10.Prim

```
/* Roll no.     : 2002
   Batch        : E-10
*/

#include <iostream>
#include<stdlib.h>
#include "graph.h"
using namespace std;

int main()
{
        graph g;

        do
                {
                        int ch;
                        cout<<"\tMENU\n";

                        cout<<"\t\t1.Enter Adjecency list\n\t\t2.Display graph\n\t\t3.Prim\n\t\t"
                                "4.Display minimum spanning tree"
                                "\n\t\t5.Re_Enter\n\t\t6.Exit\n";
                        cout<<"Enter Choice : ";
                        cin>>ch;
                        cout<<endl;

                        switch(ch)
                        {

                                case 1:
                                                g.create();
                                                break;

                                case 2:
                                                g.display();
                                                break;

                                case 3:
                                                g.prim();
                                                break;

                                case 4:
                                                g.display_mst();
                                                break;

                                case 5:
                                                main();
                                                break;

                                case 6:
                                                exit(0);
```

```
                                                        break;

                                }
                        }
                        while(1);
        return 0;

}
```

graph.h

```
/*
 * graph.h
 *
 *  Created on: 21-Feb-2018
 *      Author: e2002
 */
#include<iostream>
#include<string>
using namespace std;

struct frnd
{
	string f_name;
	int dist;
	frnd * link;
	int visited;

	struct node * current_vertex;

};

struct node
{
	string Name;
	int id;
	node * down;
	frnd * next;
	int visited;
};

#ifndef ADJ_LIST_H_
#define ADJ_LIST_H_

class graph
{
	node * root;
	int n;
	int cost;
public:
		graph();
		void create();
		frnd * add_friend(node *);
		void display();
		node * search(string);
```

```
                node * min_edge(node *,node *);
                //node * prims();
                void prim();
                void display_mst();
};
#endif /* ADJ_LIST_H_ */
```

graph.cpp

```
#include "graph.h"
#include<iostream>
#include<string>
#include<malloc.h>
#include<iomanip>
#include<queue>
using namespace std;

graph :: graph()
{
        root=NULL;
        n=0;
        cost=0;
}

void getdata(node * p)
{
        cout<<"Enter Name : ";
        cin>>p->Name;
        cout<<endl;

        p->visited=0;

}

node * graph :: search(string val)
{
        if(root->Name == val)
                return root;

        node * p;
        p=root;

        while(p != NULL && p->Name != val)
                p=p->down;;

        if(p->Name == val)
                return p;

        else
                return NULL;
}
```

```
frnd * graph:: add_friend(node * root)
{
        frnd *head,*p,*q;

                head = new frnd;

                cout<<"Enter Name of city no. 1 connected to "<<root->Name<<" : ";
                cin>>head->f_name;
                cout<<endl;

                cout<<"Enter distance between these cities : ";
                cin>>head->dist;
                cout<<endl;

                head->visited=0;

                node * this_node;
                this_node=search(head->f_name);
                head->current_vertex=this_node;

                head->link=NULL;

                p=head;
                int i=1;

                cout<<"More cities connected?(Y/N) to "<<root->Name<<" : ";
                char ch;
                cin>>ch;
                cout<<endl;

                if(ch=='N' || ch=='n')
                        return head;


                do
                {
                        q=new frnd;

                        cout<<"Enter name of city  no. "<<i+1<<" : ";
                        cin>>q->f_name;
                        cout<<endl;

                        cout<<"Enter distance between these cities : ";
                        cin>>q->dist;
                        cout<<endl;

                        q->visited=0;

                        node * this_node;
                        this_node=search(q->f_name);
                        q->current_vertex=this_node;
```

```
                q->link=NULL;

                p->link=q;
                p=q;

                i++;

                cout<<"More cities connected?(Y/N)? to "<<root->Name<<" : ";
                char ch;
                cin>>ch;
                cout<<endl;

                if(ch=='N' || ch=='n')
                        break;

        }
        while(1);

        return head;
}

void graph:: create()
{
        cout<<"Enter number of cities : ";
        cin>>n;
        cout<<endl;

        node *p,*q;
        root = new node;

        cout<<"Enter name of city no. 1 : ";
        cout<<endl<<endl;
        getdata(root);
        cout<<endl;

        root->id=0;

        root->down=NULL;
        root->next=NULL;

        p=root;

        for(int i=1;i<n;i++)
        {
                q=new node;

                cout<<"Enter name of city no. "<<i+1<<" : ";
                cout<<endl<<endl;
                getdata(q);
                cout<<endl;

                q->id=i;
```

```
                q->down=NULL;
                q->next=NULL;

                p->down=q;
                p=q;

        }

        p=root;

        while(p != NULL)
        {
                p->next=add_friend(p);
                p=p->down;
        }

}



void graph :: display()
{
        node *head;

        head=root;

        while(head != NULL)
        {
                cout<<""<<head->Name;

                frnd  *q;
                        q=head->next;

                        while(q != NULL)
                        {
                                cout<<"<- "<<q->f_name;
                                q=q->link;
                        }

                cout<<endl;
                head=head->down;
        }

        cout<<"NULL";
        cout<<endl;
}


void graph :: prim()
{
        root->visited=1;
```

```
	bool tree_complete=false;

	while(! tree_complete)
	{
		tree_complete=true;

		node * current;
		current=root;
		frnd * min_edge=NULL;

		int min_dist=999;

		while(current != NULL)
		{
			if(current->visited == 1)
			{

				frnd * f;
				f=current->next;

				while(f != NULL)
				{

					if((f->current_vertex)->visited == 0)
					{
						tree_complete=false;
						if(f->dist < min_dist)
						{
							min_dist=f->dist;
							min_edge=f;
						}
					}

					f=f->link;
				}
			}

			current = current->down;
		}

		if(min_edge != NULL)
		{
			min_edge->visited=1;
			cout<<"This is marked in friends : "<<min_edge->f_name<<endl<<endl;
			(min_edge->current_vertex)->visited=1;
		}
	}

}
```

```
void graph :: display_mst()
{
	node *head;
		head=root;

		while(head != NULL)
		{
			cout<<setw(5)<<""<<head->Name<<" :: ";

			frnd  *q;
				q=head->next;

				while(q != NULL )
				{
					if(q->visited==1)
					{
						cout<<setw(5)<<"<- "<<q->f_name;
						cout<<"( "<<q->dist<<" )"<<"";

						cost += q->dist;
					}
					q=q->link;
				}

			cout<<endl;
			head=head->down;
		}

		cout<<"Min dist : "<<cost<<endl;
		cout<<endl;
}
```