Dijkstra Sequence

author: An ordinary student

date: 2020-11

1. Introduction

Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. It was conceived by computer scientist Edsger W. Dijkstra in 1956 and published three years later. See http://en.wikipedia.org/wiki/Dijkstra's_algorithm

In this algorithm, 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, let's call it **Dijkstra sequence**, is generated by Dijkstra's algorithm. There could be more than one Dijkstra sequence.

Now, given the total number of vertices $N_v(\leq 10^3)$,edges $N_e(\leq 10^5)$ and N_e lines describing an edge by giving the indices of the vertices at the two ends, followed by a positive integer weight (≤100) of the edge, the goal is to design an algorithm to tell if k given sequences are Dijkstra sequences.

2. Algorithm Specification

- **Main data structure 1**: Adjacency matrix

Definition: The adjacency matrix used a two-dimentional array to represent a labelled graph, with the weight of edge(Vi,Vj) in the position of (Vi , Vj) . The adjacency matrix for an undirected graph is symmetric.

Main Idea: The adjacency matrix used a two-dimensional array to represent a connected graph. The value of each element in the array represents the weight of each line.

- **Main data structure 2** : A linear list of vertices.

Definition: the array-based list implementation is defined to store list elements in contiguous cells of the array. Each element is defined as

```
struct vertex{
    int known;
    int dist;
};
```

Main Idea: The array-based list implementation is used to store the labels of each vertex, including whether or not they have been included in the path(`known`), and the shortest distance from the source vertex to it (`dist`). These labels will be updated in the Dijkstra's algorithm.

- **Algorithm** : Jugding a given sequence through Dijkstra's algorithm

Input: a sequence of n integers S[n]

Output: True (integer 1) or False (integer 0)

Main Idea: The algorithm judges whether a given sequence is a Dijkstra sequence through Dijkstra algorithm.

Pseudo Code:

```
Procedure IsDseq(S[n]:the given sequence){
    for all the vertices v in G
    set v.known=0
    set v.dist=infinity
    end
    G[s[0]].dist=0
    for each element i in S[n]:
        G[i] is known;
        for all the vertices in G:
        Find Vertices that is adjacent to G[i] and compute the shortest path
(contains only known vertices) from the source vertex to them
        end

        for all the vertices in G:
        Find the minimum length of path min
        end

        next:=the next element of i
        if(G[next].dist!=min) return false
    end
    return true
}
```

3. Testing Results

- Case 1

Input	Output
1 0 1 1	Yes

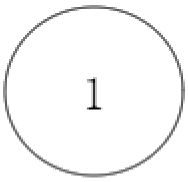


Fig 1. Case 1

- Case 2

Input	Output
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

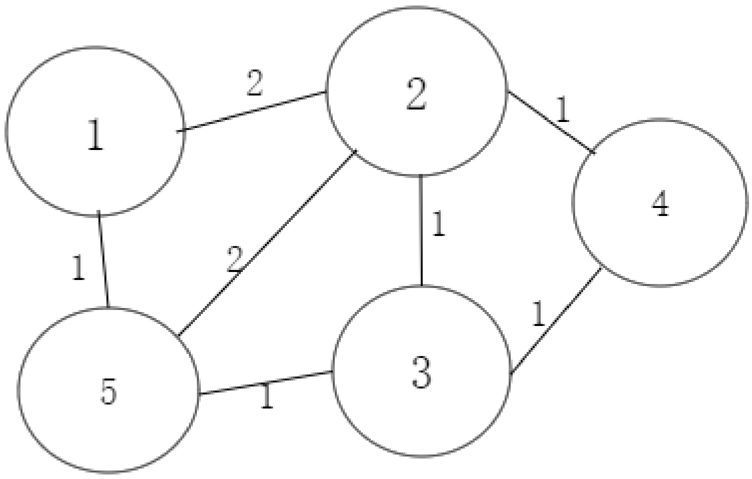


Fig 2. Case 2

- Case 3

Input	Output
5 6 1 2 88 2 3 4 2 4 1 2 5 99 3 5 1 3 4 1 3 2 4 1 3 5 2 4 3 1 5 2 4 3 5 1	No No Yes

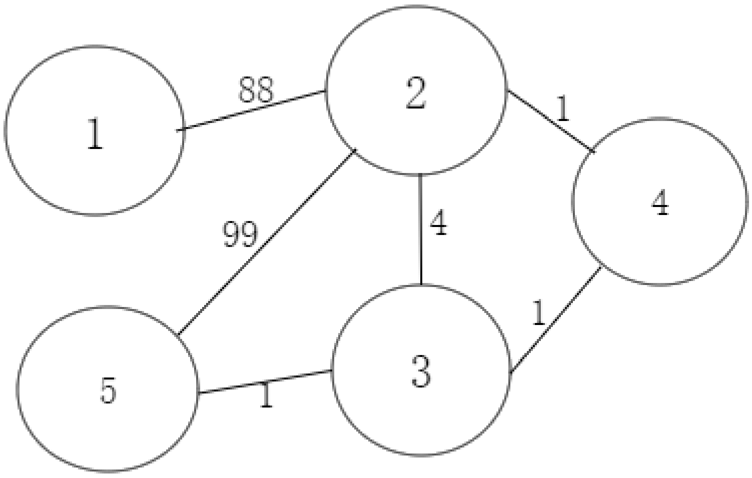
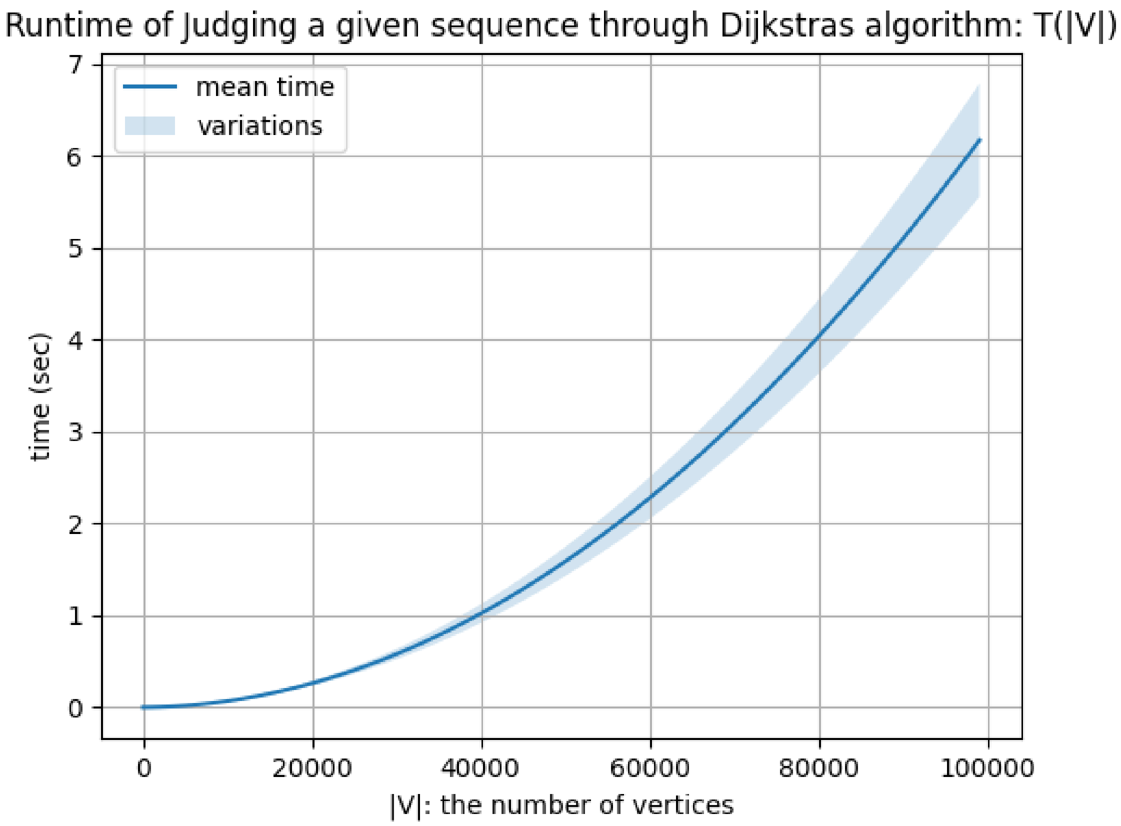


Fig 3. Case 3

Test cases	Design Purpose	expected result	actual behavior	possible cause for a bug	status
Case 1	A graph with one vertex	[Yes]	[Yes]	/	*pass*
Case 2	A weighted graph with test cases starting from different source vertices; with cases changing the order of vertices with same distance	[Yes Yes Yes No]	[Yes Yes Yes No]	/	*pass*
Case 3	A weighted graph with some vertices whose label of distance will change between each steps	[No No Yes]	[No No Yes]	/	pass

Table: Test cases for the algorithms implementation.



Figures 4 shows the running time of Jugding a given sequence through Dijkstra's algorithm. We observe a curve similar with $y = x^2$ which implies an algorithm with time complexity $O(|V|^2)$

4. Analysis and Comments

Jugding a given sequence through Dijkstra's algorithm

For the runtime of Jugding a given sequence through Dijkstra's algorithm, there are two *for* loop in side a *for* loop.

The outside *for* loop go through each vertice v:

inside:

- The first *for* loop find all the vertices that's adjacent to v, and update the label for distance, which takes $O(|V|)$ in the worst case.
- The second *for* loop go through every vertices to find the minium length of path present, which takes $O(|V|)$ in the worst case.

Therefore, The time complexity of Jugding a given sequence through Dijkstra's algorithm with |V| vertices is :

$$O(|V|^2)$$

For the space requirement, since we need $|V|$ *struct vertex* to store the $|V|$vertices and a two-dimentional array to store the lines,which is $\Theta(|V|^2)$, the total space complexity should be $\Theta(|V|^2)$.

A better method is to use a min-heap to organize the data and treats the update as a *decrease_key* operation. Thus, the time to find the minimum and *delete_min* is O(log|V|),the time for an update is O(log|V|), too . Besides, a queue can be used to store all the adjacent vertices to v so that there's at most one update for per edge for a total of O(|E|).
Therefore, the time complexity in this algorithm is

$$T(|V|) = O(|E|\log|V| + |V|\log|V|) = O(|E|\log|V|)$$

Appendix: Source Code (in C)

```
#include <stdio.h>
#include <stdlib.h>
#define max 1005
#define infty 200000
#define false 0
#define true 1
//use struct vertex to store each vertex,with label "known" and "dist".
typedef struct vertex *PtrToV;
struct vertex{
    int known; //the status of a vertex, indicating whether or not it is visited
    int dist;  // the value of dist shows the length of path from the source
vertex to it
};
int admatrix[max][max];
void readgraph(int a[][max],int ne,int nv);
void initial(int nv,PtrToV G);
int IsDseq(int s[],int nv,PtrToV G);

main(){
    int nv,ne,k;
    scanf("%d %d",&nv,&ne);//get the number of vertices and edges
    PtrToV G=(PtrToV)malloc((nv+1)*sizeof(struct vertex));  //apply for enough
space to store vertices and their labels
    readgraph(admatrix,ne,nv);//construct a two-dimensional array to represent a
graph
    // get k sequences and judge if they are Dijkstra sequences
    scanf("%d",&k);
    int seq[nv];
    for(int i=0;i<k;i++){
        for(int j=0;j<nv;j++){
            scanf("%d",&seq[j]);
        }
        if(IsDseq(seq,nv,G))printf("Yes\n");
        else printf("No\n");
    }
    return 0;
}


void readgraph(int a[][max],int ne,int nv){
    int i,j;
    int v1,v2,w;
```

```
    // initialize the adjacency matrix, using the value infty to represent that
two vertices are not connected
    for(i=1;i<=nv;i++){
        for(j=1;j<=nv;j++){
            a[i][j]=infty;
        }
    }
    // get information for each edge and store their weight.The adjacency matrix
for a undirected graph is symmetrical
    for(i=0;i<ne;i++){
        scanf("%d %d %d",&v1,&v2,&w);
        a[v1][v2]=w;
        a[v2][v1]=w;
    }
}

//Since each given sequence has different source vertices, the labels of each
vertex will be different.
//the function initial is used to clear labels of all vertices before examing
each sequence.
void initial(int nv,PtrToV G){
    for(int i=1;i<=nv;i++){
        G[i].known=0;
        G[i].dist=infty;
    }
}
//the function IsDseq judges each given sequence through Dijkstra's algorithm
int IsDseq(int s[],int nv,PtrToV G){
    initial(nv,G);
    int i,tem,j,next,min;
    G[s[0]].dist=0; //set the label of dist to be 0 for the source vertex
    //for each vertex in the given sequence,judges if its following vertex is
legal
    for(i=0;i<nv-1;i++){
        tem=s[i];
        G[tem].known=1; //visit one vertex in each turn
        for(j=1;j<=nv;j++){
            //find all the vertices w that has not been visited and has a
shorter length of path through the present vertex s[i]
            if(G[j].known==0&&(G[tem].dist+admatrix[tem][j]<G[j].dist))
            G[j].dist=G[tem].dist+admatrix[tem][j];//update the label of
distance for w
        }
        //in all the vertices whose label of dist has been updated, find the
minimum
        min=infty;next=i+1;
        for(j=1;j<=nv;j++){
            if(G[j].known==0&&G[j].dist<min)
                min=G[j].dist;
        }
        //for the next vertex a to be legal, a.dist should be minimum among
vertices that has been labeled
        if(G[s[next]].dist!=min) return false;
    }
    //if all the vertex in the given sequence is legal , return true
    return true;
}
```

Declaration

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