## Kruskal 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.

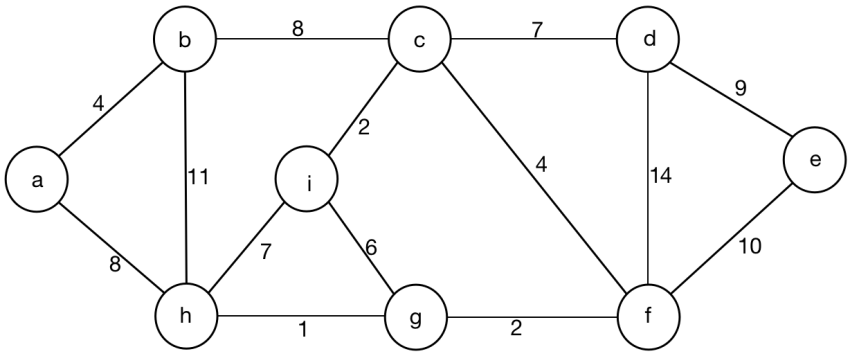
However, each collection would represent one tree among the current forest. The operation FIND\_SET(u) is used to find the collection which includes node u, and the operation FIND\_SET(v) is used to find the collection which includes node v.

*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 ) } |

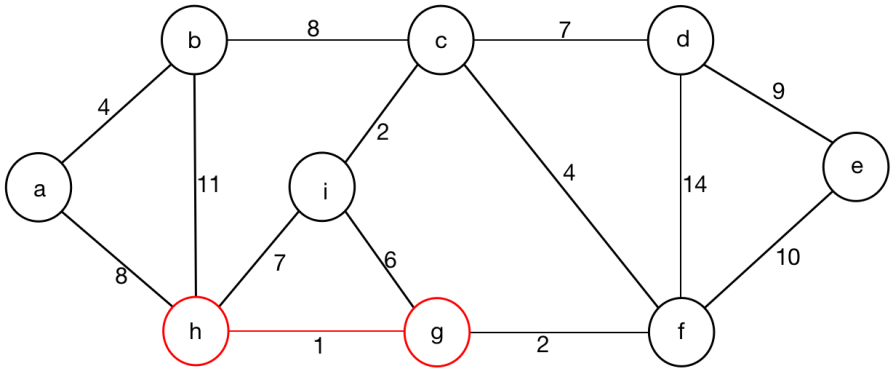
Sort all 14 edges based on Weight according to its ascending sequence.

|  |
| --- |
| ( 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 ) |
| ( b, c, 8 ) |
| ( d, e, 9 ) |
| ( e, f, 10 ) |
| ( b, h, 11 ) |
| ( d, f, 14 ) |

*First Step:*

|  |
| --- |
| ( 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 ) |
| ( b, c, 8 ) |
| ( d, e, 9 ) |
| ( e, f, 10 ) |
| ( b, h, 11 ) |
| ( d, f, 14 ) |

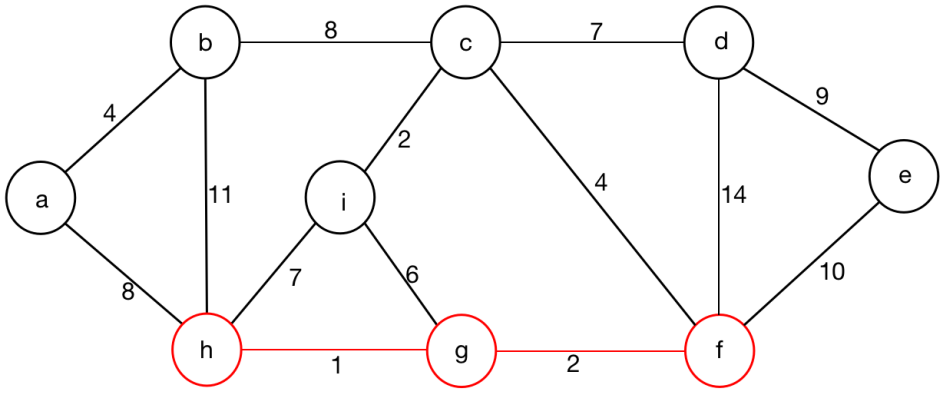
* Select the Minimum Weight Edge ( h, g, 1 ).
* FIND\_SET( h ) = H, FIND\_SET( g ) = G.
* H DO NOT EQUAL TO G.
* Add the Edge ( h, g, 1 ) into the Result Set S, S = { ( h, g, 1 ) }.
* One tree exists, which is { ( h, g, 1 ) }.



*Second Step:*

|  |
| --- |
| ( 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 ) |
| ( b, c, 8 ) |
| ( d, e, 9 ) |
| ( e, f, 10 ) |
| ( b, h, 11 ) |
| ( d, f, 14 ) |

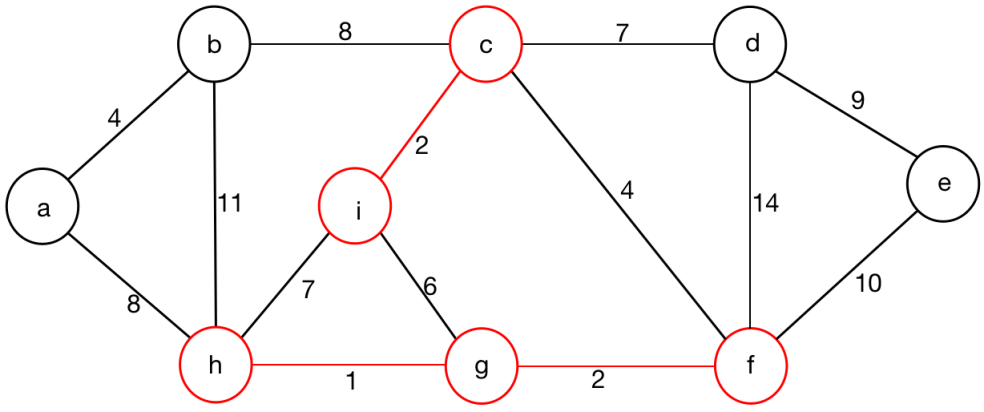
* Select the Second Minimum Weight Edge ( g, f, 2 ).
* FIND\_SET( g ) = G, FIND\_SET( f ) = F.
* G DO NOT EQUAL TO F.
* Add the Edge ( g, f, 2) into the Result Set S, S = { ( h, g, 1 ), ( g, f, 2 ) }.
* One tree exists, which is { ( h, g, 1 ), ( g, f, 2 ) }.



*Third Step:*

|  |
| --- |
| ( 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 ) |
| ( b, c, 8 ) |
| ( d, e, 9 ) |
| ( e, f, 10 ) |
| ( b, h, 11 ) |
| ( d, f, 14 ) |

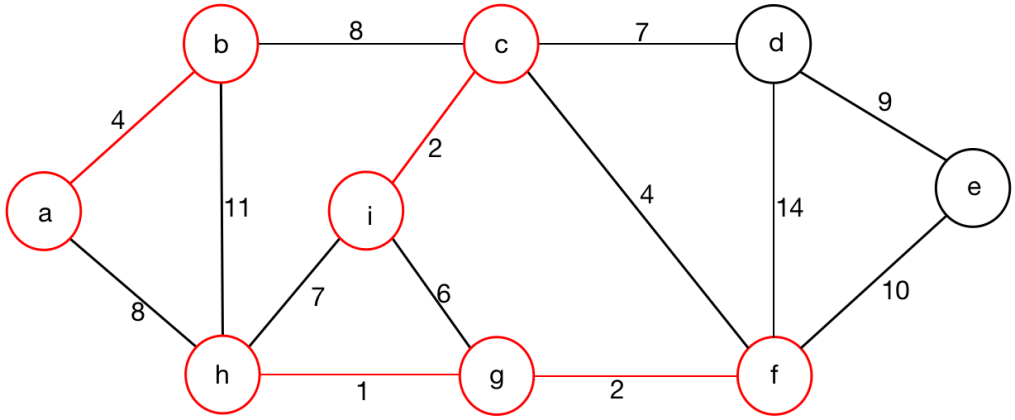
* Select the Third Minimum Weight Edge ( c, i, 2 ).
* FIND\_SET( c ) = C, FIND\_SET( i ) = I.
* C DO NOT EQUAL TO I.
* Add the Edge ( c, i, 2 ) into the Result Set S, S = { ( h, g, 1 ), ( g, f, 2 ), ( c, i, 2 ) }.
* Two trees exist, which are { ( c, i, 2 ) }, { ( h, g, 1 ), ( g, f, 2 ) }.



*Forth Step:*

|  |
| --- |
| ( 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 ) |
| ( b, c, 8 ) |
| ( d, e, 9 ) |
| ( e, f, 10 ) |
| ( b, h, 11 ) |
| ( d, f, 14 ) |

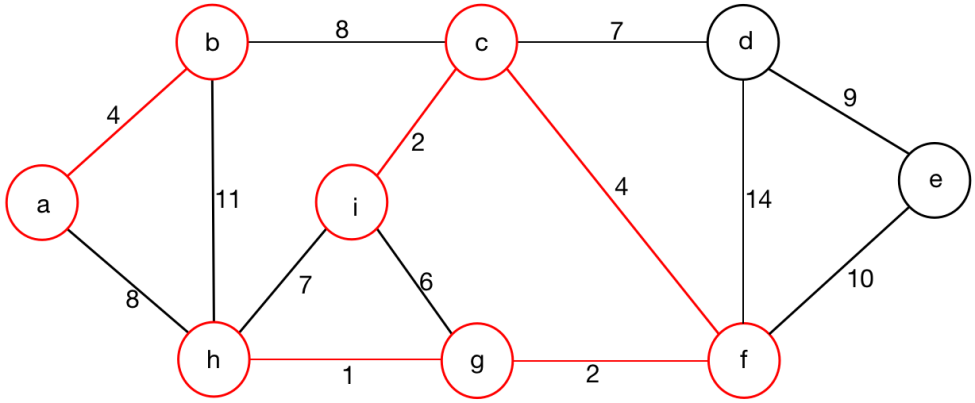
* Select the Forth Minimum Weight Edge ( a, b, 4 ).
* FIND\_SET( a ) = A, FIND\_SET( b ) = B.
* A DO NOT EQUAL TO B.
* Add the Edge ( a, b, 4 ) into the Result Set S, S = { ( h, g, 1 ), ( g, f, 2 ), ( c, i, 2 ), (a, b, 4 ) }.
* Three trees exist, which are { ( a, b, 4 ) }, { ( c, i, 2 ) }, { ( h, g, 1 ), ( g, f, 2 ) }.



*Fifth Step:*

|  |
| --- |
| ( 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 ) |
| ( b, c, 8 ) |
| ( d, e, 9 ) |
| ( e, f, 10 ) |
| ( b, h, 11 ) |
| ( d, f, 14 ) |

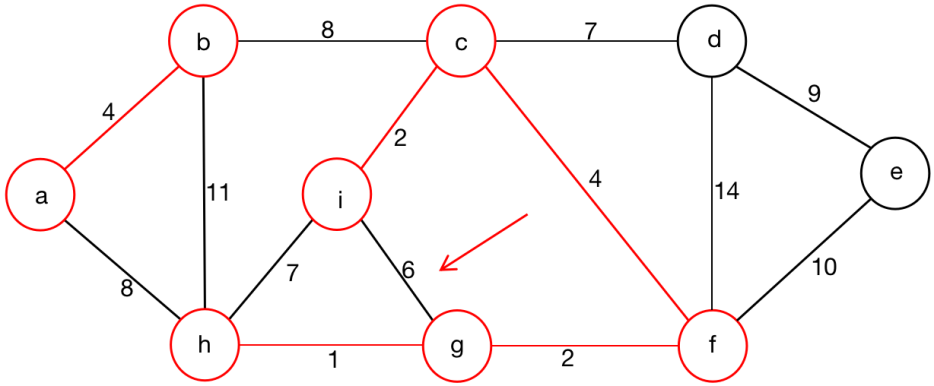
* Select the Fifth Minimum Weight Edge ( c, f, 4).
* FIND\_SET ( c ) = C, FIND\_SET( f ) = F.
* C DO NOT EQUAL TO F.
* Add the Edge ( c, f, 4 ) into the Result Set S, S = { ( h, g, 1 ), ( g, f, 2 ), ( c, i, 2 ), ( a, b, 4 ), ( c, f, 4 ) }.
* Two trees exist, which are { ( a, b, 4 ) }, { ( c, i, 2 ), ( h, g, 1 ), ( g, f, 2 ), ( c, f, 4 ) }.



*Sixth Step:*

|  |
| --- |
| ( 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 ) |
| ( b, c, 8 ) |
| ( d, e, 9 ) |
| ( e, f, 10 ) |
| ( b, h, 11 ) |
| ( d, f, 14 ) |

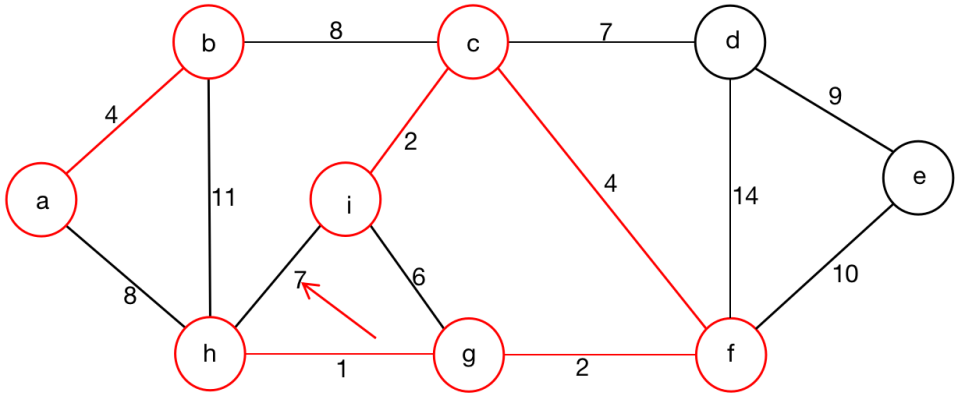
* Select the Sixth Minimum Weight Edge ( i, g, 6 ).
* FIND\_SET( i ) = FIND\_SET( g ), pass this round.



*Seventh Step:*

|  |
| --- |
| ( 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 ) |
| ( b, c, 8 ) |
| ( d, e, 9 ) |
| ( e, f, 10 ) |
| ( b, h, 11 ) |
| ( d, f, 14 ) |

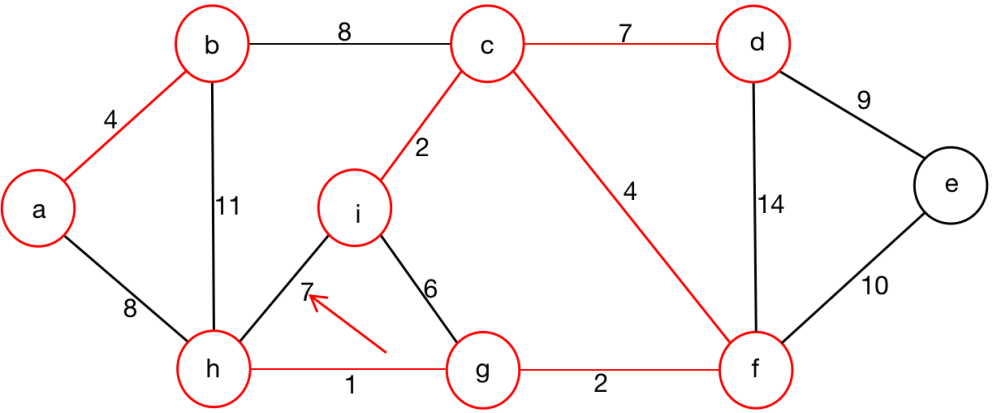
* Select the Seventh Minimum Weight Edge ( i, h, 7 ).
* FIND\_SET( i ) = FIND\_SET( h ), pass this round.



*Eighth Step:*

|  |
| --- |
| ( 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 ) |
| ( b, c, 8 ) |
| ( d, e, 9 ) |
| ( e, f, 10 ) |
| ( b, h, 11 ) |
| ( d, f, 14 ) |

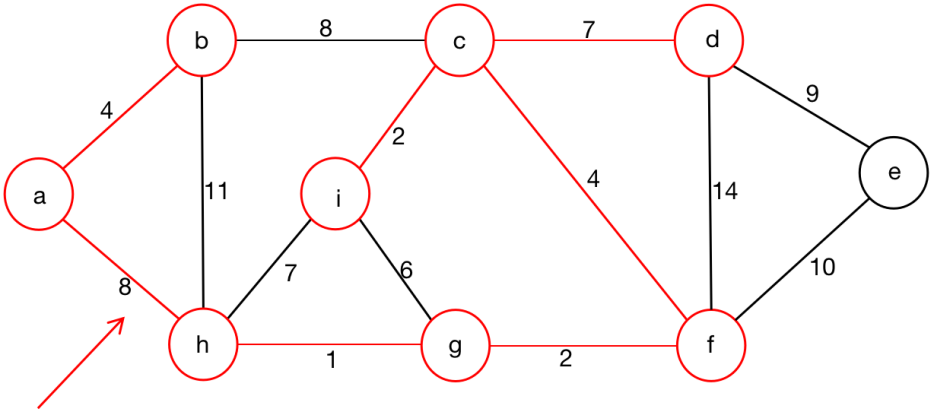
* Select the Eighth Minimum Weight Edge ( c, d, 7 ).
* FIND\_SET( c ) = C, FIND\_SET( d ) = D.
* C DO NOT EQUAL TO D.
* Add the Edge ( c, d, 7 ) into the Result Set S, S = { ( h, g, 1 ), ( g, f, 2 ), ( c, i, 2 ), ( a, b, 4 ), ( c, f, 4 ), ( c, d, 7 ) }.
* Two trees exist, which are { ( a, b, 4 ) }, { ( c, i, 2 ), ( h, g, 1 ), ( g, f, 2 ), ( c, f, 4 ), ( c, d, 7 ) }.



*Ninth Step:*

|  |
| --- |
| ( 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 ) |
| ( b, c, 8 ) |
| ( d, e, 9 ) |
| ( e, f, 10 ) |
| ( b, h, 11 ) |
| ( d, f, 14 ) |

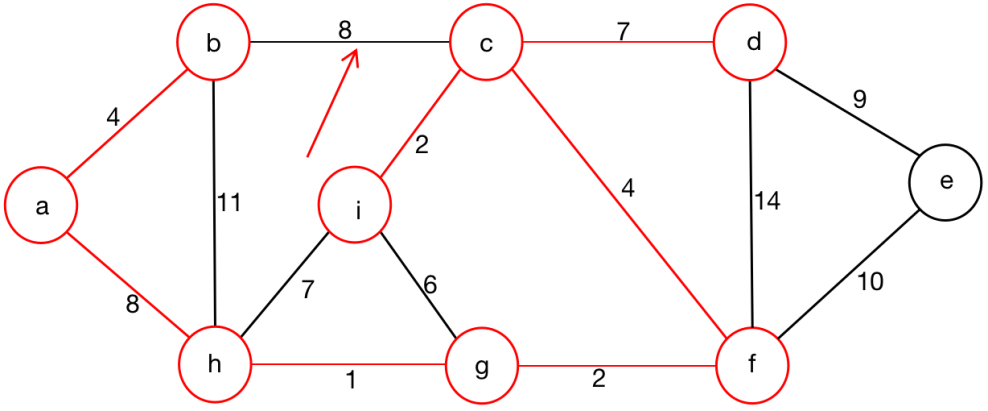
* Select the Ninth Minimum Weight Edge ( a, h, 8 ).
* FIND\_SET( a ) = 1, FIND\_SET( h ) = 2.
* 1 DO NOT EQUAL TO 2.
* Add the Edge ( a, h, 8 ) into the Result Set S, S = { ( h, g, 1 ), ( g, f, 2 ), ( c, i, 2 ), ( a, b, 4 ), ( c, f, 4 ), ( c, d, 7 ), ( a, h, 8 ) }.
* One tree exists, which is { ( h, g, 1 ), ( g, f, 2 ), ( c, i, 2 ), ( a, b, 4 ), ( c, f, 4 ), ( c, d, 7 ), ( a, h, 8 ) }.



*Tenth Step:*

|  |
| --- |
| ( 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 ) |
| ( b, c, 8 ) |
| ( d, e, 9 ) |
| ( e, f, 10 ) |
| ( b, h, 11 ) |
| ( d, f, 14 ) |

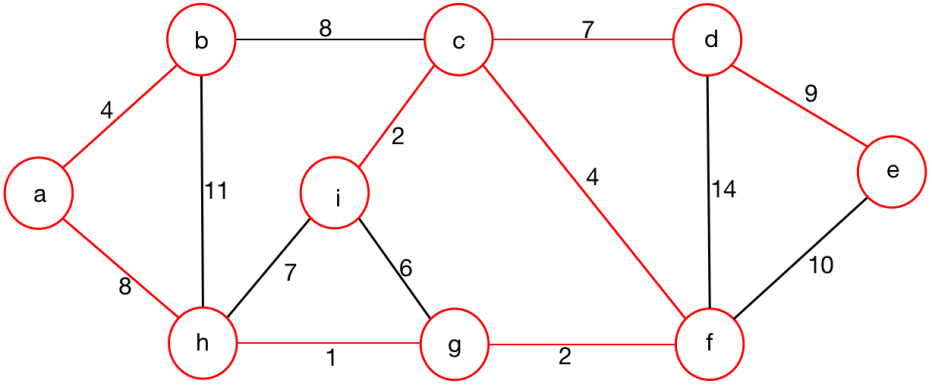
* Select the Tenth Minimum Weight Edge ( b, c, 8 ).
* FIND\_SET(b) = 1, FIND\_SET(c) = 1.
* 1 DO EQUAL TO 1, pass this round.



*Eleventh Step:*

|  |
| --- |
| ( 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 ) |
| ( b, c, 8 ) |
| ( d, e, 9 ) |
| ( e, f, 10 ) |
| ( b, h, 11 ) |
| ( d, f, 14 ) |

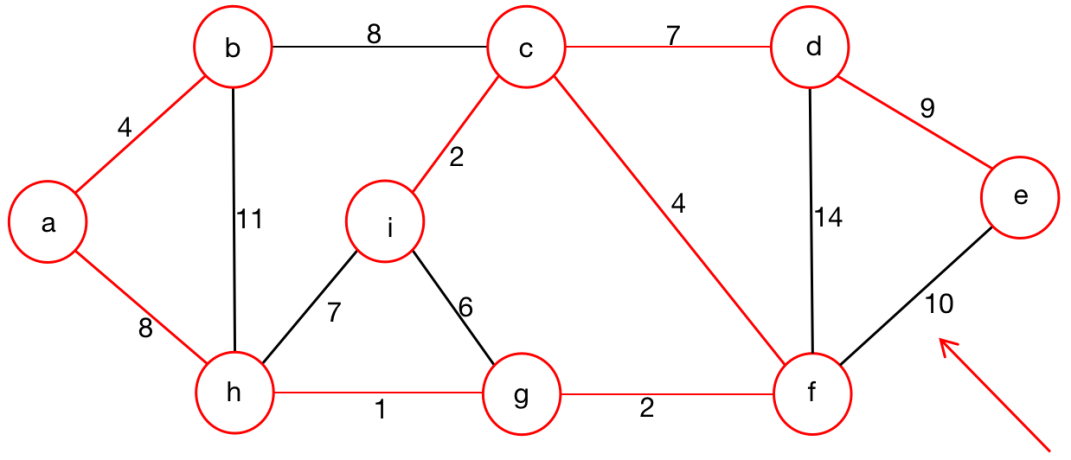
* Select the Eleventh Minimum Weight Edge ( d, e, 9 ).
* FIND\_SET( d ) = 1, FIND\_SET( e ) = E.
* 1 DO NOT EQUAL TO E.
* Add the Edge ( d, e, 9 ) into the Result Set S, S = { ( h, g, 1 ), ( g, f, 2 ), ( c, i, 2 ), ( a, b, 4 ), ( c, f, 4 ), ( c, d, 7 ), ( a, h, 8 ), ( d, e, 9 ) }.
* One tree exists, which is { ( h, g, 1 ), ( g, f, 2 ), ( c, i, 2 ), ( a, b, 4 ), ( c, f, 4 ), ( c, d, 7 ), ( a, h, 8 ), ( d, e, 9 ) }.



*Twelfth Step:*

|  |
| --- |
| ( 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 ) |
| ( b, c, 8 ) |
| ( d, e, 9 ) |
| ( e, f, 10 ) |
| ( b, h, 11 ) |
| ( d, f, 14 ) |

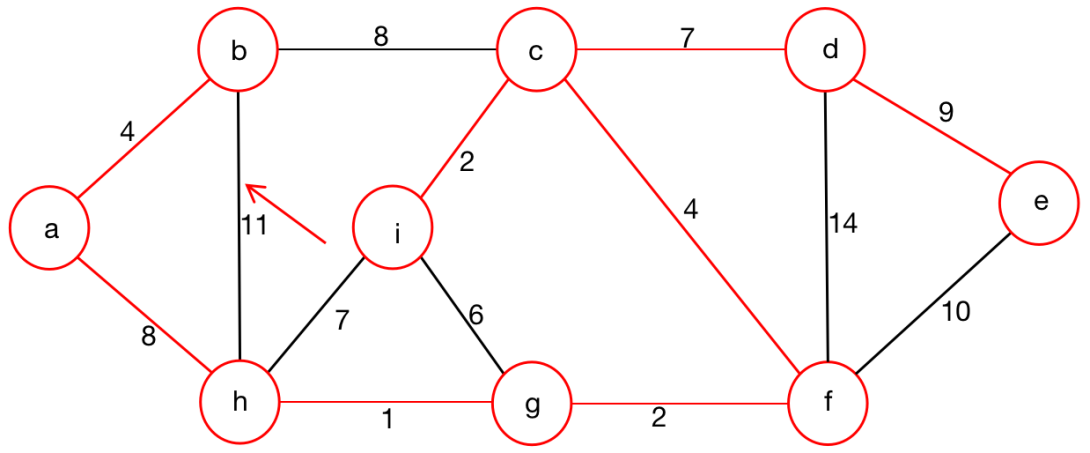
* Select the Twelfth Minimum Weight Edge ( e, f, 10 ).
* FIND\_SET( e ) = 1, FIND\_SET( f ) = 1.
* 1 DO EQUAL TO 1, pass this round.



*Thirteenth Step:*

|  |
| --- |
| ( 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 ) |
| ( b, c, 8 ) |
| ( d, e, 9 ) |
| ( e, f, 10 ) |
| ( b, h, 11 ) |
| ( d, f, 14 ) |

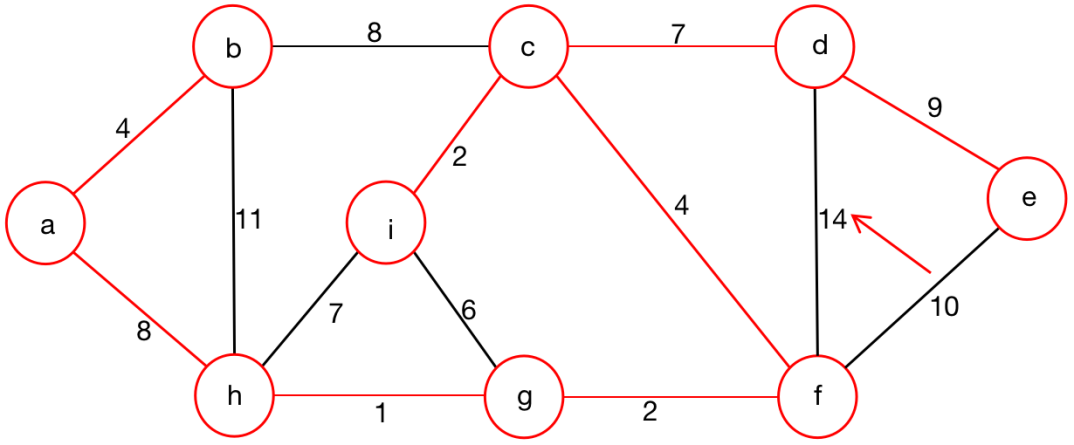
* Select the Thirteenth Minimum Weight Edge ( b, h, 11 ).
* FIND\_SET( b ) = 1, FIND\_SET( h ) = 1.
* 1 DO EQUAL TO 1, pass this round.



*Fourteenth Step:*

|  |
| --- |
| ( 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 ) |
| ( b, c, 8 ) |
| ( d, e, 9 ) |
| ( e, f, 10 ) |
| ( b, h, 11 ) |
| ( d, f, 14 ) |

* Select the Fourteenth Minimum Weight Edge ( b, f, 14 ).
* FIND\_SET( b ) = 1, FIND\_SET( f ) = 1.
* 1 DO EQUAL TO 1, pass this round.



*Pseudo Code:*

***MST\_KRUSKAL(G, w):***

*A = EMPTY;*

*Sort the Edge Set into the non - decreasing sequence;*

*For each edge (u, v) in the Edge Set:*

*Check whether FIND\_SET(u) equals to FIND\_SET(v);*

*If FIND\_SET(u) == FIND\_SET(v):*

*Then skip this round;*

*Otherwise*

*Add the Edge (u, v) into the specific collection;*

*{ The edge ( u, v ) is one safety Edge. }*

*Explanation:*

In *MST\_KRUSKAL(G, w)* procedure, we need to assign the current edge into specific collection. We need to check which collections these two nodes belong to. After that, combine two collections of two nodes, otherwise, just neglect this kind of situation, skip to check the next Edge with the Minimum Weight.