**国王的烦恼**

问题描述

　　C国由n个小岛组成，为了方便小岛之间联络，C国在小岛间建立了m座大桥，每座大桥连接两座小岛。两个小岛间可能存在多座桥连接。然而，由于海水冲刷，有一些大桥面临着不能使用的危险。  
  
　　如果两个小岛间的所有大桥都不能使用，则这两座小岛就不能直接到达了。然而，只要这两座小岛的居民能通过其他的桥或者其他的小岛互相到达，他们就会安然无事。但是，如果前一天两个小岛之间还有方法可以到达，后一天却不能到达了，居民们就会一起抗议。  
  
　　现在C国的国王已经知道了每座桥能使用的天数，超过这个天数就不能使用了。现在他想知道居民们会有多少天进行抗议。

输入格式

　　输入的第一行包含两个整数n, m，分别表示小岛的个数和桥的数量。  
　　接下来m行，每行三个整数a, b, t，分别表示该座桥连接a号和b号两个小岛，能使用t天。小岛的编号从1开始递增。

输出格式

　　输出一个整数，表示居民们会抗议的天数。

样例输入

4 4  
1 2 2  
1 3 2  
2 3 1  
3 4 3

样例输出

2

样例说明

　　第一天后2和3之间的桥不能使用，不影响。  
　　第二天后1和2之间，以及1和3之间的桥不能使用，居民们会抗议。  
　　第三天后3和4之间的桥不能使用，居民们会抗议。

数据规模和约定

　　对于30%的数据，1<=n<=20，1<=m<=100；  
　　对于50%的数据，1<=n<=500，1<=m<=10000；  
　　对于100%的数据，1<=n<=10000，1<=m<=100000，1<=a, b<=n， 1<=t<=100000。

本题的C++参考代码如下：

#include <iostream>

#include <vector>

#include <algorithm>

#include <cstring>

using namespace std;

int fa[10005];

struct Edge

{

int x,y;

int t;

friend bool operator <(Edge e1,Edge e2)

{

return e1.t>e2.t;

}

}e[100005];

int vis[100005];

int findfa(int x)

{

if (fa[x]==x) return x;

else return fa[x]=findfa(fa[x]);

}

int merges(int a,int b)

{

if (findfa(a)!=findfa(b)){

fa[findfa(a)]=findfa(b);

return 1;

}

else return 0;

}

int main()

{

memset(vis,0,sizeof(vis));

ios::sync\_with\_stdio(false);

int n,m;

cin >>n>>m;

for (int i=1;i<=n;i++)

{

fa[i]=i;

}

int a,b,c;

for (int i=0;i<m;i++)

{

cin >>a>>b>>c;

e[i].x=a;e[i].y=b;e[i].t=c;

}

sort(e,e+m);

for (int i=0;i<m;i++)

{

//cout <<endl;

//cout <<e[i].x<<" "<<e[i].y<<" "<<e[i].t<<" "<<fa[e[i].x]<<" "<<fa[e[i].y]<<endl;

int flag=merges(e[i].x,e[i].y);

//cout <<flag<<endl;

if (!vis[e[i].t]) vis[e[i].t]=flag;

}

int ans=0;

for (int i=1;i<=100000;i++)

{

ans+=vis[i];

}

cout <<ans<<endl;

}

本题的C参考代码如下：

#include <stdio.h>

#include <stdlib.h>

struct NODE

{

int u,v,w;}edge[100000];

int father[10001];

int cmp(const void \*a,const void \*b)

{

return (\*(struct NODE \*)b).w-(\*(struct NODE \*)a).w;

}

int find(int num)

{

int root=num,tmp;

while(father[root]!=root)

root=father[root];

while(root!=num)

{

tmp=father[num];

father[num]=root;

num=tmp;

}

return root;

}

int merge(int n1,int n2)

{

int root1=find(n1),root2=find(n2);

if(root1!=root2)

{

father[root1]=root2;

return 1;

}

return 0;

}

int main()

{

int i,n,m,cnt=0,tmp=0,ans=0;

scanf("%d%d",&n,&m);

for(i=0;i<m;i++)

scanf("%d%d%d",&edge[i].u,&edge[i].v,&edge[i].w);

for(i=1;i<=n;i++)

father[i]=i;

qsort(edge,m,sizeof(edge[0]),cmp);

for(i=0;i<m;i++)

{

if(merge(edge[i].u,edge[i].v))

{

if(tmp!=edge[i].w)

ans++;

tmp=edge[i].w;

cnt++;

}

if(cnt==n-1)

break;

}

printf("%d\n",ans);

return 0;

}

本题的Java参考代码如下：

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.util.\*;

public class Main {

static int n;

static int sum = 0;

static int a[];

static Edge p[];

public static void main(String[] args) throws Exception {

BufferedReader buf = new BufferedReader(

new InputStreamReader(System.in));

String read[] = buf.readLine().split("\\s+");

n = Integer.parseInt(read[0]);

int m = Integer.parseInt(read[1]);

p = new Edge[m];

a = new int[n];

int s, e, w, t;

for (int i = 0; i < m; i++) {

read = buf.readLine().split("\\s+");

s = Integer.parseInt(read[0]) - 1;

e = Integer.parseInt(read[1]) - 1;

w = Integer.parseInt(read[2]);

p[i] = new Edge(s, e, w);

}

java.util.Arrays.sort(p);

boolean flag = false;

init();

for (int i = 0; i < m; i++) {

flag = false;

s = p[i].s;

e = p[i].e;

if (!isConnect(s, e)) {

flag = true;

union(s, e);

}

while (i < m - 1 && p[i].w == p[i + 1].w) {

s = p[i + 1].s;

e = p[i + 1].e;

if (flag && !isConnect(s, e))

union(s, e);

if (!flag && !isConnect(s, e)) {

flag = true;

union(s, e);

}

i++;

}

if (flag)

sum++;

}

System.out.println(sum);

}

private static boolean isConnect(int i, int j) {

if (find(i) == find(j))

return true;

return false;

}

private static void init() {

for (int i = 0; i < n; i++)

a[i] = i;

}

private static void union(int x, int y) {

a[find(x)] = find(y);

}

private static int find(int x) {

if (a[x] == x)

return x;

a[x] = find(a[x]);

return a[x];

}

}

class Edge implements Comparable<Edge> {

int s;

int e;

int w;

public Edge(int s, int e, int w) {

super();

this.s = s;

this.e = e;

this.w = w;

}

@Override

public int compareTo(Edge p) {

if (this.w < p.w)

return 1;

else if (this.w > p.w)

return -1;

return 0;

}

}