Dijkstra Sequence

汪珉凯

Date: 2023-12-03

Chapter 1: Introduction

Problem description:

A Dijkstra sequence is an ordered sequence of vertices generated by Dijkstra's algorithm during its execution on a specific graph. But in a given graph , there could be more than one Dijkstra sequence. The problem is to judge whether the given sequence is a Dijkstra sequence in the certain given graph.

Background of the algorithms:

1. Single Source Shortest Path Problem:

The goal of Dijkstra's algorithm is to find the shortest path from a specified source vertex to all other vertices in the graph.

2.Greedy Approach:

Dijkstra's algorithm follows a greedy approach. At each step, it selects the vertex with the minimum distance from the source that is not yet included in the shortest path tree.

3.Maintaining Shortest Path Tree:

The algorithm maintains a set of vertices included in the shortest path tree.

Chapter 2: Algorithm Specification

Description of all the algorithms:

1.Dijkstra: At each step, the algorithm selects the vertex with the minimum distance from the source that is not yet included in the shortest path tree. This vertex is then added to the set of vertices included in the shortest path tree and the distance of the vertexes adjacent to it will be updated. Loop this this process until all nodes are added to the minimum distance set.

pseudo-code

```
function dijkstra(int v):
    for i from 0 to MAX:
        dis[i] = INT_MAX
        visit[i] = 0  // Clear arrays 'visit' and 'dis' every time Dijkstra's is applied
    end for

    dis[v] = 0  // The distance to itself is 0

    for i from 0 to V:  // Dijkstra's algorithm loops V times
        min = INT_MAX
        tempmin = -1  // 'min' stores the minimal distance, 'tempmin' stores the index of the vertex to be added

        for j from 1 to V:
            if visit[j] is 0 and dis[j] < min:
                tempmin = j
                min = dis[j]
            end if
        end for

        if dis[tempmin] is not equal to dis[temp[i]]:
            return 0  // Not a valid Dijkstra sequence

        visit[tempmin] = 1  // Mark this vertex as visited

        for j from 1 to V:
            if visit[j] is 0 and graph[tempmin][j] is not equal to INT_MAX and dis[tempmin] + graph[tempmin][j] < dis[j]:
                dis[j] = dis[tempmin] + graph[tempmin][j]
            end if
        end for
    end for

    return 1  // Valid Dijkstra sequence
end function
```

2.Creat_graph : The function initializes a 2D array graph with dimensions MAX * MAX and sets the initial distances between any two vertices as infinity. It then enters a loop to read input for each edge. For each edge, it reads three values: v1 and v2 (vertices), and weight (weight of the edge) and updates the matrix respectively.

pseudo-code

```
function CreateGraph(edge):
    for i from 0 to MAX-1:
        for j from 0 to MAX-1:
            graph[i][j] = INT_MAX
    // Initialize the matrix, setting initial distance between any vertices to infinity

    for i from 0 to edge-1:
        read v1, v2, weight from input
        graph[v1][v2] = weight
        graph[v2][v1] = weight
    // Read in the weight of each edge and update the corresponding entries in the matrix
end function
```

Main data structures:

Graph: represented by a matrix.

Chapter 3: Testing Results

The current status :PASS.

Table of test cases:

Graph	Input sequence	Expected output	Actual result
Graph1: 5 7 1 2 2 1 5 1 2 3 1 2 4 1	5 1 3 4 2	Yes	Yes
	5 3 1 2 4	Yes	Yes
	2 3 4 5 1	Yes	Yes

2 5 2 3 5 1 3 4 1	3 2 1 5 4	No	No
Graph2: 5 6 1 2 6 5 1 1 2 3 4 3 4 3 4 1 7 2 4 2	5 1 2 4 3	Yes	Yes
	5 1 4 2 3	Yes	Yes
	5 2 3 4 1	No	No
	2 4 3 1 5	Yes	Yes
Graph3: 4 6 1 2 1 1 3 2 1 4 1 2 3 2 2 4 3 3 4 4	3 2 1 4	Yes	Yes
	3 1 2 4	Yes	Yes
	1 4 3 2	No	No
	1 2 4 3	Yes	Yes

Chapter 4: Analysis and Comments

Time complexities: O(|V|² +|E|).

V is the number of vertex and E is the number of edges.

Reason: The algorithm loops for V times ; for each time the algorithm traverses all the V vertex to find the desired node , and update the distance of vertex adjacent to it which takes K time units. So the time complexity of Dijkstra is O(V*(V+K))=O(V² +V*K), while V*K=E. S o the time complexity of Dijkstra is O(|V|² +|E|).

Space complexity: O(|V|²). V is the number of vertex.

Reason: The algorithm uses a matrix of V*V to represent the graph, so it will take O(V²) space .Other arrays are all of 1 dimension. So the space

complexity of Dijkstra is $O(|V|^2)$.

Further possible improvements:

If the graph is dense , then the implementation of Dijkstra above is pretty good. But if the graph is sparse, then we can improve the implementation above by keeping distances in a priority queue. That way the time complexity of Dijkstra will be decreased to $O(|E|*\log|V|)$.

Appendix: Source Code (in C)

```c
#include <stdio.h>
#include <limits.h>//to use infinity

#define MAX 1005

int graph[MAX][MAX];  //use a matrix to represent the graph
                      //graph[i][j] is the weight between vertice i and vertex j
int dis[MAX];         //use an array to record the distance from a vertex to others
int visit[MAX];       //use an array to record if every node is ever visited;
                      //This array should be initializea again everytime we apply a new dijkastra
int temp[MAX];        //The temporary sequence to be checked.

int V,E;              //the number of vertexs and edges

void Creat_graph(int edge);
int dijkastra(int v);

int main(){
    scanf("%d %d",&V,&E);
    Creat_graph(E);//creat the graph

    int K;//the number of queries
    scanf("%d",&K);
    for(int i=0;i<K;i++){
        for(int j=0;j<V;j++){
            scanf("%d",temp+j);
        }//read in the sequence

        int flag=1;
        flag=dijkastra(temp[0]);    //calculate record[temp[0]]
        if(flag)printf("Yes\n");
        else printf("No\n");
    }

}
```

```
void Creat_graph(int edge){//This is the function to creat the graph.
    for(int i=0;i<MAX;i++){
        for(int j=0;j<MAX;j++)
            graph[i][j]=INT_MAX;
    }//initialize the matrix
     //the initial distance between any vertices is infinity.

    for(int i=0;i<edge;i++){
        int v1,v2,weight;
        scanf("%d %d %d",&v1,&v2,&weight);//Read in the weight of each edge.
        graph[v1][v2]=weight;
        graph[v2][v1]=weight;
    }
}

int dijkastra(int v){

    for(int i=0;i<MAX;i++){
        dis[i]=INT_MAX;
        visit[i]=0;//clear array 'visit' and 'dis' everytime dijkastra is applied
    }//initialization
    dis[v]=0;//the distance to itself is 0.


    for(int i=0;i<V;i++){//dijkastra is a algorithm which loops for V times
        int min=INT_MAX, tempmin=-1;//min is the minimal distnce;
                                    //tempmin is the index of the vertex to be added.
        for(int j=1;j<=V;j++){
            if(visit[j]==0 && dis[j]<min){
                tempmin=j;
                min=dis[j];
            }
        }

        if(dis[tempmin]!=dis[temp[i]])return 0;

        visit[tempmin]=1;//this vertex has been visited;
        for(int j=1;j<=V;j++){
            if(visit[j]==0 && graph[tempmin][j]!=INT_MAX &&
            dis[tempmin]+graph[tempmin][j]<dis[j])
                dis[j]=dis[tempmin]+graph[tempmin][j];
        }
    }
    return 1;
}
```

Declaration

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