目录

Last Edit: 2017 年 12 月 14 日

1. 矩阵 Hash

```
/**
* 查询 500*500 的字符矩阵中是否包含两个子矩阵。
* 求最大子矩阵的边长。
*/
const int MAXN = 510;
int n, m;
char a[MAXN][MAXN];
/**Hash 表**/
const ULL HASH_SIZE = 1000007;
struct HNode {
    ULL hv; int nxt;
} hd[HASH_SIZE];
int head[HASH_SIZE], tot;
bool HQuery(ULL hv) {
    int u = hv % HASH_SIZE;
    assert(u >= 0);
    for(int i = head[u]; ~i; i = hd[i].nxt) {
        if(hd[i].hv == hv) return true;
    }
    return false;
}
void HAdd(ULL hv) {
    int u = hv % HASH_SIZE;
    assert(u >= 0);
    hd[tot].hv = hv;
    hd[tot].nxt = head[u];
    head[u] = tot ++;
}
void HInit() {
    tot = 0;
    memset(head, -1, sizeof(head));
}
/**矩阵 Hash 部分**/
ULL seed[2] = {131, 13331}; /**行列种子不能相同**/
ULL qz[2][MAXN]; /**行列的权值**/
ULL Hash1[MAXN][MAXN]; /**列 Hash**/
ULL Hash2[MAXN][MAXN]; /**再对 Hash1 行 Hash**/
/**
* 差分求子矩阵 Hash。
* (xr, yr) 表示矩阵右下点的坐标
* (nn, mm) 表示矩阵的行高(x 轴)和列宽（y 轴）
*/
inline ULL getHashV(int xr, int yr, int nn, int mm) {
    assert(xr - nn >= 0 && yr - mm >= 0);
    return Hash2[xr][yr] + Hash2[xr - nn][yr - mm] * qz[1][nn] * qz[0][mm]
          - Hash2[xr - nn][yr] * qz[1][nn] - Hash2[xr][yr - mm] * qz[0][mm];
}
bool check(int h) {
    for(int i = h; i <= n; ++i) {
        for(int j = h; j <= m; ++j) {
            ULL hv = getHashV(i, j, h, h);
            if(HQuery(hv)) return true;
            HAdd(hv);
        }
    }
    return false;
}
int main() {
    qz[0][0] = qz[1][0] = 1; // 求权值
    for(int i = 1; i < MAXN; ++i) {
        qz[0][i] = qz[0][i - 1] * seed[0];
        qz[1][i] = qz[1][i - 1] * seed[1];
    }
```

```
    scanf("%d %d", &n, &m);
    HInit();
    for(int i = 1; i <= n; ++i) scanf("%s", a[i] + 1);
    for(int i = 0; i <= n; ++i) Hash1[i][0] = 0;
    for(int j = 0; j <= m; ++j) Hash2[0][j] = 0;
    /**先列 Hash**/
    for(int i = 1; i <= n; ++i) {
        for(int j = 1; j <= m; ++j) {
            Hash1[i][j] = Hash1[i][j - 1] * seed[0] + a[i][j] - 'a';
        }
    }
    /**再行 Hash**/
    for(int i = 1; i <= n; ++i) {
        for(int j = 1; j <= m; ++j) {
            Hash2[i][j] = Hash2[i - 1][j] * seed[1] + Hash1[i][j];
        }
    }
    /**二分长度**/
    int ans = 0, lb = 1, ub = n == m ? n - 1 : min(n, m), md;
    while(lb <= ub) {
        md = (lb + ub) >> 1;
        if(check(md)) ans = md, lb = md + 1;
        else ub = md - 1;
    }
    printf("%d\n", ans);
    return 0;
}
```

2. 树的重心

```
1. 树中所有点到某个点的距离和中，到重心的距离和是最小的；如果有两个重心，那么他们的距离和一样。
2. 把两个树通过一条边相连得到一个新的树，那么新的树的重心在连接原来两个树的重心的路径上。
3. 把一个树添加或删除一个叶子，那么它的重心最多只移动一条边的距离。
int siz[MAXN], mx_sum, g[MAXN], g_cnt;
inline void center_init() {
    g_cnt = 0;
    mx_sum = INF;
}
void center_dfs(int u, int fa) {
    int temp = 0;
    siz[u] = 1;
    for (int i = head[u]; ~i; i = edge[i].next) {
        int v = edge[i].v;
        if (v == fa) continue;
        center_dfs(v, u);
        siz[u] += siz[v];
        umax(temp, siz[v] + 1);
    }
    umax(temp, n - siz[u] + 1);
    if (temp < mx_sum) {
        mx_sum = temp;
        g_cnt = 0;
        g[g_cnt ++] = u;
    } else if (mx_sum == temp) {
        g[g_cnt ++] = u;
    }
}
```

3. 树 Hash

```
/**
 * 无顺序，树的 hash 值只需要取重心的 hv。注意，两个重心的情况。
 */
const ull PA = 13331, PB = 9857877; // 随便取两个不同的数
ull hv[MAXN];int val[MAXN];
```

```
void hash_dfs(int u, int fa) {
    if (fa == -1) hv[u] = PA;
    else hv[u] = (ull)(val[fa] - val[u]) ^ PA;
    for (int i = head[u]; ~i; i = edge[i].next) {
        int v = edge[i].v;
        if (v == fa) continue;
        hash_dfs(v, u);
        hv[u] *= hv[v] ^ PB;
    }
}
```

```
/**
 * 有顺序，树的 hash 值只需要取重心的 hv。注意，两个重心的情况。
 */
ull qz1[MAXN], qz2[MAXN];
void hash_init() {
    for (int i = 0; i < MAXN; ++i) {
        qz1[i] = rand();
        qz2[i] = rand();
    }
}
void hash_dfs1(int u, int fa) {
    hv[u] = val[u];
    int cnt = 0;
    for (int i = head[u]; ~i; i = edge[i].next) {
        int v = edge[i].v;
        if (v == fa) continue;
        hash_dfs1(v, u);
        hv[u] = hv[u] * qz1[++ cnt] + hv[v];
        hv[u] ^= qz2[cnt];
    }
}
```

4. 线性递推拟合

```
/**
 * 根据前若干项求线性递推式。
 * 打表打出前面若干项即可。
 */
typedef long long LL;
typedef vector<int> VI;
#define rep(i,a,n) for (int i=a;i<n;i++)
#define per(i,a,n) for (int i=n-1;i>=a;i--)
#define pb push_back
#define mp make_pair
#define all(x) (x).begin(),(x).end()
#define fi first
#define se second
#define SZ(x) ((int)(x).size())

const LL mod = 1000000007;            /** 注意取模 **/
LL qmod(LL a, LL b) {
    LL res = 1;
    a %= mod;
    assert(b >= 0);
    for(; b; b >>= 1) {
        if(b & 1)res = res * a % mod;
        a = a * a % mod;
    }
    return res;
}
namespace linear_seq {
const int N = 100100;
LL res[N], base[N], _c[N], _md[N];
vector<int> Md;
void mul(LL *a, LL *b, int k) {
    rep(i, 0, k + k) _c[i] = 0;
```

```
    rep(i, 0, k) if (a[i]) rep(j, 0, k) _c[i + j] = (_c[i + j] + a[i] * b[j]) %
mod;
    for (int i = k + k - 1; i >= k; i--) if (_c[i])
            rep(j, 0, SZ(Md)) _c[i - k + Md[j]] = (_c[i - k + Md[j]] - _c[i] *
_md[Md[j]]) % mod;
    rep(i, 0, k) a[i] = _c[i];
}
LL solve(LL n, VI a, VI b) { // a 系数 b 初值 b[n+1]=a[0]*b[n]+...
//        printf("%d\n",SZ(b));
    LL ans = 0, pnt = 0;
    int k = SZ(a);
    assert(SZ(a) == SZ(b));
    rep(i, 0, k) _md[k - 1 - i] = -a[i];
    _md[k] = 1;
    Md.clear();
    rep(i, 0, k) if (_md[i] != 0) Md.push_back(i);
    rep(i, 0, k) res[i] = base[i] = 0;
    res[0] = 1;
    while ((1ll << pnt) <= n) pnt++;
    for(int p = pnt; p >= 0; p--) {
        mul(res, res, k);
        if((n >> p) & 1) {
            for(int i = k - 1; i >= 0; i--) res[i + 1] = res[i];
            res[0] = 0;
            rep(j, 0, SZ(Md)) res[Md[j]] = (res[Md[j]] - res[k] * _md[Md[j]]) % mod;
        }
    }
    rep(i, 0, k) ans = (ans + res[i] * b[i]) % mod;
    if (ans < 0) ans += mod;
    return ans;
}
VI BM(VI s) {
    VI C(1, 1), B(1, 1);
    int L = 0, m = 1, b = 1;
    rep(n, 0, SZ(s)) {
        LL d = 0;
        rep(i, 0, L + 1) d = (d + (LL)C[i] * s[n - i]) % mod;
        if (d == 0) ++m;
        else if (2 * L <= n) {
            VI T = C;
            LL c = mod - d * qmod(b, mod - 2) % mod;
            while (SZ(C) < SZ(B) + m) C.pb(0);
            rep(i, 0, SZ(B)) C[i + m] = (C[i + m] + c * B[i]) % mod;
            L = n + 1 - L, B = T, b = d, m = 1;
        } else {
            LL c = mod - d * qmod(b, mod - 2) % mod;
            while (SZ(C) < SZ(B) + m) C.pb(0);
            rep(i, 0, SZ(B)) C[i + m] = (C[i + m] + c * B[i]) % mod;
            ++m;
        }
    }
    return C;
}
LL gao(VI a, LL n) {
    VI c = BM(a);
    c.erase(c.begin());
    rep(i, 0, SZ(c)) c[i] = (mod - c[i]) % mod;
    return solve(n, c, VI(a.begin(), a.begin() + SZ(c)));
}
};
int main() {
    int T; LL n, ans;
    scanf("%d", &T);
    while(T --) {
        scanf("%d", &n);
        /**输入打出的表，并调整第二个参数**/
```

```
        ans = linear_seq::gao(VI{31, 197, 1255, 7997, 50959, 324725, 2069239,
13185773, 84023455, 535421093}, n - 2);
        printf("%lld\n", ans);
    }
    return 0;
}
```

5. 树上莫队

```
/**
 * 求树上路径上的不同数个数
 */
#include <bits/stdc++.h>
using namespace std;

const int MAXN = 40005;
const int MAXM = 100005;
const int DEEP = 30;

int n, m, block;
struct Edge {
    int v, next;
} edge[MAXN << 1];
/** st 入时间戳, ed 出时间戳, dep 节点深度,  fa 祖先**/
int head[MAXN], tot, st[MAXN], ed[MAXN], tim, dep[MAXN], fa[MAXN][DEEP];
int seq[MAXN << 1]; /* 时间戳对应的顶点序列 */
int w[MAXN], f[MAXN], fsz; /* 权值、离散化数组 */

void init_edge() {
    tim = tot = 0;
    memset(head, -1, sizeof(head));
}
inline void add_edge(int u, int v) {
    edge[tot] = Edge{v, head[u]};
    head[u] = tot ++;
}

void dfs(int u, int pre, int deep) {
    int v;
    dep[u] = deep; fa[u][0] = pre;
    st[u] = ++ tim; seq[tim] = u;
    for(int i = head[u]; ~i; i = edge[i].next) {
        v = edge[i].v;
        if(v == pre || v == u) continue;
        dfs(v, u, deep + 1);
    }
    ed[u] = ++ tim; seq[tim] = u;
}

int lca(int u, int v) {
    while(dep[u] != dep[v]) {
        if(dep[u] < dep[v]) swap(u, v);
        int d = dep[u] - dep[v];
        for(int i = 0; i < DEEP; ++i) {
            if(d >> i & 1) u = fa[u][i];
        }
    }
    if(u == v) return u;
    for(int i = DEEP - 1; i >= 0; --i) {
        if(fa[u][i] != fa[v][i]) {
            u = fa[u][i];
            v = fa[v][i];
        }
    }
    return fa[u][0];
```

```
}

struct Query {
    int L, R, bid, qid; /*bid 表示块编号，qid 表示询问编号 */
    int lca;
    Query() {}
    Query(int _L, int _R, int _lca, int _qid) : L(_L), R(_R), lca(_lca), qid(_qid)
{
        bid = L / block;
    }
    bool operator < (const Query& e) {
        if(bid == e.bid) return R < e.R;
        return bid < e.bid;
    }
} qr[MAXM];
int Ans[MAXM];
/**完成 multiset 操作**/
struct Set {
    int vis[MAXN];
    int sz;
    void clear() { sz = 0; memset(vis, 0, sizeof(vis)); }
    void insert(int x) {
        if(!vis[x]) ++ sz;
        ++ vis[x];
    }
    void erase(int x) {
        -- vis[x];
        if(!vis[x]) -- sz;
    }
    int size() { return sz; }
} path_set;
bool in_path[MAXN]; /**判断节点是否在路径上**/
inline void QModify(int x) {
    int u = seq[x];
    if(in_path[u]) path_set.erase(w[u]);
    else path_set.insert(w[u]);
    in_path[u] ^= 1;
}
int main() {
    int u, v;
    scanf("%d %d", &n, &m);
    fsz = 0;
    for(int i = 1; i <= n; ++i) {
        scanf("%d", &w[i]);
        f[++ fsz] = w[i];
    }
    sort(f + 1, f + fsz + 1);
    fsz = unique(f + 1, f + fsz + 1) - f - 1;
    for(int i = 1; i <= n; ++i)  w[i] = lower_bound(f + 1, f + fsz + 1, w[i]) - f;
    init_edge();
    for(int i = 2; i <= n; ++i) {
        scanf("%d %d", &u, &v);
        add_edge(u, v);
        add_edge(v, u);
    }
    dfs(1, 1, 0);   // dfs(u, pre, deep);
    for(int i = 1; i < DEEP; i++) {
        for(int j = 1; j <= n; j++) {
            fa[j][i] = fa[fa[j][i - 1]][i - 1];
        }
    }
    block = sqrt(n * 2);    // 块大小
    int mm = 0;
    for(int i = 1; i <= m; ++i) {
        scanf("%d %d", &u, &v);
        if(u == v) {
            Ans[i] = 1;
```

```
            continue;
        }
        if(st[u] > st[v]) swap(u, v);
        int p = lca(u, v);
        if(p == u) {
            qr[++ mm] = Query(st[u], st[v], p, i);
        } else {
            qr[++ mm] = Query(ed[u], st[v], p, i);
        }
    }
    sort(qr + 1, qr + mm + 1);
    int L = 1, R = 0;
    memset(in_path, 0, sizeof(in_path));
    path_set.clear();

    for(int i = 1; i <= mm; ++i) {
        while(L > qr[i].L) QModify(--L);    //ins
        while(R < qr[i].R) QModify(++R);   //ins
        while(L < qr[i].L) QModify(L++);   //del
        while(R > qr[i].R) QModify(R--);   //del
        if(qr[i].lca != seq[qr[i].L] && qr[i].lca != seq[qr[i].R])
QModify(st[qr[i].lca]);  // ins lca
        Ans[qr[i].qid] = path_set.size();
        if(qr[i].lca != seq[qr[i].L] && qr[i].lca != seq[qr[i].R])
QModify(st[qr[i].lca]);  // del lca
    }
    for(int i = 1; i <= m; ++i) printf("%d\n", Ans[i]);
    return 0;
}
```

6. 全局最小割

```
/*
Poj 2914
Stoer_Wagner: 复杂度 O(n^3)
1. 设最小割 cut=INF, 任选一个点 s 到集合 A 中, 定义 W(A, p)为 A 中的所有点到 A 外一点 p 的权总和.
2. 对刚才选定的 s, 更新 W(A,p)(该值递增).
3. 选出 A 外一点 p, 且 W(A,p)最大的作为新的 s, 若 A!=G(V), 则继续 2.
4. 把最后进入 A 的两点记为 s 和 t, 用 W(A,t)更新 cut.
5. 合并 st，即新建顶点 u, 边权 w(u, v)=w(s, v)+w(t, v), 删除顶点 s 和 t, 以及与它们相连的边.
6. 若|V|!=1 则继续 1.
*/
const int MAXN = 510;
const int INF = 0x3f3f3f3f;
int g[MAXN][MAXN];
int vis[MAXN];// 判断 i 是否加入 A 集合中
int w[MAXN];  // w[i] 代表 A 集合中所有点到 i 点的距离
int v[MAXN];  // v[i] 代表 i 点所合并到的点
int Stoer_Wagner(int n) {
    int ret = INF;
    for(int i = 0; i < n; i++)v[i] = i;
    while(n > 1) {
        memset(w, 0, sizeof(w));
        memset(vis, 0, sizeof(vis));
        int pre = 0;
        for(int i = 1; i < n; i++) {
            int k = -1;
            for(int j = 1; j < n; j++) { // 寻找"距离"A 集合最大的点
                if(!vis[v[j]]) {
                    w[v[j]] += g[v[pre]][v[j]];
                    if(k == -1 || w[v[j]] > w[v[k]])k = j;
                }
            }
            vis[v[k]] = 1;
            if(i == n - 1) {
                int s = v[pre], t = v[k];
```

```
            ret = min(ret, w[t]);
            for(int j = 0; j < n; j++) {
                g[s][v[j]] += g[v[j]][t];
                g[v[j]][s] += g[v[j]][t];
            }
            v[k] = v[--n];
        }
        pre = k;
    }
}
    return ret;
}
int main() {
    int n, m;
    while(~scanf("%d%d", &n, &m)) {
        memset(g, 0, sizeof(g));
        for(int i = 0; i < m; i++) {
            int u, v, w;
            scanf("%d%d%d", &u, &v, &w);
            g[u][v] += w;
            g[v][u] += w;
        }
        int ans = Stoer_Wagner(n);
        printf("%d\n", ans);
    }
}
```

7. 行列式

```
/*整数行列式（取模）*/
LL det(int dim) {
    LL ans = 1;
    for(int k = 1; k <= dim; k++) {
        LL pos = -1;
        for(int i = k; i <= dim; i++)
            if(mat[i][k]) {
                pos = i;
                break;
            }
        if(pos == -1) return 0;
        if(pos != k)
            for(int j = k; j <= dim; j++) swap(mat[pos][j], mat[k][j]);
        LL inv = qmod(mat[k][k], MOD - 2);
        for(int i = k + 1; i <= dim; i++)
            if(mat[i][k]) {
                ans = ans * inv % MOD;
                for(int j = k + 1; j <= dim; j++)
                    mat[i][j] = ((mat[i][j] * mat[k][k] % MOD - mat[k][j] *
mat[i][k] % MOD) % MOD + MOD) % MOD;
                mat[i][k] = 0;
            }
    }
    for(int i = 1; i <= dim; i++) ans = ans * mat[i][i] % MOD;
    return ans;
}
```

```
/*浮点行列式*/
double Det(int dim) {
    double ans = 1;
    int cur = 1, sgn = 1;
    for(int i = 0; i <= dim; ++ i) {
        int nxt = -1;
        for(int j = cur; j <= dim; ++ j) if(fabs(mat[j][i]) > 1e-6) {
                nxt = j;
                break;
            }
        if(nxt == -1) continue;
```

```
        for( int j = 1; j <= dim; ++ j) swap(mat[cur][j], mat[nxt][j]);
        if(nxt != cur) sgn = - sgn;
        for(int j = 1; j <= dim ; ++ j) if(j != cur) {
              double s = mat[j][i] / mat[cur][i];
              for(int k = i; k <= N; ++ k) mat[j][k] -= s * mat[cur][k];
           }
        ++ cur;
    }
    for(int i = 1; i <= dim; ++ i) ans = ans * mat[i][i];
    return ans * sgn;}
```

8. 其他公式

降幂公式

1. A^x = A^(x % Phi(C) + Phi(C)) (mod C)，其中 x≥Phi(C)　这个降幂公式适用于 C 不是素数的情况
2. A^X % C = A ^ (X % (C - 1))　这个降幂公式只适用于 C 是素数的情况

O(1) 快速乘

```
// P 为模数
LL mul(LL a, LL b, LL p) {
    a = a % p, b = b % p;
    return ((a * b - (LL)(((long double)a * b + 0.5) / p) * p) % p + p) % p;
}
```

几何公式:

三角形:

1. 半周长 P=(a+b+c)/2
2. 面积 S=aHa/2=absin(C)/2=sqrt(P(P-a)(P-b)(P-c))
3. 中线 Ma=sqrt(2(b^2+c^2)-a^2)/2=sqrt(b^2+c^2+2bccos(A))/2
4. 角平分线 Ta=sqrt(bc((b+c)^2-a^2))/(b+c)=2bccos(A/2)/(b+c)
5. 高线 Ha=bsin(C)=csin(B)=sqrt(b^2-((a^2+b^2-c^2)/(2a))^2)
6. 内切圆半径 r=S/P=asin(B/2)sin(C/2)/sin((B+C)/2)
   =4Rsin(A/2)sin(B/2)sin(C/2)=sqrt((P-a)(P-b)(P-c)/P)
   =Ptan(A/2)tan(B/2)tan(C/2)
7. 外接圆半径 R=abc/(4S)=a/(2sin(A))=b/(2sin(B))=c/(2sin(C))

四边形:

D1,D2 为对角线,M 对角线中点连线,A 为对角线夹角

1. a^2+b^2+c^2+d^2=D1^2+D2^2+4M^2
2. S=D1D2sin(A)/2

(以下对圆的内接四边形)

3. ac+bd=D1D2
4. S=sqrt((P-a)(P-b)(P-c)(P-d)),P 为半周长

正 n 边形:

R 为外接圆半径,r 为内切圆半径

1. 中心角 A=2PI/n
2. 内角 C=(n-2)PI/n
3. 边长 a=2sqrt(R^2-r^2)=2Rsin(A/2)=2rtan(A/2)
4. 面积 S=nar/2=nr^2tan(A/2)=nR^2sin(A)/2=na^2/(4tan(A/2))

圆:

1. 弧长 l=rA
2. 弦长 a=2sqrt(2hr-h^2)=2rsin(A/2)
3. 弓形高 h=r-sqrt(r^2-a^2/4)=r(1-cos(A/2))=atan(A/4)/2
4. 扇形面积 S1=rl/2=r^2A/2
5. 弓形面积 S2=(rl-a(r-h))/2=r^2(A-sin(A))/2

棱柱:

1. 体积 V=Ah,A 为底面积,h 为高
2. 侧面积 S=lp,l 为棱长,p 为直截面周长
3. 全面积 T=S+2A

棱锥:

1. 体积 V=Ah/3,A 为底面积,h 为高

(以下对正棱锥)

```
2. 侧面积 S=lp/2,l 为斜高,p 为底面周长
3. 全面积 T=S+A

棱台:
1. 体积 V=(A1+A2+sqrt(A1A2))h/3,A1.A2 为上下底面积,h 为高
(以下为正棱台)
2. 侧面积 S=(p1+p2)l/2,p1.p2 为上下底面周长,l 为斜高
3. 全面积 T=S+A1+A2

圆柱:
1. 侧面积 S=2PIrh
2. 全面积 T=2PIr(h+r)
3. 体积 V=PIr^2h

圆锥:
1. 母线 l=sqrt(h^2+r^2)
2. 侧面积 S=PIrl
3. 全面积 T=PIr(l+r)
4. 体积 V=PIr^2h/3

圆台:
1. 母线 l=sqrt(h^2+(r1-r2)^2)
2. 侧面积 S=PI(r1+r2)l
3. 全面积 T=PIr1(l+r1)+PIr2(l+r2)
4. 体积 V=PI(r1^2+r2^2+r1r2)h/3

球:
1. 全面积 T=4PIr^2
2. 体积 V=4PIr^3/3

球台:
1. 侧面积 S=2PIrh
2. 全面积 T=PI(2rh+r1^2+r2^2)
3. 体积 V=PIh(3(r1^2+r2^2)+h^2)/6

球扇形:
1. 全面积 T=PIr(2h+r0),h 为球冠高,r0 为球冠底面半径
2. 体积 V=2PIr^2h/3
```

9. pbds 大法

```
#include <ext/pb_ds/priority_queue.hpp>

gnu_pbds::priority_queue<int> Q;
优先队列，配对堆默认，从小到大!

gnu_pbds::priority_queue < int , greater < int > , pairing_heap_tag  > Q;
gnu_pbds::priority_queue < int , greater < int > , pairing_heap_tag  > ::
point_iterator id[ maxn ];

id[x] = Q.push(5) ;
Q.modify(id[x], 6) ; //直接修改

支持 join , push , pop 操作

#include <ext/pb_ds/assoc_container.hpp>

using namespace gnu_pbds;

tree<int,null_type,less<int>,rb_tree_tag,tree_order_statistics_node_update> rbt;
tree<int,null_type,less<int>,rb_tree_tag,tree_order_statistics_node_update> ::
iterator it ;

find_by_order(size_type order) 找第 order+1 小的元素的迭代器
```

```
order_of_key(int val) 问有多少个比val小

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/hash_policy.hpp>

gnu_pbds::gp_hash_table<key, value> hs; 哈希 支持[]和find操作
```

10. Polya 定理

```
polya 定理（染色有使用次数限制）
void init() {
    int n = 40;
    c[0][0] = c[1][0] = c[1][1] = 1;
    for(int i = 2; i <= n; i++) {
        c[i][0] = 1;
        for(int j = 1; j <= i; j++) {
            c[i][j] = c[i - 1][j - 1] + c[i - 1][j];
        }
    }
}

LL calcul(int m) { //m为循环的长度
    int n = 0;
    LL ret = 1;
    for(int i = 0; i < 3; i++) {
        if(b[i] % m != 0)
            return 0;
        b[i] /= m;
        n += b[i];
    }
    for(int i = 0; i < 3; i++) {
        ret *= c[n][b[i]];
        n -= b[i];
    }
    return ret;
}
```

11. 树状数组求区间最值

```
int lowbit(int x) { return x & (-x); }
void update(int x) {
    int lx, i;
    while (x <= n) {
        h[x] = a[x];
        lx = lowbit(x);
        for (i = 1; i < lx; i <<= 1)
            h[x] = max(h[x], h[x - i]);
        x += lowbit(x);
    }
}
int query(int x, int y) {
    int ans = 0;
    while (y >= x) {
        ans = max(a[y], ans);
        y --;
        for (; y - lowbit(y) >= x; y -= lowbit(y))
            ans = max(h[y], ans);
    }
    return ans;
}
```

12. 斯坦纳生成树

```
/*  Steiner Tree：求，使得指定K个点连通的生成树的最小总权值
```

```
 *  st[i] 表示顶点 i 的标记值，如果 i 是指定集合内第 m(0<=m<K)个点，则 st[i]=1<<m
 *  endSt=1<<K
 *  dptree[i][state] 表示以 i 为根，连通状态为 state 的生成树值
 */
#define CLR(x,a) memset(x,a,sizeof(x))
int dptree[N][1 << K], st[N], endSt;
bool vis[N][1 << K];
queue<int> que;

int input() {
    /*   输入，并且返回指定集合元素个数 K
     *   因为有时候元素个数需要通过输入数据处理出来，所以单独开个输入函数。
     */
}

void initst_Tree() {
    memset(dp, -1, sizeof(dp));
    memset(st, 0, sizeof(st));
    for(int i = 0; i <= n; i++) memset(vis[i], 0, sizeof(vis[i]));
    ed = 1 << input();
    for(int i = 0; i <= n; i++)
        dp[i][st[i]] = 0;
}

void update(int &a, int x) {
    a = (a > x || a == -1) ? x : a;
}

void SPFA(int s) {
    while(!que.empty()) {
        int u = que.front();
        que.pop();
        vis[u][s] = false;
        for(int i = head[u]; i != -1; i = e[i].nxt) {
            int v = e[i].v;
            if(dp[v][st[v] | s] == -1 || dp[v][st[v] | s] > dp[u][s] + e[i].w) {
                dp[v][st[v] | s] = dp[u][s] + e[i].w;
                if(st[v] | s != s || vis[v][s])
                    continue; //只更新当前连通状态
                vis[v][s] = true;
                que.push(v);
            }
        }
    }
}

void st_Tree() {
    for(int j = 0; j < ed; j++) {
        for(int i = 0; i <= n; i++) {
            if(st[i] && (st[i]&j) == 0) continue;
            for(int s = (j - 1)&j; s; s = (s - 1)&j) {
                int x = st[i] | s, y = st[i] | (j - s);
                if(dp[i][x] != -1 && dp[i][y] != -1)
                    update(dp[i][j], dp[i][x] + dp[i][y]);
            }
            if(dp[i][j] != -1)
                que.push(i), vis[i][j] = true;
        }
        SPFA(j);
    }
}
```

13. cdq 分治+树状数组

```
struct node {
    int type, id, x, y, aid;
```

```
    node() {}
    node(int t, int i, int xx, int yy, int ai) {
        type = t;
        id = i;
        x = xx;
        y = yy;
        aid = ai;
    }
    bool operator < (const node & rhs) const {
        if(id == rhs.id)
            return type <  rhs.type;
        return id < rhs.id;
    }
} que[maxm], temp[maxm];

void cdq(int l, int r) {
    if(l == r) return;
    int mid = l + r >> 1;
    cdq(lson);  cdq(rson);
    int p = l,  q = mid + 1, now = l;
    while(p <= mid && q <= r) {
        if(que[p].x >= que[q].x) {
            if(que[p].type == -2) add(que[p].y, 1);
            temp[now++] = que[p++];
        } else {
            if(que[q].type == -1 || que[q].type == 1)
                ans[que[q].aid] += que[q].type * (query(maxn) - query(que[q].y - 1));
            temp[now++] = que[q++];
        }
    }

    while(p <= mid) temp[now++] = que[p++];
    while(q <= r) {
        if(que[q].type == -1 || que[q].type == 1)
            ans[que[q].aid] += que[q].type * (query(maxn) - query(que[q].y - 1));
        temp[now++] = que[q++];
    }

    for(int i = l; i <= r; i++) {
        que[i] = temp[i];
        if(temp[i].type == -2) Clear(que[i].y);
    }
}
```

14. KDTree

```
const int N = 2e5 + 10;

typedef long long LL;
#define rep(i, x, y) for (int i = (x), _ = (y); i <= _; ++i)
#define down(i, x, y) for (int i = (x), _ = (y); i >= _; --i)
template<typename T> inline void up_max(T & x, T y) { x < y ? x = y : 0; }
template<typename T> inline void up_min(T & x, T y) { x > y ? x = y : 0; }
namespace KD_Tree {
struct node {
    node *ch[2];
    int d[2], mx[2], my[2], size, val;
    LL sum;

    inline void push_up() {
        sum = val, size = 1;
        rep (i, 0, 1) if (ch[i]) {
            sum += ch[i]->sum, size += ch[i]->size;
            up_min(mx[0], ch[i]->mx[0]);
            up_max(mx[1], ch[i]->mx[1]);
```

```
            up_min(my[0], ch[i]->my[0]);
            up_max(my[1], ch[i]->my[1]);
        }
    }
} pool_node[N], *pool_top = pool_node;

node *del_pool[N], **del_top = del_pool;

inline node * newnode() {
    return del_top == del_pool ? ++pool_top : *(del_top--);
}

bool cmp_D;

struct Point {
    int d[2], val;
    inline bool operator < (const Point & b) const {
        return d[cmp_D] < b.d[cmp_D];
    }
} p[N];

int top = 0;

node *build(int l, int r, bool f) {
    node *o = newnode();
    int mid = (l + r) >> 1;
    cmp_D = f;
    nth_element(p + l, p + mid, p + r + 1);

    o->mx[0] = o->mx[1] = o->d[0] = p[mid].d[0];
    o->my[0] = o->my[1] = o->d[1] = p[mid].d[1];
    o->val = p[mid].val;
    o->ch[0] = l < mid ? build(l, mid - 1, f ^ 1) : 0;
    o->ch[1] = mid < r ? build(mid + 1, r, f ^ 1) : 0;
    o->push_up();

    return o;
}

void remove(node *o) {
    if (o->ch[0])
        remove(o->ch[0]);
    if (o->ch[1])
        remove(o->ch[1]);

    p[++top].val = o->val;
    p[top].d[0] = o->d[0], p[top].d[1] = o->d[1];
    *(++del_top) = o;
}

node ** rebuild_need;
bool rebuild_d;

void rebuild(node ** o) {
    top = 0, remove(*o);
    *o = build(1, top, rebuild_d);
}

void insert(node *&o, int x, int y, int v, bool f) {
    if (!o) {
        o = newnode();
        o->d[0] = o->mx[0] = o->mx[1] = x;
        o->d[1] = o->my[0] = o->my[1] = y;
        o->val = o->sum = v;
    } else if (o->d[0] == x && o->d[1] == y)
        o->sum += v, o->val += v;
    else {
```

```
        int d = !f ? o->d[0] < x : o->d[1] < y;
        insert(o->ch[d], x, y, v, f ^ 1);
        o->push_up();
        if (o->ch[d]->size * 10 >= o->size * 7)
            rebuild_need = &o, rebuild_d = f;
    }
}

inline bool in(int x1, int y1, int x2, int y2, int a1, int b1, int a2, int b2) {
    return a1 <= x1 && b1 <= y1 && x2 <= a2 && y2 <= b2;
}
inline bool out(int x1, int y1, int x2, int y2, int a1, int b1, int a2, int b2) {
    return x2 < a1 || a2 < x1 || b2 < y1 || y2 < b1;
}

LL query(node *o, int x1, int y1, int x2, int y2) {
    if (!o || out(x1, y1, x2, y2, o->mx[0], o->my[0], o->mx[1], o->my[1]))
        return 0;
    if (in(o->mx[0], o->my[0], o->mx[1], o->my[1], x1, y1, x2, y2))
        return o->sum;
    LL ret = in(o->d[0], o->d[1], o->d[0], o->d[1], x1, y1, x2, y2) ? o->val : 0;
    return ret + query(o->ch[0], x1, y1, x2, y2) + query(o->ch[1], x1, y1, x2, y2);
}
}
```

15. 上下界网络流模型

1. 无源汇上下界可行流：

a) 建立附加源点 ss 和附加汇点 tt；
b) 对于原图中的边 x->y，若限制为[b,c]，那么连边 x->y，流量为 c-b；
c) 对于原图中的某一个点 i，记 d(i)为流入这个点的所有边的下界和减去流出这个点的所有边的下界和。若 d(i)>0，那么连边 ss->i，流量为 d(i)；若 d(i)<0，那么连边 i->tt，流量为-d(i)
d) 跑一次最大流，若新图满流，则一定存在一种可行流。此时，原图中每一条边的流量应为新图中对应的边的流量+这条边的流量下界

---

2. 有源汇上下界可行流：

a) 在原图中添加一条边 t->s，流量限制为[0,inf]. 即让源点和汇点也满足流量平衡条件，这样就改造成了无源汇的网络流图。
b) 其余方法同上。

---

3. 有源汇上下界最大流：

a) 建图方法同"有源汇上下界可行流"；
b) 在新图上跑 ss 到 tt 的最大流；若新图满流，那么一定存在一种可行流。记此时$\sum f(s,i) = sum1$，将 t->s 这条边拆掉，在新图上跑 s 到 t 的最大流；记此时$\sum f(s,i) = sum2$，最终答案即为$\sum sum1 + sum2$.

---

4. 有源汇上下界最小流：

a) 建图方法同"有源汇上下界可行流"；
b) 求 ss->tt 最大流，连边 t->s,容量为 inf, 求 ss->tt 最大流. 答案即为边 t->s,inf 这条边的实际流量.

---

5. 有源汇上下界费用流：

a) 首先建立附加源点 ss 和附加汇点 tt；
b) 对于原图中的边 x->y，若限制为[b,c]，费用为 cost，那么连边 x->y，流量为 c-b，费用为 cost；

c） 对于原图中的某一个点 i，记 d(i)为流入这个点的所有边的下界和减去流出这个点的所有边的下界和。若 d(i)>0，那么连边 ss->i，流量为 d(i)，费用为 0；若 d(i)<0，那么连边 i->tt，流量为-d(i)，费用为 0；
d） 连边 t->s，流量为 inf，费用为 0
e） 跑 ss->tt 的最小费用最大流，答案即为（求出的费用+原图中边的下界*边的费用）。

注意：有上下界的费用流指的是在满足流量限制条件和流量平衡条件的情况下的最小费用流。而不是在满足流量限制条件和流量平衡条件并且满足最大流的情况下的最小费用流。也就是说，有上下界的费用流只需要满足网络流的条件就可以了，而普通的费用流是满足一般条件并且满足是最大流的基础上的最小费用。

16. 循环后缀数组

```
const int MX = 2e5 + 5;
char s[MX];
int SA[MX], R[MX], H[MX];
int wa[MX], wb[MX], wv[MX], wc[MX];
int nxt[MX],nn[2][MX];
queue <int> que[MX];
bool cmp(int *r, int a, int b, int a2, int b2) {
    return r[a] == r[b] && r[a2] == r[b2];
}
void Suffix(char *r, int m = 128) {
    int n = strlen(r) + 1, cur = 0;
    int i, j, p, *x = wa, *y = wb, *t;
    for(i = 0; i < m; i++) wc[i] = 0;
    for(i = 0; i < n; i++) wc[x[i] = r[i]]++;
    for(i = 1; i < m; i++) wc[i] += wc[i - 1];
    for(i = n - 1; i >= 0; i--) SA[--wc[x[i]]] = i;
    for(j = 1, p = 1; j <= n; j *= 2, m = p) {
        for(i = 0; i < n; i++) que[nn[cur][i]].push(i);
        for(i = 0, p = 0; i < n; i++){
            while(que[SA[i]].size()) {
                y[p++] = que[SA[i]].front();
                que[SA[i]].pop();
            }
        }
        for(i = 0; i < n; i++) wv[i] = x[y[i]];
        for(i = 0; i < m; i++) wc[i] = 0;
        for(i = 0; i < n; i++) wc[wv[i]]++;
        for(i = 1; i < m; i++) wc[i] += wc[i - 1];
        for(i = n - 1; i >= 0; i--) SA[--wc[wv[i]]] = y[i];
        for(t = x, x = y, y = t, p = 1, x[SA[0]] = 0, i = 1; i < n; i++) {
            x[SA[i]] = cmp(y, SA[i - 1], SA[i], nn[cur][SA[i-1]], nn[cur][SA[i]]) ? 
p - 1 : p++;
        }
        for(i = 0; i < n; i++) nn[cur^1][i] = nn[cur][nn[cur][i]];
        cur ^= 1;
    }
}
```

17. 一般图匹配

```
//输入格式
//第一行两个正整数，n,m。保证 n≥2。n 为点数，m 为边数
//接下来 m 行，每行两个整数 v,u 表示 uv 之间有边。保证 1≤v,u≤n，保证 v≠u,保证同一个条件不会出现
两次。
//输出格式
//第一行一个整数，表示最多产生多少个匹配。
//接下来一行 n 个整数，描述一组最优方案。第 v 个整数表示 v 号点匹配点的编号，如果 v 号没有被匹配输
出 0
#include <cstdio>
```

```
#include <cstring>
#include <algorithm>
#define M 250010
using namespace std;
char inp[33554432], *inpch = inp;
int Head[M], Next[M], Go[M], Pre[510], Nxt[510], F[510], S[510], Q[510], Vis[510],
*Top = Q, Cnt = 0, Tim = 0, n, m, x, y;
inline void addedge(int x, int y) {
    Go[++Cnt] = y;
    Next[Cnt] = Head[x];
    Head[x] = Cnt;
}
int find(int x) {
    return x == F[x] ? x : F[x] = find(F[x]);
}
int lca(int x, int y) {
    for(Tim++, x = find(x), y = find(y); ; x ^= y ^= x ^= y)
        if(x) {
            if(Vis[x] == Tim) return x;
            Vis[x] = Tim;
            x = find(Pre[Nxt[x]]);
        }
}
void blossom(int x, int y, int l) {
    while(find(x) != l) {
        Pre[x] = y;
        if(S[Nxt[x]] == 1) S[*Top = Nxt[x]] = 0, *Top++;
        if(F[x] == x) F[x] = l;
        if(F[Nxt[x]] == Nxt[x]) F[Nxt[x]] = l;
        y = Nxt[x];
        x = Pre[y];
    }
}
int Match(int x) {
    for(int i = 1; i <= n; i++) F[i] = i;
    memset(S, -1, sizeof S);
    S[*(Top = Q) = x] = 0, Top++;
    for(int *i = Q; i != Top; *i++)
        for(int T = Head[*i]; T; T = Next[T]) {
            int g = Go[T];
            if(S[g] == -1) {
                Pre[g] = *i, S[g] = 1;
                if(!Nxt[g]) {
                    for(int u = g, v = *i, lst; v; u = lst, v = Pre[u])
                        lst = Nxt[v], Nxt[v] = u, Nxt[u] = v;
                    return 1;
                }
                S[*Top = Nxt[g]] = 0, *Top++;
            } else if(!S[g] && find(g) != find(*i)) {
                int l = lca(g, *i);
                blossom(g, *i, l);
                blossom(*i, g, l);
            }
        }
    return 0;
}
inline void Read(int& x) {
    x = 0;
    while(*inpch < '0') *inpch++;
    while(*inpch >= '0') x = x * 10 + *inpch++ - '0';
}
int main() {
    fread(inp, 1, 33554432, stdin);
    Read(n), Read(m);
    for(int i = 1; i <= m; i++) {
        Read(x), Read(y);
        addedge(x, y);
```

```
        addedge(y, x);
    }
    int ans = 0;
    for(int i = n; i >= 1; i--)
        if(!Nxt[i]) ans += Match(i);
    printf("%d\n", ans);
    for(int i = 1; i <= n; i++) printf("%d ", Nxt[i]);
    putchar('\n');
    return 0;
}
```

18. 一般图最大权匹配

```
//输入格式
//第一行两个正整数，n,m。保证 n≥2。n 为点数，m 为边数
//接下来 m 行，每行两个整数 v,u，w 表示 uv 之间有边，边权为 w。保证 1≤v,u≤n，保证 v≠u,保证同一个条件不会出现两次。
//输出格式
//第一行一个整数，表示最大权。
//接下来一行 n 个整数，描述一组最优方案。第 v 个整数表示 v 号点匹配点的编号，如果 v 号没有被匹配输出 0
#include<bits/stdc++.h>
using namespace std;
//from vfleaking
//自己進行一些進行一些小修改
#define INF INT_MAX
#define MAXN 400
struct edge {
    int u,v,w;
    edge() {}
    edge(int u,int v,int w):u(u),v(v),w(w) {}
};
int n,n_x;
edge g[MAXN*2+1][MAXN*2+1];
int lab[MAXN*2+1];
int match[MAXN*2+1],slack[MAXN*2+1],st[MAXN*2+1],pa[MAXN*2+1];
int flower_from[MAXN*2+1][MAXN+1],S[MAXN*2+1],vis[MAXN*2+1];
vector<int> flower[MAXN*2+1];
queue<int> q;
inline int e_delta(const edge &e) { // does not work inside blossoms
    return lab[e.u]+lab[e.v]-g[e.u][e.v].w*2;
}
inline void update_slack(int u,int x) {
    if(!slack[x]||e_delta(g[u][x])<e_delta(g[slack[x]][x]))slack[x]=u;
}
inline void set_slack(int x) {
    slack[x]=0;
    for(int u=1; u<=n; ++u)
        if(g[u][x].w>0&&st[u]!=x&&S[st[u]]==0)update_slack(u,x);
}
void q_push(int x) {
    if(x<=n)q.push(x);
    else for(size_t i=0; i<flower[x].size(); i++)q_push(flower[x][i]);
}
inline void set_st(int x,int b) {
    st[x]=b;
    if(x>n)for(size_t i=0; i<flower[x].size(); ++i)
            set_st(flower[x][i],b);
}
inline int get_pr(int b,int xr) {
    int pr=find(flower[b].begin(),flower[b].end(),xr)-flower[b].begin();
    if(pr%2==1) { //檢查他在前一層圖是奇點還是偶點
        reverse(flower[b].begin()+1,flower[b].end());
        return (int)flower[b].size()-pr;
    } else return pr;
}
```

```
inline void set_match(int u,int v) {
    match[u]=g[u][v].v;
    if(u>n) {
        edge e=g[u][v];
        int xr=flower_from[u][e.u],pr=get_pr(u,xr);
        for(int i=0; i<pr; ++i)set_match(flower[u][i],flower[u][i^1]);
        set_match(xr,v);
        rotate(flower[u].begin(),flower[u].begin()+pr,flower[u].end());
    }
}
inline void augment(int u,int v) {
    for(;;) {
        int xnv=st[match[u]];
        set_match(u,v);
        if(!xnv)return;
        set_match(xnv,st[pa[xnv]]);
        u=st[pa[xnv]],v=xnv;
    }
}
inline int get_lca(int u,int v) {
    static int t=0;
    for(++t; u||v; swap(u,v)) {
        if(u==0)continue;
        if(vis[u]==t)return u;
        vis[u]=t;//這種方法可以不用清空 v 陣列
        u=st[match[u]];
        if(u)u=st[pa[u]];
    }
    return 0;
}
inline void add_blossom(int u,int lca,int v) {
    int b=n+1;
    while(b<=n_x&&st[b])++b;
    if(b>n_x)++n_x;
    lab[b]=0,S[b]=0;
    match[b]=match[lca];
    flower[b].clear();
    flower[b].push_back(lca);
    for(int x=u,y; x!=lca; x=st[pa[y]])
        flower[b].push_back(x),flower[b].push_back(y=st[match[x]]),q_push(y);
    reverse(flower[b].begin()+1,flower[b].end());
    for(int x=v,y; x!=lca; x=st[pa[y]])
        flower[b].push_back(x),flower[b].push_back(y=st[match[x]]),q_push(y);
    set_st(b,b);
    for(int x=1; x<=n_x; ++x)g[b][x].w=g[x][b].w=0;
    for(int x=1; x<=n; ++x)flower_from[b][x]=0;
    for(size_t i=0; i<flower[b].size(); ++i) {
        int xs=flower[b][i];
        for(int x=1; x<=n_x; ++x)
            if(g[b][x].w==0||e_delta(g[xs][x])<e_delta(g[b][x]))
                g[b][x]=g[xs][x],g[x][b]=g[x][xs];
        for(int x=1; x<=n; ++x)
            if(flower_from[xs][x])flower_from[b][x]=xs;
    }
    set_slack(b);
}
inline void expand_blossom(int b) { // S[b] == 1
    for(size_t i=0; i<flower[b].size(); ++i)
        set_st(flower[b][i],flower[b][i]);
    int xr=flower_from[b][g[b][pa[b]].u],pr=get_pr(b,xr);
    for(int i=0; i<pr; i+=2) {
        int xs=flower[b][i],xns=flower[b][i+1];
        pa[xs]=g[xns][xs].u;
        S[xs]=1,S[xns]=0;
        slack[xs]=0,set_slack(xns);
        q_push(xns);
    }
```

```
    S[xr]=1,pa[xr]=pa[b];
    for(size_t i=pr+1; i<flower[b].size(); ++i) {
        int xs=flower[b][i];
        S[xs]=-1,set_slack(xs);
    }
    st[b]=0;
}
inline bool on_found_edge(const edge &e) {
    int u=st[e.u],v=st[e.v];
    if(S[v]==-1) {
        pa[v]=e.u,S[v]=1;
        int nu=st[match[v]];
        slack[v]=slack[nu]=0;
        S[nu]=0,q_push(nu);
    } else if(S[v]==0) {
        int lca=get_lca(u,v);
        if(!lca)return augment(u,v),augment(v,u),true;
        else add_blossom(u,lca,v);
    }
    return false;
}
inline bool matching() {
    memset(S+1,-1,sizeof(int)*n_x);
    memset(slack+1,0,sizeof(int)*n_x);
    q=queue<int>();
    for(int x=1; x<=n_x; ++x)
        if(st[x]==x&&!match[x])pa[x]=0,S[x]=0,q_push(x);
    if(q.empty())return false;
    for(;;) {
        while(q.size()) {
            int u=q.front(); q.pop();
            if(S[st[u]]==1)continue;
            for(int v=1; v<=n; ++v)
                if(g[u][v].w>0&&st[u]!=st[v]) {
                    if(e_delta(g[u][v])==0) {
                        if(on_found_edge(g[u][v]))return true;
                    } else update_slack(u,st[v]);
                }
        }
        int d=INF;
        for(int b=n+1; b<=n_x; ++b)
            if(st[b]==b&&S[b]==1)d=min(d,lab[b]/2);
        for(int x=1; x<=n_x; ++x)
            if(st[x]==x&&slack[x]) {
                if(S[x]==-1)d=min(d,e_delta(g[slack[x]][x]));
                else if(S[x]==0)d=min(d,e_delta(g[slack[x]][x])/2);
            }
        for(int u=1; u<=n; ++u) {
            if(S[st[u]]==0) {
                if(lab[u]<=d)return 0;
                lab[u]-=d;
            } else if(S[st[u]]==1)lab[u]+=d;
        }
        for(int b=n+1; b<=n_x; ++b)
            if(st[b]==b) {
                if(S[st[b]]==0)lab[b]+=d*2;
                else if(S[st[b]]==1)lab[b]-=d*2;
            }
        q=queue<int>();
        for(int x=1; x<=n_x; ++x)
            if(st[x]==x&&slack[x]&&st[slack[x]]!=x&&e_delta(g[slack[x]][x])==0)
                if(on_found_edge(g[slack[x]][x]))return true;
        for(int b=n+1; b<=n_x; ++b)
            if(st[b]==b&&S[b]==1&&lab[b]==0)expand_blossom(b);
    }
    return false;
}
```

```
inline pair<long long,int> weight_blossom() {
    memset(match+1,0,sizeof(int)*n);
    n_x=n;
    int n_matches=0;
    long long tot_weight=0;
    for(int u=0; u<=n; ++u)st[u]=u,flower[u].clear();
    int w_max=0;
    for(int u=1; u<=n; ++u)
        for(int v=1; v<=n; ++v) {
            flower_from[u][v]=(u==v?u:0);
            w_max=max(w_max,g[u][v].w);
        }
    for(int u=1; u<=n; ++u)lab[u]=w_max;
    while(matching())++n_matches;
    for(int u=1; u<=n; ++u)
        if(match[u]&&match[u]<u)
            tot_weight+=g[u][match[u]].w;
    return make_pair(tot_weight,n_matches);
}
inline void init_weight_graph() {
    for(int u=1; u<=n; ++u)
        for(int v=1; v<=n; ++v)
            g[u][v]=edge(u,v,0);
}
int main() {
    int m;
    scanf("%d%d",&n,&m);
    init_weight_graph();
    for(int i=0; i<m; ++i) {
        int u,v,w;
        scanf("%d%d%d",&u,&v,&w);
        g[u][v].w=g[v][u].w=w;
    }
    printf("%lld\n",weight_blossom().first);
    for(int u=1; u<=n; ++u)printf("%d ",match[u]); puts("");
    return 0;
}
```

19. 大数模板

```
const int base = 1000000000;
const int base_digits = 9;

struct bigint {
    vector<int> z;
    int sign;

    bigint() : sign(1) { }

    bigint(long long v) { *this = v; }

    bigint(const string &s) { read(s); }

    void operator=(const bigint &v) {
        sign = v.sign;
        z = v.z;
    }

    void operator=(long long v) {
        sign = 1;
        if (v < 0)
            sign = -1, v = -v;
        z.clear();
        for (; v > 0; v = v / base)
            z.push_back(v % base);
    }
```

```
    bigint operator+(const bigint &v) const {
        if (sign == v.sign) {
            bigint res = v;

            for (int i = 0, carry = 0; i < (int) max(z.size(), v.z.size()) || carry;
++i) {
                if (i == (int) res.z.size())
                    res.z.push_back(0);
                res.z[i] += carry + (i < (int) z.size() ? z[i] : 0);
                carry = res.z[i] >= base;
                if (carry)
                    res.z[i] -= base;
            }
            return res;
        }
        return *this - (-v);
    }

    bigint operator-(const bigint &v) const {
        if (sign == v.sign) {
            if (abs() >= v.abs()) {
                bigint res = *this;
                for (int i = 0, carry = 0; i < (int) v.z.size() || carry; ++i) {
                    res.z[i] -= carry + (i < (int) v.z.size() ? v.z[i] : 0);
                    carry = res.z[i] < 0;
                    if (carry)
                        res.z[i] += base;
                }
                res.trim();
                return res;
            }
            return -(v - *this);
        }
        return *this + (-v);
    }

    void operator*=(int v) {
        if (v < 0)
            sign = -sign, v = -v;
        for (int i = 0, carry = 0; i < (int) z.size() || carry; ++i) {
            if (i == (int) z.size())
                z.push_back(0);
            long long cur = z[i] * (long long) v + carry;
            carry = (int) (cur / base);
            z[i] = (int) (cur % base);
            //asm("divl %%ecx" : "=a"(carry), "=d"(a[i]) : "A"(cur), "c"(base));
        }
        trim();
    }

    bigint operator*(int v) const {
        bigint res = *this;
        res *= v;
        return res;
    }

    friend pair<bigint, bigint> divmod(const bigint &a1, const bigint &b1) {
        int norm = base / (b1.z.back() + 1);
        bigint a = a1.abs() * norm;
        bigint b = b1.abs() * norm;
        bigint q, r;
        q.z.resize(a.z.size());

        for (int i = a.z.size() - 1; i >= 0; i--) {
            r *= base;
            r += a.z[i];
```

```
            int s1 = b.z.size() < r.z.size() ? r.z[b.z.size()] : 0;
            int s2 = b.z.size() - 1 < r.z.size() ? r.z[b.z.size() - 1] : 0;
            int d = ((long long) s1 * base + s2) / b.z.back();
            r -= b * d;
            while (r < 0)
                r += b, --d;
            q.z[i] = d;
        }

        q.sign = a1.sign * b1.sign;
        r.sign = a1.sign;
        q.trim();
        r.trim();
        return make_pair(q, r / norm);
    }

    friend bigint sqrt(const bigint &a1) {
        bigint a = a1;
        while (a.z.empty() || a.z.size() % 2 == 1)
            a.z.push_back(0);

        int n = a.z.size();

        int firstDigit = (int) sqrt((double) a.z[n - 1] * base + a.z[n - 2]);
        int norm = base / (firstDigit + 1);
        a *= norm;
        a *= norm;
        while (a.z.empty() || a.z.size() % 2 == 1)
            a.z.push_back(0);

        bigint r = (long long) a.z[n - 1] * base + a.z[n - 2];
        firstDigit = (int) sqrt((double) a.z[n - 1] * base + a.z[n - 2]);
        int q = firstDigit;
        bigint res;

        for (int j = n / 2 - 1; j >= 0; j--) {
            for (; ; --q) {
                bigint r1 = (r - (res * 2 * base + q) * q) * base * base + (j > 0 ?
(long long) a.z[2 * j - 1] * base + a.z[2 * j - 2] : 0);
                if (r1 >= 0) {
                    r = r1;
                    break;
                }
            }
            res *= base;
            res += q;

            if (j > 0) {
                int d1 = res.z.size() + 2 < r.z.size() ? r.z[res.z.size() + 2] : 0;
                int d2 = res.z.size() + 1 < r.z.size() ? r.z[res.z.size() + 1] : 0;
                int d3 = res.z.size() < r.z.size() ? r.z[res.z.size()] : 0;
                q = ((long long) d1 * base * base + (long long) d2 * base + d3) /
(firstDigit * 2);
            }
        }

        res.trim();
        return res / norm;
    }

    bigint operator/(const bigint &v) const {
        return divmod(*this, v).first;
    }

    bigint operator%(const bigint &v) const {
        return divmod(*this, v).second;
    }
```

```
void operator/=(int v) {
    if (v < 0)
        sign = -sign, v = -v;
    for (int i = (int) z.size() - 1, rem = 0; i >= 0; --i) {
        long long cur = z[i] + rem * (long long) base;
        z[i] = (int) (cur / v);
        rem = (int) (cur % v);
    }
    trim();
}

bigint operator/(int v) const {
    bigint res = *this;
    res /= v;
    return res;
}

int operator%(int v) const {
    if (v < 0)
        v = -v;
    int m = 0;
    for (int i = z.size() - 1; i >= 0; --i)
        m = (z[i] + m * (long long) base) % v;
    return m * sign;
}

void operator+=(const bigint &v) {
    *this = *this + v;
}
void operator-=(const bigint &v) {
    *this = *this - v;
}
void operator*=(const bigint &v) {
    *this = *this * v;
}
void operator/=(const bigint &v) {
    *this = *this / v;
}

bool operator<(const bigint &v) const {
    if (sign != v.sign)
        return sign < v.sign;
    if (z.size() != v.z.size())
        return z.size() * sign < v.z.size() * v.sign;
    for (int i = z.size() - 1; i >= 0; i--)
        if (z[i] != v.z[i])
            return z[i] * sign < v.z[i] * sign;
    return false;
}

bool operator>(const bigint &v) const {
    return v < *this;
}
bool operator<=(const bigint &v) const {
    return !(v < *this);
}
bool operator>=(const bigint &v) const {
    return !(*this < v);
}
bool operator==(const bigint &v) const {
    return !(*this < v) && !(v < *this);
}
bool operator!=(const bigint &v) const {
    return *this < v || v < *this;
}
```

```
    void trim() {
        while (!z.empty() && z.back() == 0)
            z.pop_back();
        if (z.empty())
            sign = 1;
    }

    bool isZero() const {
        return z.empty() || (z.size() == 1 && !z[0]);
    }

    bigint operator-() const {
        bigint res = *this;
        res.sign = -sign;
        return res;
    }

    bigint abs() const {
        bigint res = *this;
        res.sign *= res.sign;
        return res;
    }

    long long longValue() const {
        long long res = 0;
        for (int i = z.size() - 1; i >= 0; i--)
            res = res * base + z[i];
        return res * sign;
    }

    friend bigint gcd(const bigint &a, const bigint &b) {
        return b.isZero() ? a : gcd(b, a % b);
    }
    friend bigint lcm(const bigint &a, const bigint &b) {
        return a / gcd(a, b) * b;
    }

    void read(const string &s) {
        sign = 1;
        z.clear();
        int pos = 0;
        while (pos < (int) s.size() && (s[pos] == '-' || s[pos] == '+')) {
            if (s[pos] == '-')
                sign = -sign;
            ++pos;
        }
        for (int i = s.size() - 1; i >= pos; i -= base_digits) {
            int x = 0;
            for (int j = max(pos, i - base_digits + 1); j <= i; j++)
                x = x * 10 + s[j] - '0';
            z.push_back(x);
        }
        trim();
    }

    friend istream& operator>>(istream &stream, bigint &v) {
        string s;
        stream >> s;
        v.read(s);
        return stream;
    }

    friend ostream& operator<<(ostream &stream, const bigint &v) {
        if (v.sign == -1)
            stream << '-';
        stream << (v.z.empty() ? 0 : v.z.back());
        for (int i = (int) v.z.size() - 2; i >= 0; --i)
```

```
            stream << setw(base_digits) << setfill('0') << v.z[i];
        return stream;
    }

    static vector<int> convert_base(const vector<int> &a, int old_digits, int
new_digits) {
        vector<long long> p(max(old_digits, new_digits) + 1);
        p[0] = 1;
        for (int i = 1; i < (int) p.size(); i++)
            p[i] = p[i - 1] * 10;
        vector<int> res;
        long long cur = 0;
        int cur_digits = 0;
        for (int i = 0; i < (int) a.size(); i++) {
            cur += a[i] * p[cur_digits];
            cur_digits += old_digits;
            while (cur_digits >= new_digits) {
                res.push_back(int(cur % p[new_digits]));
                cur /= p[new_digits];
                cur_digits -= new_digits;
            }
        }
        res.push_back((int) cur);
        while (!res.empty() && res.back() == 0)
            res.pop_back();
        return res;
    }

    typedef vector<long long> vll;

    static vll karatsubaMultiply(const vll &a, const vll &b) {
        int n = a.size();
        vll res(n + n);
        if (n <= 32) {
            for (int i = 0; i < n; i++)
                for (int j = 0; j < n; j++)
                    res[i + j] += a[i] * b[j];
            return res;
        }

        int k = n >> 1;
        vll a1(a.begin(), a.begin() + k);
        vll a2(a.begin() + k, a.end());
        vll b1(b.begin(), b.begin() + k);
        vll b2(b.begin() + k, b.end());

        vll a1b1 = karatsubaMultiply(a1, b1);
        vll a2b2 = karatsubaMultiply(a2, b2);

        for (int i = 0; i < k; i++)
            a2[i] += a1[i];
        for (int i = 0; i < k; i++)
            b2[i] += b1[i];

        vll r = karatsubaMultiply(a2, b2);
        for (int i = 0; i < (int) a1b1.size(); i++)
            r[i] -= a1b1[i];
        for (int i = 0; i < (int) a2b2.size(); i++)
            r[i] -= a2b2[i];

        for (int i = 0; i < (int) r.size(); i++)
            res[i + k] += r[i];
        for (int i = 0; i < (int) a1b1.size(); i++)
            res[i] += a1b1[i];
        for (int i = 0; i < (int) a2b2.size(); i++)
            res[i + n] += a2b2[i];
        return res;
```

```
    }

    bigint operator*(const bigint &v) const {
        vector<int> a6 = convert_base(this->z, base_digits, 6);
        vector<int> b6 = convert_base(v.z, base_digits, 6);
        vll a(a6.begin(), a6.end());
        vll b(b6.begin(), b6.end());
        while (a.size() < b.size())
            a.push_back(0);
        while (b.size() < a.size())
            b.push_back(0);
        while (a.size() & (a.size() - 1))
            a.push_back(0), b.push_back(0);
        vll c = karatsubaMultiply(a, b);
        bigint res;
        res.sign = sign * v.sign;
        for (int i = 0, carry = 0; i < (int) c.size(); i++) {
            long long cur = c[i] + carry;
            res.z.push_back((int) (cur % 1000000));
            carry = (int) (cur / 1000000);
        }
        res.z = convert_base(res.z, 6, base_digits);
        res.trim();
        return res;
    }
};
```