Dijkstra Sequence

Name:姜雨童

Date:2023.11.28

Chapter 1： Introduction

1 Background

Dijkstra's algorithm is a popular algorithm in computer science, specifically in the field of graph theory. It is used to find the single source shortest path between two nodes in a weighted graph (without negative weight). The algorithm was conceived by Dutch computer scientist Edsger Dijkstra in 1956. This is a **greedy algorithm**, but unlike typical greedy algorithms, it can find the **global optimum**.

2 Problem

For a given graph, there could be more than one Dijkstra sequence. And the program is asked to check whether a given sequence is Dijkstra sequence or not.

2.1 Input

For each case, the first line contains two positive integers **Nv** ($\leq 10^3$, the number of vertices) and **Ne** ($\leq 10^5$, the number of edges). As a consequence, the vertices are numbered from 1 to Nv.
Then Ne lines follow, each describes an edge by giving **two vertices** and the **positive integer weight** (≤ 100). (It is guaranteed that the given graph is connected.)
Finally a positive integer **K** (≤ 100) is the number of sequence, followed by K lines of **sequences**, each contains a permutation of the Nv vertices. (It is the sequence to judge.)

2.2 Output

For each of the K sequences, print in a line `Yes` if it is a Dijkstra sequence, or `No` if not.

Chapter 2： Algorithm Specification

1 main data structures

```
int M[MaxNum][MaxNum] = {0}; /* the adjacent matrix of the graph */
int Seq[MaxNum]; /* the array for sequence */
int Sel[MaxNum] = {0}, Dist[MaxNum] = {0};
/* the array for selected vertices, and the array for their distances */
```

In this project, I use an 2D arrry M as the adjacency matrix of a graph. And other arrays to facilitate the implementation of Dijkstra's algorithm and track the selected vertices and their distances. (Seq: stores the input sequence of vertices, Sel: represents the selected vertices, Dist: the shortest distance from the source vertex to each vertex.)

2 main algorithm

The main algorithm is the Dijkstra Algorithm, and this program also consists only of the **main** function and the **Dijkstra** function. The main function reads the graph and builds the adjacent matrix, and the Dijkstra function reads the sequence and determines whether it is a Dijkstra sequence.

And the pseudocode of Dijkstra Function is listed:

```
Function Dijkstra():
    Seq[0] = LoadSequence()
    Sel[0] = Seq[0]

    for i from 1 to Nv - 1:
        for p from i + 1 to Nv - 1:
            if !Dist[Seq[p]] && Dist[Seq[p]] < Dist[Seq[i]]
                return 0

        Sel[i] = Seq[i]

        for p from i + 1 to Nv - 1:
            if !M[Seq[i]][Seq[p]]:
                if Dist[Seq[p]] || Dist[Seq[p]] > Dist[Seq[i]] + Cip
                    Dist[Seq[p]] = Dist[Seq[i]] + Cip
    return 1
```

Chapter 3: Testing Results

\	Input	Expected Output	Actual Output	Current status
test sample given by PTA	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	Yes Yes Yes No	Yes Yes Yes No	pass
the minimum graph	1 0 1 1	Yes	Yes	pass
extra test sample	6 6 1 2 2 1 3 1 1 4 1 2 5 3 3 4 3 4 6 2 4 4 1 3 2 5 6 4 1 3 6 2 5 4 1 3 6 2 5 3 1 4 6 2 5	No Yes Yes No	No Yes Yes No	pass

Besides, it also pass the judge of PTA. link:1163

测试点	提示	内存(KB)	用时(ms)	结果	得分
0		384	4	答案正确	15 / 15
1		400	5	答案正确	8 / 8
2		512	4	答案正确	2 / 2
3		4364	257	答案正确	5 / 5

提交代码

复制内容

```
#define MaxNum 1001
int Nv, Ne; /* Nv is the number of vertices, and Ne is the number of the edges */
int M[MaxNum][MaxNum] = {0}; /* the adjacent matrix of the graph */
int Dijkstra();
int main(void)
```

Chapter 4: Analysis and Comments

1 Time Complexity

Initializing the adjacency matrix (M) takes $O(Nv^2)$ time. Reading the number of vertices (Nv) and edges (Ne), and constructing the adjacency matrix takes O(Ne) time. And in Dijkstra Function, the main loop takes O(Nv) time, the nested loop takes O(Ne - i) time. So the total time complexity is $\mathbf{O(Nv^2)}$.

2 Space Complexity

Adjacent matrix M takes $O(Nv^2)$, Sel,Seq and Dist all takes O(Nv). So the total space complexity is $\mathbf{O(Nv^2)}$.

3 Comments

When the reading graph is large and there are many vertices, the construction of the graph using adjacent matrix will take up a lot of space and spend a lot of time, so it can be considered to use adjacency lists (linked lists).
Besides, ‘typedef’ for Data Types also makes code more readable, such as:

```
typedef int Vertex;
```

Chapter 5: Source Code (in C)

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

#define MaxNum 1001
int Nv, Ne; /* Nv is the number of vertices, and Ne is the number of the edges */
int M[MaxNum][MaxNum] = {0}; /* the adjacent matrix of the graph */
int Dijkstra();
int main(void)
{
    int k;  /* there are totally k cases */
    int i, opr1, opr2, weight;

    scanf("%d %d",&Nv, &Ne);
    for(i=0; i<Ne; i++){
        scanf("%d %d %d",&opr1, &opr2, &weight);
        M[opr1][opr2] = M[opr2][opr1] = weight; /* create the adjacent matrix of the graph */
    }
    scanf("%d",&k);
    for(i=0; i<k; i++){
        if(Dijkstra()) printf("Yes\n");
        else printf("No\n");
    }
    return 0;
}
int Dijkstra()
{
    int i, p, Seq[MaxNum]; /* the array for sequence */
    int Sel[MaxNum] = {0}, Dist[MaxNum] = {0}; /* the array for selected vertices, and the array for their distances */

    for(i=0; i<Nv; i++) scanf("%d", &Seq[i]); /* load the sequence */
    Sel[0] = Seq[0];
    for(i=1; i<Nv; i++) /* initialize the distance array */
        if(M[Seq[0]][Seq[i]]) Dist[Seq[i]] = M[Seq[0]][Seq[i]];
    for(i=1; i<Nv; i++){ /* whether vertex_i is to be selected next */
        for(p=i+1; p<Nv; p++)
            if(Dist[Seq[p]])
                if(Dist[Seq[p]] < Dist[Seq[i]]) return 0;   /* if vertex_i is not the smallest node, it's not Dijkstra sequence */
        Sel[i] = Seq[i]; /* else vertex_i is selected */
        for(p=i+1; p<Nv; p++)
            if(M[Seq[i]][Seq[p]]) /* if vertex_p isn't adjacent to vertex_i, the distance is no need to update */
                if(!Dist[Seq[p]] || Dist[Seq[p]] > Dist[Seq[i]] + M[Seq[i]][Seq[p]]) /* remember to change distance 'zero' */
                    Dist[Seq[p]] = Dist[Seq[i]] + M[Seq[i]][Seq[p]]; /* update the distance array */
    }
    return 1;
}
```

Declaration

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