算法设计与分析实验报告

倪玮昊

2020211346

一、 实验题目

2.1 题目描述

求出给定顶点 1 与顶点 V 之间的最短路。

2.2 输入格式

输入文件名为 dijkstra.in。
第一行两个正整数 V、E，分别代表顶点数、边数。
接下来 E 行包含三个正整数 u、v、w，代表 u、v 之间存在一条权值为 w 的**无向边**。

2.3 输出格式

输出文件名为 dijkstra.out。输出共一行。
第一行包含一个整数为最短路上权值之和，若顶点 1 与顶点 V 不连通，输出-1。

2.4 输入输出样例

dijkstra.in	dijkstra.out
3 3 1 2 5 2 3 5 3 1 2	2

2.5 数据范围

$0 < V \leq 5000$,
$0 < E \leq 2 \times 10^5$,
$0 < u, v \leq V$,
$0 < w \leq 2 \times 10^5$。

2.6 说明/提示

保证图中没有自环。

二、实验过程

算法思路

1.指定一个节点，例如我们要计算 'A' 到其他节点的最短路径

2.引入两个集合（S、U），S集合包含已求出的最短路径的点（以及相应的最短长度），U集合包含未求出最短路径的点（以及A到该点的路径，注意 如上图所示，A->C由于没有直接相连 初始时为∞）

3.初始化两个集合，S集合初始时 只有当前要计算的节点，A->A = 0，U集合初始时为 A->B = 4, A->C = ∞, A->D = 2, A->E = ∞,

4.从U集合中找出路径最短的点，加入S集合，例如 A->D = 2

5.更新U集合路径，if (‘D 到 B,C,E 的距离’ + ‘AD 距离’ < ‘A 到 B,C,E 的距离’) 则更新U

6.循环执行 4、5 两步骤，直至遍历结束，得到A 到其他节点的最短路径

关键函数及代码段的描述

贪心算法找最优解

```
int S[n];
    int D[n];
    int P[n];
    for (int con = 0; con < n; con++) {
        P[con] = startpoint;
        S[con] = 0;
    }
    P[startpoint] = -1;
    for (int count = 0; count < n; count++) {
        D[count] = a[startpoint][count];
    }
    S[startpoint] = 1;
    for (int count = 0; count < n; count++) {
        int minnum=findmin(D,S);
        if (minnum >= 0) S[minnum] = 1;
        for (int con1 = 0; con1 < n; con1++) {
            if (S[con1] == 0 && D[con1] > D[minnum] + a[minnum][con1]) {
                D[con1] = D[minnum] + a[minnum][con1];
                P[con1] = minnum;
            }
        }
    }
```

找最短路径加入结果

```
int findmin(int D[],int S[]) {
    int min = 10000;
    int minnum = -1;
    for (int count = 0; count < n; count++) {
        if (D[count] < min && D[count]!=0 && S[count]!=1) {
            min = D[count];
            minnum = count;
        }
    }
    return minnum;
}
```

通过回溯来输出最短路径

```
int way[n];
int con2 = 0;
int pre = P[endpoint];
while (pre != -1) {
    way[con2] = pre;
    con2++;
```

```
    pre = P[pre];
}
int total=0;
int con3;
for (con3 = con2 - 2; con3 >= 0; con3--) {

    if(a[way[con3 + 1]][way[con3]]==tem) return -1;
    else total= a[way[con3 + 1]][way[con3]]+total;

}
if(a[way[con3 + 1]][endpoint]==tem) return -1;
else total= a[way[con3 + 1]][endpoint]+total;
return total;
```

算法时间及空间复杂性分析

时间复杂度是O(n2),空间复杂度为O(n),用一维数组存数据

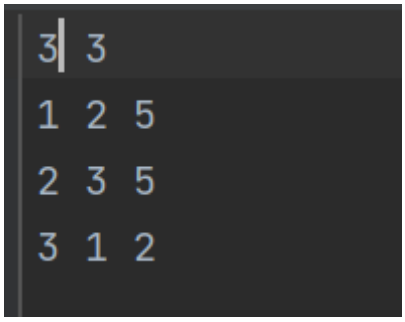
三、 实验结果

程序执行环境及运行方式

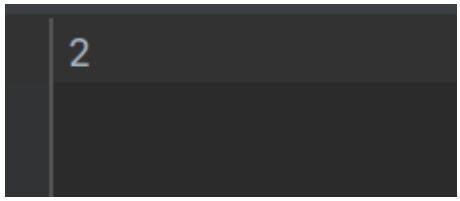
win11,CLion

程序执行示例

输入:



输出:



四、 实验总结

接触过迪杰斯特拉算法,不难,时间复杂度可以用堆栈优化,效果更好

五、 算法源代码

```
#include <deque>
#include <algorithm>
#include <iostream>
#include <fstream>
using namespace std;
```

```
int n;
int lineNum;
int a[5001][5001];
int tem;
#define rep(i, a, b) for (int i = a; i <= b; ++i)
void startA(){
    for(int i=0;i<5001;i++){
        for(int i1=0;i1<5001;i1++){
            a[i][i1]=tem;
        }
    }

}
int findmin(int D[],int S[]) {
    int min = 10000;
    int minnum = -1;
    for (int count = 0; count < n; count++) {
        if (D[count] < min && D[count]!=0 && S[count]!=1) {
            min = D[count];
            minnum = count;
        }
    }
    return minnum;
}
int Dijkstra(int startpoint, int endpoint){
    int S[n];
    int D[n];
    int P[n];
    for (int con = 0; con < n; con++) {
        P[con] = startpoint;
        S[con] = 0;
    }
    P[startpoint] = -1;
    for (int count = 0; count < n; count++) {
        D[count] = a[startpoint][count];
    }
    S[startpoint] = 1;
    for (int count = 0; count < n; count++) {
        int minnum=findmin(D,S);
        if (minnum >= 0) S[minnum] = 1;
        for (int con1 = 0; con1 < n; con1++) {
            if (S[con1] == 0 && D[con1] > D[minnum] + a[minnum][con1]) {
                D[con1] = D[minnum] + a[minnum][con1];
                P[con1] = minnum;
            }
        }
    }
    int way[n];
    int con2 = 0;
    int pre = P[endpoint];
    while (pre != -1) {
        way[con2] = pre;
        con2++;
        pre = P[pre];
    }
```

```
    int total=0;
    int con3;
    for (con3 = con2 - 2; con3 >= 0; con3--) {

        if(a[way[con3 + 1]][way[con3]]==tem) return -1;
        else total= a[way[con3 + 1]][way[con3]]+total;

    }
    if(a[way[con3 + 1]][endpoint]==tem) return -1;
    else total= a[way[con3 + 1]][endpoint]+total;
    return total;


}
int main()
{
    fstream fin("dijkstra.in", ios::in), fout("dijkstra.out", ios::out);
    fin >> n;
    tem=200000*n+1;
    fin>>lineNum;
    startA();
    int start,end;
    rep(i, 0, lineNum-1) {
        fin >> start;
        fin >> end;
        fin >> a[start-1][end-1];
        a[end-1][start-1]=a[start-1][end-1];
    }
    int distance=Dijkstra(0,n-1);
    fout<<distance;
    return 0;

}
```