

Dijkstra Sequence

Author's Name: Zhang Youchao

Date: 2010-12-4

Chapter 1: Introduction

Dijkstra sequence: a set contains vertices included in shortest path tree is maintained. During each step, we find one vertex which is not yet included and has a minimum distance from the source, and collect it into the set. Hence step by step an ordered sequence of vertices is generated by Dijkstra's algorithm.

There could be more than one Dijkstra sequence. I am going to check whether a given sequence is Dijkstra sequence or not.

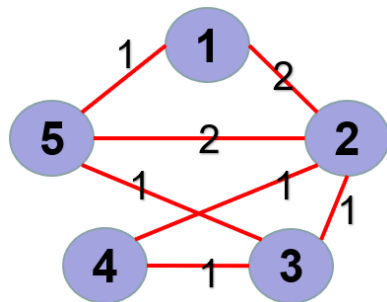


Fig 1. Sample input Graph
(undirected)

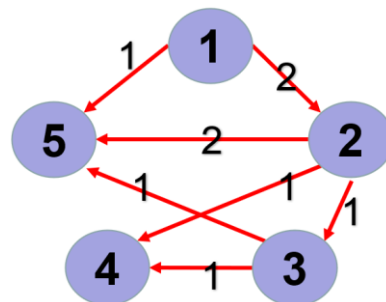


Fig 2. Sample input Graph
(directed)

The question stem does NOT specify whether it is a directed graph but from the sample we can analyze it is an undirected graph.

Chapter 2: Algorithm Specification

1. The data structure

```
/*Build adjacency matrix*/
int G[Max_Num_Vertex][Max_Num_Vertex] = {0};
```

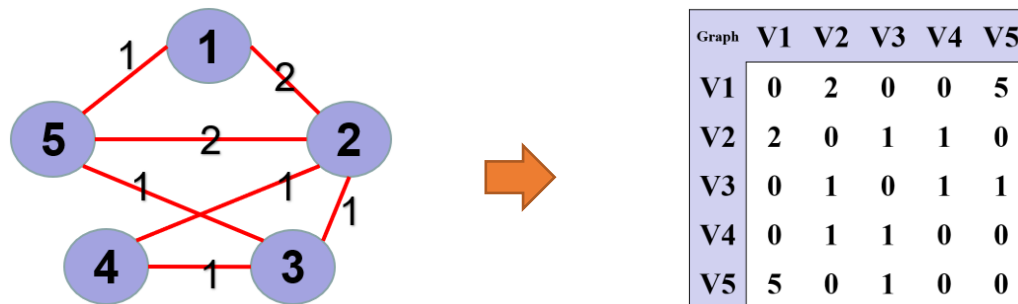


Fig 3. Undirected Graph to Adjacency Matrix

2. Dijkstra's algorithm is one of the very famous greedy algorithms. It is used for solving the single source shortest path problem which gives the shortest paths from one particular source vertex to all the other vertices of the given graph.

Algorithm1: Dijkstra

Input : Graph G, Weight w, Start point s;

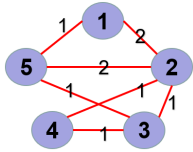
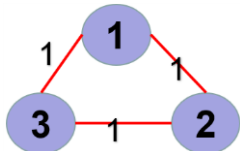
Output: 0→wrong 1→right

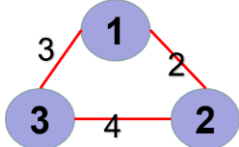

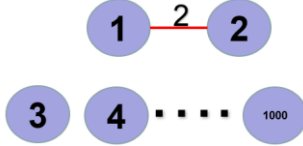
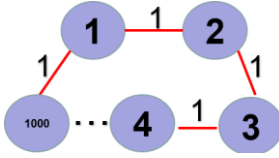
```
1:  function Dijkstra(G, w, s)
2:      for each vertex v in V[G]
3:          d[v] := infinity
4:          previous[v] := undefined
5:  d[s] := 0
6:  S := empty set
7:  Q := set of all vertices
8:  while Q is NOT an empty set
9:      u := Extract_Min(Q)
10:     S.append(u)
11:     for each edge outgoing from u as (u,v)
12:         if d[v] > d[u] + w(u,v)
13:             d[v] := d[u] + w(u,v)
14:             previous[v] := u
```

To judge whether the seq is a **Dijkstra sequence**: The 'distance' array of Dijkstra sequence is a non-descending sequence. If descending, NOT a Dijkstra sequence

Chapter 3: Testing Results

Result table:

Test case	a brief description of the purpose	Expected result	Actual behavior	Possible cause of a bug	Current status
 5 7 1 2 2 1 5 1 2 3 1 2 4 1 2 5 2 3 5 1 3 4 1 4 5 1 3 4 2 5 3 1 2 4 2 3 4 5 1 3 2 1 5 4	To test whether the program can detect the input given by the project	YES YES YES NO	YES YES YES NO	NO	Pass
 3 3 1 2 1 1 3 1 2 3 1 6 1 2 3 1 3 2 2 1 3 2 3 1 3 1 2 3 2 1	To test whether the program can detect the simple input	YES YES YES YES YES YES	YES YES YES YES YES YES	NO	Pass

 <p>3 3 1 2 2 1 3 3 2 3 4 6 1 2 3 1 3 2 2 1 3 2 3 1 3 1 2 3 2 1</p>	To test whether the program can detect the simple input	YES NO YES NO YES NO	YES NO YES NO YES NO	NO	Pass
 <p>1 0 1 1</p>	To test whether the program can detect the simplest input	YES	YES	NO	Pass
 <p>1000 1 1 2 2 2 2 1 3 4 5.....1000 1 2 3 4 5.....1000</p>	To test whether the program can detect the Max_Num_Vertex input	YES YES	YES YES	NO	Pass
 <p>1000 1000 1 2 1 2 3 1 999 1000 1 1000 1 1 1 1 2 3...1000</p>	To test whether the program can detect the Max_Num_Vertex input	No	NO	NO	Pass

Chapter 4: Analysis and Comments

Analysis of the time and space complexities of the algorithms. Comments on further possible improvements.

Algorithm1 Dijkstra:

To find the smallest unknown distance vertex I simply scan the table so the time complexity is $O(V)$. There is a loop deal with the set of all vertices and there is a nother loop also deal with the set of all vertices embedded in so the time complexity of this part is $O(V^2)$. We need to compare all edges to determine if we need to update the path so the time complexity of this part is $O(E)$.

In conclusion, the time complexity of this Dijkstra is $O(E + V^2)$.

The data structure is adjacency matrix, so the space complexity is $O(V^2)$

Further possible improvements

1. V = smallest unknown distance vertex;
/* keep distances in a priority queue and call DeleteMin – $O(\log|V|)$ */
2. The writing of function names and other code specifications can be improved, and when time permits, other methods can be devised to solve the practical problem.
3. Other improvements: Pairing heap and Fibonacci heap

Appendix: Source Code (in C)

At least 30% of the lines must be commented. Otherwise the code will NOT be evaluated.

```
#include <stdio.h>
#include <stdlib.h>

#define Max_Num_Vertex 1001 //Max_Num_Vertex
#define Max_Num_Edge 100001 //Max_Num_Edge
#define Infinity 100000000 //Infinity

int G[Max_Num_Vertex][Max_Num_Vertex] = {0}; /*Build adjacency matrix*/
int visit[Max_Num_Vertex] = {0}; /*To record weather the point is visited*/
int distance[Max_Num_Vertex] = {0}; /*To record the distance from current point to start po
int*/

/*Graph structure initialization*/
void Init_graph()
{
    for (int i = 0; i < Max_Num_Vertex; i++)
        for (int j = 0; j < Max_Num_Vertex; j++)
            G[i][j] = Infinity;
}

/*To solve the problem*/
int Solve(int seq[], int num)
{
    int start = seq[1]; //seq[1] is the start point
    Dijkstra(start, num); //use the Dijkstra to find the shortest path to start point
    /*To judge weather the array 'distance' is nondescending sort*/
    for (int i = 1; i < num; ++i)
        if (distance[seq[i]] > distance[seq[i + 1]]) //if descending sort, not a Dijkstra sequence
            return 0;
    return 1;
}

/*Impliment the Dijkstra*/
void Dijkstra(int start, int num)
{
    /*To record the distance from start to start is 0*/
    for (int i = 0; i <= num; ++i)
        distance[i] = G[i][start];
    distance[start] = 0; //The distance from start to start is 0
    visit[start] = 1; //We visited the start just now

    int cnt = 1; //To count the number of Vertex
    while (cnt < num) //The loop should find the shortest path from all the vertex to start point
```

```

{
    cnt++;

    /*Find the smallest unknown distance vertex*/
    int minedge = Infinity; //initialize the minedge
    int vertex = -1;        //initialize the smallest unknown distance vertex
    for (int i = 1; i <= num; ++i)
    {
        if (visit[i] == 0 && distance[i] < minedge)
        {
            vertex = i;
            minedge = distance[i];
        }
    }
    visit[vertex] = 1; //We visited the smallest unknown distance vertex just NOW

    /*For each unknown vertex adjacent from the smallest unknown distance vertex */
    for (int j = 1; j <= num; ++j)
    {
        if (G[j][vertex] == Infinity) //if the distance is Inf which means that the 2 vertex is
NOT connected
            continue;
        if (visit[j] == 0 && distance[vertex] + G[vertex][j] < distance[j]) //if NOT visited and
new distance is smaller
        {
            distance[j] = distance[vertex] + G[vertex][j]; //update the distance from current di
stance to start point
        }
    }
}
}

int main()
{
    Init_graph(); /*Graph structure initialization*/

    int v = 0, e = 0; //Number of Vertex and Edge
    scanf("%d%d", &v, &e);

    for (int i = 0; i < e; i++) /*Build adjacency matrix*/
    {
        int from = 0, to = 0, weight = 0;
        scanf("%d%d%d", &from, &to, &weight);
        G[from][to] = weight; //Undirect Graph
        G[to][from] = weight;
    }
}

```



```

}

int seq[Max_Num_Vertex] = {0}; /*Initialize the test sequence*/
int k = 0;
scanf("%d", &k); //Input the number of test sequence
for (int i = 0; i < k; i++)
{
    for (int j = 1; j <= v; j++) /*Input the test sequence*/
        scanf("%d", seq + j);
    if (Solve(seq, v))
        printf("YES\n");
    else
        printf("NO\n");
    for (int i = 0; i < Max_Num_Vertex; ++i) /*refresh visit[] to justify next sequence*/
        visit[i] = 0;
}
system("pause");
return 0;
}

```

Declaration

I hereby declare that all the work done in this project titled " Dijkstra Sequence " is of my independent effort.