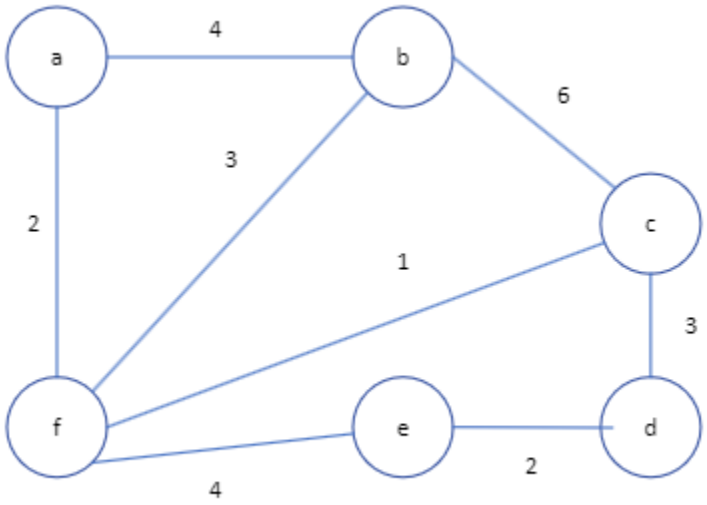
Assignment-4

Due Date: 11:59 pm on Friday, April 29, 2022

This assignment consists of two parts. Each part is 50 pts. The first part is to understand Prim's Algorithm, and the second part is to understand the Dijkstra algorithm.

1) Implement Prim's algorithm for the following vertices and edges to find a minimum spanning tree for a weighted undirected graph.



Code -

```
#include<iostream>

using namespace std;

// Number of vertices in the graph
const int N=6;

int extract_min(int key[], bool isVisited[])
{
   int minimumKey = 99999, min_index;

   for (int v = 0; v < N; v++) {
      // For minimum weighted node, node should not be visited and key is less than other node's key
      if (isVisited[v] == false && key[v] < minimumKey) {
         minimumKey = key[v];
                              min_index = v;
      }
   }
   return min_index;
}



void print(int parent[], int weight[N][N])
{
```

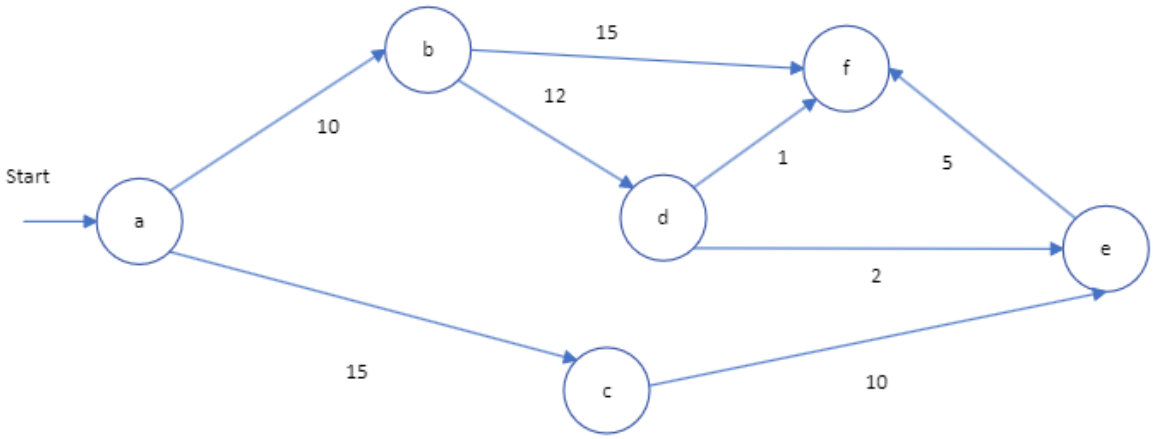
```
   int minWeight=0;
         cout<<"\n\nEdge \tWeight\n";
   for (int i = 1; i< N; i++) {
      char parentNode = 65 + parent[i];
      char currentNode = 65 + i;
                  cout<<parentNode<<" - "<<currentNode<<" \t"<<weight[i][parent[i]]<<" \n";
                  minWeight+=weight[i][parent[i]];
   }
         cout<<"\nTotal cost for PrimsAlgo is "<<minWeight;
}


void primsAlgo(int weight[N][N])
{
   int parent[N], key[N];
   bool isVisited[N];

   // Initialize key array with maz value (99999) and mark isVisited for node as false
   for (int i = 0; i< N; i++) {
      isVisited[i] = false;
      parent[i]=-1;
      key[i] = 99999;
   }

   key[0] = 0;  // weight for First node is 0
   parent[0] = -1; // First node is our root, so no parent node

   for (int x = 0; x < N - 1; x++)
   {
      // Finding the minimum weight of any edge available in the list
      // This extract_min function works like priority queue, it will give minimum cost value node
      int u = extract_min(key, isVisited);
      char currNode = 65 + u;
      cout<<"\nMinimum node is "<<currNode;

      isVisited[u] = true;  // mark current node as visited

      // Update keys of u's non-tree neighbours
      for (int v = 0; v < N; v++)
      {
         // check weight is not zero and current node is not visited and
         // current path weight should be less than total key value
         if (weight[u][v]!=0 && isVisited[v] == false && weight[u][v] < key[v])
         {
            parent[v] = u;  // store visited node
            key[v] = weight[u][v];  // save weight value of current path
         }
      }
   }

         print(parent, weight);
}
```

```
int main()
{
        cout<<"Enter the vertices for a graph with 6 vetices";
  int weight[N][N]={
     {0, 4, 0, 0, 0, 2},
     {4, 0, 6, 0, 0, 3},
     {0, 6, 0, 3, 0, 1},
     {0, 0, 3, 0, 2, 0},
     {0, 0, 0, 2, 0, 4},
     {2, 3, 1, 0, 4, 0}
  };

  cout<<"\n";
  for(int i=0; i<N; i++)
  {
     for(int j=0; j<N; j++)
     {
        cout<<weight[i][j]<<" ";
     }
     cout<<"\n";
  }

  cout<<"\nResult is - \n";
        primsAlgo(weight);

  return 0;
}
```

Result –

```
Enter the vertices for a graph with 6 vetices
0 4 0 0 0 2
4 0 6 0 0 3
0 6 0 3 0 1
0 0 3 0 2 0
0 0 0 2 0 4
2 3 1 0 4 0

Result is -

Minimum node is A
Minimum node is F
Minimum node is C
Minimum node is B
Minimum node is D

Edge    Weight
F - B   3
F - C   1
C - D   3
D - E   2
A - F   2

Total cost for PrimsAlgo is 11
```

2) Implement Dijkstra's algorithm for the following graph to find the shortest path from the source to all destinations.



Code -

```
#include<iostream>
using namespace std;

int calculateMinimumDistance(bool isVisited[], int distance[])
{
   int minimumDistance=999999;
   int nearestNode;

   // calculate minimumDistance from currentNode to its adjacent node
   for(int i=0;i<6;i++)
   {
      if(isVisited[i]==false && distance[i]<=minimumDistance)
      {
         minimumDistance=distance[i];
         nearestNode=i;
      }
   }
   return nearestNode;
}

void calculateDistance(int path[6][6], int src)
{
   int distance[6];
   bool isVisited[6];


   // Initialize all node weights with max int value (999999) and visited as false
   for(int i = 0; i<6; i++)
   {
      distance[i] = 999999;
      isVisited[i] = false;
   }

   distance[src] = 0;

   cout<<"Edge Visited\n"<<endl;
```

```
   for(int j = 0; j<6; j++)
   {
      // calculate minimumDistance from current node to adjacent nodes available in isVisited list
      int min = calculateMinimumDistance(isVisited, distance);

      // Mark minimumDistance node as visited
      isVisited[min]=true;

      for(int k = 0; k<6; k++)
      {
         // Check adjacent nodes are not visited and distance of currentNode is less than total distance calculatd
         if(!isVisited[k] && path[min][k] && distance[min]!=999999 && distance[min]+path[min][k]<distance[k])
         {
            distance[k]=distance[min]+path[min][k];

            char visitedNode = 65 + min;
            char current = 65 + k;

            // print visited edge
            cout<<"Edge "<<visitedNode<<" - "<<current<<"\n";
         }
      }
   }

   cout<<endl;

   cout<<"Vertex\tDistance from source vertex"<<endl;
   for(int k = 0; k<6; k++)
   {
      char currentNode=65+k;
      cout<<currentNode<<"\t\t"<<distance[k]<<endl;
   }
}

int main()
{
   int path[6][6]={
      {0, 10, 15, 0, 0, 0},
      {0, 0, 0, 12, 0, 15},
      {0, 0, 0, 0, 10, 0},
      {0, 0, 0, 0, 2, 1},
      {0, 0, 0, 0, 0, 5},
      {0, 0, 0, 0, 0, 0}
   };

   calculateDistance(path,0);
   return 0;
}
```

Result –

```
Edge Visited

Edge A - B
Edge A - C
Edge B - D
Edge B - F
Edge C - E
Edge D - E
Edge D - F

Vertex  Distance from source vertex
A               0
B               10
C               15
D               22
E               24
F               23
```