CSE 2003: Lab Assignment #14

Due on Thursday, April 19, 2017

Prof. Shaik Naseera 2:00pm

Jacob John

Contents

Problem 1

Write a C program to Implement Dijkstra's shortest path algorithm

Listing 1: Dijkstra's shortest path program in C

```
/*Program to find shortest distances using Dijkstra's algorithm */
#include<stdio.h>

#define MAX 100
#define TEMP 0
#define PERM 1
#define infinity 9999
#define NIL -1

void findPath(int s, int v );
void Dijkstra( int s);
int min_temp( );
void create_graph();

int n;    /* Denotes number of vertices in the graph */
int adj[MAX][MAX];
int predecessor[MAX];    /*predecessor of each vertex in shortest path*/
int pathLength[MAX];
int status[MAX];

int main()
{
     int s,v;

     create_graph();

     printf("Enter source vertex : ");
     scanf("%d",&s);

     Dijkstra(s);

     while(1)
     {
          printf("Enter destination vertex(-1 to quit): ");
          scanf("%d",&v);
          if(v == -1)
               break;
          if(v < 0 || v >= n )
               printf("This vertex does not exist\n");
          else if(v == s)
               printf("Source and destination vertices are same\n");
          else if( pathLength[v] == infinity )
            printf("There is no path from source to destination vertex\n");
          else
               findPath(s,v);
     }
}/*End of main()*/
```

Problem 1 continued on next page. . .

```
void Dijkstra( int s)
{
     int i,current;

     /* Make all vertices temporary */
     for(i=0; i<n; i++)
     {
          predecessor[i] =  NIL;
          pathLength[i] = infinity;
          status[i] = TEMP;
     }
     /* Make pathLength of source vertex equal to 0 */
     pathLength[s] = 0;

     while(1)
     {
          /*Search for temporary vertex with minimum pathLength
          and make it current vertex*/
          current = min_temp( );

          if( current == NIL )
               return;

          status[current] = PERM;

          for(i=0; i<n; i++)
          {
               /*Checks for adjacent temporary vertices */
               if ( adj[current][i] !=0 && status[i] == TEMP )
                    if( pathLength[current] + adj[current][i] < pathLength[i] )
                    {
                         predecessor[i] = current;  /*Relabel*/
                         pathLength[i] = pathLength[current] + adj[current][i];
                    }
          }
     }
}/*End of Dijkstra( )*/

/*Returns the temporary vertex with minimum value of pathLength
  Returns NIL if no temporary vertex left or
  all temporary vertices left have pathLength infinity*/
int min_temp( )
{
     int i;
     int min = infinity;
     int k = NIL;
     for(i=0;i<n;i++)
     {
          if(status[i] == TEMP && pathLength[i] < min)
          {
               min = pathLength[i];
               k = i;
          }
```

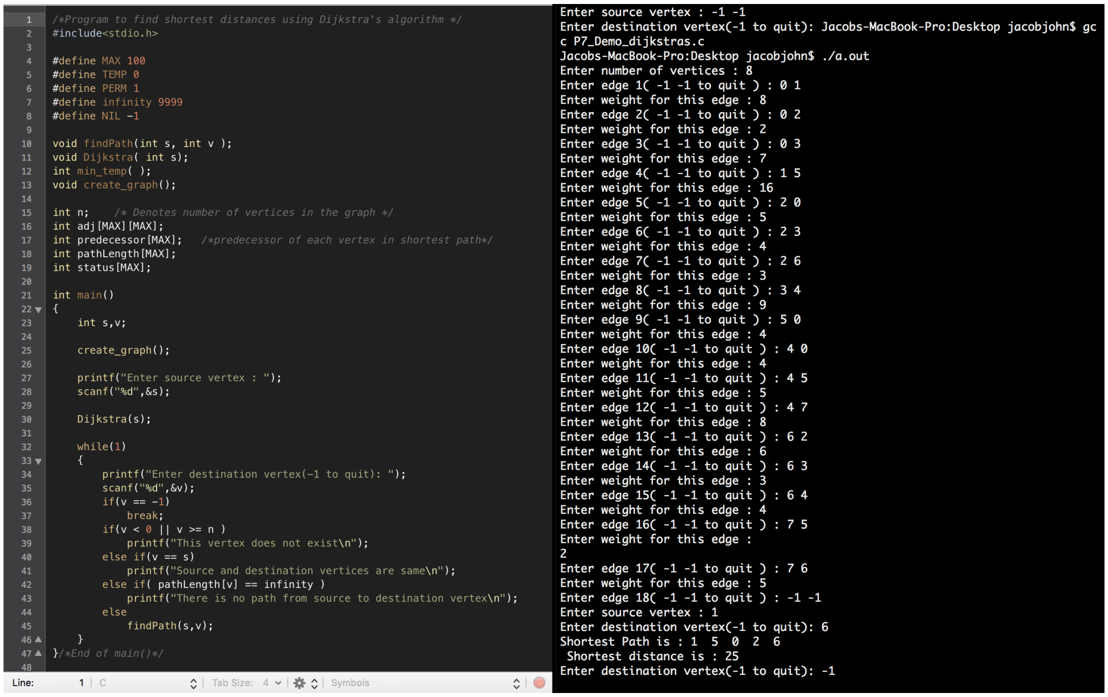
Problem 1 continued on next page. . .

```
     }
     return k;
}/*End of min_temp( )*/


void findPath(int s, int v )
{
     int i,u;
     int path[MAX];      /*stores the shortest path*/
     int shortdist = 0;  /*length of shortest path*/
     int count = 0;      /*number of vertices in the shortest path*/

     /*Store the full path in the array path*/
     while( v != s )
     {
          count++;
          path[count] = v;
          u = predecessor[v];
          shortdist += adj[u][v];
          v = u;
     }
     count++;
     path[count]=s;

     printf("Shortest Path is : ");
     for(i=count; i>=1; i--)
          printf("%d  ",path[i]);
     printf("\n Shortest distance is : %d\n", shortdist);
}/*End of findPath()*/

void create_graph()
{
     int i,max_edges,origin,destin, wt;

     printf("Enter number of vertices : ");
     scanf("%d",&n);
     max_edges = n*(n-1);

     for(i=1;i<=max_edges;i++)
     {
          printf("Enter edge %d( -1 -1 to quit ) : ",i);
          scanf("%d %d",&origin,&destin);

          if( (origin == -1) && (destin == -1) )
               break;

          printf("Enter weight for this edge : ");
          scanf("%d",&wt);

          if( origin >= n || destin >= n || origin<0 || destin<0)
          {
               printf("Invalid edge!\n");
               i--;
```

Problem 1 continued on next page. . .

```
                    }
                    else
                            adj[origin][destin] = wt;
            }
}
```

Output:



```
/*Program to find shortest distances using Dijkstra's algorithm */
#include<stdio.h>

#define MAX 100
#define TEMP 0
#define PERM 1
#define infinity 9999
#define NIL -1

void findPath(int s, int v );
void Dijkstra( int s);
int min_temp( );
void create_graph();

int n;    /* Denotes number of vertices in the graph */
int adj[MAX][MAX];
int predecessor[MAX];   /*predecessor of each vertex in shortest path*/
int pathLength[MAX];
int status[MAX];

int main()
{
    int s,v;

    create_graph();

    printf("Enter source vertex : ");
    scanf("%d",&s);

    Dijkstra(s);

    while(1)
    {
        printf("Enter destination vertex(-1 to quit): ");
        scanf("%d",&v);
        if(v == -1)
            break;
        if(v < 0 || v >= n )
            printf("This vertex does not exist\n");
        else if(v == s)
            printf("Source and destination vertices are same\n");
        else if( pathLength[v] == infinity )
            printf("There is no path from source to destination vertex\n");
        else
            findPath(s,v);
    }
}/*End of main()*/
```

```
Enter source vertex : -1 -1
Enter destination vertex(-1 to quit): Jacobs-MacBook-Pro:Desktop jacobjohn$ gc
c P7_Demo_dijkstras.c
Jacobs-MacBook-Pro:Desktop jacobjohn$ ./a.out
Enter number of vertices : 8
Enter edge 1( -1 -1 to quit ) : 0 1
Enter weight for this edge : 8
Enter edge 2( -1 -1 to quit ) : 0 2
Enter weight for this edge : 2
Enter edge 3( -1 -1 to quit ) : 0 3
Enter weight for this edge : 7
Enter edge 4( -1 -1 to quit ) : 1 5
Enter weight for this edge : 16
Enter edge 5( -1 -1 to quit ) : 2 0
Enter weight for this edge : 5
Enter edge 6( -1 -1 to quit ) : 2 3
Enter weight for this edge : 4
Enter edge 7( -1 -1 to quit ) : 2 6
Enter weight for this edge : 3
Enter edge 8( -1 -1 to quit ) : 3 4
Enter weight for this edge : 9
Enter edge 9( -1 -1 to quit ) : 5 0
Enter weight for this edge : 4
Enter edge 10( -1 -1 to quit ) : 4 0
Enter weight for this edge : 4
Enter edge 11( -1 -1 to quit ) : 4 5
Enter weight for this edge : 5
Enter edge 12( -1 -1 to quit ) : 4 7
Enter weight for this edge : 8
Enter edge 13( -1 -1 to quit ) : 6 2
Enter weight for this edge : 6
Enter edge 14( -1 -1 to quit ) : 6 3
Enter weight for this edge : 3
Enter edge 15( -1 -1 to quit ) : 6 4
Enter weight for this edge : 4
Enter edge 16( -1 -1 to quit ) : 7 5
Enter weight for this edge :
2
Enter edge 17( -1 -1 to quit ) : 7 6
Enter weight for this edge : 5
Enter edge 18( -1 -1 to quit ) : -1 -1
Enter source vertex : 1
Enter destination vertex(-1 to quit): 6
Shortest Path is : 1  5  0  2  6
 Shortest distance is : 25
Enter destination vertex(-1 to quit): -1
```