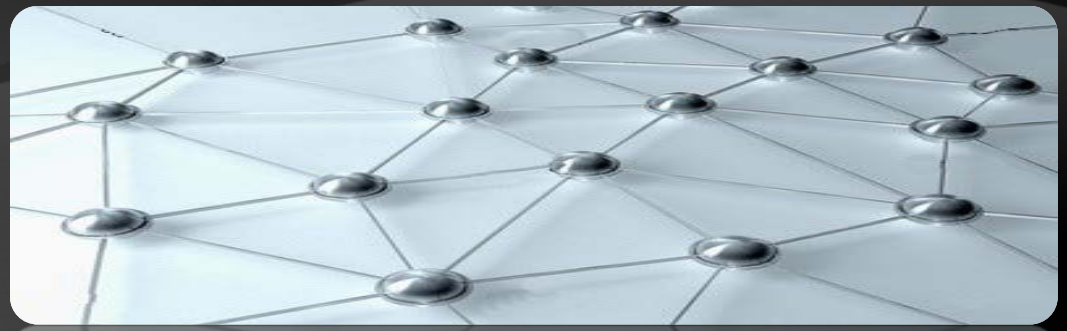
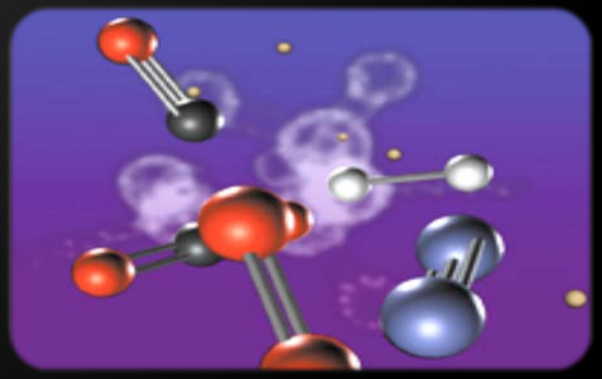
Telerik Academy



Graphs

Fundamentals, Terminology, Traversal, Algorithms



Data Structures and Algorithms

Telerik Software Academy

http://academy.telerik.com

Telerik Academy

Table of Contents

1. Graph Definitions and Terminology
2. Representing Graphs
3. Graph Traversal Algorithms
4. Connectivity
5. Dijkstra's Algorithm
6. Topological sorting
7. Prim and Kruskal

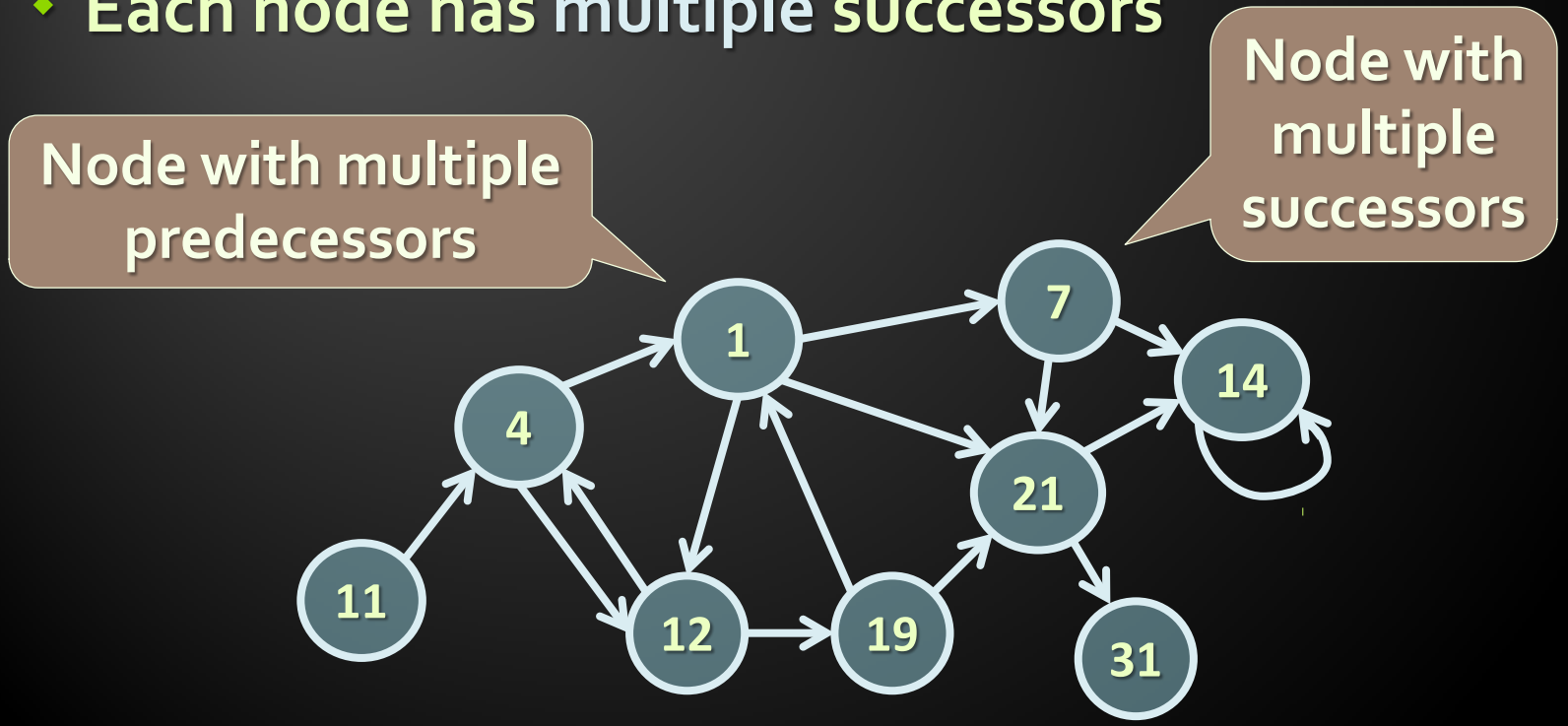
Telerik Academy

Graphs

Definitions and Terminology

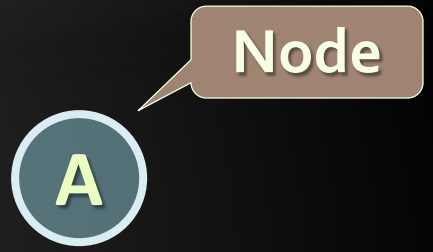
Graph Data Structure

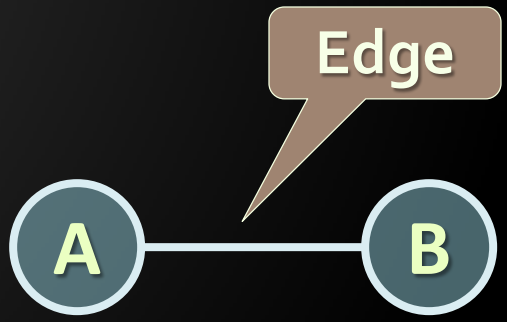
- Set of nodes with many-to-many relationship between them is called graph
  - Each node has multiple predecessors
  - Each node has multiple successors



Graph Definitions

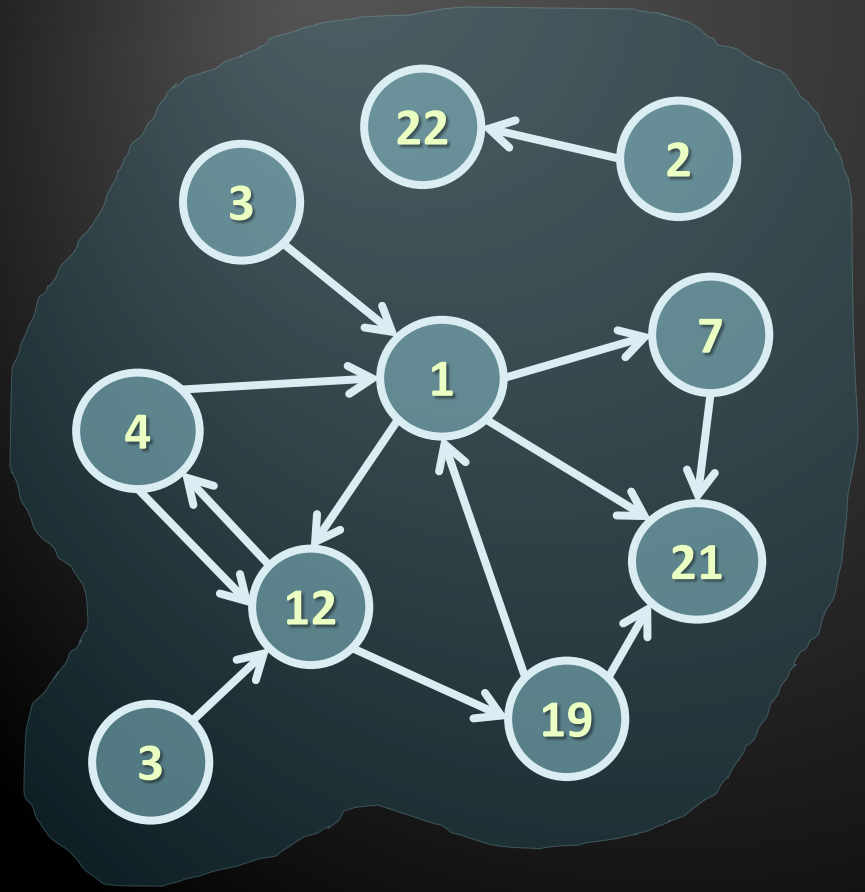
- Node (vertex)
  - Element of graph
  - Can have name or value
  - Keeps a list of adjacent nodes
- Edge
  - Connection between two nodes
  - Can be directed / undirected
  - Can be weighted / unweighted
  - Can have name / value



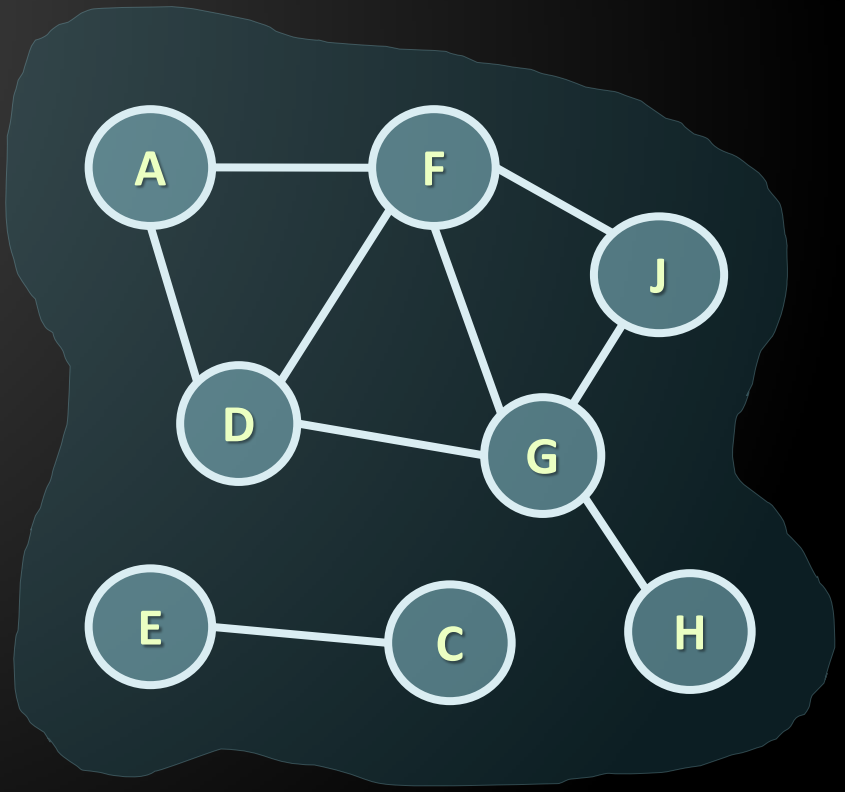


Graph Definitions (2)

- Directed graph
  - Edges have direction

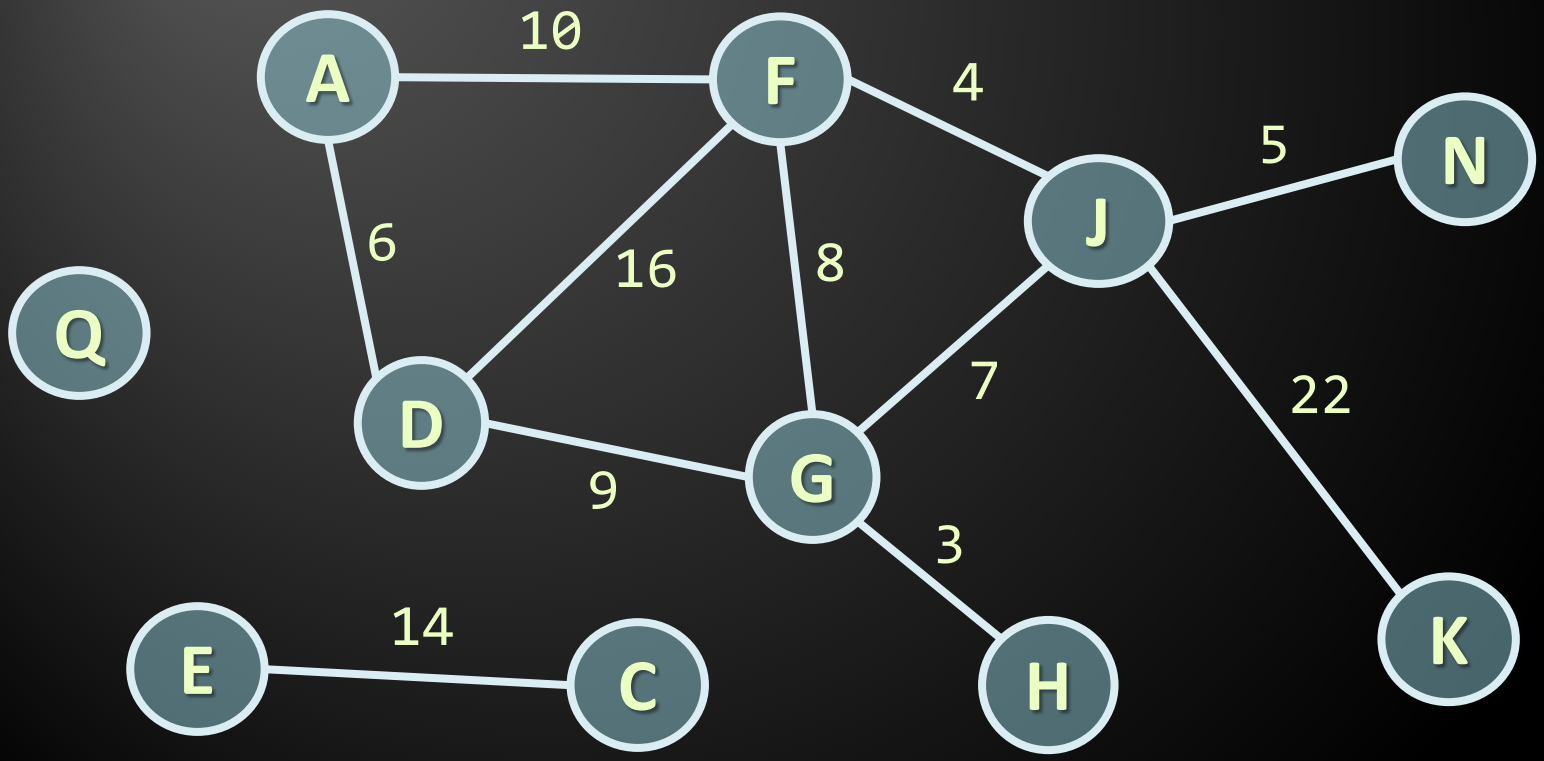


- Undirected graph
  - Undirected edges



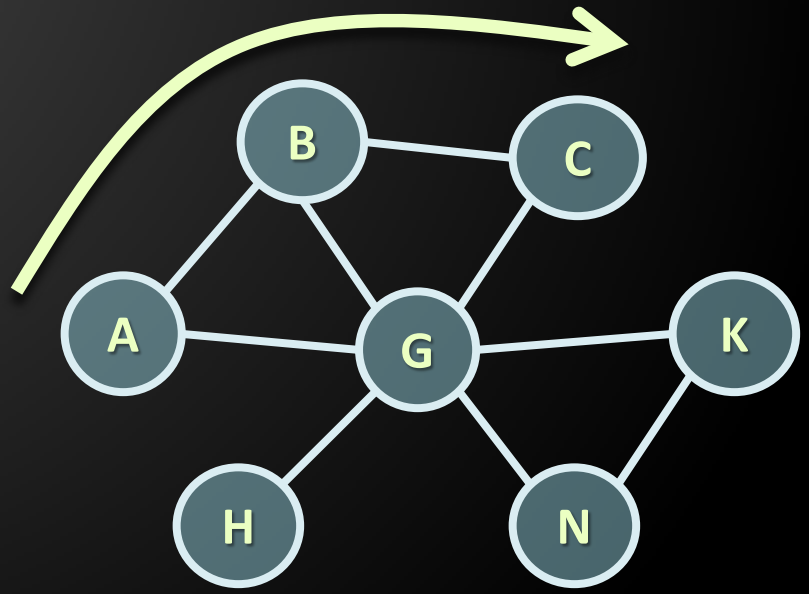
Graph Definitions (3)

- Weighted graph
  - Weight (cost) is associated with each edge



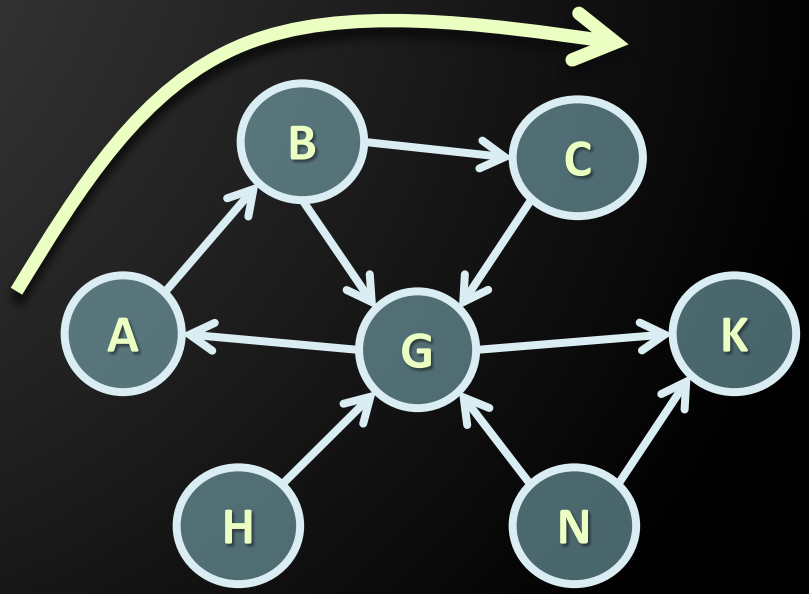
Graph Definitions (4)

- Path (in undirected graph)
  - Sequence of nodes n_1, n_2, … n_k
  - Edge exists between each pair of nodes n_i, n_{i+1}
  - Examples:
    - A, B, C is a path
    - H, K, C is not a path



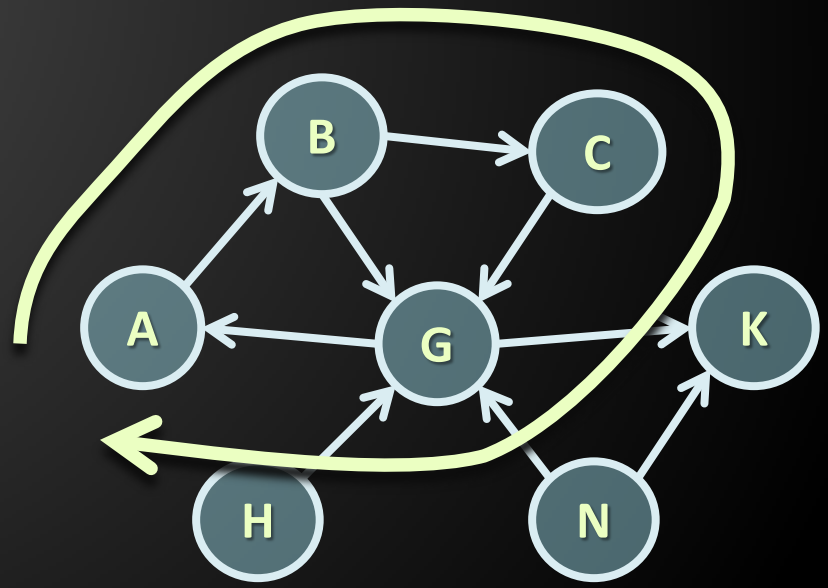
Graph Definitions (5)

- Path (in directed graph)
  - Sequence of nodes n_1, n_2, … n_k
  - Directed edge exists between each pair of nodes n_i, n_{i+1}
  - Examples:
    - A, B, C is a path
    - A, G, K is not a path



Graph Definitions (6)

- Cycle
  - Path that ends back at the starting node
  - Example:
    - A, B, C, G, A
- Simple path
  - No cycles in path
- Acyclic graph
  - Graph with no cycles
  - Acyclic undirected graphs are trees

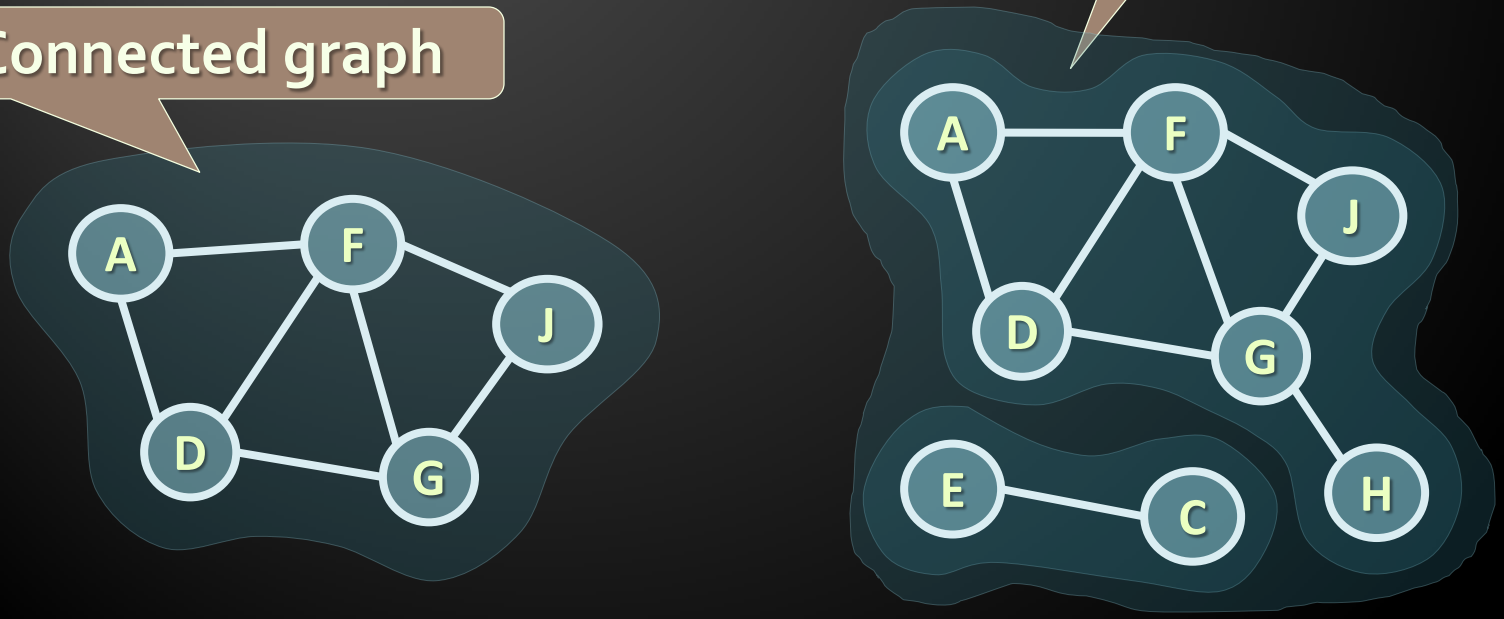


Graph Definitions (7)

- Two nodes are reachable if
  - Path exists between them
- Connected graph
  - Every node is reachable from any other node

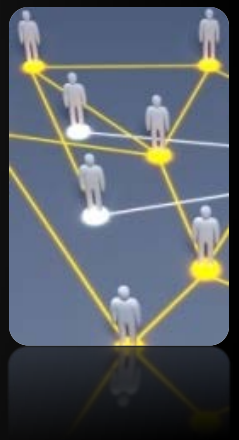
Unconnected graph with two connected components

Connected graph



Graphs and Their Applications

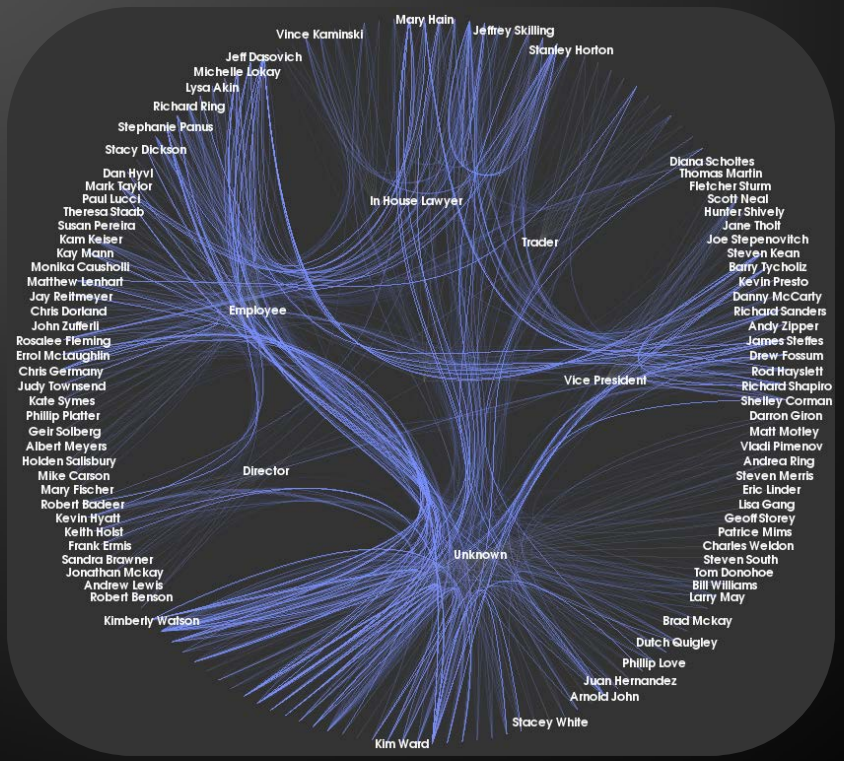
- Graphs have many real-world applications
  - Modeling a computer network like Internet
    - Routes are simple paths in the network
  - Modeling a city map
    - Streets are edges, crossings are vertices
  - Social networks
    - People are nodes and their connections are edges
  - State machines
    - States are nodes, transitions are edges



Telerik Academy

Representing Graphs

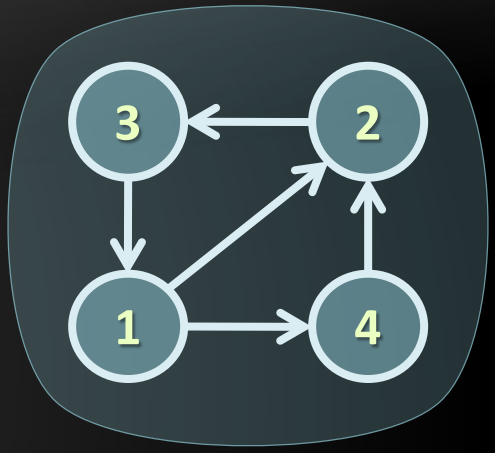
Classic and OOP Ways



Representing Graphs

- Adjacency list
  - Each node holds a list of its neighbors

```
1 → {2, 4}
2 → {3}
3 → {1}
4 → {2}
```



- Adjacency matrix
  - Each cell keeps whether and how two nodes are connected

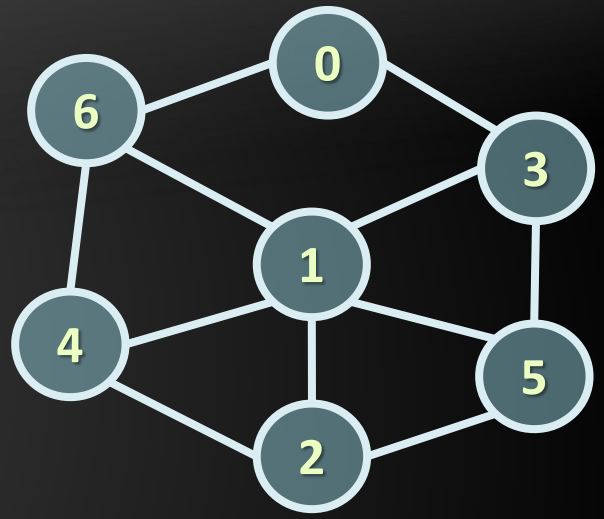
	1	2	3	4
1	0	1	0	1
2	0	0	1	0
3	1	0	0	0
4	0	1	0	0

- Set of edges

```
{1,2} {1,4} {2,3} {3,1} {4,2}
```

Simple C# Representation

```
public class Graph
{
  List<int>[] childNodes;
  public Graph(List<int>[] nodes)
  {
    this.childNodes = nodes;
  }
}
```




```
Graph g = new Graph(new List<int>[] {
  new List<int> {3, 6}, // successors of vertice 0
  new List<int> {2, 3, 4, 5, 6},// successors of vertice 1
  new List<int> {1, 4, 5}, // successors of vertice 2
  new List<int> {0, 1, 5}, // successors of vertice 3
  new List<int> {1, 2, 6}, // successors of vertice 4
  new List<int> {1, 2, 3}, // successors of vertice 5
  new List<int> {0, 1, 4}  // successors of vertice 6
});
```

Telerik Academy

Advanced C# Representation

- Using OOP:
  - Class Node
  - Class Connection (Edge)
  - Class Graph
  - Optional classes
- Using external library:
  - QuickGraph - http://quickgraph.codeplex.com/

Telerik Academy

Representing Graphs

Live Demo

Telerik Academy

Traversing Graphs

Good old DFS and BFS

Graph Traversal Algorithms

- Depth-First Search (DFS) and Breadth-First Search (BFS) can traverse graphs
  - Each vertex should be visited at most once

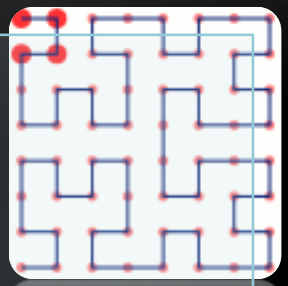
```
BFS(node)
{
  queue ← node
  visited[node] = true
  while queue not empty
    v ← queue
    print v
    for each child c of v
      if not visited[c]
        queue ← c
        visited[c] = true
}
```

```
DFS(node)
{
  stack ← node
  visited[node] = true
  while stack not empty
    v ← stack
    print v
    for each child c of v
      if not visited[c]
        stack ← c
        visited[c] = true
}
```

Recursive DFS Graph Traversal

```
void TraverseDFSRecursive(node)
{
  if (not visited[node])
  {
    visited[node] = true;
    print node;
    foreach child node c of node
    {
       TraverseDFSRecursive(c);
    }
  }
}

vois Main()
{
  TraverseDFS(firstNode);
}
```



Telerik Academy

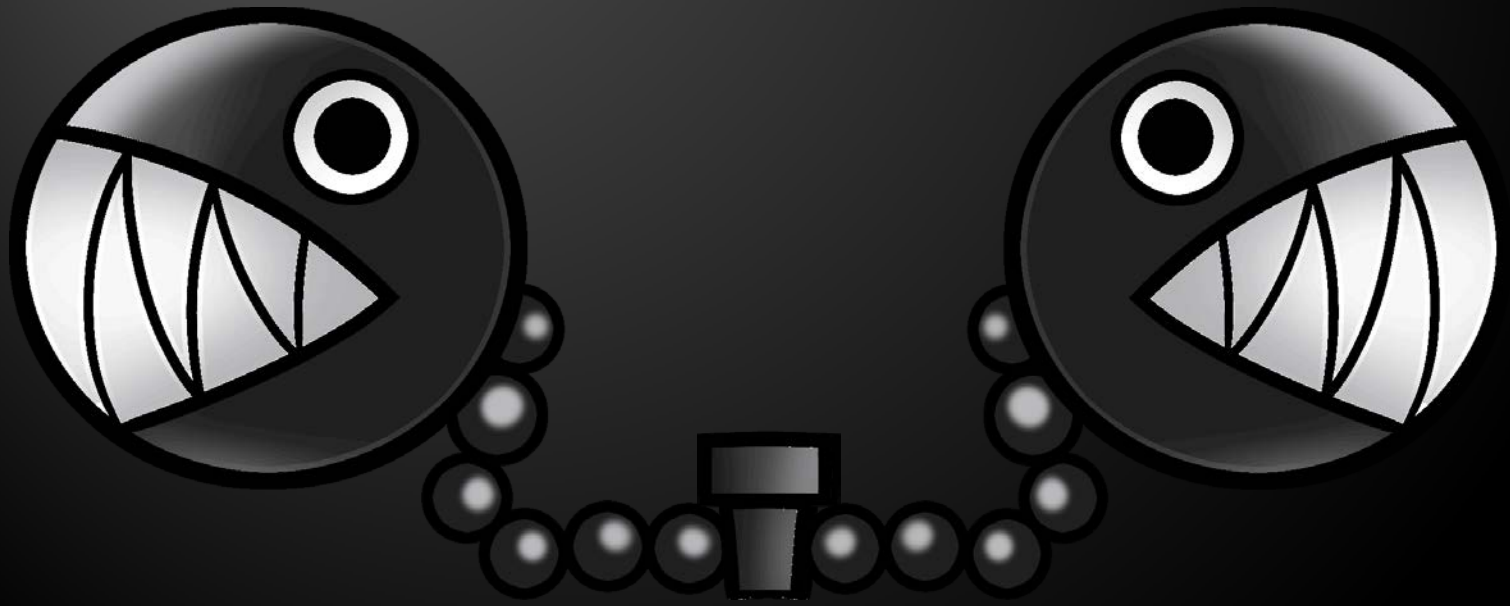
Graphs and Traversals

Live Demo

Telerik Academy

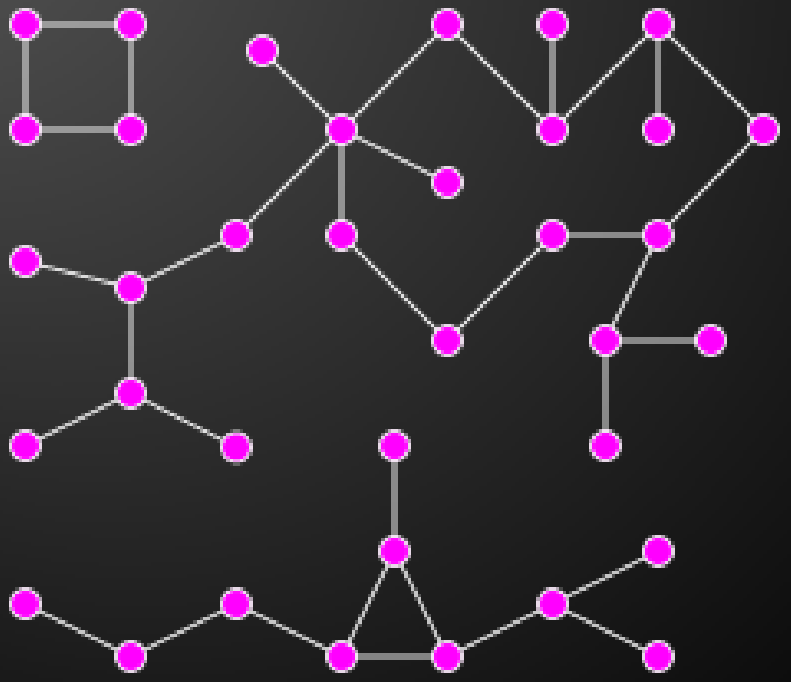
Connectivity

Connecting the chain



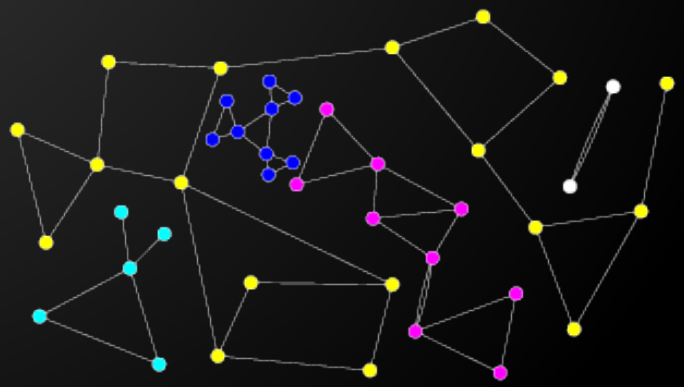
Connectivity

- Connected component of undirected graph
  - A sub-graph in which any two nodes are connected to each other by paths



Connectivity (2)

- A simple way to find number of connected components
  - A loop through all nodes and start a DFS or BFS traversing from any unvisited node
- Each time you start a new traversing
  - You find a new connected component!

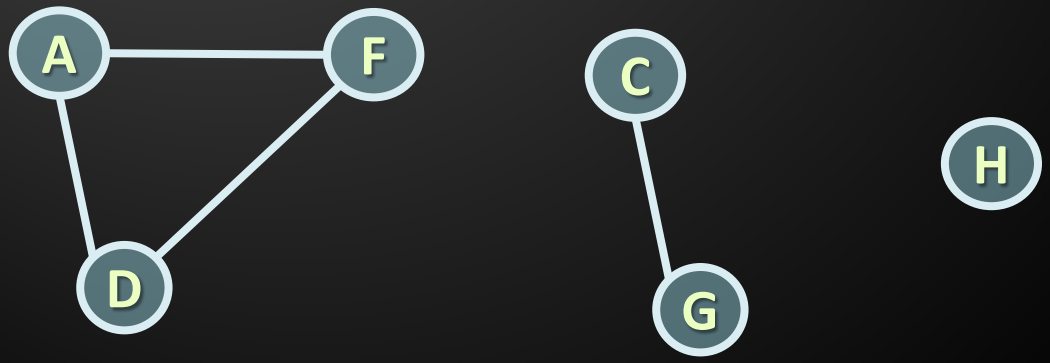


Connectivity (3)

- Algorithm:

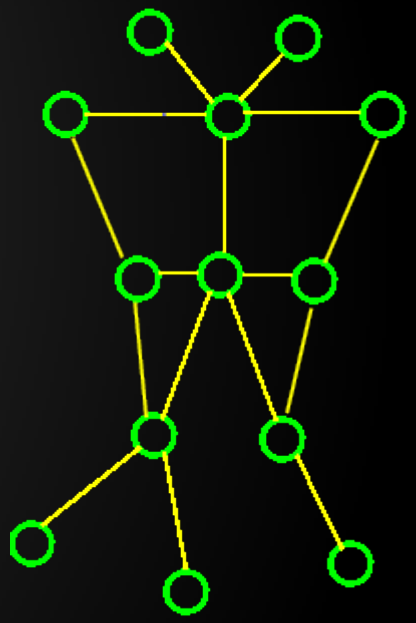
```
foreach node from graph G
{
    if node is unvisited
    {
        DFS(node);
        countOfComponents++;
    }
}
```

*Note: Do not forget to mark each node in the DFS as visited!



Connectivity (4)

- Connected graph
  - A graph with only one connected component
- In every connected graph a path exists between any two nodes
- Checking whether a graph is connected
  - If DFS / BFS passes through all vertices → graph is connected!



Telerik Academy

Connectivity

Live Demo

Telerik Academy

Dijkstra's Algorithm

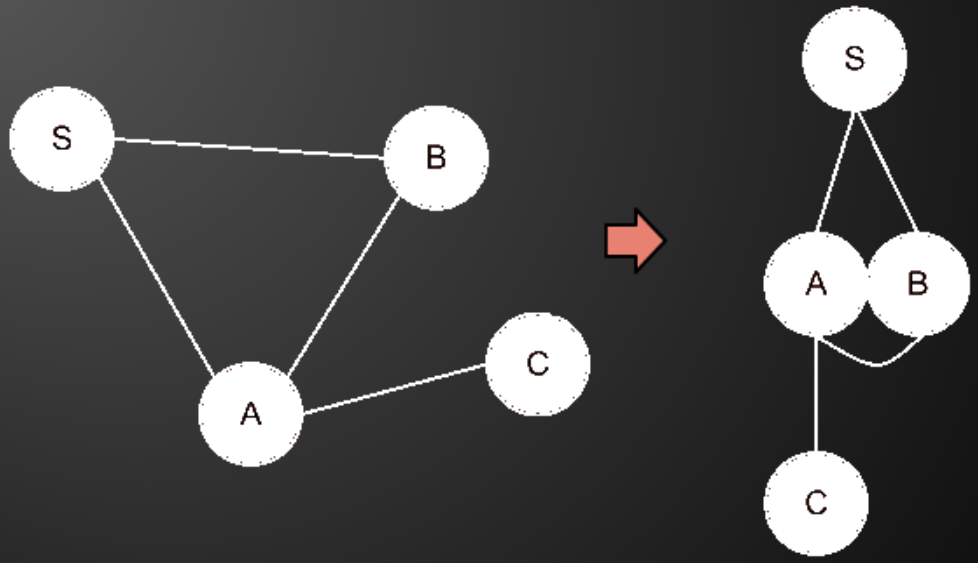
Shortest path in graph

Dijkstra's Algorithm

- Find the *shortest path* from vertex A to vertex B - a directed path between them such that no other path has a lower weight.
- Assumptions
  - Edges can be directed or not
  - Weight does not have to be distance
  - Weights are positive or zero
  - Shortest path is not necessary unique
  - Not all edges need to be reachable

Dijkstra's Algorithm (2)

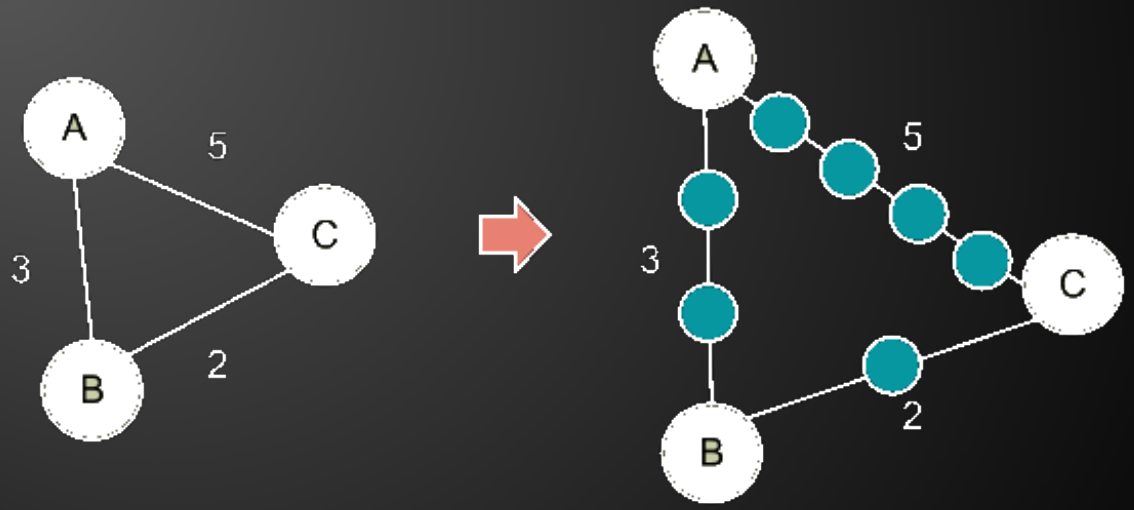
- In non-weighted graphs or edges with same weight finding shortest path can be done with BFS



*Note: Path from A to B does not matter – triangle inequality

Dijkstra's Algorithm (3)

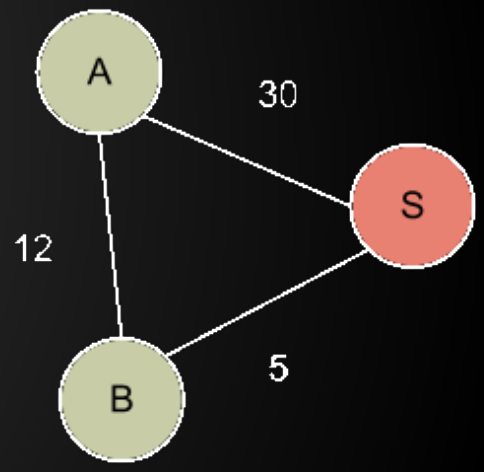
- In weighted graphs – simple solution can be done with breaking the edges in sub-vertexes



*Too much memory usage even for smaller graphs!

Dijkstra's Algorithm (4)

- Solution to this problem – priority queue instead of queue + keeping information about the shortest distance so far
- Steps:
  - Enqueue all distances from S
  - Get the lowest in priority - B
  - If edge B-A exists, check (S-B) + (B-A) and save the lower one
  - Overcome the triangle inequality miss



Dijkstra's Algorithm (5)

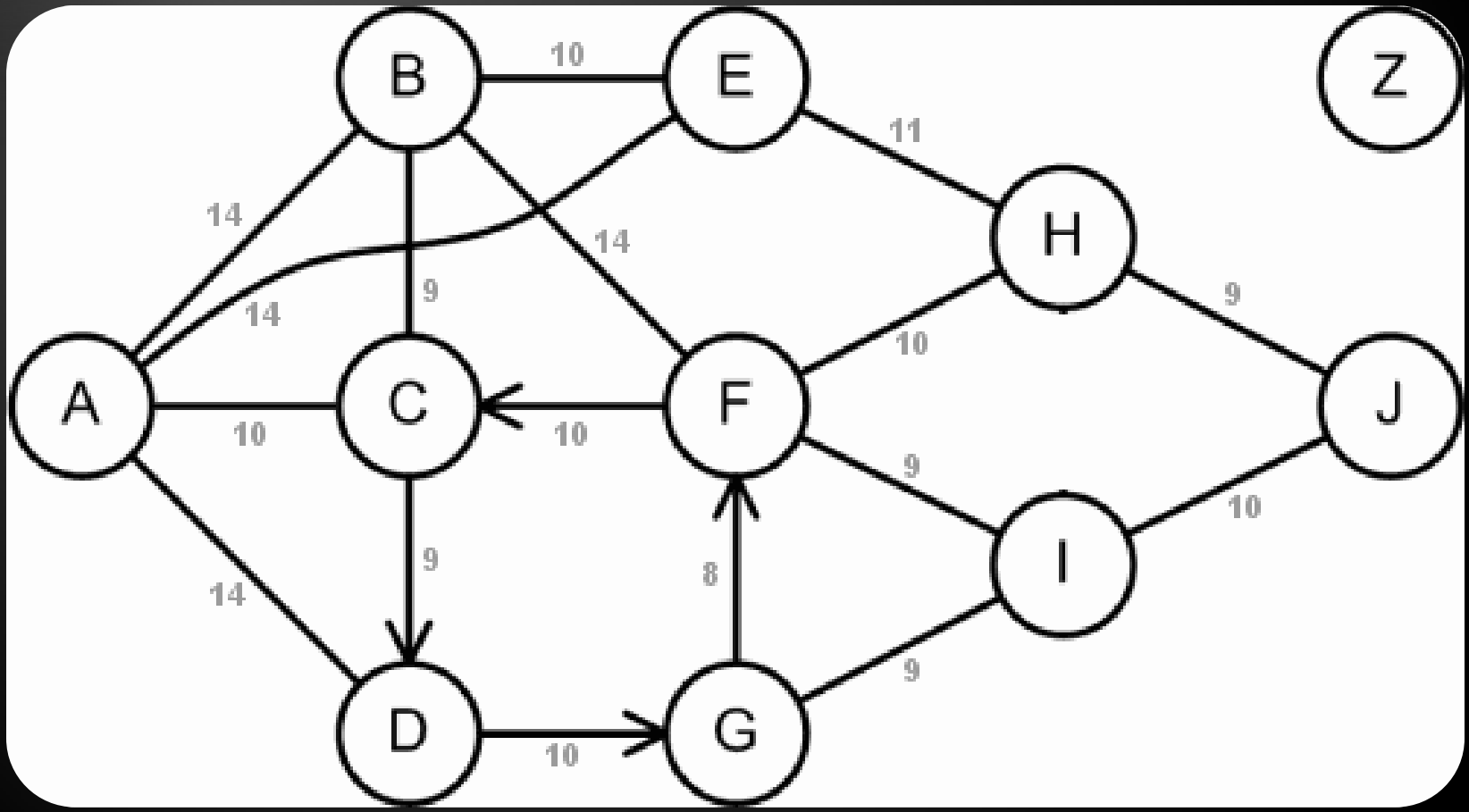
- Dijkstra's algorithm:
  1. Set the distance to every node to Infinity except the source node – must be zero
  2. Mark every node as unprocessed
  3. Choose the first unprocessed node with smallest non-infinity distance as current. If such does not exist, the algorithm has finished
  4. At first we set the current node our Source

Dijkstra's Algorithm (6)

- Dijkstra's algorithm:
  5. Calculate the distance for all unprocessed neighbors by adding the current distance to the already calculated one
  6. If the new distance is smaller than the previous one – set the new value
  7. Mark the current node as processed
  8. Repeat step 3.

Dijkstra's Algorithm (7)

- Example graph:



Dijkstra's Algorithm (8)

- Pseudo code

```
set all nodes DIST = INFINITY;
set current node the source and distance = 0;
Q -> all nodes from graph, ordered by distance;
while (Q is not empty)
{
  a = dequeue the smallest element (first in PriorityQueue);
     if (distance of a == INFINITY) break;
     foreach neighbour v of a
        {
            potDistance = distance of a + distance of (a-v);
            if (potDistance < distance of v)
            {
                distance of v = potDistance;
                reorder Q;
            }
        }
}
```

Dijkstra's Algorithm (9)

- Modifications
  - Saving the route
  - Having a target node
  - Array implementation, Queue, Priority Queue
  - A*
- Complexity
  - O((|V| + |E|).log(|V|))
- Applications –GPS, Networks, Air travels, etc.

Telerik Academy

Dijkstra's Algorithm

Live Demo

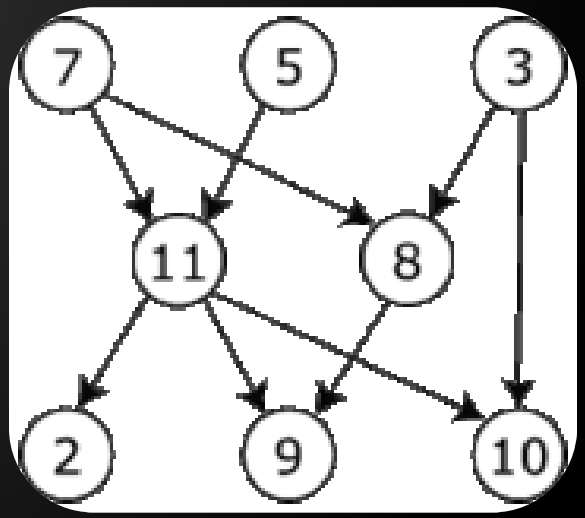
Telerik Academy

Topological Sorting

Order it!

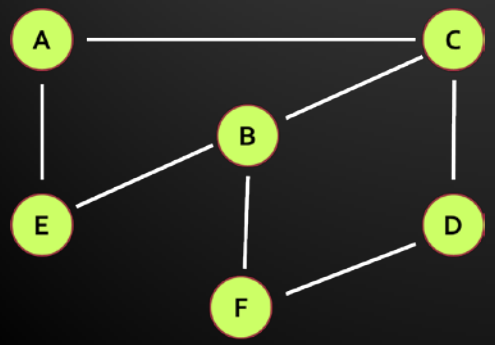
Topological Sorting

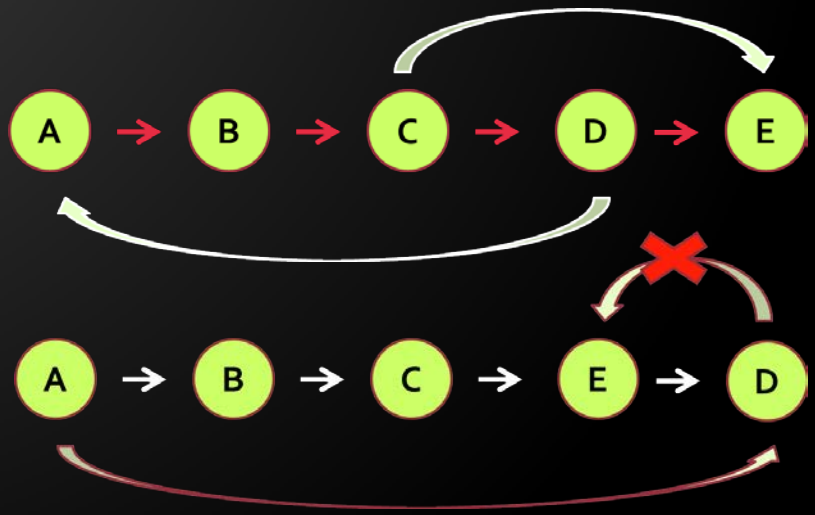
- Topological ordering of a directed graph
  - linear ordering of its vertices
  - for every directed edge from vertex u to vertex v, u comes before v in the ordering
- Example:
  - 7, 5, 3, 11, 8, 2, 9, 10
  - 3, 5, 7, 8, 11, 2, 9, 10
  - 5, 7, 3, 8, 11, 10, 9, 2



Topological Sorting (2)

- Rules
  - Undirected graph cannot be sorted
  - Directed graphs with cycles cannot be sorted
  - Sorting is not unique
  - Various sorting algorithms exists and they give different results





Topological Sorting (3)

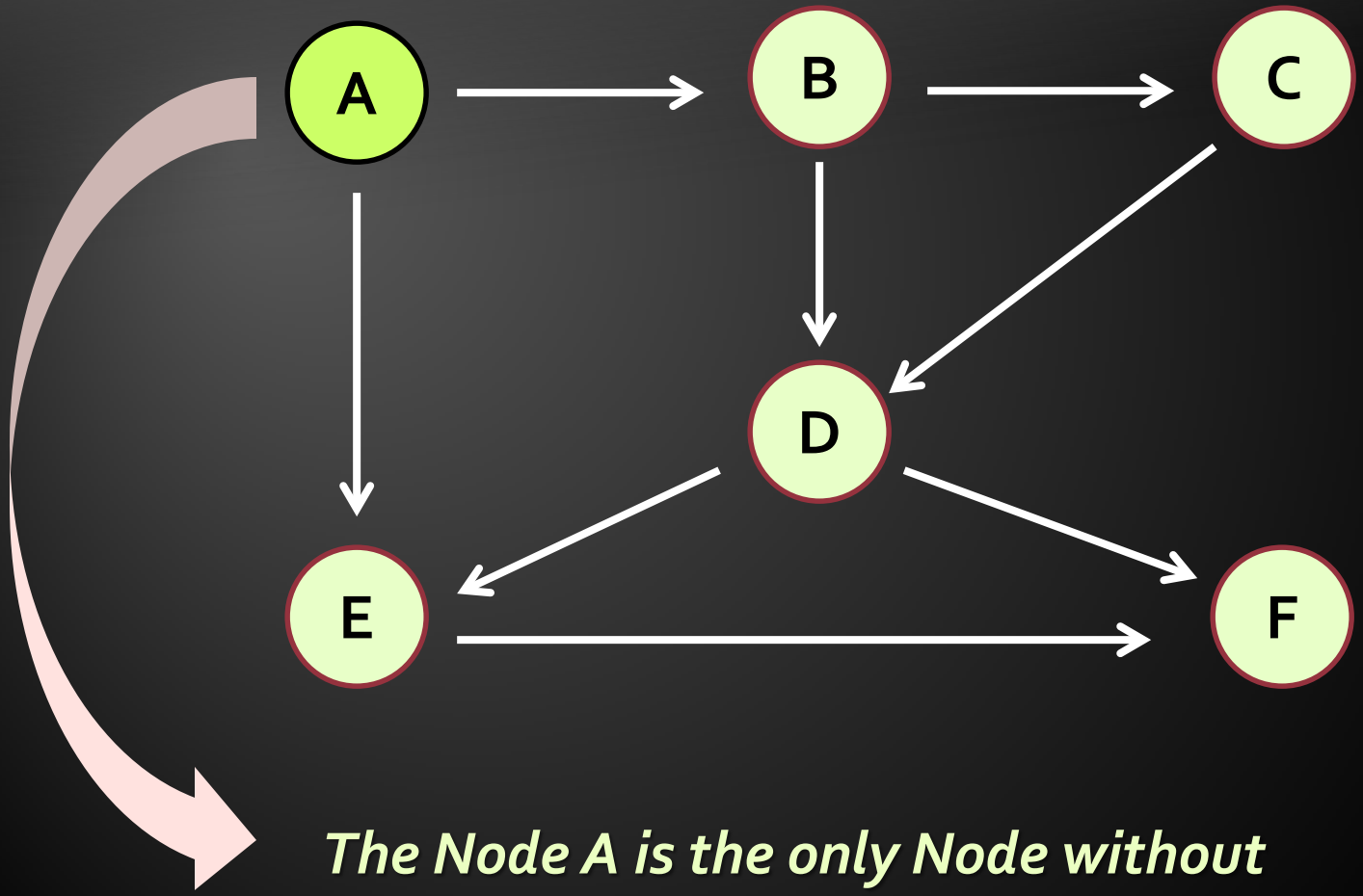
- Source removal algorithm
  - Create an Empty List
  - Find a Node without incoming Edges
  - Add this Node to the end of the List
  - Remove the Edge from the Graph
  - Repeat until the Graph is empty

Topological Sorting (4)

- Pseudo code

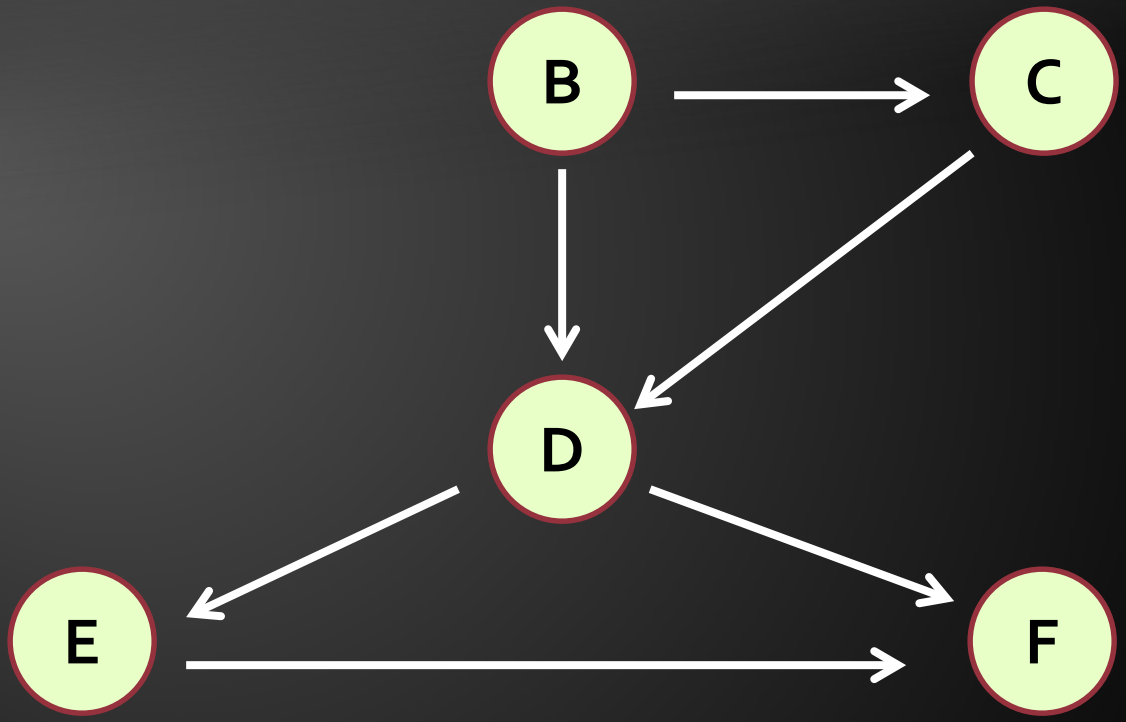
```
L ← Empty list that will contain the sorted elements
S ← Set of all nodes with no incoming edges
while S is non-empty do
    remove a node n from S
    insert n into L
    for each node m with an edge e from n to m do
        remove edge e from the graph
        if m has no other incoming edges then
            insert m into S
if graph has edges then
    return error (graph has at least one cycle)
else
    return L (a topologically sorted order)
```

Step #1-2



The Node A is the only Node without Incoming Edges

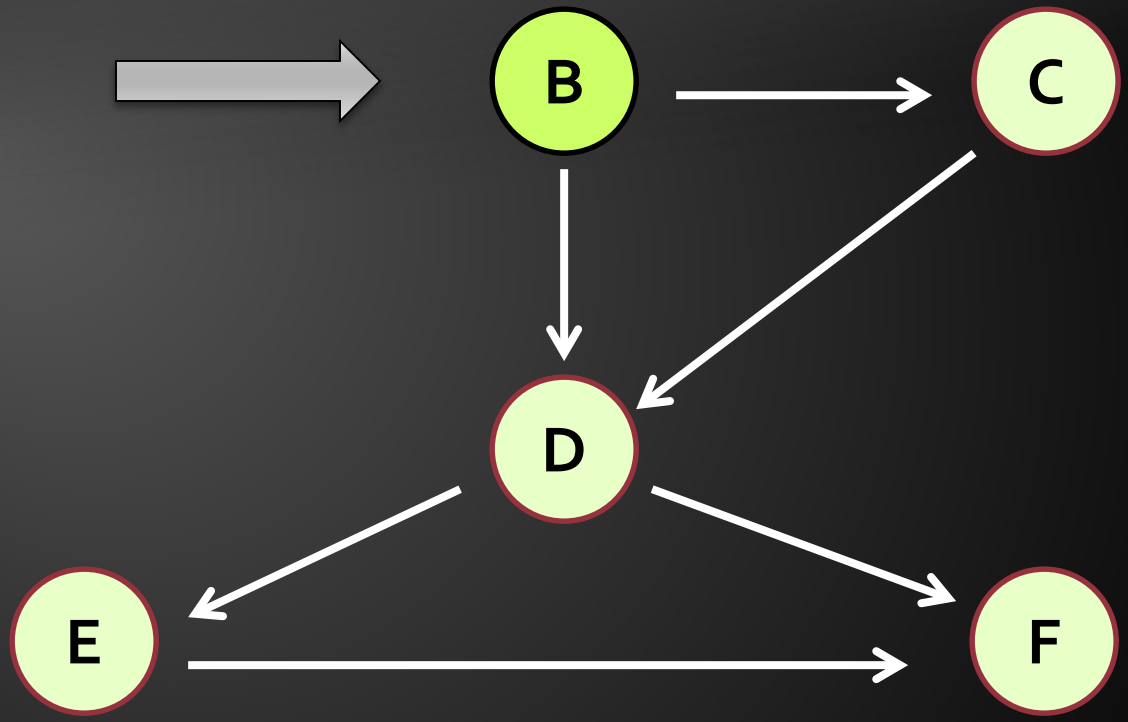
Step #3-4



L

A

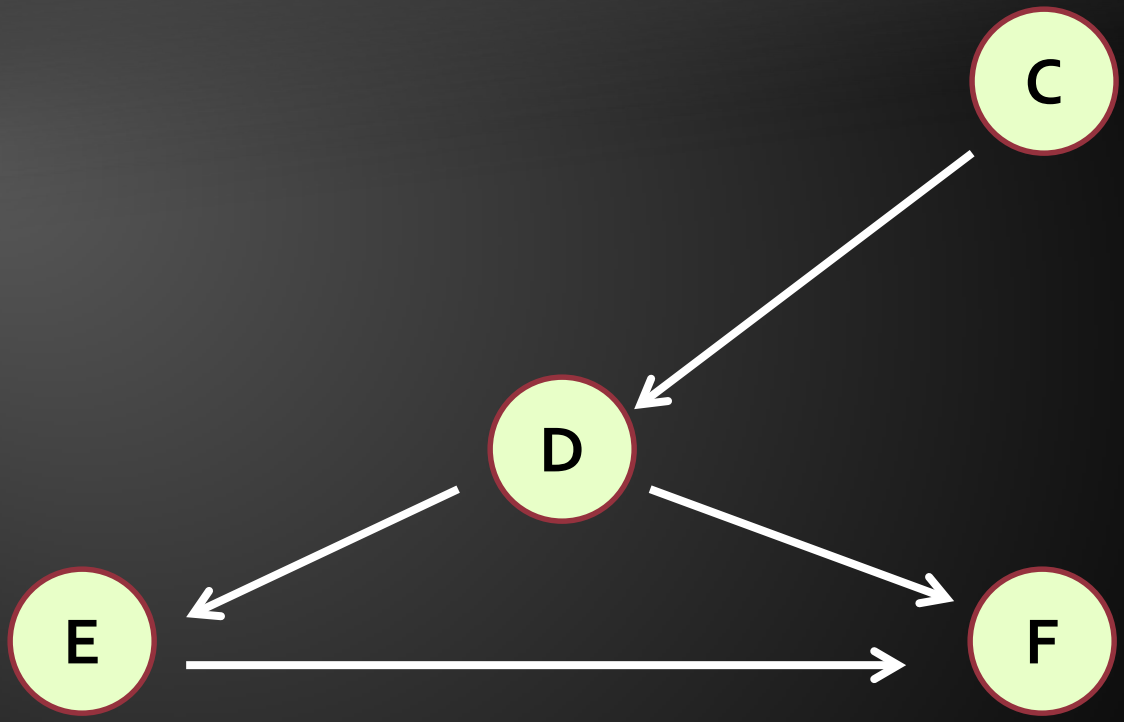
Step #2-3



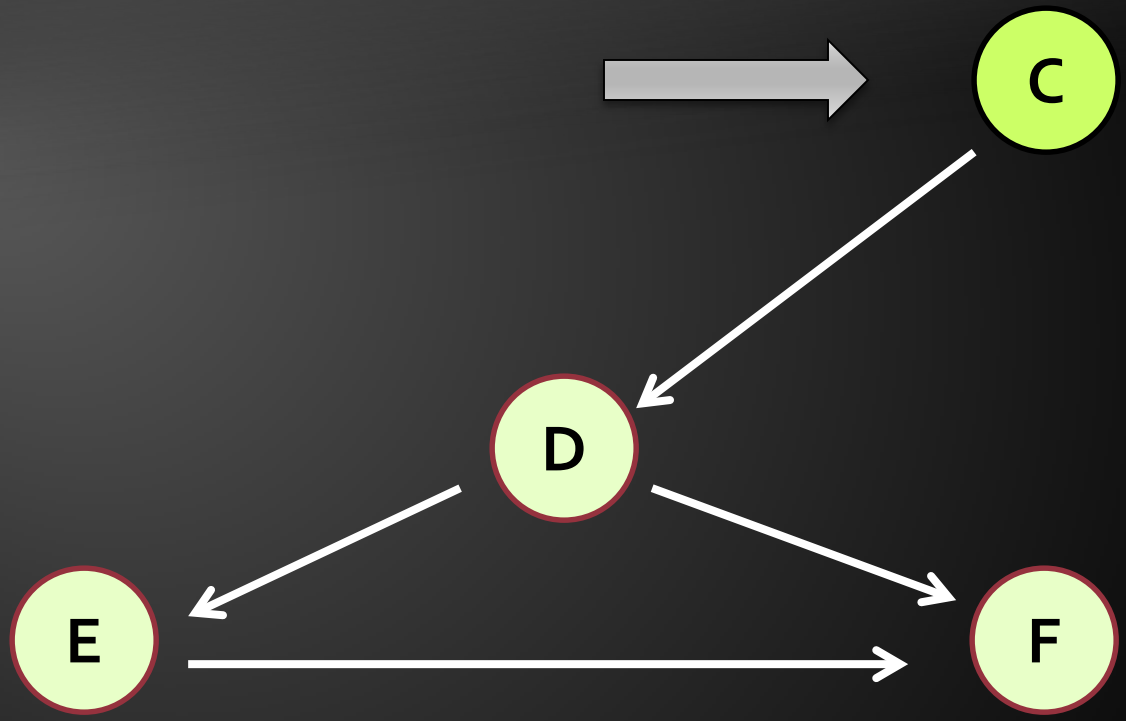
L

A

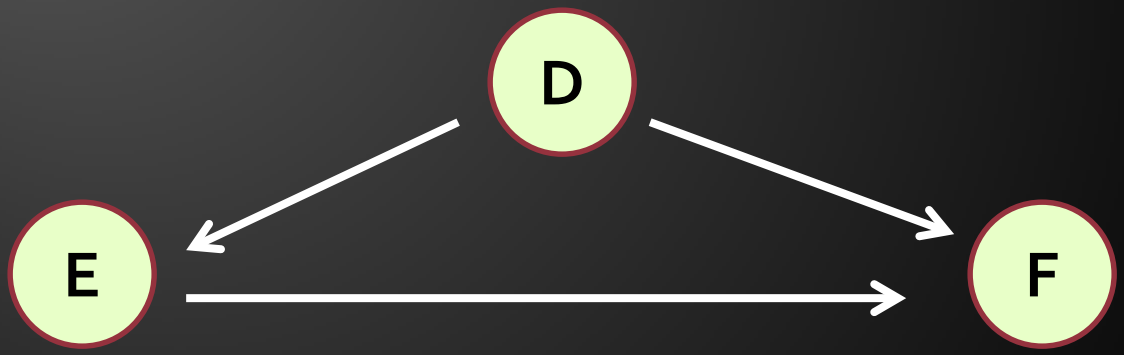
Step #2-3

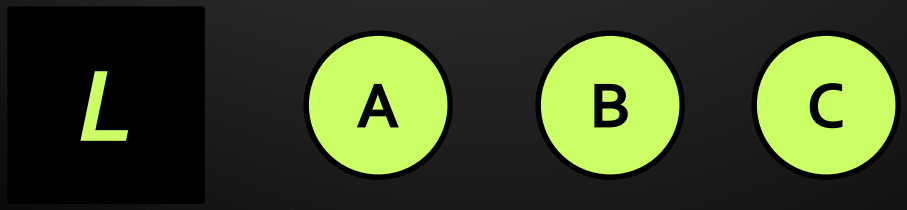


Step #2-3

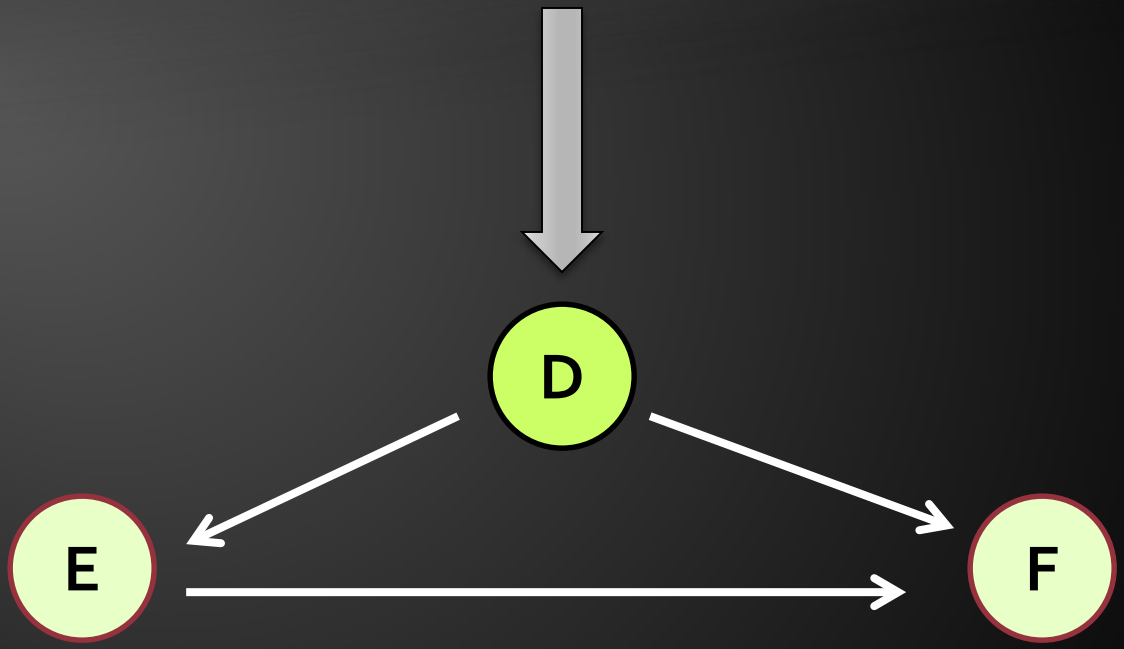


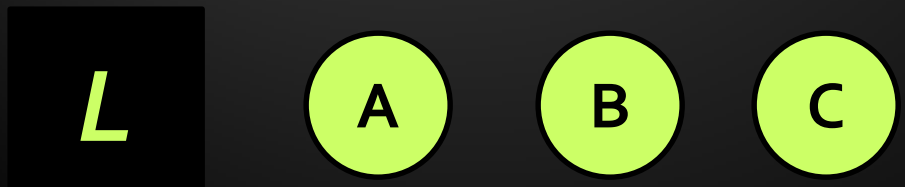
Step #2-3



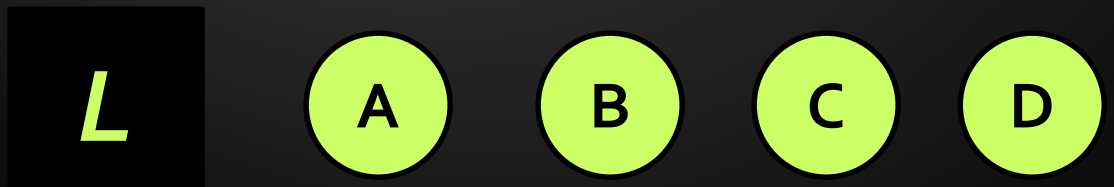


Step #2-3

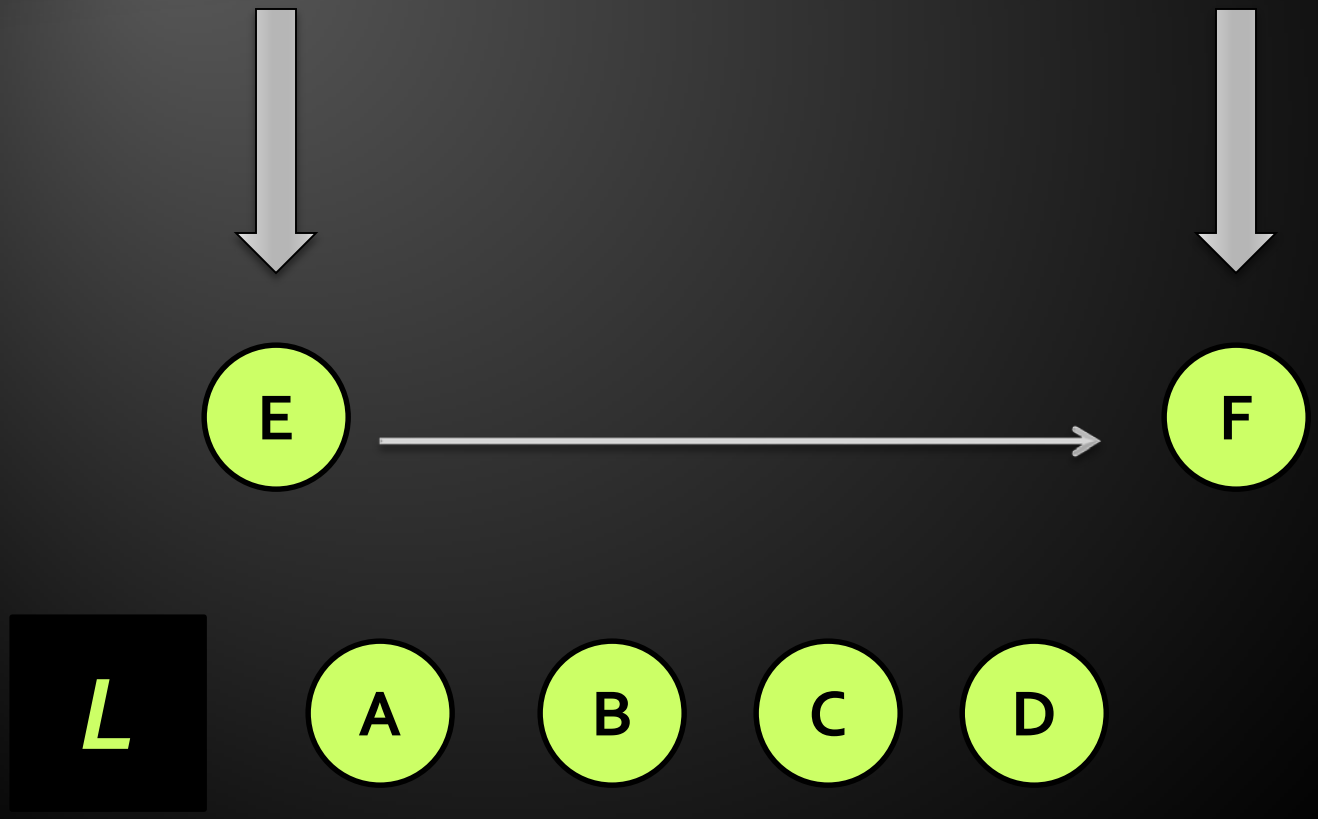




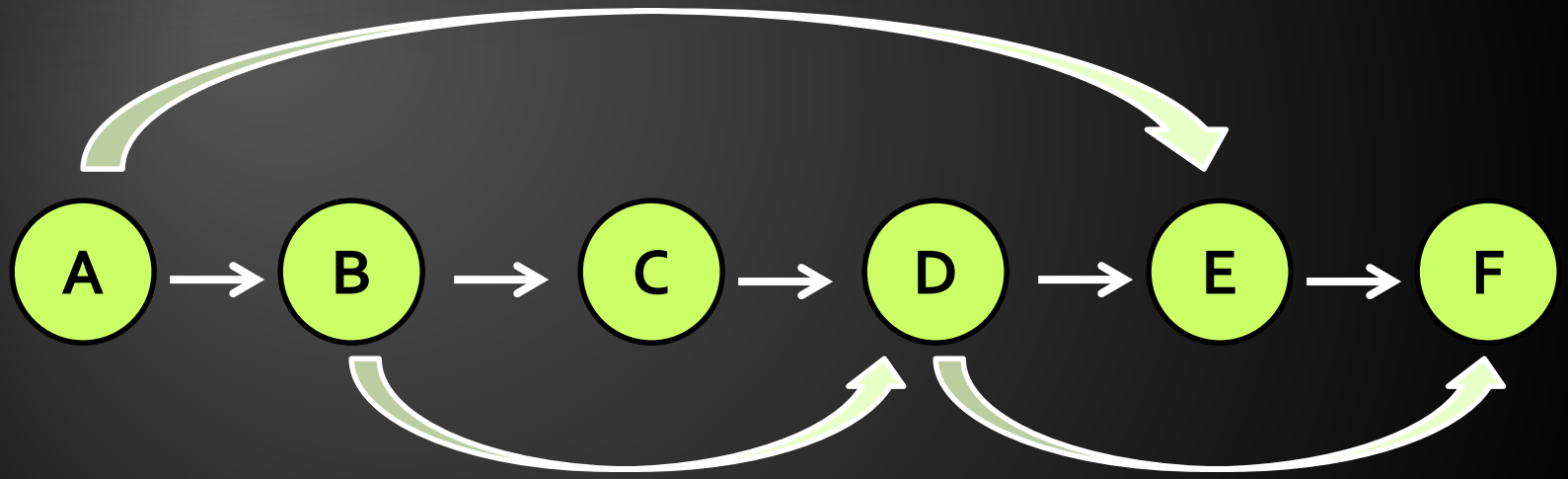
Step #2-3



Step #2-3



Result:TS



Topological Sorting (5)

- DFS algorithm
  - Create an empty List
  - Find a Node without Outgoing Edges
  - Mark the Node as visited
  - Add the node to the List
  - Stop when reach visited node
  - Reverse the List and get the TS of the Elements

Topological Sorting (6)

- Pseudo code

```
L ← Empty list that will contain the sorted nodes
while there are unmarked nodes do
    select an unmarked node n
    visit(n)
function visit(node n)
    if n has a temporary mark then stop (not a DAG)
    if n is not marked (i.e. has not been visited yet) then
        mark n temporarily
        for each node m with an edge from n to m do
            visit(m)
        mark n permanently
        add n to head of L
```

- http://www.geeksforgeeks.org/topological-sorting/

Telerik Academy

TS Using DFS

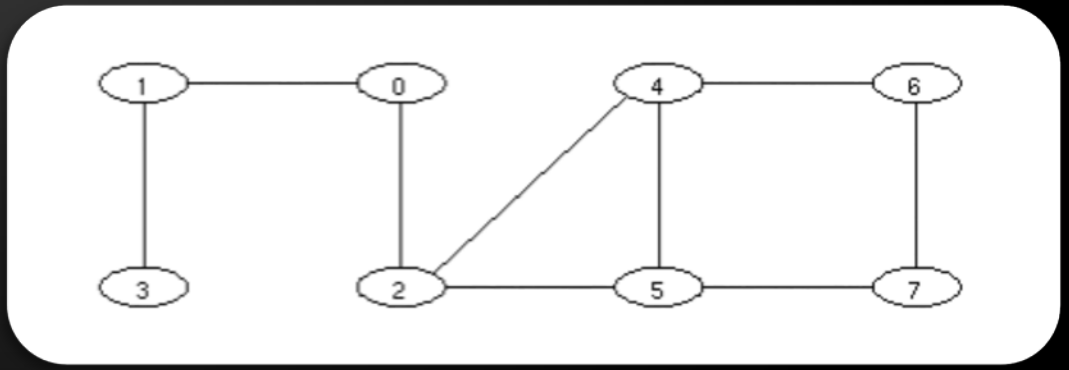
Live Demo

Telerik Academy

Minimum Spanning Tree

Minimum Spanning Tree

- Spanning Tree
  - Subgraph (Tree)
  - Connects all vertices together
- All connected graphs have spanning tree

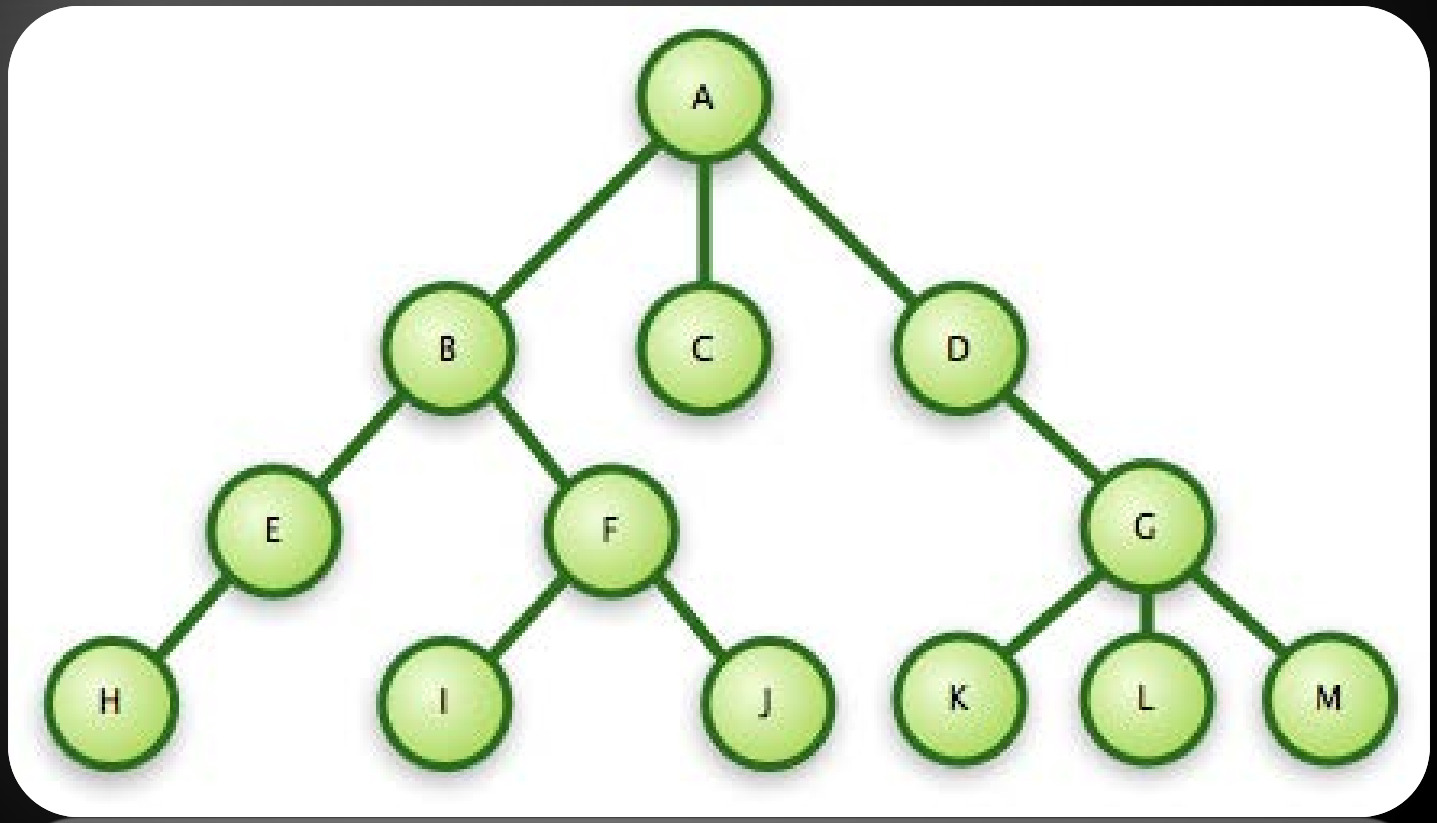


Minimum Spanning Tree

- Minimum Spanning Tree
  - weight <=

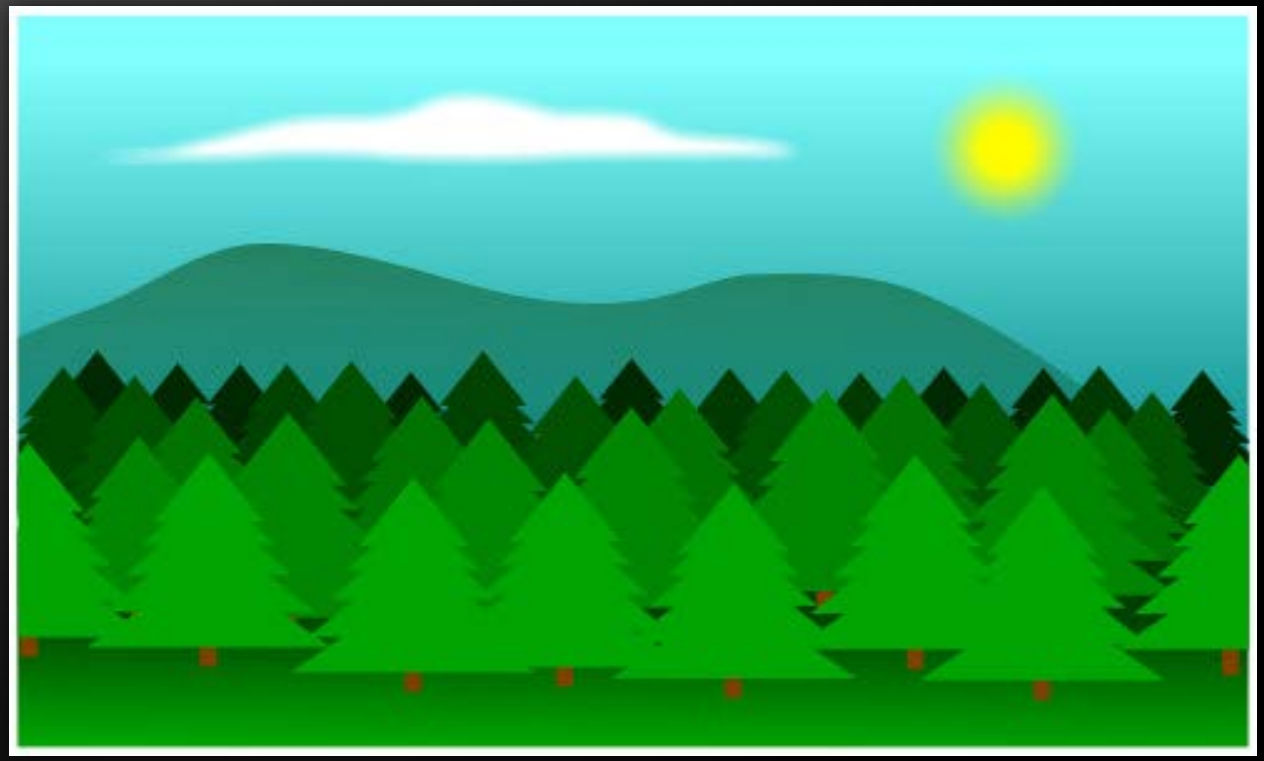
    weight(all other spanning trees)
- First used in electrical network
  - Minimal cost of wiring

Minimum Spanning Forest

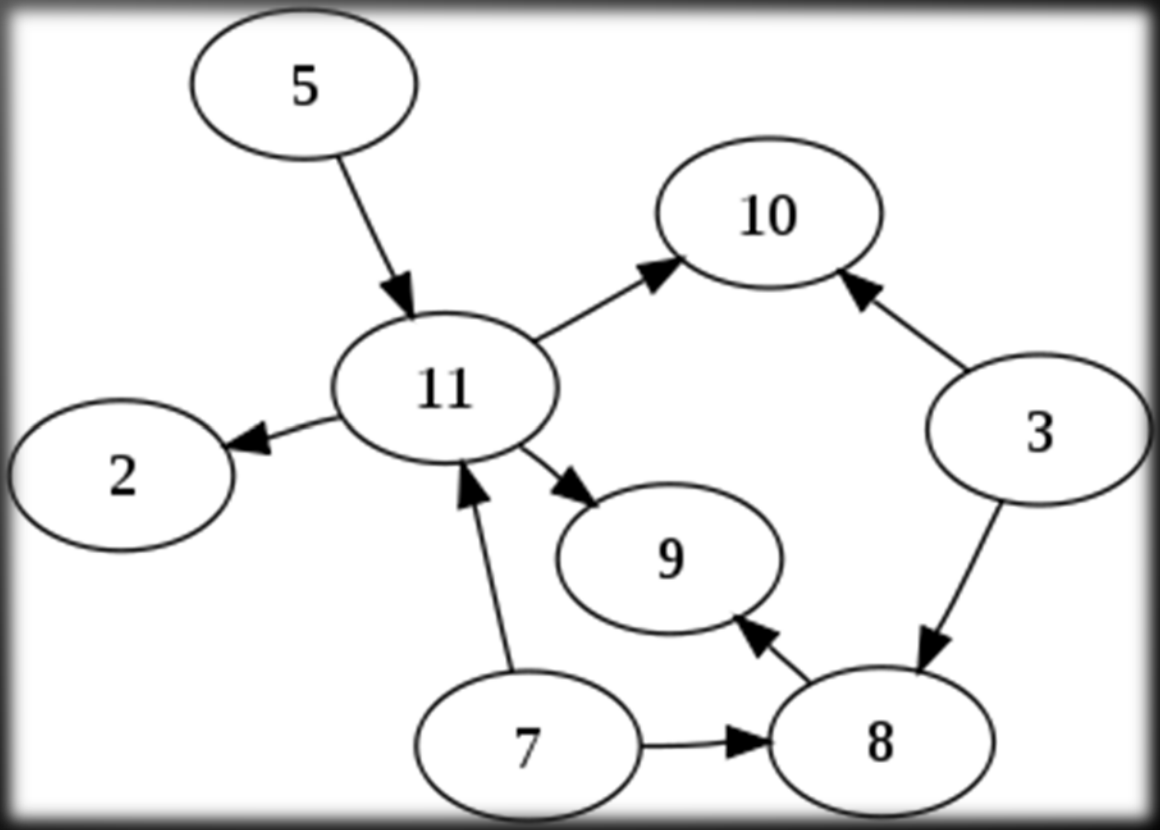


Minimum Spanning Forest

- Minimum Spanning Forest – set of all minimum spanning trees (when the graph is not connected)



Prim's Algorithm

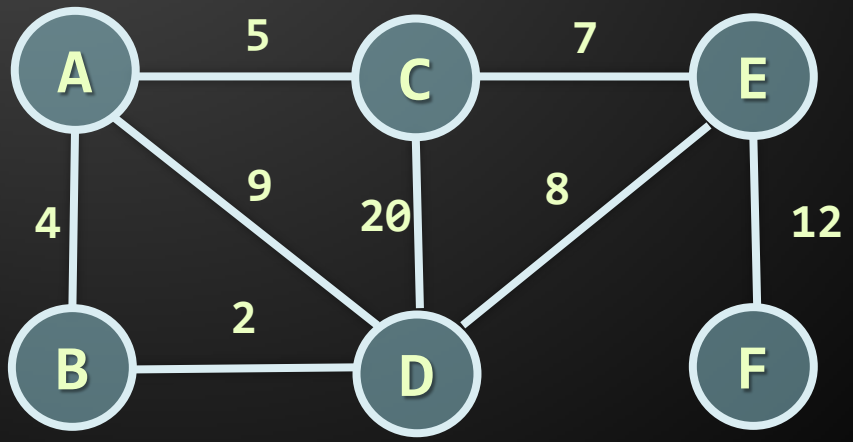


Prim's Algorithm

- Create a tree T containing a single vertex (chosen randomly)
- Create a set S from all the edges in the graph
- Loop until every edge in the set connects two vertices in the tree
  - Remove from the set an edge with minimum weight that connects a vertex in the tree with a vertex not in the tree
  - Add that edge to the tree
- Note: the graph must be connected

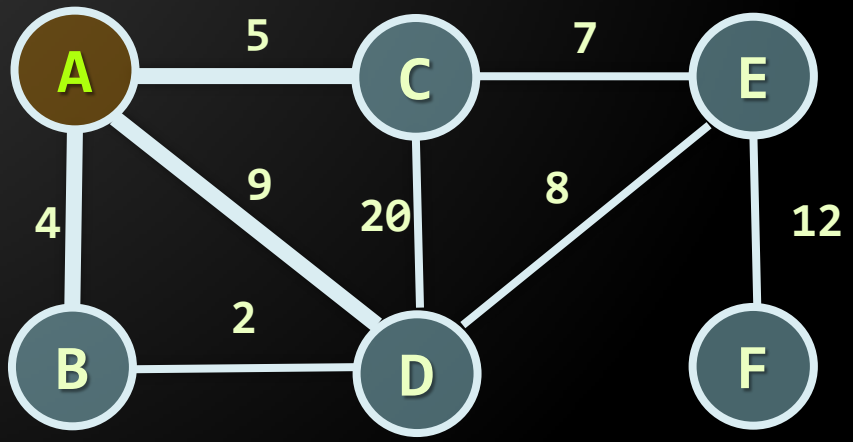
Prim's Algorithm

- Note: at every step before adding an edge to the tree we check if it makes a cycle in the tree or if it is already in the queue
- When we add a vertex we check if it is the last which is not visited



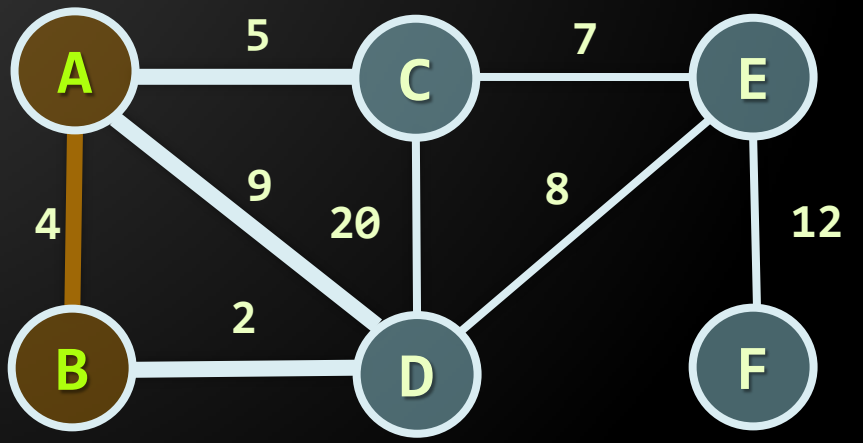
Prim's Algorithm

- We build a tree with the single vertex A
- Priority queue which contains all edges that connect A with the other nodes (AB, AC, AD)



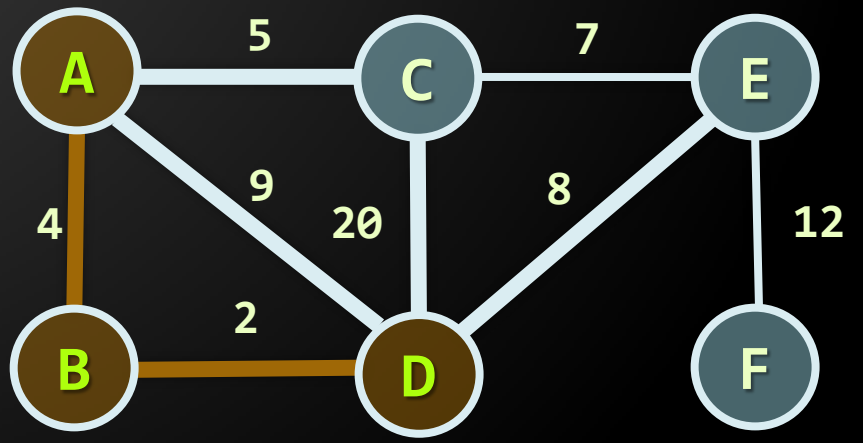
Prim's Algorithm

- The tree still contains the only vertex A
- We dequeue the first edge from the priority queue (4) and we add the edge and the other vertex (B) form that edge to the tree
- We push all edges that connect B with other nodes in the queue
  - Note that the edges 5 and 9 are still in the queue



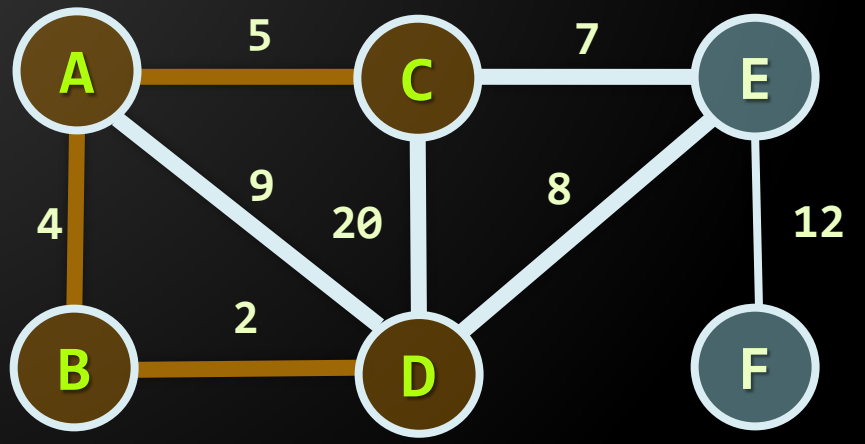
Prim's Algorithm

- Now the tree contains vertices A and B and the edge between them
- We dequeue the first edge from the priority queue (2) and we add the edge and the other vertex (D) from that edge to the tree
- We push all edges that connect D with other nodes in the queue



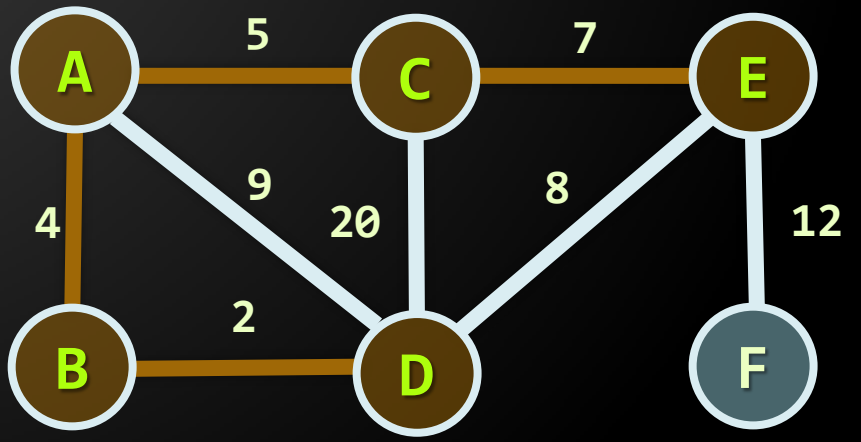
Prim's Algorithm

- Now the tree contains vertices A, B and D and the edges (4, 2) between them
- We dequeue the first edge from the priority queue (5) and we add the edge and the other vertex (C) from that edge to the tree
- We push all edges that connect C with other nodes in the queue



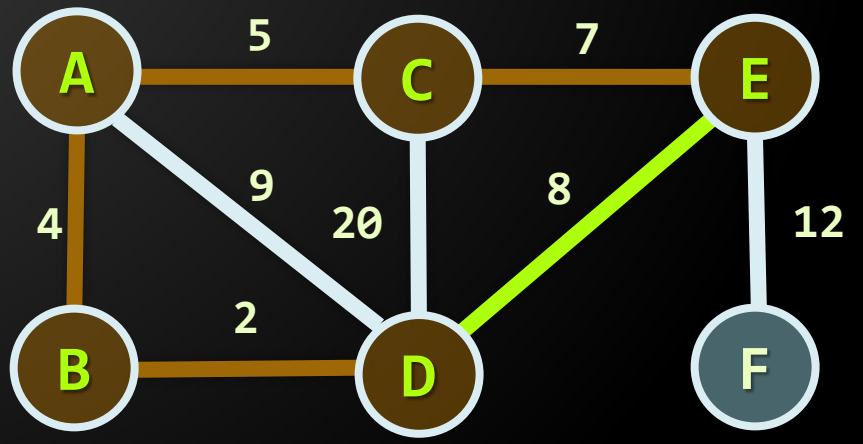
Prim's Algorithm

- Now the tree contains vertices A, B, D and C and the edges (4, 2, 5) between them
- We dequeue the first edge from the priority queue (7) and we add the edge and the other vertex (E) from that edge to the tree
- We push all edges that connect C with other nodes in the queue



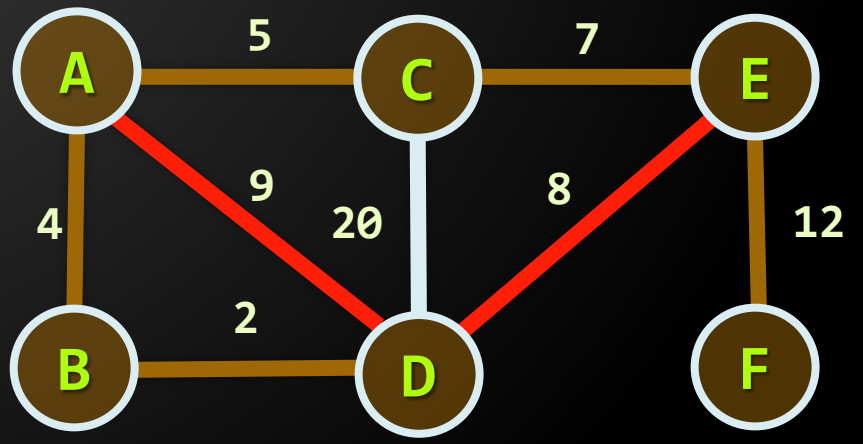
Prim's Algorithm

- Now the tree contains vertices A, B, D, C and E and the edges (4, 2, 5, 7) between them
- We dequeue the first edge from the priority queue (8)



Prim's Algorithm

- This edge will cost a cycle
- So we get the next one – 9
  - This edge will also cost a cycle
- So we get the next one – 12
  - We add it to the tree
  - We add the vertex F to the tree



Prim

Live Demo

Telerik Academy

Kruskal's Algorithm

Kruskal's Algorithm

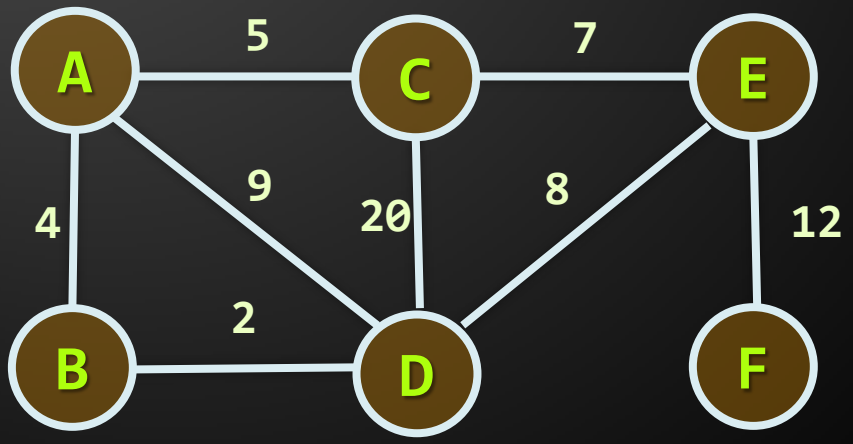
- The graph may not be connected
  - If the graph is not connected – minimum spanning forest

Kruskal's Algorithm

- Create forest F (each tree is a vertex)
- Set S – all edges in the graph
- While S if nonempty and F is not spanning
  - Remove edge with min cost from S
  - If that edge connects two different trees – add it to the forest (these two trees are now a single tree)
  - Else discard the edge
- The graph may not be connected

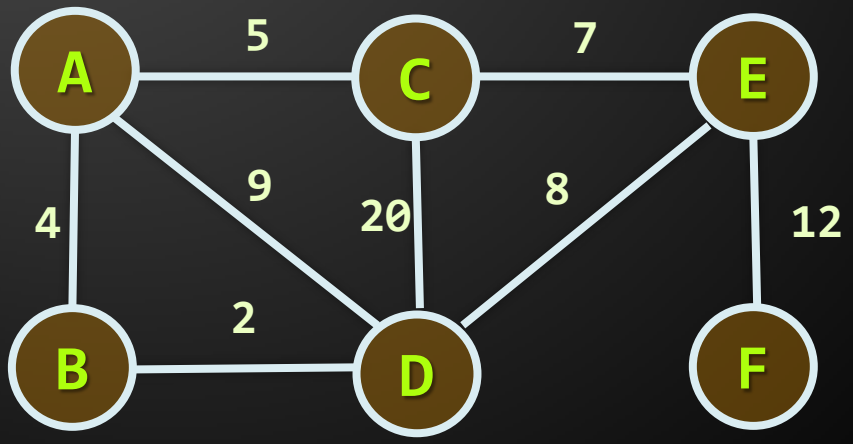
Kruskal's Algorithm

- We build a forest containing all vertices from the graph
- We sort all edges
- Edges are – 2, 4, 5, 7, 8, 9, 12, 20



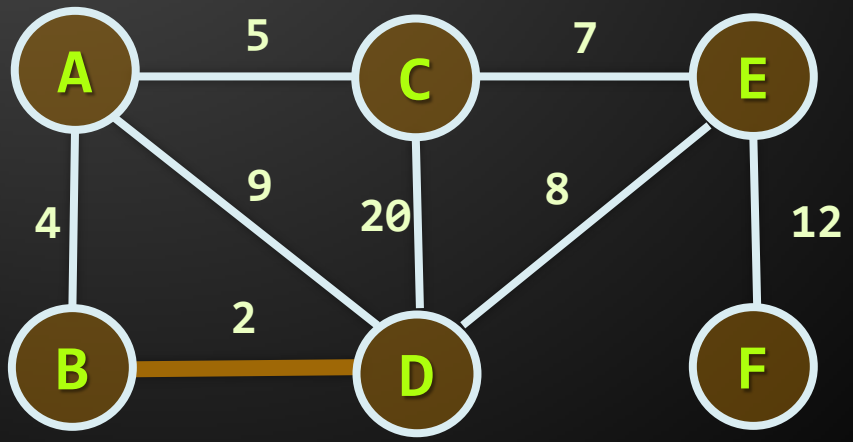
Kruskal's Algorithm

- At every step we select the edge with the smallest weight and remove it from the list with edges
- If it connects two different trees from the forest we add it and connect these trees



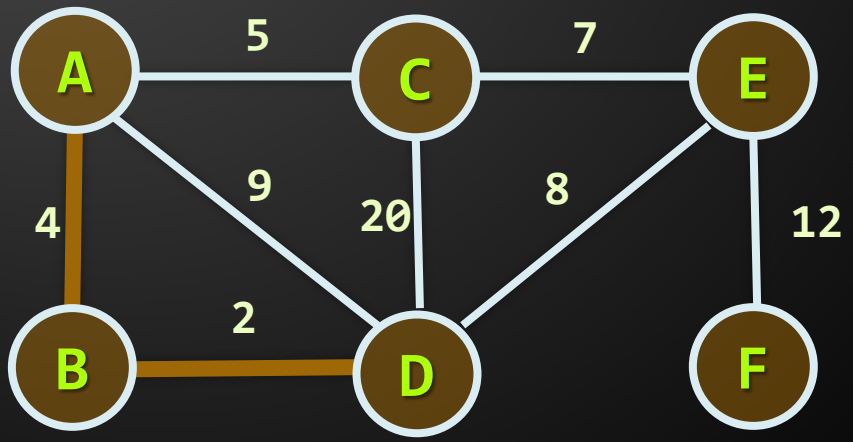
Kruskal's Algorithm

- We select the edge 2
- This edge connects the vertices B and D (they are in different trees)
- We add it



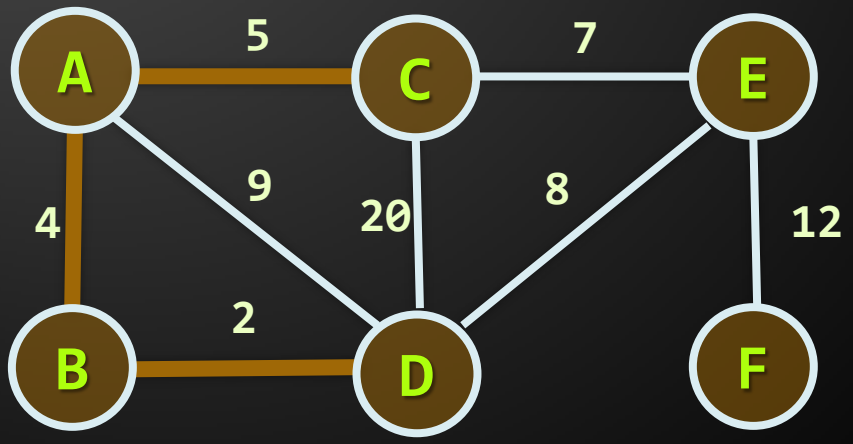
Kruskal's Algorithm

- We select the edge 4
- This edge connects the vertices A and B (they are different trees)
- We add it



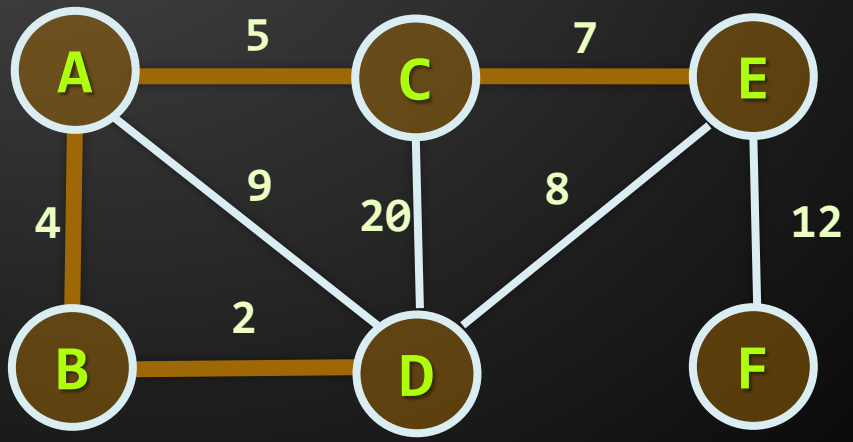
Kruskal's Algorithm

- We select the edge 5
- This edge connects the vertices A and C (they are different trees)
- We add it



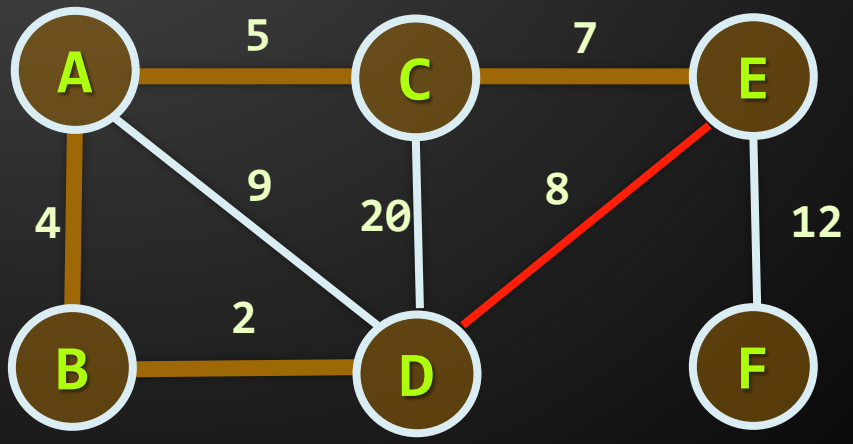
Kruskal's Algorithm

- We select the edge 7
- This edge connects the vertices C and E (they are different trees)
- We add it



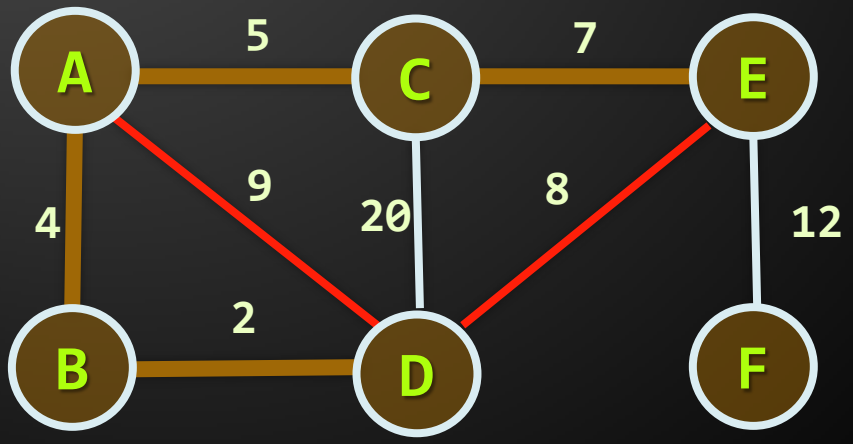
Kruskal's Algorithm

- We select the edge 8
- This edge connects the vertices E and D (they are not different trees)
- We don't add it



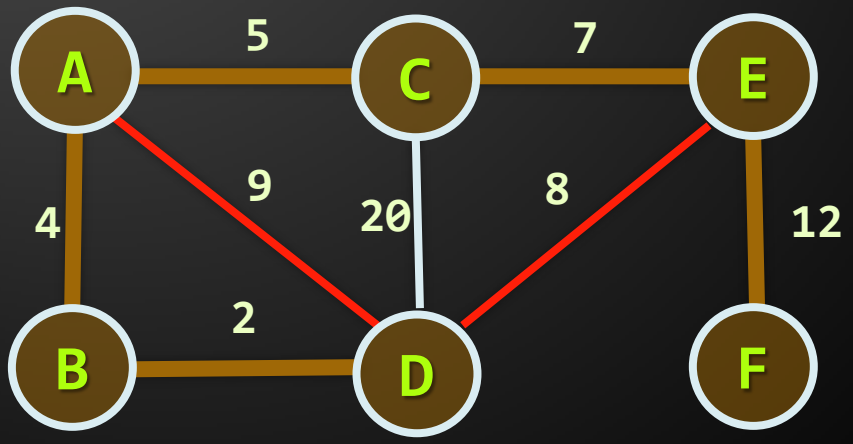
Kruskal's Algorithm

- We select the edge 9
- This edge connects the vertices A and D (they are not different trees)
- We don't add it



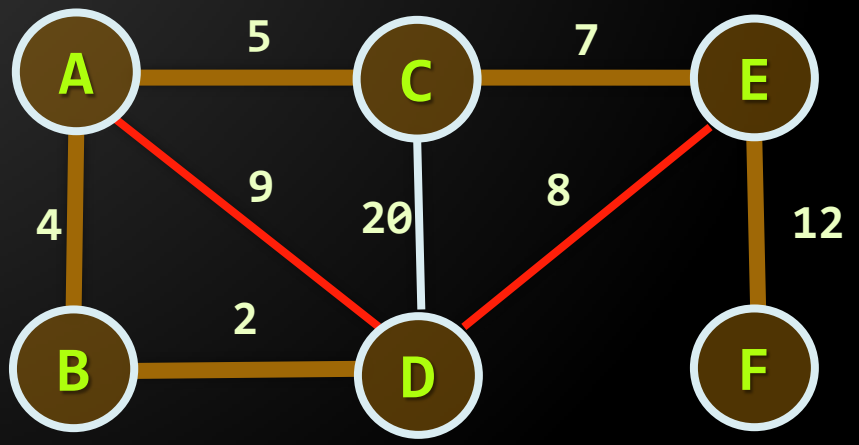
Kruskal's Algorithm

- We select the edge 12
- This edge connects the vertices E and F (they are not different trees)
- We add it



Kruskal's Algorithm

- We can have a function that checks at every step if all vertices are connected and the tree that we build is spanning
- If we have such function we stop
- Otherwise we check for the other edges
  - We just won't add them to the tree



Telerik Academy

Kruskal

Live Demo

Exercises

1. Solve these problems from BGCoder:
   1. Algo Academy March 2012 – Problem 05 – Friends of Pesho
   2. Algo Academy February 2013 – Problem 04 – Salaries
2. You are given a cable TV company. The company needs to lay cable to a new neighborhood (for every house). If it is constrained to bury the cable only along certain paths, then there would be a graph representing which points are connected by those paths. But the cost of some of the paths is more expensive because they are longer. If every house is a node and every path from house to house is an edge, find a way to minimize the cost for cables.

Trees and Graphs

Questions?