**Analysis of algorithms**

**Lab 2**

**Greedy Algorithms**

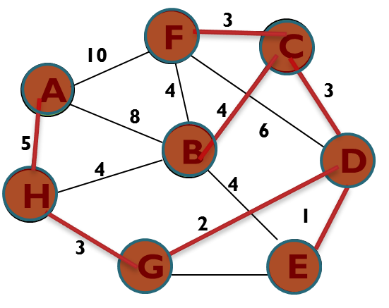
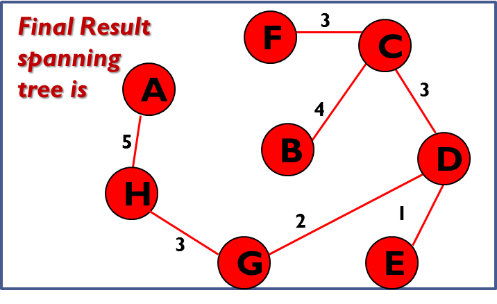
***Problem 1*:**

Check lecture number 2 to implement the ***Kruskal*** algorithm using C++ code. The input of the program from user is as follows:

1. The number of vertices n.
2. Array of the n characters defining the n vertices of a graph G.
3. A set of edge weights between vertices.

***The output***:

1. The result of the program should print the minimum spanning tree based on the input of the user. The printed output should print A-H, H-G, G-D, D-E, D-C, C-B, C-F, if the printed output is as follows:

 🡺 

1. Also the output of the program should show the number of steps required from the starting step to the ending step.
2. The output should print the processing time from the starting step to the ending step.

***Kruskal Algorithm***

T = {};

n = |V|

1. Sort the edges of E in an ascending order according to weight w;

2. While |T| < n

Remove (u,v)edge of lowest weight w from E

if( ((x,u)edge ∈ T ) && ((y,v)edge ∈ T) )

disregard (u,v)edge

else

add (u,v)edge to T

return T;

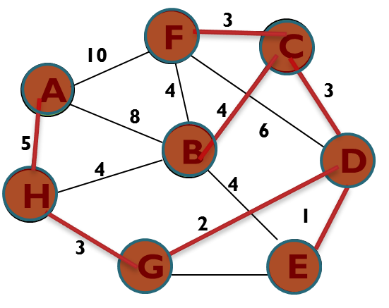
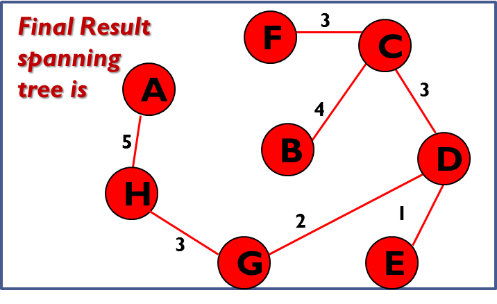
***Problem 2*:**

Check lecture number 2 to implement the ***Prim*** algorithm using C++ code. The input of the program from user is as follows:

1. The number of vertices n.
2. Array of the n characters defining the n vertices of a graph G.
3. A set of edge weights between vertices.

***The output***:

1. The result of the program should print the minimum spanning tree based on the input of the user. The printed output should print A-H, H-G, G-D, D-E, D-C, C-B, C-F, if the printed output is as follows:

 🡺 

1. Also the output of the program should show the number of steps required from the starting step to the ending step.
2. The output should print the processing time from the starting step to the ending step.

***Prim Algorithm***

***T*** = ϕ; //Set of selected edges 1 step

***S =*** ϕ; //Set of selected vertexes in MST 1 step

For every vertex ***v*** in ***V***; key(***v***) = ∞; initial key value ∞ |V| steps

Pick random vertex ***a*** in ***V***; key(***a***)= 0; initial key value 0 1 step

while (***S***≠***V***){ n=|***V***| steps

Get vertex ***u*** of minimum key value in ***V***, such that ***u***∉***S*** n=|***V***| steps

***T*** = ***T*** ⋃ {(***u***, ***a***)}, such that ***a***∈***S* &&** weight(***u-a***)=key(***u)*** 1 step

***S*** = ***S*** ⋃{***u***}, and remove ***u*** from ***V*** 1 step

For every vertex ***v*** connected to ***u*** such that ***v***∉***S*** n=|***V***| steps

If weight(***u***-***v***) weight is less than key(***v***) 1 step

Then key(***v***) = weight(***u***-***v***)

***Problem 3*:**

Run both algorithms concurrently in one program using threads and show which algorithm finishes first.