```
                                                                    /*SPRING 20*/
                                                                    /*AKHIL PAL*/
                                                                /*SANMAN PRADHAN*/
/**********************Breadth First Search and Depth first Search**********************/
        #include <iostream>
        #include <string>
        #include <vector>
        #include <queue>
        #include <stack>
        #include <algorithm>

        using namespace std;

        class Node{
                char value;
                vector<Node> child_node;
        public:
                Node(char c){value = c;}

                void addChild(Node n){
                        child_node.push_back(n);
                        return;
                }

                void addChild(char n){
                        Node foo(n);
                        child_node.push_back(foo);
                }

                char getValue(){return value;}
                vector<Node> getChildren(){return child_node;}
                bool isLeaf(){return child_node.size()==0;}
                bool operator==(Node b){return b.value==value;}
        };


        void construct(Node *r)
        {
                string foo;
                cout<<"Enter child_node for "<< r->getValue() <<" (-1 for leaf)"<<endl;
                cin>>foo;

                if(foo == "-1")
                        return;
                else{
                        for(int i = 0; i < foo.length(); i++)
                        {
                                Node t(foo[i]);
```

```
                construct(&t);
                r->addChild(t);
            }
        }
}

string depthFirstSearch(Node root, Node goal)
{
        std::stack<Node> Q;
        std::vector<Node> child_node;
        string path = "--> ";

        Q.push(root);


        while(!Q.empty())
        {
            Node t = Q.top();
            path += t.getValue();
            Q.pop();

            if(t == goal)
            {
              return path;
            }
            child_node = t.getChildren();
            std::reverse(child_node.begin(),child_node.end());

            for (int i = 0; i < child_node.size(); ++i)
            {
              Q.push(child_node[i]);
              Q.push('-');
            }
        }
        return path;
}

string breadthFirstSearch(Node root, Node goal)
{
        std::queue<Node> Q;
        std::vector<Node> child_node;
        string path = "--> ";
        Q.push(root);
        Q.push('-');

        while(!Q.empty())
        {
            Node t = Q.front();
            path += t.getValue();
```

```
                Q.pop();

                if(t == goal)
                {
                        return path;
                }
                child_node = t.getChildren();
                for (int i = 0; i < child_node.size(); ++i)
                {
                        Q.push(child_node[i]);
                        Q.push('-');
                }


        }
        return path;
}


int main(int argc, char** args)
{
        char r;
        cout<<"Enter root node"<<endl;
        cin>>r;

        Node root(r);
        construct(&root);

        cout<<"Enter Node to search for: ";
        cin>>r;

        cout<<endl;

        cout<<"BFS Traversal: "<<breadthFirstSearch(root, Node(' '))<<endl;
        cout<<"BFS Search Path: "<<breadthFirstSearch(root, Node(r))<<endl<<endl;

        cout<<"DFS Traversal: "<<depthFirstSearch(root, Node(' '))<<endl;
        cout<<"DFS Search Path: "<<depthFirstSearch(root, Node(r))<<endl;

        return 0;
}
```

```
/**********************Shortest Path Graph**********************/
  #include <iostream>
  #include <stdio.h>
  #include <limits.h>
  #include <map>
  #include <vector>
  #include <string>
  #include <list>
  #include <functional>

  using namespace std;
  struct vert;
  struct Edge
  {
      vert *dest;
      double dist;
      Edge(vert *d=0,double c=0.0)
       : dest(d),dist(c){}
  };
  struct vert
  {

      string name;
      vector<Edge> adj;
      double dist;
      vert *prev;
      unsigned int scratch;
      vert(const string &character):name(character)
      {reset();}
      void reset()
      {
          dist=3000000;
          prev=0;
          scratch=0;
    }
  };

//Class Declaration
  class graph
  {
      public:
        graph(){};
       ~graph();
        void addlink(const string & source,const string & destination,double dist);
        void showPath(const string & destination) const;
        bool directacyclicgraph(const string & init);
        void printdag();
```

```
   private:
      vert * get_V(const string & v_name);
      void showPath(const vert & dest) const;
      void clearAll();
      typedef map<string,vert *,less<string> > vmap;
      vmap vert_map;
};
graph::~graph()
{
    for(vmap::iterator itr=vert_map.begin();itr!=vert_map.end();++itr)
        delete (*itr).second;
}
vert * graph::get_V(const string & v_name)
{
    vmap::iterator itr=vert_map.find(v_name);
    if(itr==vert_map.end())
    {
        vert *newv=new vert(v_name);
        vert_map[v_name]=newv;
        return newv;
    }
    return (*itr).second;
}
void graph::addlink(const string & source,
        const string & destination,double dist)
{
    vert *v=get_V(source);
    vert *w=get_V(destination);
    v->adj.push_back(Edge(w,dist));
}

void graph::printdag()
{
    for(vmap::iterator itr =vert_map.begin();itr!=vert_map.end();++itr)
     {
        cout<<(*itr).first;
        showPath((*itr).first);
     }
}
void graph::clearAll()
{
    for(vmap::iterator itr=vert_map.begin();itr!=vert_map.end();++itr)
        (*itr).second->reset();
}
void graph::showPath(const vert & dest) const
{
    if(dest.prev!=0)
    {
        showPath(*dest.prev);
```

```
        cout<<"-->";
    }
    cout<<dest.name;
}
void graph::showPath(const string &destination) const
{
    vmap::const_iterator itr=vert_map.find(destination);
    if(itr==vert_map.end())
    {
        cout<<destination<<"is not present in graph"<<endl;
        return;
    }
    vert & w =*(*itr).second;
    if(w.dist==3000000)
        cout<<destination<<"cannot access"<<endl;
    else
    {
        cout<<"(Distance is: "<<w.dist<< ")";
        showPath(w);
    }
    cout<<endl;
}

bool graph::directacyclicgraph(const string & init)
{
    vmap::iterator itr=vert_map.find(init);
    if(itr==vert_map.end())
    {
        cout<<"Vertex not found"<<endl;
        return false;
  }
  clearAll();
  vert *start=(*itr).second;
  start->dist=0;
  list<vert *> q;
  for(itr=vert_map.begin();itr!=vert_map.end();++itr)  //Computation of Indegree of each vertices
  {
        vert *v =(*itr).second;
        for(unsigned int i=0;i<v->adj.size();i++)
        {
            v->adj[i].dest->scratch++;
        }
    }
    for(itr=vert_map.begin();itr!=vert_map.end();++itr)
    {
        vert *v=(*itr).second;              //Make a queue of vertices by pushing them
        if(v->scratch==0)
        {                                   //The vertices having 0 indegree pushed first
            q.push_back(v);
```

```
        }
    }
    unsigned int y;
    for(y=0;!q.empty();y++)
    {
        vert *v=q.front();
        q.pop_front();
        for(unsigned int i =0;i<v->adj.size();i++)
        {
            Edge e=v->adj[i];
            vert *w=e.dest;
            double cvw=e.dist;
            if(--w->scratch==0)
                q.push_back(w);
            if(v->dist==3000000)
                continue;
            if(w->dist>v->dist+cvw)
            {
                w->dist=v->dist+cvw;
                w->prev=v;
            }
        }
    }
    if(y!=vert_map.size())
        {
            cout<<"Cycle is present in graph"<<endl;
            return false;
        }
    else
         return true;
}


int main(int argc, char** argv)
{
    graph g;
    cout<<"Enter the graph description in following pattern:->"<<endl;
    cout<<"SourceCity_Initials | DestinationCity_Initials | Distance"<<endl;

    int a,dist;
    a=1;
    string src,dest;
    while(a==1)
    {
     cin>>src>>dest>>dist;
     g.addlink(src,dest,dist);
     cout<<"To Continue enter 1 or Enter 0 to stop: ";

     cin>>a;
```

```
    }
    cout<<"Enter the Source:";
    cin>>src;
    bool flag;
    flag=g.directacyclicgraph(src);
    if(flag==true)
    {
      cout<<"Shortest path to Distinct nodes are :-->"<<endl;
      g.printdag();
      return 0;
    }
    else
      return 0;
 }
```

/* TEST RUN OUTPUTS BELOW*/

TEST RUN OUTPUTS: -

BFS and DFS traversal

```
sanman@sanman-Inspiron-5558:~/Desktop/OOPM /DAG$ g++ bfsdfs.c
sanman@sanman-Inspiron-5558:~/Desktop/OOPM /DAG$ ./a.out
Enter root node
1
Enter child_node for 1 (-1 for leaf)
6
Enter child_node for 6 (-1 for leaf)
7
Enter child_node for 7 (-1 for leaf)
8
Enter child_node for 8 (-1 for leaf)
9
Enter child_node for 9 (-1 for leaf)
2
Enter child_node for 2 (-1 for leaf)
3
Enter child_node for 3 (-1 for leaf)
-1
Enter Node to search for: 2

BFS Traversal: --> 1-6-7-8-9-2-3-
BFS Search Path: --> 1-6-7-8-9-2

DFS Traversal: --> 1-6-7-8-9-2-3
DFS Search Path: --> 1-6-7-8-9-2
sanman@sanman-Inspiron-5558:~/Desktop/OOPM /DAG$
```

Invalid Output and Cyclic error :-

```
sanman@sanman-Inspiron-5558:~/Desktop/OOPM /29th April$ g++ graphak.c
sanman@sanman-Inspiron-5558:~/Desktop/OOPM /29th April$ ./a.out
Enter the Graph description in following pattern:->
SourceCity_Initials | DestinationCity_Initials | Cost
NY   LA   10
To Continue enter 1 or Enter 0 to stop: 1
SA   SJC   5
To Continue enter 1 or Enter 0 to stop: 1
NY   SJC   18
To Continue enter 1 or Enter 0 to stop: 0
Enter the Source:SJC
Shortest path to Distinct nodes are :-->
LALAcannot access

NYNYcannot access

SASAcannot access

SJC(Cost is: 0)SJC
sanman@sanman-Inspiron-5558:~/Desktop/OOPM /29th April$ g++ graphak.c
^[[Asanman@sanman-Inspiron-5558:~/Desktop/OOPM /29th April$ ./a.out
Enter the Graph description in following pattern:->
SourceCity_Initials | DestinationCity_Initials | Cost
LA NY 10
To Continue enter 1 or Enter 0 to stop: 1
CH NY 5
To Continue enter 1 or Enter 0 to stop: 1
LA PH 12
To Continue enter 1 or Enter 0 to stop: 1
NY SJC 15
To Continue enter 1 or Enter 0 to stop: 1
SJC LA 8
To Continue enter 1 or Enter 0 to stop: 0
Enter the Source:LA
Cycle is present in graph
```

Output distances between the cities/states of USA based on user input:-

```
sanman@sanman-Inspiron-5558:~/Desktop/OOPM /29th April$ ./a.out
Enter the Graph description in following pattern:->
SourceCity_Initials | DestinationCity_Initials | Cost
la ny 10
To Continue enter 1 or Enter 0 to stop: 1
ny ph 8
To Continue enter 1 or Enter 0 to stop: 1
ph fl 3
To Continue enter 1 or Enter 0 to stop: 1
fl tx 15
To Continue enter 1 or Enter 0 to stop: 0
Enter the Source:la
Shortest path to Distinct nodes are :-->
fl(Cost is: 21)la-->ny-->ph-->fl
la(Cost is: 0)la
ny(Cost is: 10)la-->ny
ph(Cost is: 18)la-->ny-->ph
tx(Cost is: 36)la-->ny-->ph-->fl-->tx
sanman@sanman-Inspiron-5558:~/Desktop/OOPM /29th April$
```

Diagram of Capital Cities