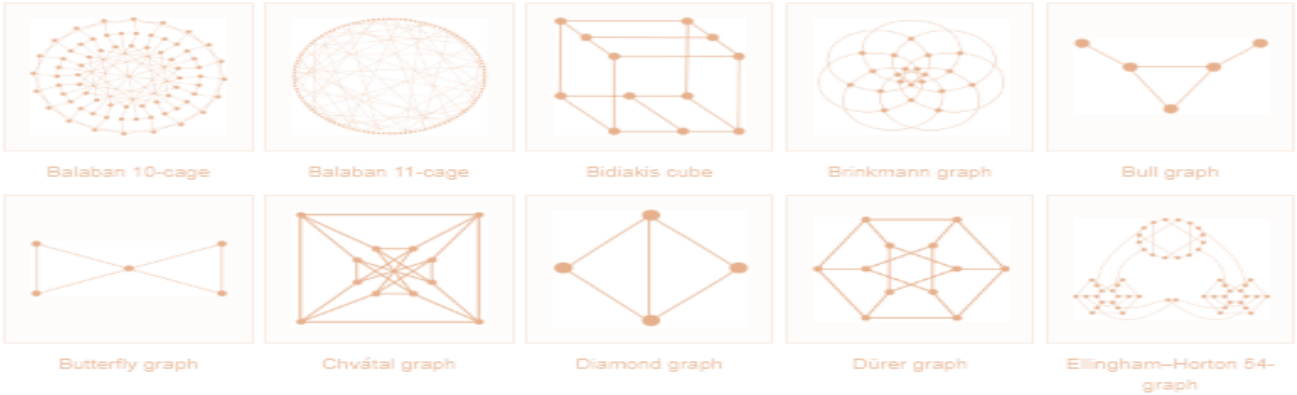
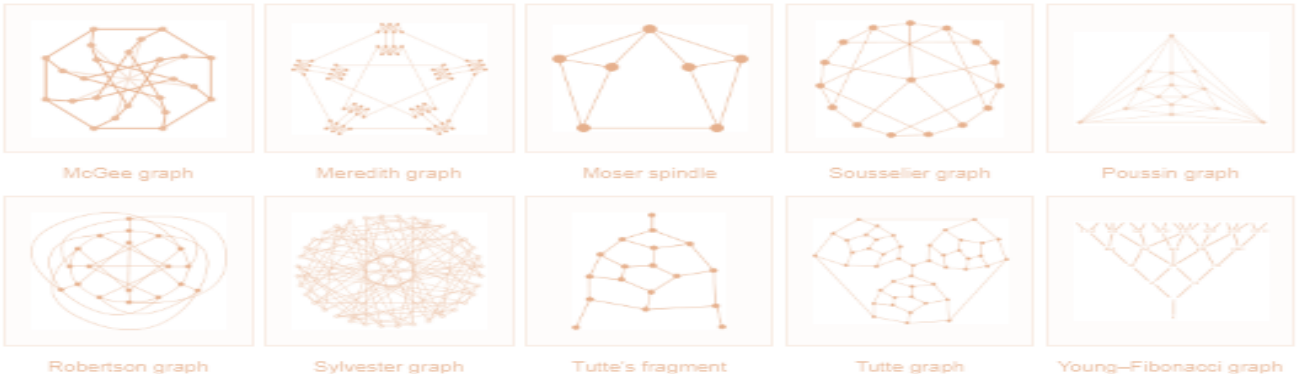
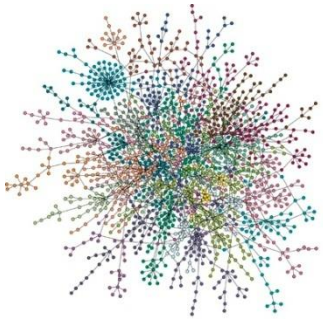


Graph Theory

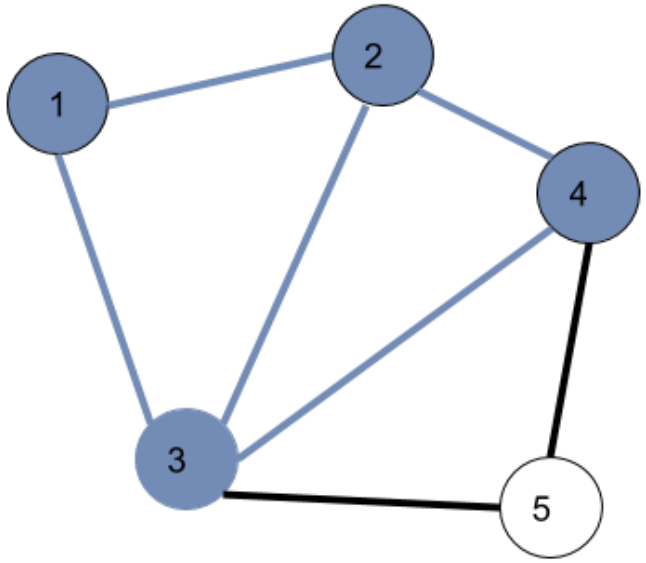
Paths and Cycles



Paths and Cycles …

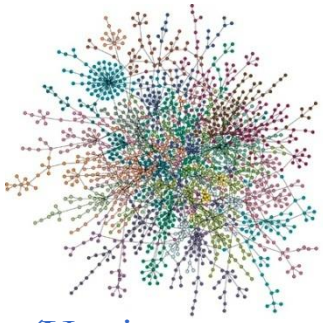


- **Walks, Trails and Paths**
  - A **walk** is a sequence of vertices and edges of a graph i.e. if we traverse a graph then we get a walk. (Vertices and Edges can be repeated)

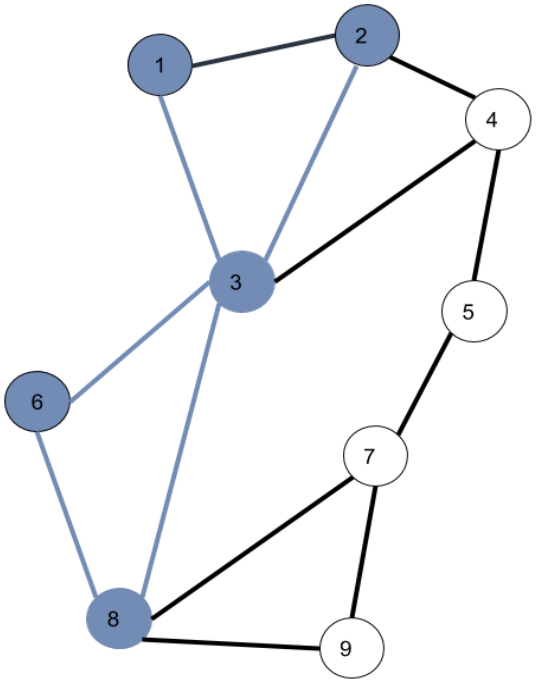


$1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 1 \rightarrow 3$ is a walk

Paths and Cycles …



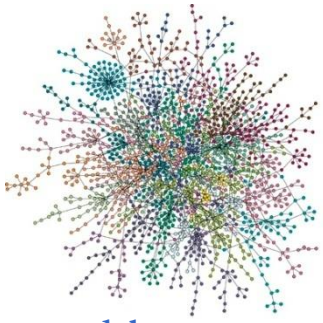
- **Walks, Trails and Paths**
  - A **Trail** is a walk in which no edge is repeated then we get a trail. (Vertices can be repeated but Edges can not be repeated)



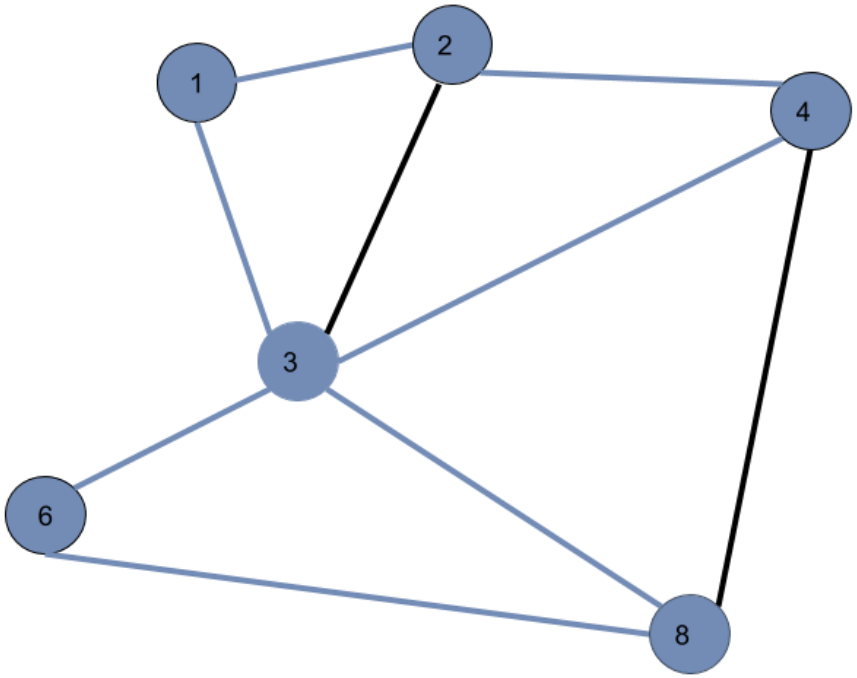
$1 \rightarrow 3 \rightarrow 8 \rightarrow 6 \rightarrow 3 \rightarrow 2$ is trail

$1 \rightarrow 3 \rightarrow 8 \rightarrow 6 \rightarrow 3 \rightarrow 2 \rightarrow 1$ is a closed trail

Paths and Cycles …

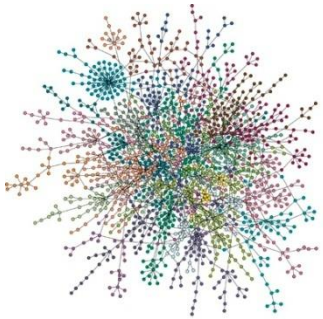


- **Walks, Trails and Paths**
  - A **Circuit** is the traversal of a graph such that not an edge is repeated but vertex can be repeated and it is closed also i.e. it is a closed trail. (Vertices can be repeated, Edges can not be repeated)

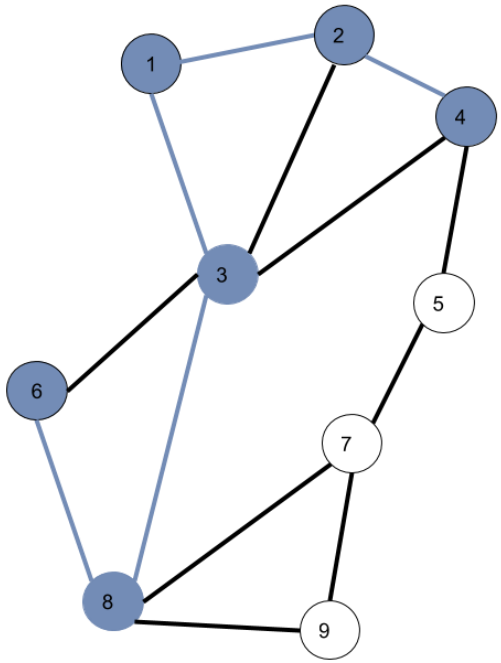


$1 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 6 \rightarrow 8 \rightarrow 3 \rightarrow 1$ is a circuit

Paths and Cycles …

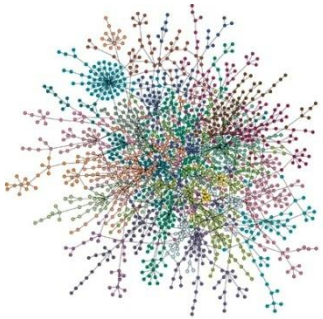


- **Walks, Trails and Paths**
  - A **Path** is a trail in which neither vertices nor edges are repeated i.e. if we traverse a graph such that we do not repeat a vertex and nor we repeat an edge. (Vertices can not be repeated, Edges can not be repeated).

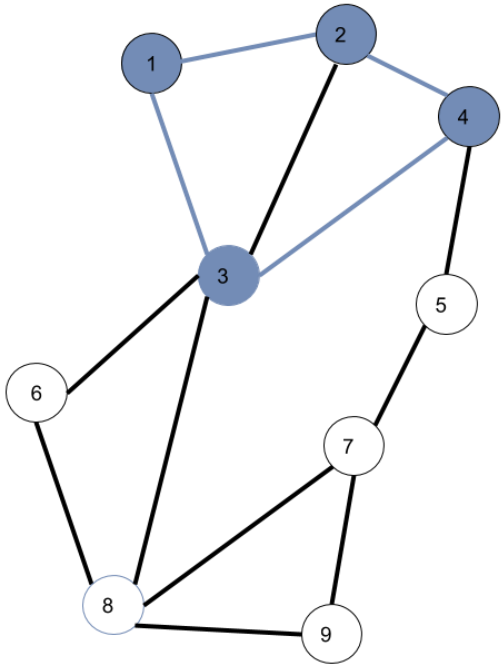


$6 \rightarrow 8 \rightarrow 3 \rightarrow 1 \rightarrow 2 \rightarrow 4$ is a Path

Paths and Cycles …

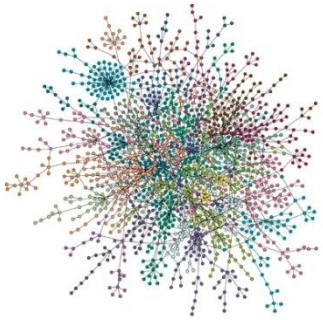


- **Walks, Trails and Paths**
  - A **Cycle** is the traversal of a graph such that we do not repeat a vertex nor we repeat a edge but the starting and ending vertex must be same i.e. we can repeat starting and ending vertex only then we get a cycle. (Vertices can not be repeated, Edges can not be repeated) Cycle is a closed path.



1→2 → 4 → 3 → 1 is a cycle.

Paths and Cycles …

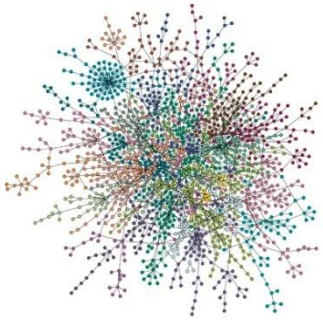


- **Walks, Trails and Paths**

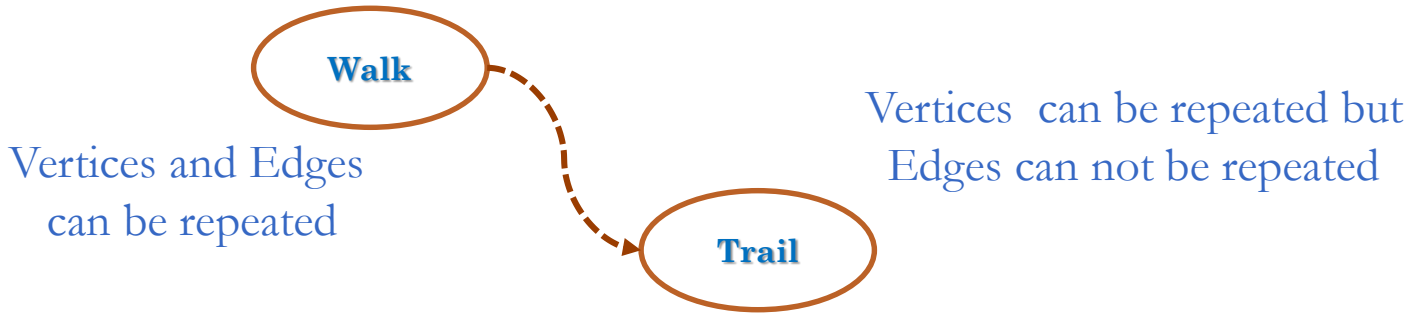
Walk

Vertices and Edges
can be repeated

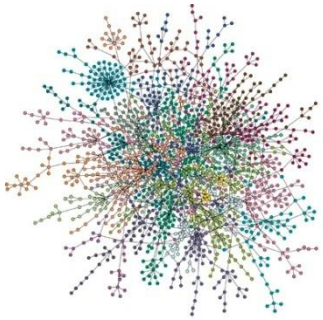
Paths and Cycles …



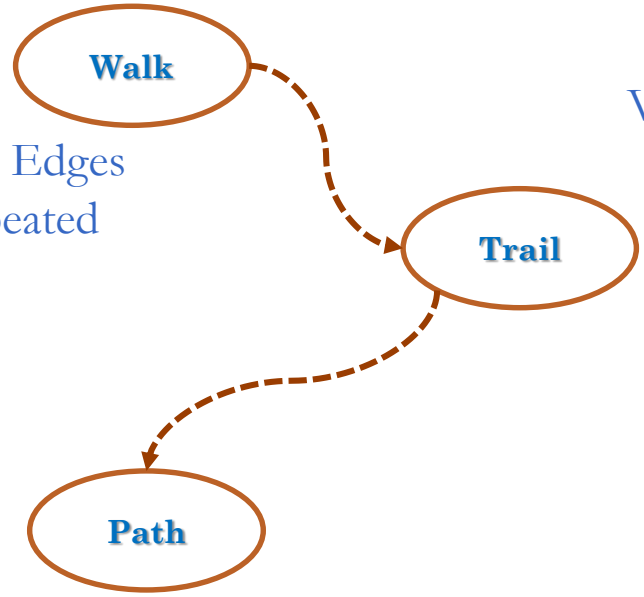
- **Walks, Trails and Paths**



Paths and Cycles …



- **Walks, Trails and Paths**

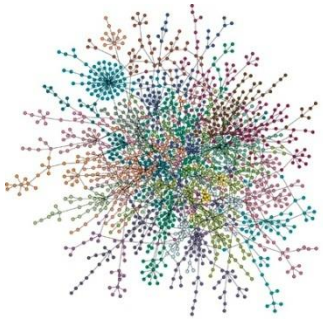


Vertices and Edges can be repeated

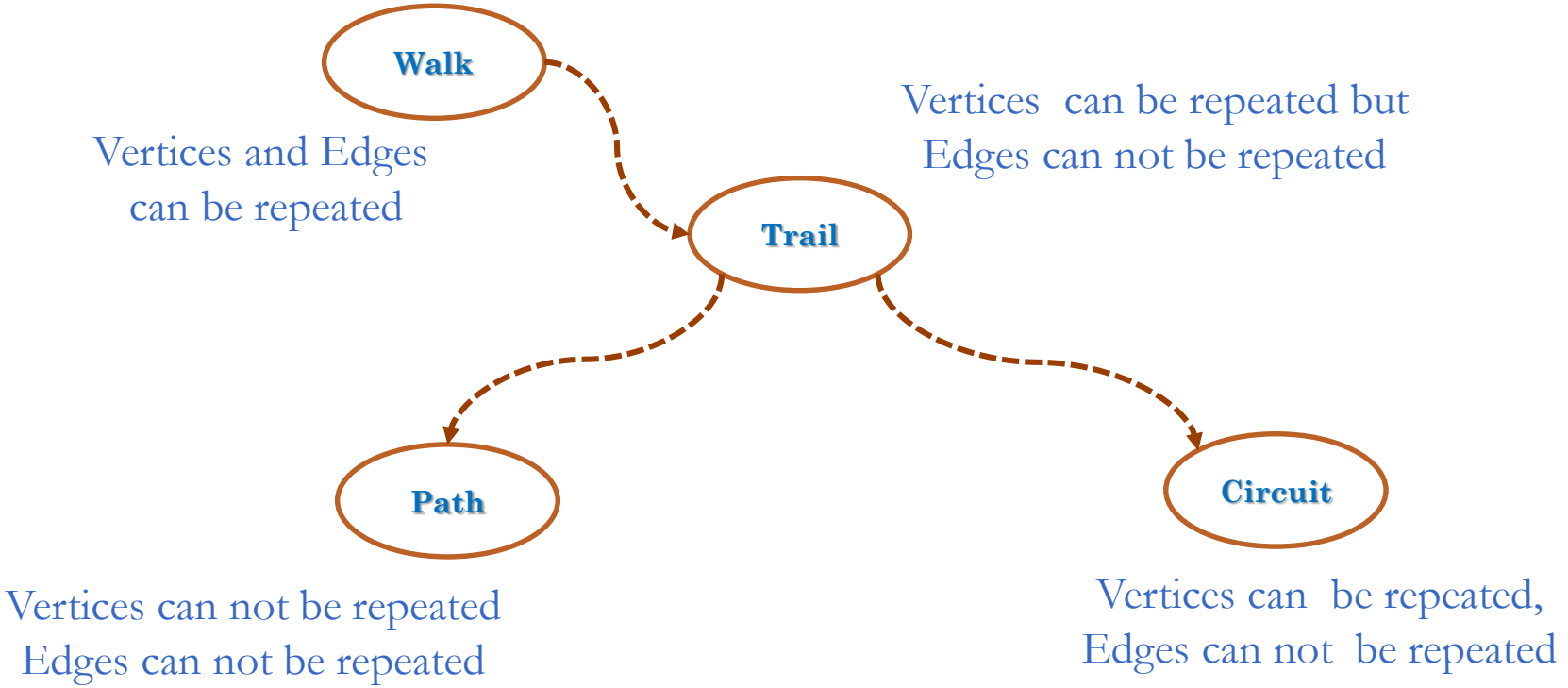
Vertices can be repeated but Edges can not be repeated

Vertices can not be repeated
Edges can not be repeated

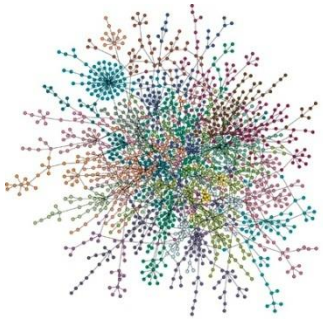
Paths and Cycles …



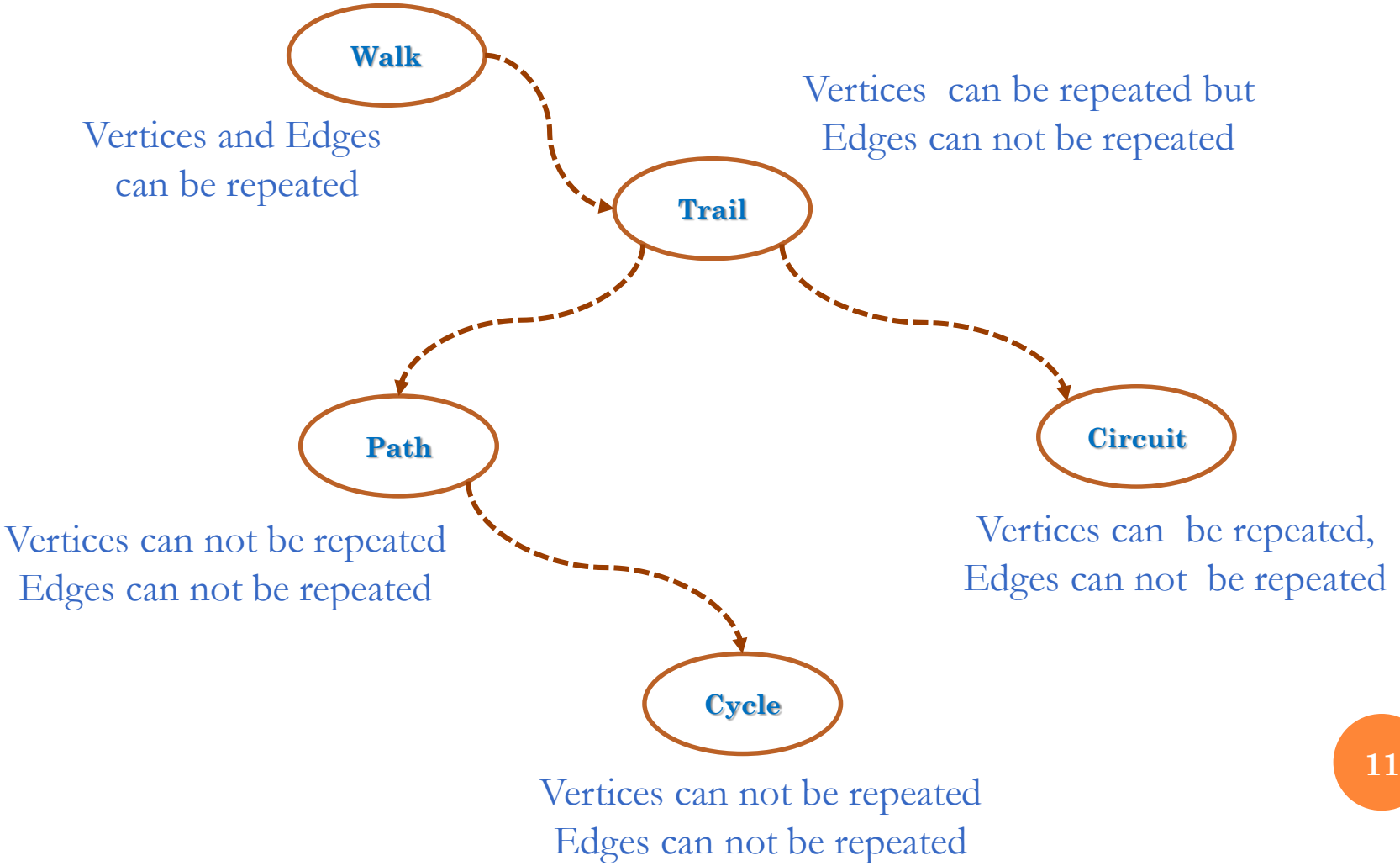
- **Walks, Trails and Paths**



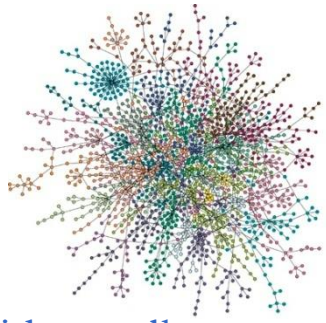
Paths and Cycles …



- **Walks, Trails and Paths**

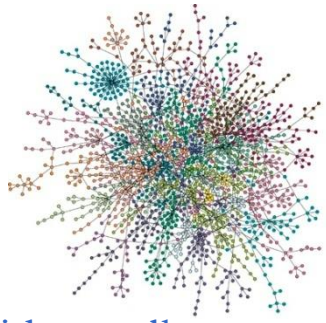


Paths and Cycles …



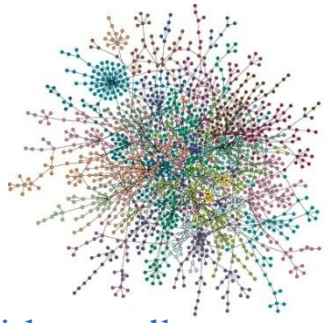
- **Walks, Trails and Paths**
  - APPLICATION: Prove that if two vertices v, w are connected with a walk, then they are connected with a path.

Paths and Cycles …



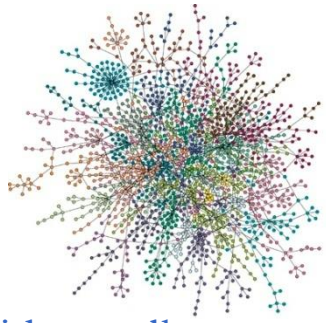
- **Walks, Trails and Paths**
  - APPLICATION: Prove that if two vertices v, w are connected with a walk, then they are connected with a path.
    - Let $v, \ldots, u, \ldots, u, \ldots, w$, and let a vertex u that appears two or more times.

Paths and Cycles …



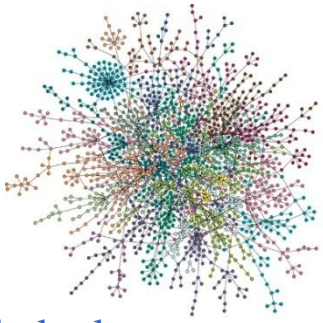
- **Walks, Trails and Paths**
  - APPLICATION: Prove that if two vertices v, w are connected with a walk, then they are connected with a path.
    - Let $v, \ldots, u, \ldots, u, \ldots, w$, and let a vertex u that appears two or more times.
    - Removing from the walk the vertices that appear between the first and the last appearance of vertex u (including the last appearance of u) it results to a walk where u appears only once.

Paths and Cycles …

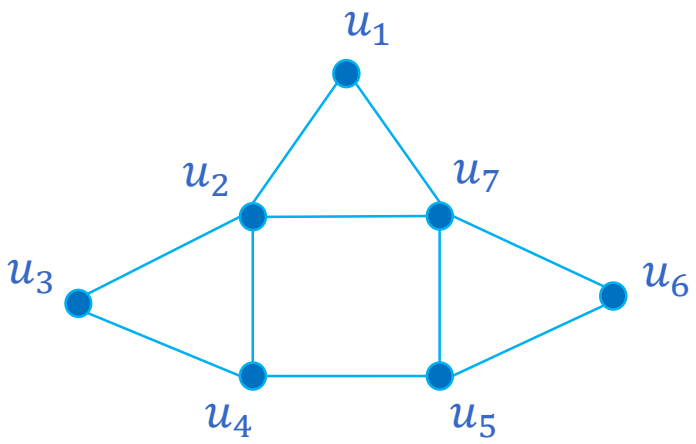


- **Walks, Trails and Paths**
  - APPLICATION: Prove that if two vertices v, w are connected with a walk, then they are connected with a path.
    - Let $v, \dots, u, \dots, u, \dots, w$, and let a vertex u that appears two or more times.
    - Removing from the walk the vertices that appear between the first and the last appearance of vertex u (including the last appearance of u) it results to a walk where u appears only once.
    - Iterate over all vertices with multiple instances on the walk.

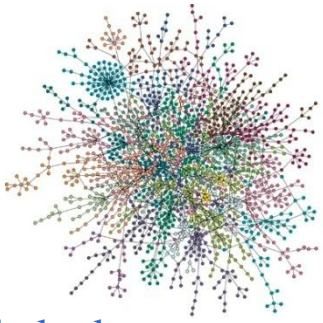
Paths and Cycles …



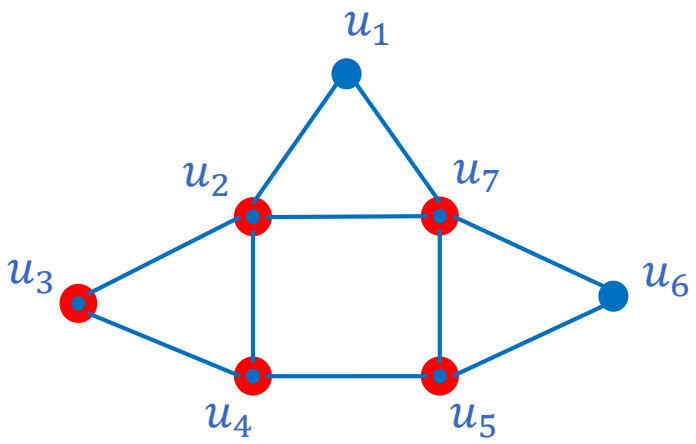
- **Walks, Trails and Paths**
  - The length of a walk/trail/path equals the cardinality of the included edges.



Paths and Cycles …

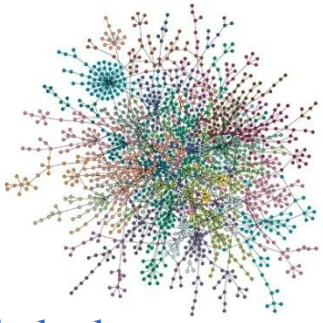


- **Walks, Trails and Paths**
  - The length of a walk/trail/path equals the cardinality of the included edges.

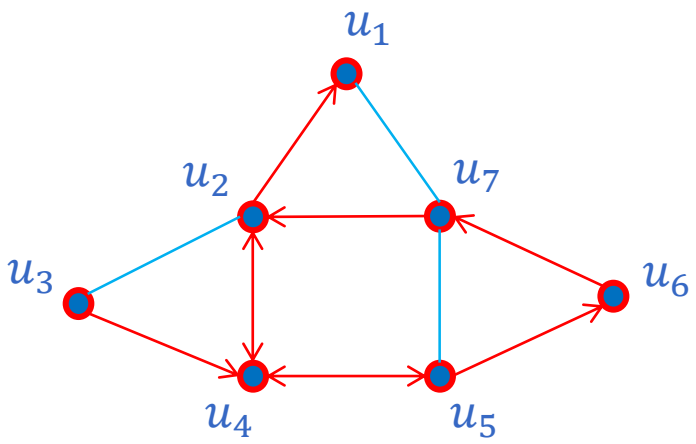


$$A_1 = (v_3, v_4, v_5, v_7, v_2, v_4, v_5, v_2, v_1)$$

Paths and Cycles …

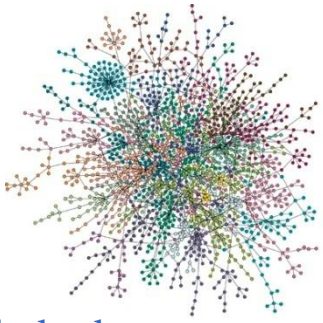


- **Walks, Trails and Paths**
  - The length of a walk/trail/path equals the cardinality of the included edges.

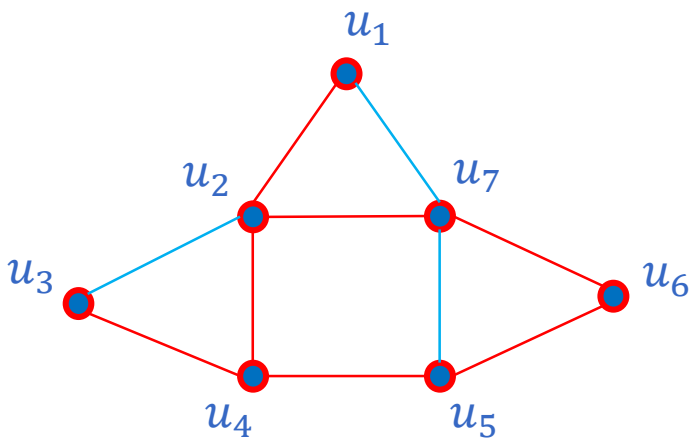


$$A_1 = (v_3, v_4, v_5, v_7, v_2, v_4, v_5, v_2, v_1)$$
$$A_2 = (v_3, v_4, v_5, v_6, v_7, v_5, v_4, v_2, v_1)$$

Paths and Cycles …

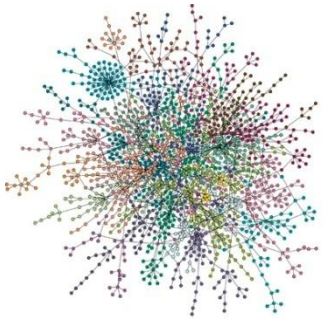


- **Walks, Trails and Paths**
  - The length of a walk/trail/path equals the cardinality of the included edges.

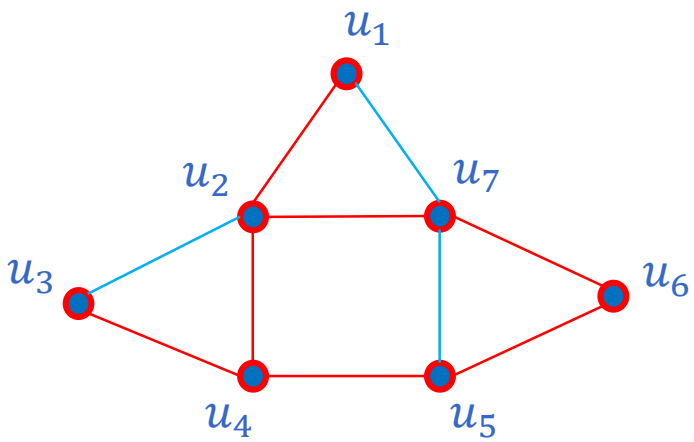


$$A_1 = (v_3, v_4, v_5, v_7, v_2, v_4, v_5, v_2, v_1)$$
$$A_2 = (v_3, v_4, v_5, v_6, v_7, v_5, v_4, v_2, v_1)$$
$$A_3 = (v_3, v_4, v_5, v_6, v_7, v_5, v_2, v_1)$$

Paths and Cycles …

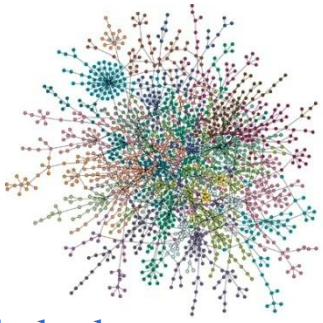


- **Walks, Trails and Paths**
  - The length of a walk/trail/path equals the cardinality of the included edges.

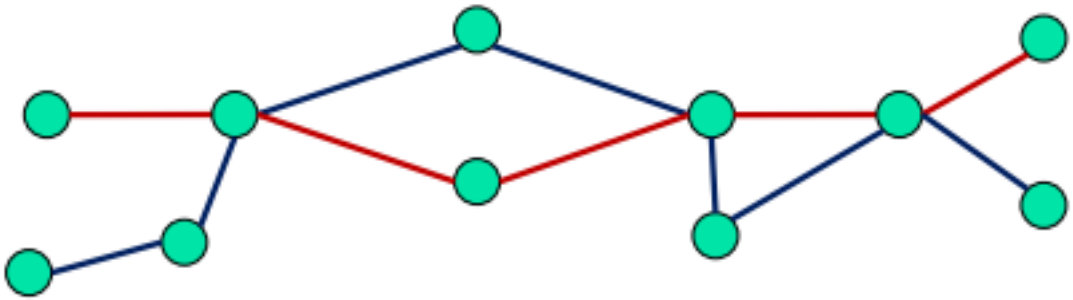


$$A_1 = (v_3, v_4, v_5, v_7, v_2, v_4, v_5, v_2, v_1)$$
$$A_2 = (v_3, v_4, v_5, v_6, v_7, v_5, v_4, v_2, v_1)$$
$$A_3 = (v_3, v_4, v_5, v_6, v_7, v_5, v_2, v_1)$$
$$A_4 = (v_3, v_4, v_5, v_6, v_7, v_2, v_1)$$

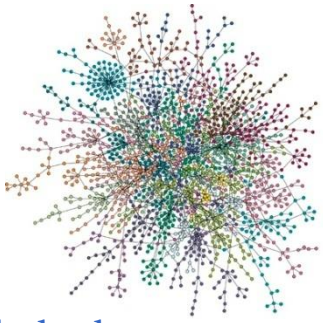
Paths and Cycles …



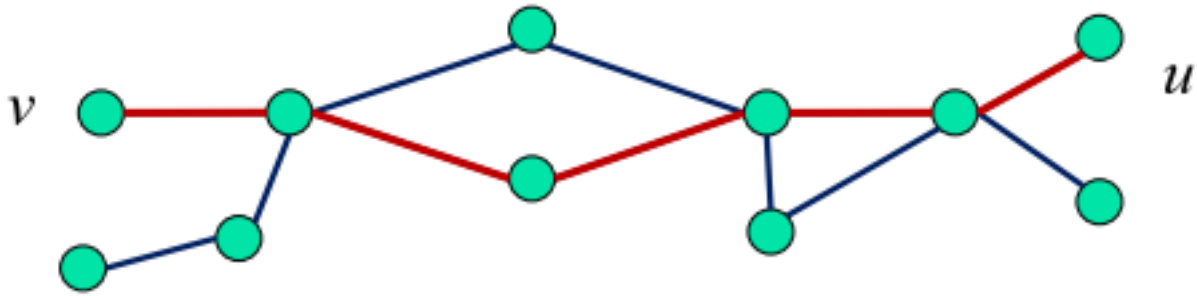
- **Distances**
  - The length of a walk/trail/path equals the cardinality of the included edges.
  - Edge Disjoint Paths:



Paths and Cycles …

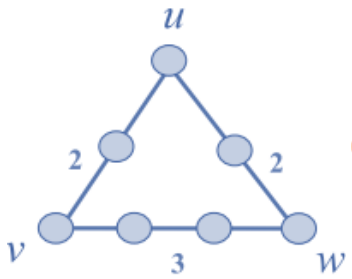


- **Distances**
  - The length of a walk/trail/path equals the cardinality of the included edges.
  - Distance between two vertices:

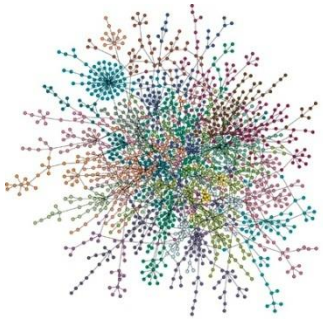


The length of a shortest path from u to v.

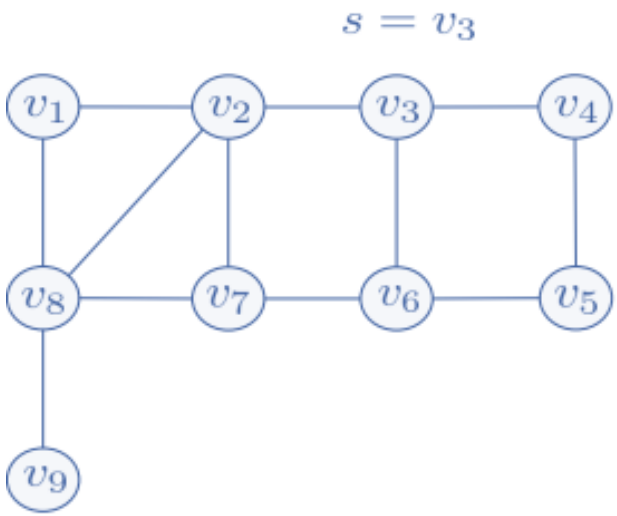
- Non-negative: $dist(v,u) > 0$ for $v \neq u$ $(dist(v,u) = 0,\ if\ v = u)$
- Symmetrical: $dist(v,u) = dist(u,v)$
- Triangle Inequality: $dist(v,u) + dist(u,w) \geq dist(v,w)$

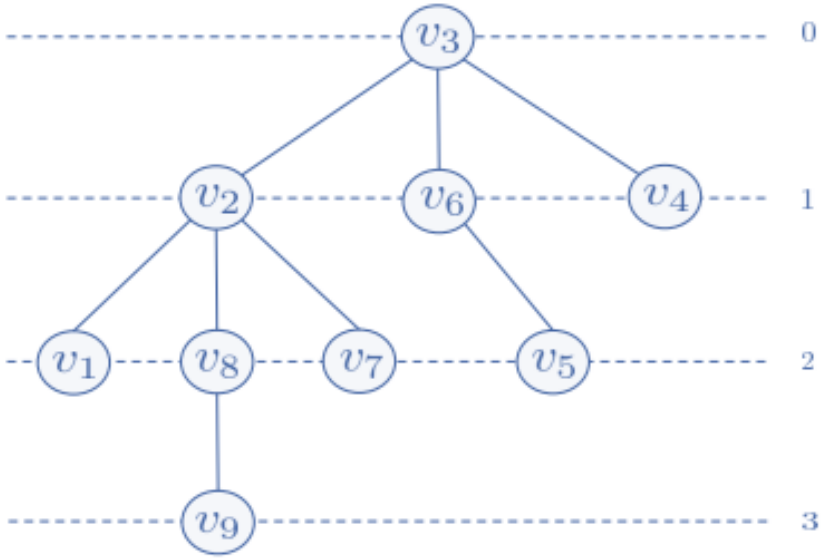


Paths and Cycles …



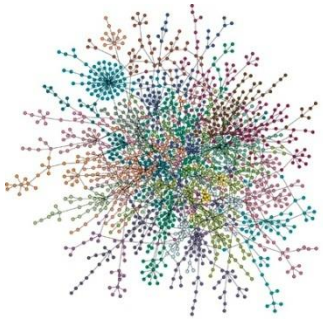
- **Distances**



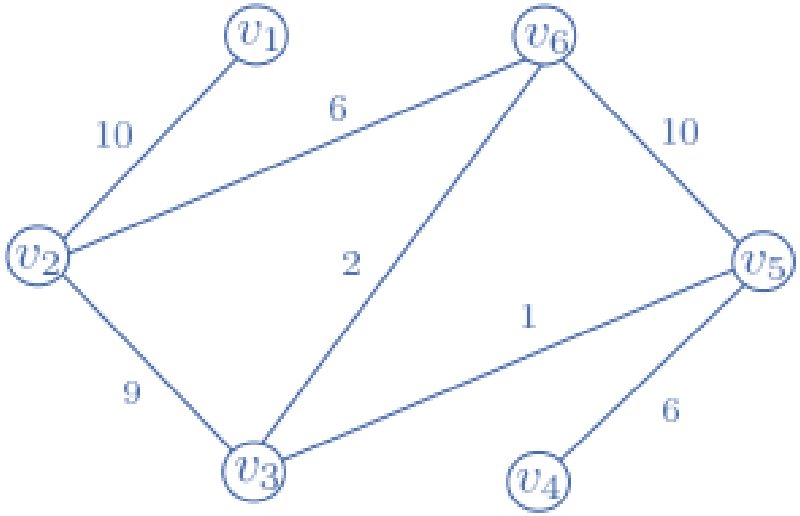


Compute Distances between vertices of simple graphs (Moore, 1959)

Paths and Cycles …



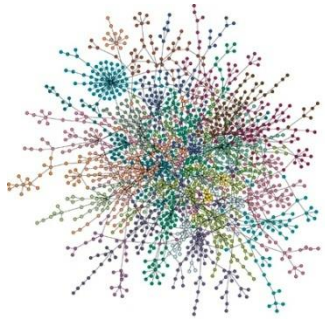
- **Distances**



$dist(v_1, v_4) = 25$

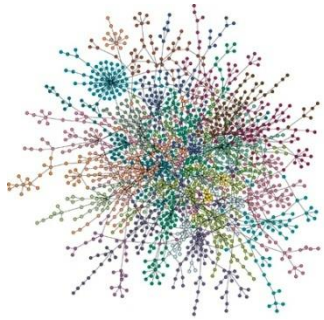
Compute Distances between vertices of weighted graphs (Dijkstra, 1959)

Paths and Cycles …



- **Distances**
  - **Eccentricity** is the maximum distance from vertex v to any other vertex u of the graph. → $e(v) = \max\{d(v,u) | u \in V(G)\}$
  - **Radius** of a graph G, radius(G) is the minimum eccentricity exhibited across the vertices of G. → $radius(G) = \min\{e(v) | v \in V(G)\}$
  - **Diameter** of a graph G, is the maximum eccentricity exhibited across the vertices of G G. → $diameter(G) = \max\{e(v) | v \in V(G)\}$

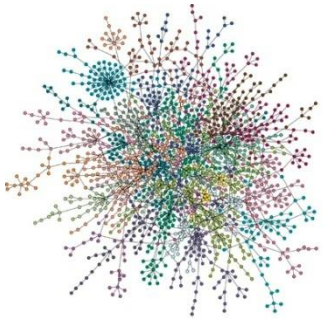
Paths and Cycles …



- **Distances**
  - **Eccentricity** is the maximum distance from vertex v to any other vertex u of the graph. → $e(v) = \max\{d(v,u) | u \in V(G)\}$
  - **Radius** of a graph G, radius(G) is the minimum eccentricity exhibited across the vertices of G. → $radius(G) = \min\{e(v) | v \in V(G)\}$
  - **Diameter** of a graph G, is the maximum eccentricity exhibited across the vertices of G G. → $diameter(G) = \max\{e(v) | v \in V(G)\}$

$$\boldsymbol{radius(G) \leq diameter(G) \leq 2 \cdot radius(G)}$$

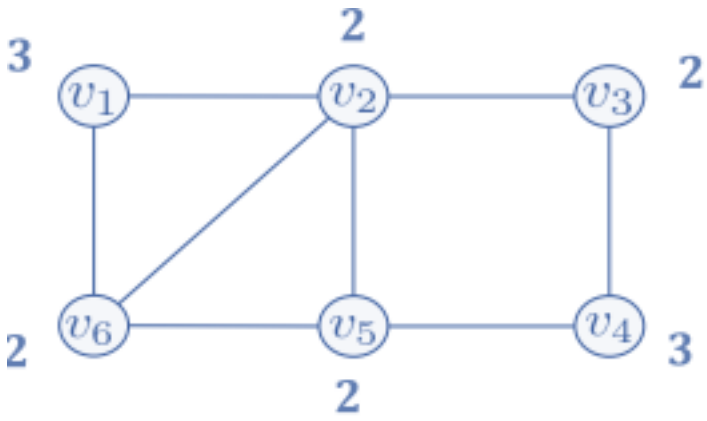
Paths and Cycles …



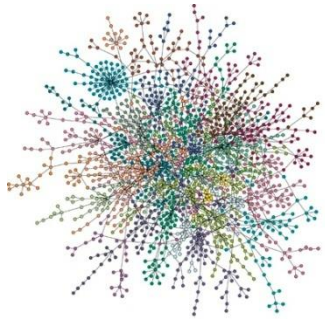
Distances

- **Eccentricity** is the maximum distance from vertex v to any other vertex u of the graph. → $e(v) = \max\{d(v,u) | u \in V(G)\}$
- **Radius** of a graph G, radius(G) is the minimum eccentricity exhibited across the vertices of G. → $radius(G) = \min\{e(v) | v \in V(G)\}$
- **Diameter** of a graph G, is the maximum eccentricity exhibited across the vertices of G G. → $diameter(G) = \max\{e(v) | v \in V(G)\}$

$$\boldsymbol{radius(G) \leq diameter(G) \leq 2 \cdot radius(G)}$$

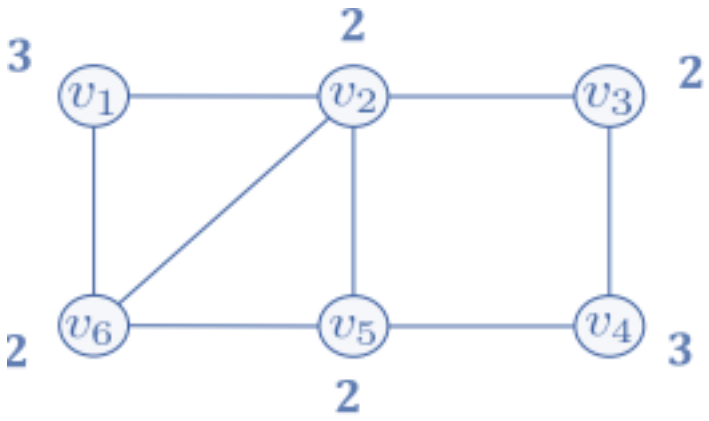


Paths and Cycles …



- **Distances**
  - **Eccentricity** is the maximum distance from vertex v to any other vertex u of the graph. → $e(v) = \max\{d(v,u) | u \in V(G)\}$
  - **Radius** of a graph G, radius(G) is the minimum eccentricity exhibited across the vertices of G. → $radius(G) = \min\{e(v) | v \in V(G)\}$
  - **Diameter** of a graph G, is the maximum eccentricity exhibited across the vertices of G G. → $diameter(G) = \max\{e(v) | v \in V(G)\}$

$$\boldsymbol{radius(G) \leq diameter(G) \leq 2 \cdot radius(G)}$$

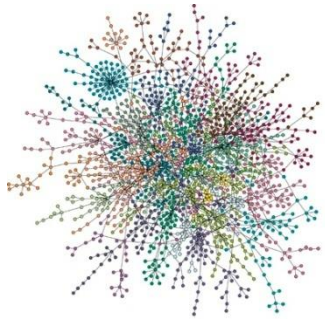


$radius(G)$

$diameter(G)$

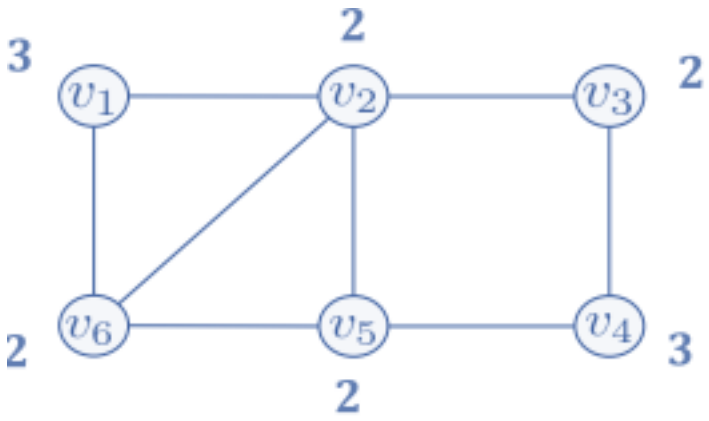
?

Paths and Cycles …



- **Distances**
  - **Eccentricity** is the maximum distance from vertex v to any other vertex u of the graph. → $e(v) = \max\{d(v,u) | u \in V(G)\}$
  - **Radius** of a graph G, radius(G) is the minimum eccentricity exhibited across the vertices of G. → $radius(G) = \min\{e(v) | v \in V(G)\}$
  - **Diameter** of a graph G, is the maximum eccentricity exhibited across the vertices of G G. → $diameter(G) = \max\{e(v) | v \in V(G)\}$

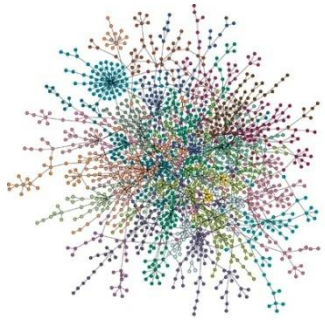
$$\boldsymbol{radius(G) \leq diameter(G) \leq 2 \cdot radius(G)}$$



$radius(G) = 2$

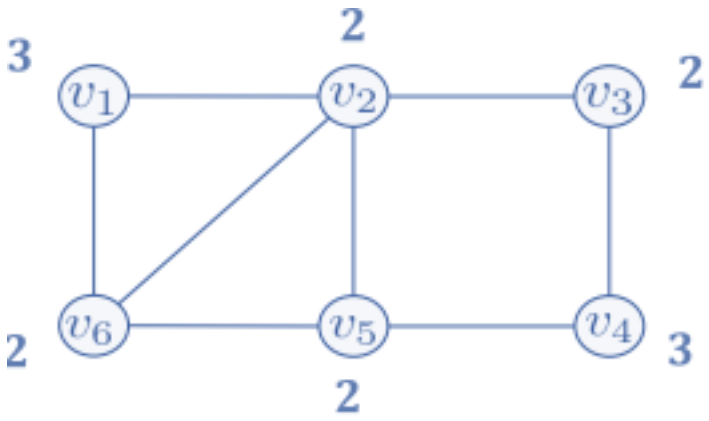
$diameter(G) = 3$

Paths and Cycles …



- **Distances**
  - **Eccentricity** is the maximum distance from vertex v to any other vertex u of the graph. → $e(v) = \max\{d(v,u) | u \in V(G)\}$
  - **Radius** of a graph G, radius(G) is the minimum eccentricity exhibited across the vertices of G. → $radius(G) = \min\{e(v) | v \in V(G)\}$
  - **Diameter** of a graph G, is the maximum eccentricity exhibited across the vertices of G G. → $diameter(G) = \max\{e(v) | v \in V(G)\}$

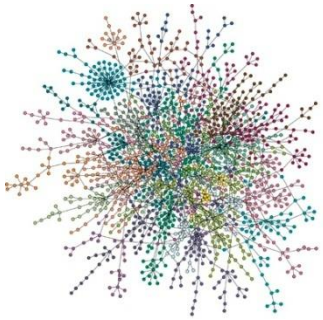
$$\boldsymbol{radius(G) \leq diameter(G) \leq 2 \cdot radius(G)}$$



$radius(G) = 2$

$diameter(G) = 3$

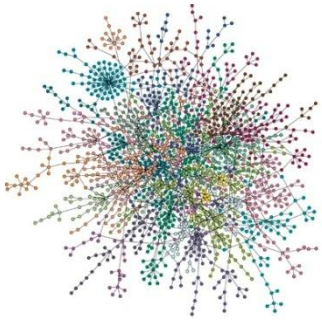
Paths and Cycles …



- **Distances**
  - APPLICATION:

    Prove that in each graph it holds that: $\boldsymbol{radius(G) \leq diameter(G) \leq 2 \cdot radius(G)}$

Paths and Cycles …

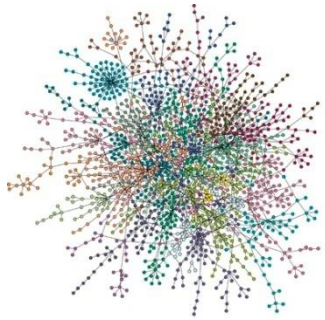


- **Distances**
  - APPLICATION:

    Prove that in each graph it holds that: $\boldsymbol{radius(G) \leq diameter(G) \leq 2 \cdot radius(G)}$

    - By definition it holds that the left part is truth.

Paths and Cycles …

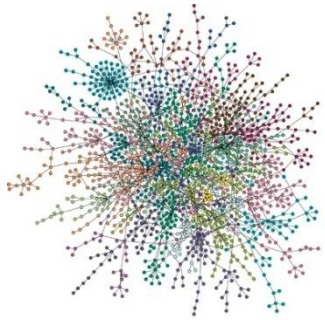


- **Distances**
  - APPLICATION:

    Prove that in each graph it holds that: $\boldsymbol{radius(G) \leq diameter(G) \leq 2 \cdot radius(G)}$

    - By definition it holds that the left part is truth.
    - For the right part, we assume that
      - there exist two vertices x, y: $dist(x, y) = diameter(G)$
      - Let z a vertex: the largest shortest path from z has length equal the radius of $G \rightarrow e(z) = radius(G)$

Paths and Cycles …



- **Distances**
  - APPLICATION:

    Prove that in each graph it holds that: $\boldsymbol{radius(G) \leq diameter(G) \leq 2 \cdot radius(G)}$

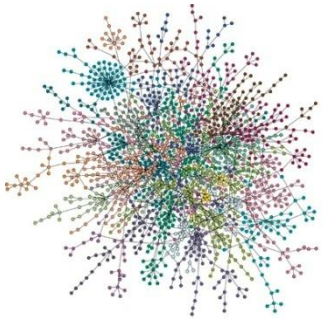
    - By definition it holds that the left part is truth.
    - For the right part, we assume that
      - there exist two vertices x, y: $dist(x, y) = diameter(G)$
      - Let z a vertex: the largest shortest path from z has length equal the radius of $G \rightarrow e(z) = radius(G)$

      - From the Triangle Inequality it holds that:

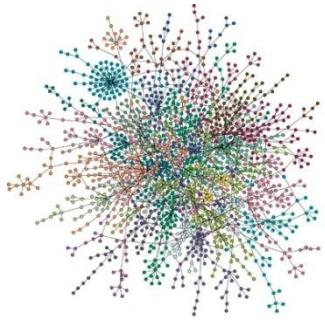
        $\boldsymbol{diameter(G) = dist(x, y) \leq dist(x, z) + dist(z, y) \leq e(z) + e(z) = 2 \cdot radius(G)}$

Paths and Cycles …

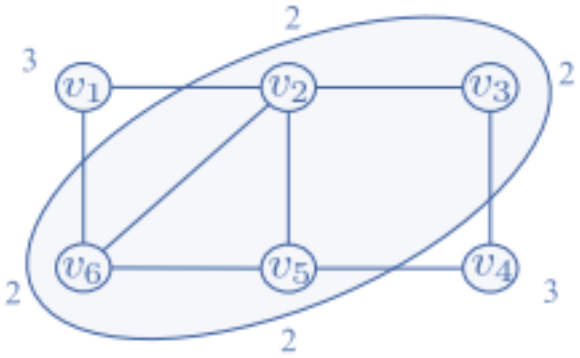


- **Graph Center and Graph Median**
  - The subgraph of a graph G induced by the vertices of G with the minimum eccentricity is called **Center** of graph G (denoted as center(G)).

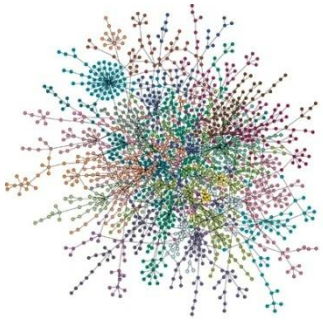
Paths and Cycles …



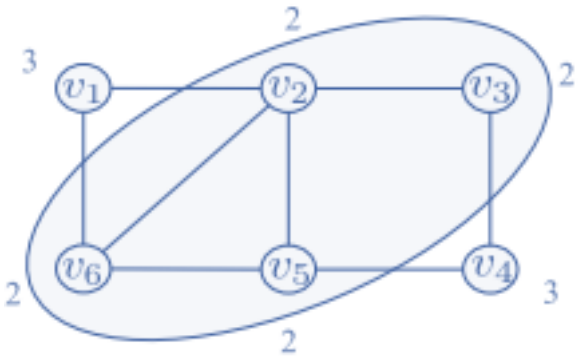
- **Graph Center and Graph Median**
  - The subgraph of a graph G induced by the vertices of G with the minimum eccentricity is called **Center** of graph G (denoted as center(G)).

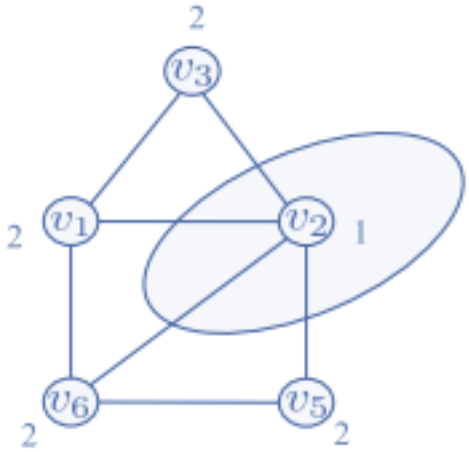


Paths and Cycles …

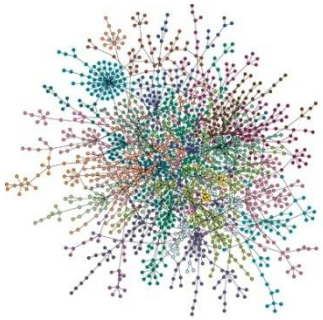


- **Graph Center and Graph Median**
  - The subgraph of a graph G induced by the vertices of G with the minimum eccentricity is called **Center** of graph G (denoted as center(G)).

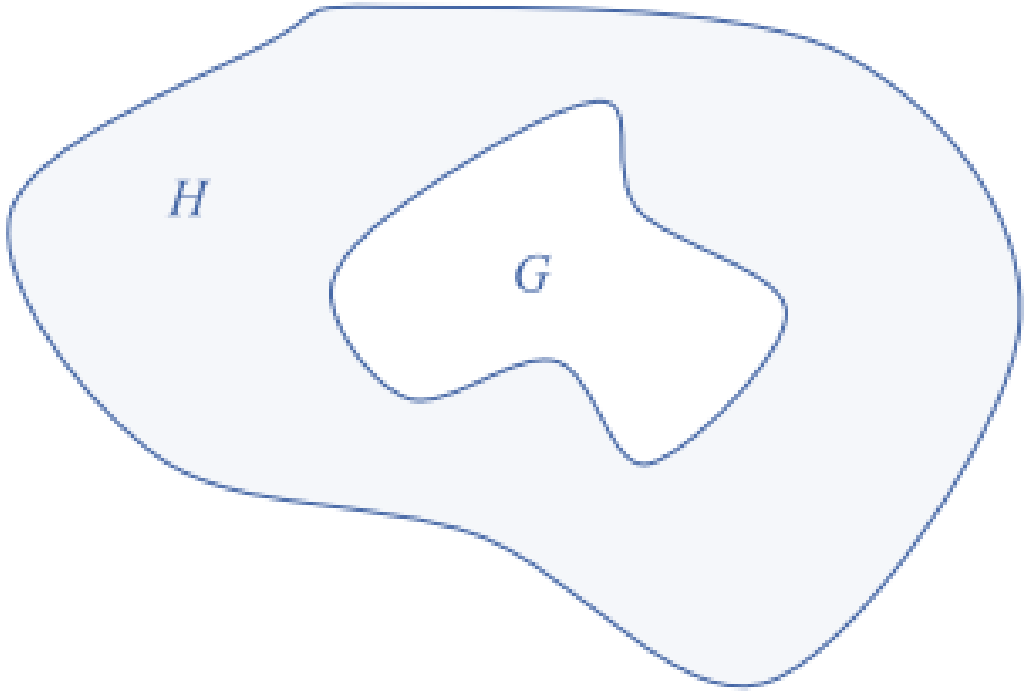




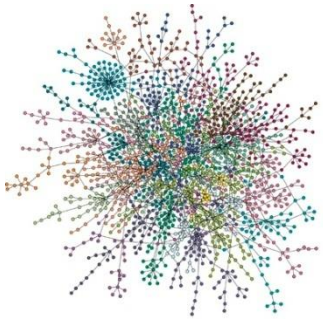
Paths and Cycles …



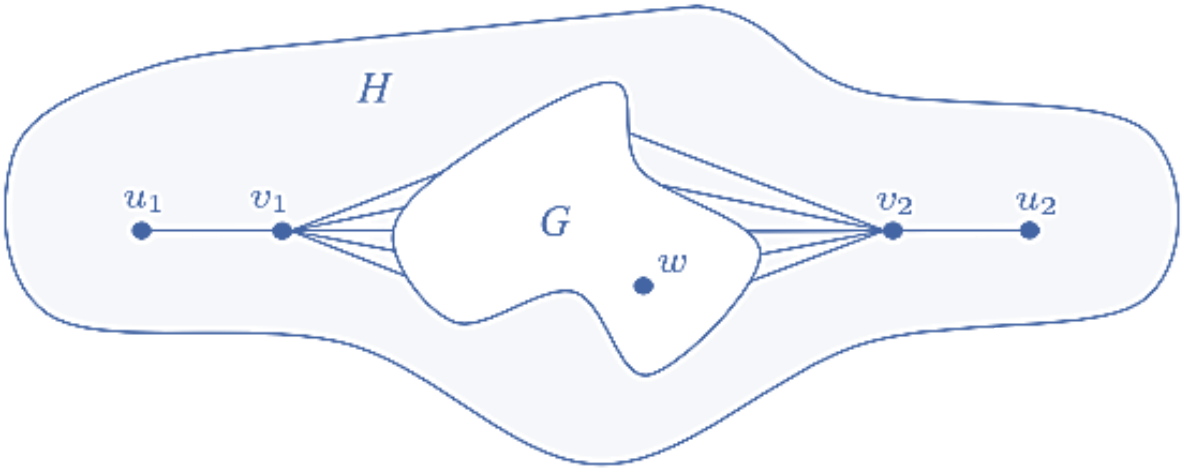
- **Graph Center and Graph Median**
  - Theorem: Each graph G is the center of a connected graph H.



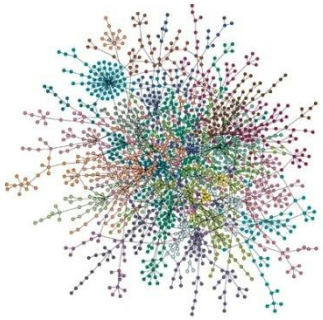
Paths and Cycles …



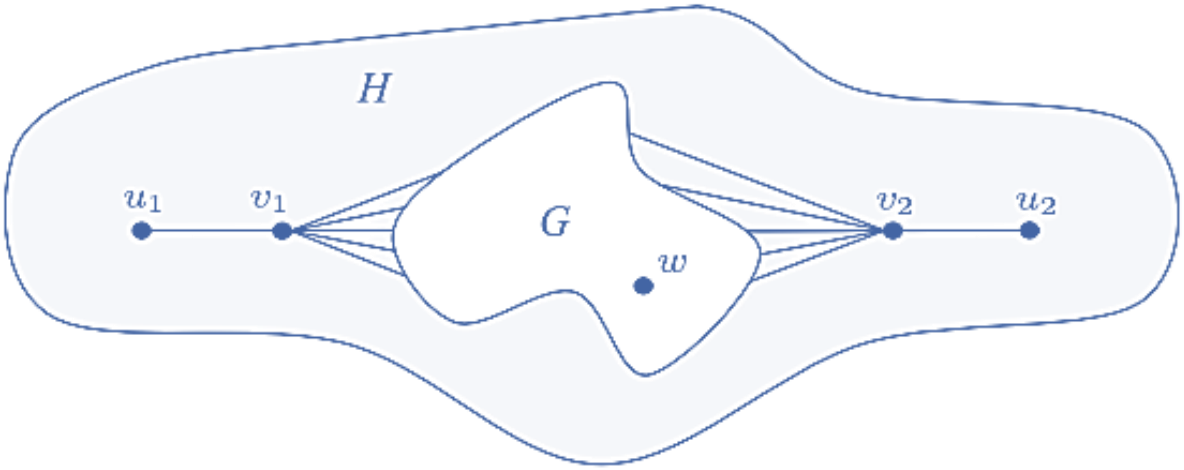
- **Graph Center and Graph Median**
  - Theorem: Each graph G is the center of a connected graph H.



Paths and Cycles …

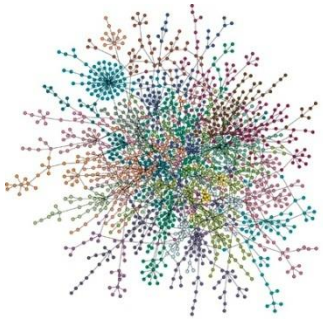


- **Graph Center and Graph Median**
  - Theorem: Each graph G is the center of a connected graph H.

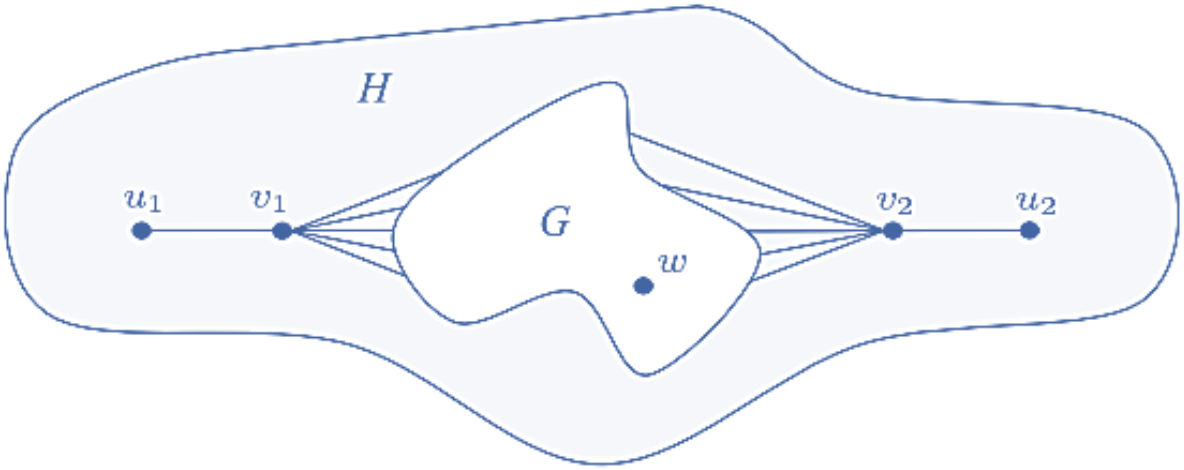


1. Construct the hyper-graph H …

Paths and Cycles …

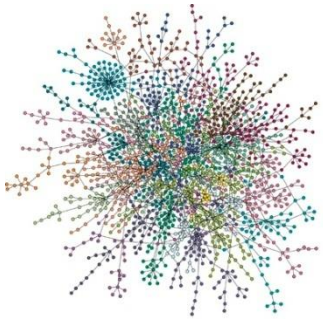


- **Graph Center and Graph Median**
  - Theorem: Each graph G is the center of a connected graph H.



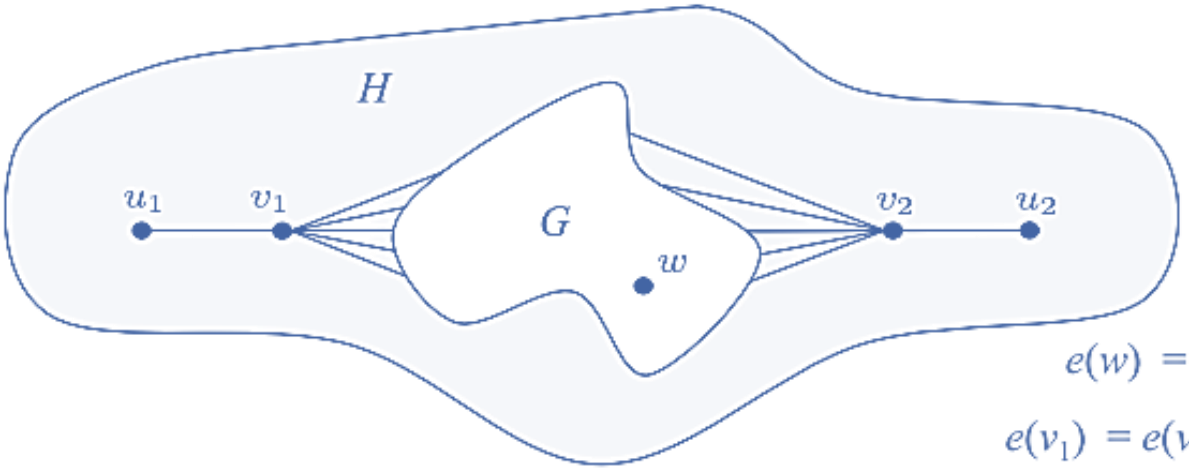
1. Construct the hyper-graph H …
2. Vertices v_1, v_2 are connected to all the vertices of H while u_1, u_2 are connected to v_1, v_2.

Paths and Cycles …



Graph Center and Graph Median

- Theorem: Each graph G is the center of a connected graph H.



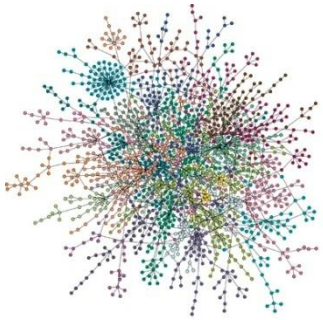
$e(w) = 2$

$e(v_1) = e(v_2) = 3$

$e(u_1) = e(u_2) = 4$

1. Construct the hyper-graph H …
2. Vertices v_1, v_2 are connected to all the vertices of H while u_1, u_2 are connected to v_1, v_2.
3. In hyper-graph H it holds that $e(v) = 2 \; \forall \, v \in E(G)$, while it holds that $e(v_1) = e(v_2) = 3$ and $e(u_1) = e(u_2) = 4$, hence the graph G consists the center of graph H.

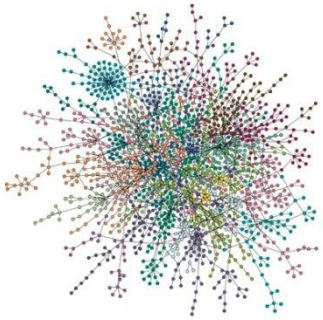
Paths and Cycles …



- **Graph Center and Graph Median**
  - **Vertex Distance** is defined as the sum of the distances between a specific vertex and all the vertices of the graph.

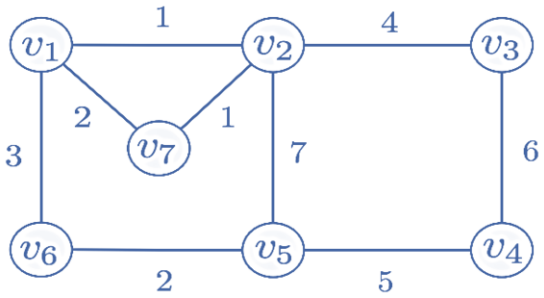
$$dist(v) = \sum_{u \in V(G)} dist(v, u)$$

Paths and Cycles …

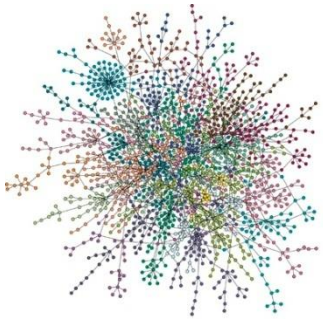


- **Graph Center and Graph Median**
  - **Vertex Distance** is defined as the sum of the distances between a specific vertex and all the vertices of the graph.

$$dist(v) = \sum_{u \in V(G)} dist(v,u)$$

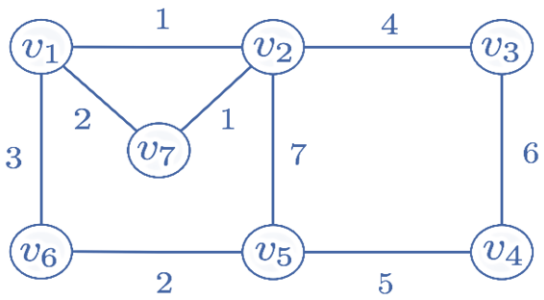


Paths and Cycles …



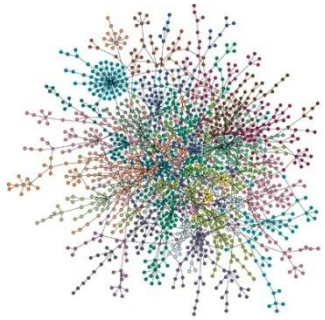
- **Graph Center and Graph Median**
  - **Vertex Distance** is defined as the sum of the distances between a specific vertex and all the vertices of the graph.

$$dist(v) = \sum_{u \in V(G)} dist(v,u)$$



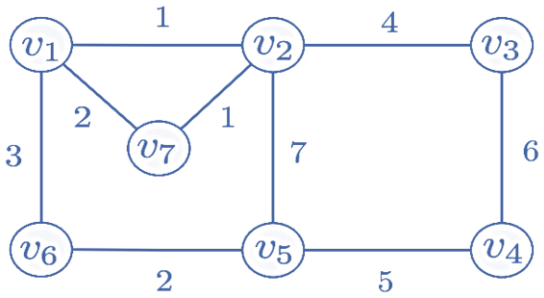
$dist(v_1) =$

Paths and Cycles …

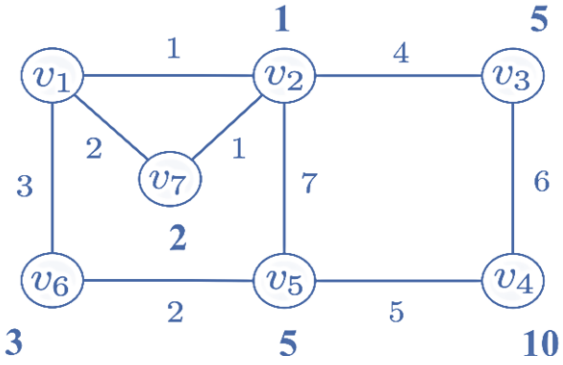


- **Graph Center and Graph Median**
  - **Vertex Distance** is defined as the sum of the distances between a specific vertex and all the vertices of the graph.

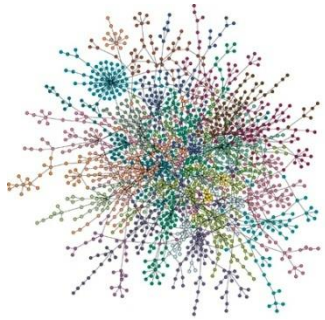
$$dist(v) = \sum_{u \in V(G)} dist(v,u)$$



$dist(v_1)$ =

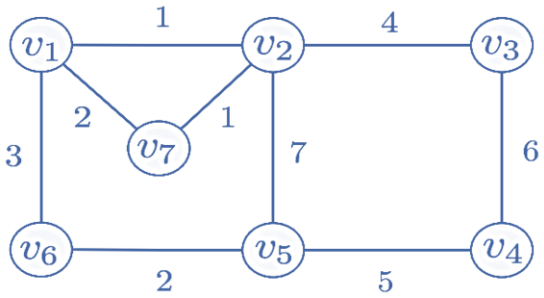


Paths and Cycles …

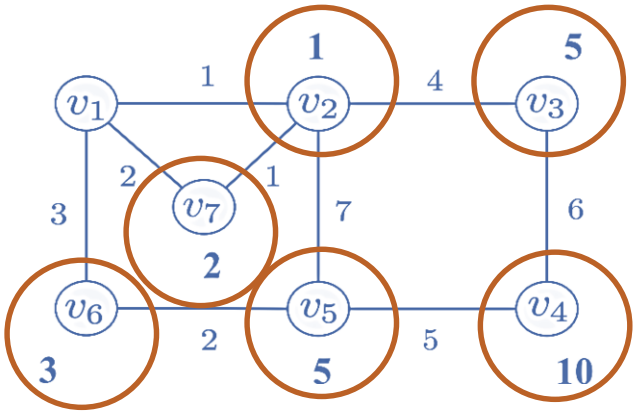


- **Graph Center and Graph Median**
  - **Vertex Distance** is defined as the sum of the distances between a specific vertex and all the vertices of the graph.

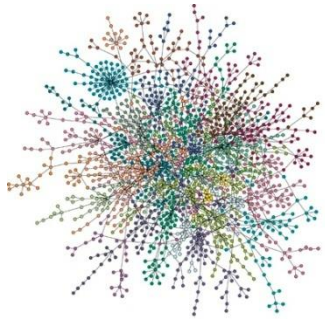
$$dist(v) = \sum_{u \in V(G)} dist(v,u)$$



$dist(v_1)$ =

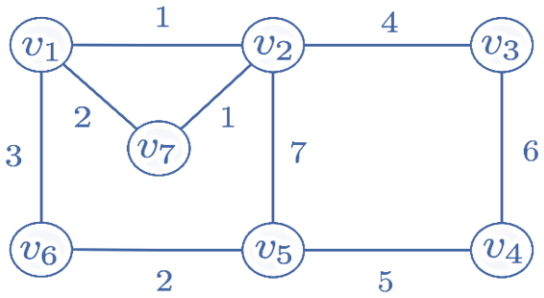


Paths and Cycles …

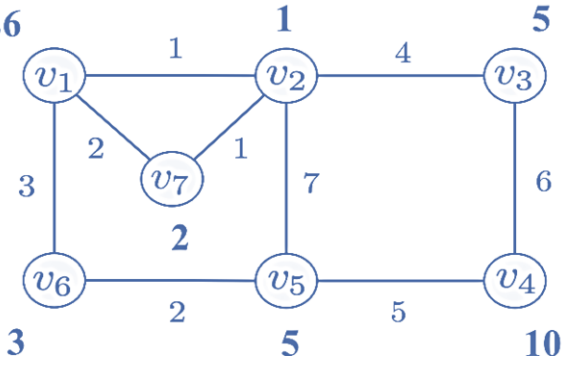


- **Graph Center and Graph Median**
  - **Vertex Distance** is defined as the sum of the distances between a specific vertex and all the vertices of the graph.

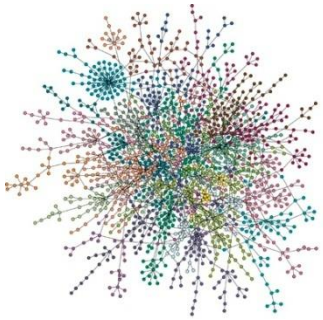
$$dist(v) = \sum_{u \in V(G)} dist(v,u)$$



$dist(v_1) = 26$

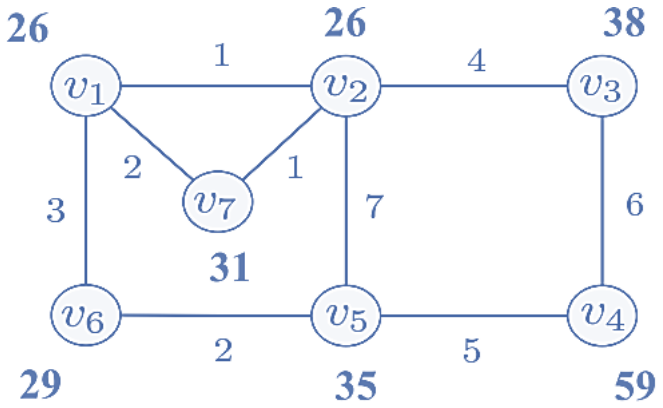


Paths and Cycles …

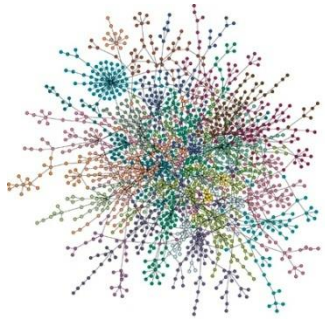


- **Graph Center and Graph Median**
  - **Vertex Distance** is defined as the sum of the distances between a specific vertex and all the vertices of the graph.

$$dist(v) = \sum_{u \in V(G)} dist(v,u)$$



Paths and Cycles …



- **Diameter of a Graph**

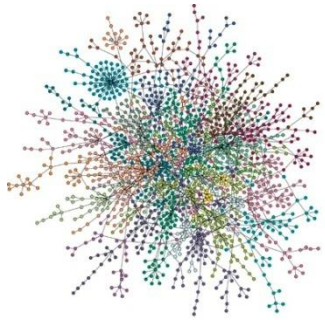
  Algorithm Center Computation

  Input: Matrix D (Floyd) with the Distances of the vertices of a graph G

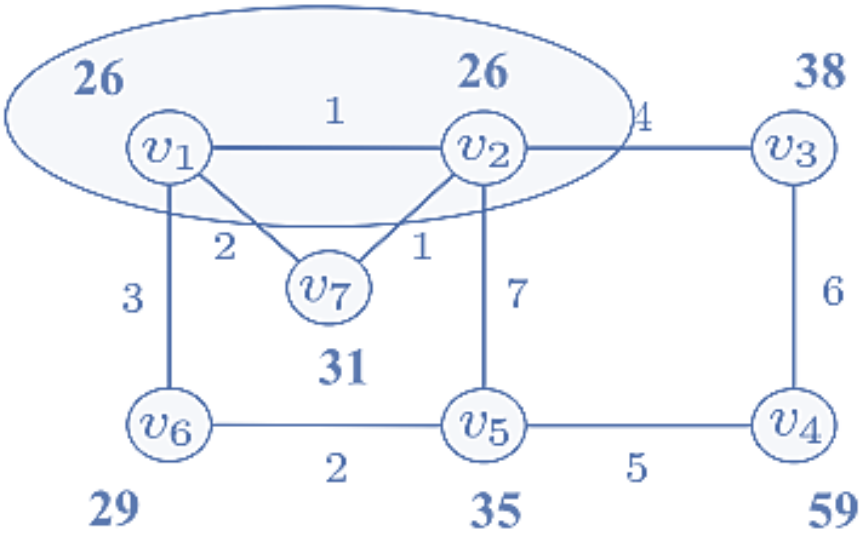
  Output: The Center of the graph G.

```
1. For i=1:1:n
2.       dist(i)← 0
3. For j=1:1:n
4.       if  (i ≠ j AND D(i,j) >dist(i))
5.           dist(i) ← D(i,j)
6. Center ← 1
7. For i=1:1:n
8.        if(dist(i)<dist(Center))
9.            Center ← i
```

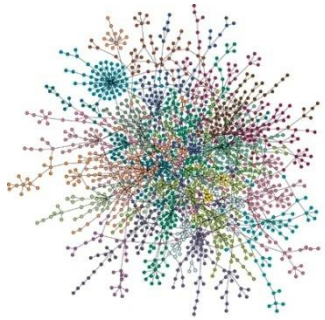
Paths and Cycles …



- **Graph Center and Graph Median**
  - The subgraph of a graph G induced by the vertices of G with the minimum distance is called **Median** of graph G (denoted as median(G)).



Paths and Cycles …



- **Graph Center and Graph Median**

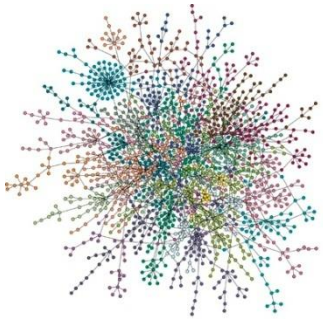
  Algorithm Median Computation

  Input: Matrix D (Floyd) with the Distances of the vertices of a graph G

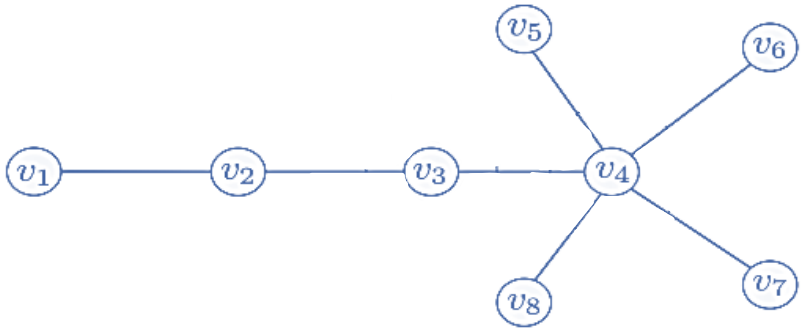
  Output: The Median of the graph G.

```
1. For i=1:1:n
2.       sum(i)← 0
3. For j=1:1:n
4.       if  (i ≠ j)
5.           sum(i) ← sum(i) + D(i,j)
6. Median ← 1
7. For i=1:1:n
8.        if(sum(i)<sum(Median))
9.            Median ← i
```

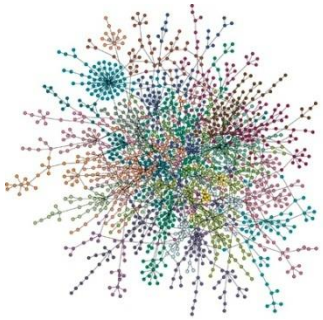
Paths and Cycles …



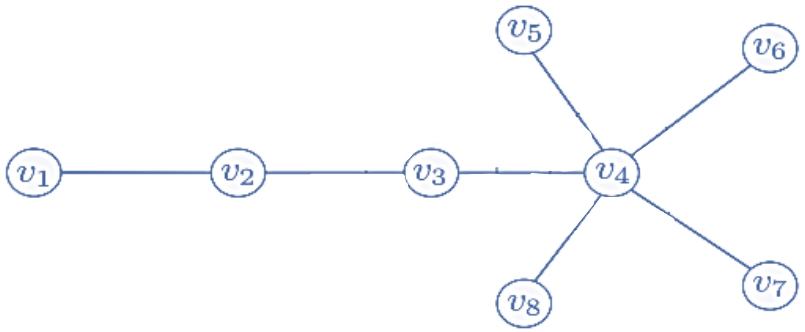
- **Graph Center and Graph Median**
  - The **Median** and the **Center** of a graph G are not always the same.



Paths and Cycles …

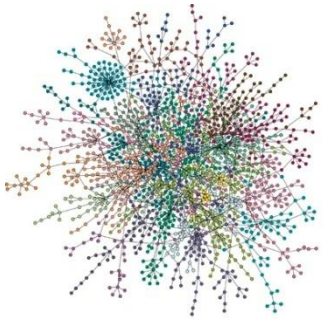


- **Graph Center and Graph Median**
  - The **Median** and the **Center** of a graph G are not always the same.

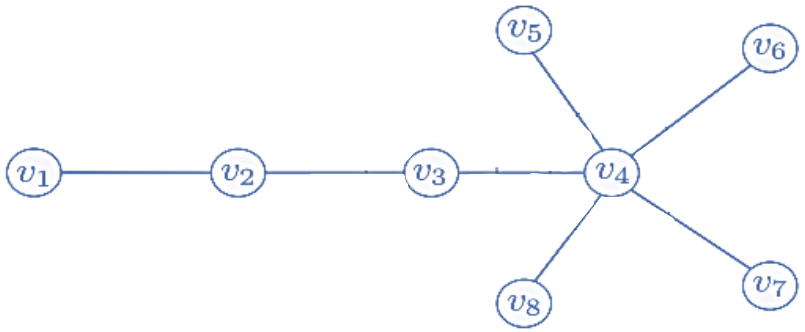


$e(v_1) = 4$
$e(v_2) = 3$
$e(v_3) = 2$
$e(v_4) = 3$
$e(v_5) = 4$
$e(v_6) = 4$
$e(v_7) = 4$
$e(v_8) = 4$

Paths and Cycles …



- **Graph Center and Graph Median**
  - The **Median** and the **Center** of a graph G are not always the same.



$dist(v_1) = 0+1+2+3+4+4+4+4 = 22$

…

$dist(v_4) = 3+2+1+0+1+1+1+1 = 10$

$dist(v_5) = 4+3+2+1+0+2+2+2 = 16$

…

$e(v_1) = 4$

$e(v_2) = 3$

$e(v_3) = 2$

$e(v_4) = 3$

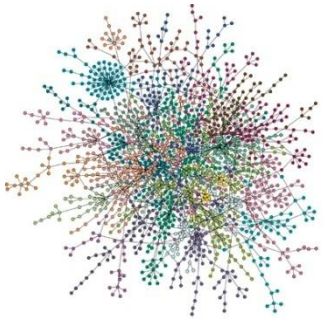
$e(v_5) = 4$

$e(v_6) = 4$

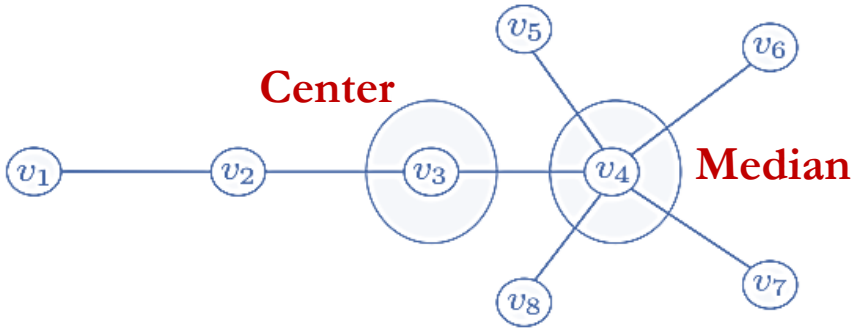
$e(v_7) = 4$

$e(v_8) = 4$

Paths and Cycles …



- **Graph Center and Graph Median**
  - The **Median** and the **Center** of a graph G are not always the same.



$dist(v_1) = 0+1+2+3+4+4+4+4 = 22$

…

$dist(v_4) = 3+2+1+0+1+1+1+1 = 10$

$dist(v_5) = 4+3+2+1+0+2+2+2 = 16$

…

$e(v_1) = 4$

$e(v_2) = 3$

$e(v_3) = 2$

$e(v_4) = 3$

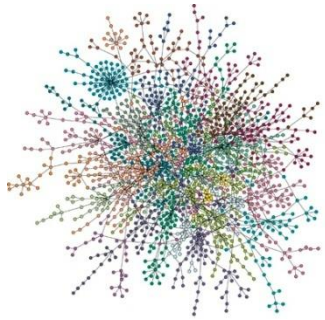
$e(v_5) = 4$

$e(v_6) = 4$

$e(v_7) = 4$

$e(v_8) = 4$

Paths and Cycles …



Diameter of a Graph

- The Diameter of a graph is defined to be the longest shortest path exhibited across all possible pairs of vertices.

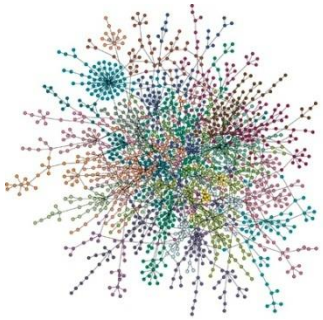
Algorithm Diameter Computation

Input: A graph $G = (V, E)$ with vertices $1,2, \ldots, n$

Output: The value of the Diameter of the Graph.

```
1.  Set MaxLength ← 0
2.  For i=1:1:n
3.      dist(i) ← ∞ and flag(i) ← false
4.  For i=1:1:n
5.  temp ← 0 , dist(i) ← ∞ and flag(i) ← true
6.  Enqueue(Q,i)
7.  while Q ≠ isnull();
8.          u ← Dequeue(Q)
9.          ∀ (u, v) ∈ E(G)
10.            if flag(v)=false
11.               flag(v) ← true and dist(v) ← dist(u)+1
12.               if dist(v) > temp
13.                  temp ← dist(v)
14.                  Enqueue(Q,v)
15. if MaxLenth < temp
16.    MaxLength ← temp
```

Paths and Cycles …

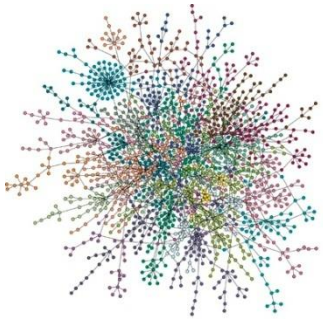


Centrality

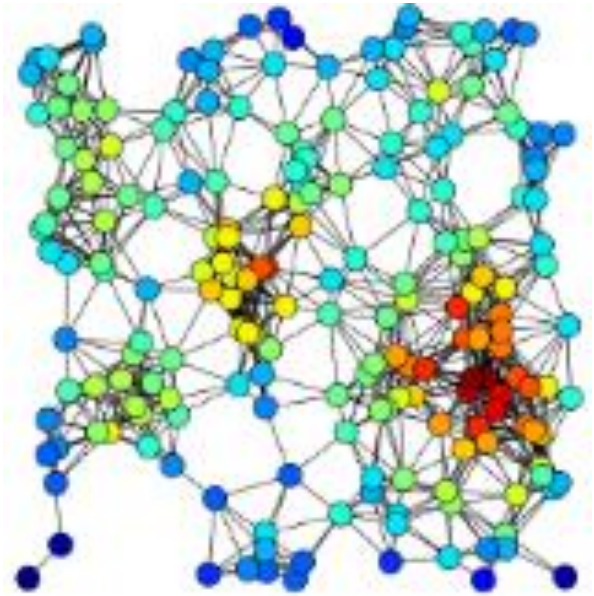
- In applications of Graph Theory, the term **Centrality** identify the most important vertices within a graph.
- Centrality indices are answers to the question "What characterizes an important vertex?“, while the word "importance" has a wide number of meanings, leading to many different definitions of centrality.
- Regarding the criteria deployed to measure centrality, it can be categorized to the following types:

1. Degree Centrality
2. Closeness Centrality
3. Betweeness Centrality

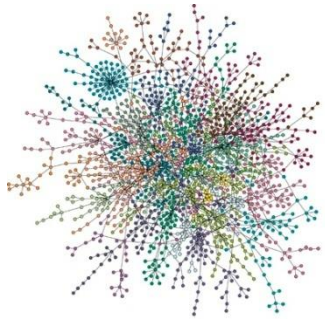
Paths and Cycles …



- **Centrality**
  - The term of **Degree Centrality** is defined as the number of links incident upon a node (i.e., the number of ties that a node has).
  - The degree can be interpreted in terms of the immediate risk of a node for catching whatever is flowing through the network (such as a virus, or some information).



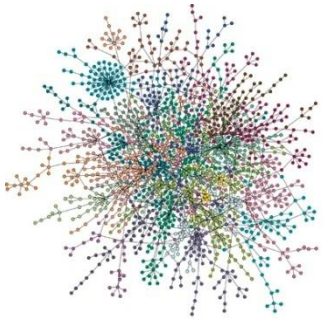
Paths and Cycles …



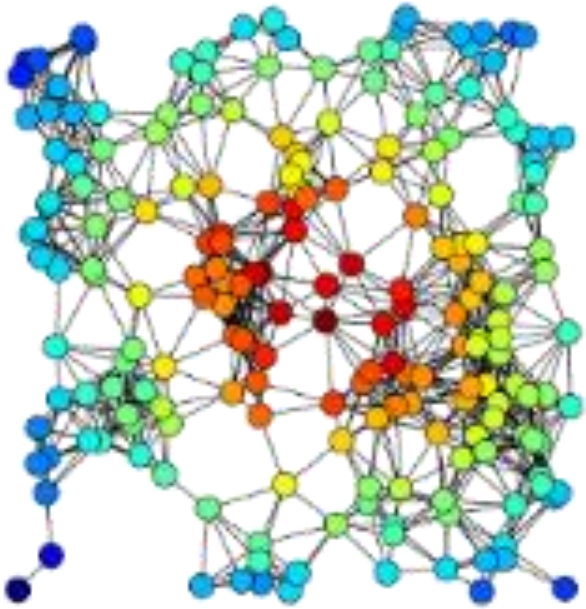
- **Centrality**
  - The term of **Degree Centrality** is defined as the number of links incident upon a node (i.e., the number of ties that a node has).
  - The degree can be interpreted in terms of the immediate risk of a node for catching whatever is flowing through the network (such as a virus, or some information).
  - The **Degree Centrality** of a vertex v, for a given graph $G = (V, E)$ with $|V|$ vertices and $|E|$ edges, is defined as:

$$C_D(v) = \deg(v)$$

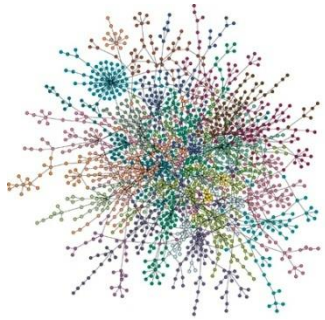
Paths and Cycles …



- **Centrality**
  - In a connected graph, the normalized **Closeness Centrality** of a node is the average length of the shortest path between the node and all other nodes in the graph.
  - The more central a node is, the closer it is to all other nodes.



Paths and Cycles …

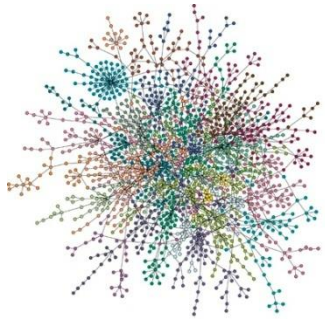


- **Centrality**
  - In a connected graph, the normalized **Closeness Centrality** of a node is the average length of the shortest path between the node and all other nodes in the graph.
  - The more central a node is, the closer it is to all other nodes.
  - **Closeness Centrality** is computed as:

$$C(x) = \frac{1}{\sum_y d(y,x)}$$

    where d(y,x) is the distance between vertices x and y.
  - In order to adjust the formula to compare graphs of different sizes, the normalized form is computed, given by the previous formula multiplied by N-1, where N is the number of nodes in the graph.

Paths and Cycles …



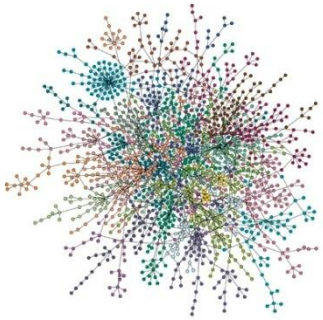
- **Centrality**
  - In a connected graph, the normalized **Closeness Centrality** of a node is the average length of the shortest path between the node and all other nodes in the graph.
  - The more central a node is, the closer it is to all other nodes.
  - **Closeness Centrality** is computed as:

$$C(x) = \frac{1}{\sum_y d(y,x)}$$

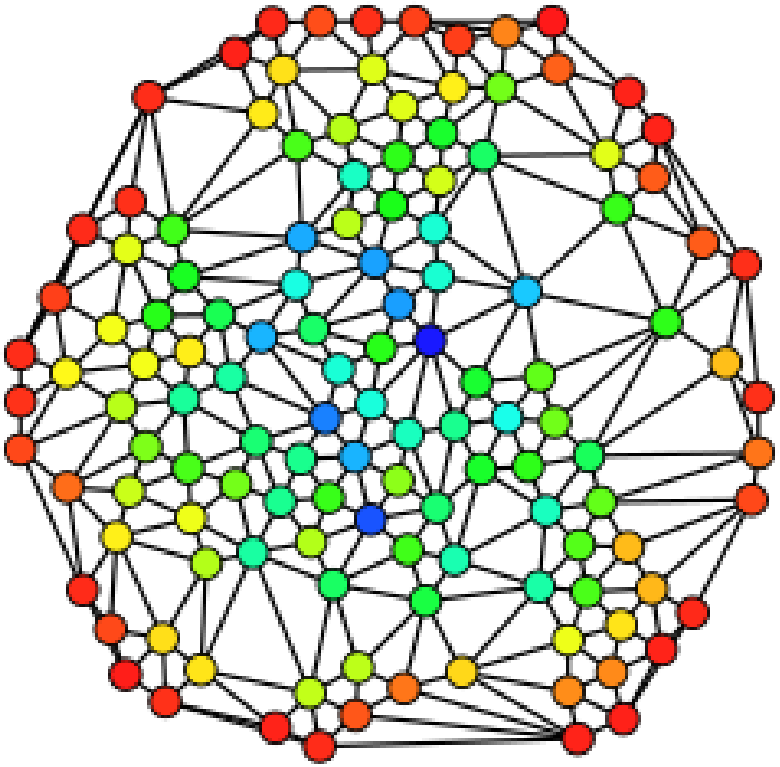
where d(y,x) is the distance between vertices x and y.

  - Taking distances from or to all other nodes is irrelevant in undirected graphs, whereas it can produce totally different results in directed graphs.

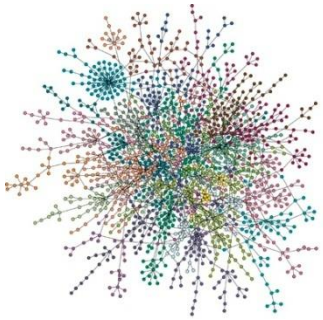
Paths and Cycles …



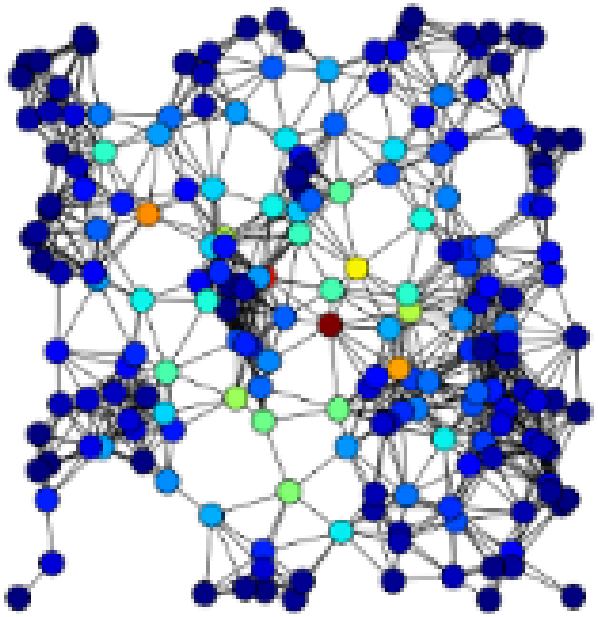
- **Centrality**
  - The term **Betweenness Centrality** quantifies the number of times a node acts as a bridge along the shortest path between two other nodes.



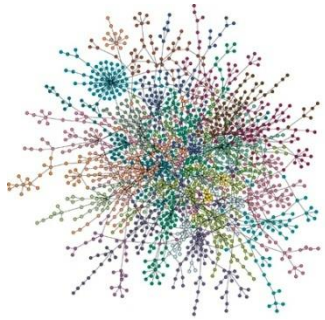
Paths and Cycles …



- **Centrality**
  - The term **Betweenness Centrality** quantifies the number of times a node acts as a bridge along the shortest path between two other nodes.
  - Under this concept, vertices that have a high probability to occur on a randomly chosen shortest path between two randomly chosen vertices have a high betweenness.



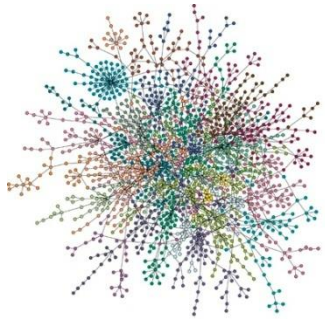
Paths and Cycles …



Centrality

- The term **Betweenness Centrality** quantifies the number of times a node acts as a bridge along the shortest path between two other nodes.
- Under this concept, vertices that have a high probability to occur on a randomly chosen shortest path between two randomly chosen vertices have a high betweenness.
- The **Betweenness Centrality** of a vertex v in a graph G=(V,E) with V vertices is computed as follows:
  1. For each pair of vertices (s,t), compute the shortest paths between them.
  2. For each pair of vertices (s,t), determine the fraction of shortest paths that pass through the vertex in question (here, vertex v).
  3. Sum this fraction over all pairs of vertices (s,t).

Paths and Cycles …



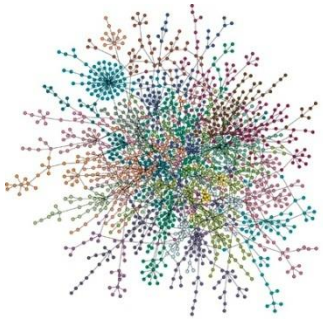
Centrality

- The term **Betweenness Centrality** quantifies the number of times a node acts as a bridge along the shortest path between two other nodes.
- More compactly the betweenness can be represented as

$$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

- where σ_{st} is total number of shortest paths from node s to node t and $\sigma_{st}(v)$ is the number of those paths that pass through v.
- The betweenness may be normalised by dividing through the number of pairs of vertices not including v, which:
  - for directed graphs is $(n-1)(n-2)\}(n-1)(n-2)$ and
  - for undirected graphs is $(n-1)(n-2)/2\}(n-1)(n-2)/2$.

Paths and Cycles …

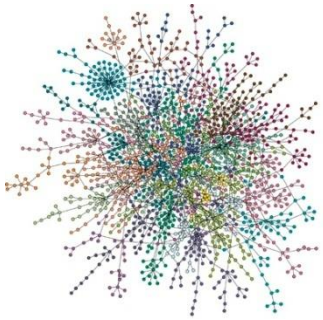


- **Centrality**
  - The term **Betweenness Centrality** quantifies the number of times a node acts as a bridge along the shortest path between two other nodes.
  - More compactly the betweenness can be represented as

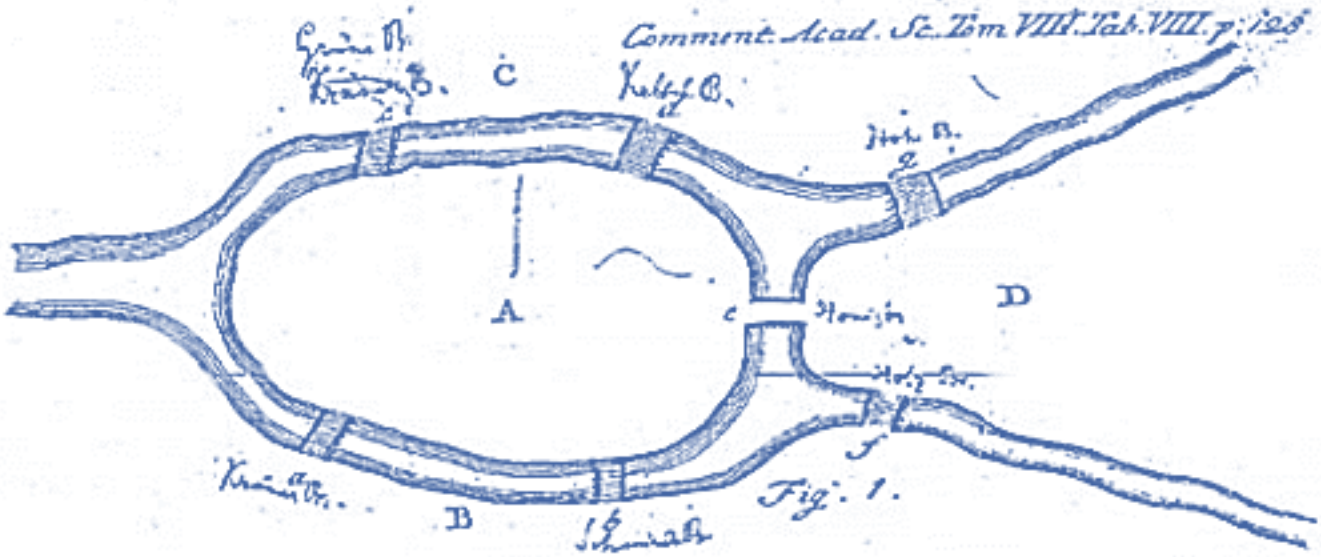
$$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

  - where σ_{st} is total number of shortest paths from node s to node t and $\sigma_{st}(v)$ is the number of those paths that pass through v.
  - For example, in an undirected star graph, the center vertex (which is contained in every possible shortest path) would have a betweenness of $(n-1)(n-2)/2\}(n-1)(n-2)/2$ (1, if normalised) while the leaves (which are contained in no shortest paths) would have a betweenness of 0.

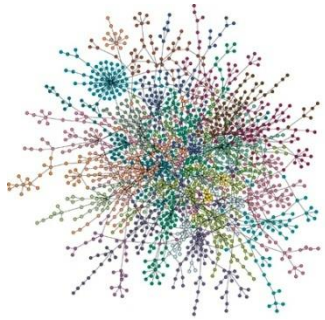
Paths and Cycles …



- **Eulerian Graphs**
  - The **Seven Bridges of Königsberg** is a historically notable problem in mathematics. Its negative resolution by **Leonhard Euler** in 1736 laid the foundations of graph theory and prefigured the idea of topology.



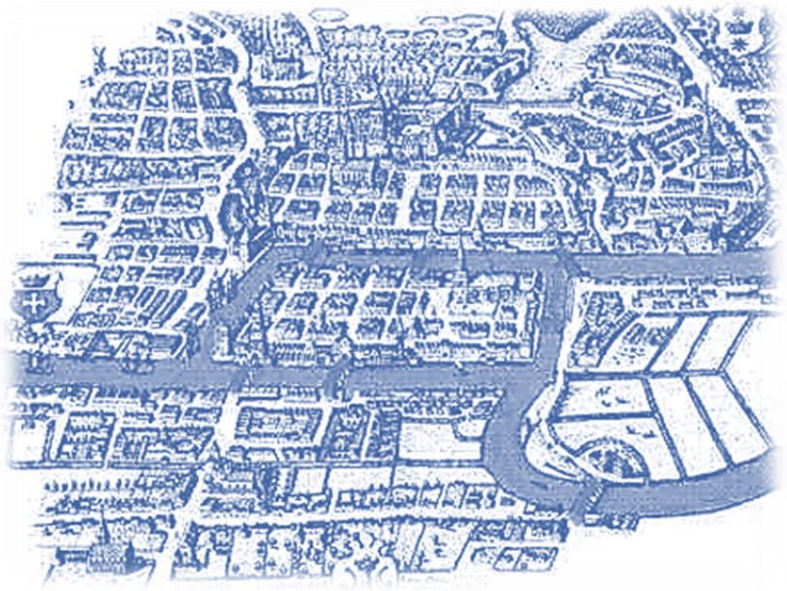
Paths and Cycles …



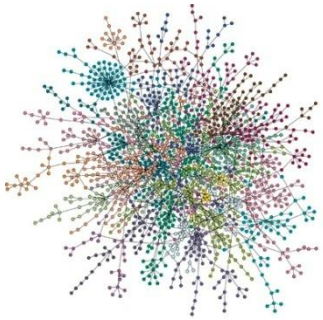
Eulerian Graphs

- The city of Königsberg in Prussia (now Kaliningrad, Russia) was set on both sides of the Pregel River, and included two large islands - Kneiphof and Lomse - which were connected to each other, or to the two mainland portions of the city, by seven bridges.

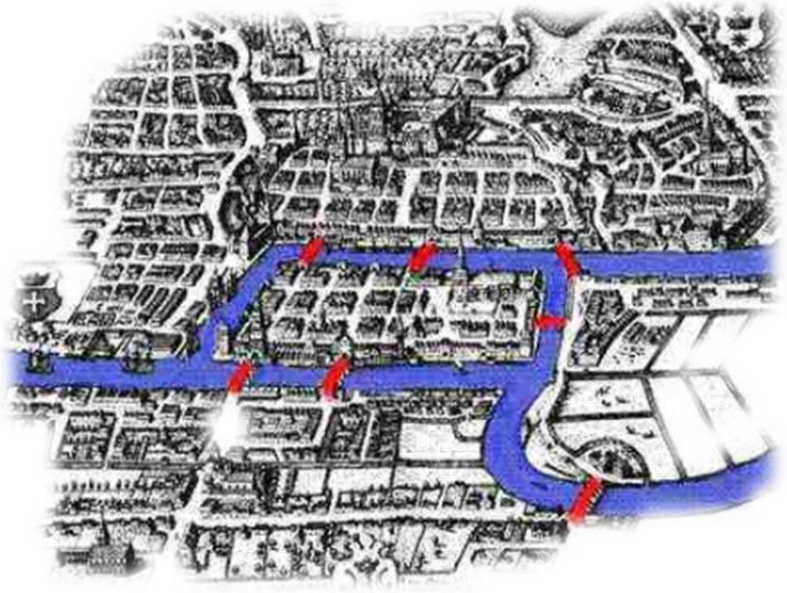
- **The problem was to devise a walk through the city that would cross each of those bridges once and only once**.



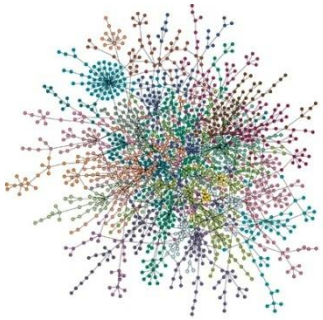
Paths and Cycles …



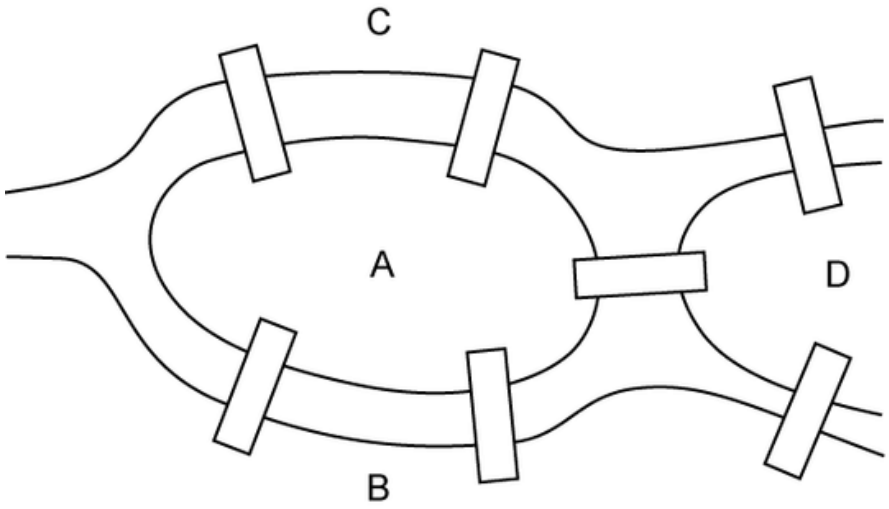
- **Eulerian Graphs**
  - By way of specifying the logical task unambiguously, explicitly unacceptable solutions involved either:
    a) reaching an island/mainland bank other than via one of the bridges, or
    b) accessing any bridge without crossing to its other end are.

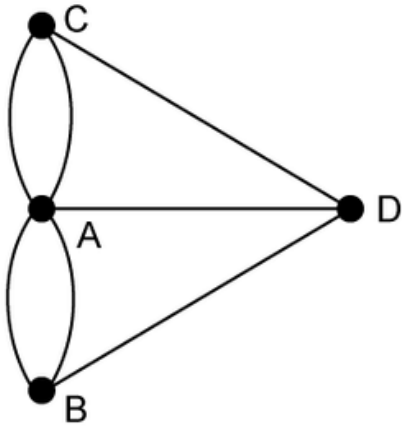


Paths and Cycles …

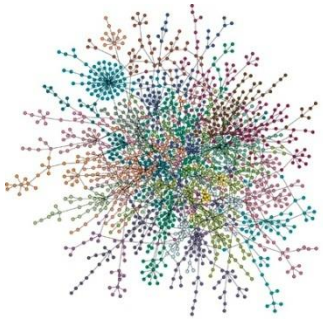


- **Eulerian Graphs**
  - Euler proved that the problem has no solution. The difficulty he faced was the development of a suitable technique of analysis, and of subsequent tests that established this assertion with mathematical rigor.



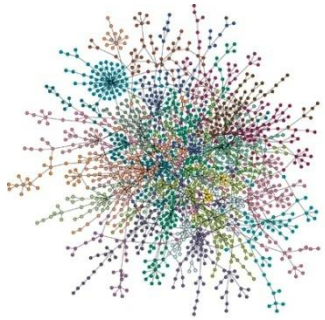


Paths and Cycles …



- **Eulerian Graphs**
  - First, Euler pointed out that the choice of route inside each land mass is irrelevant.
  - The only important feature of a route is the sequence of bridges crossed.
  - This allowed him to reformulate the problem in abstract terms (laying the foundations of graph theory), eliminating all features except the list of land masses and the bridges connecting them.

Paths and Cycles …

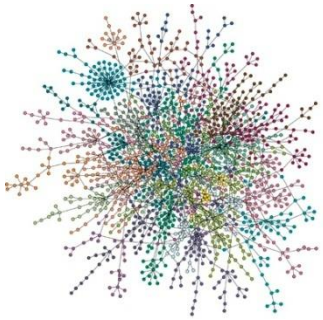


- **Eulerian Graphs**
  - First, Euler pointed out that the choice of route inside each land mass is irrelevant.
  - The only important feature of a route is the sequence of bridges crossed.
  - This allowed him to reformulate the problem in abstract terms (laying the foundations of graph theory), eliminating all features except the list of land masses and the bridges connecting them.

He replaced …

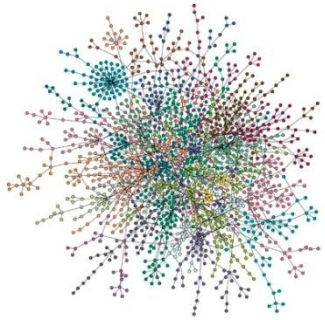
each land mass with an abstract "vertex" or node, and each bridge with an abstract connection, an "edge", which only serves to record which pair of vertices is connected by that bridge.

Paths and Cycles …



- **Eulerian Graphs**
  - First, Euler pointed out that the choice of route inside each land mass is irrelevant.
  - The only important feature of a route is the sequence of bridges crossed.
  - This allowed him to reformulate the problem in abstract terms (laying the foundations of graph theory), eliminating all features except the list of land masses and the bridges connecting them.

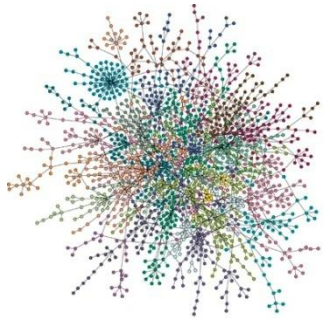
Paths and Cycles …



Eulerian Graphs

- Next, Euler observed that (except at the endpoints of the walk), whenever one enters a vertex by a bridge, one leaves the vertex by a bridge.
  - During any walk in the graph, the number of times one enters a non-terminal vertex equals the number of times one leaves it.
  - If every bridge has been traversed exactly once, it follows that, for each land mass (except start/finish), the number of bridges touching that land mass must be even (half "toward" and half "away" from the landmass).
  - All four of the land masses in the original problem are touched by an odd number of bridges (one is touched by 5 bridges, and each of the other three is touched by 3).
  - Since, at most, two land masses can serve as the endpoints of a walk, the proposition of a walk traversing each bridge once leads to a contradiction.

Paths and Cycles …

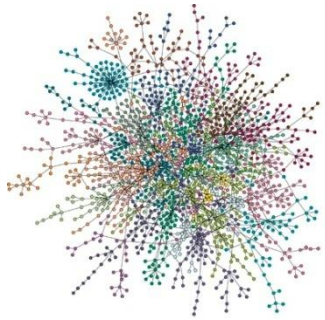


Eulerian Graphs

- Is it possible in a graph to find a closed trace (circuit) crossing all the edges of the graph, exactly one time?
- **Theorem - Eulerian Graph:** Include closed trace (circuit)
  - A graph G is Eulerian iff it includes exactly 0 vertices of odd degree.
- **Theorem - Semi-Eulerian Graphs:** Include open trace (path)
  - A graph G is Semi-Eulerian iff it include exactly 2 vertices of odd degree.

- How can it be effectively checked if a graph is Eulerian?

Paths and Cycles …

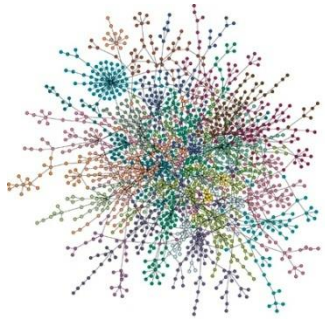


Eulerian Graphs

- Is it possible in a graph to find a closed trace (circuit) crossing all the edges of the graph, exactly one time?
- **Theorem - Eulerian Graph:** Include closed trace (circuit)
  - A graph G is Eulerian iff it includes exactly 0 vertices of odd degree.
- **Theorem - Semi-Eulerian Graphs:** Include open trace (path)
  - A graph G is Semi-Eulerian iff it include exactly 2 vertices of odd degree.

- How can it be effectively checked if a graph is Eulerian?
  - Deploying DFS and the Theorem
  - In linear time $O(n+m)$

Paths and Cycles …

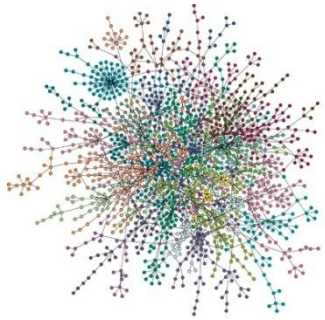


Eulerian Graphs

- Is it possible in a graph to find a closed trace (circuit) crossing all the edges of the graph, exactly one time?
- **Theorem - Eulerian Graph:** Include closed trace (circuit)
  - A graph G is Eulerian iff it includes exactly 0 vertices of odd degree.
- **Theorem - Semi-Eulerian Graphs:** Include open trace (path)
  - A graph G is Semi-Eulerian iff it include exactly 2 vertices of odd degree.

- **Theorem:** A simple connected graph G=(V,E) is Eulerian iff each of its edges belongs to an odd number of cycles.

Paths and Cycles …



Eulerian Graphs

- **Theorem:** A simple connected graph G is Eulerian (resp., Semi-Eulerian) *iff* it includes exactly 0 (resp., exactly 2) vertices of odd degree.

  Proof.

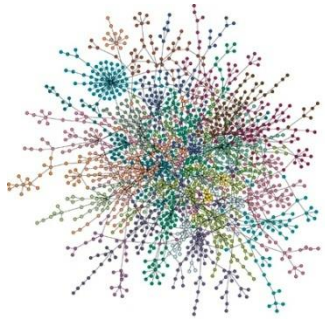
  Necessity of Condition:

  If there exists Eulerian Circuit (resp., open trace), then G should be connected and the number of odd-degree vertices should be 0 (resp., 2).

  Sufficiency of Condition: (*Inductively on the number of edges*)

  a) If $|E| = 2$, the truth is obvious. Let the proposition is truth for $|E| > 2$. Let us assume that graph Gcontains two vertices of odd degree (let, v_1, v_2). Let as assume a walk from vertex v_i (v_1 if there exist odd-degree vertices). Let the walk W cross various vertices until it reaches vertex v_j, without omitting the corresponding edges. If the graph G does not contain odd-degree edges, then $v_j = v_i$, else $v_j = v_2$.

Paths and Cycles …



Eulerian Graphs

- **Theorem:** A simple connected graph G is Eulerian (resp., Semi-Eulerian) *iff* it includes exactly 0 (resp., exactly 2) vertices of odd degree.

  Proof.

  Necessity of Condition:

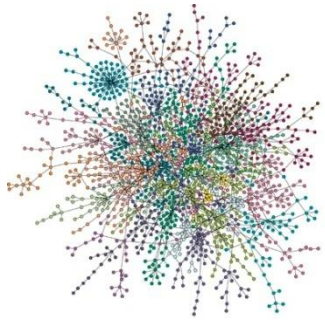
  If there exists Eulerian Circuit (resp., open trace), then G should be connected and the number of odd-degree vertices should be 0 (resp., 2).

  Sufficiency of Condition: (*Inductively on the number of edges*)

  b) However, it is not obvious if the walk W has crossed through all the vertices. If so, there exist edges that have not been utilized.

     1. If we remove the rest of the utilized edge, then a subgraph G' results, that is not necessarily connected.
     2. The graph G' should contain only even-degree vertices.

Paths and Cycles …



Eulerian Graphs

- **Theorem:** A simple connected graph G is Eulerian (resp., Semi-Eulerian) *iff* it includes exactly 0 (resp., exactly 2) vertices of odd degree.

  Proof.

  Necessity of Condition:

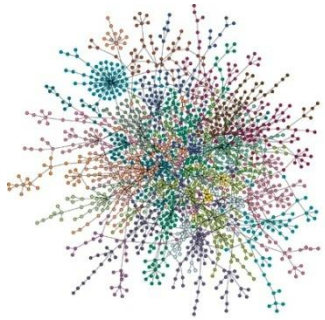
  If there exists Eulerian Circuit (resp., open trace), then G should be connected and the number of odd-degree vertices should be 0 (resp., 2).

  Sufficiency of Condition: (*Inductively on the number of edges*)

  b) However, it is not obvious if the walk W has crossed through all the vertices. If so, there exist edges that have not been utilized.

     3. According to the assumption of the induction, each component of G' contains an Eulerian circuit. Since graph G is connected, then the walk should pass from at least one vertex of each component of G'.

Paths and Cycles …



Eulerian Graphs

- **Theorem:** A simple connected graph G is Eulerian (resp., Semi-Eulerian) *iff* it includes exactly 0 (resp., exactly 2) vertices of odd degree.

  Proof.

  Necessity of Condition:

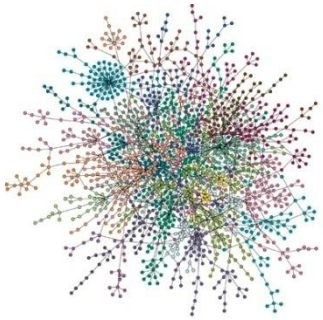
  If there exists Eulerian Circuit (resp., open trace), then G should be connected and the number of odd-degree vertices should be 0 (resp., 2).

  Sufficiency of Condition: (*Inductively on the number of edges*)

  b) However, it is not obvious if the walk W has crossed through all the vertices. If so, there exist edges that have not been utilized.

     4. It can be constructed an Eulerian circuit (or open trace) for the graph G by introducing the Eulerian circuits of the components of the subgraph G' to the walk W.

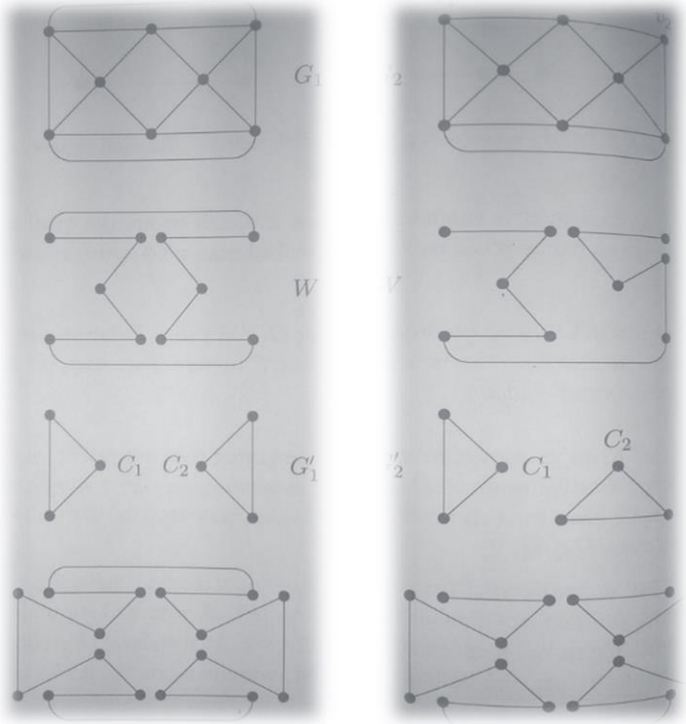
Paths and Cycles …



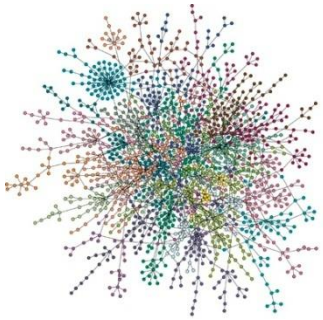
Eulerian Graphs

- **Theorem :** A simple connected graph G is Eulerian (resp., Semi-Eulerian) *iff* it includes exactly 0 (resp., exactly 2) vertices of odd degree.

  Proof.of the Sufficiency of Condition: (*Inductively on the number of edges*)

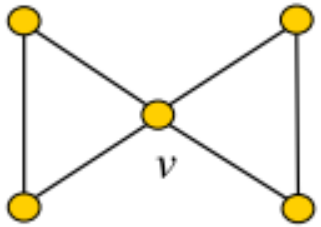


Paths and Cycles …

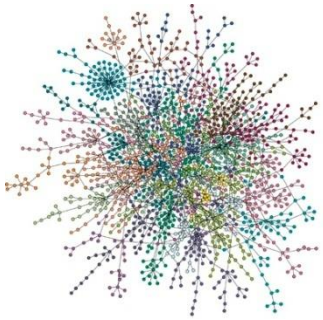


Discovering Eulerian Circuits in Graphs

- Let us assume that we follow a naive approach of discovering an Eulerian circuit in a given graph,
  - starting from an arbitrary vertex and
  - proceeding over the edges that have not been examined yet.
- … such an approach does not guarantee that it can discover the circuit!
- A graph is called **arbitrarily traceable** from vertex v, if it is guaranteed that we can draw a closed trace based on the previous naive approach starting from vertex v.

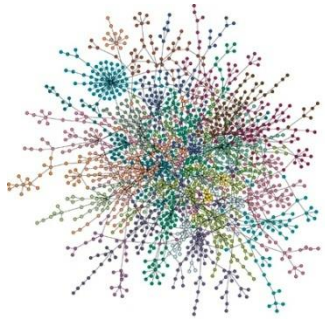


Paths and Cycles …



- **Discovering Eulerian Circuits in Graphs**
  - Algorithms to discover Eulerian circuits in graphs.
    - Hierholtzer (1873): Concatenation of circuits
    - Fleury (1883): Gradual extension of tract T_i, avoiding bridges in the subgraph $G - E(T_i)$ unless there is no other option.
    - Tucker (1976): Vertex split as to form individual cycles and concatenation of cycles.

Paths and Cycles …



Discovering Eulerian Circuits in Graphs

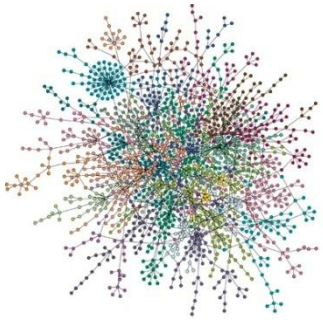
- Hierholtzer

  Input: A connected graph $G = (V, E)$ with vertices of even degree

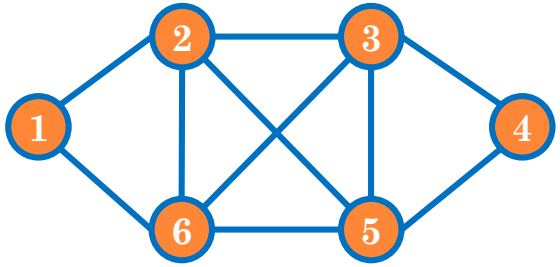
  Output: An Eulerian circuit of graph G.

1. **Select a vertex $v \in V$ creating circuit C_0 selecting edges $\notin C_0$**
2. **Set $i \leftarrow 0$**
3. **If $E(C_i) = E(G)$**
   $C = C_i$ is an Eulerian Circuit
4. **Else**
5. **Select a vertex v_i of C_i adjacent to an edge in C_i and proceed by constructing another circuit C'_i starting wit vertex v_i inside the subgraph $G - E(C_i)$**
6. **From the circuits C_i and C'_i , construct a hyper-circuit C_{i+1} starting from vertex v_{i-1} crossing circuit C_i and continuing to circuit C'_i frinishing to vertex v_i**
7. **$i \leftarrow i + 1$**
8. **Goto step 3.**

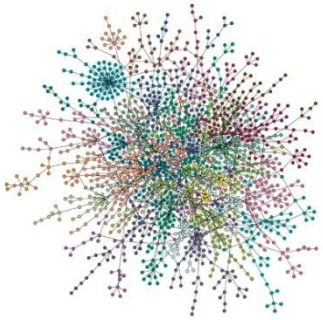
Paths and Cycles …



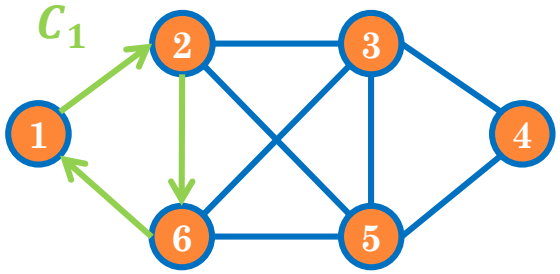
- **Discovering Eulerian Circuits in Graphs**
  - Hierholtzer



Paths and Cycles …

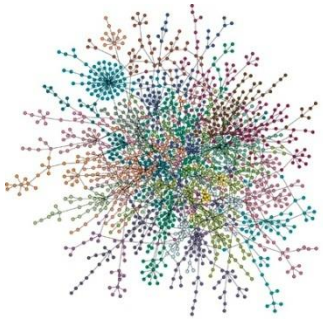


- **Discovering Eulerian Circuits in Graphs**
  - Hierholtzer

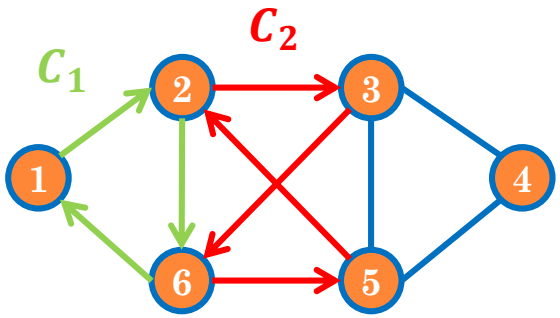


- $C_1: 1 \rightarrow 2 \rightarrow 6 \rightarrow 1$

Paths and Cycles …

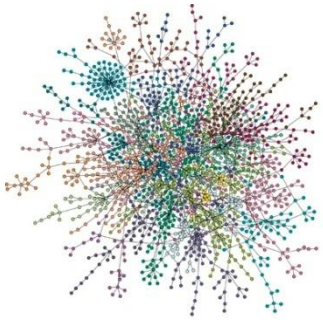


- **Discovering Eulerian Circuits in Graphs**
  - Hierholtzer

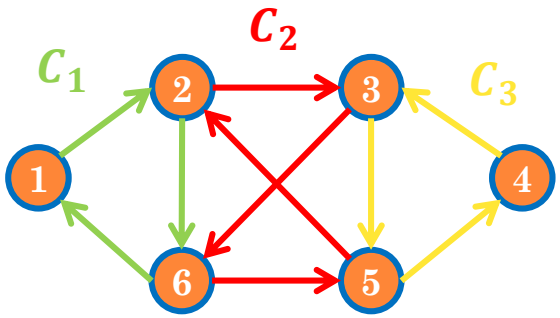


- $C_1: 1 \rightarrow 2 \rightarrow 6 \rightarrow 1$
- $C_2: 2 \rightarrow 3 \rightarrow 6 \rightarrow 5 \rightarrow 2$

Paths and Cycles …

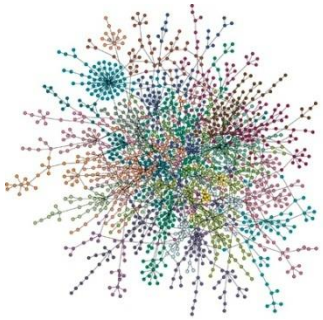


- **Discovering Eulerian Circuits in Graphs**
  - Hierholtzer

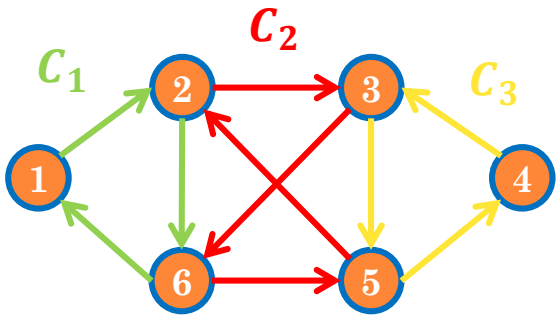


- $C_1: 1 \rightarrow 2 \rightarrow 6 \rightarrow 1$
- $C_2: 2 \rightarrow 3 \rightarrow 6 \rightarrow 5 \rightarrow 2$
- $C_3: 3 \rightarrow 5 \rightarrow 4 \rightarrow 3$

Paths and Cycles …

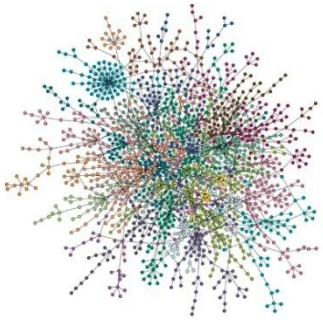


- **Discovering Eulerian Circuits in Graphs**
  - Hierholtzer

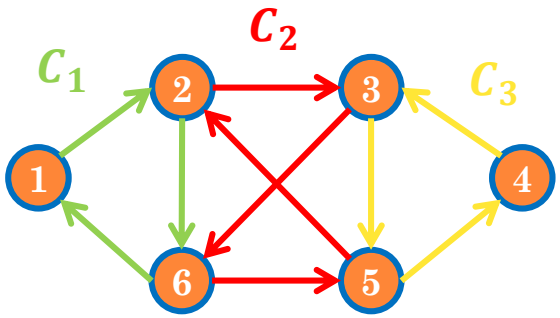


- C_1: $1 \rightarrow 2 \rightarrow 6 \rightarrow 1$
- C_2: $2 \rightarrow 3 \rightarrow 6 \rightarrow 5 \rightarrow 2$
- C_3: $3 \rightarrow 5 \rightarrow 4 \rightarrow 3$

Paths and Cycles …



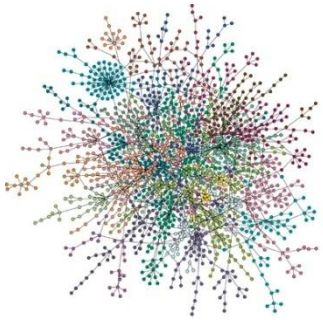
- **Discovering Eulerian Circuits in Graphs**
  - Hierholtzer



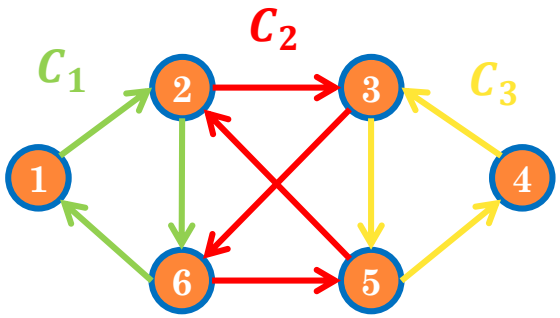
Select a vertex in $v \in C_i$
that is adjacent to edge $\notin C_i$

- C_1: $1 \rightarrow 2 \rightarrow 6 \rightarrow 1$
- C_2: $2 \rightarrow 3 \rightarrow 6 \rightarrow 5 \rightarrow 2$
- C_3: $3 \rightarrow 5 \rightarrow 4 \rightarrow 3$

Paths and Cycles …



- **Discovering Eulerian Circuits in Graphs**
  - Hierholtzer

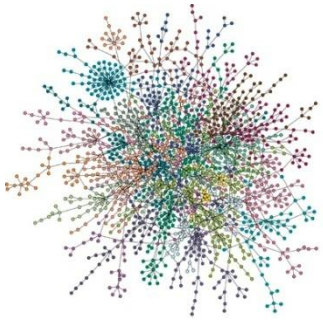


- C_1: 1 → 2 → 6 → 1
- C_2: 2 → 3 → 6 → 5 → 2
- C_3: 3 → 5 → 4 → 3

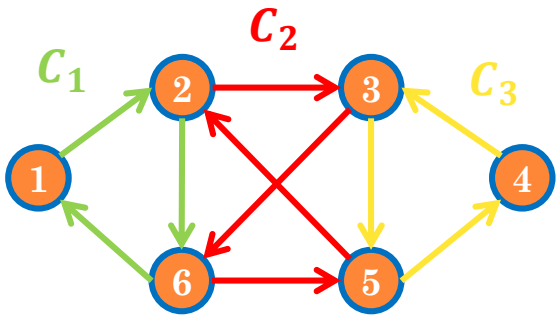
Select a vertex in $v \in C_i$
that is adjacent to edge $\notin C_i$

Merge consecutively $C' = C_1 + C_2 =$
$= 1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 5 \rightarrow 2 \rightarrow 6 \rightarrow 1$

Paths and Cycles …



- **Discovering Eulerian Circuits in Graphs**
  - Hierholtzer

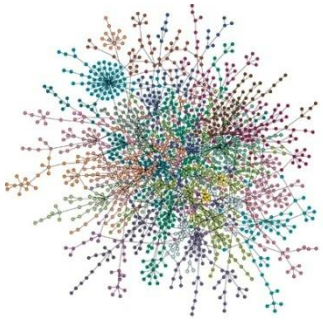


- C_1: $1 \rightarrow 2 \rightarrow 6 \rightarrow 1$
- C_2: $2 \rightarrow 3 \rightarrow 6 \rightarrow 5 \rightarrow 2$
- C_3: $3 \rightarrow 5 \rightarrow 4 \rightarrow 3$

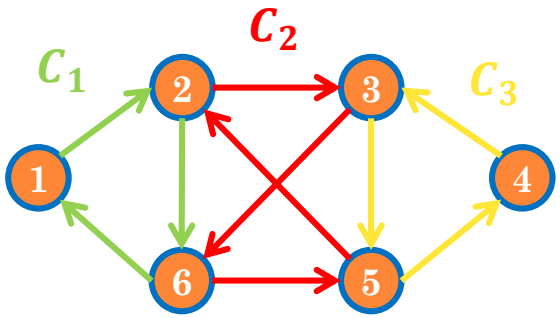
Select a vertex in $v \in C_i$
that is adjacent to edge $\notin C_i$

Merge consecutively $C' = C_1 + C_2 =$
$= 1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 5 \rightarrow 2 \rightarrow 6 \rightarrow 1$

Paths and Cycles …



- **Discovering Eulerian Circuits in Graphs**
  - Hierholtzer

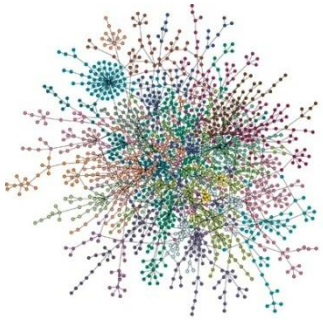


- C_1: 1 → 2 → 6 → 1
- C_2: 2 → 3 → 6 → 5 → 2
- C_3: 3 → 5 → 4 → 3

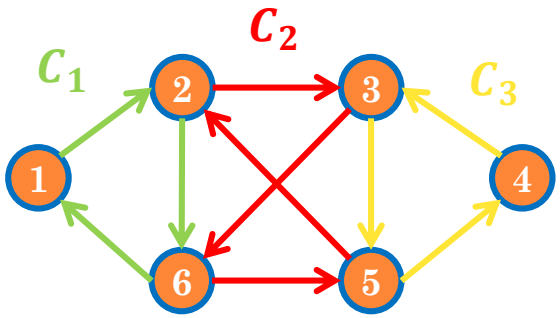
Select a vertex in $v \in C_i$
that is adjacent to edge $\notin C_i$

Merge consecutively $C' \mathrel{+}= C_3 =$
$=1 \to 2 \to 3 \to 5 \to 4 \to 3 \to 6 \to 5 \to 2 \to 6 \to 1$

Paths and Cycles …



- **Discovering Eulerian Circuits in Graphs**
  - Hierholtzer

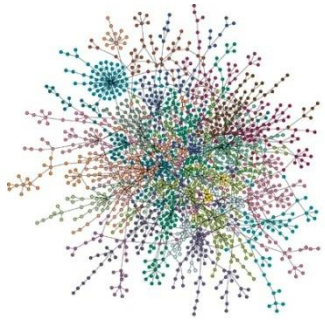


- C_1: $1 \rightarrow 2 \rightarrow 6 \rightarrow 1$
- C_2: $2 \rightarrow 3 \rightarrow 6 \rightarrow 5 \rightarrow 2$
- C_3: $3 \rightarrow 5 \rightarrow 4 \rightarrow 3$

Select a vertex in $v \in C_i$
that is adjacent to edge $\notin C_i$

Merge consecutively $C' \mathrel{+}= C_3 =$
$=1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 4 \rightarrow 3 \rightarrow 6 \rightarrow 5 \rightarrow 2 \rightarrow 6 \rightarrow 1$

Paths and Cycles …



Discovering Eulerian Circuits in Graphs

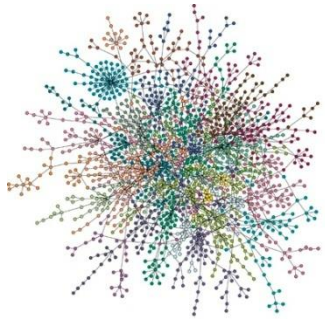
- Fleury

  Input: A connected graph $G = (V, E)$ with vertices of even degree

  Output: An Eulerian circuit of graph G.

1. **Select a vertex $v_0 \in V$ as first vertex of the trace**
2. **Set $T_0 \leftarrow (V_0)$ and $i \leftarrow 0$**
3. **Let the trace $T_i = (v_0, e_1, v_1, \ldots, e_i, v_i)$**
4. **From vertex v_i Select arbitrarily an edge $e_{i+1} \notin T_i$ that is not a bridge in the subgraph $G - E(T\_i)$, unless there is no other option.**
5. **Define path $T_{i+1} = (T_i, e_{i+1}, v_{i+1})$**
6. **$i \leftarrow i + 1$**
7. **If $(i = |E|)$**
   $C = T_i$ is an Eulerian circuit
8. **Else**
   Goto step 3.

Paths and Cycles …



Discovering Eulerian Circuits in Graphs

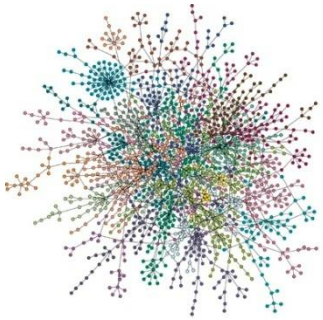
- Tucker

  Input: A connected graph $G = (V, E)$ with vertices of even degree

  Output: An Eulerian circuit of graph G.

1. **Split the vertices until there exist only vertices of degree "2"**
2. **Let G_1 the produced graph**
3. **$i \leftarrow 1$**
4. **Let c_1 the even number of components of graph G_1**
5. **If $c_i = 1$**
   $C = G_i$ is Eulerian circuit
6. **Else**
   Search two components T and T' of G_i having v_i as common vertex
7. **Construct circuit C_{i+1} starting from vertex v_i, crossing T and T' finishing to v_i**
8. **Define graph $G_{i+1} = G_i - \{T, T'\} \cup C_{i+1}$**
9. **$C\_(i+1)$ is a component of the subgraph G_{i+1}**
10. **$T = C_{i+1}$**
11. **$i \leftarrow i + 1$**
12. **Let c_i is the number of components of subgraph G_i**
13. **Goto step 5**

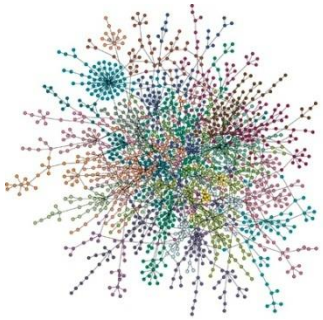
Paths and Cycles …



- **Discovering Eulerian Circuits in Graphs**
  - Tucker

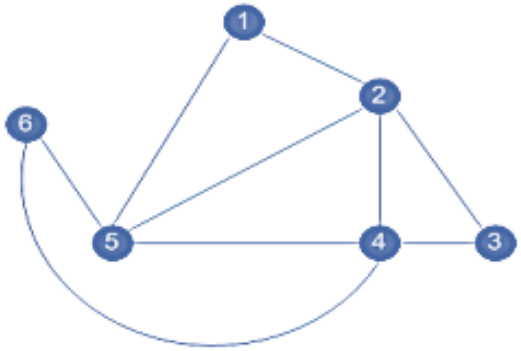
Concept: ***Split the vertices until there exist only vertices of degree "2"***

Paths and Cycles …

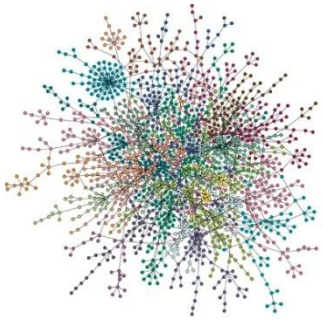


- **Discovering Eulerian Circuits in Graphs**
  - Tucker

Concept: ***Split the vertices until there exist only vertices of degree "2"***



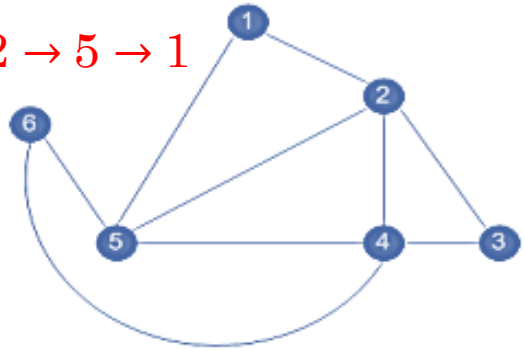
Paths and Cycles …



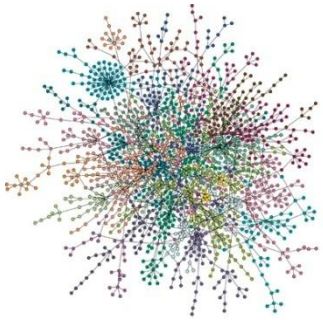
- **Discovering Eulerian Circuits in Graphs**
  - Tucker

Concept: ***Split the vertices until there exist only vertices of degree "2"***

$1 \rightarrow 2 \rightarrow 5 \rightarrow 1$



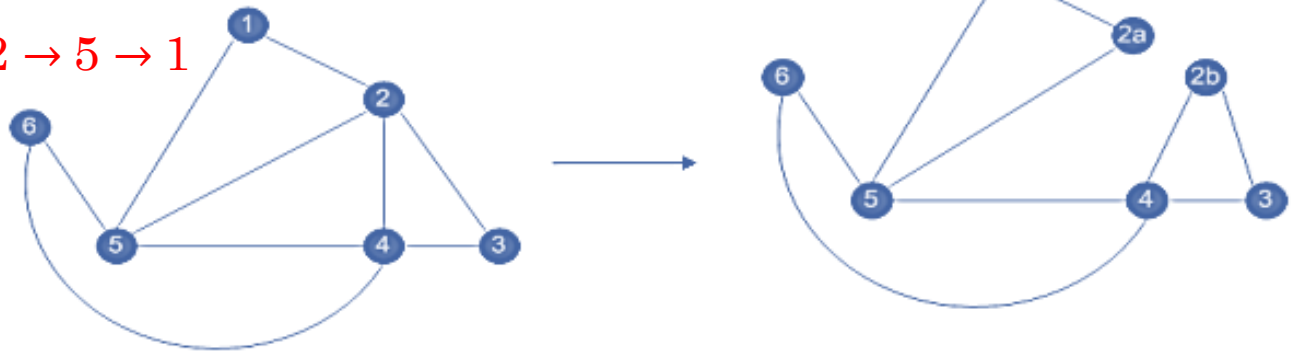
Paths and Cycles …



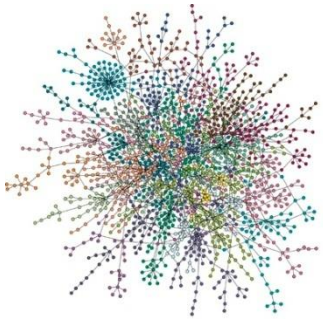
- **Discovering Eulerian Circuits in Graphs**
  - Tucker

Concept: ***Split the vertices until there exist only vertices of degree "2"***

$1 \rightarrow 2 \rightarrow 5 \rightarrow 1$



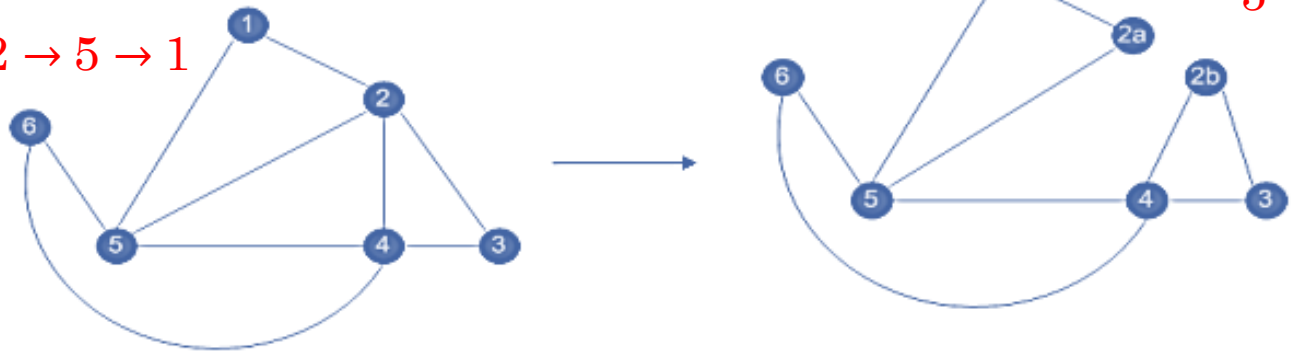
Paths and Cycles …



- **Discovering Eulerian Circuits in Graphs**
  - Tucker

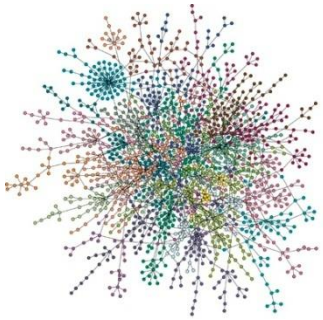
Concept: ***Split the vertices until there exist only vertices of degree "2"***

$1 \rightarrow 2 \rightarrow 5 \rightarrow 1$



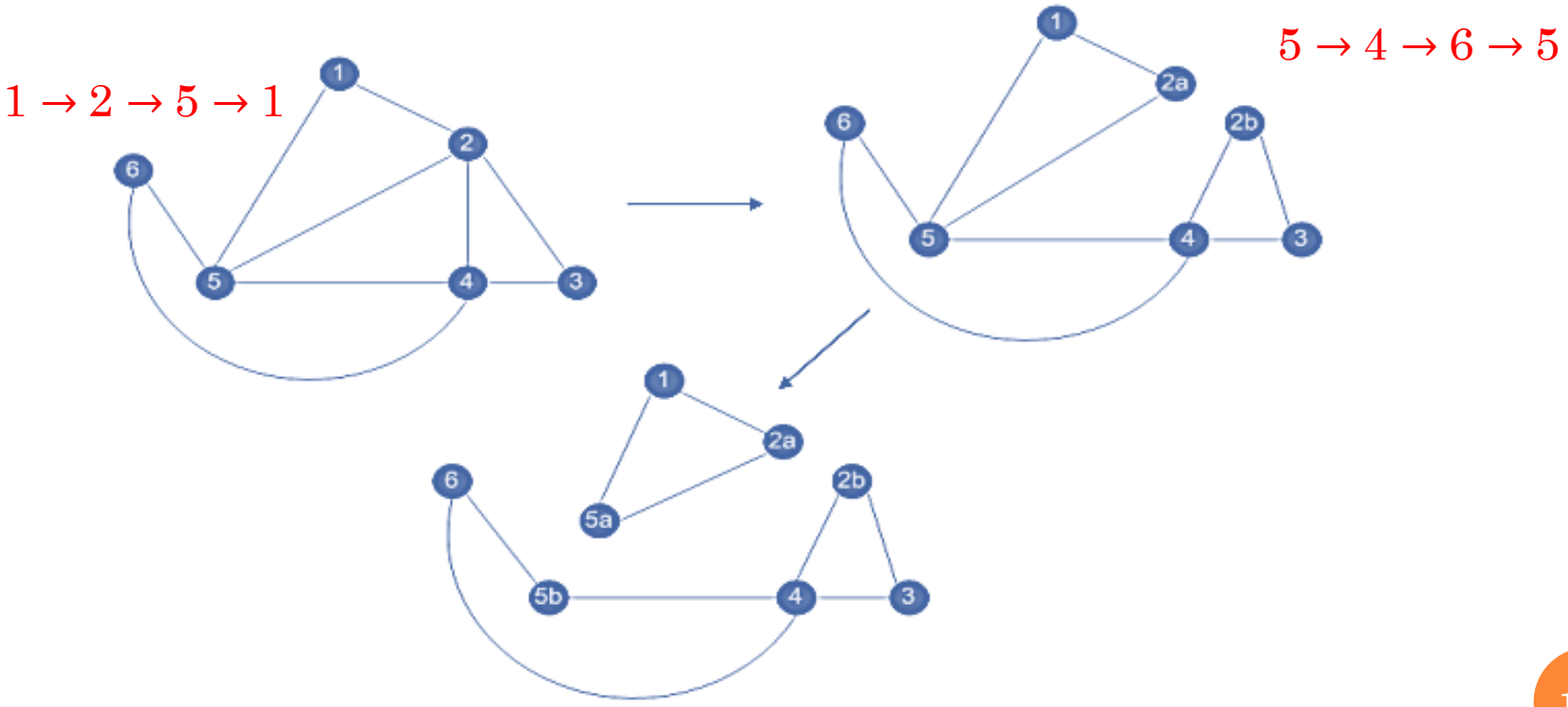
$5 \rightarrow 4 \rightarrow 6 \rightarrow 5$

Paths and Cycles …

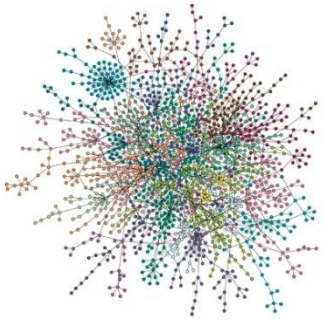


- **Discovering Eulerian Circuits in Graphs**
  - Tucker

Concept: ***Split the vertices until there exist only vertices of degree "2"***

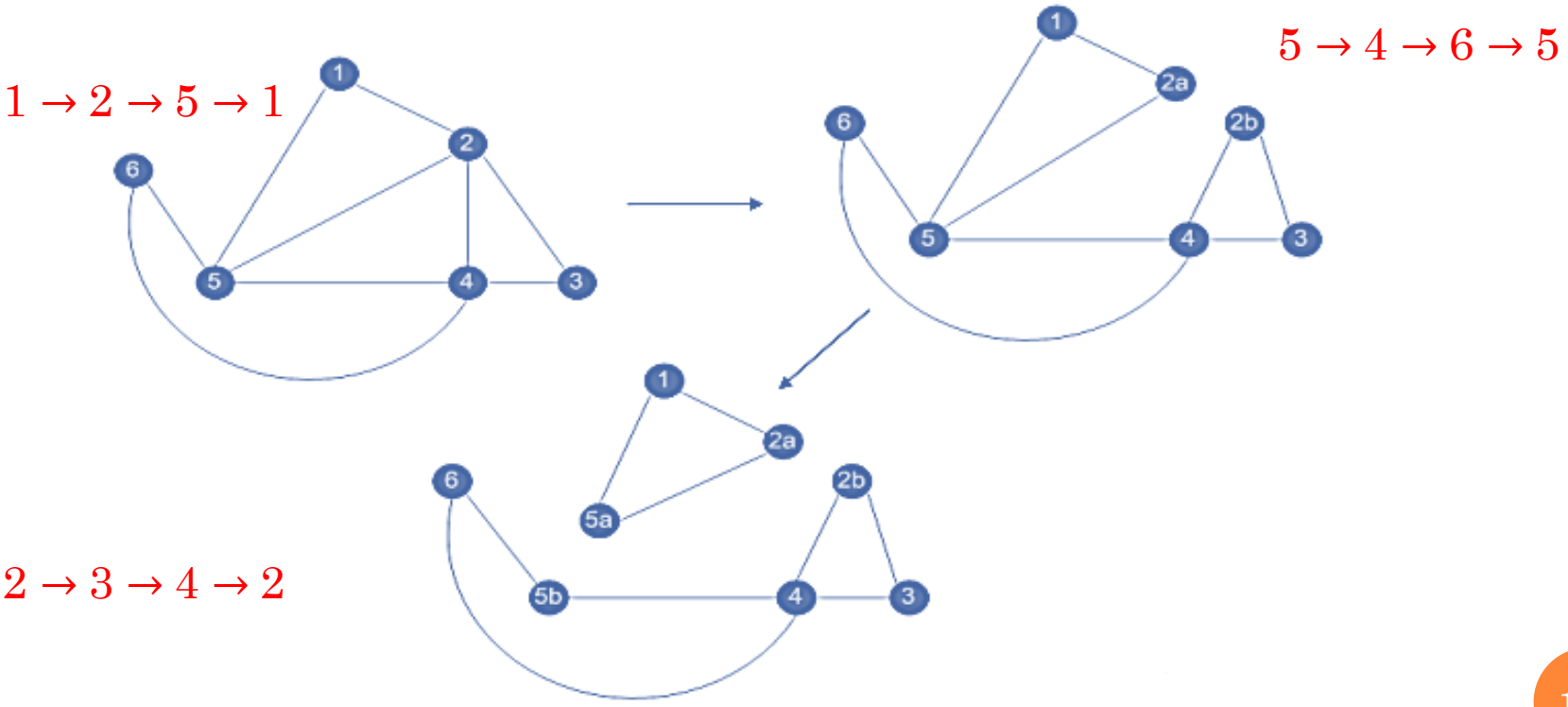


Paths and Cycles …

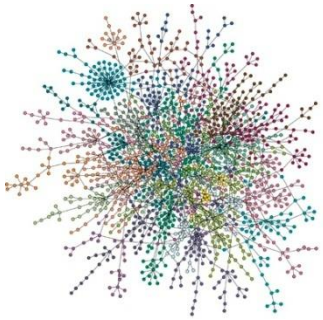


- **Discovering Eulerian Circuits in Graphs**
  - Tucker

Concept: ***Split the vertices until there exist only vertices of degree "2"***

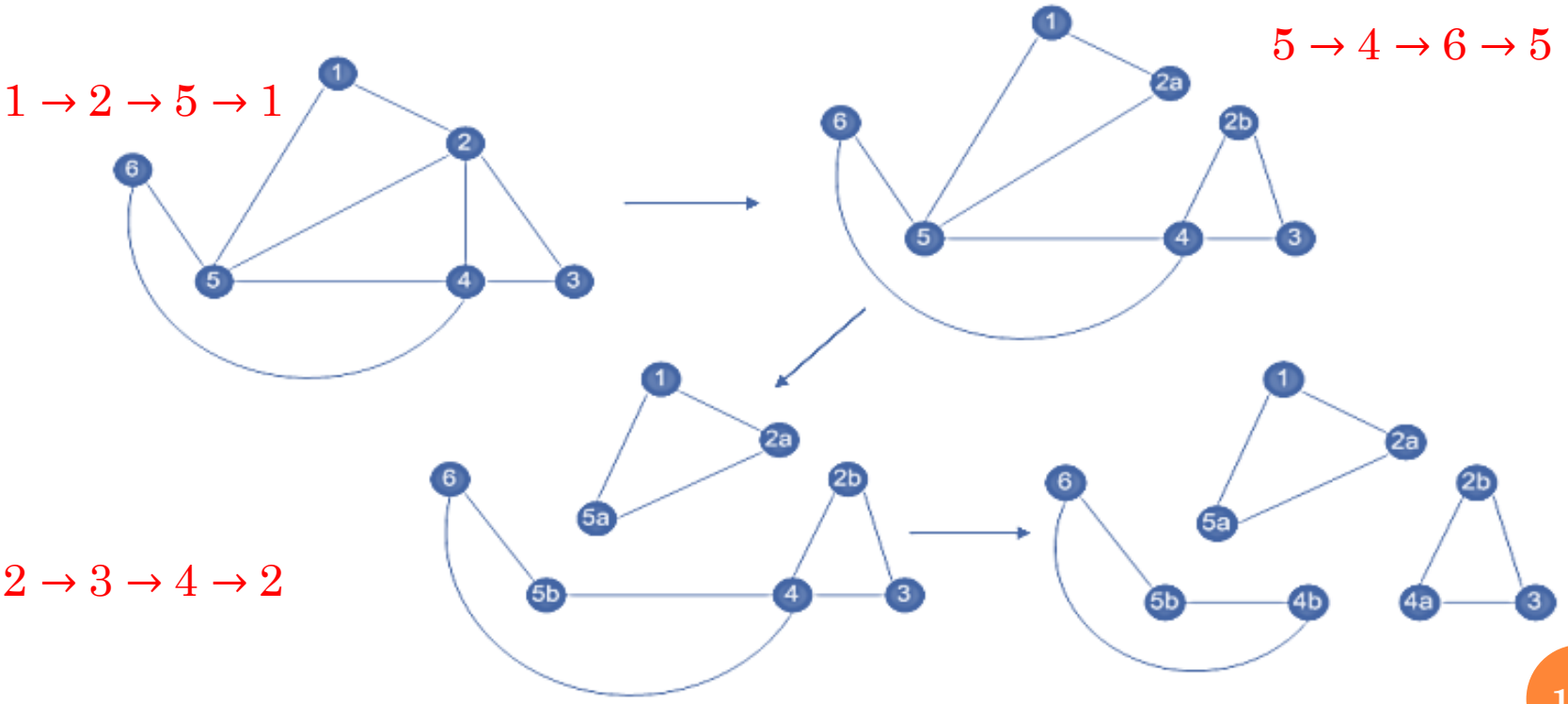


Paths and Cycles …

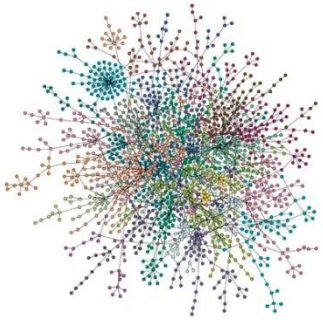


- **Discovering Eulerian Circuits in Graphs**
  - Tucker

Concept: ***Split the vertices until there exist only vertices of degree "2"***

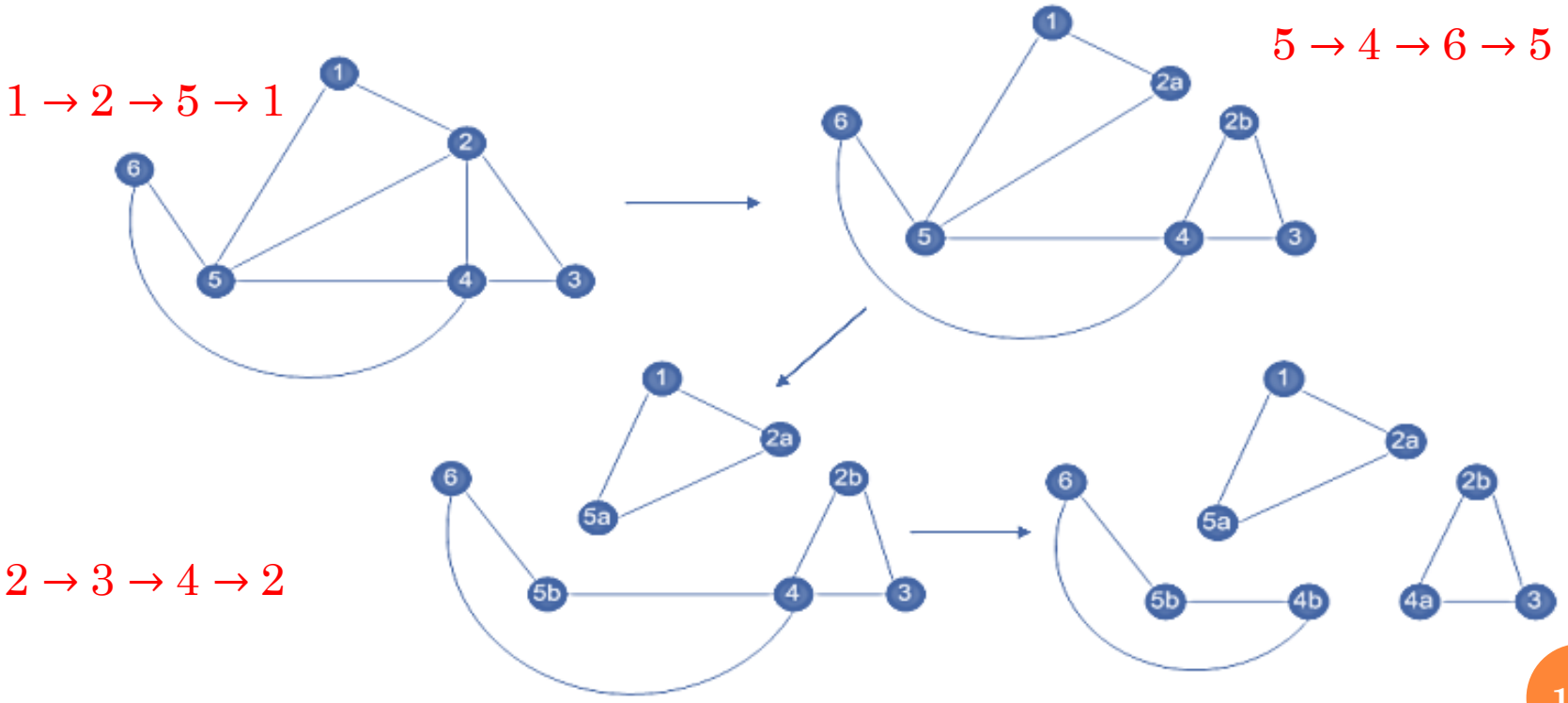


Paths and Cycles …



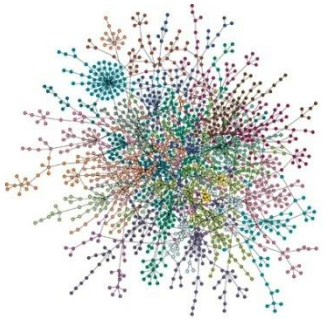
- **Discovering Eulerian Circuits in Graphs**
  - Tucker

Concept: ***Split the vertices until there exist only vertices of degree "2"***



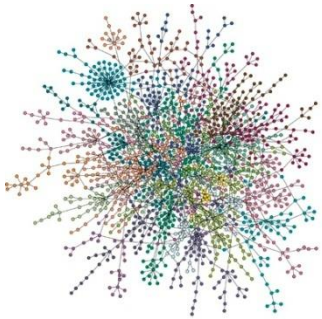
1 → 2 → 3 → 4 → 2 → 5 → 4 → 6 → 5

Paths and Cycles …



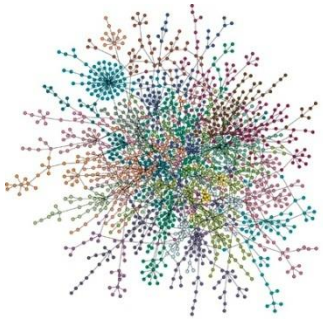
- **Chinese Post Office …**

Paths and Cycles …



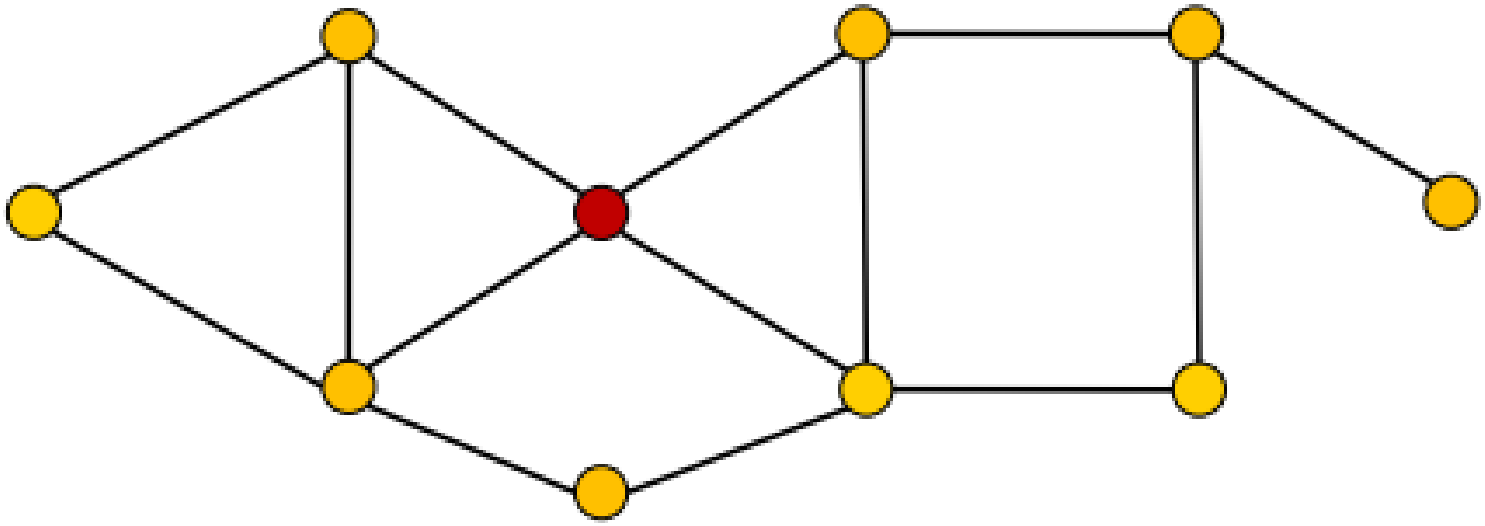
- **Chinese Postman …**

Paths and Cycles …

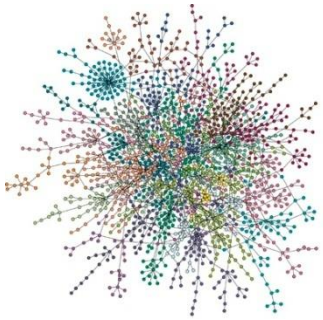


- **Chinese Postman Problem…**
  - The "CPP" Problem:

    A postman starting from his office has to cross all the streets and return to his office as soon as possible!

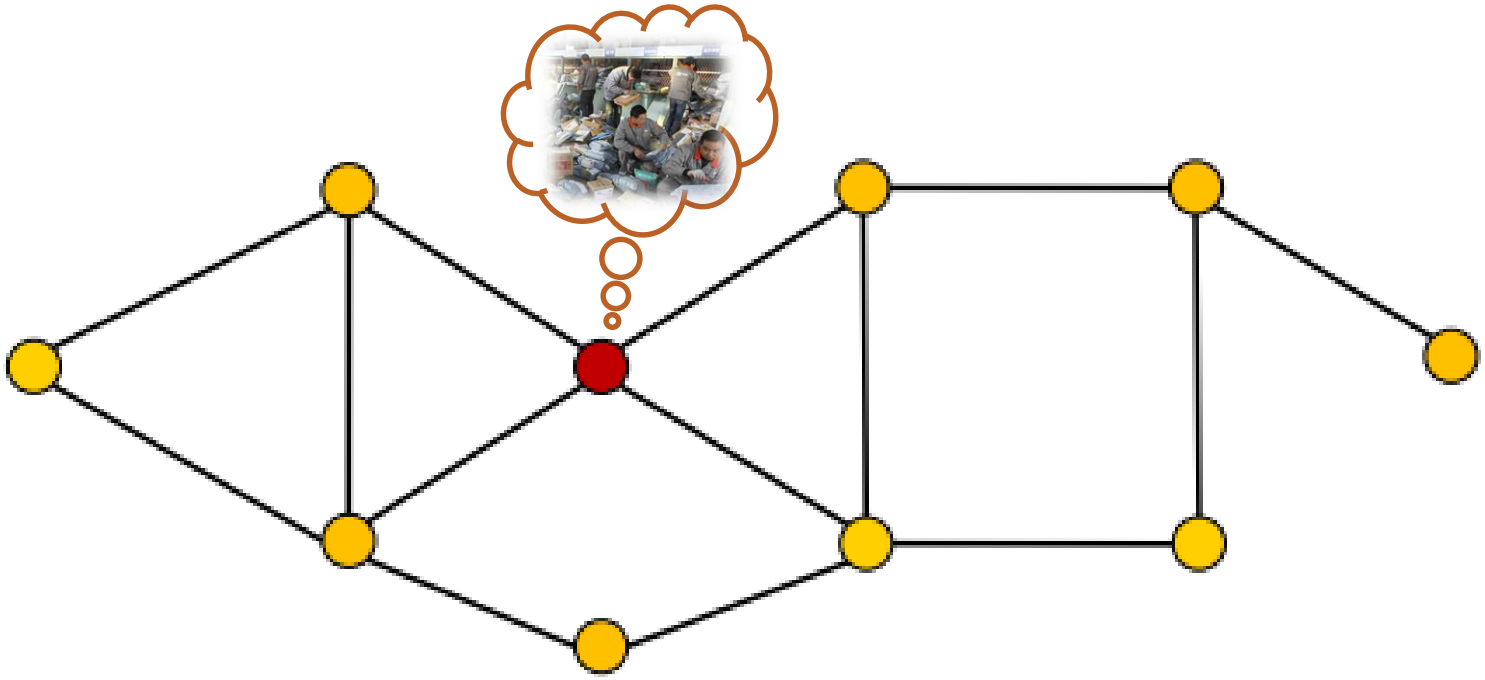


Paths and Cycles …

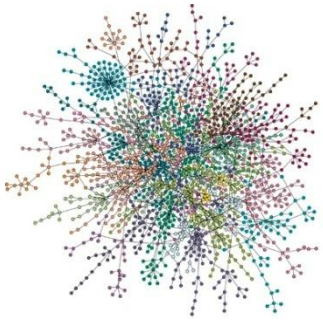


- **Chinese Postman Problem…**
  - The "CPP" Problem:

    A postman starting from his office has to cross all the streets and return to his office as soon as possible!

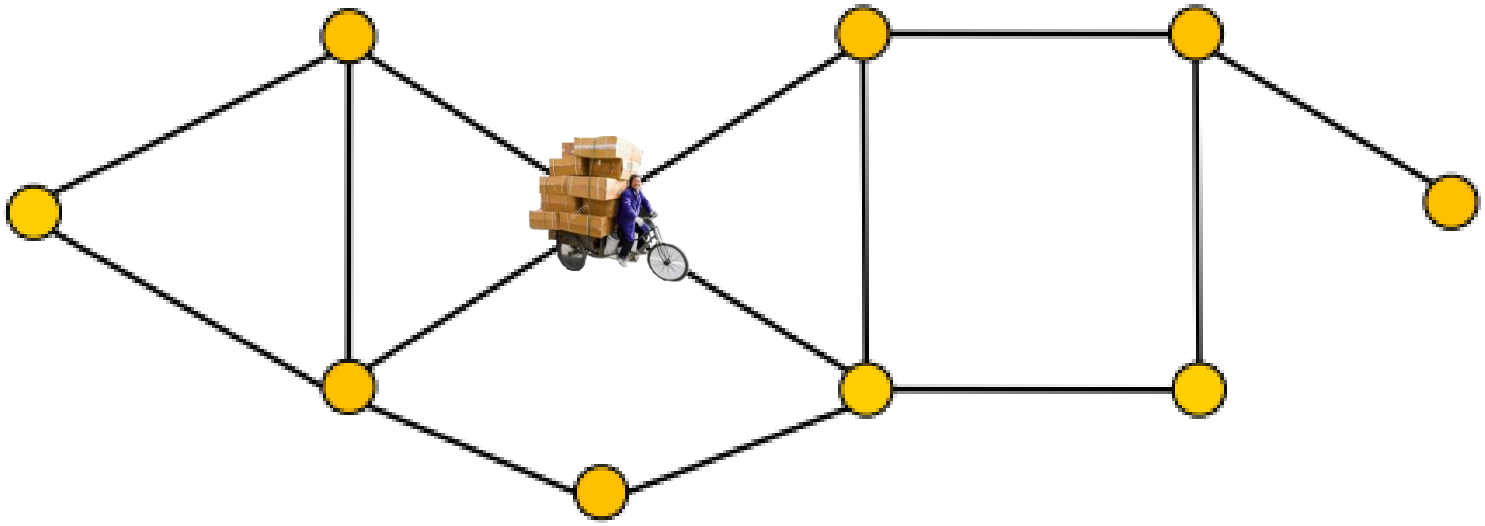


Paths and Cycles …

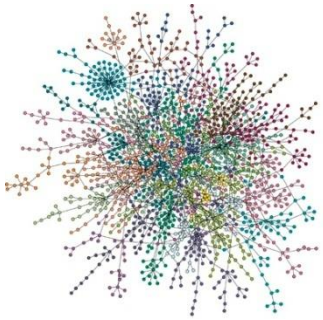


- **Chinese Postman Problem…**
  - The "CPP" Problem:

    A postman starting from his office has to cross all the streets and return to his office as soon as possible!

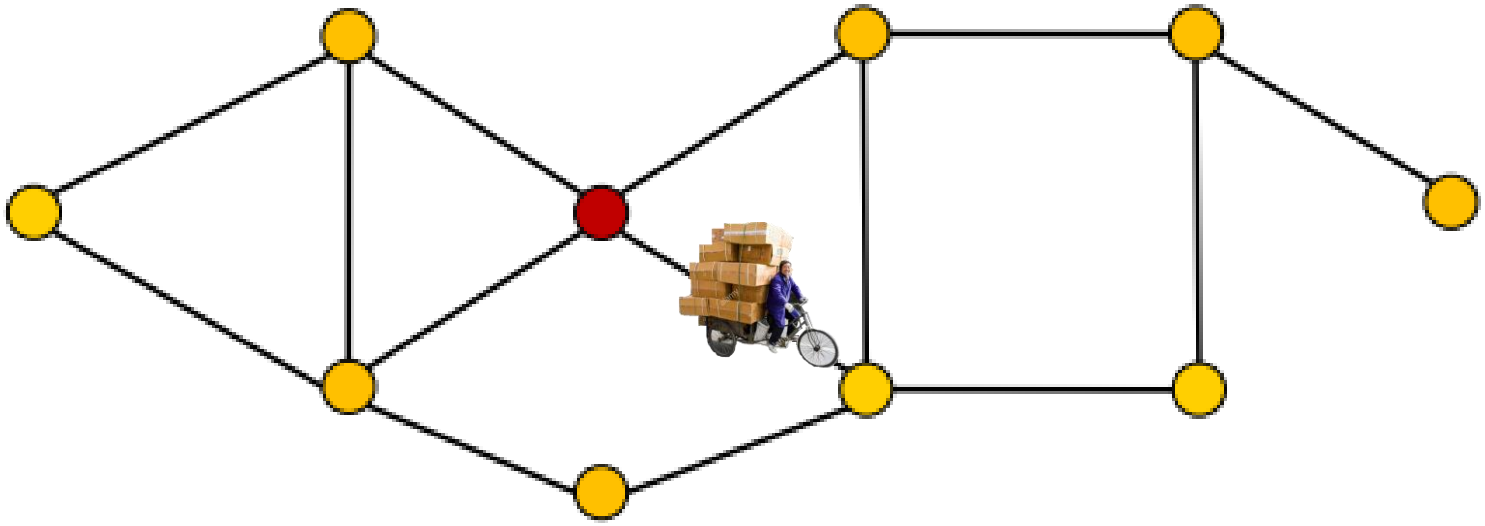


Paths and Cycles …

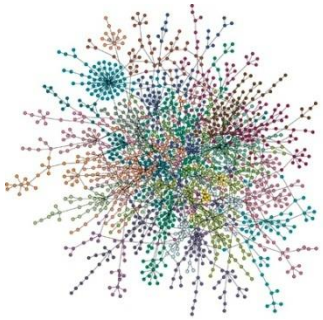


- **Chinese Postman Problem…**
  - The "CPP" Problem:

    A postman starting from his office has to cross all the streets and return to his office as soon as possible!

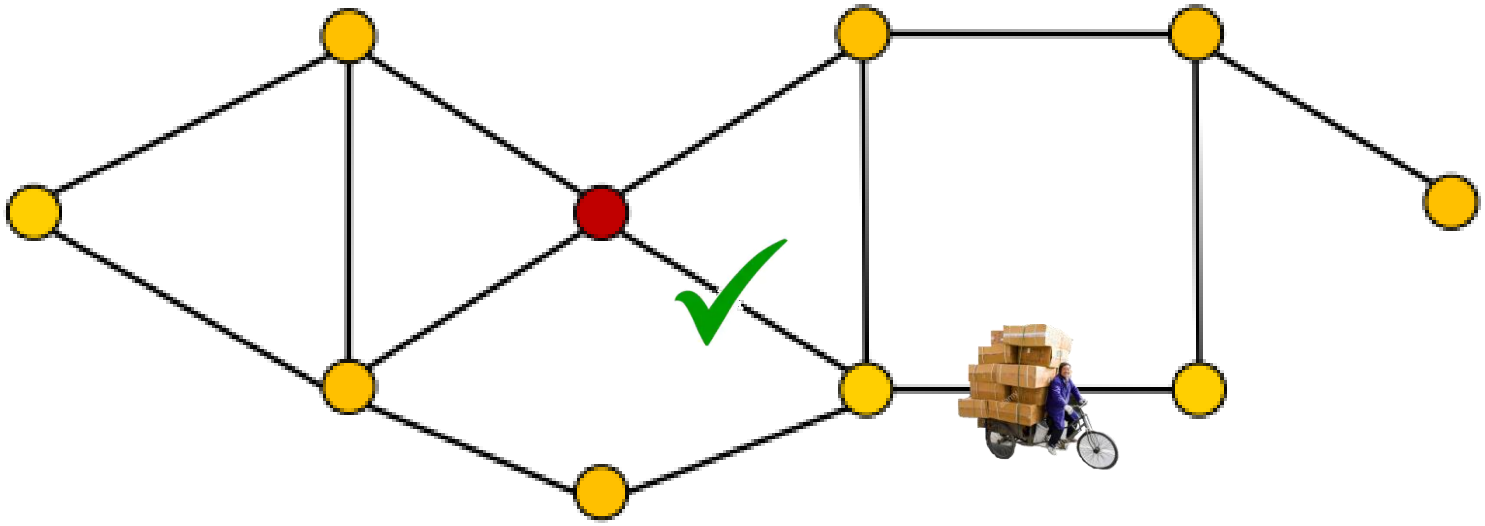


Paths and Cycles …

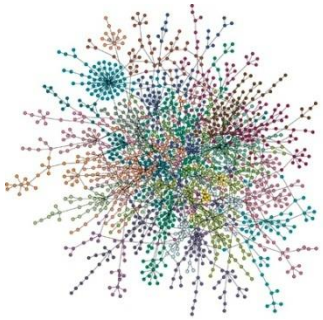


- **Chinese Postman Problem…**
  - The "CPP" Problem:

    A postman starting from his office has to cross all the streets and return to his office as soon as possible!

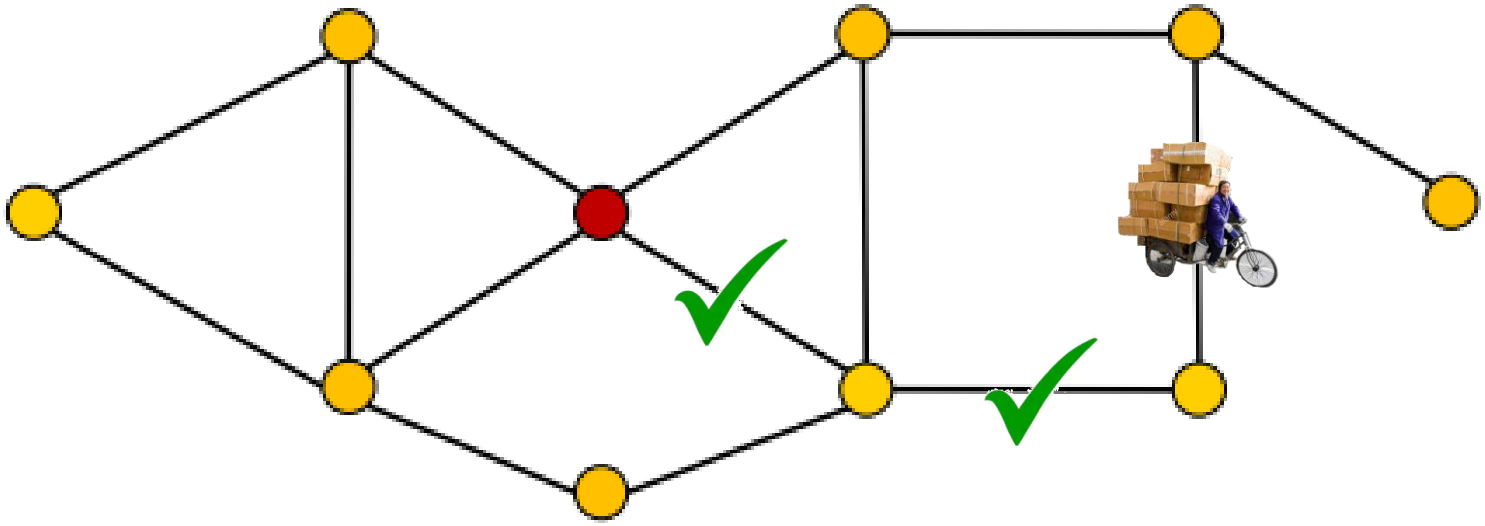


Paths and Cycles …

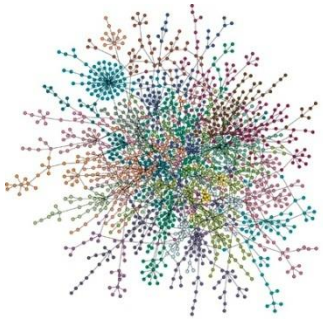


- **Chinese Postman Problem…**
  - The "CPP" Problem:

    A postman starting from his office has to cross all the streets and return to his office as soon as possible!

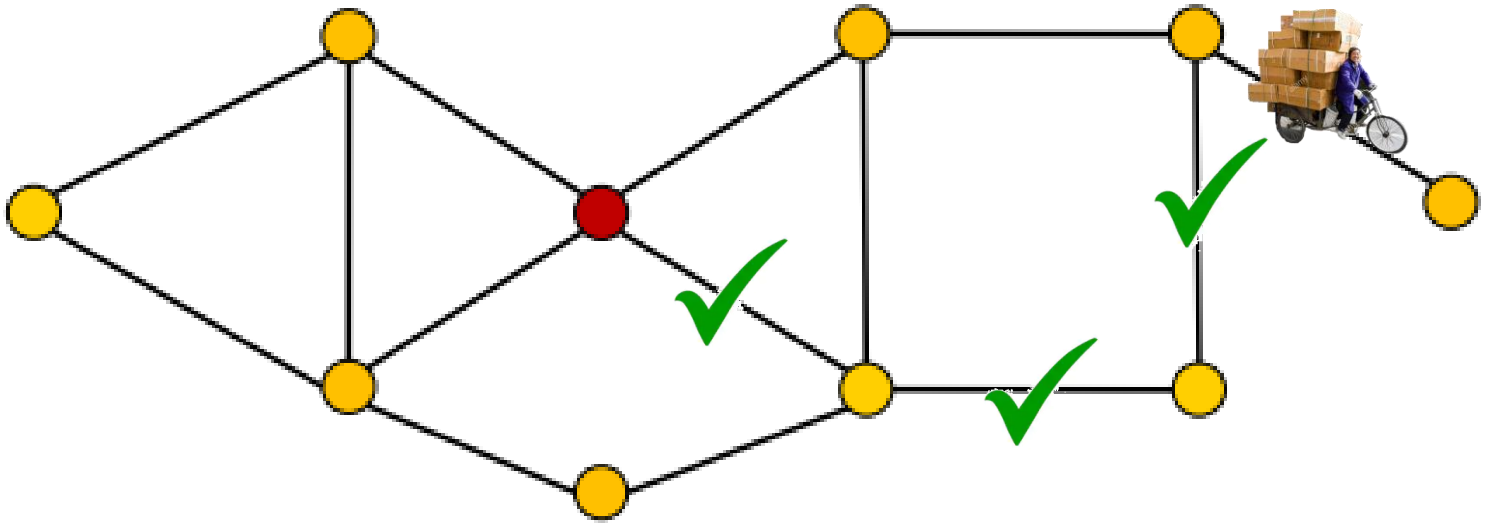


Paths and Cycles …

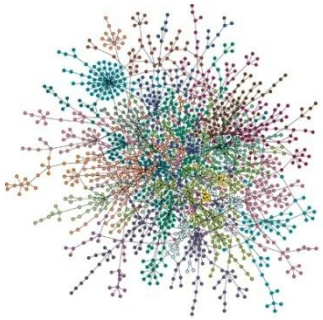


- **Chinese Postman Problem…**
  - The "CPP" Problem:

    A postman starting from his office has to cross all the streets and return to his office as soon as possible!

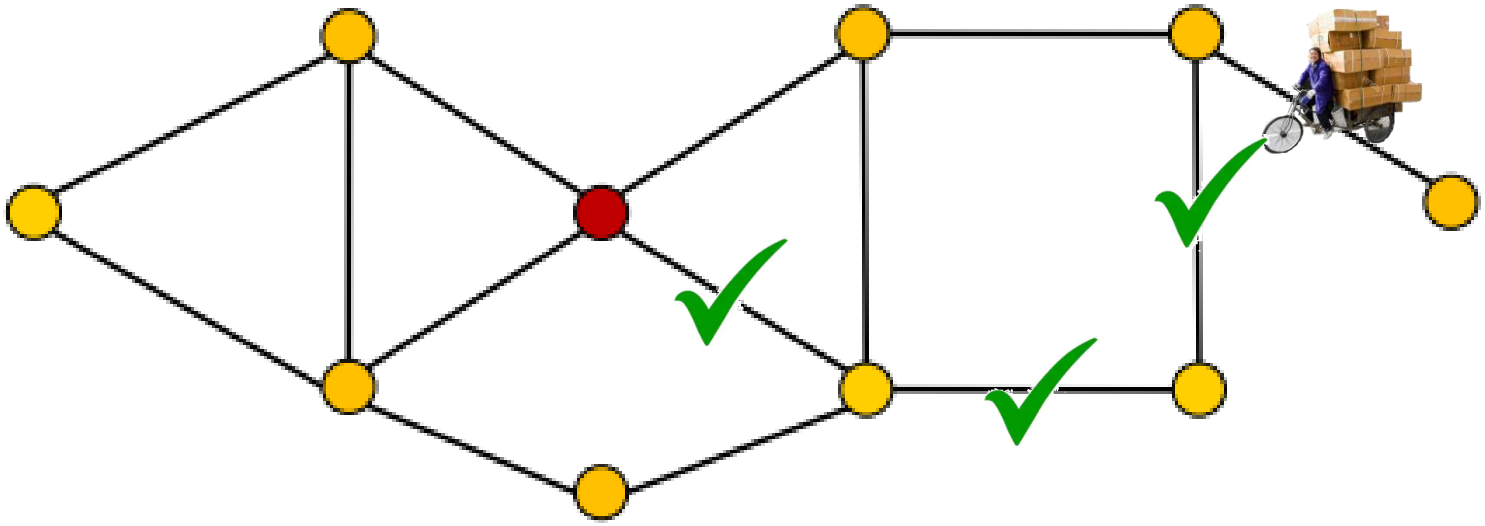


Paths and Cycles …

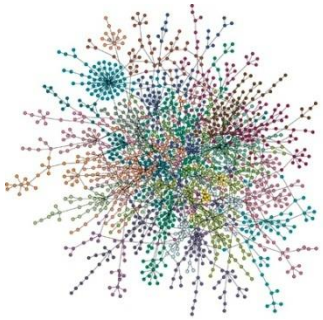


- **Chinese Postman Problem…**
  - The "CPP" Problem:

    A postman starting from his office has to cross all the streets and return to his office as soon as possible!

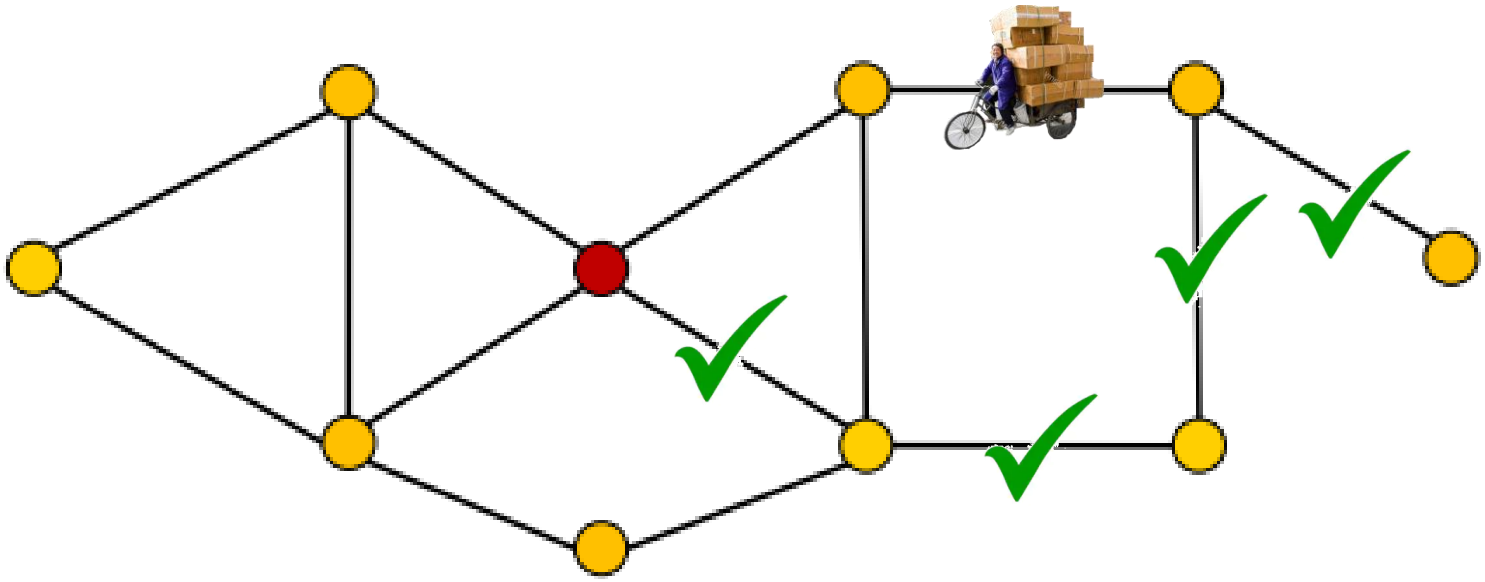


Paths and Cycles …

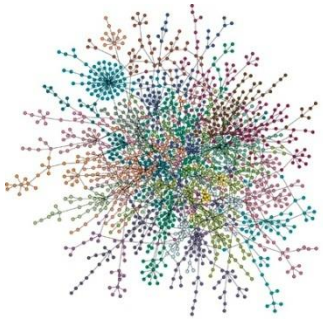


- **Chinese Postman Problem…**
  - The "CPP" Problem:

    A postman starting from his office has to cross all the streets and return to his office as soon as possible!

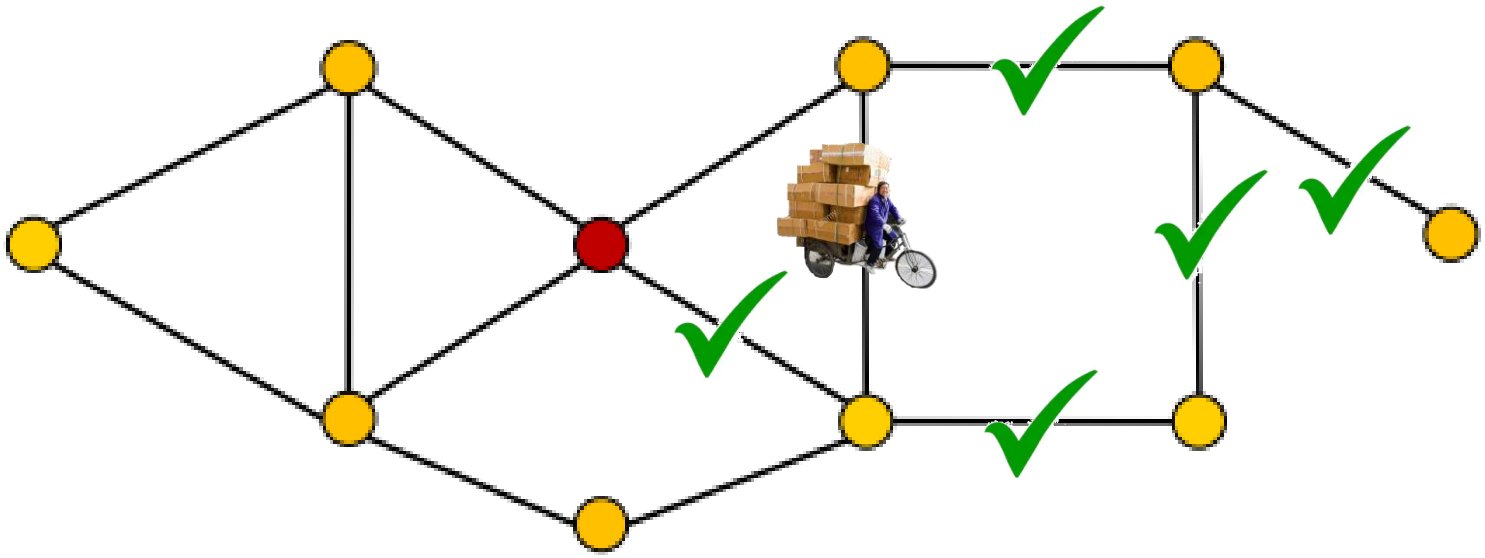


Paths and Cycles …

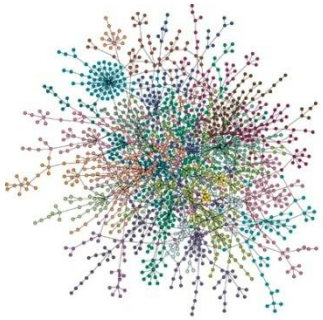


- **Chinese Postman Problem…**
  - The "CPP" Problem:

    A postman starting from his office has to cross all the streets and return to his office as soon as possible!

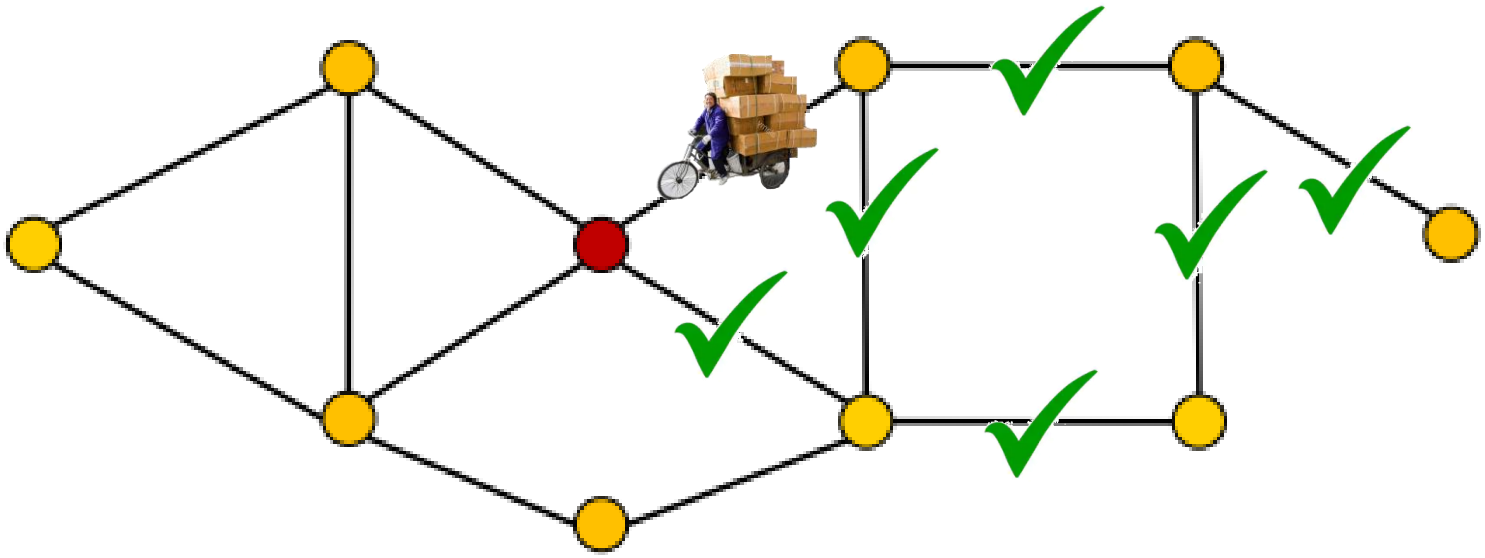


Paths and Cycles …

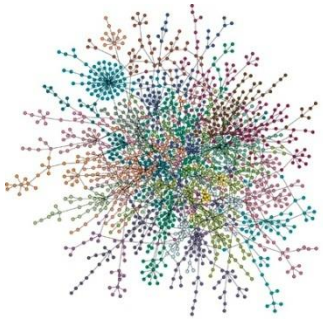


- **Chinese Postman Problem…**
  - The "CPP" Problem:

    A postman starting from his office has to cross all the streets and return to his office as soon as possible!

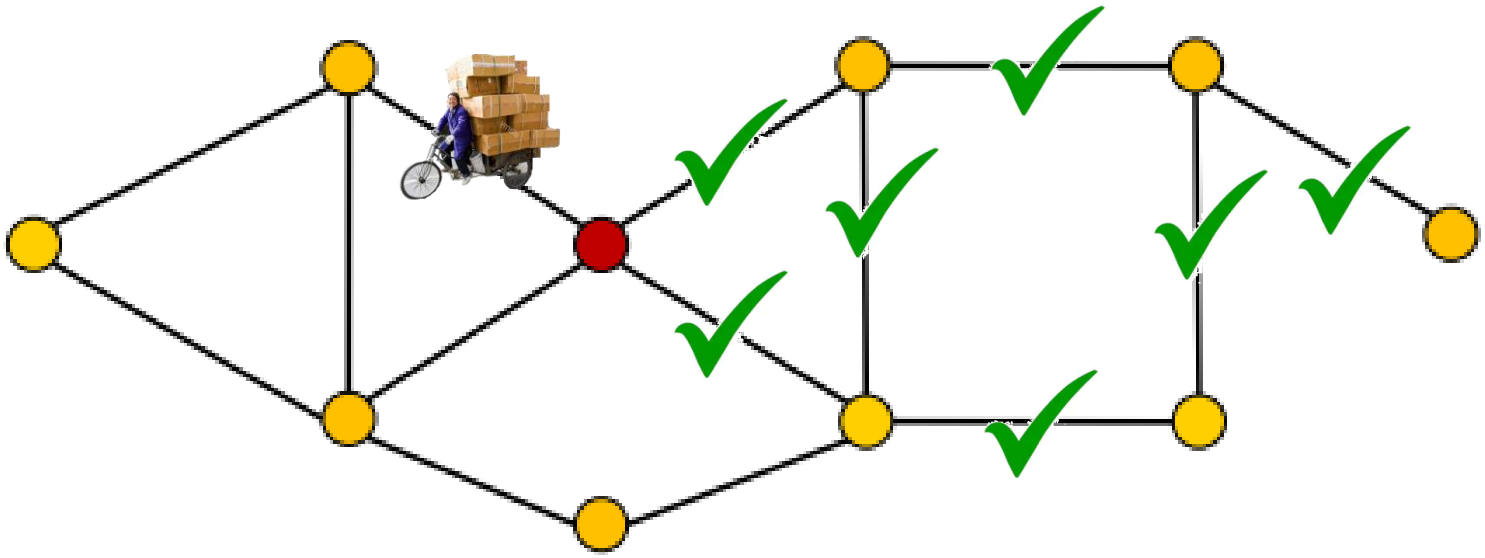


Paths and Cycles …

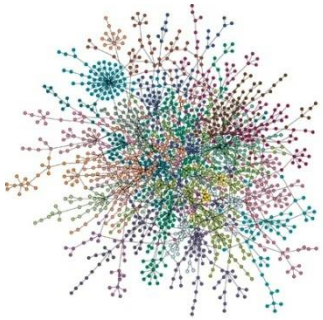


- **Chinese Postman Problem…**
  - The "CPP" Problem:

    A postman starting from his office has to cross all the streets and return to his office as soon as possible!

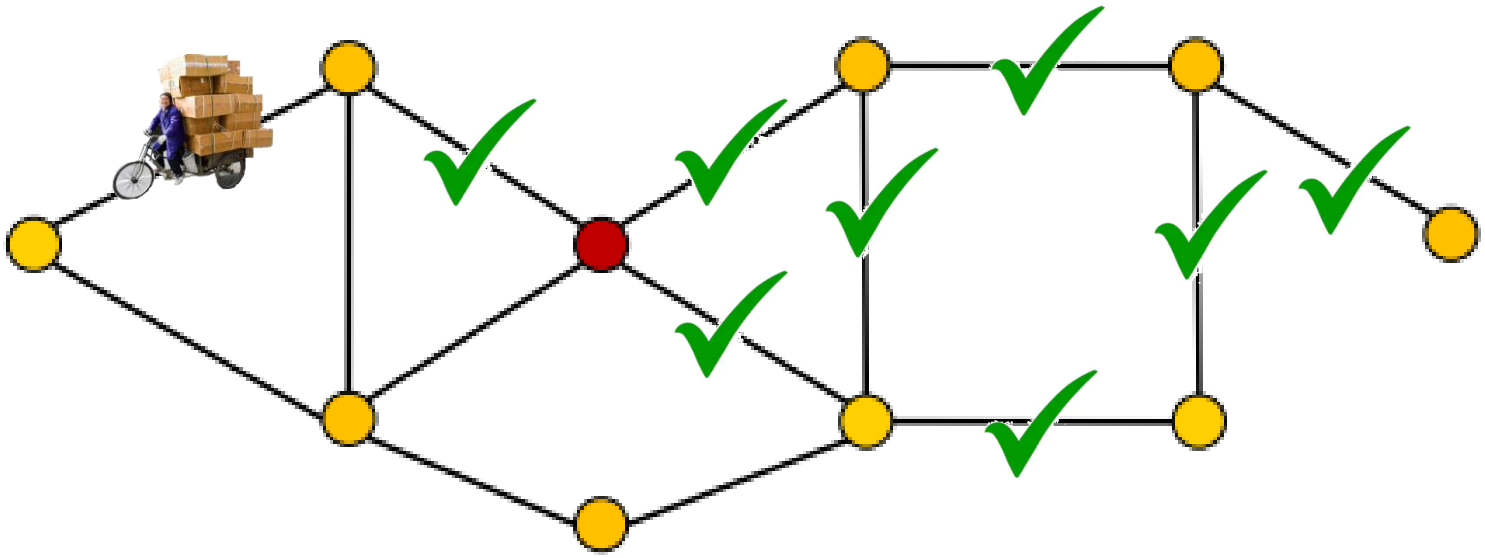


Paths and Cycles …

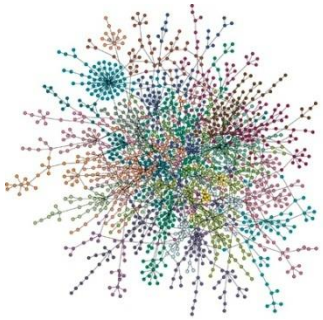


- **Chinese Postman Problem…**
  - The "CPP" Problem:

    A postman starting from his office has to cross all the streets and return to his office as soon as possible!

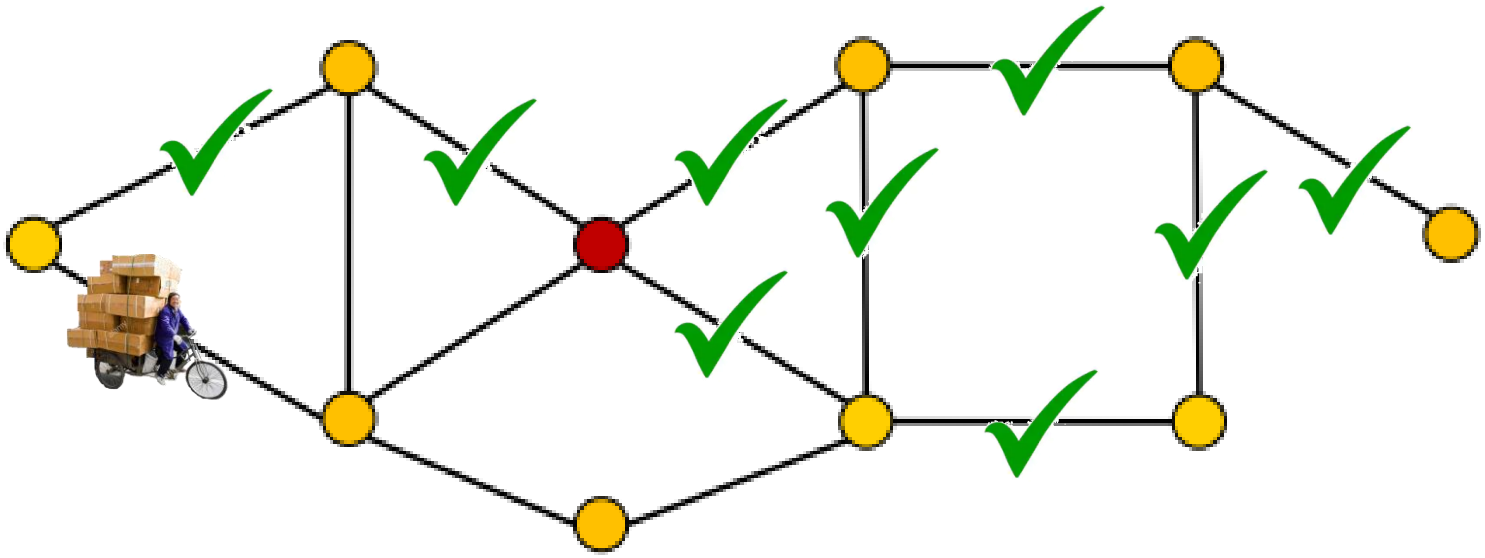


Paths and Cycles …

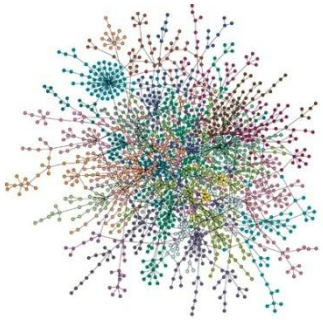


- **Chinese Postman Problem…**
  - The "CPP" Problem:

    A postman starting from his office has to cross all the streets and return to his office as soon as possible!

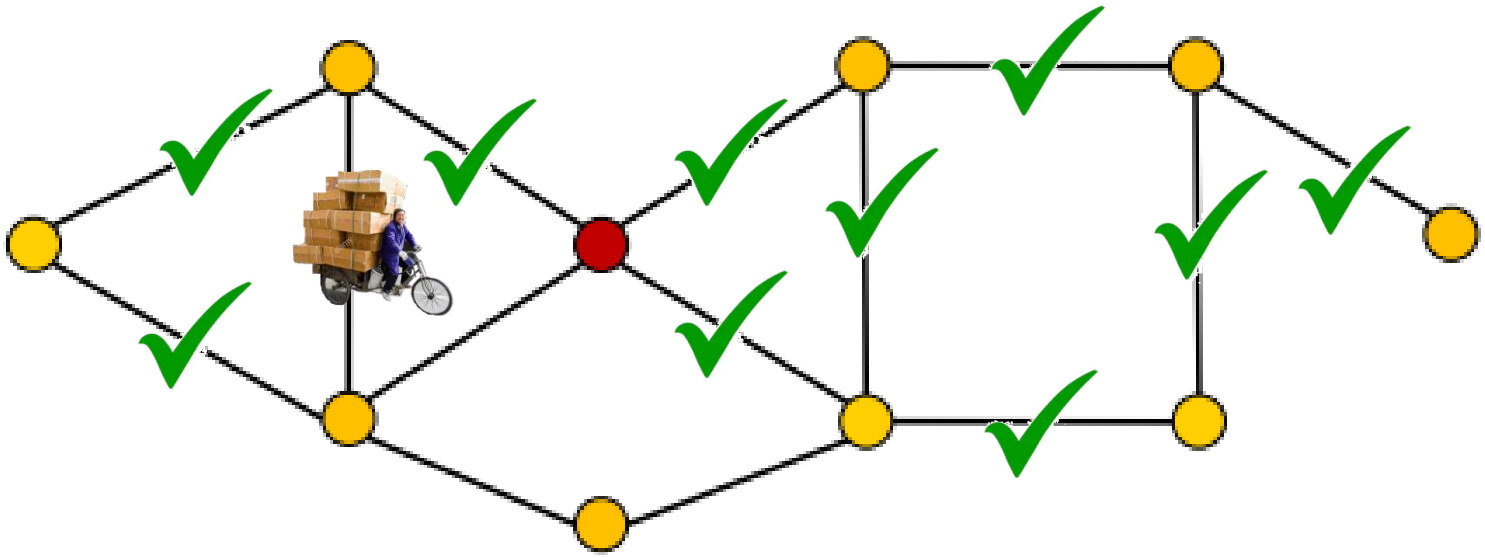


Paths and Cycles …

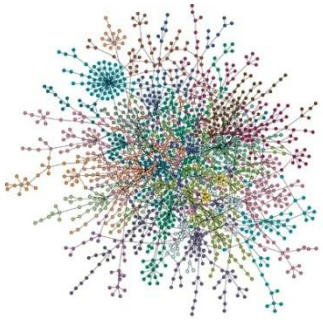


- **Chinese Postman Problem…**
  - The "CPP" Problem:

    A postman starting from his office has to cross all the streets and return to his office as soon as possible!

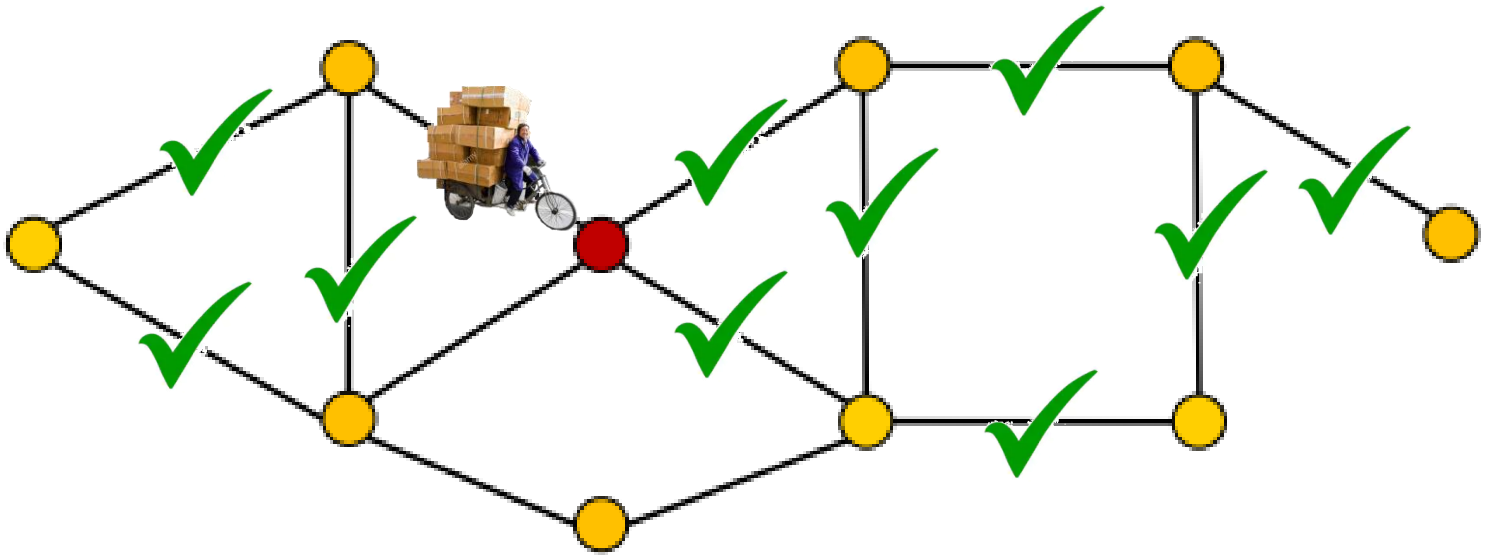


Paths and Cycles …

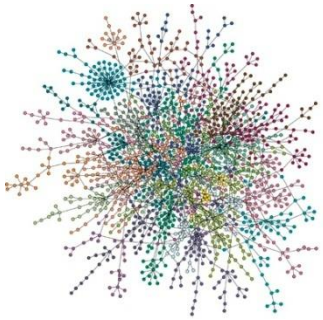


- **Chinese Postman Problem…**
  - The "CPP" Problem:

    A postman starting from his office has to cross all the streets and return to his office as soon as possible!

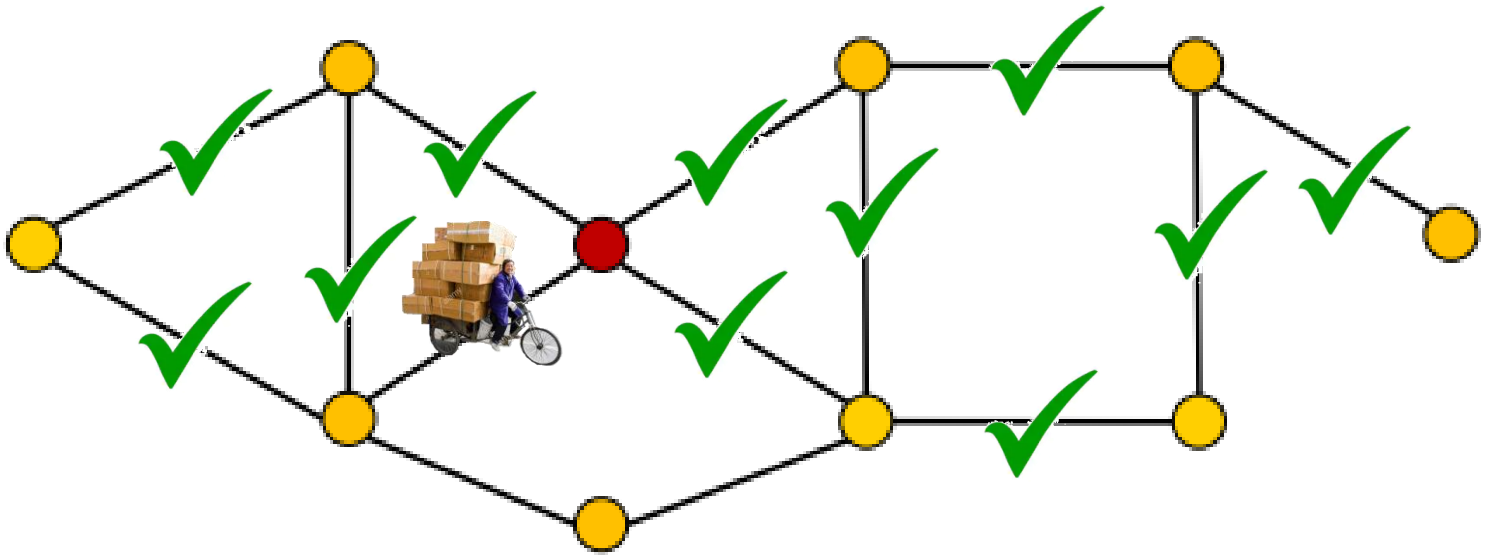


Paths and Cycles …

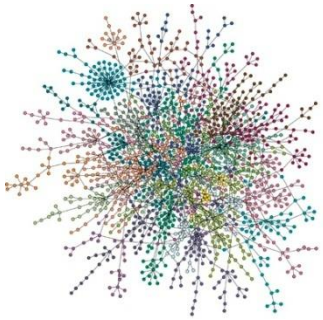


- **Chinese Postman Problem…**
  - The "CPP" Problem:

    A postman starting from his office has to cross all the streets and return to his office as soon as possible!

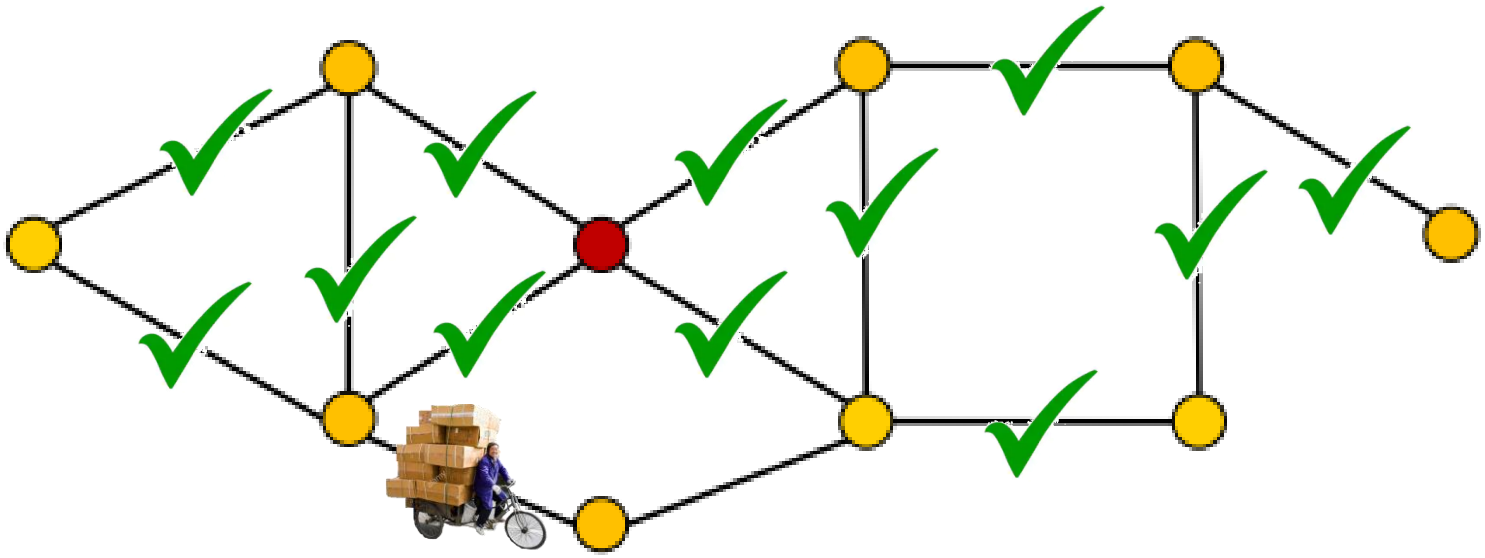


Paths and Cycles …

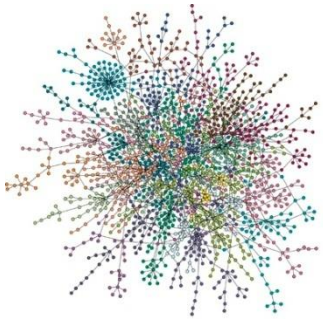


- **Chinese Postman Problem…**
  - The "CPP" Problem:

    A postman starting from his office has to cross all the streets and return to his office as soon as possible!

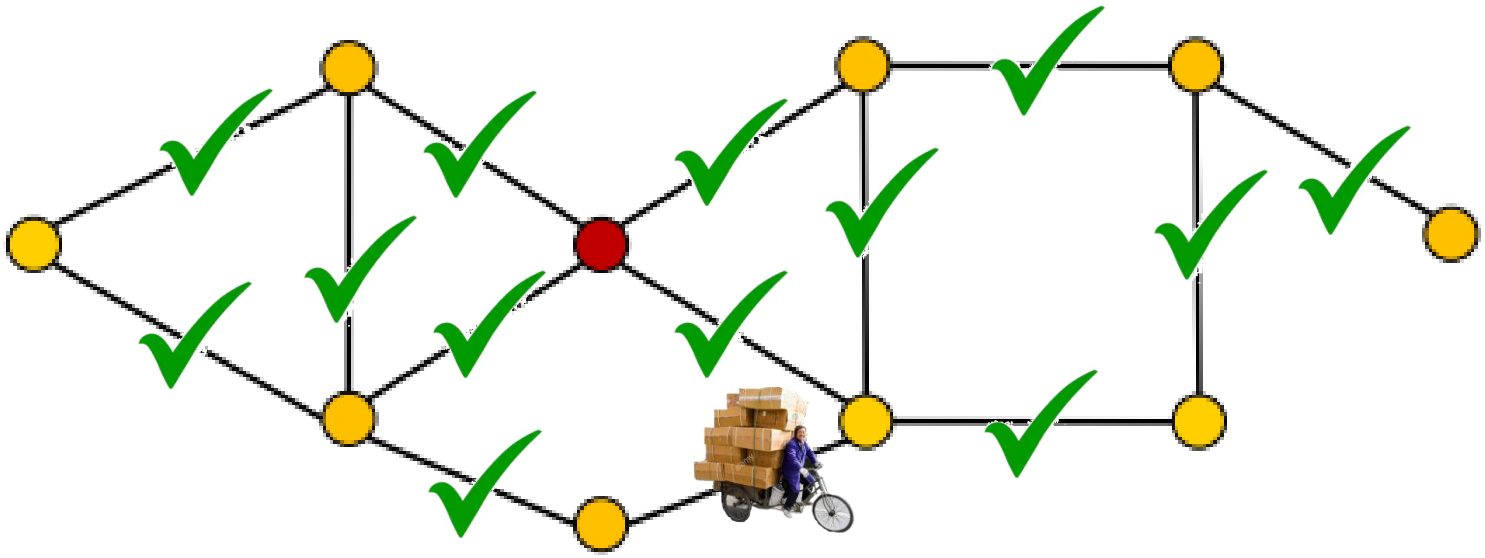


Paths and Cycles …

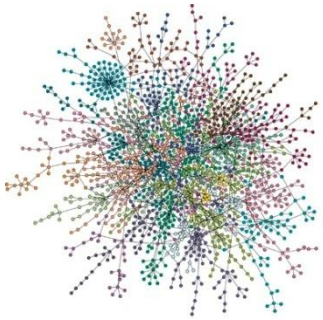


- **Chinese Postman Problem…**
  - The "CPP" Problem:

    A postman starting from his office has to cross all the streets and return to his office as soon as possible!

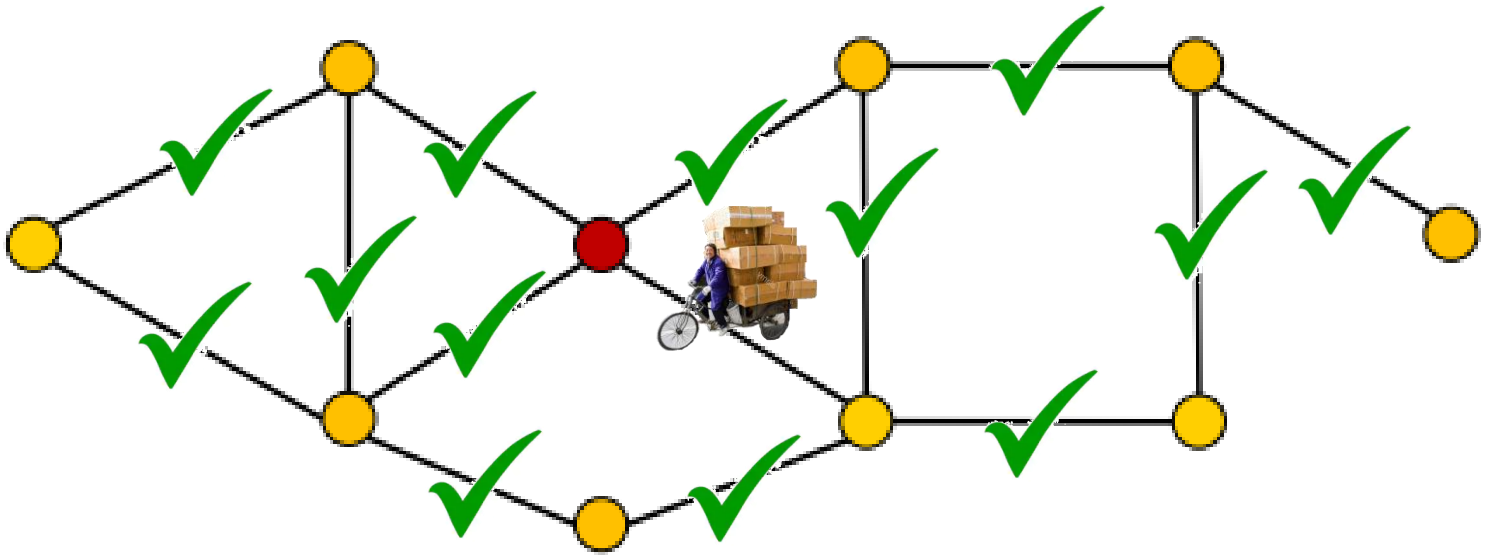


Paths and Cycles …

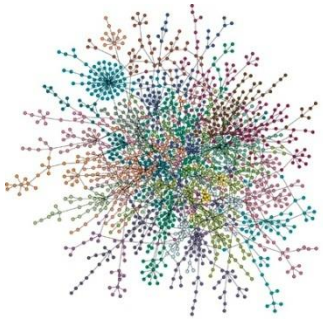


- **Chinese Postman Problem…**
  - The "CPP" Problem:

    A postman starting from his office has to cross all the streets and return to his office as soon as possible!

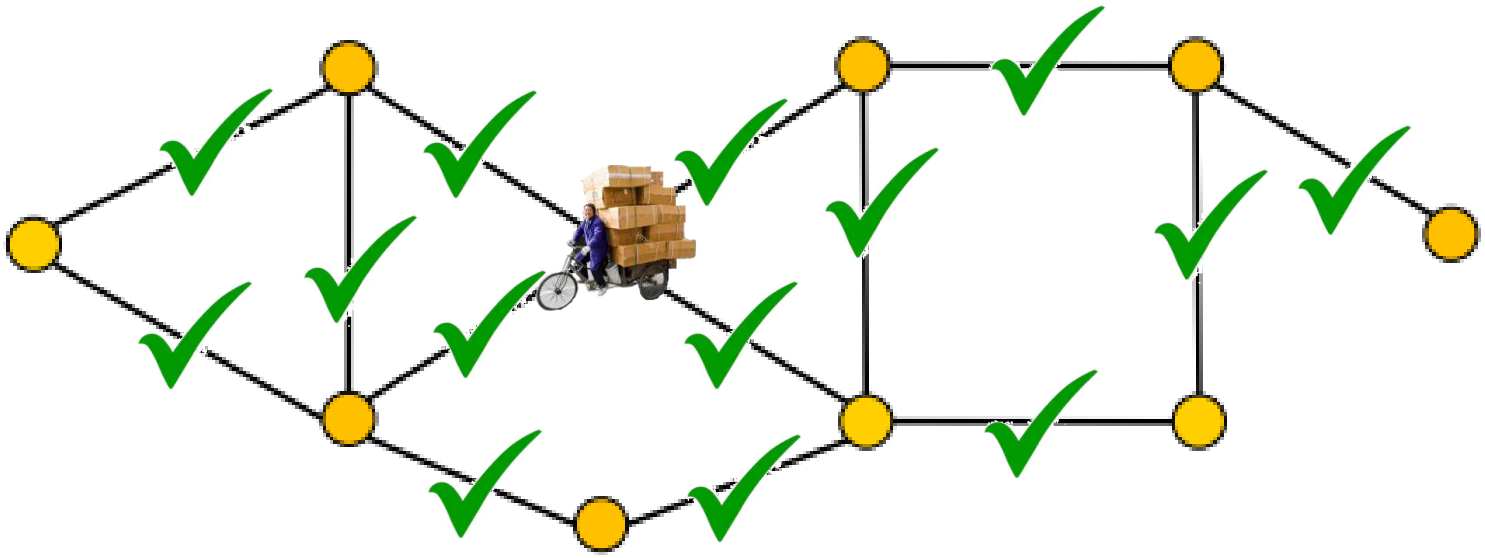


Paths and Cycles …

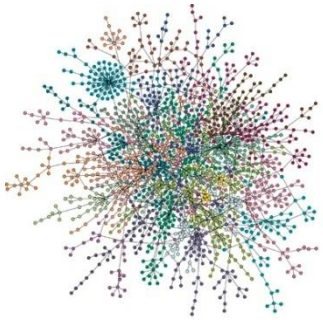


- **Chinese Postman Problem…**
  - The "CPP" Problem:

    A postman starting from his office has to cross all the streets and return to his office as soon as possible!



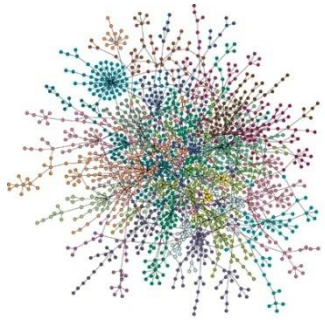
Paths and Cycles …



- **Chinese Postman Problem…**
  - The "CPP" Problem:

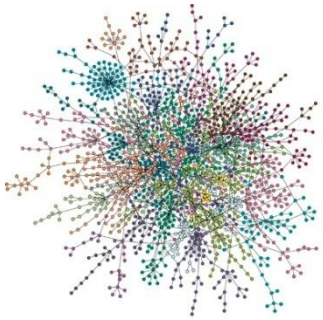
    A postman starting from his office has to cross all the streets and return to his office as soon as possible!

Paths and Cycles …



- **Chinese Postman Problem…**
  - The "CPP" Problem:
    A postman starting from his office has to cross all the streets and return to his office as soon as possible!
  - find a shortest closed path or circuit that visits every edge of an (connected) undirected graph.
  - When the graph has an Eulerian circuit that is an optimal solution.
  - Otherwise, the optimization problem is to find the smallest number of graph edges to duplicate (or the subset of edges with the minimum possible total weight) so that the resulting multigraph does have an Eulerian circuit.
  - It may be solved in polynomial time.

Paths and Cycles …

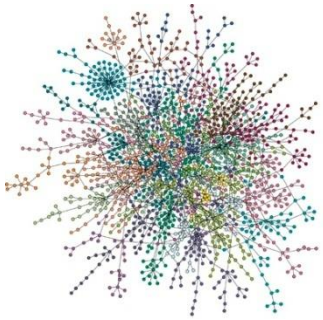


- **Chinese Postman Problem…**
  - The "CPP" Problem:

    A postman starting from his office has to cross all the streets and return to his office as soon as possible!

  - If the graph G is not Eulerian, then some edges have to be crossed more than one times … how many?

Paths and Cycles …

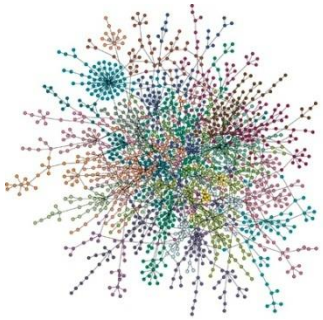


- **Chinese Postman Problem…**
  - The "CPP" Problem:

    A postman starting from his office has to cross all the streets and return to his office as soon as possible!

  - If the graph G is not Eulerian, then some edges have to be crossed more than one times … how many?
    - If graph G is a tree → … times

Paths and Cycles …



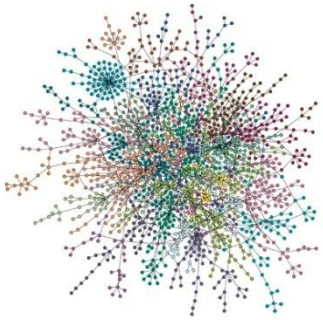
- **Chinese Postman Problem…**
  - The "CPP" Problem:

    A postman starting from his office has to cross all the streets and return to his office as soon as possible!

  - If the graph G is not Eulerian, then some edges have to be crossed more than one times … how many?
    - If graph G is a tree → two times

      The length l of the optimal solution is $|E| \leq l \leq 2|E|$

Paths and Cycles …



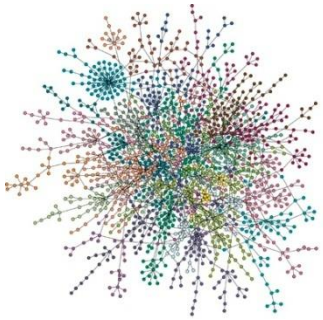
- **Chinese Postman Problem…**
  - The "CPP" Problem:

    A postman starting from his office has to cross all the streets and return to his office as soon as possible!

  - If the graph G is not Eulerian, then some edges have to be crossed more than one times … how many?
    - If graph G is a tree → two times

      The length l of the optimal solution is $|E| \leq l \leq 2|E|$
    - Otherwise ???

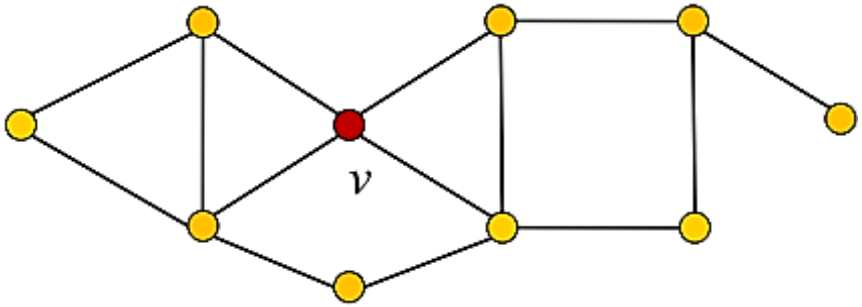
Paths and Cycles …



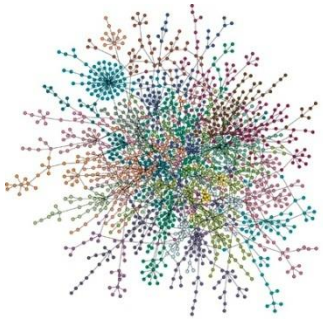
- **Chinese Postman Problem…**
  - The "CPP" Problem:

    A postman starting from his office has to cross all the streets and return to his office as soon as possible!

  - If the graph G is not Eulerian, then some edges have to be crossed more than one times … how many?
    - Solution in case the graph G is not a tree …



Paths and Cycles …

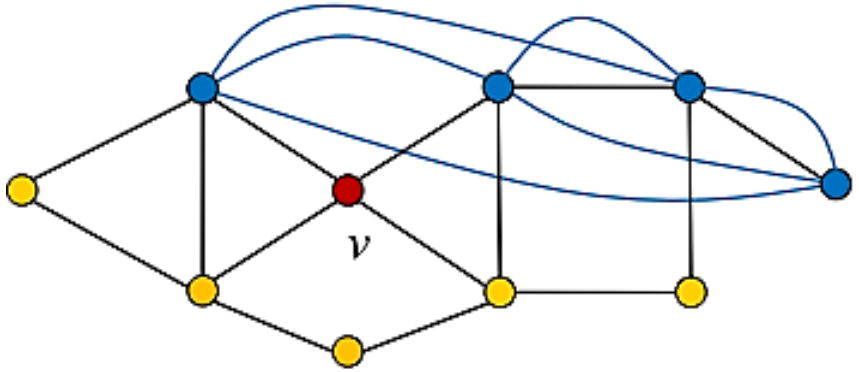


- **Chinese Postman Problem…**
  - The "CPP" Problem:

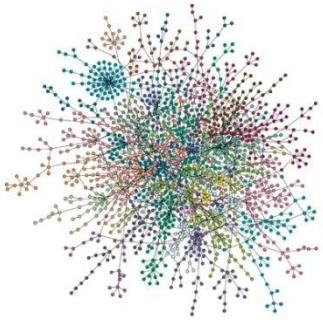
    A postman starting from his office has to cross all the streets and return to his office as soon as possible!

  - If the graph G is not Eulerian, then some edges have to be crossed more than one times … how many?
    - Solution in case the graph G is not a tree …

      We select the vertices of odd-degree and connect them with "virtual edges" (if necessary), of weight equal to the weight (length) of the shortest path between them.



Paths and Cycles …

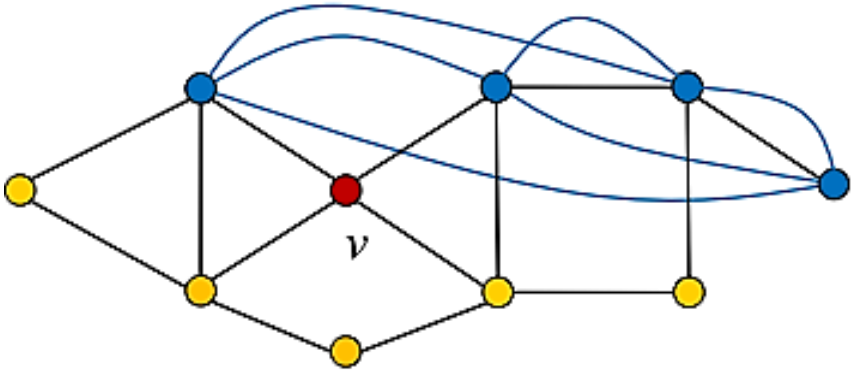


- **Chinese Postman Problem…**
  - The "CPP" Problem:

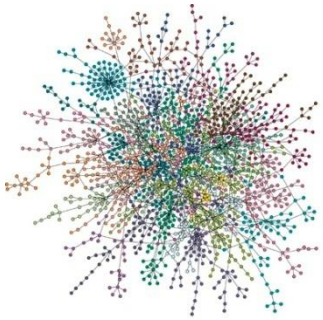
    A postman starting from his office has to cross all the streets and return to his office as soon as possible!

  - If the graph G is not Eulerian, then some edges have to be crossed more than one times … how many?
    - Solution in case the graph G is not a tree …

      We apply an algorithm for the computation of Eulerian circuit, and replace each "virtual-edge", let xy, with a real shortest path from vertex x to vertex y.



Paths and Cycles …

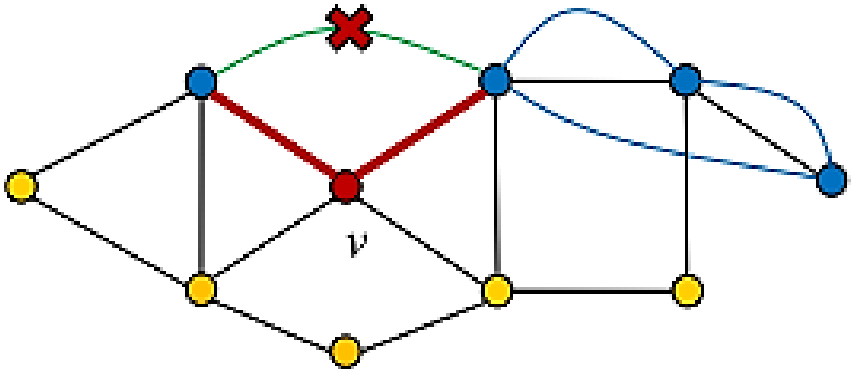


- **Chinese Postman Problem…**
  - The "CPP" Problem:

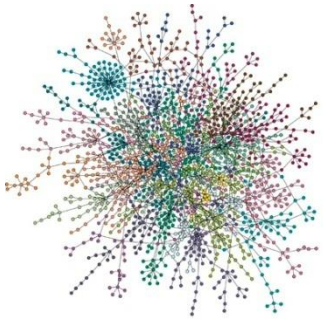
    A postman starting from his office has to cross all the streets and return to his office as soon as possible!

  - If the graph G is not Eulerian, then some edges have to be crossed more than one times … how many?
    - Solution in case the graph G is not a tree …

      We apply an algorithm for the computation of Eulerian circuit, and replace each "virtual-edge", let xy, with a real shortest path from vertex x to vertex y.



Paths and Cycles …

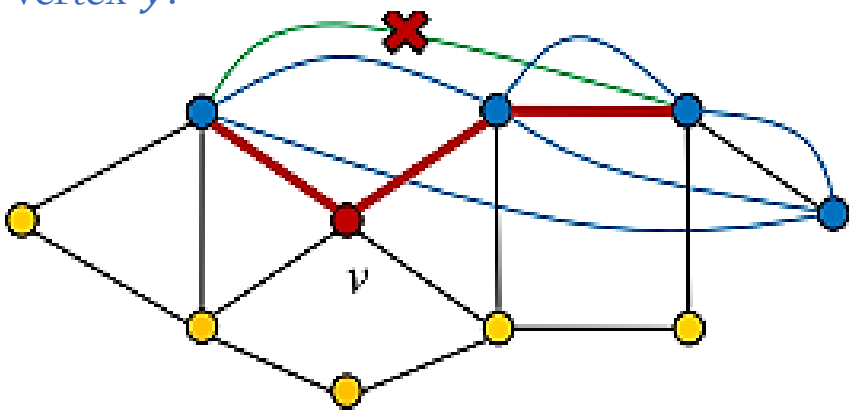


- **Chinese Postman Problem…**
  - The "CPP" Problem:

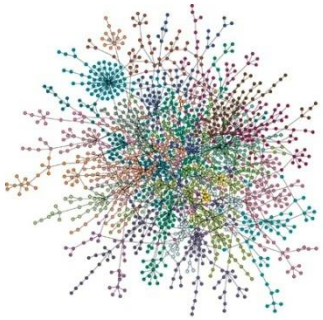
    A postman starting from his office has to cross all the streets and return to his office as soon as possible!

  - If the graph G is not Eulerian, then some edges have to be crossed more than one times … how many?
    - Solution in case the graph G is not a tree …

      We apply an algorithm for the computation of Eulerian circuit, and replace each "virtual-edge", let xy, with a real shortest path from vertex x to vertex y.



Paths and Cycles …

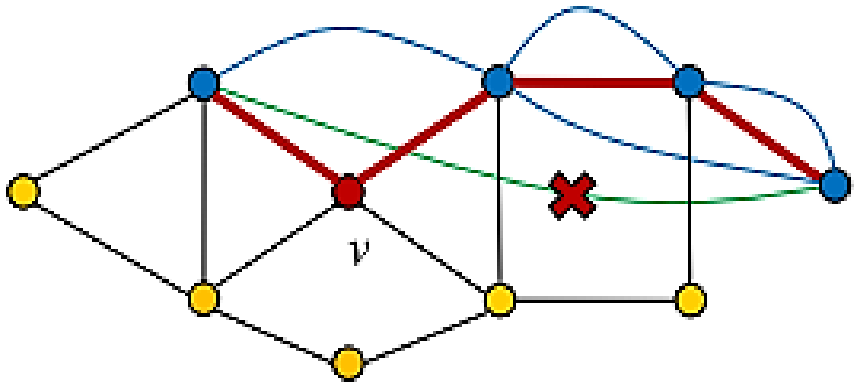


- **Chinese Postman Problem…**
  - The "CPP" Problem:

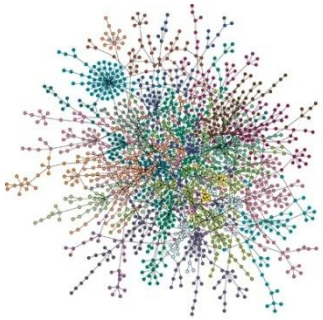
    A postman starting from his office has to cross all the streets and return to his office as soon as possible!

  - If the graph G is not Eulerian, then some edges have to be crossed more than one times … how many?
    - Solution in case the graph G is not a tree …

      We apply an algorithm for the computation of Eulerian circuit, and replace each "virtual-edge", let xy, with a real shortest path from vertex x to vertex y.



Paths and Cycles …

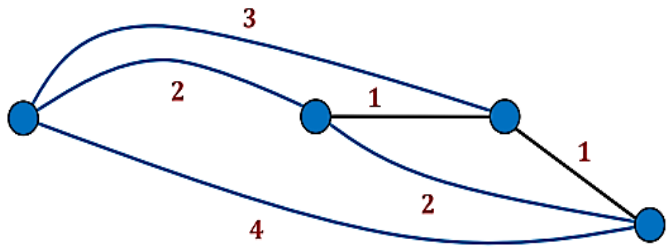


- **Chinese Postman Problem…**
  - The "CPP" Problem:

    A postman starting from his office has to cross all the streets and return to his office as soon as possible!

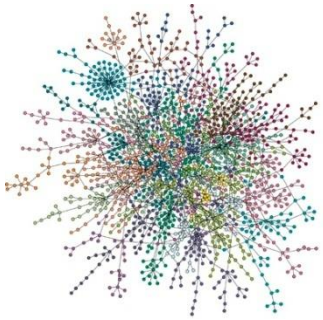
  - If the graph G is not Eulerian, then some edges have to be crossed more than one times … how many?
    - Solution in case the graph G is not a tree …

      We apply an algorithm for the computation of Eulerian circuit, and replace each "virtual-edge", let xy, with a real shortest path from vertex x to vertex y.



Optimal Matching

Paths and Cycles …

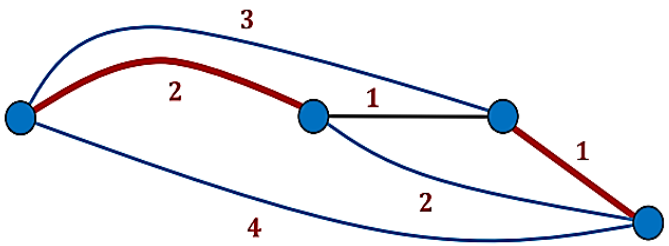


- **Chinese Postman Problem…**
  - The "CPP" Problem:

    A postman starting from his office has to cross all the streets and return to his office as soon as possible!

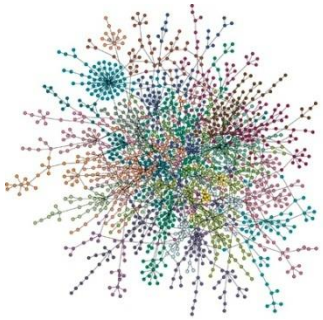
  - If the graph G is not Eulerian, then some edges have to be crossed more than one times … how many?
    - Solution in case the graph G is not a tree …

      We apply an algorithm for the computation of Eulerian circuit, and replace each "virtual-edge", let xy, with a real shortest path from vertex x to vertex y.



Optimal Matching

Paths and Cycles …

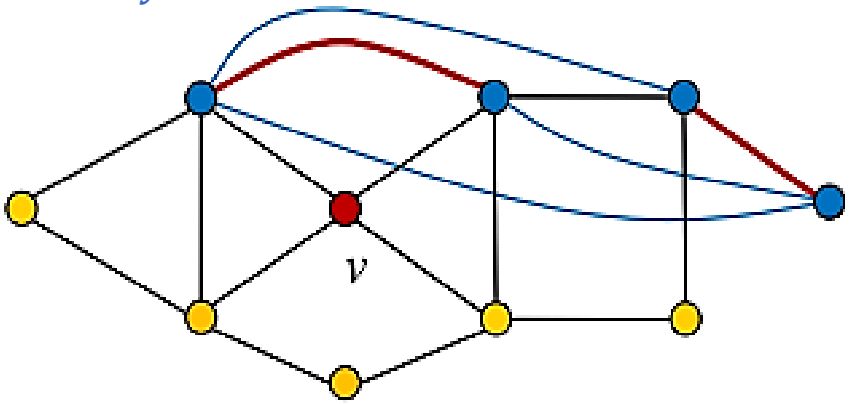


- **Chinese Postman Problem…**
  - The "CPP" Problem:

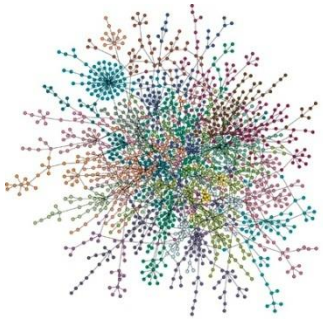
    A postman starting from his office has to cross all the streets and return to his office as soon as possible!

  - If the graph G is not Eulerian, then some edges have to be crossed more than one times … how many?
    - Solution in case the graph G is not a tree …

      We apply an algorithm for the computation of Eulerian circuit, and replace each "virtual-edge", let xy, with a real shortest path from vertex x to vertex y.



Paths and Cycles …

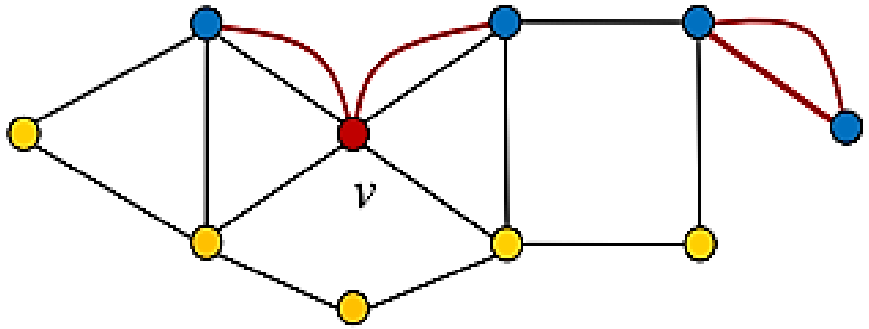


- **Chinese Postman Problem…**
  - The "CPP" Problem:

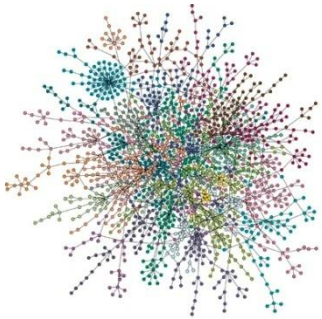
    A postman starting from his office has to cross all the streets and return to his office as soon as possible!

  - If the graph G is not Eulerian, then some edges have to be crossed more than one times … how many?
    - Solution in case the graph G is not a tree …

      We apply an algorithm for the computation of Eulerian circuit, and replace each "virtual-edge", let xy, with a real shortest path from vertex x to vertex y.

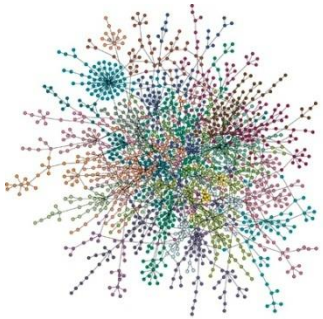


Paths and Cycles …



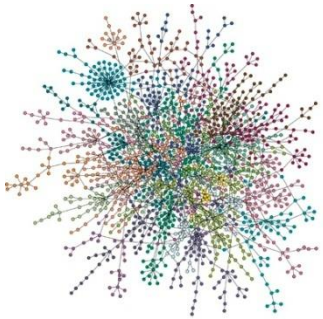
- **Around the World …**

Paths and Cycles …



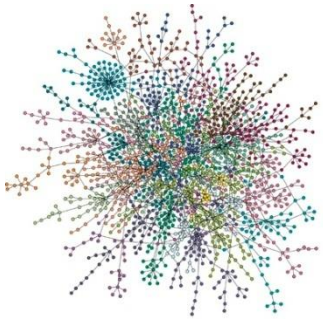
- **Around the World …**

Paths and Cycles …



- **Around the World …**

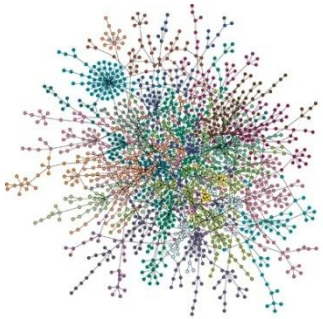
Paths and Cycles …



- Around the World …

Sir William Rowan Hamilton (1806-1865)

Paths and Cycles …

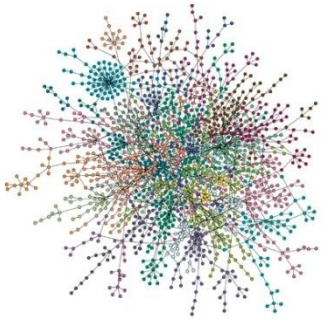


- **Around the World …**

Sir William Rowan Hamilton (1806-1865)

Given a 12-side geometrical shape (regular graph of degree 3, 20 vertices and 30 edges), each vertex represents a city.

Paths and Cycles …

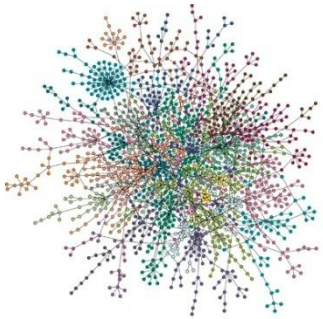


- **Around the World …**

Sir William Rowan Hamilton (1806-1865)

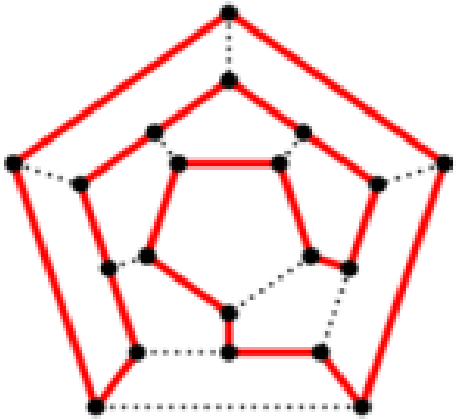
Given a 12-side geometrical shape (regular graph of degree 3, 20 vertices and 30 edges), each vertex represents a city. Let the players "A" and "B".

Paths and Cycles …



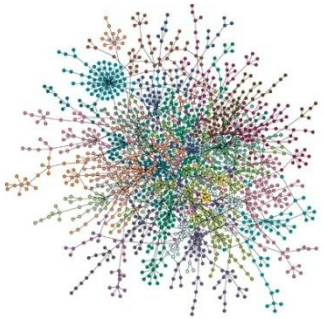
- **Around the World …**

Sir William Rowan Hamilton (1806-1865)



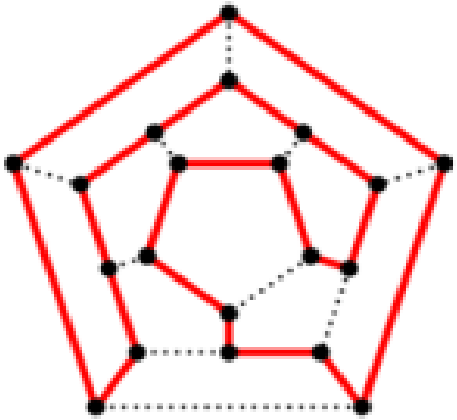
Given a 12-side geometrical shape (regular graph of degree 3, 20 vertices and 30 edges), each vertex represents a city. Let the players "A" and "B". Player "A" defines the 5 first cities and challenges "B" to draw a path crossing only once each of the rest cities finishing, to the city given first by "A".

Paths and Cycles …

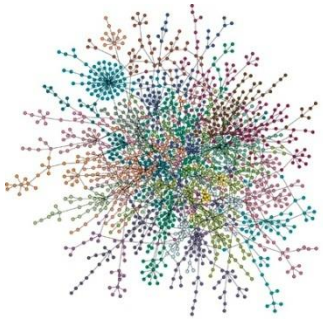


- Around the World …

Sir William Rowan Hamilton (1806-1865)



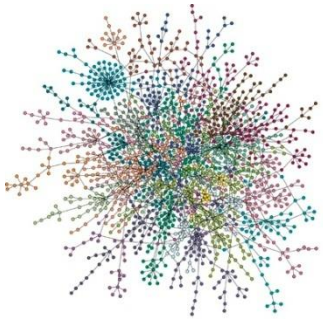
Paths and Cycles …



- **Hamiltonian Graphs**
  - Problem:

    Is it possible in each graph to find a closed path (cycle) that crosses all vertices only once?

Paths and Cycles …

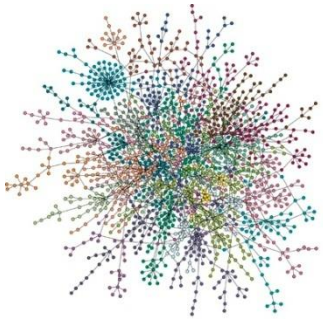


- **Hamiltonian Graphs**
  - Problem:

    Is it possible in each graph to find a closed path (cycle) that crosses all vertices only once?

  - A **Hamiltonian path** (or traceable path) is a path in an undirected or directed graph that visits each vertex exactly once.
  - A **Hamiltonian cycle** (or Hamiltonian circuit) is a Hamiltonian path that is a cycle.

Paths and Cycles …

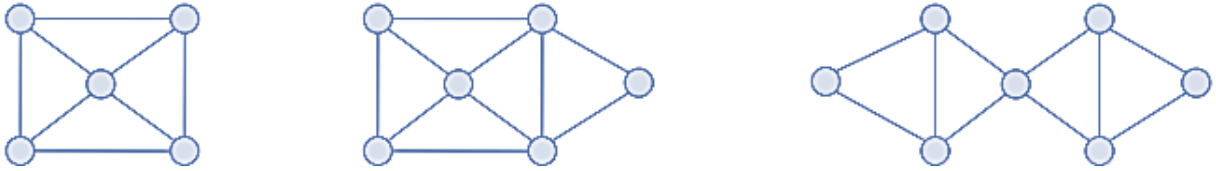


Hamiltonian Graphs

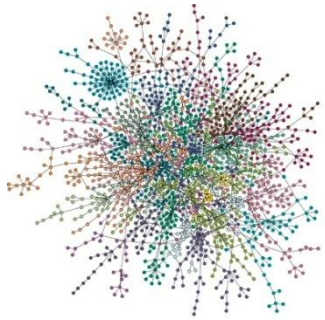
- Problem:

  Is it possible in each graph to find a closed path (cycle) that crosses all vertices only once?

- A **Hamiltonian path** (or traceable path) is a path in an undirected or directed graph that visits each vertex exactly once.
- A **Hamiltonian cycle** (or Hamiltonian circuit) is a Hamiltonian path that is a cycle.
- A **Hamiltonian graph**, is a graph that contains Hamiltonian cycle.



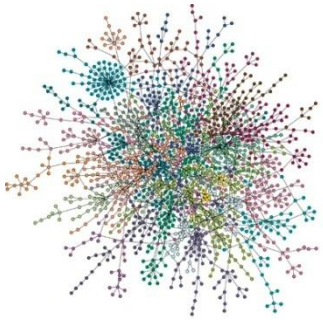
Paths and Cycles …



Hamiltonian Graphs

- A graph G is called **traceable** if it has Hamiltonian path.
- Every Hamiltonian graph is **traceable**, but the inverse does not hold.
- A graph G is called **homogeneously traceable** if there exists a Hamiltonian path starting from a random vertex.
- A graph G is called **hypo-Hamiltonian**, if G is not Hamiltonian but the graph $G - v$ is Hamiltonian.
- Every hypo-Hamiltonian graph is homogeneously traceable, but the inverse does not hold.
- A graph G is called **Hamiltonian-connected** if every two vertices are connected by a Hamiltonian path.
- Every Hamiltonian-connected graph where $n \geq 3$ is Hamiltonian, but the inverse does not hold.

Paths and Cycles …



- **Hamiltonian Graphs**
  - Problem:

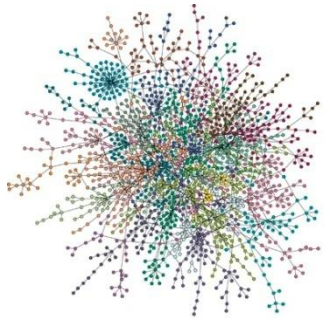
    The necessary and sufficient condition for a graph to be Hamiltonian?
    … NP-Complete

  - **Theorem 1:** Every complete graph is Hamiltonian

    … the graph K_n is Hamiltonian $\forall\, n \geq 1$

  - **Theorem 2:** Each K_n graph of odd cardinality of vertices has $(n-1)/2$ Hamiltonian cycles with different edges.

Paths and Cycles …



Hamiltonian Graphs

- Problem:

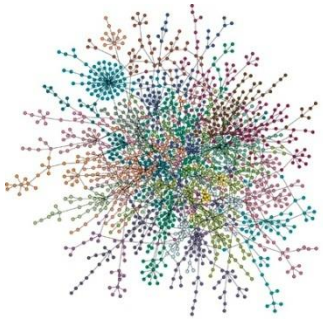
  The necessary and sufficient condition for a graph to be Hamiltonian? … NP-Complete

- **Theorem 1:** Every complete graph is Hamiltonian

  … the graph K_n is Hamiltonian $\forall\, n \geq 1$

- **Theorem 2:** Each K_n graph of odd cardinality of vertices has $(n-1)/2$ Hamiltonian cycles with different edges.
  - A complete graph has n(n-1)/2 edges.
  - A Hamiltonian graph cycle has length n.
  - The graph may has at most (n-1)/2 cycles of different edges.
  - It has to be proven that is has exactly as many cycles …

Paths and Cycles …



Hamiltonian Graphs

- Problem:

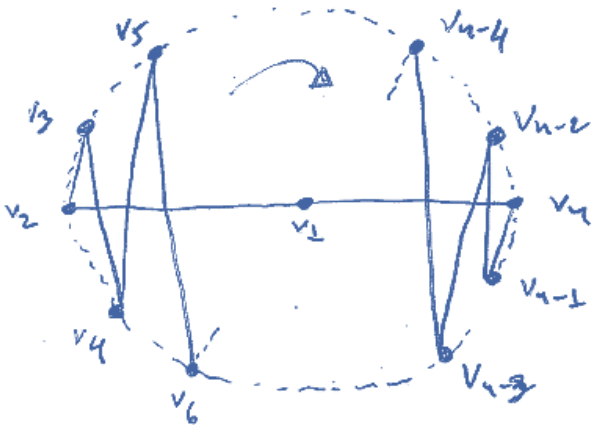
  The necessary and sufficient condition for a graph to be Hamiltonian?
  … NP-Complete

- **Theorem 1:** Every complete graph is Hamiltonian

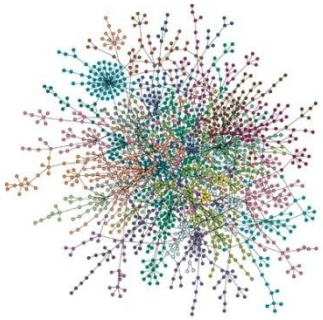
  … the graph K_n is Hamiltonian $\forall\, n \geq 1$

- **Theorem 2:** Each K_n graph of odd cardinality of vertices has $(n-1)/2$ Hamiltonian cycles with different edges.

  Proof:



Paths and Cycles …



Hamiltonian Graphs

- Problem:

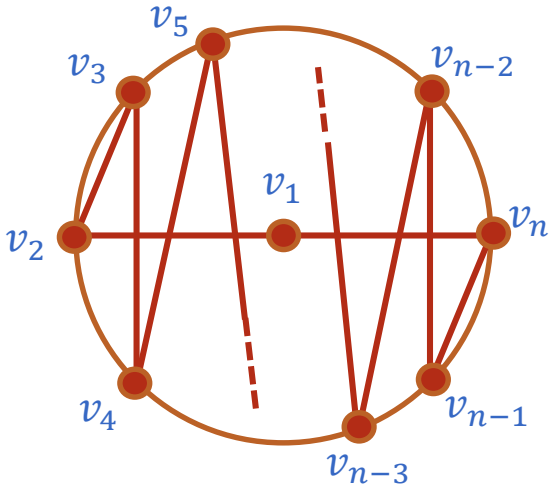
  The necessary and sufficient condition for a graph to be Hamiltonian?
  … NP-Complete

- **Theorem 1:** Every complete graph is Hamiltonian

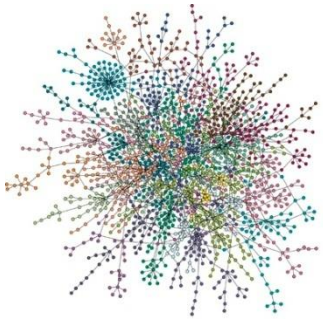
  … the graph K_n is Hamiltonian $\forall\, n \geq 1$

- **Theorem 2:** Each K_n graph of odd cardinality of vertices has $(n-1)/2$ Hamiltonian cycles with different edges.

Proof:



Paths and Cycles …



Hamiltonian Graphs

- Problem:

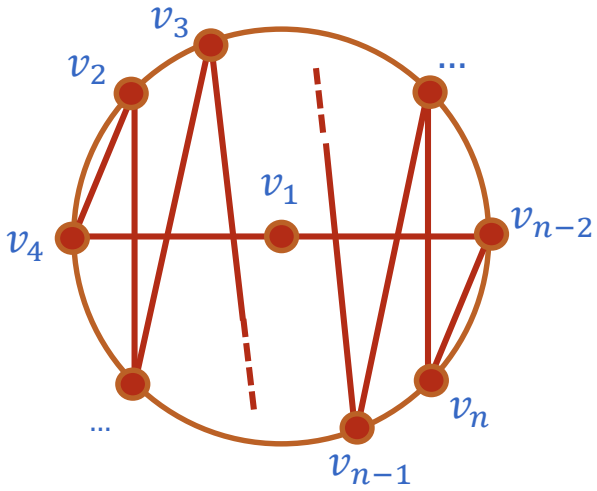
  The necessary and sufficient condition for a graph to be Hamiltonian?
  … NP-Complete

- **Theorem 1:** Every complete graph is Hamiltonian

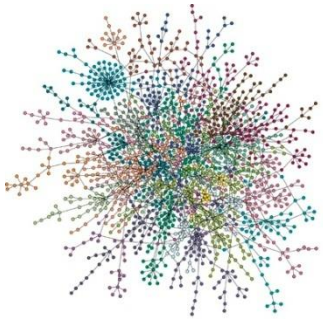
  … the graph K_n is Hamiltonian $\forall\, n \geq 1$

- **Theorem 2:** Each K_n graph of odd cardinality of vertices has $(n-1)/2$ Hamiltonian cycles with different edges.

Proof:



Paths and Cycles …



Hamiltonian Graphs

- Problem:

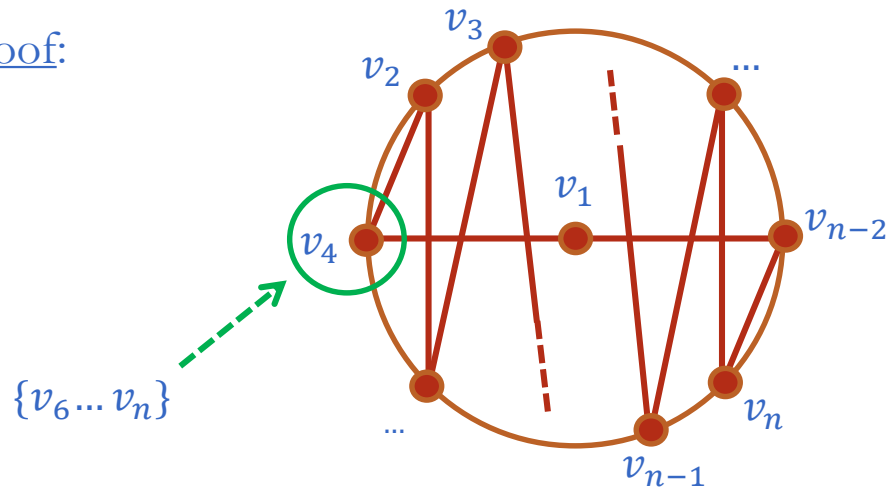
  The necessary and sufficient condition for a graph to be Hamiltonian?
  … NP-Complete

- **Theorem 1:** Every complete graph is Hamiltonian

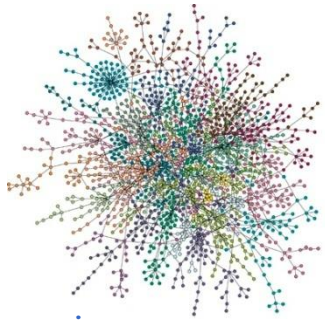
  … the graph K_n is Hamiltonian $\forall\, n \geq 1$

- **Theorem 2:** Each K_n graph of odd cardinality of vertices has $(n-1)/2$ Hamiltonian cycles with different edges.

Proof:



Paths and Cycles …



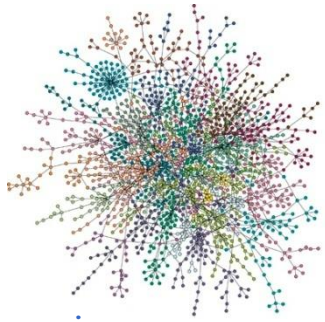
Hamiltonian Graphs

- **Theorem 3 (Dirac, 1952)**: If G is a simple graph with $n \geq 3$ vertices and it holds that $d(G) \geq n/2$, then the graph is Hamiltonian.

  Proof:

  - Let us assume that there exist k new vertices connected to all the vertices of G.
  - Let that k is as small as possible as the produced graph G' to be Hamiltonian.
  - We will assume that $k \geq 0$ and we will result to contradiction.

Paths and Cycles …



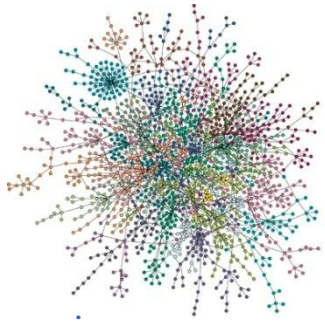
Hamiltonian Graphs

- **Theorem 3 (Dirac, 1952)**: If G is a simple graph with $n \geq 3$ vertices and it holds that $d(G) \geq n/2$, then the graph is Hamiltonian.

  Proof:

  - Let us assume $C = (v, p, w, \cdots, v)$ a Hamiltonian cycle of graph G', where v, w are vertices of G, while vertex p is one of the k vertices.
  - Vertices v, w are not adjacent, otherwise vertex p could be bypassed negating the assumption that k is as small as possible.
  - A vertex w' adjacent to w, can not appear after a vertex v' that is adjacent to v, because the sequence $v \to p \to w \to \cdots \to v' \to w' \to \cdots \to v$ could be substituted with the sequence $v \to v' \to \cdots \to w \to w' \to \cdots \to v$, inverting the part between w and v',

Paths and Cycles …



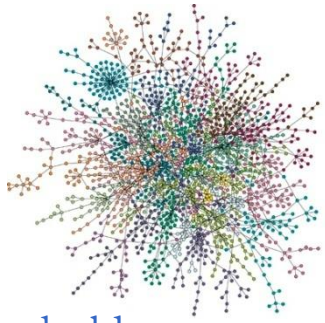
Hamiltonian Graphs

- **Theorem 3 (Dirac, 1952)**: If G is a simple graph with $n \geq 3$ vertices and it holds that $d(G) \geq n/2$, then the graph is Hamiltonian.

  Proof:

  - Hence, the vertices of G' that are non adjacent to w are at least as many as the vertices adjacent to v (i.e., at least $n/2 + k$).
  - The of vertices of G' that are adjacent to w are at least $n/2 + k$
  - Since there does not exist a vertex in G' that is both adjacent and non-adjacent to w, it follows that the total number $n + k$ of the vertices of G' is no less than $n + 2k$ …which is a contradiction.

Paths and Cycles …



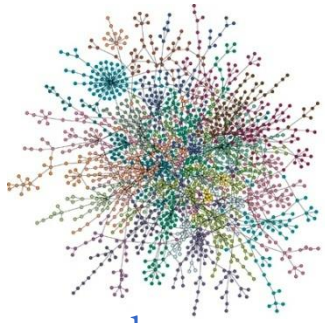
Hamiltonian Graphs

- **Corollary**: If the simple graph G with $n \geq 3$ vertices and it holds that $d(G) \geq (n-1)/2$, then the graph has a Hamiltonian path .

  Proof:

  - Let the hyper graph $H = G + K_1$, and let v a vertex of $H \notin V(G)$.
  - Since the hyper graph H is of order $n+1$, it holds that $d(H) = d(G) \geq (n-1)/2 + 1 = (n+1)/2$.
  - From Dirac's Theorem, the graph H has a Hamiltonian cycle.
  - Then, omitting vertex v it results to a Hamiltonian path.

Paths and Cycles …



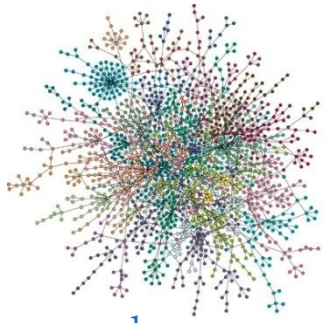
Hamiltonian Graphs

- **Theorem 4 (Ore 1960)**: If the simple graph G with $n \geq 3$ vertices and for all its non-adjacent vertices x and y it holds that $d(x) + d(y) \geq n$, then the graph G is Hamiltonian.

  Proof:

  - Let us assume that the proposition of the Theorem is false.
  - Then, there exist a non-Hamiltonian graph G with n vertices and the maximum number of edges, in order for our assumption to be satisfied.
    - If we add one extra edge (x, y) on this graph between the vertices x and y, then the graph $H = G + (x, y)$ will be Hamiltonian.
    - Since $n \geq 3$ it follows that the graph G is not complete.

Paths and Cycles …



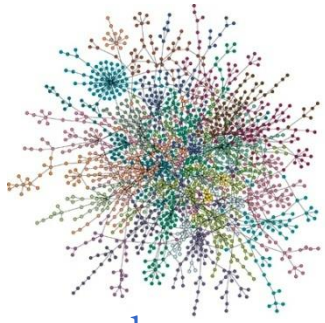
Hamiltonian Graphs

- **Theorem 4 (Ore 1960)**: If the simple graph G with $n \geq 3$ vertices and for all its non-adjacent vertices x and y it holds that $d(x) + d(y) \geq n$, then the graph G is Hamiltonian.

  Proof:

  - Let us assume that the proposition of the Theorem is false.
  - Let us assume that the graph H is Hamiltonian an each Hamiltonian cycle crosses edge (x, y).
    - In the graph G there exist a Hamiltonian path $P = (v_1, v_2, \dots, v_n)$, where $v_1 = x$ and $v_n = y$.
    - If in this graph G, vertex x is adjacent to v_i, then it is not possible vertex v_n to be adjacent to vertex v_{i-1}, because then graph G there would exist a Hamiltonian cycle $(x, v_i, v_{i+1}, \dots, v_n, v_{i-1}, v_{i-2}, \dots, v_1)$.

Paths and Cycles …



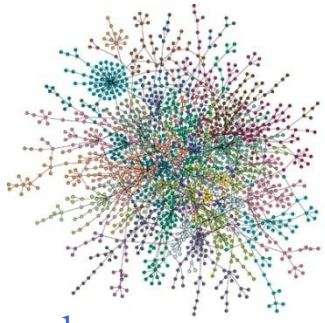
Hamiltonian Graphs

- **Theorem 4 (Ore 1960)**: If the simple graph G with $n \geq 3$ vertices and for all its non-adjacent vertices x and y it holds that $d(x) + d(y) \geq n$, then the graph G is Hamiltonian.

  Proof:

  - Let us assume that the proposition of the Theorem is false.
  - Let us assume that the graph H is Hamiltonian an each Hamiltonian cycle crosses edge (x, y).
    - Then, for each vertex of G that is adjacent to vertex x there exist a vertex in the graph $G - y$ that is not adjacent to vertex y.
    - Then, it holds: $d(y) \leq (n-1) - d(x)$, that is a contradiction.

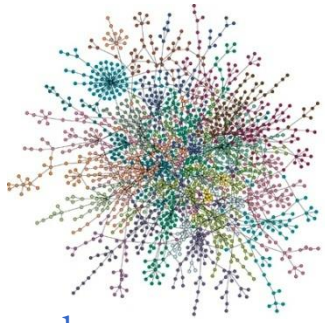
Paths and Cycles …



- **Hamiltonian Graphs**
  - **Theorem 5 (Bondy & Chvatal, 1976)**: Let G be a simple graph, where for two distinct non adjacent vertices x and y it holds that $d(x) + d(y) \geq n$. Then the graph G is Hamiltonian iff the graph $G + (x, y)$ is Hamiltonian.

    Proof:

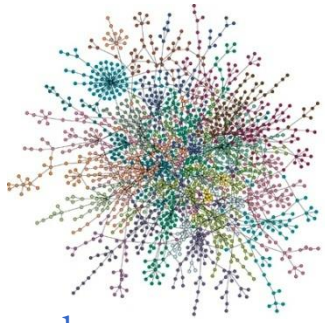
Paths and Cycles …



- **Hamiltonian Graphs**
  - **Theorem 5 (Bondy & Chvatal, 1976)**: Let G be a simple graph, where for two distinct non adjacent vertices x and y it holds that $d(x) + d(y) \geq n$. Then the graph G is Hamiltonian ***iff*** the graph $G + (x, y)$ is Hamiltonian.

    Proof:

Paths and Cycles …

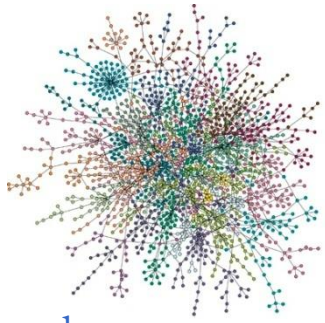


- **Hamiltonian Graphs**
  - **Theorem 5 (Bondy & Chvatal, 1976)**: Let G be a simple graph, where for two distinct non adjacent vertices x and y it holds that $d(x) + d(y) \geq n$. Then the graph G is Hamiltonian ***iff*** the graph $G + (x, y)$ is Hamiltonian.

    Proof:

    - ***iff*** → if and only if

Paths and Cycles …



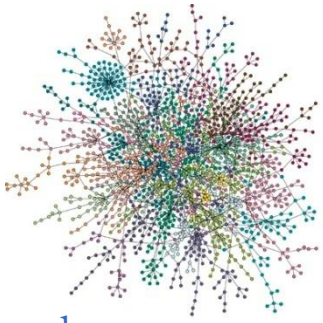
Hamiltonian Graphs

- **Theorem 5 (Bondy & Chvatal, 1976)**: Let G be a simple graph, where for two distinct non adjacent vertices x and y it holds that $d(x) + d(y) \geq n$. Then the graph G is Hamiltonian ***iff*** the graph $G + (x, y)$ is Hamiltonian.

  Proof:

  - ***iff*** → if and only if
  - Ore's Theorem utilizes the strong assumption that this relation holds for every pair of non adjacent vertices.

Paths and Cycles …



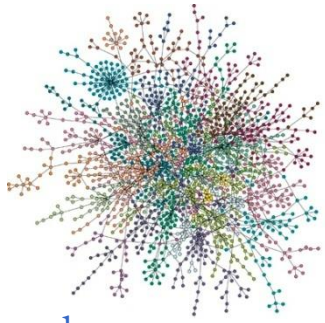
Hamiltonian Graphs

- **Theorem 5 (Bondy & Chvatal, 1976)**: Let G be a simple graph, where for two distinct non adjacent vertices x and y it holds that $d(x) + d(y) \geq n$. Then the graph G is Hamiltonian ***iff*** the graph $G + (x, y)$ is Hamiltonian.

  Proof:

  - ***iff*** → if and only if
  - Ore's Theorem utilizes the strong assumption that this relation holds for every pair of non adjacent vertices.
    - The "if" is obvious

Paths and Cycles …



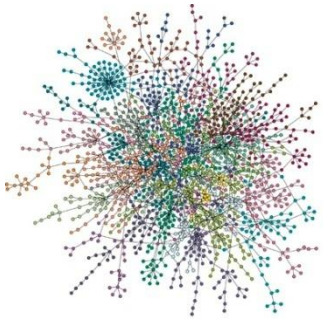
Hamiltonian Graphs

- **Theorem 5 (Bondy & Chvatal, 1976)**: Let G be a simple graph, where for two distinct non adjacent vertices x and y it holds that $d(x) + d(y) \geq n$. Then the graph G is Hamiltonian ***iff*** the graph $G + (x, y)$ is Hamiltonian.

  Proof:

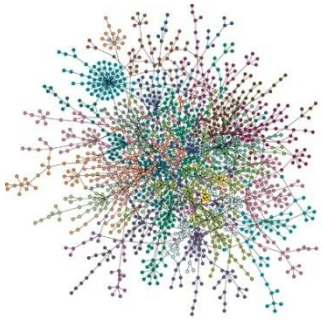
  - ***iff*** → if and only if
  - Ore's Theorem utilizes the strong assumption that this relation holds for every pair of non adjacent vertices.
    - The "if" is obvious
    - The "only if" is proven following the same process as in Ore's Theorem.

Paths and Cycles …



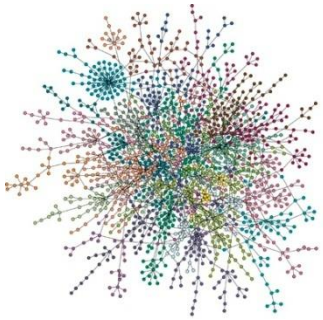
- **Discovering Hamiltonian Cycles**
  - Let a graph G with $n = 5$ vertices …
  - Define Matrices M_1 and M
  - After sequential $n - 1$ matrix multiplications and input concatenations result a matrix M^*
  - Check if the inputs of M^* are Hamiltonian paths /cycles.

Paths and Cycles …

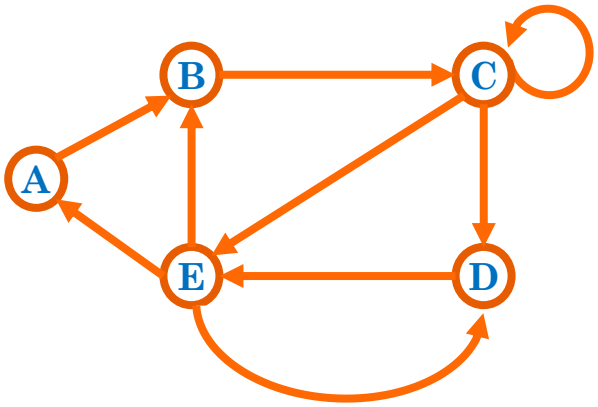


- **Discovering Hamiltonian Cycles**
  - Let a graph G with $n = 5$ vertices …
  - Define Matrices M_1 and M as follows:

Paths and Cycles …



- **Discovering Hamiltonian Cycles**
  - Let a graph G with $n = 5$ vertices …
  - Define Matrices M_1 and M as follows:



G

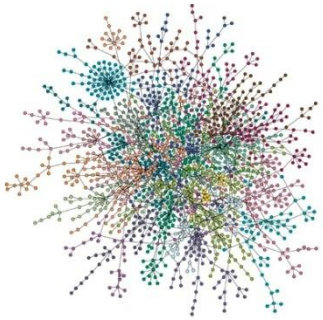
0	AB	0	0	0
0	0	BC	0	0
0	0	0	CD	CE
0	0	0	0	DE
EA	EB	0	ED	0

M_1

0	B	0	0	0
0	0	C	0	0
0	0	0	D	E
0	0	0	0	E
A	B	0	D	0

M

Paths and Cycles …

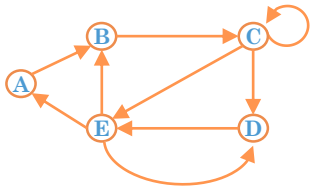


Discovering Hamiltonian Cycles

- Let a graph G with $n = 5$ vertices …
- Define Matrices M_1 and M as follows:
- Utilizing these matrices we proceed to sequential matrix multiplications in order to construct matrices: $M_j = M_{j-1} * M,\ 1 < j < n$, where, for each element (r, s) of M_j it holds that:

$$M_{j(r,s)} = \sum_{t=1}^{n} M_{j-1}(r,t) * M(t,s)$$

and the symbol " * " indicates the concatenation of the corresponding elements of the two matrices M_{j-1} and M, if both the elements are non-zero, while the symbol of the element of M is not included in the symbol of element in M_{j-1} (Hadamard Product).



G

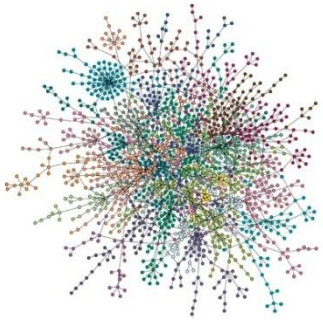
0	AB	0	0	0
0	0	BC	0	0
0	0	0	CD	CE
0	0	0	0	DE
EA	EB	0	ED	0

M_1

0	B	0	0	0
0	0	C	0	0
0	0	0	D	E
0	0	0	0	E
A	B	0	D	0

M

Paths and Cycles …



- **Discovering Hamiltonian Cycles**

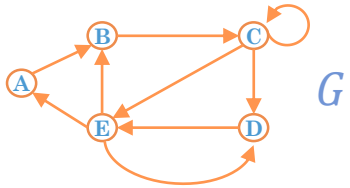
$$M_2 = M_1 * M$$

$$M_2(1,3) = \sum_{t=1}^{5} M_1(1,t) * M(t,3)$$

$M_1(1,:)$ → 0 AB 0 0 0

$M(:,3)$ → 0 C 0 0 0

M_2 (1,3) → 0 ABC 0 0 0



G

0	B	0	0	0
0	0	C	0	0
0	0	0	D	E
0	0	0	0	E
A	B	0	D	0

M

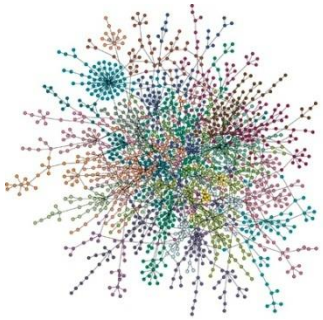
0	0	ABC	0	0
0	0	0	BCD	BCE
CEA	CEB	0	CED	CDE
DEA	DEB	0	0	0
0	EAB	EBC	0	0

M_2

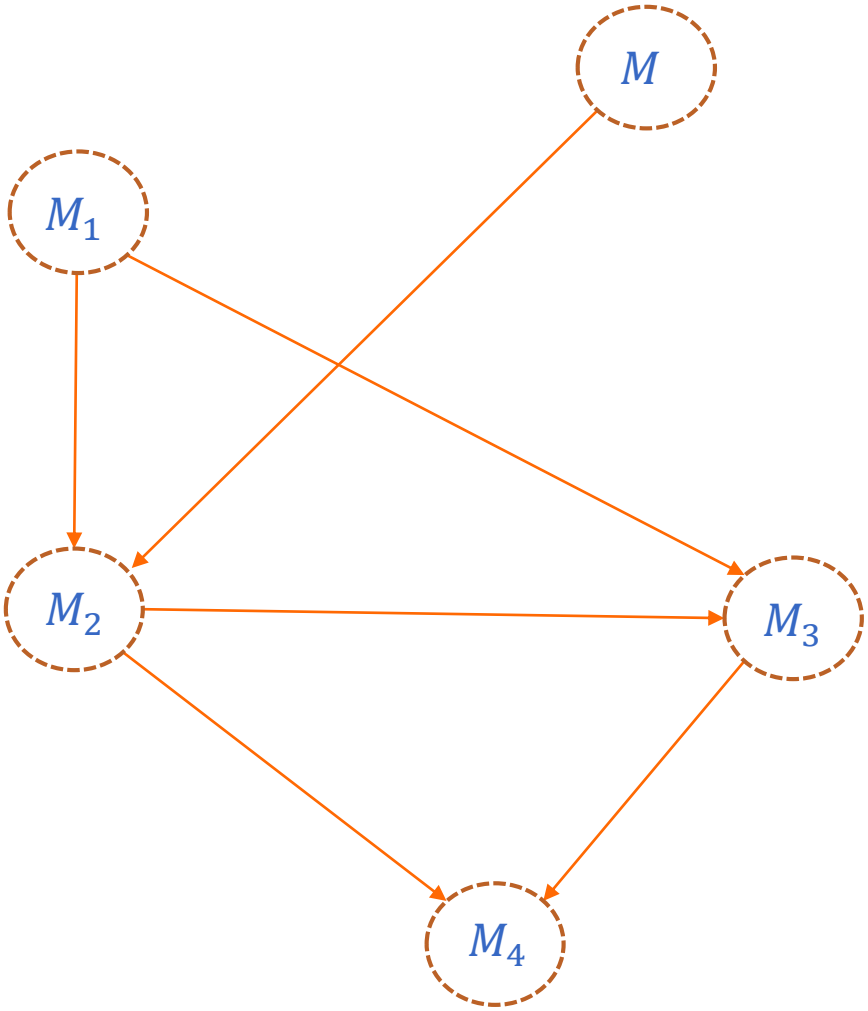
0	AB	0	0	0
0	0	BC	0	0
0	0	0	CD	CE
0	0	0	0	DE
EA	EB	0	ED	0

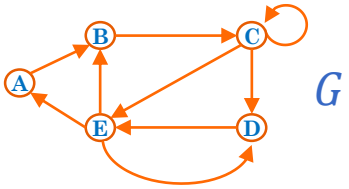
M_1

Paths and Cycles …

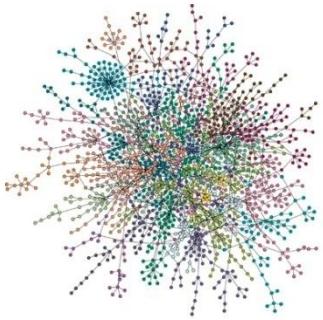


- **Discovering Hamiltonian Cycles**





Paths and Cycles …



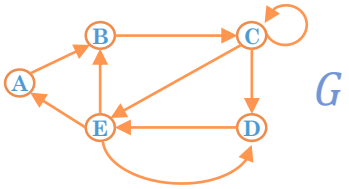
- **Discovering Hamiltonian Cycles**

0	AB	0	0	0
0	0	BC	0	0
0	0	0	CD	CE
0	0	0	0	DE
EA	EB	0	ED	0

M_1

M

0	B	0	0	0
0	0	C	0	0
0	0	0	D	E
0	0	0	0	E
A	B	0	D	0



G

0	0	ABC	0	0
0	0	0	BCD	BCE
CEA	CEB	0	CED	CDE
DEA	DEB	0	0	0
0	EAB	EBC	0	0

M_2

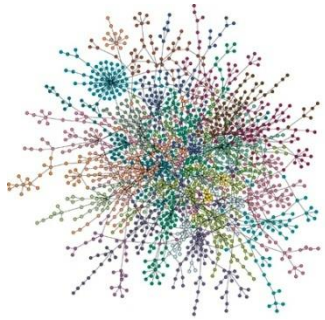
0	0	0	ABCD	ABCE
BCEA	0	0	BCED	BCDE
CDEA	CEAB CDEB	0	0	0
0	DEAB	DEBC	0	0
0	0	EABC	EBCD	0

M_3

0	0	0	ABCED	ABCDE
BCDEA	0	0	0	0
0	CDEAB	0	0	0
0	0	DEABC	0	0
0	0	0	EABCD	0

M_4

Paths and Cycles …



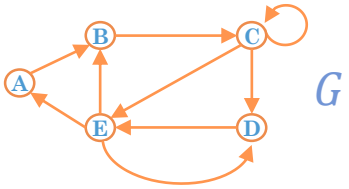
- **Discovering Hamiltonian Cycles**

0	AB	0	0	0
0	0	BC	0	0
0	0	0	CD	CE
0	0	0	0	DE
EA	EB	0	ED	0

M_1

M

0	B	0	0	0
0	0	C	0	0
0	0	0	D	E
0	0	0	0	E
A	B	0	D	0



G

0	0	ABC	0	0
0	0	0	BCD	BCE
CEA	CEB	0	CED	CDE
DEA	DEB	0	0	0
0	EAB	EBC	0	0

M_2

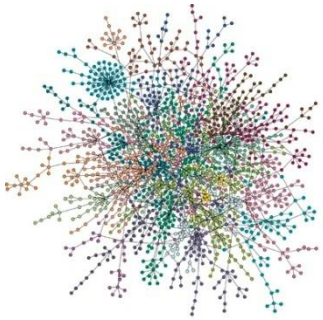
0	0	0	ABCD	ABCE
BCEA	0	0	BCED	BCDE
CDEA	CEAB CDEB	0	0	0
0	DEAB	DEBC	0	0
0	0	EABC	EBCD	0

M_3

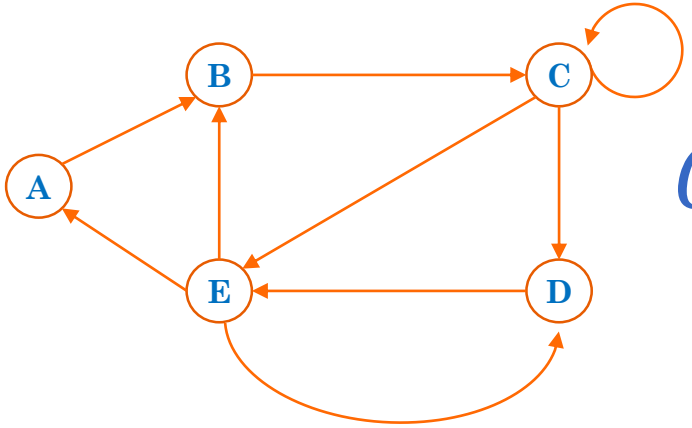
0	0	0	ABCED	ABCDE
BCDEA	0	0	0	0
0	CDEAB	0	0	0
0	0	DEABC	0	0
0	0	0	EABCD	0

M_4

Paths and Cycles …



- **Discovering Hamiltonian Cycles**



G

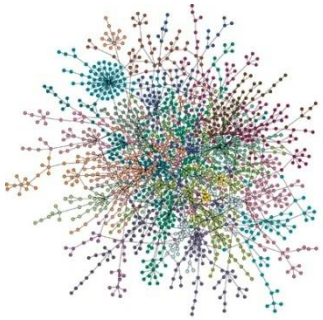
0	0	0	ABCED	ABCDE
BCDEA	0	0	0	0
0	CDEAB	0	0	0
0	0	DEABC	0	0
0	0	0	EABCD	0

M_4

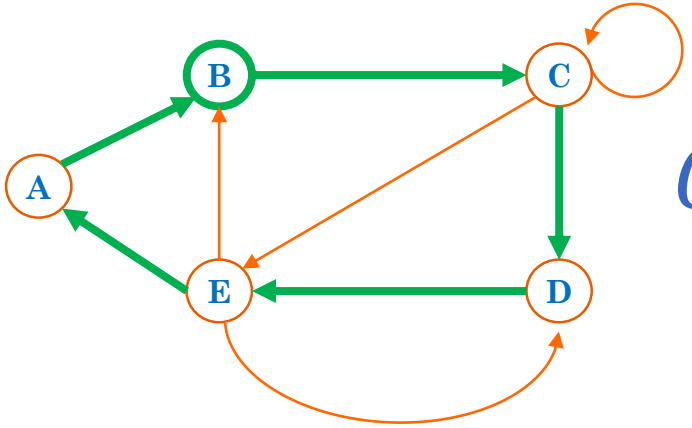
0	B	0	0	0
0	0	C	0	0
0	0	0	D	E
0	0	0	0	E
A	B	0	D	0

M

Paths and Cycles …



- **Discovering Hamiltonian Cycles**



G

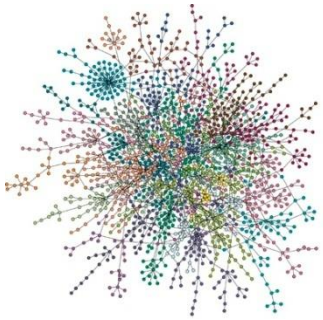
0	0	0	ABCED	ABCDE
BCDEA	0	0	0	0
0	CDEAB	0	0	0
0	0	DEABC	0	0
0	0	0	EABCD	0

M_4

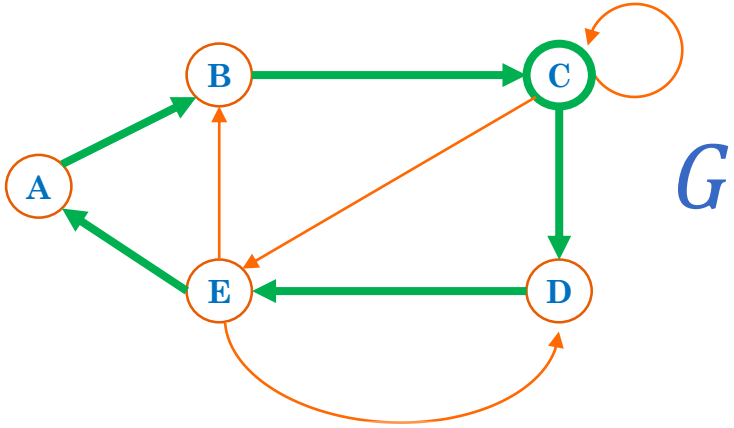
0	B	0	0	0
0	0	C	0	0
0	0	0	D	E
0	0	0	0	E
A	B	0	D	0

M

Paths and Cycles …



- **Discovering Hamiltonian Cycles**



G

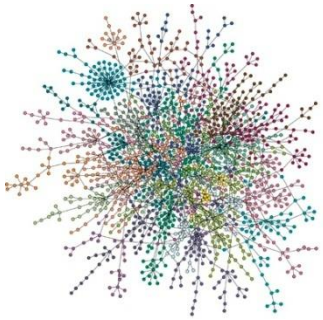
0	0	0	ABCED	ABCDE
BCDEA	0	0	0	0
0	CDEAB	0	0	0
0	0	DEABC	0	0
0	0	0	EABCD	0

M_4

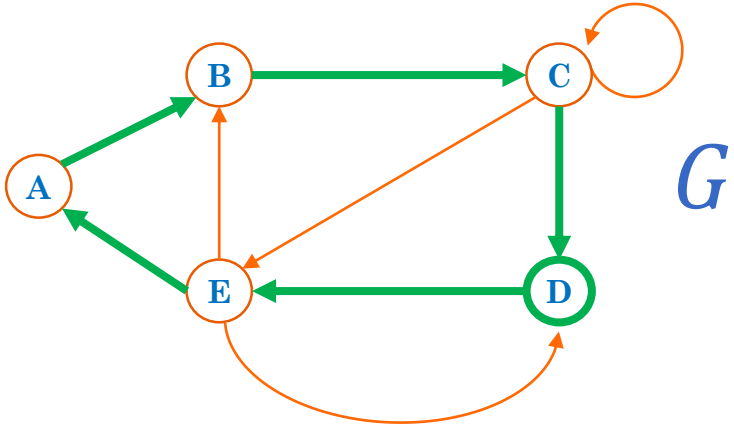
0	B	0	0	0
0	0	C	0	0
0	0	0	D	E
0	0	0	0	E
A	B	0	D	0

M

Paths and Cycles …



- **Discovering Hamiltonian Cycles**



G

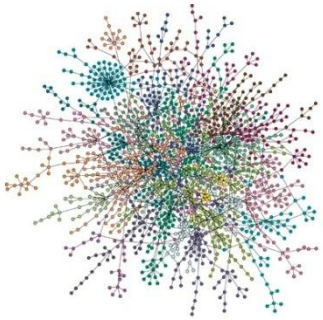
0	0	0	ABCED	ABCDE
BCDEA	0	0	0	0
0	CDEAB	0	0	0
0	0	DEABC	0	0
0	0	0	EABCD	0

M_4

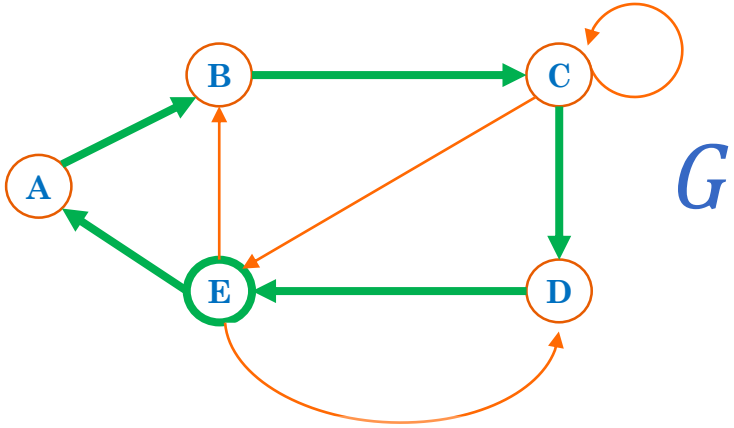
0	B	0	0	0
0	0	C	0	0
0	0	0	D	E
0	0	0	0	E
A	B	0	D	0

M

Paths and Cycles …



- **Discovering Hamiltonian Cycles**



G

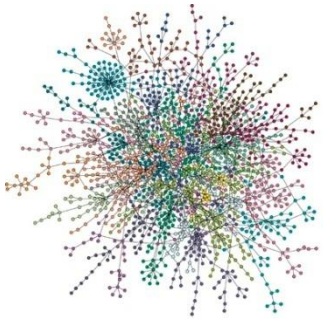
0	0	0	ABCED	ABCDE
BCDEA	0	0	0	0
0	CDEAB	0	0	0
0	0	DEABC	0	0
0	0	0	EABCD	0

M_4

0	B	0	0	0
0	0	C	0	0
0	0	0	D	E
0	0	0	0	E
A	B	0	D	0

M

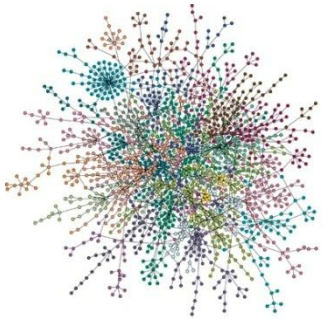
Paths and Cycles …



- **The Traveling Salesman Problem**
  - The "TSP" Problem:

    Given a list of cities and the distances between each pair of cities, what is the shortest possible route a salesman should take in order to visit each city and return to the origin city?

Paths and Cycles …



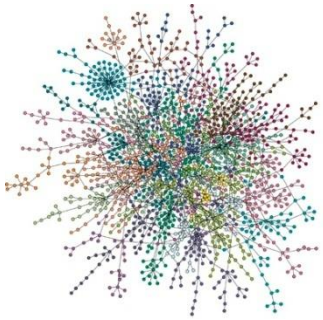
- **The Traveling Salesman Problem**
  - The "TSP" Problem:

    Given a list of cities and the distances between each pair of cities, what is the shortest possible route a salesman should take in order to visit each city and return to the origin city?

    - Passing only once from each city …
    - Passing more than once from each city …

Paths and Cycles …



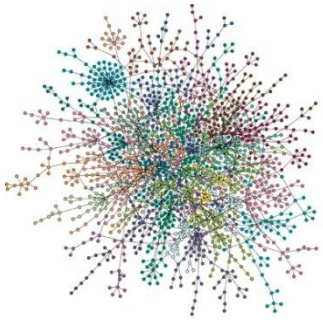
The Traveling Salesman Problem

- The "TSP" Problem:

  Given a list of cities and the distances between each pair of cities, what is the shortest possible route a salesman should take in order to visit each city and return to the origin city?

  - Passing only once from each city …
  - Passing more than once from each city …
    - If the graph G is an Euclidean graph (i.e., it holds the triangle inequality for every three vertices that form a triangle), it is not preferable for the salesman to pass more than once from the cities already visited.
    - The number of Hamiltonian cycles in an Euclidean graph is $(n-1)!/2$. Starting from an origin city, the salesman can move to $n-1$ cities, then to $n-2$ cities, to $n-3$,…, etc., while the destination counts twice.

Paths and Cycles …



The Traveling Salesman Problem

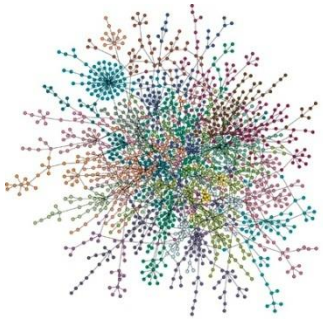
- The "TSP" Problem:

  Given a list of cities and the distances between each pair of cities, what is the shortest possible route a salesman should take in order to visit each city and return to the origin city?

  - Passing only once from each city …
  - Passing more than once from each city …
    - Brute Force on $(n-1)!/2$ Hamiltonian cycles $\rightarrow O(n^n)$
    - Dynamic Programming (Branch and Bound) $\rightarrow O(n^2\ 2^n)$

TSP ∈ NP – Complete

Paths and Cycles …

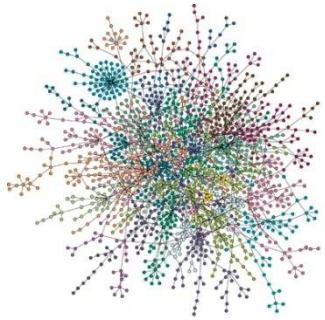


- **The Traveling Salesman Problem**
  - "TSP" Problem Definition:

    Given a complete undirected graph G, with weighted edges $w_{ij} \in N, \forall (i,j) \in E(G)$, find a Hamiltonian cycle W in G, such that the sum of the weights on the edges to be minimum?

$$\min \sum_{(i,j) \in W} w_{ij}$$

Paths and Cycles …

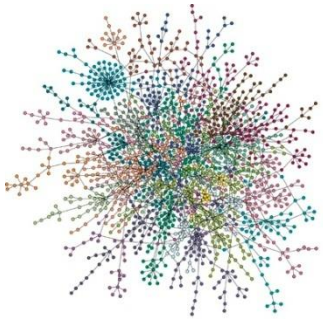


- **The Traveling Salesman Problem**
  - "TSP" Problem Definition:

    Given a complete undirected graph G, with weighted edges $w_{ij} \in N, \forall (i,j) \in E(G)$ and a number $k \in N$, find if there exists a Hamiltonian cycle W in G, such that the sum of the weights on the edges to be $\leq k$?

$$\sum_{(i,j)\in W} w_{ij} \leq k$$

Paths and Cycles …

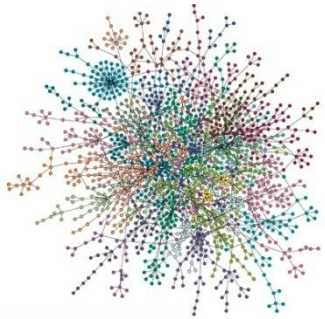


- **The Traveling Salesman Problem**
  - "TSP" Problem Definition:

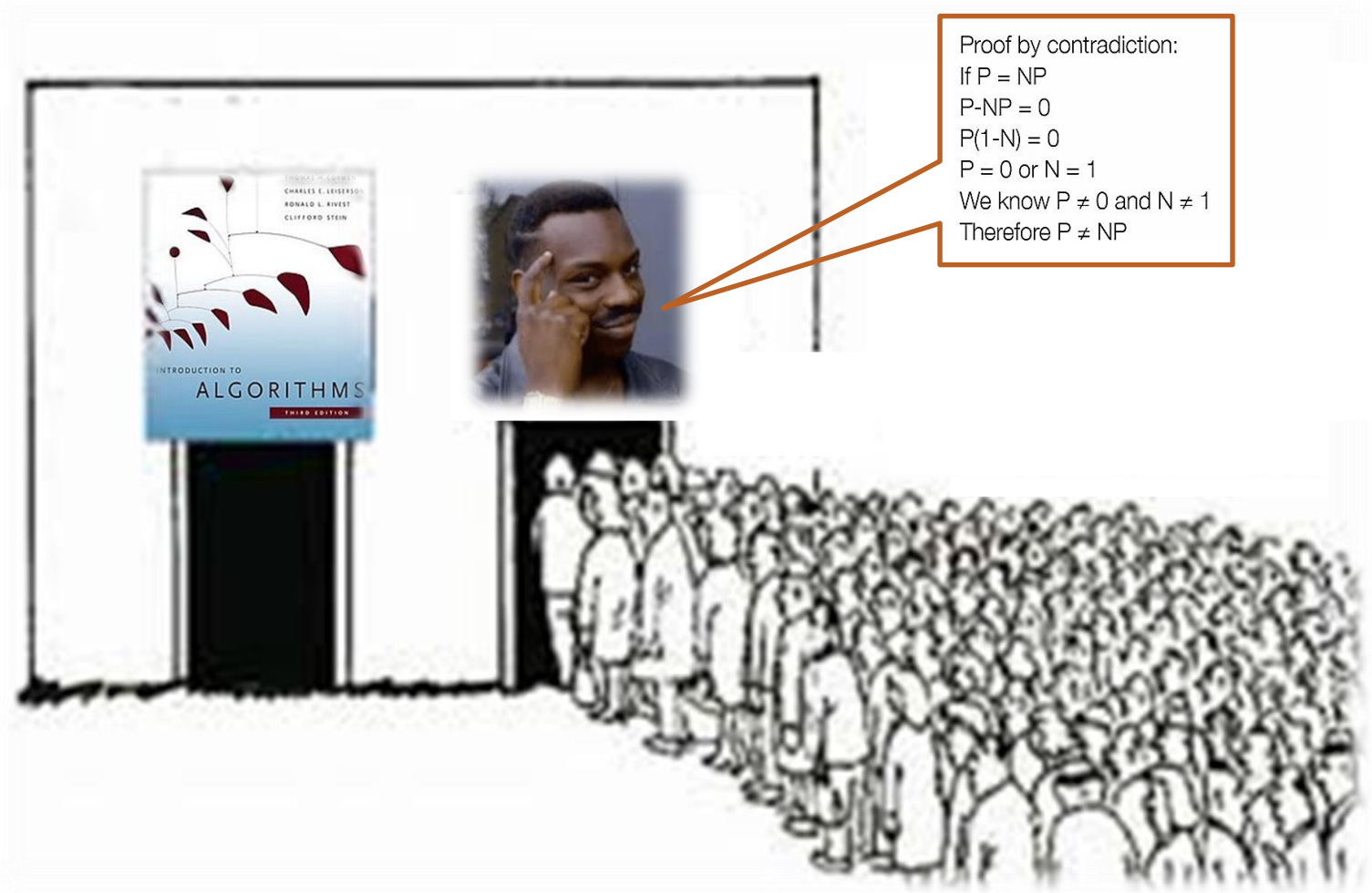
    Given a complete undirected graph G, with weighted edges $w_{ij} \in N, \forall (i,j) \in E(G)$ and a number $k \in N$, find if there exists a Hamiltonian cycle W in G, such that the sum of the weights on the edges to be $\leq k$?

$$\boldsymbol{TSP \in NP - Complete}$$

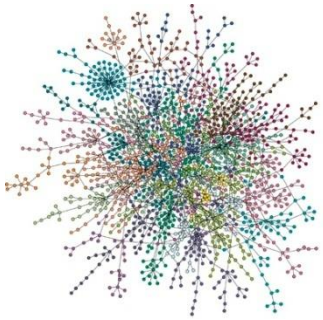
Paths and Cycles …



- **The Traveling Salesman Problem**



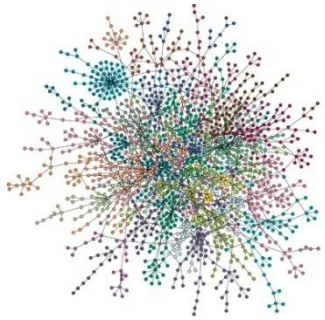
Paths and Cycles …



- **The Traveling Salesman Problem**

I can't find an efficient algorithm, but neither can all these famous people.

Paths and Cycles …



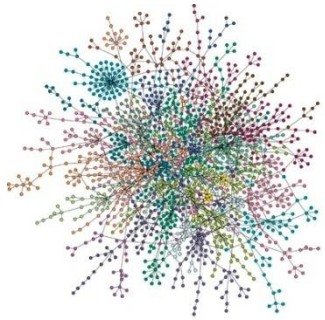
- **The Traveling Salesman Problem**
  - Reduction:

    The most important method (tool) to prove that a problem P belongs to a complexity class is the reduction?

    - Given a problem A we transform the instance p_1 of problem A to the instance p_2 of problem B, such that a solution to p_2 to provide a solution to p_1 (i.e., $A \leq B$ or $A \rightarrow B$).

$$\boldsymbol{TSP \in NP - Complete}$$

Paths and Cycles …



The Traveling Salesman Problem

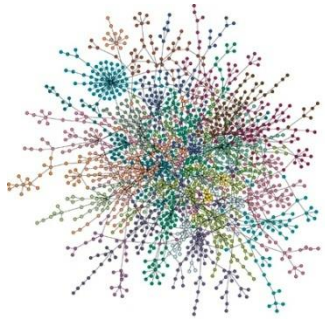
- Reduction:

  The most important method (tool) to prove that a problem P belongs to a complexity class is the reduction?

  A ≤ B or A→B (R reduction)

  - If p_1 is a "YES" instance of problem A, then the R reduction will produce a "YES" instance p_2 (i.e., $p_2 = R(p_1)$) of problem B.
  - If p_1 is a "NO" instance of problem A, then the R reduction will produce a "NO" instance p_2 (i.e., $p_2 = R(p_1)$) of problem B.
  - Utilizing set-notation we should prove that the R reduction is of polynomial time and that $p_1 \in A \ iff \ p_2 \in B$ (A and B are sets of instances having the property P, i.e., "YES" - satisfiability of P)
  - Then we say that the problem A is reduced to B, or equivalently that we can utilize the solution of B to solve A.

Paths and Cycles …

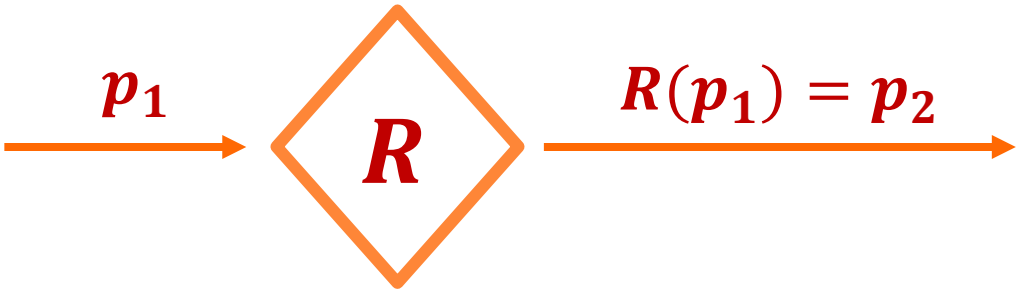


- **The Traveling Salesman Problem**
  - Reduction:

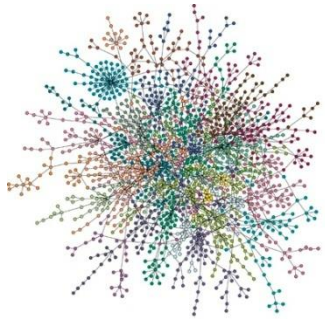
    The most important method (tool) to prove that a problem P belongs to a complexity class is the reduction?

    $A \leq B$ or $A \rightarrow B$ (R reduction)

    - Lemma: If problem A is reduced to B in polynomial time, and problem B can be solved in polynomial time, then problem A can be solved in polynomial time.



Paths and Cycles …

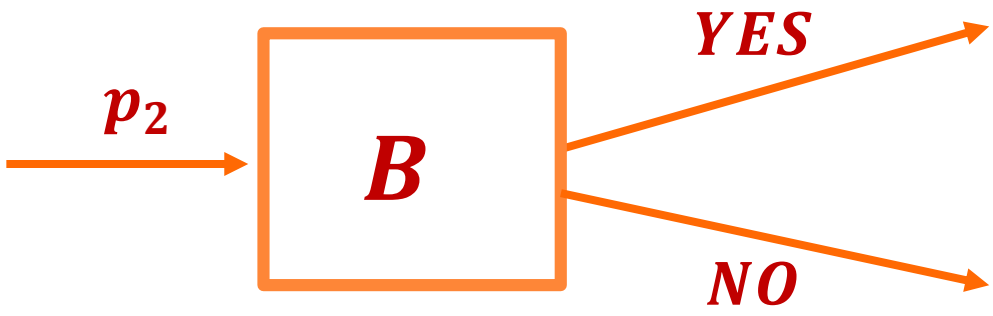


- **The Traveling Salesman Problem**
  - Reduction:

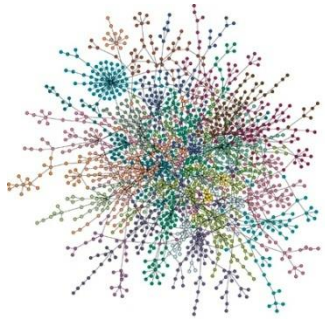
    The most important method (tool) to prove that a problem P belongs to a complexity class is the reduction?

    $A \leq B$ or $A \rightarrow B$ (R reduction)

    - Lemma: If problem A is reduced to B in polynomial time, and problem B can be solved in polynomial time, then problem A can be solved in polynomial time.



Paths and Cycles …

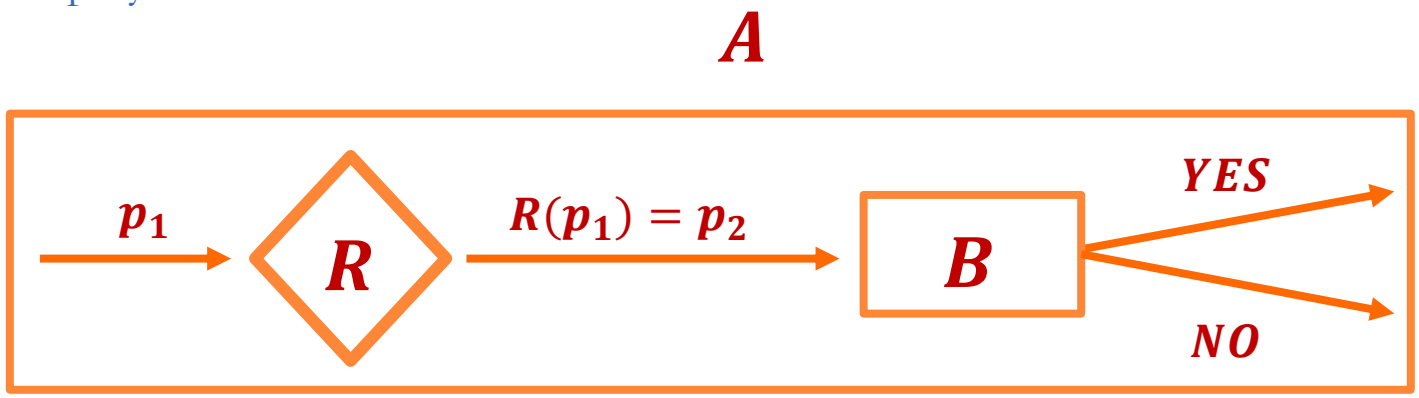


- **The Traveling Salesman Problem**
  - Reduction:

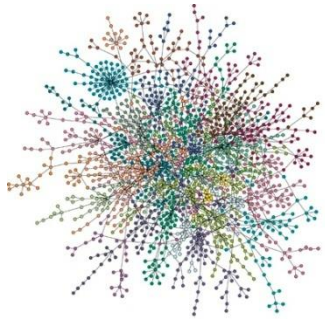
    The most important method (tool) to prove that a problem P belongs to a complexity class is the reduction?

    $A \leq B$ or $A \rightarrow B$ (R reduction)

    - Lemma: If problem A is reduced to B in polynomial time, and problem B can be solved in polynomial time, then problem A can be solved in polynomial time.



Paths and Cycles …



The Traveling Salesman Problem

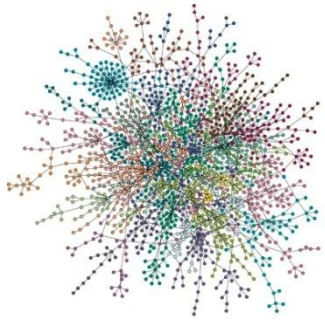
- Reduction:

  The most important method (tool) to prove that a problem P belongs to a complexity class is the reduction?

  $A \leq B$ or $A \rightarrow B$ (R reduction)

  - Reversing the Lemma: If problem A is reduced to B in polynomial time, and problem A can be solved in exponential time, then problem B should be solved in exponential time.
  - This form of reduction we will deploy in order to prove that many problems of graph theory, like TSP in our case, are hard to be solved.

Paths and Cycles …



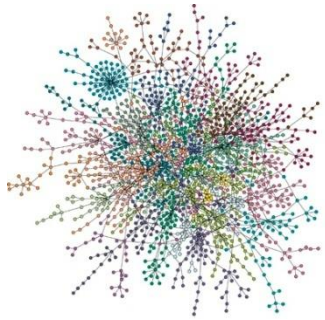
- **The Traveling Salesman Problem**
  - Theorem (TSP ∈ NP-Comlpete):

    Given a complete undirected graph G, with weighted edges $w_{ij} \in N, \forall (i,j) \in E(G)$ and a number $k \in N$, find if there exists a Hamiltonian cycle W in G, such that the sum of the weights on the edges to be $\leq k$?

$$\sum_{(i,j)\in W} w_{ij} \leq k$$

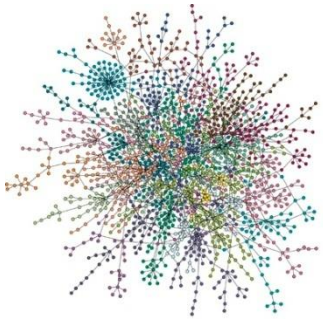
1. The TSP problem belongs to NP class
2. There exists a known to be NP-Complete problem A and $A \leq TSP$

Paths and Cycles …

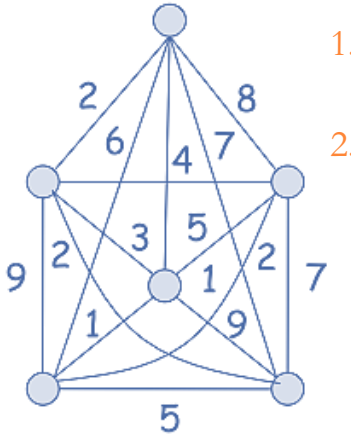


- **The Traveling Salesman Problem**
  1. The TSP problem belongs to NP class
     - We will construct a polynomial verifier
     - The verifier (i.e., an algorithm for the problem) has as input a weighted graph G and an integer number k, while the certificate is a cycle C.
       1. First, the cycle has to be checked if indeed is Hamiltonian
       2. If not, then it returns "NO" and terminates
       3. Else, it checks if the sum of the weights of C's edges is $\leq k$
          1. If it is not truth, then it returns "NO" and terminates,
          2. Else, it returns "YES"
     - The steps followed in the verifier are implemented in polynomial time.

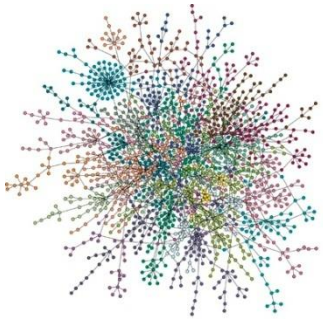
Paths and Cycles …



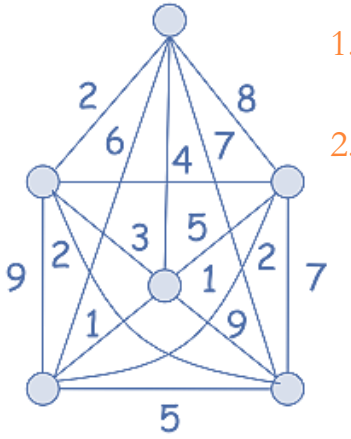
- **The Traveling Salesman Problem**
  1. The TSP problem belongs to NP class
     - We will construct a polynomial verifier
     - The verifier (i.e., an algorithm for the problem) has as input a weighted graph G and an integer number k, while the certificate is a cycle C.
       1. First, the cycle has to be checked if indeed is Hamiltonian
       2. If not, then it returns "NO" and terminates
       3. Else, it checks if the sum of the weights of C's edges is $\leq k$
          1. If it is not truth, then it returns "NO" and terminates,
          2. Else, it returns "YES"

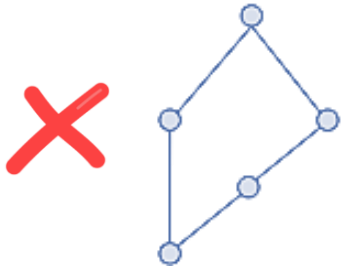


Paths and Cycles …

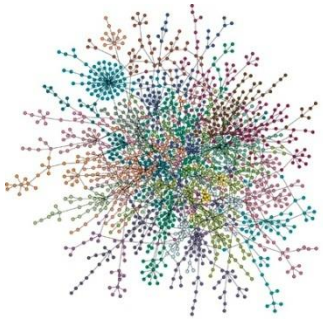


- **The Traveling Salesman Problem**
  1. The TSP problem belongs to NP class
     - We will construct a polynomial verifier
     - The verifier (i.e., an algorithm for the problem) has as input a weighted graph G and an integer number k, while the certificate is a cycle C.
       1. **First, the cycle has to be checked if indeed is Hamiltonian**
       2. **If not, then it returns "NO" and terminates**
       3. Else, it checks if the sum of the weights of C's edges is $\leq k$
          1. If it is not truth, then it returns "NO" and terminates,
          2. Else, it returns "YES"

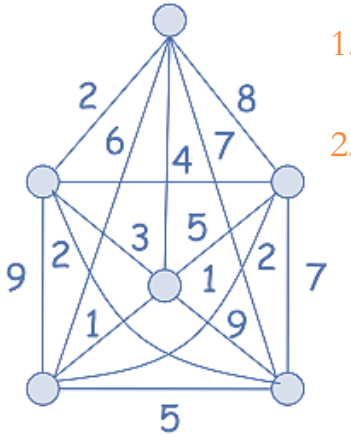


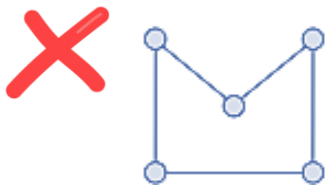


Paths and Cycles …

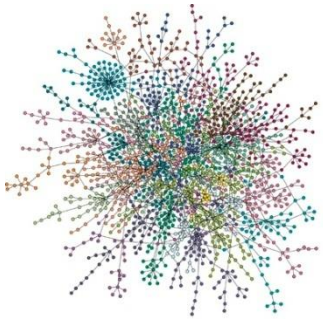


- **The Traveling Salesman Problem**
  1. The TSP problem belongs to NP class
     - We will construct a polynomial verifier
     - The verifier (i.e., an algorithm for the problem) has as input a weighted graph G and an integer number k, while the certificate is a cycle C.
       1. **First, the cycle has to be checked if indeed is Hamiltonian**
       2. **If not, then it returns "NO" and terminates**
       3. Else, it checks if the sum of the weights of C's edges is $\leq k$
          1. If it is not truth, then it returns "NO" and terminates,
          2. Else, it returns "YES"

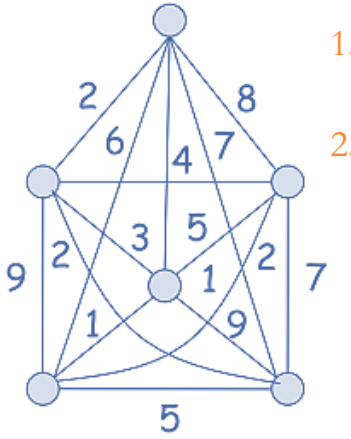


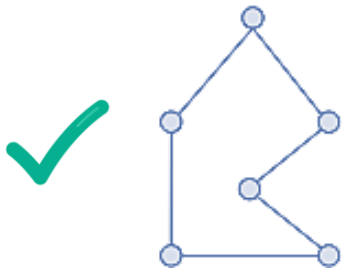


Paths and Cycles …

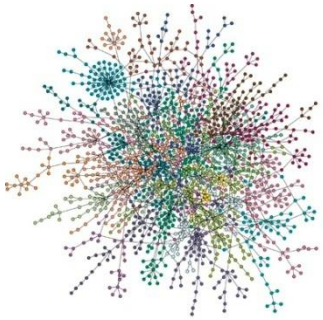


- **The Traveling Salesman Problem**
  1. The TSP problem belongs to NP class
     - We will construct a polynomial verifier
     - The verifier (i.e., an algorithm for the problem) has as input a weighted graph G and an integer number k, while the certificate is a cycle C.
       1. **First, the cycle has to be checked if indeed is Hamiltonian**
       2. If not, then it returns "NO" and terminates
       3. Else, it checks if the sum of the weights of C's edges is $\leq k$
          1. If it is not truth, then it returns "NO" and terminates,
          2. Else, it returns "YES"

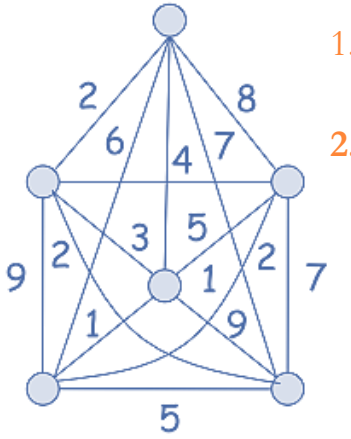


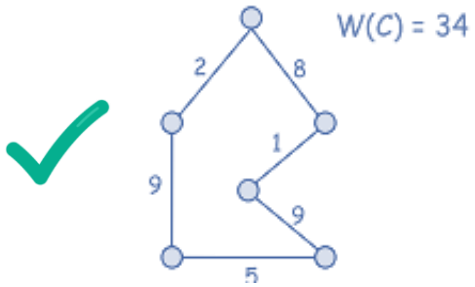


Paths and Cycles …

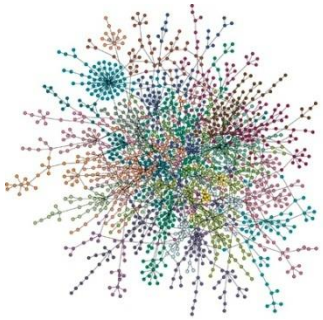


- **The Traveling Salesman Problem**
  1. The TSP problem belongs to NP class
     - We will construct a polynomial verifier
     - The verifier (i.e., an algorithm for the problem) has as input a weighted graph G and an integer number k, while the certificate is a cycle C.
       1. **First, the cycle has to be checked if indeed is Hamiltonian**
       2. If not, then it returns "NO" and terminates
       3. **Else, it checks if the sum of the weights of C's edges is $\leq k$**
          1. If it is not truth, then it returns "NO" and terminates,
          2. **Else, it returns "YES"**

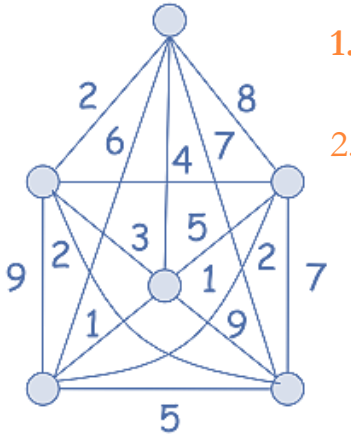


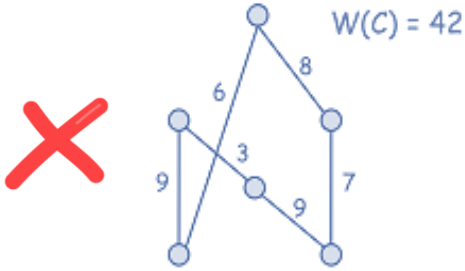


Paths and Cycles …

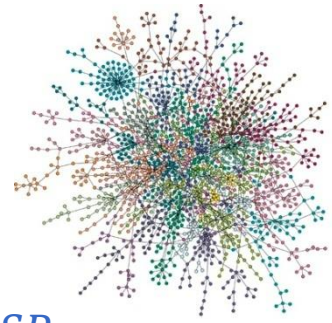


- **The Traveling Salesman Problem**
  1. The TSP problem belongs to NP class
     - We will construct a polynomial verifier
     - The verifier (i.e., an algorithm for the problem) has as input a weighted graph G and an integer number k, while the certificate is a cycle C.
       1. **First, the cycle has to be checked if indeed is Hamiltonian**
       2. If not, then it returns "NO" and terminates
       3. **Else, it checks if the sum of the weights of C's edges is $\leq k$**
          1. **If it is not truth, then it returns "NO" and terminates,**
          2. Else, it returns "YES"



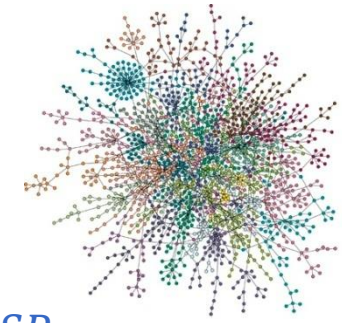


Paths and Cycles …



- **The Traveling Salesman Problem**
  2. There exists a known to be NP-Complete problem A and $A \leq TSP$
     - We select A =Hamiltonian cycle (HAMC)
     - Hamiltonian cycle in a simple connected graph G is a closed path that passes from each vertex of G.
     - HAMC $\in$ NP-Complete
     - HAMC $\leq$ TSP

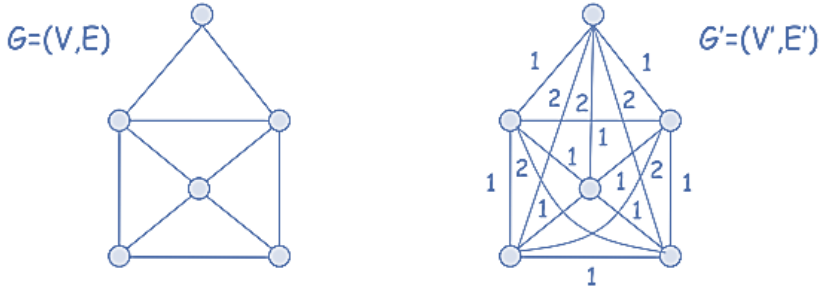
Paths and Cycles …



- **The Traveling Salesman Problem**
  2. There exists a known to be NP-complete problem A and $A \leq TSP$
     - Let p_1 an instance of the HAMC by the graph $G = (V, E)$
     - We construct a new complete weighted graph $G' = (V', E')$: $V' = V$ and define a weight function $w: E' \rightarrow \{1,2\}$ such that

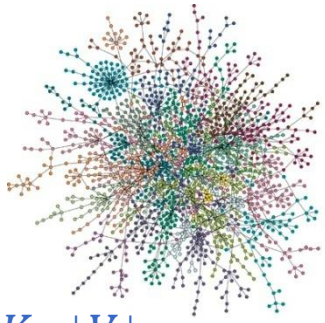
$$\forall (i,j) \in E', w_{ij} = \begin{cases} 1, if\ (i,j) \in E \\ 2, if\ (i,j) \notin E \end{cases}$$

     - Finally we set $k = |V|$



     - In order to complete the reduction we should prove that the graph G has HAMC iff the produced graph G' has HAMC $\leq K = |V|$

Paths and Cycles …

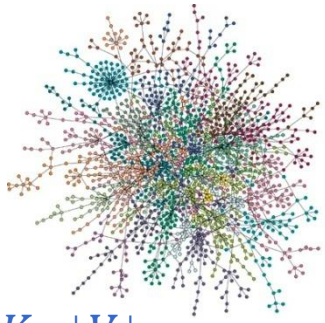


- **The Traveling Salesman Problem**
  2. The graph G has HAMC iff the produce graph G' has HAMC $\leq K = |V|$

     $(\Longrightarrow)$

     - Let G has a Hamiltonian cycle W.
     - Then, G' has also the Hamiltonian cycle W since it has the same vertex set with G while $E \subseteq E'$.
     - Since all the vertices of W in G' have weight 1 it follows that cycle W has weight in G' equal to $k = |V|$.

Paths and Cycles …

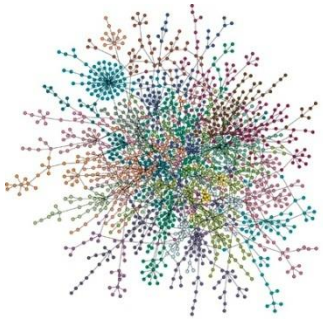


- **The Traveling Salesman Problem**
  2. The graph G has HAMC iff the produce graph G' has HAMC $\leq K = |V|$

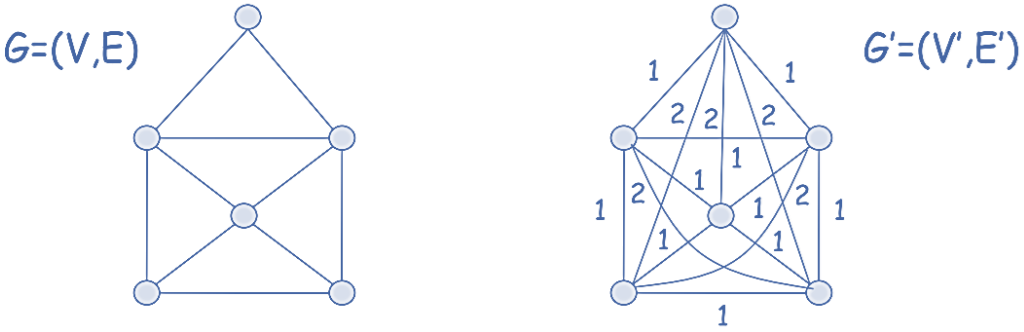
     $(\Leftarrow)$

     - Let G has a Hamiltonian cycle W with weight $\leq k = |V|$.
     - Then, the cycle W can not include any of the edges of E' that do not belong to E, as they have weight 2 and hence the $|V|$ edges of W can not have total weight $\leq |V|$.
     - Hence, the initial graph G has a Hamiltonian cycle W

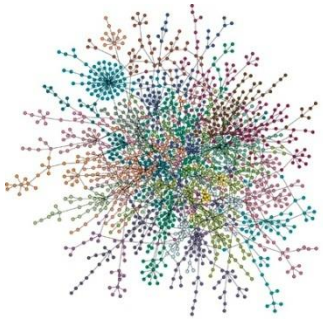
Paths and Cycles …



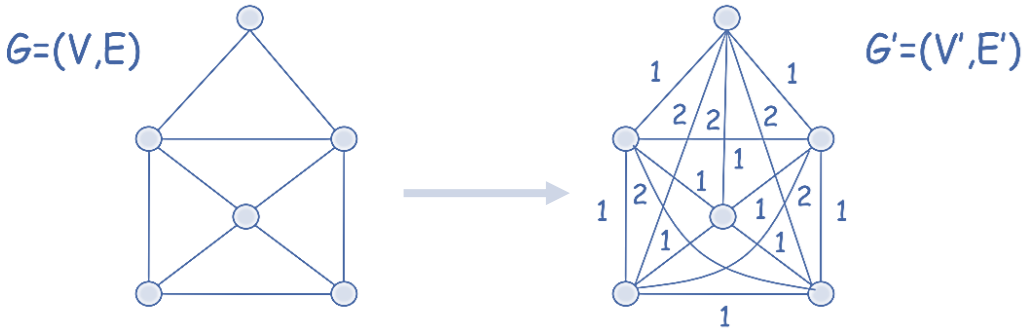
- **The Traveling Salesman Problem**
  2. The correctness of the reduction HAMC ≤ TSP has been proven.
     - Now it lefts to prove that the complexity of the reduction is polynomial.



Paths and Cycles …



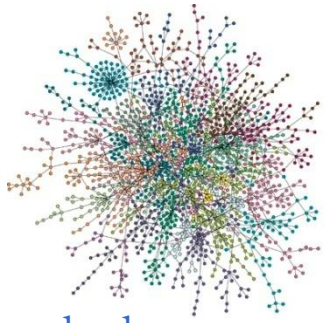
- **The Traveling Salesman Problem**
  2. The correctness of the reduction HAMC ≤ TSP has been proven.
     - Now it lefts to prove that the complexity of the reduction is polynomial.



     - Indeed the graph G' can be produced from G in polynomial time!
     - Hence, since
       - $TSP \in NP$ and
       - $HAMC \leq TSP$

       $\Rightarrow TSP \in NP - complete$

Paths and Cycles …



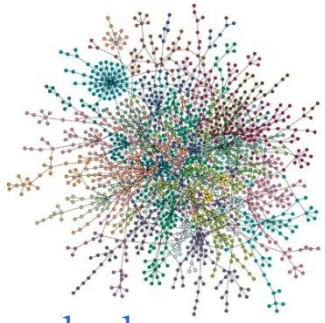
- **Decision TSP & Optimization TSP**
  - **Decision TSP**: Given a complete undirected graph G, with weighted edges $w_{ij} \in N, \forall (i,j) \in E(G)$ and a number $k \in N$, find if there exists a Hamiltonian cycle W in G, such that the sum of the weights on the edges to be $\leq k$?

$$\sum_{(i,j)\in W} w_{ij} \leq k$$

  - **Optimization TSP**: Given a complete undirected graph G, with weighted edges $w_{ij} \in N, \forall (i,j) \in E(G)$, find a Hamiltonian cycle W in G, such that the sum of the weights on the edges to be minimum?

$$\min \sum_{(i,j)\in W} w_{ij}$$

Paths and Cycles …

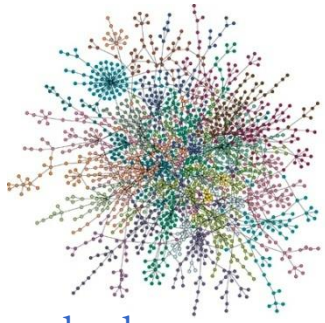


- **Decision TSP & Optimization TSP**
  - **Decision TSP**: Given a complete undirected graph G, with weighted edges $w_{ij} \in N, \forall (i,j) \in E(G)$ and a number $k \in N$, find if there exists a Hamiltonian cycle W in G, such that the sum of the weights on the edges to be $\leq k$?

$$\sum_{(i,j)\in W} w_{ij} \leq k$$

  - The Traveling Salesman Problem has been proven to be NP-Complete.
  - There does not exist an algorithm that solves the Decision TSP in polynomial time, and hence neither for the Optimization TSP.

Paths and Cycles …

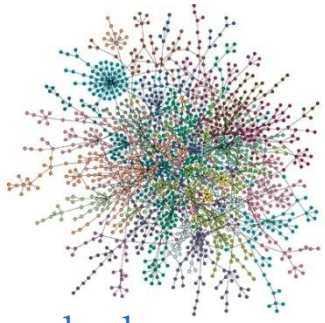


- **Decision TSP & Optimization TSP**
  - **Decision TSP**: Given a complete undirected graph G, with weighted edges $w_{ij} \in N, \forall (i,j) \in E(G)$ and a number $k \in N$, find if there exists a Hamiltonian cycle W in G, such that the sum of the weights on the edges to be $\leq k$?

$$\sum_{(i,j)\in W} w_{ij} \leq k$$

  - The Traveling Salesman Problem has been proven to be NP-complete.
  - There does not exist an algorithm that solves the Decision TSP in polynomial time, and hence neither for the Optimization TSP.
  - Why?

Paths and Cycles …

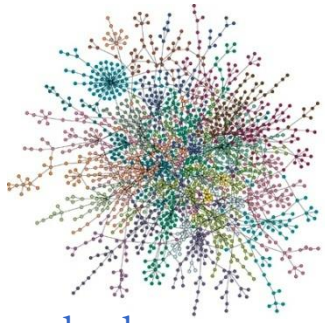


- **Decision TSP & Optimization TSP**
  - **Decision TSP**: Given a complete undirected graph G, with weighted edges $w_{ij} \in N, \forall (i,j) \in E(G)$ and a number $k \in N$, find if there exists a Hamiltonian cycle W in G, such that the sum of the weights on the edges to be $\leq k$?

$$\sum_{(i,j)\in W} w_{ij} \leq k$$

  - The Traveling Salesman Problem has been proven to be NP-complete.
  - There does not exist an algorithm that solves the Decision TSP in polynomial time, and hence neither for the Optimization TSP.
  - If we design an algorithm that solves the Decision TSP in polynomial time, would it solve the Optimization TSP in polynomial time too?

Paths and Cycles …



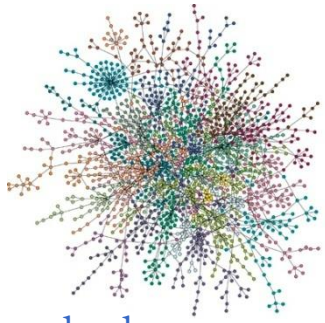
- **Decision TSP & Optimization TSP**
  - **Decision TSP**: Given a complete undirected graph G, with weighted edges $w_{ij} \in N, \forall (i,j) \in E(G)$ and a number $k \in N$, find if there exists a Hamiltonian cycle W in G, such that the sum of the weights on the edges to be $\leq k$?

$$\sum_{(i,j)\in W} w_{ij} \leq k$$

  - Having an algorithm that solves the Decision TSP in polynomial time, would it solve the Optimization TSP in polynomial time too?

  - If we had an algorithm let A_d for the Decision TSP problem, then it would return "YES" if there would exist Hamiltonian cycle of weight $\leq k$, or it would return "NO" otherwise.

Paths and Cycles …



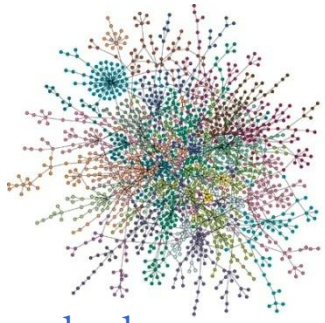
- **Decision TSP & Optimization TSP**
  - **Decision TSP**: Given a complete undirected graph G, with weighted edges $w_{ij} \in N, \forall (i,j) \in E(G)$ and a number $k \in N$, find if there exists a Hamiltonian cycle W in G, such that the sum of the weights on the edges to be $\leq k$?

$$\sum_{(i,j)\in W} w_{ij} \leq k$$

  - Having an algorithm that solves the Decision TSP in polynomial time, would it solve the Optimization TSP in polynomial time too?

  - Let M the maximum weight of the edges of graph G, we search the value of k in the range $[0, nM]$ and apply the algorithm A_d

Paths and Cycles …



Decision TSP & Optimization TSP

- **Decision TSP**: Given a complete undirected graph G, with weighted edges $w_{ij} \in N, \forall (i,j) \in E(G)$ and a number $k \in N$, find if there exists a Hamiltonian cycle W in G, such that the sum of the weights on the edges to be $\leq k$?

$$\sum_{(i,j)\in W} w_{ij} \leq k$$

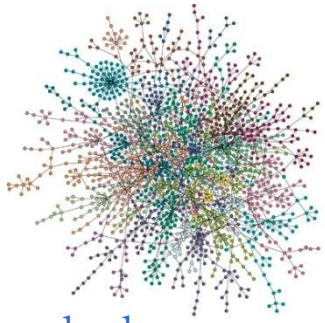
- Having an algorithm that solves the Decision TSP in polynomial time, would it solve the Optimization TSP in polynomial time too?

- Algorithm A_{opt} (Linear Search)

$$[\mathbf{0}, 1, 2, 3, \dots, m-1, m, m+1, \dots, nM-1, nM]$$

- $k = 0$, If A_d returns "NO", then $k++$

Paths and Cycles …



- **Decision TSP & Optimization TSP**
  - **Decision TSP**: Given a complete undirected graph G, with weighted edges $w_{ij} \in N, \forall (i,j) \in E(G)$ and a number $k \in N$, find if there exists a Hamiltonian cycle W in G, such that the sum of the weights on the edges to be $\leq k$?

$$\sum_{(i,j)\in W} w_{ij} \leq k$$

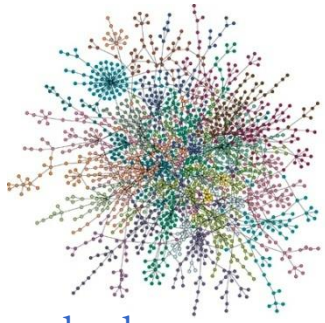
  - Having an algorithm that solves the Decision TSP in polynomial time, would it solve the Optimization TSP in polynomial time too?

  - Algorithm A_{opt} (Linear Search)

$$[0, \mathbf{1}, 2, 3, \dots, m-1, m, m+1, \dots, nM-1, nM]$$

  - $k = 1$, If A_d returns "NO", then $k++$

Paths and Cycles …



Decision TSP & Optimization TSP

- **Decision TSP**: Given a complete undirected graph G, with weighted edges $w_{ij} \in N, \forall (i,j) \in E(G)$ and a number $k \in N$, find if there exists a Hamiltonian cycle W in G, such that the sum of the weights on the edges to be $\leq k$?

$$\sum_{(i,j)\in W} w_{ij} \leq k$$

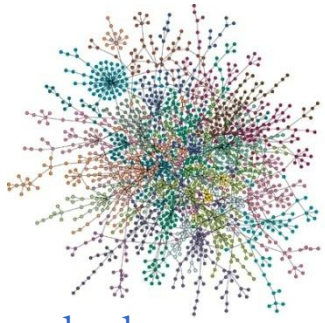
- Having an algorithm that solves the Decision TSP in polynomial time, would it solve the Optimization TSP in polynomial time too?

- Algorithm A_{opt} (Linear Search)

$$[0, 1, \mathbf{2}, 3, \dots, m-1, m, m+1, \dots, nM-1, nM]$$

- $k = 2$, If A_d returns "NO", then $k++$

Paths and Cycles …



- **Decision TSP & Optimization TSP**
  - **Decision TSP**: Given a complete undirected graph G, with weighted edges $w_{ij} \in N, \forall (i,j) \in E(G)$ and a number $k \in N$, find if there exists a Hamiltonian cycle W in G, such that the sum of the weights on the edges to be $\leq k$?

$$\sum_{(i,j)\in W} w_{ij} \leq k$$

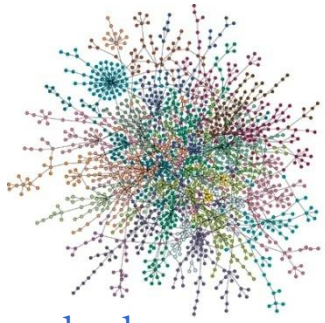
  - Having an algorithm that solves the Decision TSP in polynomial time, would it solve the Optimization TSP in polynomial time too?

  - Algorithm A_{opt} (Linear Search)

$$[0, 1, 2, 3, \dots, \boldsymbol{m-1}, m, m+1, \dots, nM-1, nM]$$

  - $k = m - 1$, If A_d returns "NO", then $k++$

Paths and Cycles …



- **Decision TSP & Optimization TSP**
  - **Decision TSP**: Given a complete undirected graph G, with weighted edges $w_{ij} \in N, \forall (i,j) \in E(G)$ and a number $k \in N$, find if there exists a Hamiltonian cycle W in G, such that the sum of the weights on the edges to be $\leq k$?

$$\sum_{(i,j)\in W} w_{ij} \leq k$$

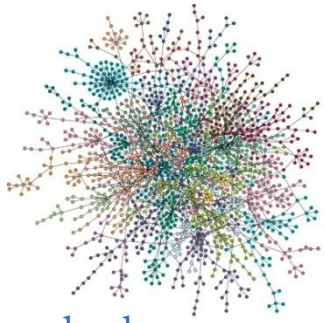
  - Having an algorithm that solves the Decision TSP in polynomial time, would it solve the Optimization TSP in polynomial time too?

  - Algorithm A_{opt} (Linear Search)

$$[0, 1, 2, 3, \dots, m-1, \boldsymbol{m}, m+1, \dots, nM-1, nM]$$

  - $k = m$, If A_d returns "YES", then $exit$

Paths and Cycles …



- **Decision TSP & Optimization TSP**
  - **Decision TSP**: Given a complete undirected graph G, with weighted edges $w_{ij} \in N, \forall (i,j) \in E(G)$ and a number $k \in N$, find if there exists a Hamiltonian cycle W in G, such that the sum of the weights on the edges to be $\leq k$?

$$\sum_{(i,j)\in W} w_{ij} \leq k$$

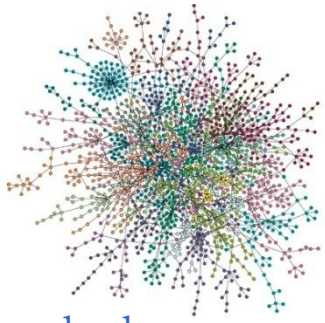
  - Having an algorithm that solves the Decision TSP in polynomial time, would it solve the Optimization TSP in polynomial time too?

  - Algorithm A_{opt} (Linear Search)

$$[0, 1, 2, 3, \dots, m-1, m, m+1, \dots, nM-1, nM]$$

  - Algorithm A_d will be executed at most $O(nM)$ times (worst case)
    Hence, if A_d is polynomial then A_{opt} is polynomial too.

Paths and Cycles …



- **Decision TSP & Optimization TSP**
  - **Decision TSP**: Given a complete undirected graph G, with weighted edges $w_{ij} \in N, \forall (i,j) \in E(G)$ and a number $k \in N$, find if there exists a Hamiltonian cycle W in G, such that the sum of the weights on the edges to be $\leq k$?

$$\sum_{(i,j)\in W} w_{ij} \leq k$$

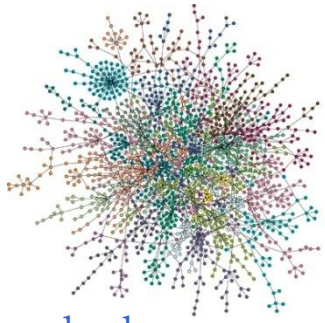
  - Having an algorithm that solves the Decision TSP in polynomial time, would it solve the Optimization TSP in polynomial time too?

  - Algorithm A_{opt} (Binary Search)

$$[\mathbf{0}, 1, 2, 3, \dots, m-1, m, m+1, \dots, nM-1, \boldsymbol{nM}]$$

  - $K = \lceil (right - left)/2 \rceil, \dots$

Paths and Cycles …



- **Decision TSP & Optimization TSP**
  - **Decision TSP**: Given a complete undirected graph G, with weighted edges $w_{ij} \in N, \forall (i,j) \in E(G)$ and a number $k \in N$, find if there exists a Hamiltonian cycle W in G, such that the sum of the weights on the edges to be $\leq k$?

$$\sum_{(i,j)\in W} w_{ij} \leq k$$

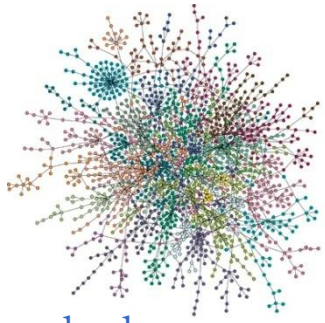
  - Having an algorithm that solves the Decision TSP in polynomial time, would it solve the Optimization TSP in polynomial time too?

  - Algorithm A_{opt} (Binary Search)

$$[\mathbf{0}, 1, 2, 3, \dots, m-1, m, m+1, \dots, nM-1, \boldsymbol{nM}]$$

  - $K = \lceil (right - left)/2 \rceil$, … Algorithm A_d in worst cast will be executed $O(lognM)$ times, and hence if A_d is polynomial, then A_{opt} will be polynomial too.

Paths and Cycles …

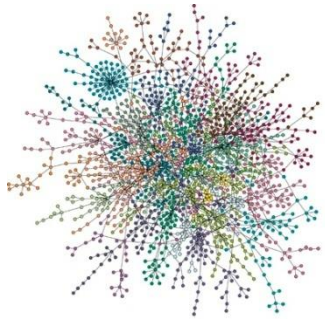


- **Decision TSP & Optimization TSP**
  - **Decision TSP**: Given a complete undirected graph G, with weighted edges $w_{ij} \in N, \forall (i,j) \in E(G)$ and a number $k \in N$, find if there exists a Hamiltonian cycle W in G, such that the sum of the weights on the edges to be $\leq k$?

$$\sum_{(i,j)\in W} w_{ij} \leq k$$

  - The performance of the Optimization TSP is close enough to the performance of the Decision TSP.

Paths and Cycles …



- **3SAT ∈ NP-complete (SAT ≤ 3SAT)**
  - **SAT Problem**: We are given a Boolean formula in conjunctive form:

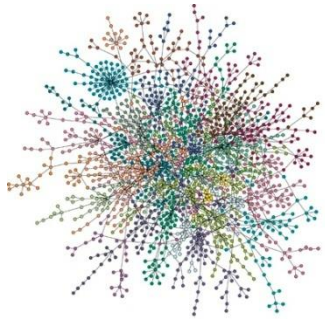
    $$(x \vee \bar{y} \vee z \vee \bar{w}) \wedge (x \vee \bar{y}) \wedge (z \vee \bar{x} \vee \bar{w} \vee p \vee y) \wedge (x \vee \bar{y} \vee z)$$

    and we need to find a satisfying truth assignment (or to claim that there does not exist any)

  - **3SAT Problem:**

    $$(x \vee \bar{y} \vee z) \wedge (w \vee \bar{y}) \wedge (y \vee \bar{z}) \wedge (\bar{p} \vee x) \wedge (\bar{x} \vee \bar{y} \vee z)$$

Paths and Cycles …



- **3SAT ∈ NP-complete (SAT ≤ 3SAT)**
  - **SAT Problem**: We are given a Boolean formula in conjunctive form:

    $(x \vee \bar{y} \vee z \vee \bar{w}) \wedge (x \vee \bar{y}) \wedge (z \vee \bar{x} \vee \bar{w} \vee p \vee y) \wedge (x \vee \bar{y} \vee z)$

    and we need to find a satisfying truth assignment (or to claim that there does not exist any)

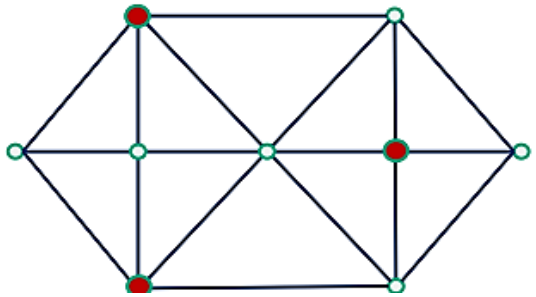
  - **3SAT Problem:**

    $(\bar{x} \vee y \vee \bar{z}) \wedge (x \vee \bar{y} \vee z) \wedge (x \vee y \vee z) \wedge (\bar{x} \vee \bar{y})$

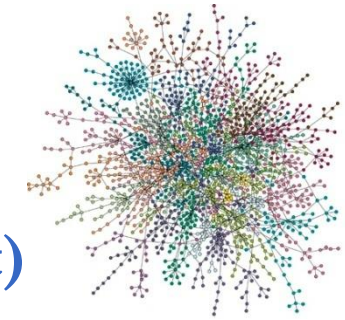
    3SAT ≤ IS

  - **Independent-Set**: We are given a graph of n vertices and an integer k.

    We are asked to find an I.S. of k vertices, where each pair of vertices in the set are disjoint, or to point out that there does not exist such a set.



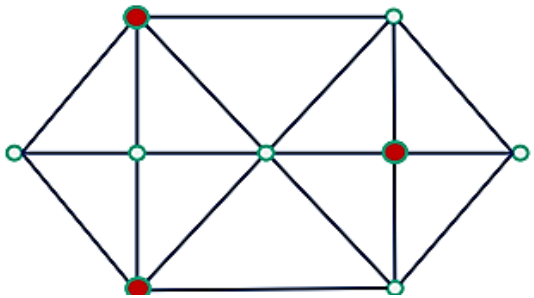
Paths and Cycles …



- **3SAT ∈ NP-complete (3SAT ≤ Independent Set)**
  - **3SAT Problem:**

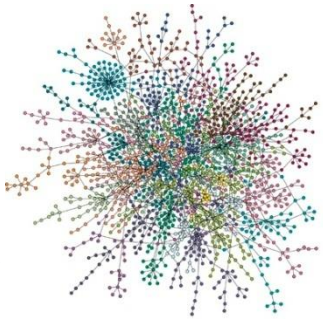
$$(\bar{x} \vee y \vee \bar{z}) \wedge (x \vee \bar{y} \vee z) \wedge (x \vee y \vee z) \wedge (\bar{x} \vee \bar{y})$$

  - **Independent-Set**:



  - Correlation between Boolean Logic and Graphs
  - In a satisfying truth assignment, an element from each term should be true, and hence since x could be chosen to be true in a term we are not allowed to set $\bar{x}$ to be true in another term.

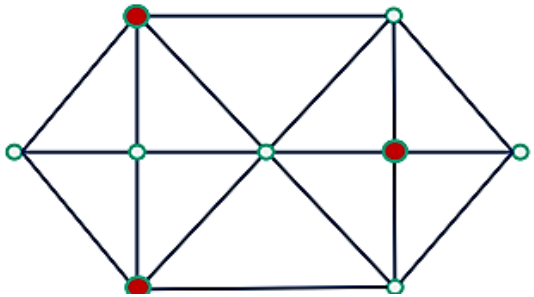
Paths and Cycles …



- **3SAT ∈ NP-complete (3SAT ≤ Independent Set)**
  - **3SAT Problem:**

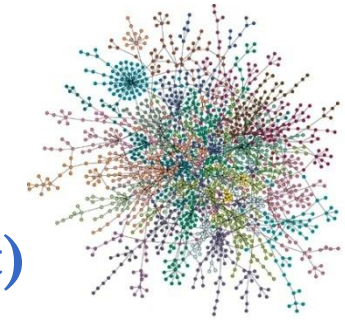
$$(\bar{x} \vee y \vee \bar{z}) \wedge (x \vee \bar{y} \vee z) \wedge (x \vee y \vee z) \wedge (\bar{x} \vee \bar{y})$$

  - **Independent-Set:**



  - Correlation between Boolean Logic and Graphs

Paths and Cycles …

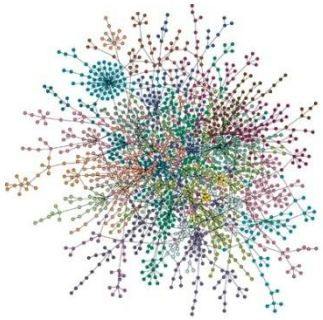


- **3SAT ∈ NP-complete (3SAT ≤ Independent Set)**
  - **3SAT Problem and Independent Set:**

$$(\bar{x} \vee y \vee \bar{z}) \wedge (x \vee \bar{y} \vee z) \wedge (x \vee y \vee z) \wedge (\bar{x} \vee \bar{y})$$

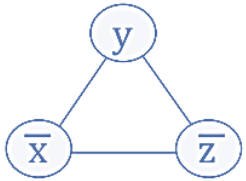
  - In the produced graph, let G, each Independent Set contains at most one element from each triangle/term.

Paths and Cycles …



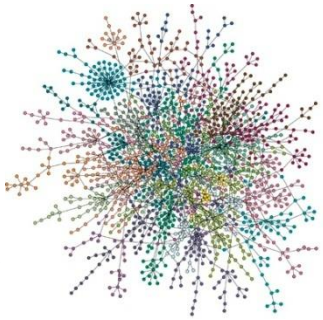
- **3SAT ∈ NP-complete (3SAT ≤ Independent Set)**
  - **3SAT Problem and Independent Set:**

$$(\bar{x} \vee y \vee \bar{z}) \wedge (x \vee \bar{y} \vee z) \wedge (x \vee y \vee z) \wedge (\bar{x} \vee \bar{y})$$



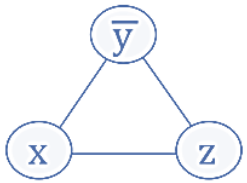
  - In the produced graph, let G, each Independent Set contains at most one element from each triangle/term.

Paths and Cycles …



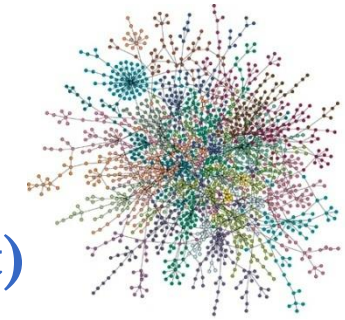
- **3SAT ∈ NP-complete (3SAT ≤ Independent Set)**
  - **3SAT Problem and Independent Set:**

$$(\bar{x} \vee y \vee \bar{z}) \wedge (x \vee \bar{y} \vee z) \wedge (x \vee y \vee z) \wedge (\bar{x} \vee \bar{y})$$



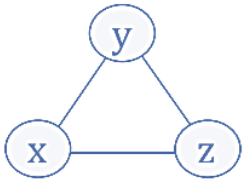
  - In the produced graph, let G, each Independent Set contains at most one element from each triangle/term.

Paths and Cycles …



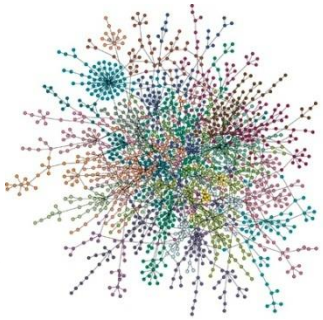
- **3SAT ∈ NP-complete (3SAT ≤ Independent Set)**
  - **3SAT Problem and Independent Set:**

$$(\bar{x} \vee y \vee \bar{z}) \wedge (x \vee \bar{y} \vee z) \wedge (x \vee y \vee z) \wedge (\bar{x} \vee \bar{y})$$



  - In the produced graph, let G, each Independent Set contains at most one element from each triangle/term.

Paths and Cycles …

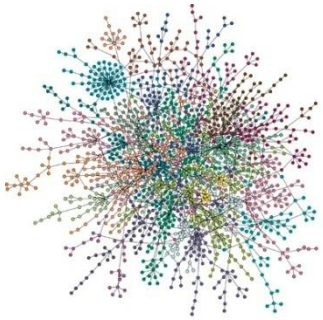


- **3SAT ∈ NP-complete (3SAT ≤ Independent Set)**
  - **3SAT Problem and Independent Set:**

$$(\bar{x} \vee y \vee \bar{z}) \wedge (x \vee \bar{y} \vee z) \wedge (x \vee y \vee z) \wedge (\bar{x} \vee \bar{y})$$

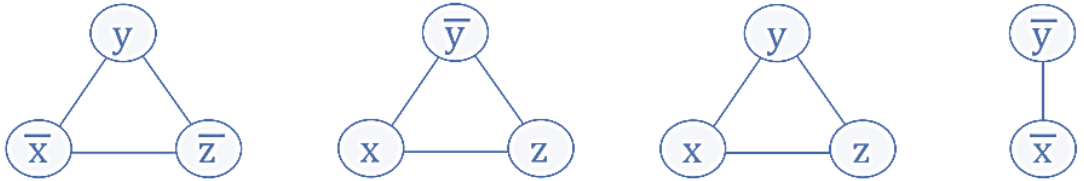
  - In the produced graph, let G, each Independent Set contains at most one element from each triangle/term.

Paths and Cycles …



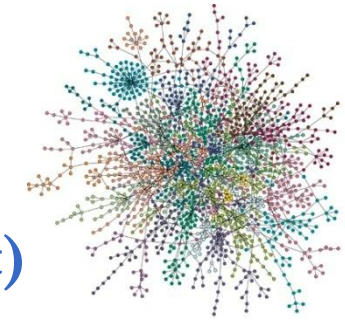
- **3SAT ∈ NP-complete (3SAT ≤ Independent Set)**
  - **3SAT Problem and Independent Set:**

$$(\bar{x} \vee y \vee \bar{z}) \wedge (x \vee \bar{y} \vee z) \wedge (x \vee y \vee z) \wedge (\bar{x} \vee \bar{y})$$



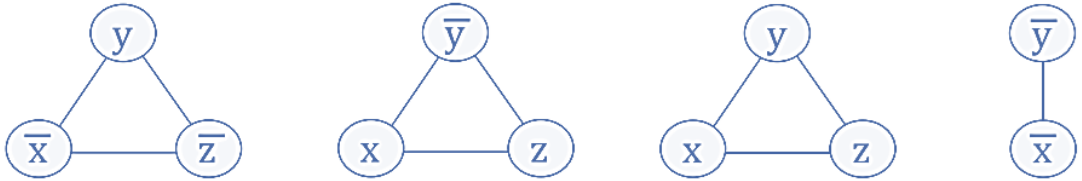
  - In the produced graph, let G, each Independent Set contains at most one element from each triangle/term.

Paths and Cycles …



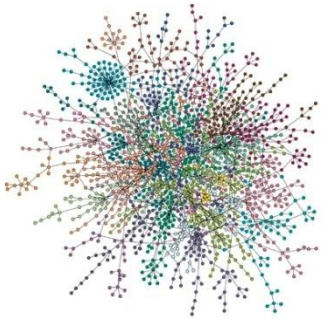
- **3SAT ∈ NP-complete (3SAT ≤ Independent Set)**
  - **3SAT Problem and Independent Set:**

$$(\bar{x} \vee y \vee \bar{z}) \wedge (x \vee \bar{y} \vee z) \wedge (x \vee y \vee z) \wedge (\bar{x} \vee \bar{y})$$



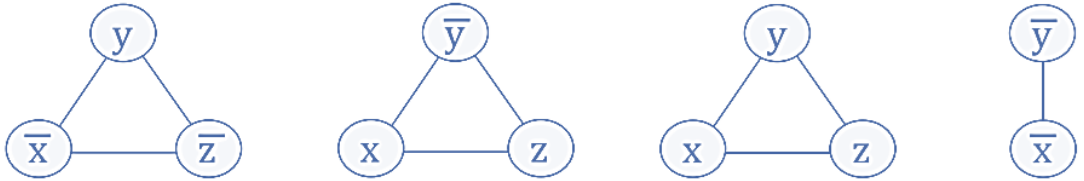
- In the produced graph, let G, each Independent Set contains at most one element from each triangle/term.
- In order to set an option from each triangle/term, we set k =the cardinality of the terms.
- It is needed a method that would deter us from selecting opposite elements, i.e., x and $\bar{x}$.

Paths and Cycles …

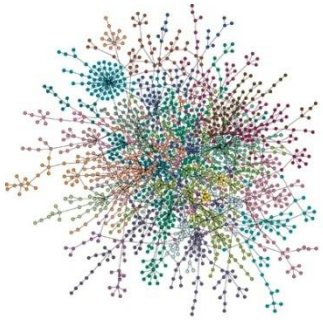


- **3SAT ∈ NP-complete (3SAT ≤ Independent Set)**
  - **3SAT Problem and Independent Set:**

$$(\bar{x} \vee y \vee \bar{z}) \wedge (x \vee \bar{y} \vee z) \wedge (x \vee y \vee z) \wedge (\bar{x} \vee \bar{y})$$

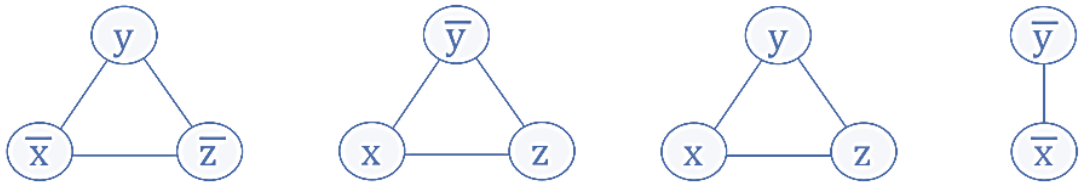


Paths and Cycles …

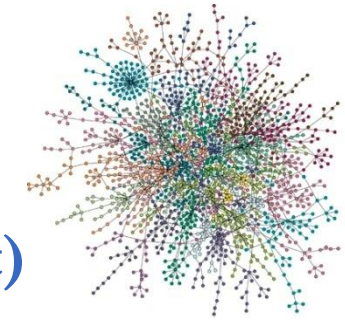


- **3SAT ∈ NP-complete (3SAT ≤ Independent Set)**
  - **3SAT Problem and Independent Set:**

$$(\bar{x} \vee y \vee \bar{z}) \wedge (x \vee \bar{y} \vee z) \wedge (x \vee y \vee z) \wedge (\bar{x} \vee \bar{y})$$

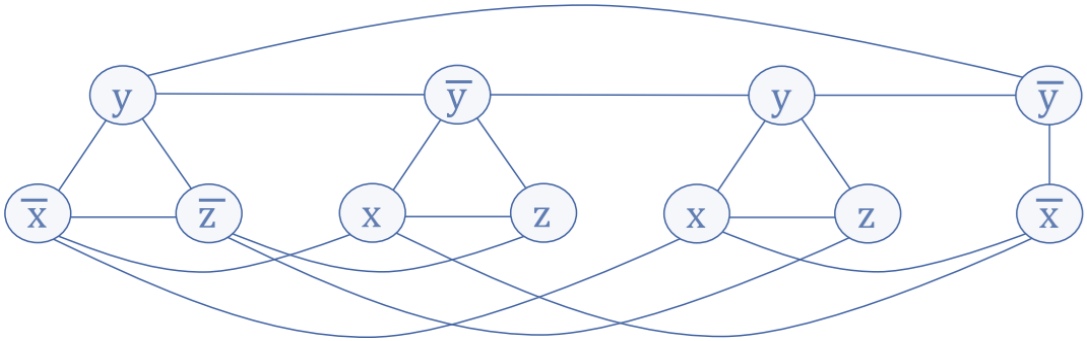


Paths and Cycles …



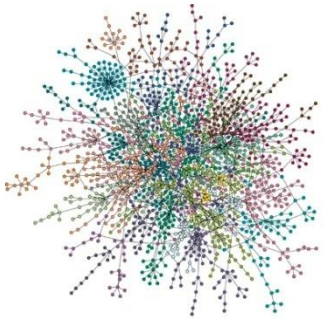
- **3SAT ∈ NP-complete (3SAT ≤ Independent Set)**
  - **3SAT Problem and Independent Set:**

$$(\bar{x} \vee y \vee \bar{z}) \wedge (x \vee \bar{y} \vee z) \wedge (x \vee y \vee z) \wedge (\bar{x} \vee \bar{y})$$



Given an instance I of problem 3SAT, construct an instance (G, k) of the problem Independent Set.

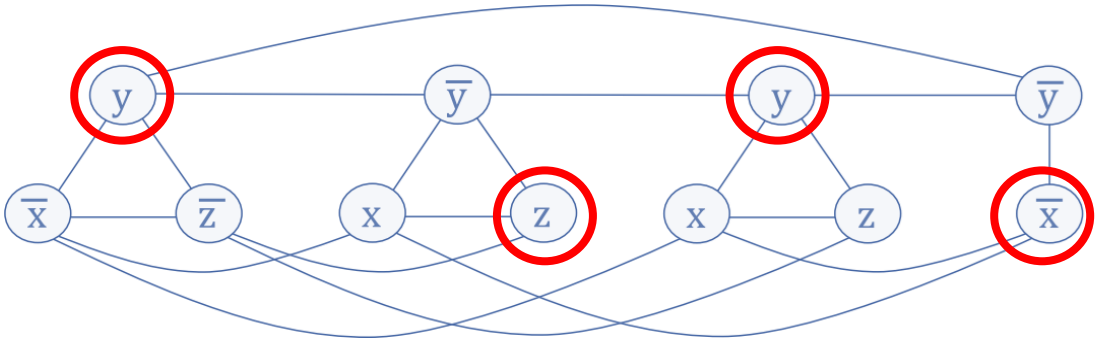
Paths and Cycles …



- **3SAT ∈ NP-complete (3SAT ≤ Independent Set)**
  - **3SAT Problem and Independent Set:**

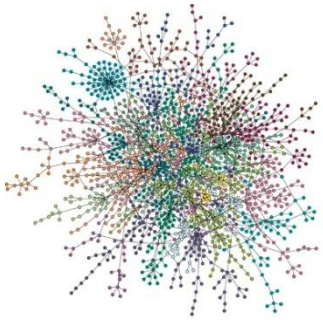
$$(\bar{x} \vee y \vee \bar{z}) \wedge (x \vee \bar{y} \vee z) \wedge (x \vee y \vee z) \wedge (\bar{x} \vee \bar{y})$$

$$x = false, y = true, z = true$$



Given an instance I of problem 3SAT, construct an instance (G, k) of the problem Independent Set.

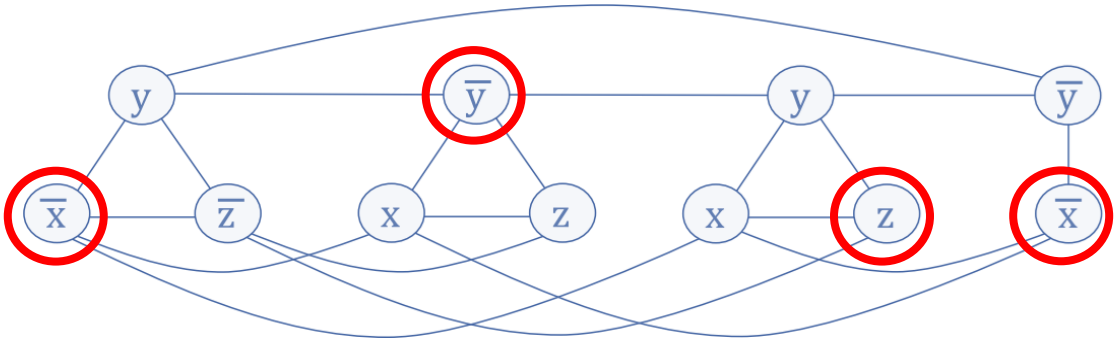
Paths and Cycles …



- **3SAT ∈ NP-complete (3SAT ≤ Independent Set)**
  - **3SAT Problem and Independent Set:**

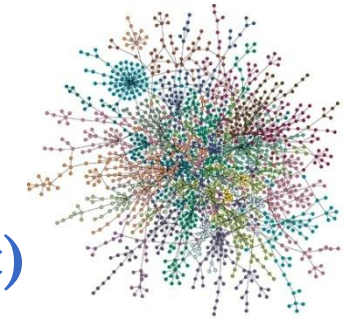
$$(\bar{x} \vee y \vee \bar{z}) \wedge (x \vee \bar{y} \vee z) \wedge (x \vee y \vee z) \wedge (\bar{x} \vee \bar{y})$$

$$x = false, y = false, z = true$$



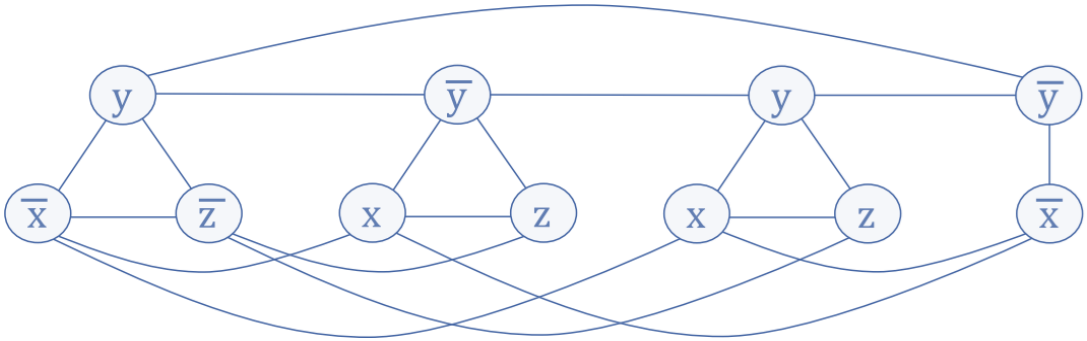
Given an instance I of problem 3SAT, construct an instance (G, k) of the problem Independent Set.

Paths and Cycles …



- **3SAT ∈ NP-complete (3SAT ≤ Independent Set)**
  - **3SAT Problem and Independent Set:**

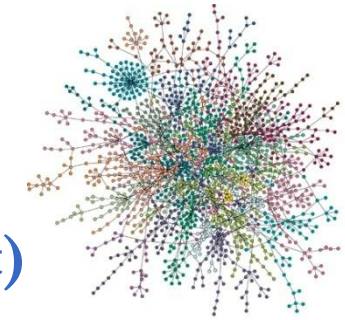
$$(\bar{x} \vee y \vee \bar{z}) \wedge (x \vee \bar{y} \vee z) \wedge (x \vee y \vee z) \wedge (\bar{x} \vee \bar{y})$$



Given an instance I of problem 3SAT, construct an instance (G, k) of the problem Independent Set.

Polynomial Time

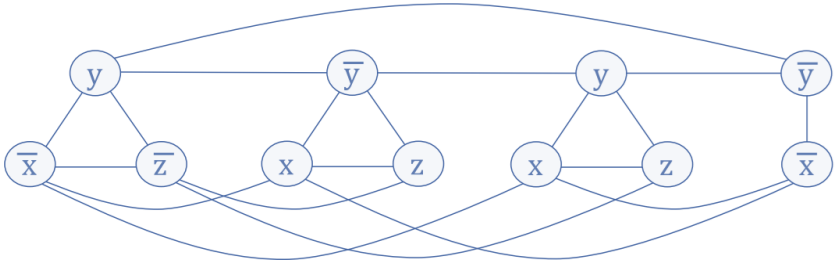
Paths and Cycles …



- **3SAT ∈ NP-complete (3SAT ≤ Independent Set)**
  - **3SAT Problem and Independent Set:**

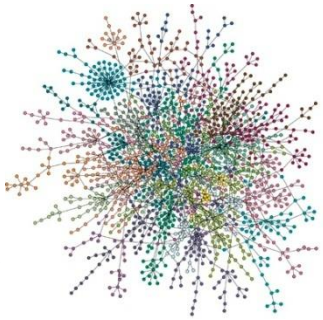
    We should prove that 3SAT ⇔ IS

    $$(\bar{x} \vee y \vee \bar{z}) \wedge (x \vee \bar{y} \vee z) \wedge (x \vee y \vee z) \wedge (\bar{x} \vee \bar{y})$$



  - (⟸) Given an Independent Set S with k vertices in the graph G, we can always get a satisfying truth assignment in I.
  - (⟹) If the graph G has no Independent Set S of size k, then the logical formula I is not satisfiable.

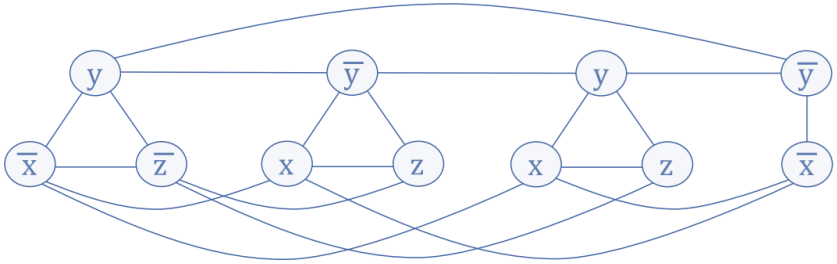
Paths and Cycles …



- **3SAT ∈ NP-complete (3SAT ≤ Independent Set)**
  - **3SAT Problem and Independent Set:**

    We should prove that 3SAT ⇔ IS

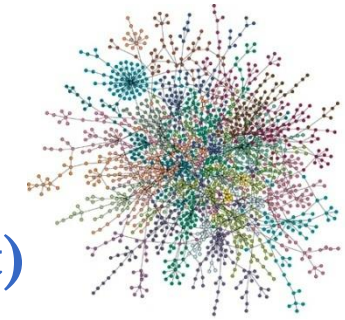
    $$(\bar{x} \vee y \vee \bar{z}) \wedge (x \vee \bar{y} \vee z) \wedge (x \vee y \vee z) \wedge (\bar{x} \vee \bar{y})$$



  - (⇐) Given an Independent Set S with k vertices in the graph G, we can always get a satisfying truth assignment in I.

    For each variable x, set S contains no vertices with labels x and $\bar{x}$. We assign $x = true$ if s contains vertex with label x and $x = false$ if S contains vertex with label $\bar{x}$. Since $|S| = k$, it should contain a vertex per term. Such an assignment satisfies all terms.

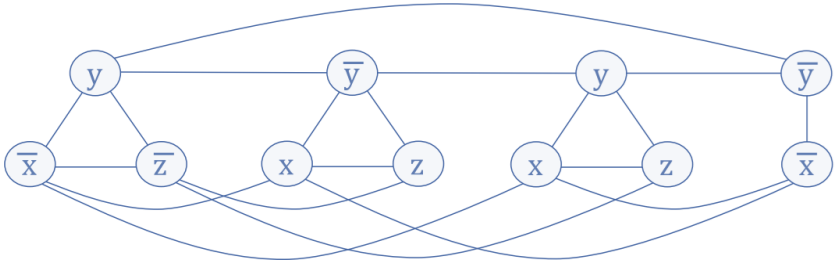
Paths and Cycles …



- **3SAT ∈ NP-complete (3SAT ≤ Independent Set)**
  - **3SAT Problem and Independent Set:**

    We should prove that 3SAT ⇔ IS

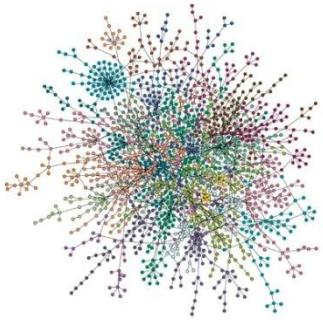
    $$(\bar{x} \vee y \vee \bar{z}) \wedge (x \vee \bar{y} \vee z) \wedge (x \vee y \vee z) \wedge (\bar{x} \vee \bar{y})$$



  - (⟹) If the graph G has no Independent Set S of size k, then the logical formula I is not satisfiable.

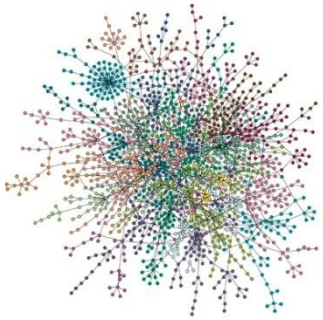
    It is adequate to prove that if the logical formula I has a satisfying truth assignment, then G has an Independent Set S of size k. Hence, for each term of the 3SAT we select an element which its value in the satisfying truth assignment is true and add the corresponding vertex in S .

Paths and Cycles …



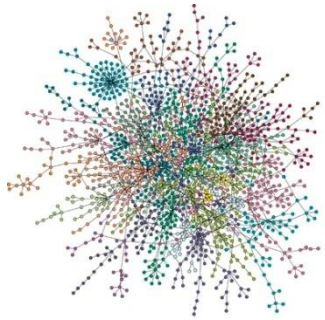
- **Approximation Algorithms on TSP**
  - **Heuristic Sub-Optimal Solutions:**
    1. **Greedy TSP with DFS**
    2. **Greedy TSP with Bilateral Path Extension (Nearest Insertion)**
    3. **Greedy TSP with Closest Insertion (Cheapest Insertion)**
    4. **TSP with Minimum Spanning Tree**
    5. **TSP with Vertex Substitution**

Paths and Cycles …



- **Approximation Algorithms on TSP**
  - **Heuristic Sub-Optimal Solutions:**
    1. **Greedy TSP with DFS**
    2. Greedy TSP with Bilateral Path Extension (Nearest Insertion)
    3. Greedy TSP with Closest Insertion (Cheapest Insertion)
    4. TSP with Minimum Spanning Tree
    5. TSP with Vertex Substitution

Paths and Cycles …

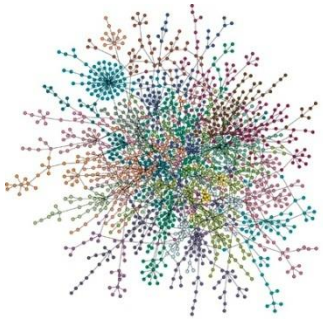


- **Approximation Algorithms on TSP**
  - **Heuristic Sub-Optimal Solutions:**

    Greedy TSP with DFS

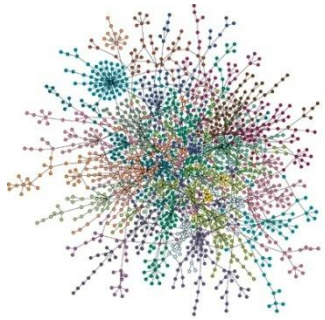
    1) Select a random vertex v_0
    2) set $i \leftarrow 1$ and $l(v) \leftarrow 0$
    3) while (∃ unmarked vertices)
    4) select the edge with the less weight, from the edges that connect vertex v with an unmarked vertex w.
    5) set $l(w) \leftarrow i$ and $v \leftarrow w$.

Paths and Cycles …



- **Approximation Algorithms on TSP**
  - **Heuristic Sub-Optimal Solutions:**
    1. Greedy TSP with DFS
    2. **Greedy TSP with Bilateral Path Extension (Nearest Insertion)**
    3. Greedy TSP with Closest Insertion (Cheapest Insertion)
    4. TSP with Minimum Spanning Tree
    5. TSP with Vertex Substitution

Paths and Cycles …

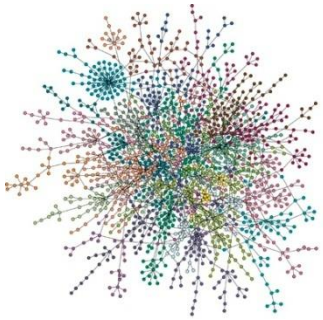


- **Approximation Algorithms on TSP**
  - **Heuristic Sub-Optimal Solutions:**

Greedy TSP with Bilateral Path Extension (Nearest Insertion)

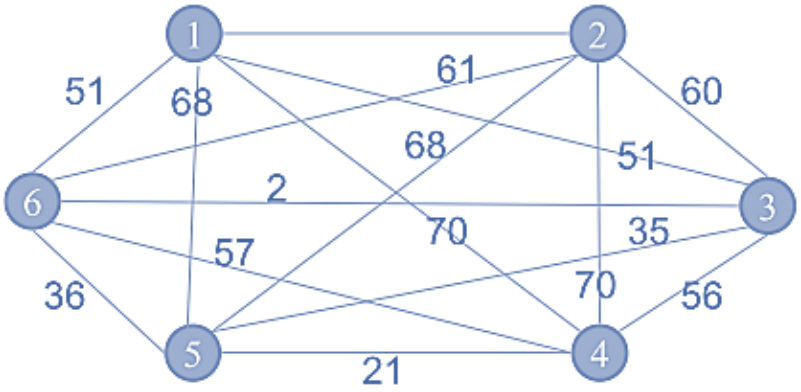
1) Set $i \leftarrow 1$.
2) Select a random peak v_0 and consider the path $P_i = v_0$
3) If $(i = n)$
   $C = P_n$ is a Hamilton circle,
4) Else
   select the lightest edge e adjacent to one of the two terminal vertices of P_i such that no cycle with the peaks of P_i to be created.
5) Construct the path $P_{i+1} = (P_i \cup e)$
6) Set $i \leftarrow i + 1$.
7) Goto step 3

Paths and Cycles …



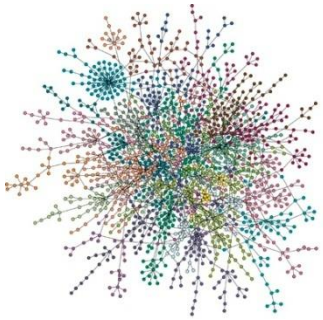
- **Approximation Algorithms on TSP**
  - **Heuristic Sub-Optimal Solutions:**

Greedy TSP with Bilateral Path Extension (Nearest Insertion)



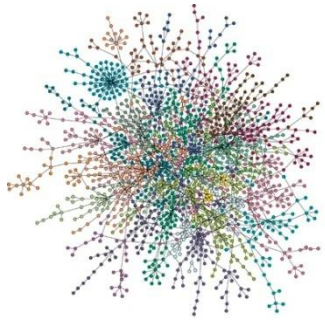
Path	Weight
3-6	2
3-6-5	38
3-6-5-4	59
3-6-5-4-2	129
3-6-5-4-2-1	142
3-6-5-4-2-1-3	193

Paths and Cycles …



- **Approximation Algorithms on TSP**
  - **Heuristic Sub-Optimal Solutions:**
    1. Greedy TSP with DFS
    2. Greedy TSP with Bilateral Path Extension (Nearest Insertion)
    3. **Greedy TSP with Closest Insertion (Cheapest Insertion)**
    4. TSP with Minimum Spanning Tree
    5. TSP with Vertex Substitution

Paths and Cycles …

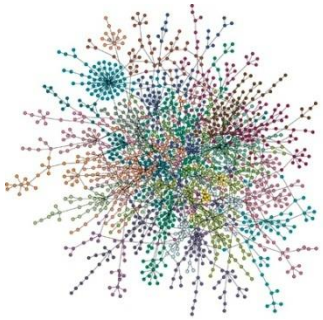


- **Approximation Algorithms on TSP**
  - **Heuristic Sub-Optimal Solutions:**

Greedy TSP with Closest Insertion (Cheapest Insertion)

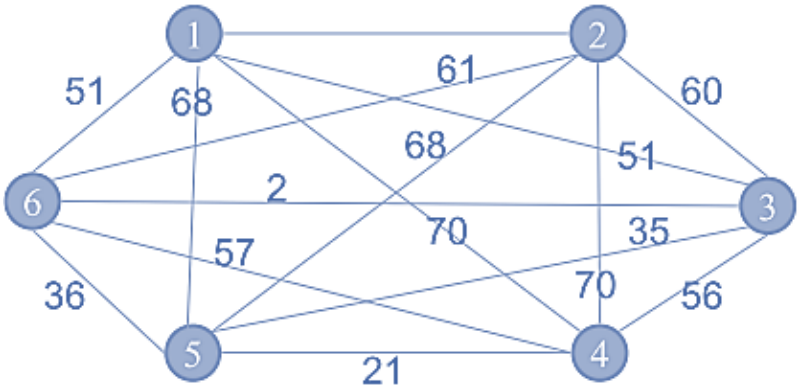
1) Set $i \leftarrow 1$.
2) Select a random peak v_0 and consider the path $C_i = v_0$
3) If $(i = n)$
   $C = C_n$ is a Hamilton circle
4) Else
   search for a vertex v_i that $\notin$ in the circle C_i and are closest to a pair in sequential vertices $\{v_i, v_{i+1}\} \in C_i$.
5) Construct cycle C_{i+1} inserting the vertex v_i between v_i and v_{i+1}.
6) Set $i \leftarrow i + 1$.
7) Goto step 3

Paths and Cycles …



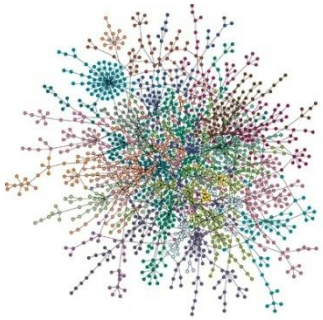
- **Approximation Algorithms on TSP**
  - **Heuristic Sub-Optimal Solutions:**

Greedy TSP with Closest Insertion (Cheapest Insertion)



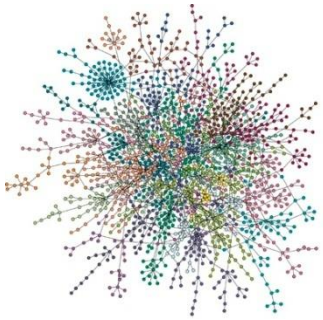
Path	Weight
3	…
3-6-3	…
3-6-5-3	…
3-6-5-4-3	…
3-6-1-5-4-3	…
3-6-2-1-5-4-3	192

Paths and Cycles …



- **Approximation Algorithms on TSP**
  - **Heuristic Sub-Optimal Solutions:**
    1. Greedy TSP with DFS
    2. Greedy TSP with Bilateral Path Extension (Nearest Insertion)
    3. Greedy TSP with Closest Insertion (Cheapest Insertion)
    4. **TSP with Minimum Spanning Tree**
    5. TSP with Vertex Substitution

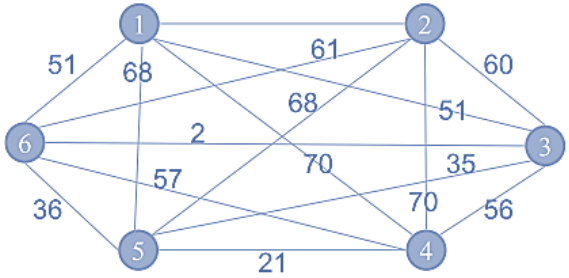
Paths and Cycles …



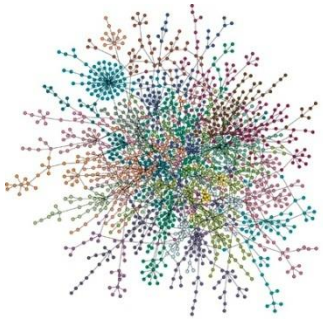
- **Approximation Algorithms on TSP**
  - **Heuristic Sub-Optimal Solutions:**

TSP with Minimum Spanning Tree

1) Find a minimum spanning tree T
2) Perform a Depth First Search
3) If $v_i, v_2, \ldots, v_p$ is the sequence of visiting the vertices of T from step 2, then the Hamiltonian cycle is $v_i, v_2, \ldots, v_p, v_1$.



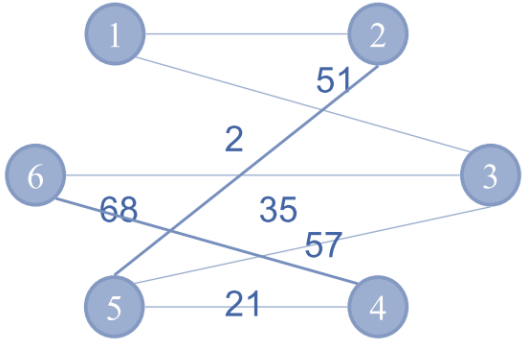
Paths and Cycles …



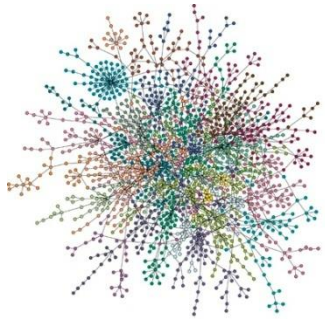
- **Approximation Algorithms on TSP**
  - **Heuristic Sub-Optimal Solutions:**

TSP with Minimum Spanning Tree

1) Find a minimum spanning tree T
2) Perform a Depth First Search
3) If $v_i, v_2, \ldots, v_p$ is the sequence of visiting the vertices of T from step 2, then the Hamiltonian cycle is $v_i, v_2, \ldots, v_p, v_1$.



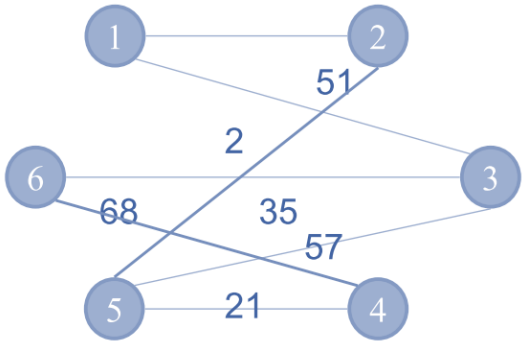
Paths and Cycles …



- **Approximation Algorithms on TSP**
  - **Heuristic Sub-Optimal Solutions:**

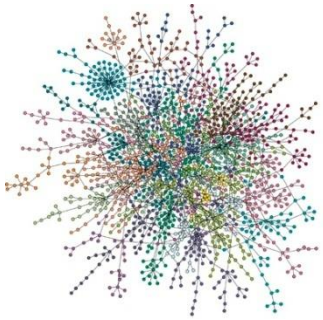
TSP with Minimum Spanning Tree

1) Find a minimum spanning tree T
2) Perform a Depth First Search
3) If $v_i, v_2, \ldots, v_p$ is the sequence of visiting the vertices of T from step 2, then the Hamiltonian cycle is $v_i, v_2, \ldots, v_p, v_1$.



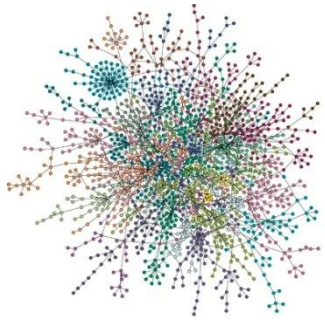
Path	Weight
(3,1,2,5,4,6,3)	212
(1,2,3,5,4,6,1)	237
…	…

Paths and Cycles …



- **Approximation Algorithms on TSP**
  - **Heuristic Sub-Optimal Solutions:**
    1. Greedy TSP with DFS
    2. Greedy TSP with Bilateral Path Extension (Nearest Insertion)
    3. Greedy TSP with Closest Insertion (Cheapest Insertion)
    4. TSP with Minimum Spanning Tree
    5. **TSP with Vertex Substitution**

Paths and Cycles …

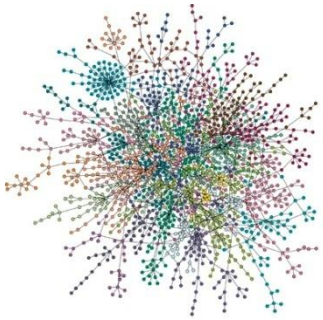


- **Approximation Algorithms on TSP**
  - **Heuristic Sub-Optimal Solutions:**

TSP with Vertex Substitution

1) Consider a Hamiltonian cycle $C = (v_1, v_2, \ldots, v_n, v_1)$
2) $\forall\ i, j, : 1 < i + 1 < j < n$, obtain a new Hamiltonian cycle $C_{i,j} = (v_1, v_2, \ldots\ v_i, v_j, v_{j-1}, \ldots, v_{i+1}, v_{j+1}, v_{j+2}, \ldots, v_n, v_1)$ deleting the edges (v_i, v_{i+1}) *and* (v_j, v_{j+1}) and adding the edges (v_i, v_j) *and* (v_{i+1}, v_{j+1}).
3) If (for some i, j occurs $w(v_i, v_j) + w(v_{i+1}, v_{j+1}) < w(v_i, v_{i+1}) + w(v_j, v_{j+1})$) set $C = C_{i,j}$.
4) Goto step 2.

Paths and Cycles …

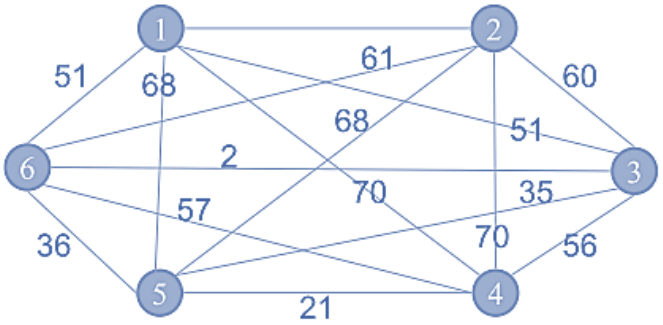


Approximation Algorithms on TSP

- **Heuristic Sub-Optimal Solutions:**

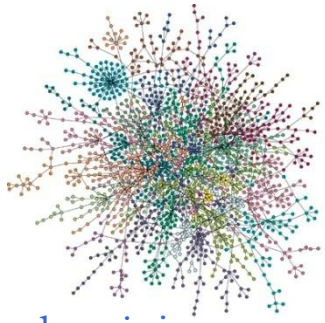
TSP with Vertex Substitution

1) Consider a Hamiltonian cycle $C = (v_1, v_2, \ldots, v_n, v_1)$
2) $\forall\ i, j, : 1 < i + 1 < j < n$, obtain a new Hamiltonian cycle $C_{i,j} = (v_1, v_2, \ldots\ v_i, v_j, v_{j-1}, \ldots, v_{i+1}, v_{j+1}, v_{j+2}, \ldots, v_n, v_1)$ deleting the edges (v_i, v_{i+1}) *and* (v_j, v_{j+1}) and adding the edges (v_i, v_j) *and* (v_{i+1}, v_{j+1}).
3) If (for some i, j occurs $w(v_i, v_j) + w(v_{i+1}, v_{j+1}) < w(v_i, v_{i+1}) + w(v_j, v_{j+1})$) set $C = C_{i,j}$.
4) Goto step 2.

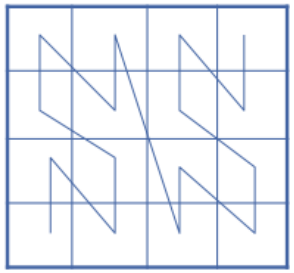


Path	Weight
(3,4,5,6,1,2,3)	237
(3,6,5,4,1,2,3)	210
(3,6,5,4,2,1,3)	193
(3,6,1,2,4,5,3)	192

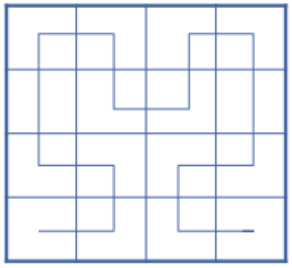
Paths and Cycles …



- **Infinite Graphs**
  - Vertices are points of the plane with integer coordinates, while the edges join vertices at a distance of 1
  - In infinite graphs there is no Eulerian circuit or Hamiltonian circle, but there are corresponding paths
  - An "one-way" Eulerian / Hamiltonian path is the path that starts from a vertex and expands infinitely (space filling curve)

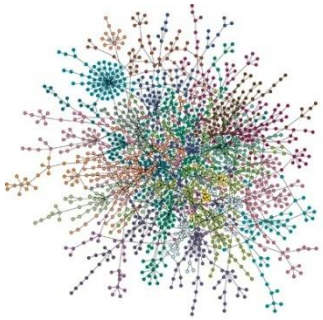


Peano/z-order



Hilbert

Paths and Cycles …



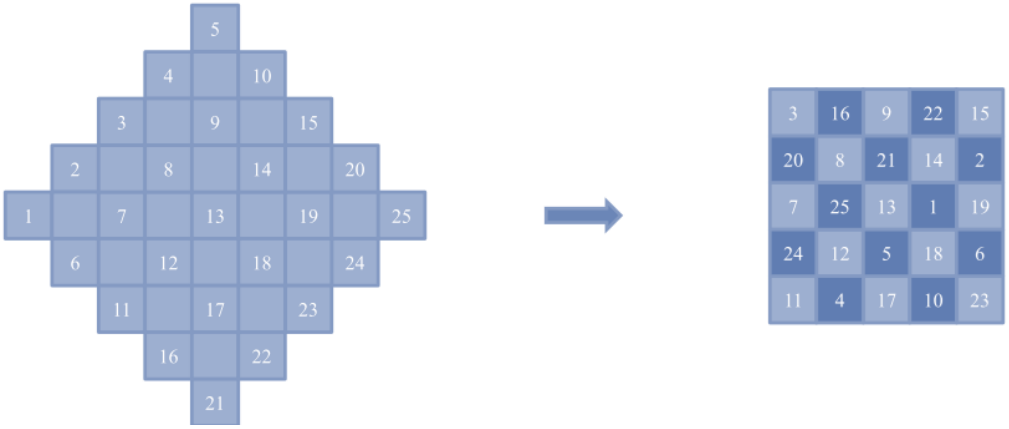
Magic Squares

- The rows, columns and diagonals have an equal sum

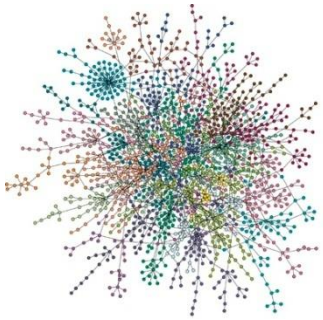
16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1

23	1	2	20	19
22	16	9	14	4
5	11	13	15	21
8	12	17	10	18
7	25	24	6	3

- Magic squares algorithms (odd order):
  - Bachet method (with rhombus)



Paths and Cycles …



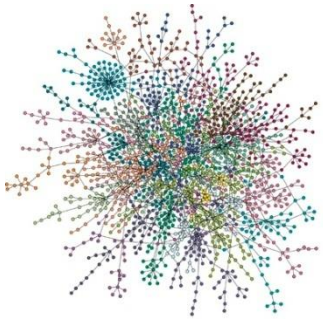
- **Magic Squares**
  - The rows, columns and diagonals have an equal sum

16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1

23	1	2	20	19
22	16	9	14	4
5	11	13	15	21
8	12	17	10	18
7	25	24	6	3

  - Magic squares algorithms (odd order):
    - Bachet method (with rhombus)
    - By tricking the three random numbers (eg 3,2,5)
    - Replacing unnecessary numbers 3-17 in positions 1-9
    - Adding the same number to each position

Paths and Cycles …



- **Magic Squares**
  - The rows, columns and diagonals have an equal sum

16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1

23	1	2	20	19
22	16	9	14	4
5	11	13	15	21
8	12	17	10	18
7	25	24	6	3

  - **Magic** is called the graph where the sum of their inscriptions of edges that fall on all nodes is equal
  - **Theorem**: if a bipartite graph can be decomposed into 2 Hamiltonian circles, then the graph is magical
  - **Antimagic** is called the graph where the sums of inscriptions of the edges of all vertices is not equal.
  - Number of magic objects (squares, dominoes, polygons, etc)