图论

图的相关概念

图论相关概念 - OI Wiki (oi-wiki.org)

图的存储

1.接邻矩阵

```
for (int i = 1; i <= m; i++)
{
    int u, v;
    cin >> u >> v;
    mp[u][v] = 1;
}
```

复杂度

查询是否存在某条边:o(1)

遍历一个点的所有出边:o(n)

遍历整张图:o(n^2)

空间复杂度：o(n^2)

应用

邻接矩阵只适用于没有重边（或重边可以忽略）的情况。

其最显著的优点是可以 查询一条边是否存在。

由于邻接矩阵在稀疏图上效率很低（尤其是在点数较多的图上，空间无法承受），所以一般只会在稠密图上使用邻接矩阵。

2.接邻表（推荐）

```
 for (int i = 1; i <= m; i++)
{
    int u, v;
    cin >> u >> v;
    adj[u].push_back(v);
  }

//带权图存储
#include<cstdio>
#include<algorithm>
#include<cstring>
#include<iostream>
#include<vector>
#define N 10000
using namespace std;
struct EDGE
{
    int to;//终点
    int cost;//边的权值
};
vector<EDGE>G[N];//G[i]中i表示出发点
int m,n;
int temp1;//出发点
int main()
{
    scanf("%d%d",&n,&m);
    while(m--)
    {
        EDGE e;
        scanf("%d%d%d",&temp1,&e.to,&e.cost);//输入出发点，终点，边的权值
        G[temp1].push_back(e);//将数据压入动态数组，表示在这个出发点下引出的边
                              //相当于二维动态数组
    }
    for (int i=1;i<=n;i++)//按照出发点的顺序遍历
    {
        for(int j=0;j<G[i].size();j++)//遍历出发点所引出的边
        {
            EDGE e=G[i][j];//1以二维数组形式输出
            printf("from %d to %d,the cost is %d\n",i,e.to,e.cost);
```

```
            }
        }
        return 0;
    }
```

复杂度

查询是否存在u到v的边：o(d+(u))（如果事先进行了排序就可以使用 二分查找 做到 o(log(d+(u))))

遍历点u的所有出边:o(d+(u))

遍历整张图:o(m+n)

空间复杂度:o(m)

应用

存各种图都很适合，除非有特殊需求（如需要快速查询一条边是否存在，且点数较少，可以使用邻接矩阵）。

尤其适用于需要对一个点的所有出边进行排序的场合。

图的遍历

bfs，dfs等

最短路

FLOYD算法

是用来求任意两个结点之间的最短路的。

复杂度比较高，但是常数小，容易实现。（三重 `for` ）

适用于任何图，不管有向无向，边权正负，但是最短路必须存在。（不能有个负环）

实现

```
for(int i=1;i<=n;i++)//floyd精髓-*从j号顶点到k号顶点只经过前i号点的最短路程*
   for(int j=1;j<=n;j++)
       for(int k=1;k<=n;k++)
```

```
            if(dis[j][i]+dis[i][k]<dis[j][k])
                dis[j][k]=dis[j][i]+dis[i][k];
```

Dijkstra 算法

是一种求解 **非负权图** 上单源最短路径的算法。

Dijkstra算法算是**贪心思想**实现的，首先把起点到所有点的距离存下来找个最短的，然后松弛一次再找出最短的，（所谓的松弛操作就是，遍历一遍看通过刚刚找到的距离最短的点作为中转站会不会更近，如果更近了就更新距离）这样把所有的点找遍之后就存下了起点到其他所有点的最短距离。

```
//暴力解法
struct edge {
  int v, w;
};

vector<edge> e[maxn];
int dis[maxn], vis[maxn];

void dijkstra(int n, int s) {
  memset(dis, 63, sizeof(dis));
  dis[s] = 0;
  for (int i = 1; i <= n; i++) {
    int u = 0, mind = 0x3f3f3f3f;
    for (int j = 1; j <= n; j++)
      if (!vis[j] && dis[j] < mind) u = j, mind = dis[j];
    vis[u] = true;
    for (auto ed : e[u]) {
      int v = ed.v, w = ed.w;
      if (dis[v] > dis[u] + w) dis[v] = dis[u] + w;
    }
  }
}
//优先队列
struct edge {
  int v, w;
};

struct node {
  int dis, u;

  bool operator>(const node& a) const { return dis > a.dis; }
};

vector<edge> e[maxn];
int dis[maxn], vis[maxn];
priority_queue<node, vector<node>, greater<node> > q;

void dijkstra(int n, int s) {
  memset(dis, 63, sizeof(dis));
```

```
    dis[s] = 0;
    q.push({0, s});
    while (!q.empty()) {
      int u = q.top().u;
      q.pop();
      if (vis[u]) continue;
      vis[u] = 1;
      for (auto ed : e[u]) {
        int v = ed.v, w = ed.w;
        if (dis[v] > dis[u] + w) {
          dis[v] = dis[u] + w;
          q.push({dis[v], v});
        }
      }
    }
  }
```

并查集

【算法与数据结构】— 并查集

生成树

最小生成树详解(模板 + 例题)

例题

【T1】P8794 [蓝桥杯 2022 国 A] 环境治理

P8794 [蓝桥杯 2022 国 A] 环境治理

提交答案 加入题单

题目描述

复制Markdown 展开

LQ 国拥有 n 个城市，从 0 到 $n-1$ 编号，这 n 个城市两两之间都有且仅有一条双向道路连接，这意味着任意两个城市之间都是可达的。每条道路都有一个属性 D，表示这条道路的灰尘度。当从一个城市 A 前往另一个城市 B 时，可能存在多条路线，每条路线的灰尘度定义为这条路线所经过的所有道路的灰尘度之和，LQ 国的人都很讨厌灰尘，所以他们总会优先选择灰尘度最小的路线。

LQ 国很看重居民的出行环境，他们用一个指标 P 来衡量 LQ 国的出行环境，P 定义为：

$$P=\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}d(i,j)$$

其中 $d(i,j)$ 表示城市 i 到城市 j 之间灰尘度最小的路线对应的灰尘度的值。

为了改善出行环境，每个城市都要有所作为，当某个城市进行道路改善时，会将与这个城市直接相连的所有道路的灰尘度都减少 1，但每条道路都有一个灰尘度的下限值 L，当灰尘度达到道路的下限值时，无论再怎么改善，道路的灰尘度也不会再减小了。

具体的计划是这样的：

- 第 1 天，0 号城市对与其直接相连的道路环境进行改善；
- 第 2 天，1 号城市对与其直接相连的道路环境进行改善；

……

- 第 n 天，$n-1$ 号城市对与其直接相连的道路环境进行改善；
- 第 $n+1$ 天，0 号城市对与其直接相连的道路环境进行改善；
- 第 $n+2$ 天，1 号城市对与其直接相连的道路环境进行改善；

……

LQ 国想要使得 P 指标满足 $P\le Q$。请问最少要经过多少天之后，P 指标可以满足 $P\le Q$。如果在初始时就已经满足条件，则输出 0；如果永远不可能满足，则输出 -1。

输入格式

输入的第一行包含两个整数 n,Q，用一个空格分隔，分别表示城市个数和期望达到的 P 指标。

接下来 n 行，每行包含 n 个整数，相邻两个整数之间用一个空格分隔，其中第 i 行第 j 列的值 $D_{i,j}(D_{i,j}=D_{j,i},D_{i,i}=0)$ 表示城市 i 与城市 j 之间直接相连的那条道路的灰尘度。

接下来 n 行，每行包含 n 个整数，相邻两个整数之间用一个空格分隔，其中第 i 行第 j 列的值 $L_{i,j}(L_{i,j}=L_{j,i},L_{i,i}=0)$ 表示城市 i 与城市 j 之间直接相连的那条道路的灰尘度的下限值。

输出格式

输出一行包含一个整数表示答案。

输入输出样例

输入 #1 复制

```
3 10
0 2 4
2 0 1
4 1 0
0 2 2
2 0 0
```

输出 #1 复制

```
2
```

```
200
```

```cpp
#include<bits/stdc++.h>
using namespace std;
int mp[105][105];//存图
int lim[105][105];//最小灰尘度
int dust[105][105];
int ans=-1,n;
long long q;
bool check(int day)
{
    for(int i=1;i<=n;i++)//算这一天每个城市的灰尘度
        for(int j=1;j<=n;j++)
            dust[i][j]=max(mp[i][j]-day/n-(day%n>=i?1:0),lim[i][j]);
    for(int k=1;k<=n;k++)//floyd
        for(int i=1;i<=n;i++)
            for(int j=1;j<=n;j++)
                dust[i][j]=min(dust[i][j],dust[i][k]+dust[k][j]);
    long long sum=0;
    for(int i=1;i<=n;i++)//算P
        for(int j=1;j<=n;j++)
            sum+=dust[i][j];
    if(sum>=0&&sum<=q)//判断是否符合要求
    {
        ans=day;
        return true;
    }
    else return false;
}
void search()//二分搜答案
{
    int l=0,r=0x3f3f3f3f;//为啥l=-1不行呀....
    while(l<=r)         //r=INT_MAX会TLE..QWQ
    {
        int mid=(l+r)/2;
        if(check(mid)) r=mid-1;
        else l=mid+1;
    }
}
int main()
{
    cin>>n>>q;
    for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++)
            cin>>mp[i][j];
    for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++)
            cin>>lim[i][j];
    search();
    cout<<ans;
}
```

【T2】P8654 [蓝桥杯 2017 国 C] 合根植物

题目描述

复制Markdown

w 星球的一个种植园，被分成 $m \times n$ 个小格子（东西方向 m 行，南北方向 n 列）。每个格子里种了一株合根植物。

这种植物有个特点，它的根可能会沿着南北或东西方向伸展，从而与另一个格子的植物合成为一体。

如果我们告诉你哪些小格子间出现了连根现象，你能说出这个园中一共有多少株合根植物吗？

输入格式

第一行，两个整数 m，n，用空格分开，表示格子的行数、列数（$1 < m, n < 1000$）。

接下来一行，一个整数 k，表示下面还有 k 行数据（$0 < k < 10^5$）。

接下来 k 行，第行两个整数 a，b，表示编号为 a 的小格子和编号为 b 的小格子合根了。

格子的编号一行一行，从上到下，从左到右编号。

比如：5×4 的小格子，编号：

```
1  2  3  4
5  6  7  8
9  10 11 12
13 14 15 16
17 18 19 20
```

输出格式

一行一个整数，表示答案

输入输出样例

输入 #1 复制

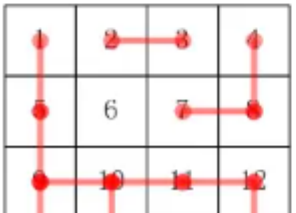
```
5 4
16
2 3
1 5
5 9
4 8
7 8
9 10
10 11
11 12
10 14
12 16
14 18
17 18
15 19
19 20
9 13
13 17
```

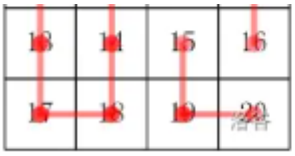
输出 #1

```
5
```

说明/提示

样例解释





时限 1 秒, 256M。蓝桥杯 2017 年第八届国赛

```
#include<bits/stdc++.h>
using namespace std;
int n,m;
int num,ans;
const int N=1e6+5;
int disjoint_set[N],depth[N];//并查集和深度

void init()
{
    for(int i=1;i<=n;i++)
    {
        disjoint_set[i]=i;//初始化每个节点的父节点为本身
        depth[i]=0;//深度
    }
}
int find(int a)//查找这个集合最终父节点
{
    if(disjoint_set[a]==a)
        return a;
    else
        return disjoint_set[a]=find(disjoint_set[a]);
}
void  merge(int a,int b)//合并
{
    a=find(a);
    b=find(b);//先找到两个集合的源头
    if(a==b) return;//如果源头是同个节点说明属于同个集合
    else if(depth[a]>depth[b])
    {
        disjoint_set[b]=a;//如果a集合深度大于b，把b挂到a上
    }
    else//这里包含两种情况，d[a]<d[b]&d[a]=d[b]
    {
        if(depth[a]==depth[b])//深度相等不妨把a挂b上
            depth[b]++;
        disjoint_set[a]=b;//d[a]<d[b]的话不执行if直接把a挂b上
    }
}
int main()
{
    cin>>m>>n;
    n*=m;//n现在代表总格子数
    cin>>num;
    init();
    for(int i=1;i<=num;i++)
    {
        int x,y;
        cin>>x>>y;
```

```
            merge(x,y);
        }
        for(int i=1;i<=n;i++)//对每个格子寻根
        {
            if(find(i)==i)
                ans++;
        }
        cout<<ans;
    }
```

【T3】P3958 [NOIP2017 提高组] 奶酪

P3958 [NOIP2017 提高组] 奶酪

提交答案 加入题单

题目背景

复制Markdown 展开

NOIP2017 提高组 D2T1

题目描述

现有一块大奶酪，它的高度为 h，它的长度和宽度我们可以认为是无限大的，奶酪中间有许多半径相同的球形空洞。我们可以在这块奶酪中建立空间坐标系，在坐标系中，奶酪的下表面为 $z=0$，奶酪的上表面为 $z=h$。

现在，奶酪的下表面有一只小老鼠 Jerry，它知道奶酪中所有空洞的球心所在的坐标。如果两个空洞相切或是相交，则 Jerry 可以从其中一个空洞跑到另一个空洞，特别地，如果一个空洞与下表面相切或是相交，Jerry 则可以从奶酪下表面跑进空洞；如果一个空洞与上表面相切或是相交，Jerry 则可以从空洞跑到奶酪上表面。

位于奶酪下表面的 Jerry 想知道，在不破坏奶酪的情况下，能否利用已有的空洞跑到奶酪的上表面去？

空间内两点 $P_1(x_1,y_1,z_1)$、$P2(x_2,y_2,z_2)$ 的距离公式如下：

$$\operatorname{dist}(P_1,P_2)=\sqrt{(x_1-x_2)^2+(y_1-y_2)^2+(z_1-z_2)^2}$$

输入格式

每个输入文件包含多组数据。

第一行，包含一个正整数 T，代表该输入文件中所含的数据组数。

接下来是 T 组数据，每组数据的格式如下：第一行包含三个正整数 n,h,r，两个数之间以一个空格分开，分别代表奶酪中空洞的数量，奶酪的高度和空洞的半径。

接下来的 n 行，每行包含三个整数 x,y,z，两个数之间以一个空格分开，表示空洞球心坐标为 (x,y,z)。

输出格式

T 行，分别对应 T 组数据的答案，如果在第 i 组数据中，Jerry 能从下表面跑到上表面，则输出 `Yes`，如果不能，则输出 `No`。

输入输出样例

输入 #1 复制

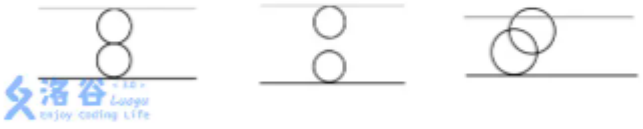
```
3
2 4 1
0 0 1
0 0 3
2 5 1
0 0 1
0 0 4
2 5 2
0 0 2
2 0 4
```

输出 #1 复制

```
Yes
No
Yes
```

说明/提示

【输入输出样例 1 说明】



第一组数据，由奶酪的剖面图可见：

第一个空洞在 $(0,0,0)$ 与下表面相切；

第二个空洞在 $(0,0,4)$ 与上表面相切；

两个空洞在 $(0,0,2)$ 相切。

输出 `Yes`。

第二组数据，由奶酪的剖面图可见：

两个空洞既不相交也不相切。

输出 `No`。

第三组数据，由奶酪的剖面图可见：

两个空洞相交，且与上下表面相切或相交。

输出 `Yes`。

【数据规模与约定】

对于 20% 的数据，$n=1$，$1\le h,r\le 10^4$，坐标的绝对值不超过 10^4。

对于 40% 的数据，$1\le n\le 8$，$1\le h,r\le 10^4$，坐标的绝对值不超过 10^4。

对于 80% 的数据，$1\le n\le 10^3$，$1\le h,r\le 10^4$，坐标的绝对值不超过 10^4。

对于100%的数据，$1 \le n \le 1 \times 10^3$，$1 \le h,r \le 10^9$，$T \le 20$，坐标的绝对值不超过$10^9$。

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e5+5;
int disjoint_set[1005];
long long x[N],y[N],z[N];
int up_number[N],down_number[N];//up_number记录与顶面相交的洞的序号,down_number记录与底面相交的洞的序号

int find(int x)//查找+路径压缩
{
    if(x!=disjoint_set[x])
        disjoint_set[x]=find(disjoint_set[x]);
    return disjoint_set[x];
}
long long distance(long long x1,long long y1,long long z1,long long x2,long long y2,long long z2)
{
    return(x1-x2)*(x1-x2)+(y1-y2)*(y1-y2)+(z1-z2)*(z1-z2);
}//两点距离公式，注意这里算的是距离平方。

int main()
{
    int t,n,h;
    long long r;
    cin>>t;
    for(int i=1;i<=t;i++)
    {
        cin>>n>>h>>r;
        int up=0,down=0;//up->跟顶面相交的洞个数,down->底
        for(int j=1;j<=n;j++)
            disjoint_set[j]=j;//并查集初始化
        for(int j=1;j<=n;j++)/**主体**/
        {
            cin>>x[j]>>y[j]>>z[j];
            if(z[j]+r>=h)//判断是否跟顶面相交
            {
                up++;
                up_number[up]=j;
            }
            if(z[j]-r<=0)//判断是否跟底相交
            {
                down++;
                down_number[down]=j;
            }
            for(int k=1;k<=j;k++)//枚举之前的洞是否跟这个洞相交，相交就合并
            {
                if((x[j]-x[k])*(x[j]-x[k])+(y[j]-y[k])*(y[j]-y[k])>4*r*r)
continue;//防止爆ll特判(如果两维都超那就不需要考虑第三维)
                else if(distance(x[j],y[j],z[j],x[k],y[k],z[k])<=4*r*r)
                {
                    int a1=find(j);
                    int a2=find(k);
                    if(a1!=a2) disjoint_set[a1]=a2;
```

```
                }
            }
        }
        bool flag=false;
        for(int j=1;j<=up;j++)//看每个并查集是否有洞链接上下面
        {
            for(int k=1;k<=down;k++)
            {
                if(find(up_number[j])==find(down_number[k]))//这样说明相通
                {
                    flag=true;
                    break;
                }
            }
            if(flag) break;
        }
        if(flag) cout<<"Yes"<<endl;
        else cout<<"No"<<endl;
    }
}
```

【T4】P2504 [HAOI2006]聪明的猴子

P2504 [HAOI2006]聪明的猴子

提交答案 加入题单

题目描述

复制Markdown 展开

在一个热带雨林中生存着一群猴子，它们以树上的果子为生。昨天下了一场大雨，现在雨过天晴，但整个雨林的地表还是被大水淹没着，部分植物的树冠露在水面上。猴子不会游泳，但跳跃能力比较强，它们仍然可以在露出水面的不同树冠上来回穿梭，以找到喜欢吃的果实。

现在，在这个地区露出水面的有N棵树，假设每棵树本身的直径都很小，可以忽略不计。我们在这块区域上建立直角坐标系，则每一棵树的位置由其所对应的坐标表示(任意两棵树的坐标都不相同)。

在这个地区住着的猴子有M个，下雨时，它们都躲到了茂密高大的树冠中，没有被大水冲走。由于各个猴子的年龄不同、身体素质不同，它们跳跃的能力不同。有的猴子跳跃的距离比较远(当然也可以跳到较近的树上)，而有些猴子跳跃的距离就比较近。这些猴子非常聪明，它们通过目测就可以准确地判断出自己能否跳到对面的树上。

【问题】现已知猴子的数量及每一个猴子的最大跳跃距离，还知道露出水面的每一棵树的坐标，你的任务是统计有多少个猴子可以在这个地区露出水面的所有树冠上觅食。

输入格式

输入文件monkey.in包括：

第1行为一个整数，表示猴子的个数M(2<=M<=500)；

第2行为M个整数，依次表示猴子的最大跳跃距离（每个整数值在1--1000之间）；

第3行为一个整数表示树的总棵数N(2<=N<=1000)；

第4行至第N+3行为N棵树的坐标（横纵坐标均为整数，范围为：-1000--1000）。

（同一行的整数间用空格分开）

输出格式

输出文件monkey.out包括一个整数，表示可以在这个地区的所有树冠上觅食的猴子数。

输入输出样例

输入 #1 复制

```
4
 1 2 3 4
6
0 0
1 0
1 2
-1 -1
-2 0
2 2
```

输出 #1 复制

```
3
```

说明/提示

【数据规模】

对于40%的数据，保证有2<=N <=100，1<=M<=100

对于全部的数据，保证有2<=N <= 1000，1<=M=500

感谢@charlie003 修正数据

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e6+5;
int n,m,k,ans=0;
double sum=-1;
int a[N][3],b[N],pre[N];

struct coordinate
{
    int x,y;
    double p;//p->长度
}z[N];
int cmp(coordinate a,coordinate b)//按边的长短从小到大排序
{
    return a.p<b.p;
}
int find(int x)//找根节点
{
    if(pre[x]==x)
        return x;
    return pre[x]=find(pre[x]);
}
int main()
{
    cin>>m;
    for(int i=1;i<=m;i++)
        cin>>b[i];
    cin>>n;
    for(int i=1;i<=n;i++)
        cin>>a[i][1]>>a[i][2];
    for(int i=1;i<=n;i++)//建边
    {
        for(int j=1;j<=n;j++)
        {
            if(i!=j)
            {
                k++;
                z[k].x=i;//记录这条边连的两个点
                z[k].y=j;//
                z[k].p=sqrt((a[i][1]-a[j][1])*(a[i][1]-a[j][1])+(a[i][2]-a[j][2])*
(a[i][2]-a[j][2]));
            }
        }
    }
    int cnt=n;//记录树上有几条边
    sort(z+1,z+k+1,cmp);
    for(int i=1;i<=n;i++)
        pre[i]=i;
    for(int i=1;i<=k;i++)/**Kruskal**/
    {
        if(cnt==1) break;//表示所有的点都在树上
        int s1=find(z[i].x),s2=find(z[i].y);//判断起点终点在哪棵树上
        if(s1!=s2)//如果起点终点不在同个树就合并
        {
            pre[s1]=s2;
```

```
            cnt--;
            sum=z[i].p;
        }
    }
    for(int i=1;i<=m;i++)
    {
        if(sum<=b[i])
            ans++;
    }
    cout<<ans<<endl;
}
```

【T5】P1636 Einstein学画画

P1636 Einstein学画画

提交答案 加入题单

题目描述

复制Markdown 展开

Einstein 学起了画画。

此人比较懒~~，他希望用最少的笔画画出一张画……

给定一个无向图，包含 n 个顶点（编号 $1 \sim n$），m 条边，求最少用多少笔可以画出图中所有的边。

输入格式

第一行两个整数 n, m。

接下来 m 行，每行两个数 a, b（$a \neq b$），表示 a, b 两点之间有一条边相连。

一条边不会被描述多次。

输出格式

一个数，即问题的答案。

输入输出样例

输入 #1 复制

```
5 5
2 3
2 4
2 5
3 4
4 5
```

输出 #1 复制

```
1
```

说明/提示

对于 50% 的数据，$n \leq 50$，$m \leq 100$。

对于 100% 的数据，$1 \leq n \leq 1000$，$1 \leq m \leq 10^5$。

```cpp
#include<bits/stdc++.h>
using namespace std;
int mp[10005];//mp[i]:点i的度数
int n,m ,ans;
int main()
{
    cin>>n>>m;
    for(int i=1;i<=m;i++)
```

```
    {
        int x,y;
        cin>>x>>y;
        mp[x]++;
        mp[y]++;//联通的话两点度数都+1
    }
    for(int i=1;i<=n;i++)
    {
        if(mp[i]%2==1)//如果度数是奇数就要多一笔画完（实际就是这样至于为啥..）
            ans++;
    }
    if(ans)
        cout<<ans/2;//因为一次可以连两个点所以/2(注意：一个连通图只可能有偶数个奇点)
    else cout<<1;//都是偶数度数就一笔画完啦（每个点都有进有出）
    return 0;
}
/*
补充:
点的度，就是指和该顶点相关联的边数。

出度：有向图中从某顶点出发的边数。

入度：有向图中在某顶点结束的边数。

欧拉回路:
若恰通过图中每条边一次回到起点,则称该回路为欧拉(Euler)回路。
具有欧拉回路的图称为欧拉图。
定理1:
一个无向图是欧拉图，当且仅当该图所有顶点度数都是偶数。
一个有向图是欧拉图，当且仅当该图所有顶点度数都是0(入度与出度之和)。
定理2:
存在欧拉回路的条件：图是连通的，且不存在奇点（顶点度数为奇数）

欧拉路（此题解法 ！）
若从起点到终点的路径恰通过图中每条边一次（起点与终点是不同的点），
则该路径称为欧拉路。
定理1:
存在欧拉路的条件：图是连通的，且存在2个奇点。
如果存在2个奇点，则欧拉路一定是从一个奇点出发，以另一个奇点结束。
*/
```

【T6】P1119 灾后重建

P1119 灾后重建

提交答案 加入题单

题目背景

复制Markdown 展开

B 地区在地震过后，所有村庄都造成了一定的损毁，而这场地震却没对公路造成什么影响。但是在村庄重建好之前，所有与未重建完成的村庄的公路均无法通车。换句话说，只有连接着两个重建完成的村庄的公路才能通车，只能到达重建完成的村庄。

题目描述

给出 B 地区的村庄数 N，村庄编号从 0 到 $N-1$，和所有 M 条公路的长度，公路是双向的。并给出第 i 个村庄重建完成的时间 t_i，你可以认为是同时开始重建并在第 t_i 天重建完成，并且在当天即可通车。若 t_i 为 0 则说明地震未对此地区造成损坏，一开始就可以通车。之后有 Q 个询问 (x,y,t)，对于每个询问你要回答在第 t 天，从村庄 x 到村庄 y 的最短路径长度为多少。如果无法找到从 x 村庄到 y 村庄的路径，经过若干个已重建完成的村庄，或者村庄 x 或村庄 y 在第 t 天仍未重建完成，则需要返回 `-1`。

输入格式

第一行包含两个正整数N,M，表示了村庄的数目与公路的数量。

第二行包含N个非负整数$t_0,t_1,\ldots,t_{N-1}$，表示了每个村庄重建完成的时间，数据保证了$t_0 \le t_1 \le \ldots \le t_{N-1}$。

接下来M行，每行3个非负整数i,j,w，w为不超过10000的正整数，表示了有一条连接村庄i与村庄j的道路，长度为w，保证$i \neq j$，且对于任意一对村庄只会存在一条道路。

接下来一行也就是$M+3$行包含一个正整数Q，表示Q个询问。

接下来Q行，每行3个非负整数x,y,t，询问在第t天，从村庄x到村庄y的最短路径长度为多少，数据保证了t是不下降的。

输出格式

共Q行，对每一个询问(x,y,t)输出对应的答案，即在第t天，从村庄x到村庄y的最短路径长度为多少。如果在第t天无法找到从x村庄到y村庄的路径，经过若干个已重建完成的村庄，或者村庄x或村庄y在第t天仍未修复完成，则输出-1。

输入输出样例

输入 #1 复制

```
4 5
1 2 3 4
0 2 1
2 3 1
3 1 2
2 1 4
0 3 5
4
2 0 2
0 1 2
0 1 3
0 1 4
```

输出 #1 复制

```
-1
-1
5
4
```

说明/提示

对于30%的数据，有$N \le 50$；

对于30%的数据，有$t_i=0$，其中有20%的数据有$t_i=0$且$N>50$；

对于50%的数据，有$Q\leq 100$；

对于100%的数据，有$N\leq 200$，$M\leq N\times(N-1)/2$，$Q\leq 50000$，所有输入数据涉及整数均不超过100000。

```
#include<bits/stdc++.h>
using namespace std;
const int N=205;
int n,m;
int a[N],f[N][N];//f[][]->邻接矩阵存边,a[]->修复完成时间
void floyd(int k)
{
    for(int i=0;i<n;i++)
        for(int j=0;j<n;j++)
        {
            if(f[i][j]>f[i][k]+f[j][k])
                f[i][j]=f[j][i]=f[i][k]+f[j][k];
        }
}
int main()
{
    cin>>n>>m;
    for(int i=0;i<n;i++)
        cin>>a[i];//输入每个村庄修好要的时间
    for(int i=0;i<n;i++)
        for(int j=0;j<n;j++)
            f[i][j]=1e9;//先初始化为最大值(用INT_MAX会炸???)
    for(int i=0;i<n;i++)
        f[i][i]=0;//一个点到自己距离为0
    int s1,s2,s3;
    for(int i=1;i<=m;i++)
    {
        cin>>s1>>s2>>s3;
        f[s1][s2]=f[s2][s1]=s3;//初始化边长(无向边，存两次)
    }
    int q;
    cin>>q;
    int now=0;//表示现在是第几天
    for(int i=1;i<=q;i++)//处理询问
    {
        cin>>s1>>s2>>s3;//第s3天村s1到s2最短路
        while(a[now]<=s3&&now<n)//模拟第s3天各条路修复情况
        {
            floyd(now);
            now++;
        }
        if(a[s1]>s3||a[s2]>s3) cout<<-1<<endl;//村庄没建好
        else
        {
            if(f[s1][s2]==1e9) cout<<-1<<endl;//两点不连通
            else cout<<f[s1][s2]<<endl;
        }
    }
```

```
}
```

【T7】U80592 【模板】floyd

洛谷 / 题目列表 / 题目详情

U80592 【模板】floyd

提交答案 加入题单

题目背景

复制Markdown 展开

模板题，无背景

题目描述

给出n个点，m条边的无向图，求每个点到其他点的距离之和%998244354的值

输入格式

第一行两个数n，m含义如上 从第二行开始，共m行，每行三个数x，y，l，代表从x到y点的长度为l

输出格式

n行，每行一个数，第i行代表点i到其他点的距离之和

输入输出样例

输入 #1 复制

```
2 1
1 2 4
```

输出 #1 复制

```
4
4
```

输入 #2 复制

```
4 5
1 2 1
1 3 2
2 3 2
3 4 3
2 4 4
```

输出 #2 复制

```
8
7
7
12
```

说明/提示

模板题，保证图联通 n<=500 m<=10000 1<=x,y<=n l<=1e9

```cpp
#include<bits/stdc++.h>
using namespace std;
const long long mod=998244354;
long long dis[5005][5005];
int main()
{
    int n,m;
    cin>>n>>m;
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=n;j++)
        {
            dis[i][j]=INT_MAX;
            dis[i][i]=0;
        }
    }
    for(int i=1;i<=m;i++)
    {
        long long x,y,l;
        cin>>x>>y>>l;
        dis[x][y]=min(l,dis[x][y]);
        dis[y][x]=min(l,dis[y][x]);
    }
    for(int i=1;i<=n;i++)//floyd精髓-*从j号顶点到k号顶点只经过前i号点的最短路程*
        for(int j=1;j<=n;j++)//
            for(int k=1;k<=n;k++)//
                if(dis[j][i]+dis[i][k]<dis[j][k])//
                    dis[j][k]=dis[j][i]+dis[i][k];//
    for(int i=1;i<=n;i++)
    {
        int ans=0;
        for(int j=1;j<=n;j++)
            if(dis[i][j]!=INT_MAX) ans=(ans+dis[i][j])%mod;
        cout<<ans<<endl;
    }
}
```

【T8】P5318 【深基18.例3】查找文献

P5318 【深基18.例3】查找文献

提交答案 加入题单

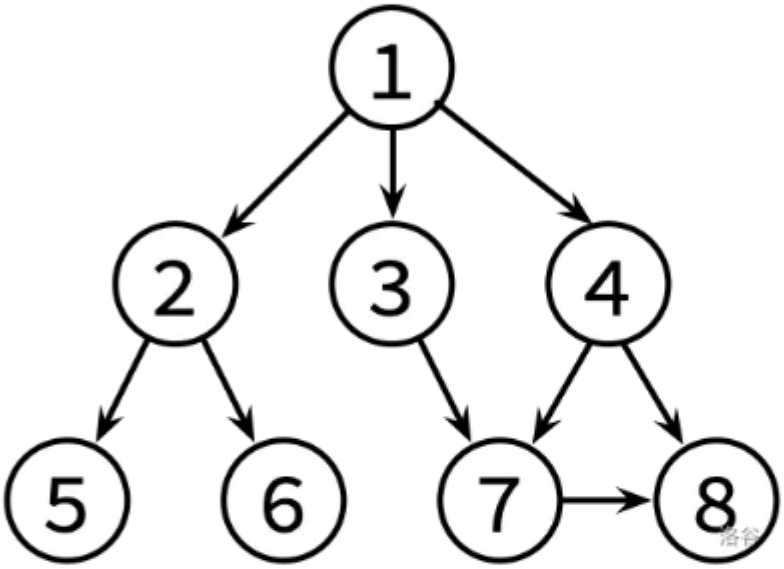
题目描述

复制Markdown 展开

小K 喜欢翻看洛谷博客获取知识。每篇文章可能会有若干个（也有可能没有）参考文献的链接指向别的博客文章。小K 求知欲旺盛，如果他看了某篇文章，那么他一定会去看这篇文章的参考文献（如果他之前已经看过这篇参考文献的话就不用再看它了）。

假设洛谷博客里面一共有 $n(n \leq 10^5)$ 篇文章（编号为 1 到 n）以及 $m(m \leq 10^6)$ 条参考文献引用关系。目前小 K 已经打开了编号为 1 的一篇文章，请帮助小 K 设计一种方法，使小 K 可以不重复、不遗漏的看完所有他能看到的文章。

这边是已经整理好的参考文献关系图，其中，文献 X → Y 表示文章 X 有参考文献 Y。不保证编号为 1 的文章没有被其他文章引用。



请对这个图分别进行 DFS 和 BFS，并输出遍历结果。如果有很多篇文章可以参阅，请先看编号较小的那篇(因此你可能需要先排序)。

输入格式

共 $m+1$ 行，第 1 行为 2 个数，n 和 m，分别表示一共有 $n(n \leq 10^5)$ 篇文章（编号为 1 到 n）以及 $m(m \leq 10^6)$ 条参考文献引用关系。

接下来 m 行，每行有两个整数 X, Y 表示文章 X 有参考文献 Y。

输出格式

共 2 行。 第一行为 DFS 遍历结果，第二行为 BFS 遍历结果。

输入输出样例

输入 #1 复制

```
8 9
1 2
1 3
1 4
2 5
2 6
3 7
4 7
4 8
```

输出 #1 复制

```
1 2 5 6 3 7 8 4
1 2 3 4 5 6 7 8
```

```
7 8
```

```cpp
#include<bits/stdc++.h>
using namespace std;
const int N=1e5+5;
struct edge
{
    int u;
    int v;
};
vector <int> mp[N];//存具体信息的二维vector --这样写有点别扭哇
vector <edge> edg;//存边的结构体            --
bool vis_dfs[N],vis_bfs[N];
bool cmp(edge x, edge y)
{
    if(x.v==y.v) return x.u<y.u;//v相同按u排
    else return x.v<y.v;//否则按v从大到小排序
}
void dfs(int x)
{
    vis_dfs[x]=true;
    cout<<x<<' ';
    for(int i=0;i<mp[x].size();i++)
    {
        int next_point=edg[mp[x][i]].v;
        if(!vis_dfs[next_point]) dfs(next_point);
    }
}
void bfs(int x)
{
    queue<int> q ;
    q.push(x);
    cout<<x<<' ';
    vis_bfs[x]=true;
    while(!q.empty())
    {
        int top=q.front();
        for(int i=0;i<mp[top].size();i++)
        {
            int point=edg[mp[top][i]].v;
            if(!vis_bfs[point])
            {
                q.push(point);
                cout<<point<<' ';
                vis_bfs[point]=true;
            }
        }
        q.pop();
    }
}
int main()
{
    int n,m;
```

```
    cin>>n>>m;
    for(int i=0;i<m;i++)
    {
        int u,v;
        cin>>u>>v;
        edg.push_back((edge){u,v});//初始化存边的s数组
    }
    sort(edg.begin(),edg.end(),cmp);
    for(int i=0;i<m;i++) mp[edg[i].u].push_back(i);//初始化mp
    dfs(1);          //在mp[edg[i].u]（也就是i号边的起点edg[i].u连接的边的数组）中存入i
号边
    cout<<endl;
    bfs(1);
}
```

感觉这样写比较好（图的遍历和存储上）

```
#include<bits/stdc++.h>
using namespace std;
int v[100005];
vector<int> e[100005];
void dfs(int cur)
{
    v[x]=true;*
    cout<<cur<<' ';
    for(int i=0;i<e[cur].size();i++)
    {
        int search=e[cur][i];
        if(!v[search])
        {
            v[search]=1;
            dfs(search);
        }

     }
}
void bfs(int cur)---这个bfs写的有点别扭的说.....
{
    int q[100005],f=0,r=0;
    q[r++]=cur;
    v[cur]=1;
    while(f!=r)
    {
        int tmp=q[f++];//出队
        cout<<tmp<<' ';
        for(int i=0;i<e[tmp].size();i++)
        {
            int search=e[tmp][i];
            if(!v[search])
            {
                q[r++]=search;
                v[search]=1;
            }
        }
    }
}
int main()
{
```

```
    int i,j,n,m,x,y;
    cin>>n>>m;
    for(i=1;i<=m;i++)
    {
        cin>>x>>y;
        e[x].push_back(y);
    }
    for(i=1;i<=n;i++)
    sort(e[i].begin(),e[i].end());
    dfs(1);
    memset(v,0,sizeof(v));
    cout<<endl;
    bfs(1);
    return 0;
}
```

【T9】P4779 【模板】单源最短路径（标准版）

题目描述

给定一个 n 个点，m 条有向边的带非负权图，请你计算从 s 出发，到每个点的距离。

数据保证你能从 s 出发到任意点。

输入格式

第一行为三个正整数 n, m, s。 第二行起 m 行，每行三个非负整数 u_i, v_i, w_i，表示从 u_i 到 v_i 有一条权值为 w_i 的有向边。

输出格式

输出一行 n 个空格分隔的非负整数，表示 s 到每个点的距离。

输入输出样例

输入 #1 复制

```
4 6 1
1 2 2
2 3 2
2 4 1
1 3 5
3 4 3
1 4 4
```

输出 #1 复制

```
0 2 4 3
```

说明/提示

样例解释请参考 数据随机的模板题。

$1 \le n \le 10^5$;

$1 \le m \le 2 \times 10^5$;

$s = 1$;

$1 \le u_i, v_i \le n$;

$0 \le w_i \le 10^9$,

$0 \le \sum w_i \le 10^9$。

```cpp
#include<bits/stdc++.h>
using namespace std;
const int maxn=1e5+5;
const int INF=1e9+5;
int dis[maxn];
bool vis[maxn];
int n,m,w;
struct node1
{
    int v;
```

```
    int s;
};
struct node2
{
    int to;
    int len;
};
bool operator<(node1 x,node1 y)
{
    return x.s>y.s;
}
vector<node2> mp[maxn];
priority_queue<node1> s;
void dijkstra()
{
    for(int i=1;i<=n;i++)
        dis[i]=INF;
    dis[w]=0;
    node1 p;
    p.v=w,p.s=0;
    s.push(p);
    while(!s.empty())
    {
        int top=s.top().v;
        s.pop();
        if(vis[top]) continue;
        for(int i=0;i<mp[top].size();i++)
        {
            int dv=mp[top][i].to;
            int ds=mp[top][i].len;
            if(dis[dv]>dis[top]+ds)
                dis[dv]=dis[top]+ds;
            node1 temp;
            temp.s=dis[dv],temp.v=dv;
            s.push(temp);
        }
        vis[top]=1;
    }
}
int main()
{
    int x,y,z;
    cin>>n>>m>>w;
    for(int i=1;i<=m;i++)
    {
        cin>>x>>y>>z;
        node2 p;
        p.to=y;p.len=z;
        mp[x].push_back(p);
    }
    dijkstra();
    for(int i=1;i<=n;i++)
        cout<<dis[i]<<' ';
}
```

【T10】P3905 道路重建

P3905 道路重建

提交答案 加入题单

题目描述

复制Markdown 展开

从前，在一个王国中，在n个城市间有m条道路连接，而且任意两个城市之间至多有一条道路直接相连。在经过一次严重的战争之后，有d条道路被破坏了。国王想要修复国家的道路系统，现在有两个重要城市A和B之间的交通中断，国王希望尽快的恢复两个城市之间的连接。你的任务就是修复一些道路使A与B之间的连接恢复，并要求修复的道路长度最小。

输入格式

输入文件第一行为一个整数n（$2 < n \leq 100$）,表示城市的个数。这些城市编号从1到n。

第二行为一个整数m（$n-1 \leq m \leq \frac{1}{2}n(n-1)$），表示道路的数目。

接下来的m行，每行3个整数i,j,k（$1 \leq i,j \leq n, i \neq j, 0 < k \leq 100$），表示城市$i$与$j$之间有一条长为$k$的道路相连。

接下来一行为一个整数d（$1 \leq d \leq m$），表示战后被破坏的道路的数目。在接下来的d行中，每行两个整数i和j，表示城市i与j之间直接相连的道路被破坏。

最后一行为两个整数A和B，代表需要恢复交通的两个重要城市。

输出格式

输出文件仅一个整数，表示恢复A与B间的交通需要修复的道路总长度的最小值。

输入输出样例

输入 #1 复制

```
3
2
1 2 1
2 3 2
1
1 2
1 3
```

输出 #1 复制

```
1
```

```cpp
#include<bits/stdc++.h>
using namespace std;
int ans[105][105],dis[105][105];//ans[x][y]->xy修路最小值，dis[x][y]->xy距离
void floyd(int n)
{
    for(int k=1;k<=n;k++)
        for(int i=1;i<=n;i++)
            for(int j=1;j<=n;j++)
```

```
                ans[i][j]=min(ans[i][j],ans[i][k]+ans[k][j]);
}
int main()
{
    memset(ans,0x3f3f3f3f,sizeof(ans));
    int n,m,d;
    cin>>n>>m;
    for(int i=1;i<=m;i++)//建边
    {
        int u,v;
        cin>>u>>v>>dis[u][v];
        dis[v][u]=dis[u][v];//注意左值右值！！！
        ans[u][v]=ans[v][u]=0;//有路的话就不需要再修
    }
    cin>>d;
    for(int i=1;i<=d;i++)//删边
    {
        int u,v;
        cin>>u>>v;
        ans[u][v]=ans[v][u]=dis[u][v];//修的路等于两点距离
    }
    floyd(n);
    int st,ed;
    cin>>st>>ed;
    cout<<ans[st][ed];
}
```

【T11】P1330 封锁阳光大学

题目描述

复制Markdown 展开

曹是一只爱刷街的老曹，暑假期间，他每天都欢快地在阳光大学的校园里刷街。河蟹看到欢快的曹，感到不爽。河蟹决定封锁阳光大学，不让曹刷街。

阳光大学的校园是一张由 n 个点构成的无向图，n 个点之间由 m 条道路连接。每只河蟹可以对一个点进行封锁，当某个点被封锁后，与这个点相连的道路就被封锁了，曹就无法在这些道路上刷街了。非常悲剧的一点是，河蟹是一种不和谐的生物，当两只河蟹封锁了相邻的两个点时，他们会发生冲突。

询问：最少需要多少只河蟹，可以封锁所有道路并且不发生冲突。

输入格式

第一行两个正整数，表示节点数和边数。 接下来 m 行，每行两个整数 u, v，表示点 u 到点 v 之间有道路相连。

输出格式

仅一行如果河蟹无法封锁所有道路，则输出 `Impossible`，否则输出一个整数，表示最少需要多少只河蟹。

输入输出样例

输入 #1 复制

```
3 3
1 2
1 3
2 3
```

输出 #1 复制

```
Impossible
```

输入 #2 复制

```
3 2
1 2
2 3
```

输出 #2 复制

```
1
```

说明/提示

【数据规模】
对于 100% 的数据，$1 \leq n \leq 10^4$，$1 \leq m \leq 10^5$，保证没有重边。

```cpp
#include<bits/stdc++.h>
using namespace std;
const int N=1e5+5;
int n,m;
vector<int>mp[N];
int color[N];//每一个点的染色
bool vis[N];//是否遍历过
int sum[2];//两种染色各自的的操作次数
bool dfs(int now,int colour)//对每个独立的图进行dfs
{
    if(vis[now])//如果已经被染过色
    {
```

```
        if(color[now]==colour)return true;//如果仍是原来的颜色，就没啥问题
        return false;//如果不是原来的颜色说明能够产生冲突
    }
    vis[now]=true;//记录走过
    sum[color[now]=colour]++;//这一种颜色的个数+1，这个点的颜色也记录下来
    bool flag=true;//是否可行
    for(int i=0;i<mp[now].size();i++)//对边进行遍历
    {
        int next=mp[now][i];
        flag=(flag&&dfs(next,1-colour));//是否可以继续染色
    }
    return flag;//返回染色是否完成
}
int main()
{
    int flag=1,ans=0;
    cin>>n>>m;
    memset(vis,false,sizeof(vis));
    memset(color,0,sizeof(color));
    for(int i=1;i<=m;i++)
    {
        int u,v;
        cin>>u>>v;
        mp[u].push_back(v);
        mp[v].push_back(u);
    }                          //以上都是初始化
    for(int i=1;i<=n;i++)
    {
        if(vis[i])continue;//如果此点已被包含为一个已经被遍历过的子图，则不需重复遍历
        sum[0]=sum[1]=0;//初始化
        if(!dfs(i,0))//如果不能染色
        {
            flag=0;
            break;
        }
        ans+=min(sum[0],sum[1]);//ans加上其中小的那个
    }
    if(!flag)cout<<"Impossible";
    else cout<<ans;
}
/*这一题是一个好题。如果知道思路了，便会非常简单。但是如果不知道思路，却比较的难想出来。

我们来分析一下题目。首先，肯定要明确一点，那就是这个图是不一定联通的。于是，我们就可以将整张图切分成许多分开的连同子图来处理。然而最重要的事情是：如何处理一个连通图？

乍看下去，似乎无从下手，因为方案好像有很多种，根本就枚举不完。但是，关键要注意到题目中重要的两个条件，我们把它抽象成这两个要素：

##①每一条边所连接的点中，至少要有一个被选中。②每一条边所连接的两个点，不能被同时选中。由此，可以推断出：

#每一条边都有且仅有一个被它所连接的点被选中。

又因为我们要处理的是一个连通图。所以，对于这一个图的点的选法，可以考虑到相邻的点染成不同的颜色。

#于是，对于一个连通图，要不就只有两种选法（因为可以全部选染成一种色的，也可以全部选染成另一
```

种色的），要不就是impossible!

所以，**我们只需要找到每一个子连通图，对它进行黑白染色，然后取两种染色中的最小值，然后最后汇总，就可以了。**

另外，要判断impossible，只需要加一个used数组，记录已经遍历了哪些点。如果重复遍历一个点，且与上一次的颜色不同，则必然是impossible的。*/

【T12】P8604 [蓝桥杯 2013 国 C] 危险系数

题目背景

复制Markdown 展开

抗日战争时期，冀中平原的地道战曾发挥重要作用。

题目描述

地道的多个站点间有通道连接，形成了庞大的网络。但也有隐患，当敌人发现了某个站点后，其它站点间可能因此会失去联系。

我们来定义一个危险系数 $DF(x,y)$：

对于两个站点 x 和 $y(x \neq y)$，如果能找到一个站点 z，当 z 被敌人破坏后，x 和 y 不连通，那么我们称 z 为关于 x,y 的关键点。相应的，对于任意一对站点 x 和 y，危险系数 $DF(x,y)$ 就表示为这两点之间的关键点个数。

本题的任务是：已知网络结构，求两站点之间的危险系数。

输入格式

输入数据第一行包含 2 个整数 $n(2 \leq n \leq 1000)$，$m(0 \leq m \leq 2000)$，分别代表站点数，通道数。

接下来 m 行，每行两个整数 $u,v(1 \leq u,v \leq n, u \neq v)$ 代表一条通道。

最后 1 行，两个数 u,v，代表询问两点之间的危险系数 $DF(u,v)$。

输出格式

一个整数，如果询问的两点不连通则输出 -1。

输入输出样例

输入 #1 复制

```
7 6
1 3
2 3
3 4
3 5
4 5
5 6
1 6
```

输出 #1 复制

```
2
```

说明/提示

时限 1 秒, 64M。蓝桥杯 2013 年第四届国赛

```cpp
#include<bits/stdc++.h>
using namespace std;
const int N=1005;
int u,v,st,ed,n,m;
int cnt=0;
vector<int>mp[N];//用vector存图
```

```cpp
bool vis[N];//是否来过
int total[N];//这个点被用了几次
void dfs(int now)
{
    if(now==ed)
    {
        cnt++;
        for(int i=1;i<=n;i++)//统计用到了哪些点
        {
            if(vis[i])
             total[i]++;
        }
        return;
    }
    for(int i=0;i<mp[now].size();i++)//注意遍历方式
    {
        int next=mp[now][i];//
        if(!vis[next])
        {
            vis[next]=true;
            dfs(next);
            vis[next]=false;
        }
    }
}
int main()
{
    cin>>n>>m;
    for(int i=1;i<=m;i++)//注意要从一开始，根据题意最前面的点编号为1
    {
        cin>>u>>v;
        mp[u].push_back(v);//
        mp[v].push_back(u);//初始化图
    }
    cin>>st>>ed;
    vis[st]=true;//先把起点标记为走过
    dfs(st);
    int ans=0;
    for(int i=1;i<=n;i++)
    {
        if(total[i]==cnt)//代表从st到ed不管怎么走都会经过他
            ans++;
    }
    cout<<ans-2;//最后要减去起点和终点
}
```

【T13】P3916 图的遍历

P3916 图的遍历

提交答案 加入题单

题目描述

复制Markdown 展开

给出 N 个点，M 条边的有向图，对于每个点 v，求 $A(v)$ 表示从点 v 出发，能到达的编号最大的点。

输入格式

第 1 行 2 个整数 N, M，表示点数和边数。

接下来 M 行，每行 2 个整数 U_i, V_i，表示边 (U_i, V_i)。点用 $1, 2, \ldots, N$ 编号。

输出格式

一行 N 个整数 $A(1), A(2), \ldots, A(N)$。

输入输出样例

输入 #1 复制

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

输出 #1 复制

```
4 4 3 4
```

说明/提示

- 对于 60% 的数据，$1 \leq N, M \leq 10^3$。
- 对于 100% 的数据，$1 \leq N, M \leq 10^5$。

```cpp
#include<bits/stdc++.h>
using namespace std;
const int N=1e5+10;
int u,v,m,n;
vector<int>mp[N];
int biggest[N];//也可以认为存的是每个节点可达的最大节点
void dfs(int x,int d)//从x开始，尽头是d，即可以从x到达d
{
    if(biggest[x]) return;
    biggest[x]=d;
    for(int i=0;i<mp[x].size();i++)
        dfs(mp[x][i],d);
}
int main()
{
    cin>>n>>m;
    for(int i=1;i<=m;i++)
    {
```

```
        cin>>u>>v;
        mp[v].push_back(u);//反向建边，优点是：递归由大向小，减少搜索次数
    }                      //反向建立边的作用相当于让之前的路径有可以反悔的余地。
    for(int i=n;i>0;i--)
        dfs(i,i);
    for(int i=1;i<=n;i++)
        cout<<biggest[i]<<' ';
    cout<<endl;
}
//按题目来每次考虑每个点可以到达点编号最大的点，不如考虑较大的点可以反向到达哪些点

//循环从N到1，则每个点i能访问到的结点的biggest值都是i

//每个点访问一次，这个biggest值就是最优的，因为之后如果再访问到这个结点那么答案肯定没当前大了
```