一、拓扑排序

例题：Genealogical tree

给一个有向无环图图，输出任一拓扑排序

样例输入

5

#5 个点

0 #1 号点没出边

4 5 1 0 #2 号点有边连到 4,5, 1

1 0

5 3 0

3 0

样例输出

2 4 5 3 1

```
import queue
n = int(input())
G = [[] for i in range(n+1)]
inDegree = [0] * (n+1) #G 是邻接表，inDegree[i]是 i 的入度
for i in range(1,n+1):
lst = list(map(int,input().split()))
G[i] = lst[:-1]
q = queue.Queue()
for i in range(1,n+1):
for v in G[i]:
inDegree[v] += 1
for i in range(1,n+1):
if inDegree[i] == 0:
q.put(i)
seq = []
```



```
while not q.empty():
k = q.get()
seq.append(k)
for v in G[k]:
inDegree[v] -= 1 #删除边(k,v)后将 v 入度减 1
if inDegree[v] == 0:
q.put(v)
if len(seq) != n: #如果拓扑序列长度少于点数，则说明有环
```

```
print("error")
else:
for x in seq:
print(x,end = " ")
print("")
```

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

bool TopSort(vector<vector<int> > &G, int n, vector<int> &inDegree) {
    /*
    *   param
    *   G：邻接表
    *   n：顶点数
    *   InDegree：   记录顶点的入度
    */
    int num = 0;                //记录加入拓扑排序的顶点数
    queue<int> q;
    for (int i = 0; i < n; i++)
        if (inDegree[i] == 0)
            q.push(i);          //将所有入度为 0 的顶点入队
    while (!q.empty()) {
        int u = q.front();      //取队首顶点 u
        cout << u << " ";
        q.pop();
        for (int i = 0; i < G[u].size(); i++) {
            int v = G[u][i];        //u 的后继节点
            inDegree[v]--;          //v 的入度减 1
            if (inDegree[v] == 0)       //顶点 v 的入度减为 0 则入队
                q.push(v);
        }
        G[u].clear();           //清空顶点 u 的所有出边
        num++;
    }
    if (num == n)               //加入拓扑序列的顶点数为 n，说明拓扑排序成功，否则，
失败
```

```
            return true;
        else
            return false;
}

int main() {
        int n, m;
        cout << "请输入顶点数和边数:";
        cin >> n >> m;
        vector<vector<int> > G(n);
        for (int i = 0; i < m; i++) {
            int x, y;
            cout << "请输入第" << i+1 << "条边的顶点:";
            cin >> x >> y;
            G[x].push_back(y);
        }
        cout << "拓扑排序为:";
        vector<int> inDegree(n);
        for ( int i = 0 ; i< n ; i++ )
        {
            for ( int j = 0 ; j < G[i].size() ; j ++ )
            {
                inDegree[G[i][j]]++;
            }
         }
//      for (auto x : G) {
//          for (auto y : x)
//              inDegree[y]++;
//      }
        bool res = TopSort(G,n,inDegree);

        return 0;

}
```

二、最小生成树

（一）、prim

POJ 1258 Agri-Net 最小生成树模版题

输入图的邻接矩阵，求最小生成树的总权值(多组数据)

输入样例:

4

0 4 9 21

4 0 8 17

9 8 0 16

21 17 16 0

输出样例:

28

Prim + 堆 完成 POJ1258 Agri-Net

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <queue>
using namespace std;
const int
INFINITE = 1 << 30;
struct Edge
{
int v; //边端点，另一端点已知
int w; //边权值，也用来表示 v 到在建最小生成树的距离
Edge(int v_ = 0, int w_ = INFINITE):v(v_),w(w_) { }
bool operator <(const Edge & e) const
{
return w > e.w; //在队列里，边权值越小越优先
}
};
vector< vector <Edge> > G(110); //图的邻接表北京大学信息学院 郭炜
int HeapPrim(const vector<vector<Edge> > & G, int n)
//G 是邻接表,n 是顶点数目，返回值是最小生成树权值和
{
int i,j,k;
Edge xDist(0,0);
priority_queue<Edge> pq; //存放顶点及其到在建生成树的距离
vector<int> vDist(n); //各顶点到已经建好的那部分树的距离
vector<int> vUsed(n);//标记顶点是否已经被加入最小生成树
int nDoneNum = 0; //已经被加入最小生成树的顶点数目
for( i = 0;i < n;i ++ ) {
```

```
vUsed[i] = 0;
vDist[i] = INFINITE;
}
nDoneNum = 0;
int nTotalW = 0; //最小生成树总权值
pq.push(Edge(0,0)); //开始只有顶点 0，它到最小生成树距离 0while( nDoneNum < n
&& !pq.empty() ) {
```

```
do {//每次从队列里面拿离在建生成树最近的点
xDist = pq.top();
pq.pop();
} while( vUsed[xDist.v] == 1 && ! pq.empty());
if( vUsed[xDist.v] == 0 ) {
nTotalW += xDist.w; vUsed[xDist.v] = 1;
nDoneNum ++;
for( i = 0;i < G[xDist.v].size();i ++ ) {
//更新新加入点的邻点
int k = G[xDist.v][i].v;
if( vUsed[k] == 0) {
int w = G[xDist.v][i].w ;
if( vDist[k] > w ) {
vDist[k] = w; pq.push(Edge(k,w));
}
}
}
}
}
if( nDoneNum < n )
return -1; //图不连通
return nTotalW;
}
```

```
int main()
{
int N;
while(cin >> N) {
for( int i = 0;i < N; ++i)
G[i].clear();
for( int i = 0; i < N; ++i)
```

```
for( int j = 0; j < N; ++j) {
int w;
cin >> w;
G[i].push_back(Edge(j,w));
}
cout << HeapPrim(G,N) << endl;
}
}
```

（二）、Kruskal

Kruskal 算法完成 POJ1258 Agri-Net

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
struct Edge
{
int s,e,w; //起点，终点，权值
Edge(int ss,int ee,int ww):s(ss),e(ee),w(ww) { }
Edge() { }
bool operator < (const Edge & e1) const {
return w < e1.w;
}
};
vector <Edge> edges;
vector <int> parent;北京大学信息学院 郭炜
int GetRoot(int a)
{
if( parent[a] == a)
return a;
parent[a] = GetRoot(parent[a]);
return parent[a];
}
void Merge(int a,int b)
{
int p1 = GetRoot(a);
int p2 = GetRoot(b);
if( p1 == p2)
return;
```

```
parent[p2] = p1;
}int main() {
```

```
int N;
while(cin >> N) {
parent.clear();
edges.clear();
for( int i = 0;i < N; ++i) parent.push_back(i);
for( int i = 0; i < N; ++i)
for( int j = 0; j < N; ++j) { int w;
cin >> w;
edges.push_back(Edge(i,j,w));
}
sort(edges.begin(),edges.end()); //排序复杂度 O(ElogE）
int done = 0;
int totalLen = 0;
for( int i = 0;i < edges.size(); ++i) {
if( GetRoot(edges[i].s) != GetRoot(edges[i].e)) {
Merge(edges[i].s,edges[i].e);
++done;
totalLen += edges[i].w;
}
if( done == N – 1) break;
}
cout << totalLen << endl;
}
}
```

三、最短路径

（一）Dijkstra

POJ3159 Candies

有 N 个孩子（N<=3000)分糖果。

有 M 个关系(M<=150,000)。每个关系形如：

A B C (A,B,C 是孩子编号)

表示 A 比 B 少的糖果数目，不能超过 C

求第 N 个学生最多比第 1 个学生能多分几个糖果

思路：30000 点，150000 边的稀疏图求单源最短路

读入 “A B C”，就添加 A->B 的有向边，权值为 C

然后求 1 到 N 的最短路

用 prioirty_queue 实现 dijkstra + 堆的 POJ 3159 Candies

北京大学信息学院 郭炜

```
#include <cstdio>
#include <iostream>
#include <vector>
#include <queue>
#include <cstring>
using namespace std;
struct CNode {
int k; //有向边的终点
int w; //边权值，或当前 k 到源点的距离
};
bool operator < ( const CNode & d1, const CNode & d2 )
{ return d1.w > d2.w; } //priority_queue 总是将最大的元素出列
priority_queue<CNode> pq;
bool bUsed[30010]={0}; // bUsed[i]为 true 表示源到 i 的最短路已经求出
vector<vector<CNode> > v; //v 是整个图的邻接表
const unsigned int INFINITE = 100000000;
```



```
int main()
```

北京大学信息学院 郭炜

```
{
int N,M,a,b,c;
int i,j,k;
CNode p;
scanf("%d%d", & N, & M );
v.clear();
v.resize(N+1);
memset( bUsed,0,sizeof(bUsed));
for( i = 1;i <= M; i ++ ) {
scanf("%d%d%d", & a, & b, & c);
p.k = b;
p.w = c;
v[a].push_back( p);
}
p.k = 1; //源点是 1 号点
p.w = 0; //1 号点到自己的距离是 0
```

```
pq.push (p);
```



```
while( !pq.empty ()) {
p = pq.top ();
pq.pop();
if( bUsed[p.k]) //已经求出了最短路
continue;
bUsed[p.k] = true;
if( p.k == N ) //因只要求 1-N 的最短路，所以要 break
break;
for( i = 0, j = v[p.k].size(); i < j; i ++ ) {
CNode q; q.k = v[p.k][i].k;
if( bUsed[q.k] ) continue;
q.w = p.w + v[p.k][i].w ;
pq.push (q); //队列里面已经有 q.k 点也没关系
}
}
printf("%d", p.w ) ;
return 0;
}
```

（二）、Bellman

POJ3259 Wormholes

Bellman

要求判断任意两点都能仅通过正边就互相可达的有向图(图中有重边）中是否存在负权环

Sample Input

2
3 3 1
1 2 2
1 3 4
2 3 1
3 1 3
3 2 1
1 2 3
2 3 4
3 1 8

```
#include <iostream>
```

```
#include <vector>
using namespace std;
int F,N,M,W;
const int INF = 1 << 30;
struct Edge {
int s,e,w;
Edge(int ss,int ee,int ww):s(ss),e(ee),w(ww) { }
Edge() { }
};
vector<Edge> edges; //所有的边
int dist[1000];
```

```
int Bellman_ford(int v) {
```

```
for( int i = 1; i <= N; ++i)
dist[i] = INF;
dist[v] = 0;
for( int k = 1; k < N; ++k) { //经过不超过 k 条边
for( int i = 0;i < edges.size(); ++i) {
int s = edges[i].s;
int e = edges[i].e;
if(dist[s] != INF &&
dist[s] + edges[i].w < dist[e])
dist[e] = dist[s] + edges[i].w;
}
}
for( int i = 0;i < edges.size(); ++ i) {
int s = edges[i].s;
int e = edges[i].e;
if(dist[s] != INF &&
dist[s] + edges[i].w < dist[e])
return true;
}
return false;
}
```

```
int main() {
```

```
cin >> F;
```

```
while( F--) {
edges.clear();
cin >> N >> M >> W;
for( int i = 0; i < M; ++ i) {
int s,e,t;
cin >> s >> e >> t;
edges.push_back(Edge(s,e,t)); //双向边等于两条边
edges.push_back(Edge(e,s,t));
}
for( int i = 0;i < W; ++i) {
int s,e,t;
cin >> s >> e >> t;
edges.push_back(Edge(s,e,-t));
}
if( Bellman_ford(1))//从 1 可达所有点
cout << "YES" <<endl;
else cout << "NO" <<endl;
}
}
```

61

北京大学信息学院 郭炜

```
for( int k = 1; k < N; ++k) { //经过不超过 k 条边
for( int i = 0;i < edges.size(); ++i) {
int s = edges[i].s;
int e = edges[i].e;
if( dist[s] + edges[i].w < dist[e])
dist[e] = dist[s] + edges[i].w;
}
}
```

会导致在一次内层循环中，更新了某个 dist[x]后，以后又用 dist[x]去更新 dist[y]，这样 dist[y]就是经过最多不超过 k+1 条边的情况了

出现这种情况没有关系，因为整个 for(int k = 1; k < N; ++k) 循环的目的是要确保,对任意点 u,如果从源 s 到 u 的最短路是经过不超过 n-1 条边的,则这条最短路不会被忽略。至于计算过程中对某些点 v 计算出了从 s->v 的经过超过 N-1 条边的最短路的情况，也不影响结果正确性。若是从 s->v 的经过超过 N-1 条边的结果比经过最多 N-1 条边的结果更小，那一定就有负权回路。有负权回路的情况下，再多做任意多次循环，每次都会发现到有些点的最短路变得更短了。

（三）、SPFA

维护一个队列，里面存放所有需要进行迭代的点。初始时队列中只有一个

源点 S。用一个布尔数组记录每个点是否处在队列中。

每次迭代，取出队头的点 v，依次枚举从 v 出发的边 v->u，若 Dist[v]+len(v->u) 小于 Dist[u]，则改进 Dist[u]（可同时将 u 前驱记为 v）。此时由于 S 到 u 的最短距离变小了，有可能 u 可以改进其它的点，所以若 u 不在队列中，就将它放入队尾。这样一直迭代下去直到队列变空，也就是 S 到所有节点的最短距离都确定下来，结束算法。**若一个点最短路被改进的次数达到 n ，则有负权环(原因同 B-F 算法)** 。可以用 spfa 算法判断图有无负权环

在平均情况下，SPFA 算法的期望时间复杂度为 O(E)。

POJ3259 Wormholes 判断有没有负权环 spfa

```
//by guo wei
#include <iostream>
#include <vector>
#include <queue>
#include <cstring>
using namespace std;
int F,N,M,W;
const int INF = 1 << 30;
struct Edge {
int e,w;
Edge(int ee,int ww):e(ee),w(ww) { }
Edge() { }
};
vector<Edge> G[1000]; //整个有向图
int updateTimes[1000]; //最短路的改进次数
int dist[1000]; //dist[i]是源到 i 的目前最短路长度
```



```
int Spfa(int v) {
```



```
for( int i = 1; i <= N; ++i)
dist[i] = INF;
dist[v] = 0;
queue<int> que; que.push(v);
memset(updateTimes ,0,sizeof(updateTimes));
while( !que.empty()) {
int s = que.front();
que.pop();
for( int i = 0;i < G[s].size(); ++i) {
int e = G[s][i].e;
if(dist[s] != INF &&
```

```
dist[e] > dist[s] + G[s][i].w ) {
dist[e] = dist[s] + G[s][i].w;
que.push(e); //没判队列里是否已经有 e,可能会慢一些
++updateTimes[e];
if( updateTimes[e] >= N) return true;
}
}
}
return false;
}
```


```
int main(){
cin >> F;
while( F--) {
cin >> N >> M >> W;
for( int i = 1; i <1000; ++i)
G[i].clear();
int s,e,t;
for( int i = 0; i < M; ++ i) {
cin >> s >> e >> t;
G[s].push_back(Edge(e,t));
G[e].push_back(Edge(s,t));
}
for( int i = 0;i < W; ++i) {
cin >> s >> e >> t;
G[s].push_back(Edge(e,-t));
}
if( Spfa(1))
cout << "YES" <<endl;
else
cout << "NO" <<endl;
}
}
```

（四）、floyed

Floyed

```
for( int i = 1 ;i <= vtxnum; ++i )
for( int j = 1; j <= vtxnum; ++j) {
dist[i][j] = cost[i][j]; // cost 是边权值, dist 是两点间最短距离
```

```
if( dist[i][j] < INFINITE) //i 到 j 有边
path[i,j] = [i]+[j]; //path 是路径
}
for( k = 1; k <= vtxnum; ++k) //每次求中间点标号不超过 k 的 i 到 j 最短路
for( int i = 1; i <= vtxnum; ++i)
for(int j = 1; j <= vtxnum ; ++j)
if( dist[i][k] + dist[k][j] < dist[i][j]) {
dist[i][j] = dist[i][k]+dist[k][j];
path[i,j] = path[i,k]+path[k,j];
}
```

四、求连通分量 Tarjan

无重边连通无向图求割点和桥

Input: （11 点 13 边）

11 13
1 2
1 4
1 5
1 6
2 11
2 3
4 3
4 9
5 8
5 7
6 7
7 10
11 3

output:

1
4
5
7
5,8
4,9
7,10

```
//无重边连通无向图求割点和桥的程序
#include <iostream>
```

```
#include <vector>
using namespace std;
#define MyMax 200
typedef vector<int> Edge;
vector<Edge> G(MyMax);
bool Visited[MyMax] ;
int dfn[MyMax] ;
int low[MyMax] ;
int Father[MyMax]; //DFS 树中每个点的父节点
bool bIsCutVetext[MyMax]; //每个点是不是割点
int nTime; //Dfs 时间戳
int n,m; //n 是点数，m 是边数 void Tarjan(int u, int father) //father 是 u 的父节点
{
Father[u] = father;
int i,j,k;
low[u] = dfn[u] = nTime ++;
for( i = 0;i < G[u].size() ;i ++ ) {
int v = G[u][i];
if( ! dfn[v]) {
Tarjan(v,u);
low[u] = min(low[u],low[v]);
}
else if( father != v ) //连到父节点的回边不考虑，否则求不出桥
low[u] = min(low[u],dfn[v]);
}
}void Count()
{ //计算割点和桥
int i,nRootSons = 0;
Tarjan(1,0);
for( i = 2;i <= n;i ++ ) {
int v = Father[i];
if( v == 1 )
nRootSons ++; //DFS 树中根节点有几个子树
else if( dfn[v] <= low[i])
bIsCutVetext[v] = true;
}
if( nRootSons > 1)
bIsCutVetext[1] = true;
```

```
for( i = 1;i <= n;i ++ )
if( bIsCutVetext[i] )
cout << i << endl;
for( i = 1;i <= n;i ++) {
int v = Father[i];
if(v >0 && dfn[v] < low[i])
cout << v << "," << i <<endl;
}
}int main()
{
int u,v;
int i;
nTime = 1;
cin >> n >> m ; //n 是点数，m 是边数
for( i = 1;i <= m;i ++ ) {
cin >> u >> v; //点编号从 1 开始
G[v].push_back(u);
G[u].push_back(v);
}
memset( dfn,0,sizeof(dfn));
memset( Father,0,sizeof(Father));
memset( bIsCutVetext,0,sizeof(bIsCutVetext));
Count();
return 0;
}
```

五、CCF 图论

（一）、最小生成树

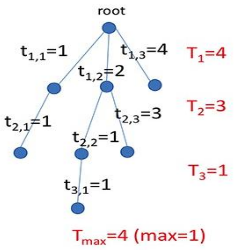
1、数据中心（最小生成树）

【知识背景】

在一个集中式网络中，存在一个根节点，需要长时间接收其余所有节点传输给它的反馈数据。

【题目描述】

存在一个 n 节点的网络图，编号从 1 到 n。该网络的传输是全双工的，所以是无向图。如果两节点 v_i, u_i 相连，表明 v_i, u_i 之间可以互相收发数据，边权是传输数据所需时间 t_i。**现在每个节点需要选择一条路径将数据发送到 $root$ 号节点。**希望求出一个最优的树结构传输图，使得完成这个任务所需要的时间最少。$root$ 节点只能接收数据，其余任何一个节点可以将数据传输给另外的一个节点，但是不能将数据传输给多个节点。所有节点可以接收多个不同节点的数据。



一个树结构传输图的传输时间为 $T_{\max}$，其中 $T_{\max} = \max(T_h)$，h 为接收点在树中的深度，$T_h = \max(t_{h,j})$，$t_{h,j}$ 表示 j 条不同的边，这 j 条边接收点的深度都为 h。

【输入格式】

从标准输入读入数据。

输入的第 1 行包含一个正整数 n，保证 $n \le 5 \times 10^4$。

输入的第 2 行包含一个正整数 m，保证 $m \le 10^5$。

输入的第 3 行包含一个正整数 $root$，保证 $root \le 5 \times 10^4$。

输入的第 4 行至第 $3+m$ 行包含 3 个正整数 v_i, u_i, t_i，保证 $v_i \le 5\times10^4$，$u_i \le 5\times10^4$，$t_i \le 10^6$，$u_i \ne v_i$。

问题描述：

【输出格式】

输出到标准输出。

输出仅有一行，包含一个正整数 ans，表示最优的树结构流水线所耗时 $T_{\max}$。

这道题题目描述不是很清楚，其实这道题是要求最小生成树中的最长边。

```
#include<bits/stdc++.h>
using namespace std;
struct Edge{//边的类
    int v1,v2,cost;
    Edge(int vv1,int vv2,int c):v1(vv1),v2(vv2),cost(c){}
    bool operator <(const Edge&e)const{//重载小于运算符
        return this->cost>e.cost;
    }
};
priority_queue<Edge>edges;
int father[50005];//并查集
int findFather(int x){//查找根结点并进行路径压缩
    if(father[x]==x)
        return x;
    int temp=findFather(father[x]);
    father[x]=temp;
    return temp;
}
int main(){
    int n,m,root,ans=0;
    scanf("%d%d%d",&n,&m,&root);
    iota(father,father+n+1,0);//初始化并查集
    while(m--){
```

```
        int a,b,c;
        scanf("%d%d%d",&a,&b,&c);
        edges.push(Edge(a,b,c));
    }
    while(!edges.empty()){
        Edge e=edges.top();
        edges.pop();
        int ua=findFather(e.v1),ub=findFather(e.v2);
        if(ua!=ub){//边的两个端点不属于同一个集合
            father[ua]=ub;
            ans=e.cost;//更新最长边
        }
    }
    printf("%d",ans);
    return 0;
}
```

2、地铁修建（最小生成树）

问题描述

A市有n个交通枢纽，其中1号和n号非常重要，为了加强运输能力，A市决定在1号到n号枢纽间修建一条地铁。

地铁由很多段隧道组成，每段隧道连接两个交通枢纽。经过勘探，有m段隧道作为候选，两个交通枢纽之间最多只有一条候选的隧道，没有隧道两端连接着同一个交通枢纽。

现在有n家隧道施工的公司，每段候选的隧道只能由一个公司施工，每家公司施工需要的天数一致。而每家公司最多只能修建一条候选隧道。所有公司同时开始施工。

作为项目负责人，你获得了候选隧道的信息，现在你可以按自己的想法选择一部分隧道进行施工，请问修建整条地铁最少需要多少天。

输入格式

输入的第一行包含两个整数n, m，用一个空格分隔，分别表示交通枢纽的数量和候选隧道的数量。

第2行到第m+1行，每行包含三个整数a, b, c，表示枢纽a和枢纽b之间可以修建一条隧道，需要的时间为c天。

输出格式

输出一个整数，修建整条地铁线路最少需要的天数。

本题实际上是一道求解最小生成树的最长边问题，可以使用生成算法 Kruskal 算法，只不过这道题不需要生成整棵最小生成树，只需要保证 1 号结点和 n 号结点属于同一个并查集，即 1 号结点到 n 号结点间有路径时即可结束算法。

```
#include<bits/stdc++.h>
using namespace std;
struct Edge{//边的类
    int v1,v2,cost;
    Edge(int vv1,int vv2,int c):v1(vv1),v2(vv2),cost(c){}
    bool operator <(const Edge&e)const{//重载小于运算符
        return this->cost>e.cost;
    }
};
priority_queue<Edge>edges;
int father[100005];//并查集
int findFather(int x){//查找根结点并进行路径压缩
    if(father[x]==x)
        return x;
```

```
	int temp=findFather(father[x]);
	father[x]=temp;
	return temp;
}
int main(){
	int n,m,ans=0;
	scanf("%d%d",&n,&m);
	iota(father,father+n+1,0);//初始化并查集
	while(m--){
		int a,b,c;
		scanf("%d%d%d",&a,&b,&c);
		edges.push(Edge(a,b,c));
	}
	while(findFather(1)!=findFather(n)){
		Edge e=edges.top();
		edges.pop();
		int ua=findFather(e.v1),ub=findFather(e.v2);
		if(ua!=ub){//边的两个端点不属于同一个集合
			father[ua]=ub;
			ans=max(ans,e.cost);//更新最长边
		}
	}
	printf("%d",ans);
	return 0;
}
```

```
#include<cstdio>
#include<iostream>
#include<vector>
#include<queue>
#include<cstring>
using namespace std ;

struct cnode {
	int k ;
	int w ;
};

bool operator < ( const cnode &d1 , const cnode & d2 )
{
	return d1.w > d2 .w ;
}

priority_queue<cnode >pq ;
```

```
bool bused[100001] = {0};

vector < vector <cnode > > v ;
const unsigned int inf = 1 <<30 ;
int main()
{
	int    n , m , a ,    b ,c ;
	cnode p ;
	cin >> n >>    m ;
	v.clear() ;
	v.resize(n+1) ;

	memset(bused , 0 , sizeof(bused)) ;

	for    ( int i = 1 ; i <= m ; i ++ )
	{
		cin    >>a >> b >> c ;
		p.k = b ;
		p.w = c ;
		v[a].push_back(p);
	}

	p.k = 1 ;
	p.w = 0 ;
	pq.push(p) ;

	while(!pq.empty())
	{
		p = pq.top() ;

		pq.pop() ;
		if( bused[p.k])
			continue ;

		bused[p.k] = true ;

		if(p.k == n)
			break ;


		for( int i = 0 ,    j = v[p.k].size() ; i < j ;i ++ )
		{
			cnode q ;
			q.k = v[p.k][i].k ;
```

```
            if(bused[q.k])
                continue ;
            q.w = p.w + v[p.k][i].w ;
            pq.push(q) ;
        }
    }

    printf("%d" , p.w) ;
    return    0;
}
```

二、最短路径

1、通信网络（暴力搜素）

问题描述

某国的军队由N个部门组成，为了提高安全性，部门之间建立了M条通路，每条通路只能单向传递信息，即一条从部门a到部门b的通路只能由a向b传递信息。信息可以通过中转的方式进行传递，即如果a能将信息传递到b，b又能将信息传递到c，则a能将信息传递到c。一条信息可能通过多次中转最终到达目的地。

由于保密工作做得很好，并不是所有部门之间都互相知道彼此的存在。只有当两个部门之间可以直接或间接传递信息时，他们才彼此知道对方的存在。部门之间不会把自己知道哪些部门告诉其他部门。

上图中给了一个4个部门的例子，图中的单向边表示通路。部门1可以将消息发送给所有部门，部门4可以接收所有部门的消息，所以部门1和部门4知道所有其他部门的存在。部门2和部门3之间没有任何方式可以发送消息，所以部门2和部门3互相不知道彼此的存在。

现在请问，有多少个部门知道所有N个部门的存在。或者说，有多少个部门所知道的部门数量（包括自己）正好是N。

输入格式

输入的第一行包含两个整数N, M，分别表示部门的数量和单向通路的数量。所有部门从1到N标号。

接下来M行，每行两个整数a, b，表示部门a到部门b有一条单向通路。

输出格式

输出一行，包含一个整数，表示答案。

由于部门数量最多只有 1000，完全可以用暴力搜索的方法，维护一个 1005*1005 的 bool 型数组 know，表示两个部门之间是否互相知晓另一个部门的存在。从每一个部门所代表的点 start 都发起一次深度优先遍历，对于遍历到的点 v，都置 know[start][v]=know[v][start]=true，表示 start 与 v 所代表的两个部门之间互相知晓另一个部门的存在。所有的点都发起一次深度优先遍历之后，如果点 v 下 know[v]数组中有 N 个为 true 的元素，则该点为知道所有其他部门存在的点。统计出所有这样的点即可。

```
#include<bits/stdc++.h>
using namespace std;
vector<int>graph[1005];
bool visit[1005];
int N,M;
bool know[1005][1005];
void DFS(int v,int start){
    visit[v]=true;
    know[start][v]=know[v][start]=true;
    for(int i=0;i<graph[v].size();++i)
        if(!visit[graph[v][i]])
            DFS(graph[v][i],start);
```

```
}
int main(){
	scanf("%d%d",&N,&M);
	while(M--){
		int a,b;
		scanf("%d%d",&a,&b);
		graph[a].push_back(b);
	}
	int result=0;
	for(int i=1;i<=N;++i){
		fill(visit+1,visit+N+1,false);
		DFS(i,i);
	}
	for(int i=1;i<=N;++i)
		if(count(know[i]+1,know[i]+N+1,true)==N)
			++result;
	printf("%d",result);
	return 0;
```

2、交通规划（最短路径）

问题描述

G国国王来中国参观后，被中国的高速铁路深深的震撼，决定为自己的国家也建设一个高速铁路系统。

建设高速铁路投入非常大，为了节约建设成本，G国国王决定不新建铁路，而是将已有的铁路改造成高速铁路。现在，请你为G国国王提供一个方案，将现有的一部分铁路改造成高速铁路，使得任何两个城市间都可以通过高速铁路到达，而且从所有城市乘坐高速铁路到首都的最短路程和原来一样长。请你告诉G国国王在这些条件下最少要改造多长的铁路。

输入格式

输入的第一行包含两个整数n, m，分别表示G国城市的数量和城市间铁路的数量。所有的城市由1到n编号，首都为1号。

接下来m行，每行三个整数a, b, c，表示城市a和城市b之间有一条长度为c的双向铁路。这条铁路不会经过a和b以外的城市。

输出格式

输出一行，表示在满足条件的情况下最少要改造的铁路长度。

算法设计：

可以直接利用 Dijkstra 算法，计算出从首都 1 号顶点到达其他顶点的最短距离，其中，如果有多条路径到达某一顶点，选择路径和最小的。为了实现上述要求，可以在定义的记录从首都到达该点的最短距离的数组 dis 之外再定义一个记录到达该点的最小边长的数组 cost。例如样例中，2 号、3 号结点都可以到达 4 号结点，但是 2 号到 4 号边长为 3，3 号到 4 号边长为 2，故 cost[4]=2。在 Dijkstra 算法中如果遇到最短距离相等的情况取能够令 cost 最小的边，那么得出的最终结果即为所求的最短路径和。

```
#include<bits/stdc++.h>
using namespace std;
struct Edge{
	int v,cost;
};
vector<Edge>graph[10005];//图
int N,M,result=0;
bool visit[10005];//当前顶点是否已访问过
int dis[10005],cost[10005];
```

```
void Dijkstra(){
    for(int ii=0;ii<N;++ii){//循环 N 次
        int v=-1,MIN=INT_MAX;
        for(int i=1;i<=N;++i)//得出当前有最短距离的未被访问的结点
            if(!visit[i]&&dis[i]<MIN){
                MIN=dis[i];
                v=i;
            }
        visit[v]=true;//当前结点已访问过
        result+=cost[v];//将到达当前结点的边长加和到最终结果中
        for(int i=0;i<graph[v].size();++i){//遍历当前结点能到达的结点
            int temp=graph[v][i].v;
            //取最短距离，如果最短距离相等，取到达结点最小的边

if((!visit[temp]&&dis[temp]>dis[v]+graph[v][i].cost)||(dis[temp]==dis[v]+graph[v][i].cost&&cost
[temp]>graph[v][i].cost)){
                dis[temp]=dis[v]+graph[v][i].cost;
                cost[temp]=graph[v][i].cost;
            }
        }
    }
}
int main(){
    scanf("%d%d",&N,&M);
    while(M--){//读取数据，注意所给图为无向图
        int a,b,c;
        scanf("%d%d%d",&a,&b,&c);
        graph[a].push_back({b,c});
        graph[b].push_back({a,c});
    }
    fill(dis+2,dis+N+1,INT_MAX);
    Dijkstra();
    printf("%d",result);
    return 0;
}
```

（二）、连通分量

高速公路（强连通分量）

问题描述

某国有n个城市，为了使得城市间的交通更便利，该国国王打算在城市之间修一些高速公路，由于经费限制，国王打算第一阶段先在部分城市之间修一些单向的高速公路。

现在，大臣们帮国王拟了一个修高速公路的计划。看了计划后，国王发现，有些城市之间可以通过高速公路直接（不经过其他城市）或间接（经过一个或多个其他城市）到达，而有的却不能。如果城市A可以通过高速公路到达城市B，而且城市B也可以通过高速公路到达城市A，则这两个城市被称为便利城市对。

国王想知道，在大臣们给他的计划中，有多少个便利城市对。

输入格式

输入的第一行包含两个整数n, m，分别表示城市和单向高速公路的数量。

接下来m行，每行两个整数a, b，表示城市a有一条单向的高速公路连向城市b。

输出格式

这是一道求强连通分量的题目。在有向图 G 中，如果两个顶点间至少存在一条互相可达路径，称两个顶点强连通。如果有向图 G 的每两个顶点都强连通，称 G 是一个强连通图。非强连通图有向图的极大强连通子图，称为强连通分量。容易知道假设一个强连通分量中结点个数为 n，则便利城市对的数量为

求解有向图的强连通分量算法有很多，可以采用 Tarjan 算法来求解

```
#include<bits/stdc++.h>
using namespace std;
const int MAX=10005;
vector<int>graph[MAX];
//index[i]表示 i 是第几个被访问的结点,lowLink[i]表示从 i 出发经有向边可到达的所有节点中最小的 index,sccno[i]表示 i 所属的强连通分量的编号
int index[MAX],lowLink[MAX],sccno[MAX],dfsNo=0,scc_cnt=0;
int ans=0;//最终结果
stack<int>s;
void DFS(int v){
    index[v]=lowLink[v]=++dfsNo;
    s.push(v);
    for(int i:graph[v]){
        if(index[i]==0){
            DFS(i);
            lowLink[v]=min(lowLink[v],lowLink[i]);
        }else if(sccno[i]==0)
            lowLink[v]=min(lowLink[v],index[i]);
    }
    if(lowLink[v]==index[v]){//是一个强连通分支的根结点
        ++scc_cnt;
        int t,num=0;//num 表示该强连通分量中结点的个数
        do{
            t=s.top();
            s.pop();
            ++num;
            sccno[t]=scc_cnt;
        }while(t!=v);
        ans+=(num-1)*num/2;//加上该强连通分量中的便利城市对个数
```

```
	}
}
int main(){
	int n,m,k,a,b;
	scanf("%d%d",&n,&m);
	while(m--){
		scanf("%d%d",&a,&b);
		graph[a].push_back(b);
	}
	for(int i=1;i<=n;++i)
		if(index[i]==0)
			DFS(i);
	printf("%d",ans);
	return 0;
}
```

三、CCF 常用 STL

（一）、链表

学生排队（链表）

```
#include<bits/stdc++.h>
using namespace std;
int main(){
	int N,M;
	scanf("%d%d",&N,&M);
	list<int>l;//存储学号的链表
	for(int i=0;i<N;++i)//将所有学号加入链表中
		l.push_back(i+1);
	while(M--){
		int a,b;
		scanf("%d%d",&a,&b);//读取移动的学号，和移动的长度
		list<int>::iterator i=l.begin();
		while(*i!=a)//遍历链表查找要移动的学号在链表中的位置
			++i;
		i=l.erase(i);//删除该元素
		while(b<0){//找到移动后的位置
			--i;
			++b;
		}
		while(b>0){
			++i;
			--b;
		}
		l.insert(i,a);//插入该元素
```

```
    }
    for(list<int>::iterator i=l.begin();i!=l.end();++i)//遍历输出
        printf("%d ",*i);
    return 0;
```

（三）、队列

1、游戏

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    int N,K;
    scanf("%d%d",&N,&K);
    queue<int>q;
    for(int i=1;i<=N;++i)//将所有人编号压入队列
        q.push(i);
    int num=1;//当前的报数
    while(q.size()>1){
        int t=q.front();//获取当前报数的人的编号
        q.pop();
        if(!(num%K==0||num%10==K))//如果既不是 K 的倍数，末位也不为 K
            q.push(t);//将这个人编号加入队列中 ，表示没有被淘汰
        ++num;//递增当前报数
    }
    printf("%d",q.front());//输出最后获胜的人的编号
```

3、二十四点

【题目描述】

定义每一个游戏由 4 个从 1-9 的数字和 3 个四则运算符组成，保证四则运算符将数字两两隔开，不存在括号和其他字符，运算顺序按照四则运算顺序进行。其中加法用符号 `+` 表示，减法用符号 `-` 表示，乘法用小写字母 `x` 表示，除法用符号 `/` 表示。在游戏里除法为整除，例如 `2 / 3 = 0`，`3 / 2 = 1`, `4 / 2 = 2`。

老师给了你 n 个游戏的解，请你编写程序验证每个游戏的结果是否为 24 。

【输入格式】

从标准输入读入数据。

第一行输入一个整数 n，从第 2 行开始到第 $n+1$ 行中，每一行包含一个长度为 7 的字符串，为上述的 24 点游戏，保证数据格式合法。

【输出格式】

输出到标准输出。

包含 n 行，对于每一个游戏，如果其结果为 24 则输出字符串 `Yes`，否则输出字符串 `No`。

这是一道求解四则表达式结果的题目。由于只涉及到四则表达式运算符，只有两个优先级，可以进行两次遍历，第一次先求解出所有乘除法的结果，第二次遍历求解出所有加减法的结果。为此，可以定义两个队列：

queue<int>num：存储加减法的操作数和乘除法的结果

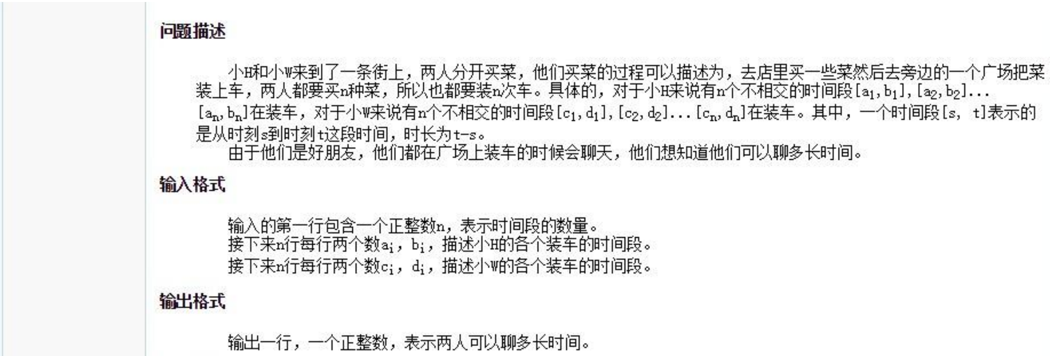
queue<char>op：存储+、-符号

第一次遍历整个表达式，将加减法的操作数和加减号存储起来，第二次同时遍历两个队列，求解出最终结果。为了编码方便，可以在每一个表达式的末尾添加上+0 字符，最终结果不变，但是编码会方便很多。具体实现可见代码。

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    int n;
    string s;
    cin>>n;
    queue<int>num;//存储加减法的操作数和乘除法的结果
    queue<char>op;//存储+、-符号
    while(n--){
        cin>>s;
        s.push_back('+');//在每个表达式末尾加上"+"字符
        for(int i=1;i<s.size();i+=2){//遍历整个字符串
            int t=s[i-1]-'0';
            for(;i<s.size()&&s[i]=='x'||s[i]=='/';i+=2){//求出连续乘除运算的结果
                t=(s[i]=='x')?t*(s[i+1]-'0'):t/(s[i+1]-'0');
            }
            num.push(t);
            op.push(s[i]);
        }
        num.push(0);//加减法操作数再放入一个 0，保证在整个表达式末尾添上了+0 运算
        int t=num.front();//第一个加减法操作数
        num.pop();
        while(!op.empty()){//同时遍历两个队列，求出加减运算的结果
            char c=op.front();
            op.pop();
            t=(c=='+')?t+num.front():t-num.front();
            num.pop();
        }
        puts(t==24?"Yes":"No");
    }
    return 0;
}
```

（三）、pair

1、买菜（pair）



问题描述

小H和小W来到了一条街上，两人分开买菜，他们买菜的过程可以描述为，去店里买一些菜然后去旁边的一个广场把菜装上车，两人都要买n种菜，所以也都要装n次车。具体的，对于小H来说有n个不相交的时间段[a_1,b_1],[a_2,b_2]...[a_n,b_n]在装车，对于小W来说有n个不相交的时间段[c_1,d_1],[c_2,d_2]...[c_n,d_n]在装车。其中，一个时间段[s, t]表示的是从时刻s到时刻t这段时间，时长为t-s。

由于他们是好朋友，他们都在广场上装车的时候会聊天，他们想知道他们可以聊多长时间。

输入格式

输入的第一行包含一个正整数n，表示时间段的数量。
接下来n行每行两个数a_i，b_i，描述小H的各个装车的时间段。
接下来n行每行两个数c_i，d_i，描述小W的各个装车的时间段。

输出格式

输出一行，一个正整数，表示两人可以聊多长时间。

本题实际上可以简化成给出两个区间，求重叠区间长度的问题。

对于给定的两个区间(a,b)和(c,d)，显然，当且仅当 a≤da\leq da≤d 且 b≥cb\geq cb≥c 时才会有重叠区间，此时重叠区间长度 L 为

L=min(b,d)−max(a,c)L=min(b,d)-max(a,c)

L=min(b,d)−max(a,c)

由于数据量比较小（最大数据量才 2000），故本题可以直接采取暴力搜索的方式，即对于小 H 的每一个时间段，计算它与小 W 的每一个时间段的重合区间，时间复杂度为 O(n2)O(n^2)O(n

2

)。

```
#include<bits/stdc++.h>
using namespace std;
int main(){
	int n,ans=0;//ans 存储最终结果
	scanf("%d",&n);
	vector<pair<int,int>>v1(n),v2(n);//分别存储小 H 和小 W 的装车时间段
	for(int i=0;i<n;++i)
		scanf("%d%d",&v1[i].first,&v1[i].second);
	for(int i=0;i<n;++i)
		scanf("%d%d",&v2[i].first,&v2[i].second);
	for(pair<int,int>p1:v1)
		for(pair<int,int>p2:v2)
			if(p1.first<=p2.second&&p1.second>=p2.first)//判断有无重叠区间
				ans+=min(p1.second,p2.second)-max(p1.first,p2.first);//加上重叠区间
	printf("%d",ans);
	return 0;
}
```

2、碰撞的小球（pair）

```
#include<bits/stdc++.h>
using namespace std;
int main(){
	int N,L,T;
	scanf("%d%d%d",&N,&L,&T);
	pair<int,int>ball[N+1];//记录小球信息,小球编号从 1 开始,first 成员记录小球所处位置，
```

second 成员记录小球当前的运动方向

```
    int line[L+1]={0};//记录线段上小球信息，为 0 表示没有小球，否则表示小球编号
    for(int i=1;i<=N;++i){//读取数据
        scanf("%d",&ball[i].first);
        ball[i].second=1;
        line[ball[i].first]=i;
    }
    while(T--)//更新 T 次小球位置信息
        for(int i=1;i<=N;++i){//遍历 N 个小球
            line[ball[i].first]=0;//小球从当前位置移走
            ball[i].first=ball[i].first+ball[i].second;//小球移动到即将到达的位置处
            if(line[ball[i].first]!=0){//小球移动到即将到达的位置处有其他小球，将这两个小球运动方向均置反向
                ball[i].second=-ball[i].second;
                ball[line[ball[i].first]].second=-ball[line[ball[i].first]].second;
            }else if(ball[i].first==0||ball[i].first==L)//小球移动到即将到达的位置是线段两端
                ball[i].second=-ball[i].second;//将这个小球运动方向均置反向
            line[ball[i].first]=i;//小球移动到即将到达的位置处
        }
    for(int i=1;i<=N;++i)//输出
        printf("%d ",ball[i].first);
   return 0;
```

3、优先级队列（默认大的先出）

```
#include<bits/stdc++.h>
using namespace std;
struct Key{//定义 Key 类
    int num;//钥匙编号
    int time;//当前时间
    bool borrow;//表示是取钥匙还是还钥匙 ，true 表示取，false 表示还
    Key(int n,int t,bool b):num(n),time(t),borrow(b){}//构造函数
    bool operator<(const Key&k)const{//重载<运算符 ，注意优先级队列默认以最大元素为队首元素
        if(this->time!=k.time)//以时间最早的位于队首
            return this->time>k.time;
        else if(this->borrow!=k.borrow)//先还钥匙再取钥匙
            return this->borrow&&!k.borrow;
        else//以钥匙编号最小的位于队首
            return this->num>k.num;
    }
};
priority_queue<Key>pq;//优先级队列
int main(){
    int N,K;
```

```
    scanf("%d%d",&N,&K);
    int a[N+1];//表示当前挂钩上钥匙顺序的数组
    for(int i=0;i<N+1;++i)
        a[i]=i;
    for(int i=0;i<K;++i){//读取数据
        int a,b,c;
        scanf("%d%d%d",&a,&b,&c);
        pq.push(Key(a,b,true));
        pq.push(Key(a,b+c,false));
    }
    while(!pq.empty()){//队列不空，继续循环
        Key k=pq.top();
        pq.pop();
        if(k.borrow){//如果是取钥匙
            int i=1;
            while(a[i]!=k.num)//查找取的钥匙编号在数组中的位置
                ++i;
            a[i]=-1;//令该位置处钥匙编号为-1，表示该挂钩没有挂钥匙
        }else{//如果是还钥匙
            int i=1;
            while(a[i]!=-1)//查找数组中第一个没挂钥匙的挂钩
                ++i;
            a[i]=k.num;//令该位置处钥匙编号为还的钥匙编号
        }
    }
    for(int i=1;i<N+1;++i)//输出
        printf("%d ",a[i]);
```

（四）、map

问题描述

给定n个整数，请统计出每个整数出现的次数，按出现次数从多到少的顺序输出。

输入格式

输入的第一行包含一个整数n，表示给定数字的个数。

第二行包含n个整数，相邻的整数之间用一个空格分隔，表示所给定的整数。

输出格式

输出多行，每行包含两个整数，分别表示一个给定的整数和它出现的次数。按出现次数递减的顺序输出。如果两个整数出现的次数一样多，则先输出值较小的，然后输出值较大的。

样例输入

12
5 2 3 3 1 3 4 2 5 2 3 5

样例输出

3 4
2 3
5 3
1 1
4 1

题目比较简单，先用 map 将遇到的数字及其出现次数储存起来，然后将 map 中所有元素搬迁到 vector 中，对 vector 按要求排序输出即可。

```
#include<bits/stdc++.h>
using namespace std;
int main(){
	int N,a;
	scanf("%d",&N);
	unordered_map<int,int>m;
	for(int i=0;i<N;++i){
		scanf("%d",&a);
		++m[a];//将读取的数字出现次数递增
	}
	vector<pair<int,int>>v;
	for(auto i:m)//将 map 中元素搬迁到 vector 中,注意 first 表示数字,second 表示数字出现的次数
		v.push_back(i);
	sort(v.begin(),v.end(),[](const pair<int,int>&p1,const pair<int,int>&p2){
		return p1.second!=p2.second?p1.second>p2.second:p1.first<p2.first;
	});//排序
	//输出
	for(auto i:v)
		printf("%d %d\n",i.first,i.second);
return 0;
```

（五）、vector

push_back 在数组的最后添加一个数据

pop_back 去掉数组的最后一个数据

```
obj[i].resize(M);
obj.push_back(i);
obj.pop_back();
```