Minimum Spanning Trees

Paniz Abedin, Assistant Professor of Computer Science,
Florida Polytechnic University

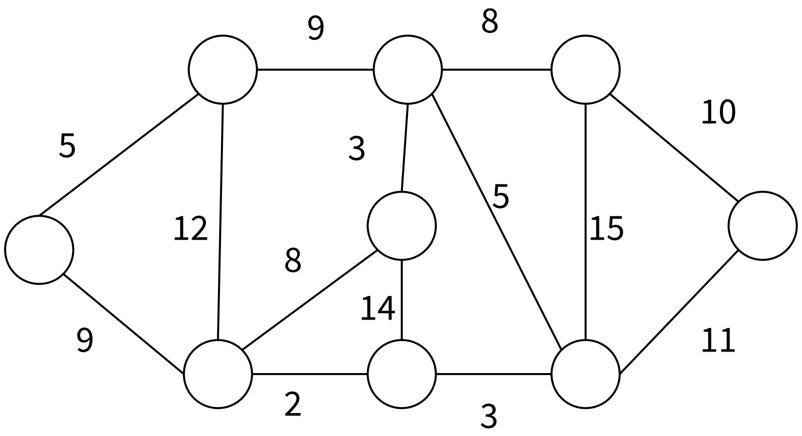
Tree

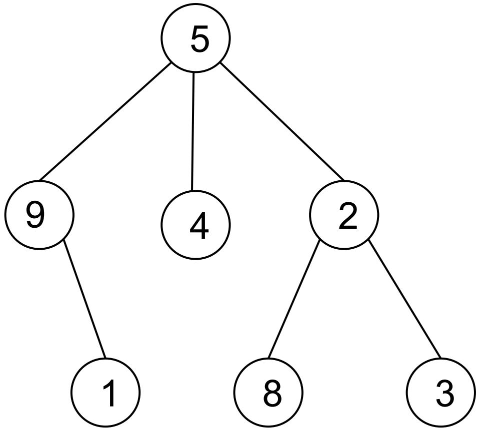
- A connected acyclic graph is called a tree. In other words, a connected graph with no cycles is called a tree.

- A tree with ‘n’ vertices has ‘n-1’ edges. If it has one more edge extra than ‘n-1’, then the extra edge should obviously has to pair up with two vertices which leads to form a cycle. Then, it becomes a cyclic graph which is a violation for the tree graph.

Weighted Graphs/Trees

- Vertices or edges in the graph have assigned weights.





Minimum Spanning Tree

- Spanning Tree: Given an undirected and connected graph **G** = (V, E), a spanning tree of the graph **G** is a tree that spans **G** (that is, it includes every vertex of) and is a subgraph of **G** (every edge in the tree belongs to **G**)

Minimum Spanning Tree

- Spanning Tree: Given an undirected and connected graph **G** = (V, E), a spanning tree of the graph **G** is a tree that spans **G** (that is, it includes every vertex of) and is a subgraph of **G** (every edge in the tree belongs to **G**)

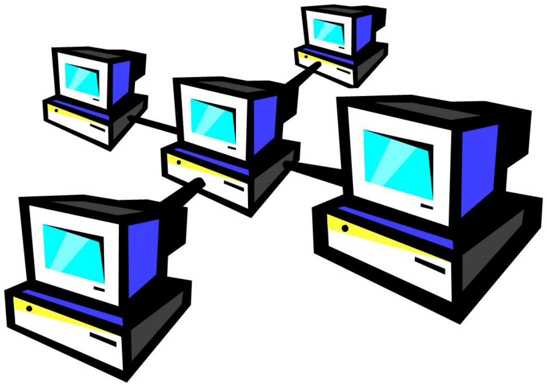
- The cost of the spanning tree is the sum of the weights of all the edges in the tree. There can be many spanning trees.

- Minimum Spanning Tree: is the spanning tree where the cost is minimum among all the spanning trees. There also can be many minimum spanning trees.

- Spanning forest: If a graph is not connected, then there is a spanning tree for each connected component of the graph

Applications of MST

- Constructing highways or railroads spanning several cities
- Laying pipelines connecting offshore drilling sites, refineries and consumer markets.
- Find the least expensive way to connect a set of cities, terminals, computers, etc.

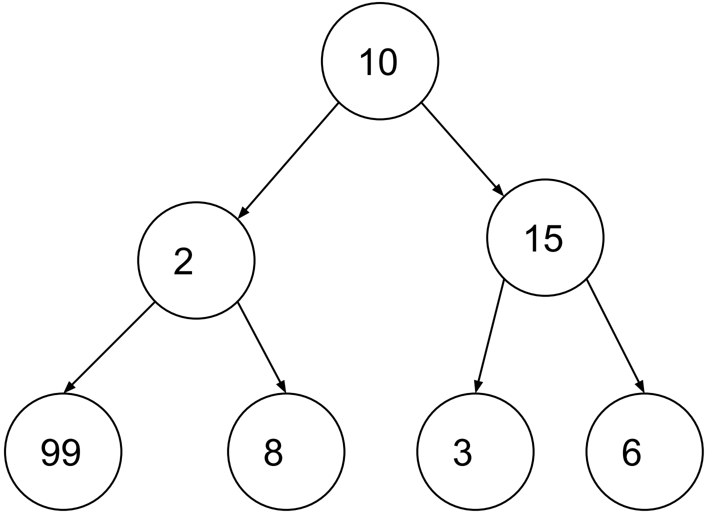


Greedy Algorithms

- A greedy algorithm is a simple, intuitive algorithm that is used in optimization problems.
- The algorithm makes the optimal choice at each step as it attempts to find the overall optimal way to solve the entire problem
- A greedy strategy does not produce an optimal solution

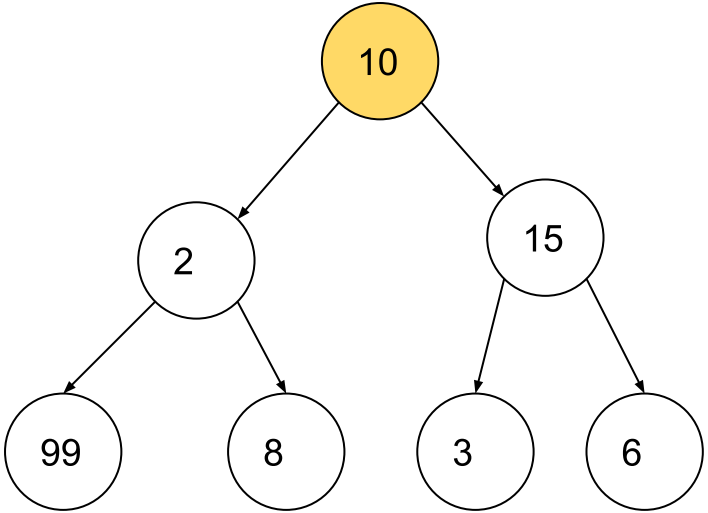
EX

Find the path with the largest sum:



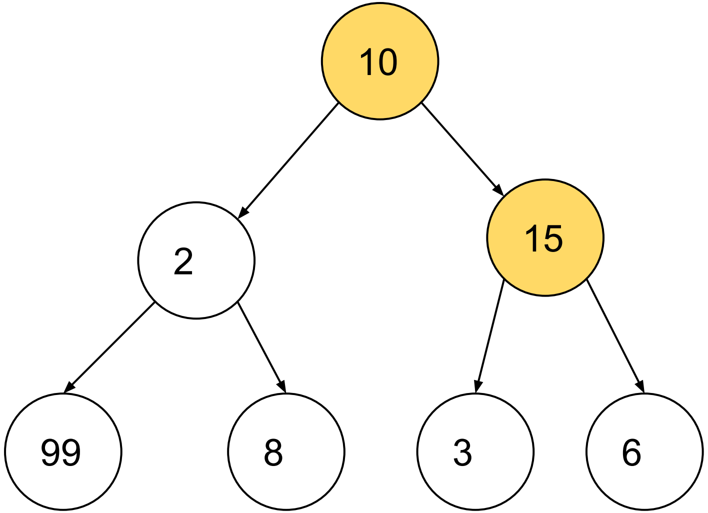
EX

Find the path with the largest sum



EX

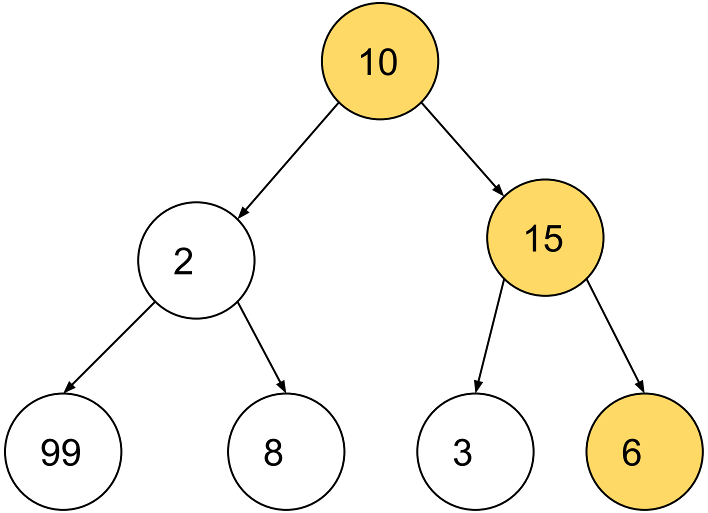
Find the path with the largest sum



Pick greedy!

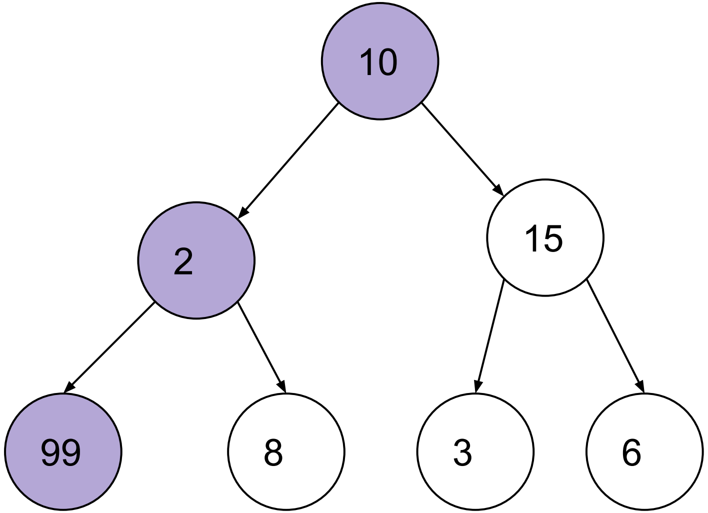
EX

Find the path with the largest sum



EX

Find the path with the largest sum:



Actual answer

Greedy Algorithms

- If both of the properties below are true, a greedy algorithm can be used to solve the problem.
  - Greedy choice property: A global (overall) optimal solution can be reached by choosing the optimal choice at each step.
  - Optimal substructure: A problem has an optimal substructure if an optimal solution to the entire problem contains the optimal solutions to the sub-problems.

Minimum Spanning Trees

Input: A connected, weighted undirected graph:

- A weight $w(u, v)$ on each edge $(u, v) \in E$

Output: A tree T which connects all vertices such that $w(T) = \sum_{(u, v) \in E} W(u, v)$ Is minimized

Kruskal’s Algorithm

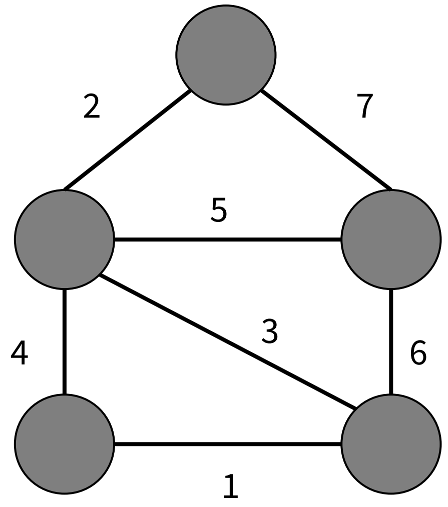
- Kruskal’s Algorithm builds the spanning tree by adding edges one by one into a growing spanning tree. Kruskal's algorithm follows greedy approach as in each iteration it finds an edge which has least weight and add it to the growing spanning tree

Kruskal’s Algorithm

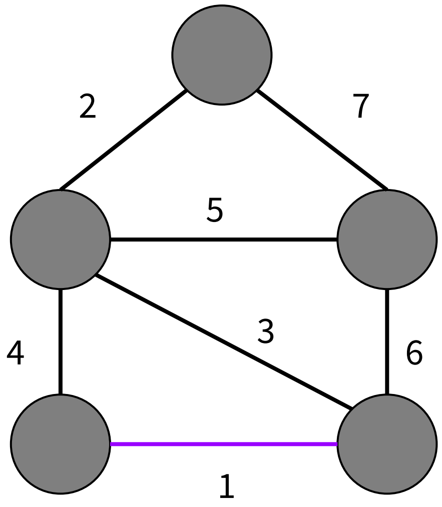
Algorithm:

1. Sort the graph edges with respect to their weights.
2. Start adding edges to the MST from the edge with the smallest weight until the edge of the largest weight.
3. Only add edges which doesn't form a cycle , edges which connect only disconnected components.

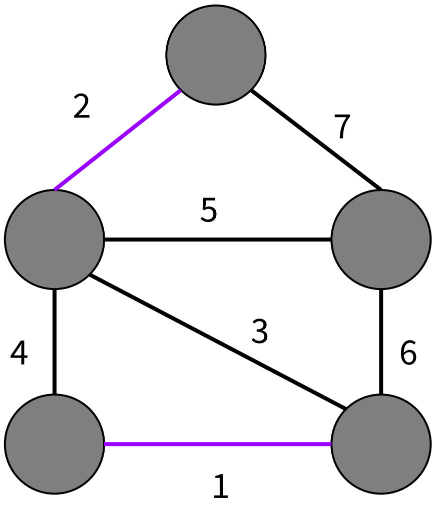
Kruskal's Algorithm



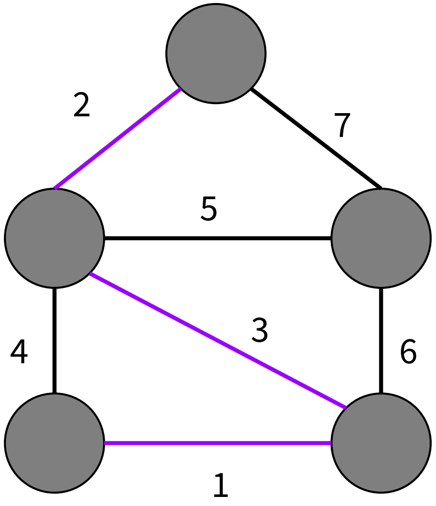
Kruskal’s Algorithm



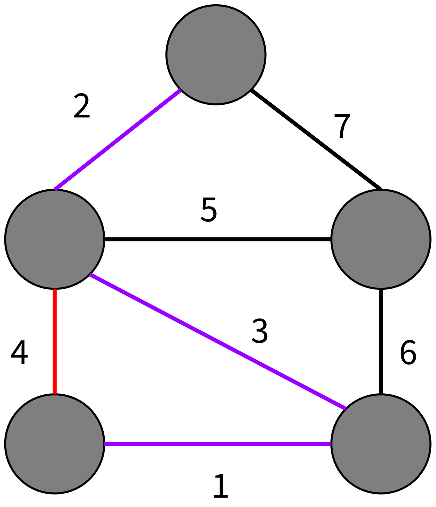
Kruskal's Algorithm



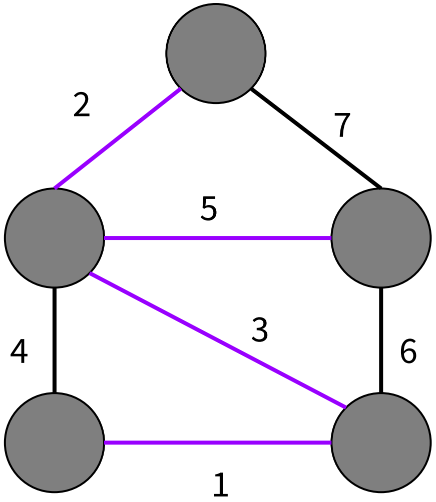
Kruskal's Algorithm



Kruskal’s Algorithm



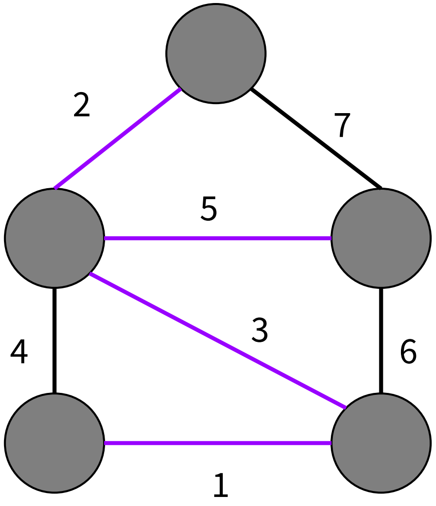
Kruskal's Algorithm



Done!

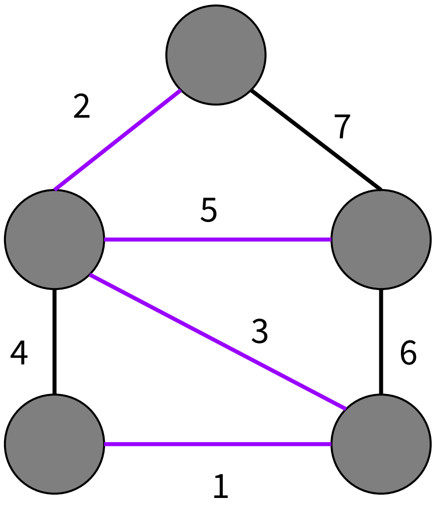
Cost: 11

Kruskal’s Algorithm



Time Complexity?

Kruskal's Algorithm



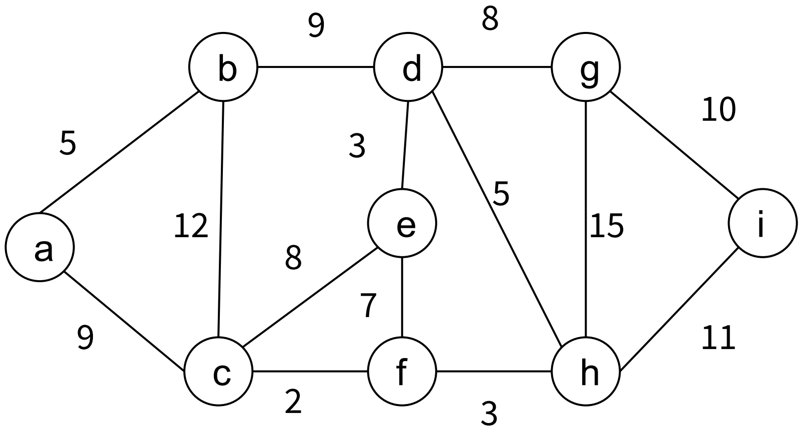
Time Complexity?

O(|E|log|E|)

Prim's Algorithm

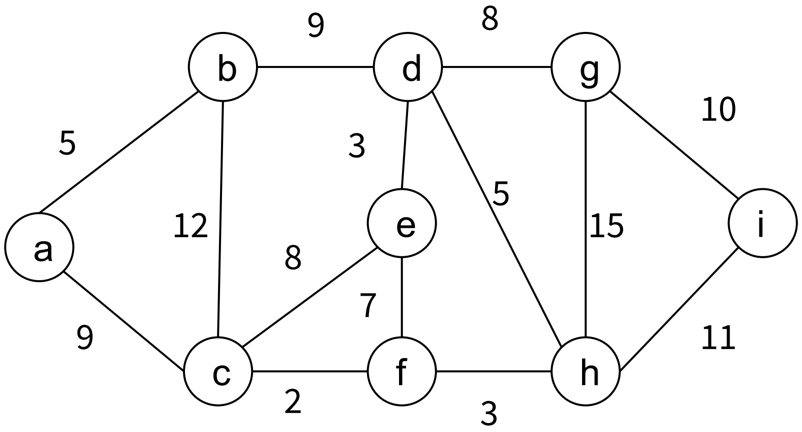
1. Create a set *MSTVertices* that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While *MSTVertices* doesn't include all vertices
   a. Pick a vertex *u* which is not there in *MSTVertices* and has minimum key value.
   b. Include *u* to *MSTVertices* .
   c. Update key value of all adjacent vertices of *u*. To update the key values, iterate through all adjacent vertices. For every adjacent vertex *v*, if weight of edge *u*-*v* is less than the previous key value of *v*, update the key value as weight of *u*-*v*

Prim’s Algorithm



Prim’s Algorithm

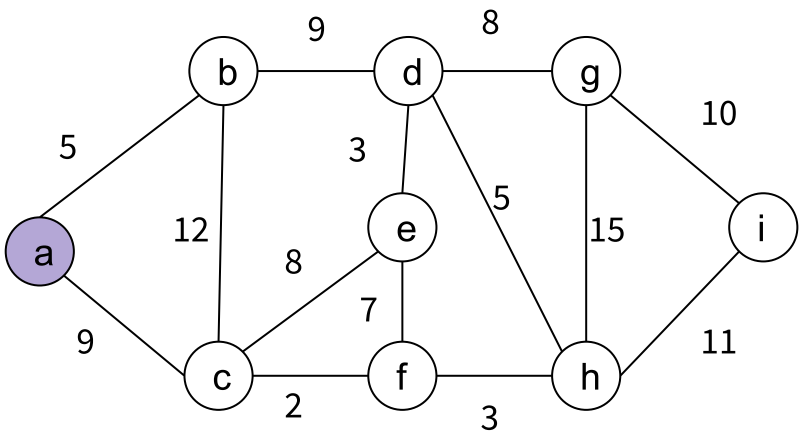
MSTVertices={}
source : a



key	a	b	c	d	e	f	g	h	i
value	0	∞	∞	∞	∞	∞	∞	∞	∞

Prim’s Algorithm

MSTVertices={a}
source : a



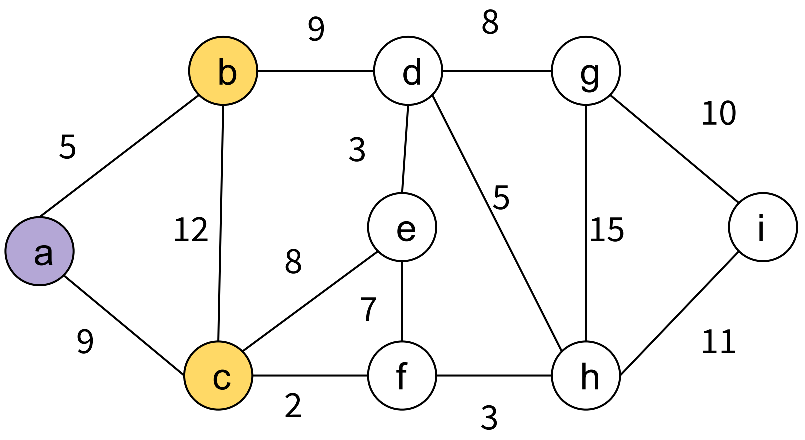
key	a	b	c	d	e	f	g	h	i
value	0	∞	∞	∞	∞	∞	∞	∞	∞

While MSTVertices doesn’t include all vertices

- Pick a vertex u which is not there in MSTVertices and has minimum key value
- Include u to MSTVertices .
- Update key value of all adjacent vertices of u. To update the key values, iterate through all adjacent vertices. For every adjacent vertex v, if weight of edge u-v is less than the previous key value of v, update the key value as weight of u-v

Prim's Algorithm

MSTVertices={a}
source : a



While MSTVertices doesn't include all vertices

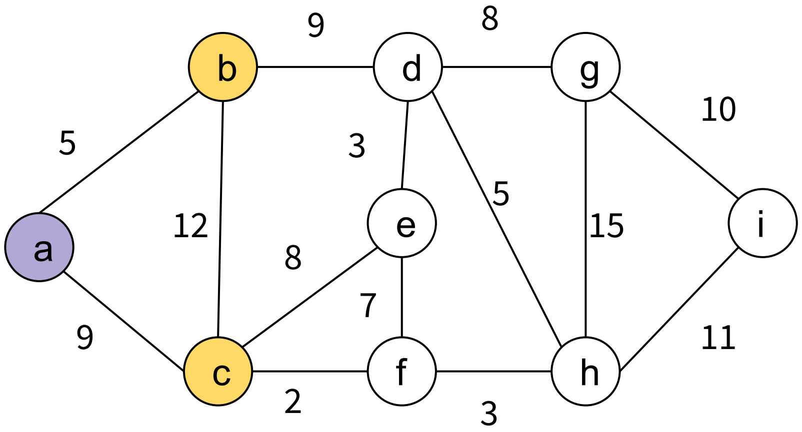
- Pick a vertex u which is not there in MSTVertices and has minimum key value.
- Include u to MSTVertices .
- Update key value of all adjacent vertices of u.

If w(u,v) < v.key
v.key= w(u,v)

key	a	b	c	d	e	f	g	h	i
value	0	∞	∞	∞	∞	∞	∞	∞	∞

Prim's Algorithm

MSTVertices={a}
source : a



While MSTVertices doesn't include all vertices

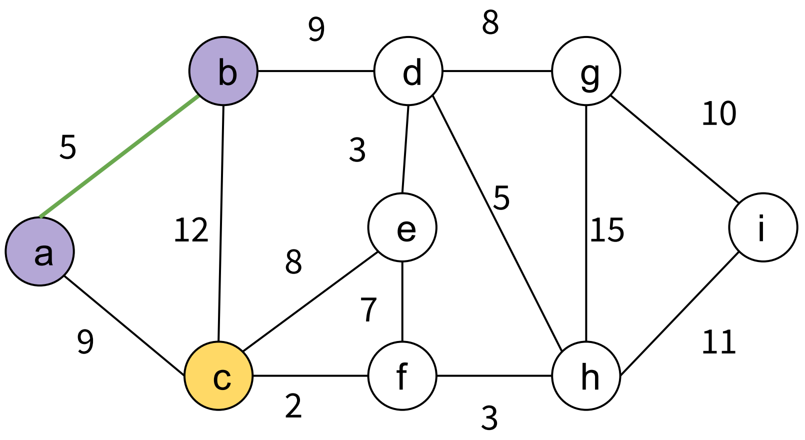
- Pick a vertex u which is not there in MSTVertices and has minimum key value.
- Include u to MSTVertices .
- Update key value of all adjacent vertices of u.

If w(u,v) < v.key
    v.key= w(u,v)

key	a	b	c	d	e	f	g	h	i
value	0	5	9	∞	∞	∞	∞	∞	∞

Prim's Algorithm

MSTVertices={a, b}
source : a



key	a	b	c	d	e	f	g	h	i
value	0	5	9	∞	∞	∞	∞	∞	∞

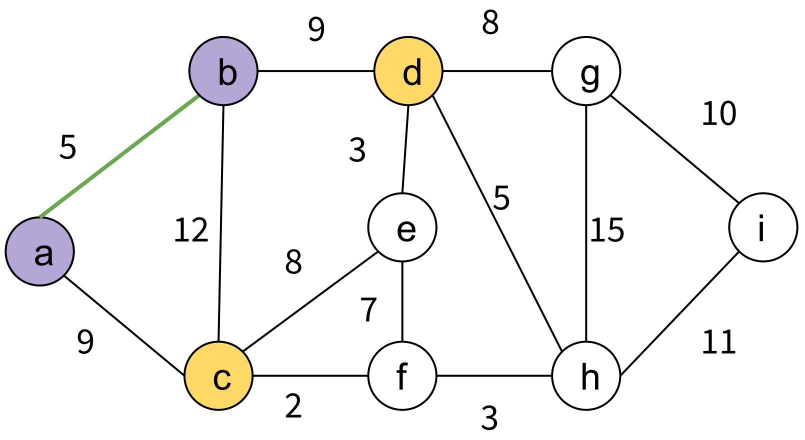
While MSTVertices doesn't include all vertices

- Pick a vertex u which is not there in MSTVertices and has minimum key value.
- Include u to MSTVertices .
- Update key value of all adjacent vertices of u.

If w(u,v) < v.key
    v.key= w(u,v)

Prim's Algorithm

MSTVertices={a, b}
source : a



While MSTVertices doesn't include all vertices

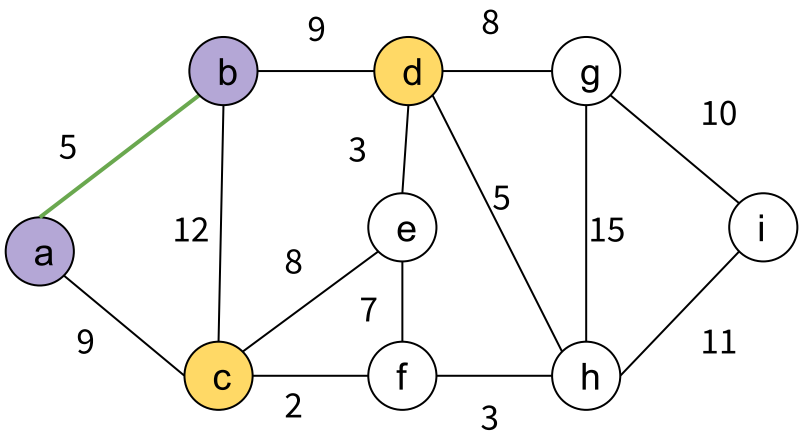
- Pick a vertex u which is not there in MSTVertices and has minimum key value.
- Include u to MSTVertices .
- Update key value of all adjacent vertices of u.

If w(u,v) < v.key
    v.key= w(u,v)

key	a	b	c	d	e	f	g	h	i
value	0	5	9	9	∞	∞	∞	∞	∞

Prim's Algorithm

MSTVertices={a, b}
source : a



key	a	b	c	d	e	f	g	h	i
value	0	5	9	9	∞	∞	∞	∞	∞

While MSTVertices doesn't include all vertices

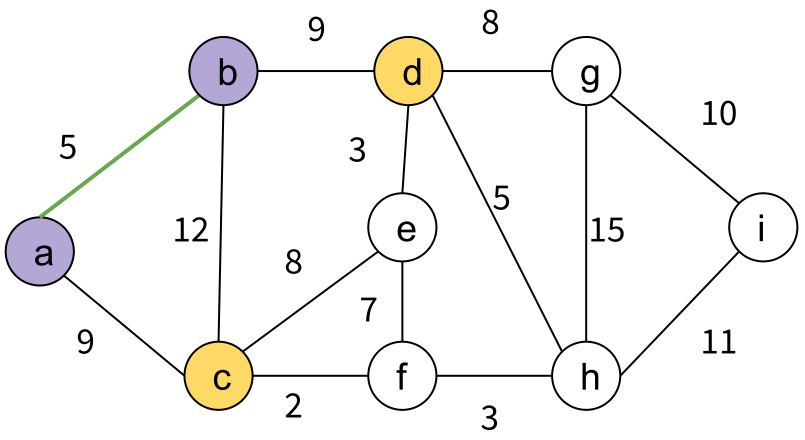
- Pick a vertex u which is not there in MSTVertices and has minimum key value.
- Include u to MSTVertices .
- Update key value of all adjacent vertices of u.

If w(u,v) < v.key
v.key= w(u,v)

v(c) won't be updated because 12 > 9

Prim’s Algorithm

MSTVertices={a, b}
source : a



While MSTVertices doesn’t include all vertices

- Pick a vertex u which is not there in MSTVertices and has minimum key value.
- Include u to MSTVertices .
- Update key value of all adjacent vertices of u.

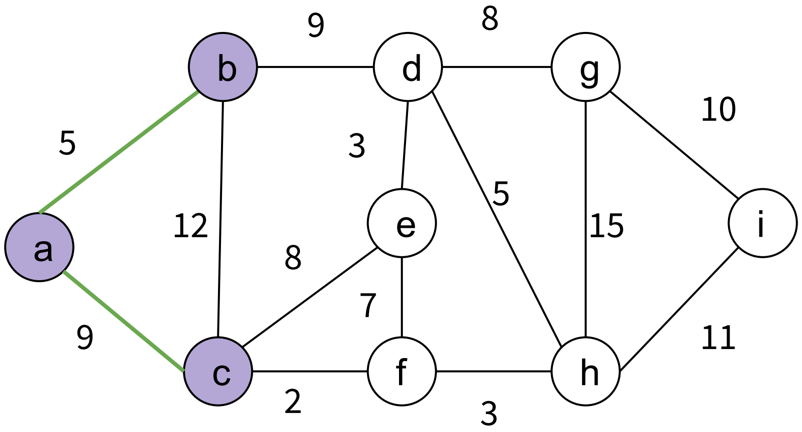
If w(u,v) < v.key
v.key= w(u,v)

key	a	b	c	d	e	f	g	h	i
value	0	5	9	9	∞	∞	∞	∞	∞

Pick either c or d and add it to MSTvertices

Prim's Algorithm

MSTVertices={a, b, c}
source : a



While MSTVertices doesn't include all vertices

- Pick a vertex u which is not there in MSTVertices and has minimum key value.
- Include u to MSTVertices .
- Update key value of all adjacent vertices of u.

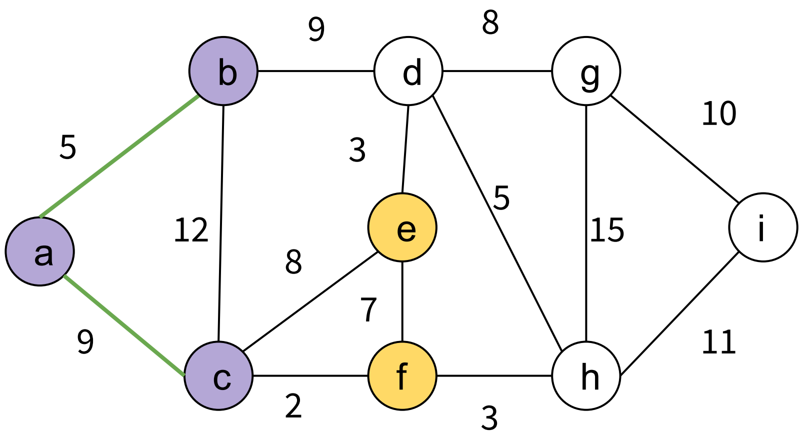
If w(u,v) < v.key
v.key= w(u,v)

key	a	b	c	d	e	f	g	h	i
value	0	5	9	9	∞	∞	∞	∞	∞

Pick either c or d and add it to MSTvertices

Prim's Algorithm

MSTVertices={a, b, c}
source : a



While MSTVertices doesn't include all vertices

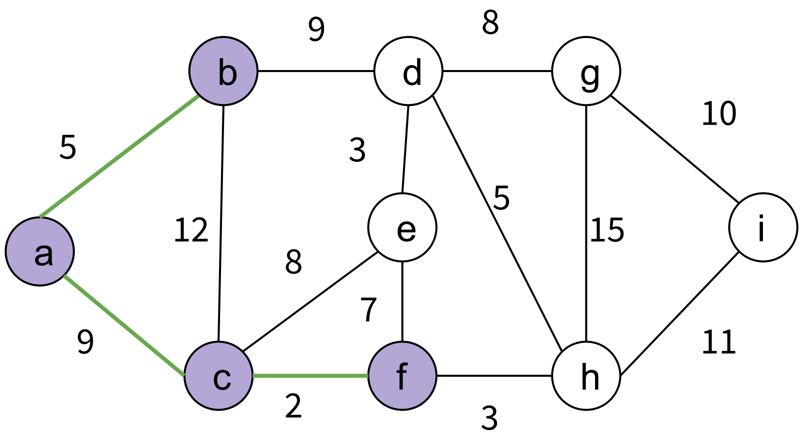
- Pick a vertex u which is not there in MSTVertices and has minimum key value.
- Include u to MSTVertices .
- Update key value of all adjacent vertices of u.

If w(u,v) < v.key
  v.key= w(u,v)

key	a	b	c	d	e	f	g	h	i
value	0	5	9	9	8	2	∞	∞	∞

Prim’s Algorithm

MSTVertices={a, b, c, f}
source : a



While MSTVertices doesn’t include all vertices

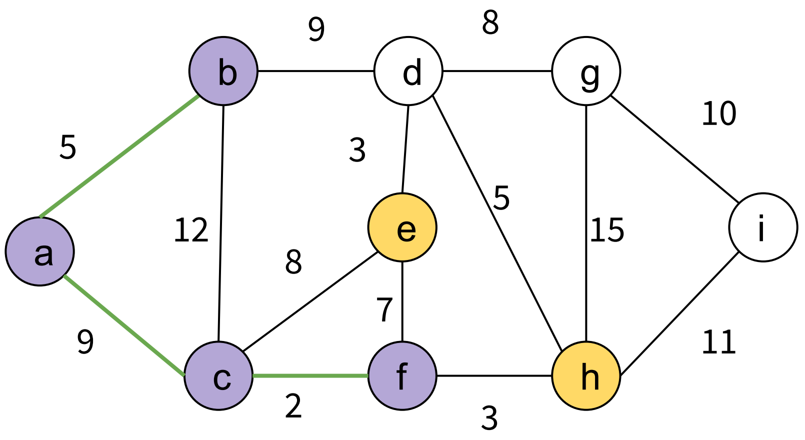
- Pick a vertex u which is not there in MSTVertices and has minimum key value.
- Include u to MSTVertices .
- Update key value of all adjacent vertices of u.

If w(u,v) < v.key
v.key= w(u,v)

key	a	b	c	d	e	f	g	h	i
value	0	5	9	9	8	2	∞	∞	∞

Prim’s Algorithm

MSTVertices={a, b, c, f}
source : a



While MSTVertices doesn’t include all vertices

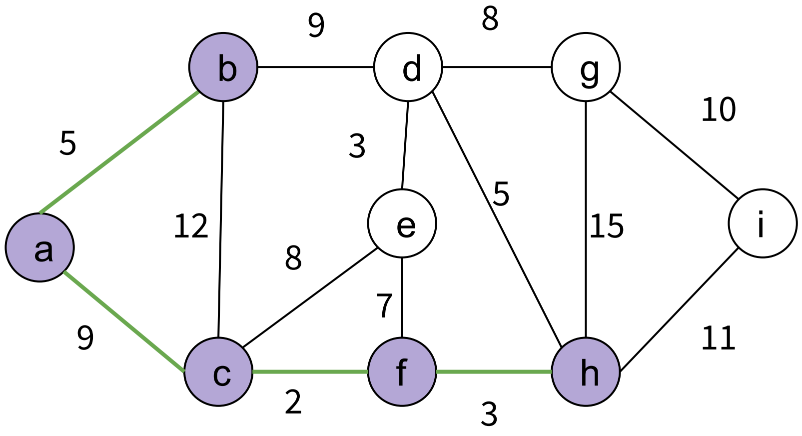
- Pick a vertex u which is not there in MSTVertices and has minimum key value.
- Include u to MSTVertices .
- Update key value of all adjacent vertices of u.

If w(u,v) < v.key
v.key= w(u,v)

key	a	b	c	d	e	f	g	h	i
value	0	5	9	9	7	2	∞	3	∞

Prim's Algorithm

MSTVertices={a, b, c, f, h}
source : a



While MSTVertices doesn't include all vertices

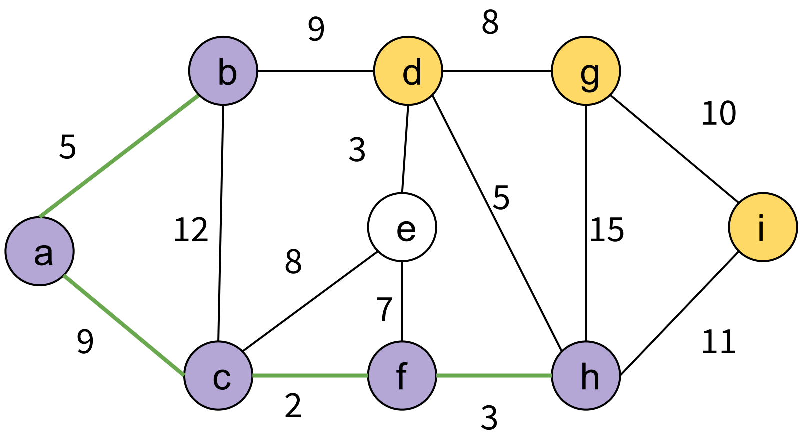
- Pick a vertex u which is not there in MSTVertices and has minimum key value.
- Include u to MSTVertices .
- Update key value of all adjacent vertices of u.

If w(u,v) < v.key
v.key= w(u,v)

key	a	b	c	d	e	f	g	h	i
value	0	5	9	9	7	2	∞	3	∞

Prim's Algorithm

MSTVertices={a, b, c, f, h}
source : a



While MSTVertices doesn't include all vertices

- Pick a vertex u which is not there in MSTVertices and has minimum key value.
- Include u to MSTVertices .
- Update key value of all adjacent vertices of u.

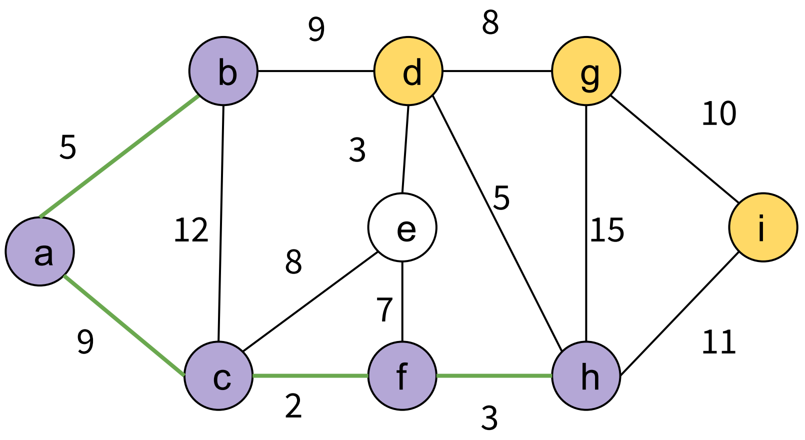
If w(u,v) < v.key
 v.key= w(u,v)

key	a	b	c	d	e	f	g	h	i
value	0	5	9	9	7	2	∞	3	∞

Update
value of d!

Prim's Algorithm

MSTVertices={a, b, c, f, h}
source : a



While MSTVertices doesn't include all vertices

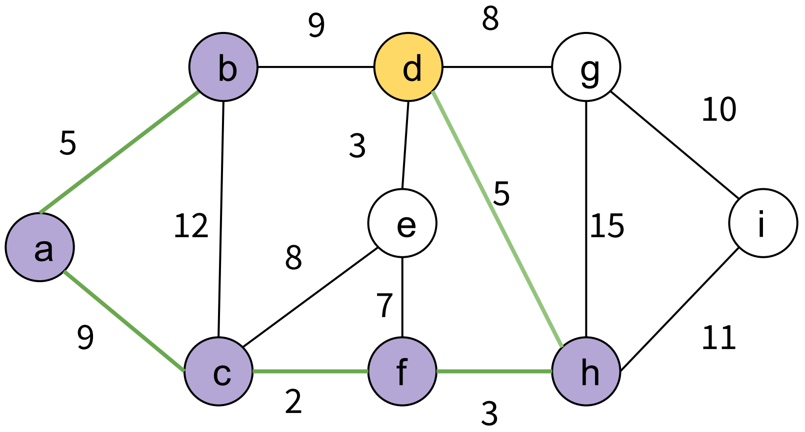
- Pick a vertex u which is not there in MSTVertices and has minimum key value.
- Include u to MSTVertices .
- Update key value of all adjacent vertices of u.

If w(u,v) < v.key
v.key= w(u,v)

key	a	b	c	d	e	f	g	h	i
value	0	5	9	5	7	2	15	3	11

Prim’s Algorithm

MSTVertices={a, b, c, f, h}
source : a



While MSTVertices doesn’t include all vertices

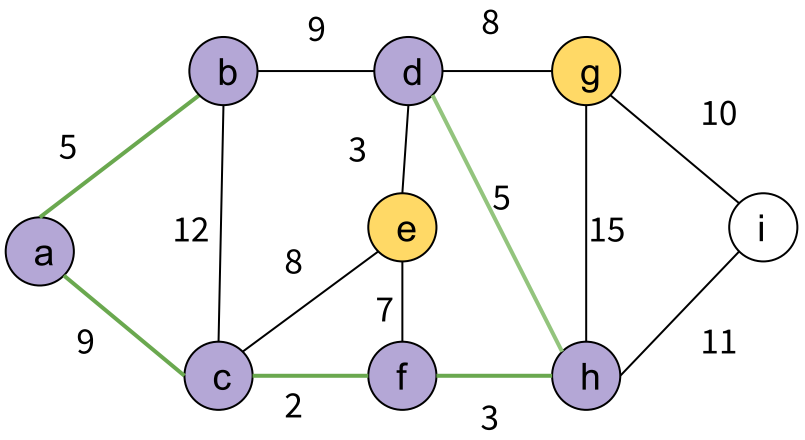
- Pick a vertex u which is not there in MSTVertices and has minimum key value.
- Include u to MSTVertices .
- Update key value of all adjacent vertices of u.

If w(u,v) < v.key
    v.key= w(u,v)

key	a	b	c	d	e	f	g	h	i
value	0	5	9	5	7	2	15	3	11

Prim's Algorithm

MSTVertices={a, b, c, f, h, d}
source : a



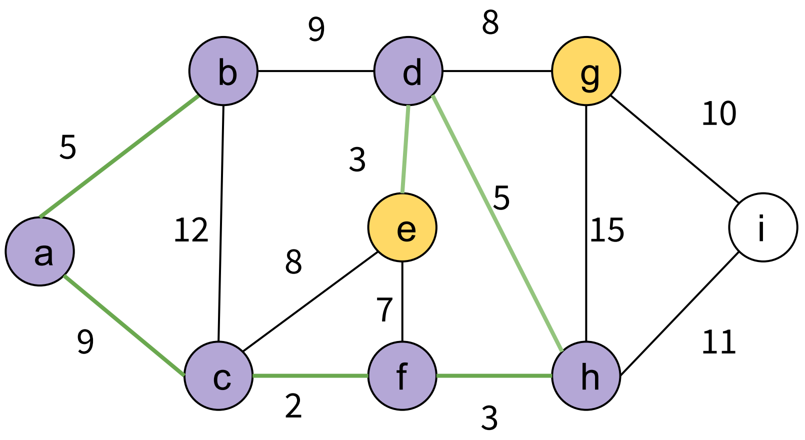
While MSTVertices doesn't include all vertices
- Pick a vertex u which is not there in MSTVertices and has minimum key value.
- Include u to MSTVertices .
- Update key value of all adjacent vertices of u.

If w(u,v) < v.key
v.key= w(u,v)

key	a	b	c	d	e	f	g	h	i
value	0	5	9	5	7	2	15	3	11

Prim's Algorithm

MSTVertices={a, b, c, f, h, d}
source : a



key	a	b	c	d	e	f	g	h	i
value	0	5	9	5	3	2	8	3	11

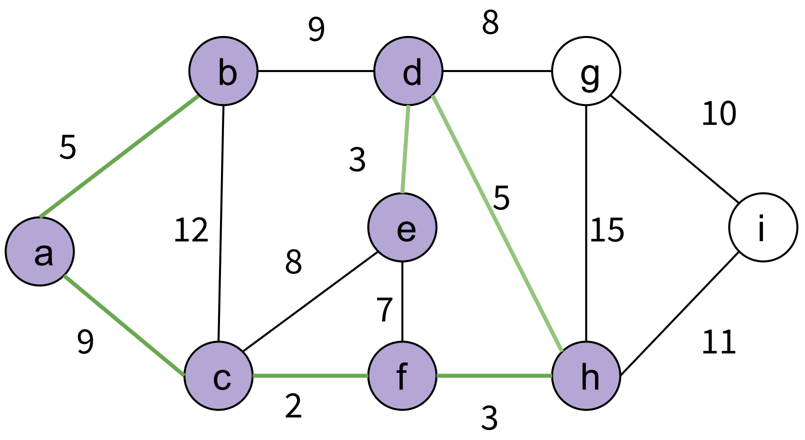
While MSTVertices doesn't include all vertices

- Pick a vertex u which is not there in MSTVertices and has minimum key value.
- Include u to MSTVertices .
- Update key value of all adjacent vertices of u.

If w(u,v) < v.key
    v.key= w(u,v)

Prim's Algorithm

MSTVertices={a, b, c, f, h, d, e}
source : a



While MSTVertices doesn't include all vertices

- Pick a vertex u which is not there in MSTVertices and has minimum key value.
- Include u to MSTVertices .
- Update key value of all adjacent vertices of u.

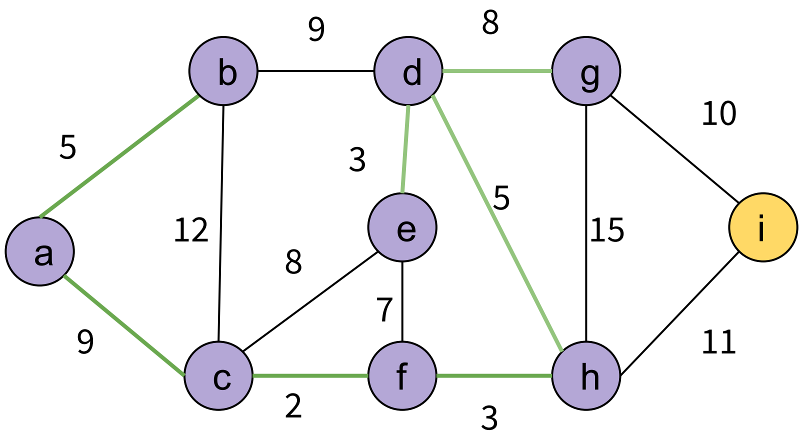
If w(u,v) < v.key
v.key= w(u,v)

key	a	b	c	d	e	f	g	h	i
value	0	5	9	5	3	2	8	3	11

Pick the next min

Prim's Algorithm

MSTVertices={a, b, c, f, h, d, e, g}
source : a



While MSTVertices doesn't include all vertices

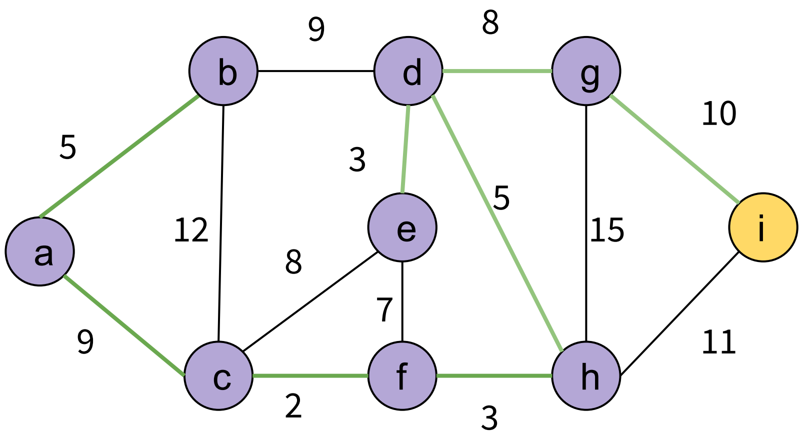
- Pick a vertex u which is not there in MSTVertices and has minimum key value.
- Include u to MSTVertices .
- Update key value of all adjacent vertices of u.

If w(u,v) < v.key
v.key= w(u,v)

key	a	b	c	d	e	f	g	h	i
value	0	5	9	5	3	2	8	3	11

Prim’s Algorithm

MSTVertices={a, b, c, f, h, d, e, g, i}
source : a



While MSTVertices doesn’t include all vertices

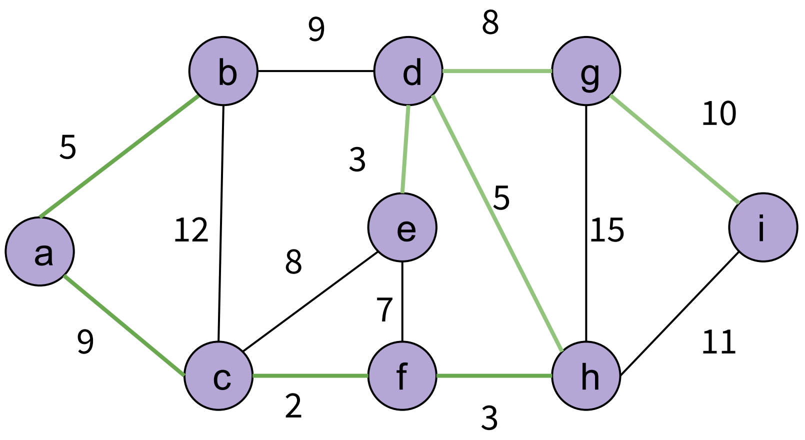
- Pick a vertex u which is not there in MSTVertices and has minimum key value.
- Include u to MSTVertices .
- Update key value of all adjacent vertices of u.

If w(u,v) < v.key
v.key= w(u,v)

key	a	b	c	d	e	f	g	h	i
value	0	5	9	5	3	2	8	3	10

Prim's Algorithm

MSTVertices={a, b, c, f, h, d, e, g, i}
source : a

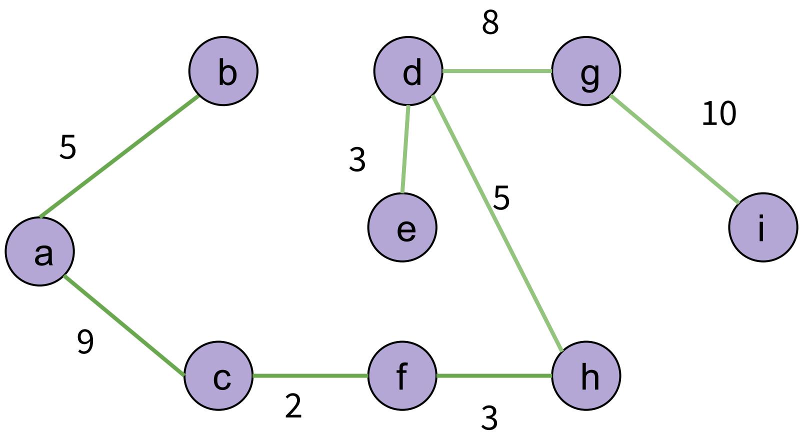


Every node is in MSTVertices set.
Done!

key	a	b	c	d	e	f	g	h	i
value	0	5	9	5	3	2	8	3	10

Prim's Algorithm

MSTVertices={a, b, c, f, h, d, e, g, i}
source : a

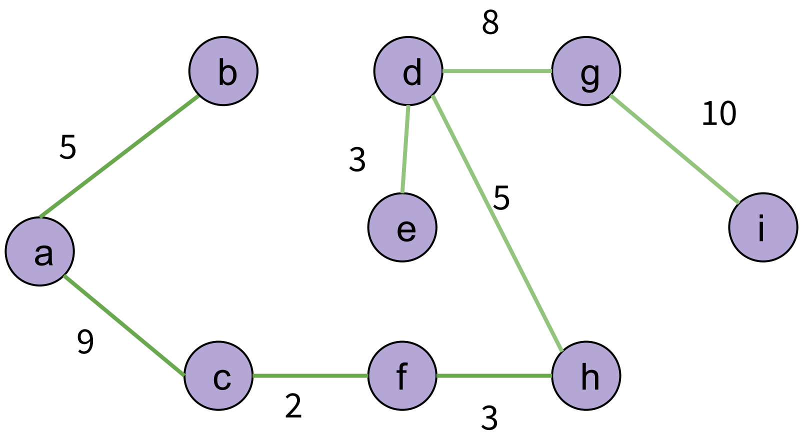


Total weight: 45

key	a	b	c	d	e	f	g	h	i
value	0	5	9	5	3	2	8	3	10

Prim's Algorithm

MSTVertices={a, b, c, f, h, d, e, g, i}
source : a

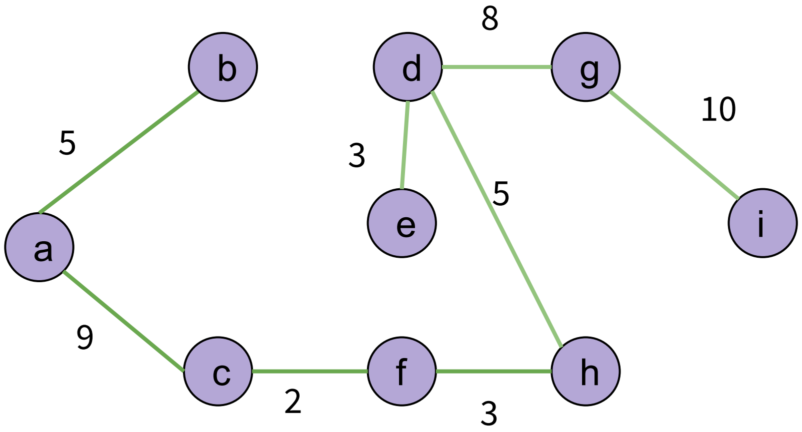


Time Complexity?

key	a	b	c	d	e	f	g	h	i
value	0	5	9	5	3	2	8	3	10

Prim's Algorithm

MSTVertices={a, b, c, f, h, d, e, g, i}
source : a



Time Complexity?

O(ElogV)

key	a	b	c	d	e	f	g	h	i
value	0	5	9	5	3	2	8	3	10

Prim Complexity

- It can be implemented efficiently using Binary Heap/Priority Queue H:
- First, insert all edges adjacent to u into H
- At each step, extract the cheapest edge
- If an end-point(vertex), say v, is not in MST, include this edge and v to MST
- Insert all edges adjacent to v into H
- At most O(E) Insert/Extract-Min
- Total Time: O(E log V)

- Kruskal algorithm can be implemented similarly : O(E log V)

Priority Queue Implementation using STL

- A priority queue is a container adaptor that provides constant time lookup of the largest (by default) element, at the expense of logarithmic insertion and extraction.
- #include <queue>
- Syntax of Priority Queue:
    - priority_queue<int> variableName;
- Syntax to create min-heap for the Priority Queue:
    - priority_queue <int, vector<int>, greater<int>> q;

Where vector<int> is a STL container and greater<int> is comparator class.

Priority Queue Implementation using STL

Inserting elements in a Priority Queue:

```
#include<iostream>
#include<queue>
using namespace std;

int main()
{
	priority_queue<int> pq;
	pq.push(40);
	pq.push(30);
	pq.push(90);
	while (!pq.empty())
	{
		cout << ' ' << pq.top();
		pq.pop();
	}
}
```

Priority Queue Implementation using STL

Inserting elements in a Priority Queue:

```
#include<iostream>
#include<queue>
using namespace std;

int main()
{
        priority_queue<int> pq;
        pq.push(40);
        pq.push(30);
        pq.push(90);
        while (!pq.empty())
        {
                cout << ' ' << pq.top();
                pq.pop();
        }
}
```

Output:
90 40 30

MinHeap Implementation using STL

```
#include<iostream>
#include<queue>

using namespace std;
int main()
{
        // creates a min heap
        priority_queue <int, vector<int>, greater<int> > pq;

          pq.push(45);
          pq.push(10);
          pq.push(72);
          pq.push(40);
          pq.push(32);

          while (!pq.empty())
          {
                    cout <<"  "<< pq.top() << " ";
                    pq.pop();
          }

   return 0;
}
```

Output:
10 32 40 45 72

Priority Queue Implementation using STL

Inserting elements in a Priority Queue:

```
#include<iostream>
#include<queue>
using namespace std;

int main()
{
	priority_queue<int> pq;
	pq.push(40);
	pq.push(30);
	pq.push(90);
	while (!pq.empty())
	{
		cout << ' ' << pq.top();
		pq.pop();
	}
}
```

Prim Implementation using STL

- It can be done using priority queue

```
class Graph
{
    // Number. of vertices
    int V;

   // In a weighted graph, we need to store vertex
   // and weight pair for every edge
    list< pair<int, int> > *adj;

public:
    Graph(int V);  // Constructor

   // function to add an edge to graph
    void addEdge(int u, int v, int w);

   // Print MST using Prim's algorithm
    void primMST();
};
```

Prim Implementation using STL

- It can be done using priority queue

```
class Graph
{
   int V;
    list< pair<int, int> > *adj;

public:
   Graph(int V);
    void addEdge(int u, int v, int w);
    void primMST();
};
```

- We need to create a priority queue to store vertices that are being primMST

```
priority_queue< pair<int, int>, vector <pair<int, int>> , greater<pair<int, int>> >   pq;
```

- Create a vector for keys and initialize all keys as infinite (INF)
  vector<int> key(V, INF);

- Can you think of the implementation of the algorithm using this?