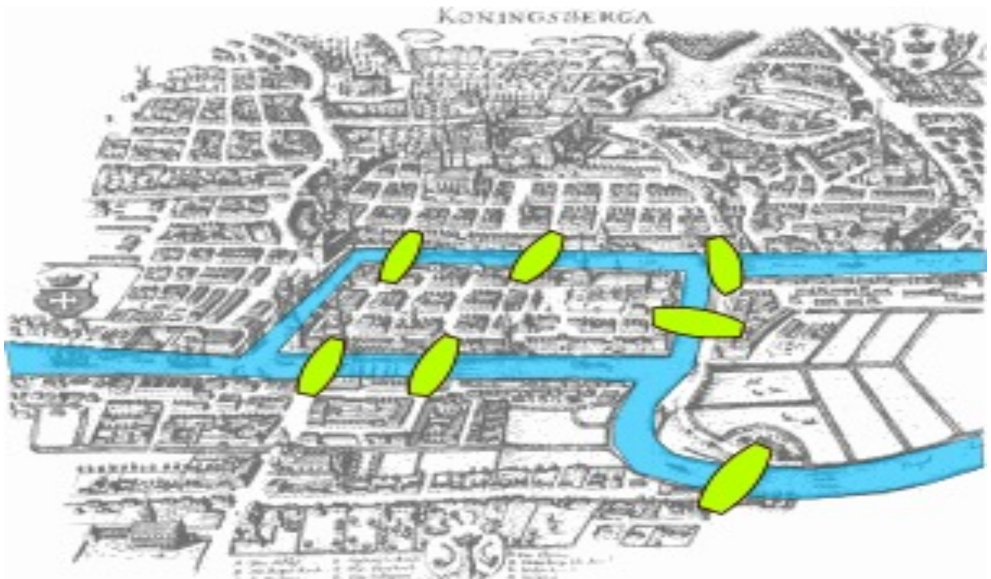
LECTURE 8:GRAPH THEORY

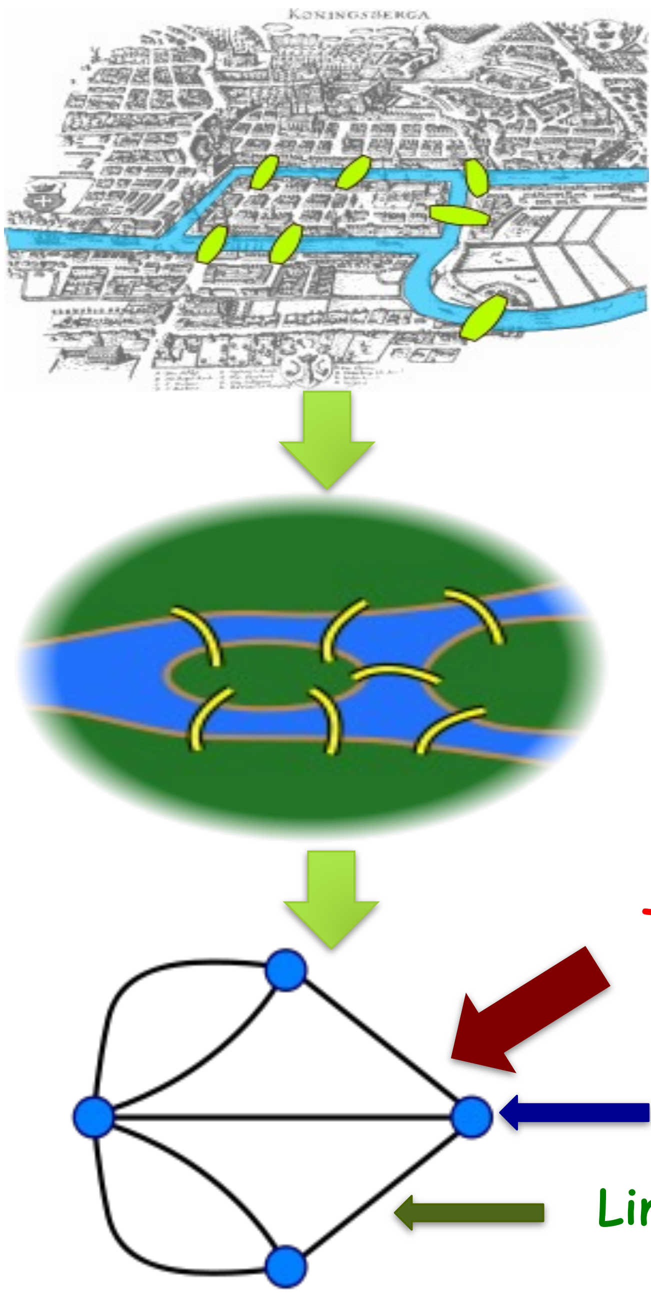
SEVEN BRIDGES OF KONIGSBURG



The city of Königsberg was set on both sides of the Pregel River, and included two large islands which were connected to each other and the mainland by seven bridges.

Is there a walk through the city that would cross each bridge once and only once.

The islands could not be reached by any route other than the bridges, and every bridge must have been crossed completely every time



Can you draw the final figure without retracing the lines and not lifting the pencil from the paper ?

This is a GRAPH

Points: Vertices or Nodes

Lines: Edges or Links

SEVEN BRIDGES OF KONIGSBURG

The Konigsburg bridge problem is one of the first problem to be solved using graph theory

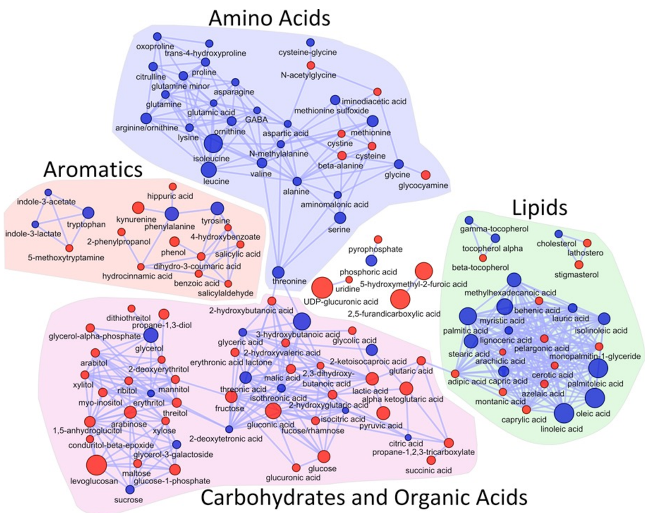
Solved by Euler in 1735

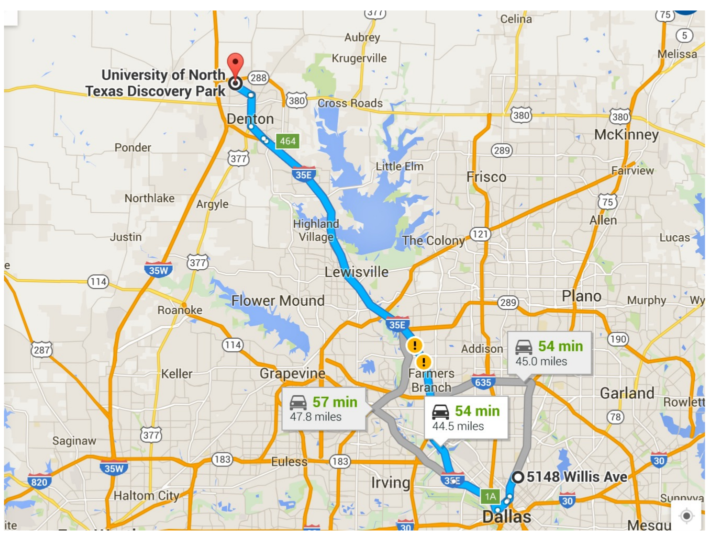
Mathematically showed that such a walk cannot exist

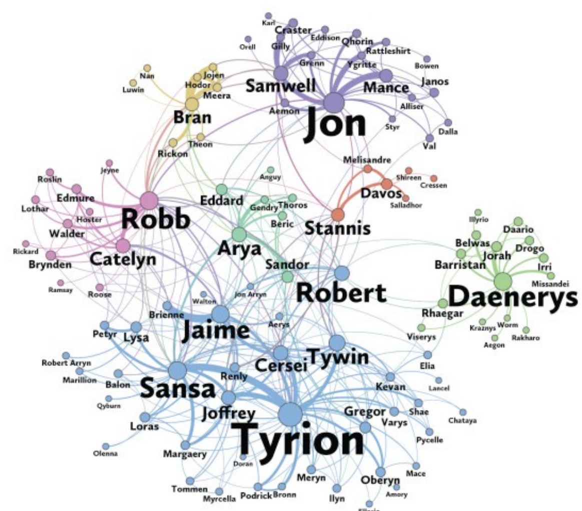
For a solution to exist, no more than two vertices can have odd number of edges coming out of them. Why ?

Can you add or subtract a bridge to make the problem solvable ?

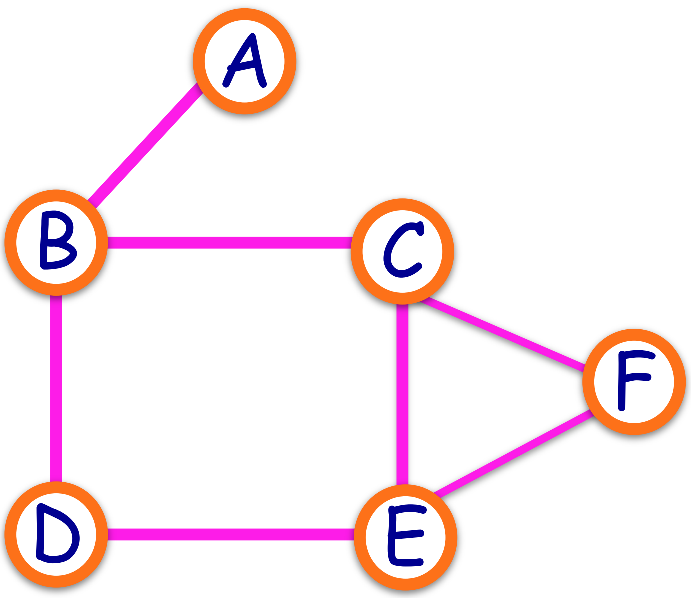
EXAMPLES


Amino Acids
Aromatics
Lipids
Carbohydrates and Organic Acids


University of North Texas Discovery Park
Denton
Celina
Aubrey
Krugerville
Cross Roads
McKinney
Ponder
Little Elm
Frisco
Fairview
Northlake
Argyle
Highland Village
The Colony
Allen
Justin
Lucas
Lewisville
Plano
Roanoke
Flower Mound
Murphy
Keller
Grapevine
Addison
Farmers Branch
Garland
Rowlett
Saginaw
Euless
Irving
Haltom City
Dallas
57 min
47.8 miles
54 min
45.0 miles
54 min
44.5 miles
5148 Willis Ave


Jon
Samwell
Mance
Bran
Robb
Eddard
Davos
Stannis
Arya
Catelyn
Sandor
Robert
Daenerys
Jaime
Cersei
Tywin
Sansa
Joffrey
Gregor
Tyrion

PROPERTIES OF GRAPHS



Vertex Set={A,B,C,D,E}

Edge Set={ (A,B), (B,A), (B,C), (B,D),(C,B), (C,E), (C,F),(D,B),(D,E), (E,C),(E,D),(E,F), (F,C),(F,E)}

An undirected graph is a graph where if there is an edges (A,B) there will also be an edge (B,A)

Vertex v is a neighbor of vertex w if (v,w) is an edge in the list

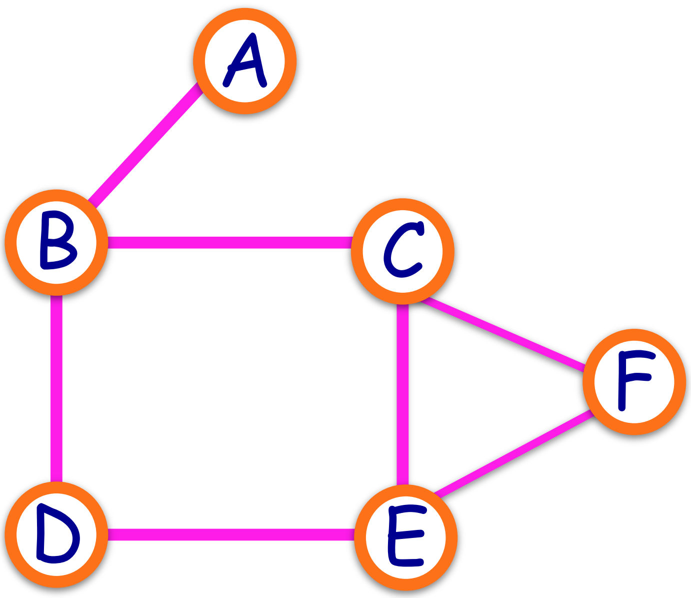
- C is a neighbor of B, but not of D

Degree of a vertex is its number of neighbors--Degree of C is 3

Degree distribution is the degree of the vertices in increasing order

- 1(A), 2(D), 2(F),3(B),3(C), 3(D)

PROPERTIES OF GRAPHS



A path is a sequence of vertices (v1,v2, ..,vn), such that (v1, v2) is an edge

- (A,B,C,E) is a path
- (A,B,C,D) is NOT a path—just a sequence of vertices

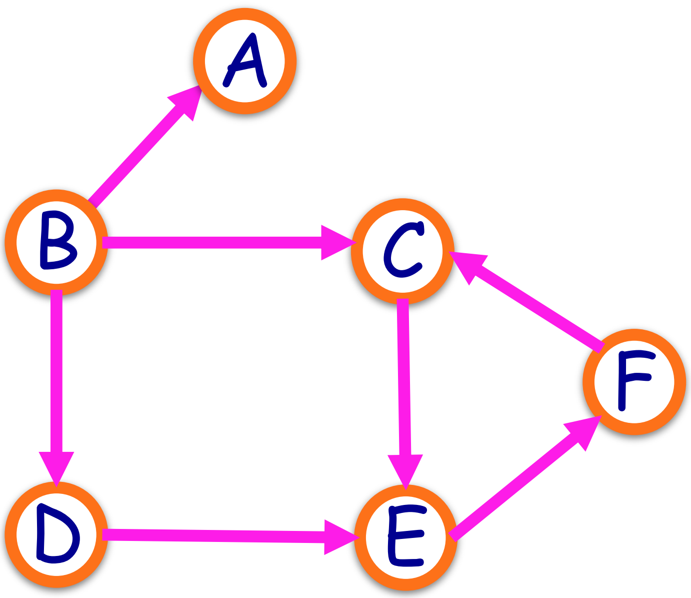
The length of a path is the number of edges in the path

- Length of path (A,B, C,E) is 3

A cycle is a path where the beginning and the end vertices are the same; v1=vn

- (B,C,E,D,B) is a cycle of length 4
- (C,E,F,C) is a cycle of length 3

PROPERTIES OF GRAPHS



Vertex Set={A,B,C,D,E}

Edge Set={ (B,A), (B,C), (B,D), (C,E), (D,E), (D,F),(F,C)}

A directed graph is a graph where if there is an edges (A,B) there may NOT be an edge (B,A)

In-Degree of a vertex is the number of edges pointing to it. Vertex with only in-degrees is a sink

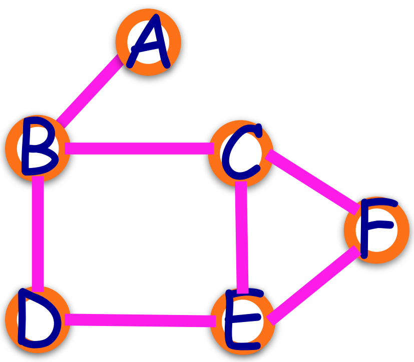
- In-degree of C is 2
- A is a sink

Out-Degree of a vertex is the number of edges going out from it

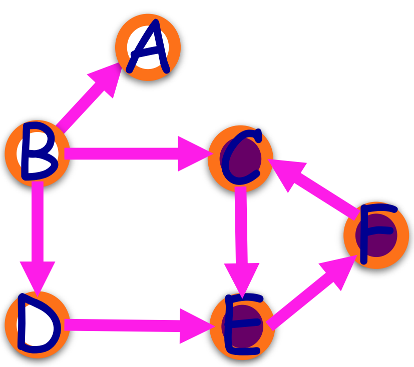
Vertex with only out-degrees is a source

- Out-degree of C is 2
- B is a source

CONNECTIVITY



(a) Undirected Graph



(b) Directed Graph

A graph is connected if there is a path from every vertex to every other vertex

A directed graph is strongly connected if there is a path from every vertex to every other vertex while maintaining the direction of the edges

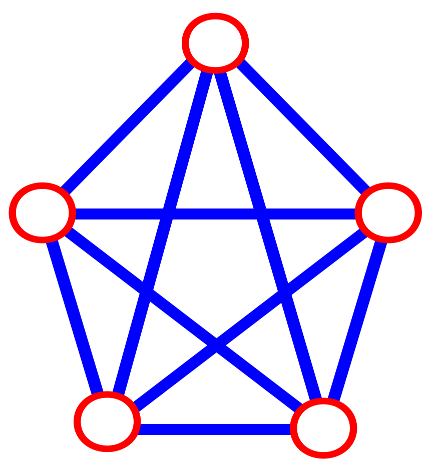
A directed graph is weakly connected if there is a path from every vertex to every other vertex while ignoring the direction of the edges

- The directed graph (b) is weakly connected
- The subgraph with purple nodes is strongly connected

DEFINITIONS (COMPLETE GRAPH)

A *complete graph* is a graph in which there is an edge between every pair of vertices.

What is the total number of edges in a complete graph ?



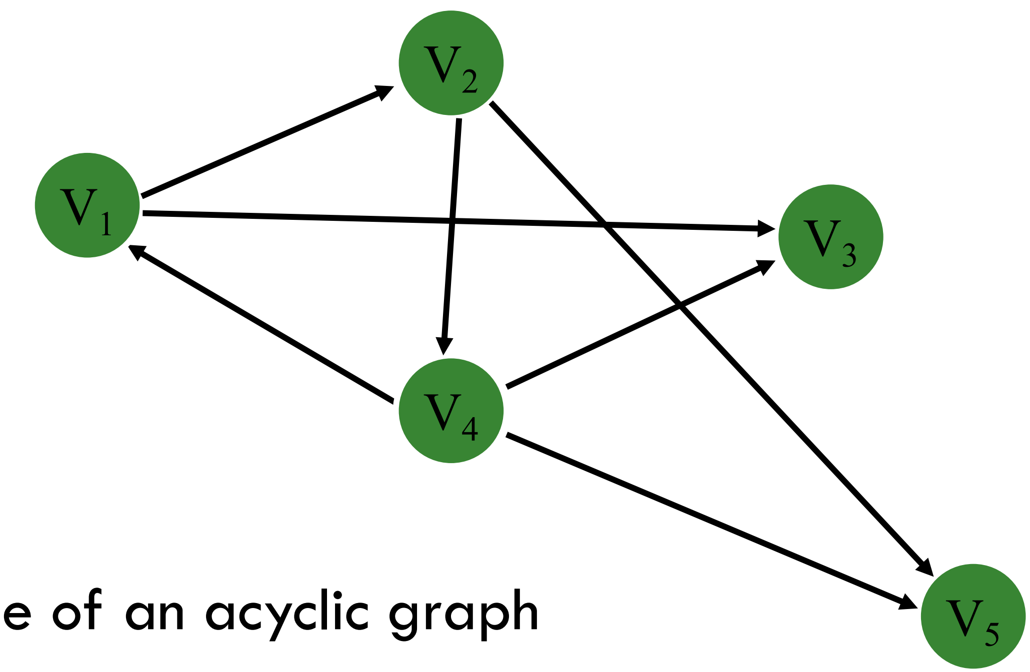
n = 5

m = $(5 * 4)/2 = 10$

DEFINITIONS (DAG)

A directed graph is *acyclic* if it has no cycles

DAG (directed acyclic graph)

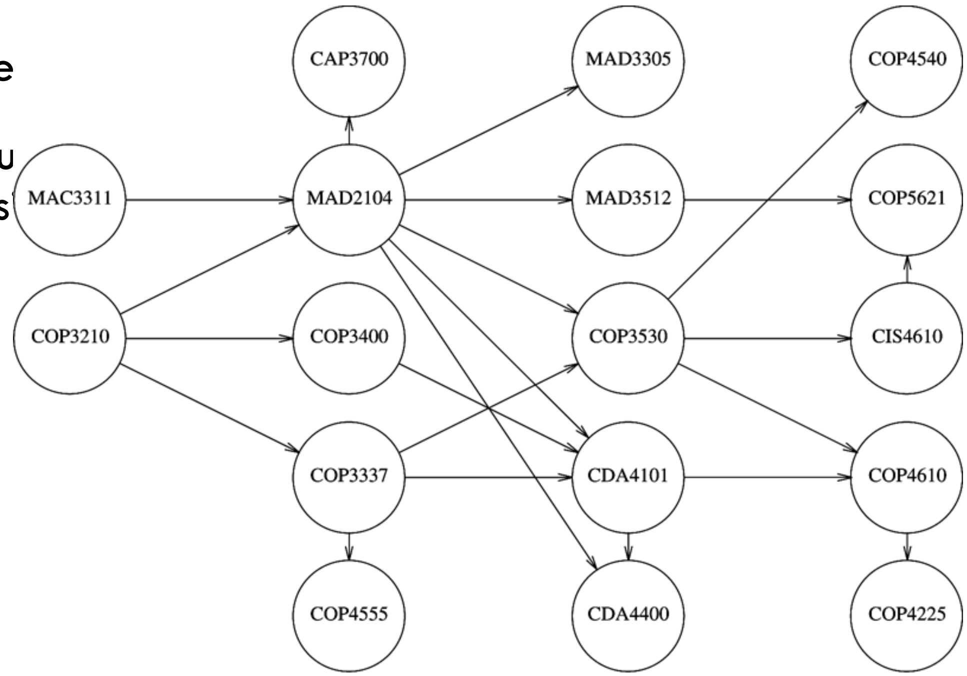


Give an example of an acyclic graph

TOPOLOGICAL SORT

Given a course prerequisite structure

Arrange the courses in a sequence su
that it does not violate the prerequis
criteria.



- An ordering of vertices in a *directed acyclic graph such that if there is a path from V_i to V_j , then V_j appears after V_i in the ordering.*
  - *A topological ordering is not possible if the graph has a cycle.*
  - The ordering is not necessarily unique.

PSEUDOCODE TO PERFORM TOPOLOGICAL SORT

```
void Graph::topsort( )
{
    Queue<Vertex> q;
    int counter = 0;

    q.makeEmpty( );
    for each Vertex v
        if( v.indegree == 0 )
            q.enqueue( v );

    while( !q.isEmpty( ) )
    {
        Vertex v = q.dequeue( );
        v.topNum = ++counter;  // Assign next number

        for each Vertex w adjacent to v
            if( --w.indegree == 0 )
                q.enqueue( w );
    }

    if( counter != NUM_VERTICES )
        throw CycleFoundException( );
}
```

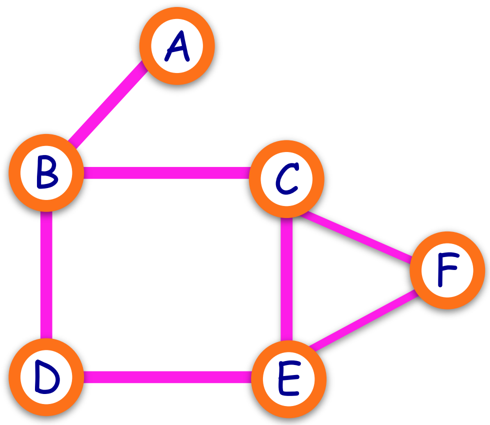
First, the indegree is computed for every vertex. Then all vertices of indegree 0 are placed on an initially empty queue.

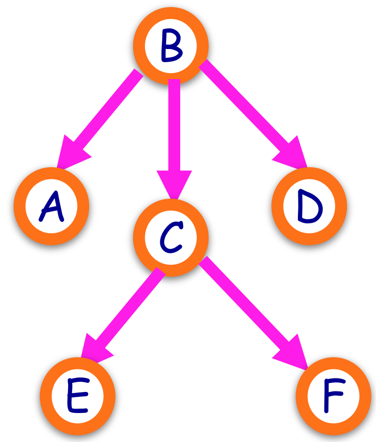
While the queue is not empty, a vertex v is removed, and all vertices adjacent to v have their indegrees decremented.

A vertex is put on the queue as soon as its indegree falls to 0.

Complexity O(|E|+|V|)

GRAPH TRAVERSAL

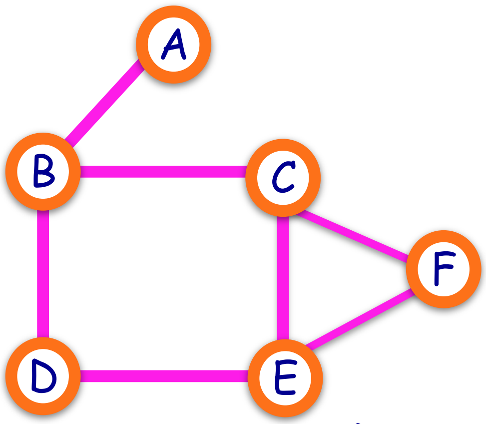


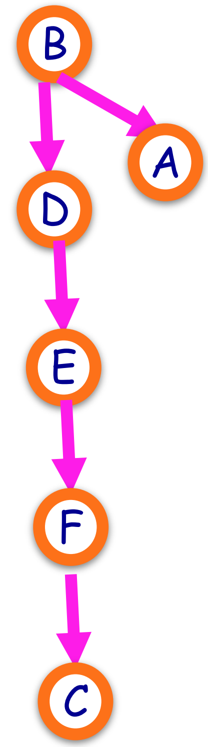


Breadth First Search O(V+E)

1. Start at a vertex
2. Mark as visited
3. Visit all unvisited neighbors
4. Mark neighbors as visited
5. Stop when all nodes are visited

GRAPH TRAVERSAL

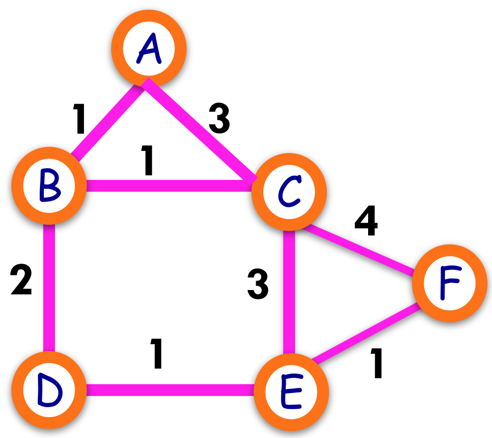


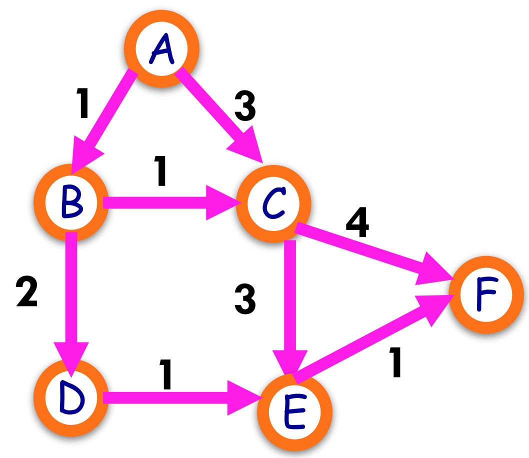


Depth First Search O(V+E)

1. Start at a vertex
2. Mark as visited
3. Visit any one of the unvisited neighbors
4. Mark the neighbor as visited
5. Continue until all vertices in a path are visited
6. Return to start with next unvisited vertex
7. Stop when all nodes are visited

TRAVERSING WEIGHTED GRAPHS





A graph is weighted if the edges have values associated with them

The single source shortest path (SSSP) problem considers the problem of finding the shortest distance from a given vertex to all other vertices.

Solved by Dijkstra's algorithm (Edsgar Dijkstra1956)
Similar to BFS, except we also visit previously visited nodes
And update the distance as needed

SINGLE SOURCE SHORTEST PATH

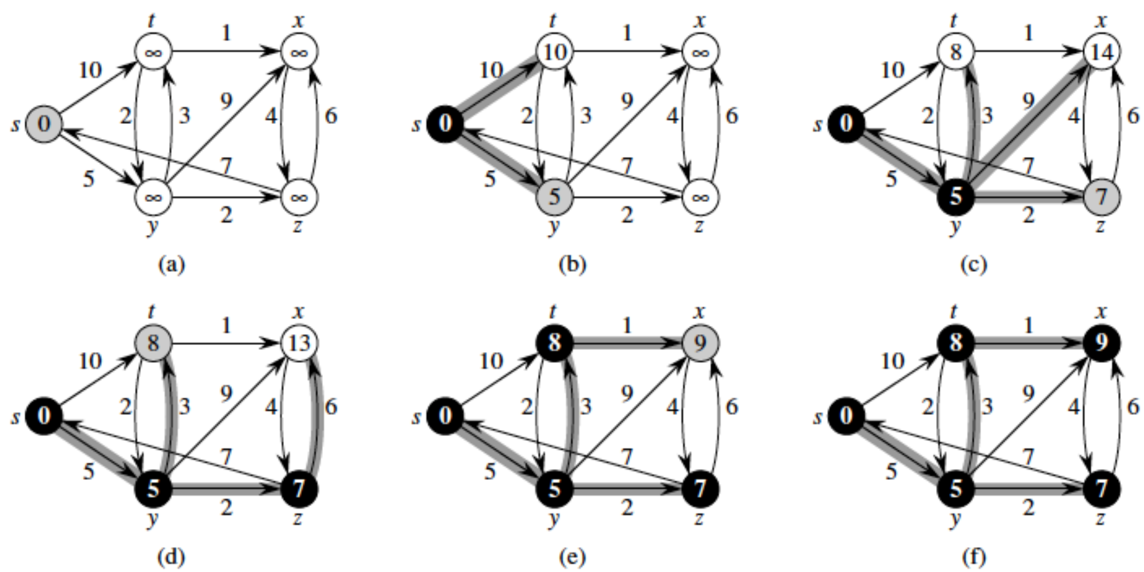
With Negative Weights (Bellman-Ford)

Without Negative Weights (Djikstra)

DIJKSTRA

```
DIJKSTRA(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
S ← Ø
Q ← V[G]
while Q ≠ Ø
     do u ← EXTRACT-MIN(Q)
          S ← S ∪ {u}
          for each vertex v ∈ Adj[u]
               do RELAX(u, v, w)
```

Complexity $O(V^2+E)$ or $O(V\log V+E)$ with better data structure



GRAPHS WITH NEGATIVE WEIGHTS

In Dijkstra's method once a node is extracted from the priority queue, it is not considered again.

Thus effect of negative weights not accounted

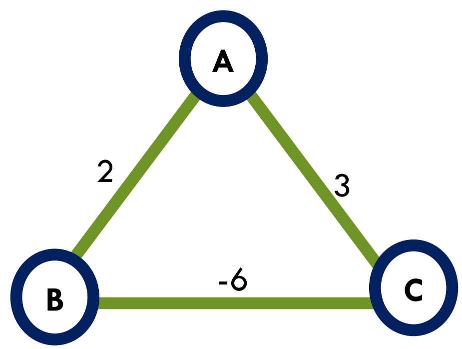
Bellman-Ford-Moore Algorithm (1958,1956,1959)

- For each vertex (Step 1: Initialize)
  - Distance d[v]=INF; pred[v]=NULL
- Repeat V-1 times (Step 2: Update repeatedly)
  - For each edge (u,v)
    - If d[v]>d[u]+weight(u,v)
    - d[v]=d[u]+weight(u,v) [update distance]
    - pred[v]=u;
- For each edge (u,v) (Step 3: Check for negative cycle)
  - If d[v]>d[u]+weight(u,v)
  - Report negative cycle

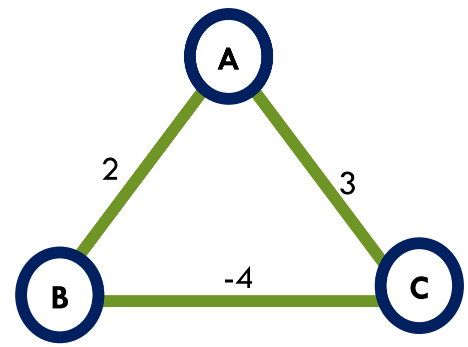
Total complexity O(|V|+|V|*|E|)

Shortest distance A-B is (A-C-B)

Dijkstra's algorithm will find 2 (A-B)



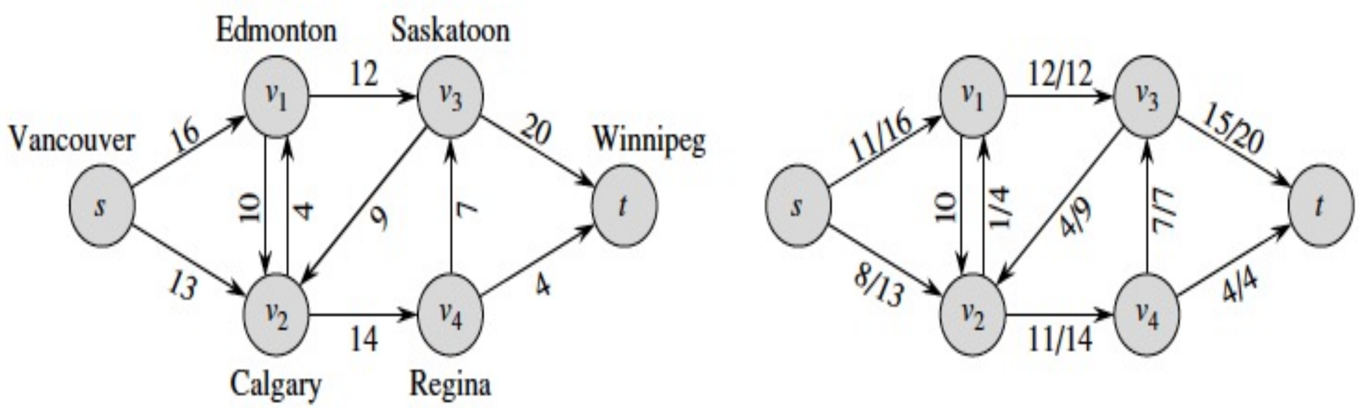
Negative Cycle



No Negative Cycle

MAX FLOW ALGORITHM

How do we find the maximum flow through a set of pipes/routes, etc.



(a) (b)

FLOW NETWORKS

A flow network is a directed graph where each edge has a non negative capacity

Follows the three properties;

Capacity constraint: For all $u, v \in V$, we require $f(u, v) \leq c(u, v)$.

Skew symmetry: For all $u, v \in V$, we require $f(u, v) = -f(v, u)$.

Flow conservation: For all $u \in V - \{s, t\}$, we require

$$\sum_{v \in V} f(u, v) = 0 .$$

BASIC ALGORITHMIC STEP

Find a Path from Source to Sink

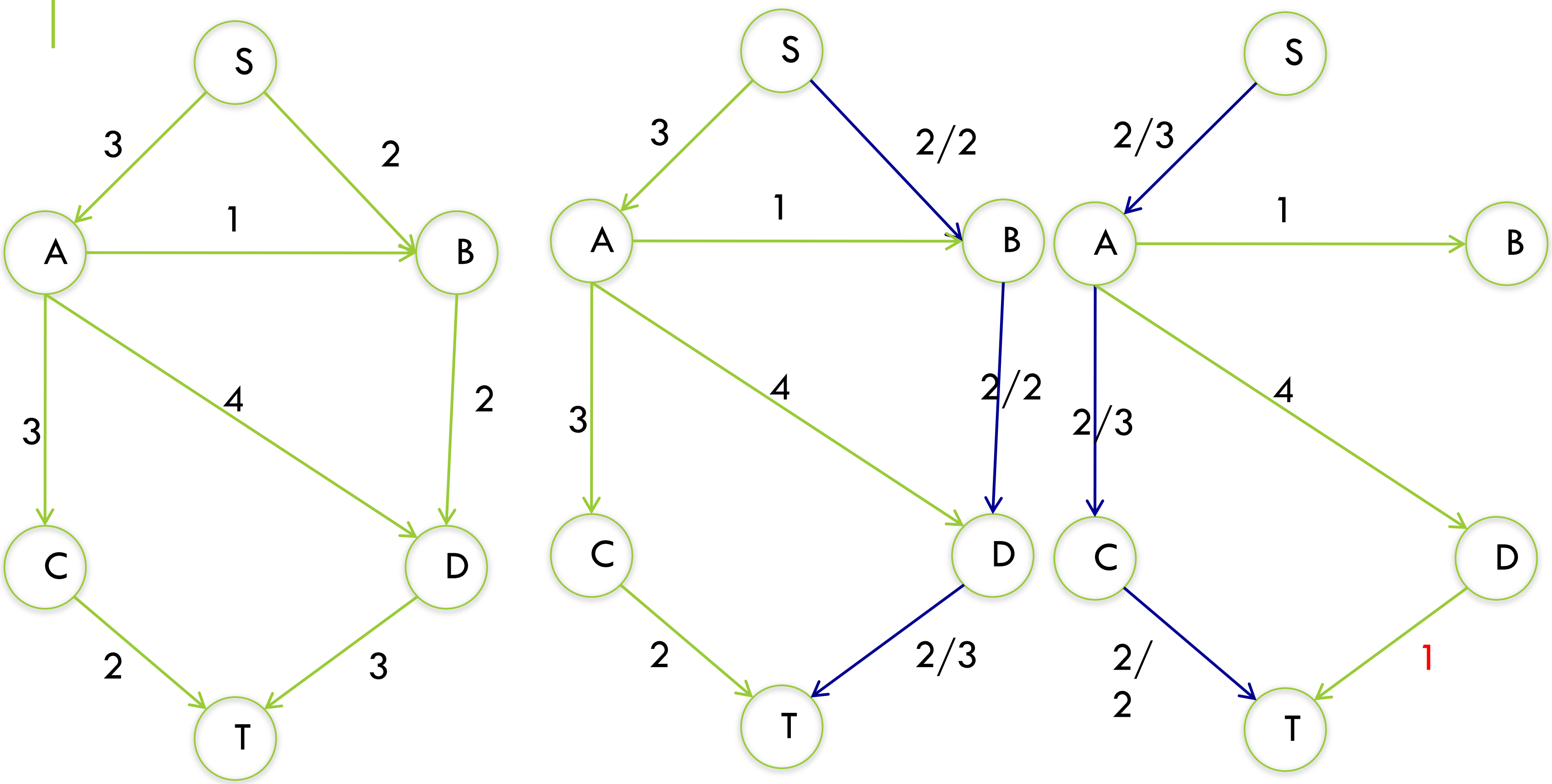
- Depth First from Source until we hit Sink

Update the flow along the path

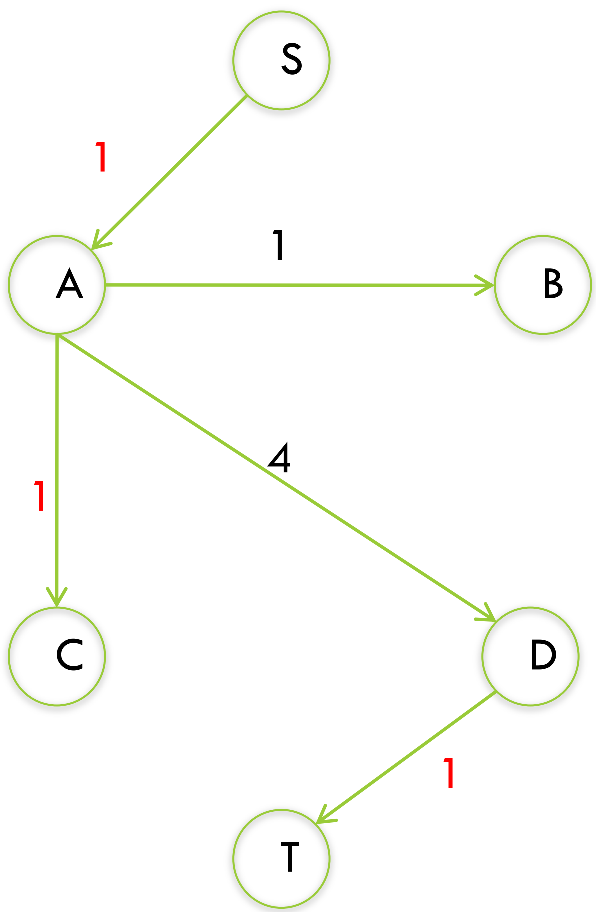
Remove the edges with 0 flow

Continue until there are no paths from Source to Sink

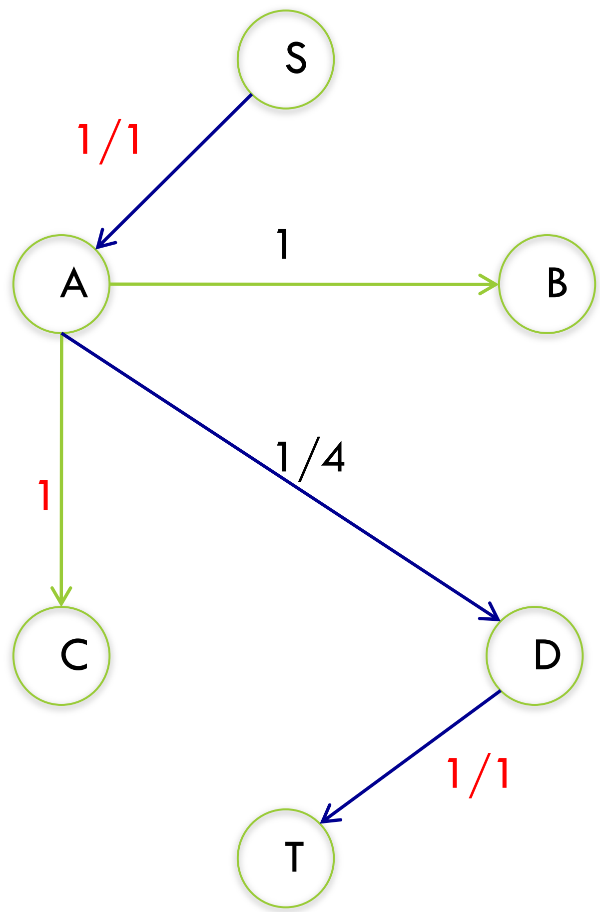
EXAMPLE



EXAMPLE -CONTINUED



Maximum Flow 5



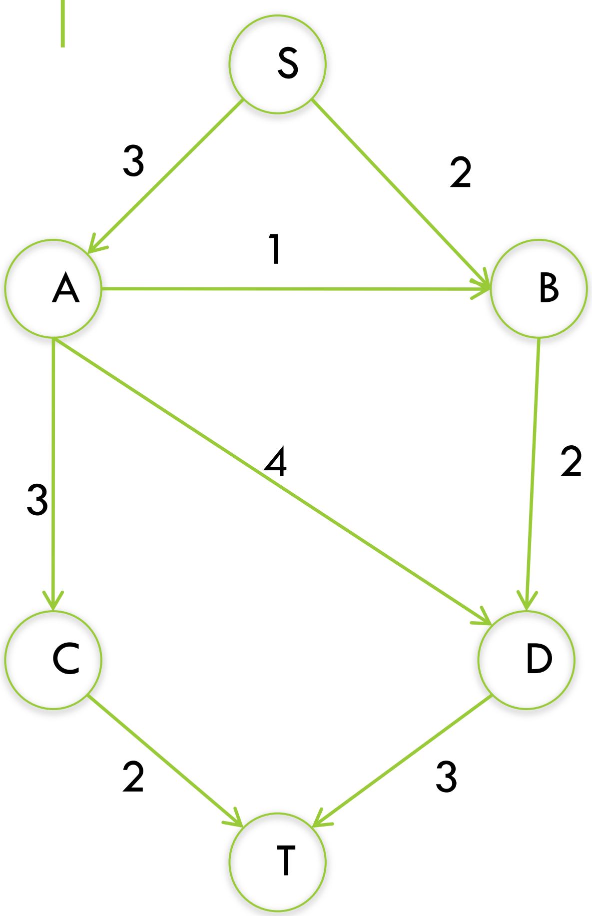
What is the problem with this algorithm ?

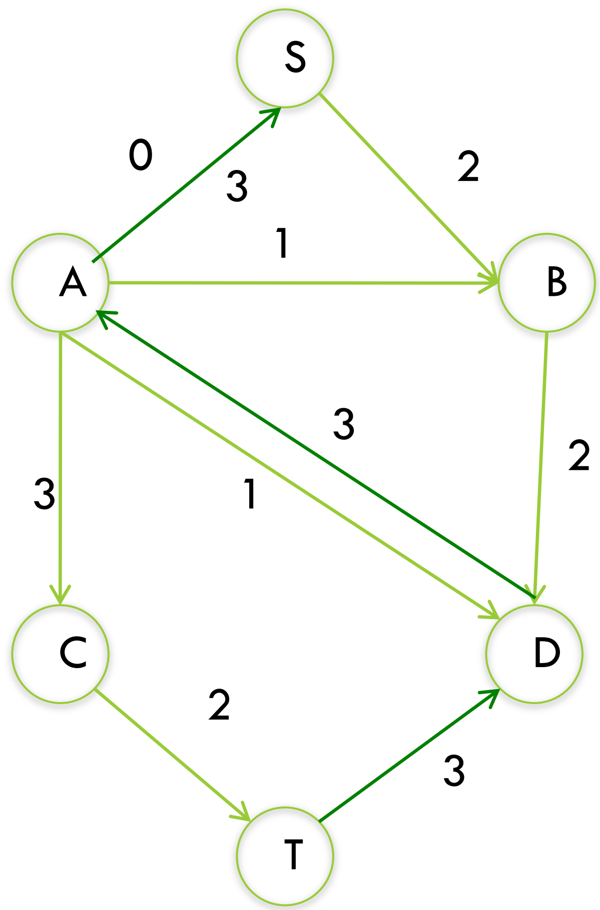
PROBLEM

Not guaranteed to find maximum flow if we start from the wrong edge

Need provision to retrace our steps

EXAMPLE





FORD-FULKERSON ALGORITHM

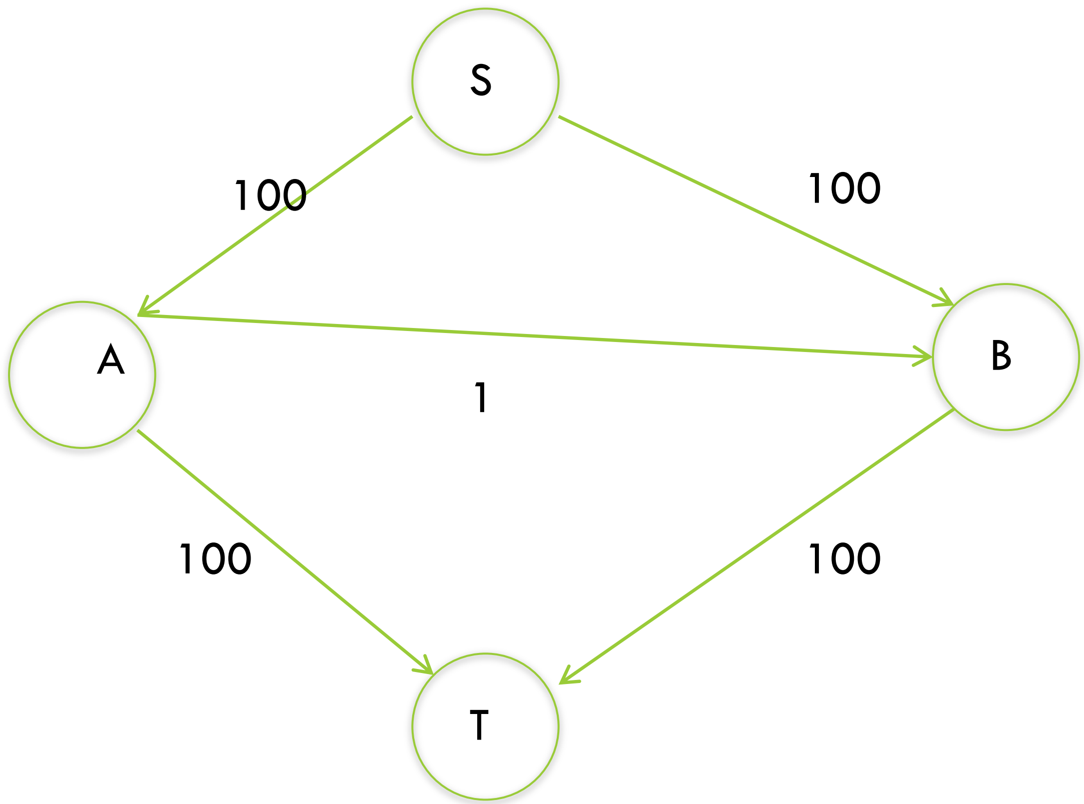
- Find a Path from Source to Sink
- Depth First from Source until we hit Sink
- Update the flow along the path
- Remove edges with 0 flow
- Add edges in opposite direction for reverse flow
- Continue until there are no paths from Source to Sink

Flow increases by at most 1 for each new path

Guaranteed to find maximum flow if the capacities are rational numbers

Execution time O(E *max_flow)

A BAD EXAMPLE



If we always choose the edge (a,b) or (b,a) we will need 100 steps to terminate

MODIFICATION

Always choose the augmenting path that allows largest increase to flow.

Like weighted shortest-paths (other way round)

Modification of Dijkstra's algorithm (Breadth First)

Edmond's-Karp Algorithm

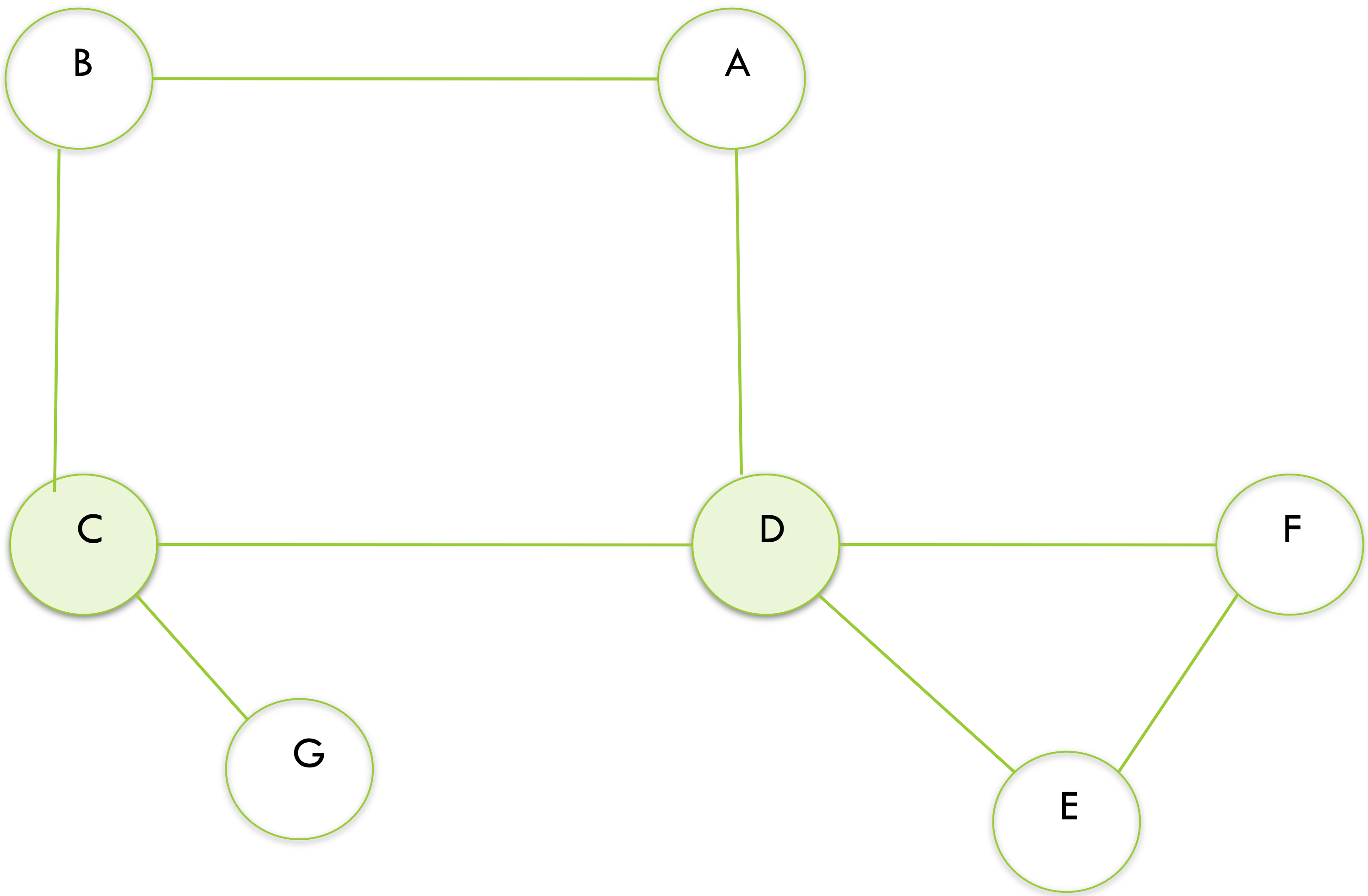
BICONNECTED GRAPHS

A connected undirected graph is biconnected if there are no vertices whose removal disconnects the rest of the graph

Vertices whose removal disconnects graphs are known as articulation points

Used in detecting crucial points in networks

EXAMPLE



FINDING ARTICULATION POINT

Do a depth First Traversal of the Graph

Create a tree with directed edges showing order of traversal

Set Num(v)=the order in which graphs are visted

Add remaining edges to the tree as back edges

Set Low(v) as the lowest vertex that can be reached using tree edges and at most one back edge

If for any vertex (other than root) v with child w Low(w) >= Num(v) , then v is an articulation point

If root has more than one child root is an articulation point

EXAMPLE