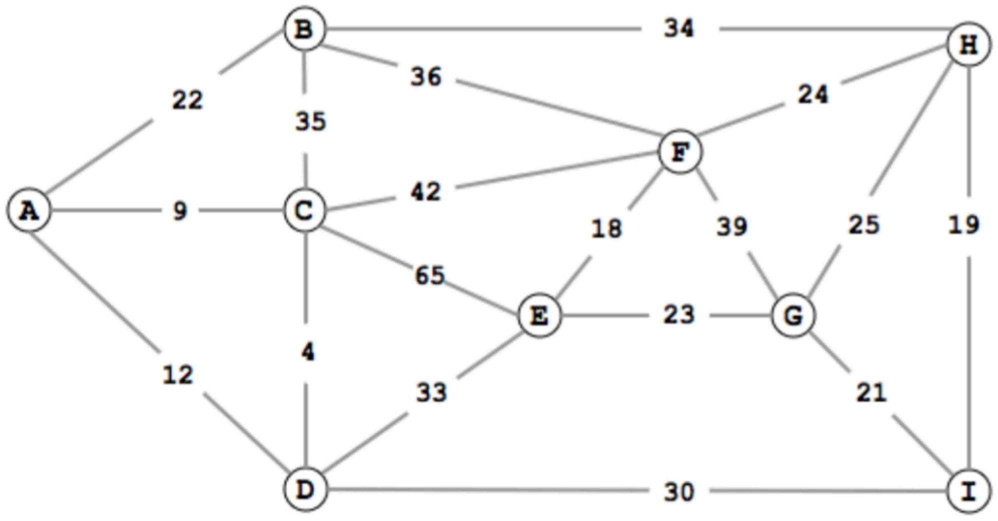
Graph G: Undirected weighted graph.



Adjacency Matrix Representation of the above weighted graph G.

	A	B	C	D	E	F	G	H	I
A	0	22	9	12	0	0	0	0	0
B	22	0	35	0	0	36	0	34	0
C	9	35	0	4	65	42	0	0	0
D	12	0	4	0	33	0	0	0	30
E	0	0	65	33	0	18	23	0	0
F	0	36	42	0	18	0	39	24	0
G	0	0	0	0	23	39	0	25	21
H	0	34	0	0	0	24	25	0	19
I	0	0	0	30	0	0	21	19	0

Adjacency List representation of the above weighted graph.

A B 22 C 9 D 12
B A 22 C 35 F 36 H 34
C A 9 B 35 D 4 E 65 F 42
D A 12 C 4 E 33 I 30
E C 65 D 33 F 18 G 23
F B 36 C 42 E 18 G 39 H 24
G E 23 F 39 H 25 I 21
H B 34 F 24 G 25 I 19
I D 30 G 21 H 19

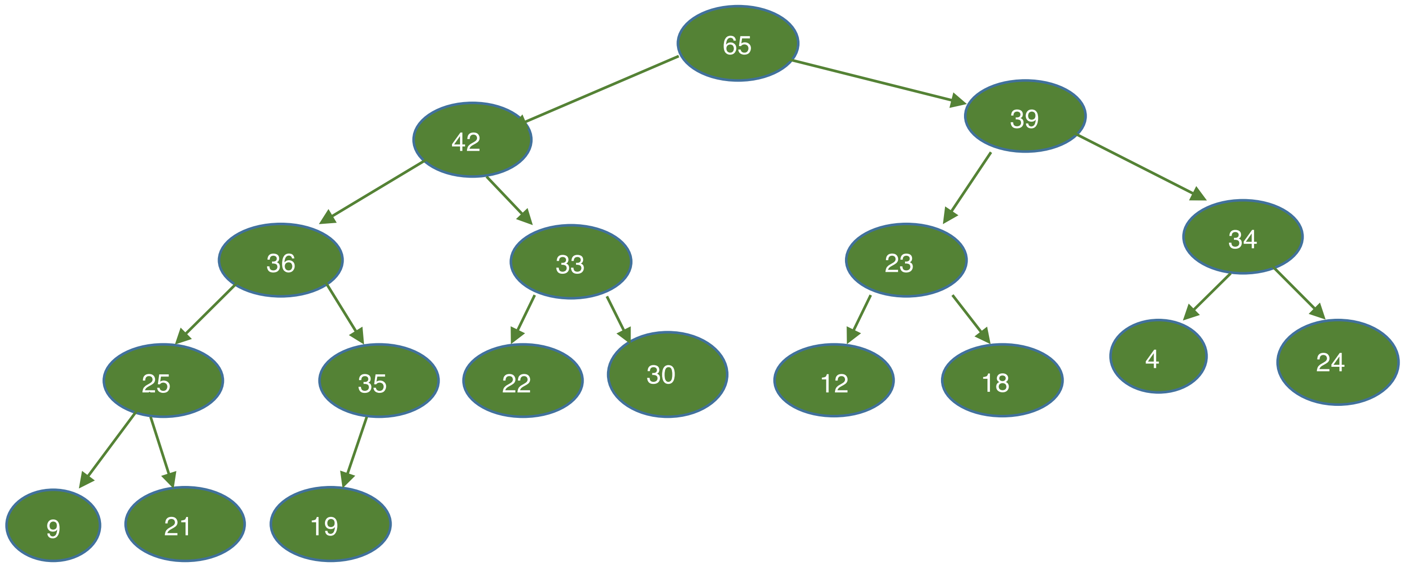
1. **Heap and Heapsort:**

Algorithm:

```
Step1: Heapify the array;

Step2:

while the array isn't empty {

    Remove and replace the root;

    Re-heap the new root node;

}
```

Language: C++
Platform: Compiled and executed in Mac/Linux(Ubuntu v 14.04.1)

Compilation: %> g++ main.C
Heapified BinaryTree:



2.Shortest Path Dijkstra Algorithm.

Description:

Dijkstra's shortest path algorithm finds shortest path from source to any nodes of a Graph. It is a greedy algorithm as it picks unvisited node with lowest distance in each step. The input the algorithm is the "Adjacency List" representation of the above undirected weighted graph It find out the shortest path from source node to all other nodes.

Adjacency List
Dijkstra
Shortest Path, Cost

Algorithm:

Shortest path Disjkstra algorithm has been implemented based on following algorithm.

```
Initialization:
  S = {s}
  for all nodes v
    if v adjacent to node s
      then D'(v) = c(s,v)
    else D'(v) = infinity

 Loop
    find v not in S such that D'(v) is a minimum
    add v to S and set D(v) =D'(v)
    update D'(w) for all w adjacent to v and not in S:
       D'(w) = min( D'(w), D(v) + c(v,w) )
     /* new cost to w is either old cost to w or known
        shortest path cost to v plus direct link cost from v to w */
  until all nodes in S
```

Language: C++

Platform: Compiled and executed in Mac/Linux(Ubuntu v 14.04.1)

Compilation: `%> g++  main.C`

4. (Minimum Spanning Tree): Kruskal's Algorithm to generate one MST for this graph.

Description:

Kruskal's algorithm finds minimum spanning tree (MST) of a Graph where algorithm finds an edge of least possible weight which connects two trees in the forest. It is a greedy algorithm as it finds MST for a weighted connected graph adding edges in increasing cost at each steps. Input to the Kruskal's algorithm is an Adjacency List representation of specified Graph in the assignment.

Algorithm:

```
Kruskal(G, c) {
  Sort edges weights so that $c_1 \leq c_2 \leq ... \leq c_m$.
  $T \leftarrow \phi$
  foreach ($u \in V$) make a set containing the
      single node u
  for i = 1 to m
    (u,v) = $e_i$   % nodes u and v are for this edge
    if (u and v are in different sets) {
      $T \leftarrow T \cup \{e_i\}$
      merge the sets containing u and v
    }
  return T
}
```

Language: C++
Platform: Compiled and executed in Mac as well as in Eustis (eustis.eecs.ucf.edu) machine.
Compilation: `%> g++ main.C`