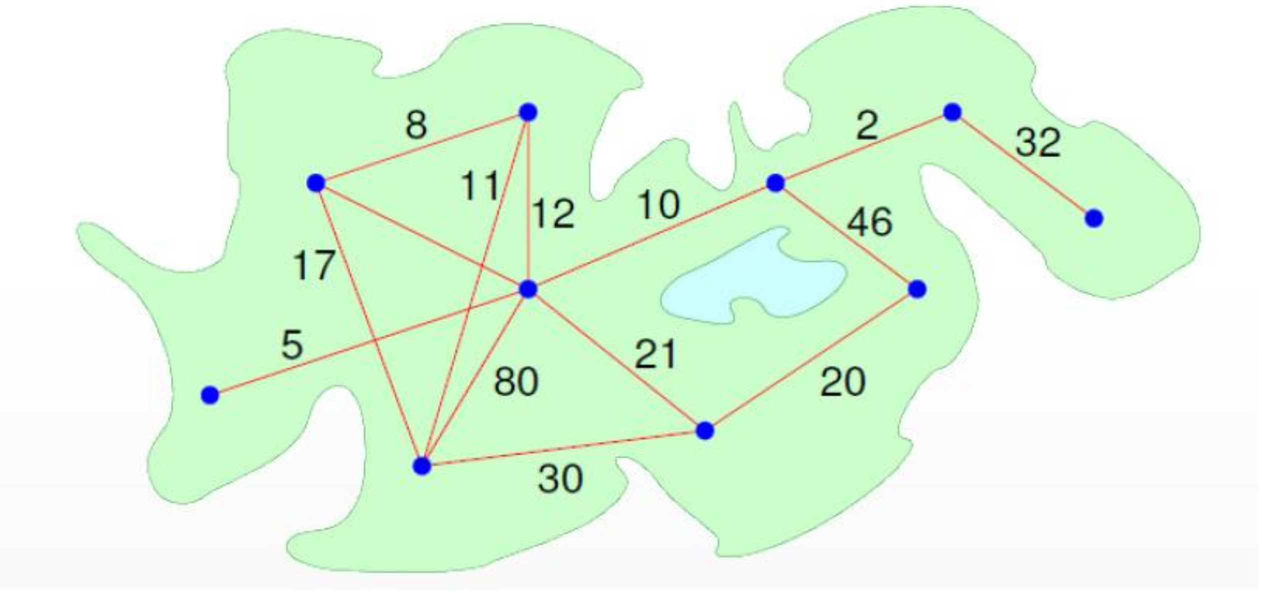
COSC 222 Data Structure

Graphs – Part 3

Spanning Tree

- A company has a contract to provide high-speed internet to an area.
- Each client must be connected to the network while minimizing the total cost of building the network.
- Your are provided cost estimates for various possible links in the network.

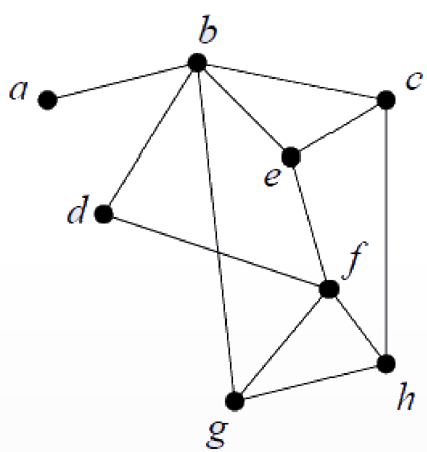


Spanning Tree

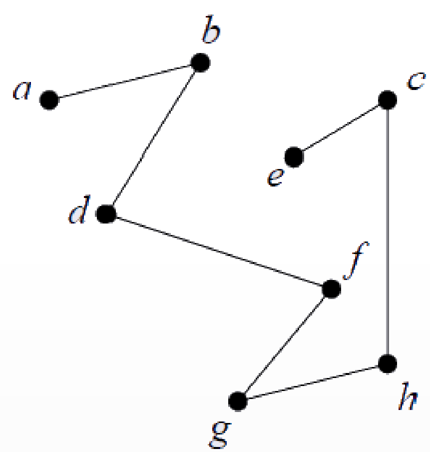
- A spanning tree for a graph G is a spanning subgraph of G that is a tree.
  - Every connected graph has a spanning tree.
  - Any spanning tree for a graph G = (V; E)

    has |V| vertices.

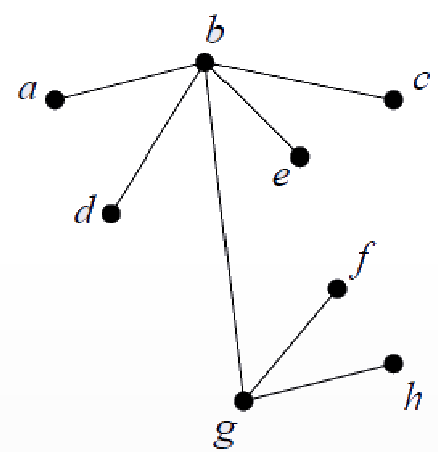
    has |V| - 1 edges.



Graph G



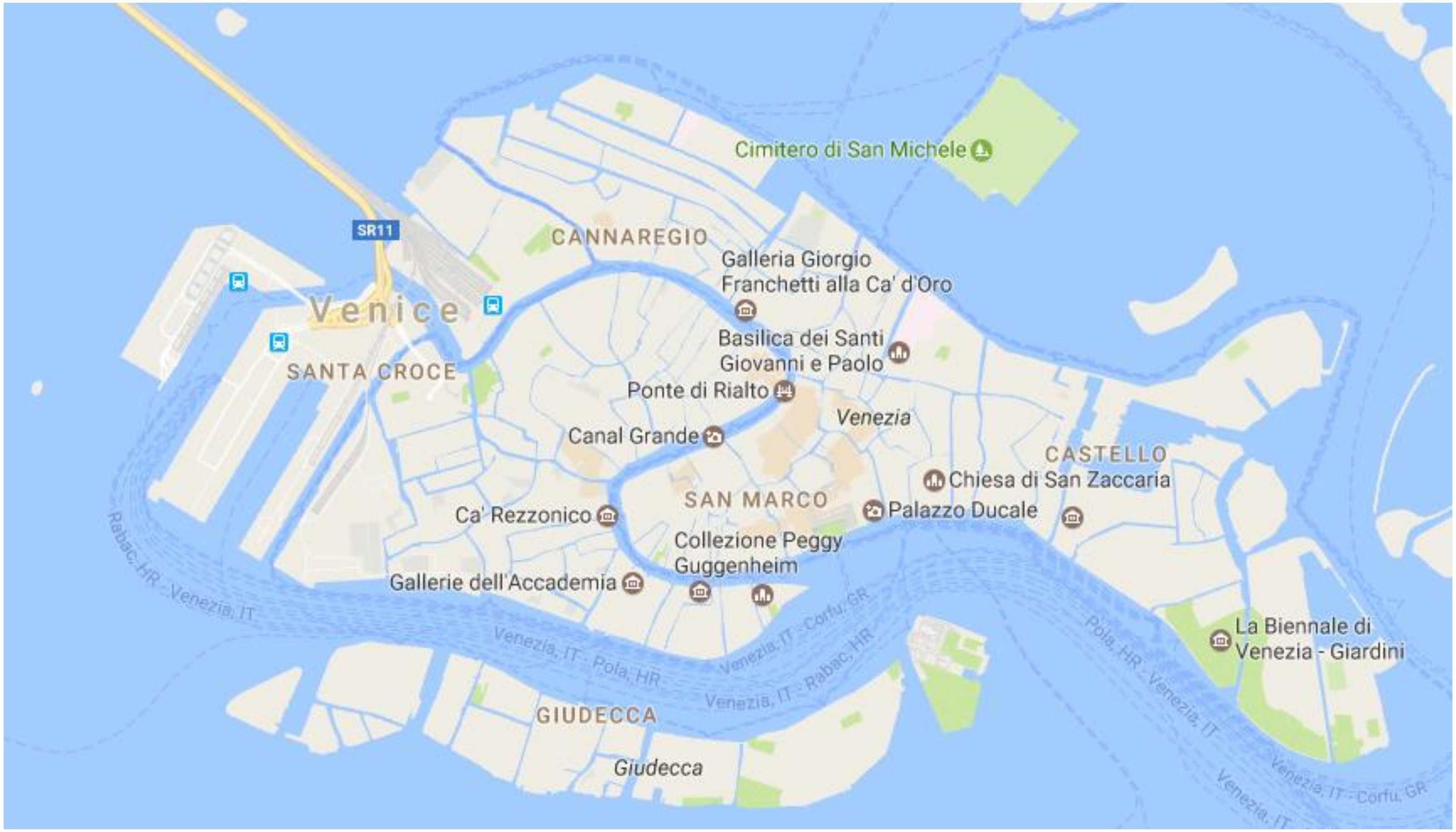
Spanning Tree of G



Spanning Tree of G

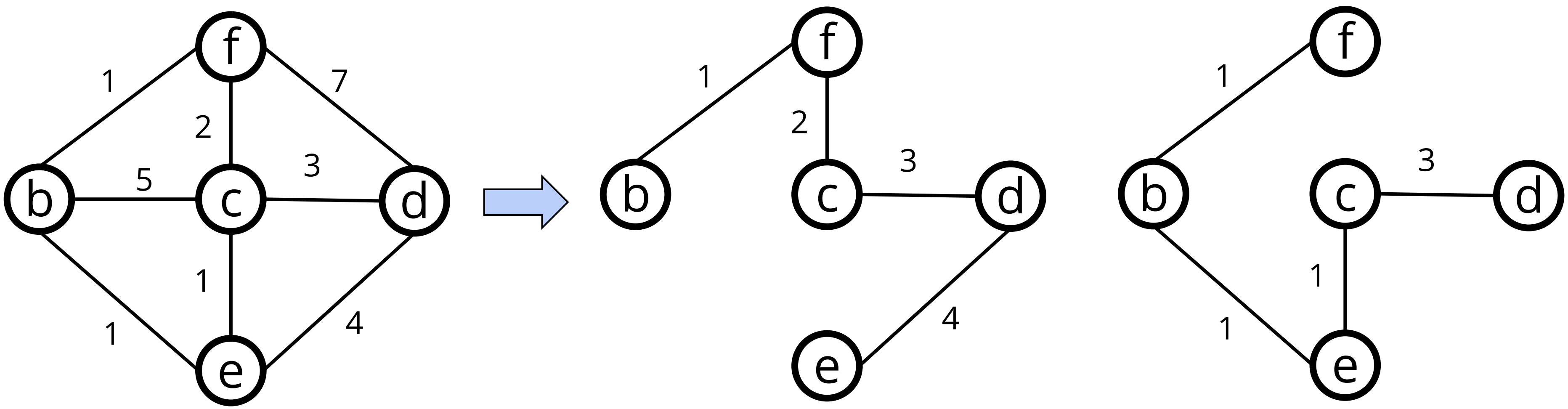
Another Application

- You plan to visit all the important heritage sites, but in short time



Minimum Spanning Tree

- A Minimum Spanning Tree (MST) of a weighted graph G is a spanning tree of G that has the least possible total weight compared to all other spanning trees of G.

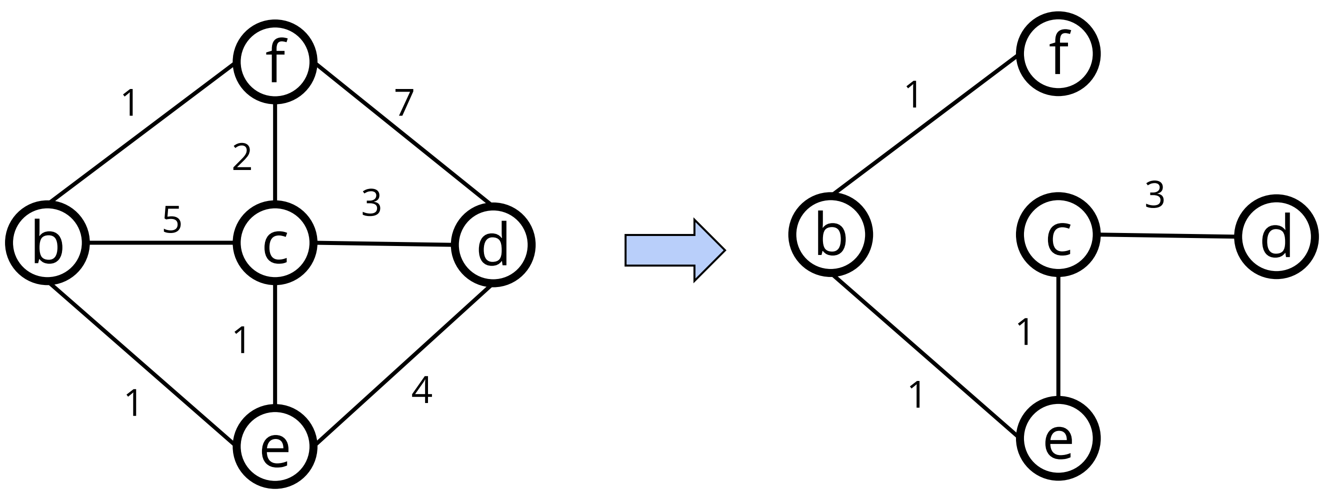


Total weight = 10

Total weight = 6

Minimum spanning trees: properties

- A valid MST cannot contain a cycle
- If we add or remove an edge from an MST, it's no longer a valid MST for that graph.
  - Adding an edge introduces a cycle; removing an edge means vertices are no longer connected.
  - An MST is always a tree.
  - If every edge has a unique weight, there exists a unique MST.



Total weight = 6

Finding a minimum spanning

- It is not always easy to derive a minimum spanning tree 'with eyes’.
- Two efficient algorithms for finding a minimum spanning tree:
  - Prim's algorithm
  - Kruskal's algorithm

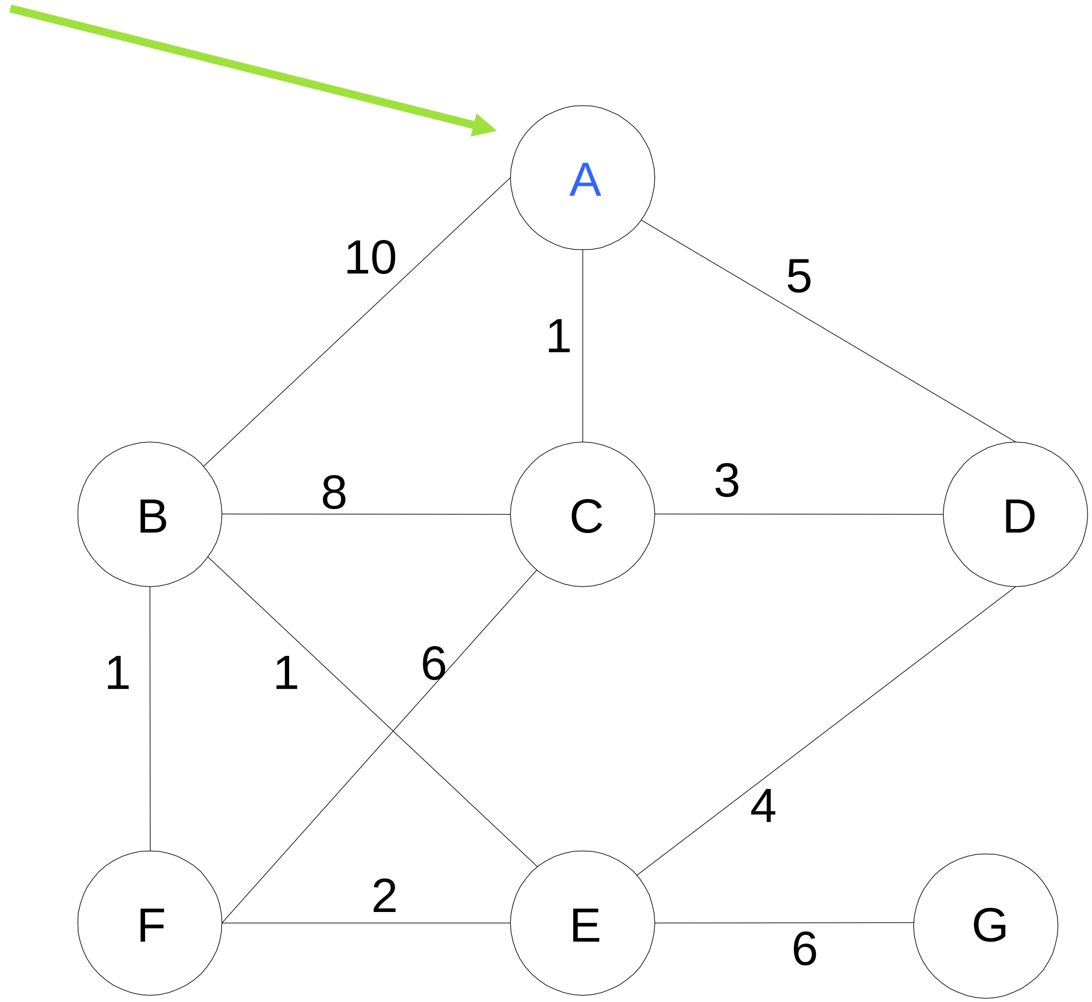
Prim’s algorithm

- Greedy approach to find the minimum spanning tree.
- **Idea**: Grow a tree by adding an edge from the “known” vertices to the “unknown” vertices.
  - Pick the edge with the smallest weight.

Prim’s algorithm

Starting from empty *T*, choose a vertex at random and initialize

V = {A}, T ={}

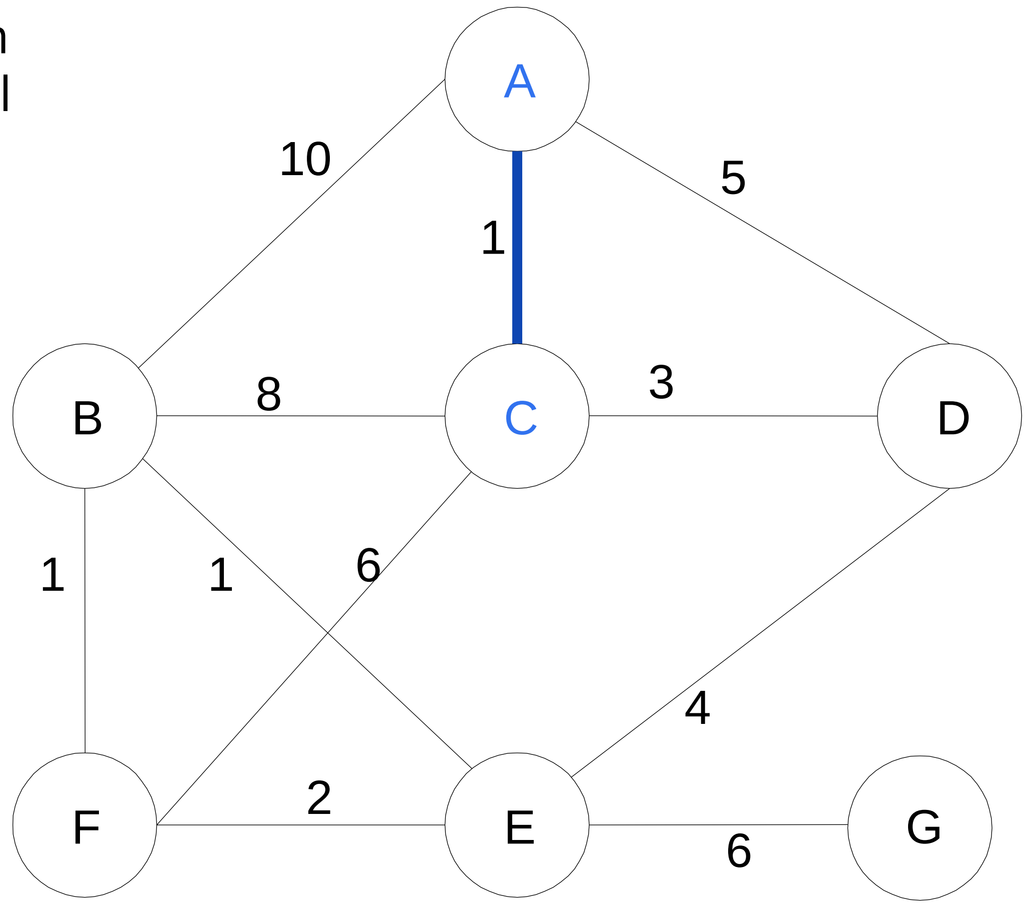


Prim’s algorithm

Choose the vertex u not in V such that edge weight from u to a vertex in V is minimal (greedy!)

V = {A,C}

T = { (A,C) }

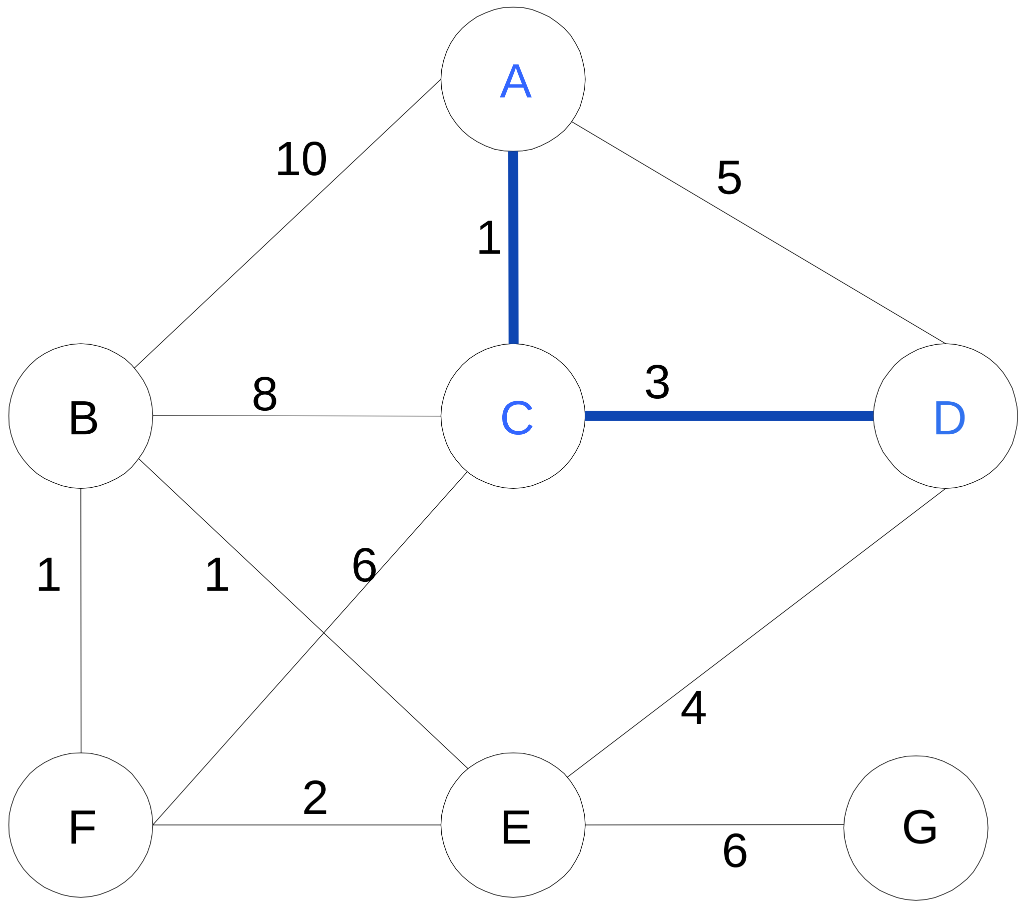


Prim's algorithm

Repeat until all vertices have been chosen

V = {A,C,D}

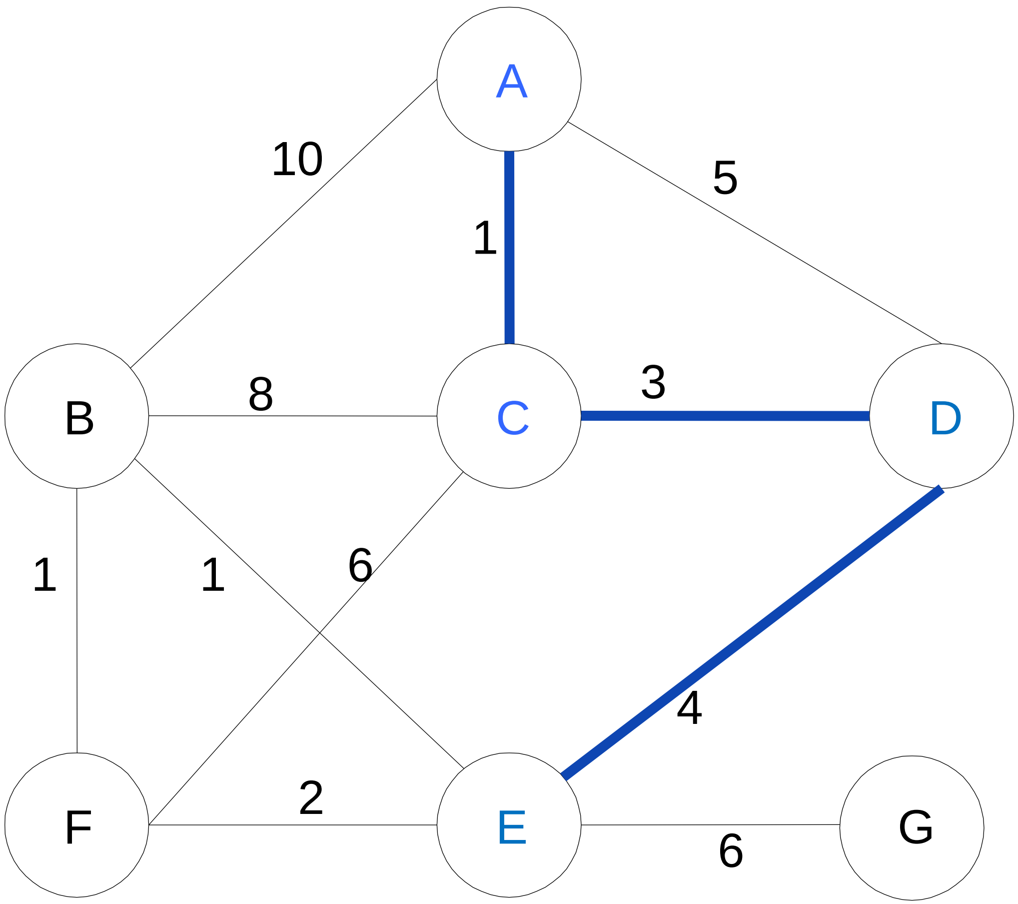
T= { (A,C), (C,D)}



Prim's algorithm

V = {A,C,D,E}

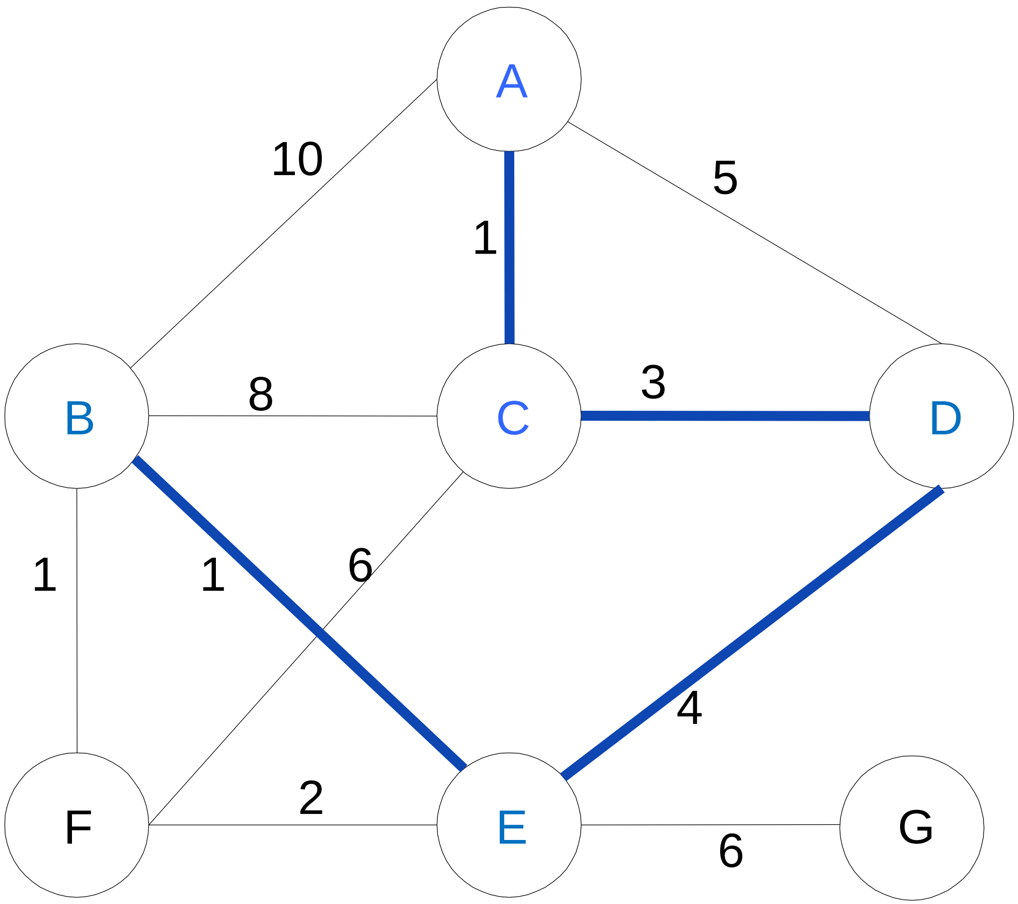
T = { (A,C), (C,D), (D,E) }



Prim's algorithm

V = {A,C,D,E,B}

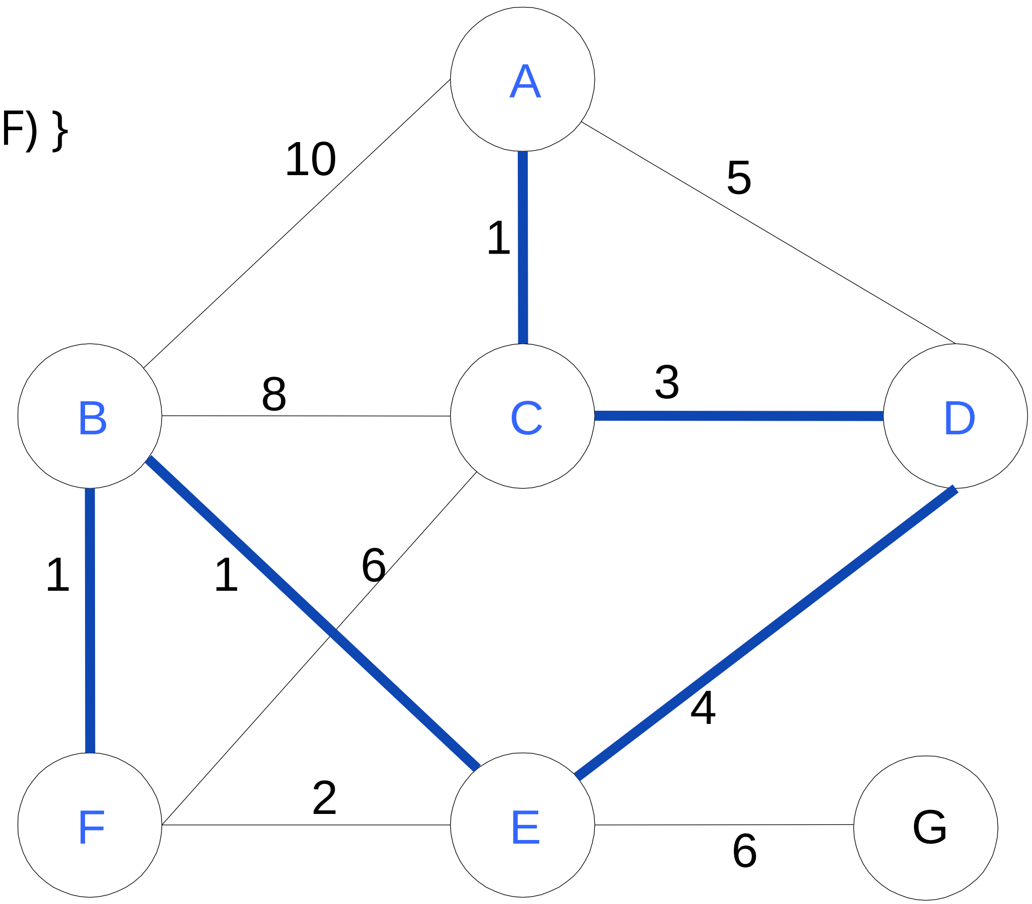
T = { (A,C), (C,D), (D,E), (E,B) }



Prim's algorithm

V = {A,C,D,E,B,F}

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

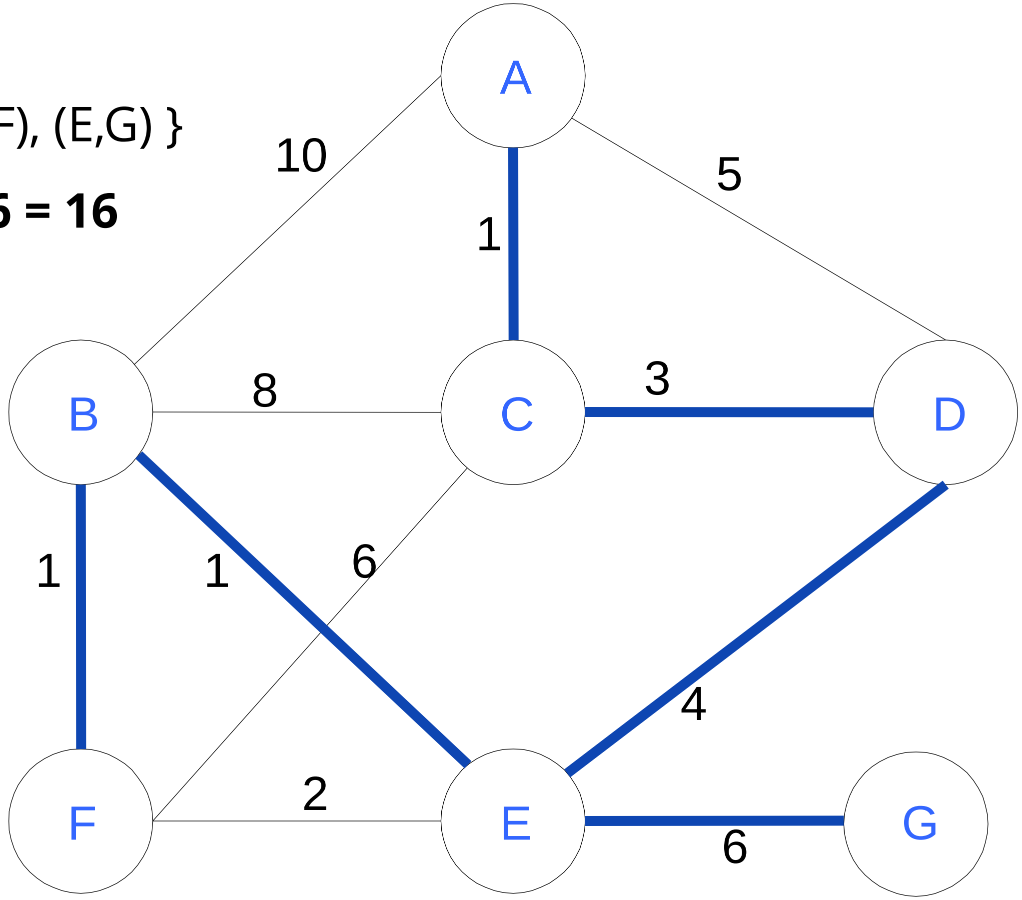


Prim's algorithm

V = {A,C,D,E,B,F,G}

T = { (A,C), (C,D), (D,E), (E,B), (B,F), (E,G) }

Final Cost: 1 + 3 + 4 + 1 + 1 + 6 = 16



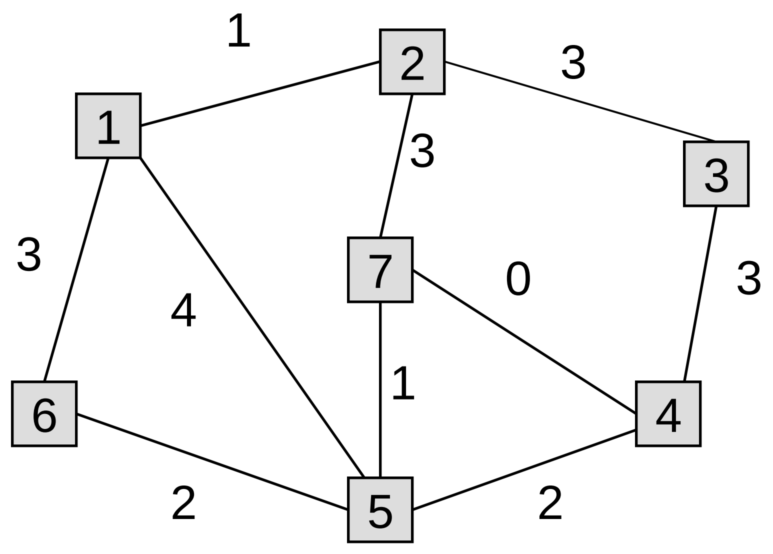
Prim's Algorithm Implementation

```
Select a random vertex v.
Initialize the variables as: X:={v}, Y=V-X, E={}
while X != V do:
     Select e{x,y} edge of the graph where x ∈ X, y ∈ Y
       and w(e) is minimal. //weight of an edge e is w(e)
     X:= X ∪ {y};
     E:= E ∪ {(x,y)};
     Y:= Y-y
Return with the (X,E) tree
```

Kruskal's MST Algorithm

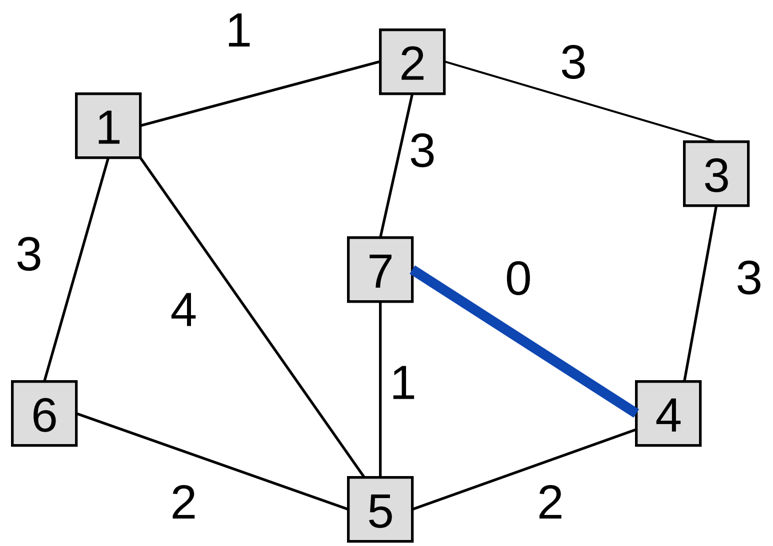
- Idea: Grow a forest out of edges that do not create a cycle.
- Pick an edge with the smallest weight.
- Steps
  - Remove all loops and parallel edges (keep the minimum one)
  - Arrange all edges in their increasing order of weight
  - Add the edge which has the least weight

Kruskal's Algorithm



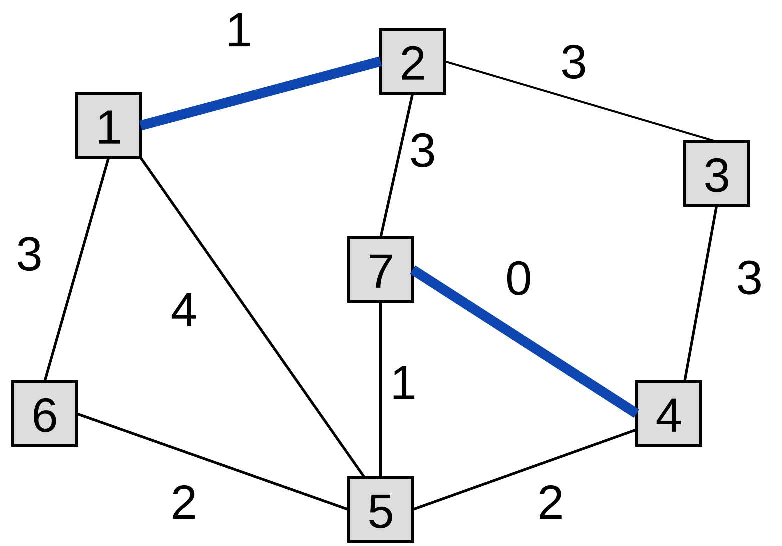
{7,4}	{2,1}	{7,5}	{5,6}	{5,4}	{1,6}	{2,7}	{2,3}	{3,4}	{1,5}
0	1	1	2	2	3	3	3	3	4

Kruskal’s Algorithm



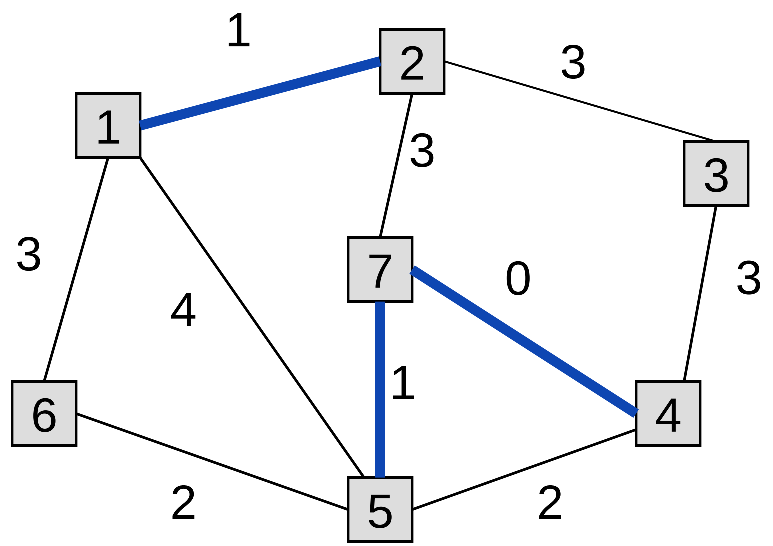
~~{7,4}~~	{2,1}	{7,5}	{5,6}	{5,4}	{1,6}	{2,7}	{2,3}	{3,4}	{1,5}
~~0~~	1	1	2	2	3	3	3	3	4

Kruskal’s Algorithm



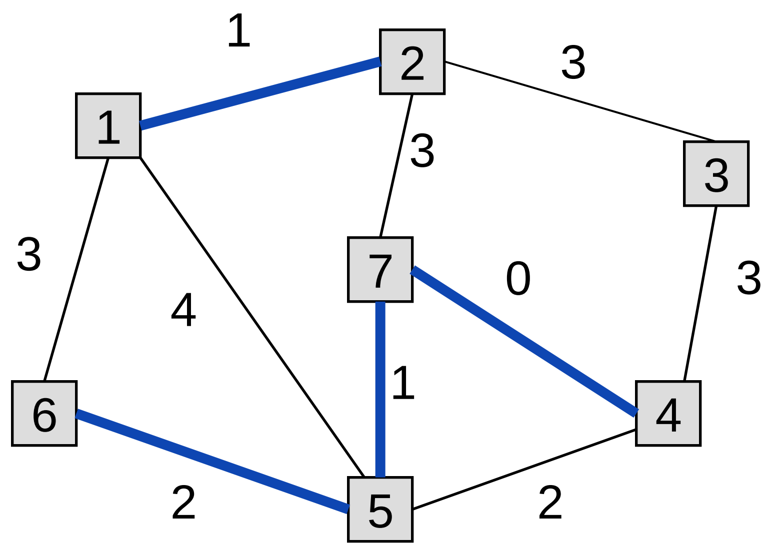
~~{7,4}~~	~~{2,1}~~	{7,5}	{5,6}	{5,4}	{1,6}	{2,7}	{2,3}	{3,4}	{1,5}
~~0~~	~~1~~	1	2	2	3	3	3	3	4

Kruskal’s Algorithm



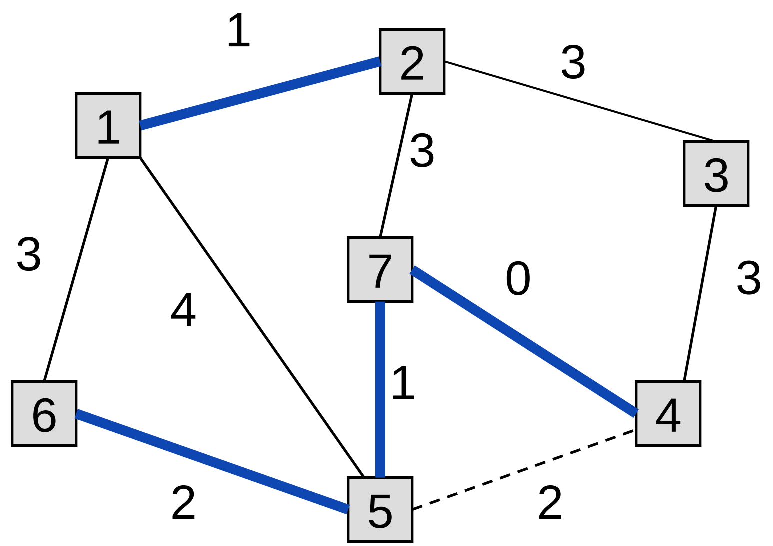
~~{7,4}~~	~~{2,1}~~	~~{7,5}~~	{5,6}	{5,4}	{1,6}	{2,7}	{2,3}	{3,4}	{1,5}
~~0~~	~~1~~	~~1~~	2	2	3	3	3	3	4

Kruskal’s Algorithm



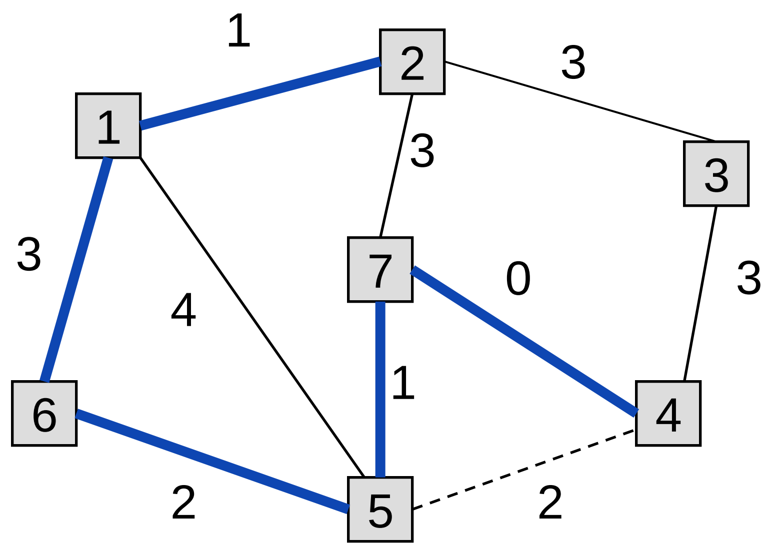
~~{7,4}~~	~~{2,1}~~	~~{7,5}~~	~~{5,6}~~	{5,4}	{1,6}	{2,7}	{2,3}	{3,4}	{1,5}
~~0~~	~~1~~	~~1~~	~~2~~	2	3	3	3	3	4

Kruskal’s Algorithm



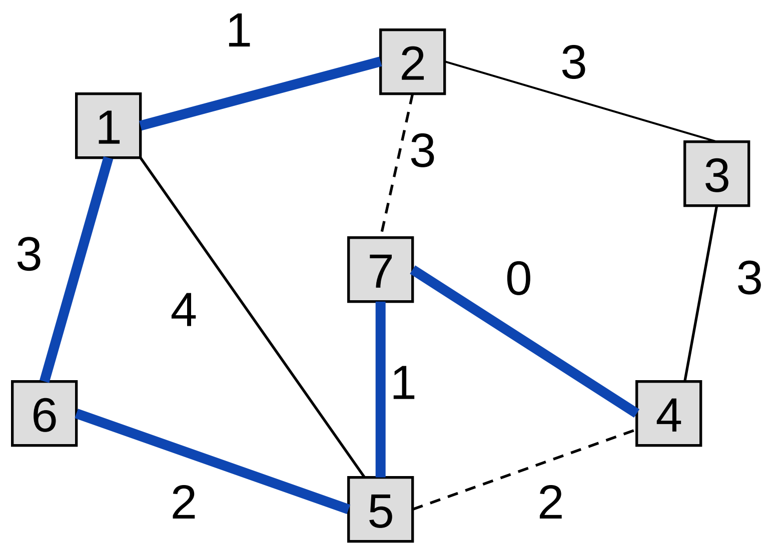
~~{7,4}~~	~~{2,1}~~	~~{7,5}~~	~~{5,6}~~	~~{5,4}~~	{1,6}	{2,7}	{2,3}	{3,4}	{1,5}
~~0~~	~~1~~	~~1~~	~~2~~	~~2~~	3	3	3	3	4

Kruskal's Algorithm



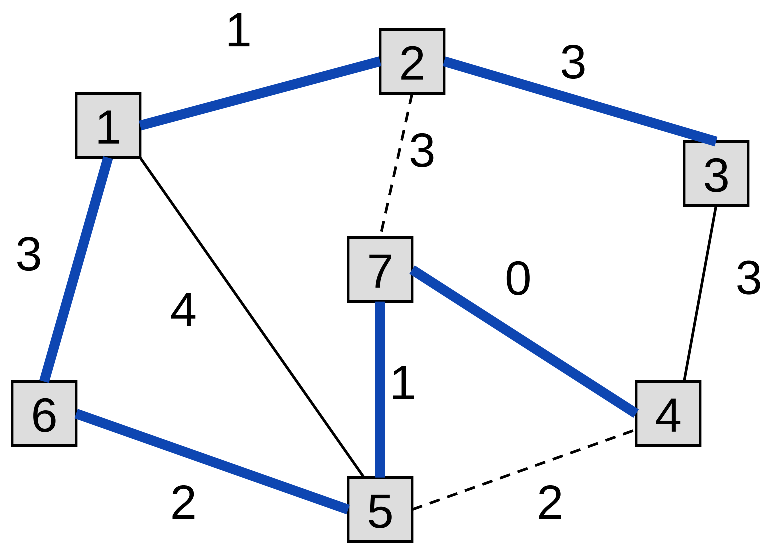
~~{7,4}~~	~~{2,1}~~	~~{7,5}~~	~~{5,6}~~	~~{5,4}~~	~~{1,6}~~	{2,7}	{2,3}	{3,4}	{1,5}
~~0~~	~~1~~	~~1~~	~~2~~	~~2~~	~~3~~	3	3	3	4

Kruskal’s Algorithm



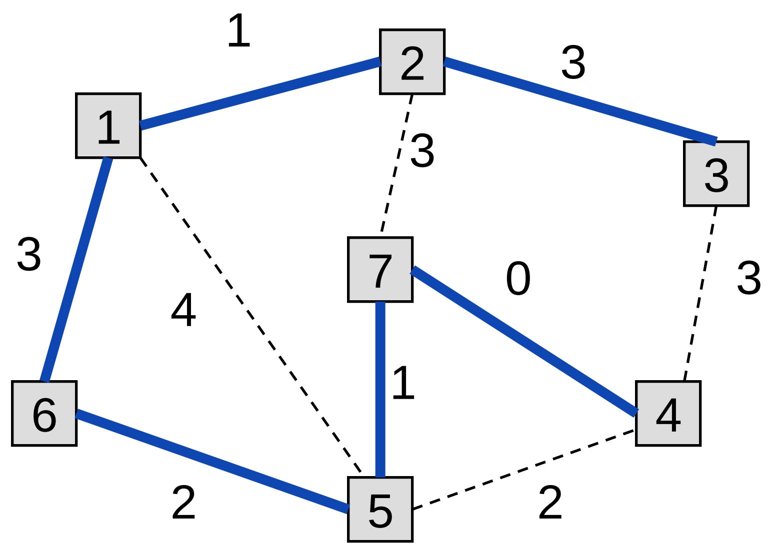
~~{7,4}~~	~~{2,1}~~	~~{7,5}~~	~~{5,6}~~	~~{5,4}~~	~~{1,6}~~	~~{2,7}~~	{2,3}	{3,4}	{1,5}
~~0~~	~~1~~	~~1~~	~~2~~	~~2~~	~~3~~	~~3~~	3	3	4

Kruskal’s Algorithm



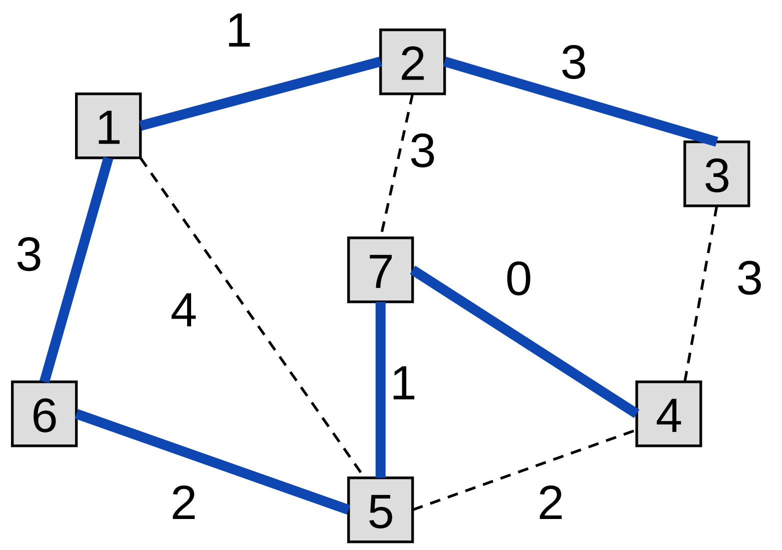
~~{7,4}~~	~~{2,1}~~	~~{7,5}~~	~~{5,6}~~	~~{5,4}~~	~~{1,6}~~	~~{2,7}~~	~~{2,3}~~	~~{3,4}~~	{1,5}
~~0~~	~~1~~	~~1~~	~~2~~	~~2~~	~~3~~	~~3~~	~~3~~	~~3~~	4

Kruskal’s Algorithm



~~{7,4}~~	~~{2,1}~~	~~{7,5}~~	~~{5,6}~~	~~{5,4}~~	~~{1,6}~~	~~{2,7}~~	~~{2,3}~~	~~{3,4}~~	{1,5}
~~0~~	~~1~~	~~1~~	~~2~~	~~2~~	~~3~~	~~3~~	~~3~~	~~3~~	4

Kruskal's Algorithm



~~{7,4}~~	~~{2,1}~~	~~{7,5}~~	~~{5,6}~~	~~{5,4}~~	~~{1,6}~~	~~{2,7}~~	~~{2,3}~~	~~{3,4}~~	~~{1,5}~~
~~0~~	~~1~~	~~1~~	~~2~~	~~2~~	~~3~~	~~3~~	~~3~~	~~3~~	~~4~~

Kruskal's Algorithm Implementation

```
Kruskals():
    sort edges in increasing order of length (e1, e2, e3,
  ..., em).

    T := {}.

    for i = 1 to m
        if ei  does not add a cycle:
            add ei to T.

    return T.
```

All-Pairs Shortest Path

- Dijkstra's algorithm requires a starting vertex
- What if you want to find the shortest path between all pairs of vertices in the graph?
  - Run Dijkstra's for each vertex v?
  - Can we do better?

    Use dynamic programming: An algorithmic technique that systematically records the answers to sub-problems in a table and re-uses those recorded results.

Floyd-Warshall algorithm

- optimal substructure property

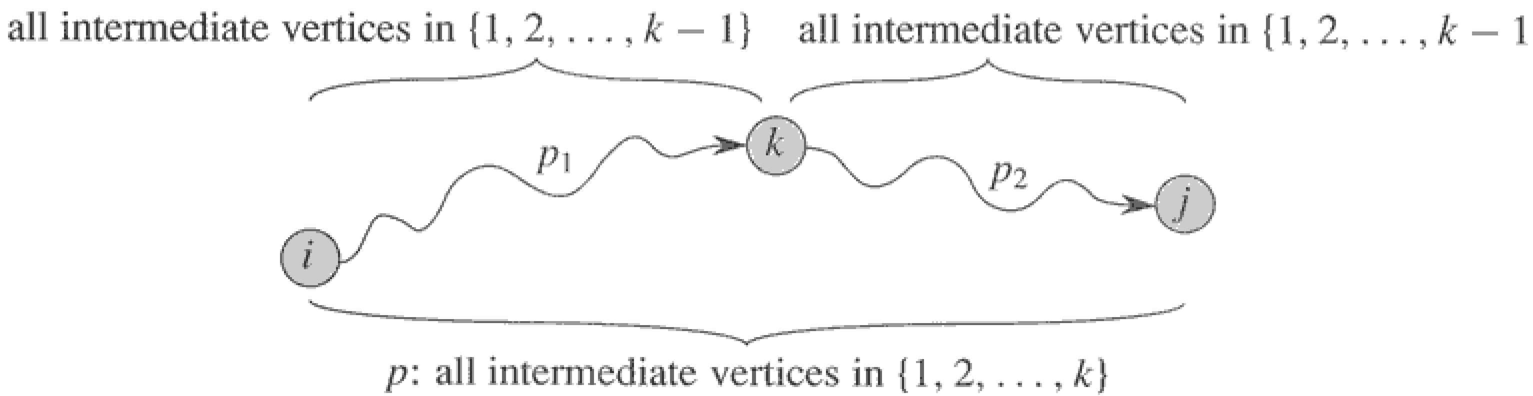


Figure 25.3 Path p is a shortest path from vertex i to vertex j, and k is the highest-numbered intermediate vertex of p. Path p_1, the portion of path p from vertex i to vertex k, has all intermediate vertices in the set $\{1, 2, \ldots, k-1\}$. The same holds for path p_2 from vertex k to vertex j.

The Algorithm

M[u][v] stores the cost of the best path from u to v

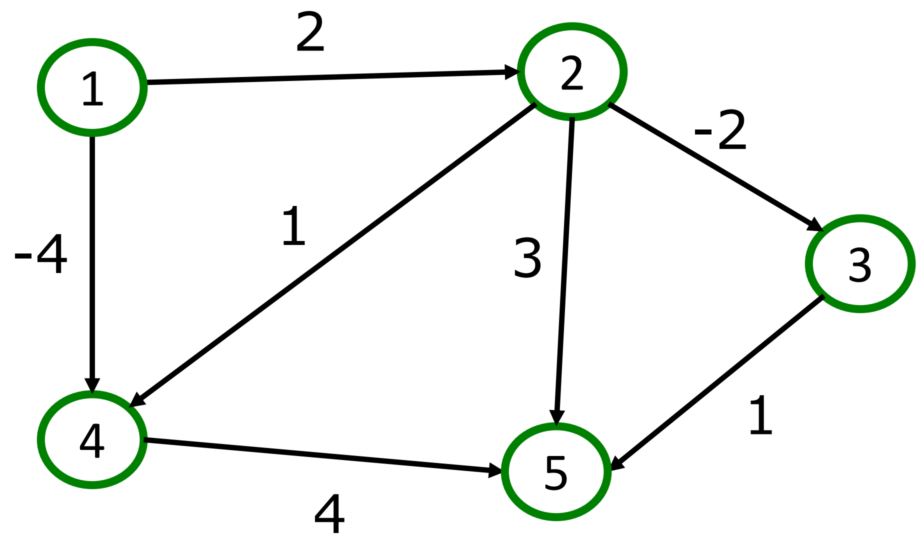
Initialized to cost of edge between M[u][v]

The algorithm:

```
for (int k = 1; k =< V; k++)
  for (int i = 1; i =< V; i++)
    for (int j = 1; j =< V; j++)
      if ( M[i][k]+ M[k][j] < M[i][j] )
         M[i][j] = M[i][k]+ M[k][j]
```

After the k^{th} iteration, the matrix M includes the shortest path between all pairs that use on only vertices 1..k as intermediate vertices in the paths

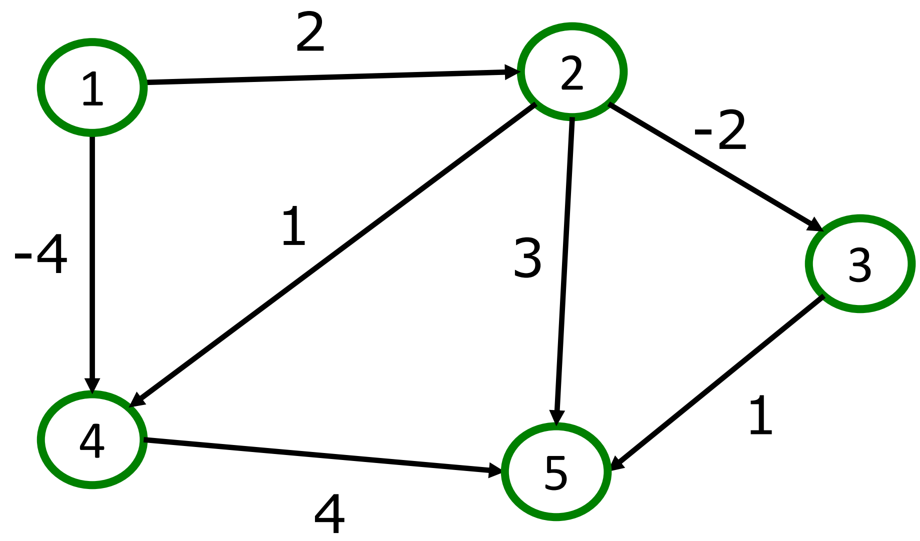
Floyd-Warshall algorithm



Initial state of the matrix:

	1	2	3	4	5
1	0	2	∞	-4	∞
2	∞	0	-2	1	3
3	∞	∞	0	∞	1
4	∞	∞	∞	0	4
5	∞	∞	∞	∞	0

Floyd-Warshall algorithm

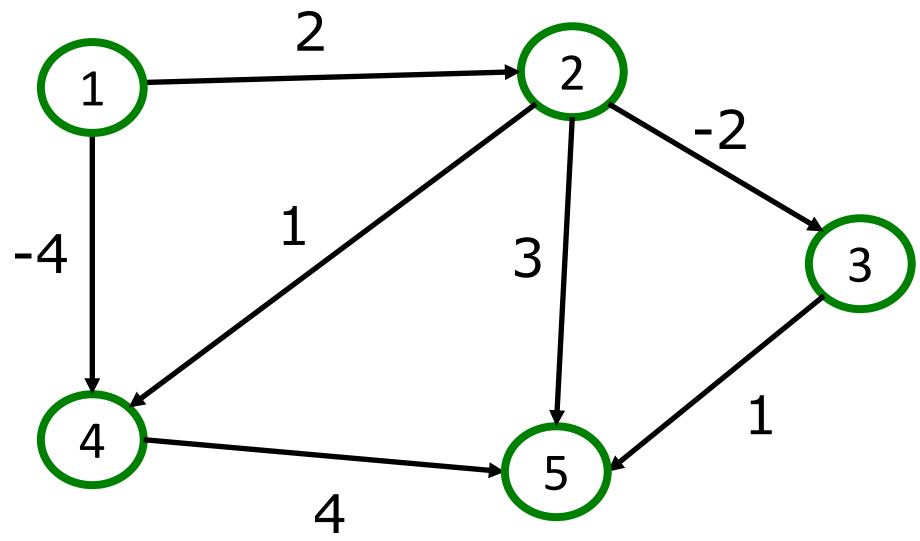


k = 1

	1	2	3	4	5
1	0	2	∞	-4	∞
2	∞	0	-2	1	3
3	∞	∞	0	∞	1
4	∞	∞	∞	0	4
5	∞	∞	∞	∞	0

M[i][j] =
min(M[i][j], M[i][k]+ M[k][j])

Floyd-Warshall algorithm

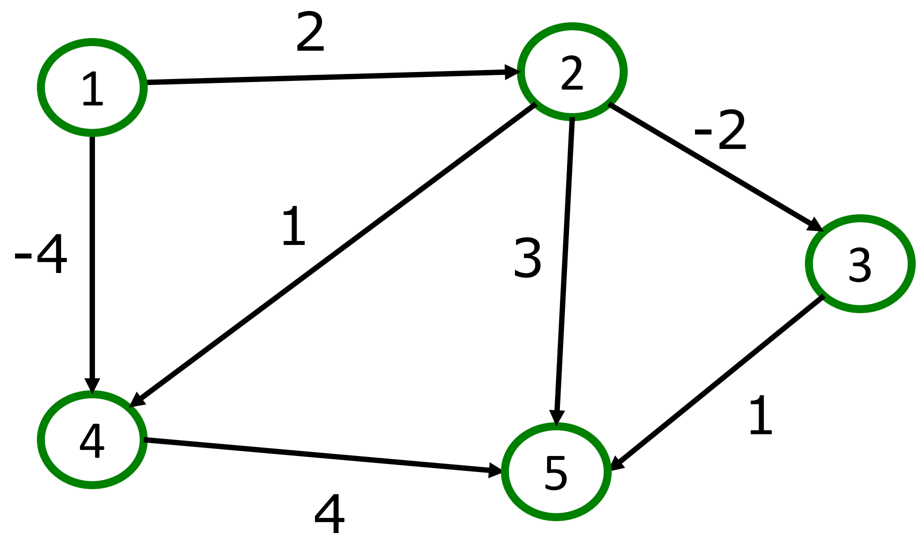


k = 2

	1	2	3	4	5
1	0	2	0	-4	5
2	∞	0	-2	1	3
3	∞	∞	0	∞	1
4	∞	∞	∞	0	4
5	∞	∞	∞	∞	0

M[i][j] =
  min(M[i][j], M[i][k]+ M[k][j])

Floyd-Warshall algorithm

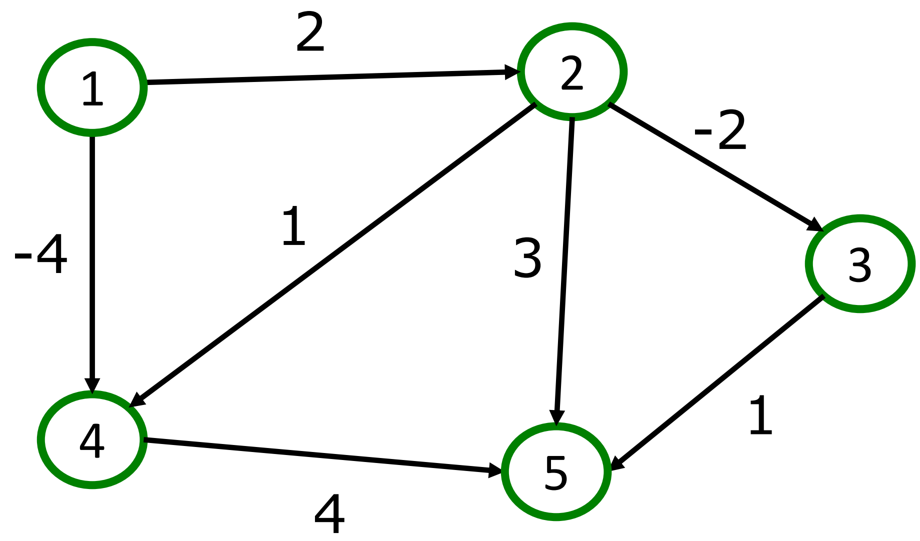


k = 3

	1	2	3	4	5
1	0	2	0	-4	1
2	∞	0	-2	1	-1
3	∞	∞	0	∞	1
4	∞	∞	∞	0	4
5	∞	∞	∞	∞	0

M[i][j] =
min(M[i][j], M[i][k]+ M[k][j])

Floyd-Warshall algorithm

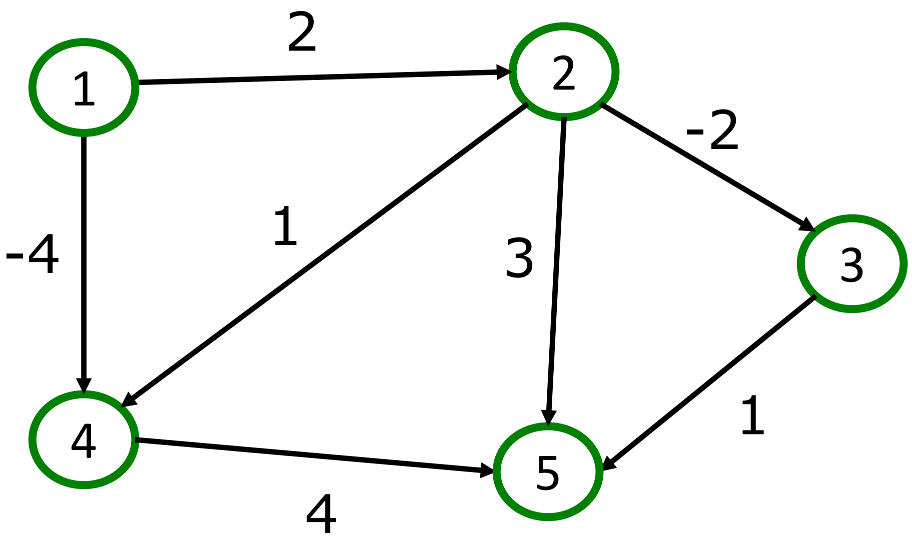


k = 4

	1	2	3	4	5
1	0	2	0	-4	0
2	∞	0	-2	1	-1
3	∞	∞	0	∞	1
4	∞	∞	∞	0	4
5	∞	∞	∞	∞	0

M[i][j] =
min(M[i][j], M[i][k]+ M[k][j])

Floyd-Warshall algorithm



k = 5

	1	2	3	4	5
1	0	2	0	-4	0
2	∞	0	-2	1	-1
3	∞	∞	0	∞	1
4	∞	∞	∞	0	4
5	∞	∞	∞	∞	0

M[i][j] =
min(M[i][j], M[i][k]+ M[k][j])

Graph Algorithms Summary

- Search
  - Depth-first search
  - Breadth-first search
- Topological sort
- Shortest paths
  - Breadth-first search (unweighted only)
  - Dijkstra's algorithm (requires starting vertex)
  - Floyd-Warshall algorithm (all-pairs)
- Minimum spanning trees
  - Prim's algorithm
  - Kruskal's algorithm

Thank You

Review

1. 2-3 trees are B trees of order ____
   a) 1
   b) 2
   c) 3
   d) 4
   e) none of the above

2. In a 2-3 Tree, a 3-Node has ______ data items
   a) 1
   b) 2
   c) 3
   d) 4
   e) none of the above

Review

3. During insertion and deletion operations, Red Black trees require fewer rotations than AVL trees.
   - True
   - False

- A Red-Black tree always has a black root

- A Red-Black tree can have two consecutive red nodes

- A Red-Black tree is always coloured either red or black

- Every path from a node to a null link must have the same number of red nodes

4. How many of the above statements are correct?
   a) 1
   b) 2
   c) 3
   d) 4

Review

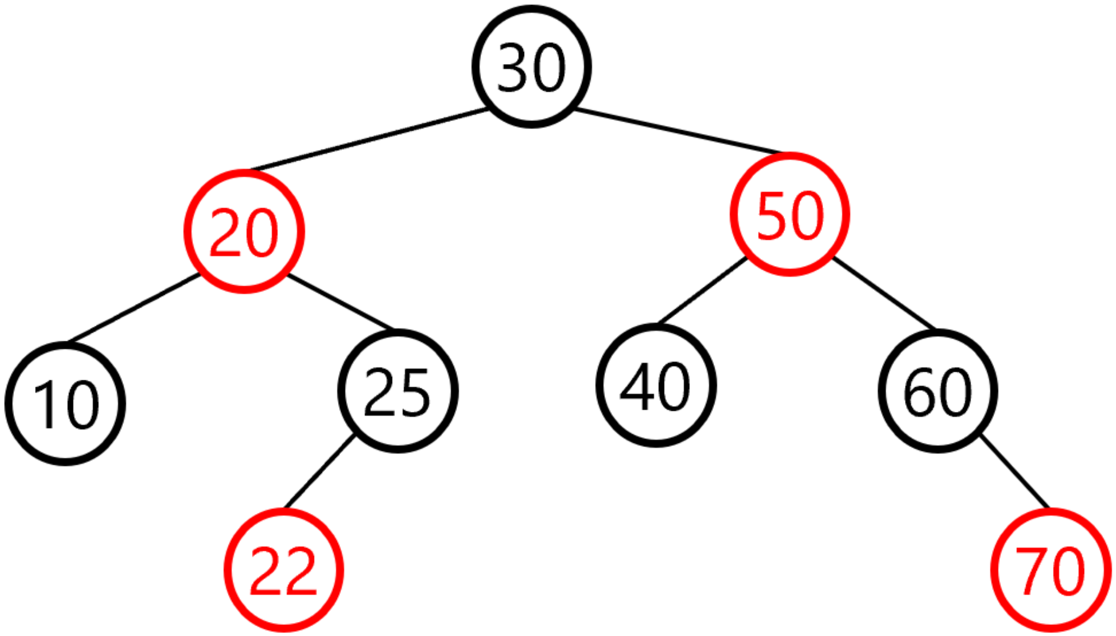
5. When a node is first inserted in a red-black tree, it is placed according to the insert procedure in a binary search tree. What colour is this newly inserted node (initially) if it is NOT the root?
   a) Red
   b) Black
   c) Either Red or Black
   d) It has no colour

6. The time complexity for peek operation on a min-heap is
   a) O(1)
   b) O(n)
   c) O(logn)
   d) O(nlogn)
   e) none of the above

7. Bubble up and bubble down are used to restore heap ordering.
   - True
   - False

Review

8. The following is an example of a **valid** Red-Black tree



- True
- False

Review

9. Show the red-black trees after inserting the keys
   42, 39, 30, 11, 18, 7
   into an initially empty tree.
   Draw the tree after each insertion. For the red nodes, you may use labels (e.g., R) to indicate their colour.

10. Insert the following values into an initially empty 2-3tree:
    1, 2, 3, 4, 5, 6, 7, 8, 9.
    Draw the tree after each insertion.

Review

11. Construct a max-heap by adding the following elements onto the heap in the given order:
    7, 2, 1, 9, 12, 3, 14.
    Draw the heap after each completed insertion of an element.