Dijkstra's SSSP(Single source shortest path) algorithm

```
//This is an optimized algorithm running in O(E*log(V))
#define INF INT_MAX
const int sz=10001;
vector<pair<int,int> > a[sz];
int dis[sz];
bool vis[sz]= {0};
int parent[sz];

void printPath(int j)
{
  if (parent[j] == - 1)
    return;
  printPath(parent[j]);
  printf("%d ", j);
}

void Dijkstra(int source, int n)
{
  for(int i=0; i<sz; i++)
    dis[i]=INF;
  parent[0] = -1 ;
  ///Custom Comparator for Determining priority for priority queue (shortest edge comes first)
  class prioritize
  {
  public:
    bool operator ()(pair<int, int>&p1,pair<int, int>&p2)
    {
      return p1.second>p2.second;
      }
   };
   priority_queue<pair<int,int>,vector<pair<int,int> >, prioritize> pq;  //Priority queue to store vertex,weight pairs
   pq.push(make_pair(source,dis[source]=0));
   while(!pq.empty())
   {
      pair<int, int> curr= pq.top(); //Current vertex. The shortest distance for this has been found
      pq.pop();
      int cv=curr.first,cw=curr.second; ///'cw' the final shortest distance for this vertex
      if(vis[cv]) ///If the vertex is already visited, no point in exploring adjacent vertices
         continue;
      vis[cv]=true;
      for(int i=0; i<a[cv].size(); i++)
      {
         int v = a[cv][i].first ;
         if(!vis[a[cv][i].first] && a[cv][i].second+cw<dis[a[cv][i].first]) //If this node is not visited and the current parent node distance+distance from there to this node is shorted than the initial distace set to this node, update it
         {
            pq.push(make_pair(a[cv][i].first,(dis[a[cv][i].first]=a[cv][i].second+cw))); //Set the new distance and add to priority queue
            parent[v] = a[cv][i].second;
         }
      }
   }
}
int main() //Driver Function for Dijkstra SSSP
{
   int n,m,x,y,w;//Number of vertices and edges
```

```
    //cout<<"Enter number of vertices and edges in the graph\n";
    cin>>n>>m;
    for(int i=0; i<m; i++) //Building Graph
    {
        cin>>x>>y>>w; //Vertex1, Vertex2, weight of edge
        a[x].push_back(make_pair(y,w));
      //  a[y].push_back(make_pair(x,w));
    }
    //cout<<"Enter source for Dijkstra's SSSP algorithm\n";
    int source;
    cin>>source;
    Dijkstra(source,n);//SSSP from source (Also passing number of vertices as parameter)
    for(int i=0; i<=n; i++) //Printing final shortest distances from source
    {
        cout<<"Vertex: "<<i<<" , Distance: ";
        dis[i]!=INF? cout<<dis[i]<<"\n" : cout<<"-1\n";
    }
}
```