Analysis of Minimum Spanning Tree using Prim's and Kruskal's Algorithm

Upasana Ghosh

Roll no: UE143110
U.I.E.T, Panjab University
Email: ghoshupasana05@gmail.com

AIM

To draw a minimum spanning tree using Prim's and Kruskal's algorithm and represent it graphically by drawing nodes and edges.

APPROACH

Minimum spanning tree problem can be solved using greedy approach. A greedy algorithm always makes the choice that look best at the moment. It makes a locally optimal choice in the hope that this choice will lead to a global optimal solution. It does not always lead to optimal solution but for many problems optimal results can be obtained. It first make a greedy choice, the choice that looks best at that moment and then solve the resulting sub-problems without bothering to solve other sub-problems. It generates only one decision sequence unlike dynamic programming which generates multiple decision sequence.

A connected graph with n vertices and n-1 edges is spanning tree and n-1 edges with no cycle is a tree. Let G=(V,E) be a connected graph in which each edge E(u,v) has an associated cost C(u,v). A spanning tree for G is a subgraph of G that is a free tree connecting all vertices in v. The cost of spanning tree is the sum of cost of its edges. A minimum spanning tree in an undirected connected weighted graph is a spanning tree of minimum weight.

For finding minimum spanning tree in a connected weighted graph, Prim's algorithm is used. In this algorithm, first minimum cost edge is selected from the graph and vertices of the selected edge is stored in a resultant matrix *t.* Total number of vertices in the graph is *vcount*. The size of *t* matrix would be *vcount* x 2. The cost associated with the selected edge is added in *mincost* variable. Near edges to these selected vertices with minimum cost is then selected and the vertices of that selected edge is again placed in *t* matrix. This step will repeat *vcount*-1 times so that the resulting spanning tree would consist of *vcount*-1 number of edges.

Kruskal's algorithm is a minimum spanning tree algorithm where the algorithm finds an edge of the lease possible weight that connects any two trees in the forest. It is a greedy algorithm in a graph theory as it finds a minimum spanning tree for a connected weighted graph adding increasing cost at each step. In this algorithm, edge with minimum cost is selected and vertices of the selected edge is stored in a resultant matrix *t.* Total number of vertices in the graph is *vcount.* The size of *t* matrix would be *vcount* x 2. After accessing the selected vertices, these vertices are merged so to avoid formation of cycle in spanning tree. The cost associated with the selected edge is added in *mincost* variable. This procedure will repeat vcount-1 number of times.

EXPERIMENT

In this experiment, vertices and edges are drawn on mouse click and keyboard button click. C programming language is used in this experiment.

As the vertices and edges are represented graphically, graphics.h and dos.h header files are used to implement them. C graphics using graphics.h functions can be used to draw different shapes, display text in different fonts, colors and many more. This header file can be used for animations, projects and games. Different sizes of circle, rectangle, ellipse, lines, bars and other geometrical figures can be drawn with this header file. The header file dos.h of C language contains functions for handling interrupts, producing sound, date and time functions etc. It is borland specific and works in turbo c compiler.

In this program, vertices are created graphically in vetex_fill() function. This function contain fillellipse(), setcolor(),outtextxy() functions whose definitions are available in graphics.h library. fillellipse() functions is used to create vertex of the graph. Setcolor() function is used to set color to this vertex. Outtextxy() function is used to print vertex number and vertex name on the graphical screen. Vertices would keeps on creating on mouse click until the user press 1 from keyboard.

Edges are created graphically in edge_fill() function. This function contain a line(), setcolor(), outtextxy()whose definitions are available in graphics.h library. Line() is used to create line in graphical screen according to the given x and y coordinates. Edge_fill() function first draws vertex graphically and then create line joining those vertices.

As the vertices and edges are drawn using mouse and keyboard click, mouse.h header file is created manually using graphic.h and dos.h header files and is added in this program. Mouse.h header file consist of several functions like initMouse(), showMouse(), hideMouse() and Isclick(). All these functions uses AX register to access mouse pointer and keyboard keys. Interrupts are called to use graphics and provide delay in taking inputs. Mouse events are tracked by using software interrupt '0x33'. Here '0x' means that the number is in hexadecimal. The interrupt call is made using int86( ) function. This function is defined in dos.h. They are of type union REGS. The union inreg and outreg hold values of registers before and after the call has been made.

To initialize a mouse, the value of AX register is set as '0x00'. This tells the system that when the '0x33' interrupt is called, it should initialize the mouse (basically get it ready for other functions). showMouse() function set the AX register value to 1 and display the mouse pointer in graphical screen. hideMouse() function set the AX register value to 2 and is used to hide the mouse pointer from the graphical screen. Isclick() function set the AX register value to 3 and is used to get the x and y coordinate of the screen where user has clicked with mouse pointer. It also tells the mouse button (left, right or middle) the user has pressed on the graphical screen. kbhit( ) function returns nonzero integer if any key is pressed else it returns zero.

In this program, two 2-dimension array is taken, that is matrix *adj* and matrix *extra*. GetVertices() is called which stores the values of vertex coordinates and vertex name and draw vertex as soon as user clicks on the graphical screen with the mouse pointer. If user press keyboard key 1 then the program will prompt the user to input edge weight using GetEdges() and edgecost() functions and matrix *adj* is updated. Once, all edge costs are inserted by the user, it would redraw the graph using ReDraw(). Then PrimsAlgo() function is called which computes minimum cost spanning tree and display the result on the graphical screen. Similarly, KruskalAlgo() function is called which computes minimum cost spanning tree and display the result on the graphical screen.

Platform used for compilation and execution:

Operating system: Window 10

RAM: 4GB

Compiler: Turbo C++ 4.0

Processor: intel core i3

ALGORITHM

Prim's algorithm:

Algorithm Prim's(adj,n,t)

```
{//adj[1:n,1:n]is the cost of adjacency matrix
//of a graph with n vertex
for i=1 to n
      near[i]=-1;
  for k=1 to n
    {     for l=1to n
           {         if(adj[k][l]<min_edge)
                       {min_edge=adj[k][l];
                         min_x=k;
                         min_y=l;
                     }
                      }
          }
  int mincost=0;
  mincost+=min_edge;
  int t[n1][2];
  t[0][0]=min_x;
  t[0][1]=min_y;
    for i=1 to n
  {      if(adj[i][min_x]<adj[i][min_y])
                near[i]=min_x;
         else
                near[i]=min_y;
  }
    near[min_x]=near[min_y]=-1;
  for i=1 to n
  {  min_j=max_num;
            for j=1 to n
    {if(adj[j][near[j]]<min_num &  near[j]!=-1)
                    min_j=j;
 }
    t[i][0]=min_j;
    t[i][1]=near[min_j];
          mincost+=adj[min_j][near[min_j]];
          near[min_j]=-1;
          for k=1 to n
          {        if(near[k]!=-1 &
adj[k][near[k]]>adj[k][min_j])
                    near[k]=min_j;
          }
 return mincost;
}
```

Kruskal's algorithm:

Algorithm Kruskal'sAlgo(E,cost,n,t)

```
{ construct a minheap out of edge cost using
heapify
for i=1 to n
```

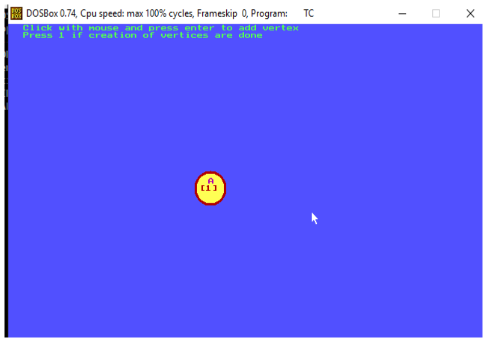
```
    parent[i]=-1;
  i=0,mincost=0
while((i<n-1)and (heap not empty))
do{ delete a mincost edge (u,v) from heap and
heapify using adjust,
 j=find(u), k=find(v)

    if(j!=k)
then { i=i+1;
  t[i,1]=u, t[i,2]=v;
mincost+=cost(u,v)
union(j,k);
}
if(i!=n+1)
 then write ("No Mst")
else return mincost
}
}
```

RESULT

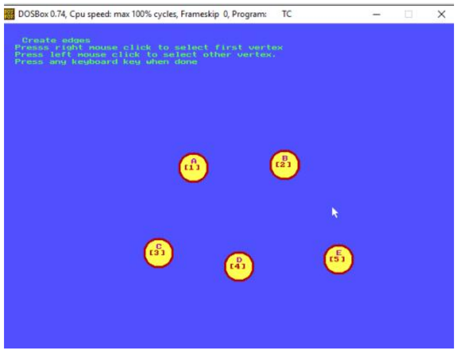
Figure 1 to Figure 10 represents the formation of MST using Kruskal's algorithm and Figure 11 to Figure 18 represents the formation of MST using Prim's algorithm.

MST from Kruskal's algorithm


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
A
[1]

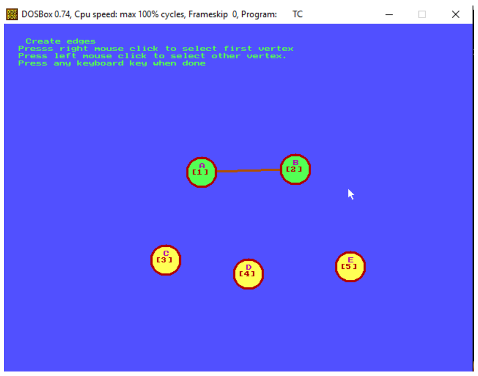
```
Click with mouse and press enter to add vertex
Press 1 if creation of vertices are done
```

Fig 1: 1st vertex created


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
A
[1]
B
[2]
C
[3]
D
[4]
E
[5]

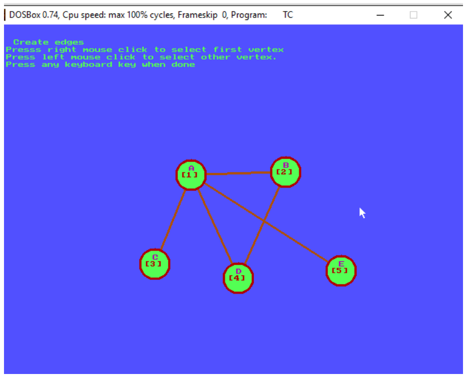
```
 Create edges
Presss right mouse click to select first vertex
Press left mouse click to select other vertex.
Press any keyboard key when done
```

Fig 2: 5th vertex created


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
A
[1]
B
[2]
C
[3]
D
[4]
E
[5]

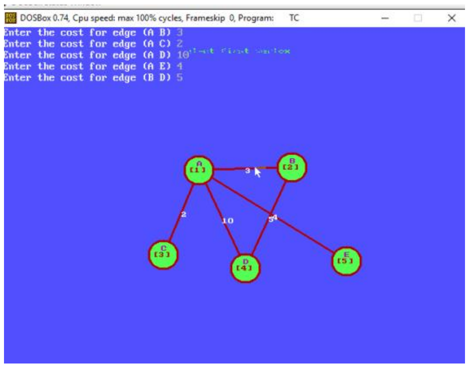
```
 Create edges
Presss right mouse click to select first vertex
Press left mouse click to select other vertex.
Press any keyboard key when done
```

Fig 3: 1st edge created


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

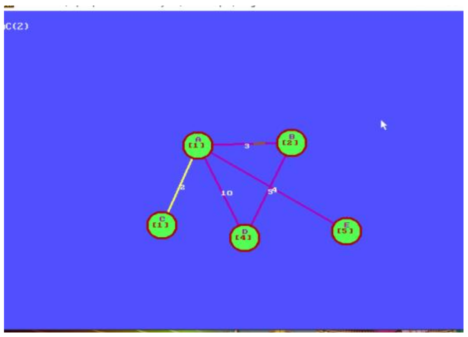
```
 Create edges
Presss right mouse click to select first vertex
Press left mouse click to select other vertex.
Press any keyboard key when done
```

Fig 4: 5th edge created


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

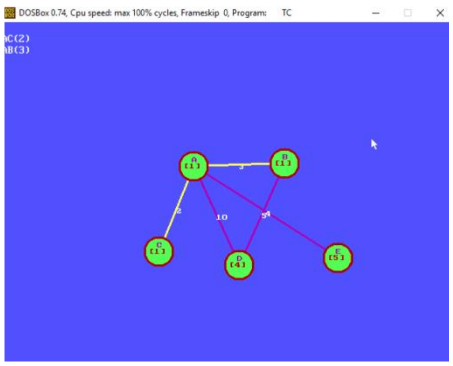
```
Enter the cost for edge (A B) 3
Enter the cost for edge (A C) 2
Enter the cost for edge (A D) 10
Enter the cost for edge (A E) 4
Enter the cost for edge (B D) 5
```

Fig 5: 5th edge created



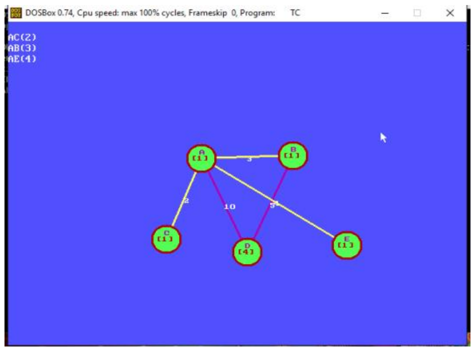
```
C(2)
```

Fig 6: 1st edge selected


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

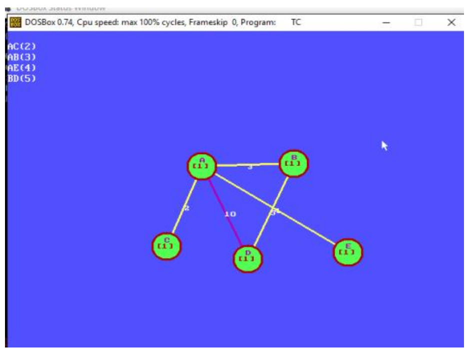
```
C(2)
B(3)
```

Fig 7: 5th edge cost entered


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

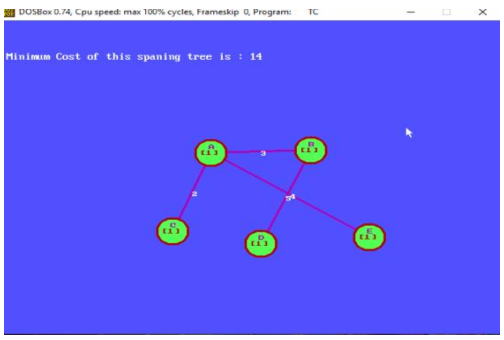
```
AC(2)
AB(3)
AE(4)
```

Fig 8: Graph created


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
AC(2)
AB(3)
AE(4)
BD(5)
```

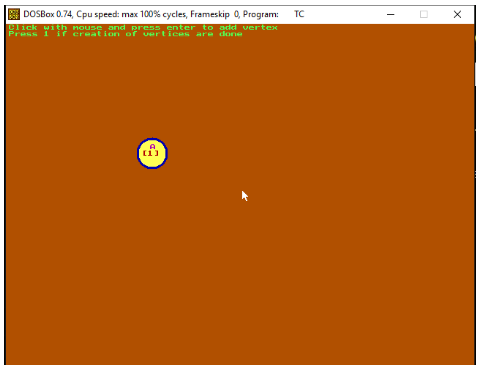
Fig 9: MST with mincost



```
Minimum Cost of this spaning tree is : 14
```

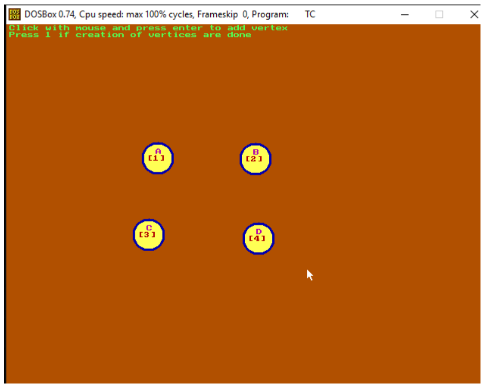
Fig 10: MST with cost

MST from Prims's algorithm



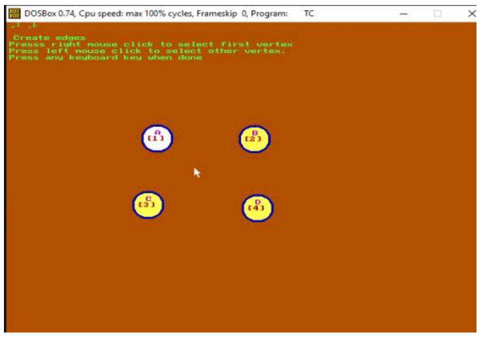
```
Click with mouse and press enter to add vertex
Press 1 if creation of vertices are done
```

Fig 11: 1st vertex created



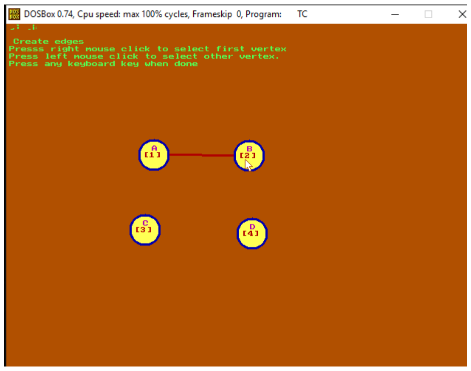
```
Click with mouse and press enter to add vertex
Press 1 if creation of vertices are done
```

Fig 12: 4th vertex created



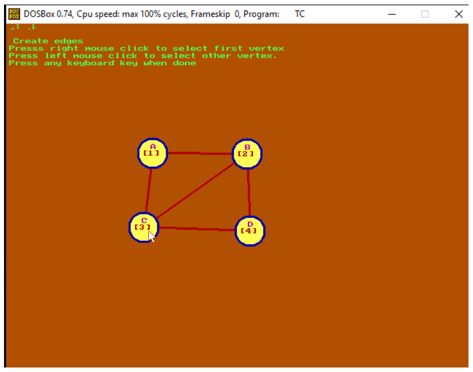
```
 Create edges
Presss right mouse click to select first vertex
Press left mouse click to select other vertex.
Press any keyboard key when done
```

Fig 13: Vertex A selected for edge creation



```
 Create edges
Presss right mouse click to select first vertex
Press left mouse click to select other vertex.
Press any keyboard key when done
```

Fig 14: 1st edge created



```
 Create edges
Presss right mouse click to select first vertex
Press left mouse click to select other vertex.
Press any keyboard key when done
```

Fig 15: 5th vertex created

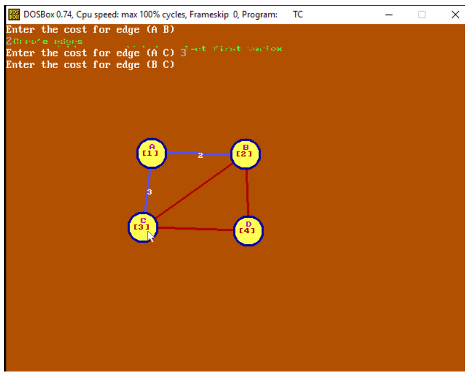

```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip  0, Program:     TC
Enter the cost for edge (A B)
2Create edges
Enter the cost for edge (A C) 3 [illegible]ct first vertex
Enter the cost for edge (B C)
```

Fig 16: Edge cost is inserted for 2nd edge

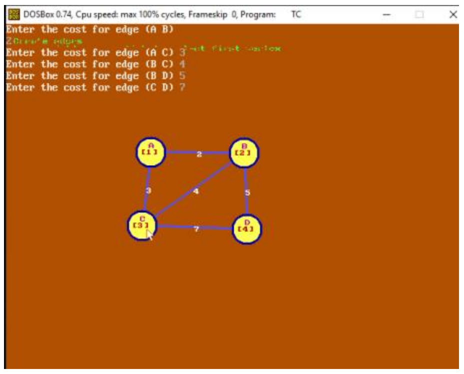

```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip  0, Program:     TC
Enter the cost for edge (A B)
2Create edges
Enter the cost for edge (A C) 3 [illegible]ct first vertex
Enter the cost for edge (B C) 4
Enter the cost for edge (B D) 5
Enter the cost for edge (C D) 7
```

Fig 17 : Edge cost is inserted for 5th edge

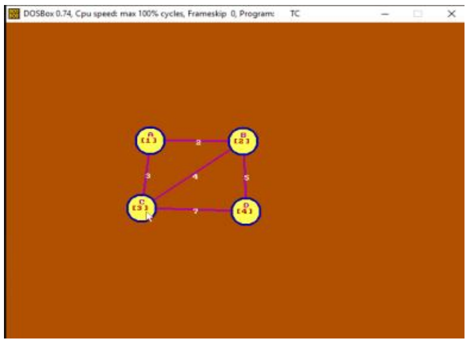
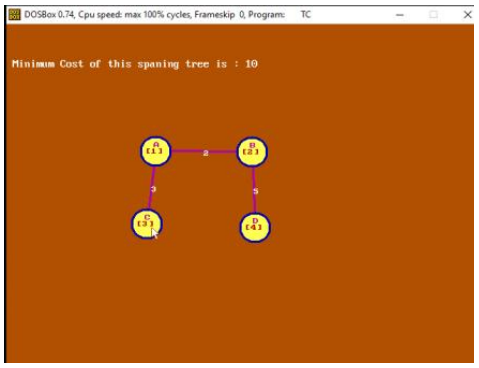

```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip  0, Program:     TC
```

Fig 18: Graph created


```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip  0, Program:     TC

Minimum Cost of this spaning tree is : 10
```

Fig 19: MST with mincost

DISCUSSION

Both Prim's and Kruskal's algorithm results in the formation of minimum spanning tree. Both these algorithms follow different approach to form minimum spanning tree. In Prim's algorithm, trees are formed and that goes on expanding to form minimum spanning tree. In Kruskal's algorithm forests are formed that keeps on merging. A graph G = (V,E) having n number of vertices, time complexity of prim's algorithm is $O(n^2)$ while time complexity of Kruskal's algorithm is O(e log(e)) where e is the number of edges. Kruskal's algorithm looks for smallest remaining edges and create forest avoiding cycle formation. Prim's algorithm looks for smallest edges from which are connected to the previous selected vertices. With this approach, there is no need to take care of cycle formation as this approach will never leads to cycle formation where the cost of edges are positive number. Prim's algorithm is usually used when the graph is dense, i.e. all the vertices are connected to one another as it search all the nearest vertices of the selected vertices. If the graph is sparse then a lot of time would be wasted searching for nearest vertices which will ultimately increase computation time. Kruskal's algorithm can be used here as it searches remaining smallest vertex from the entire graph. Its complexity depends entirely on the number of edges in a graph.

CONCLUSION

It is observed that minimum spanning tree from both these algorithm yields the same result but if graph is dense then performance of Prim's algorithm is better than Kruskal's algorithm. If the graph is sparse then Kruskal's algorithm outperform Prim's algorithm. Performance of Prim's algorithm depends on the number of vertices while performance of Kruskal's algorithm depends on the number of edges assuming numbers of edges are more than the number of vertices in a graph.

REFERENCES

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stin *Introduction to Algorithm*, 3rd ed. The MIT Press, Cambridge, Massachusetts, London, England
[2] Mouse Programming in Turbo C in *electrosofts*. Retrieved April 2, 2016 from http://electrosofts.com/cgraphics/mouse.html
[3] Greedy Algorithm in *Wikipedia*. Retrieved April 2, 2016 from https://en.wikipedia.org/wiki/Greedy_algorithm