ACM/ICPC Template Library

Beijing Jiao Tong University
Math_Magic Zhou Xinming
2014.10

一．配置环境 1
1. 头文件 1
2. 输入输出挂 2
二．杂项 3
1. C++大整数类 3
2. Manacher 7
3.RMQ 8
一维 8
二维 9
4.LCA 9
三．图论 11
1.最短路 11
1.1 Dijstra 11
1.2 最小环 11
1.3 最短路次短路计数 12
1.4 K 短路 13
2.最小生成树 14
2.1 Prim 14
2.2 最小树形图 14
2.3 度限制最小生成树 15
2.4 最优比例生成树 19
2.5 生成树计数 19
3.拓扑排序 21
4.连通性 23
4.1 SCC 23
4.2 点双联通分量 24
4.3 边双联通分量 26
4.4 无向图全局最小割 27
4.5 2-SAT 28
5.匹配 32
5.1 最大匹配 匈牙利 32
5.2 一般图最大匹配——带花树 33
5.3 最大权匹配——KM 35
6.欧拉回路 36
7.网络流 36
7.1 最大流 36
7.2 最小费用流 38
四．博弈 43
1 翻硬币游戏 43
2 anti-nim 游戏 43
3 every-SG 游戏 43
4 删边游戏 43
5 Ferguson 博弈 44
6 staircase nim 44
7 N 阶 Nim 游戏 44
8 Nim 积 44

一. 配置环境

- **codeblocks　在 codeblocks --> setting -->环境变量**

  terminal：　gnome-terminal -t $TITLE -x

- **eclipse**

Eclipse 中默认是输入"."后出现自动提示，用于类成员的自动提示，可是有时候我们希望它能在我们输入类的首字母后就出现自动提示，可以节省大量的输入时间（虽然按 alt + /会出现提示，但还是要多按一次按键，太麻烦了）。　从 Window -> preferences -> Java -> Editor -> Content assist -> Auto-Activation 下，我们可以在"."号后面加入我们需要自动提示的首字幕，比如"ahiz"。　然后我们回到 Eclipse 的开发环境，输入"a"，提示就出现了。但是我们可以发现，这个 Auto-Activation 下的输入框里最多只能输入 5 个字母，也许是 Eclipse 的开发人员担心我们输入的太多会影响性能，但计算机的性能不用白不用，所以我们要打破这个限制。其实上面都是铺垫，制造一下气氛，以显得我们下面要做的事情很牛似的，其实不然，一切都很简单。嘿嘿 :)

在"."后面随便输入几个字符，比如"abij"，然后回到开发环境，File -> export -> general -> preferences -> 选一个地方保存你的首选项，比如 C:"a.epf 用任何文本编辑器打开 a.epf，查找字符串“abij”，找到以后，替换成“abcdefghijklmnopqrstuvwxyz”，总之就是你想怎样就怎样！!然后回到 Eclipse，File -> import -> general -> preferences -> 导入刚才的 a.epf 文件。此时你会发现输入任何字幕都可以得到自动提示了。爽!!! 最后：自动提示弹出的时间最好改成 100 毫秒以下，这样会比较爽一点，不然你都完事了，自动提示才弹出来:)，不过也要看机器性能。

```
FileWriter fileWrite
r=new FileWriter("c:\\Result.txt");
    int [] a=new int[]{11112,222,333,444,555,666};
    for (int i = 0; i < a.length; i++) {
        fileWriter.write(String.valueOf(a[i])+" "
);
```

1.头文件

```
#include <algorithm>
#include <bitset>
#include <cctype>
#include <cerrno>
#include <clocale>
#include <cmath>
#include <complex>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <ctime>
#include <deque>
#include <exception>
#include <fstream>
#include <functional>
#include <limits>
#include <list>
#include <map>
#include <iomanip>
```

```
#include <ios>
#include <iosfwd>
#include <iostream>
#include <istream>
#include <ostream>
#include <queue>
#include <set>
#include <sstream>
#include <stack>
#include <stdexcept>
#include <streambuf>
#include <string>
#include <utility>
#include <vector>
#include <cwchar>
#include <cwctype>

using namespace std;
#define outstars cout << "**********************" <<
endl;
#define clr(a,b) memset(a,b,sizeof(a))
#define mk make_pair
#define pb push_back
#define sz size()
#define AA first
#define BB second
#define eps 1e-10
#define zero(x) fabs(x) < eps
#define deq(a , b) fabs(a - b) < eps

const int MAXN = 40000 + 50;
const int MAXE = 4000 + 50;
const int MAXQ = 1000000 + 50;
```

```
const int inf = 0x3f3f3f3f;
const int INF = ~0U >> 1;
const long long LLinf = 0x3FFFFFFFFFFFFFFFLL;
const long long LLINF = (1LL << 63) - 1;
const int IMIN = 0x80000000;

const int mod = 10007;
const long long MOD = 1000000000 + 7;
const double PI = acos(-1.0);

typedef long long LL;
typedef pair<int , int> pii;
typedef vector<int> vec;
///#pragma comment(linker,
"/STACK:102400000,102400000")
```

2. 输入输出挂

```
int Scan()
{
    int flag = 1;
    char ch;
    int a = 0;
    while((ch = getchar()) == ' ' || ch == '\n');
    if(ch == '-') flag = -1;
    else
        a += ch - '0';
    while((ch = getchar()) != ' ' && ch != '\n') {
        a *= 10;
        a += ch - '0';
 }
    return flag * a;
}
```

```
void Out(int a)
{
    if(a < 0) {
        putchar('-');
        a = -a;
    }
    if(a >= 10)Out(a / 10);
        putchar(a % 10 + '0');
}
```

二. 杂项

1. C++大整数类

```
#define MAXN 9999
#define MAXSIZE 10
#define DLEN 4
class BigNum
{
private:
    int a[500];    //可以控制大数的位数
    int len;       //大数长度
public:
    BigNum()
    {
        len = 1;    //构造函数
        memset(a,0,sizeof(a));
    }
    BigNum(const int);       //将一个 int 类型的变量转化为大数
    BigNum(const char*);     //将一个字符串类型的变量转化为大数
    BigNum(const BigNum &);  //拷贝构造函数
    BigNum &operator=(const BigNum &);   //重载赋值运算符，大数之间进行赋值运算

    friend istream& operator>>(istream&,  BigNum&);   //重载输入运算符
    friend ostream& operator<<(ostream&,  BigNum&);   //重载输出运算符

    BigNum operator+(const BigNum &) const;   //重载加法运算符，两个大数之间的相加运算
    BigNum operator-(const BigNum &) const;   //重载减法运算符，两个大数之间的相减运算
    BigNum operator*(const BigNum &) const;   //重载乘法运算符，两个大数之间的相乘运算
    BigNum operator/(const int   &) const;    //重载除法运算符，大数对一个整数进行相除运算

    BigNum operator^(const int  &) const;    //大数的 n 次方运算
    int    operator%(const int  &) const;    //大数对一个 int 类型的变量进行取模运算
    bool   operator>(const BigNum & T)const;   //大数和另一个大数的大小比较
    bool   operator>(const int & t)const;      //大数和一个 int 类型的变量的大小比较

    void print();       //输出大数
};
```

```
BigNum::BigNum(const int b)     //将一个 int 类型的变量转化为大数
{
    int c,d = b;
    len = 0;
    memset(a,0,sizeof(a));
    while(d > MAXN) {
        c = d - (d / (MAXN + 1)) * (MAXN + 1);
        d = d / (MAXN + 1);
        a[len++] = c;
    }
    a[len++] = d;
}
BigNum::BigNum(const char*s)     //将一个字符串类型的变量转化为大数
{
    int t,k,index,l,i;
    memset(a,0,sizeof(a));
    l=strlen(s);
    len=l/DLEN;
    if(l%DLEN)
        len++;
    index=0;
    for(i=l-1; i>=0; i-=DLEN) {
        t=0;
        k=i-DLEN+1;
        if(k<0)
            k=0;
        for(int j=k; j<=i; j++)
            t=t*10+s[j]-'0';
        a[index++]=t;
    }
}
```

```
BigNum::BigNum(const BigNum & T) : len(T.len)  //拷贝构造函数
{
    int i;
    memset(a,0,sizeof(a));
    for(i = 0 ; i < len ; i++)
        a[i] = T.a[i];
}
BigNum & BigNum::operator=(const BigNum & n)   //重载赋值运算符，大数之间进行赋值运算
{
    int i;
    len = n.len;
    memset(a,0,sizeof(a));
    for(i = 0 ; i < len ; i++)
        a[i] = n.a[i];
    return *this;
}
istream& operator>>(istream & in,  BigNum & b)   //重载输入运算符
{
    char ch[MAXSIZE*4];
    int i = -1;
    in>>ch;
    int l=strlen(ch);
    int count=0,sum=0;
    for(i=l-1; i>=0;) {
        sum = 0;
        int t=1;
        for(int j=0; j<4&&i>=0; j++,i--,t*=10) {
            sum+=(ch[i]-'0')*t;
        }
        b.a[count]=sum;
        count++;
```

```
    }
    b.len =count++;
    return in;

}
ostream& operator<<(ostream& out,  BigNum& b)   //重载输出运算符
{
    int i;
    cout << b.a[b.len - 1];
    for(i = b.len - 2 ; i >= 0 ; i--) {
        cout.width(DLEN);
        cout.fill('0');
        cout << b.a[i];
    }
    return out;
}

BigNum BigNum::operator+(const BigNum & T) const   //两个大数之间的相加运算
{
    BigNum t(*this);
    int i,big;      //位数
    big = T.len > len ? T.len : len;
    for(i = 0 ; i < big ; i++) {
        t.a[i] +=T.a[i];
        if(t.a[i] > MAXN) {
            t.a[i + 1]++;
            t.a[i] -=MAXN+1;
        }
    }
    if(t.a[big] != 0)
        t.len = big + 1;
    else
        t.len = big;
    return t;
}
BigNum BigNum::operator-(const BigNum & T) const   //两个大数之间的相减运算
{
    int i,j,big;
    bool flag;
    BigNum t1,t2;
    if(*this>T) {
        t1=*this;
        t2=T;
        flag=0;
    } else {
        t1=T;
        t2=*this;
        flag=1;
    }
    big=t1.len;
    for(i = 0 ; i < big ; i++) {
        if(t1.a[i] < t2.a[i]) {
            j = i + 1;
            while(t1.a[j] == 0)
                j++;
            t1.a[j--]--;
            while(j > i)
                t1.a[j--] += MAXN;
            t1.a[i] += MAXN + 1 - t2.a[i];
        } else
            t1.a[i] -= t2.a[i];
    }
    t1.len = big;
    while(t1.a[len - 1] == 0 && t1.len > 1) {
        t1.len--;
```

```
        big--;
    }
    if(flag)
        t1.a[big-1]=0-t1.a[big-1];
    return t1;
}

BigNum BigNum::operator*(const BigNum & T) const   //两个大数之间的相乘运算
{
    BigNum ret;
    int i,j,up;
    int temp,temp1;
    for(i = 0 ; i < len ; i++) {
        up = 0;
        for(j = 0 ; j < T.len ; j++) {
            temp = a[i] * T.a[j] + ret.a[i + j] + up;
            if(temp > MAXN) {
                temp1 = temp - temp / (MAXN + 1) * (MAXN + 1);
                up = temp / (MAXN + 1);
                ret.a[i + j] = temp1;
            } else {
                up = 0;
                ret.a[i + j] = temp;
            }
        }
        if(up != 0)
            ret.a[i + j] = up;
    }
    ret.len = i + j;
    while(ret.a[ret.len - 1] == 0 && ret.len > 1)
        ret.len--;
    return ret;
}
BigNum BigNum::operator/(const int & b) const   //大数对一个整数进行相除运算
{
    BigNum ret;
    int i,down = 0;
    for(i = len - 1 ; i >= 0 ; i--) {
        ret.a[i] = (a[i] + down * (MAXN + 1)) / b;
        down = a[i] + down * (MAXN + 1) - ret.a[i] * b;
    }
    ret.len = len;
    while(ret.a[ret.len - 1] == 0 && ret.len > 1)
        ret.len--;
    return ret;
}
int BigNum::operator %(const int & b) const    //大数对一个 int 类型的变量进行取模运算
{
    int i,d=0;
    for (i = len-1; i>=0; i--) {
        d = ((d * (MAXN+1))% b + a[i])% b;
    }
    return d;
}
BigNum BigNum::operator^(const int & n) const    //大数的 n 次方运算
{
    BigNum t,ret(1);
    int i;
    if(n<0)
        exit(-1);
    if(n==0)
        return 1;
```

```
    if(n==1)
        return *this;
    int m=n;
    while(m>1) {
        t=*this;
        for( i=1; i<<1<=m; i<<=1) {
            t=t*t;
        }
        m-=i;
        ret=ret*t;
        if(m==1)
            ret=ret*(*this);
    }
    return ret;
}
bool BigNum::operator>(const BigNum & T) const   //大数和另一个大数的大小比较
{
    int ln;
    if(len > T.len)
        return true;
    else if(len == T.len) {
        ln = len - 1;
        while(a[ln] == T.a[ln] && ln >= 0)
            ln--;
        if(ln >= 0 && a[ln] > T.a[ln])
            return true;
        else
            return false;
    } else
        return false;
}
bool BigNum::operator >(const int & t) const    //大数和一个 int 类型的变量的大小比较
{
    BigNum b(t);
    return *this>b;
}

void BigNum::print()    //输出大数
{
    int i;
    cout << a[len - 1];
    for(i = len - 2 ; i >= 0 ; i--) {
        cout.width(DLEN);
        cout.fill('0');
        cout << a[i];
    }
    cout << endl;
}
int main(void)
{
    int i,n;
    BigNum x[101];      //定义大数的对象数组
    x[0]=1;
    for(i=1; i<101; i++)
        x[i]=x[i-1]*(4*i-2)/(i+1);
    while(scanf("%d",&n)==1 && n!=-1) {
        x[n].print();
    }
}
```

2. Manacher

算法大致过程是这样。先在每两个相邻字符中间插入一个分隔符，当然这个分隔符要在原串中没有出现过。一般可以用‘#’分隔。这样就非常巧妙

的将奇数长度回文串与偶数长度回文串统一起来考虑了（见下面的一个例子，回文串长度全为奇数了），然后用一个辅助数组 P 记录以每个字符为中心的最长回文串的信息。P［id］记录的是以字符 str［id］为中心的最长回文串，当以 str［id］为第一个字符，这个最长回文串向右延伸了 P［id］个字符。

原串：　　w aa bwsw f d

新串：　　# w# a # a # b# w # s # w # f # d #

辅助数组 P：　1 2 1 2 3 2 1 2 1 2 1 4 1 2 1 2 1 2 1

这里有一个很好的性质，P[id]-1 就是该回文子串在原串中的长度（包括‘#’）。如果这里不是特别清楚，可以自己拿出纸来画一画，自己体会体会。当然这里可能每个人写法不尽相同，不过我想大致思路应该是一样的吧。

好，我们继续。现在的关键问题就在于怎么在 O（n）时间复杂度内求出 P 数组了。只要把这个 P 数组求出来，最长回文子串就可以直接扫一遍得出来了。

由于这个算法是线性从前往后扫的。那么当我们准备求 P[i]的时候，i 以前的 P[j]我们是已经得到了的。我们用 mx 记在 i 之前的回文串中，延伸至最右端的位置。同时用 id 这个变量记下取得这个最优 mx 时的 id 值。（注：为了防止字符比较的时候越界，我在这个加了‘#’的字符串之前还加了另一个特殊字符‘$’，故我的新串下标是从 1 开始的）

```
void pk()
{
    int i;
    int mx = 0;
    int id;
    for(i=1; i<n; i++)
    {
        if( mx > i )
            p[i] = MIN( p[2*id-i], mx-i );
        else
            p[i] = 1;
        for(; str[i+p[i]] == str[i-p[i]]; p[i]++)
            ;
        if( p[i] + i > mx )
        {
            mx = p[i] + i;
            id = i;
        }
    }
}
```

3.RMQ

一维

```
int a[MAXN];
int mm[MAXN];
int dp[MAXN][20];
int n , k;
void initRMQ()
{
    mm[0] = -1;
    for(int i = 1 ; i <= MAXN ; i++) {
        mm[i] = ((i & (i - 1)) == 0) ? mm[i - 1] + 1 : mm[i - 1];
    }
    for(int i = 1 ; i <= n ; i++) {
```

```
        dp[i][0] = a[i];
    }
    for(int i = 1 ; i <= mm[n] + 1 ; i++) {
        for(int j = 1 ; j + (1 << i) - 1 <= n ; j++) {
            dp[j][i] = max(dp[j][i - 1] , dp[j + (1 << (i
- 1))][i - 1]);
        }
    }
}
int rmq(int L , int R)
{
    int k = mm[R - L + 1];
    return max(dp[L][k] , dp[R - (1 << k) + 1][k]);
}
```

二维

```
int a[MAXN][MAXN];
int mm[MAXN];
int dp[MAXN][MAXN][9][9];
int n , m , q;
void initRMQ()
{
    mm[0] = -1;
    for(int i = 1 ; i <= MAXN ; i++) {
        mm[i] = ((i & (i - 1)) == 0) ? mm[i - 1] + 1 : mm[i
- 1];
    }
    for(int i = 1 ; i <= n ; i++) {
        for(int j = 1 ; j <= m ; j++) {
            dp[i][j][0][0] = a[i][j];
        }
    }
    for(int ii = 0 ; ii <= mm[n] ; ii++) {
        for(int jj = 0 ; jj <= mm[m] ; jj++) {
            if(ii + jj) {
                for(int i = 1 ; i + (1 << ii) - 1 <= n ; i++)
{
                    for(int j = 1 ; j + (1 << jj) - 1 <= m ;
j++) {
                        if(ii)dp[i][j][ii][jj] =
max(dp[i][j][ii - 1][jj] , dp[i + (1 << (ii - 1))][j][ii
- 1][jj]);
                        else dp[i][j][ii][jj] =
max(dp[i][j][ii][jj - 1] , dp[i][j + (1 << (jj -
1))][ii][jj - 1]);
                    }
                }
            }
        }
    }
}
int rmq(int x1 , int y1 , int x2 , int y2)
{
    int k1 = mm[x2 - x1 + 1];
    int k2 = mm[y2 - y1 + 1];
    x2 = x2 - (1 << k1) + 1;
    y2 = y2 - (1 << k2) + 1;
    return max(max(dp[x1][y1][k1][k2] ,
dp[x1][y2][k1][k2]) , max(dp[x2][y1][k1][k2] ,
dp[x2][y2][k1][k2]));
}
```

4.LCA

离线——Tarjan

```
//flag 初始化为 0 ， 需要并查集的 find ， g 记录每个点的 query 对点 ， dis 记录该点距根的距离
//初始 tarjan 根（1）
void tarjan(int u)
{
    flag[u] = 1;
    for(int i = 0 ; i < g[u].sz ; i++) {
        int v = g[u][i].v , id = g[u][i].id;
        if(flag[v])query[id][2] = find(v);
    }
    for(int i = head[u] ; i != -1 ; i = e[i].next) {
        int v = e[i].v , w = e[i].c;
        if(!flag[v]) {
            dis[v] = dis[u] + w;
            tarjan(v);
            fa[v] = u;
        }
    }
}
```

在线——基于 RMQ

```
vector<pair<int,int> >e[MAXN];
int dist[MAXN];
int root;
int depth,b[MAXN*2],a[MAXN*2],tot;
int p[MAXN],f[MAXN];
int dp[MAXN*2][20];
int ori[MAXN];
void init()
{
    tot = 0;
    depth = 0;
    clr(ori , 0);
    for(int i = 0 ; i <= n ; i++)e[i].clear();
}
int find_root()
{
    for(int i = 1 ; i <= n; i++) {
        if(!ori[i])return i;
    }
}
void dfs()
{
    stack<pair<int,pair<int,int> > >s;
    s.push(mk(root,mk(0,0)));
    while(!s.empty()) {
        pair<int,pair<int,int> >now=s.top();
        s.pop();
        int
u=now.first,pre=now.second.first,i=now.second.second;
        if(i==0) {
            int t=++depth;
            b[++tot]=t;
            f[t]=u;
            p[u]=tot;
        }
        if(i<e[u].size()) {
            int v=e[u][i].first,w=e[u][i].second;
            s.push(mk(u,mk(pre,i+1)));
            if(v==pre) continue;
            dist[v]=dist[u]+w;
            s.push(mk(v,mk(u,0)));
        } else
            b[++tot]=b[p[pre]];
```

```
    }
}

void Init_rmq(int n)
{
    for(int i=1; i<=n; i++)
        dp[i][0]=b[i];
    int m=floor(log(n*1.0)/log(2.0));
    for(int j=1; j<=m; j++)
        for(int i=1; i<=n-(1<<j)+1; i++)

dp[i][j]=min(dp[i][j-1],dp[i+(1<<(j-1))][j-1]);
}
int rmq(int l,int r)
{
    int k=floor(log((r-l+1)*1.0)/log(2.0));
    return min(dp[l][k],dp[r-(1<<k)+1][k]);
}
int lca(int a,int b)
{
    if(p[a]>p[b]) swap(a,b);
    return f[rmq(p[a],p[b])];
}
```

三. 图论

1.最短路

1.1 Dijstra

```
int dij()
{
    priority_queue<pii, vector<pii>, greater<pii> >Q;
    memset(vis, false, sizeof(vis));
    for(int i = 0; i <= 4 * n * m + 2; i++) dis[i] = inf;
    dis[st] = 0;
    Q.push(pii(dis[st], st));
    while(!Q.empty()) {
        pii x = Q.top();
        Q.pop();
        int u = x.second;
        if(vis[u]) continue;
        vis[u] = true;
        for(int i = head[u]; i != -1; i = e[i].next) {
            int v = e[i].v, w = e[i].c;
            if(dis[v] > dis[u] + w) {
                dis[v] = dis[u] + w;
                Q.push(pii(dis[v], v));
            }
        }
    }
    return dis[en];
}
```

1.2 最小环

```
///inf 不要开太大
void init()
```

```
{
    for(int i = 1; i <= n ; i++) {
        for(int j = 1 ; j <= n ; j ++) {
            g[i][j] = dist[i][j] = inf;
            pre[i][j] = i;
        }
    }
}
void floyd()
{
    for(int k = 1 ; k <= n ; k++) {
        //求最大环不要这部分循环，普通的 floyd 就可以
        //最后最大环为 max(dist[i][i])
        for(int i = 1;  i < k ; i++) {
            for(int j = i + 1 ; j < k ; j++) {
                if(ans > g[i][j] + dist[i][k] + dist[k][j])
{
                    ans = g[i][j] + dist[i][k] + dist[k][j];
                    num = 0;
                    int p = j;
                    while(p != i) {
                        path[num++] = p;
                        p = pre[i][p];
                    }
                    path[num++] = i;
                    path[num++] = k;
                }
            }
        }
        for(int i = 1 ; i <= n ; i++) {
            for(int j = 1 ; j <= n ; j++) {
                if(g[i][j] > g[i][k] + g[k][j]) {
                    g[i][j] = g[i][k] + g[k][j];
                    pre[i][j] = pre[k][j];
                }
            }
        }
    }
}
```

1.3 最短路次短路计数

```
void dij()
{
    clr(cnt , 0);
    clr(vis , 0);
    for(int i = 0 ; i <= n ; i++) {
        dis[i][0] = inf;
        dis[i][1] = inf;
    }
    dis[st][0] = 0;
    cnt[st][0] = 1;
    for(int i = 1 ; i <= 2 * n ; i++) {
        int tmp = inf , tx = -1 , flag;
        for(int j = 1 ; j <= n ; j++) {
            if(!vis[j][0] && dis[j][0] < tmp) {
                tmp = dis[j][0];
                tx = j;
                flag = 0;
            } else if(!vis[j][1] && dis[j][1] < tmp) {
                tmp = dis[j][1];
                tx = j;
                flag = 1;
            }
        }
        if(tx == -1)break;
```

```
        vis[tx][flag] = 1;
        for(int j = head[tx] ; ~j ; j = e[j].next) {
            int v = e[j].v , c = e[j].c;
            if(dis[tx][flag] + c < dis[v][0]) {
                dis[v][1] = dis[v][0];
                cnt[v][1] = cnt[v][0];
                dis[v][0] = dis[tx][flag] + c;
                cnt[v][0] = cnt[tx][flag];
            } else if(dis[tx][flag] + c == dis[v][0]) {
                cnt[v][0] += cnt[tx][flag];
            } else if(dis[tx][flag] + c == dis[v][1]) {
                cnt[v][1] += cnt[tx][flag];
            } else if(dis[tx][flag] + c < dis[v][1]) {
                dis[v][1] = dis[tx][flag] + c;
                cnt[v][1] = cnt[tx][flag];
            }
        }
    }
}
```

1.4 K 短路

```
//以终点 t 为源对反图 spfa
struct Edge {
    int v , c , next;
} e[MAXE] , re[MAXE];
struct node {
    int now,g,f;
    bool operator <(const node a)const
    {
        if(a.f == f) return a.g < g;
        return a.f < f;
    }
};

int dis[MAXN];
int in[MAXN];
int head[MAXN];
int rehead[MAXN];
int cnt[MAXN];
int num;
void init()
{
    num = 0;
    clr(head , -1);
    clr(rehead , -1);
}
void adde(int u , int v , int c)
{
    e[num].v = v;
    e[num].c = c;
    e[num].next = head[u];
    head[u] = num;
    re[num].v = u;
    re[num].c = c;
    re[num].next = rehead[v];
    rehead[v] = num++;
}
int Astar(int src,int to)
{
    priority_queue<node> Q;
    int i,cnt = 0;
    if(src == to) k++;//在起点与终点是同一点的情况下，k 要+1
    if(dis[src] == inf) return -1;
    node a,next;
    a.now = src;
```

```
    a.g = 0;
    a.f = dis[src];
    Q.push(a);
    while(!Q.empty()) {
        a = Q.top();
        Q.pop();
        if(a.now == to) {
            cnt++;
            if(cnt == k)
                return a.g;
        }
        for(i = head[a.now]; i!=-1; i = e[i].next) {
            next = a;
            next.now = e[i].v;
            next.g = a.g+e[i].c;
            next.f = next.g+dis[next.now];
            Q.push(next);

        }
    }
    return -1;//不能到达
}
```

2.最小生成树

2.1 Prim

```
int lowcost[MAXN],closest[MAXN];
int prim(int v0)
{
    int i,j,mindis,minone;
    int ans = 0;/*用来记录最小生成树的总长度*/
    /*各点距离初始化*/
    for(i = 0; i < n; i++) {
        lowcost[i] = cost[v0][i];
        closest[i] = v0;
    }
    for(i = 0; i < n-1; i++) {
        mindis = inf;
        for(j = 0; j < n; j++)
            if(lowcost[j] && mindis > lowcost[j]) {
                mindis = lowcost[j];
                minone = j;
            }
        /*将找到的最近点加入最小生成树*/
        ans += lowcost[minone];
        lowcost[minone] = 0;
        /*修正其他点到最小生成树的距离*/
        for(j = 0; j < n; j++)
            if(cost[j][minone] < lowcost[j]) {
                lowcost[j] = cost[j][minone];//邻接矩阵
                closest[j] = minone;
            }
    }
    return ans;
}
```

2.2 最小树形图

最小树形图（根固定） O(VE)

有向图最小生成树

根不固定，添加一个根节点与所有点连无穷大的边！

根据 pre 的信息能构造出这棵树！

```
int n , m;
int pre[MAXN] , in[MAXN] , vis[MAXN];
int id[MAXN];
int Directed_mst(int root)
{
    n++;
    int ret = 0;
    while(1) {
        for(int i = 0 ; i < n ; i++) {
            in[i] = inf;
            id[i] = -1;
            vis[i] = -1;
        }
        for(int i = 0 ; i < m ; i++) {
            int u = e[i].u;
            int v = e[i].v;
            if(e[i].c >= in[v] || u == v)continue;
            pre[v] = u;
            in[v] = e[i].c;
        }
        in[root] = 0;
        pre[root] = root;
        for(int i = 0 ; i < n ; i++) {
            ret += in[i];
            if(in[i] == inf) {
                return -1;
            }
        }
        int cnt = 0;
        for(int i = 0 ; i < n ; i++) {
            if(vis[i] == -1) {
                int v = i;
                while(vis[v] == -1) {
                    vis[v] = i;
                    v = pre[v];
                }
                if(vis[v] != i || v == root)continue;
                for(int u = pre[v] ; u != v ; u = pre[u])
{
                    id[u] = cnt;
                }
                id[v] = cnt++;
            }
        }
        if(!cnt)break;
        for(int i = 0 ; i < n ; i++) {
            if(id[i] == -1)id[i] = cnt++;
        }
        for(int i = 0 ; i < m ; i++) {
            int v = e[i].v;
            e[i].u = id[e[i].u];
            e[i].v = id[e[i].v];
            e[i].c -= in[v];
        }
        n = cnt;
        root = id[root];
    }
    return ret;
}
```

2.3 度限制最小生成树

就是图中的某些点在生成树时有度的限制，找满足这些约束的最小生成树。如果所有点都有度限制，那么这个问题将是 NP 难题），具体算法如下：

1. 先求出最小 m 度限制生成树：原图中去掉和 V0 相连的所有边（可以事先存两个图， Ray 的方法是一个邻接矩阵，一个邻接表，用方便枚举边的邻接表来构造新图），得到 m 个连通分量，则这 m 个连通分量必须通过 v0 来连接，所以，在图 G 的所有生成树中 dT(v0)≥m 。也就是说，当 k<m 时，问题无解。对每个连通分量求一次最小生成树（哪个算法都行），对于每个连通分量 V′ ，用一条与 V0 直接连接的最小的边把它与 V0 点连接起来，使其整体成为一个生成树。于是，我们就得到了一个 m 度限制生成树，不难证明，这就是最小 m 度限制生成树。

2. 由最小 m 度限制生成树得到最小 m+1 度限制生成树;：连接和 V0 相邻的点 v ，则可以知道一定会有一个环出现（因为原来是一个生成树），只要找到这个环上的最大权边（不能与 v0 点直接相连）并删除，就可以得到一个 m+1 度限制生成树，枚举所有和 V0 相邻点 v ，找到替换后，增加权值最小的一次替换 (当然，找不到这样的边时，就说明已经求出) ，就可以求得 m+1 度限制生成树。。如果每添加一条边，都需要对环上的边一一枚举，时间复杂度将比较高（但这个题数据比较小，所以这样也没问题，事实上，直接枚举都能过这个题），这里，用动态规划解决。设 Best(v) 为路径 v0—v 上与 v0 无关联且权值最大的边。定义 father(v) 为 v 的父结点，由此可以得到动态转移方程：
Best(v)=max(Best(father(v)),ω(father(v),v)) ，边界条件为 Best[v0]=-∞ (因为我们每次寻找的是最大边，所以 -∞ 不会被考虑) ， Best[v′]=-∞|(v0,v′)∈E(T) 。

3. 当 dT(v0)=k 时停止(即当 V0 的度为 k 的时候停止)，但不一定 k 的时候最优。

接下来说一下算法的实现：

采用并查集+ kruskal 代码量还少一点。

首先，每个连通分量的的最小生成树可以直接用一个循环，循环着 Kruskal 求出。这里利用了联通分量间的独立性，对每个连通分量分别求最小生成树，和放在一起求，毫不影响。而且 kruskral 算法保证了各连通分量边的有序性。

找最小边的时候，上面讲的动态规划无疑是一种好方法，但是也可以这么做：

先走一个循环，但我们需要逆过来加边，将与 v0 关联的所有边从小到达排序，然后将各连通分量连接起来，利用并查集可以保证每个连通分量只有一条边与 v0 相连，由于边已经从小到达排序，故与每个连通分量相连的边就是每个连通分量与 v0 相连中的最小边。

然后求 m+1 度的最小生成树时，可以直接用 DFS ，最小生成树要一直求到 k 度，然后从中找出一个最优值。

```
int vis[MAXN];
int lowcost[MAXN];
int best[MAXN];
int pre[MAXN];
int n , k , ans , cnt;
int g[MAXN][MAXN];
int edge[MAXN][MAXN];
int fa[MAXN];
int mark[MAXN];
map <string ,int> m;
void init()
{
    clr(g , 0x3f);
```

```
    cnt = 1;
}
void dfs(int s)
{
    for(int i = 1 ; i <= cnt ; i++){
        if(mark[i] && edge[s][i]){
            mark[i] = 0;
            fa[i] = s;
            dfs(i);
        }
    }
}
int prim(int s)
{
    clr(mark , 0);
    int sum , pos;
    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[pos] > lowcost[j]){
                    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;
                }
            }
        }
    }
    int minedge = inf;
    int root = -1;
    for(int i = 1 ; i <= cnt ; i++){
        if(mark[i] && g[i][1] < minedge){
            minedge = g[i][1];
            root = i;
        }
    }
    mark[root] = 0;
    dfs(root);
    fa[root] = 1;
    return sum + minedge;
}
int Best(int s)
{
    if(fa[s] == 1)return -1;
    if(best[s] != -1)return best[s];
    int tmp = Best(fa[s]);
    if(tmp != -1 && g[fa[tmp]][tmp] > g[fa[s]][s]){
        best[s] = tmp;
    }
    else best[s] = s;
    return best[s];
}
```

```
void solve()
{
    clr(vis , 0);
    clr(fa , -1);
    clr(edge , 0);
    vis[1] = 1;
    int num = 0;
    ans = 0;
    for(int i = 1 ; i <= cnt ; i++){
        if(!vis[i]){
            num++;
            ans += prim(i);
        }
    }
    int minadd , change;
    for(int i = num + 1 ; i <= k && i <= cnt; i ++){
        clr(best , -1);
        for(int j = 1 ; j <= cnt ; j++){
            if(fa[j] != 1 && best[j] == -1){
                Best(j);
            }
        }
        minadd = inf;
        for(int j = 1 ;  j <= cnt ; j++){
            if(g[1][j] != inf && fa[j] != 1){
                int a = best[j];
                int b = fa[best[j]];
                int tmp = g[1][j] - g[a][b];
                if(minadd > tmp){
                    minadd = tmp;
                    change = j;
                }
            }
        }
        if(minadd >= 0)break;
        ans += minadd;
        int a = best[change];
        int b = fa[best[change]];
        g[a][b] = g[b][a] = inf;
        fa[a] = 1;
        g[1][a] = g[a][1] = g[change][1];
        g[1][change] = g[change][1] = inf;
    }
}
int main()
{
    m.clear();
    m["Park"] = 1;
    init();
    scanf("%d" , &n);
    string str1 , str2;
    int v;
    for(int i = 0 ; i < n ; i++){
        cin >> str1 >> str2;
        scanf("%d" , &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)g[a][b] = g[b][a] = v;
    }
    scanf("%d" , &k);
    solve();
    printf("Total miles driven: %d\n" , ans);
```

```
    return 0;
}
```

2.4 最优比例生成树

```
double prim(double s)
{
    clr(vis , 0);
    double retdis = 0;
    int retcost = 0;
    vis[1] = 1;
    for(int i = 2 ; i <= n ; i++) {
        lowcost[i] = abs(v[1].h - v[i].h) - s * dis[1][i];
        closest[i] = 1;
    }
    for(int num = 0 ; num < n - 1 ; num++) {
        double minans = inf;
        int u;
        for(int i = 1 ; i <= n ; i++) {
            if(vis[i])continue;
            if(lowcost[i] < minans) {
                minans = lowcost[i];
                u = i;
            }
        }
        vis[u] = 1;
        retcost += abs(v[closest[u]].h - v[u].h);
        retdis += dis[closest[u]][u];
        for(int i = 1 ; i <= n ; i++) {
            double cal = abs(v[u].h - v[i].h) - s *
dis[u][i];
            if(!vis[i] &&cal < lowcost[i]) {
                lowcost[i] = cal;
                closest[i] = u;
            }
        }
    }
    return 1.0 * retcost / retdis;
}
int main()
{
    double ans = 0;
    while(1) {
        double tmp = prim(ans);
        if(fabs(tmp - ans) < eps) {
            break;
        }
        ans = tmp;
    }
    printf("%.3f\n" , ans);
}
```

2.5 生成树计数

```
//对 mod 取模
int INV[mod];
//求 ax = 1( mod m) 的 x 值，就是逆元(0<a<m)
long long inv(long long a,long long m)
{
    if(a == 1)return 1;
    return inv(m%a,m)*(m-m/a)%m;
}
struct Matrix {
```

```
    int mat[330][330];
    void init()
    {
        clr(mat , 0);
    }
    int det(int n)//求行列式的值模上 mod，需要使用逆元
    {
        for(int i = 0; i < n; i++)
            for(int j = 0; j < n; j++)
                mat[i][j] = (mat[i][j]%mod+mod)%mod;
        int res = 1;
        for(int i = 0; i < n; i++) {
            for(int j = i; j < n; j++)
                if(mat[j][i]!=0) {
                    for(int k = i; k < n; k++)
                        swap(mat[i][k],mat[j][k]);
                    if(i != j)
                        res = (-res+mod)%mod;
                    break;
                }
            if(mat[i][i] == 0) {
                res = -1;//不存在(也就是行列式值为 0)
                break;
            }
            for(int j = i+1; j < n; j++) {
                //int mut =
(mat[j][i]*INV[mat[i][i]])%mod;//打表逆元
                int mut =
(mat[j][i]*inv(mat[i][i],mod))%mod;
                for(int k = i; k < n; k++)
                    mat[j][k] =
(mat[j][k]-(mat[i][k]*mut)%mod+mod)%mod;
            }
            res = (res * mat[i][i])%mod;
        }
        return res;
    }
};
```

主函数：

```
/*
    对逆元打表
    for(int i = 1;i < MOD;i++)
        INV[i] = inv(i,MOD);
*/
Matrix ret;
ret.init();
for(int i = 0; i < n; i++)
    for(int j = 0; j < n; j++)
        if(i != j && g[i][j])
        {
            ret.mat[i][j] = -1;
            ret.mat[i][i]++;
        }
printf("%d\n",ret.det(n-1));
```

```
//不取模
double b[MAXN][MAXN];
double det(double a[][MAXN],int n)
{
    int i, j, k, sign = 0;
    double ret = 1;
    for(i = 0; i < n; i++)
        for(j = 0; j < n; j++)
            b[i][j] = a[i][j];
    for(i = 0; i < n; i++) {
        if(sgn(b[i][i]) == 0) {
```

```
            for(j = i + 1; j < n; j++)
                if(sgn(b[j][i]) != 0)
                    break;
            if(j == n)return 0;
            for(k = i; k < n; k++)
                swap(b[i][k],b[j][k]);
            sign++;
        }
        ret *= b[i][i];
        for(k = i + 1; k < n; k++)
            b[i][k]/=b[i][i];
        for(j = i+1; j < n; j++)
            for(k = i+1; k < n; k++)
                b[j][k] -= b[j][i]*b[i][k];
    }
    if(sign & 1)ret = -ret;
    return ret;
}
```

3.拓扑排序

3.1 朴素算法

将入度为 0 的点加入队列，再依次取出队列中的元素，把其出边依次遍历，并且将边指向的节点入度减一，同时将入度为 0 的点加入队列。

```
queue < int > Q;
memset(idegree, 0, sizeof(idegree));
memset(last, -1, sizeof(last));
tot = 0;
for(int i = 0; i < m; i++)
{
    int x, y;
    scanf("%d%d", &x, &y);
    addedge(x, y);
    idegree[y]++;
}
int now = 0;
for(int i = 1; i <= n; i++)
    if(idegree[i] == 0)
        Q.push(i);
while(!Q.empty())
{
    int u = Q.top();
    Q.pop();
    now++;
    printf("%d", u);
    if(now < n) putchar(' ');
    else putchar('\n');
    for(int j = last[u]; -1 != j; j = edge[j].next) {
        int v = edge[j].v;
        idegree[v]--;
        if(idegree[v] == 0)
            Q.push(v);
    }
}
```

3.2 求字典序最小的方案

当各点标号均互不相同时，直接将队列改为优先队列；
当存在重复标号时？？？？

3.3 基于 DFS 的拓扑排序（很短很好写）

记录各点完成访问的时刻(完成时间),用 DFS 遍历一次整个图,得出各结点的完成时间,然后按完成时间倒序排列就得到了图的拓扑序列

```
/* 拓扑排序        O(e)
 * 确保是有向无环图!
 * 结果逆序存放在 sta 中!
 * */
VI ve[Maxn];
bool vis[Maxn];
int sta[Maxn], top;
void dfsTopo(int u)
{
    vis[u] = true;
    for (int i = 0; i < ve[u].size(); i ++ )
        if (!vis[ve[u][i]])
            dfsTopo(ve[u][i]);
    sta[top ++ ] = u;
}
void Toposort(int n)
{
    memset(vis, 0, sizeof(vis));
    top = 0;
    for (int i = 1; i <= n; i ++ )
        if (!vis[i]) dfsTopo(i);
}
```

3.4 拓扑排序数

求树的拓扑排序数：

```
dp[root] = num[root] ! / (num[i] , i in tree[root])
```

3.5 关键路径

关键节点：

正向拓扑序，求每个点最早到达时间 `early[i] = max(early[i], early[j] + edge[k]);`

利用汇点的 early 值，反向拓扑求每个点迟到达时间 `late[i] = min(late[i], late[j] - edge[k]);`

关键节点 `early == late`

PS:

最早开始时间可以理解为是必须等到之前的任务完成才能做

最迟开始时间可以理解为是必须为后面的任务留出足够的时间

关键路径：

源点到汇点的最长路（可能有多条）

```
/*
* 求出 early 和 latest
* idegree  odegree 出入度
* 中间有重建图，所以保存了一条边的 u 和 v
*/

memset(idegree, 0, sizeof(idegree));
memset(odegree, 0, sizeof(odegree));
memset(last, -1, sizeof(last));
tot = 0;
for(int i = 0; i < m; i++)
{
```

```
    int x, y, z;
    scanf("%d%d%d", &x, &y, &z);
    addedge(x, y, z);
    idegree[y]++;
    odegree[x]++;
}
memset(early, 0, sizeof(early));
for(int i = 1; i <= n; i++)
    if(!idegree[i])
        Q.push(i);
while(!Q.empty())
{
    int u = Q.front();
    Q.pop();
    for(int j = last[u]; -1 != j; j = edge[j].next) {
        int v = edge[j].v;
        idegree[v]--;
        if(!idegree[v])
            Q.push(v);
        int temp = early[u] + edge[j].len;
        early[v] = max(early[v], temp);
    }
}
memset(last, -1, sizeof(last));
tot = 0;
for(int i = 0; i < m; i++)
    addedge(edge[i].v, edge[i].u, edge[i].len);
memset(latest, CLRINF, sizeof(latest));
cout << latest[0] << endl;
for(int i = 1; i <= n; i++)
    if(odegree[i] == 0)
    {
        latest[i] = early[i];
        Q.push(i);
    }
while(!Q.empty())
{
    int u = Q.front();
    Q.pop();
    for(int j = last[u]; -1 != j; j = edge[j].next) {
        int v = edge[j].v;
        odegree[v]--;
        if(!odegree[v])
            Q.push(v);
        int temp = latest[u] - edge[j].len;
        latest[v] = min(latest[v], temp);
    }
}
```

4.连通性

4.1 SCC

```
//注释部分为判断仙人掌图
void tarjan(int u)
{
    low[u] = dfn[u] = ++dindex;
    st[++tail] = u;
    instack[u] = 1;
    //  int cnt = 0;
    for(int j = head[u] ; ~j ; j = e[j].next) {
        int v = e[j].v;
        //        if(color[v])ok = 0;
```

```
        if(!dfn[v]) {
            tataarrjan(v);
            //          if(low[v] > dfn[u])ok = 0;
            //          if(low[v] < dfn[u])cnt++;
            //          if(cnt == 2)ok = 0;
            low[u] = min(low[u] , low[v]);
        } else if(instack[v]) {
            low[u] = min(low[u] , dfn[v]);
            //          cnt++;
            //          if(cnt == 2)ok = 0;
        }
    }
    if(dfn[u] == low[u]) {
        int i;
        bcnt++;
        do {
            i = st[tail--];
            instack[i] = 0;
            cmp[i] = bcnt;
        } while(i != u);
    }
    //    color[u] = 1;
}
void scc()
{
    clr(dfn , 0);
    clr(low , 0);
    clr(instack , 0);
    clr(cmp , 0);
    //  clr(color , 0);
    tail = bcnt = dindex = 0;
    for(int i = 1 ; i <= N ; i++) {
        if(!dfn[i])tarjan(i);
    }
```

```
}
```

4.2 点双联通分量

POJ 2942 Knights of the Round Table

亚瑟王要在圆桌上召开骑士会议，为了不引发骑士之间的冲突，
并且能够让会议的议题有令人满意的结果，每次开会前都必须对出席会议的骑士有如下要求：

1、 相互憎恨的两个骑士不能坐在直接相邻的 2 个位置；

2、 出席会议的骑士数必须是奇数，这是为了让投票表决议题时都能有结果。

注意：1、所给出的憎恨关系一定是双向的，不存在单向憎恨关系。

2、由于是圆桌会议，则每个出席的骑士身边必定刚好有 2 个骑士。

即每个骑士的座位两边都必定各有一个骑士。

3、一个骑士无法开会，就是说至少有 3 个骑士才可能开会。

首先根据给出的互相憎恨的图中得到补图。

然后就相当于找出不能形成奇圈的点。

利用下面两个定理：

（1）如果一个双连通分量内的某些顶点在一个奇圈中（即双连通分量含有奇圈），

那么这个双连通分量的其他顶点也在某个奇圈中；

（2）如果一个双连通分量含有奇圈，则他必定不是一个二分图。反过来也成立，这是一个充要条件。

所以本题的做法，就是对补图求点双连通分量。

然后对于求得的点双连通分量，使用染色法判断是不是二分图，不是二分图，这个双连通分量的点是可以存在的

```
int n , m , k;
int head[MAXN];
int num;
int dfn[MAXN];
int low[MAXN];
int st[MAXN];
int g[MAXN][MAXN];
int top , nindex , ncnt;
int N;
struct Edge {
    int u , v , vis , next;
} e[MAXV] ;

void init()
{
    num = 0;
    clr(head , -1);
}
void adde(int u , int v)
{
    e[num].u = u;
    e[num].v = v;
    e[num].vis = 0;
    e[num].next = head[u];
    head[u] = num++;
}
int mark[MAXN];
int color[MAXN];
int odd[MAXN];
int find(int u)
{
    for(int j = head[u] ; ~ j ; j = e[j].next) {
        int v = e[j].v;
        if(mark[v]) {
            if(color[v] == -1) {
                color[v] = !color[u];
                return find(v);
            } else if(color[v] == color[u])return 1;
        }
    }
    return 0;
}
void col(int u)
{
    clr(mark , 0);
    int i;
    do {
        i = st[top--];
        mark[e[i].u] = 1;
        mark[e[i].v] = 1;
    } while(e[i].u != u);
    clr(color , -1);
    color[u] = 0;
    if(find(u)) {
        for(int i = 1 ; i <= n ; i++) {
            if(mark[i])odd[i] = 1;
        }
    }
}
void tarjan(int u)
{
    dfn[u] = low[u] = ++ nindex;
    for (int j = head[u]; j != -1; j = e[j].next) {
        int v = e[j].v;
        if(e[j].vis)continue;
        e[j].vis = e[j ^ 1].vis = 1;
        st[++top] = j;
        if(!dfn[v]) {
```

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

void solve()
{
    clr(dfn , 0);
    clr(low , 0);
    top = nindex = 0;
    for (int i = 1; i <= n; i ++ )
        if (!dfn[i]) {
            tarjan(i);
        }
}

int main()
{
    while(~scanf("%d%d" , &n , &m) , n) {
        init();
        clr(g , 0);
        for(int i = 0 ; i < m  ; i++) {
            int a , b;
            scanf("%d%d" , &a , &b);
            g[a][b] = g[b][a] = 1;
        }
        for(int i = 1 ; i <= n ; i++) {
            for(int j = i + 1 ; j <= n ; j++) {
                if(!g[i][j])adde(i , j) , adde(j , i);
            }
        }
        clr(odd , 0);
```

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

4.3 边双联通分量

```
void tarjan(int u, int pre)
{
    dfn[u] = low[u] = nindex++;
    instack[u] = 1;
    st[++top] = u;
    int v;
    for(int j = head[u]; ~j ; j = e[j].next) {
        v = e[j].v;
        if(v == pre)continue;
        if(!dfn[v]) {
            tarjan(v, u);
            low[u] = min(low[u] , low[v]);
            if(low[v] > dfn[u]) {
                bridge++;
                e[j].cut = 1;
                e[j ^ 1].cut = 1;
            }
```

```
        } else if(instack[v]) {
            low[u] = min(low[u] , dfn[v]);
        }
    }
    if(dfn[u] == low[u]) {
        ncnt++;
        do {
            v = st[top--];
            instack[v] = 0;
            cmp[v] = ncnt;
        } while(v != u);
    }
}
void solve()
{
    clr(dfn , 0);
    clr(low , 0);
    clr(instack , 0);
    clr(cmp , 0);
    ncnt = nindex = top = 0;
    bridge = 0;
    tarjan(1 , -1);
}
```

4.4 无向图全局最小割

```
int mat[600][600];
int res;
void Mincut(int n)
{
    int node[600], dist[600];
    bool visit[600];
    int i, prev, j, k;
    for (i = 0; i < n; i++)
        node[i] = i;
    while (n > 1) {
        int maxj = 1;
        for (i = 1; i <n; i++) { //初始化到已圈集合的割大小
            dist[node[i]] = mat[node[0]][node[i]];
            if (dist[node[i]] > dist[node[maxj]])
                maxj = i;
        }
        prev = 0;
        memset(visit, false, sizeof (visit));
        visit[node[0]] = true;
        for (i = 1; i < n; i++) {
            if (i == n - 1) { //只剩最后一个没加入集合的点，更新最小割
                res = min(res, dist[node[maxj]]);
                for (k = 0; k < n; k++) { //合并最后一个点以及推出它的集合中的点
mat[node[k]][node[prev]]=(mat[node[prev]][node[k]]+=mat[node[k]][node[maxj]]);
                }
                node[maxj] = node[--n]; //缩点后的图
            }
            visit[node[maxj]] = true;
            prev = maxj;
            maxj = -1;
            for (j = 1; j < n; j++) {
                if (!visit[node[j]]) { //将上次求的 maxj 加入集合，合并与它相邻的边到割集
                    dist[node[j]] += mat[node[prev]][node[j]];
```

```
                if (maxj == -1 || dist[node[maxj]] <
dist[node[j]])
                    maxj = j;
            }
        }
    }

    }
    return;
}

int main()
{
    int n, m, a, b, v;
    while (scanf("%d%d", &n, &m) != EOF) {
        res = (1 << 29);
        memset(mat, 0, sizeof (mat));
        while (m--) {
            scanf("%d%d%d", &a, &b, &v);
            mat[a][b] += v;
            mat[b][a] += v;
        }
        Mincut(n);
        printf("%d\n", res);
    }
    return 0;
}
```

4.5 2-SAT

综述：每个条件的形式都是 x[i]为真/假或者 x[j]为真/假，
每个 x[i]拆成 2*i 和 2*i+1 两个点，分别表示 x[i]为真，x[i]为假；加的每一条边之间的关系是 and

模型一：两者（A，B）不能同时取（但可以两个都不选）
说明：A 为假或 B 为假
那么选择了 A 就只能选择 B’，选择了 B 就只能选择 A’
连边 A→B’，B→A’

模型二：两者（A，B）不能同时不取（但可以两个都选）
说明：A 为真或 B 为真
那么选择了 A’就只能选择 B，选择了 B’就只能选择 A
连边 A’→B，B’→A

模型三：两者（A，B）要么都取，要么都不取
说明：……
那么选择了 A，就只能选择 B，选择了 B 就只能选择 A，选择了 A’就只能选择 B’，
选择了 B’就只能选择 A’
连边 A→B，B→A，A’→B’，B’→A’

模型四：两者（A，A’）必取 A
那么，那么，该怎么说呢？先说连边吧。
连边 A’→A

模型五（补充）：两者（A，B）两个必须不相同，即要么选 A，要么选 B
逻辑表达：A||B 非 A||非 B
连边：A 为真或 B 为真： A’--->B B’--->A;

A 为假或 B 为假: A-->B’ B-->A

说明：A 或 B，非 A 或非 B，前者表示两者至少有一个 true，后者表示至少有一个 false

2-SAT + 二分答案 统一建模的方式：

同一组的两个状态分别存储在 2*i 和 2*i+1 两个节点，产生 2*n 个节点

```
for(int i=1;i<2*n;i++)
    for(int j=0;j<i;j++)
    {
        if (i==(j^1)) continue;//记得 j^1 加上小括号
        sat.add_clause(i,j);//枚举出的不属于同一组的不相容的两点
    }
```

```
//求字典序最小解
struct Edge {
    int v,next;
} e[MAXE];
int head[MAXN],tot;
void init()
{
    tot = 0;
    memset(head,-1,sizeof(head));
}
void addedge(int u,int v)
{
    e[tot].v = v;
    e[tot].next = head[u];
    head[u] = tot++;
}
bool vis[MAXN];//染色标记，为 true 表示选择
int S[MAXN],top;//栈
bool dfs(int u)
{
    if(vis[u^1])return false;
    if(vis[u])return true;
    vis[u] = true;
    S[top++] = u;
    for(int i = head[u]; i != -1; i = e[i].next)
        if(!dfs(e[i].v))
            return false;
    return true;
}
bool Twosat(int n)
{
    memset(vis,false,sizeof(vis));
    for(int i = 0; i < n; i += 2) {
        if(vis[i] || vis[i^1])continue;
        top = 0;
        if(!dfs(i)) {
            while(top)vis[S[--top]] = false;
            if(!dfs(i^1)) return false;
        }
    }
    return true;
}
int main()
{
    int n,m;
    int u,v;
    while(scanf("%d%d",&n,&m) == 2) {
        init();
```

```
        while(m--) {
            scanf("%d%d",&u,&v);
            u--;
            v--;
            addedge(u,v^1);
            addedge(v,u^1);
        }
        if(Twosat(2*n)) {
            for(int i = 0; i < 2*n; i++)
                if(vis[i])
                    printf("%d\n",i+1);
        } else printf("NIE\n");
    }
    return 0;
}
```

```
//输出任意解
struct Edge {
    int v,next;
} e[MAXE];
int head[MAXN],num;
void init()
{
    num = 0;
    memset(head,-1,sizeof(head));
}
void addedge(int u,int v)
{
    e[num].v = v;
    e[num].next = head[u];
    head[u] = num++;
}
```

```
int
Low[MAXN],DFN[MAXN],Stack[MAXN],Belong[MAXN];//Belong
数组的值 1~scc
int Index,top;
int scc;
bool Instack[MAXN];
void Tarjan(int u)
{
    int v;
    Low[u] = DFN[u] = ++Index;
    Stack[top++] = u;
    Instack[u] = true;
    for(int i = head[u]; i != -1; i = e[i].next) {
        v = e[i].v;
        if( !DFN[v] ) {
            Tarjan(v);
            if(Low[u] > Low[v])Low[u] = Low[v];
        } else if(Instack[v] && Low[u] > DFN[v])
            Low[u] = DFN[v];
    }
    if(Low[u] == DFN[u]) {
        scc++;
        do {
            v = Stack[--top];
            Instack[v] = false;
            Belong[v] = scc;
        } while(v != u);
    }
}
bool solvable(int n)//n 是总个数,需要选择一半
{
    memset(DFN,0,sizeof(DFN));
    memset(Instack,false,sizeof(Instack));
    Index = scc = top = 0;
```

```
    for(int i = 0; i < n; i++)
        if(!DFN[i])
            Tarjan(i);
    for(int i = 0; i < n; i += 2) {
        if(Belong[i] == Belong[i^1])
            return false;
    }
    return true;
}
//*************************************************
//拓扑排序求任意一组解部分
queue<int>q1,q2;
vector<vector<int> > dag;//缩点后的逆向 DAG 图
char color[MAXN];//染色，为'R'是选择的
int indeg[MAXN];//入度
int cf[MAXN];
void solve(int n)
{
    dag.assign(scc+1,vector<int>());
    memset(indeg,0,sizeof(indeg));
    memset(color,0,sizeof(color));
    for(int u = 0; u < n; u++)
        for(int i = head[u]; i != -1; i = e[i].next) {
            int v = e[i].v;
            if(Belong[u] != Belong[v]) {
                dag[Belong[v]].push_back(Belong[u]);
                indeg[Belong[u]]++;
            }
        }
    for(int i = 0; i < n; i += 2) {
        cf[Belong[i]] = Belong[i^1];
        cf[Belong[i^1]] = Belong[i];
    }
    while(!q1.empty())q1.pop();
    while(!q2.empty())q2.pop();
    for(int i = 1; i <= scc; i++)
        if(indeg[i] == 0)
            q1.push(i);
    while(!q1.empty()) {
        int u = q1.front();
        q1.pop();
        if(color[u] == 0) {
            color[u] = 'R';
            color[cf[u]] = 'B';
        }
        int siz = dag[u].size();
        for(int i = 0; i < siz; i++) {
            indeg[dag[u][i]]--;
            if(indeg[dag[u][i]] == 0)
                q1.push(dag[u][i]);
        }
    }
}
int change(char s[])
{
    int ret = 0;
    int i = 0;
    while(s[i]>='0' && s[i]<='9') {
        ret *= 10;
        ret += s[i]-'0';
        i++;
    }
    if(s[i] == 'w')return 2*ret;
    else return 2*ret+1;
}
int main()
{
    int n,m;
```

```
    char s1[10],s2[10];
    while(scanf("%d%d",&n,&m) == 2) {
        if(n == 0 && m == 0)break;
        init();
        while(m--) {
            scanf("%s%s",s1,s2);
            int u = change(s1);
            int v = change(s2);
            addedge(u^1,v);
            addedge(v^1,u);
        }
        addedge(1,0);
        if(solvable(2*n)) {
            solve(2*n);
            for(int i = 1; i < n; i++) {
                //注意这一定是判断 color[Belong[
                if(color[Belong[2*i]] ==
'R')printf("%dw",i);
                else printf("%dh",i);
                if(i < n-1)printf(" ");
                else printf("\n");
            }
        } else printf("bad luck\n");
    }
    return 0;
}
```

5.匹配

5.1 最大匹配 匈牙利

```
int V;
int match;
int matchx[MAXN], matchy[MAXN];
int vis[MAXN];
int preHungary()
{
    int res = 0, u, v;
    for(int i = 1; i <= V; i++) {
        u = i;
        for(int j = head1[i]; ~j ; j = e1[j].next) {
            v = e1[j].v;
            if(matchy[v] == -1) {
                matchx[u] = v;
                matchy[v] = u;
                res++;
                break;
            }
        }
    }
    return res;
}
bool dfs(int u)
{
    int v;
    for(int j = head1[u]; ~j ; j = e1[j].next) {
        v = e1[j].v;
        if(vis[v]) continue;
        vis[v] = 1;
        if(matchy[v] == -1 || dfs(matchy[v])) {
            matchy[v] = u;
            matchx[u] = v;
            return 1;
        }
    }
```

```
    return 0;
}
void solve()
{
    match = 0;
    clr(matchx, -1);
    clr(matchy, -1);
    match += preHungary();
    for(int i = 1; i <= V; i++)
        if(matchx[i] == -1) {
            clr(vis, 0);
            if(dfs(i)) match++;
        }
}
```

5.2 一般图最大匹配——带花树

```
#define N 1100
#define M 200100
#define LEN 100100
#define INF (1 << 30)
typedef long long LL;

int n, head, tail, Start, Finish;
int match[N];     //表示哪个点匹配了哪个点
int Father[N];   //这个就是增广路的 father……但是用起来太精髓了
int base[N];     //该点属于哪朵花
int Q[N];
bool mark[N], map[N][N], InBlossom[N], in_Queue[N];
```

```
void init()
{
    int x,y;
    scanf("%d",&n);
    while (scanf("%d%d",&x,&y)!=EOF)
        map[x][y]=map[y][x]=1;
}

void BlossomContract(int x,int y)
{
    clr(mark, false);
    clr(InBlossom,false);
#define pre father[match[i]]
    int lca,i;
    for (i=x; i; i=pre) {
        i=base[i];
        mark[i]=true;
    }
    for (i=y; i; i=pre) {
        i=base[i];    //寻找 lca 之旅……一定要注意 i=base[i]
        if (mark[i]) {
            lca=i;
            break;
        }
    }
    for (i=x; base[i]!=lca; i=pre) {
        if (base[pre]!=lca) father[pre]=match[i]; //对于 BFS 树中的父边是匹配边的点，father 向后跳
        InBlossom[base[i]]=true;
        InBlossom[base[match[i]]]=true;
    }
    for (i=y; base[i]!=lca; i=pre) {
        if (base[pre]!=lca) father[pre]=match[i]; //同理
```

```
        InBlossom[base[i]]=true;
        InBlossom[base[match[i]]]=true;
    }
#undef pre
    if (base[x]!=lca) father[x]=y;      //注意不能从 lca 这
个奇环的关键点跳回来
    if (base[y]!=lca) father[y]=x;
    for (i=1; i<=n; i++)
        if (InBlossom[base[i]]) {
            base[i]=lca;
            if (!in_Queue[i]) {
                Q[++tail]=i;
                in_Queue[i]=true;      //要注意如果本来连向
BFS 树中父结点的边是非匹配边的点，可能是没有入队的
            }
        }
}

void Change()
{
    int x,y,z;
    z=Finish;
    while (z) {
        y=father[z];
        x=match[y];
        match[y]=z;
        match[z]=y;
        z=x;
    }
}

void FindAugmentPath()
{
    clr(father, 0);
    clr(in_Queue, false);
    for (int i=1; i<=n; i++) base[i]=i;
    head=0;
    tail=1;
    Q[1]=Start;
    in_Queue[Start]=1;
    while (head!=tail) {
        int x=Q[++head];
        for (int y=1; y<=n; y++)
            if (map[x][y] && base[x]!=base[y] &&
match[x]!=y)   //无意义的边
                if ( Start==y || match[y] &&
father[match[y]] )    //精髓地用 father 表示该点是否
                    BlossomContract(x,y);
                else if (!father[y]) {
                    father[y]=x;
                    if (match[y]) {
                        Q[++tail]=match[y];
                        in_Queue[match[y]]=true;
                    } else {
                        Finish=y;
                        Change();
                        return;
                    }
                }
    }
}

void Edmonds()
{
    clr(match, 0);
    for (Start=1; Start<=n; Start++)
        if (match[Start]==0)
            FindAugmentPath();
```

```
}

void output()
{
    clr(mark, false);
    int cnt=0;
    for (int i=1; i<=n; i++)
        if (match[i]) cnt++;
    printf("%d\n",cnt);
    for (int i=1; i<=n; i++)
        if (!mark[i] && match[i]) {
            mark[i]=true;
            mark[match[i]]=true;
            printf("%d %d\n",i,match[i]);
        }
}

int main()
{
    //    freopen("input.txt","r",stdin);
    init();
    Edmonds();
    output();
    return 0;
}
```

5.3 最大权匹配——KM

```
int nx,ny;//两边的点数
int g[MAXN][MAXN];//二分图描述
int linker[MAXN],lx[MAXN],ly[MAXN];//y 中各点匹配状态，x,y 中的点标号
```

```
int slack[MAXN];
bool visx[MAXN],visy[MAXN];

bool DFS(int x)
{
    visx[x] = true;
    for(int y = 0; y < ny; y++) {
        if(visy[y])continue;
        int tmp = lx[x] + ly[y] - g[x][y];
        if(tmp == 0) {
            visy[y] = true;
            if(linker[y] == -1 || DFS(linker[y])) {
                linker[y] = x;
                return true;
            }
        } else if(slack[y] > tmp)
            slack[y] = tmp;
    }
    return false;
}
int KM()
{
    memset(linker,-1,sizeof(linker));
    memset(ly,0,sizeof(ly));
    for(int i = 0; i < nx; i++) {
        lx[i] = -INF;
        for(int j = 0; j < ny; j++)
            if(g[i][j] > lx[i])
                lx[i] = g[i][j];
    }
    for(int x = 0; x < nx; x++) {
        for(int i = 0; i < ny; i++)
            slack[i] = INF;
        while(true) {
```

```
            memset(visx,false,sizeof(visx));
            memset(visy,false,sizeof(visy));
            if(DFS(x))break;
            int d = INF;
            for(int i = 0; i < ny; i++)
                if(!visy[i] && d > slack[i])
                    d = slack[i];
            for(int i = 0; i < nx; i++)
                if(visx[i])
                    lx[i] -= d;
            for(int i = 0; i < ny; i++) {
                if(visy[i])ly[i] += d;
                else slack[i] -= d;
            }
        }
    }
    int res = 0;
    for(int i = 0; i < ny; i++)
        if(linker[i] != -1)
            res += g[linker[i]][i];
    return res;
}
```

6.欧拉回路

```
/*欧拉回路，有向图*/
vec ve[Maxn];
int cur[Maxn];
stack<int> eulerianWalk(int u)    //返回欧拉回路的逆序
{
    stack<int> sta, ret;
    sta.push(u);
    cur[u] = 0;
    while (!sta.empty()) {
        u = sta.top();
        sta.pop();
        while (cur[u] < ve[u].size()) {
            sta.push(u);
            u = ve[u][cur[u] ++ ];
        }
        ret.push(u);
    }
    return ret;
}
```

7.网络流

7.1 最大流

Dinic

```
void adde(int u , int v , int c , int cc)
{
    e[num].u = u , e[num].v = v , e[num].c = c , e[num].next
= head[u] , head[u] = num++;
    e[num].u = v , e[num].v = u , e[num].c = cc , e[num].next
= head[v] , head[v] = num++;
}
int dis[MAXN] , cur[MAXN] , sta[MAXN] , que[MAXN] ,
pre[MAXN];
bool bfs(int s , int t , int n)
```

```
{
    int front = 0 , tail = 0;
    clr(dis , -1);
    dis[s] = 0;
    que[tail++] = s;
    while(front < tail) {
        for(int i = head[que[front ++ ]] ; i != -1 ; i =
e[i].next) {
            if(e[i].c > 0 && dis[e[i].v] == -1) {
                dis[e[i].v] = dis[e[i].u] + 1;
                if(e[i].v == t)return 1;
                que[tail ++] = e[i].v;
            }
        }
    }
    return 0;
}
int dinic(int s , int t , int n)
{
    int maxflow = 0;
    while(bfs(s , t , n)) {
        for(int i = 0 ; i < n ; i++)cur[i] = head[i];
        int u = s , tail = 0;
        while(cur[s] != -1) {
            if(u == t) {
                int det = INF;
                for(int i = tail - 1 ; i >= 0 ; i--) {
                    det = min(det , e[sta[i]].c);
                }
                maxflow += det;
                for(int i = tail - 1 ; i >= 0 ; i--) {
                    e[sta[i]].c -= det;
                    e[sta[i] ^ 1].c += det;
                    if(e[sta[i]].c == 0)tail = i;
                }
                u = e[sta[tail]].u;
            } else if(cur[u] != -1 && e[cur[u]].c > 0 &&
dis[u] + 1 == dis[e[cur[u]].v]) {
                sta[tail ++] = cur[u];
                u = e[cur[u]].v;
            } else {
                while(u != s && cur[u] == -1) {
                    u = e[sta[--tail]].u;
                }
                cur[u] = e[cur[u]].next;
            }
        }
    }
    return maxflow;
}
```

ISAP

```
void adde(int u , int v , int c , int cc)
{
    e[num].u = u , e[num].v = v , e[num].c = c , e[num].next
= head[u] , head[u] = num++;
    e[num].u = v , e[num].v = u , e[num].c = cc , e[num].next
= head[v] , head[v] = num++;
}

int dis[MAXN] , pre[MAXN] , cur[MAXN] , gap[MAXN];
int ISAP(int s , int t , int n)
{
    clr(dis , 0);
    clr(gap , 0);
    clr(cur , 0);
    for(int i = 0 ; i < n ; i++)cur[i] = head[i];
    gap[0] = n;
```

```
    pre[s] = s;
    int u = s , maxflow = 0;
    while(dis[s] <= n) {
        bool flag = false;
        for(int i = cur[u] ; i != -1;  i = e[i].next) {
            if(e[i].c > 0 && dis[u] == dis[e[i].v] + 1) {
                int v = e[i].v;
                cur[u] = i;
                pre[v] = u;
                flag = true;
                u = v;
                break;
            }
        }
        if(flag) {
            if(u == t) {
                int det = INF;
                for(int j = u ; j != s ; j = pre[j]) {
                    det = min(det , e[cur[pre[j]]].c);
                }
                for(int j = u ; j != s ; j = pre[j]) {
                    e[cur[pre[j]]].c -= det;
                    e[cur[pre[j]] ^ 1].c += det;
                }
                maxflow += det;
                u = s;
            }

        } else {
            int mind = n;
            for(int i = head[u] ; i != -1 ; i = e[i].next)
{
                if(e[i].c > 0 && dis[e[i].v] < mind) {
                    mind = dis[e[i].v];
                    cur[u] = i;
                }

            }
            if((--gap[dis[u]]) == 0)break;
            gap[dis[u] = mind + 1] ++;
            if(u != s)u = pre[u];
        }
    }
    return maxflow;
}
```

7.2 最小费用流

spfa 增广费用流

```
struct Edge {
    int from;
    int to;
    int next;
    int re;//记录逆边的下标
    int cap;//容量
    int cost;//费用
} e[MAXE];
int pre[MAXN];
int head[MAXN];
bool in[MAXN];
int que[MAXN];
int dis[MAXN];
int num;//边的总数
void init()
```

```
{
    num = 0;
    clr(head , -1);
}
void add(int u,int v,int ca,int co)
{
    e[num].from=u;
    e[num].to=v;
    e[num].cap=ca;
    e[num].cost=co;
    e[num].re=num+1;
    e[num].next=head[u];
    head[u]=num++;

    e[num].from=v;//加逆边
    e[num].to=u;
    e[num].cap=0;
    e[num].cost=-co;
    e[num].re=num-1;
    e[num].next=head[v];
    head[v]=num++;
}
int n;
int start;
int end;
bool SPFA()
{
    int front=0,rear=0;
    for(int v=0; v<=n; v++) {
        if(v==start) {
            que[rear++]=v;
            in[v]=true;
            dis[v]=0;
        } else {
            dis[v]=INF;
            in[v]=false;
        }
    }
    while(front!=rear) {
        int u=que[front++];
        in[u]=false;
        if(front>=MAXN)front=0;
        for(int i=head[u]; i!=-1; i=e[i].next) {
            int v=e[i].to;
            if(e[i].cap&&dis[v]>dis[u]+e[i].cost) {
                dis[v]=dis[u]+e[i].cost;
                pre[v]=i;
                if(!in[v]) {
                    que[rear++]=v;
                    in[v]=true;
                    if(rear>=MAXN)rear=0;
                }
            }
        }
    }
    if(dis[end]==INF)return false;
    return true;
}
int c;//费用
int f;//最大流

void minCostMaxflow()
{
    c=f=0;
    int u,p;
    while(SPFA()) {
        int Min=INF;
        for(u=end; u!=start; u=e[p].from) {
```

```
            p=pre[u];
            Min=min(Min,e[p].cap);
        }
        for(u=end; u!=start; u=e[p].from) {
            p=pre[u];
            e[p].cap-=Min;
            e[e[p].re].cap+=Min;

        }
        c+=dis[end]*Min;
        f+=Min;
    }
}
```

ZKW 流

```
struct node {
    int u, v, c, w, next;
} edge[M];
int tot, last[N];
int dist[N],pre[N];
bool visit[N];
int n, m, src, des;
int flow, cost, value;

void addedge(int u, int v, int c, int w)
{
    edge[tot].u = u;
    edge[tot].v = v;
    edge[tot].c = c;
    edge[tot].w = w;
    edge[tot].next = last[u];
    last[u] = tot++;
    edge[tot].u = v;
    edge[tot].v = u;
    edge[tot].c = 0;
    edge[tot].w = -w;
    edge[tot].next = last[v];
    last[v] = tot++;
}

int Aug(int u, int m)
{
    if(u == des) {
        cost += value * m;
        flow += m;
        return m;
    }
    visit[u] = true;
    int l = m;
    for(int j = last[u]; j != -1; j = edge[j].next) {
        int v = edge[j].v, c = edge[j].c, w = edge[j].w;
        if(c && !w && !visit[v]) {
            int delta = Aug(v, l < c ? l : c);
            edge[j].c -= delta;
            edge[j ^ 1].c += delta;
            l -= delta;
            if(!l) return m;
        }
    }
    return(m - l);
}

bool ModLabel(int src, int des)
{
    int d = INF;
    for(int i = 0; i <= des; i++)
        if(visit[i]) {
```

```
            for(int j = last[i]; j != -1; j = edge[j].next)
{
                if(edge[j].c && !visit[edge[j].v] &&
edge[j].w < d) d = edge[j].w;
            }
        }
    if(d == INF) return false;
    for(int i = 0; i <= des; i++)
        if(visit[i]) {
            for(int j = last[i]; j != -1; j = edge[j].next)
{
                edge[j].w -= d;
                edge[j^1].w += d;
            }
        }
    value += d;
    return true;
}

void MinCostMaxFlow(int src, int des)
{
    flow = cost = value = 0;
    int xx = 0;
    do {
        //cout << xx ++ << endl;
        do {
            memset(visit, 0, sizeof(visit));
        } while(Aug(src, INF));
    } while(ModLabel(src, des));
}
```

注意：

B[u,v]表示(u,v)流量的下限，C[u,v]表示(u,v)流量的上限，F[u,v]表示(u,v)的流量，

g[u,v]表示 F[u,v]-B[u,v] 显然 0<=g[u,v]<=C[u,v]-B[u,v]

1.无源汇的可行流 ：

我们要想办法转换为有源汇的最大流问题.

考虑流量都为 g[,]且容量为 C[,]-B[,]的网络，貌似有点接近最后的转换方式了，

为了不忽略 B[,]这一条件，我们把 g[,]最后强制加上 B[,].

但会发现一个致命漏洞，加上后就未必满足流量平衡了！

对于这个有两种办法解决..

一种方法是添加附加源汇 S,T 对于某点 u, 设 M(u)=sigma(B[i,u])-sigma(B[u,j]) ，

则根据流量平衡条件有 M(u)同时等于 sigma(g[u,j])-sigma(g[i,u])

若 M(u)<0，即 sigma(g[u,j]) < sigma(g[i,u]) 进入 u 的流量比从 u 出去的多，所以 u -> T 连容量为 -(sigma(B[i,u])-sigma(B[u,j])) 的边

同理. M(u)>0 时，即 S->u 连容量为 sigma(B[i,u])-sigma(B[u,j]) 的边.

然后再对于任意边(i,u)/(u,j) 连一条 C[u,v]-B[u,v]的边.

这样只需对新的网络求一遍最大流即可. 若出附加源点的边都满流即是存在可行流，反之不然.

满流的必要条件是显然的. 不满流不能保证加上 B[,]后流量平衡. 前面都白费了.

另一种方法相对简单.其实类似，本质相同.

仍添加附加源汇 S,T 对于某边 (u,v) 在新网络中连边 S->v 容量 B[u,v] ，u->T 容量 B[u,v] ， u->v 容量 C[u,v]-B[u,v]

可以这样理解，边 S->v：求的时候直接从 S 流过来的流量值 B[u,v]，与最终解中边(u,v)强制加上的从 u 流过来的流量 B[u,v]，对 v 点的流量平衡条件的

影响实质等价.
边 u->T 同理.
最后，一样也是求一下新网络的最大流，判断从附加源点的边，是否都满流即可.

具体的解？根据最前面提出的强制转换方式，边(u,v)的最终解中的实际流量即为 g[u,v]+B[u,v]
为什么这种方法只适用于无源汇上下界可行流？
本质上是因为 S,T 并不满足流量平衡，而上述的方法都是考虑到每点的流量平衡而建的. 但有些时候貌似还是可以出正确解. 至于有没有什么解决方法，下次再想想吧~【标记下】
例 题 ZOJ 2314 / SGU 194 Reactor Cooling
http://acm.sgu.ru/problem.php?contest=0&problem=194

2.有源汇的上下界可行流

从汇点到源点连一条上限为 INF，下限为 0 的边. 按照 1.无源汇的上下界可行流一样做即可.
改成无源汇后，求的可行流是类似环的，流量即 T->S 边上的流量. 这样做显然使 S,T 也变得流量平衡了.

3.有源汇的上下界最大流

方法一： 2.有源汇上下界可行流中，从汇点到源点的边改为连一条上限为 INF，下限为 x 的边.
因为显然 x>ans 即 MIN(T->S)> MAX(S->T) ,会使求新网络的无源汇可行流无解的（S,T 流量怎样都不能平衡）
而 x<=ans 会有解.
所以满足二分性质，二分 x，最大的 x 使得新网络有解的即是所求答案原图最大流.

方法二：从汇点 T 到源点 S 连一条上限为 INF，下限为 0 的边，变成无源汇的网络. 照求无源汇可行流的方法(如 1)，建附加源点 S'与汇点 T'，求一遍 S'->T'的最大流. 再把从汇点 T 到源点 S 的这条边拆掉 . 求一次从 S 到 T 的最大流即可.（关于 S',T'的边好像可以不拆？）(这样一定满足流量平衡？）
表示这方法我也没有怎么理解.

4.有源汇的上下界最小流

方法一： 2.有源汇上下界可行流中，从汇点到源点的边改为连一条上限为 x，下限为 0 的边.
与 3 同理，二分上限，最小的 x 使新网络无源汇可行流有解，即是所求答案原图最小流.
方法二: 照求无源汇可行流的方法(如 1)，建附加源点 S'与汇点 T'，求一遍 S'->T'的最大流. 但是注意这一遍不加汇点到源点的这条边，即不使之改为无源汇的网络去求解. 求完后，再加上那条汇点到源点上限 INF 的边. 因为这条边下限为 0，所以 S',T'无影响. 再直接求一遍 S'->T'的最大流. 若 S'出去的边全满流，T->S 边上的流量即为答案原图最小流，否则若不全满流即无解.
和求 3.有源汇的上下界最大流过程相反，感性理解是:
首先明确，我们的方法是通过加边转化成对任一点都有流量平衡的无源汇的网络，进行求解.
即最终解只能是加上边后，求的无源汇可行流，即 T->S 这边上的流量. 不改成无源汇的直接求的解是未必正确的，在（1）中已经提到.
然后，因为第一遍做的时候并无这条边，所以 S->T 的流量在第一遍做的时候都已经尽力往其他边流了. 于是加上 T->S 这条边后，都是些剩余的流不到其他边的流量. 从而达到尽可能减少 T->S 这边上的流量的效果，即减小了最终答案.
感觉上第一遍做的既然是不改成无源汇直接求的，应该是错误的？

这里不是错误的. 首先我们的解都是按照第二遍所求的而定，其次这里这样做本质是延迟对 T->S 这条边的增流.

四．博弈

1 翻硬币游戏

所翻动的硬币中，最右边那个硬币的必须是从正面翻到反面,谁不能翻谁输
局面的 SG 值为局面中每个正面朝上的棋子单一存在时的 SG 值的异或和
每次能翻转一个或两个硬币。(不用连续)

每个硬币的 SG 值为它的编号，初始编号为 0
每次必须连续翻转 k 个硬币

sg 的形式为 000…01 000…01，其中一小段 0 的个数为 k-1
每次必须翻动两个硬币，而且这两个硬币的距离要在可行集 S={1,2,3}中，硬币序号从 0 开始。(Twins 游戏)

位置 x,sg[x]=x%3;
每次可以翻动一个、二个或三个硬币。（Mock Turtles 游戏）

一个非负整数为 odious，当且仅当该数的二进制形式的 1 出现的次数是奇数，否则称作 evil

当 2x 为 odious 时，sg 值是 2x，当 2x 是 evil 时，sg 值是 2x+1

2 anti-nim 游戏

拿最后一个棋子的人输
先手必胜有两种状态：
1.如果每一个小游戏都只剩下一个石子了，SG 为 0。
2.至少一堆石子>1,且 SG 不为 0.

3 every-SG 游戏

多线程博弈。形象的说就是红队和蓝队每个队 n 个人，然后进行 n 个博弈，最后结束的一场博弈的胜者胜利。
如果 v 是先手必胜，则 f[v]=max(f[u])+1,其中 u 为 v 的后继且 u 为先手必败。
否则 f[v]=min(f[u])+1,u 为 v 后继。

4 删边游戏

移除一个有根图的某些边，直到没有与地板的相连的边
Colon Principle：当树枝在一个顶点上时，用一个非树枝的杆的长度来替代，相当于他们的 n 异或之和。
The Fusion Principle：任何环内的节点可以融合成一点而不会改变图的 sg 值。
拥有奇数条边的环可简化为一条边，偶数条边的环可简化为一个节点

5 Ferguson 博弈

第一个盒子中有 n 枚石子，第二个盒子中有 m 个石子(n, m > 0),清空一个盒子中的石子，然后从另一个盒子中拿若干石子到被清空的盒子中，使得最后两个盒子都不空。当两个盒子中都只有一枚石子时，游戏结束。最后成功执行操作的玩家获胜

(x,y)至少一偶时，先手胜；都为奇时，先手败

6 staircase nim

许多硬币任意分布在楼梯上，共 n 阶楼梯从地面由下向上编号为 0 到 n。游戏者在每次操作时可以将楼梯 j(1<=j<=n)上的任意多但至少一个硬币移动到楼梯 j-1 上。游戏者轮流操作，将最后一枚硬币移至地上的人获胜。

SG=奇数台阶的硬币数 nim 和

7 N 阶 Nim 游戏

有 k 堆石子，各包含 x1,x2…xk 颗石子。双方玩家轮流操作，每次操作选择其中非空的若干堆，至少一堆但不超过 N 堆，在这若干堆中的每堆各取走其中的若干颗石子（1 颗，2 颗…甚至整堆），数目可以不同，取走最后一颗石子的玩家获胜。

当且仅当在每一个不同的二进制位上，x1,x2…xk 中在该位上 1 的个数是 N+1 的倍数时，后手方有必胜策略，否则先手必胜。

8 Nim 积

验题: hdu 3404

```
int m[2][2]={0,0,0,1};
int Nim_Mult_Power(int x,int y){
    if(x<2)
        return m[x][y];
    int a=0;
    for(;;a++)
        if(x>=(1<<(1<<a))&&x<(1<<(1<<(a+1))))
            break;
    int m=1<<(1<<a);
    int p=x/m,s=y/m,t=y%m;
    int d1=Nim_Mult_Power(p,s);
    int d2=Nim_Mult_Power(p,t);
    return (m*(d1^d2))^Nim_Mult_Power(m/2,d1);
}
int Nim_Mult(int x,int y){
    if(x<y)
        return Nim_Mult(y,x);
    if(x<2)
        return m[x][y];
    int a=0;
    for(;;a++)
        if(x>=(1<<(1<<a))&&x<(1<<(1<<(a+1))))
            break;
    int m=1<<(1<<a);
    int p=x/m,q=x%m,s=y/m,t=y%m;
    int
c1=Nim_Mult(p,s),c2=Nim_Mult(p,t)^Nim_Mult(q,s),c3=Ni
m_Mult(q,t);
    return (m*(c1^c2))^c3^Nim_Mult_Power(m/2,c1);
}
int main()
{
    int t;
```

```
    cin >> t;
    while(t--){
        int n;
        scanf("%d" , &n);
        int res = 0;
        while(n--){
            int x , y;
            scanf("%d%d" , &x , &y);
            res ^=Nim_Mult(x , y);
        }
        if(!res)puts("Don't waste your time.");
        else puts("Have a try, lxhgww.");
    }
    return 0;
}
```