CS301
Data Structure and Algorithms
Lecture 13: Minimum spanning tree

Pandav Patel
Assistant Professor

Computer Engineering Department
Dharmsinh Desai University
Nadiad, Gujarat, India

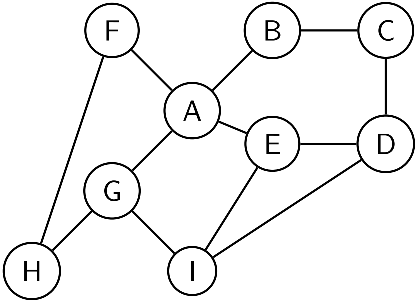
Objective

- To understand what is spanning tree for a graph
- To understand what is minimum spanning tree (MST) for a graph
- To learns algorithms to find out MST for a given graph

Overview

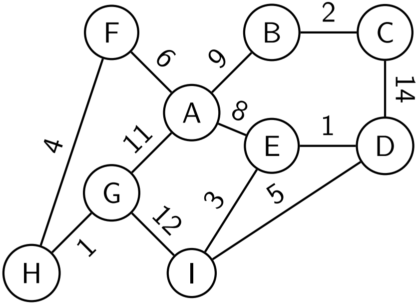
1. Objective
2. Minimum Spanning Tree (MST)
   - What is spanning tree?
   - What is Minimum Spanning Tree (MST)
3. Algorithms to find MST
   - Kruskal's algorithm
   - Prim's algorithm

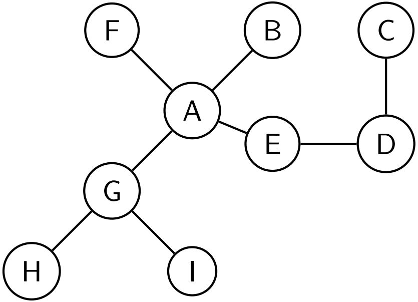
What is spanning tree?



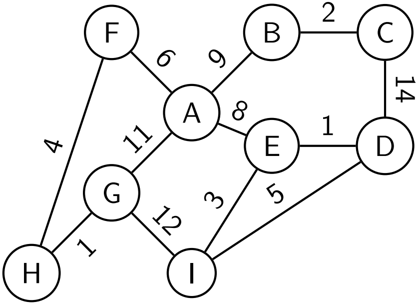
- What is tree?
  - A connected acyclic undirected graph
- Spanning tree for an undirected graph G(V, E) is
  - Connected subgraph (subset of E) without cycles which includes all the vertices (V)
- How many edges in the spanning tree of a graph with V vertices?
  - Always $V-1$.
  - If it is less than $V-1$, then all vertices can not be connected.
  - If it is more than $V-1$, then it must contain a cycle and hence it will not be tree anymore.
- Can a graph have more than one spanning tree?

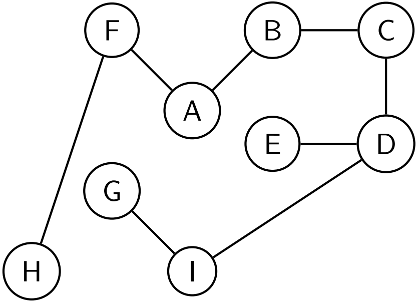
What is spanning tree? (cont...)



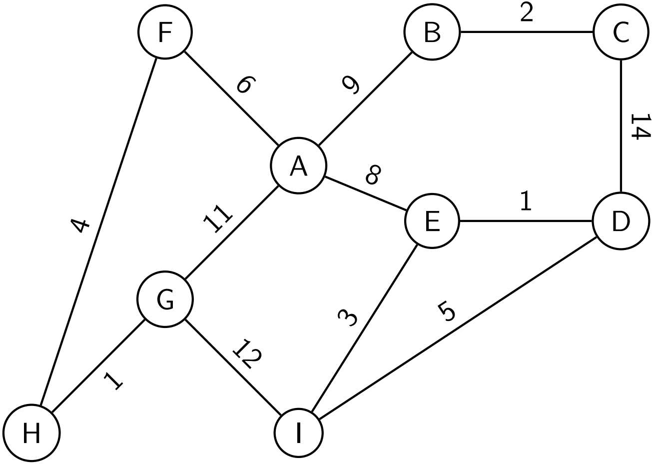


What is spanning tree? (cont...)



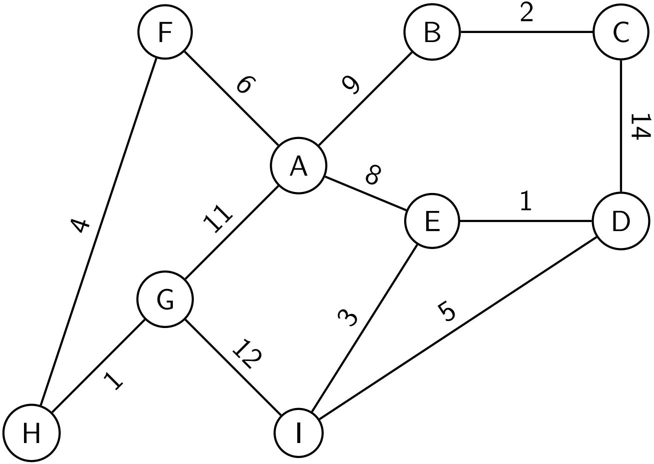


What is Minimum Spanning Tree (MST)?



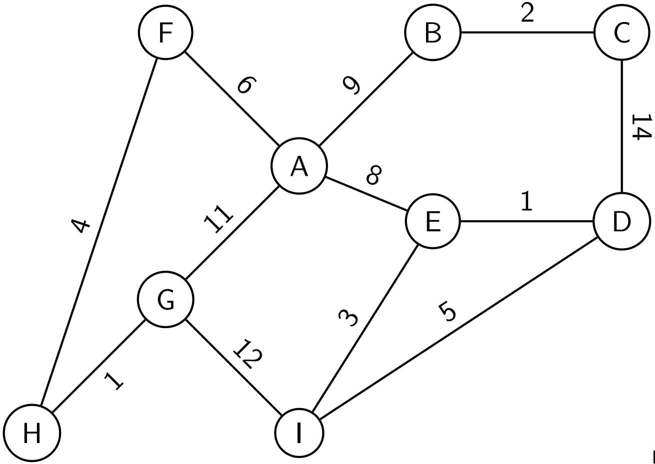
- Spanning tree with minimum cost (sum of weights of edges included in spanning tree) is called Minimum Spanning Tree (**MST**)
- Is MST unique?
  - Yes, If edge weights of a graph are distinct
  - There may be more than one MST for a graph if edge weights are not unique

Kruskal's algorithm (cont...)



- Edges are sorted in ascending order of their weights
- Sets are used to detect cycle in the tree. Initially each vertex has its own set
- Picks edges one by one in ascending order
- If two endpoints of an edge are in the same set then it forms a cycle. Otherwise that edge is added to the spanning tree and sets corresponding to endpoints are combined

Kruskal's algorithm (cont...)

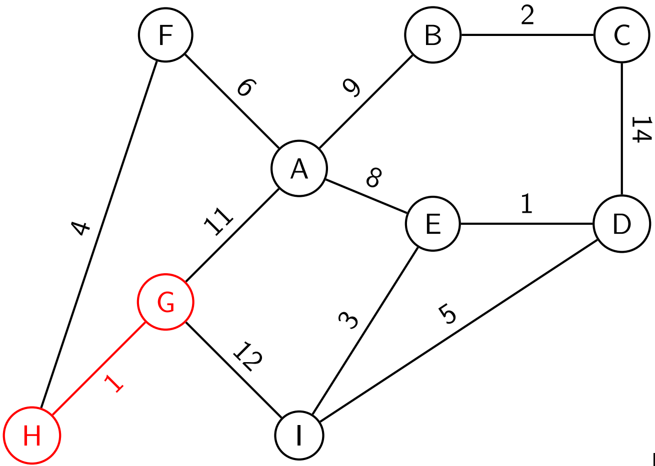


{A}
{B}
{C}
{D}
{E}
{F}
{G}
{H}
{I}

- Cost: 0

1 G-H
1 D-E
2 B-C
3 E-I
4 F-H
5 D-I
6 A-F
8 A-E
9 A-B
11 A-G
12 G-I

Kruskal's algorithm (cont...)



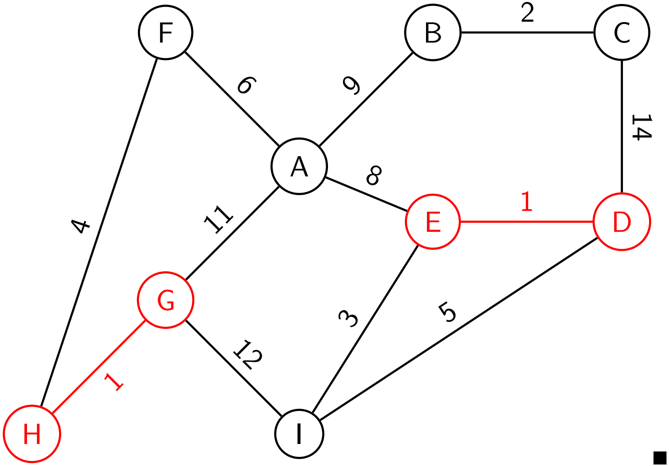
{A}
{B}
{C}
{D}
{E}
{F}
{G, H}

{I}

- Cost: 1

1 G-H
1 D-E
2 B-C
3 E-I
4 F-H
5 D-I
6 A-F
8 A-E
9 A-B
11 A-G
12 G-I

Kruskal's algorithm (cont...)



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

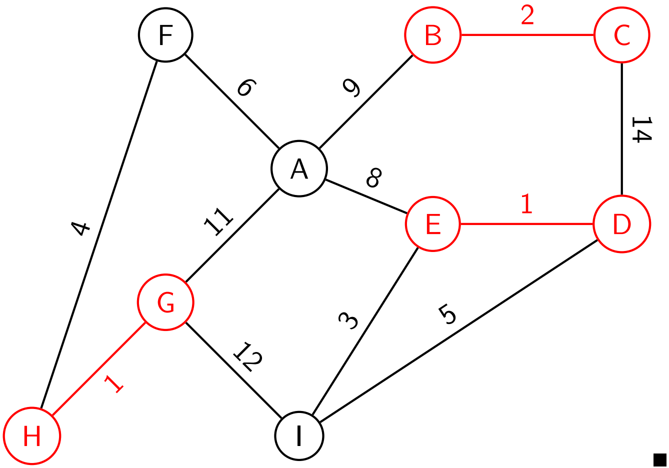
{F}
{G, H}

{I}

- Cost: 2

1 D-E
2 B-C
3 E-I
4 F-H
5 D-I
6 A-F
8 A-E
9 A-B
11 A-G
12 G-I
14 C-D

Kruskal's algorithm (cont...)

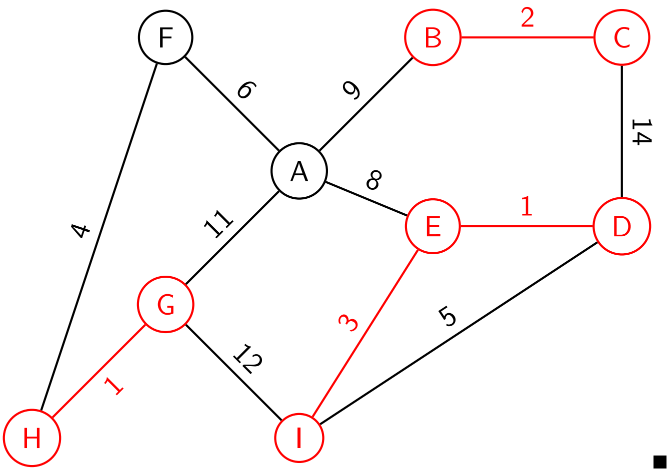


{A}
{B, C}
{D, E}
{F}
{G, H}
{I}

- Cost: 4

2 B-C
3 E-I
4 F-H
5 D-I
6 A-F
8 A-E
9 A-B
11 A-G
12 G-I
14 C-D

Kruskal's algorithm (cont...)



{A}

{B, C}

{D, E, I}

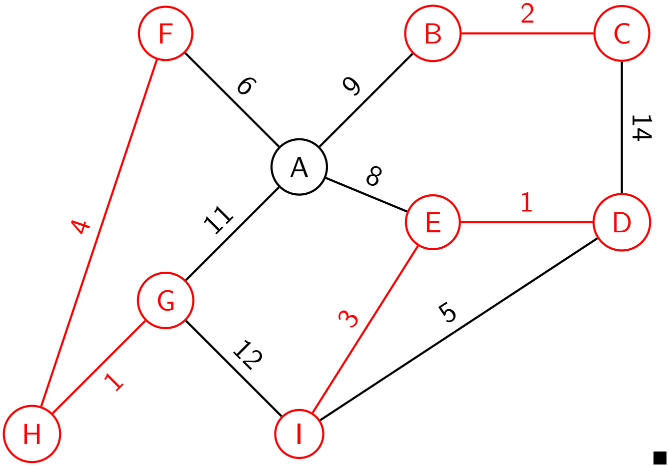
{F}

{G, H}

- Cost: 7

3 E-I
4 F-H
5 D-I
6 A-F
8 A-E
9 A-B
11 A-G
12 G-I
14 C-D

Kruskal's algorithm (cont...)



{A}

{B, C}

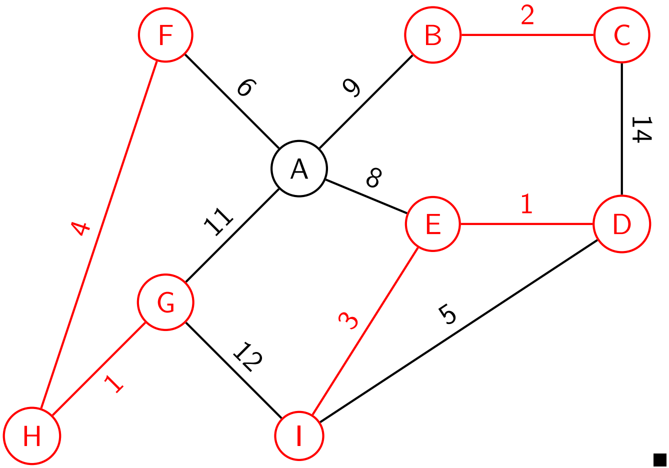
{D, E, I}

{F, G, H}

- Cost: 11

4 F-H
5 D-I
6 A-F
8 A-E
9 A-B
11 A-G
12 G-I
14 C-D

Kruskal's algorithm (cont...)



{A}

{B, C}

{D, E, I}

{F, G, H}

- Cost: 11

5 D-I

6 A-F

8 A-E

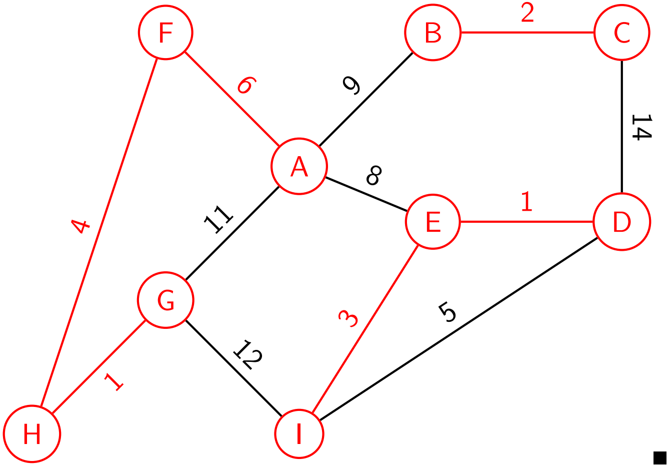
9 A-B

11 A-G

12 G-I

14 C-D

Kruskal's algorithm (cont...)



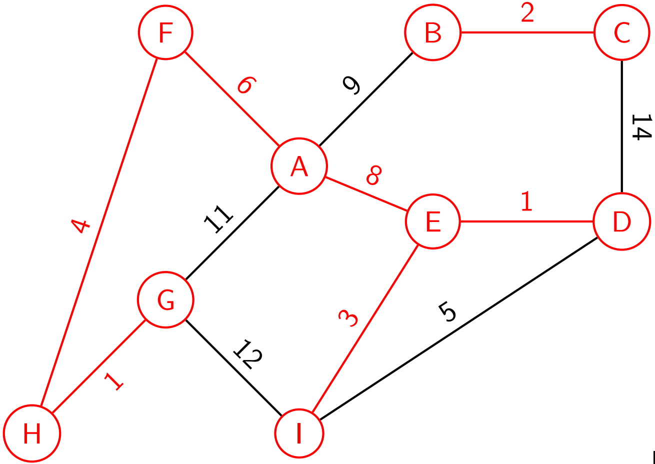
{A, F, G, H}
{B, C}

{D, E, I}

- Cost: 17

6 A-F
8 A-E
9 A-B
11 A-G
12 G-I
14 C-D

Kruskal's algorithm (cont...)

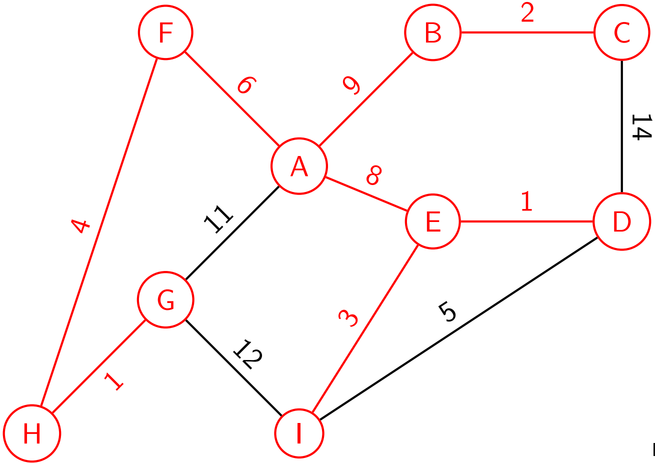


{A, F, G, H, D, E, I}
{B, C}

8 A-E
9 A-B
11 A-G
12 G-I
14 C-D

- Cost: 25

Kruskal's algorithm (cont...)



{A, F, G, H, D, E, I, B, C}

- Cost: 34

9 A-B

11 A-G

12 G-I

14 C-D

Kruskal's algorithm (cont...)

- Is it guaranteed that Kruskal's algo will produce optimal solution (tree with least possible weight)? proof? Let us prove it by contradiction (Assumption: All edges of the graph have unique weight). Link to relevant NPTEL video
  - Assume that Kruskal's algorithm picks following edges. $K_1, K_2....$ are representing weights of respective edges
    - $K_1 < K_2 < K_3...... < K_i...... < K_{V-1}$
  - Say there exists an optimal spanning tree with less weight than one produced by Kruskal's algo. Edges are as follow
    - $O_1 < O_2 < O_3...... < O_i...... < O_{V-1}$
  - Assume $K_j = O_j$ for all $j < i$. In other words i^{th} edge is the first edge with different weight in both trees
    - K_i can not be greater than O_i, otherwise Kruskal's algo would have picked O_i over K_i. As Picking O_i can not form cycle because K_1 to K_{i-1} is same as O_1 to O_{i-1}. If it forms cycle in Kruskal's by picking O_i then it would have formed cycle in optimal solution as well. It is contradiction as tree can not contain cycle
    - Can K_i be smaller than O_i?

Kruskal's algorithm (cont...)

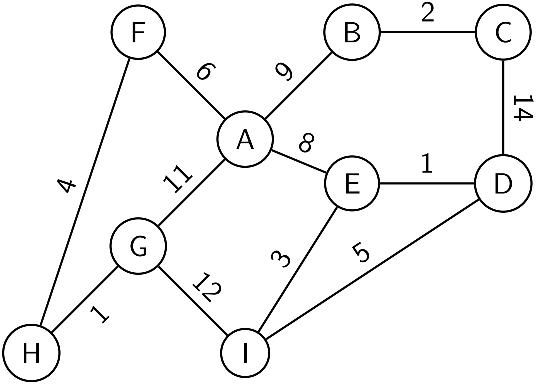
- Continuing from previous slide...
- Can K_i be smaller than O_i? If so...
- Add K_i to optimal tree. It will form a cycle. Say O_p, O_q, O_r, and K_i are part of the cycle
- If K_i is not the largest among O_p, O_q, O_r, and K_i then largest edge can be removed and K_i be added to optimal tree and it will reduce its weight. Means original optimal tree was not actually optimal. Hence the contradiction.
- So K_i should be the largest of all the edges in the cycle. In that case O_p, O_q and O_r must be from O_1 to O_{i-1}, as they all need to be smaller than K_i (hence they must be smaller than O_i). As O_p, O_q and O_r are less than K_i, they must be present in K_1 to K_{i-1}. And in that case they would have formed a cycle in tree produced by Kruskal's algorithm (K_p, K_q, K_r, and K_i), which is a contradiction as it is spanning tree which could not have a cycle

- Algorithm works fine when there are edges with same weight in the graph. We assumed that edge weights are unique just to simplify our proof.
- When edges with same weights are present graph may have more than one MST

Kruskal's algorithm (cont...)

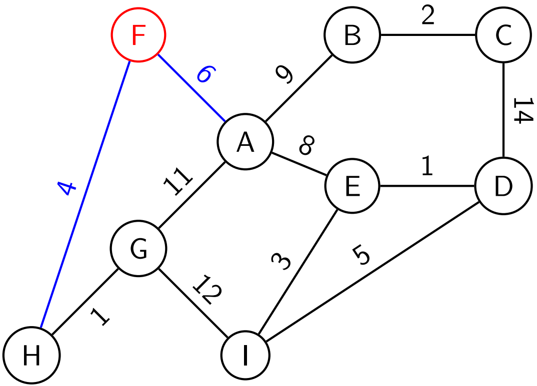
- How would you implement Kruskal's algorithm? What data structures are required?
- How to maintain sets? Using hashmap? What is the time complexity of merging two hashmaps?
- Any other data structure? What will be the complexity of find and union for that data structure?
- What is overall complexity of your approach?
- By using a data structure called union set, complexity can be reduced to **E*log(E)**
  - Click this to learn disjoint set data structure

Prim's algorithm



- Cut the graph into two sets. Let us call them S and S'. A vertex is picked at random and is added to S. Rest of the vertices are part of S'
- Smallest edge that is connecting a vertex in S and another vertex in S' is added to the solution and its endpoint in S' is moved from S' to S
- Repeat above step until S' becomes empty
- NOTE: Edges connecting two sets will change as vertex moves from S' to S

Prim's algorithm (cont...)

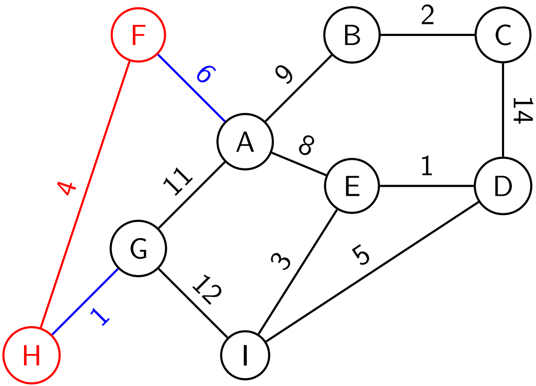


$S = \{F\}$

$S' = \{A, B, C, D, E, G, H, I\}$

- Cost: 0
- Red edges are part of MST and blue edges are connecting two sets
- Red vertices are present in S, rest of the vertices are in S'

Prim's algorithm (cont...)

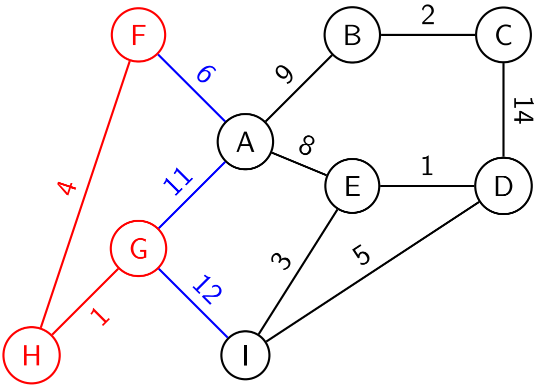


$S = \{F, H\}$

$S' = \{A, B, C, D, E, G, I\}$

- Cost: 4
- Red edges are part of MST and blue edges are connecting two sets
- Red vertices are present in S, rest of the vertices are in S'

Prim's algorithm (cont...)

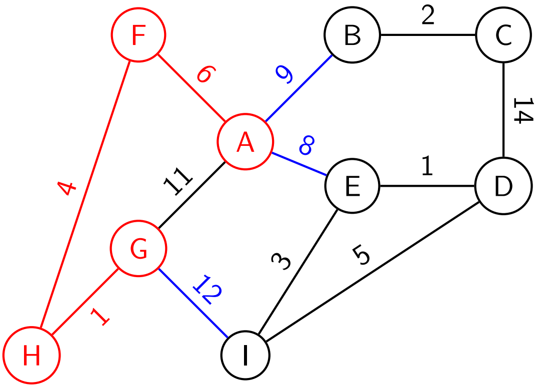


$S = \{F, H, G\}$

$S' = \{A, B, C, D, E, I\}$

- Cost: 5
- Red edges are part of MST and blue edges are connecting two sets
- Red vertices are present in S, rest of the vertices are in S'

Prim's algorithm (cont...)

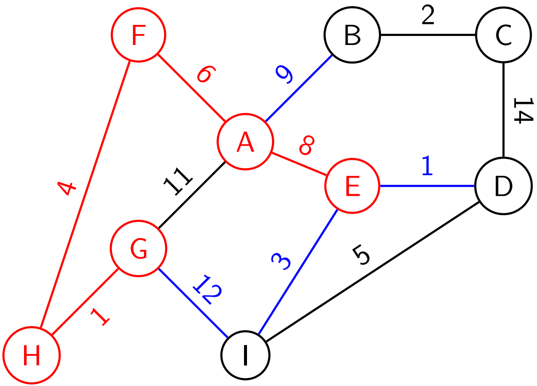


$S = \{F, H, G, A\}$

$S' = \{B, C, D, E, I\}$

- Cost: 11
- Red edges are part of MST and blue edges are connecting two sets
- Red vertices are present in S, rest of the vertices are in S'
- Note: Edge with weight 11 changed from blue to black

Prim's algorithm (cont...)

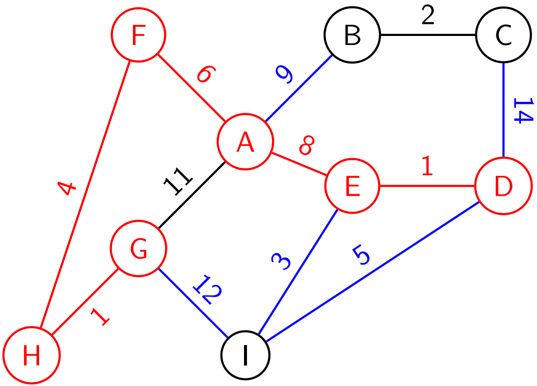


$S = \{F, H, G, A, E\}$

$S' = \{B, C, D, I\}$

- Cost: 19
- Red edges are part of MST and blue edges are connecting two sets
- Red vertices are present in S, rest of the vertices are in S'

Prim's algorithm (cont...)

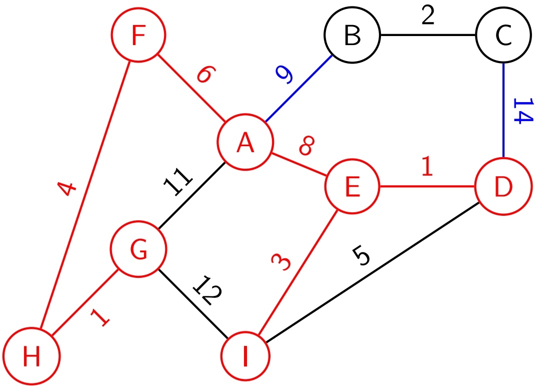


$S = \{F, H, G, A, E, D\}$

$S' = \{B, C, I\}$

- Cost: 20
- Red edges are part of MST and blue edges are connecting two sets
- Red vertices are present in S, rest of the vertices are in S'

Prim's algorithm (cont...)

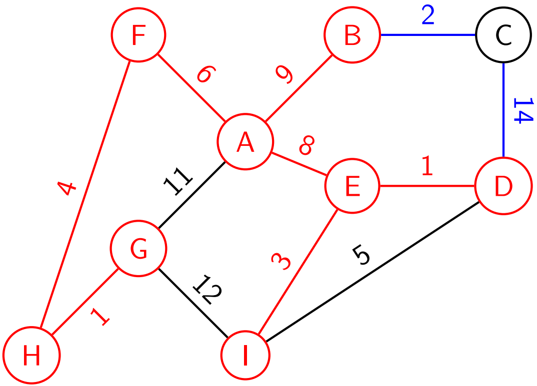


$S = \{F, H, G, A, E, D, I\}$

$S' = \{B, C\}$

- Cost: 23
- Red edges are part of MST and blue edges are connecting two sets
- Red vertices are present in S, rest of the vertices are in S'

Prim's algorithm (cont...)

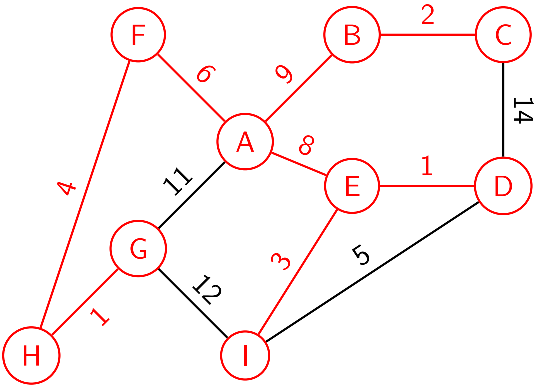


$S = \{F, H, G, A, E, D, I, B\}$

$S' = \{C\}$

- Cost: 32
- Red edges are part of MST and blue edges are connecting two sets
- Red vertices are present in S, rest of the vertices are in S'

Prim's algorithm (cont...)



$S = \{F, H, G, A, E, D, I, B, C\}$

$S' = \{\}$

- Cost: 34
- Red edges are part of MST and blue edges are connecting two sets
- Red vertices are present in S, rest of the vertices are in S'

Prim's algorithm (cont...)

- Prim's algorithm is based on that fact that minimum edge between two sets of a cut will always be part of MST. (Assumption: All edges have distinct weight. In case edges have same weights then one of the two must be part of the MST)
  - Proof? Let us prove it by contradiction. Link to relevant NPTEL video
    - Say at some point while running Prim's algo, there is a minimum edge e connecting two sets of the cut which is not included in MST
    - So if we add edge e to MST then it will create a cycle
    - And that cycle must contain at least one edge (other than edge e) which is connecting the two sets (at that point during execution of prim's algo). Because if no such edge is present in the cycle then how are endpoints of edge e connected in MST? MST must have at least one edge which connects two sets.
    - Now we can remove one edge from the cycle and reduce the cost of spanning tree.
    - Should we remove edge e or the other edge with more weight than edge e (other edge must have more weight than edge e as edge e is *minimum* edge connecting two sets at that time)? We must remove the other edge. Hence we prove that edge e will be part of the MST

Prim's algorithm (cont...)

- How would you implement Prim's algorithm? What data structures are required?
- How to maintain list of edges across sets? How will you update them after inclusion of new edge in to the solution
- Any other data structure that can be used?
- What is overall complexity of your approach?
- By using a *heap* data structure, complexity can be reduced to **E*log(E)**
  - We will learn *heap* data structure in coming days