632Team

板子君

V1.0

GooZy

2016-10-12

目录

一、字符串

1.统计以某个字符结尾的回文个数

```
/*
 * 原题 HDU5785：任意给定三元组 i<j<=k,使得 s[i,j - 1],s[j, k]
 * 均为回文串，问这样的三元组值的和？(i,j,k)值为 i*k
 *
 * 主要思想就是回文中心+1，回文边界-1，然后累加和即可，代码
 * 中 cnt 数组即可求以某个字符结尾的回文个数
 * 对称点下标 = 回文中心 * 2 - 当前点的下标
 * 所以在回文中心加上一个中心*2,边界减去这个值
 * suml 代表：以 i 开头的回文串值的和
 * sumr 代表：以 i 结尾的回文串值的和
 */

const int MAXN = 1e6 + 111;
const int MOD = 1000000007;

char Ma[MAXN << 1];
int Mp[MAXN << 1];
char s[MAXN];
ll suml[MAXN], sumr[MAXN], cnt[MAXN];

int Manacher(int len) {
    int l = 0;
    Ma[l++] = '$';
    Ma[l++] = '#';
    for (int i = 0; i < len; ++i) {
        Ma[l++] = s[i];
        Ma[l++] = '#';
    }
    Ma[l] = 0;
    int mx = 0, id = 0, cnt = 0;
    for (int i = 0; i < l; ++i) {
        Mp[i] = mx > i ? min(Mp[2*id - i], mx - i) : 1;
        while (Ma[i + Mp[i]]==Ma[i - Mp[i]]) ++Mp[i];
        if (i + Mp[i] > mx) {
            mx = i + Mp[i];
            id = i;
        }
    }
    return l;
}
```

```
int main()
{
    while (~scanf("%s", s)) {
        int n = strlen(s);
        int l = Manacher(n);
        memset(cnt, 0, sizeof cnt);
        memset(sumr, 0, sizeof sumr);
        for (int i = 2; i < l; ++i) {
            int len = (Mp[i] - 1) / 2;
            if (i % 2 == 0) {
                int base = i / 2;
                ++cnt[base];
                --cnt[base + len + 1];
                sumr[base] += 2 * base;
                sumr[base + len + 1] -= 2 * base;
            }
            else {
                int base = i / 2;
                ++cnt[base + 1];
                --cnt[base + len + 1];
                sumr[base + 1] += 2 * base + 1;
                sumr[base + len + 1] -= 2 * base + 1;
            }
        }
        for (int i = 1; i <= n; ++i) {
            sumr[i] += sumr[i - 1];
            cnt[i] += cnt[i - 1];
            cnt[i] %= MOD;
            sumr[i] %= MOD;
        }
        for (int i = 1; i <= n; ++i) {
            sumr[i] = (sumr[i] - i * cnt[i]) % MOD;
            sumr[i] += MOD;
        }

        memset(cnt, 0, sizeof cnt);
        memset(suml, 0, sizeof suml);
        for (int i = l - 2; i >= 2; --i) {
            int len = (Mp[i] - 1) / 2;
            if (i % 2 == 0) {
                int base = i / 2;
                ++cnt[base];
                --cnt[base - len - 1];
                suml[base] += 2 * base;
                suml[base - len - 1] -= 2 * base;
```

```
            }
            else {
                int base = i / 2;
                ++cnt[base];
                --cnt[base - len];
                suml[base] += 2 * base + 1;
                suml[base - len] -= 2 * base + 1;
            }
        }
        for (int i = n; i >= 1; --i) {
            suml[i] += suml[i + 1];
            cnt[i] += cnt[i + 1];
            cnt[i] %= MOD;
            suml[i] %= MOD;
        }
        for (int i = n; i >= 1; --i) {
            suml[i] = (suml[i] - i * cnt[i]) % MOD;
            suml[i] += MOD;
        }
        ll ans = 0;
        for (int i = 1; i < n; ++i) {
            ans = (ans + sumr[i] * suml[i + 1]) % MOD;
        }
        printf("%lld\n", ans);
    }
    return 0;
}
```

2.回文树

```
/*
 * 1.统计到某个前缀为止的本质不同回文串个数
 *   新增结点则代表多了一个
 * 2.统计两个字符串中相同的字符串个数(可重复)
 *   建立两棵树，count 之后，分别从结点 0 和结点 1 一起 dfs 遍历，ans += cnt[0][u] *
cnt[1][v]
 */

const int MAXN = 100005 ;
const int N = 26 ;
```

```
struct Palindromic_Tree {
    int next[MAXN][N] ;//next 指针，next 指针和字典树类似，指向的串为当前串两端加上同一个字符构成
    int fail[MAXN] ;//fail 指针，失配后跳转到 fail 指针指向的节点
    int cnt[MAXN] ; //表示节点 i 表示的本质不同的串的个数（建树时求出的不是完全的，最后 count()函数跑一遍以后才是正确的）
    int num[MAXN] ; //以 i 结尾的回文串个数，不包括本身。所以 num 相加，加上串长度为串内所有回文个数。
    int len[MAXN] ;//len[i]表示节点 i 表示的回文串的长度
    int S[MAXN] ;//存放添加的字符
    int last ;//指向上一个字符所在的节点，方便下一次 add
    int n ;//字符数组指针
    int p ;//节点指针

    int newnode ( int l ) {//新建节点
        for ( int i = 0 ; i < N ; ++ i ) next[p][i] = 0 ;
        cnt[p] = 0 ;
        num[p] = 0 ;
        len[p] = l ;
        return p ++ ;
    }

    void init () {//初始化
        p = 0 ;
        newnode ( 0 ) ;
        newnode ( -1 ) ;
        last = 0 ;
        n = 0 ;
        S[n] = -1 ;//开头放一个字符集中没有的字符，减少特判
        fail[0] = 1 ;
    }

    int get_fail ( int x ) {//和 KMP 一样，失配后找一个尽量最长的
        while ( S[n - len[x] - 1] != S[n] ) x = fail[x] ;
        return x ;
    }

    void add ( int c ) {
        c -= 'a' ;
        S[++ n] = c ;
        int cur = get_fail ( last ) ;//通过上一个回文串找这个回文串的匹配位置
        // 每增加一个新结点，本质不同回文串个数+1
        if ( !next[cur][c] ) {//如果这个回文串没有出现过，说明出现了一个新的本质不同的回文串
            int now = newnode ( len[cur] + 2 ) ;//新建节点
```

```
            fail[now] = next[get_fail ( fail[cur] )][c] ;//和 AC 自动机一样建立 fail 指针，以便失配后跳转
            next[cur][c] = now ;
            num[now] = num[fail[now]] + 1 ;
        }
        last = next[cur][c] ;
        cnt[last] ++ ;
    }

    void count () {
        for ( int i = p - 1 ; i >= 0 ; -- i ) cnt[fail[i]] += cnt[i] ;
        //父亲累加儿子的 cnt，因为如果 fail[v]=u，则 u 一定是 v 的子回文串！
    }
} ;
```

3.最大表示法

```
// s.length() == 2 * l
int MR(string &s, int l)
{
    int i = 0, j = 1, k = 0;
    while (i < l && j < l)
    {
        k = 0;
        while (k < l && s[i + k] == s[j + k]) ++k;
        if (k == l) return min(i, j);
        if (s[i + k] < s[j + k])
        {
            if (i + k + 1 <= j) i = j + 1;
            else i = i + k + 1;
        }
        else
        {
            if (j + k + 1 <= i) j = i + 1;
            else j = j + k + 1;
        }
    }
    return min(i, j);
}
```

二、数学

1.自适应 simpson 积分

```
/*
 * 近似计算积分 f(x)dx [a, b]
 * 调用：asr(a, b, 1e-6);
 */
double simpson(double a,double b) {
    double c = a + (b-a)/2;
    return (F(a) + 4*F(c) + F(b))*(b-a)/6; // 公式
}

double asr(double a,double b,double eps,double A) {
    double c = a + (b-a)/2;
    double L = simpson(a,c), R = simpson(c,b);
    if(fabs(L + R - A) <= 15*eps)return L + R + (L + R - A)/15.0;
    return asr(a,c,eps/2,L) + asr(c,b,eps/2,R);
}

double asr(double a,double b,double eps) {
    return asr(a,b,eps,simpson(a,b));
}
```

2.康托展开

```
/*
 * 用于求某个数，各位拆开后的排列组合中，
 * 这个数是排行第几。
 * 比如 132 是 1、2、3 排列中的第 2
 * NL 为数字长度
 */
```

```
int KT(string tm)
{
    int ret = 0, t;
    for (int i = 0; i < NL; ++i){
        t = 0;
        for (int j = i + 1; j < NL; ++j){
            if (tm[j] < tm[i]){
                t++;
            }
        }
        ret += t * fac[NL - i - 1];
    }
    return ret+1;
}

void initFac()
{
    fac[0] = 1;
    for (int i = 1; i <= NL; ++i){
        fac[i] = fac[i-1] * i;
    }
}
```

三、图论

1.生成树

1.1.最小生成树（Prim+heap）

```
/*
 * 最小生成树注意判断无解的情况
 * 判断无解可以在 prim 函数中，最
 * 终判断是否全部点都被访问过
 * 一个性质：单边权值 0 或者 1 的最小生成树
 * 其最小权值~最大权值都是能构造出来的。
 */
#define pii pair<int, int>
#define pr(x) cout << #x << " = " << (x) << " ; ";
#define prln(x) cout << #x << " = " << (x) << '\n';
using namespace std;
```

```cpp
const int INF = 0x7f7f7f7f;
const int MAXN = 111;

vector<pii> G[MAXN];
int cost[MAXN];
bool vis[MAXN];

int prim(int s) {
    priority_queue<pii, vector<pii>, greater<pii> > pq;
    pq.push(pii(0, s));
    cost[s] = 0;
    int ret = 0;
    while (pq.size()) {
        int u = pq.top().second, cc = pq.top().first;
        pq.pop();
        if (vis[u]) continue;
        for (int i = 0; i < G[u].size(); ++i) {
            int v = G[u][i].first, add = G[u][i].second;
            if (cost[v] > add) {
                cost[v] = add;
                pq.push(pii(cost[v], v));
            }
        }
        ret += cc;
        vis[u] = 1;
    }
    return ret;
}

int main()
{
    int n, m, u, v;
    while (~scanf("%d", &n)) {
        for (int i = 1; i <= n; ++i) {
            G[i].clear();
            cost[i] = INF;
            vis[i] = 0;
        }
        int w;
        for (int i = 1; i <= n; ++i) {
            for (int j = 1; j <= n; ++j) {
                scanf("%d", &w);
                if (i != j) G[i].push_back(pii(j, w));
            }
        }
        scanf("%d", &m);
```

```
        while (m --) {
            scanf("%d%d", &u, &v);
            G[u].push_back(pii(v, 0));
            G[v].push_back(pii(u, 0));
        }
        printf("%d\n", prim(1));
    }
    return 0;
}
```

1.2.次小生成树

```
/*
 * 求出最小生成树，然后用 mx[i][j]记录 i 到 j 的最大边，
 * 每次添加不在最小生成树上的边(u,v)时，先删除 u,v
 * 在树上的最大边，然后添加进去
 */
const int INF = 0x3f3f3f3f;
const int MAXN = 111;
const int MAXM = 111*111;

bool used[MAXN][MAXN], vis[MAXN];
int cost[MAXN], mp[MAXN][MAXN];
int pre[MAXN];
int mx[MAXN][MAXN];
int u[MAXM], v[MAXM];

int prime(int n) {
    int ret = 0;
    memset(vis, 0, sizeof vis);
    memset(used, 0, sizeof used);
    memset(mx, 0, sizeof mx);
    pre[1] = -1;
    vis[1] = 1;
    cost[1] = 0;
    for (int i = 2; i <= n; ++i) {
        cost[i] = mp[1][i];
        pre[i] = 1;
    }
```

```
    for (int i = 1; i < n; ++i) {
        int mi = INF;
        int p = -1;
        for (int j = 1; j <= n; ++j) {
            if (!vis[j] && cost[j] < mi) {
                mi = cost[j];
                p = j;
            }
        }
        // 不连通
        if (mi == INF) return -1;
        ret += mi;
        vis[p] = 1;
        used[p][pre[p]] = used[pre[p]][p] = 1;
        for (int j = 1; j <= n; ++j) {
            if (vis[j]) mx[j][p] = mx[p][j] = max(mx[j][pre[p]], cost[p]);
            if (!vis[j] && cost[j] > mp[p][j]) {
                cost[j] = mp[p][j];
                pre[j] = p;
            }
        }
    }
    return ret;
}



int main()
{
    int t, n, m; scanf("%d", &t);
    while (t --) {
        scanf("%d%d", &n, &m);
        memset(mp, 0x3f, sizeof mp);
        for (int i = 0; i < m; ++i) {
            scanf("%d%d", &u[i], &v[i]);
            scanf("%d", &mp[u[i]][v[i]]);
            mp[v[i]][u[i]] = mp[u[i]][v[i]];
        }
        int ans = prime(n);
        // 求出最小生成树
        int sub = INF;
        for (int i = 0; i < m; ++i) {
            // 边不在生成树上
            if (!used[u[i]][v[i]]) {
                sub = min(sub, ans - mx[u[i]][v[i]] + mp[u[i]][v[i]]);
            }
```

```
        }
        if (ans == sub) {
            puts("Not Unique!");
        }
        else printf("%d\n", ans);
    }
    return 0;
}
```

1.3.最小树形图（有向图最小生成树）

```
/*
 * 如果无定根，那么新建一个虚拟结点，向各个结点连边，
 * 边权值为图中所有边权之和+1，算最小树形图权值，如果
 * 权值大于两倍（边权值之和+1），则不存在，否则答案为
 * 返回值 -（边权值之和+1）
 * 下方为定根代码
 */

const int INF = 0x7f7f7f7f;
const int MAXN = 111;

struct Edge{
    int u, v;
    double w;
}edge[MAXN * MAXN];

int n, m, pre[MAXN], newid[MAXN], vis[MAXN];
double x[MAXN], y[MAXN], in[MAXN];

double getdis(int a, int b)
{
    double delx = x[a] - x[b], dely = y[a] - y[b];
    return sqrt(delx * delx + dely * dely);
}

double zhuLiu(int rt)
{
    double ret = 0;
```

```
    while (1)
    {
        for (int i = 0; i < n; ++i) in[i] = INF;
        // 第一步：找出每个点的最小权入边
        for (int i = 0; i < m; ++i)
        {
            int u = edge[i].u, v = edge[i].v;
            if (in[v] > edge[i].w && u != v) in[v] = edge[i].w, pre[v] = u;
        }
        for (int i = 0; i < n; ++i)
        {
            if (i == rt) continue;
            if (in[i] == INF) return -1; // 判断是否无法构成最小树形图
        }
        // 第二步：判环
        int cnt = 0;
        memset(newid, -1, sizeof newid);
        memset(vis, -1, sizeof vis);
        in[rt] = 0;
        for (int i = 0; i < n; ++i)
        {
            ret += in[i];
            int v = i;
            while (vis[v] != i && newid[v] == -1 && v != rt) // 找环。vis[]用来标记
点在哪个点为首的环中
            {
                vis[v] = i;
                v = pre[v];
            }
            if (v != rt && newid[v] == -1) // 找到环了，缩点
            {
                for (int u = pre[v]; u != v; u = pre[u]) newid[u] = cnt;
                newid[v] = cnt++;
            }
        }
        if (cnt == 0) break; // 没有环

        for (int i = 0; i < n; ++i) // 重新赋予其他点标号
            if (newid[i] == -1) newid[i] = cnt++;

        for (int i = 0; i < m; ++i) // 建立新图
        {
            int u = edge[i].u, v = edge[i].v;
            edge[i].u = newid[u];
            edge[i].v = newid[v];
```

```
            if (newid[u] != newid[v]) edge[i].w -= in[v]; // 选择当前边的同时便放弃了原来的最小入边.原来的已经加到 ret 中了
        }
        n = cnt;
        rt = newid[rt];
    }
    return ret;
}

int main()
{
    while (~scanf("%d%d", &n, &m))
    {
        for (int i = 0; i < n; ++i) scanf("%lf%lf", &x[i], &y[i]);
        for (int i = 0; i < m; ++i)
        {
            scanf("%d%d", &edge[i].u, &edge[i].v);
            --edge[i].u;
            --edge[i].v;
            edge[i].w = getdis(edge[i].u, edge[i].v);
        }

        double ans = zhuLiu(0);
        if (ans != -1) printf("%.2f\n", ans);
        else printf("poor snoopy\n");
    }
    return 0;
}
```

1.4.限制 K 度的最小生成树

```
/*
 * 考虑把起点相连的边去掉，求出 m 个连通分量
 * 然后起点向各个连通分量权值最小的连边，得到
 * 限制 m 度的最小生成树，然后考虑 m+1，枚举还未
 * 添加的边，每次添加后肯定构成环，去掉环上权
 * 值最大的边，枚举所有组合，取最小的得到 m+1
 */

using namespace std;
#define MAXN 30
#define INF 0x7FFFFFFF
```

```
int n , k , ans , cnt;//边的长度 n 和 k 度 , cnt 表示有几个点
int vis[MAXN];//标记点 i 是否加入了生成树
int mark[MAXN];//在 prime 算法里面会用到
int pre[MAXN];//点 i 的前驱节点
int father[MAXN]; // 生成树中父节点的编号
int best[MAXN]; // 记录点 i 到限制点并且和限制点没有关联的最大边的点的编号
int edge[MAXN][MAXN];// 用来表示边是否已在生成树中
int G[MAXN][MAXN];// 保存两点之间的权值
int lowcost[MAXN];
map<string , int>m;

void init(){
   for(int i = 0 ; i < MAXN ; i++){
      for(int j = 0 ; j < MAXN ; j++)
         G[i][j] = INF;
   }
}

//dfs 把一个连通分支里面的点全部指向 s
void dfs(int s){
   for(int i = 1 ; i <= cnt ; i++){
      if(mark[i] && edge[i][s]){
        father[i] = s;
        mark[i] = 0;
        dfs(i);
      }
   }
}

int prime(int s){
   int sum , pos;
   memset(mark , 0 ,sizeof(mark));
   vis[s] = mark[s] = 1;
   sum = 0;
   for(int i = 1 ; i <= cnt ; i++){
      lowcost[i] = G[s][i];
      pre[i] = s;
   }
   for(int i = 1 ; i <= cnt ; i++){
      pos = -1;
      for(int j = 1 ; j <= cnt ; j++){
         if(!vis[j] && !mark[j]){
           if(pos == -1 || lowcost[j] < lowcost[pos])
             pos = j;
         }
      }
```

```
        if(pos == -1)
          break;
        vis[pos] = mark[pos] = 1;
        edge[pre[pos]][pos] = edge[pos][pre[pos]] = 1;
        sum += G[pre[pos]][pos];
        for(int j = 1 ; j <= cnt ; j++){
           if(!vis[j] && !mark[j]){
             if(lowcost[j] > G[pos][j]){
               lowcost[j] = G[pos][j];
               pre[j] = pos;
             }
           }
        }
    }
    //以下是找到一条最小权值的边把该连通分量连接到限制点 1
    int min = INF;
    int root = -1;/*要和 1 点连接的点*/
    for(int i = 1 ; i <= cnt ; i++){
        if(mark[i] && G[i][1] < min){
          min = G[i][1];
          root = i;
        }
    }
    //把当前的连通
    mark[root] = 0;
    dfs(root);
    father[root] = 1;
    return sum+min;
}

//求 best 数组函数,求解 s-1 路径上权值最大的边的终点
int Best(int s){
    if(father[s] == 1)
      return -1;
    if(best[s] != -1)
      return best[s];
    int tmp = Best(father[s]);
    if(tmp != -1 && G[father[tmp]][tmp] > G[father[s]][s])
      best[s] = tmp;
    else
      best[s] = s;
    return best[s];
}

void solve(){
    memset(father , -1 , sizeof(father));
```

```
memset(vis , 0 , sizeof(vis));
memset(edge , 0 , sizeof(edge));
vis[1] = 1;//把 1 这个点当成限制点
int num = 0;//把 1 限制点去掉以后的连通分支的个数
ans = 0;

//先求最小 num 度限制树
for(int i = 1 ; i <= cnt ; i++){
   if(!vis[i]){
     num++;
     ans += prime(i);
   }
}

//再由 m 度限制生成树->k 度生成树
int minAdd;//增加一条边改变的权值大小
int change;// 记录回路上要删除的边的终点
// 循环 k-num 次
for(int i = num+1 ; i <= k && i <= cnt ; i++){
   memset(best , -1 , sizeof(best));
   // 求出 best 数组
   for(int j =1 ; j <= cnt ; j++){
      if(best[j] == -1 && father[j] != 1)
        Best(j);
   }
   minAdd = INF;
   for(int j = 1 ; j <= cnt ; j++){
      if(G[1][j] != INF && father[j] != 1){
        int a = best[j];
        int b = father[best[j]];
        int tmp = G[1][j]-G[a][b];
        if(tmp < minAdd){
          minAdd = tmp;
          change = j;
        }
      }
   }
   if(minAdd >= 0)//用于度数不大于 k 的限制，如果 k 限制，就不用 break 了
     break;
   ans += minAdd;
   int a = best[change];
   int b = father[change];
   G[a][b] = G[b][a] = INF;/*把这一条边去掉就是赋值为 INF*/
   father[a] = 1;/*把 a 的父亲节点指向为限制点 1*/
   G[a][1] = G[1][a] = G[change][1];/*新增加的一条边的权值*/
   G[1][change] = G[change][1] = INF;
```

```
    }
}

int main(){
   int v;
   string str1 , str2;
   m.clear();
   m["Park"] = 1;
   cnt = 1;//初始化有一个点
   init();
   scanf("%d" , &n);
   for(int i = 0 ; i < n ; i++){
      cin>>str1>>str2>>v;
      int a = m[str1];
      int b = m[str2];
      if(!a)
        m[str1] = a = ++cnt;
      if(!b)
        m[str2] = b = ++cnt;
      if(G[a][b] > v)//a b 为点的编号，所以上面不能直接把 m[str1] = 1
        G[a][b] = G[b][a] = v;
   }
   scanf("%d" , &k);
   solve();
   printf("Total miles driven: %d\n" , ans);
   return 0;
}
```

1.5.最优比率生成树

```
/*
 * 本题要求 dis/cost 的比率最小化，就是求：
 * cost/dis = r 的比率最大化。将式子化为：
 * z(r) = cost - r * dis。这里的 cost、dis
 * 都是 sigma，即实际被选入边的和，由这个等式
 * 新边 i 的边权为：cost[i] - r * dis[i]，z(r)
 * 这值就是求新图中最小生成树的权值
 * 根据新边的定义，r 越大，z(r)越小，是递减的
 * 然后当 r 取最大值时，z(r)等于 0，用反证法证明。
 */

const int INF = 0x7f7f7f7f;
const int MAXN = 1e3 + 111;
const double eps = 1e-6;
```

```
struct P {
    double x, y, h;
    double getdis(P a) {
        double dx = x - a.x, dy = y - a.y;
        return sqrt(dx * dx + dy * dy);
    }
    double getheight(P a) {
        return fabs(h - a.h);
    }
    void read() {
        scanf("%lf%lf%lf", &x, &y, &h);
    }
}p[MAXN];

int n;
int pre[MAXN];
double mp[MAXN][MAXN]; // 新边
double dis[MAXN][MAXN], cost[MAXN][MAXN];
bool vis[MAXN];
double dist[MAXN];

double prime() {
    int s = 0;
    for (int i = 0; i < n; ++i) {
        dist[i] = INF;
        pre[i] = 0;
        vis[i] = false;
    }
    vis[s] = true;
    dist[s] = 0;
    int mip = s;
    double mi = 0.0, dissum = 0, costsum = 0;
    for(int i = 1; i < n; i++){
        for(int j = 0; j < n; j++)
            if(!vis[j] && dist[j] > mp[mip][j]){
                dist[j] = mp[mip][j];
                pre[j] = mip;
            }
        mi = INF;
        for(int j = 0; j < n; j++)
            if(!vis[j] && mi > dist[j]) mi = dist[j], mip = j;
        vis[mip] = true;
        dissum += dis[mip][pre[mip]];
        costsum += cost[mip][pre[mip]];
    }
    return costsum / dissum;
```

```
}

inline void build(double rate) {
    for (int i = 0; i < n; ++i) {
        for (int j = i + 1; j < n; ++j) {
            mp[i][j] = mp[j][i] = cost[i][j] - rate * dis[i][j];
        }
    }
}

// 牛顿迭代
double solve() {
    double last = 0, cur = 1.0;
    while (fabs(cur - last) > eps) {
        last = cur;
        build(last);
        cur = prime();
    }
    return last;
}

int main()
{
    while (~scanf("%d", &n) && n) {
        for (int i = 0; i < n; ++i) {
            p[i].read();
        }
        for (int i = 0; i < n; ++i) {
            dis[i][i] = cost[i][i] = 0;
            for (int j = i + 1; j < n; ++j) {
                dis[i][j] = dis[j][i] = p[i].getdis(p[j]);
                cost[i][j] = cost[j][i] = p[i].getheight(p[j]);
            }
        }
        printf("%.3f\n", solve());
    }
    return 0;
}
```

2.最短路

2.1.次短路

```
/*
 * 转移的时候是使用优先队列中的距离来更新，因为这里面
 * 的距离是随着弹出点代表的次短还是最短中的点而不同的。
 */
const int INF = 0x7f7f7f7f;
const int N = 5e3 + 111;
const int M = 2e5 + 111;

struct E {
    int nxt, to, w;
}edge[M];
int dis1[N], dis2[N], head[M], cnt; // dis1:最短 dis2:次短

void add_edge(int u, int v, int w) {
    edge[cnt].to = v;
    edge[cnt].w = w;
    edge[cnt].nxt = head[u];
    head[u] = cnt++;
}

void dijkstra(int s) {
    dis1[s] = 0;
    priority_queue<pii, vector<pii>, greater<pii> > pq;
    pq.push(pii(dis1[s], s));
    while (pq.size()) {
        pii temp = pq.top(); pq.pop();
        int u = temp.second, d = temp.first;
        if (dis2[u] < d) continue;
        for (int i = head[u]; ~i; i = edge[i].nxt) {
            int v = edge[i].to;
            int temp = d + edge[i].w; // 由于维护的是最短次短两者，所以我们使用 d 来更新，
d 随着弹出点的定义不同而不同
            if (temp < dis1[v]) { // 最短路更新，次短路必定更新
                dis2[v] = dis1[v];
                dis1[v] = temp;
                pq.push(pii(dis2[v], v));
                pq.push(pii(dis1[v], v));
            }
            else if (dis1[v] < temp && temp < dis2[v]) { // 否则，在当前值在两者之间
时更新
                dis2[v] = temp;
```

```
                pq.push(pii(dis2[v], v));
            }
        }
    }
}

int main()
{
    cnt = 0;
    int n, m;
    scanf("%d%d", &n, &m);
    memset(head, -1, sizeof(int) * (n + 5));
    int u, v, w;
    for (int i = 0; i < m; ++i) {
        scanf("%d%d%d", &u, &v, &w);
        add_edge(u, v, w);
        add_edge(v, u, w);
    }

    for (int i = 1; i <= n; ++i) dis1[i] = dis2[i] = INF;
    dijkstra(1);

    printf("%d\n", dis2[n]);
    return 0;
}
```

2.2.K 短路

```
/*
 * 设估价函数 f(x) = g(x) + d(x);
 * g(x)为当前到点 x 的距离，d(x)为
 * x 到终点的最短距离，通过反向边
 * 求出。以 f(x)为第一优先，其次 g(x)
 * 即可求得 k 短路
 */
const int INF = 0x3f3f3f3f;
const int MAXN = 1e3 + 11;
const int MAXM = 1e5 + 11;

int n, m, s, t, k;
struct Edge {
    int to, w, nxt;
}edge[MAXM], redge[MAXM];
int head[MAXN], rhead[MAXN], tol;
```

```
void init() {
    memset(head, -1, sizeof head);
    memset(rhead, -1, sizeof rhead);
    tol = 0;
}

void addedge(int u, int v, int w) {
    edge[tol].to = v;
    edge[tol].w = w;
    edge[tol].nxt = head[u];
    head[u] = tol;

    redge[tol].to = u;
    redge[tol].w = w;
    redge[tol].nxt = rhead[v];
    rhead[v] = tol++;
}

bool in[MAXN];
int dis[MAXN];
void spfa(int src) {
    for (int i = 1; i <= n; ++i) {
        dis[i] = INF;
        in[i] = 0;
    }
    queue<int> q;
    q.push(src);
    in[src] = 1;
    dis[src] = 0;
    while (q.size()) {
        int u = q.front(); q.pop();
        in[u] = 0;
        for (int i = rhead[u]; ~i; i = redge[i].nxt) {
            int v = redge[i].to, w = redge[i].w;
            if (dis[v] > dis[u] + w) {
                dis[v] = dis[u] + w;
                if (!in[v]) {
                    in[v] = 1;
                    q.push(v);
                }
            }
        }
    }
}
```

```
struct A
{
    int f, g, v; // f = g + dis
    bool operator <(const A a)const {
        if(a.f == f) return a.g < g;
        return a.f < f;
    }
};
int astar(int src, int des) {
    int cnt = 0;
    priority_queue<A> pq;
    if (src == des) ++k; // ¸HÇd¸
    if (dis[src] == INF) return -1;
    A nxt;
    nxt.v = src, nxt.g = 0, nxt.f = nxt.g + dis[nxt.v];
    pq.push(nxt);
    while (pq.size()) {
        A cur = pq.top(); pq.pop();
        if (cur.v == des && ++cnt == k) {
            return cur.g;
        }
        for (int i = head[cur.v]; ~i; i = edge[i].nxt) {
            nxt.v = edge[i].to;
            nxt.g = cur.g + edge[i].w;
            nxt.f = nxt.g + dis[nxt.v];
            pq.push(nxt);
        }
    }
    return -1;
}

int main()
{
    int u, v, w;
    while (~scanf("%d%d", &n, &m)) {
        init();
        while (m --) {
            scanf("%d%d%d", &u, &v, &w);
            addedge(u, v, w);
        }
        scanf("%d%d%d", &s, &t, &k);
        spfa(t);
        printf("%d\n", astar(s, t));
    }
    return 0;
}
```

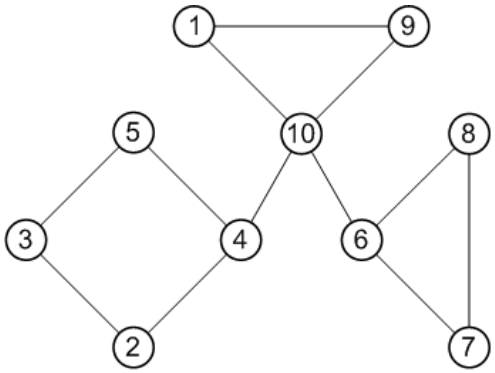
3.连通分量

3.1.强连通分量

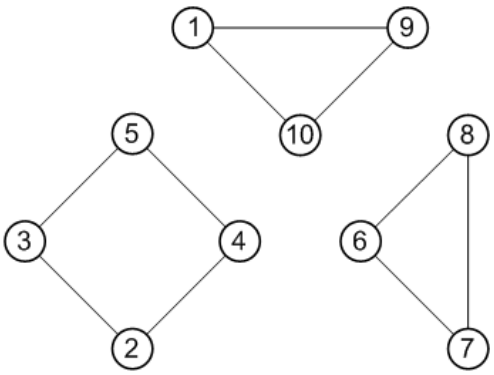
```
/*
 * 强连通分量这一概念属于有向图
 * tarjan 复杂度: O(n + m)
 * 本题判断是否只有一个连通块，来自 HDU 1269
 */

const int MAXN = 1e4 + 11;
const int MAXM = 1e5 + 11;

struct Edge {
    int to, next;
}edge[MAXM];
int head[MAXN], tol;
int low[MAXN], dfn[MAXN], Stack[MAXN], Belong[MAXN];
int index, top;
int block;
bool inStack[MAXN];

void add_edge(int u, int v) {
    edge[tol].to = v;
    edge[tol].next = head[u];
    head[u] = tol++;
}

void init() {
    memset(head, -1, sizeof head);
    memset(dfn, 0, sizeof dfn);
    memset(inStack, 0, sizeof inStack);
    index = top = block = 0;
    tol = 0;
}

void tarjan(int u) {
    int v;
    low[u] = dfn[u] = ++index;
    Stack[top++] = u;
    inStack[u] = 1;

    for (int i = head[u]; ~i; i = edge[i].next) {
        v = edge[i].to;
        if (!dfn[v]) {
```

```
            tarjan(v);
            low[u] = min(low[u], low[v]);
        }
        else if (inStack[v])
            low[u] = min(low[u], dfn[v]);
    }

    if (low[u] == dfn[u]) { // 缩点
        ++block;
        do {
            v = Stack[--top];
            inStack[v] = 0;
            Belong[v] = block;
        }while (v != u);
    }
}

int main()
{
    int n, m, u, v;
    while (~scanf("%d%d", &n, &m) && (n | m)) {
        init();
        while (m --) {
            scanf("%d%d", &u, &v);
            add_edge(u, v);
        }
        for (int i = 1; i <= n; ++i) {
            if (!dfn[i]) {
                tarjan(i);
            }
        }
        puts(block == 1 ? "Yes" : "No");
    }
    return 0;
}
```

3.2.边双连通分量



(a) 连通无向图



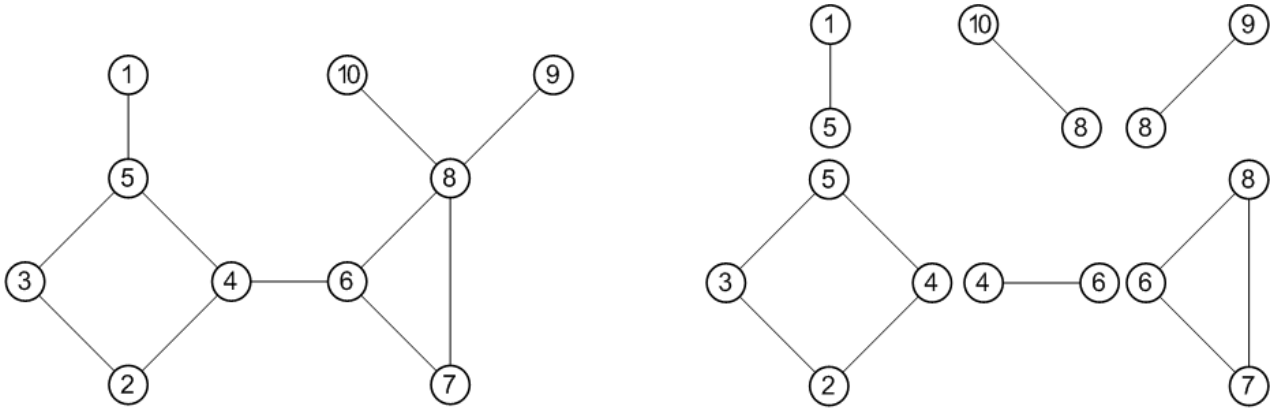
(b) 边双连通分量

```
/*
 * 原题：POJ 3177
 * 题意：给出一幅含有重边的无向图，问至少连多少条边，
 * 使得图中任意两个点 u、v 都有 u->v 的路径，和 v->u 的路径
 * ，且两者没有相同的边。
 * 本题重边当做一条边考虑，然后缩点，最后变成一棵树，答案即为：(度数为 1 的点 + 1) / 2。
 * 对于需要考虑重边的题目，我们 tarjan 的时候按边标记即可。
 */

const int MAXN = 5e3 + 111;
const int MAXM = 2e4 + 111; // 边的两倍

struct Edge {
    int to, next;
    bool cut;
}edge[MAXM];
int head[MAXN], tol;
int low[MAXN], dfn[MAXN], Stack[MAXN], Belong[MAXN];
int index, top;
int block;
bool inStack[MAXN];
int bridge;

void add_edge(int u, int v) {
    edge[tol].to = v;
    edge[tol].cut = 0;
    edge[tol].next = head[u];
    head[u] = tol++;
}

void init() {
    memset(head, -1, sizeof head);
    memset(dfn, 0, sizeof dfn);
    memset(inStack, 0, sizeof inStack);
```

```
    index = top = block = 0;
    tol = 0;
}

void tarjan(int u, int fa) {
    int v;
    low[u] = dfn[u] = ++index;
    Stack[top++] = u;
    inStack[u] = 1;

    for (int i = head[u]; ~i; i = edge[i].next) {
        v = edge[i].to;
        if (v == fa) continue;
        if (!dfn[v]) {
            tarjan(v, u);
            low[u] = min(low[u], low[v]);
            if (low[v] > dfn[u]) { // low[v] < dfn[u]说明子结点能到达 u 的父节点，也就是形成环了
                ++bridge;
                edge[i].cut = edge[i^1].cut = 1;
            }
        }
        else if (inStack[v])
            low[u] = min(low[u], dfn[v]);
    }

    if (low[u] == dfn[u]) { // 缩点
        ++block;
        do {
            v = Stack[--top];
            inStack[v] = 0;
            Belong[v] = block;
        }while (v != u);
    }
}

int deg[MAXN];

int main()
{
    int f, r, u, v;
    init();
    scanf("%d%d", &f, &r);
    while (r --) {
        scanf("%d%d", &u, &v);
        add_edge(u, v);
```

```
            add_edge(v, u);
        }

        tarjan(1, 0);

        int ans = 0;
        memset(deg, 0, sizeof deg);
        for (int i = 1; i <= f; ++i) {
            for (int j = head[i]; ~j; j = edge[j].next) {
                if (edge[j].cut) ++deg[Belong[i]]; // 新图中，割边就是连结所有点的边
            }
        }

        for (int i = 1; i <= block; ++i) if (deg[i] == 1) ++ans;

        printf("%d\n", (ans + 1) / 2);
        return 0;
}
```

3.3.点双连通分量



(a) 连通无向图　　(b) 重连通分量

```
/*
 * 原题：POJ 2942
 * 题意：给出骑士们的憎恨关系，为了让会议进行下去，
 * 凡是无法在所有奇圈中留下的骑士都要被剔除，问要
 * 剔除多少个？
 * 求补图，然后求含有奇圈的双连通分量，判定奇圈用二分图染色
 */

const int INF = 0x7f7f7f7f;
const int MAXN = 1e3 + 111;
```

```
vector<int> G[MAXN];
bool mp[MAXN][MAXN];
int n, m;
int dfn[MAXN], low[MAXN], bccno[MAXN], indx, block, top, sta[MAXN];
bool in[MAXN], vis[MAXN], can[MAXN];
int temp[MAXN], cnt, color[MAXN];

bool dfs(int u, int c) {
    color[u] = c;
    for (int i = 0; i < G[u].size(); ++i) {
        int v = G[u][i];
        if (!vis[v]) continue;
        if (color[v] == -1 && !dfs(v, !c)) return 0;
        else if (color[v] != -1) {
            if (color[v] == c) return 0;
        }
    }
    return 1;
}

void tarjan(int u, int p) {
    dfn[u] = low[u] = ++indx;
    sta[top++] = u;
    in[u] = 1;

    for (int i = 0; i < G[u].size(); ++i) {
        int v = G[u][i];
        if (v == p) continue;
        if (!dfn[v]) {
            tarjan(v, u);
            low[u] = min(low[u], low[v]);
            if (low[v] >= dfn[u]) {
                ++block;
                int sv;
                cnt = 0;
                memset(vis, 0, sizeof vis);
                do {
                    sv = sta[--top];
                    bccno[sv] = block;
                    in[sv] = 0;
                    vis[sv] = 1;
                    temp[cnt++] = sv;
                }while (sv != v);
                vis[u] = 1;

                memset(color, -1, sizeof color);
```

```
                if (!dfs(u, 0)) {
                    can[u] = 1;
                    while (cnt--)
                        can[temp[cnt]] = 1;
                }
            }
        }
        else if (in[v] && low[u] > dfn[v])
            low[u] = dfn[v];
    }
}

void find_bcc() {
    memset(dfn, 0, sizeof dfn);
    memset(low, 0, sizeof low);
    memset(in, 0, sizeof in);
    memset(bccno, 0, sizeof bccno);
    memset(can, 0, sizeof can);
    indx = block = top = 0;
    for (int i = 1; i <= n; ++i) {
        if (!dfn[i]) {
            tarjan(i, -1);
        }
    }
}

int main()
{
    while (~scanf("%d%d", &n, &m) && (n|m)) {
        int u, v;
        memset(mp, 0, sizeof mp);
        for (int i = 1; i <= n; ++i) G[i].clear();
        while (m --) {
            scanf("%d%d", &u, &v);
            mp[u][v] = mp[v][u] = 1;
        }
        for (int i = 1; i <= n; ++i) {
            for (int j = i + 1; j <= n; ++j) {
                if (!mp[i][j]) {
                    G[i].push_back(j);
                    G[j].push_back(i);
                }
            }
        }

        find_bcc();
```

```
        int ans = n;
        for (int i = 1; i <= n; ++i) {
            if (can[i]) --ans;
        }
        printf("%d\n", ans);
    }
    return 0;
}
```

3.4.2-SAT

```
/*
 * 原题：HDU 1824
 * 题意：n 支队伍，m 个关系。满足：1.队伍里面，队长和其余
 * 两名队员两者中必须有一个要留下来；2.关系中，A 队员留下
 * B 队员就得离开，B 队员留下，A 队员就要离开。问：能否合理
 * 安排满足以上两个关系。
 */
```

解题思路：

将每个点拆分成两种状态：留下和不留下，然后根据题目，设：A为队长，B和C为队员，那么就有!A->(B^C), (!Bv!C)->A, 然后每个关系中，有：B->!C, C->!B. 最终根据关系建边，判断矛盾关系是否在同一个连通分量内即可。

2-SAT类型的题，重要的就在于把这些对象间的关系理清楚。

```
const int INF = 0x7f7f7f7f;
const int MAXN = 6e3 + 111;

vector<int> G[MAXN];
int n, m;
int dfn[MAXN], low[MAXN], sta[MAXN], id[MAXN], indx, scc, top;
bool in[MAXN];

void init() {
    for (int i = 0; i <= 6 * n; ++i) {
        G[i].clear();
        in[i] = 0;
        dfn[i] = 0;
    }
    indx = scc = top = 0;
```

```
}

void tarjan(int u) {
    dfn[u] = low[u] = ++indx;
    sta[top++] = u;
    in[u] = 1;
    for (int i = 0; i < G[u].size(); ++i) {
        int v = G[u][i];
        if (!dfn[v]) {
            tarjan(v);
            low[u] = min(low[u], low[v]);
        }
        else if (in[v]) low[u] = min(low[u], dfn[v]);
    }

    if (dfn[u] == low[u]) {
        ++scc;
        int v;
        do {
            v = sta[--top];
            in[v] = 0;
            id[v] = scc;
        } while (v != u);
    }
}

int main()
{
    int a, b, c;
    while (~scanf("%d%d", &n, &m)) {
        init();
        int add = 3 * n;
        for (int i = 0; i < n; ++i) {
            scanf("%d%d%d", &a, &b, &c);
            G[a + add].push_back(c);
            G[a + add].push_back(b);
            G[b + add].push_back(a);
            G[c + add].push_back(a);
        }
        int u, v;
        while (m --) {
            scanf("%d%d", &u, &v);
            G[u].push_back(v + add);
            G[v].push_back(u + add);
        }
```

```
        for (int i = 0; i < 6 * n; ++i) {
            if (!dfn[i]) tarjan(i);
        }

        bool flag = 1;
        for (int i = 0; i < 3 * n; ++i) {
            if (id[i] == id[i + 3 * n]) {
                flag = 0;
                break;
            }
        }
        printf("%s\n", flag? "yes" : "no");
    }
    return 0;
}
```

4.最大团

4.1.最大团

```
/*
 * 返回最大团是多大
 * 最大团即最大的完全图
 */

const int INF = 0x7f7f7f7f;
const int MAXN = 51;

int n, ans;
int mp[MAXN][MAXN];
int avl[MAXN][MAXN]; // 每层的可行点
int cnt[MAXN]; // i~n 的点集构成的最大团数目
int id[MAXN]; // 团中元素的标号
int group[MAXN]; // 最大团中的元素
```

```
// 当前可行结点个数为 cur， 团中元素个数为 tol
bool dfs(int cur, int tol) {
    if (cur == 0) {
        if (tol > ans) {
            ans = tol;
            return 1;
        }
        return 0;
    }
    for (int i = 0; i < cur; ++i) {
        if (cur - i + tol <= ans) return 0;
        int u = avl[tol][i];
        // 如果取点 i 但是点 i 能达到的最大和目前的相加比答案还小，可以直接返回了。
        if (cnt[u] + tol <= ans) return 0;
        int nxt = 0;
        for (int j = i + 1; j < cur; ++j) {
            if (mp[u][avl[tol][j]])
                avl[tol + 1][nxt++] = avl[tol][j];
        }
        id[tol] = u;
        if (dfs(nxt, tol + 1)) return 1;
    }
    // 每增加一个点，最大团个数至多加 1，如果比答案大，说明就是目前最优了。
    if (tol > ans) {
// 记录路径用
//        for (int i = 0; i < tol; ++i)
//            group[i] = id[i];
        ans = tol;
        return 1;
    }
    return 0;
}

void MaxClique() {
    ans = -1;
    for (int i = n - 1; i >= 0; --i) {
        int cur = 0;
        for (int j = i + 1; j < n; ++j) {
            if (mp[i][j]) avl[1][cur++] = j; // 可达点才是应该访问的点
        }
        id[0] = i;
        dfs(cur, 1);
        cnt[i] = ans;
        if (ans == n) return; // 达到上界就别计算下去了
    }
}
```

```
int main()
{
    while (~scanf("%d", &n) && n) {
        for (int i = 0; i < n; ++i)
            for (int j = 0; j < n; ++j)
                scanf("%d", &mp[i][j]);
        MaxClique();
        printf("%d\n", ans);
    }
    return 0;
}
```

4.2.最大团计数

```
/*
 * 统计最大团有多少个
 */

const int INF = 0x7f7f7f7f;
const int MAXN = 150;

int n, ans;
int g[MAXN][MAXN];
int S, all[MAXN][MAXN], some[MAXN][MAXN], none[MAXN][MAXN];
//下标从 1 开始
//all 为已取顶点集，some 为未处理顶点集，none 为不取的顶点集
//我们求最大团顶点数时只要 some，要求记录路径时要 all 和 some，这里求极大团数量，需要 all、
some、none
void dfs(int d, int an, int sn, int nn)
{
    if(S > 1000) return ; // 极大团数量超过 1000 就不再统计
    if(sn == 0 && nn == 0) ++S;//sn==0 搜索到终点，只有 nn==0 时，才是一个极大团
    int u = some[d][0];//pivot vertex
    for(int i = 0; i < sn; ++i)
    {
        int v = some[d][i];
        if(g[u][v] || !v) continue;
        int tsn = 0, tnn = 0;
        for(int j = 0; j < an; j ++) all[d + 1][j] = all[d][j];
        all[d + 1][an] = v;
        for(int j = 0; j < sn; ++j) if(g[v][some[d][j]]) some[d + 1][tsn++] =
some[d][j];
```

```
        for(int j = 0; j < nn; ++j) if(g[v][none[d][j]]) none[d + 1][tnn++] =
none[d][j];
        dfs(d + 1, an + 1, tsn, tnn);
        //把 v 从 some 取出，放入 none
        some[d][i] = 0, none[d][nn++] = v;
    }
}

void MaxClique() {
    S = 0;
    for(int i = 0; i < n; ++i) some[0][i] = i + 1;
    dfs(0, 0, n, 0);
}

int main()
{
    int m, u, v;
    while (~scanf("%d%d", &n, &m)) {
        memset(g, 0, sizeof g);
        for (int i = 0; i < m; ++i) {
            scanf("%d%d", &u, &v);
            g[u][v] = g[v][u] = 1;
        }
        MaxClique();
        if (S > 1000) puts("Too many maximal sets of friends.");
        else printf("%d\n", S);
    }
    return 0;
}
```

5.网络流

5.1.最大流

```
const int MAXN = 510;//点数的最大值
const int MAXM = 540010;//边数的最大值的两倍
const int INF = 0x3f3f3f3f;

struct Edge
{
    int to,next,cap,flow;
}edge[MAXM];//注意是 MAXM
int tol, src, des;
int head[MAXN];
int gap[MAXN],dep[MAXN],pre[MAXN],cur[MAXN];

void init()
{
    tol = 0;
    memset(head,-1,sizeof(head));
}
//加边，单向图三个参数，双向图四个参数
void addedge(int u,int v,int w,int rw=0)
{
    edge[tol].to = v;edge[tol].cap = w;edge[tol].next = head[u];
    edge[tol].flow = 0;head[u] = tol++;
    edge[tol].to = u;edge[tol].cap = rw;edge[tol].next = head[v];
    edge[tol].flow = 0;head[v]=tol++;
}
//输入参数：起点、终点、点的总数
//点的编号没有影响，只要输入点的总数
int sap(int start,int end,int N)
{
    memset(gap,0,sizeof(gap));
    memset(dep,0,sizeof(dep));
    memcpy(cur,head,sizeof(head));
    int u = start;
    pre[u] = -1;
    gap[0] = N;
    int ans = 0;
    while(dep[start] < N)
    {
        if(u == end)
        {
            int Min = INF;
```

```
            for(int i = pre[u];i != -1; i = pre[edge[i^1].to])
            if(Min > edge[i].cap - edge[i].flow)
                Min = edge[i].cap - edge[i].flow;
            for(int i = pre[u];i != -1; i = pre[edge[i^1].to])
            {
                edge[i].flow += Min;
                edge[i^1].flow -= Min;
            }
            u = start;
            ans += Min;
            continue;
        }
        bool flag = false;
        int v;
        for(int i = cur[u]; i != -1;i = edge[i].next)
        {
            v = edge[i].to;
            if(edge[i].cap - edge[i].flow && dep[v]+1 == dep[u])
            {
                flag = true;
                cur[u] = pre[v] = i;
                break;
            }
        }
        if(flag)
        {
            u = v;
            continue;
        }
        int Min = N;
        for(int i = head[u]; i != -1;i = edge[i].next)
            if(edge[i].cap - edge[i].flow && dep[edge[i].to] < Min)
            {
                Min = dep[edge[i].to];
                cur[u] = i;
            }
        gap[dep[u]]--;
        if(!gap[dep[u]])return ans;
        dep[u] = Min+1;
        gap[dep[u]]++;
        if(u != start) u = edge[pre[u]^1].to;
    }
    return ans;
}

int main()
```

```
{
    for (int kk, kase = scanf("%d", &kk); kase <= kk; ++kase) {
        init();

    }
    return 0;
}
```

5.2.费用流

```
const int MAXN = 300;
const int MAXM = 1e5; // 边数最大值的两倍
const int INF = 0x3f3f3f3f;
struct Edge
{
    int to,next,cap,flow,cost;
}edge[MAXM];
int head[MAXN],tol;
int pre[MAXN],dis[MAXN];
bool vis[MAXN];
int N;//节点总个数，节点编号从 0~N-1
void init(int n)
{
    N = n;
    tol = 0;
    memset(head,-1,sizeof(head));
}
void addedge(int u,int v,int cap,int cost)
{
    edge[tol].to = v;
    edge[tol].cap = cap;
    edge[tol].cost = cost;
    edge[tol].flow = 0;
    edge[tol].next = head[u];
    head[u] = tol++;
    edge[tol].to = u;
    edge[tol].cap = 0;
    edge[tol].cost = -cost;
    edge[tol].flow = 0;
    edge[tol].next = head[v];
    head[v] = tol++;
}
```

```
bool spfa(int s,int t)
{
    queue<int>q;
    for(int i = 0;i < N;i++)
    {
        dis[i] = INF;
        vis[i] = false;
        pre[i] = -1;
    }
    dis[s] = 0;
    vis[s] = true;
    q.push(s);
    while(!q.empty())
    {
        int u = q.front();
        q.pop();
        vis[u] = false;
        for(int i = head[u]; i != -1;i = edge[i].next)
        {
            int v = edge[i].to;
            if(edge[i].cap > edge[i].flow && dis[v] > dis[u] + edge[i].cost )
            {
                dis[v] = dis[u] + edge[i].cost;
                pre[v] = i;
                if(!vis[v])
                {
                    vis[v] = true;
                    q.push(v);
                }
            }
        }
    }
    if(pre[t] == -1)return false;
    else return true;
}
//返回的是最大流， cost 存的是最小费用
int minCostMaxflow(int s,int t,int &cost)
{
    int flow = 0;
    cost = 0;
    while(spfa(s,t))
    {
        int Min = INF;
        for(int i = pre[t];i != -1;i = pre[edge[i^1].to])
        {
            if(Min > edge[i].cap - edge[i].flow)
```

```
            Min = edge[i].cap - edge[i].flow;
        }
        for(int i = pre[t];i != -1;i = pre[edge[i^1].to])
        {
            edge[i].flow += Min;
            edge[i^1].flow -= Min;
            cost += edge[i].cost * Min;
        }
        flow += Min;
    }
    return flow;
}
```

6.匹配

```
/*
    1） 一个二分图中的最大匹配数等于这个图中的最小点覆盖数
        König 定理是一个二分图中很重要的定理，它的意思是，一个二分图
    中的最大匹配数等于这个图中的最小点覆盖数。最小点覆盖：选择最少的
    点来覆盖所有的边。
    2） 最小路径覆盖＝|G|- 最大匹配数
        在一个 N*N 的有向图中，路径覆盖就是在图中找一些路经，使之覆盖
    了图中的所有顶点，且任何一个顶点有且只有一条路径与之关联;(如果把
    这些路径中的每条路径从它的起始点走到它的终点，那么恰好可以经过图中
    的每个顶点一次且仅一次);如果不考虑图中存在回路，那么每每条路径就是
    一个弱连通子集. 由上面可以得出:
        1.一个单独的顶点是一条路径;
        2.如果存在一路径 p1,p2,......pk，其中 p1 为起点，pk 为终点，那么在
    覆盖图中，顶点 p1,p2,......pk 不再与其它的顶点之间存在有向边.
        最小路径覆盖就是找出最小的路径条数，使之成为 G 的一个路径覆盖.
    3） 二分图最小边覆盖=最大独立集=顶点数-二分图最大匹配
        独立集：图中任意两个顶点都不相连的顶点集合。
 */
```

6.1.二分图匹配

```
/*
 * 原题：ZOJ 3646
 * 题意：任意交换两列或者两行，能否使得对角线都是 U
 */
const int INF = 0x7f7f7f7f, MAXN = 211;
int line[MAXN][MAXN], match[MAXN], n;
bool vis[MAXN];
```

```cpp
bool find(int x)
{
    for (int i = 0; i < n; ++i)
    {
        if (line[x][i] && !vis[i])
        {
            vis[i] = 1;
            if (match[i] == -1 || find(match[i]))
            {
                match[i] = x;
                return 1;
            }
        }
    }
    return 0;
}

int main()
{
    while (cin >> n)
    {
        char x;
        memset(line, 0, sizeof line);
        memset(match, -1, sizeof match);
        for (int i = 0; i < n; ++i)
            for (int j = 0; j < n; ++j)
            {
                cin >> x;
                if (x == 'U') line[i][j] = 1;
            }
        int ans = 0;

        for (int i = 0; i < n; ++i)
        {
            memset(vis, 0, sizeof vis);
            ans += find(i);
        }
        if (ans == n) cout << "YES\n";
        else cout << "NO\n";
    }
    return 0;
}
```

6.2.二分图多重匹配

```
/*
 * 多重匹配其实和二分差不多，也可用网络流求解
 * 本题求是否能够全部都能匹配上，所以只要一个不匹配即可返回
 */
const int MAXN = 100010;
const int MAXM = 12;
int n, m;
int g[MAXN][MAXM];
int linker[MAXM][MAXN];
bool used[MAXM];
int num[MAXM];//右边最大的匹配数
bool dfs(int u) {
    for(int v = 0; v < m;v++) {
        if(g[u][v] && !used[v]) {
            used[v] = true;
            if(linker[v][0] < num[v]) {
                linker[v][++linker[v][0]] = u;
                return true;
            }
            else {
                for(int i = 1;i <= num[v]; ++i) {
                    if(dfs(linker[v][i])){
                        linker[v][i] = u;
                        return true;
                    }
                }
            }
        }
    }
    return false;
}

int hungary() {
    int res = 0;
    for(int i = 0;i < m; ++i) linker[i][0] = 0;
    for(int u = 0; u < n; ++u) {
        memset(used,false,sizeof(used));
        if(dfs(u))res++;
        else return false;
    }
    return true;
}

int main()
```

```
{
    while (~scanf("%d%d", &n, &m)) {
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < m; ++j) {
                scanf("%d", &g[i][j]);
            }
        }
        for(int i = 0;i < m; ++i) scanf("%d", &num[i]);
        if(hungary())
            puts("YES");
        else
            puts("NO");
    }
    return 0;
}
```

6.3.二分图最大权匹配

```
/*
 * 二分图最大权匹配
 * 复杂度 n^3
 */
#include<cstdio>
#include<cstring>
#include<iostream>
using namespace std;

const int INF = 0x7f7f7f7f;
const int MAXN = 311;

int n, nx, ny;
int match[MAXN], lx[MAXN], ly[MAXN], slack[MAXN]; // lx[]左标杆 ly[]右标杆
slack[]松弛量
int visx[MAXN], visy[MAXN], w[MAXN][MAXN];

int dfs(int x)
{
    visx[x] = 1;
    for (int y = 1; y <= ny; ++y)
    {
        if (visy[y]) continue;
```

```
        int temp = lx[x] + ly[y] - w[x][y];
        if (temp == 0)
        {
            visy[y] = 1;
            if (match[y] == -1 || dfs(match[y]))
            {
                match[y] = x;
                return 1;
            }
        }
        else if (slack[y] > temp) slack[y] = temp;
    }
    return 0;
}

int KM()
{
    // 1.initialize
    int i, j;
    memset(match, -1, sizeof match);
    memset(ly, 0, sizeof ly);
    for (i = 1; i <= nx; ++i)
        for (j = 1, lx[i] = -INF; j <= ny; ++j)
            if (w[i][j] > lx[i]) lx[i] = w[i][j];

    // 2.find
    for (int x = 1; x <= nx; ++x)
    {
        for (i = 1; i <= ny; ++i) slack[i] = INF;
        while (1)
        {
            memset(visx, 0, sizeof visx);
            memset(visy, 0, sizeof visy);
            if (dfs(x)) break;

            // 能到达这一步，说明再也挪不到新边给 x 了，所以要进行增广，增加新边来选择
            int d = INF;
            // 没有匹配的 y，且两者有边相连
            for (i = 1; i <= ny; ++i) if (!visy[i] && d > slack[i])
                d = slack[i];
            for (i = 1; i <= nx; ++i) if (visx[i]) lx[i] -= d;
            for (i = 1; i <= ny; ++i)
            {
                if (visy[i]) ly[i] += d;
                else slack[i] -= d; // 更改松弛量
```

```
            }                       // 本来这些点是要松弛 D 的，结果已经松弛了 d，那么之后如果要松弛只用松弛 D - d 就行了
        }
    }

    int ret = 0;
    // 等于-1 说明匹配没有匹配完!
    for (i = 1; i <= ny; ++i)
        if (match[i] != -1) ret += w[match[i]][i];
    return ret;
}

int main()
{
    int n;
    while (~scanf("%d", &n))
    {
        nx = ny = n;
        for (int i = 1; i <= n; ++i)
            for (int j = 1; j <= n; ++j)
                scanf("%d", &w[i][j]);
        printf("%d\n", KM());
    }
    return 0;
}
```

6.4.最小路径覆盖

```
/*
 * 本题需要缩点，然后建图，二分匹配
 * 拆点后连边，如果 u->v 则连边 u->v'
 * 对于可重复经过点的最小路径覆盖，我们需要对最终的图进行一次传递闭包再匹配
 */

const int INF = 0x7f7f7f7f;
const int MAXN = 5e3 + 111;
const int MAXM = 1e5 + 111;

vector<int> G[MAXN];
int dfn[MAXN], low[MAXN], id[MAXN], sta[MAXN], in[MAXN];
int scc, top, indx;
int u[MAXM], v[MAXM];

void init(int n) {
    for (int i = 1; i <= n; ++i) {
        G[i].clear();
        dfn[i] = 0;
    }
    scc = top = indx = 0;
}

void tarjan(int u) {
    dfn[u] = low[u] = ++indx;
    sta[top++] = u;
    in[u] = 1;
    int v;
    for (int i = 0; i < G[u].size(); ++i) {
        v = G[u][i];
        if (!dfn[v]) {
            tarjan(v);
            low[u] = min(low[u], low[v]);
        }
        else if (in[v]) low[u] = min(low[u], dfn[v]);
    }

    if (dfn[u] == low[u]) {
        ++scc;
        do {
            v = sta[--top];
            in[v] = 0;
            id[v] = scc;
        } while (u != v);
```

```
    }
}

int match[MAXN * 2];
bool vis[MAXN * 2];
bool dfs(int u) {
    for (int i = 0; i < G[u].size(); ++i) {
        int v = G[u][i];
        if (!vis[v]) {
            vis[v] = 1;
            if (!match[v] || dfs(match[v])) {
                match[v] = u;
                return 1;
            }
        }
    }
    return 0;
}

int solve() {
    int ret = 0;
    memset(match, 0, sizeof match);
    for (int i = 1; i <= scc; ++i) {
        memset(vis, 0, sizeof vis);
        if (dfs(i)) ++ret;
    }
    return ret;
}

int main()
{
    int t, n, m; scanf("%d", &t);
    while (t --) {
        scanf("%d%d", &n, &m);
        init(n);
        for (int i = 0; i < m; ++i) {
            scanf("%d%d", &u[i], &v[i]);
            G[u[i]].push_back(v[i]);
        }
        for (int i = 1; i <= n; ++i) {
            if (!dfn[i]) tarjan(i);
        }
        int ans = n;
        for (int i = 1; i <= scc; ++i) G[i].clear();
        for (int i = 0; i < m; ++i) {
            int x = id[u[i]], y = id[v[i]];
```

```
            if (x == y) continue;
            G[x].push_back(y + scc);
        }
        printf("%d\n", scc - solve());
    }
    return 0;
}
```

6.5.婚姻匹配

```
/*
 * 带喜好的匹配
 * 女士心仪的男生有第一优先权
 */
const int INF = 0x7f7f7f7f;
const int MAXN = 210;

struct P {
    int cap, id;
    double x, y, z;
    P() {}
    P(int a, int b, double c) {
        id = a; cap = b; x = c;
    }
    void read() {
        scanf("%d%d%lf%lf%lf", &id, &cap, &x, &y, &z);
    }
    bool operator < (const P&t)const {
        if (x == t.x) return cap > t.cap;
        return x < t.x;
    }
}boy[MAXN], girl[MAXN];

int len[MAXN];
vector<P> glike[MAXN], blike[MAXN];
int match[MAXN], n;

double getdis(int a, int b) {
    P &t1 = boy[a], &t2 = girl[b];
    double dx = t1.x - t2.x;
    double dy = t1.y - t2.y;
    double dz = t1.z - t2.z;
    return sqrt(dx*dx + dy*dy + dz*dz);
```

```
}

// 不断给男士匹配女士，用 len 数组记录下标，已经被拒绝过的女士不可能再匹配了。
void GaleShapley(){
    queue<int> q;
    for (int i = 1; i <= n; ++i) {
        match[i] = -1;
        len[i] = 0;
        q.push(i);
    }
    while (q.size()) {
        int bid = q.front(); q.pop();
        for (int &i = len[bid]; i < n; ++i) {
            int gid = blike[bid][i].id;
            if (match[gid] == -1) {
                match[gid] = bid;
                break;
            }
            int j;
            for (j = 0; j < n; ++j) {
                if (glike[gid][j].id == bid || glike[gid][j].id == match[gid])
                    break;
            }
            if (glike[gid][j].id == bid) {
                q.push(match[gid]);
                match[gid] = bid;
                break;
            }
        }
    }
}

int main()
{
    for (int t, kk = scanf("%d", &t); kk <= t; ++kk) {
        scanf("%d", &n);
        for (int i = 1; i <= n; ++i) boy[i].read(), blike[i].clear();
        for (int i = 1; i <= n; ++i) girl[i].read(), glike[i].clear();
        for (int i = 1; i <= n; ++i) {
            for (int j = 1; j <= n; ++j) {
                double dis = getdis(i, j);
                blike[boy[i].id].push_back(P(girl[j].id, girl[j].cap, dis));
                glike[girl[j].id].push_back(P(boy[i].id, boy[i].cap, dis));
            }
        }
        for (int i = 1; i <= n; ++i) {
```

```
            sort(blike[i].begin(), blike[i].end());
            sort(glike[i].begin(), glike[i].end());
        }
        GaleShapley();
        for (int i = 1; i <= n; ++i) {
            printf("%d %d\n", match[i], i);
        }
        putchar('\n');
    }
    return 0;
}
```

7.可行性遍历

7.1.欧拉路

欧拉回路：每条边只经过一次，而且回到起点
欧拉路径：每条边只经过一次，不要求回到起点
欧拉回路判断：

无向图：连通（不考虑度为 0 的点），每个顶点度数都为偶数。

有向图：基图连通（把边当成无向边，同样不考虑度为 0 的点），每个顶点出度等于入度。

混合图（有无向边和有向边）：首先是基图连通（不考虑度为 0 的点），然后需要借助网络流判定。首先给原图中的每条无向边随便指定一个方向（称为初始定向），将原图改为有向图 G'，然后的任务就是改变 G'中某些边的方向（当然是无向边转化来的，原混合图中的有向边不能动）使其满足每个点的入度等于出度。

设 D[i]为 G'中(点 i 的出度 - 点 i 的入度)。 可以发现，在改变 G'中边的方向的过程中，任何点的 D 值的奇偶性都不会发生改变（设将边<i, j>改为<j, i>，则 i 入度加 1 出度减 1， j 入度减 1 出度加 1，两者之差加 2 或减 2，奇偶性不变）！而最终要求的是每个点的入度等于出度，即每个点的 D 值都为 0，是偶数，故可得：

若初始定向得到的 G'中任意一个点的 D 值是奇数，那么原图中一定不存在欧拉环！

若初始 D 值都是偶数，则将 G'改装成网络：设立源点 S 和汇点 T，对于每个 D[i]>0 的点 i，连边<S, i>，容量为 D[i]/2；对于每个 D[j]<0 的点 j，连边<j, T>，容量为-D[j]/2； G'中的每条边在网络中仍保留，容量为 1（表示该边最多只能被改变方向一次）。求这个网络的最大流，**若 S 引出的所有边均满流，则原混合图是欧拉图**，将网络中所有流量为 1 的中间边（就是不与 S 或 T 关联的边）在 G'中改变方向，形成的新图 G''一定是有向欧拉图；若 S 引出的边中有的没有满流，则原混合图不是欧拉图。

欧拉路径的判断：

无向图：连通（不考虑度为 0 的点），每个顶点度数都为偶数或者仅有两个点的度数为奇数。

有向图：基图连通（把边当成无向边，同样不考虑度为 0 的点），每个顶点出度等于入度 或者 有且仅有一个点的出度比入度多 1，有且仅有一个点的出度比入度少 1，其余出度等于入度。

混合图： 如果存在欧拉回路，一点存在欧拉路径了。否则如果有且仅有两个点的（出度-入度）是奇数，那么给这个两个点加边，判断是否存在欧拉回路

```cpp
/*
 * 原题：UVA 10441
 * 将首尾相同的单词连接起来成为一个串，输出这个串
 * 实质就是判定是否存在欧拉回路/通路，输出路径
 */
const int INF = 0x7f7f7f7f;
const int MAXN = 1e3 + 111;

struct Edge
{
    int to, nxt, id;
    bool flag;
}edge[MAXN * 2];

string s[MAXN];
int n, ans[MAXN], in[33], out[33], tot, head[33], cnt;

void init()
{
    tot = 0;
    memset(in, 0, sizeof in);
    memset(out, 0, sizeof out);
    memset(head, -1, sizeof head);
}

void add_edge(int u, int v, int id)
{
    edge[tot].to = v;
    edge[tot].flag = 0;
    edge[tot].id = id;
    edge[tot].nxt = head[u];
    head[u] = tot++;
}

void dfs(int u)
{
    for (int i = head[u]; ~i; i = edge[i].nxt)
    {
        if (!edge[i].flag)
        {
            edge[i].flag = 1;
            dfs(edge[i].to);
            ans[cnt++] = edge[i].id;
        }
    }
}
```

```
int main()
{
    ios_base::sync_with_stdio(0);
    int t; cin >> t;
    while (t --)
    {
        cin >> n;
        init();

        for (int i = 1; i <= n; ++i) cin >> s[i];
        sort(s + 1, s + n + 1);
        int st = 26;
        for (int i = n; i >= 1; --i) // 链式前向星是倒着遍历边，所以倒着来，最后就是顺着字典序了
        {
            int u = s[i][0] - 'a', v = s[i].back() - 'a';
            add_edge(u, v, i);
            ++in[v], ++out[u];
            st = min(st, min(u, v));  // 如果是回路，字典序最小的作为开头
        }

        int cnt1 = 0, cnt2 = 0;
        // 判断欧拉回路/欧拉通路
        for (int i = 0; i < 26; ++i)
        {
            if (out[i] - in[i] == 1)
            {
                ++cnt1;
                st = i;
            }
            else if (out[i] - in[i] == -1)
                ++cnt2;
            else if (out[i] - in[i] != 0)
                cnt1 = 3;
        }

        if (!(cnt1 == cnt2 && cnt1 <= 1))
        {
            cout << "***\n";
            continue;
        }

        cnt = 0;
        dfs(st);
        if (cnt != n)
```

```
        {
            cout << "***\n";
            continue;
        }

        for (int i = n - 1; i >= 0; --i)
        {
            cout << s[ans[i]];
            if (i > 0) cout << '.';
            else cout << '\n';
        }
    }
    return 0;
}
```

7.2.哈密顿

```
/*
 * 竞赛图寻找哈密顿回路
 * 竞赛图中是一定存在哈密顿路的，但不一定有回路
 */
const int MAXN = 1010;

int map[MAXN][MAXN], nxt[MAXN], ans[MAXN];

void Hamilton(int n, int st){
    memset(nxt, -1, sizeof(nxt));
    int head = st;
    for(int i = 1; i <= n; i++){
        if(i == st)continue;
        if(map[i][head]){
            nxt[i] = head;
            head = i;
        }else{
            int pre = head, pos = nxt[head];
            while(pos != -1 && !map[i][pos]){
                pre = pos;
                pos = nxt[pre];
            }
            nxt[pre] = i;
            nxt[i] = pos;
        }
    }
```

```
    int cnt = 0;
    for(int i = head; i != -1; i = nxt[i])
        ans[++cnt] = i;
}

int main()
{
    int N;
    while(~scanf("%d", &N) && N){
        memset(map, 0, sizeof map);
        memset(ans, 0, sizeof ans);
        for(int i = 1; i <= N; i++){
            for(int j = 1; j <= N; j++){
                int u;
                scanf("%d", &u);
                map[i][j] = u;
            }
        }
        if(N == 1){printf("1\n");continue;}
        int i;
        for(i = 1; i<= N; i++){
            Hamilton(N, i);
            if(map[ans[N]][ans[1]]){
                for(int j = 1; j <= N; j++){
                    printf(j == 1 ? "%d":" %d", ans[j]);
                }
                break;
            }
        }
        if(i > N)printf("-1");
        printf("\n");
    }
    return 0;
}
```

7.3.拓扑排序

```
const int MAXN = 1111;

int c[MAXN], G[MAXN][MAXN];
int topo[MAXN], t, n;
```

```
bool dfs(int u) {
    c[u] = -1; // 访问标记
    for (int v = 0; v < n; ++v) {
        if (G[u][v]) {
            if (c[v] < 0) return false;
            else if (!c[v] && !dfs(v)) return false;
        }
    }
    c[u] = 1;
    topo[--t] = u;
    return true;
}

bool toposort() {
    t = n;
    memset(c, 0, sizeof c);
    for (int u = 0; u < n; ++u) if (!c[u] && !dfs(u)) {
        return false;
    }
    return true;
}
```

四、树

1.树的最小表示

```
/*
 * 原题：POJ1635
 * 字符 1 表示向下走，0 表示向上走。
 * 树的最小表示:递归子树，然后给子树排序
 */

string mRept(string s) {
    string ret = "0";
    vector<string> temp;
    int cnt = 0, p = 0;
    for (int i = 0; i < s.length(); ++i) {
        if (s[i] == '0') ++cnt;
        else --cnt;
```

```
        if (cnt == 0) {
            temp.push_back(mRept(s.substr(p + 1, i - 1 - (p + 1) + 1)));
            p = i + 1;
        }
    }
    sort(temp.begin(), temp.end());
    for (int i = 0; i < temp.size(); ++i) ret += temp[i];
    return ret + "1";
}
```

2.扫描线

```
/*
 * 扫描线求重叠矩形周长
 * 分 x 和 y 进行计算
 */
const int INF = 0x7f7f7f7f;
const int MAXN = 1e4 + 111;

struct Seg {
    int l, r, h, d;
    bool operator < (const Seg&t)const {
        return h < t.h;
    }
}a[2][MAXN];
int all[2][MAXN], len[2];

int sum[MAXN << 3], cnt[MAXN << 3];

void push_up(int l, int r, int rt, int op) {
    if (cnt[rt]) sum[rt] = all[op][r + 1] - all[op][l];
    else if (l == r) sum[rt] = 0;
    else sum[rt] = sum[ls] + sum[rs];
}

void update(int L, int R, int l, int r, int rt, int val, int op) {
    if (L <= l && r <= R) {
        cnt[rt] += val;
        push_up(l, r, rt, op);
        return;
    }
    int mid = (l + r) >> 1;
    if (L <= mid) update(L, R, lson, val, op);
    if (mid < R) update(L, R, rson, val, op);
```

```
    push_up(l, r, rt, op);
    return;
}

int main()
{
    int n, x1, y1, x2, y2;
    while (~scanf("%d", &n)) {
        for (int i = 1; i <= n; ++i) {
            scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
            a[0][i] = Seg{x1, x2, y1, 1}, a[0][i + n] = Seg{x1, x2, y2, -1};
            a[1][i] = Seg{y1, y2, x1, 1}, a[1][i + n] = Seg{y1, y2, x2, -1};
            all[0][i] = x1, all[0][i + n] = x2;
            all[1][i] = y1, all[1][i + n] = y2;
        }
        n <<= 1;
        for (int i = 0; i < 2; ++i) {
            sort(a[i] + 1, a[i] + 1 + n);
            sort(all[i] + 1, all[i] + 1 + n);
            len[i] = unique(all[i] + 1, all[i] + 1 + n) - all[i] - 1;
        }
        int ans = 0;
        for (int i = 0; i < 2; ++i) {
            int last = 0;
            memset(sum, 0, sizeof sum);
            memset(cnt, 0, sizeof cnt);
            for (int j = 1; j <= n; ++j) {
                int l = lower_bound(all[i] + 1, all[i] + 1 + len[i], a[i][j].l) -
all[i];
                int r = lower_bound(all[i] + 1, all[i] + 1 + len[i], a[i][j].r) -
all[i];
                // 如果更新 l,r 那么中间值 mid,mid+1 这个区间会丢失
                update(l, r - 1, root, a[i][j].d, i);
                ans += abs(sum[1] - last);
                last = sum[1];
            }
        }
        printf("%d\n", ans);
    }
    return 0;
}
```

3.二维线段树

```
/*
  二维线段树：单点更新，区间求最大最小
  二维线段树可以想象成这个样子
  设 x 轴某颗线段树 id 为 ii，对应 x 轴区间[4,6]
  它对应的 y 轴线段树相当于把这个区间内的所有
  y 轴合并在一起变成一个长条(设 y 轴总长度为 3，[1,3])
   3 3 10      10
   4 9 2   --> 9
   1 6 8       8
  [4 5 6]      ii
  所以更新的时候，当更新的点在这个区域中时，我们就
  把它更新了。
 */
#define root 1, n, 1
#define ls rt << 1
#define rs rt << 1 | 1
#define lson l, mid, ls
#define rson mid + 1, r, rs
#define pr(x) cout << #x << " = " << (x) << " I ";
#define prln(x) cout << #x << " = " << (x) << '\n';
using namespace std;

const int INF = 0x7f7f7f7f;
const int MAXN = 808;
const double eps = 1e-6;

int ami, amx, n, mat[MAXN][MAXN];
int mi[MAXN << 3][MAXN << 3], mx[MAXN << 3][MAXN << 3];

void push_up(int xrt, int rt) {
    mx[xrt][rt] = max(mx[xrt][ls], mx[xrt][rs]);
    mi[xrt][rt] = min(mi[xrt][ls], mi[xrt][rs]);
}

void buildy(int xrt, int x, int l, int r, int rt) {
    if (l == r) {
        if (~x) {
            mi[xrt][rt] = mx[xrt][rt] = mat[x][l];
        }
        else {
            mi[xrt][rt] = min(mi[xrt << 1][rt], mi[xrt << 1|1][rt]);
            mx[xrt][rt] = max(mx[xrt << 1][rt], mx[xrt << 1|1][rt]);
        }
        return;
```

```
    }
    int mid = l + r >> 1;
    buildy(xrt, x, lson);
    buildy(xrt, x, rson);
    push_up(xrt, rt); // 类似 push up
}

void buildx(int l, int r, int rt) {
    if (l == r) {
        buildy(rt, l, root);
        return;
    }
    int mid = l + r >> 1;
    buildx(lson);
    buildx(rson);
    buildy(rt, -1, root);
}

void updatey(int xrt, int op, int y, int val, int l, int r, int rt) {
    if (l == r) {
        if (~op) {
            mi[xrt][rt] = mx[xrt][rt] = val;
        }
        else {
            mi[xrt][rt] = min(mi[xrt << 1][rt], mi[xrt << 1|1][rt]);
            mx[xrt][rt] = max(mx[xrt << 1][rt], mx[xrt << 1|1][rt]);
        }
        return;
    }
    int mid = l + r >> 1;
    if (y <= mid) updatey(xrt, op, y, val, lson);
    else updatey(xrt, op, y, val, rson);
    push_up(xrt, rt);
}

void updatex(int x, int y, int val, int l, int r, int rt) {
    if (l == r) {
        updatey(rt, 1, y, val, root);
        return;
    }
    int mid = l + r >> 1;
    if (x <= mid) updatex(x, y, val, lson);
    else updatex(x, y, val, rson);
    updatey(rt, -1, y, val, root);
}
```

```
void queryy(int xrt, int y1, int y2, int l, int r, int rt) {
    if (y1 <= l && r <= y2) {
        ami = min(ami, mi[xrt][rt]);
        amx = max(amx, mx[xrt][rt]);
        return;
    }
    int mid = l + r >> 1;
    if (y1 <= mid) queryy(xrt, y1, y2, lson);
    if (mid < y2) queryy(xrt, y1, y2, rson);
}

void queryx(int x1, int x2, int y1, int y2, int l, int r, int rt) {
    if (x1 <= l && r <= x2) {
        queryy(rt, y1, y2, root);
        return;
    }
    int mid = l + r >> 1;
    if (x1 <= mid) queryx(x1, x2, y1, y2, lson);
    if (mid < x2) queryx(x1, x2, y1, y2, rson);
}

int main()
{
    int kase = 0, t, x, q, y, l; scanf("%d", &t);
    while (t --) {
        scanf("%d", &n);
        for (int i = 1; i <= n; ++i) {
            for (int j = 1; j <= n; ++j) {
                scanf("%d", &mat[i][j]);
            }
        }
        buildx(root);
        scanf("%d", &q);
        printf("Case #%d:\n", ++kase);
        while (q --) {
            scanf("%d%d%d", &x, &y, &l);
            l = (l - 1) / 2;
            ami = INF, amx = 0;
            queryx(max(1, x - l), min(n, x + l), max(1, y - l), min(n, y + l),
root);
            int nc = (ami + amx) / 2;
            printf("%d\n", nc);
            updatex(x, y, nc, root);
        }
    }
    return 0;}
```

4.二维 BIT

```
// 单点更新区间求和
#define lowbit(x) x & (-x)
using namespace std;

const int INF = 0x7f7f7f7f;
const int MAXN = 1e3 + 111;

int n = 1024;
int bit[MAXN][MAXN];

inline void update(int x, int y, int val) {
    for (int i = x; i <= n; i += lowbit(i)) {
        for (int j = y; j <= n; j += lowbit(j))
            bit[i][j] += val;
    }
}

int query(int x, int y) {
    int ret = 0;
    for (int i = x; i > 0; i -= lowbit(i)) {
        for (int j = y; j > 0; j -= lowbit(j))
            ret += bit[i][j];
    }
    return ret;
}

int main()
{
    int op, x, y, val, x1, y1, x2, y2;
    while (~scanf("%d", &op) && op != 3) {
        if (op == 0) {
            scanf("%d", &op);
            memset(bit, 0, sizeof bit);
        }
        else if (op == 1) {
            scanf("%d%d%d", &x, &y, &val);
            ++x, ++y;
            update(x, y, val);
        }
        else {
            scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
            ++x1, ++y1, ++x2, ++y2;
            printf("%d\n", query(x2, y2) - query(x1 - 1, y2) - query(x2, y1 - 1)
+ query(x1 - 1, y1 - 1));
```

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

5.树分治

```
/*
 * 统计树上距离小于 k 的点对的个数
 */
using namespace std;

const int INF = 0x7f7f7f7f;
const int MAXN = 1e4 + 111;

struct Edge {
    int to, next, w;
}edge[2 * MAXN];
int head[MAXN], tot;

int n, k, ans;
int subSize[MAXN];
bool vis[MAXN];

void init() {
    ans = tot = 0;
    memset(vis, 0, sizeof vis);
    memset(head, -1, sizeof head);
}

void add_edge(int u, int v, int w) {
    edge[tot].to = v;
    edge[tot].w = w;
    edge[tot].next = head[u];
    head[u] = tot++;
}

int compute_subSize(int u, int p) {
    int c = 1;
    for (int i = head[u]; ~i; i = edge[i].next) {
        int v = edge[i].to;
        if (v != p && !vis[v]) c += compute_subSize(v, u);
    }
    return subSize[u] = c;
}
```

```
// 寻找删除该点后最大子树的顶点数最少的点
pii search_it(int u, int p, int t) {
    pii ret = make_pair(INF, -1);
    int s = 1, m = 0;
    for (int i = head[u]; ~i; i = edge[i].next) {
        int v = edge[i].to;
        if (v != p && !vis[v]) {
            ret = min(ret, search_it(v, u, t));
            m = max(m, subSize[v]);
            s += subSize[v];
        }
    }
    m = max(m, t - s); // t - s:包含 u 点的子树顶点个数
    return min(ret, make_pair(m, u));
}

void getdis(int u, int p, int d, vector<int> &ds) {
    ds.push_back(d);
    for (int i = head[u]; ~i; i = edge[i].next) {
        int v = edge[i].to;
        if (v != p && !vis[v]) getdis(v, u, d + edge[i].w, ds);
    }
}

int countAns(vector<int> &ds) {
    int ret = 0;
    sort(ds.begin(), ds.end());
    int i = 0, j = ds.size() - 1;
    while (i < j) {
        while (ds[i] + ds[j] > k && i < j) --j;
        ret += j - i;
        ++i;
    }
    return ret;
}

void solve(int v) {
    // 找重心
    compute_subSize(v, -1);
    int s = search_it(v, -1, subSize[v]).second;
    vis[s] = 1;

    // 继续划分
    for (int i = head[s]; ~i; i = edge[i].next) {
        if (!vis[edge[i].to]) solve(edge[i].to);
```

```
    }

    // 计算个数
    vector<int> ds;
    ds.push_back(0); // 距离 s 为 0 的虚拟点
    for (int i = head[s]; ~i; i = edge[i].next) {
        int v = edge[i].to;
        if (vis[v]) continue;

        vector<int> tds;
        getdis(v, s, edge[i].w, tds);

        // 减去同一侧的对数
        ans -= countAns(tds);
        ds.insert(ds.end(), tds.begin(), tds.end());
    }

    ans += countAns(ds);
    vis[s] = 0; // 复原，为大子树计算服务 2333
}

int main()
{
    while (~scanf("%d%d", &n, &k) && (n | k)) {
        init();
        int u, v, l;
        for (int i = 1; i < n; ++i) {
            scanf("%d%d%d", &u, &v, &l);
            add_edge(u, v, l);
            add_edge(v, u, l);
        }
        solve(1);
        printf("%d\n", ans);
    }
    return 0;
}
```

五、其它

1.java 大整数

```
import java.util.Scanner;
import java.math.BigInteger;

public class Main {
    public static void main(String[] args) {
    Scanner cin = new Scanner(System.in);
    BigInteger p[] = new BigInteger[60];
    BigInteger ok[] = new BigInteger[60];
    BigInteger nok[] = new BigInteger[60];
    BigInteger tol[] = new BigInteger[60];
    BigInteger C[][] = new BigInteger[60][60];
    p[0] = BigInteger.valueOf(1); // 使用整数给大整数赋值的用法
    BigInteger two = new BigInteger("2");
    for (int i = 1; i <= 50; ++i) p[i] = p[i - 1].multiply(two);
    for (int i = 0; i <= 50; ++i) {
        nok[i] = BigInteger.ZERO;
        C[i][0] = C[i][i] = BigInteger.ONE;
        for (int j = 1; j < i; ++j) {
            C[i][j] = C[i - 1][j - 1].add(C[i - 1][j]);
        }
    }

    tol[1] = BigInteger.ONE;
    ok[1] = BigInteger.ONE;
    for (int i = 2; i <= 50; ++i) {
        for (int j = 1; j < i; ++j) {
            nok[i] = nok[i].add((C[i - 1][j - 1].multiply(ok[j])).multiply(tol[i -
j]));
        }
        tol[i] = tol[i - 1].multiply(p[i - 1]);
        ok[i] = tol[i].subtract(nok[i]);
    }
    while (cin.hasNext()) {
        int n = cin.nextInt();
        if (n == 0) break;
        System.out.println(ok[n]);
    }
    cin.close();
    }
}
```

2.fastIO

```
struct FastIO {
  static const int S = 1310720;
  int wpos; char wbuf[S];
  FastIO() : wpos(0) {}
  inline int xchar() { // 读入字符
    static char buf[S];
    static int len = 0, pos = 0;
    if (pos == len)
      pos = 0, len = fread(buf, 1, S, stdin);
    if (pos == len) return -1;
    return buf[pos ++];
  }
  inline int xuint() { // 读入 unsigned int
    int c = xchar(), x = 0;
    while (c <= 32) c = xchar();
    for (;'0' <= c && c <= '9'; c = xchar()) x = x * 10 + c - '0';
    return x;
  }
  inline int xint() { // 读入 int
    int s = 1, c = xchar(), x = 0;
    while (c <= 32) c = xchar();
    if (c == '-') s = -1, c = xchar();
    for (; '0' <= c && c <= '9'; c = xchar()) x = x * 10 + c - '0';
    return x * s;
  }
  inline void xstring(char *s) { // 读入 string
    int c = xchar();
    while (c <= 32) c = xchar();
    for(; c > 32; c = xchar()) *s++ = c;
    *s = 0;
  }
  inline void wchar(int x) { // 输出
    if (wpos == S) fwrite(wbuf, 1, S, stdout), wpos = 0;
    wbuf[wpos ++] = x;
  }
  inline void wint(ll x) {
    if (x < 0) wchar('-'), x = -x;

    char s[24];
    int n = 0;
    while (x || !n) s[n ++] = '0' + x % 10, x /= 10;
    while (n--) wchar(s[n]);
  }
  inline void wstring(const char *s) {
```

```
    while (*s) wchar(*s++);
  }
  ~FastIO() {
    if (wpos) fwrite(wbuf, 1, wpos, stdout), wpos = 0;
  }
} io;
```

六、一些笔记

1.判定最小生成树是否唯一

1)删除拥有相同权值的树边，重新求一次权值和，看是否相同。

2)求次小生成树，看权值是否和最小生成树一样。Max[i][j]记录 MST 上点 i 到 j 的最大边权。每次添加新边到 MST，必定构成环，删除环上权值最大的边，计算 MST 值。全部步骤后，即可知道次小生成树值

2.判断最小割是否唯一

如果源点能够到达点的个数 + 汇点能够到达点的个数 = 所有点个数。则最小割唯一。

证明：假设 S 沿着不满流的边不可到达点 V，则说明 S->V 的可行边满流，假设 T 沿着不满流的边不可到达点 V，则说明 T->V 的可行边满流，这样 V 点可以属于 S 集也能属于 T 集，最小割方案不唯一。

3.流量有上下界的网络的最大流（无源汇）

以前写的最大流默认的下界为 0，而这里的下界却不为 0，所以我们要进行再构造让每条边的下界为 0，这样做是为了方便处理。对于每根管子有一个上界容量 up 和一个下界容量 low，我们让这根管子的容量下界变为 0，上界为 up-low。可是这样做了的话流量就不守恒了，为了再次满足流量守恒，即每个节点"入流=出流”，我们增设一个超级源点 st 和一个超级终点 sd。我们开设一个数组 du[]来记录每个节点的流量情况。

du[i]=in[i]（i 节点所有入流下界之和）-out[i]（i 节点所有出流下界之和）。

当 du[i]大于 0 的时候，st 到 i 连一条流量为 du[i]的边。

当 du[i]小于 0 的时候，i 到 sd 连一条流量为-du[i]的边。

一次最大流，判断是否满流，满流答案就是 f + b（每条边的流量+每条边的下界）

4.（有源汇）

同上建图。

增设一条从 d->s 没有下界容量为无穷的边对（sd，st）进行一次最大流，当 maxflow 等于所有(du[]>0)之和时，有可行流，否则没有。

当有可行流时，删除超级源点 sd 和超级终点 st（其实不用删除，反正第二次流量也流不到这边），再对（s，d）进行一次最大流，此时得到的 maxflow 则为题目的解。

5.流量有上下界的网络的最小流量（有源汇）

建图同上。

先不增加 d->s 容量为无穷的边，进行一次 maxflow（），加一条（d，s）容量为无穷的边，再进行一次 maxflow（），当且仅当所有附加弧满载时，有可行解，解为 flow[（d->s）^1]（即 d 到 s 的后悔边权值）。

6.设 A、 B 是无向连通图 G 的两个不相邻的顶点，最少要删除多少个顶点才能使得 A 和 B 不再连通？

答案是 P(A, B)<A 到 B 的强独立轨的最大条数>个

6.1 无向图 G 的顶点连通度（删除多少个点后图不连通）

完全图为 V - 1, 其它图为 min<AB 不相连>P(A,B) 废话么= = 变成网络流求解。1.拆点，容量 1；2. 2. 任意两点 u'',v' v'', u' 容量无穷；3.求最大流

7.设 A、 B 是无向连通图 G 的两个不相邻的顶点，最少要删除多少条边才能使得 A 和 B 不再连通？

答案是 P'(A, B)<A 到 B 的弱独立轨的最大条数>个

7.1 无向图 G 的边连通度（删除多少条边后图不连通）

1) 原图 G 中的每条边 e = (u, v)变成重边， 再将这两条边加上互为反向的方向， 设 e'为<u, v>，e''为<v, u>， e'和 e''的容量均为 1；

2) 以 A 为源点， B 为汇点。

8.二维向量 p1 = (x1, y1), p2 = (x2, y2)，内积：p1*p2 = x1x2 + y1y2；外积：p1Xp2 = x1y2 - x2y1；

8.1 判断点是否在线段上

外积判断是否在一条直线（等于 0），内积判断是否在线段上（小于 0）

9.判断增加哪条边能增加网络中的流量

最大流，然后从 s 开始 dfs 标记 s 能到的点，从 t 开始沿着逆向边 dfs 标记 t 能到的点，如果 s->u v->t && u->v 是边，则这条边就是能加大的。

10.

点支配：点覆盖点

点覆盖：点覆盖边

点独立：谁都不碰谁

边覆盖：边覆盖点

边独立：谁都不抢谁的点，匹配哦~

最小路径覆盖：有向图。V - 最大匹配数。（如果是可重复点的最小路径覆盖，需要做一次传递闭包）1->2->3; 4->2->5;

最大团：图中点数最多的完全子图。

最小边覆盖：二分图。V - 最大匹配数

最小顶点覆盖：等于最大匹配数

最大独立集：V - 最小顶点覆盖

最小割 = 最小点权覆盖集 = 点权和 - 最大点权独立集

11.n - m + r = 2 欧拉公式（点-边+面=2）

12.最大密度子图

h（g）= |E'|-g*|V'|

二分密度值 g。

h（g）= （U*n-mincut）/2

源点与所有点容量 U（足够大，使得 U+2g-dv 不为负数。无点权，取 m，有边权取边权和，有点边权，取两者总和)

汇点与所有点容量 U + 2g - dv(dv 是点的度数)<有边权这里代表 dv 点相连的边权和>

Ps：如果有边权+点权，上方式子为 U+2(g-p(点权))-dv(边权和)

点与点间，建立容量 1 的两条边<有边权，这里容量该位边权>

13.线段树 （模多少的和）

离散化坐标，记录每个膜的和，记录偏移的个数，显然，更新某点时，左子树偏移不变，右子树偏移变了。

14.最大闭权子图

A 依赖 B，A->B 无穷容量，点 A 盈利，S->A 盈利量，A 花费，A->T 花费量

15.什么是代码能力？

简单地说，如果你觉得学会了某一个算法或者数据结构，你能否在半个小时（或一小时）内完成程序，并且经过很少的调试一次通过？如果做不到的话，只能说你还没彻底掌握这个算法或数据结构，要不断地训练，直到达到要求，只有这样才能保证在高度紧张的比赛中敢于实现而不犯错误。

16.贪心算法的正确性

要证明一个贪心算法是正确的，需要证明我们可以把一个最优解逐步转化为我们用贪心算法所得到的解，而解不会更差，从而证明贪心算法得到的解和最优解是一样好的（显然，最优解不可能更好)。而要证明一个贪心算法是错误的，只需要找到一个反例就可以了。

七、DEBUG 心得

程序逻辑对了吗？
特殊样例过了吗？
数组开够大了吗？
变量爆 int 了吗？
取模取对了吗？
0 做除数了吗？