Dijkstra(算法模板)

#include<stdio.h>

#include<stdlib.h>

#include<limits.h>

#define V 5

#define INF INT\_MAX

Typedef struct {

Int graph[v][v];//邻接矩阵表示图

Int dist[v];//存储到各个顶点的最短距离

Int visited[v];//标记每个顶点是否被访问过

}Dijkstra；

Void initDijkstra(Dijkstra \*a){//初始化

Int i,j;

For(i=0;i<v;I++){

For(j=0;j<v;j++){

A-> Graph[v][v]=INF;

}

A->Dist[v]=INF;

A->Visited[v]=0;

}

}

Void printfsolution(Dijksta \*a){

int i=0;

Printf(“到各个顶点的最短距离为：\n”)；

For（i=0;i<v;i++）{

Printf（“顶点%d：/t%d\n”,i,d->dist[i]);

}

}

Void dijkdstra(Dijksta \*d ,int start){

Int minDist,minindex;

Int i,j,u;

initDijkstra(d);

1. >dist[start]=0;

For(i=0;i<v-1;i++){

minDist = INF ;//存储当前最短特殊路径长度

MinIdex=-1；//存储当前最短特殊路径所连接的顶点编号

For(j=0;j<v;j++){

If(!d->visited[j]&&d->dist[j]<mindist){

Mindist= d->dist[j];

minIndex=j;

}

}

If(minindex==-1){//如果所有顶点都被访问过了，则结束循环

Break;

}

U=minindex；

d->visited[u]=1;//选出了节点

For(j=0;j<V;j++){//更新最短路径存储的边长

If(d->visited[j]==0&&d->dist[u]+d->graph[u][j]<d->dist[j]&&graph[u][j]!=INF){

D->dist[v]=d->dist[u]+d->graph[u][v];//更新路径

}

}

}

总结：1.需要用到邻接矩阵这种数据结构

1. 需要用到至少两个数组 visited[v]数组用来放已经遍历节点,抽象为集合，数组dist[v]用来存储源点到其他节点的最短路径
2. 在代码部分将代码分为两个操作 在放节点的循环中 先通过dist数组进行找要放到visited集合的节点，再进行记忆的最短路径的更新

#include<iostream>

#include<string>

#include<cmath>

#include<stdlib.h>

#include <vector>

#include<algorithm>

using namespace std;

class Dijkstra\_os{

public:

  vector<bool> visited;

  vector<int> dist;

  int v;

  int start;

  Dijkstra\_os(int num){

    start=0;

     visited.resize(num,false);

     v=num;

     dist.resize(num,INT\_MAX);

  }

  void print\_road(){

    for(int i=0;i<v;i++){

        if(i!=start){

            cout << "start\_node  end\_node  wight"<<endl;

            cout << start<<"              "<<i<<"          "<<dist[i]<<endl;

        }

    }

  }

  int get\_minroad(){

    int minidx=-1;

    int min\_road=INT\_MAX;

    for(int i=0;i<v;i++){

        if(visited[i]==false&&dist[i]<min\_road){//找到下一步要访问的节点

              minidx=i;

              min\_road=dist[i];

        }

    }

    return minidx;

  }

  void key\_DJ(vector<vector<int>>&grap,int st){

    //先将开始的顶点放到集合中,但为了逻辑的连贯性，假设有一个编号为-1的节点，从-1的节点出发再导start，再遍历其他的节点

     start =st;

     dist[start]=0;

     for(int i=0;i<v;i++){

        int mid\_u=get\_minroad();//第一部分

        if(mid\_u==-1){//表明所有节点都放到了节点中

            break;

        }

       visited[mid\_u]=true;

       for(int j=0;j<v;j++){

        if(visited[j]==false&&grap[mid\_u][j]!=0&&dist[mid\_u]+grap[mid\_u][j]<dist[j]){

            dist[j]=dist[mid\_u]+grap[mid\_u][j];

        }

       }

     }

print\_road();

  }

};

# Floyd 算法模板(一开始需要用到邻接矩阵辅助)

#include<iostream>

#include<cstring>

#include<cmath>

Using namespace std;

Class floyd\_model{

Vector<vector<int>> grap\_reach;//到达矩阵 grap\_reach[i][j] 表示节点 i 到节点 j 的距离 若为无穷大则表示两个节点不可到达

Int v;

Grap\_reach.resize(num,vector<int>(num,0));

Floyd\_model(int num ,vector<vector<int>> &grap){

v=num；

For(int i=0;i<v;i++){

For(int j =0;j<;j++){

If(grap[i][j]==0){

Grap\_reach[i][j]=INT\_MAX;

}

Else {

Grap\_reach[i][j]=grap[i][j];

}

}

}

For(int k=0;k<v;k++){

Grap\_reach[k][k]=0;

}  
}

Void floyd\_key(){

For(int i=0;i<v;i++){//一个节点一个节点的当中转站

For(int p=0;p<v;p++){

For(int d=0;p<v;p++){

If(p!=i&&d!=i&&grap\_reach[p][i]+grap\_reach[i][d]<grap\_reach[p][d]){

Grap\_reach[p][d]=grap\_reach[p][i]+grap\_reach[i][d];

}

}

}

}

}

Void print\_road(){

For(int i=0;i<v;i++){

For(int j=0;j<v;j++){

If(grap\_reach!=INT\_MAX){

Cout <<”e\tw”<<endl;

Cout <<i<<”--”<<j<<’\t’<<grap[i][j];

}

}

}

}

};

#include<iostream>

#include<string>

#include<cmath>

#include<stdlib.h>

#include <vector>

#include<algorithm>

using namespace std;

class Floy\_ed{

public:

vector<vector<int>> grap\_reach;

int v;

Floy\_ed(int num,vector<vector<int>> grap){

    v=num;

    grap\_reach.resize(num,vector<int>(num,0));

    for(int i=0;i<v;i++){

        for(int j=0;j<v;j++){

            if(grap[i][j]==0){

                grap\_reach[i][j]=65535;

            }

            else{

                grap\_reach[i][j]=grap[i][j];

            }

        }

    }

    for(int k=0;k<v;k++){

        grap\_reach[k][k]=0;

    }

}

void floyd\_key(){

    for(int k=0;k<v;k++){

        for(int i=0;i<v;i++){

            for(int j=0;j<v;j++){

                if(i!=k&&j!=k&&grap\_reach[i][k]+grap\_reach[k][j]<grap\_reach[i][j]){

                    grap\_reach[i][j]=grap\_reach[i][k]+grap\_reach[k][j];

                }

            }

        }

    }

}

void print\_road(int num){

    floyd\_key();

    for(int j=0;j<v;j++){

        cout <<"edges\tweigt"<<endl;

        cout <<num<<"--" <<j<<'\t'<<grap\_reach[num][j]<<endl;

    }

}

};

第一步： 构建出邻接矩阵，再根据邻接矩阵构造可达矩阵

第二步： 按照算法的思想不断更新 可达矩阵的值 ，更新过程中，要切记 中转节点的 同一行和一列上的点不跟新 （优化计算）

在写模板的时候遇到的问题，INT\_MAX的使用文件没有导入.

