## Kruskal Algorithm and Prim Algorithm

*Introduction:*

In this Chapter, two classic algorithms of the Formation of Minimum Spanning Tree problem would be discussed. These two algorithms would use the specific rules to describe the algorithm discussed in the Chapter Formation of Minimum Spanning Tree. These two algorithms expand the method to find a safety edge. *(The method in the Formation of Minimum Spanning Tree refers to the algorithm Generic\_MST(G, w).)*

* *In the Kruskal Algorithm, the collection A is one forest. The nodes in the forest are all nodes in the Graph. Each time the Safety Edge would be added into the collection A, and it surely be the Minimum Weight connected with two separate collections.*
* *In the Prim Algorithm, the collection A is one tree. The Safety Edge would be connected the node among the collection A and the node outside the collection A.*

*Kruskal Algorithm:*

The way the Kruskal Algorithm to find its Safety Edge (u, v) is to find the Edge with Minimum Weight crossing two Trees among the forest.

Assume that the collection C1 and C2 are two separate trees which are connected by the Edge (u, v). Since Edge (u, v) must be the Minimum Weight Edge connecting the collection C1 and C2, such Edge must be the Safety Edge.

*Apparently, the Kruskal Algorithm must be one Greedy Algorithm, since each time it would select one Edge with the Minimum Weight and add into the forest.*

*Realization:*

In the Kruskal Algorithm, it uses one Non - Intersect Element Collection Data Structure to maintain several Non - Intersect Element Collections.

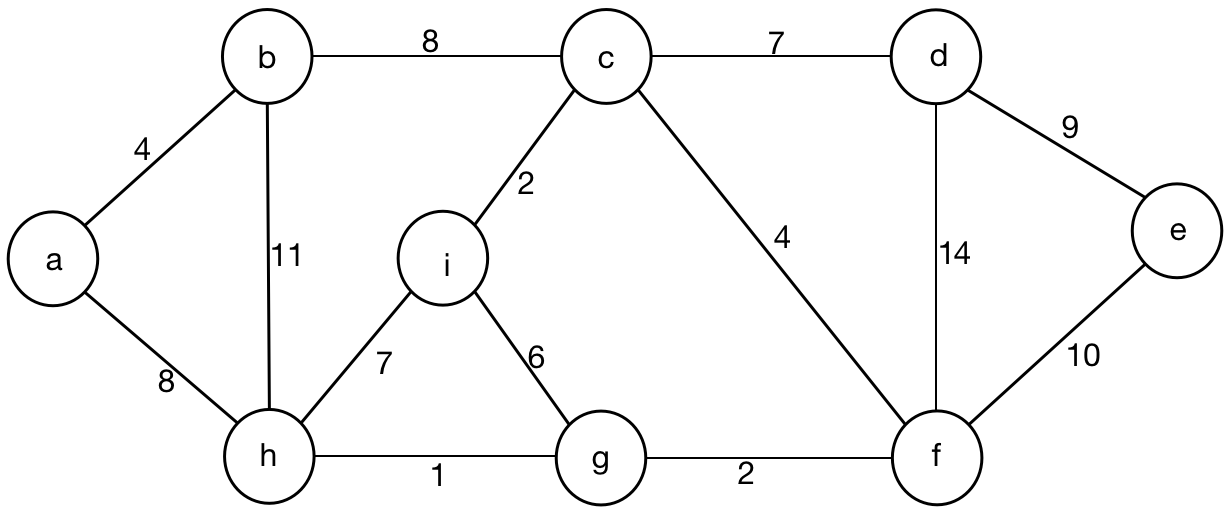
However, each collection would represent one tree among the current forest. The operation FIND\_SET(u) is used to find the collection which includes node u, and the operation FIND\_SET(v) is used to find the collection which includes node v.

*Safety Edge:*

The Safety Edge (u, v) must be the Edge with below property. Using FIND\_SET(u) to find the collection which includes the node u, and FIND\_SET(v) to find the collection which includes the node v. If these two collections are not equal, then such Edge must be the Safety Edge.

*Procedure:*

The Original Graph =>



Initialize the Result Set with name S = EMPTY.

Initialize all sets of the original Graph, then there would exist 9 trees which can use 9 separate sets to represent.

|  |  |  |
| --- | --- | --- |
| A = { ( a ) } | D = { ( d ) } | G = { ( g ) } |
| B = { ( b ) } | E = { ( e ) } | H = { ( h ) } |
| C = { ( c ) } | F = { ( f ) } | I = { ( i ) } |

Sort all 14 edges based on Weight according to its ascending sequence.

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

First Step:

* Select the Minimum Weight Edge h - g with Weight 1.
* FIND\_SET( h ) = H, FIND\_SET( g ) = G.
* H NOT EQUAL TO G.
* Add Edge ( h, g, 1 ) into the Result Set S, S = { ( h, g, 5 ) }.

Second Step:

Third Step:

Forth Step:

Fifth Step:

*Pseudo Code:*

MST\_KRUSKAL(G, w):