# 优化

## bascic

```
namespace BasicT{
    mt19937
mtrandom(std::chrono::system_clock::now().time_since_epoch().count());
    template<typename T>T getRandom(T l,T r){return uniform_int_distribution<T>
(l,r)(mtrandom);}
    template<typename T>T gcd(T a,T b){return b==0?a:gcd(b,a%b);}
    ll qmul(ll a,ll b){ll r=0;while(b){if(b&1)r=(r+a)%mod;b>>=1;a=
(a+a)%mod;}return r;}
    ll qpow(ll a,ll n){ll r=1;while(n){if(n&1)r=(r*a)%mod;n>>=1;a=
(a*a)%mod;}return r;}
    ll qpow(ll a,ll n,ll p){ll r=1;while(n){if(n&1)r=(r*a)%p;n>>=1;a=
(a*a)%p;}return r;}
}
```

## debug

```
void debug(const std::string &file, i32 line, const std::string &name, const
auto &value) {
    std::cerr << file << ":" << line << " | " << name << " = " << value <<
std::endl;
}

template <class T>
std::ostream &operator<<(std::ostream &os, const std::vector<T> &v) {
    os << "[ ";
    for (const T &vi : v) {
        os << vi << ", ";
    }
    return os << "]";
}
```

## Fast I/O

## I/O

之后可以用fin，fout读入输出，1e5没有明显优势

```
struct BasicBuffer {
    std::vector<char> s;
    BasicBuffer() : s(1 << 18) {}
    char *p = s.data(), *beg = p, *end = p + s.size();
    inline char getc() {
        if (p == end)
            readAll();
        return *p++;
    }
    inline void putc(char c) {
```

```
        if (p == end)
            writeAll();
        *p++ = c;
    }
    inline void puts(const char *x) {
        while (*x != 0)
            putc(*x++);
    }
    void readAll() {
        std::fread(beg, 1, end - beg, stdin);
        p = s.data();
    }
    void writeAll() {
        std::fwrite(beg, 1, p - beg, stdout);
        p = s.data();
    }
};

struct FastI : BasicBuffer {
    FastI() {
        readAll();
    }
    ll read() {
        ll x = 0;
        char c = getc();
        bool sgn = true;
        while (!std::isdigit(c))
            sgn = sgn && c != '-', c = getc();
        while (std::isdigit(c))
            x = x * 10 + c - '0', c = getc();
        return sgn ? x : -x;
    }
    template <class T>
    FastI &operator>>(T &x) {
        return x = read(), *this;
    }
    FastI &operator>>(char &x) {
        return x = getc(), *this;
    }
};

struct FastO : BasicBuffer {
    std::array<char, 32> u{};
    ~FastO() {
        writeAll();
    }
    void output(ll x) {
        char *i = u.data() + 20;
        if (x < 0)
            putc('-'), x = -x;
        do
            *--i = x % 10 + '0', x /= 10;
        while (x > 0);
        puts(i);
    }
    template <class T>
    FastO &operator<<(const T &x) {
        return output(x), *this;
```

```
    }
    FastO &operator<<(char x) {
        return putc(x), *this;
    }
    FastO &operator<<(const char *x) {
        return puts(x), *this;
    }
    FastO &operator<<(const std::string &x) {
        return puts(x.c_str()), *this;
    }
};

FastI fin;
FastO fout;
```

# 第二

```
namespace FastIOT{
    const int bsz=1<<18;
    char bf[bsz],*head,*tail;
    IL char gc(){if(head==tail)tail=
(head=bf)+fread(bf,1,bsz,stdin);if(head==tail)return 0;return *head++;}
    template<typename T>IL void read(T &x){T f=1;x=0;char
c=gc();for(;c>'9'||c<'0';c=gc())if(c=='-')f=-1;
    for(;c<='9'&&c>='0';c=gc())x=(x<<3)+(x<<1)+(c^48);x*=f;}
    template<typename T>IL void print(T x){if(x<0)putchar(45),x=-
x;if(x>9)print(x/10);putchar(x%10+48);}
    template<typename T>IL void println(T x){print(x);putchar('\n');}
}
using namespace FastIOT;
```

```
struct IO {
    char a[1 << 25], b[1 << 25], *s, *t;
    IO() : s(a), t(b) {
        a[std::fread(a, 1, sizeof a, stdin)] = 0;
    }
    ~IO() {
        std::fwrite(b, 1, t - b, stdout);
    }
    IO &operator>>(std::uint64_t &x);
    IO &operator>>(std::int64_t &x);
    IO &operator>>(std::int32_t &x);
    IO &operator>>(std::uint32_t &x) {
        x = 0;

        while (*s < '0' || *s > '9')
            ++s;

        while (*s >= '0' && *s <= '9')
            x = x * 10 + *s++ - '0';

        return *this;
    }
    IO &operator<<(const char *tmp) {
        return std::fwrite(tmp, 1, std::strlen(tmp), stdout), *this;
    }
```

```
    IO &operator<<(char x) {
        return *t++ = x, *this;
    }
    IO &operator<<(std::int32_t x);
    IO &operator<<(std::uint64_t x);
    IO &operator<<(std::int64_t x);
    IO &operator<<(std::uint32_t x) {
        static char c[16], *i;
        i = c;

        if (x == 0) {
            *t++ = '0';
        } else {
            while (x != 0) {
                std::uint32_t y = x / 10;
                *i++ = x - y * 10 + '0', x = y;
            }

            while (i != c)
                *t++ = *--i;
        }

        return *this;
    }
} io;
io >> a
io << a
```

```
numeric_limits<T>::max();//返回数据类型T的最大值
```

# 图论

## 必经点

### 题目

- 给出一张有 $n$ 个结点、 m 条边的无向联通图；
- 图上有两个特殊点 $a$ 和 $b$ （$1 \leq a, b \leq n, a \neq b$）；
- 求出满足下列条件的二元组 $(u,v)$ 的对数：
  - $1 \leq u < v \leq n$
  - $u \neq a, v \neq a, u \neq b, v \neq b$
  - 任意一条从$u$到$v$的路径$(u, e_1, e_2, \ldots, e_k, v)$都经过$a$和$b$。
- 包含 $T$ 组测试数据。

### 分析

只需要对整张图扫两边，标记出只有$a$能到的点和只有$b$能到的点就可以

```
vector<int>G[N];
int col[N];
void dfs(int x,int c,int f)
{
```

```
    for(auto y:G[x])
    {
        if(y==f)continue;
        if(col[y]&c)continue;
        col[y]|=c;
        dfs(y,c,f);
    }
}
void solve()
{
    int n,m;cin>>n>>m;
    int a,b;cin>>a>>b;
    for(int i=1;i<=n;++i)
    {
        G[i].clear();
        col[i]=0;
    }
    for(int i=1;i<=m;++i)
    {
        int x,y;cin>>x>>y;
        G[x].push_back(y);
        G[y].push_back(x);
    }
    col[a]=1;
    dfs(a,1,b);
    col[b]=2;
    dfs(b,2,a);
    int cnt1=-1,cnt2=-1;
    for(int i=1;i<=n;++i)
    {
        if(col[i]==1)cnt1++;
        else if(col[i]==2)cnt2++;
    }
    cout<<1ll*cnt1*cnt2<<'\n';
}
```

# 二分图匹配

## 前置

## 定义

图 $G=(V,E)$，其中$V$是点集，$E$是边集

一组两两没有公共点的边集 $(M(M\in E))$称为这张图的 **匹配。**

定义匹配的大小为其中边的数量 $|M|$，其中边数最大的大的 $M$ 为 **最大匹配。**

当图中的边带权的时候，边权和最大的为 **最大权匹配。**

匹配中的边称为 **匹配边**，反之称为 **未匹配边。**

一个点如果属于 $M$且为至多一条边的端点，称为 **匹配点**，反之称为 **未匹配点。**

完美匹配：所有点都属于匹配点，同时也符合最大匹配

增广路：也称**交错路**，是一条连接两个非匹配点，且匹配边与非匹配边交错出现的路径

增广路显然具有以下性质：

1.长度$len$是奇数

2.路径上第$1,3,5,\cdots,len$是非匹配边，第$2,4,5,\cdots,len-1$是匹配边

推论

**二分图的一组匹配$M$是最大匹配，当且仅当图中不存在$S$的增广路**

# 二分图最大匹配

# code

因为增广路长度为奇数，路径起始点非左即右，所以我们先考虑从左边的未匹配点找增广路。 注意到因为交错路的关系，增广路上的第奇数条边都是非匹配边，第偶数条边都是匹配边，于是左到右都是非匹配边，右到左都是匹配边。 于是我们给二分图 **定向**，问题转换成，有向图中从给定起点找一条简单路径走到某个未匹配点，此问题等价给定起始点 $s$ 能否走到终点 $t$。 那么只要从起始点开始 DFS 遍历直到找到某个未匹配点，。 未找到增广路时，我们拓展的路也称为 **交错树**。

$O(NM)$

```
//点0~n-1 ,
namespace augment_path {
    vector<vector<int> > g;
    vector<int> pa;      // 左部点匹配
    vector<int> pb;      // 右部点匹配
    vector<int> vis;     // 访问
    vector<int> ext;
    int n, m;            // 两个点集中的顶点数量 左部点,右部点
    int dfn;           // 时间戳记，代替memset加快效率
    int res;           // 匹配数

    void init(int _n,int _m){
        n = _n, m = _m;
        assert(0 <= n && 0 <= m);
        pa = vector<int>(n, -1);
        pb = vector<int>(m, -1);
        vis = vector<int>(n);
        g.resize(n);
        res = 0;
        dfn = 0;
    }
    void add(int from, int to) { //注意减一
        assert(0 <= from && from < n && 0 <= to && to < m);
        g[from].push_back(to);
    }

    bool bfs(int v) {
        vis[v] = dfn; //标记访问
        for (int u : g[v]) {
            if (pb[u] == -1) { //如果右部点u没有匹配，我们就直接连接u,v
                pb[u] = v;
                pa[v] = u;
                return true;
            }
```

```
        }
        for (int u : g[v]) {  //否则尝试找一条增广路
          if (vis[pb[u]] != dfn && bfs(pb[u])) { //没有访问过就尝试找增广路
                pa[v] = u;
                pb[u] = v;
                return true;
            }
        }
        return false;
    }

    int solve() {
        while (true) {
          dfn++;
          int cnt = 0;
          for (int i = 0; i < n; i++) {
            if (pa[i] == -1 && bfs(i)) { //如果左部点i没有匹配，我们就dfs尝试匹配
              cnt++;
            }
          }
          if (cnt == 0) {
            break;
          }
          res += cnt;
        }
        return res;
    }
    //上面是普通的版本


    //用于字典序最小 ,左部点每个点度数不超过2
    bool dfs(int v) {
        for (int u : g[v]) {
            if(vis[u] != dfn){
                vis[u] = dfn; //标记访问
                if(pb[u] == -1 || dfs(pb[u])){//如果右部点u没有匹配或者能找到增广路，我
们就直接连接u,v
                    pb[u] = v;
                    pa[v] = u;
                    return true;
                }
            }
        }
        return false;
    }

    int ssolve() { //作用和上面那个solve一样，效率低但是便于修改
        int cnt = 0;

        for (int i = n - 1; i >= 0; i--) {
            dfn++;
            if (pa[i] == -1 && dfs(i)) { //如果左部点i没有匹配，我们就dfs尝试匹配
                    cnt++;
            }

        }
        res += cnt;
        return res;
```

```
    }

    //用于求字典序最小的完美匹配
    bool dfs_min(int v) {
        if(vis[v] == dfn || ext[v])return 0;
        vis[v] = dfn;
        for (int u : g[v]) {
            if(ext[n + u])continue;
            if(pb[u] == -1 || dfs_min(pb[u]))
            {
                pb[u] = v;
                pa[v] = u;
                return true;
            }
        }
        return false;
    }

    bool solve_min() {
        solve();
        if(res != n)return false;

        //先看看可不可以，再考虑最小

        ext.resize(2 * n, 0);
        for(int i = 0; i < n; ++i)
        {
            for(int p = 0; p < g[i].size(); ++p)//每一次都强制匹配i和y，然后pa[i]与
pb[i]失配重连
            {
                int y = g[i][p];
                bool check = 0;
                if(pa[i] == y)check = 1;
                else
                {
                    ext[i] = 1;
                    ext[y + n] = 1;
                    pb[pa[i]] = -1;
                    dfn++;
                    if(dfs_min(pb[y]))check = 1;
                    else pb[pa[i]] = i;
                    ext[i] = 0;
                    ext[y + n] = 0;
                }
                if(check)
                {
                    pa[i] = y;
                    pb[y] = i;
                    ext[i] = ext[n + y] = 1;
                    break;
                }

            }
        }

        return 1;
    }
};
```

对于字典序最小的方案,如果左部点最多只有两条边与之相连,那么我们倒着枚举左部点,同时让边的终点从小到大放入图即可

证明

首先，我们不停地选择右边只有一条连边的点。由于要达到完美匹配，与它相连的左边的点必须与它匹配。所以可以把那个点连的两条边删掉。

此时，所有右边度数为 11 的点都被删完了，所以 $min >= 2$。而右边总度数开始时为 $2n$，每匹配一个点度数 -2，所以假设剩下 $k$ 个点，总度数一定为 $2n-2(n-k)=2n-2(n-k)=2k$，又因为$min >= 2$，所以每个右边的点度数都为 2。

然后因为图中每个点度数都为 2，所以被分成了若干个度数为 2 的环。倒着匹配的过程中，确定了一个匹配后，两个端点所连的另两个点的匹配也就确定了，一直循环下去，环中所有剩余的匹配也就确定了（自己模拟一下有助于理解）。所以虽然有后效性，但在后面的过程中只有一种选择，就一定可以保证字典序最小了。

# 二分图带权完备匹配

## 前置

相等子图$(A_i+B_j=w(i,j))$的完备匹配是带权最大匹配。

对于该问题，我们应当至少还应理解以下内容：

1. 最大权但不是完美匹配
2. 我们可以求想要的顺序，如最小字典序
3. 特判无解的情况

对于第一种情况，我们只需要考虑将初始值设为0，然后当作重边取$max$即可

对于第二种情况，我们应该考虑改变边的权重然后看$i-j$是否连边，假设前$i-1$个点已经是我们要求的顺序了，那么前$i-1$个点，只保留匹配边，而对于这些点连出的非匹配边，当成不存在（即$-inf$)。对于$i$来说，将右边点的顺序按照我们想要的加上不同的微小量，以确保在其他不变的情况下保证想要的顺序。具体实现，保留边的边权乘以点的大小，特别边加上0~点的大小-1

对于第三种情况，我们应该考虑匹配边是否有$-inf$的情况（不是0是因为可能有负权边）

## code

顶标全称"定点标记值"，满足$A_i+B_j\geq w(i,j)$。

$lx[i]$左部点的顶标

$ly[i]$右部点的顶标

$visx[i]$左部点遍历标记

$visy[i]$右部点遍历标记

$matchx[i]$左部点匹配

$matchx[i]$右部点匹配

$slack[i]$对于指向右部点$i$的所有边，$min(lx[u]+ly[i]-e[u][i])$的值，即松弛量（初始化为$inf$）

PS:当$slack[i]$，表示对于右部点$i$，相等子图中有一条指向它的边

km算法大致思想：

带权匹配在这里我们转换为完备匹配来做，依据就是相等子图的特点。km算法实质是维护一个相等子图并在上面做完备匹配的过程。

一开始，我们将所有顶点加入子图$G'$，然后考虑与左部点直接相连的边中最大的几条，将其加入$G'$中。（这是为什么我们顶标是$la[i]=max(g[i][u]),lb[i]=0,$）

然后我们去求这张图的完备匹配，若存在那么我们就得到一个最大权，如果不存在，我们就考虑扩大这张子图。

如何扩大呢，贪心去考虑的话，就是将次小边也加进去看看，但是这样又怎么保证子图是相等子图。

于是引入了一个重要操作，将相等子图中已匹配的左部点顶标减小，右部点顶标增大，这样，原本在子图里的边仍然在图上，原来不在的，就有可能引进去了。更具体地说，由于原本子图不是完备匹配，那么一定有一个点，在寻找增广路时寻找失败，那么寻找路径时就形成了一颗交错树。这棵树右部点都是已匹配的点，否则就有一条增广路。那么左部点连向的右部点如果不在这棵树上，那么这条边对应的顶标值就变小，也就有可能加入子图中了。

更进一步地说，我们其实没有必要等子图跑完完备匹配再去更新，因为我们最后想要的一定得是完备匹配，否则为什么不用网络流呢。因此一旦匹配失败，我们就去扩展子图，然后从新加入的边开始搜索，就可以都得到一个较优的复杂度。

```
//0~n-1
namespace hungarian {   // km
    int n;                  //max(左部点个数，右部点个数)
    vector<int> matchx;     // 左集合对应的匹配点
    vector<int> matchy;     // 右集合对应的匹配点
    vector<int> pre;        // 连接右集合的左点
    vector<int> visx;       // 左部点遍历标记
    vector<int> visy;       // 右部点遍历标记
    vector<ll> lx;          //左部顶标
    vector<ll> ly;          //右部顶标
    vector<vector<ll> > g;  //图
    vector<ll> slack;       //松弛量
    ll inf;
    ll res;
    int tmp;
    queue<int> q;
    int org_n;
    int org_m;

    void init(int _n, int _m){
        org_n = _n;
        org_m = _m;
        n = max(_n, _m);
        inf = 1ll << 60;
        res = 0;
        g = vector<vector<ll> >(n, vector<ll>(n, -inf));
        matchx = vector<int>(n, -1);
        matchy = vector<int>(n, -1);
        pre = vector<int>(n);
        visx = vector<int>(n, 0);
```

```
        visy = vector<int>(n, 0);
        lx = vector<ll>(n, -inf);
        ly = vector<ll>(n, 0);
        slack = vector<ll>(n);
    }


    void addEdge(int u, int v, ll w) {
        assert(0 <= u && u < n && 0 <= v && v < n);
        if(w>=0)g[u][v] = max(g[u][v], w);
    }

    bool check(int v) {
        visy[v] = tmp;
        if (matchy[v] != -1) {
            q.push(matchy[v]);
            visx[matchy[v]] = tmp;  // in S
            return false;
        }
        // 找到新的未匹配点 更新匹配点 pre 数组记录着"非匹配边"上与之相连的点
        while (v != -1) {
            matchy[v] = pre[v];
            swap(v, matchx[pre[v]]);
        }
        return true;
    }

    void bfs(int i) {
        while (!q.empty()) {
            q.pop();
        }
        q.push(i);
        visx[i] = tmp;
        while (true) {
            while (!q.empty()) {
                int u = q.front();
                q.pop();
                for (int v = 0; v < n; v++) {   //遍历连向右边的所有边
                    if (visy[v] != tmp) {
                        ll delta = lx[u] + ly[v] - g[u][v];
                        if (slack[v] >= delta) {
                            pre[v] = u;
                            if (delta) {
                                slack[v] = delta;
                            }
                            else if (check(v)) {  // delta=0 代表有在相等子图中 找增
广路
                                                  // 找到就return 重建交错树
                                return;
                            }
                        }
                    }
                }
            }
            // 没有增广路 修改顶标
            ll a = inf;
            for (int j = 0; j < n; j++) {
                if (visy[j] != tmp) {
```

```
                    a = min(a, slack[j]);
                }
            }
            for (int j = 0; j < n; j++) {
                if (visx[j] == tmp) {  // S
                    lx[j] -= a;
                }
                if (visy[j] == tmp) {  // T
                    ly[j] += a;
                } else {  // T'
                    slack[j] -= a;
                }
            }
            for (int j = 0; j < n; j++) {
                if (visy[j] != tmp && slack[j] == 0 && check(j)) {
                    return;
                }
            }
        }
    }

    void solve() {
        // 初始顶标
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                lx[i] = max(lx[i], g[i][j]);
            }
        }

        for (int i = 0; i < n; i++) {
            tmp++;
            fill(slack.begin(), slack.end(), inf);
            bfs(i);
        }

        //
        for(int i = 0; i < n; ++i)
        {
            if(g[i][matchx[i]] > 0){
                res += g[i][matchx[i]];
            }else{
                matchx[i] = -1;
            }
        }
        cout << res << "\n";
        for (int i = 0; i < org_n; i++) {
            cout << matchx[i] + 1 << " ";
        }
        cout << "\n";
    }
};
```

# 二者取其一

## 问题模型

有$n$个元素，每个元素属于$A$，$B$两个集合中的一个，当这个元素属于$A$时会有贡献$a_i$，属于$B$时会有贡献$b_i$，同时有若干个组合，当组合里的元素都在$A$时会产生贡献$c_i$，都在$B$时会产生贡献$d_i$。求最大贡献。

## 分析

求最大权闭合子图的问题

### 问题

每一个元素有两个三元组$(a_i,b_i,c_i)$，$(a_i',b_i;,c_i')$，只能选择其中一个三元组

同时我们希望最小化
$\max(\max_{1\le i\le n}a_i-\min_{1\le i\le n}a_i,\max_{1\le i\le n}b_i-\min_{1\le i\le n}b_i,\max_{1\le i\le n}c_i-\min_{1\le i\le n}c_i)$

这种问题我们考虑二分$2-sat$

# 费用流

### 费用流定义

给定一个网络$G(V，E)$，每条边有容量限制$c(u,v)$和单位流量的费用$w(u,v)$

当边$(u,v)$的流量为$f(u,v)$时，需要花费$f(u,v)\times w(u,v)$的费用

注意，费用也满足斜对称性，即$w(u,v)=-w(v,u)$。

以下只针对最小费用最大流，即满足最大流的前提下最小化费用

### SSP算法

每次寻找单位费用最小的增广路进行增广，直到图上不存在增广路为止。

简单说就是把$dinic$算法的$bfs$换成最短路算法即可

复杂度为$O(nmf)$，$f$表示最大费用

```
struct MF {
    struct edge {
        int v, nxt;
        ll cap, cost;
    } e[M];

    int head[N], cnt = 1;
    ll INF = 1e18;
    int n, S, T;    //点的个数，源点，汇点
    ll maxflow = 0, ret = 0;
    int cur[N];
    ll dis[N];
    bool vis[N];

    void init(int _n, int s, int t)
    {
        n = _n;
```

```
        for(int i = 0; i <= n; ++i) head[i] = 0;
        for(int i = 0; i <= n; ++i) cur[i] = 0;
        for(int i = 0; i <= n; ++i) vis[i] = 0;
        S = s;
        T = t;
        cnt = 1;
        maxflow = 0;
        ret = 0;
    }
    void addedge(int u, int v, ll w, ll c) {
        e[++cnt] = {v, head[u], w, c};
        head[u] = cnt;
        e[++cnt] = {u, head[v], 0, -c};
        head[v] = cnt;
    }

    bool spfa()
    {
        for(int i = 0; i <= n; ++i) dis[i] = LINF;
        for(int i = 0; i <= n; ++i) cur[i] = head[i];
        queue<int>q;
        q.push(S), dis[S] = 0, vis[S] = 1;
        while(!q.empty())
        {
            int u = q.front(); q.pop();
            vis[u] = 0;
            for(int i = head[u]; i; i=e[i].nxt)
            {
                int y = e[i].v;
                if(e[i].cap && dis[y] > dis[u] + e[i].cost)
                {
                    dis[y] = dis[u] + e[i].cost;
                    if(!vis[y]) q.push(y), vis[y] = 1;
                }
            }
        }
        return dis[T] != LINF;
    }

    ll dfs(int u, ll flow) {

        if(u == T) return flow;
        vis[u] = 1;
        ll ans = 0;
        for (int &i = cur[u]; i && ans < flow; i = e[i].nxt) {
            int v = e[i].v;
            if(!vis[v] && e[i].cap && dis[v] == dis[u] + e[i].cost)
            {
                ll k = dfs(v, min(flow - ans, e[i].cap));
                if(k) ret += k * e[i].cost, e[i].cap -= k, e[i ^ 1].cap += k,
ans += k;
            }
        }
        vis[u] = 0;
        return ans;
    }

    void mcmf() {
```

```
        ll flow = 0;
        while (spfa()) {
//          flow = dfs(S, INF);
            while((flow = dfs(S, INF)))  {
                maxflow += flow;
            }
        }
    }
} mf;

mf.init(n, s, t);
mf.mcmf();
cout << mf.maxflow << ' ' << mf.ret << '\n';
```

## Primal-Dual 原始对偶算法

仅仅处理负权边保证可以用$dijkstra$

大概是$O(n \times \log m \times f)$

反正比上一个快，虽然网络流都挺玄学的，模板题快近一倍

```
struct MF {
    struct edge {
        int v, nxt;
        ll cap, cost;
    } e[M];

    int head[N], cnt = 1;
    ll INF = 1e18;
    int n, S, T;    //点的个数，源点，汇点
    ll maxflow = 0, ret = 0;    //流量，费用
    int cur[N];
    ll dis[N], h[N];
    bool vis[N];

    void init(int _n, int s, int t)
    {
        n = _n;
        for(int i = 0; i <= n; ++i) head[i] = 0;
        for(int i = 0; i <= n; ++i) cur[i] = 0;
        for(int i = 0; i <= n; ++i) vis[i] = 0;
        S = s;
        T = t;
        cnt = 1;
        maxflow = 0;
        ret = 0;
    }
    void addedge(int u, int v, ll w, ll c) {
        e[++cnt] = {v, head[u], w, c};
        head[u] = cnt;
        e[++cnt] = {u, head[v], 0, -c};
        head[v] = cnt;
    }

    void spfa()
    {
        queue<int> q;
```

```
        for(int i = 0; i <= n; ++i) h[i] = LINF;
        h[S] = 0, vis[S] = 1;
        q.push(S);
        while(!q.empty())
        {
            int u = q.front();
            q.pop();
            vis[u] = 0;
            for(int i = head[u]; i; i = e[i].nxt)
            {
                int v = e[i].v;
                if(e[i].cap && h[v] > h[u] + e[i].cost)
                {
                    h[v] = h[u] + e[i].cost;
                    if(!vis[v]) q.push(v), vis[v] = 1;
                }
            }
        }
    }

    struct mypair{  //dij用的大根堆，重载一下运算符
        ll dis;
        int id;
        bool operator<(const mypair& a)const{
            return dis > a.dis;
        }
        mypair(ll d, int x){ dis = d, id = x; }
    };

    struct node     //用于记录前一位
    {
        int v,e;
    }p[N];

    bool dijkstra()
    {
        priority_queue<mypair> q;
        for(int i = 0; i <= n; ++i) dis[i] = LINF;
        for(int i = 0; i <= n; ++i) vis[i] = 0;
        dis[S] = 0;
        q.push(mypair(0, S));
        while(!q.empty())
        {
            int u = q.top().id;
            q.pop();
            if(vis[u]) continue;
            vis[u] = 1;
            for(int i = head[u]; i; i = e[i].nxt)
            {
                int v = e[i].v;
                ll nc = e[i].cost + h[u] - h[v];
                if(e[i].cap && dis[v] > dis[u] + nc)
                {
                    dis[v] = dis[u] + nc;
                    p[v].v = u;
                    p[v].e = i;
                    if(!vis[v]) q.push(mypair(dis[v], v));
                }
```

```
            }
        }
        return dis[T] != LINF;
    }


    void solve() {
        spfa();
        while (dijkstra()) {
            ll minf = LINF;
            for(int i = 1; i <= n; ++i) h[i] += dis[i];
            for(int i = T; i != S; i = p[i].v) minf = min(minf, e[p[i].e].cap);
            for(int i = T; i != S; i = p[i].v)
            {
                e[p[i].e].cap -= minf;
                e[p[i].e ^ 1].cap += minf;
            }
            maxflow += minf;
            ret += minf *h[T];
        }
    }
} mf;

mf.init(n,s,t);
mf.solve();
cout << mf.maxflow << ' ' << mf.ret << '\n';
```

## zwk费用流

好像挺快的

```
bool vis[200001];int dist[200001];
//解释一下各数组的含义：vis两个用处：spfa里的访问标记，增广时候的访问标记，dist是每个点的距离标号
int n,m,s,t,ans=0;
//s是起点，t是终点，ans是费用答案
int nedge=-1,p[200001],c[200001],cc[200001],nex[200001],head[200001];
//这里是边表，解释一下各数组的含义：p[i]表示以某一点出发的编号为i的边对应点，c表示编号为i的边的流量，cc表示编号为i的边的费用，nex和head不说了吧。。。
inline void addedge(int x,int y,int z,int zz){
    p[++nedge]=y;c[nedge]=z;cc[nedge]=zz;nex[nedge]=head[x];head[x]=nedge;
}
//建边（数组模拟边表倒挂）
inline bool spfa(int s,int t){
    memset(vis,0,sizeof vis);
    for(int i=0;i<=n;i++)dist[i]=1e9;dist[t]=0;vis[t]=1;
//首先SPFA我们维护距离标号的时候要倒着跑，这样可以维护出到终点的最短路径
    deque<int>q;q.push_back(t);
//使用了SPFA的SLF优化（SLF可以自行百度或Google）
    while(!q.empty()){
        int now=q.front();q.pop_front();
        for(int k=head[now];k>-1;k=nex[k])if(c[k^1]&&dist[p[k]]>dist[now]-cc[k])
{
//首先c[k^1]是为什么呢，因为我们要保证正流，但是SPFA是倒着跑的，所以说我们要求c[k]的对应反向边是正的，这样保证走的方向是正确的
            dist[p[k]]=dist[now]-cc[k];
```

```
//因为已经是倒着的了，我们也可以很清楚明白地知道建边的时候反向边的边权是负的，所以减一下就对了
（负负得正）
            if(!vis[p[k]]){
                vis[p[k]]=1;
                if(!q.empty()&&dist[p[k]]
<dist[q.front()])q.push_front(p[k]);else q.push_back(p[k]);
//SLF优化
            }
        }
        vis[now]=0;
    }
    return dist[s]<1e9;
//判断起点终点是否连通
}
inline int dfs(int x,int low){
//这里就是进行增广了
    if(x==t){vis[t]=1;return low;}
    int used=0,a;vis[x]=1;
//这边是不是和dinic很像啊
    for(int k=head[x];k>-1;k=nex[k])if(!vis[p[k]]&&c[k]&&dist[x]-
cc[k]==dist[p[k]]){
//这个条件就表示这条边可以进行增广
        a=dfs(p[k],min(c[k],low-used));
        if(a)ans+=a*cc[k],c[k]-=a,c[k^1]+=a,used+=a;
//累加答案，加流等操作都在这了
        if(used==low)break;
    }
    return used;
}
inline int costflow(){
    int flow=0;
    while(spfa(s,t)){
//判断起点终点是否连通，不连通说明满流，做完了退出
        vis[t]=1;
        while(vis[t]){
            memset(vis,0,sizeof vis);
            flow+=dfs(s,1e9);
//一直增广直到走不到为止（这样也可以省时间哦）
        }
    }
    return flow;//这里返回的是最大流，费用的答案在ans里
}
int main()
{
    memset(nex,-1,sizeof nex);memset(head,-1,sizeof head);
    scanf("%d%d%d%d",&n,&m,&s,&t);
    for(int i=1;i<=m;i++){
        int x,y,z,zz;scanf("%d%d%d%d",&x,&y,&z,&zz);
        addedge(x,y,z,zz);addedge(y,x,0,-zz);
    }
    printf("%d ",costflow());printf("%d",ans);
    return 0;
}
```

# 竞赛图

一个无向完全图给每条边定向，得到的图即为竞赛图

## 性质

竞赛图没有自环和二元环，如果存在环，必然存在三元环。

假设存在比三元环大的环，由于基图是完全图，隔一个点的两点之间必有连线，那么对于点$i$来说，$i$连向的边必须和环的方向一致，否则存在两个相邻的点方向不一致就构成了环，此时$pre[pre[i]]$，我们发现$i, pre[i], pre[pre[i]]$，就构成一个三元环。

任意竞赛图都存在哈密顿路径，即从一条经过所有点一次的路径。

首先容易得到将竞赛图缩点后，必然形成一条链，$n = 3$时显然，考虑新加进来的点，如果不构成环，则必须前面的点可以进，后面的点可以出，因此仍然构成链。

# 克鲁斯卡尔重构树

我们按照克鲁斯卡尔建立最小生成树的过程，最后产生的二叉树为重构树

这棵树有很多奇妙的性质

1. **它是一个二叉堆。**
2. **若边权升序，则它是一个大根堆**
3. **任意两点路径边权最大值为$Kruskal$重构树上$LCA$的点权。**

有了这个东西，我们就可以维护

1. 普通图上，任意两点之间路径上的最大边权的最小值。建完最小生成树后，任意两点之间的最大边权必须大于等于最小生成树上的两点间的最大边权，否则我们就找到一条路径，上面边权最大的边去替换原先的最大边，这与最小生成树的定义不符。于是我们有**任意两点间的最大边权的最小值等于最小生成树上的两点最短路径上的最大值**，即**也等于重构树上这两点的$LCA$的权值**
2. 树上，任意两点的最大距离等于重构树上的两点的$LCA$

# 拓扑

## 定义

给一张有向无环图上的点排序

使得拓扑序在前面的点只有可能向拓扑序在后面的点连边。

```
for(int i = 1; i <= n; ++i)
{
    for(int y:G[x])
    {
        in[y]++;
    }
}
for(int i = 1; i <= n; ++i)
{
    if(!in[i]) q.push(i);
}
while(!q.empty())
{
    int x = q.front();
    q.pop();
```

```
    for(int y:G[x])
    {
        if(--in[y]) q.push(y);
    }
}
```

## 改版

我们可以将队列换成优先队列，从而实现最小/最大字典序的拓扑序。

# 网络流

## 基本概念

我们可以将流网络的每一条有向边看成看成一个流通通道，每条边的容量看成流经该通道的最大速率。

我们将$c(u,v)$记作有向边$u-v$的容量，而$f(u,v)$记作这条边的流量。同时具有以下性质：

1. 容量限制。$0\leq f(u,v)\leq c(u,v)$
2. 流量守恒。$\sum_{v\in V}f(v,u)=\sum_{v\in V}f(u,v)$。即流入等于流出

一个流$f$的值定义如下：

$$|f|=\sum_{v\in V}f(s,v)-\sum_{v\in V}f(v,s)$$

即流$f$的值定义为从源节点流出的总流量减去流入源节点的总流量

## 基础网络流

在网络流中有一个比较重要的概念：残存网络。

假如我们已经找到一个流，那么我们可以在整张图里把这个流给去掉，得到了一个残留图，因为我们的目标是想方设法增大容量，那么我们还需要一个反向流量，使其可以减小正向流量然后增大总的流。

形式化的表示为

$$c_f(u,v)=\begin{cases}c(u,v)-f(u,v) & \text{if}(u,v)\in E\\ f(v,u) & \text{if}(v,u)\in E\\ 0\end{cases}$$

对于抵消操作，可以理解为$u$送给$v$五箱橘子，$v$送给$u$三箱橘子，那么就等价于$u$给$v$两箱橘子。

那么我们就可以在残量图中继续求解最大流

## code

dinic算法复杂度为$O(n^2m)$，通常$1e4-1e5$都能处理

求解二分图匹配复杂度为$O(m\sqrt{n})$

```
struct MF {
    struct edge {
        int v, nxt;
        ll cap;
    } e[M];
```

```
    int head[N], cnt = 1;
    ll INF = 1e18;
    int n, S, T;    //点的个数，源点，汇点
    ll maxflow = 0;
    int dep[N], cur[N], now[N];

    void init(int _n, int s, int t)
    {
        n = _n;
        for(int i = 0; i <= n; ++i) head[i] = 0;
        for(int i = 0; i <= n; ++i) now[i] = 0;
        S = s;
        T = t;
        cnt = 1;
        maxflow = 0;
    }
    void addedge(int u, int v, ll w, ll f = 0) {
        e[++cnt] = {v, head[u], w};
        head[u] = cnt;
        e[++cnt] = {u, head[v], f};
        head[v] = cnt;
    }

    bool bfs() {
        queue<int> q;
//      memset(dep, 0, sizeof(dep));
        for(int i = 0; i <= n; ++i)dep[i] = 0;

        dep[S] = 1;
        q.push(S);
        now[S] = head[S];
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            for (int i = head[u]; i; i = e[i].nxt) {
                int v = e[i].v;
                if (!dep[v] && e[i].cap) {
                    dep[v] = dep[u] + 1;
                    q.push(v);
                    now[v] = head[v];
                    if(v == T) return 1;
                }
            }
        }
        return 0;
    }

    ll dfs(int u, ll flow) {

        if(u == T) return flow;
        ll ret = flow, k;
        for (int i = now[u]; i && ret; i = e[i].nxt) {
            int v = e[i].v;
            now[u] = i;
            if(e[i].cap && dep[v] == dep[u] + 1)
            {
                k = dfs(v, min(ret, e[i].cap));
```

```
                if(!k) dep[v] = 0;
                e[i].cap -= k;
                e[i ^ 1].cap += k;
                ret -= k;
            }
        }
        return flow - ret;
    }

    void dinic() {
        ll flow = 0;
        while (bfs()) {
//          flow = dfs(S, INF);
            while(flow = dfs(S, INF))  {
                maxflow += flow;

            }
        }
    }
} mf;
```

HLPP，$O(n^2\sqrt{m})$

空间比较大

```
template<class T = int>
struct HLPP{
    const int MAXN = 1e5 + 5;
    const T INF = 0x3f3f3f3f;
    struct edge{
        int to, rev;
        T f;
    };
    vector<edge> adj[M];
    deque<int> lst[N];
    vector<int> gap[N];
    T excess[N];
    int highest, height[N], cnt[N], ptr[N], work, N;

    void addEdge(int u, int v, int f, bool isdirected = true){
        adj[u].push_back({v, adj[v].size(), f});
        adj[v].push_back({u, adj[u].size() - 1, isdirected? 0 : f});
    }

    void clear(int n){
        N = n;
        for(int i = 0; i <= n; i++){
            adj[i].clear(), lst[i].clear();
            gap[i].clear();
        }
    }

    void upHeight(int v, int nh){
        ++work;
        if(height[v] != N) --cnt[height[v]];
        height[v] = nh;
        if(nh == N) return;
```

```
        cnt[nh]++, highest = nh;
        gap[nh].push_back(v);
        if(excess[v] > 0){
            lst[nh].push_back(v);
            ++ptr[nh];
        }
    }

    void glovalRelabel(int s, int t){
        work = 0;
        fill(height, height + N + 1, N);
        fill(cnt, cnt + N + 1, 0);
        for(int i = 0; i <= highest; i++){
            lst[i].clear();
            gap[i].clear();
            ptr[i] = 0;
        }
        height[t] = 0;
        queue<int> q({t});
        while(!q.empty()){
            int v = q.front();
            q.pop();
            for(auto &e : adj[v])
            if(height[e.to] == N && adj[e.to][e.rev].f > 0){
                q.push(e.to);
                upHeight(e.to, height[v] + 1);
            }
            highest = height[v];
        }
    }
    void push(int v, edge& e){
        if(excess[e.to] == 0){
            lst[height[e.to]].push_back(e.to);
            ++ptr[height[e.to]];
        }
        T df = min(excess[v], e.f);
        e.f -= df;
        adj[e.to][e.rev].f += df;
        excess[v] -= df;
        excess[e.to] += df;
    }
    void discharge(int v){
        int nh = N;
        for(auto &e : adj[v]){
            if(e.f > 0){
                if(height[v] == height[e.to] + 1){
                    push(v, e);
                    if(excess[v] <= 0) return;
                }
                else{
                    nh = min(nh, height[e.to] + 1);
                }
            }
        }
        if(cnt[height[v]] > 1){
            upHeight(v, nh);
        }
        else{
```

```
            for(int i = height[v]; i < N; i++){
                for(auto j : gap[i]) upHeight(j, N);
                gap[i].clear(); ptr[i] = 0;
            }
        }
    }

    T hlpp(int s, int t){
        fill(excess, excess + N + 1, 0);
        excess[s] = INF, excess[t] = -INF;
        glovalRelabel(s, t);
        for(auto &e : adj[s]) push(s, e);
        for(; highest >= 0; -- highest){
            while(lst[highest].size()){
                int v = lst[highest].back();
                lst[highest].pop_back();
                discharge(v);
                if(work > 4 * N) glovalRelabel(s, t);
            }
        }
        return excess[t] + INF;
    }

};
HLPP<int>mf;
```

IASP，空间较小，时间比DINIC快小一倍

注意从零开始标号

```
struct ISAP {
    int n, m, s, t;
    vector<Edge> edges;
    vector<int> G[maxn];
    bool vis[maxn];
    int d[maxn];
    int cur[maxn];
    int p[maxn];
    int num[maxn];

    void AddEdge(int from, int to, ll cap, ll f = 0) {
        edges.push_back(Edge(from, to, cap, 0));
        edges.push_back(Edge(to, from, f, 0));
        m = edges.size();
        G[from].push_back(m - 2);
        G[to].push_back(m - 1);
    }

    bool BFS() {
        memset(vis, 0, sizeof(vis));
        queue<int> Q;
        Q.push(t);
        vis[t] = 1;
        d[t] = 0;
        while (!Q.empty()) {
            int x = Q.front();
            Q.pop();
```

```
            for (int i = 0; i < G[x].size(); i++) {
                Edge& e = edges[G[x][i] ^ 1];
                if (!vis[e.from] && e.cap > e.flow) {
                    vis[e.from] = 1;
                    d[e.from] = d[x] + 1;
                    Q.push(e.from);
                }
            }
        }
        return vis[s];
    }

    void init(int n) {
        this->n = n;
        for (int i = 0; i < n; i++) G[i].clear();
        edges.clear();
    }

    ll Augment() {
        int x = t; ll a = LINF;
        while (x != s) {
            Edge& e = edges[p[x]];
            a = min(a, e.cap - e.flow);
            x = edges[p[x]].from;
        }
        x = t;
        while (x != s) {
            edges[p[x]].flow += a;
            edges[p[x] ^ 1].flow -= a;
            x = edges[p[x]].from;
        }
        return a;
    }

    ll Maxflow(int s, int t) {
        this->s = s;
        this->t = t;
        ll flow = 0;
        BFS();
        memset(num, 0, sizeof(num));
        for (int i = 0; i < n; i++) num[d[i]]++;
        int x = s;
        memset(cur, 0, sizeof(cur));
        while (d[s] < n) {
            if (x == t) {
                flow += Augment();
                x = s;
            }
            int ok = 0;
            for (int i = cur[x]; i < G[x].size(); i++) {
                Edge& e = edges[G[x][i]];
                if (e.cap > e.flow && d[x] == d[e.to] + 1) {
                    ok = 1;
                    p[e.to] = G[x][i];
                    cur[x] = i;
                    x = e.to;
                    break;
                }
```

```
            }
            if (!ok) {
                int m = n - 1;
                for (int i = 0; i < G[x].size(); i++) {
                    Edge& e = edges[G[x][i]];
                    if (e.cap > e.flow) m = min(m, d[e.to]);
                }
                if (--num[d[x]] == 0) break;
                num[d[x] = m + 1]++;
                cur[x] = 0;
                if (x != s) x = edges[p[x]].from;
            }
        }
        return flow;
    }
} mf;

mf.init(n);
mf.AddEdge(u - 1,v - 1,w,w);//或者mf.AddEdge(u - 1,v - 1,w);前面是双向边
cout<<mf.Maxflow(s,t)<<'\n';
```

# 二分图匹配的可行边与必须边

## 必须边

满足两个条件:

1. $(x,y)$是匹配边
2. 删除$(x,y)$后，不能找到一条从$x$到$y$的增广路

## 可行边

满足两个条件之一:

1. $(x,y)$是匹配边
2. $(x,y)$不是匹配边，假设当前$x$与$v$匹配，$y$与$u$匹配，连接边$(x,y)$后，还能找到一条$u$到$v$的增广路

## 转化

我们不妨将非匹配边看作从左部到右部的有向边，匹配边看作右部到左部的有向边。

必须边：$(x,y)$是匹配边并且$x$，$y$在新图中属于不同的强连通分量

可行边：$(x,y)$是匹配边或者$x$，$y$在新图中属于相同同的强连通分量

# 最小割

## 割的定义

给定一个网络$G=(V,E)$，源点是$S$，汇点是$T$。若一个边集$E'\subset E$被删去之后，源点和汇点不再联通，则该边集称为网络的**割**。其中边容量最小的割称为最小割。

## 最大流最小割

直观来说，因为$S$和$T$的最大流就在那里，至少得删去这些边才能保证$S$和T不再联通。

### 方案

我们可以通过源点开始$DFS$，每次走残量大于零的边，找到所有$S$相连的点

```
//val是边残量
void dfs(int u) {
    vis[u] = 1;
    for (int i = lnk[u]; i; i = nxt[i]) {
        int v = ter[i];
        if (!vis[v] && val[i]) dfs(v);
    }
}
```

## 割边

如果需要在最小割的前提下最小化割边数量，那么先求出最小割，把没有满流的边容量改成 $\infty$，满流的边容量改成1，重新跑一遍最小割就可求出最小割边数量；如果没有最小割的前提，直接把所有边的容量设成 $1$ ，求一遍最小割就好了。

## 问题模型1

有$n$个物品和两个集合$A, B$，如果一个物品**没有**放在$A$集合会花费$a_i$，**没有**放在$B$集合会花费$b_i$。同时还有m个限制条件$(u, v, w)$，表示$u$和$v$**没有**在一个集合会花费$w$。每个物品必须且只能属于一个集合。求最小代价。

## 分析

我们不妨在$A$集合引入源点$S$，$B$集合引入源点$T$。对于每个点，由源点连一条$a_i$的边，向汇点连一条$b_i$的边，那么$A$集合和$B$集合不连通，表示这个点一定属于其中一个集合。那么割去的边就是我们需要支付的代价。考虑限制条件，我们可以让$u$和$v$连一条$w$的双向边，那么我们仍然可以沿用上面的结论。求一个最小割。

## 问题模型2

最大权值闭合图，即在一张有向图中，每个点都有一个权值（可以为正或负或0），你需要选择一个权值最大的子图，使得子图中每个点的后继都在这个子图里面。

## 常见的问题描述

1. 我们有一个集合$A = \{a_1, a_2, \cdots, a_n\}$和一个集合$B = \{b_1, b_2, \cdots, b_m\}$，对于$a_i$，我们必须选择$b_{k_1}, b_{k_2}, \cdots, b_{k_r}$，然后求最大的收益。
2. e.g. W 教授正在为国家航天中心计划一系列的太空飞行。每次太空飞行可进行一系列商业性实验而获取利润。现已确定了一个可供选择的实验集合$E = \{E_1, E_2, \cdots, E_m\}$，和进行这些实验需要使用的全部仪器的集合 $I = \{I_1, I_2, \cdots, I_n\}$。实验$E_j$ 需要用到的仪器是 $I$ 的子集$R_j \subseteq I$。

   配置仪器 $I_k$ 的费用为$c_k$ 美元。实验$E_j$ 的赞助商已同意为该实验结果支付$p_j$美元。W 教授的任务是找出一个有效算法，确定在一次太空飞行中要进行哪些实验并因此而配置哪些仪器才能使太空飞行的净收益最大。这里净收益是指进行实验所获得的全部收入与配置仪器的全部费用的差额。

   对于给定的实验和仪器配置情况，编程找出净收益最大的试验计划。

### 分析

首先每一个闭合子图都对应原图中的一个割。因为每一个割都将图分成两部分，那么与$S$相连的部分就是一个闭合子图

然后我们希望最小割去除的边不是原图的，因此我们可以将原图的边容量设为$+\infty$。

如果一个点连向$T$的边被割掉了，那么等价于我们选择了这个点，而$S$连向这个点的边被割掉等价于我们放弃了这个点。

### 做法

建立超级源点和超级汇点，若某个点权值为正，那么连一条$S$连向这个点的边，为负则连一条这个点向$T$的边，边容量都是这个点权值的绝对值。这样我们就可以把负点与我们不要的正点同号。

**最大权值和 = 所有正权值之和 − 最小割 = 所有正权值之和 − 最大流**

### 问题

给定一张 n 个点的无向图。求最少删除多少个点，使得图不连通。

### 分析

将每个点拆成出点和入点，将其转化为割断出点入点。

其他边都设为$+\infty$。

### 网络流的优化

网络流处理的其实是单向边，对于双向边，可以正反连一条容量都为$c$的。

# 有向图强连通分量

### 核心思想在于锁住连通分量往外走到路

对于有环图，我们可以进行缩点将其当成有向无环图考虑

我们考虑$dfs$，我们在结束对这个点的后续遍历时将这个点放入栈中，那么对于任意一条有向边$(u,v)$，$u$一定在$v$的上面。

如果我们先$dfs(u)$，那么只有在$dfs(v)$结束后才会结束因此$u$一定在$v$的上面。

如果我们先$dfs(v)$，那么不会遍历到$u$，因此$u$还是在$v$的上面。

接着我们按照自底向上的顺序访问栈，并且尝试反向$dfs$

如果$u$在$v$的上面，我们反向$dfs(v)$时，遍历到了$u$，说明$u,v$互相可达，也就是构成了环。

## code

```
bool v1[N],v2[N];
vector<int>G[N],rG[N];
int st[N];
int top;
//原图dfs
void dfs(int x,int f)
{
    if(v1[x]) return ;
    v1[x] = 1;
```

```
    for(auto y : G[x])
    {
        if(y == f) continue;
        dfs(y, x);
    }
    st[++top] = x;
}
//反图dfs
void rdfs(int x,int f)
{
    if(v2[x]) return ;
    v2[x] = 1;
    for(auto y : rG[x])
    {
        if(y == f) continue;
        rdfs(y, x);
    }
}
void solve()
{
    for(int i = 1; i <= n; ++i)
    {
        if(!v1[i]) dfs(i, 0);
    }
    for(int i = 1; i <= n; ++i)
    {
        if(!v2[st[i]])
        {
            rdfs(st[i], 0);
        }
    }
}
```

# 最短路

## 单源最短路

### Dijkstra

时间复杂度$O((n+m)\log m)$

无法处理负权边

```
ll dist[N];
bool vis[N];
priority_queue<pair<ll,int> >q;
void dijkstra(int s)
{
    memset(dist, 0, sizeof(dist));
    memset(v, 0, sizeof(v));
    dist[s] = 0;
    q.push({0, s});
    while(!q.empty())
    {
        int x = q.top().second; q.pop();
```

```
        if(v[x]) continue;
        v[x] = 1;
        for(int i = head[x]; i; i=nxt[i])
        {
            int y = ver[i];
            ll w = edge[i];
            if(d[y] > d[x] + w)
            {
                d[y] = d[x] + w;
                q.push({-d[y], y});
            }
        }
    }
}
```

### SPFA

通常时间复杂度为$O(km)$，$k$是比较小的常数

但是可以卡到$O(nm)$

```
ll dist[N];
bool vis[N];
queue<int>q;
void spfa(int s)
{
    memset(dist, 0, sizeof(dist));
    memset(v, 0, sizeof(v));
    dist[s] = 0;
    v[x] = 1;
    q.push(s);
    while(!q.empty())
    {
        int x = q.front(); q.pop();
        v[x] = 0;
        for(int i = head[x]; i; i=nxt[i])
        {
            int y = ver[i];
            ll w = edge[i];
            if(d[y] > d[x] + w)
            {
                d[y] = d[x] + w;
                if(!v[y]) q.push(y), v[y] = 1;
            }
        }
    }
}
```

### 双端队列BFS

针对边权只有0或1的情况，只有1我们可以直接bfs

```
ll dist[N];
bool vis[N];
deeue<int>q;
void bfs(int s)
{
```

```
    memset(dist, 0, sizeof(dist));
    memset(v, 0, sizeof(v));
    q.push(s);
    dist[v] = 0;
    while(!q.empty())
    {
        int x = q.front(); q.pop_front();
        if(v[x]) continue;
        v[x] = 1;
        for(int i = head[x]; i; i = nxt[i])
        {
            int y = ver[i];
            int w = edge[i];
            if(w)q.push_back(y);
            else q.push_front(y);
        }
    }
}
```

# 多源最短路

## flody

时间复杂度$O(n^3)$

$d[i][i]=0$，不存在的边为$\infty$

```
void floyd(int s)
{
    for(int k = 1; k <= n; ++k)
    {
        for(int i = 1; i <= n; ++i)
        {
            for(int j = 1; j <= n; ++j)
            {
                d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
            }
        }
    }
}
```

也可以用来求传递闭包

```
void floyd(int s)
{
    for(int k = 1; k <= n; ++k)
    {
        for(int i = 1; i <= n; ++i)
        {
            for(int j = 1; j <= n; ++j)
            {
                d[i][j] |= d[i][k] & d[k][j];
            }
        }
    }
}
```

### Johnson 全源最短路径算法

对于负权图又要跑全源最短路，我们可以利用Johnson 全源最短路径算法

我们新建一个虚拟节点（在这里我们就设它的编号为0）。从这个点向其他所有点连一条边权为0的边。

接下来用 Bellman–Ford 算法求出从 0号点到其他所有点的最短路，记为 $h_i$。

假如存在一条从 $u$ 点到 $v$ 点，边权为 $w$ 的边，则我们将该边的边权重新设置为 $w + h_u - h_v$。

接下来以每个点为起点，跑 $n$ 轮 Dijkstra 算法即可求出任意两点间的最短路了。

# tanjan

## 有向图

将任意有向图转化为强连通，需要$\max(p,q)$条有向边，其中$p$和$q$表示零入度点和零出度点的个数

注意重边

```
//有向图tanjan
namespace tanjan
{
    vector<int>G[N];
    int st[N], ins[N], c[N];    //ins表示已经找到的点中不在环上的点，标记为1
    int dfn[N], low[N];
    vector<int>scc[N];
    int ind = 0, top = 0, cnt = 0;
    void add(int x, int y){G[x].push_back(y);}
    void tanjan(int x)
    {
        dfn[x] = low[x] = ++ind;
        st[++top] = x, ins[x] = 1;
        for(auto y:G[x])
        {
            if(!dfn[y])
            {
                tanjan(y);
                low[x] = min(low[x], low[y]);
            }
            else if(ins[y])
            {
                low[x] = min(low[x], dfn[y]);
            }
```

```
        }
        if(low[x] == dfn[x])
        {
            cnt++;
            int y;
            do{
                y = st[top--]; ins[y] = 0;
                c[y] = cnt; scc[cnt].push_back(y);
            }while(x != y);
        }
    }

    void solve(int n)
    {
        for(int i = 1; i <= n; ++i)
        {
            if(!dfn[i]) tanjan(i);
        }
    }

    vector<int>G_c[N];
    void add_c(int x, int y){ G_c[x].push_back(y);}

    void get(int n) //缩点
    {
        for(int x = 1; x <= n; ++x)
        {
            for(auto y:G[x])
            {
                if(c[x] == c[y])continue;
                add_c(c[x],c[y]);
            }
        }
     }
}
using tanjan::add;
void solve()
{
    int n,m;cin>>n>>m;
    for(int i=1;i<=m;++i)
    {
        int x,y;cin>>x>>y;
        add(x,y);
    }
    tanjan::solve(n);
}
```

# 数学

## 常用优化

// 向下取整
$r = n/(n/l);$
// 向上取整
$r = (n-1)/((n+l-1)/l-1);$

## 位运算

### 返回x 的最后一位1 是从后向前第几位：

```
__builtin_ffs(unsigned x)
__builtin_ffsll(unsigned long long x)

__builtin_ffs(3) = 1
```

### 返回 x 的二进制下前导的 0 的个数：

```
__builtin_clz(unsigned x)
__builtin_clzll(unsigned long long x)

__builtin_clz(4) = 29//int
__builtin_clzll(4) = 61//long long
```

### 返回 x 的二进制下末尾的 0 的个数：

```
__builtin_ctz(unsigned x)
__builtin_ctzll(unsigned long long x)

__builtin_ctz(4) = 2
```

### 返回 x 的二进制下 1 的个数

```
__builtin_popcount(unsigned x)
__builtin_popcountll(unsigned long long x)

__builtin_popcount(3) = 2
```

### 返回 x 的二进制下 1 的个数的奇偶性

```
__builtin_parity(unsigned x)
__builtin_parityll(unsigned long long x)

__builtin_parity(3) = 0//偶数是 0 奇数是 1
```

### 进制转换

```
itoa(int x,char *s ,int _radix)
itoa(3,a,2)
a="11"
```

### 给定数组$a$，询问$x$,查询$max_{y\in a}(popcount(x\wedge y))$

#### 法1

$$max_{y\in a}(popcount(x\wedge y))=m-min_{y\in a}(popcount(x^c\wedge y))$$

其中$x^c$是$x$的翻转$(0变1,1变0)$

然后就是最短路的问题

#### 法2

$$popcount(x\wedge y)=popcount(x)+popcount(y)-2*popcount(x\&y)$$

[COCI '22 Contest 5 #2 Diskurs 的题解 - DMOJ: Modern Online Judge]

```
for(int s = 0; s < (1 << m); ++s) {
        f[s] = -m;
        if(b[s]) f[s] = __builtin_popcount(s);
        for(int i = 0; i < m; ++i) {
            if((1 << i) & s) f[s] = std::max(f[s], f[s ^ (1 << i)]);
        }
    }
    for(int s = (1 << m) - 1; s >= 0; --s) {
        dp[s] = f[s];
        for(int i = 0; i < m; ++i) {
            if(s & (1 << i)) continue;
            dp[s] = std::max(dp[s], dp[s | (1 << i)] - 2);
        }
    }
```

# 蒙哥马利模乘

## 蒙哥马利约减

#### 看不懂

通常我们计算$xy \mod n$，我们需要$x*y-\lfloor\frac{x*y}{n}\rfloor$

蒙哥马利模乘约减的思路是通过变换，将需要取模的数控制到很小的范围，最需要最多一次减法完成取模运算。通过选择除数为2的幂次，从而通过移位加快速度。

在竞赛范围(模数为正奇数)内，我们认为以下式子总会成立

$$RR'\equiv 1(\mod N)$$
$$RR'-NN'=1$$
$$RR'-NN'\equiv 1(\mod R)$$
$$-NN'\equiv 1(\mod R)$$
$$0<R'<N$$
$$0<N'<R$$

如果我们需要计算约减形式，即对于$T$我们想要求$TR'$

$$T=T(RR'-NN')=TRR-TNN'$$
$$TR'=\frac{T+TNN'}{R}$$

记$m=TN'$

$$\begin{aligned}
&TR' \mod N \\
=&\frac{T+mN}{R} \mod N \\
=&\frac{T+(\lfloor\frac{T}{R}\rfloor R+(T \mod R))NN'}{R} \mod N \\
=&\frac{T+(T \mod R)NN'}{R}+\lfloor\frac{T}{R}\rfloor NN' \mod N \\
=&\frac{T+(T \mod R)NN'}{R} \mod N
\end{aligned}$$

**个人想法?**

利用$exgcd$求$RR'-NN'=1$中的$R',N'$

约简的话，对于数$x$，计算$m=xN'(\mod R)$

```
template <std::uint32_t P>struct MontInt {
    using u32 = std::uint32_t;
    using u64 = std::uint64_t;
    u32 v;
    static constexpr u32 get_r() {
        u32 iv = P;

        for (u32 i = 0; i != 4; ++i)
            iv *= 2U - P * iv;

        return -iv;
    }
    static constexpr u32 r = get_r(), r2 = -u64(P) % P;//限定词不能省

    MontInt() = default;
    ~MontInt() = default;
    MontInt(u32 v) : v(reduce(u64(v) * r2)) {}
    MontInt(const MontInt &rhs) : v(rhs.v) {}

    u32 reduce(u64 x) {
        return x + (u64(u32(x) * r) * P) >> 32;
    }

    u32 norm(u32 x) {
        return x - (P & -(x >= P));
    }
    u32 get() {
        u32 res = reduce(v) - P;
        return res + (P & -(res >> 31));
    }


    MontInt operator-() const {
        MontInt res;
        return res.v = (P << 1 & -(v != 0)) - v, res;
    }
    MontInt inv() const {
        return pow(-1);
    }
```

```
    MontInt &operator=(const MontInt &rhs) {
        return v = rhs.v, *this;
    }
    MontInt &operator+=(const MontInt &rhs) {
        return v += rhs.v - (P << 1), v += P << 1 & -(v >> 31), *this;
    }
    MontInt &operator-=(const MontInt &rhs) {
        return v -= rhs.v, v += P << 1 & -(v >> 31), *this;
    }
    MontInt &operator*=(const MontInt &rhs) {
        return v = reduce(u64(v) * rhs.v), *this;
    }
    MontInt &operator/=(const MontInt &rhs) {
        return this->operator*=(rhs.inv());
    }
    friend MontInt operator+(const MontInt &lhs,
                                        const MontInt &rhs) {
        return MontInt(lhs) += rhs;
    }
    friend MontInt operator-(const MontInt &lhs,
                                        const MontInt &rhs) {
        return MontInt(lhs) -= rhs;
    }
    friend MontInt operator*(const MontInt &lhs,
                                        const MontInt &rhs) {
        return MontInt(lhs) *= rhs;
    }
    friend MontInt operator/(const MontInt &lhs,
                                        const MontInt &rhs) {
        return MontInt(lhs) /= rhs;
    }
    friend bool operator==(const MontInt &lhs, const MontInt &rhs) {
        return norm(lhs.v) == norm(rhs.v);
    }
    friend bool operator!=(const MontInt &lhs, const MontInt &rhs) {
        return norm(lhs.v) != norm(rhs.v);
    }
    constexpr MontInt pow(ll y) const {
        if ((y %= P - 1) < 0)
            y += P - 1; // phi(P) = P - 1, assume P is a prime number

        MontInt res(1), x(*this);

        for (; y != 0; y >>= 1, x *= x)
            if (y & 1)
                res *= x;

        return res;
    }
};
auto ans = MontInt<998244353>(i);
cout << ans.inv().get() << '\n';
```

```
template <std::uint32_t P> struct MontgomeryModInt32 {
public:
```

```
    using i32 = std::int32_t;
    using u32 = std::uint32_t;
    using i64 = std::int64_t;
    using u64 = std::uint64_t;

private:
    u32 v;

    static constexpr u32 get_r() {
        u32 iv = P;

        for (u32 i = 0; i != 4; ++i)
            iv *= 2U - P * iv;

        return -iv;
    }

    static constexpr u32 r = get_r(), r2 = -u64(P) % P;

    static_assert((P & 1) == 1);
    static_assert(-r * P == 1);
    static_assert(P < (1 << 30));

public:
    static constexpr u32 pow_mod(u32 x, u64 y) {
        if ((y %= P - 1) < 0)
            y += P - 1;

        u32 res = 1;

        for (; y != 0; y >>= 1, x = u64(x) * x % P)
            if (y & 1)
                res = u64(res) * x % P;

        return res;
    }

    static constexpr u32 get_pr() {
        u32 tmp[32] = {}, cnt = 0;
        const u64 phi = P - 1;
        u64 m = phi;

        for (u64 i = 2; i * i <= m; ++i) {
            if (m % i == 0) {
                tmp[cnt++] = i;

                while (m % i == 0)
                    m /= i;
            }
        }

        if (m > 1)
            tmp[cnt++] = m;

        for (u64 res = 2; res <= phi; ++res) {
            bool flag = true;

            for (u32 i = 0; i != cnt && flag; ++i)
```

```
            flag &= pow_mod(res, phi / tmp[i]) != 1;

        if (flag)
            return res;
    }

    return 0;
}

MontgomeryModInt32() = default;
~MontgomeryModInt32() = default;
constexpr MontgomeryModInt32(u32 v) : v(reduce(u64(v) * r2)) {}
constexpr MontgomeryModInt32(const MontgomeryModInt32 &rhs) : v(rhs.v) {}
static constexpr u32 reduce(u64 x) {
    return x + (u64(u32(x) * r) * P) >> 32;
}
static constexpr u32 norm(u32 x) {
    return x - (P & -(x >= P));
}
constexpr u32 get() const {
    u32 res = reduce(v) - P;
    return res + (P & -(res >> 31));
}
explicit constexpr operator u32() const {
    return get();
}
explicit constexpr operator i32() const {
    return i32(get());
}
constexpr MontgomeryModInt32 &operator=(const MontgomeryModInt32 &rhs) {
    return v = rhs.v, *this;
}
constexpr MontgomeryModInt32 operator-() const {
    MontgomeryModInt32 res;
    return res.v = (P << 1 & -(v != 0)) - v, res;
}
constexpr MontgomeryModInt32 inv() const {
    return pow(-1);
}
constexpr MontgomeryModInt32 &operator+=(const MontgomeryModInt32 &rhs) {
    return v += rhs.v - (P << 1), v += P << 1 & -(v >> 31), *this;
}
constexpr MontgomeryModInt32 &operator-=(const MontgomeryModInt32 &rhs) {
    return v -= rhs.v, v += P << 1 & -(v >> 31), *this;
}
constexpr MontgomeryModInt32 &operator*=(const MontgomeryModInt32 &rhs) {
    return v = reduce(u64(v) * rhs.v), *this;
}
constexpr MontgomeryModInt32 &operator/=(const MontgomeryModInt32 &rhs) {
    return this->operator*=(rhs.inv());
}
friend MontgomeryModInt32 operator+(const MontgomeryModInt32 &lhs,
                                    const MontgomeryModInt32 &rhs) {
    return MontgomeryModInt32(lhs) += rhs;
}
friend MontgomeryModInt32 operator-(const MontgomeryModInt32 &lhs,
                                    const MontgomeryModInt32 &rhs) {
    return MontgomeryModInt32(lhs) -= rhs;
```

```
    }
    friend MontgomeryModInt32 operator*(const MontgomeryModInt32 &lhs,
                                        const MontgomeryModInt32 &rhs) {
        return MontgomeryModInt32(lhs) *= rhs;
    }
    friend MontgomeryModInt32 operator/(const MontgomeryModInt32 &lhs,
                                        const MontgomeryModInt32 &rhs) {
        return MontgomeryModInt32(lhs) /= rhs;
    }
    friend bool operator==(const MontgomeryModInt32 &lhs, const
MontgomeryModInt32 &rhs) {
        return norm(lhs.v) == norm(rhs.v);
    }
    friend bool operator!=(const MontgomeryModInt32 &lhs, const
MontgomeryModInt32 &rhs) {
        return norm(lhs.v) != norm(rhs.v);
    }
    friend std::istream &operator>>(std::istream &is, MontgomeryModInt32 &rhs) {
        return is >> rhs.v, rhs.v = reduce(u64(rhs.v) * r2), is;
    }
    friend std::ostream &operator<<(std::ostream &os, const MontgomeryModInt32
&rhs) {
        return os << rhs.get();
    }
    constexpr MontgomeryModInt32 pow(i64 y) const {
        if ((y %= P - 1) < 0)
            y += P - 1; // phi(P) = P - 1, assume P is a prime number

        MontgomeryModInt32 res(1), x(*this);

        for (; y != 0; y >>= 1, x *= x)
            if (y & 1)
                res *= x;

        return res;
    }
};
```

# 多项式

## 多项式概述

### 常用多项式

| 名称 | 公式 | 方法 | 复杂度 |
| --- | --- | --- | --- |
| 多项式乘法/卷积 | $F(x)=A(x)B(x)$ | NTT/FFT | $O(nlogn)$ |
| 多项式求逆 | $B(x)\equiv 2B_1(x)-B_1^2(x)A(x)(\ mod\ \ x^n)$ | 分治+多项式乘法 | $O(nlogn)$ |
| 多项式除法 | $A^R(x)B^{R^{-1}}(x)\equiv D^R(x)(\ mod\ \ x^{n-m+1})$ | 多项式求逆 | $O(nlogn)$ |
| 多项式取模 | $P(x)=A(X)-D(X)B(X)$ | 多项式乘法 | $O(nlogn)$ |
| 多项式求导 | $F'(x)=\sum_{i=1}^{n}ia_ix^{i-1}$ | --- | $O(n)$ |
| 积分 | $\int F(x)dx=\sum_{i=1}^{n}\frac{a_ix^{i+1}}{i+1}$ | --- | $O(n)$ |
| 开根 | $B(x)\equiv\frac{B_1^2(x)+A(x)}{2B_1(x)}$ | 分治+多项式求逆 | $O(nlogn)$ |
| ln | $ln(F(x))=\int F'(x)F^{-1}(x)$ | 多项式求逆 | $O(nlogn)$ |
| exp | $F(x)\equiv F_0(x)(1-lnF_0(x)+A(x))(\ mod\ \ x^n)$ | 多项式ln | $O(nlogn)$ |

## 关于limit,L,RR[N]

```
for(limit = 1, L = 0; limit <= (n + m) * 2; limit <<= 1) L ++ ;
for(int i = 0; i < limit; ++ i)
   RR[i] = (RR[i >> 1] >> 1) | ((i & 1) << (L - 1));
```

## 卷积的性质

交换律

$$f*g=g*f$$

结合律

$$(f*g)*h=f*(g*h)$$

## 大部分操作

## note

1. **0次项，1次项，。。。**
2. **N的大小至少为1<<t**
3. **记得init()**
4. **对于大数幂次，要注意乘回来的幂次对$\phi(p)$取模**

```
namespace Poly
{
```

```
    #define mul(x, y) (1ll * x * y >= mod ? 1ll * x * y % mod : 1ll * x * y)
    #define minus(x, y) (1ll * x - y < 0 ? 1ll * x - y + mod : 1ll * x - y)
    #define plus(x, y) (1ll * x + y >= mod ? 1ll * x + y - mod : 1ll * x + y)//上面其实没用到
    #define ck(x) (x >= mod ? x - mod : x)//取模运算太慢了

    typedef vector<int> poly;
    const int G = 3;//根据具体的模数而定，原根可不一定不一样！！！
    //一般模数的原根为 2 3 5 7 10 6
    const int inv_G = qpow(G, mod - 2),tt = 22;
    int deer[2][tt][(1 << tt)];
    vector<int>RR(1 << (tt + 1), 0),inv(1 << tt, 0);

    void init(const int t) {//预处理出来NTT里需要的w和wn，砍掉了一个log的时间
        assert(t < tt);//一定要注意！！
        for(int p = 1; p <= t; ++ p) {
            int buf1 = qpow(G, (mod - 1) / (1 << p));
            int buf0 = qpow(inv_G, (mod - 1) / (1 << p));
            deer[0][p][0] = deer[1][p][0] = 1;
            for(int i = 1; i < (1 << p); ++ i) {
                deer[0][p][i] = 1ll * deer[0][p][i - 1] * buf0 % mod;//逆
                deer[1][p][i] = 1ll * deer[1][p][i - 1] * buf1 % mod;
            }
        }
        inv[1] = 1;
        for(int i = 2; i <= (1 << t); ++ i)
            inv[i] = 1ll * inv[mod % i] * (mod - mod / i) % mod;
    }

    int NTT_init(int n) {//快速数论变换预处理
        int limit = 1, L = 0;
        while(limit <= n) limit <<= 1, L ++ ;
        assert(L < tt);
        assert(limit < 1 << (tt + 1));
        for(int i = 0; i < limit; ++ i)
            RR[i] = (RR[i >> 1] >> 1) | ((i & 1) << (L - 1));
        return limit;
    }

    void NTT(poly &A, bool type, int limit) {//快速数论变换
        A.resize(limit);
        for(int i = 0; i < limit; ++ i)
            if(i < RR[i])
                swap(A[i], A[RR[i]]);
        for(int mid = 2, j = 1; mid <= limit; mid <<= 1, ++ j) {
            int len = mid >> 1;
            for(int pos = 0; pos < limit; pos += mid) {
//                auto wn = deer[type][j].begin();
                for(int i = pos, p = 0; i < pos + len; ++ i, ++ p) {
                    int tmp = 1ll * deer[type][j][p] * A[i + len] % mod;
                    A[i + len] = ck(A[i] - tmp + mod);
                    A[i] = ck(A[i] + tmp);
                }
            }
        }
        if(type == 0) {
            for(int i = 0; i < limit; ++ i)
                A[i] = 1ll * A[i] * inv[limit] % mod;
```

```
        }
    }

    poly poly_mul(poly A, poly B) {//多项式乘法
        int deg = A.size() + B.size() - 1;
        int limit = NTT_init(deg);
        poly C(limit);
        NTT(A, 1, limit);
        NTT(B, 1, limit);
        for(int i = 0; i < limit; ++ i)
            C[i] = 1ll * A[i] * B[i] % mod;
        NTT(C, 0, limit);
        C.resize(deg);
        return C;
    }

    poly poly_inv(poly &f, int deg) {//多项式求逆 deg<f.szie()
        if(deg == 1)
            return poly(1, qpow(f[0], mod - 2));

        poly A(f.begin(), f.begin() + deg);
        poly B = poly_inv(f, (deg + 1) >> 1);
        int limit = NTT_init(deg << 1);
        NTT(A, 1, limit), NTT(B, 1, limit);
        for(int i = 0; i < limit; ++ i)
            A[i] = B[i] * (2 - 1ll * A[i] * B[i] % mod + mod) % mod;
        NTT(A, 0, limit);
        A.resize(deg);
        return A;
    }

    poly poly_dev(poly f) {//多项式求导
        int n = f.size();
        for(int i = 1; i < n; ++ i) f[i - 1] = 1ll * f[i] * i % mod;
        if(n > 1)f.resize(n - 1);
        else f[0] = 0;
        return f.resize(n - 1), f;//求导整体左移，第0项不要
    }

    poly poly_idev(poly f) {//多项式求积分
        int n = f.size();
        for(int i = n - 1; i ; -- i) f[i] = 1ll * f[i - 1] * inv[i] % mod;
        return f[0] = 0, f;//积分整体右移，第0项默认为0
    }

    poly poly_ln(poly f, int deg) {//多项式求对数，第一项为1
        poly A = poly_idev(poly_mul(poly_dev(f), poly_inv(f, deg)));
        return A.resize(deg), A;
    }

    poly poly_exp(poly &f, int deg) {//多项式求指数，第一项为0
        if(deg == 1)
            return poly(1, 1);

        poly B = poly_exp(f, (deg + 1) >> 1);
        B.resize(deg);
        poly lnB = poly_ln(B, deg);
        for(int i = 0; i < deg; ++ i)
```

```
            lnB[i] = ck(f[i] - lnB[i] + mod);

        int limit = NTT_init(deg << 1);//n -> n^2
        NTT(B, 1, limit), NTT(lnB, 1, limit);
        for(int i = 0; i < limit; ++ i)
            B[i] = 1ll * B[i] * (1 + lnB[i]) % mod;
        NTT(B, 0, limit);
        B.resize(deg);
        return B;
    }

    poly poly_sqrt(poly &f, int deg) {//多项式开方,第一项是1
        if(deg == 1) return poly(1, 1);
        poly A(f.begin(), f.begin() + deg);
        poly B = poly_sqrt(f, (deg + 1) >> 1);
        poly IB = poly_inv(B, deg);
        int limit = NTT_init(deg << 1);
        NTT(A, 1, limit), NTT(IB, 1, limit);
        for(int i = 0; i < limit; ++ i)
            A[i] = 1ll * A[i] * IB[i] % mod;
        NTT(A, 0, limit);
        for(int i =0; i < deg; ++ i)
            A[i] = 1ll * (A[i] + B[i]) * inv[2] % mod;
        A.resize(deg);
        return A;
    }

    poly poly_pow(poly f, int k) {//多项式快速幂,第一项得是1
        f = poly_ln(f, f.size());
        for(auto &x : f) x = 1ll * x * k % mod;
        return poly_exp(f, f.size());
    }

    poly poly_ksm(poly f, int k) {//多项式快速幂,适用于初始只有几项，同时所有项都需要的情况,会比上面那个快一点
        poly res(1, 1);
        while(k){
            if(k & 1)res = poly_mul(res, f);
            f = poly_mul(f, f);
            k >>= 1;
        }
        return res;
    }

    poly poly_ppow(poly f, int k, int k2)   //多项式快速幂,允许前几项为0,k2是大数幂取模后的结果
    {
        poly g;int m;int invg_0, qg_0;
        for(m = 0; m < f.size(); ++m)
        {
            if(f[m] != 0)
            {
                invg_0 = qpow(f[m], mod - 2);
                qg_0 = qpow(f[m], k2);
                for(int i = m; i < f.size(); ++i)
                {
                    int x = 1ll * f[i] * invg_0 % mod;
                    g.pb(x);
```

```
                }
                break;
            }
        }
        fill(f.begin(), f.end(), 0);
        if(1ll * k * m >= f.size())
        {
            return f;
        }

        g = poly_ln(g, g.size());
        for(auto &x : g) x = 1ll * x * k % mod;
        g = poly_exp(g, g.size());

        for(int i = k * m, j = 0; i < f.size(); ++i, ++j)//注意起点
        {
            f[i] = 1ll * g[j] * qg_0 % mod;
        }
        return f;
    }

    poly operator / (poly A,poly B){
        reverse(A.begin(),A.end());
        reverse(B.begin(),B.end());
        int n = A.size(), m = B.size();
        A.resize(n - m + 1);
        B.resize(n - m + 1);
        B = poly_mul(poly_inv(B, n - m + 1), A);
        B.resize(n - m + 1);
        reverse(B.begin(), B.end());
        return B;
    }
    poly operator % (poly A,poly B){
        int n = A.size(), m = B.size();
        B = poly_mul(B, A / B);
        B.resize(m);
        for(int i = 0; i < m; ++i)B[i] = minus(A[i] , B[i]);//做差取模
        return B;
    }

    poly poly_cos(poly f, int deg) {//多项式三角函数（cos）
        poly A(f.begin(), f.begin() + deg);
        poly B(deg), C(deg);
        for(int i = 0; i < deg; ++ i)
            A[i] = 1ll * A[i] * img % mod;

        B = poly_exp(A, deg);
        C = poly_inv(B, deg);
        int inv2 = qpow(2, mod - 2);
        for(int i = 0; i < deg; ++ i)
            A[i] = 1ll * (1ll * B[i] + C[i]) % mod * inv2 % mod;
        return A;
    }

    poly poly_sin(poly f, int deg) {//多项式三角函数（sin）
        poly A(f.begin(), f.begin() + deg);
        poly B(deg), C(deg);
        for(int i = 0; i < deg; ++ i)
```

```
            A[i] = 1ll * A[i] * img % mod;

        B = poly_exp(A, deg);
        C = poly_inv(B, deg);
        int inv2i = qpow(img << 1, mod - 2);
        for(int i = 0; i < deg; ++ i)
            A[i] = 1ll * (1ll * B[i] - C[i] + mod) % mod * inv2i % mod;
        return A;
    }

    poly poly_arcsin(poly f, int deg) {
        poly A(f.size()), B(f.size()), C(f.size());
        A = poly_dev(f);
        B = poly_mul(f, f);
        for(int i = 0; i < deg; ++ i)
            B[i] = minus(mod, B[i]);
        B[0] = plus(B[0], 1);
        C = poly_sqrt(B, deg);
        C = poly_inv(C, deg);
        C = poly_mul(A, C);
        C = poly_idev(C);
        return C;
    }

    poly poly_arctan(poly f, int deg) {
        poly A(f.size()), B(f.size()), C(f.size());
        A = poly_dev(f);
        B = poly_mul(f, f);
        B[0] = plus(B[0], 1);
        C = poly_inv(B, deg);
        C = poly_mul(A, C);
        C = poly_idev(C);
        return C;
    }
}

using Poly::poly;
using Poly::poly_arcsin;
using Poly::poly_arctan;

int n, m, x, k, type;
poly f, g;
char s[N];

int main()
{
    Poly::init(18);//2^21 = 2,097,152,根据题目数据多项式项数的大小自由调整，注意大小需要跟deer数组同步(21+1=22) 开到比需要大的的最小2的幂*2 如1e6应该是21，1e5可以开18

    read(n), read(type);

    for(int i = 0; i < n; ++ i)
        read(x), f.push_back(x);

    if(type == 0) g = poly_arcsin(f, n);
    else g = poly_arctan(f, n);

    for(int i = 0; i < n; ++ i)
```

```
        printf("%d ", g[i]);
    return 0;
}
```

## 据说飞快的板子，但是长

```
//poly_base (变换基础)NTT以及INTT
namespace poly_base {
    int l, n; u64 iv; vec w2;

    void init(int n = N, bool dont_calc_factorials = true) {
        int i, t;
        for (inv[1] = 1, i = 2; i < n; ++i) inv[i] = u64(mod - mod / i) *
inv[mod % i] % mod;
        if (!dont_calc_factorials) for (*finv = *fact = i = 1; i < n; ++i)
fact[i] = (u64)fact[i - 1] * i % mod, finv[i] = (u64)finv[i - 1] * inv[i] % mod;
        t = min(n > 1 ? lg2(n - 1) : 0, 21),
        *w2 = 1, w2[1 << t] = PowerMod(unity, 1 << (21 - t));
        for (i = t; i; --i) w2[1 << (i - 1)] = (u64)w2[1 << i] * w2[1 << i] %
mod;
        for (i = 1; i < n; ++i) w2[i] = (u64)w2[i & (i - 1)] * w2[i & -i] % mod;
    }

    inline void NTT_init(int len) {n = 1 << (l = len), iv = mod - (mod - 1) /
n;}

    void DIF(int *a) {
        int i, *j, *k, len = n >> 1, R, *o;
        for (i = 0; i < l; ++i, len >>= 1)
            for (j = a, o = w2; j != a + n; j += len << 1, ++o)
                for (k = j; k != j + len; ++k)
                    R = (u64)*o * k[len] % mod, reduce(k[len] = *k - R),
reduce(*k += R - mod);
    }

    void DIT(int *a) {
        int i, *j, *k, len = 1, R, *o;
        for (i = 0; i < l; ++i, len <<= 1)
            for (j = a, o = w2; j != a + n; j += len << 1, ++o)
                for (k = j; k != j + len; ++k)
                    reduce(R = *k + k[len] - mod), k[len] = u64(*k - k[len] +
mod) * *o % mod, *k = R;
    }

    inline void DNTT(int *a) {DIF(a);}
    inline void IDNTT(int *a) {
        DIT(a), std::reverse(a + 1, a + n);
        for (int i = 0; i < n; ++i) a[i] = a[i] * iv % mod;
    }

    inline void DIF(int *a, int *b) {memcpy(b, a, n << 2), DIF(b);}
    inline void DIT(int *a, int *b) {memcpy(b, a, n << 2), DIT(b);}
    inline void DNTT(int *a, int *b) {memcpy(b, a, n << 2), DNTT(b);}
    inline void IDNTT(int *a, int *b) {memcpy(b, a, n << 2), IDNTT(b);}
}
//poly (多项式初等函数)乘除逆
```

```
namespace poly {
    using namespace poly_base;

    vec B1, B2, B3, B4, B5, B6;

    // Multiplication (use one buffer, 3-dft of length 2n)
    void mul(int deg, pvec a, pvec b, pvec c) {
        if (!deg) {*c = (u64)*a * *b % mod; return;}
        NTT_init(lg2(deg) + 1), DNTT(a, c), DNTT(b, B1);
        for (int i = 0; i < n; ++i) c[i] = (u64)c[i] * B1[i] % mod;
        IDNTT(c);
    }

    // Inversion (use three buffers, 5-dft)
    void inv(int deg, pvec a, pvec b) {
        int i, len; assert(*a);
        if (*b = PowerMod(*a, mod - 2), deg <= 1) return;
        memset(b + 1, 0, i = 8 << lg2(deg - 1)), memset(B1, 0, i), *B1 = *a;

        for (len = 0; 1 << len < deg; ++len) {
            NTT_init(len + 1);

            memcpy(B1 + (n >> 1), a + (n >> 1), n << 1), DIF(b, B2), DIF(B1,
B3);
            for (i = 0; i < n; ++i) B3[i] = (u64)B3[i] * B2[i] % mod; DIT(B3);
            for (i = n >> 1; i < n; ++i) B3[i] = B3[n - i] * iv % mod;

            memset(B3, 0, n << 1), DIF(B3);
            for (i = 0; i < n; ++i) B3[i] = (u64)B3[i] * B2[i] % mod; DIT(B3);
            for (i = n >> 1; i < n; ++i) b[i] = B3[n - i] * (mod - iv) % mod;
        }
    }

    // Division and Modulo Operation (use five buffers)
    void div_mod(int A, int B, pvec a, pvec b, pvec q, pvec r) {
        if (A < B) {memcpy(r, a, (A + 1) << 2), memset(r + (A + 1), 0, (B - A)
<< 2); return;}
        int Q = A - B, i, l_ = Q ? lg2(Q) + 1 : 0; NTT_init(l_);
        for (i = 0; i <= Q && i <= B; ++i) B4[i] = b[B - i];
        memset(B4 + i, 0, (n - i) << 2), inv(i = Q + 1, B4, B5);

        std::reverse_copy(a + B, a + (A + 1), B4), NTT_init(++l_),
        memset(B4 + i, 0, (n - i) << 2), memset(B5 + i, 0, (n - i) << 2),
        mul(2 * Q, B4, B5, q), std::reverse(q, q + (Q + 1)),
        memset(q + i, 0, (n - i) << 2);

        if (!B) return;
        NTT_init(lg2(2 * B - 1) + 1);
        for (i = 0; i <= Q && i < B; ++i) B2[i] = b[i], B3[i] = q[i];
        memset(B2 + i, 0, (n - i) << 2), memset(B3 + i, 0, (n - i) << 2),
        mul(2 * (B - 1), B2, B3, r), memset(r + i, 0, (n - i) << 2);
        for (i = 0; i < B; ++i) reduce(r[i] = a[i] - r[i]);
    }

    // Multiplication with std::vector (use two buffers, 3-dft)
    void mul(vector &a, vector &b, vector &ret) {
        int A = a.size() - 1, B = b.size() - 1;
        if (!(A || B)) {ret.EB((u64)a[0] * b[0] % mod); return;}
```

```
        NTT_init(lg2(A + B) + 1),
        memcpy(B1, a.data(), (A + 1) << 2), memset(B1 + (A + 1), 0, (n - A - 1)
<< 2),
        memcpy(B2, b.data(), (B + 1) << 2), memset(B2 + (B + 1), 0, (n - B - 1)
<< 2),
        DNTT(B1), DNTT(B2);
        for (int i = 0; i < n; ++i) B1[i] = (u64)B1[i] * B2[i] % mod;
        IDNTT(B1), ret.assign(B1, B1 + (A + B + 1));
    }

    // Differential
    void diff(int deg, pvec a, pvec b) {for (int i = 1; i <= deg; ++i) b[i - 1]
= (u64)a[i] * i % mod;}

    // Integral
    void intg(int deg, pvec a, pvec b, int constant = 0) {for (int i = deg; i; -
-i) b[i] = (u64)a[i - 1] * ::inv[i] % mod; *b = constant;}

    // f'[x] / f[x] (use four buffers, 6.5-dft)
    void dif_quo(int deg, pvec a, pvec b) {
        assert(*a);
        if (deg <= 1) {*b = PowerMod(*a, mod - 2, a[1]); return;}

        int i, len = lg2(deg - 1);
        inv((deg + 1) / 2, a, B4), NTT_init(len + 1),
        memset(B4 + (n >> 1), 0, n << 1), DIF(B4, B2),

        diff(deg, a, B1), memcpy(B3, B1, n << 1),
        memset(B3 + (n >> 1), 0, n << 1), DIF(B3);

        for (i = 0; i < n; ++i) B3[i] = (u64)B3[i] * B2[i] % mod;
        DIT(B3, b), *b = *b * iv % mod;
        for (i = 1; i < n >> 1; ++i) b[i] = b[n - i] * iv % mod;
        memset(b + (n >> 1), 0, n << 1);

        DIF(b, B4), DIF(a, B3);
        for (i = 0; i < n; ++i) B3[i] = (u64)B3[i] * B4[i] % mod; DIT(B3);
        for (i = n >> 1; i < n; ++i) B3[i] = (B3[n - i] * iv + mod - B1[i]) %
mod;

        memset(B3, 0, n << 1), DIF(B3);
        for (i = 0; i < n; ++i) B3[i] = (u64)B3[i] * B2[i] % mod; DIT(B3);
        for (i = n >> 1; i < n; ++i) b[i] = B3[n - i] * (mod - iv) % mod;
    }

    // Logarithm (use DifQuo)
    inline void ln(int deg, pvec a, pvec b) {assert(*a == 1), --deg ?
(dif_quo(deg, a, b), intg(deg, b, b)) : void(*b = 0);}

    // Exponentiation (use six buffers, 12-dft)
    // WARNING : this implementation of exponentiation is SLOWER than the CDQ_NTT
ver.
    void exp(int deg, pvec a, pvec b) {
        int i, len; pvec c = B6; assert(!*a);
        if (*b = 1, deg <= 1) return;
        if (b[1] = a[1], deg == 2) return;
```

```
        memset(b + 2, 0, i = 8 << lg2(deg - 1)), memset(c, 0, i), memset(B1, 0,
i),
        *c = 1, neg(c[1] = b[1]);

        for (len = 1; 1 << len < deg; ++len) {
            NTT_init(len + 1);

            DIF(c, B2), DIF(b, B3);
            for (i = 0; i < n; ++i) B4[i] = (u64)B3[i] * B2[i] % mod; DIT(B4);
            for (i = n >> 1; i < n; ++i) B4[i] = B4[n - i] * iv % mod;

            memset(B4, 0, n << 1), DIF(B4);
            for (i = 0; i < n; ++i) B4[i] = (u64)B4[i] * B2[i] % mod; DIT(B4);
            for (i = n >> 1; i < n; ++i) B4[i] = B4[n - i] * (mod - iv) % mod;

            memcpy(B4, c, n << 1), DIF(B4);
            diff(n >> 1, b, B1), DIF(B1, B5);
            for (i = 0; i < n; ++i) B4[i] = (u64)B4[i] * B5[i] % mod; DIT(B4);
            for (i = n >> 1; i < n; ++i) reduce(B5[i] = (a[i] + B4[n - i + 1] *
(mod - iv) % mod * ::inv[i]) % mod);

            memset(B5, 0, n << 1), DIF(B5);
            for (i = 0; i < n; ++i) B5[i] = (u64)B5[i] * B3[i] % mod; DIT(B5);
            for (i = n >> 1; i < n; ++i) b[i] = B5[n - i] * iv % mod;

            if (2 << len >= deg) return;

            DIF(b, B3);
            for (i = 0; i < n; ++i) B3[i] = (u64)B3[i] * B2[i] % mod; DIT(B3);
            for (i = n >> 1; i < n; ++i) B3[i] = B3[n - i] * iv % mod;

            memset(B3, 0, n << 1), DIF(B3);
            for (i = 0; i < n; ++i) B3[i] = (u64)B3[i] * B2[i] % mod; DIT(B3);
            for (i = n >> 1; i < n; ++i) c[i] = B3[n - i] * (mod - iv) % mod;
        }
    }
}

//CDQ_NTT (分治多项式技巧)
namespace CDQ_NTT {
    using namespace poly_base;

    int lim;
    vec f, g, C1;
    int fn[N * 2], gn[N * 2];

    inline void register_g(pvec g) {for (int i = 1; 1 << (i - 1) <= lim; ++i)
NTT_init(i), DIF(g, gn + (1 << i));}

    // Standard CDQ-NTT Algorithm, type `UK`
    void solve(int L, int w) {
        int i, R = L + (1 << w), M;
        if (!w) {
            // something depend on problem
            return;
        }
        solve(L, w - 1);
        if ((M = (1 << (w - 1)) + L) > lim) return;
```

```
        NTT_init(w);
        pvec ga = gn + (1 << w);
        memcpy(C1, f + L, 2 << w), memset(C1 + (1 << (w - 1)), 0, 2 << w),
        DIF(C1);
        for (i = 0; i < n; ++i) C1[i] = (u64)C1[i] * ga[i] % mod;
        DIT(C1);
        for (i = M; i < R; ++i) f[i] = (f[i] + C1[n - (i - L)] * iv) % mod;
        solve(M, w - 1);
    }

    void solve(int L, int w) {
        int i, R = L + (1 << w), M;
        if (!w) {
            // something depend on problem
            return;
        }
        solve(L, w - 1);
        if ((M = (1 << (w - 1)) + L) > lim) return;
        NTT_init(w);
        if (L) {
            pvec fa = fn + (1 << w), ga = gn + (1 << w);
            memcpy(C1, f + L, 2 << w), memset(C1 + (1 << (w - 1)), 0, 2 << w),
            memcpy(C2, g + L, 2 << w), memset(C2 + (1 << (w - 1)), 0, 2 << w),
            DIF(C1), DIF(C2);
            for (i = 0; i < n; ++i) C1[i] = ((u64)C1[i] * ga[i] + (u64)C2[i] *
fa[i]) % mod;
            DIT(C1);
            for (i = M; i < R; ++i) f[i] = (f[i] + C1[n - (i - L)] * iv) % mod;
        } else {
            memcpy(C1, f, 2 << w), memset(C1 + M, 0, 2 << w),
            memcpy(C2, g, 2 << w), memset(C2 + M, 0, 2 << w),
            DIF(C1), DIF(C2),
            memcpy(fn + (1 << (w - 1)), C1, 2 << w),
            memcpy(gn + (1 << (w - 1)), C2, 2 << w);
            for (i = 0; i < n; ++i) C1[i] = (u64)C1[i] * C2[i] % mod;
            DIT(C1);
            for (i = M; i < R; ++i) f[i] = (f[i] + C1[n - i] * iv) % mod;
        }
        solve(M, w - 1);
    }
}

//poly_evaluation (多点求值)
namespace poly_evaluation {
    using namespace poly_base;

    int cnt = 0, lc[N], rc[N];
    vec Prd_, E1, E2, E3;
    vector g[N], tmp_;

    int solve(int L, int R) {
        if (L + 1 == R) return L;
        int M = (L + R) / 2, id = cnt++, lp = solve(L, M), rp = solve(M, R);
        return poly::mul(g[lp], g[rp], g[id]), lc[id] = lp, rc[id] = rp, id;
    }

    void recursion(int id, int L, int R, const vector &poly) {
        if (L + 1 == R) return tmp_.EB(poly.back());
```

```
        int i, n = poly.size() - 1, M = (L + R) / 2, lp = lc[id], rp = rc[id],
            dl = min(n, g[lp].size() - 1), dr = min(n, g[rp].size() - 1);
        if (L + 2 == R) return
            tmp_.EB((poly[n] + (u64)poly[n - 1] * g[rp].back()) % mod),
            tmp_.EB((poly[n] + (u64)poly[n - 1] * g[lp].back()) % mod);

        vector ly, ry; ly.reserve(dl + 1), ry.reserve(dr + 1);
        NTT_init(lg2(dl + dr) + 1);
        memcpy(E1, poly.data(), (n + 1) << 2), DIF(E1, E2), memset(E1, 0, (n +
1) << 2);

        memcpy(E1, g[rp].data(), (dr + 1) << 2), DIF(E1, E3), memset(E1, 0, (dr
+ 1) << 2);
        for (i = 0; i < poly::n; ++i) E3[i] = (u64)E3[i] * E2[i] % mod;
        DIT(E3), std::reverse(E3 + 1, E3 + poly::n);
        for (i = n - dl; i <= n; ++i) ly.EB(E3[i] * iv % mod);

        memcpy(E1, g[lp].data(), (dl + 1) << 2), DIF(E1, E3), memset(E1, 0, (dl
+ 1) << 2);
        for (i = 0; i < poly::n; ++i) E3[i] = (u64)E3[i] * E2[i] % mod;
        DIT(E3), std::reverse(E3 + 1, E3 + poly::n);
        for (i = n - dr; i <= n; ++i) ry.EB(E3[i] * iv % mod);

        recursion(lp, L, M, ly), recursion(rp, M, R, ry);
    }

    vector emain(int n, pvec f, const vector &pts) {
        int i, id, m = pts.size(), q;
        if (!m) return vector();
        if (!n) return vector(m, *f);
        for (i = 0; i < m; ++i) g[i].clear(), g[i].EB(1), g[i].EB(neg(q =
pts[i]));

        id = solve(0, cnt = m), memcpy(Prd_, g[id].data(), (m + 1) << 2);
        poly::inv(n + 1, Prd_, E2), memset(Prd_, 0, (m + 1) << 2);
        if (n > 0) memset(E2 + (n + 1), 0, (poly::n - n - 1) << 2);

        std::reverse_copy(f, f + (n + 1), E1), poly::mul(2 * n, E1, E2, E3),
        memset(E1, 0, (n + 1) << 2), memset(E2, 0, (n + 1) << 2);

        return tmp_.clear(), tmp_.reserve(m), recursion(id, 0, m, vector(E3 +
max(n - m + 1, 0), E3 + (n + 1))), tmp_;
    }
}

//poly_interpolation (快速插值)
//linear_recur (常系数线性齐次递推式 - Fiduccia)
//linear_recur_single (常系数线性齐次递推式 - 单点)
//miscellaneous (杂项)
```

# 多项式除法

## 推导

目前我们已知$F(x)$，$G(x)$，要求$Q(x)$，$R(x)$。$R(x)$最高次低于$F(x)$

$$F(x)=Q(x)G(x)+R(x)$$

不妨设$F(x)$最高次数为$n$，$G(x)$最高次数为$m$。

$n=m$时，$Q(x)$是常数

$n<m$时，$Q(x)$恒为0

$n>m$时，$Q(x)$最高次项为$n-m$，$R(x)$的最高次项系数小于$m$。

不妨将$R(x)$系数用0补全到m-1

考虑移除 $R(x)$的影响，将系数反转得到

$$x^nF(\frac{1}{x})=x^{n-m}*Q(\frac{1}{x})*x^m*G(\frac{1}{x})+x^{n-m+1}*x^{m-1}*R(\frac{1}{x})$$

记$F_R(x)$为$F(x)$系数反转后的结果，则有

$$F_R(x)=Q_R(x)*G_R(x)+x^{n-m+1}*R_R(x)$$

可以发现此时对$x^{n-m+1}$取模即可消去$R(x)$项同时$Q(x)$最高次项因为不足$x^{n-m+1}$，模意义下求出来的结果就是答案

即对于$Q(x)$，先求出反转后的结果再求出$R(x)$

## code

```
cin>>n>>m;
bool ck = 0;
for(int i=0;i<=n;++i)
{
    int x;cin>>x;
    f.push_back(x);
}
for(int i=0;i<=m;++i)
{
    int x;cin>>x;
    g.push_back(x);
}
Poly::init(18);
poly fr(n+1);
poly gr(m+1);

for(int i=0;i<=n;++i)fr[i]=f[n-i];
for(int i=0;i<=m;++i)gr[i]=g[m-i];

fr.resize(n-m+1);gr.resize(n-m+1);
gr = Poly::poly_inv(gr,n-m+1);
gr = Poly::poly_mul(gr,fr);

gr.resize(n-m+1);
reverse(gr.begin(),gr.end());

for(int i=0;i<n-m+1;++i)cout<<gr[i]<<' ';
cout<<'\n';
```

```
    gr = Poly::poly_mul(gr,g);
    for(int i=0;i<m;++i)
    {
        int x=(f[i]-gr[i]+mod)%mod;
        cout<<x<<' ';
    }
```

# 多项式优化背包

# 拉格朗日插值

## 公式

已知点对$(x_1,y_1),(x_2,y_2),\dots,(x_n,y_n)$，则有

$$f(x)=\sum_{i=1}^{n}y_i\prod_{j\neq i}\frac{x-x_j}{x_i-x_j}$$

## 任意点对，$O(n^2)$

```
//k是最后要求的F(k)
for(int i = 1; i <= n; ++ i) {
    ll s1 = A[i].y % mod;
    ll s2 = 1ll;
    for(int j = 1; j <= n; ++ j) {
        if(i != j) {
            s1 = s1 * (k - A[j].x) % mod;
            s2 = s2 * (A[i].x - A[j].x) % mod;
        }
    }
    ans += s1 * inv(s2) % mod;
}
```

## 连续点对，$O(n)$

1~n

$$pre[i]=\prod_{j=0}^{i}(k-j)$$

$$suf[i]=\prod_{j=i}^{n}(k-j)$$

$$f(x)=\sum_{i=1}^{n}y_i\frac{pre[i-1]*suf[i+1]}{fact[i-1]*fact[N-i]}((N-i)\%2\ ?\ -1:1)$$

# 斯特林数

# 第一类斯特林数

## 记法

$$s\left(n,m\right)$$

或者

$$\begin{bmatrix}n\\m\end{bmatrix}$$

## 含义

n个元素构成m个圆排列的方案数

# 第二类斯特林数

## 记法

$$S\left(n,m\right)$$

或者

$$\begin{Bmatrix}n\\m\end{Bmatrix}$$

## 含义

将n个不同小球放到m个相同盒子里的方案数。

## 递推公式

考虑第n个小球，有两种放法，第一种放到前n-1个小球所在的前m个盒子里面，由于这些盒子可以由小球区分开来，因此一共有m种放法，第二种自己放到一个单独的盒子里。

$$\begin{Bmatrix}n\\m\end{Bmatrix}=\begin{Bmatrix}n-1\\m-1\end{Bmatrix}+m*\begin{Bmatrix}n-1\\m\end{Bmatrix}$$

边界条件

$$\begin{Bmatrix}0\\0\end{Bmatrix}=1$$

$$\begin{Bmatrix}n\\0\end{Bmatrix}=0$$

## 通项公式

$$\begin{Bmatrix}n\\m\end{Bmatrix}=\frac{1}{m!}\sum_{k=0}^{m}(-1)^k\binom{m}{k}(m-k)^n$$

## 部分斯特林数

| n | 第二类斯特林数 |
|---|---|
| 0 | 1 |
| 1 | 0 1 |
| 2 | 0 1 1 |
| 3 | 0 1 3 1 |
| 4 | 0 1 7 6 1 |
| 5 | 0 1 15 25 10 1 |
| 6 | 0 1 31 90 65 15 1 |
| 7 | 0 1 63 301 350 140 21 1 |
| 8 | 0 1 127 966 1701 1050 266 28 1 |
| 9 | 0 1 255 3025 7770 6951 2646 462 36 1 |
| 10 | 0 1 511 9330 34105 42525 22827 5880 750 45 1 |

## 斯特林数性质

性质1：

$$m^n=\sum_{i=0}^{m}\binom{m}{i}\begin{Bmatrix}n\\i\end{Bmatrix}i!$$

斯特林数也可以表现为

$$\sum_{1\le x_1<x_2<\cdots<x_n\le m}\prod_{i=1}^{n}x_i$$

我们可以这样考虑，现在有$m$个盒子，每个盒子里小球数量表示值为$i$的数有多少个，那么这样对应序列产生的贡献就可以写为$\prod_{i=1}^{m}a_i*i$，现在我们有$n+m$个球（保证序列非空），那么最后一个小球对应的贡献就可以表示为$(\prod_{i=1}^{m-1}a_i*i)*(a_m-1+1)*m$，其中一部分可以理解为$(\prod_{i=1}^{m-1}a_i*i)*m$即（$n+m-1,m$)的情况作为答案在乘以$m$，另一部分即（$n+m-1,m-1$)的情况。

## 快速求（$O(nlogn)$)

## 求一行

给定n，求m= 1~$i$的斯特林数

考虑性质1

$$m^n=\sum_{i=0}^{m}\binom{m}{i}\begin{Bmatrix}n\\i\end{Bmatrix}i!$$

不妨设

$$f(m)=m^n,g(m)=\begin{Bmatrix}n\\m\end{Bmatrix}m!$$

于是有

$$\begin{aligned}f(m)&=\sum_{i=0}^{m}\binom{m}{i}g(i)\\\iff g(m)&=\sum_{i=0}^{m}\binom{m}{i}(-1)^{m-i}f(i)\\&=\sum_{i=0}^{m}\binom{m}{i}(-1)^{m-i}i^n\end{aligned}$$

那么我们可以得到

$$\begin{Bmatrix}n\\m\end{Bmatrix}=\sum_{i=0}^{m}\frac{\binom{m}{i}(-1)^{m-i}i^n}{m!}$$

化简可以得到

$$\begin{Bmatrix}n\\m\end{Bmatrix}=\sum_{i=0}^{m}\frac{(-1)^{m-i}i^n}{i!(m-i)!}$$

于是我们可以利用NTT求了

```
for (int i = 1; i <= n; ++i)
        fact[i] = 1ll * fact[i - 1] * i % mod;
inv[n] = qpow(fact[n] , mod - 2);
for (int i = n - 1; i >= 0; --i)
        inv[i] = 1ll * inv[i + 1] * (i + 1) % mod;
 poly f(n + 1), g(n + 1);
for (int i = 0; i <= n; ++i)
    g[i] = (i & 1 ? mod - 1ll : 1ll) * inv[i] % mod,
           f[i] = 1ll * qpow(i, n) * inv[i] % mod;
Poly::init(21);
f = poly_mul(f, g);
f.resize(n + 1);
```

## 求一列

考虑递推式

$$\begin{Bmatrix}n\\m\end{Bmatrix}=\begin{Bmatrix}n-1\\m-1\end{Bmatrix}+m*\begin{Bmatrix}n-1\\m\end{Bmatrix}$$

「同一列」的第二类斯特林数指的是，有着不同的 $i$，相同的 $k$的一系列 $\begin{Bmatrix}i\\k\end{Bmatrix}$。求出同一列的所有第二类斯特林数，就是对 $i=0..n$求出了将 $i$ 个不同元素划分为 $k$ 个非空集的方案数。

利用指数型生成函数计算。

一个盒子装 $i$ 个物品且盒子非空的方案数是 $[i>0]$。我们可以写出它的指数型生成函数为 $F(x)=\sum\limits_{i=1}^{+\infty}\frac{x^i}{i!}=\mathrm{e}^x-1$。经过之前的学习，我们明白 $F^k(x)$就是$i$个有标号物品放到$k$个有标号盒子里的指数型生成函数，那么除掉 $k!$ 就是 $i$ 个有标号物品放到 $k$个无标号盒子里的指数型生成函数。

$\begin{Bmatrix} i \\ k \end{Bmatrix} = \dfrac{\left[\dfrac{x^i}{i!}\right] F^k(x)}{k!}$，$O(n \log n)$ 计算多项式幂即可。

另外，$\exp F(x) = \sum\limits_{i=0}^{+\infty} \dfrac{F^i(x)}{i!}$就是 $i$ 个有标号物品放到任意多个无标号盒子里的指数型生成函数（$EXP$ 通过每项除以一个 $i!$ 去掉了盒子的标号）。这其实就是贝尔数的生成函数。

这里涉及到很多「有标号」「无标号」的内容，注意辨析。

```
int main() {
  scanf("%d%d", &n, &k);
  poly f(n + 1);
  fact[0] = 1;
  for (int i = 1; i <= n; ++i) fact[i] = (ll)fact[i - 1] * i % mod;
  for (int i = 1; i <= n; ++i) f[i] = qpow(fact[i], mod - 2);
  f = exp(log(f >> 1) * k) << k, f.resize(n + 1);
  int inv = qpow(fact[k], mod - 2);
  for (int i = 0; i <= n; ++i)
    printf("%lld ", (ll)f[i] * fact[i] % mod * inv % mod);
  return 0;
}
```

## 原根

对于模数$m$

它的原根一定满足$g^{\frac{\varphi(m)}{p}} \neq 1(\mod m)$

若$g$是一个原根，那么$(k, \varphi(m)) = 1$，$g^k$也是原根

只有满足$2, 4, p^k, 2 * p^k$，$p$为奇素数，$k$为正整数

设 $m \geq 3, (g, m) = 1$，则 $g$ 是模 $m$ 的原根的充要条件是，对于 $\varphi(m)$ 的每个素因数 $p$，都有 $g^{\frac{\varphi(m)}{p}} \neq 1 \pmod m$.

一个数$m$存在原根，当且仅当$m = 2, 4, p^\alpha, 2p^\alpha$，其中$p$是奇素数。

417934045419982028 9的原根是3

```
#include <iostream>
#include <cstdlib>
#include <cstring>
#include <cstdio>
using namespace std;
int P;
const int NUM = 32170;
int prime[NUM/4];
bool f[NUM];
int pNum = 0;
void getPrime()//线性筛选素数
{
    for (int i = 2; i < NUM; ++ i)
    {
```

```
        if (!f[i])
        {
            f[i] = 1;
            prime[pNum++] = i;
        }
        for (int j = 0; j < pNum && i*prime[j] < NUM; ++ j)
        {
            f[i*prime[j]] = 1;
            if (i%prime[j] == 0)
            {
                break;
            }
        }
    }
}
__int64 getProduct(int a,int b,int P)//快速求次幂mod
{
    __int64 ans = 1;
    __int64 tmp = a;
    while (b)
    {
        if (b&1)
        {
            ans = ans*tmp%P;
        }
        tmp = tmp*tmp%P;
        b>>=1;
    }
    return ans;
}

bool judge(int num)//求num的所有的质因子
{
    int elem[1000];
    int elemNum = 0;
    int k = P - 1;
    for (int i = 0; i < pNum; ++ i)
    {
        bool flag = false;
        while (!(k%prime[i]))
        {
            flag = true;
            k /= prime[i];
        }
        if (flag)
        {
            elem[elemNum ++] = prime[i];
        }
        if (k==1)
        {
            break;
        }
        if (k/prime[i]<prime[i])
        {
            elem[elemNum ++] = prime[i];
            break;
        }
    }
```

```
    bool flag = true;
    for (int i = 0; i < elemNum; ++ i)
    {
        if (getProduct(num,(P-1)/elem[i],P) == 1)
        {
            flag = false;
            break;
        }
    }
    return flag;
}
int main()
{

    getPrime();
    while (cin >> P)
    {
        for (int i = 2;;++i)
        {
            if (judge(i))
            {
                cout << i<< endl;
                break;
            }
        }
    }
    return 0;
}
```

# cdq分治

## 解决和点对有关的问题

这类问题多数类似于「给定一个长度为 n 的序列，统计有一些特性的点对 $(i,j)$的数量/找到一对点$(i,j)$使得一些函数的值最大」。

CDQ 分治解决这类问题的算法流程如下：

1. 找到这个序列的中点 $mid$;
2. 将所有点对$(i,j)$划分为 3 类：
    1. $1 \leq i,j \leq mid$
    2. $mid+1 \leq i,j \leq r$
    3. $1 \leq i \leq mid, mid+1 \leq j \leq r$
3. 将 $(1,n)$这个序列拆成两个序列 $(1,mid)$和 $(mid+1,n)$。此时第一类点对和第三类点对都在这两个序列之中；
4. 递归地处理这两类点对；
5. 设法处理第二类点对。

## 题目

$$f_0 = 1$$

$$f_i = \sum_{j=1}^{i} f_{i-j} g_j$$

考虑如何计算左区间对右区间的贡献即可。

```
void work(poly &g, int l, int r, int n)
{
    if(l == r)
    {
        return ;
    }
    int mid = (l + r) >> 1;

    work(g, l, mid, n);

    poly r1(mid - l + 1, 0), r2(r - l + 1, 0);
    for(int i = l; i <= mid; ++i)
    {
        r1[i - l] = f[i];
    }
    for(int i = 0; i < r - l + 1; ++i)
    {
        r2[i] = g[i];
    }
    poly res = poly_mul(r1, r2);
    for(int i = mid - l + 1; i < min((int)res.size(), r - l + 1); ++i)
    {

        f[i + l] += res[i];
        if(f[i + l] > mod) f[i + l] -= mod;

    }
    work(g, mid + 1, r, n);
}
```

传参数的时候注意$g$传引用值，不然会惨遭$TLE$

# FFT

## 前置知识

### 复数

### 性质

$$\text{性质一：}\quad \omega_{2n}^{2k} = \omega_n^k$$

$$\text{性质二：}\quad \omega_{2n}^{k+\frac{n}{2}} = -\omega_n^k$$

## 用法

多项式乘法，卷积

## 原理

一个n次多项式可以有两种表示方法，点值表示法和系数表示法

在用系数表示法的前提下，即朴素写法，我们需要$\Theta(n^2)$的时间来完成

在点值表示下，可以仅$\Theta(n)$的时间完成

因此我们考虑如何将点值表示与系数表示进行快速转换

不妨记

$$A(x)=\sum_{i=0}^{n-1}a_ix^i$$

在这里，我们可以令高次项$a_i=0$从而使n变大

考虑

$$A(x)=(a_0+a_2x^2+\cdots+a_{n-2}x^{n-2})+x(a_1+a_3x^2+\cdots+a_{n-1}x^{n-1})$$

令

$$A1(x)=(a_0+a_2x+\ldots\ldots a_{n-2}x^{\frac{n-2}{2}})$$
$$A2(x)=(a_1+a_3x+\ldots\ldots a_{n-1}x^{\frac{n-2}{2}})$$

于是有

$$A(X)=A1(x^2)+A2(x^2)$$

带入单位根$(0\le k\le \frac{n}{2}-1)$得

左半部分

$$A(\omega_n^k)=A1(\omega_n^{2k})+\omega_n^k*A2(\omega_{\frac{n}{2}}^k)$$
$$=A1(\omega_{\frac{n}{2}}^k)+\omega_n^k*A2(\omega_n^{2k})(\text{折半引理，性质}1)$$

右半部分

$$A(\omega_n^{k+\frac{n}{2}})=A1(\omega_n^{2k+n})+\omega_n^{k+\frac{n}{2}}*A2(\omega_n^{2k+n})$$

运算得

$$A(\omega_n^{k+\frac{n}{2}})=A1(\omega_{\frac{n}{2}}^k)-\omega_n^k*A2(\omega_{\frac{n}{2}}^k)$$

则问题为求$A1(x)$与$A2(x)$，总体时间复杂度为$T(n)=2T(\frac{n}{2})+O(n)=O(nlogn)$,即实现了FFT转换

## 代码

```
N 应该5倍大小
const db PI=acos(-1);
n = A的最高次数
m = B的最高次数
Lim = 1,  L = 0;
```

```
int R[N];
struct Complex
{
    double x, y;
    Complex (double x = 0, double y = 0) : x(x), y(y) { }
}a[N], b[N];
Complex operator * (Complex J, Complex Q) {
    //模长相乘，幅度相加
    return Complex(J.x * Q.x - J.y * Q.y, J.x * Q.y +           J.y * Q.x);
}
Complex operator - (Complex J, Complex Q) {
    return Complex(J.x - Q.x, J.y - Q.y);
}
Complex operator + (Complex J, Complex Q) {
    return Complex(J.x + Q.x, J.y + Q.y);
}
//for (int i = 0; i <= Lim; ++ i) {
    //换成二进制序列
    //R[i] = (R[i >> 1] >> 1) | ((i & 1) << (L - 1));
    // 在原序列中 i 与 i/2 的关系是 ： i可以看做是i/2的二进制上的每一位左移一位得来
    // 那么在反转后的数组中就需要右移一位，同时特殊处理一下奇数
//}

inline void FFT(Complex *J, double type)//1 转点值，-1 转系数
{
    for(int i = 0; i < Lim; ++ i) {
        if(i < R[i]) swap(J[i], J[R[i]]);
        //i小于R[i]时才交换，防止同一个元素交换两次，回到它原来的位置。
    }
    //从底层往上合并
    for(int mid = 1; mid < Lim; mid <<= 1) {//待合并区间长度的一半，最开始是两个长度为1
的序列合并,mid = 1;
        Complex wn(cos(PI / mid), type * sin(PI / mid));//单位根w_n^i;
        for(int len = mid << 1, pos = 0; pos < Lim; pos += len) {
        //for(int pos = 0; pos < Lim; pos += (mid << 1)) {
            //len是区间的长度，pos是当前的位置,也就是合并到了哪一位
            Complex w(1, 0);//幂,一直乘，得到平方，三次方...
            for(int k = 0; k < mid; ++ k, w = w * wn) {
                //只扫左半部分，得到右半部分的答案,w 为 w_n^k
                Complex x = J[pos + k];//左半部分
                Complex y = w * J[pos + mid + k];//右半部分
                J[pos + k] = x + y;//蝴蝶变换
                J[pos + mid + k] = x - y;
            }
        }
    }
    if(type == 1) return ;
    for(int i = 0; i <= Lim; ++ i)
        a[i].x /= Lim, a[i].y /= Lim;
}
cin>>n>>m;
for(int i=0;i<=n;++i)cin>>a[i].x;
for(int j=0;j<=m;++j)cin>>a[j].y;
while(Lim < n + m) Lim <<= 1, L ++ ;
for (int i = 0; i < Lim; ++ i) {
    //换成二进制序列
    R[i] = (R[i >> 1] >> 1) | ((i & 1) << (L - 1));
```

```
}
FFT(a, 1);
for (int i = 0; i <= Lim; ++ i)
    //对应项相乘，O(n)得到点值表示的多项式的解C，利用逆变换完成插值得到答案C的点值表示
        a[i] = a[i] * a[i];
FFT(a, -1);
```

# FWT

## 前置

我们定义 $\times$ 是多项式对应系数相乘，* 作为多项式卷积

其中 $\bigoplus$是二元位运算符，即 与$(\&)$，或$(|)$，异或$(\wedge)$

我们记对数组 $A$ 进行快速沃尔什变换后得到的结果为$FWT[A]$ 。

那么 FWT 核心思想就是：

我们需要一个新序列$C$，由序列$A$ 和序列$B$经过某运算规则得到，即 $C = A \cdot B$;

我们先正向得到 $FWT[A], FWT[B]]$，再根据 $FWT[C] = FWT[A] \times FWT[B]$在 $O(n)$ 的时间复杂度内求出 $FWT[C]$;

然后逆向运算得到原序列 $C$。时间复杂度为 $O(n \log n)$

## 用途

用于下标位运算卷积问题

## 或运算

注意到，

$$i|k = k, j|k = k \Rightarrow (i|j)|k = k$$

于是构造如下式子

$$FWT[A](i) = A' = \sum_{i=i|j} A_j$$

### 性质

$$1.\ FWT[A + B] = FWT[A] + FWT[B]$$

$FWT$是$A$序列的一个线性组合

$$2.\ FWT[A] = \begin{cases} merge(FWT[A_0], FWT[A_0] + FWT[A_1]) \ ， \ n > 1 \\ A \ ， \ n = 0 \end{cases}$$

$A_0$是$A$的左半部分，$A_1$是$A$的右半部分。对于$A_0$中的任意下标$i$，都可以在$A_1$中找到在二进制下除最高位以外都相同的下标$j$。

由于最高位不同，右半部分不会对最半部分产生贡献，$FWT[A]_0 = FWT[A_0]$

同理可得，$FWT[A]_1 = FWT[A_0] + FWT[A_1]$

$$3.\ FWT[A|B] = FWT[A] \times FWT[B]$$

展开即可证明

**逆运算**

$$IFWT[A] = \begin{cases} merge(IFWT[A_0], IFWT[A_1] - IFWT[A_0]) \ , & n > 1 \\ A \ , & n = 0 \end{cases}$$

## 与运算

同或运算

## 异或运算

fwt式子略有不同

$$FWT[A](i) = \sum_{d(i\&j) \ mod 2 \ \equiv \ 0} A_j - \sum_{d(i\&j) \ mod 2 \ \equiv \ 1} A_j$$

其中$d(x)$表示$x$在二进制下的1的个数

性质与或运算相同

## Code

**数组版本**

```
inline void in() //防止污染原数组
{
    for(int i = 0; i < n; ++ i)
        a[i] = A[i], b[i] = B[i];
}

inline void get()
{
    for(int i = 0; i < n; ++ i)
        a[i] = a[i] * b[i] % mod;
}

inline void out()
{
    for(int i = 0; i < n; ++ i)
        printf("%lld%s", (a[i] % mod + mod) % mod, i == (n - 1) ? "\n" : " ");
}


inline void OR(ll *f, int x = 1)//前半部分 f[i + j], 后半部分 f[i + j + k]
{
    for(int o = 2; o <= n; o <<= 1)
        for(int i = 0, k = o >> 1; i < n; i += o)
            for(int j = 0; j < k; ++ j)
                f[i + j + k] = (f[i + j] * x + f[i + j + k] + (x == 1 ? 0 :
mod)) % mod;

}

inline void AND(ll *f, int x = 1)//前半部分 f[i + j],后半部分 f[i + j + k]
{
    for(int o = 2; o <= n; o <<= 1)
        for(int i = 0, k = o >> 1; i < n; i += o)
            for(int j = 0; j < k; ++ j)
```

```
                f[i + j] = (f[i + j] + f[i + j + k] * x + (x == 1 ? 0 : mod)) %
mod;

}

inline void XOR(ll *f, int x = 1)//前半部分 f[i + j],后半部分 f[i + j + k]
{
    for(int o = 2; o <= n; o <<= 1)
        for(int i = 0, k = o >> 1; i < n; i += o)
            for(int j = 0; j < k; ++ j) {
                int X = f[i + j], Y = f[i + j + k];
                f[i + j] = (X + Y) % mod;
                f[i + j + k] = (X - Y % mod + mod) % mod;
                if(x != 1) {
                    f[i + j] = f[i + j] * inv2 % mod;
                    f[i + j + k] = f[i + j + k] * inv2 % mod;
                }
            }
}

int main()
{
    scanf("%d", &m);
    n = 1 << m;
    for(int i = 0; i < n; ++ i)
        scanf("%lld", &A[i]);
    for(int i = 0; i < n; ++ i)
        scanf("%lld", &B[i]);
    in(), OR(a), OR(b), get(), OR(a, -1), out();
    in(), AND(a), AND(b), get(), AND(a, -1), out();
    in(), XOR(a), XOR(b), get(), XOR(a, -1), out();
    return 0;
}
```

**Vector版本**

```
struct FWT {
    void add(int &x, int y) {
        (x += y) >= P && (x -= P);
    }
    void sub(int &x, int y) {
        (x -= y) < 0 && (x += P);
    }

    int extend(int n) {
        int N = 1;
        for (; N < n; N <<= 1);
        return N;
    }
    void FWTor( vector<int> &a, bool rev) {
        int n = a.size();
        for (int l = 2, m = 1; l <= n; l <<= 1, m <<= 1) {
            for (int j = 0; j < n; j += l) for (int i = 0; i < m; i++) {
                if (!rev) add(a[i + j + m], a[i + j]);
                else sub(a[i + j + m], a[i + j]);
```

```
            }
        }
    }
    void FWTand( vector<int> &a, bool rev) {
        int n = a.size();
        for (int l = 2, m = 1; l <= n; l <<= 1, m <<= 1) {
            for (int j = 0; j < n; j += l) for (int i = 0; i < m; i++) {
                if (!rev) add(a[i + j], a[i + j + m]);
                else sub(a[i + j], a[i + j + m]);
            }
        }
    }
    void FWTxor( vector<int> &a, bool rev) {
        int n = a.size(), inv2 = (P + 1) >> 1;
        for (int l = 2, m = 1; l <= n; l <<= 1, m <<= 1) {
            for (int j = 0; j < n; j += l) for (int i = 0; i < m; i++) {
                int x = a[i + j], y = a[i + j + m];
                if (!rev) {
                    a[i + j] = (x + y) % P;
                    a[i + j + m] = (x - y + P) % P;
                } else {
                    a[i + j] = 1LL * (x + y) * inv2 % P;
                    a[i + j + m] = 1LL * (x - y + P) * inv2 % P;
                }
            }
        }
    }
    vector<int> Or(vector<int> a1, vector<int> a2) {
        int n = max(a1.size(), a2.size()), N = extend(n);
        a1.resize(N), FWTor(a1, false);
        a2.resize(N), FWTor(a2, false);
        vector<int> A(N);
        for (int i = 0; i < N; i++) A[i] = 1LL * a1[i] * a2[i] % P;
        FWTor(A, true);
        return A;
    }
    vector<int> And(vector<int> a1, vector<int> a2) {
        int n =  max(a1.size(), a2.size()), N = extend(n);
        a1.resize(N), FWTand(a1, false);
        a2.resize(N), FWTand(a2, false);
        vector<int> A(N);
        for (int i = 0; i < N; i++) A[i] = 1LL * a1[i] * a2[i] % P;
        FWTand(A, true);
        return A;
    }
     vector<int> Xor(vector<int> a1, vector<int> a2) {
        int n = max(a1.size(), a2.size()), N = extend(n);
        a1.resize(N), FWTxor(a1, false);
        a2.resize(N), FWTxor(a2, false);
        vector<int> A(N);
        for (int i = 0; i < N; i++) A[i] = 1LL * a1[i] * a2[i] % P;
        FWTxor(A, true);
        return A;
    }
} fwt;

//main部分
    scanf("%d", &m);
```

```
    n = 1 << m;
    vector<int> a1(n), a2(n);
    for (int i = 0; i < n; i++) scanf("%d", &a1[i]);
    for (int i = 0; i < n; i++) scanf("%d", &a2[i]);
    vector<int> A;
    A = fwt.Or(a1, a2);
    for (int i = 0; i < n; i++) {
        printf("%d%c", A[i], " \n"[i == n - 1]);
    }
    A = fwt.And(a1, a2);
    for (int i = 0; i < n; i++) {
        printf("%d%c", A[i], " \n"[i == n - 1]);
    }
    A = fwt.Xor(a1, a2);
    for (int i = 0; i < n; i++) {
        printf("%d%c", A[i], " \n"[i == n - 1]);
    }
```

## MTT（任意模数）

### FFT

```
//luoguP4245 【模板】MTT
#include<cstdio>
#include<cmath>
#include<algorithm>
const int N = 4e5 + 10, M = 32767;
const double pi = acos(-1.0);
typedef long long LL;

int read() {
    char ch = getchar(); int f = 1, x = 0;
    for(;ch < '0' || ch > '9'; ch = getchar()) if(ch == '-') f = -1;
    for(;ch >= '0' && ch <= '9'; ch = getchar()) x = (x << 1) + (x << 3) - '0' +
ch;
    return x * f;
}
struct cp {
    double r, i;
    cp(double _r = 0, double _i = 0) : r(_r), i(_i) {}
    cp operator * (const cp &a) {return cp(r * a.r - i * a.i, r * a.i + i *
a.r);}
    cp operator + (const cp &a) {return cp(r + a.r, i + a.i);}
    cp operator - (const cp &a) {return cp(r - a.r, i - a.i);}
}w[N], nw[N], da[N], db[N];

cp conj(cp a) {return cp(a.r, -a.i);}

int L, n, m, a[N], b[N], c[N], R[N], P=1e9+7;

void Pre() {
    int x = 0; for(L = 1; (L <<= 1) <= n + m; ++x) ;
    for(int i = 1;i < L; ++i) R[i] = (R[i >> 1] >> 1) | (i & 1) << x;
    for(int i = 0;i < L; ++i) w[i] = cp(cos(2 * pi * i / L), sin(2 * pi * i /
L));
```

```
}

void FFT(cp *F) {
    for(int i = 0;i < L; ++i) if(i < R[i]) std::swap(F[i], F[R[i]]);
    for(int i = 2, d = L >> 1;i <= L; i <<= 1, d >>= 1)
        for(int j = 0;j < L; j += i) {
            cp *l = F + j, *r = F + j + (i >> 1), *p = w, tp;
            for(int k = 0;k < (i >> 1); ++k, ++l, ++r, p += d)
                tp = *r * *p, *r = *l - tp, *l = *l + tp;
        }
}
void Mul(int *A, int *B, int *C) {
    for(int i = 0;i < L; ++i) (A[i] += P) %= P, (B[i] += P) %= P;
    static cp a[N], b[N], Da[N], Db[N], Dc[N], Dd[N];
    for(int i = 0;i < L; ++i) a[i] = cp(A[i] & M, A[i] >> 15);
    for(int i = 0;i < L; ++i) b[i] = cp(B[i] & M, B[i] >> 15);
    FFT(a); FFT(b);
    for(int i = 0;i < L; ++i) {
        int j = (L - i) & (L - 1); static cp da, db, dc, dd;
        da = (a[i] + conj(a[j])) * cp(0.5, 0);
        db = (a[i] - conj(a[j])) * cp(0, -0.5);
        dc = (b[i] + conj(b[j])) * cp(0.5, 0);
        dd = (b[i] - conj(b[j])) * cp(0, -0.5);
        Da[j] = da * dc; Db[j] = da * dd; Dc[j] = db * dc; Dd[j] = db * dd; //顺
便区间反转，方便等会直接用DFT代替IDFT
    }
    for(int i = 0;i < L; ++i) a[i] = Da[i] + Db[i] * cp(0, 1);
    for(int i = 0;i < L; ++i) b[i] = Dc[i] + Dd[i] * cp(0, 1);
    FFT(a); FFT(b);
    for(int i = 0;i < L; ++i) {
        int da = (LL) (a[i].r / L + 0.5) % P; //直接取实部和虚部
        int db = (LL) (a[i].i / L + 0.5) % P;
        int dc = (LL) (b[i].r / L + 0.5) % P;
        int dd = (LL) (b[i].i / L + 0.5) % P;
        C[i] = (da + ((LL)(db + dc) << 15) + ((LL)dd << 30)) % P;
    }
}
int main() {
    n = read(); m = read();
    P = read();
    for(int i = 0;i <= n; ++i) a[i] = read();
    for(int j = 0;j <= m; ++j) b[j] = read();
    Pre(); Mul(a, b, c);
    for(int i = 0;i <= n + m; ++i) printf("%d ", (c[i] + P) % P); puts("");
    return 0;
}
```

## 三模数

```
#include <algorithm>
#include <cstdio>
#include <cstring>
int mod;
int qpow(int base, int p, const int mod)
{
    int res;
```

```
    for (res = 1; p; p >>= 1, base = static_cast<long long> (base) * base % mod)
if (p & 1) res = static_cast<long long> (res) * base % mod;
    return res;
}
int inv(int x, const int mod) { return qpow(x, mod - 2, mod); }


const int mod1 = 998244353, mod2 = 1004535809, mod3 = 469762049, G = 3;

const long long mod_1_2 = 1ll * mod1 * mod2;

const int inv_1 = inv(mod1, mod2), inv_2 = inv(mod_1_2 % mod3, mod3);

struct Int {
    int A, B, C;
    Int() { }
    Int(int __num) : A(__num), B(__num), C(__num) { }
    Int(int __A, int __B, int __C) : A(__A), B(__B), C(__C) { }

    static inline Int reduce(const Int &x) {
        return Int(x.A + (x.A >> 31 & mod1), x.B + (x.B >> 31 & mod2), x.C +
(x.C >> 31 & mod3));
    }
    inline friend Int operator + (const Int &lhs, const Int &rhs) {
        return reduce(Int(lhs.A + rhs.A - mod1, lhs.B + rhs.B - mod2, lhs.C +
rhs.C - mod3));
    }
    inline friend Int operator - (const Int &lhs, const Int &rhs) {
        return reduce(Int(lhs.A - rhs.A, lhs.B - rhs.B, lhs.C - rhs.C));
    }
    inline friend Int operator * (const Int &lhs, const Int &rhs) {
        return Int(static_cast<long long> (lhs.A) * rhs.A % mod1,
static_cast<long long> (lhs.B) * rhs.B % mod2, static_cast<long long> (lhs.C) *
rhs.C % mod3);
    }

    inline int get() {
        long long x = 1ll * (B - A + mod2) % mod2 * inv_1 % mod2 * mod1 + A;
        return (1ll * (C - x % mod3 + mod3) % mod3 * inv_2 % mod3 * (mod_1_2 %
mod) % mod + x) % mod;
    }
} ;

#define maxn 131072

namespace Poly {
#define N (maxn << 1)
    int lim, s, rev[N];
    Int Wn[N | 1];
    inline void init(int n) {
        s = -1, lim = 1; while (lim < n) lim <<= 1, ++s;
        for (register int i = 1; i < lim; ++i) rev[i] = rev[i >> 1] >> 1 | (i &
1) << s;
        const Int t(qpow(G, (mod1 - 1) / lim, mod1), qpow(G, (mod2 - 1) / lim,
mod2), qpow(G, (mod3 - 1) / lim, mod3));
        *Wn = Int(1); for (register Int *i = Wn; i != Wn + lim; ++i) *(i + 1) =
*i * t;
    }
```

```
    inline void NTT(Int *A, const int op = 1) {
        for (register int i = 1; i < lim; ++i) if (i < rev[i]) std::swap(A[i],
A[rev[i]]);
        for (register int mid = 1; mid < lim; mid <<= 1) {
            const int t = lim / mid >> 1;
            for (register int i = 0; i < lim; i += mid << 1) {
                for (register int j = 0; j < mid; ++j) {
                    const Int W = op ? Wn[t * j] : Wn[lim - t * j];
                    const Int X = A[i + j], Y = A[i + j + mid] * W;
                    A[i + j] = X + Y, A[i + j + mid] = X - Y;
                }
            }
        }
        if (!op) {
            const Int ilim(inv(lim, mod1), inv(lim, mod2), inv(lim, mod3));
            for (register Int *i = A; i != A + lim; ++i) *i = (*i) * ilim;
        }
    }
#undef N
}

int n, m;
Int A[maxn << 1], B[maxn << 1];
int main() {
    scanf("%d%d%d", &n, &m, &mod); ++n, ++m;
    for (int i = 0, x; i < n; ++i) scanf("%d", &x), A[i] = Int(x % mod);
    for (int i = 0, x; i < m; ++i) scanf("%d", &x), B[i] = Int(x % mod);

    Poly::init(n + m);
    Poly::NTT(A), Poly::NTT(B);

    for (int i = 0; i < Poly::lim; ++i) A[i] = A[i] * B[i];

    Poly::NTT(A, 0);

    for (int i = 0; i < n + m - 1; ++i) {
        printf("%d", A[i].get());
        putchar(i == n + m - 2 ? '\n' : ' ');
    }
    return 0;
}
```

# 二项式反演

若

$$f(n)=\sum_{i=0}^{n}\binom{n}{i}g(i)$$

那么

$$g(n)=\sum_{i=0}^{n}(-1)^{n-i}\binom{n}{i}f(i)$$

# 方程

# 扩展中国剩余定理

考虑方程组

$$x \equiv a_1(\mod m_1)$$
$$x \equiv a_2(\mod m_2)$$

对于第一个方程，解的形式是$x = a_1 + k * m_1$。带入第二个方程得到
$a_1 + k * m_1 = a_2(\mod m_2)$。用扩欧可以接出来一个$k_0$，那么我们就合并得到一个方程
$x \equiv a_1 + k_0 * m_1(\mod lcm(m_1, m_2))$

```
    ll mul(ll a, ll b, ll c)
    {
        if(b < 0)a = -a, b = -b;
        ll res = 0;
        while(b)
        {
            if(b & 1)res = (res + a) % c;
            a = (a + a) % c;
            b >>= 1;
        }
        return res;
    }
vector<ll>ai,bi;//x mod b = a;
    ll excrt()
    {
        int n = a.size();
        ll x, y, k;
        ll M = b[0], ans = a[0];//第一个方程的特解
        for(int i = 1; i < n; ++i)
        {
            ll a = M, b = bi[i], c = (ai[i] - ans % b + b) % b;
            ll d = exgcd(a, b, x, y);
            ll bg = b / d;  //lcm
            if(c % d != 0)return -1;    //判断无解
            x = mul(x, c / d, bg);
            ans += x * M;   //更新前k个方程的答案
            M *= bg;
            ans = (ans % M + M) % M;
        }
        ans = (ans % M + M) % M;
        //if(ans == 0)ans = M; //看情况，可能0是符合题意的也可能不是
        return ans;
    }
```

```
struct Congruence
{
    ll exgcd(ll a, ll b, ll &x, ll &y)  //扩欧
    {
        if(b == 0){x = 1; y = 0; return a;}
        ll d = exgcd(b, a % b, x, y);
        ll z = x; x = y; y = z - a / b * y;
        return d;
    }

    ll mul(ll a, ll b, ll c)
```

```
    {
        if(b < 0)a = -a, b = -b;
        ll res = 0;
        while(b)
        {
            if(b & 1)res = (res + a) % c;
            a = (a + a) % c;
            b >>= 1;
        }
        return res;
    }


    vector<ll>ai,bi;//x mod b = a;
    ll excrt()  //扩展中国剩余定理
    {
        int n = ai.size();
        ll x, y, k;
        ll M = bi[0], ans = ai[0];//第一个方程的特解
        for(int i = 1; i < n; ++i)
        {
            ll a = M, b = bi[i], c = (ai[i] - ans % b + b) % b;
            ll d = exgcd(a, b, x, y);
            ll bg = b / d;  //lcm
            if(c % d != 0)return -1;    //判断无解
            x = mul(x, c / d, bg);
            ans += x * M;   //更新前k个方程的答案
            M *= bg;
            ans = (ans % M + M) % M;
        }
        ans = (ans % M + M) % M;
        // if(ans == 0)ans = M; //看情况，可能0是符合题意的也可能不是
        return ans;
    }
}t;
```

## 普通方程组

```
//0~n-1 -1表示无解，0表示无穷多解，1表示有唯一解
int Gauss(int n, vector<vector<double> > a, vector<double> b, vector<double> &x)
//a是系数矩阵，b是等号右边的常数，x是返回的答案
{
    x.resize(n, 0);
    int line = 0;
    for(int i = 0; i < n; ++i)
    {
        int p = line;
        for(int k = line + 1; k < n; ++k) if(fabs(a[k][i]) > fabs(a[p][i]))p =
k;
        if(line != p) swap(a[p], a[line]), swap(b[line], b[p]);
        if(fabs(a[line][i]) < eps) continue;
        for(int k = 0; k < n; ++k)
        {
            if(k == line) continue;
            double d = a[k][i] / a[line][i];
```

```
            b[k] -= d * b[line];
            for(int j = i; j < n; ++j) a[k][j] -= d * a[line][j];
        }
        line++;
    }
    if(line < n)
    {
        while(line < n)
            if(fabs(b[line++]) > eps) return -1;
        return 0;
    }
    for(int i = n - 1; i >= 0; --i)
    {

        // for(int j = i + 1; j < n; ++j) b[i] -= x[j] * a[i][j];
        x[i] = b[i] / a[i][i];

    }
    return 1;
}
```

```
bool Gauss(int n, vector<vector<double> > a, vector<double> b, vector<double>
&x) //a是系数矩阵，b是等号右边的常数，x是返回的答案
{
    x.resize(n, 0);
    for(int i = 0; i < n; ++i)
    {
        int p = i;
        for(int k = i + 1; k < n; ++k) if(fabs(a[k][i]) > fabs(a[p][i]))p = k;
        if(i != p) swap(a[i], a[p]), swap(b[i], b[p]);
        if(a[i][i] == 0) return 0;
        for(int k = i + 1; k < n; ++k)
        {
            double d = a[k][i] / a[i][i];
            b[k] -= d * b[i];
            for(int j = i; j < n; ++j) a[k][j] -= d * a[i][j];
        }
    }
    for(int i = n - 1; i >= 0; --i)
    {

        for(int j = i + 1; j < n; ++j) b[i] -= x[j] * a[i][j];
        x[i] = b[i] / a[i][i];

    }
    return 1;
}
```

## 异或方程组

```
int gauss(int n)
{
    int row,col;
    for(row=1,col=1;col<=n;++col){
        int t = -1;
        for(int i=row;i<=n;++i){
```

```
            if(v[i][col]){
                t = i;
                break;
            }
        }
        if(t == -1) continue;
        if(t != row){
            for(int j=row;j<=n;j++) swap(v[row][j],v[t][j]);
        }
        for(int i=row+1;i<=n;i++){
            if(v[i][col]){
                for(int j=col;j<=n;j++)
                    v[i][j] ^= v[row][j];
            }
        }
        row++;
    }
    return col-row;
}
```

## 高次同余方程

```
ll BSGS(ll a, ll b, ll P) //a^x=b(mod p) return x;
{
    map<ll, ll>mp;
    ll ans = 0, m = ceil(sqrt(P)) + 1, tmp = 1ll;
    for(ll i = 1; i <= m; ++i)
    {
        tmp = tmp * a % P;
        mp[tmp * b % P] = i;

    }
    ll res = tmp;
    for(ll i = 1; i <= m; ++i)
    {
        if(mp[res])
        {
            ans = m * i - mp[res];
            return ans;
        }
        res = res * tmp % P;
    }
    return -1;
}
```

## exBSGS/exgcd

```
ll exgcd(ll a, ll b, ll &x, ll &y)
{
    if(!b)
    {
        x = 1;
        y = 0;
        return a;
    }
    ll d = exgcd(b, a % b, x, y);
```

```
    ll z = x; x = y; y = z - y * (a / b);
    return d;
}

using namespace std;

ll BSGS(ll a, ll b, ll P) //a^x=b(mod p) return x;
{
    b %= P, a %= P;
    map<ll, ll>mp;
    ll m = ceil(sqrt(P)) + 1, tmp = 1ll;
    for(ll i = 1; i <= m; ++i)
    {
        tmp = tmp * a % P;
        mp[tmp * b % P] = i;

    }
    ll res = tmp;
    for(ll i = 1; i <= m; ++i)
    {
        if(mp.find(res) != mp.end())
        {
            return (m * i - mp[res] + P) % P;
        }
        res = res * tmp % P;
    }
    return -1;
}
ll inv(ll a, ll b)
{
    ll x, y;
    exgcd(a, b, x, y);
    return (x % b + b) % b;
}
ll exBSGS(ll a, ll b, ll P) //a^x=b(mod p) return x;
{
    a %= P;
    b %= P;
    if(b == 1 || P == 1)
    {
        return 0;
    }
    ll g = gcd(a, P), k = 0, na = 1;
    while(g > 1)
    {

        if(b % g) return -1;
        k++; b /= g; P /= g; na = na * (a / g) % P;
        if(na == b) return k;
        g = gcd(a, P);
    }
    ll x, y;
    exgcd(na, P, x, y);
    x = (x % P + P) % P;
    ll f = BSGS(a, b * x % P, P);
    if(f == -1)return -1;
    return f + k;
}
```

奇怪的求，似乎快很多$p \leq 10^{18}$ 且 $p$不怎么毒瘤能够通过

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

// 2023 OneWan
long long qpow(long long a, long long b) {
    long long res = 1;
    while (b) {
        if (b & 1) res = res * a;
        a = a * a;
        b >>= 1;
    }
    return res;
}
long long qpow(long long a, long long b, long long mod) {
    long long res = 1;
    while (b) {
        if (b & 1) {
            res = (__int128) res * a % mod;
        }
        b >>= 1;
        a = (__int128) a * a % mod;
    }
    return res;
}
template<class T> struct Random {
    mt19937 mt;
    Random() : mt(chrono::steady_clock::now().time_since_epoch().count()) {}
    T operator()(T L, T R) {
        uniform_int_distribution<int64_t> dist(L, R);
        return dist(mt);
    }
};
Random<long long> rng;
namespace Miller_Rabin {
    bool Miller_Rabin(const long long& n, const vector<long long>& as) {
        long long d = n - 1;
        while (!(d & 1)) {
            d >>= 1;
        }
        long long e = 1, rev = n - 1;
        for (auto& a : as) {
            if (n <= a) {
                break;
            }
            long long t = d;
            long long y = qpow(a, t, n);
            while (t != n - 1 && y != e && y != rev) {
                y = (__int128) y * y % n;
                t <<= 1;
            }
            if (y != rev && (!(t & 1))) return false;
        }
        return true;
    }
```

```cpp
    bool is_prime(const long long& n) {
        if (!(n & 1)) {
            return n == 2;
        }
        if (n <= 1) {
            return false;
        }
        if (n < (1LL << 30)) {
            return Miller_Rabin(n, {2, 7, 61});
        }
        return Miller_Rabin(n, {2, 325, 9375, 28178, 450775, 9780504,
1795265022});
    }
} // Miller_Rabin
namespace Pollard_rho {
    long long solve(long long n) {
        if (!(n & 1)) {
            return 2;
        }
        if (Miller_Rabin::is_prime(n)) {
            return n;
        }
        long long R, one = 1;
        auto f = [&](long long x) {
            return ((__int128) x * x % n + R) % n;
        };
        auto rnd = [&]() {
            return rng(0, n - 3) + 2;
        };
        while (true) {
            long long x, y, ys, q = one;
            R = rnd(), y = rnd();
            long long g = 1;
            int m = 128;
            for (int r = 1 ; g == 1 ; r <<= 1) {
                x = y;
                for (int i = 0 ; i < r ; i++) {
                    y = f(y);
                }
                for (int k = 0 ; g == 1 && k < r ; k += m) {
                    ys = y;
                    for (int i = 0 ; i < m && i < r - k ; i++) {
                        q = (__int128) q * ((x - (y = f(y)) + n) % n) % n;
                    }
                    g = __gcd(q, n);
                }
            }
            if (g == n) {
                do {
                    g = __gcd((x - (ys = f(ys)) + n) % n, n);
                } while (g == 1);
            }
            if (g != n) {
                return g;
            }
        }
        return 0;
    }
```

```cpp
    vector<long long> factorize(long long n) {
        if (n <= 1) return {};
        long long p = solve(n);
        if (p == n) return {n};
        auto L = factorize(p);
        auto R = factorize(n / p);
        copy(R.begin(), R.end(), back_inserter(L));
        return L;
    }
    vector<pair<long long, int>> prime_factor(long long n) {
        auto ps = factorize(n);
        sort(ps.begin(), ps.end());
        vector<pair<long long, int>> ret;
        for (auto &e : ps) {
            if (!ret.empty() && ret.back().first == e) {
                ret.back().second++;
            } else {
                ret.emplace_back(e, 1);
            }
        }
        return ret;
    }
    vector<long long> divisors(long long n) {
        auto ps = prime_factor(n);
        int cnt = 1;
        for (auto& [p, t] : ps) cnt *= t + 1;
        vector<long long> ret(cnt, 1);
        cnt = 1;
        for (auto& [p, t] : ps) {
            long long pw = 1;
            for (int i = 1; i <= t; i++) {
                pw *= p;
                for (int j = 0; j < cnt; j++) ret[cnt * i + j] = ret[j] * pw;
            }
            cnt *= t + 1;
        }
        return ret;
    }
} // Pollard_rho
namespace Pohlig_Hellman {
    long long BSGS(long long A, long long B, long long P, long long mod) {
        A %= mod;
        B %= mod;
        if (B == 1) {
            return 0;
        }
        if (A == 0) {
            if (B == 0) {
                return 1;
            }
            return -1;
        }
        long long t = 1;
        int m = sqrt(1.0 * P) + 1;
        long long base = B;
        unordered_map<long long, long long> vis;
        for (int i = 0 ; i < m ; i++) {
            vis[base] = i;
```

```
            base = (__int128) base * A % mod;
        }
        base = qpow(A, m, mod);
        long long now = 1;
        for (int i = 1 ; i <= m ; i++) {
            now = (__int128) now * base % mod;
            auto k = vis.find(now);
            if (k != vis.end()) {
                return i * m - k -> second;
            }
        }
        return -1;
    }
    long long getK(long long A, long long B, long long P, long long C, long long
phi, long long mod) {
        vector<long long> pi;
        long long temp = 1;
        for (int i = 0 ; i <= C ; i++) {
            pi.emplace_back(temp);
            temp *= P;
        }
        long long k = qpow(A, pi[C - 1], mod);
        long long inv = 0;
        temp = 1;
        for (int i = C - 1 ; i >= 0 ; i--) {
            long long tp = qpow(A, pi[C] - inv, mod);
            long long tx = temp * BSGS(k, qpow((__int128)B * tp % mod, pi[i],
mod), P, mod);
            inv += tx;
            temp *= P;
        }
        return inv;
    }
    int getOrg(long long P, long long phi, const vector<pair<long long, int>>&
res) {
        for (int k = 2 ; ; k++) {
            bool flag = true;
            for (auto& [x, y] : res) {
                if (qpow(k, phi / x, P) == 1LL) {
                    flag = false;
                    break;
                }
            }
            if (flag) return k;
        }
    }
    void Exgcd(long long a, long long b, long long& x, long long& y) {
        if (b == 0) {
            x = 1;
            y = 0;
            return;
        }
        Exgcd(b, a % b, y, x);
        y -= a / b * x;
    }
    long long CRT(const vector<long long>& k, const vector<pair<long long,
int>>& res) {
        int len = res.size();
```

```
        long long M = 1, ans = 0;
        vector<long long> m(len);
        for (int i = 0 ; i < len ; i++) {
            m[i] = qpow(res[i].first, res[i].second);
            M *= m[i];
        }
        for (int i = 0 ; i < len ; i++) {
            long long Mi = M / m[i];
            long long x, y;
            Exgcd(Mi, m[i], x, y);
            ans = (ans + (__int128)Mi * ((x % m[i] + m[i]) % m[i]) * k[i]) % M;
        }
        if (ans < 0) ans += M;
        return ans;
    }
    long long getX(long long B, long long A, long long phi, long long mod,
vector<pair<long long, int>>& res) {
        vector<long long> k;
        for (auto& [x, y] : res) {
            long long z = qpow(x, y);
            long long tA = qpow(A, phi / z, mod);
            long long tB = qpow(B, phi / z, mod);
            k.emplace_back(getK(tA, tB, x, y, phi, mod));
        }
        return CRT(k, res);
    }
    long long Solve(long long A, long long B, long long P) {
        if (B == 1LL) {
            return 0LL;
        }
        long long phi = P - 1;
        vector<pair<long long, int>> res = Pollard_rho::prime_factor(phi);
        int rt = getOrg(P, phi, res);
        long long x = getX(A, rt, phi, P, res), y = getX(B, rt, phi, P, res);
        long long a, b;
        if (x == 0LL) {
            if (y == 0LL) {
                return 1LL;
            } else if (y == 1LL) {
                return 0LL;
            }
            return -1LL;
        }
        long long d;
        if (y % (d = __gcd(x, phi))) return -1;
        x /= d;
        phi /= d;
        y /= d;
        Exgcd(x, phi, a, b);
        a = ((__int128)a * y % phi + phi) % phi;
        return a;
    }
} // Pohlig_Hellman

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
```

```
    long long A, B, P;
    cin >> P >> A >> B;
    long long ans = Pohlig_Hellman::Solve(A, B, P);
    if (ans == -1) {
        cout << "no solution\n";
    } else {
        cout << ans << "\n";
    }
    return 0;
}
```

# 概率期望

## 鞅

鞅的引入是为了解决一类期望进行轮数的问题。

$A = \{A_0, A_1, A_2, \cdots, A_n\}$为随机过程，设$T$为停时（停止时刻）。

那么我们可以尝试构造这样一个势函数，他满足

1. $E(\Phi(A_{n+1}) - \Phi(A_n)|A_0, \cdots, A_n) = -1$
2. $\Phi(A_T)$为常数，且$\Phi(A_i) = \Phi(A_T)$当且仅当$i = T$

那么进一步我们可以尝试$X_n = \Phi(A_n) + n$，那么显然$\{X\}$是一个鞅，那么相应就有$E(X_T) = E(X_0)$，即$E(\Phi(A_T)) + E(T) = E(\Phi(A_0))$，那么我们就有

$$E(T) = E(\Phi(A_T)) - \Phi(A_0)$$

通常构造势函数的过程比较板。

### CF1025G Company Acquisitions

#### 题意

### CF1349D Slime and Biscuits

#### 题意

有$n$个人，第$i$个人拥有$a_i$块饼干，每次操作随机选择一块饼干，假设其属于$x$，然后再随机选择一个人$y$，求只有一个人拥有饼干的期望轮数。

#### 分析

套路的，我们设每个人的势函数$f(a)$，那么一开始的状态$A_0$对应的势函数$\Phi(A_0)$为$\sum f(a_{i,0})$，然后根据题意我们有$\Phi(A_{T+1}) = \sum_{i=1}^{n} \sum_{j!=i} \frac{1}{h_t*(n-1)} f(a_i - 1) + f(a_{j+1}) + \sum_{k!=i,k!=j} f(a_k)$，然后想办法转换求出$f$

# 莫比乌斯反演

## 前置

### 说明

$$(f + g)(x) = f(x) + g(x)$$
$$(f \cdot g)(x) = f(x) * g(x)$$

## 整除分块

对于$\lfloor \frac{n}{i} \rfloor$，我们发现其值最多只有$\sqrt{n}$ 个

同时有引理

$$\lfloor \frac{a}{b*c} \rfloor = \lfloor \frac{\lfloor \frac{a}{b} \rfloor}{c} \rfloor$$

于是我们有

```
for(int l = 1, r; l <= n; l = r + 1)
{
    r = min(n. n / (n / l));
    //code
}
```

## 积性函数

### 定义

$$\gcd(a,b) == 1, f(a*b) = f(a) * f(b)$$

称满足上式的函数为**积性函数**

其中，不满足互质条件但是式子仍然成立的称为**完全积性函数**

### 性质

若$f$是积性函数，同时$n = \prod_{i=1}^{m} p_i^{c_i}$，那么$f(n) = \prod_{i=1}^{m} f(p_i^{c_i})$

若$f, g$均为积性函数，那么以下函数必然也是积性函数

$$h(x) = f(x^p)$$
$$h(x) = f^p(x)$$
$$h(x) = f(x) * g(x)$$
$$h(x) = \sum_{d|x} f(d) * g(\frac{x}{d})$$

### 常见的积性函数

单位函数（完全积性）：$\epsilon(n) = [n == 1]$

恒等函数: $id_k(n) = n^k$

常数函数（完全积性）：$1(n) = 1$

除数函数: $\sigma_k(n) = \sum_{d|k} d^k$

欧拉函数: $\varphi_k(n) = \sum_{i=1}^{n} [gcd(i,n) == 1]$

莫比乌斯函数: $\mu(x) = \begin{cases} 1, n = 1 \\ 0, \exists d, d^2 | n \\ (-1)^k, k为x的质因子个数 \end{cases}$

# 迪利克雷卷积

$$h(x)=\sum_{d|n}f(d)g(\frac{d}{x})=\sum_{a*b=x}f(a)*g(b)$$

可以简写为

$$h=f*g$$

**性质**

$$f*g=g*f$$
$$(f*g)*h=f*(g*h)$$
$$(f+g)*h=f*h+g*h$$

$f$为任意函数，$f\times\epsilon=f$

$f,g$为任意积性函数，则$h=f\times g$也是积性函数

积性函数的逆元也是积性函数

$$\sum_{d|n}\mu(d)=\begin{cases}1,n==1\\0,n!=1\end{cases}$$

$$[gcd(i,j)==1]=\sum_{d|gcd(i,j)}\mu(d)$$

$$\varphi*1=id$$

$$1*\mu=\epsilon$$

$$\mu*id=\varphi$$

$f$，$g$为任意函数

如果$f(n)=\sum_{d|n}g(d)$，那么$g(n)=\sum_{d|n}\mu(d)*f(\frac{n}{d})$

如果$f(n)=\sum_{n|d}g(d)$，那么$g(n)=\sum_{n|d}\mu(d)*f(\frac{d}{n})$

$\sum_{d=1}^{n}\sum_{k=1}^{\lfloor\frac{n}{d}\rfloor}f(d)g(k)h(d*k)=\sum_{T=1}^{n}\sum_{k|T}f(d)g(k)h(T)$

# 杜教筛

杜教筛被用于处理一类数论函数的前缀和问题。对于数论函数$f$，杜教筛可以在低于线性时间的复杂度内计算其前缀和，而不要求$f$是数论函数。

**公式推导**

对于任意数论函数$f$，我们想求它的前缀和$F$，为此我们需要一个函数$g$

$$\begin{aligned}\sum_{i=1}^{n} f*g &= \sum_{i=1}^{n}\sum_{d|i} f(d)*g(\frac{i}{d}) \\ &= \sum_{i=1}^{n} g(i)\sum_{j=1}^{\lfloor \frac{n}{i} \rfloor} f(i) \\ &= \sum_{i=1}^{n} g(i)F(\lfloor \frac{n}{i} \rfloor)\end{aligned}$$

将我们想要的$F(n)$提出来，那么就有

$$F(n)*g(1) = \sum_{i=1}^{n}(f*g)(i) - \sum_{i=2}^{n} g(i)F(\lfloor \frac{n}{i} \rfloor)$$

那么容易发现，后面那个式子是典型的整除分块，现在我们只需要考虑

1. 快速计算$\sum_{i=1}^{n}(f*g)(i)$
2. 快速计算$\sum_{i=1}^{n} g(i)$

一般预处理$n^{\frac{2}{3}}$即可，直接做的复杂度为$O(n^{\frac{3}{4}})$

时间复杂度是$O(n^{\frac{2}{3}})$，确切说是$O(k)+O(\frac{n}{\sqrt{k}})$

$\sum_{i=1}^{n}\sum_{j=1}^{n}[gcd(i,j)==1] = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{d|gcd(i,j)}\mu(d) = \sum_{i=1}^{n}\mu(i)*(\lfloor \frac{n}{i} \rfloor)^2$

## 题目

求莫比乌斯函数和欧拉函数的前缀和

**分析**

根据$\mu*1=\epsilon$，我们就可以将1当成我们的$g$函数。

针对欧拉函数，我们可以考虑莫比乌斯反演也可以考虑杜教筛

反演

$$\sum_{i=1}^{n}\varphi(i) = \sum_{i=1}^{n}\sum_{j=1}^{i-1}[gcd(i,j)==1]$$

此时我们发现只要稍作修改就变成了我们熟悉的形式

筛

$\varphi*1=id$，同理可得

**code**

```
int mu[N], phi[N];
bool vis[N];
ll pre_mu[N], pre_phi[N];
map<ll,ll>pmu, pphi;
void init()
{
    for(int i = 1; i < N; ++i) mu[i] = 1, phi[i] = i, vis[i] = 0;
    for(int i = 2; i < N; ++i)
    {
        if(vis[i]) continue;
```

```
        mu[i] = -1;
        for(int j = 2 * i; j < N; j += i)
        {
            vis[j] = 1;
            if((j / i) % i == 0) mu[j] = 0;
            else mu[j] *= -1;
        }
        for(int j = i; j < N; j += i)
        {
            phi[j] = phi[j] / i * (i - 1);
        }
    }
    for(int i = 1; i < N; ++i)
    {
        pre_mu[i] = pre_mu[i - 1] + mu[i];
        pre_phi[i] = pre_phi[i - 1] + phi[i];
    }
}
ll query_mu(ll x)
{
    if(x < N) return pre_mu[x];
    if(pmu.find(x) != pmu.end()) return pmu[x];
    ll res = 1ll;
    for(ll l = 2, r; l <= x; l = r + 1)
    {
        r = min(x, x / (x / l));
        res -= query_mu(x / l) * (r - l + 1);
    }
    pmu[x] = res;
    return res;
}
ll query_phi(ll x)
{
    if(x < N) return pre_phi[x];
    if(pphi.find(x) != pphi.end()) return pphi[x];
    ll res = 0;
    for(ll l = 1, r; l <= x; l = r + 1)
    {
        r = min(x, x / (x / l));
        res += (query_mu(r) - query_mu(l - 1)) * (x / l) * (x / l);
    }
    return pphi[x] = (res - 1) / 2 + 1;
}
```

## 题目

求$\sum_{i=1}^{n}\sum_{j=1}^{n} i*j*\gcd(i,j)$

### 分析

$$\begin{aligned}\sum_{i=1}^{n}\sum_{j=1}^{n} i*j*\gcd(i,j) &= \sum_{d=1}^{n}\sum_{i=1}^{n}\sum_{j=1}^{n} i*j*d*[\gcd(i,j)==d] \\ &= \sum_{d=1}^{n} d^3 \sum_{i=1}^{\lfloor \frac{n}{d} \rfloor}\sum_{j=1}^{\lfloor \frac{n}{d} \rfloor} i*j*[\gcd(i,j)==1] \\ &= \sum_{d=1}^{n} d^3 \sum_{i=1}^{\lfloor \frac{n}{d} \rfloor}\sum_{j=1}^{\lfloor \frac{n}{d} \rfloor} i*j*\sum_{k|gcd(i,j)}\mu(k) \\ &= \sum_{d=1}^{n} d^3 \sum_{k=1}^{\lfloor \frac{n}{d} \rfloor} d^2\mu(d) \sum_{i=1}^{\lfloor \frac{n}{dk} \rfloor}\sum_{j=1}^{\lfloor \frac{n}{dk} \rfloor} i*j \\ &= \sum_{d=1}^{n} d^3 \sum_{k=1}^{\lfloor \frac{n}{d} \rfloor} k^2*\mu(k)*g(\lfloor \frac{n}{dk} \rfloor)\end{aligned}$$

其中$g(x)=(\frac{x(x+1)}{2})^2$

不妨令$T=d*k$

那么

$$\begin{aligned}\sum_{d=1}^{n} d^3 \sum_{k=1}^{\lfloor \frac{n}{d} \rfloor} k^2*\mu(k)*g(\lfloor \frac{n}{dk} \rfloor) &= \sum_{d=1}^{n}\sum_{k=1}^{\lfloor \frac{n}{d} \rfloor} k^2*\mu(k)*g(\lfloor \frac{n}{dk} \rfloor)*d^3 \\ &= \sum_{d=1}^{n}\sum_{d|T} T^2*\mu(\frac{T}{d})*g(\lfloor \frac{n}{T} \rfloor)*d \\ &= \sum_{T=1}^{n} g(\lfloor \frac{n}{T} \rfloor)*T^2\sum_{d|T}\mu(\frac{T}{d})*d\end{aligned}$$

由于$\mu*id=\varphi$

于是

$$\sum_{T=1}^{n} g(\lfloor \frac{n}{T} \rfloor)*T^2\sum_{d|T}\mu(\frac{T}{d})*d = \sum_{T=1}^{n} g(\lfloor \frac{n}{T} \rfloor)*T^2\varphi(T)$$

然后对于前半部分正常维护后半部分用杜教筛求即可。

```
vector<int>phi,prime,v;
vector<mint>pre;
map<ll,mint>ppre;
void init()
{
    phi.resize(N);
    prime.resize(N);
    v.resize(N);
    pre.resize(N);
    phi[1] = 1;
    int tot = 0;
    for(int i = 2; i < N; ++i)
    {
        if(v[i] == 0) {
            phi[i] = i - 1;
            v[i] = i;
            prime[++tot] = i;
```

```
        }
        for(int j = 1; j <= tot; j ++)
        {
            if(prime[j] > v[i] || prime[j] > N / i) break;
            v[i * prime[j]] = prime[j];
            phi[i * prime[j]] = phi[i] * (i % prime[j] ? prime[j] - 1 :
prime[j]);
        }
    }
    for(int i = 1; i < N; ++i)
    {
        pre[i] = pre[i - 1] + 1ll * phi[i] * i % mod * i % mod;
    }
}

//map<ll,
mint inv2,inv3;
mint query_s(ll x)
{
    mint t = mint(x);
    return t * (t + 1) * (2 * t + 1) * inv2 * inv3;
}
mint query_g(ll x)
{
    mint r = mint(x);
    r = r * (r + 1) * inv2;
    return r * r;
}
mint query(ll n)
{
    if(n < N) return pre[n];
    if(ppre.find(n) != ppre.end()) return ppre[n];
    mint res = query_g(n);
    for(ll l = 2, r; l <= n; l = r + 1)
    {
        r = min(n, n / (n / l));
        mint rk = query_s(r) - query_s(l - 1);
        res -= rk * query(n / l);
    }
    return ppre[n] = res;
}
mint G(ll x)
{
    mint r = x;
    r = r * r * (r + 1) * (r + 1);
    r *= qpow(mint(4), mod - 2);
    return r;
}
mint ask(ll n)
{
    mint res = 0;
    for(ll l = 1, r; l <= n; l = r + 1)
    {
        r = min(n, n / (n / l));
        mint g = G(n / l);
        mint k = query(r) - query(l - 1);
        res += g * k;
    }
```

```
    return res;
}
   init();
    cout<<ask(n)<<'\n';
```

## 莫比乌斯函数筛

```
int mu[N];
bool vis[N];
void init()
{
    int n = 1e6;
    for(int i = 1; i <= n; ++i) mu[i] = 1, vis[i] = 0;
    for(int i = 2; i <= n; ++i)
    {
        if(vis[i]) continue;
        mu[i] = -1;
        for(int j = 2 * i; j <= n; j += i)
        {
            vis[j] = 1;
            if((j / i) % i == 0) mu[j] = 0;
            else mu[j] *= -1;
        }
    }
}
```

## 欧拉函数筛

```
int p[N], phi[N];
int tot;
bool vis[N];
void init()
{
    for(int i = 1; i <= n; ++i) phi[i] = i;
    for(int i = 2; i <= n; ++i)
    {
        if(phi[i] == i)
        {
            for(int j = i; j <= n; j += i)
            {
                phi[j] = phi[j] /i * (i - 1);
            }
        }
    }
}
```

# 组合数学

# 母函数

## 母函数重要思想

### 将组合问题的加法与幂级数的乘幂对应起来

## hint

通常而言，我们将普通母函数处理组合问题，指数母函数处理排列问题

# 整数划分

将$n$划分为若干个不超过$m$的数的方案数

```
for(int i = 1; i <= n; ++i)
{
    for(int j = 1; j <= m; ++j)
    {
        if(i == 1 || j == 1) dp[i][j] = 1;
        else if(i < j) dp[i][j] = dp[i][i];
        else if(n == m) dp[i][j] = 1;
        else dp[i][j] = dp[i][j - 1] +  dp[i - j][j];
    }
}
```

# 组合数

## 二项式定理

$$(a+b)^n = a^n + C_n^1 a^{n-1}b + \cdots + C_n^{n-1}ab^{n-1} = \sum_{i=0}^{n} C_n^i a^i b^{n-i}$$

$$(a+b)^\alpha = \sum_{i=0}^{\infty} \binom{\alpha}{i} a^i b^{n-i}, \binom{\alpha}{i} = \frac{\alpha * (\alpha-1) * \cdots * (\alpha-i+1)}{i!}$$

$$(a+b)^{-\alpha} = \sum_{i=0}^{\infty} \binom{-\alpha}{i} a^i b^{n-i}, \binom{\alpha}{i} = (-1)^i * \left(\binom{\alpha+i-1}{i}\right)$$

$$\sum_{i=0}^{k} \binom{n}{i}\binom{m}{k-i} = \binom{n+m}{k}$$

## 常用公式

$$C_n^k = C_{n-1}^{k-1} + C_{n-1}^k$$

$$C_n^k = C_n^{n-k}$$

$$C_n^k = \frac{n-k+1}{k} C_n^{k-1}$$

$$\sum_{i=0}^{n} C_n^i = 2^n$$

$$\sum_{i=0}^{n} i * C_n^i = n * 2^{n-1}$$

$$\sum_{i=1,3,5,7,\cdots}^{n} C_n^i = \sum_{i=0,2,4,6,\cdots}^{n} C_n^i = 2^{n-1}$$

$$\sum_{i=0}^{S} C_n^i C_m^{S-i} = C_{n+m}^s$$

## $lucas$定理

若$p$为素数，则有

$$C_n^m = \prod_{i=0}^{k} C_{n_i}^{m_i} (\mod p)$$

$$n = \sum_{i=0}^{k} n_i * p^i$$

$$m = \sum_{i=0}^{k} m_i * p^i$$

$$C_n^m = C_{n\%p}^{m\%p} * C_{n\backslash p}^{m\backslash p}$$

## 康托展开

用来求排列的排名，时间复杂度$O(n \log n)$

## 伪代码

```
int res = 1;
for(int i = 1; i <= n; ++i)
{
    res += (a[i] - 1 - cnt[比第i位数小且出现过的次数]) * （n - i)!;
}
```

## 斐波那契数列

$$F_{2*k} = F_k(2F_{k+1} - F_k)$$
$$F_{2*k+1} = F_{k+1}^2 + F_k^2$$
$$F_{n-1}F_{n+1} - F_n^2 = (-1)^n$$
$$F_{n+k} = F_kF_{n+1} + F_{k-1}F_n$$
$$F_{2*n} = F_n(F_{n+1} + F_{n-1})$$
$$F_n \mid F_{nk}$$
$$\text{如果} F_a \mid F_b, \text{那么} a \mid b$$
$$(F_m, F_n) = F_{(m,n)}$$

每一个数都可以唯一的用斐波那契数和表示

模意义下具有周期性

**卢卡斯数列**

$L_0 = 2, L_1 = 1$的广义斐波那契额数列

### 卡特兰数

$cat_n = \frac{1}{n+1}\binom{2*n}{n}$

### 贝尔数

### 斯特林数

### 第一类斯特林数

### 第二类斯特林数

### 伯努利数

### 分拆数

### 小球盒子

# 质因数分解

```
namespace Factor {
//  using ull = std::uint64_t;

    /*Montgomery Multiplt Template*/

    ull modmul(ull a, ull b, ull M) {
        ll ret = a * b - M * ull(1.L / M * a * b);
        return ret + M * (ret < 0) - M * (ret >= (ll)M);
    }

    ull modpow(ull b, ull e, ull mod) {
        ull ans = 1;
        for (; e; b = modmul(b, b, mod), e /= 2)
            if (e & 1) ans = modmul(ans, b, mod);
        return ans;
    }
    bool isPrime(ull n) {
        if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
        std::vector<ull> A = {2, 325, 9375, 28178, 450775, 9780504, 1795265022};
```

```
        ull s = __builtin_ctzll(n - 1), d = n >> s;
        for (ull a : A) {  // ^ count trailing zeroes
            ull p = modpow(a % n, d, n), i = s;
            while (p != 1 && p != n - 1 && a % n && i--) p = modmul(p, p, n);
            if (p != n - 1 && i != s) return 0;
        }
        return 1;
    }
    ull pollard(ull n) {
        auto f = [n](ull x, ull k) { return modmul(x, x, n) + 1; };
        ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;
        while (t++ % 40 || std::gcd(prd, n) == 1) {
            if (x == y) x = ++i, y = f(x, i);
            if ((q = modmul(prd, std::max(x, y) - std::min(x, y), n))) prd = q;
            x = f(x, i), y = f(f(y, i), i);
        }
        return std::gcd(prd, n);
    }
    std::vector<ull> factor(ull n) {
        if (n == 1) return {};
        if (isPrime(n)) return {n};
        ull x = pollard(n);
        auto l = factor(x), r = factor(n / x);
        l.insert(l.end(), r.begin(), r.end());
        return l;
    }
}

auto fac=Factor::factor(x);
```

```
//防溢出取模乘法
inline ll ksc(ull x,ull y ,ll p){return (x*y-(ull)((ld)x/p*y)*p+p)%p;}
inline ll ksm(ll x,ll y,ll p){ll
res=1;for(;y;y>>=1,x=ksc(x,x,p))if(y&1)res=ksc(res,x,p);return res;}
inline bool mr(ll x,ll p)
{
    if(ksm(x,p-1,p)!=1)return 0;
    ll y=p-1,z;
    while(!(y&1))
    {
        y>>=1;z=ksm(x,y,p);
        if(z!=1&&z!=p-1)return 0;
        if(z==p-1)return 1;
    }
    return 1;
}
//生日攻击 第一个重复的数前面期望大约有sqrt(PI*N/2)个
ll te_per[20]=
{0,2,3,5,7,433,61,24251};//{0,2,325,9375,28178,450775,9780504,1795265022};
int te_num=7;
inline bool isprime(ll x)
{
    if(x<3)return x==2;
    if(x&1==0)return 0;
    ll d=x-1,r=0;
```

```
    for(int i=1;i<=te_num;++i)if(x==te_per[i])return 1;
    for(int i=1;i<=te_num;++i)
        if(!(x%te_per[i])||!mr(te_per[i],x))return 0;
    return 1;
}
//快速筛质因数 prho,结果存在ys里面
ll ys[N];
int ind;//使用前清空
inline ll rho(ll p)
{
    ll x,y,z,c,g;
    re int i,j;
    while(1)
    {
        y=x=rand()%p;//使用前记得srand(time(0))
        z=1,c=rand()%p;
        i=0,j=1;
        while(++i)
        {
            x=(ksc(x,x,p)+c)%p;
            z=ksc(z,Abs(y-x),p);
            if(x==y||!z)break;
            if(!(i%127)||i==j)
            {
                g=gcd(z,p);
                if(g>1)return g;
                if(i==j)y=x,j<<=1;
            }
        }

    }
}
inline void prho(ll p)
{
    if(p==1)return ;
    if(isprime(p)){ys[++ind]=p;return ;}
    ll pi=rho(p);
    while(p%pi==0)p/=pi;
    prho(pi);
    prho(p);
}
void solve()
{
    ll n;
    cin>>n;
    if(isprime(n))cout<<"Prime\n";
    else
    {
        ind=0;
        prho(n);
        ll maxx=-1;
        for(int i=1;i<=ind;++i)if(maxx<ys[i])maxx=ys[i];
        cout<<maxx<<'\n';
    }
}
```

# 取模类

```
/注意qpow在这里，base和返回值类型一致
int norm(int x) {
    if (x < 0) {
        x += mod;
    }
    if (x >= mod) {
        x -= mod;
    }
    return x;
}
template<class T>
T qpow(T base, ll power) {
    T res = 1;
    while (power) {
        if (power & 1) res = res * base;
        base = base * base;
        power >>= 1;
    }
    return res;
}

struct mint {
    int x;
    mint(int x = 0) : x(norm(x)) {}
    mint(ll x) : x(norm((int)(x % mod))) {}
    int val() const { return x; }
    mint operator-() const { return mint(norm(mod - x)); }
    mint inv() const { assert(x != 0); return qpow(*this, mod - 2); }
    mint &operator*=(const mint &rhs) { x = (ll)x * rhs.x % mod; return *this; }
    mint &operator+=(const mint &rhs) { x = norm(x + rhs.x); return *this; }
    mint &operator-=(const mint &rhs) { x = norm(x - rhs.x); return *this; }
    mint &operator/=(const mint &rhs) { return *this *= rhs.inv(); }
    friend mint operator*(const mint &lhs, const mint &rhs) { mint res = lhs;
res *= rhs; return res; }
    friend mint operator+(const mint &lhs, const mint &rhs) { mint res = lhs;
res += rhs; return res; }
    friend mint operator-(const mint &lhs, const mint &rhs) { mint res = lhs;
res -= rhs; return res; }
    friend mint operator/(const mint &lhs, const mint &rhs) { mint res = lhs;
res /= rhs; return res; }
    friend std::istream &operator>>(std::istream &is, mint &a) { ll v; is >> v;
a = mint(v); return is; }
    friend std::ostream &operator<<(std::ostream &os, const mint &a) { return os
<< a.val(); }
};
```

```
template <std::uint32_t P>struct MontInt {
    using u32 = std::uint32_t;
    using u64 = std::uint64_t;
    u32 v;
```

```
    static constexpr u32 get_r() {
        u32 iv = P;

        for (u32 i = 0; i != 4; ++i)
            iv *= 2U - P * iv;

        return -iv;
    }
    static constexpr u32 r = get_r(), r2 = -u64(P) % P;//限定词不能省

    MontInt() = default;
    ~MontInt() = default;
    constexpr MontInt(u32 v) : v(reduce(u64(v) * r2)) {}
    constexpr MontInt(const MontInt &rhs) : v(rhs.v) {}

    static constexpr u32 reduce(u64 x) {
        return x + (u64(u32(x) * r) * P) >> 32;
    }

    static constexpr u32 norm(u32 x) {
        return x - (P & -(x >= P));
    }
    constexpr u32 get() const{
        u32 res = reduce(v) - P;
        return res + (P & -(res >> 31));
    }


    constexpr MontInt operator-() const {
        MontInt res;
        return res.v = (P << 1 & -(v != 0)) - v, res;
    }
    constexpr MontInt inv() const {
        return pow(-1);
    }
    constexpr MontInt &operator=(const MontInt &rhs) {
        return v = rhs.v, *this;
    }
    constexpr MontInt &operator+=(const MontInt &rhs) {
        return v += rhs.v - (P << 1), v += P << 1 & -(v >> 31), *this;
    }
    constexpr MontInt &operator-=(const MontInt &rhs) {
        return v -= rhs.v, v += P << 1 & -(v >> 31), *this;
    }
    constexpr MontInt &operator*=(const MontInt &rhs) {
        return v = reduce(u64(v) * rhs.v), *this;
    }
    constexpr MontInt &operator/=(const MontInt &rhs) {
        return this->operator*=(rhs.inv());
    }
    friend MontInt operator+(const MontInt &lhs,
                                        const MontInt &rhs) {
        return MontInt(lhs) += rhs;
    }
    friend MontInt operator-(const MontInt &lhs,
                                        const MontInt &rhs) {
        return MontInt(lhs) -= rhs;
    }
```

```
    friend MontInt operator*(const MontInt &lhs,
                                        const MontInt &rhs) {
        return MontInt(lhs) *= rhs;
    }
    friend MontInt operator/(const MontInt &lhs,
                                        const MontInt &rhs) {
        return MontInt(lhs) /= rhs;
    }
    friend bool operator==(const MontInt &lhs, const MontInt &rhs) {
        return norm(lhs.v) == norm(rhs.v);
    }
    friend bool operator!=(const MontInt &lhs, const MontInt &rhs) {
        return norm(lhs.v) != norm(rhs.v);
    }
    constexpr MontInt pow(ll y) const {
        if ((y %= P - 1) < 0)
            y += P - 1; // phi(P) = P - 1, assume P is a prime number

        MontInt res(1), x(*this);

        for (; y != 0; y >>= 1, x *= x)
            if (y & 1)
                res *= x;

        return res;
    }
};
```

# dp

## 单调栈单调队列

### 定义

简单来说就是满足单调性的栈和队列

出去条件：下标不在范围或新来的元素会比老元素有优势，老元素出队

栈

```
void insert(int x)
{
    while(!sta.empty() && sta.top() < x)
    {
        sta.pop();
    }
    sta.push(x);
}
```

```
head = 1; tail = 0;
void insert(int x)
{
    while (head <= tail && a[q[tail]] >= a[R[i]]) tail--;
    q[++tail] = R[i];
    while(q[head] < L[i])head++;
}
```

## 优化dp

对于形如$f[i]=max_{L[i]\le j\le R[i]}(f[j]+w[i])$的式子进行优化

## 例题

### 烽火传递

#### 题意

在某两个城市之间有 n 座烽火台，每个烽火台发出信号都有一定的代价。

为了使情报准确传递，在连续 m 个烽火台中至少要有一个发出信号。

#### 数据范围

$1\le m\le n\le 2\times 10^5, 0\le a_i\le 1000$

#### 分析

首先有转移式，$dp[i]=min_{i-m+1\le j\le i-1}(dp[j]+a[i])$

然后套板子即可

```
    dp[0] = 0;
    int l = 1, r = 0;
    int ans = INF;
    for(int i = 1; i <= n; ++i)
    {
        while(l <= r && dp[sta[r]] >= dp[i - 1])r--;
        sta[++r]=i - 1;
        while(sta[l] < i - m)l++;
        dp[i] = dp[sta[l]] + a[i];
        if(i > n - m)ans=min(ans, dp[i]);
    }
```

### 修剪草坪

#### 题意

在一年前赢得了小镇的最佳草坪比赛后，FJ 变得很懒，再也没有修剪过草坪。现在，新一轮的最佳草坪比赛又开始了，FJ 希望能够再次夺冠。

然而，FJ 的草坪非常脏乱，因此，FJ 只能够让他的奶牛来完成这项工作。FJ 有 $N$ 只排成一排的奶牛，编号为 1 到 *N*。每只奶牛的效率是不同的，奶牛 *i*的效率为 $E_i$。

靠近的奶牛们很熟悉，如果 FJ 安排超过 *K* 只连续的奶牛，那么这些奶牛就会罢工去开派对。因此，现在 FJ 需要你的帮助，计算 FJ 可以得到的最大效率，并且该方案中没有连续的超过 *K* 只奶牛。

#### 分析

即$k+1$中必有一个不选，然后考虑不选之后的最小效率

同上

```
cin >> n >> k; k++;//区间长度
for(int i = 1;i <= n; ++i)cin>>a[i];
dp[0] = 0;
int l = 1, r = 0;
ll ans = LINF;
ll sum = 0;
for(int i = 1;i <= n; ++i)sum += a[i];
for(int i = 1;i <= n; ++i)
{
    while(l <= r && dp[sta[r]] >= dp[i-1])r--;
    sta[++r] = i - 1;
    while(sta[l] < i - k)l++;
    dp[i] = dp[sta[l]] + a[i];
    if(i > n - k)ans = min(ans, dp[i]);
}
cout << sum - ans << '\n';
```

### 绿色草坪

#### 题意

高二数学《绿色通道》总共有 $n$ 道题目要抄，编号 $1\cdots n$，抄第 $i$ 题要花 $a_i$ 分钟。小 Y 决定只用不超过 $t$ 分钟抄这个，因此必然有空着的题。每道题要么不写，要么抄完，不能写一半。下标连续的一些空题称为一个空题段，它的长度就是所包含的题目数。这样应付自然会引起马老师的愤怒，最长的空题段越长，马老师越生气。
现在，小 Y 想知道他在这 $t$ 分钟内写哪些题，才能够尽量减轻马老师的怒火。由于小 Y 很聪明，你只要告诉他最长的空题段至少有多长就可以了，不需输出方案。

#### 数据范围

$0<n\leq 5\times 10^4, 0<a_i\leq 3000, 0<t\leq 10^8$

#### 分析

容易发现答案具有单调性，因此不妨二分最长连续空段为$m$，然后跑一次$dp$，代码与上述相同

### The Great Wall II

#### 题意

将n个数分成m段，每一段的代价是这一段数的最大值，求$m=1\cdots n$时的最小代价

#### 分析

朴素写法即枚举前$i$个数，分成$j$段，然后考虑第$j+1$段是从那里开始到$i+1$

考虑优化，我们发现状态可以分为$a_i$作为最后一段贡献或者不是最后一段最大值

我们记$dp[i][j]$为前$i$个数分成$j$段的代价，那么就有$dp[i][j]=min_{pos\leq p\leq i-1}(dp[p][j-1]+a[i])$，表示$a_i$作为最后一段最大值，然后$pos$表示$a[pos-1]\geq a[i]$的最大的值，即上一个比$a_i$大的位置的下一位

如果不是最大值，那就有$dp[i][j]=dp[pos-1][j]$，这时将最后一段区间挂到$pos-1$的第$j$段上，那么$a_i$一定不是这一段的代价

首先单调栈维护上一个$pos$的值，$dp[i][j]$不具有单调性，因此我们考虑在为维护$pos$的时候顺便也维护对应的值，即栈内两个相邻元素的对应的下标的$min\ dp$。

# 期望dp概率dp

## 期望

### 常用公式

$$E(X) = \sum p(X_i)X_i$$
$$E(X|D) = \sum p(X_i|D)X_i$$
$$P(X|D) = \frac{P(XD)}{P(D)}$$
$$P(A) = \sum P(A|B_i)P(B_i), \quad B_i \cap B_j = \emptyset, \quad \cup B_i = \Omega$$

**通常我们期望是从后往前推，而概率一般从前往后**

**期望具有线性性**

#### 题目

#### 题意

$n$面的骰子，问每个面都被投出过的期望次数是多少

#### 分析

$dp[i]$表示已经有$i$面被投出过，那么接下来还有$n-i$没有被投出过

$$dp[i] = \frac{i}{n}dp[i] + \frac{n-i}{n}dp[i+1] + 1$$

稍作整理即可得出递推式

#### 题意

若干个宝箱，宝箱里有东西的概率是$p$，求最后$n$次中至少有$k$个宝箱期望次数

#### 分析

这题$n$，$k$很小，只有$6$

考虑枚举最后$n$位的状态，我们有

$$dp[z_0z_1z_2z_3z_4z_5] = 1 + p * dp[1z_0z_1z_2z_3z_4] + (1-p) * dp[0z_0z_1z_2z_3z_4]$$

然后高斯消元即可

#### 题意

已知牌堆一共有34种类型，每种类型的牌四张

在开局会发给你13张牌，（**保证**每种类型的牌的数量小于等于2），以此进行游戏

在每一回合，先从牌库抽一张牌，若此时十四张手牌为（7种不同的类型，每种各两张）则算作胜利，结束游戏。否则就从十四张牌中选择一张放入弃牌堆（不放回的意思）

一共T组样例，每次给你起始手牌，询问在**最优策略的情况下**的获胜的期望次数。

**分析**

首先思考什么是最优策略

如果我当前拿到的牌，我手中已经有了2张，那么我们就直接丢弃这张牌

如果我当前已经有一张，那么我们丢弃另外的只有一张的牌

如果我当前没有，那么直接把这张牌丢了

为什么呢，首先我一开始手上一定有单牌，那么最后我们肯定是和一开始留下的牌进行匹配，不然如果我们先前把某个类型的牌丢了，那么后面我们就不可能再拿这个类型的牌，因为牌库里这种牌的数量已经小于$3$了，比一开始就有的牌抽到的概率低。

于是我们发现每次我只需要关心牌库里还有几张牌，以及我当前有几张单牌。

于是设$dp[i][j]$表示牌库里还有$j$张牌，手上有$i$张单牌，然后到达全匹配的期望次数。

容易发现$dp[1][3]=1$。即当我手上有只有一张单牌，同时牌库里还有三张牌时，我们怎么样都可以抽出一个

和其匹配的牌。

同时我们有转移方程

$$dp[i][j]=1+\frac{3*i}{j}dp[i-2][j-1]+\frac{j-3*i}{j}dp[i][j-1]$$

这里$-2$是因为还有一张牌被我们丢了

当$i$等于1时，我们有

$$dp[i][j]=1+\frac{j-3*i}{j}dp[i][j-1]$$

这是因为只有我们没抽到想要的牌，我们才需要继续抽

**题意**

一个长度为$n$的序列，每次操纵随机选取两个下标，然后将这两个下标对应的数进行交换，问$m$次操作后每一位和一开始不同的个数的期望

**分析**

一道很好的期望题，虽然赛时尝试打表并没有找到什么规律

我们计$f_m$表示操作%$m$次后，$a_0=0$的概率。

因为每一位其实是等价的，我们计算时可以

$E(\sum X_i)=\sum E(X_i)=n*E(X_0)=n*f_m$

其中$E(X_i)$表示第$i$位和一开始一样的期望

因此只需要计算$f_m$即可

首先第$m$次和原来相同，来自于前一次和原来相同，然后操作一次不变，或者前一次和原来不同，但是操作完后回到原位置

原来相同，操作不变，可能是两次交换的下标都是$0$，或者都不是$0$

原来不同，那么只有两种情况回到原位置

$f_m = \frac{1+(n-1)^2}{n^2} f_{m-1} + \frac{2}{n^2}(1 - f_{m-1})$

化简得到$f_m = \frac{2}{n^2} + \frac{n-2}{n} f_{m-1}$

然后就可以用矩阵快速幂之类的方法维护了。

## 随机游走

首先对于任何一个点的期望经过次数，我们都有

$d(x)$表示点$x$的度

$$E(X) = \sum_{X \to Y} \frac{E[Y]}{d(Y)}$$

而每条边经过的期望次数我们可以写成

$$f(u,v) = \frac{E(u)}{d(u)} + \frac{E(v)}{d(v)}$$

然后通过高斯消元我们就可以解决问题

## 树上随机游走

不妨计$f[i]$为从子节点$i$走向父节点的期望步数，我们有

$$f[u] = \frac{w(u,p_u) + \sum_{v \in son_u}(w(u,v) + f[v] + f[u])}{deg[u]}$$

化简可得

$$f[u] = \sum_{(u,v) \in E} w(u,v) + \sum_{v \in son_u} f[v]$$

当边权都是1时，我们进一步可以得到

$$f[u] = 2 * size[u] - 1$$

其中$size[u]$是$u$子树的大小

计$G[x]$为从父节点$f$走向子节点$x$的期望步数

$$G[x] = \frac{w(p_x,x) + (\ w(p_x,p_{p_x}) + G[p_x] + G[x]\ ) + \sum_{v \in son_{f_x}, v \neq x}(\ w(p_x,v) + f[v] + G[x])}{deg[p_x]}$$

化简可得

$$G[x] = G[p_x] + f[p_u] - f[u]$$

```
//代码存疑需要修改，主要初始值需要斟酌
void dfs1(int u, int p) {
    bool flag = 1;
    for (auto [v,w] : G[u]) {
        if (v == p) continue;
        flag = 0;
        dfs1(v, u);
        f[u] += f[v] + w;
    }
    if(flag) f[u] = G[u][0].second;
}
```

```
void dfs2(int u, int p) {
    for (auto v : G[u]) {
        if (v == p) continue;
        g[v] = g[u] + f[u] - f[v];
        dfs2(v, u);

    }
}
```

# 奇怪dp

## 状态转换

**题意**

每一位数字$a_i$在$-m \le a_i \le m$之间，同时任意长度大于等于2的子段和非负，问这样的序列有多少个

**分析**

很不好考虑的一个问题。

关键在于状态的选择。题目要求的是任意大于2的字段和都非负，我们如果考虑最后一位放什么的话比较难处理，主要在于我们没办法保证我们选择的第$i$个数满足$a_i + a_{i-1} + a_{i-2} \ge 0$，但是仔细考虑的话可以发现我们可以将最后一位放什么的状态改成后缀最小的子段的状态，这样我们就可以转移了。

考虑最小后缀的状态，因为任意长度为2的子段和都大于等于0，于是这个最小后缀的范围也在$-m \le suf \le m$之间。

如果当前最小后缀小于零，那么只有一种情况，即前面一位放的是正数，同时这一位放的是等于这个最小后缀的负数，这样才可以满足条件。

如果当前最小后缀大于等于0，那么就有两种情况，一种是前一位放的数要大于等于当前位，或者说前一位的最小后缀要大于等于当前位的数，或者前一位虽然要小于当前位，但是是负数，那么当前位可以放一个正数使其与前一位相加等于最小后缀，形式化的说，应该是下面这个样子。

$$dp[i][j] = \begin{cases} \sum_{0 \le k \le m} dp[i-1][k] + \sum_{j-m \le k < 0} dp[i-1][k], & \text{if } j \ge 0 \\ \sum_{-j \le k \le m} dp[i-1][k], & \text{if } j < 0 \end{cases}$$

稍作整理，我们就得到

$$dp[i][j] = \begin{cases} \sum_{j-m \le k \le m} dp[i-1][k], & \text{if } j \ge 0 \\ \sum_{-j \le k \le m} dp[i-1][k], & \text{if } j < 0 \end{cases}$$

然后处理一下后缀就可$O(n^2)$完成。

**题目**

23杭电多校第六场1008

**题意**

$Alice$和$Bob$在做游戏。

一开始他们有一个序列，每次他们都可以选择一个位置$pos$，如果第一位到$pos$的和大于等于$pos+1$到最后一位的前缀和，那么我们就把$pos$后面的给截去，只留下前面的部分，否则我们留下后面的。

这个游戏显然有必胜必败策略，于是$Alice$和$Bob$希望自己赢的时候留下的数字尽可能大，不然就尽可能小。

**分析**

在不考虑额外的限制条件的情况下，我们可以轻易写出$O(n^3)$的$sg$函数

```
void dfs(int l, int r)
{
    if(vis[l][r]) return;
    vis[l][r] = 1;
    vector<bool>v(r - l + 10, 0);
    for(int k = l; k < r; ++k)
    {
        if(pre[k] - pre[l - 1] >= pre[r] - pre[l - 1])
        {
            dfs(l, k);
            v[sg[l][k]] = 1;
        }
        else
        {
            dfs(k + 1, r);
            v[sg[k + 1][r]] = 1;
        }
    }
    for(int i = 0; i < r - l + 10; ++i)
    {
        if(!vis[i])
        {
            sg[l][r] = i;
            return ;
        }
    }
}
```

现在我们要记录最后剩下的值，那么有一个比较巧妙的方法是我们将必胜态记录为大于0的状态，必败态记录为小于0的状态，那么当前这个状态就必须取后继状态的最小值，当前状态是必胜仅当后继状态有一个小于0，当前必败，仅当后继都大于0。

然后状态的绝对值我们就可以记为当前状态到最后状态的留下的值，仔细思考，这样是符合逻辑的，当前必胜，那么我们就挑一个最小的数，这样取负之后就是当前状态的答案，当前必败，我们就取一个最小的正数，同样符合条件。

然后我们就写出了一个区间一样的东西。

```
for(int i = 1; i <= n; ++i) dp[i][i] = -a[i];
for(int L = 2; L <= n; ++L)
{
    for(int l = 1; l + L - 1 <= n; ++l)
    {
        int r = l + L - 1;
        ll tmp = LINF;
        for(int k = l; k < r; ++k)
        {
            if(pre[k] - pre[l - 1] >= pre[r] - pre[l - 1])
            {
                tmp = min(tmp, dp[l][k]);
```

```
            }
            else
            {
                tmp = min(tmp, dp[k + 1][r]);
            }
        }
        dp[l][r] = -tmp;
    }
}
```

仔细考察我们写出来的东西，我们可以发现如果没有$if$语句，那么就是一个全部取$\min$的情况，似乎是比较典型的单调栈优化的过程，接下来仔细分析是否可以用单调栈优化。

我们考虑对每个点开两个单调栈，一个单调栈表示当前区间长度为$L$的情况下，所有作为左端点的有效决策，另一个就是作为右端点的有效决策。

我们发现当$p$是保留左端点的一个决策，当且仅当$pre[p]-pre[l-1]\geq pre[r]-pre[p]$，稍作整理既得$2*pre[p]\geq pre[r]+pre[l-1]$，在这个式子里，$l$是不变的，$r$是增大的，因此$p$的值越小，越容易不在有效取值范围内，我们考虑将$p$从小到大排一排，那么后面的一些$p$仍然可以作为我们的一个决策，然后我们希望这个决策带来的值越小越好，因此我们将对应的值也从小到大排，每一次新加入的决策就在最后面。也就是说，下标越大，值越小，越是我们想要的值，那么这个过程我们就可以拿单调栈维护。

保留右端点同理。

于是我们就有如下的代码

```
int n;
ll dp[3010][3010];
int a[3010];
ll pre[3010];
deque<int> top[3010],top2[3010];
void add(int l,int r)
{
    if(pre[r - 1] - pre[l - 1] >= a[r])
    {
        while(!top[l].empty())
        {
            int p = top[l].front();
            if(2 * pre[p] >= pre[l - 1] + pre[r])
            {
                break;
            }
            top[l].pop_front();
        }
        while(!top[l].empty())
        {
            int p = top[l].back();
            if(dp[l][p] < dp[l][r - 1])
            {
                break;
            }
            top[l].pop_back();
        }
        top[l].push_back(r - 1);
        //最优决策为 top[l].front()
    }
```

```
    if(a[l] < pre[r] - pre[l])
    {
        while(!top2[r].empty())
        {
            int p = top2[r].back();
            if(2 * pre[p] < pre[l - 1] + pre[r])
            {
                break;
            }
            top2[r].pop_back();
        }
        while(!top2[r].empty())
        {
            int p = top2[r].front();
            if(dp[p + 1][r] < dp[l + 1][r])
            {
                break;
            }
            top2[r].pop_front();
        }
        top2[r].push_front(l);
        //最优决策为 top2[r].back()
    }
}
ll query(int l, int r)
{
    ll w = LINF;
    if(!top[l].empty())
    {
        w = min(w, dp[l][top[l].front()]);
    }
    if(!top2[r].empty())
    {
        w = min(w, dp[top2[r].back() + 1][r]);
    }
    return w;
}
void solve(int cas)
{
    cin >> n;
    for(int i = 1; i <= n; ++i)
    {
        cin>>a[i];
        dp[i][i]=-a[i];
        pre[i]=pre[i-1]+a[i];
        top[i].clear();
        top2[i].clear();
    }
    for(int len = 2; len <= n; ++len)
    {
        for(int l = 1; l + len - 1 <= n; ++l)
        {
            int r = l + len - 1;
            add(l, r);
            dp[l][r] = -query(l, r);
        }
    }
    if(dp[1][n] >= 0)
```

```
        cout << "Alice " << dp[1][n] << '\n';
    else
        cout << "Bob " << -dp[1][n] << '\n';
}
```

# 区间dp

### 题目

### 题意

你有$n$张牌，每张牌都有类型和等级$(初始等级为1)$，每张牌打出去后都会产生$p^{x-1}*V_i$的收益，其中$x$是这张牌的等级，$i$是这张牌的种类。你可以选择两张相邻且种类相同等级相同的牌，将其合并为一张更高等级的牌，求所有牌打完后的最高收益。

### 分析

这显然是一道区间，即我们可以考虑这个区间产生的最大收益是多少。但是仅有这个状态是不够的，因为还有合并的状态，因此我们考虑这个区间只有一张牌的情况，此时任意的合并操作都可以看成两个区间留下一张牌的情况

即设状态为$dp[l][r][x][k]$，表示在$[l,r]$区间保留种类为$k$，等级为$x$的牌。

同时记$dp[l][r][0][0]$，表示这个区间牌都打光的情况。

如果留下来的牌等级为$1$，那么

$$dp[l][r][1][k]=\max_{l+1\le p\le r-1,a[i]==k}(dp[l][p-1][0][0]+dp[p+1][r][0][0])$$

同时，$p=l$或$p=r$的情况与之类似

如果留下的牌等级超过$1$，那么一定是合成来的

$$dp[l][r][x][k]=\max_{l\le p\le r-1}(dp[l][p][x-1][k]+dp[p+1][r][x-1][k])$$

计算这个区间的贡献，只需枚举这个区间最后一张牌是是什么

$$dp[l][r][0][0]=\max dp[l][r][x][k]+P^{x-1}*v[k]$$

```
for(int i=0;i<=n;++i)for(int p=0;p<=n;++p)
        for(int j=0;j<=7;++j)for(int k=0;k<=20;++k)dp[i][p][j][k]=-LINF;

    for(int i=1;i<=n;++i)dp[i][i][1][a[i]]=0,dp[i][i][0][0]=v[a[i]];
    for(int len=2;len<=n;++len)
    {
        for(int l=1;l+len-1<=n;++l)
        {
            int r=l+len-1;
            for(int p=l;p<=r;++p)
            {
                if(p!=l&&p!=r)dp[l][r][1][a[p]]=max(dp[l][r][1][a[p]],dp[l][p-1]
[0][0]+dp[p+1][r][0][0]);
                else if(p==l)dp[l][r][1][a[p]]=max(dp[l][r][1][a[p]],dp[p+1][r]
[0][0]);
                else if(p==r)dp[l][r][1][a[p]]=max(dp[l][r][1][a[p]],dp[l][p-1]
[0][0]);
```

```
            }

            for(int x=2;x<=R;++x)
            {
                for(int k=1;k<=m;++k)
                {
                    for(int p=l;p<r;++p)
                    {
                        dp[l][r][x][k]=max(dp[l][r][x][k],dp[l][p][x-1]
[k]+dp[p+1][r][x-1][k]);
                    }
                }

            }

            for(int x=1;x<=R;++x)
            {
                for(int k=1;k<=m;++k)
                {
                    dp[l][r][0][0]=max(dp[l][r][0][0],dp[l][r][x][k]+v[k]*pw[x-
1]);
                }
            }
        }
    }
    cout<<dp[1][n][0][0]<<'\n';
```

# 树形dp

## 题意

给定一棵树，你需要给树上每一个点赋**不同**的值，每个节点$u$的贡献$f[u]$是$mex_{v\in son_u}\{f[v]\}$，一颗树的贡献是所有节点的贡献，求这棵树的最大贡献

## 分析

首先对于一个节点来说，最大贡献肯定是只有它一个子树的贡献加上它这个子树的大小。

即我们可以假设这个点的所有子节点从零开始赋值，否则这个节点的贡献会减少，同时在儿子节点中只有一个产生贡献，因为0只有一个。那么贪心的选择就可以。

```
vector<int>G[N];
ll dp[N],s[N];
void dfs(int x,int f)
{
    dp[x]=0;
    s[x]=1;
    for(int y:G[x])
    {
        if(y==f)continue;
        dfs(y,x);
        dp[x]=max(dp[x],dp[y]);
        s[x]+=s[y];
    }
    dp[x]+=s[x];
```

```
}
```

## 题意

给定一棵树，每条边都有权值，每个点也都有权值。一个点$i$有一个修改代价$c[i]$，表示点$i$从权值$a$变成权值$b$需要$c_i * |a-b|$。

现在要求对于边$(u,v)$，都满足$min(w[u],w[v]) \le w(u,v) \le max(w[u],w[v])$，求最小修改代价

## 分析

每个点的取值可以认为不会超过这个点的度数+1个，枚举一下可能的取值然后对于相同情况考虑上下界取最小即可

```
ll calc(int x,int w)
{
    return c[x]*abs(d[x]-w);
}
void dfs(int x,int f,ll pre)
{
    vector<pair<ll,int> >v;
    ll s=0;
    for(int i=head[x];i;i=nxt[i])
    {
        int y=ver[i];
        if(y==f)continue;
        dfs(y,x,e[i]);
        s+=dp[y][1];
        v.emplace_back(e[i],y);
    }
    v.emplace_back(d[x],0);
    if(x!=1)v.emplace_back(pre,0);
    sort(v.begin(),v.end());
    ll ppre=0;
    for(int i=0;i<v.size();++i)
    {
        int j=i;
        ll tmp=min(dp[v[i].second][1],dp[v[i].second][0]);
        s-=dp[v[i].second][1];
        while(j<(int)v.size()-1&&v[j+1].first==v[i].first)
        {
            j++;
            s-=dp[v[j].second][1];
            tmp+=min(dp[v[j].second][1],dp[v[j].second][0]);
        }

        if(pre<=v[i].first||x==1)
        {
            dp[x][1]=min(dp[x][1],calc(x,v[i].first)+ppre+s+tmp);
        }
        if(pre>=v[i].first||x==1)
        {
            dp[x][0]=min(dp[x][0],calc(x,v[i].first)+ppre+s+tmp);
        }
        while(i<j)
        {
```

```
            ppre+=dp[v[i].second][0];
            i++;
        }
         ppre+=dp[v[i].second][0];
    }

}
```

# 数位dp

## 一些说明

**pos**为当前在第几位

**lead** 前导零标记 1表示需要判前导零

如果当前位是0直接跳过，$dfs(pos+1)$

**limit** 限制标记 1表示前面已经取到最高位

不妨计当前最高位为$res$

则下一位的$limit$为$p[i]==res\&\&limit$

$x \mid \exists v \in subtree\,(u), x$

## code

```
//lead 前导零标记 1表示需要判前导零，当前位是0直接跳过
//limit 限制标记 1表示已经取到最高位 ，当前能取到的
//pre 记录前几位的数方便状态转移
// dp初值取-1
ll dp[N];
//从高位到低位
ll dfs(int pos, int pre, int st, ... , int lead, int limit)
{
    if(pos > len) return st; //剪枝
    if(dp[pos][pre][st]...[...] != -1 && !limit && !lead) return  dp[pos][pre]
[st]...[...];
    ll res = 0; //当前的方案数
    int up = limit?a[pos]:9;
    for(int i = 0; i <= up; ++i)
    {
        //需要判前导零并且当前位是0
        if(!i && lead) res += dfs(pos + 1,...,i == res && limit);
        //需要判前导零并且当前位不是0
        else if(i && lead) res +=  dfs(pos + 1,..., 0,i == res && limit);
        else if(其他条件) res += dfs(pos + 1,...,i == res && limit);
    }
    if(!limit && !lead) dp[pos][pre][st]...[...] = res;
    return res;
}
//最高位在0c
ll pre(string s)
{
    dfs();
}
```

# min优化

## 题目

总共$n$个数，要求连续$m$个数，至少取两个的最小代价

## 分析

不同于取一个的的情况，似乎没办法转换。因为我们取数时需要考虑前两个取的位置。

因此我们可以这样写$dp[i][j]$，表示第$i$个数取走，上一个取的数是$j$。

于是有转移方程

$$dp[i][j] = \min_{i-m \le k \le j-1}(dp[j][k]) + a[i]$$

如果$k$的值小于$i-m$，那么就会有一段区间只有$j$而没有其他的。

这样我们的转移时间复杂度是$O(n^2)$的。有一个$n$可以通过取min来优化

考虑到$j$的取值，我们可以将其中一维优化成$O(n*m)$

这样我们的转移方程就可以写成

$$dp[i][j] = minn[i-j][m-j] + a[i]$$

其中$j$表示与上一位的差值，$minn[i][j]$表示第$i$位，前一位与当前位差值不超过$m-j$的最小值

```
    //前m个需要特殊考虑
    for(int i = 2; i <= m; ++i)
    {
        for(int j = i - 1; j; --j)
        {
            dp[pos[i]][i - j] = a[i] + a[j];
            minn[pos[i]][i - j] = min(minn[pos[i]][i - j - 1], dp[pos[i]][i -
j]);
        }
    }

    for(int i = m + 1; i <= n; ++i)
    {
        for(int j = 1; j < m; ++j)
        {
            dp[pos[i]][j] = minn[pos[i - j]][m - j] + a[i];
            minn[pos[i]][j] = min(dp[pos[i]][j], minn[pos[i]][j - 1]);
        }
    }
//当最后m个有两个被选中时，可以作为答案
    ll ans = LINF;
    for(int i = n - m + 1; i <= n; ++i)
    {
        for(int j = i - 1; j >= n - m + 1; --j)
        {
            ans = min(ans, dp[pos[i]][i - j]);
        }
    }
```

```
    cout << ans << '\n';
```

## ST表

### 用途

解决RMQ（区间最大最小值查询）问题

更准确说，是可重贡献问题

即某个数重复计数不会影响最后的结果，常见的还有区间GCD等

### code

```
//注意初值
log = 20
f[M][N]//表示以第N个数为起点，长度为2^j区间中的最大值
logn[N]
void init()
{
    logn[1] = 0;
    logn[2] = 1;
    for(int i = 3; i < N; ++i)
    {
        logn[i] = logn[i / 2] + 1;
    }
    for(int j = 1; j <= log; ++j)
    {
        for(int i = 1; i + (1 << j) - 1 <= n; ++i)
        {
            f[j][i] = max(f[j-1][i], f[j - 1][i + (1 << (j - 1))]);
        }
    }
}
int query(int x, int y)
{
    int s = logn[y - x + 1];
    return max(f[s][x], f[s][y - (1 << s) + 1]);
}
```

# 数据结构

## 泛化前缀和

当且仅当类型$T$和运算$\oplus$满足下面三个条件时，我们可以使用前缀和

1. 存在单位元素$\epsilon$，满足对于任意类型为$T$的元素$a$，有$a \oplus \epsilon = a, \epsilon \oplus a = a$
2. 满足结合率，即$a \oplus (b \oplus c) = (a \oplus b) \oplus c$
3. 存在逆元，即$a \oplus (\backsim a) = \epsilon$

```
template <typename T>
struct PreSum
```

```
{
    vector<T> sum;
    init(vector<T> &a)
    {
        int n = a.size();
        sum.resize(n + 10);
        sum[0] = T_0;   //单位元
        for(int i = 0; i < n; ++i)
        {
            sum[i + 1] = sum[i] + a[i];
        }
    }

    T operator+ (const T &a)const
    {
        return b;   //返回结果
    }

    T inv(T a)
    {
        return b;   //返回a的逆元
    }

    T query(int l, int r) {
        return inv(sum[l - 1]) + sum[r];
        // =   (-(a[0] + ... + a[l-1]))
        //       +   (a[0] + ... + a[l-1])
        //       +   (a[l] + ... + a[r]  )
        // = (a[l] + ... + a[r])

        // 若改为 sum[r] + (-sum[l-1]) 则可能出错，因为+不保证满足交换律
    }
};
```

用树状数组写泛化前缀和，还需要满足交换律

用线段树写泛化前缀和，不需要存在逆元

# 树状数组

```
template <typename T>
struct BIT {
    int n;
    vector<T> a;

    BIT(int n = 0) {
        init(n);
    }

    void init(int _n) {
        this->n = _n;
        a.assign(n, T());
    }

    void add(int x, T v) {
        while(x <= n)
        {
```

```
            a[x] += v;
            x += x & (-x);
        }
    }

    T sum(int x) {
        auto ans = T();
        while(x)
        {
            ans += a[x];
            x -= x & (-x);
        }
        return ans;
    }

    T rangeSum(int l, int r) {
        return sum(r) - sum(l - 1);
    }

    int kth(T k) {
        int x = 0;
        for (int i = 1 << 20; i; i /= 2) {
            if (x + i <= n && k > a[x + i]) {
                x += i;
                k -= a[x];
            }
        }
        return x + 1;
    }
};
```

# 基础线段树

注意开四倍空间

几个可能的优化：

1. 叶子节点可以不用下放$lazy$标记
2. 标记永久化

```
//区间加，区间询问
struct SegTree{
    ll val[N<<2];
    ll lazy[N<<2];
    int n = 0;
    void push_up(int p)
    {
        val[p] = val[p << 1] + val[p << 1 | 1];
    }
    void push_down(int p)
    {
        if(lazy[p])
        {
            int mid = (l + r) >> 1;
            val[p << 1] += lazy[p] * (mid - l + 1);
```

```
            val[p << 1 | 1] += lazy[p] * (r - mid);
            lazy[p << 1] += w;
            lazy[p << 1 | 1] += w;
            lazy[p] = 0;
        }
    }
    void build(int p,int l,int r)
    {
        if(p == 1) n = r;
        if(l==r)
        {
            val[p] = a[l];
            return ;
        }
        int mid = (l + r) >> 1;
        build(p << 1, l, mid);
        build(p << 1 | 1, mid + 1, r);
        push_up(p);
    }
    void add(int p,int ql, int qr, ll w, int l = 1,int r = n)
    {
        if(ql <= l && r <= qr)
        {
            lazy[p] += w;
            val[p] += w * (r - l + 1);
            return ;
        }
        push_down(p, l, r);
        int mid = (l + r) >> 1;
        if(ql <= mid) add(p << 1, ql, qr, l, mid);
        if(qr > mid) add(p <<< 1 | 1, ql, qr, mid + 1, r);
        push_up(p);
    }
    ll query(int p, int ql, int qr, int l = 1, int r = n)
    {
        if(ql <= l && r <= qr)
        {
            return val[p];
        }
        push_down(p, l, r);
        int mid = (l + r) >> 1;
        ll ans = 0;
        if(ql <= mid) ans += query(p << 1, ql, qr, l, mid);
        if(qr > mid) ans += query(p <<< 1 | 1, ql, qr, mid + 1, r);
        return ans;
    }
};
```

# 动态开点线段树

```
int n, cnt, root;//root表示根节点，cnt表示节点个数;
int val[N << 1], ls[N << 1], rs[N << 1];
void update(int& p, int l,int r,int x,int f)
{
    if(!p) p = ++cnt;
    if(l == r)
    {
```

```
        sum[p] += f;
        return ;
    }
    int mid = (l + r) >> 1;
    if(x <= mid) update(ls[p], l, mid. x, f);
    else update(ls[p], l, mid. x, f);
    sum[p] = sum[ls[p]] + sum[rs[p]];
}
int query(int p, int s, int t, int l, int r) {
  if (!p) return 0;  // 如果结点为空，返回 0
  if (s >= l && t <= r) return sum[p];
  int m = s + ((t - s) >> 1), ans = 0;
  if (l <= m) ans += query(ls[p], s, m, l, r);
  if (r > m) ans += query(rs[p], m + 1, t, l, r);
  return ans;
}
```

## 猫树

快速查询区间信息和

要点

1. 建树时维护$(l, mid]$的后缀和以及$(mid, r]$的前缀和
2. 预处理$log$数组
3. 将序列补全至2的幂次
4. $lcp(x, y) = lca(x, y) = x >> log[x^y]$

# 李超树

李超树是在线维护平面线段在整点取值这类问题的结构

抽象的说，应该具有两个功能

1. 加入一条线段
2. 给定数$k$，询问与直线$x = k$相交的线段中，纵坐标最大的线段的标号，坐标相同我们取编号最小的

由于每次询问的点都是整数，因此对于某个区间，我们可以考虑维护最大的线段。

但是线段具有方向性，我们只能保证某一个点只有一个线段是最大的，但是区间上的情况就比较复杂。因此我们利用标记永久化的方法，在询问时，将所有含这个点的线段都找出来然后取最大

时间复杂度，插入为$O(\log^2 n)$，查询为$O(\log n)$。

```
int cmp(double x, double y)
{
    if(fabs(x - y) < eps) return 0;
    if(x - y > eps) return 1;
    return -1;
}
//小于0记得改l[0]的值
struct node
{
    double k,b;
}l[N];
```

```
int tot;
void add(int x0, int y0, int x1, int y1)
{
    tot++;
    if(x0 == x1)
    {
        l[tot].k = 0;
        l[tot].b = max(y0,y1);
    }
    else
    {
        l[tot].k = (double)(y1 - y0) / (x1 - x0);
        l[tot].b = y0 - l[tot].k * x0;
    }
}
double calc(int id,int x)
{
    return l[id].b + l[id].k * x;
}
int s[N << 2];
void udp(int p, int l, int r, int id)
{
    int &v = s[p], mid = (l + r) >> 1;
    int ckmid = cmp(calc(id, mid), calc(v, mid));
    if(ckmid == 1 || (ckmid == 0 && id < v)) swap(v,id);
//  if(l == r) return ;
    int lcheck = cmp(calc(id, l), calc(v, l)), rcheck = cmp(calc(id, r), calc(v,
r));
    if(lcheck == 1 || (lcheck == 0 && id < v)) udp(p << 1, l, mid, id);
    if(rcheck == 1 || (rcheck == 0 && id < v)) udp(p << 1 | 1, mid + 1, r, id);
}
void insert(int p, int l,int r,int ql,int qr, int id)
{
    if(ql <= l && r <= qr)
    {
        udp(p, l, r, id);
        return ;
    }
    int mid = (l + r) >> 1;
    if(ql <= mid) insert(p << 1, l, mid, ql, qr, id);
    if(qr > mid) insert(p << 1 | 1, mid + 1, r, ql, qr, id);
}
pair<double, int> pmax(pair<double, int> x, pair<double, int> y)
{
    int tmp = cmp(x.first, y.first);
    if(tmp == 0) return x.second < y.second ? x : y;
    return tmp > 0 ? x : y;
}
pair<double, int> query(int p, int l, int r, int x)
{
    if(l == r)
    {
        return {calc(s[p], x), s[p]};
    }
    int mid = (l + r) >> 1;
    auto res = make_pair(calc(s[p], x),s[p]);
    if(x <= mid) res = pmax(res, query(p << 1 , l, mid, x));
    if(x > mid) res = pmax(res, query(p << 1 | 1, mid + 1, r, x));
```

```
    return res;
}

//插入线段
if(x0 > x1) swap(x0, x1), swap(y0, y1);
add(x0, y0, x1, y1);
insert(1, 1, p1, x0, x1, tot);
//询问
query(1, 1, p1, x)
```

```
struct Segment
{
    typedef int Int;
    int tot, n;
    struct func{
        Int k, b;
        func(Int k = 0, Int b = 0):k(k), b(b) {}
    }seg[N]; //线段
    vector<int>tag;
    Segment(int n) {
        init(n);
    }
    void init(int _n)
    {
        n = _n;
        tot = 0;
        tag.assign(4 * n + 10, 0);
    }
    void change(Int k, Int b, int ql = 1, int qr = n)
    {
        seg[++tot] = {k, b};
        insert(1, 1, n, ql, qr, id);
    }
    Int calc(int id, int x)
    {
        return seg[id].b + seg[id].k * x;
    }
    int cmp(int id, int y)
    {
        if(x == y) return 0;
        if(x > y) return 1;
        return -1;
    }
    int cmp(double x, double y)
    {
        if(fabs(x - y) < eps) return 0;
        if(x - y > eps) return 1;
        return -1;
    }
    void udp(int p, int l, int r, int id)
    {
        if(!tag[p])
        {
            tag[p] = id;
            return ;
        }
        int &v = tag[p], mid = (l + r) >> 1;
```

```
        int ckmid = cmp(calc(id, mid), calc(v, mid));
        if(ckmid == 1 || (ckmid == 0 && id < v)) swap(v,id);
        int lcheck = cmp(calc(id, l), calc(v, l)), rcheck = cmp(calc(id, r),
calc(v, r));
        if(lcheck == 1 || (lcheck == 0 && id < v)) udp(p << 1, l, mid, id);
        if(rcheck == 1 || (rcheck == 0 && id < v)) udp(p << 1 | 1, mid + 1, r,
id);
    }
    void insert(int p, int l, int r, int ql, int qr, int id) //插入线段斜率截距
    {
        if(ql <= l && r <= qr)
        {
            udp(p, l, r, id);
            return ;
        }
        int mid = (l + r) >> 1;
        if(ql <= mid) insert(p << 1, l, mid, ql, qr, id);
        if(qr > mid) insert(p << 1 | 1, mid + 1, ql, qr, id);
    }
    pair<Int,int> query(int p, int l, int r, int x)
    {
        if(l == r)
        {
            return {calc(tag[p], x), tag[p]};
        }
        int mid = (l + r) >> 1;
        auto res = {calc(tag[p], x), tag[p]};
        if(x <= mid) res = max(res, query(p << 1, l, mid, x));
        if(x > mid) res = max(res, query(p << 1 | 1, mid + 1, r, x));
        return res;
    }
}
```

# 技巧

## 变长bitset

```
template <int len = 1>
ll work(vector<int>&cnt, int n, int va)
{
    if(len <= n)
    {
        return work<min(len * 2, maxn)>(cnt, n, va);//注意maxn也是整数
    }
    //下面是正常代码
    bitset<len>dp;
    dp[0] = 1;
    for(auto x:cnt)
    {
        int w = x, val = a[x];
        for(int i = 1; i <= val; i <<= 1)
        {
            if(x * i <= n)dp |= (dp << (x * i));
            val -= i;
```

```
        }
        if(val)
        {
            if(x * val <= n)dp |= (dp << (x * val));
        }
    }
    ll ans = 0;
    for(int i = n / 2; i >= 0; --i)
    {
        if(dp[i]) return 1ll * i * (s[va] - i);
    }
    return 0;
}
```

# 集合枚举

## 枚举给定集合的子集

```
void solve(int x)
{
    for(int i = x; i; i = (i - 1) & x){

    }
}
```

## 枚举给定集合的超集

```
void solve(int x)
{
    for(int i = x; i < (1 << n); i = (i + 1) | x){

    }
}
```

## 枚举大小为k的子集

```
void solve(int k) {
    for(int i = (1 << k) - 1; i < (1 << n);) {
        print_subset(i);
        int x = i & -i, y = i + x;
        i = ((((i & ~y) / x ) >> 1) | y;
    }
}
```

## 将矩阵某一行或某一列加一最后模$k$为0

给定一个矩阵，每次将某一行或者某一列加一，求模$k$意义下该矩阵变成全0的最小代价。

### 分析

我们不妨假设$r_i$表示第$i$行的操作次数，$c_j$表示的$j$行的操作次数，那么我们要求的就是满足方程组$r_i + c_j + a_{i,j} \equiv 0(\mod k)$的最小解。

因为我们有$n * m$的方程只有$n + m$的未知数，所以有很多方程是冗余的。

首先我们一定有存在$r_i$或者$c_i$为$0$。

不妨假设两者都大于0。

对于$n \geq m$的情况，我们让所有的$r_i := r_i - 1$，$c_i := c_i + 1$，此时我们减少了$n - m$次操作，但是矩阵和之前一样。

对于$n < m$ 的情况，同理可得。

于是至少有一个$r_i$或者$c_I$等于0l。

于是去枚举某一行某一列为0的情况。

假设我们让$r_i = 0$那么对于列来说$c_j = -a_{i,j}(\mod k)$。

然后我们就可以求出所有的$r_i$和$c_i$了。

代码如下

```
bool check()
{
    for(int i = 1; i <= n; ++i)
    {
        for(int j= 1; j <= n; ++j)
        {
            if((a[1][1] + a[i][j]) % k != (a[1][j] + a[i][1]) % k) return 0;
        }
    }
}

ll calc1(int x)
{
    ll ans = 0;
    ll c_1 = (k - a[x][1]) % k;
    for(int i = 1; i <= n; ++i) ans += (k - a[x][i]) % k;
    for(int i = 1; i <= m; ++i) ans += (- c_1 - a[i][1] + 2 * k) % k;
    return ans;
}

ll calc2(int x)
{
    ll ans = 0;
    ll r_1 = (k - a[1][x]) % k;
    for(int i = 1; i <= m; ++i) ans += (k - a[i][x]) % k;
    for(int i = 1; i <= n; ++i) ans += (- r_1 - a[1][i] + 2 * k) % k;
    return ans;
}
```

**对于零一矩阵来说，翻转操作等价于加一操作**

# 求两个区间集合的最大交

```
    sort(a.begin(), a.end());
    sort(b.begin(), b.end());
    int n1 = a.size(), n2 = b.size();
    ll ans = 0;
    for(int i = 0, j = 0; i < n1 && j < n2;)
    {
        ll l1 = a[i].first, r1 = a[i].second;
        ll l2 = b[j].first, r2 = b[j].second;
```

```
        if(r2 >= l1 && r1 >= l2)
        {
            ans = max(ans, min(r1, r2) - max(l1, l2));
        }
        if(r2 < r1) j++;
        else i++;
    }
    cout << ans << '\n';
```

# 区间合并

```
 vector<vector<int>> merge(vector<vector<int>>& intervals) {
        sort(intervals.begin(), intervals.end());
        vector<vector<int>>ans;
        int st = INT_MIN, ed = INT_MIN;
        for(auto v:intervals)
        {
            if(ed == INT_MIN)
            {
                st = v[0];
                ed = v[1];
            }
            else if(v[0] <= ed)
            {
                ed = max(v[1], ed);
            }
            else if(v[0] > ed)
            {
                ans.push_back({st, ed});
                st = v[0];
                ed = v[1];
            }
        }
        ans.push_back({st, ed});
        return ans;
}
```

# 最多不重叠区间

```
int eraseOverlapIntervals(vector<vector<int>>& intervals) {
        sort(intervals.begin(), intervals.end(), [](const vector<int> &x, const
vector<int> &y) {
            return x[1] < y[1];  // 按照右端点从小到大排序
        });

        int res = 0, ed = INT_MIN;
        for (auto v: intervals) {
            if (ed <= v[0]) {
                res++;
                ed = v[1];
            }
        }

        return intervals.size() - res;
    }
```

## 区间点覆盖

等价于最多不重叠区间

# 计算几何

# 博弈论

## 二分图博弈

### 定义

**二分图博弈**是一类博弈模型（也是公平组合游戏），它可以被抽象为：给出一张**二分图**和**起始点**$H$，A和B轮流操作，每次只能选与上个被选择的点（第一回合则是点 $H$ ）**相邻**的点，且不能选择**已选择过**的点，**无法选点**的人输掉。一个经典的二分图博弈模型是在国际象棋棋盘上，双方轮流移动一个士兵，不能走已经走过的格子，问谁先无路可走。

### 结论

考虑二分图的最大匹配，如果最大匹配**一定**包含$H$，那么先手必胜，否则后手必胜。

### 证明

先手只需要沿着匹配边走即可。

如果最大匹配一定包含$H$，那么先手只需沿着匹配边操作，后手无论如何都会选到匹配点，否则我们将这条路径上的匹配状态取反，此时匹配数不变但是不包含$H$。更具体的说，我们按照上面的说法会形成$\{H \to^{\text{匹配边}} P_0 \to^{\text{非匹配边}} P_1 \to \cdots \to^{\text{非匹配边}} P_n\}$，然后我们就可以将边的状态取反，此时不包括$H$。这与假设不符。

如果最大匹配不一定包含$H$，那么先手必定选到匹配点，后手也选相应的匹配点即可，否则就有一条增广路，这与选点是最大匹配矛盾。

### 运用

然后我们就可以用网络流或者匈牙利去跑。可以在建图时涉及$H$的边先不管，先跑一次看看结果，再将边放进去看看结果是否变大。

### 题目

你有三个不同的棋子在长度为$n$数轴上，将三个棋子当前所在位置记为一个三元组$(x,y,z)$，每次可以移动其中一个棋子到旁边的位置，即$+1$或$-1$。要求出现的三元组不能重复，先不能操作的人判输。

### 分析

将三元组看成三维坐标上的点，这时我们会发现和黑白棋子二分匹配一样，$(x+y+z)$是奇数还是偶数就可以划分二部点，那么当$n$是偶数时，显然最大匹配就是完美匹配。$n$是奇数时，会有一个和为奇数的点没有办法匹配到，因此和为奇数先手必败否则必胜

## 公平组合游戏

## 定义

如果一个游戏满足：

1. 有两名玩家交替行动
2. 在游戏进程的任意时刻，可以执行的合法行动与轮到哪名玩家无关
3. 不能行动的玩家判负

那么该游戏就是公平组合游戏

## 有向图游戏

给定一个有向无环图，图中有一个唯一的起点，在起点上放一枚棋子。两名玩家交替移动棋子，无法移动者判负。

**任何一个**公平组合游戏都可以转化为有向图游戏。可以将每个局面看成一个节点。

## $mex$运算

我们定义

$$mex(S) = min_{x \in N, x \notin S} x$$

## $SG$函数

一个状态的$sg$函数是这个状态的一个表示。

我们定义

$$SG(x) = mex\{sg(y)\}$$

其中$y$是$x$的后继状态.

有向图游戏的某个局面必胜当且仅当该局面对应节点的$SG$函数大于$0$

有向图游戏的某个局面必败当且仅当该局面对应节点的$SG$函数等于$0$

我们可以这样打表

```
for(int i = 1; i <= n; ++i)
{
    memset(vis, 0, sizeof(vis));
    for(int y:nxt[i])   //y表示i的下一个局面
        vis[sg[y]] = 1;
    for(int j = 0; j <= n; ++j)
    {
        if(!vis[j])
        {
            sg[i] = j;
            break;
        }
    }
}
```

### 有向图游戏的和

$$SG(G) = SG(G_1) \wedge SG(G_2) \wedge \cdots \wedge SG(G_m)$$

# 常见博弈

## bash博弈

$n$个石子，每次可以取$1-m$颗石子

### 必败必胜

$n \mod (m+1) == 0$先手必败，否则先手必胜

当$n \leq m$时，甲可以一次性拿完。先手必胜

$n = m+1$时，甲无论如何都拿不完，乙可以在甲操作完后拿完，先手必败。

$n$是$m+1$的倍数时，无论怎么拿都会破坏这个状态，后者总可以回到这个状态。

### 变种

每次我们只能取$\{a_1, a_2, \cdots, a_k\}$这个集合中的石子数

通常我们可以打表找规律，也可以$sg$函数判断，一般来说都有周期性。

## NIM博弈

有$n$堆石子，每次从某一堆中取任意数量的石子。

### 必败

当$a_1 \oplus a_2 \oplus a_3 \oplus \cdots \oplus a_n == 0$，先手必败，否则先手必胜

我们一定可以从必胜态找到一个必败态的后继，而必败态找不到必败态的后继。

## 威佐夫博弈

有两堆石子，每次可以从一堆中拿任意数量的石子或者从两堆中拿走相同数量的石子。

### 必败

当且仅当当前局势为奇异局势时必败

# 字符串

## 后缀自动机

```
#include<bits/stdc++.h>
using namespace std;
const int MAXLEN=1e5+10;
struct SAIS
{
    #define L_Type 0
```

```
#define S_Type 1
//rl【i】表示排序后 第i个后缀的第一个字符在原串中的位置
//rk【i】表示 第i个字符作为后缀第一个字符的排名
//lcp【i】表示 排序后相邻两个后缀的最大公共前缀
int st[MAXLEN];
int rl[MAXLEN],rk[MAXLEN],lcp[MAXLEN];
int n;
void init(const string &s){
    n=s.size();
    for(int i=1;i<=n;++i)st[i]=s[i-1];
    //st中的字符必须调成正数
}
//判断是否为LMS字符
inline bool is_lms_char(int *type,int x){

    return x>1&&type[x]==S_Type&&type[x-1]==L_Type;
}
//判断两个LMS子串是否相同
inline bool equal_substring(int *S,int x,int y,int *type){

    do
    {
        if(S[x]!=S[y])return false;
        x++,y++;
    }while(!is_lms_char(type,x)&& !is_lms_char(type,y));
    return S[x]==S[y]&&type[x]==type[y];
}
//诱导排序
inline void induced_sort(int *S,int *SA,int *type,int *bucket,
            int *lbucket,int *sbucket, int n, int SIGMA) {

    for(int i=1;i<=n;++i)
    {
        if(SA[i]>1&&type[SA[i]-1]==L_Type)
            SA[lbucket[S[SA[i]-1]]++]=SA[i]-1;
    }
    for (int i = 0; i <= SIGMA; ++i)  // Reset S-type bucket
        sbucket[i] = bucket[i];
    for (int i = n; i >= 1; i--)
        if (SA[i] > 1 && type[SA[i] - 1] == S_Type)
            SA[sbucket[S[SA[i] - 1]]--] = SA[i] - 1;
}
// SA-IS主体
// S是输入字符串，length是字符串的长度，SIGMA是字符集的大小
int *sais(int *S,int Length,int SIGMA){

    int n=Length;
    assert(S[n]==0);
    int *type = new int[n + 5];  // 后缀类型
    int *position = new int[n + 5];  // 记录LMS子串的起始位置
    int *name = new int[n + 5];  // 记录每个LMS子串的新名称
    int *SA = new int[n + 5];  // SA数组
    int *bucket = new int[SIGMA + 5];  // 每个字符的桶
    int *lbucket = new int[SIGMA + 5];  // 每个字符的L型桶的起始位置
    int *sbucket = new int[SIGMA + 5];  // 每个字符的S型桶的起始位置
    memset(bucket, 0, sizeof(int) * (SIGMA + 5));
    for (int i = 1; i <= n; i++)
        bucket[S[i]]++;
```

```
        for (int i = 0; i <= SIGMA; i++)
        {
            if (i==0)
            {
//?              ?//bucket[i] = bucket[i];
                lbucket[i] = 1;
            }else
            {
                bucket[i] += bucket[i - 1];
                lbucket[i] = bucket[i - 1] + 1;
            }
            sbucket[i] = bucket[i];
        }

        // 确定后缀类型(利用引理2.1)
        type[n] = S_Type;
        for (int i = n - 1; i >= 1; i--)
        {
            if (S[i] < S[i + 1])
                type[i] = S_Type;
            else if (S[i] > S[i + 1])
                type[i] = L_Type;
            else
                type[i] = type[i + 1];
        }
        // 寻找每个LMS子串
        int cnt = 0;
        for (int i = 1; i <= n; i++)
            if (is_lms_char(type,i))
                position[++cnt] = i;
        // 对LMS子串进行排序
        fill(SA, SA + n + 3, -1);
        for (int i = 1; i <= cnt; i++)
            SA[sbucket[S[position[i]]]--] = position[i];
        induced_sort(S, SA, type, bucket, lbucket, sbucket, n, SIGMA);

        // 为每个LMS子串命名
        fill(name, name + n + 3, -1);
        int lastx = -1, namecnt = 1;  // 上一次处理的LMS子串与名称的计数
        bool flag = false;  // 这里顺便记录是否有重复的字符
        for (int i = 2; i <= n; i++)
        {
            int x = SA[i];

            if (is_lms_char(type, x))
            {
                if (lastx >= 0 && !equal_substring(S, x, lastx, type))
                    namecnt++;
                // 因为只有相同的LMS子串才会有同样的名称
                if (lastx >= 0 && namecnt == name[lastx])
                    flag = true;

                name[x] = namecnt;
                lastx = x;
            }
        }  // for
        name[n] = 0;
```

```
        // 生成S1
        int *S1 = new int[cnt+5];
        int pos = 0;
        for (int i = 1; i <= n; i++)
            if (name[i] >= 0)
                S1[++pos] = name[i];
        int *SA1;
        if (!flag)
        {
            // 直接计算SA1
            SA1 = new int[cnt + 5];
            for (int i = 1; i <= cnt; i++)
                SA1[S1[i]+1] = i;
        }
        else
            SA1 = sais(S1, cnt, namecnt);  // 递归计算SA1

        // 从SA1诱导到SA
        for (int i = 0; i <= SIGMA; i++)
        {
            if (i==0)
                lbucket[i] = 1;
            else
                lbucket[i] = bucket[i - 1] + 1;
            sbucket[i] = bucket[i];
        }
        fill(SA, SA + n + 3, -1);
        for (int i = cnt; i >= 1; i--)  // 这里是逆序扫描SA1，因为SA中S型桶是倒序的
            SA[sbucket[S[position[SA1[i]]]]--] = position[SA1[i]];
        induced_sort(S, SA, type, bucket, lbucket, sbucket, n, SIGMA);

        delete[] S1;
        delete[] SA1;
        delete[] bucket;
        delete[] lbucket;
        delete[] sbucket;
        delete[] position;
        delete[] type;
        delete[] name;
        // 后缀数组计算完毕
        return SA;
    }
    void build(){

        st[0]=st[n+2]=-1;
        st[n+1]=0;
        int SIGMA = 0;
        for (int i=1;i<=n;++i) SIGMA = max(SIGMA,st[i]);
        int * sa = sais(st,n+1,SIGMA);
        for (int i=2;i<=n+1;++i) rk[sa[i]]=i-1;
        delete[] sa;
        for (int i=1;i<=n;++i) rl[rk[i]]=i;
        for (int i=1,len=0;i<=n;++i)
        {
            if (len) --len;
            while (i+len<=n&&rl[rk[i]-1]+len<=n
&&st[i+len]==st[rl[rk[i]-1]+len]) ++len;
            lcp[rk[i]]=len;
```

```
        }
    }
    #undef L_TYPE
    #undef R_TYPE
}sa;
int main()
{
    ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    int n,q;cin>>n>>q;
    string s;
    cin>>s;
    sa.init(s);
    sa.build();
//  for (int i=1;i<=sa.n;++i) printf("%d%c",sa.rl[i]," \n"[i==sa.n]);
//    for (int i=1;i<=sa.n;++i) printf("%d%c",sa.lcp[i]," \n"[i==sa.n]);
    while(q--)
    {
        string ss;cin>>ss;
        int len=ss.size();
        int l=1,r=n;
        while(l<=r)
        {
            int mid=(l+r)>>1;
            //cout<<s.substr(sa.rl[mid],min(len,n-sa.rl[mid]+1))<<'\n';
            if(s.substr(sa.rl[mid],min(len,n-sa.rl[mid]+1))>=ss)r=mid-1;
            else l=mid+1;
        }
        int k1=r;
        cout<<k1<<'\n';
        l=1,r=n;
        while(l<=r)
        {
            int mid=(l+r)>>1;
            if(s.substr(sa.rl[mid],min(len,n-sa.rl[mid]+1))<=ss)r=mid-1;
            else l=mid+1;
        }
        int k2=l;
        cout<<k2-k1<<'\n';
    }
    return 0;
}
```

# KMP，扩展KMP

## 前缀函数

$\pi[i]$最长$s$的真前缀和真后缀($最后一位是s[i]$)匹配的长度

$$\pi[0]=0;\pi[i]=\max_{k=0\cdots i}\{s[0\cdots k-1]=s[i-(k-1)\cdots i]\}$$

```
vector<int> prefix_function(string s) {
  int n = (int)s.length();
  vector<int> pi(n, 0);
  for (int i = 1; i < n; i++) {
    int j = pi[i - 1];
    while (j > 0 && s[i] != s[j]) j = pi[j - 1];
    if (s[i] == s[j]) j++;
    pi[i] = j;
  }
  return pi;
}
```

## 查找$s$在$t$中的所有出现

```
vector<int> find_occurrences(string text, string pattern) {
  string cur = pattern + '#' + text;
  int sz1 = text.size(), sz2 = pattern.size();
  vector<int> v;
  vector<int> lps = prefix_function(cur);
  for (int i = sz2 + 1; i <= sz1 + sz2; i++) {
    if (lps[i] == sz2)
      v.push_back(i - 2 * sz2);
  }
  return v;
}
```

## 自动机

```
void compute_automaton(string s, vector<vector<int>>& aut) {
  s += '#';
  int n = s.size();
  vector<int> pi = prefix_function(s);
  aut.assign(n, vector<int>(26));
  for (int i = 0; i < n; i++) {
    for (int c = 0; c < 26; c++) {
      if (i > 0 && 'a' + c != s[i])
        aut[i][c] = aut[pi[i - 1]][c];
      else
        aut[i][c] = i + ('a' + c == s[i]);
    }
  }
}
```

## z函数

$z[i]$表示$s$和$s[i, n-1]$的最长公共前缀

```
vector<int> z_function(string s) {
  int n = (int)s.length();
  vector<int> z(n,0);
  for (int i = 1, l = 0, r = 0; i < n; ++i) {
    if (i <= r && z[i - l] < r - i + 1) {
      z[i] = z[i - l];
    } else {
```

```
      z[i] = max(0, r - i + 1);
      while (i + z[i] < n && s[z[i]] == s[i + z[i]]) ++z[i];
    }
    if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
  }
  return z;
}
```

# Hash

```
const ll base1=1331,base2=311,mod1=1e9+7,mod2=1000000000000002493;
inline ll ksc(ull x,ull y ,ll p){return (x*y-(ull)((ld)x/p*y)*p+p)%p;}
map<pair<ull,ull>,int>h;
ull hash[N];
void Hash()
{
    hash[0]=(ull)1;
    for(int i=1;i<N-10;++i)hash[i]=ksc(hash[i-1],base1,mod1);
}
```

```
namespace Hash
{
    const ull base1 = 1331, base2 = 31;
    const ull mod1 = 1e9 + 7, mod2 = 100000000000000249311;
    ull p;
    inline ll ksc(ull x,ull y ,ll p){return (x*y-(ull)((long
double)x/p*y)*p+p)%p;}
//  map<pair<ull,ull>,int>h;
    vector<ull> power, h;
    int n;
    void init(string const& s, const ull _p = mod2)
    {
        p = _p;
        n = s.size();
        power.resize(n + 5);
        h.resize(n + 5, 0);
        power[0]=(ull)1;
        for(int i = 1; i < n; ++i)
        {
            power[i]=ksc(power[i-1],base1,p);
        }
        for(int i = 0; i < n; ++i)
        {
            h[i + 1] = (ksc(h[i], base1, p) + s[i]) % p;
        }
    }
    ull get_hash(int l, int r)
    {
        l++, r++;
        ull t = h[r] + p - ksc(h[l - 1], power[r - l + 1], p);
        return (t % p + p) % p;
    }

};

using Hash::init;
```

```
using Hash::get_hash;
```

```
namespace Hash
{
//  const ull base1 = 1331, base2 = 31;
    const ull mod1 = 1000000007, mod2 = 10000000000000002493ll;
    const int hash_cnt = 2;
    vector<ull>p(hash_cnt), base(hash_cnt);
    inline ll ksc(ull x,ull y ,ull p){return (x*y-(ull)((long
double)x/p*y)*p+p)%p;}
//  map<pair<ull,ull>,int>h;
    vector<ull> power[hash_cnt], h[hash_cnt];
    int n;
    void init(string const& s)
    {
        base[0] = 233;
        base[1] = 1331;
        p[0] = mod1;
        p[1] = mod2;
        n = s.size();
        for(int j = 0; j < hash_cnt; ++j)
        {
            power[j].resize(n + 5);
            h[j].resize(n + 5, 0);
            power[j][0]=(ull)1;
            for(int i = 1; i <= n; ++i)
            {
                power[j][i]=ksc(power[j][i-1], base[j], p[j]);
            }
            for(int i = 1; i <= n; ++i)
            {
                h[j][i] = (ksc(h[j][i - 1], base[j], p[j]) + s[i - 1]) % p[j];
            }
        }
    }
    vector<ull> get_hash(int l, int r)
    {
        l++, r++;
        vector<ull>ans(hash_cnt);
        for(int j = 0; j < hash_cnt; ++j)
        {
            ll t = h[j][r] + p[j] - ksc(h[j][l - 1], power[j][r - l + 1], p[j]);
            ans[j] = (t % p[j] + p[j]) % p[j];
        }
        return ans;
    }

};
using Hash::init;
using Hash::get_hash;
```