费用流（EK）：

EK算法求解网络流的时间复杂度为$O(VE^2)$

求解费用流$O(flow * VE)$，$flow$为最大流量

```
#include<bits/stdc++.h>
using namespace std;
#define rep(i, a, b) for(int i = (a); i <= (b); ++i)

typedef long long ll;
const ll INF = 0x3f3f3f3f3f3f3f3f;
const int inf = 0x3f3f3f3f;
const int M = 8333333;
const int N = 52500; // 最大流为50时，可通过此数据
// 最小费用流

struct Edge {
    int from, to, w, pre;
    ll c;
} e[M];
int last[N], tot = 1;

inline void ine(int from, int to, int w, ll c) {
    e[++tot].to = to;
    e[tot].w = w;
    e[tot].c = c;
    e[tot].from = from;
    e[tot].pre = last[from];
    last[from] = tot;
}

inline void add(int a, int b, int w, ll c) {
    ine(a, b, w, c);
    ine(b, a, 0, -c);
}

int s, t;
int fa[N], flow[N], inq[N]; ll dis[N];
queue<int> Q;
bool SPFA(int n)
{
    while (!Q.empty()) Q.pop();
    for(int i = 1; i <= n; ++i) fa[i] = 0, inq[i] = 0, flow[i] = inf, dis[i] =
INF;
    dis[s] = 0; Q.push(s); inq[s] = 1;
    while (!Q.empty())
    {
        int p = Q.front(); Q.pop();
        inq[p] = 0;
        for (int eg = last[p]; eg; eg = e[eg].pre)
        {
            int to = e[eg].to, vol = e[eg].w;
            if (vol > 0 && dis[to] > dis[p] + e[eg].c) // 容量大于0才增广
            {
                fa[to] = eg; // 记录上一条边
                flow[to] = min(flow[p], vol); // 更新下一个点的流量
                dis[to] = dis[p] + e[eg].c;
                if (!inq[to])
                {
                    Q.push(to);
                    inq[to] = 1;
                }
            }
        }
    }
    return fa[t] != 0;
}

void update() {
    for (int i = t; i != s; i = e[fa[i] ^ 1].to) {
        e[fa[i]].w -= flow[t];
        e[fa[i] ^ 1].w += flow[t];
    }
}

ll maxflow, mincost;
inline void MCMF(int n) {
    maxflow = 0, mincost = 0;
    while (SPFA(n)) {
        maxflow += flow[t];
        mincost += dis[t] * flow[t];
        update();
    }
}
```

网络流（dinic）：

```
#include<bits/stdc++.h>
using namespace std;
#define rep(i, a, b) for(int i = (a); i <= (b); ++i)

typedef long long ll;
const ll INF = 0x3f3f3f3f3f3f3f3f;
const int M = 20005;
const int N = 3005;

// 最大流
struct Edge {
    int from, to, pre;
    ll w;
} e[M];
int last[M], tot = 1;

void ine(int a, int b, ll w) {
    tot++;
    e[tot].from = a; e[tot].to = b; e[tot].w = w;
    e[tot].pre = last[a];
    last[a] = tot;
}
```

```
int s, t, lv[N], cur[M];  // lv: 每点层数, cur: 当前弧
inline bool bfs(int n)  {
    rep(i, 1, n) lv[i] = -1;
    lv[s] = 0;
    memcpy(cur, last, sizeof(last));
    queue<int> q;
    q.push(s);
    while(!q.empty()) {
        int u = q.front(); q.pop();
        for(int i = cur[u]; i; i = e[i].pre) {
            int to = e[i].to;
            ll vol = e[i].w;
            if(vol > 0 && lv[to] == -1)
                lv[to] = lv[u] + 1, q.push(to);
        }
    }
    return lv[t] != -1; // 如果汇点未访问过则不可达
}

ll dfs(int u = s, ll f = INF) {
    if(u == t)
        return f;
    for(int &i = cur[u]; i; i = e[i].pre) {
        int to = e[i].to;
        ll vol = e[i].w;
        if(vol > 0 && lv[to] == lv[u] + 1) {
            ll c = dfs(to, min(vol, f));
            if(c) {
                e[i].w -= c;
                e[i ^ 1].w += c;    // 反向边
                return c;
            }
        }
    }
    return 0; // 输出流量大小
}

inline ll dinic(int n)
{
    ll ans = 0;
    while(bfs(n)) {
        ll f;
        while((f = dfs()) > 0)
            ans += f;
    }
    return ans;
}

int main() {
    int n, m, u, v; ll w;
    cin >> n >> m >> s >> t;
    tot = 1;
    rep(i, 1, m) {
        scanf("%d %d %lld", &u, &v, &w);
        ine(u, v, w); // 正向边容量为w
        ine(v, u, 0); // 反向边容量为0
    }
    cout << dinic() << endl;
}
```

一般图最大匹配（带花树）：$O(n^3)$

```
#include<bits/stdc++.h>
using namespace std;

// https://blog.bill.moe/blossom-algorithm-notes/
// 带花树: 一般图最大匹配
const int maxn=505;

struct Match {
    int n,father[maxn],vst[maxn],match[maxn],pre[maxn],Type[maxn],times;
    vector<int>edges[maxn];
    queue<int>Q;

    void ine(int x,int y) { edges[x].push_back(y); }
    void ine2(int x, int y) { ine(x, y); ine(y, x); }

    void init(int num) {
        times=0;
        n = num;
        for(int i = 0; i <= n; ++i)
edges[i].clear(),vst[i]=0,match[i]=0,pre[i]=0;
    }

    int LCA(int x,int y) {
        times++;
        x=father[x],y=father[y]; //已知环位置
        while(vst[x]!=times) {
            if(x) {
                vst[x]=times;
                x=father[pre[match[x]]];
            }
            swap(x,y);
        }
        return x;
    }

    void blossom(int x,int y,int lca) {
        while(father[x]!=lca) {
            pre[x]=y;
            y=match[x];
            if(Type[y]==1) {
                Type[y]=0;
                Q.push(y);
            }
            father[x]=father[y]=father[lca];
            x=pre[y];
        }
    }

    int Augument(int s) {
        for(int i=0; i<=n; ++i)father[i]=i,Type[i]=-1;
        Q=queue<int>();
        Type[s]=0;
        Q.push(s); //仅入队o型点
        while(!Q.empty()) {
            int Now=Q.front();
            Q.pop();
            for(int Next:edges[Now]) {
                if(Type[Next]==-1) {
                    pre[Next]=Now;
                    Type[Next]=1; //标记为i型点
                    if(!match[Next]) {
                        for(int to=Next,from=Now; to; from=pre[to]) {
                            match[to]=from;
                            swap(match[from],to);
                        }
                        return true;
                    }
                    Type[match[Next]]=0;
                    Q.push(match[Next]);
                } else if(Type[Next]==0&&father[Now]!=father[Next]) {
                    int lca=LCA(Now,Next);
                    blossom(Now,Next,lca);
                    blossom(Next,Now,lca);
                }
            }
        }
        return false;
    }

    void gao() {
        int res = 0; // 最大匹配数
        for(int i = n; i >= 1; --i) if(!match[i]) res += Augument(i);
        printf("%d\n", res);
        for(int i = 1; i <= n; ++i) printf("%d ",match[i]);
        printf("\n");
    }
} G;

int main() {
    int n, m;
    cin >> n >> m;
    G.init(n);
    for(int i=1,x,y; i<=m; i++) {
        scanf("%d %d", &x, &y);
        G.ine2(x, y);
    }
    G.gao();
    return 0;
}
```

FWT:

求解卷积运算:

$$c_n = \sum_{i \oplus j = n} a_i b_j$$

A为一向量，长度为2的整数次幂，$FWT(A)$也为一向量，长度与A相同。

满足：$FWT(A*B)=FWT(A)\cdot FWT(B)$

做法：FWT --> 内积 --> $IFWT$

注：卷积运算和FWT运算本质上都是对下标的限制，分配律自然成立

$FWT(A+B)=FWT(A)+FWT(B)$, $A*(B+C)=A*B+A*C$

```
// or
void FWT(ll *a, int len, int inv) {
    for(int h = 1; h < len; h <<= 1) {
        for(int i = 0; i < len; i += (h<<1)) {
            for(int j = 0; j < h; ++j) {
                a[i+j+h] += a[i+j] * inv;
            }
        }
    }
}
// and
void FWT(ll *a, int len, int inv) {
    for(int h = 1; h < len; h <<= 1) {
        for(int i = 0; i < len; i += (h << 1)) {
            for(int j = 0; j < h; ++j) {
                a[i+j] += a[i+j+h] * inv;
            }
        }
    }
}
// xor
void FWT(ll *a, int len, int inv) {
    ll x, y;
    for(int h = 1; h < len; h <<= 1) {
        for(int i = 0; i < len; i += (h << 1)) {
            for(int j = 0; j < h; ++j) {
                x = a[i+j], y = a[i+j+h];
                a[i+j] = x+y, a[i+j+h] = x-y;
                if(inv == -1) a[i+j] /= 2, a[i+j+h] /= 2;
            }
        }
    }
}
```

高斯消元：

```
// 高斯消元-辗转相除（求行列式）
const int N = 20;
struct Mat {
    int a[N][N], n, m;
    void init(int _n, int _m, int val) {
        n = _n, m = _m;
        for(int i = 0; i < n; ++i) for(int j = 0; j < m; ++j) a[i][j] = val;
    }
    int guass() { // 辗转相除化简(当mod大时可能会溢出)
        int ans = 1;
        for(int j = 0; j < m; ++j) {
            for(int i = j + 1; i < n; ++i) {
                while(a[i][j]) {
                    int t = a[j][j] / a[i][j];
                    for(int k = j; k < m; ++k)
                        a[j][k] = (a[j][k] - t * a[i][k] % mod + mod) % mod;
                    swap(a[i], a[j]);
                    ans = -ans;
                }
            }
        }
        for(int i = 0; i < m && i < n; ++i) ans = ans * a[i][i] % mod;
        return (ans + mod) % mod;
    }
} mat;

// 高斯消元（求秩，解方程）
const int L = 512;
ll a[L][L+1], ans[L];
void gauss(int n, int m) { // 消增广矩阵
    vector<int> pos(m, -1); // 有效pos数量即为秩
    ll inv, del;
    for(int r = 0, c = 0; c < m; ++c) {
        int sig = -1;
        for(int i = r; i < n; ++i)
            if(a[i][c]) {
                sig = i; break;
            }
        if(sig == -1) continue; // 空列
        pos[c] = r;
        if(sig != r) swap(a[sig], a[r]);
        inv = Pow(a[r][c], mod - 2);
        for(int i = 0; i < n; ++i) {
            if(i == r) continue;
            del = inv * a[i][c] % mod;
            for(int j = c; j <= m; ++j) a[i][j] = (a[i][j] - del * a[r][j] %
mod) % mod;
        }
        ++r;
    }
    for(int i = 0; i < m; ++i) { // ax = b
        if(pos[i] != -1) {
            ans[i] = Pow(a[pos[i]][i], mod - 2) * a[pos[i]][m] % mod;
        }
    }
}
```

记搜+数位DP：

```
// 记忆化搜索
struct cmp {
    bool operator()(const state& a, const state& b) const {
        // ...
    }
};
```

```
struct _hash {
    size_t operator()(const state& st) const {
        size_t res = st.cnt[0];
        for(int i = 1; i < 10; ++i) {
            res *= 19260817;
            res += st.cnt[i];
        }
        return res;
    }
};
unordered_map <state, ll, _hash, cmp> dp[20][20];

// -pos: 搜到的位置
// -st: 当前状态
// -lead: 是否有前导0
// -limit: 是否有最高位限制
ll dfs(int pos, state st, int lead, int limit){
    // 边界情况
    if(pos < 0 /* && ... */) return 0;
    // 记忆化搜索
    int wd = st.w[st.d];
    if((!limit) && (!lead) && dp[pos][wd].count(st)) return dp[pos][wd][st];

    ll res = 0;
    // 最高位最大值
    int cur = limit ? a[pos] : 9;
    for(int i = 0; i <= cur; ++i) {
        // 有前导0且当前位也是0
        if((!i) && lead) res += dfs(pos-1, st, 1, limit&&(i==cur));
        // 有前导0且当前位非0（出现最高位）
        else if(i && lead) res += dfs(pos-1, st.add(i), 0, limit&&(i==cur));
        else res += dfs(pos-1, st.add(i), 0, limit&&(i==cur));
    }
    // 没有前导0和最高限制时可以直接记录当前dp值以便下次搜到同样的情况可以直接使用。
    if(!limit&&!lead) dp[pos][wd][st] = res;
    return res;
}
ll gao(ll x) {
    memset(a, 0, sizeof(a));
    int len=0;
    while(x) a[len++]=x%10,x/=10;
    // init st
    return dfs(len-1, st, 1, 1);
}
```

极角排序：

```
// 极角排序 --象限法
struct Point {
    ll x, y; int rd;
    bool operator < (Point b)const{
        if(rd != b.rd) return rd < b.rd;
        else return sgn((*this)^b) > 0;
    }
    void calcrd() {
        rd = (x > 0 || x == 0 && y > 0) ? 0 : 1;
    }
    Point rotleft() const {
        Point res = Point(-y,x);
        res.calcrd();
        if((res.rd - rd) % 2) res.rd = rd + 1;
        else res.rd = rd;
        return res;
    }
};
// m = 2 * tot
for(int j = 0, k = 0; j < tot; ++j) {
    Point r = q[j].rotleft();
    while(k+1==j || k+1<m && q[k+1]<r) ++k;
    // [j+1, k] 为所有落在 (r, r + pi/2) 中的点
}
```

半平面交：

```
struct Line{
    Point s,e;
    double angle;
    Line(){}
    bool operator ==(Line v){
        return (s == v.s)&&(e == v.e);
    }
    void calcangle(){
        angle = atan2(e.y-s.y,e.x-s.x);
    }
    bool operator <(const Line &b)const{
        return angle < b.angle;
    }
    Line(Point _s, Point _e) {
        s = _s; e = _e;
    }
    //ax+by+c=0
    Line(double a,double b,double c){
        if(sgn(a) == 0){
            s = Point(0,-c/b);
            e = Point(1,-c/b);
        }
        else if(sgn(b) == 0){
            s = Point(-c/a,0);
            e = Point(-c/a,1);
        }
        else{
            s = Point(0,-c/b);
            e = Point(1,(-c-a)/b);
        }
        Point tmp = s + (e-s).rotleft();
        if(sgn(a*tmp.x+b*tmp.y+c) > 0) swap(s, e);
    }
    //`两向量平行(对应直线平行或重合)`
    bool parallel(Line v){
        return sgn((e-s)^(v.e-v.s)) == 0;
    }
    //`求两直线的交点`
    //`要保证两直线不平行或重合`
    Point crosspoint(Line v){
        double a1 = (v.e-v.s)^(s-v.s);
        double a2 = (v.e-v.s)^(e-v.s);
        return Point((s.x*a2-e.x*a1)/(a2-a1),(s.y*a2-e.y*a1)/(a2-a1));
    }
};
struct halfplanes{
    int n;
    Line hp[N];
    Point p[N];
    int que[N];
    int st,ed;
    void push(Line tmp){
        hp[n++] = tmp;
    }
    //去重
    void unique(){
        int m = 1;
        for(int i = 1;i < n;i++){
            if(sgn(hp[i].angle-hp[i-1].angle) != 0)
                hp[m++] = hp[i];
            else if(sgn( (hp[m-1].e-hp[m-1].s)^(hp[i].s-hp[m-1].s) ) > 0)
                hp[m-1] = hp[i];
        }
        n = m;
    }
    bool halfplaneinsert(){
        for(int i = 0;i < n;i++)hp[i].calcangle();
        sort(hp,hp+n);
        unique();
        que[st=0] = 0;
        que[ed=1] = 1;
        p[1] = hp[0].crosspoint(hp[1]);
        for(int i = 2;i < n;i++){
            while(st<ed && sgn((hp[i].e-hp[i].s)^(p[ed]-hp[i].s))<0)ed--;
            while(st<ed && sgn((hp[i].e-hp[i].s)^(p[st+1]-hp[i].s))<0)st++;
            que[++ed] = i;
            if(hp[i].parallel(hp[que[ed-1]]))return false;
            p[ed]=hp[i].crosspoint(hp[que[ed-1]]);
        }
        while(st<ed && sgn((hp[que[st]].e-hp[que[st]].s)^(p[ed]-hp[que[st]].s))
<0)ed--;
        while(st<ed && sgn((hp[que[ed]].e-hp[que[ed]].s)^(p[st+1]-
hp[que[ed]].s))<0)st++;
        if(st+1>=ed)return false;
        return true;
    }
    //`得到最后半平面交得到的凸多边形`
    //`需要先调用halfplaneinsert() 且返回true`
    void getconvex(polygon &con){
        p[st] = hp[que[st]].crosspoint(hp[que[ed]]);
        con.n = ed-st+1;
        for(int j = st,i = 0;j <= ed;i++,j++)
            con.p[i] = p[j];
    }
```

```
};
```

简单多边形与圆:

```
struct polygon {
    //`多边形和圆交的面积`
    //`测试: POJ3675 HDU3982 HDU2892`
    double areacircle(circle c){
        double ans = 0;
        for(int i = 0;i < n;i++){
            int j = (i+1)%n;
            if(sgn( (p[j]-c.p)^(p[i]-c.p) ) >= 0)
                ans += c.areatriangle(p[i],p[j]);
            else ans -= c.areatriangle(p[i],p[j]);
        }
        return fabs(ans);
    }
    //`多边形和圆关系`
    //` 2 圆完全在多边形内`
    //` 1 圆在多边形里面, 碰到了多边形边界`
    //` 0 其它`
    int relationcircle(circle c){
        getline();
        int x = 2;
        if(relationpoint(c.p) != 1)return 0;//圆心不在内部
        for(int i = 0;i < n;i++){
            if(c.relationseg(l[i])==2)return 0;
            if(c.relationseg(l[i])==1)x = 1;
        }
        return x;
    }
};
```