Dijkstra Sequence Problem

Author:陈硕
Number:3220106406
Date:2023-12-4

I.Introduction

Dijkstra's algorithm is one of the most famous greedy algorithms t solve the **single source shortest path problems**.

In this algorithm, a set contains vertices included in shortest pat tree is maintained. During each step, we find one vertex which is no yet included and has a minimum distance from thesource, and collect it into the set. Hence step by step an ordered sequence of vertices, and let's call it **Dijkstra sequence.**

On the other hand, for a given graph, there could be more than one Dijkstra sequence.

This project aims to check whether a given sequence is Dijkstra sequence or not.

II.Algorithm Specification

Firstly,a two-dimension symmetric array **g[][]** is constructed, used to **store the whole graph**.In this array,**g[a][b]=m(m!=0) means there is an edge between vertice a and vertice b with the value of the edge being m**; g[a][b]=0 means that there is no edge betweenvertice a and vertice

```c
#include<stdio.h>
#include<windows.h>
#define MaxVertexNum 1001
int g[MaxVertexNum][MaxVertexNum];  //global variable,storing the graph

int dijkstra(int vt[],int v);
int MinFromSettoVertex(int Set[],int num,int m);
void Buildset(int Remainset[],int Set[],int num,int v);
int MinFromSettoRemainset(int Set[],int Remainset[],int num,int v);  //four functions

int main(){
    int v,e;
    scanf("%d%d",&v,&e);        //v represents the number of vertices
    int i,k;                    //e represents the number of edges
    for(i=0;i<v;i++){
        for(k=0;k<v;k++){
            g[i][k]=0;         //initialize the array,while g[i][k]=0 means there is no edge from vertice i to vertice k
        }
    }
    int vertex1,vertex2,value;
    for(i=0;i<e;i++){
        scanf("%d%d%d",&vertex1,&vertex2,&value);
        g[vertex1][vertex2]=value;              //read in the graph
        g[vertex2][vertex1]=value;
    }
    scanf("%d",&k);
    int vt[MaxVertexNum];
    for(i=0;i<k;i++){                          //read in the sequences remaining to be checked
        for(int j=0;j<v;j++){
            scanf("%d",&vt[j]);
        }
        if(dijkstra(vt,v))printf("Yes\n");      //judge if the present sequence is a Dijkstra sequence
        else printf("No\n");

    }
}
```

Then,I design some functions to fulfill my object.

Function 1: dijkstra

This function is the core judge function,if its return value is 1,the sequence is a Dijkstra sequence, otherwise not.In this function,I build an array called **Set,storing the vertices we have included in** and another array called **Remainset**(built by Function 3),s**toring the vertices outside Set**. Everytime we meet a vertice in the given sequence,we first compute **min1**(computed by Function 2),**the minimum distance From Set to the vertice**.Then we compute **min2**(computed by Function 4),**the minimum distance from Set to**

Remainset.If min1!=min2,it means the vertice don't satisfy Dijkstra algorithm,then the sequence isn't a Dijkstra sequence.If all vertices satisfy the requirements,the whole sequence is a Dijkstra sequence.

```
//This function is the judge function,which involves the core algorithm
int dijkstra(int vt[],int v){
    int Set[MaxVertexNum];           //We build an array called Set,storing the vertices we have included in
    int Remainset[MaxVertexNum];     //We build another array called Remainset,storing the vertices outside Set
    int min1,min2;                   //min1 represents the minimum distance from Set to the next vertive in given sequence
    int num=0;                       //min2 represents the minimum distance from Set to Remainset
    Set[0]=vt[0];
    num++;
    int i,j;
    int flag=1;                   //Every time we meet a vertice in the given sequence,we check if min1==min2
    for(i=1;i<v;i++){
          min1= MinFromSettoVertex(Set,num,vt[i]);
          Buildset(Remainset,Set,num,v);
          min2= MinFromSettoRemainset(Set,Remainset,num,v);
          if(min1!=min2){
              flag=0;
              break;
          }
          Set[num]=vt[i];
          num++;

    }
    return flag;                  //if min1!=min2 for any vertice,the sequence isn't a Dijkstra sequence,and we break with flag=0;
}                                 //if min1==min2 for every vertice,it is a Dijkstra sequence,and we return flag=1;
```

Function 2: MinFromSettoVertex

This function aims to compute min1.**We can simply ergodic all the elements in Set to find the minimum distance**.Note that 0 can be considered as the minimum because 0 means that there is no edge between the two vertices!

Function 3: Buildset

```
//This function computes min1
int MinFromSettoVertex(int Set[],int num,int m){
    int i;
    int mindis;
    for(i=0;i<num;i++){                //we first initialize the minimum distance,noting that the distance!=0
        if(g[Set[i]][m]!=0){           //because 0 means there's no edge between 2 vertices
            mindis=g[Set[i]][m];
            break;
        }
    }
    for(i=0;i<num;i++){                          //then we ergodic all elements in Set,comparing the distance with the minimum
        if(g[Set[i]][m]<mindis&&g[Set[i]][m]!=0){
            mindis=g[Set[i]][m];
        }
    }
    return mindis;                    //finally we return the minimum distance
}
```

This function aims to build Remainset. I set a mark"flag".Then I ergodic Set and **mark all the elements that have appeared in Set with flag=0,meaning that they can't appear in Remainset**.Hence I get the final Remainset.

```
//This function builds the array "Remainset"
void Buildset(int Remainset[],int Set[],int num,int v){
    int i=0;
    int j,k;
    int flag;
    for(j=1;j<=v;j++){              //the origin vertices are 1,2,3,……, v, while v means the number of vertices
        flag=1;                     //however,if the vertices have appeared in Set,then it can't be in Remainset,so we set flag to 0 and break
        for(k=0;k<num;k++){
            if(Set[k]==j){
                flag=0;
                break;
            }
        }
        if(flag==1){              //if flag=0,it means this vertice is outside Set,so we 'enroll' it into Remainset
            Remainset[i]=j;
            i++;
        }
    }
}
```

Function 4: MinFromSettoRemainset

This function aims to compute min2.With Set and Remainset already built, **we can just simply ergodic all the vertices in Set and Remainset to find the smallest edge to connect the two set**. Also note that 0 can be considered as the minimum because 0 means that there is no edge between the two vertices!

```
//This function computes min2
int MinFromSettoRemainset(int Set[],int Remainset[],int num,int v){
   int i,j;
   int mindis;
    for(i=0;i<num;i++){                     //we also first initialize the minimum distance,noting that the distance!=0
       for(j=0;j<(v-num);j++){              //because 0 means there's no edge between 2 vertices
        if(g[Set[i]][Remainset[j]]!=0){
            mindis=g[Set[i]][Remainset[j]];
            break;
        }
        }
    }
   for(i=0;i<num;i++){                        //then we ergodic all elements in Set and Remainset,comparing the distance with the minimum
        for(j=0;j<(v-num);j++){
            if(g[Set[i]][Remainset[j]]<mindis&&g[Set[i]][Remainset[j]]!=0){
                mindis=g[Set[i]][Remainset[j]];
            }
        }
   }
   return mindis;                             //finally we return the minimum distance
}
```

III.Testing Results

Testing data and results	Test for what
1 0 1 1 Yes	A graph with only a vertice
5 7 1 2 2 1 5 1 2 3 1 2 4 1 2 5 2 3 5 1 3 4 1 1 3 2 1 5 4 No	not be Dijkstra sequence
5 7 1 2 2 1 5 1 2 3 1 2 4 1 2 5 2 3 5 1 3 4 1 1 5 1 3 4 2 Yes	Be Dijkstra sequence.

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 Yes 5 3 1 2 4 Yes 2 3 4 5 1 Yes 3 2 1 5 4 No		Several sequences input at the same time

IV.Analysis and Comments

Let v be the the number of vertices in the graph.

function	Time complexity	Space complexity
MinFromSettoVertice	O(v) The number of elements in set varies from 1 to v,**hence it's O(v)**	We build a two-dimension array g[][],storing the whole graph.**It takes v^2 space.** We build two one-dimension arrays Set and Remainset,**both taking v spaces** During the process of computing minimum distance,we first set a minimum,and compare the values computed a later with minimum,so **it takes O(1) space.** Therefore,**the space complexity of the whole program is O(v^2).**
Buildset	O(v) We actually ergodic all elements in Set to do the mark.**So it's O(v)**	
MinFromSettpRemainset	O(v^2) When elements in Set and Remainset are both v/2,we do v^2/4 computations,which is the most.**Soit'sO(v^2).**	
dijkstra (also the time complexity of the whole program)	O(v^3) We ergodic all vertices in the sequence.In the loop,we call the three functions above,**so the whole time complexity isO(v)*O(v^2)=O(v^3)**	

In a nutshell,the time and space complexity of the whole progran is respectively O(V^3) and O(v^2).

From the analysis of time and space complexity, I can feel that there's still much space for my project and algorithm toimprove (especially the time complexity is too large.And I may use more efficient algorithms. Overall, I successfully finished my project,despite many difficulties.Hope that I can do better next time!

V.Appendix

```c
#include<stdio.h>
#include<windows.h>
#define MaxVertexNum 1001
int g[MaxVertexNum][MaxVertexNum];  //global variable,storing the graph

int dijkstra(int vt[],int v);
int MinFromSettoVertex(int Set[],int num,int m);
void Buildset(int Remainset[],int Set[],int num,int v);
int MinFromSettoRemainset(int Set[],int Remainset[],int num,int v);  //four functions

int main(){
    int v,e;
    scanf("%d%d",&v,&e);        //v represents the number of vertices
    int i,k;                    //e represents the number of edges
    for(i=0;i<v;i++){
        for(k=0;k<v;k++){
            g[i][k]=0;         //initialize the array,while g[i][k]=0 means there is no edge from vertice i to vertice k
        }
    }
    int vertex1,vertex2,value;
    for(i=0;i<e;i++){
        scanf("%d%d%d",&vertex1,&vertex2,&value);
        g[vertex1][vertex2]=value;              //read in the graph
        g[vertex2][vertex1]=value;
    }
    scanf("%d",&k);
    int vt[MaxVertexNum];
    for(i=0;i<k;i++){                          //read in the sequences remaining to be checked
        for(int j=0;j<v;j++){
            scanf("%d",&vt[j]);
        }
        if(dijkstra(vt,v))printf("Yes\n");      //judge if the present sequence is a Dijkstra sequence
        else printf("No\n");

    }
}

//This function is the judge function,which involves the core algorithm
int dijkstra(int vt[],int v){
    int Set[MaxVertexNum];           //We build an array called Set,storing the vertices we have included in
    int Remainset[MaxVertexNum];     //We build another array called Remainset,storing the vertices outside Set
    int min1,min2;                  //min1 represents the minimum distance from Set to the next vertive in given sequence
    int num=0;                      //min2 represents the minimum distance from Set to Remainset
    Set[0]=vt[0];
    num++;
    int i,j;
    int flag=1;                    //Every time we meet a vertice in the given sequence,we check if min1==min2
    for(i=1;i<v;i++){
          min1= MinFromSettoVertex(Set,num,vt[i]);
          Buildset(Remainset,Set,num,v);
          min2= MinFromSettoRemainset(Set,Remainset,num,v);
          if(min1!=min2){
              flag=0;
              break;
          }
          Set[num]=vt[i];
          num++;

    }
    return flag;                  //if min1!=min2 for any vertice,the sequence isn't a Dijkstra sequence,and we break with flag=0;
}                                //if min1==min2 for every vertice,it is a Dijkstra sequence,and we return flag=1;
```

```c
//This function computes min1
int MinFromSettoVertex(int Set[],int num,int m){
    int i;
    int mindis;
    for(i=0;i<num;i++){                 //we first initialize the minimum distance,noting that the distance!=0
        if(g[Set[i]][m]!=0){            //because 0 means there's no edge between 2 vertices
            mindis=g[Set[i]][m];
            break;
        }
    }
    for(i=0;i<num;i++){                             //then we ergodic all elements in Set,comparing the distance with the minimum
        if(g[Set[i]][m]<mindis&&g[Set[i]][m]!=0){
            mindis=g[Set[i]][m];
        }
    }
    return mindis;                      //finally we return the minimum distance
}
//This function builds the array "Remainset"
void Buildset(int Remainset[],int Set[],int num,int v){
    int i=0;
    int j,k;
    int flag;
    for(j=1;j<=v;j++){              //the origin vertices are 1,2,3,……, v, while v means the number of vertices
        flag=1;                     //however,if the vertices have appeared in Set,then it can't be in Remainset,so we set flag to 0 and break
        for(k=0;k<num;k++){
            if(Set[k]==j){
                flag=0;
                break;
            }
        }
        if(flag==1){                //if flag=0,it means this vertice is outside Set,so we 'enroll' it into Remainset
            Remainset[i]=j;
            i++;
        }
    }
}

//This function computes min2
int MinFromSettoRemainset(int Set[],int Remainset[],int num,int v){
    int i,j;
    int mindis;
    for(i=0;i<num;i++){                     //we also first initialize the minimum distance,noting that the distance!=0
        for(j=0;j<(v-num);j++){             //because 0 means there's no edge between 2 vertices
            if(g[Set[i]][Remainset[j]]!=0){
                mindis=g[Set[i]][Remainset[j]];
                break;
            }
        }
    }
    for(i=0;i<num;i++){                         //then we ergodic all elements in Set and Remainset,comparing the distance with the minimum
        for(j=0;j<(v-num);j++){
            if(g[Set[i]][Remainset[j]]<mindis&&g[Set[i]][Remainset[j]]!=0){
                mindis=g[Set[i]][Remainset[j]];
            }
        }
    }
    return mindis;                              //finally we return the minimum distance
}
```

VI. Declaration

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