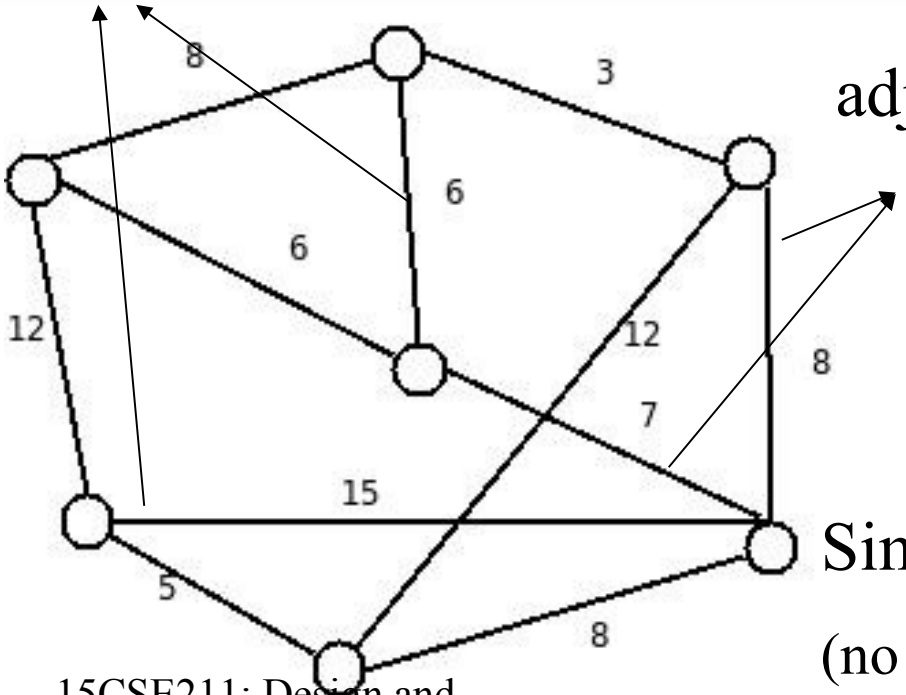
Graph Algorithms

Graph Definitions

- A graph G= (V,E) is a set of vertices V, and a collection of edges E which is a subset of VxV
  - Directed, undirected or mixed

independent

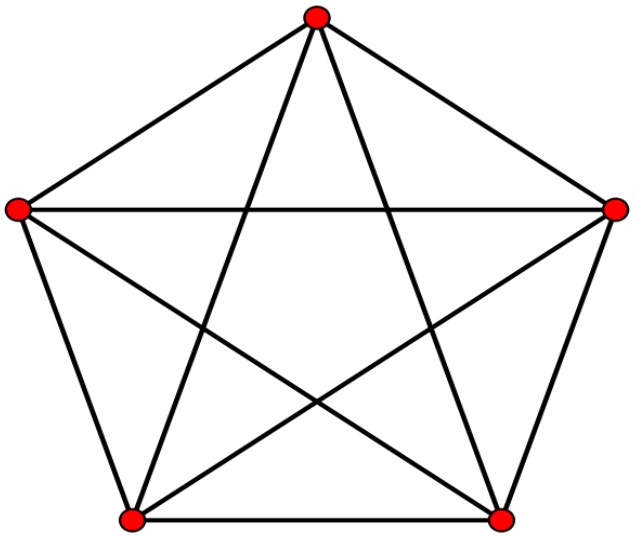


adjacent

Simple Graph

(no loops, parallel edges)

Complete Graph



Degree

- Number of edges incident at v (d(v))
  - In-degree – number of incoming edges to vertex v
  - Out-degree – number of outgoing edges
- An *isolated* vertex has degree 0
- Min degree of G --> $\delta(G) = \min \{d(v)| v \in V\}$
- Total degree of G is 2m (number of edges)
- To remember
  - Number of vertices of odd degree is always even in a graph

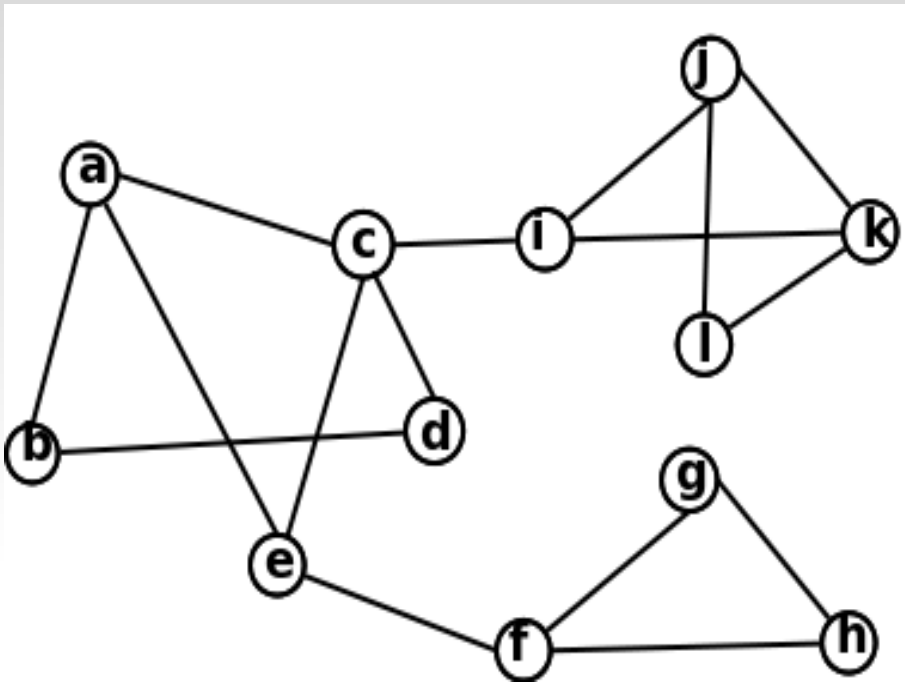
Graph Representations

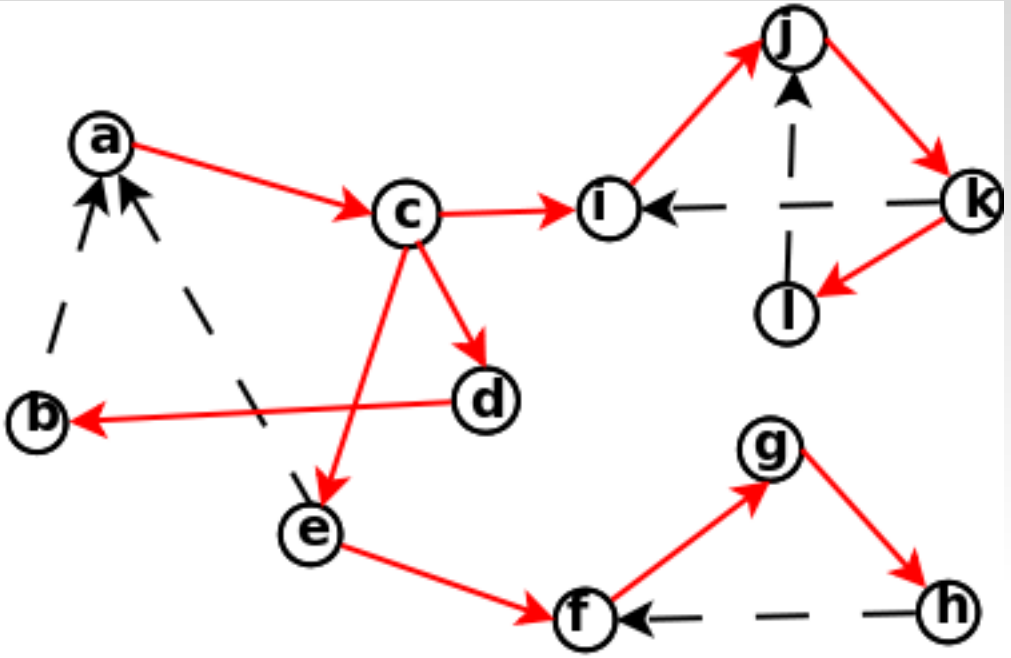
- Adjacency list
  - Each vertex has incidence container
    - List of vertices incident on v
  - Edge list
  - Useful in path algorithms or if graph is sparse
- Adjacency matrix
  - Edge list
  - Matrix A where A[i,j] represents edge
  - Edge retrieval quick
- Path – list of vertices connective u and v

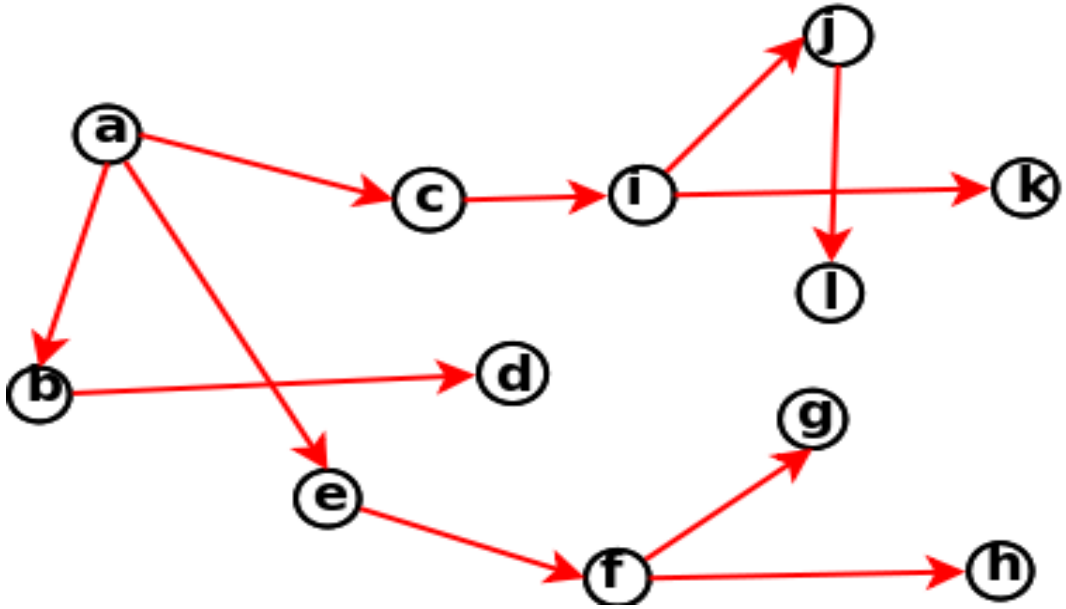
Graph Traversal

- Depth first search
  - Recursive – O(m+n) algorithm
  - Start with some node v
    - Of all neighbors of v, goto next w which is unexplored do DFS(w)
    - If w explored, then mark edge as back edge
- Breadth first search
  - Discovery in levels, marks new nodes in levels
  - O(m+n)
- Applications
  - Checking connectivity, cycle detection, finding biconnected components

Example







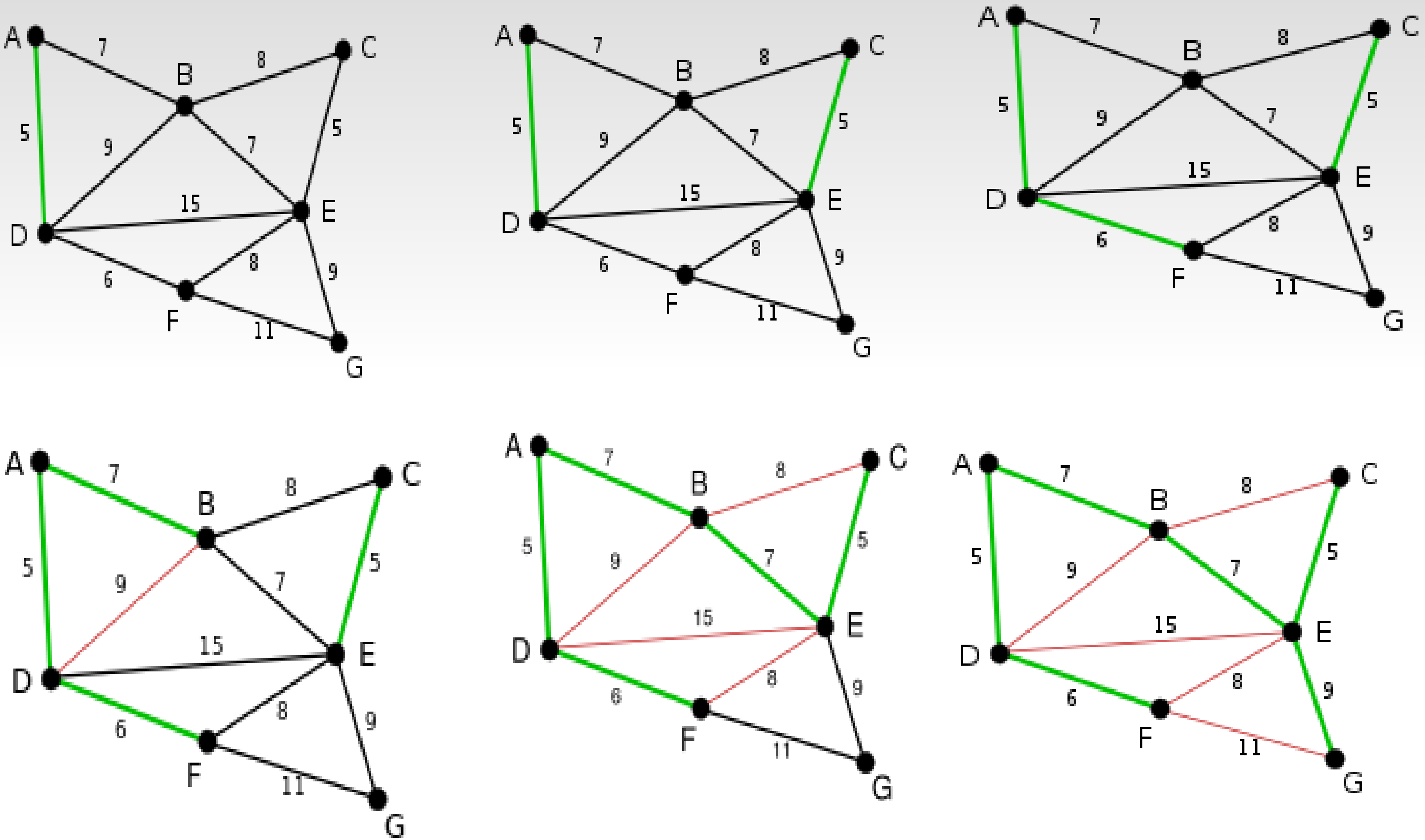
Minimum Spanning Tree

- Given a weighted undirected graph G, goal is to find a T such that
  - T contains all vertices in G
  - Sum of weights of edges in T is mimimum
- Different algorithms
  - Prim's
  - Kruskal
  - Boruvka's
- All use some greedy strategy

Kruskal's Algorithm

- Let every node in G be a cluster C(v)
- Initialize a priority queue Q with all edges in G using weights as keys
- Take the minimum weight edge in Q and if C(u) <> C(v)
  - add the edge to MST T
  - Merge the clusters C(u) and C(v)
- Repeat till there are no more clusters to merge

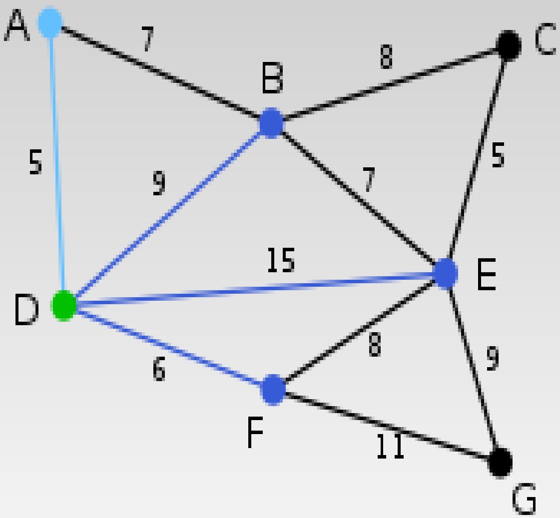
Kruskal's Algorithm: Example



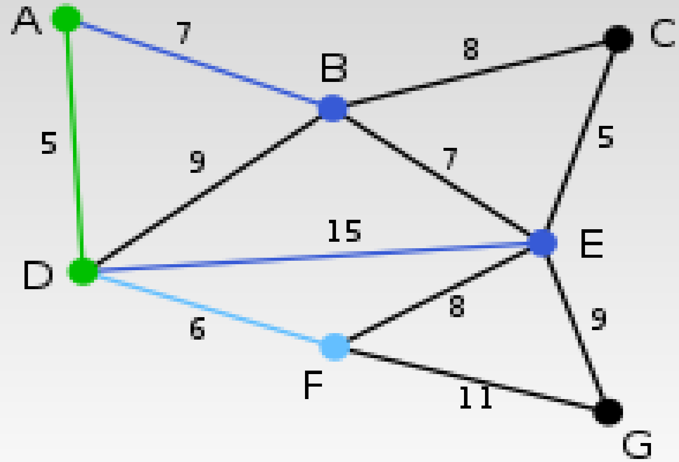
Prim's Algorithm

- Similar to Dijkstra
- Pick any vertex v of G
- Initialize for all u not v, d[u] to infinity and d[v]=0
- Remove from Q u with minimum d[u]
  - Add u and edge (u,v) to T
  - For all neighbors z of u, do relaxation by finding d[z] = w(u,z), and update Q

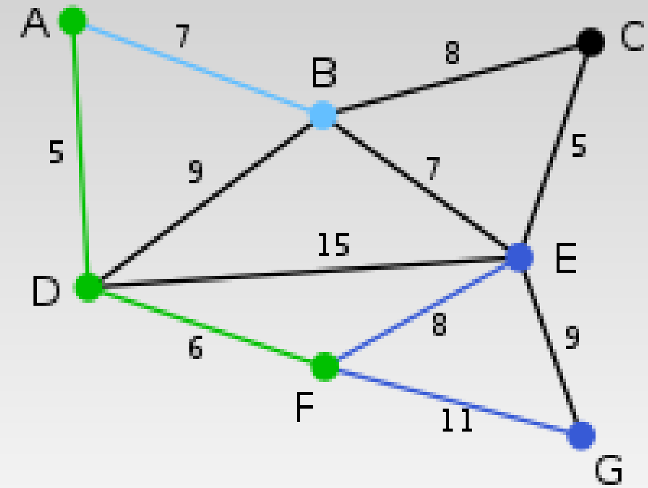
Example



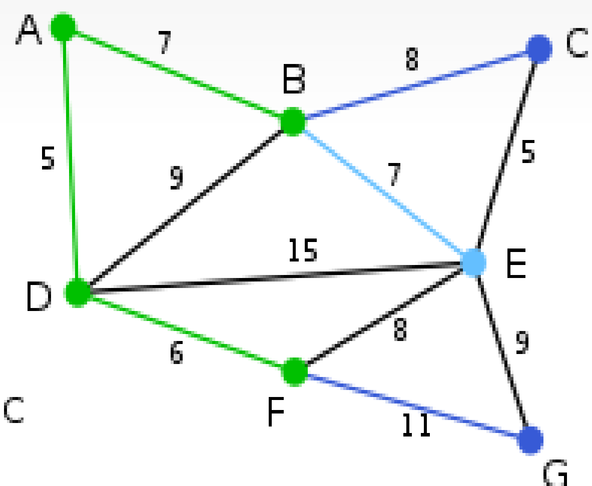
A
B
C
D
E
F
G
7
8
5
9
7
5
15
6
8
9
11



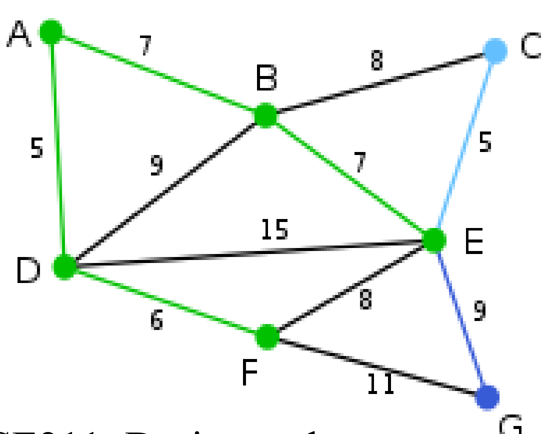
A
B
C
D
E
F
G
7
8
5
9
7
5
15
6
8
9
11



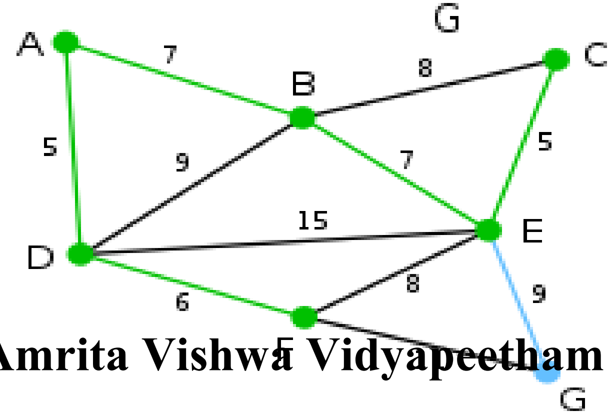
A
B
C
D
E
F
G
7
8
5
9
7
5
15
6
8
9
11



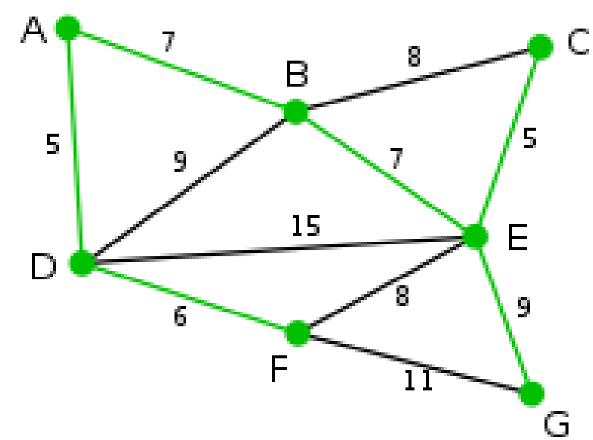
A
B
C
D
E
F
G
7
8
5
9
7
5
15
6
8
9
11



A
B
C
D
E
F
G
7
8
5
9
7
5
15
6
8
9
11



A
B
C
D
E
G
7
8
5
9
7
5
15
6
8
9

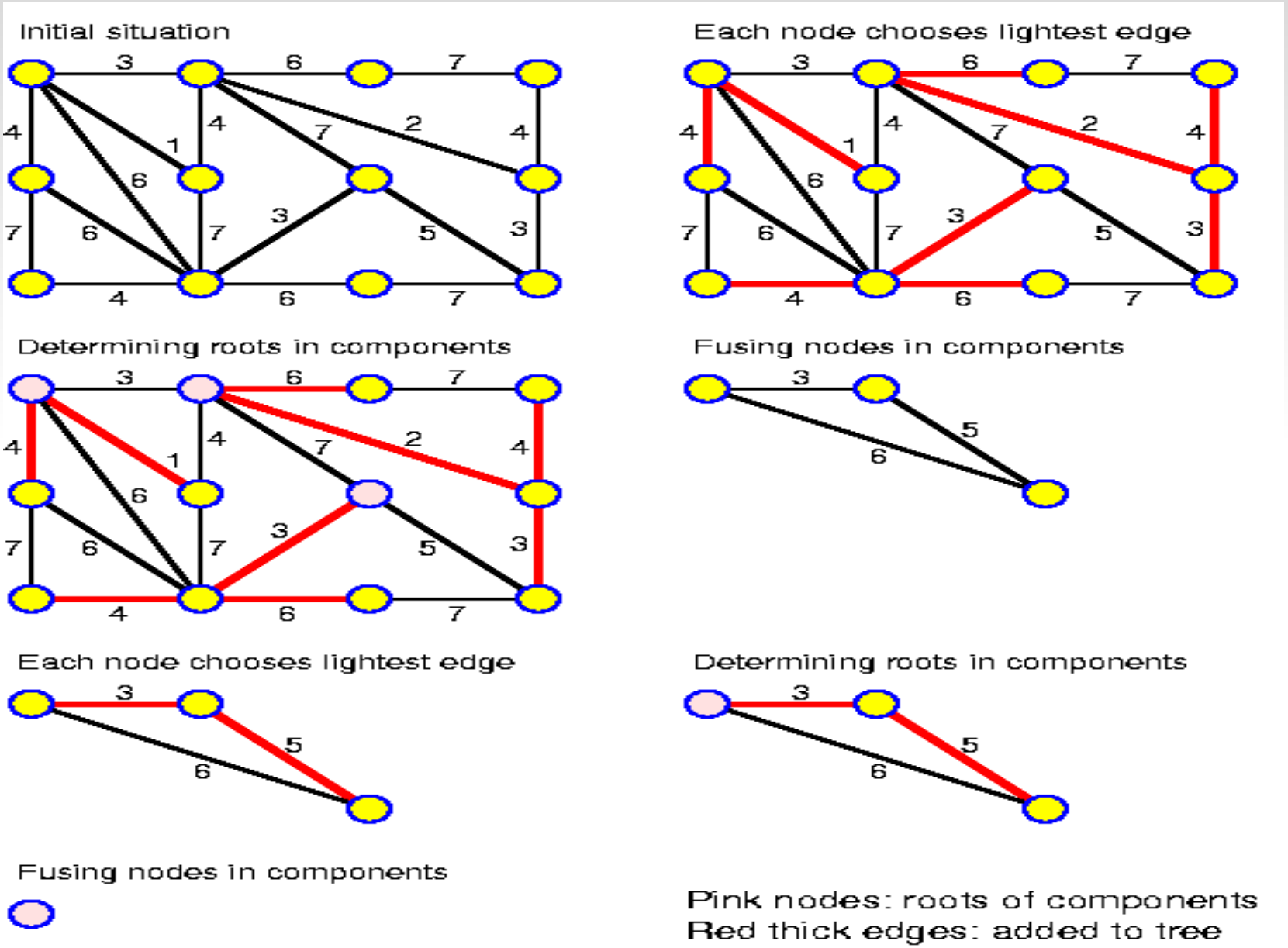


A
B
C
D
E
F
G
7
8
5
9
7
5
15
6
8
9
11

Baruvka's Algorithm

- Here greedy strategy explicitly optimizes certain priorities of vertices
- Initially every vertex is a component C_i
- Find the smallest weight edge (v,u) in E and add to C_i such that
  - v is in C_i, u is not in C_i
  - Add e to T

Example

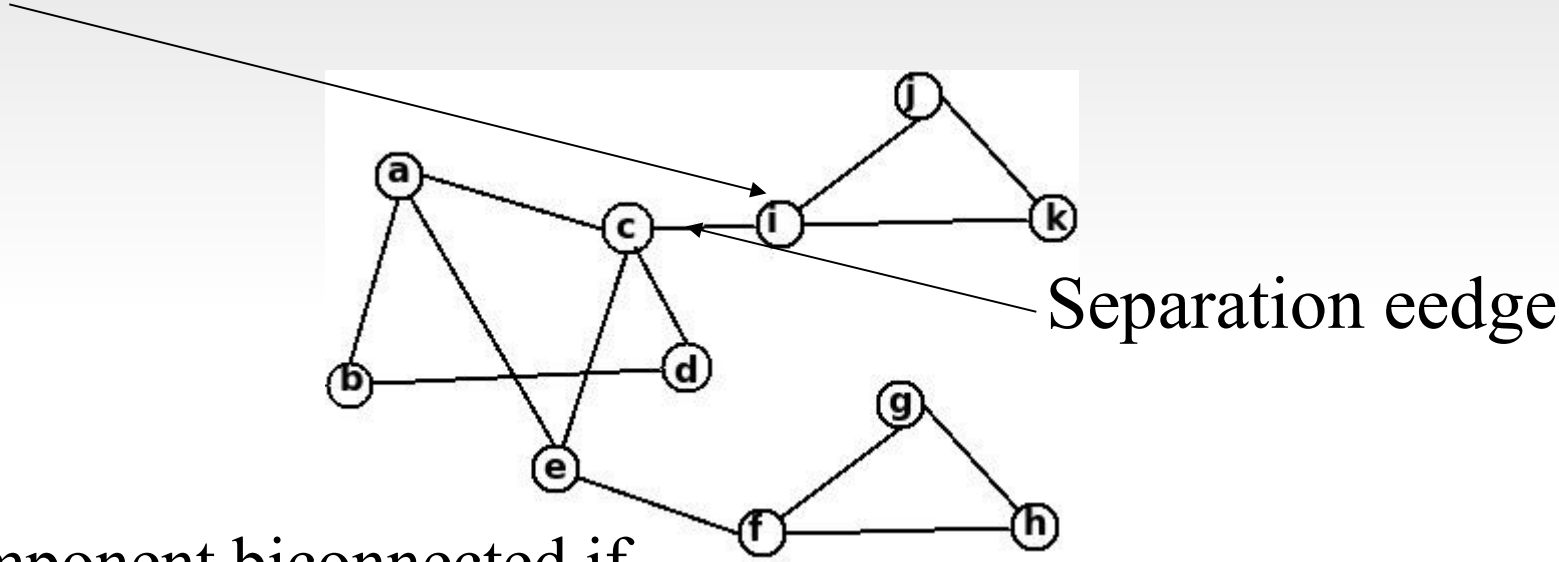


Vidhya Balasubramanian

http://www8.cs.umu.se/~jopsi/dinf504/boruvka.gif

Biconnected Components

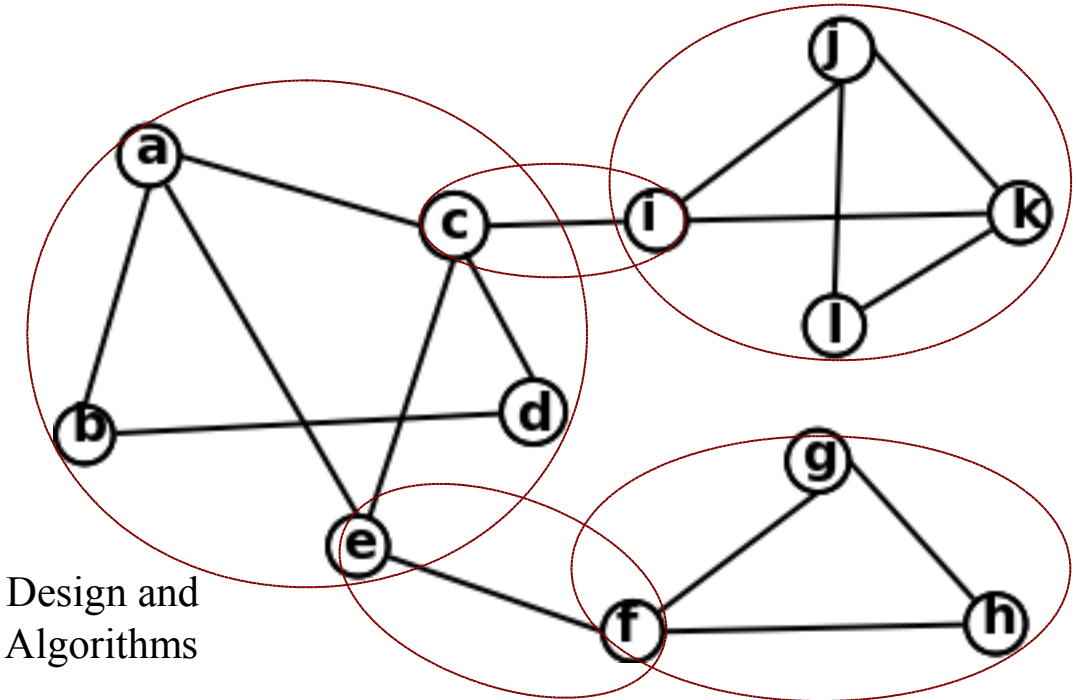
- Separation or cut edge - Edge whose removal disconnects a graph
- Separation vertex – vertex removal disconnects graph



- Graph/component biconnected if
  - There are no separation edge or vertices
  - For any two vertices u, v, there are 2 disjoint paths between them

Biconnected components

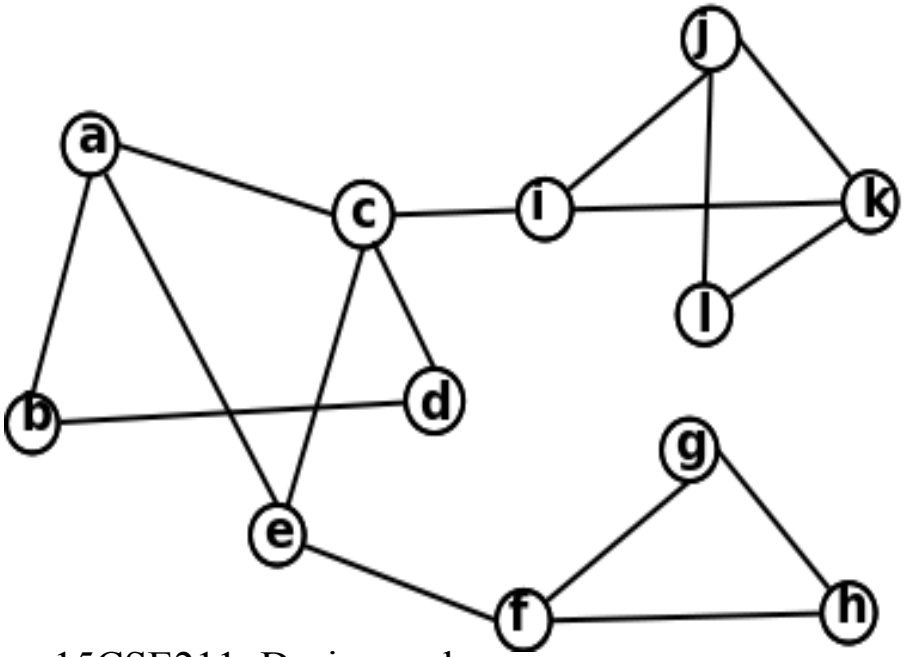
- Lemma : Let G be a biconnected graph. The following are equivalent
  - G is biconnected
  - For any two vertices of G, there is a simple cycle containing them
  - G does not have any separation vertices or separation edges

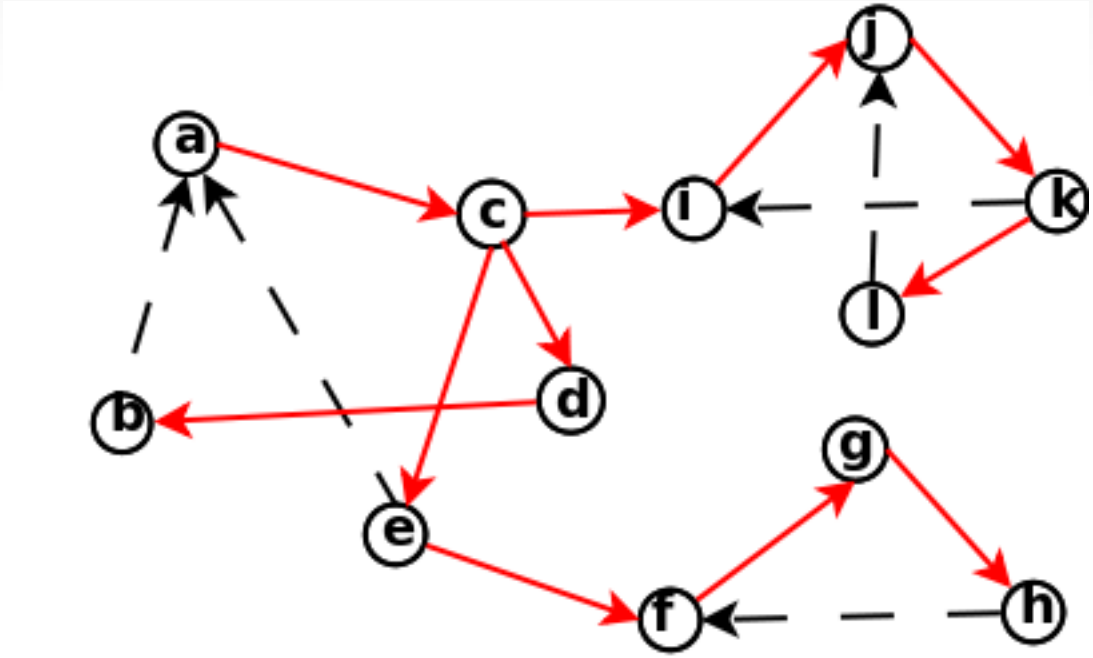


Compute Biconnected Components via DFS

- To construct biconnected components of G we need to compute the equivalence classes of the link relation among G's edges
- Steps

  1. Do a DFS traversal of G





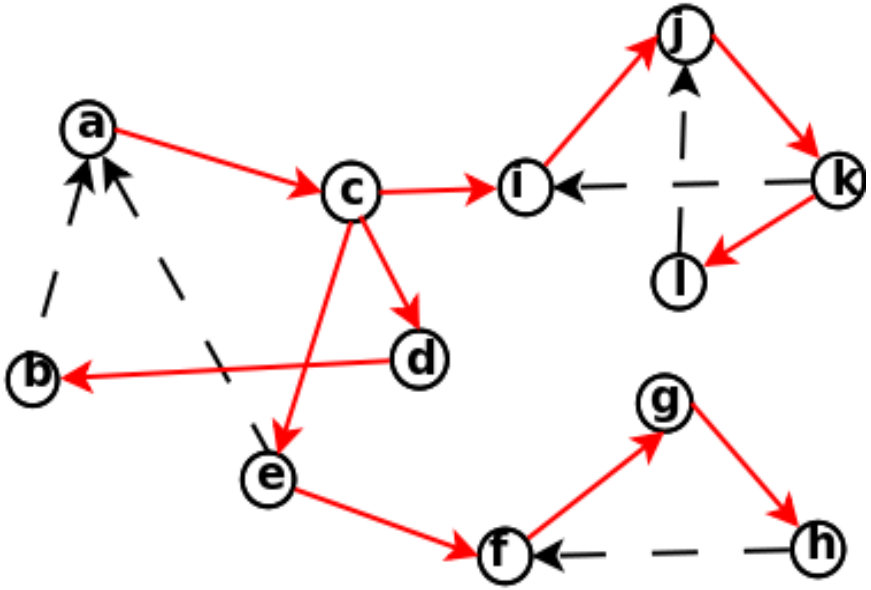
Auxiliary Graph

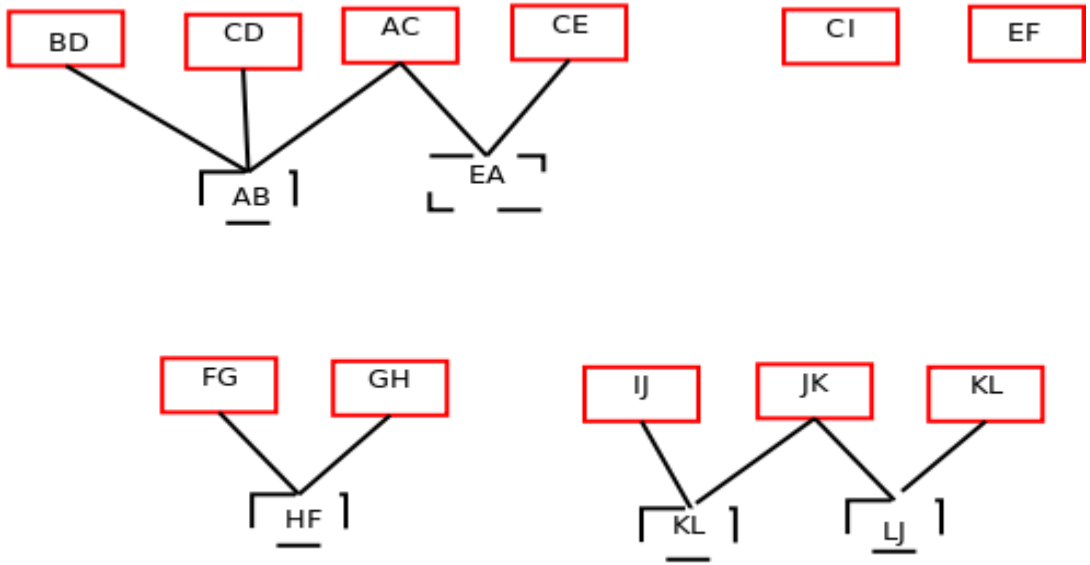
- Construct an auxiliary graph B as follows

  The vertices of B are edges of G

  For every back edge e of G, let f_1, ...f_k be the discovery edges of G that form a cylce with e

  Graph B contains edges (e,f_1), (e,f_2),...,(e,f_k)



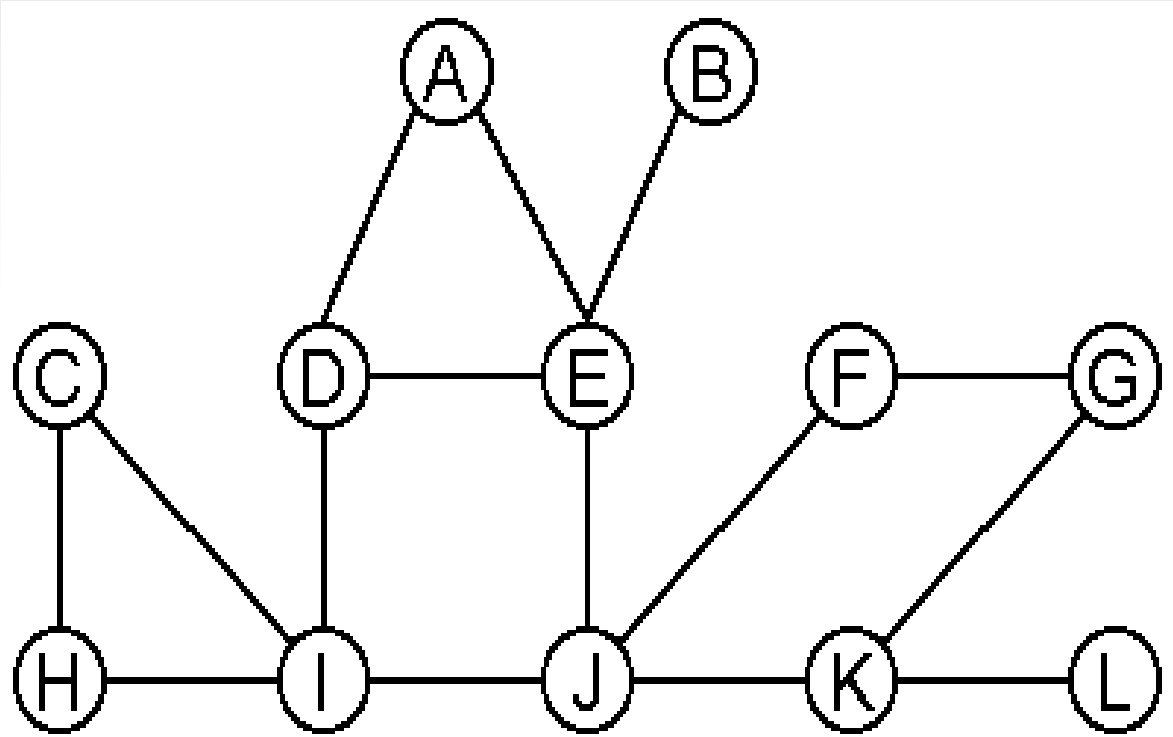


Finding the components

- The connected components of the auxiliary graph B correspond to link components of the graph G that induced B
  - Compute the connected components of B for example by performing a DFS traversal of the auxiliary graph B
  - For each connected component of B , output the vertices of B as a link component of G
    - Vertices of B are edges of G
- Complexity
  - Finding separation edges and vertices – O(m+n)
  - Finding auxiliary graph O(mn)
    - Reduce this by finding a spanning forest of B

Exercise

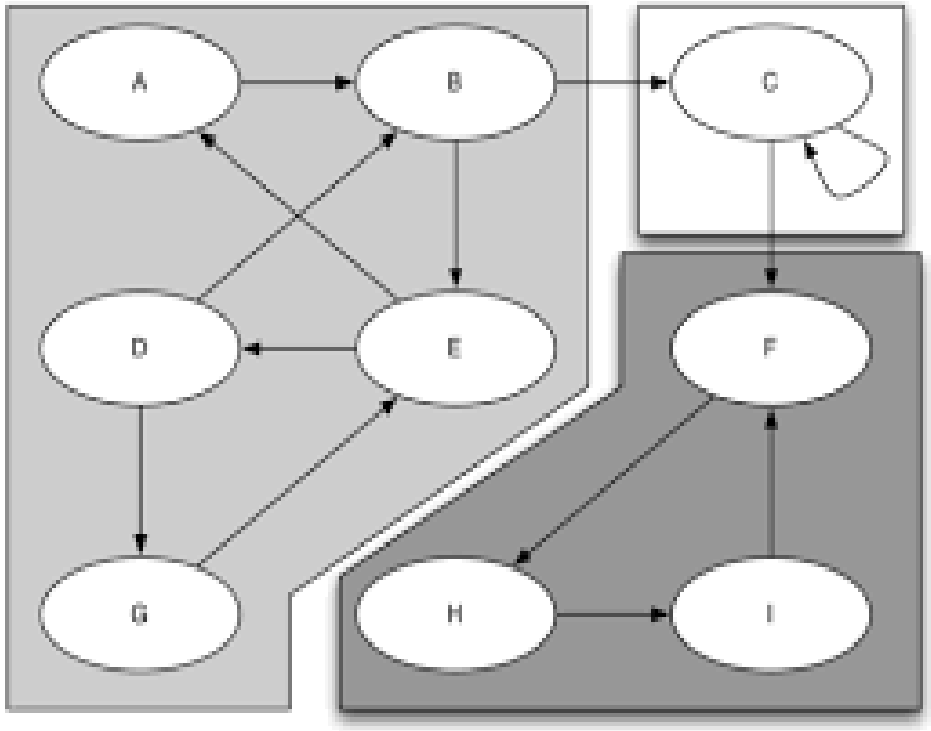
- Find all the biconnected components of this graph using DFS method



Strongly Connected Components

Two nodes a and b in a directed graph are connected if there are paths between a to b and vice versa

- A directed graph is strongly connected if there are directed paths between every pair of nodes
- Directed graph is DAG of its strongly connected components



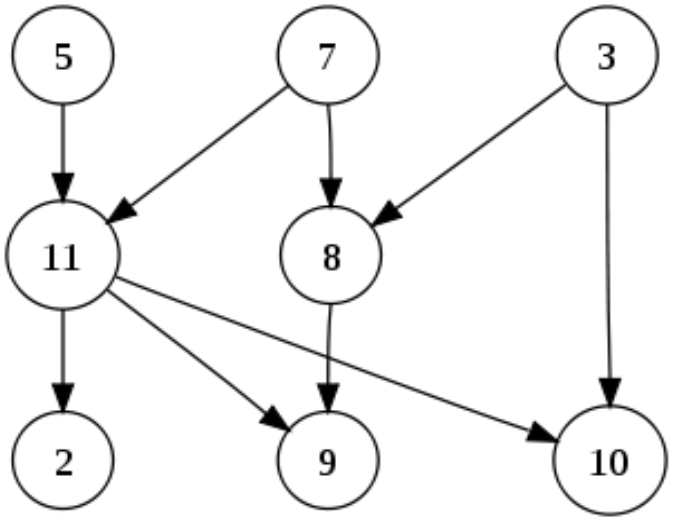
Use DFS to find the components

Apply DFS on graph and then on transpose graph G'

G' has same set of vertices with all of the edges reversed compared to the orientation of the corresponding edges in G

Digraphs

- A directed graph or digraph is a pair G =(V,E) of
  - a set V, whose elements are called vertices or nodes
  - a set A of ordered pairs of vertices, called arcs, directed edges, or arrows
- A directed acyclic graph (DAG) is a directed graph with no directed cycles



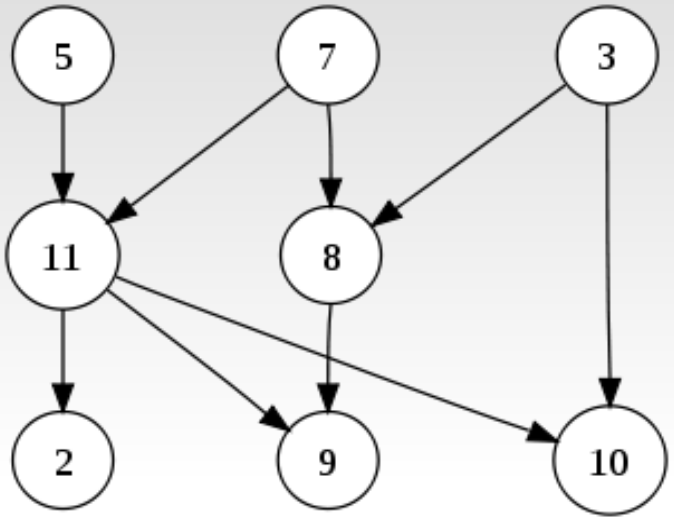
Digraph Application

- Used to model several kinds of structures in maths or science
- Scheduling
  - Edge (a, b) indicates task a must be completed before b
- Topological Ordering
- Data Processing Networks
  - data enters a processing element through its incoming edges and leaves the element through its outgoing edges
  - Bayesian Networks
  - Compilers
  - Circuit Design

Topological Ordering

- A DAG is a digraph that has no directed cycles and a topological ordering of a digraph is a numbering v_1 , ..., v_n of the vertices such that
  - for every edge (v_i , v_j), we have i < j
- Example
  - In task scheduling, a topological ordering is a task sequence that satisfies the precedence constraints
- Theorem
  - A digraph admits a topological ordering if and only if it is a DAG

Example



- For the above graph these are some valid topological ordering
  - 5, 7, 3, 11, 8, 2, 9, 10
  - 3, 5, 7, 8, 11, 10, 9, 2

Topological Sorting Algorithm

- Let S be an empty stack
- For each vertex u of G, set incounter(u) as the indegree(u)
  - If incounter(u) =0, push it in the stack
- Set i as 1
- If S is empty it has a directed cycle
- Pop the ith vertex u_i from the stack and increment i
  - For each neighbor v of u_i, decrement incounter(v) by 1
  - If incounter(v) =0, push it in stack
  - Do this this while stack has elements in it

Analysis

- Runs in O(n+m) time – n vertices and m edges
  - Initial computation of indegree O(n+m)
- Uses O(n) auxiliary space (stack)
- If some vertices are not numbered then there is a cycle
  - Any vertex on a directed cycle will not be visited
  - A vertex visited only when incounter is 0
    - Implies all its previous predecessors were previously visited