8. Dijkstra

```
/*Roll No.  2002
  Batch      E-10
*/

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

int main()
{
        graph g;
        int src;

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

                        cout<<"\t\t1.Create graph\n\t\t2.Display adjacency
list\n\t\t3.Dijkstra\n\t\t4.Distance"
                                        "\n\t\t5.Exit\n\n";
                        cout<<"Enter Choice : ";
                        cin>>ch;
                        cout<<endl;

                        switch(ch)
                        {
                        case 1 :        g.getdata();
                                        cout<<endl<<endl;
                                        break;

                        case 2 :        g.display_adj_list();
                                        cout<<endl<<endl;
                                        break;

                        case 3 :        cout<<"Enter source : ";
                                        cin>>src;
                                        cout<<endl<<endl;

                                        g.dijkstras(src);
                                        cout<<"Dijkstra successfully applied !!!\n\n ";
                                        cout<<endl<<endl;
                                        break;

                        case 4 :        g.print_solution(src);
                                        cout<<endl<<endl;
                                        break;

                        case 5 :        exit(0);
```

```
                        }
                    }
        while(1);

        return 0;
}
```

graph.cpp

```
/*
 * graph.cpp
 *
 *  Created on: 12-Mar-2018
 *      Author: deception
 */

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

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

	cout<<"Enter adjecency matrix :: \n";

	for(int i=0;i<n;i++)
	{
		for(int j=0;j<n;j++)
			cin>>G[i][j];
	}
}

void graph:: display_adj_list()
{
	cout<<"Given graph in form of adjacency list is :: \n";

	for(int i=0;i<n;i++)
	{
		for(int j=0;j<n;j++)
			cout<<G[i][j]<<" ";

		cout<<endl;
	}
}

void graph:: unvisit()
{
	for(int i=0;i<20;i++)
	{
		dist[i]=99999;
		finalized[i]=false;
	}
}

int graph:: pick_min()
```

```
{
	int min=99999,min_idx=0;

	for(int i=0;i<n;i++)
	{
		if(! finalized[i] && dist[i]<=min)
		{
			min=dist[i];
			min_idx=i;
		}
	}

	return min_idx;

}

void graph:: dijkstras(int src)
{
	unvisit();
	dist[src]=0;

	path_idx=0;
	//path[path_idx++]=src;

	for(int i=0;i<n;i++)
	{
		int current = pick_min();

		finalized[current]=true;

		for(int next=0;next<n;next++)
		{
			if( (! finalized[next]) && (G[current][next]) &&
					( dist[current] + G[current][next] < dist[next]) &&
(dist[current]!= 99999) )
				dist[next]= dist[current] + G[current][next]  ;

		}

		path[path_idx++]=current;

	}
}

void graph:: print_solution(int src)
{
		int width=10;

		cout<<"Nodes   :: Path              :: Distance\n";

		for(int i=0;i<path_idx;i++)
		{
```

```
                cout<<src<<" -> "<<path[i]<<"  :: ";
                for(int j=0;j<=i;j++)
                        cout<<path[j]<<" ";

                cout<<setw(width--);

                cout<<"  :: "<<dist[path[i]]<<endl<<endl;

        }
}
```

graph.h

```
/*
 * graph.h
 *
 *  Created on: 12-Mar-2018
 *      Author: deception
 */

#ifndef GRAPH_H_
#define GRAPH_H_

class graph
{
	int n,dist[20];
	int G[100][100];
	int path[20];
	int path_idx;

	bool finalized[20];

public:
	void getdata();
	void display_adj_list();

	void unvisit();
	int pick_min();
	void dijkstras(int);
	void print_solution(int);
};

#endif /* GRAPH_H_ */
```