香港中文大學（深圳）
The Chinese University of Hong Kong, Shenzhen

Tutorial 10
Minimum Spanning Tree (MST)

CSC 3100 Data Structures

Xiaowen Shao

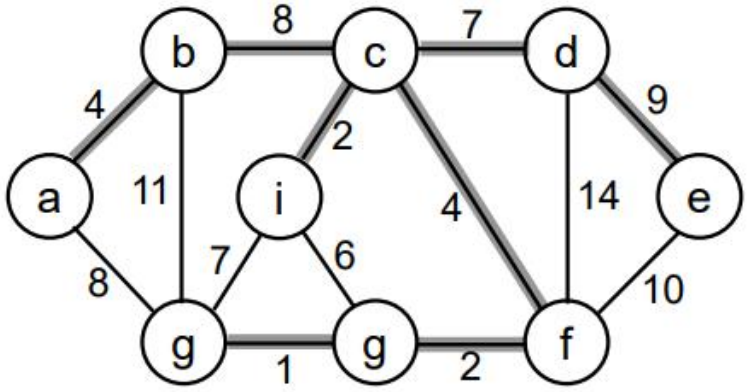
Email:119010258@link.cuhk.edu.cn

November 30, 2021

Minimum Spanning Trees

- Spanning Tree
- A tree (i.e., connected, acyclic graph) which contains all the vertices of the graph

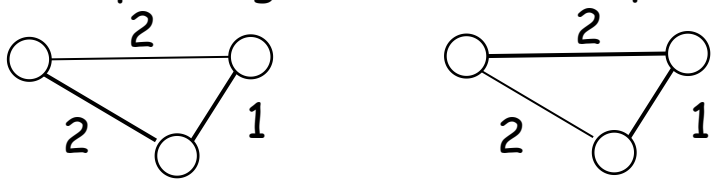
- Minimum Spanning Tree (MST)
- Spanning tree with the minimum sum of weights



- Spanning forest
- If a graph is not connected, then there is an MST for each connected component of the graph.

Properties of MSTs

- Minimum Spanning Tree is not unique

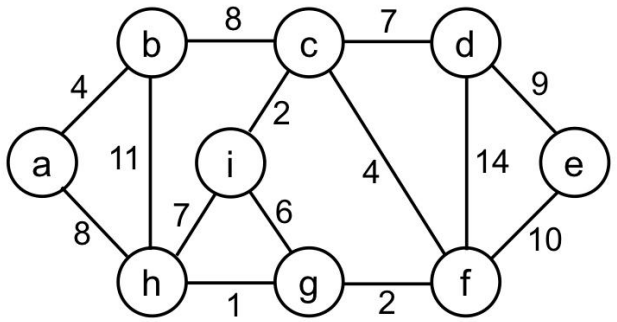


- MST has no cycles (why?)
  - We can take out an edge of a cycle, and still have the vertices connected while reducing the cost.

- # of edges in a MST:
  - |V| -1

Generic MST Algorithm

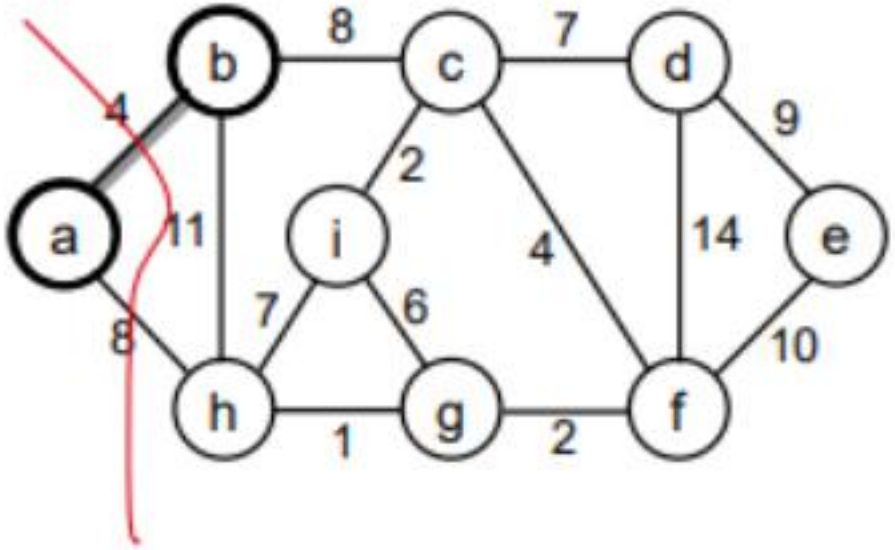
1. $A \leftarrow \varnothing$
2. **while** A is not a spanning tree
3. **do** find an edge (u, v) that is safe for A
4. $A \leftarrow A \cup \{(u, v)\}$
5. **return** A



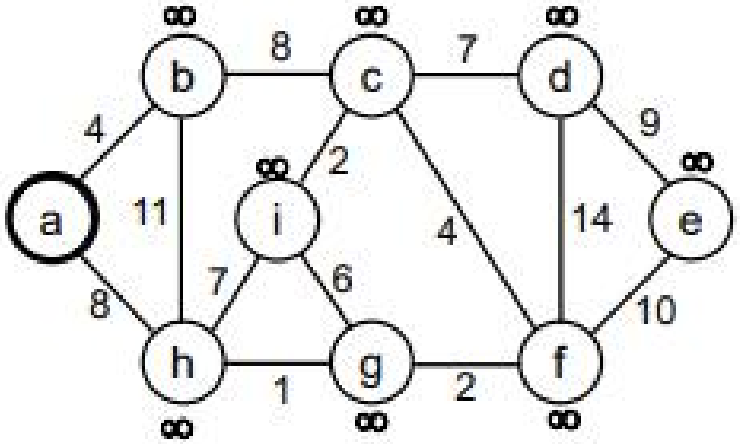
How do we find safe edges?

Prim's Algorithm

- **Prim's Algorithm to generate MST**
  - Initialize an empty set A to store MST edges
  - Starts from an arbitrary "root": $V_A = \{a\}$
  - At each step:
    - Find a light edge crossing cut $(V - V_A)$
    - Add this edge to A
    - Repeat until the tree spans all vertices



Review: Prim's Algorithm

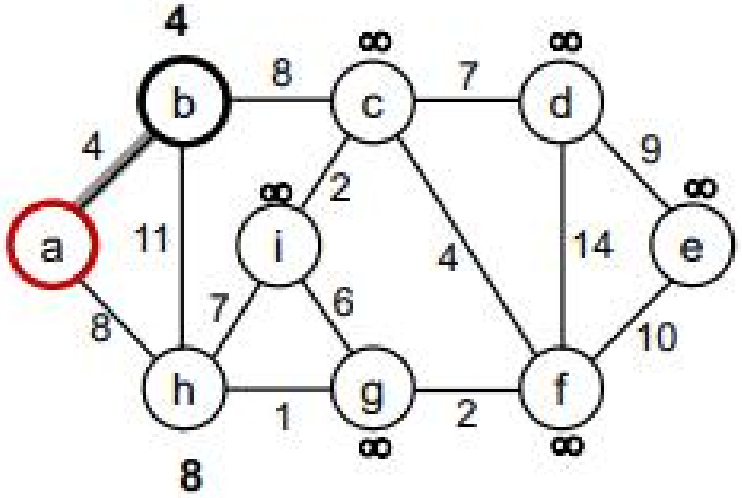


0 ∞ ∞ ∞ ∞ ∞ ∞ ∞ ∞

Q = {a, b, c, d, e, f, g, h, i}

$V_A = \varnothing$

Extract-MIN(Q) ⇒ a



key [b] = 4 π [b] = a

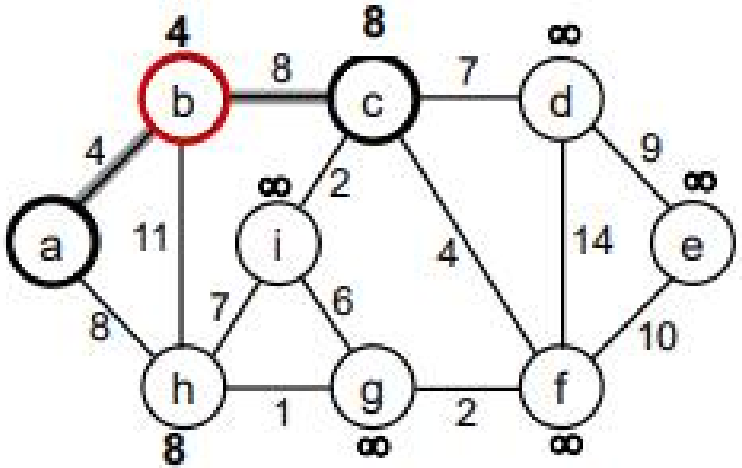
key [h] = 8 π [h] = a

4 ∞ ∞ ∞ ∞ ∞ **8** ∞

Q = {b, c, d, e, f, g, h, i} V_A = {a}

Extract-MIN(Q) ⇒ b

Review: Prim's Algorithm



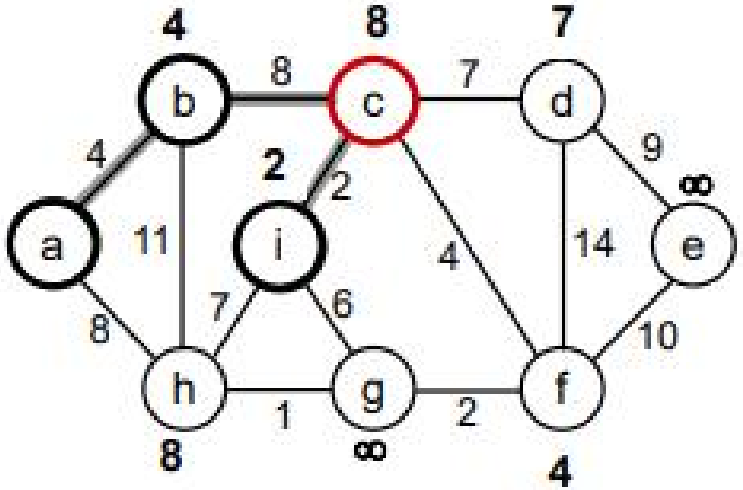
key [c] = 8 π [c] = b

key [h] = 8 π [h] = a - unchanged

8 ∞ ∞ ∞ ∞ 8 ∞

Q = {c, d, e, f, g, h, i} V_A = {a, b}

Extract-MIN(Q) ⇒ c



key [d] = 7 π [d] = c

key [f] = 4 π [f] = c

key [i] = 2 π [i] = c

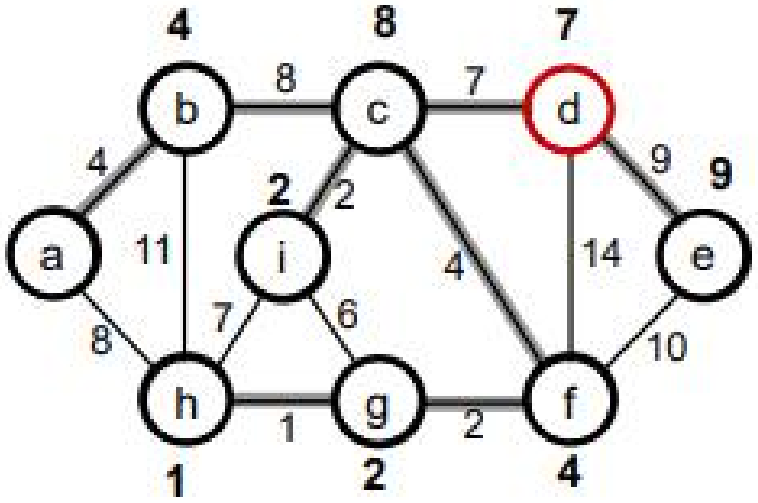
7 ∞ 4 ∞ 8 2

Q = {d, e, f, g, h, i} V_A = {a, b, c}

Extract-MIN(Q) ⇒ i

Review: Prim's Algorithm

- Repeat previous operations until we span all vertices



key [e] = 9 π [e] = d

9

Q = {e} V_A = {a, b, c, i, f, g, h, d}

Extract-MIN(Q) $\Rightarrow$ e

Q = $\varnothing$ V_A = {a, b, c, i, f, g, h, d, e}

Prim's Algorithm: Java Example

```
3  public class Prim {
4
5      int n;
6      int [][] adjMatrix;
7      public void addEdge(int v1, int v2, int weight){
12     public void initGraph(){
36
37     public void prim(){
74
75     public static void main(String[] args) {
76         Prim prim = new Prim();
77         prim.initGraph();
78         prim.prim();
79     }
80 }
81
```

- **Member variable**
  - **n** : number of vertices
  - **adjMatrix** : adjacent matrix

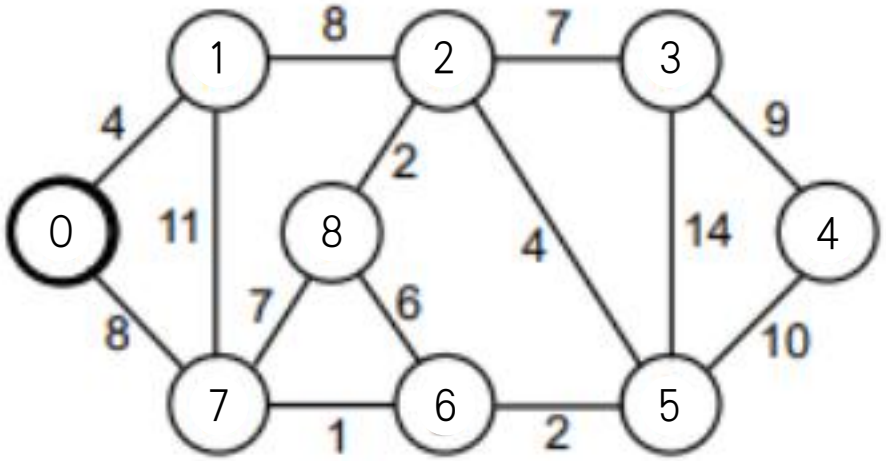
- **Member method**
  - **addEdge** : adding edge (v1,v2,weight) to the adjacent matrix
  - **initGraph** : construct the graph
  - **prim** : Prim's algorithm to generate MST for the graph constructed by initGraph

Prim's Algorithm: Java Example

```
7   public static void addEdge(int v1, int v2, int weight){
8
9       adjMatrix[v1][v2] = weight;
10      adjMatrix[v2][v1] = weight;
11  }
12  public static void initGraph(){
13      n = 9; // the number of vertices
14      adjMatrix = new int[n][n]; //adjacent matrix
15      for(int i = 0; i < n; i++){
16          for(int j = 0; j < n; j++)
17              adjMatrix[i][j] = 999;
18      }
19      //add edges
20      addEdge(0,1,4);
21      addEdge(0,7,8);
22      addEdge(1,2,8);
23      addEdge(1,7,11);
24      addEdge(2,3,7);
25      addEdge(2,5,4);
26      addEdge(2,8,2);
27      addEdge(3,4,9);
28      addEdge(3,5,14);
29      addEdge(4,5,10);
30      addEdge(5,6,2);
31      addEdge(6,7,1);
32      addEdge(6,8,6);
33      addEdge(7,8,7);
34
35  }
```

Undirected graph, the adjacent matrix is symmetric

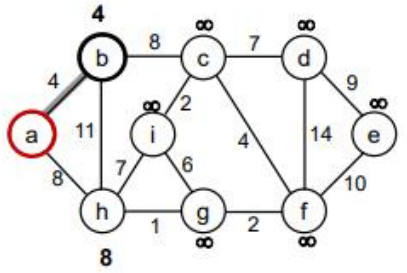
Use a sufficiently large number to represent infinite



Prim's Algorithm: Java Example

```
37  public static void prim(){
38
39      int[] key = new int[n]; //weights of crossing edges
40      int[] preNode= new int[n];//previous nodes
41      boolean[] Va = new boolean[n]; //chosen flags, true for chosen, false for not chosen
42      Va[0] = true;
43
44      for(int i=1;i<n;i++){
45          key[i] = adjMatrix[0][i];   //initialized weights of cross edges
46          preNode[i] = 0; //The first node is selected
47          Va[i] = false;
48      }
49
50
51      for(int i=1;i<n;i++){ // find the next n-1 edges
52          int minWeight = 999;// use a sufficiently large number to represent infinite
53          int minNode = 0;
54          //find the light crossing edge
55          for(int j=1;j<n;j++){
56              if(!Va[j] && key[j]<minWeight){
57                  minWeight = key[j];
58                  minNode = j;
59              }
60          }
61
62          Va[minNode] = true;
63          System.out.println(preNode[minNode]+ "---" +minNode + ", weights:" + minWeight);
64
65          //update new weights of cross edges
66          for(int j=1;j<n;j++){
67              if(!Va[j] && key[j]>adjMatrix[minNode][j]){
68                  key[j] = adjMatrix[minNode][j];
69                  preNode[j] = minNode;
70              }
71          }
72      }
73  }
```

Find the light crossing edge, namely Extract-MIN

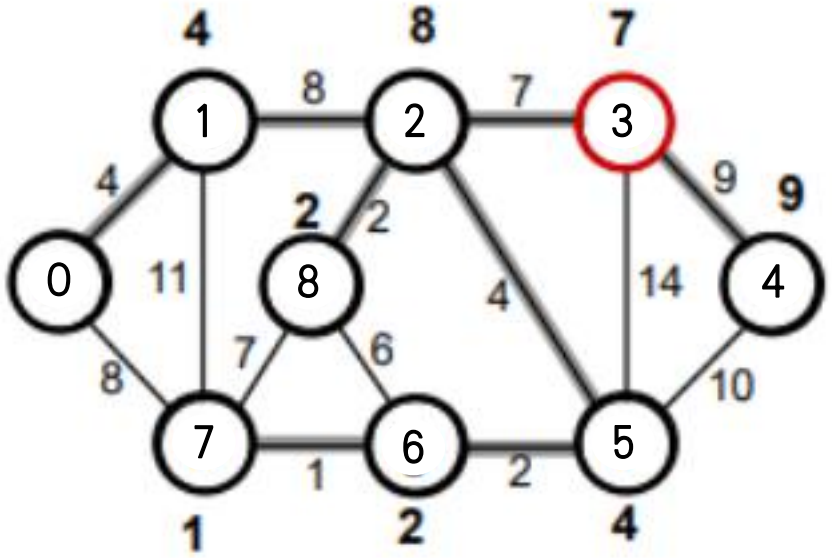


key [b] = 4 π [b] = a
key [h] = 8 π [h] = a

4 ∞ ∞ ∞ ∞ ∞ **8** ∞
Q = {b, c, d, e, f, g, h, i} V_A = {a}
Extract-MIN(Q) ⇒ b

- Here, the Extract-MIN function is realized by brute-force traversal, we can use more efficient algorithm to sort. But in Java, PriorityQueue doesn't support dynamical priority update, so it's kind of challenging to make it work.

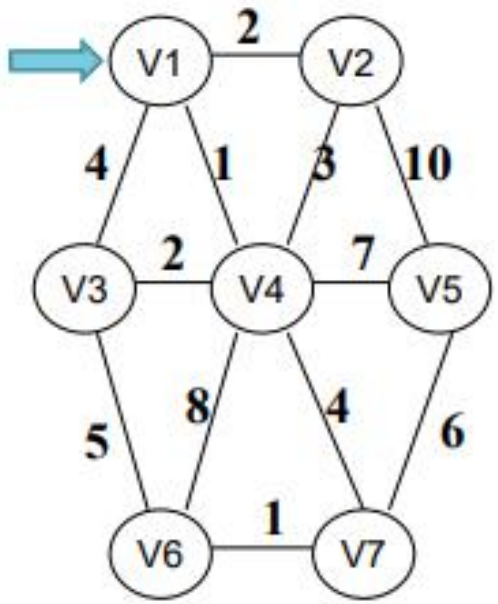
Prim's Algorithm: Java Example




```
<terminated> Prim [Java Applicatio
0---1, weights:4
1---2, weights:8
2---8, weights:2
2---5, weights:4
5---6, weights:2
6---7, weights:1
2---3, weights:7
3---4, weights:9
```

Prim's Algorithm: Java Example

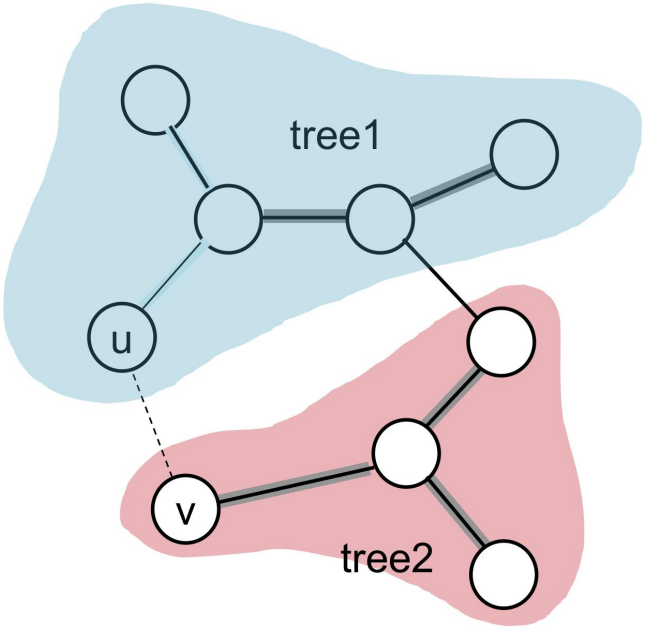
- **Exercise**
  - Modify Prim.java to construct following graph, and show the MST edges, starting from v_1



Kruskal's Algorithm

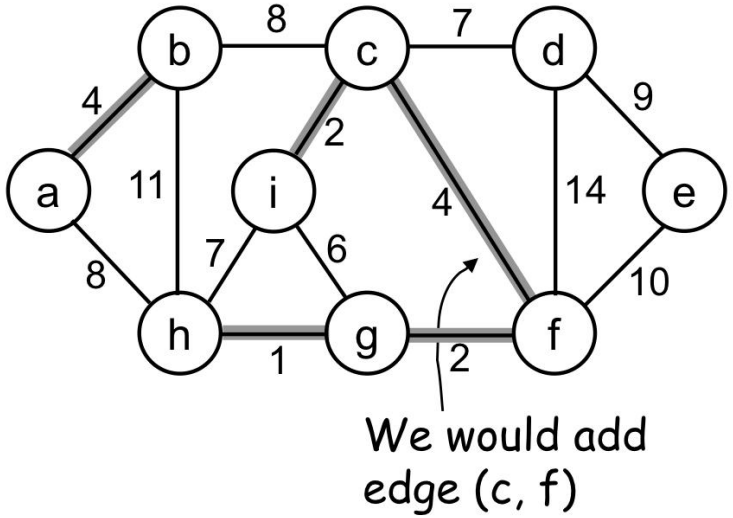
How is it different from Prim's Algorithm?

- Prim's Algorithm grows one tree all the time
- Kruskal's algorithm grows multiple trees (i.e., a forest) at the same time
- Trees are merged together using ***safe*** edges

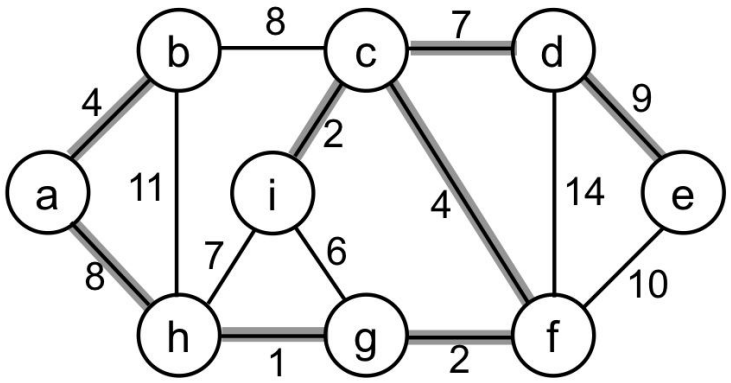


Kruskal's Algorithm

- Start with each vertex being its own component
- Repeatedly merge two components into one by choosing the **light** edge that connects them
- Which components to consider at each iteration?
  - Scan the set of edges in monotonically increasing order by weight



Kruskal's Algorithm



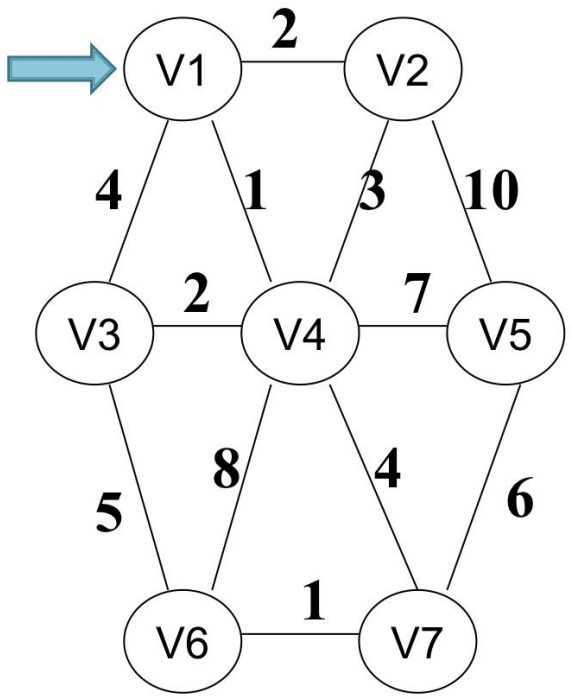
1: (h, g)
2: (c, i), (g, f)
4: (a, b), (c, f)
6: (i, g)
7: (c, d), (i, h)
8: (a, h), (b, c)
9: (d, e)
10: (e, f)
11: (b, h)
14: (d, f)

{a}, {b}, {c}, {d}, {e}, {f}, {g}, {h}, {i}

1.	Add (h, g)	{g, h}, {a}, {b}, {c}, {d}, {e}, {f}, {i}
2.	Add (c, i)	{g, h}, {c, i}, {a}, {b}, {d}, {e}, {f}
3.	Add (g, f)	{g, h, f}, {c, i}, {a}, {b}, {d}, {e}
4.	Add (a, b)	{g, h, f}, {c, i}, {a, b}, {d}, {e}
5.	Add (c, f)	{g, h, f, c, i}, {a, b}, {d}, {e}
6.	Ignore (i, g)	{g, h, f, c, i}, {a, b}, {d}, {e}
7.	Add (c, d)	{g, h, f, c, i, d}, {a, b}, {e}
8.	Ignore (i, h)	{g, h, f, c, i, d}, {a, b}, {e}
9.	Add (a, h)	{g, h, f, c, i, d, a, b}, {e}
10.	Ignore (b, c)	{g, h, f, c, i, d, a, b}, {e}
11.	Add (d, e)	{g, h, f, c, i, d, a, b, e}
12.	Ignore (e, f)	{g, h, f, c, i, d, a, b, e}
13.	Ignore (b, h)	{g, h, f, c, i, d, a, b, e}
14.	Ignore (d, f)	{g, h, f, c, i, d, a, b, e}

Exercise 2: Find an MST

- Use Prim's algorithm
- Use Kruskal's algorithm



Thank you for coming !