算法设计与分析实验报告

倪玮昊

2020211346

一、实验内容

4.1 题目描述

使用 **Kruskal 算法**计算给定无向图上以顶点 1 为根的最小生成树边权和。

4.2 输入格式

输入文件名为 kruskal.in。
第一行包含两个正整数 V、E，分别代表顶点数与边数。
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**。

4.3 输出格式

输出文件名为 kruskal.out。输出共一行。
第一行包含一个整数代表最小生成树的权值和。

4.4 输入输出样例

kruskal.in	kruskal.out
4 5 1 2 2 1 3 2 1 4 3 2 3 4 3 4 3	7

4.5 数据范围

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

4.6 说明/提示

保证图中没有自环。

二、实验过程

算法思路

1 初始化。将所有边都按权值从小到大排序，将每个节点集合号都初始化为自身编号。

2 按排序后的顺序选择权值最小的边（u,v）。

3 如果节点 u 和 v 属于两个不同的连通分支，则将边(u,v)加入边集 TE 中，并将两个连通分支合并。

4 如果选取的边数小于 n-1,则转向步骤2，否则算法结束。

关键函数及代码段的描述

找是否存在闭环

```
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;
```

快速排序

```
  if(begin >= end)//只有一个元素,递归完成
        return;
    line sign = a[end];
    int i = begin;
    int j = end;
    while(i != j){//对右边大和左边小的值进行调换
        while(a[i].value <= sign.value && j > i)
            i++;
        while(a[j].value >= sign.value && j > i)
            j--;
        if(j > i){
            line t = a[i];
            a[i] = a[j];
            a[j] = t;
        }
    }
    a[end] = a[j];//与末尾调换
    a[j] = sign;
    Quick_Sort(a, begin, i-1);//重复过程
    Quick_Sort(a, i+1, end);
```

将边一条一条加入图里

```
for(int i=0;i<lineNum && count<n-1;i++){
        int tem=a[l[i].start][l[i].end];
        a[l[i].start][l[i].end]=l[i].value;
        a[l[i].end][l[i].start]=l[i].value;
        if(kruskal(l[i].start, l[i].start, tem) == -1){
            count++;
            length=length+l[i].value;
        }
        else{
            a[l[i].start][l[i].end]=tem;
        };

    }
```

算法时间及空间复杂性分析

采用快排则时间复杂度为O(N log N)

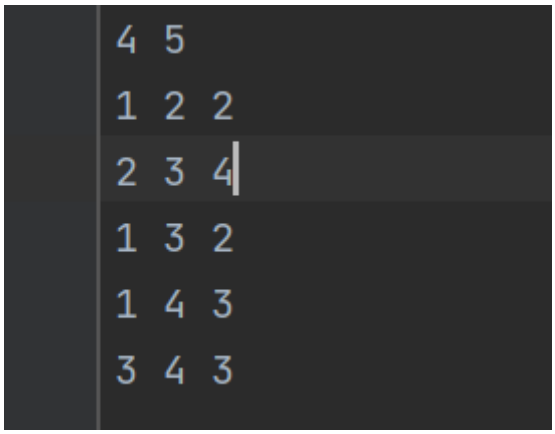
三、实验结果

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

可以用迪杰斯特拉辅助来判断闭环,可以启发现实中的一些问题比如铺设光纤

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];
class line{
public:
    int start;
    int end;
    int value=0;
};
line l[5001];
#define rep(i, a, b) for (int i = a; i <= b; ++i)
int findmin(int D[],int S[]) {
    int min = 2000001;
    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;
```

```
}
void startA(){
    for(int i=0;i<5001;i++){
        for(int i1=0;i1<5001;i1++){
            a[i][i1]=200000*n+1;
        }
    }

}
int kruskal(int startpoint, int endpoint, int tem){
    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;


}
void Quick_Sort(line a[], int begin, int end){
    if(begin >= end)//只有一个元素,递归完成
```

```
        return;
    line sign = a[end];
    int i = begin;
    int j = end;
    while(i != j){//对右边大和左边小的值进行调换
        while(a[i].value <= sign.value && j > i)
            i++;
        while(a[j].value >= sign.value && j > i)
            j--;
        if(j > i){
            line t = a[i];
            a[i] = a[j];
            a[j] = t;
        }
    }
    a[end] = a[j];//与末尾调换
    a[j] = sign;
    Quick_Sort(a, begin, i-1);//重复过程
    Quick_Sort(a, i+1, end);
}
int main()
{
    fstream fin("kruskal.in", ios::in), fout("kruskal.out", ios::out);
    fin >> n;
    fin>>lineNum;
    startA();
    int start,end;
    rep(i, 0, lineNum-1) {
        fin >> start;
        l[i].start=start;
        fin >> end;
        l[i].end=end;
        fin >>  l[i].value;
    }
    int count=0;
    int length=0;
    Quick_Sort(l,0,lineNum-1);
    for(int i=0;i<lineNum && count<n-1;i++){
        int tem=a[l[i].start][l[i].end];
        a[l[i].start][l[i].end]=l[i].value;
        a[l[i].end][l[i].start]=l[i].value;
        if(kruskal(l[i].start, l[i].start, tem) == -1){
            count++;
            length=length+l[i].value;
        }
        else{
            a[l[i].start][l[i].end]=tem;
        };

    }
    fout<<length;


    return 0;
```

```
}
```