

Содержание

| | |
|--------------------------------------|-----------|
| 1 Теория чисел | 1 |
| 1.1 КТО | 1 |
| 1.2 Алгоритм Миллера — Рабина | 1 |
| 1.3 Алгоритм Берлекэмпа — Мессе | 1 |
| 2 Графы | 2 |
| 2.1 SCC и 2-SAT | 2 |
| 2.2 Эйлеров цикл | 2 |
| 2.3 Компоненты рёберной двусвязности | 2 |
| 2.4 DCP offline | 3 |
| 3 Свёртки | 3 |
| 3.1 AND, OR, XOR свёртки | 3 |
| 3.2 NTT & co | 3 |
| 3.3 старое доброе FFT | 4 |
| 4 Структуры данных | 5 |
| 4.1 Дерево Фенвика | 5 |
| 4.2 Дерево отрезков | 5 |
| 4.3 Битовый бор | 6 |
| 4.4 Ordered set | 7 |
| 4.5 Convex hull trick | 7 |
| 4.6 Центроиды | 7 |
| 5 Строковые алгоритмы | 7 |
| 5.1 Префикс-функция | 7 |
| 5.2 Z-функция | 7 |
| 5.3 Алгоритм Манакера | 8 |
| 5.4 Суфмассив | 8 |
| 5.5 Алгоритм Ахо — Корасик | 9 |
| 5.6 Алгоритм Ахо Корасик | 9 |
| 5.7 Дерево палиндромов | 9 |
| 5.8 Дерево палиндромов | 9 |
| 6 Потoki | 9 |
| 6.1 Алгоритм Диница | 9 |
| 6.2 Mincost k-flow | 10 |
| 7 Геометрия | 11 |
| 7.1 Примитивы | 11 |
| 7.2 Выпуклая оболочка | 11 |
| 7.3 Точка внутри многоугольника | 11 |
| 7.4 Касательные | 11 |
| 8 Разное | 11 |
| 8.1 Флаги копирования | 11 |
| 8.1.1 Сеточка в vim | 11 |
| 8.2 Что сделать на пробном туре | 11 |
| 8.3 Шаблон | 12 |

1 Теория чисел

1.1 КТО

```
int gcd(int a, int b, int &x, int &y) {
    if (b==0) { x = 1; y = 0; return a; }
    int d = gcd(b, a%b, y, x);
    y -= a/b*x;
    return d;
}

int inv(int r, int m) {
    int x, y;
    gcd(r, m, x, y);
    return (x+m)%m;
}

int crt(int r, int n, int c, int m) { return r + ((c -
    r) % m + m) * inv(n, m) % m * n; }
```

1.2 Алгоритм Миллера — Рабина

```
__int128 one=1;
int po(int a, int b, int p)
{
    int res=1;
    while(b) {if(b & 1) {res=(res*one*a)%p;--b;} else {a
        =(a*one*a)%p;b>>=1;}} return res;
}

bool chprime(int n) ///miller-rabin
{
    if(n==2) return true;
    if(n<=1 || n%2==0) return false;
    int h=n-1;int d=0;while(h%2==0) {h/=2;++d;}
    for(int a:{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31,
        37})
    {
        if(a==n) return true;
        int u=po(a,h,n);bool ok=0;
        if(u%n==1) continue;
        for(int c=0;c<d;++c)
        {
            if((u+1)%n==0) {ok=1;break;}
            u=(u*one*u)%n;
        }
        if(!ok) return false;
    }
    return true;
}
```

1.3 Алгоритм Берлекэмпа — Мессе

<https://mzhang2021.github.io/cp-blog/berlekamp-massey/>

```
template<typename T>
vector<T> berlekampMassey(const vector<T> &s) {
    int n = s.size(), l = 0, m = 1;
    vector<T> b(n), c(n);
    T ld = b[0] = c[0] = 1;
    for (int i=0; i<n; i++, m++) {
        T d = s[i];
        for (int j=1; j<=l; j++)
            d += c[j] * s[i-j];
        if (d == 0) continue;
        vector<T> temp = c;
        T coef = d / ld;
        for (int j=m; j<n; j++) c[j] -= coef * b[j-m];
        if (2 * l <= i) {
            l = i + 1 - l;
            b = temp;
            ld = d;
            m = 0;
        }
    }
    c.resize(l + 1);
    c.erase(c.begin());
    for (T &x : c)
        x = -x;
    return c;
}
```

2 Графы

2.1 SCC и 2-SAT

Алгоритм ищет сильносвязные компоненты в графе g , если есть путь $i \rightarrow j$, то $scc[i] \leq scc[j]$

В случае 2-SAT рёбра $i \Rightarrow j$ и $(j \oplus 1) \Rightarrow (i \oplus 1)$ должны быть добавлены одновременно.

```
vector<vector<int>> g(2 * n);
vector<vector<int>> r(g.size());
for (int i = 0; i < g.size(); ++i) {
    for (int j : g[i]) r[j].push_back(i);
}
vector<int> used(g.size()), tout(g.size());
int time = 0;
auto dfs = [&](auto dfs, int cur) -> void {
    if (used[cur]) return;
    used[cur] = 1;
    for (int nxt : g[cur]) {
        dfs(dfs, nxt);
    }
    // used[cur] = 2;
    tout[cur] = time++;
};
for (int i = 0; i < g.size(); ++i) if (!used[i]) dfs(dfs, i);
vector<int> ind(g.size());
iota(ind.begin(), ind.end(), 0);
sort(all(ind), [&](int i, int j){return tout[i] > tout[j]});
vector<int> scc(g.size(), -1);
auto go = [&](auto go, int cur, int color) -> void {
    if (scc[cur] != -1) return;
    scc[cur] = color;
    for (int nxt : r[cur]) {
        go(go, nxt, color);
    }
};
int color = 0;
for (int i : ind) {
    if (scc[i] == -1) go(go, i, color++);
}
for (int i = 0; i < g.size() / 2; ++i) {
    if (scc[2 * i] == scc[2 * i + 1]) "IMPOSSIBLE"
    if (scc[2 * i] < scc[2 * i + 1]) {
        // !i => i, assign i = true
    } else {
        // i => !i, assign i = false
    }
}
```

2.2 Эйлеров цикл

```
vector<int> euler(vector<vector<pair<int, int>>> g,
    int src) { // g[cur][i] = pair{nxt, idx}
    int n = g.size();
    vector<int> used(m), it(n), cycle;
    auto dfs = [&](auto dfs, int cur) -> void {
        while (true) {
            while (it[cur] < g[cur].size() && used[g[cur][it[cur]].second]) it[cur]++;
            if (it[cur] == g[cur].size()) return;
            auto [nxt, idx] = g[cur][it[cur]];
            used[idx] = true;
            dfs(dfs, nxt);
            cycle.push_back(idx); // or {cur, nxt}
        }
    };
    dfs(dfs, src);
    reverse(cycle.begin(), cycle.end());
    if (cycle.size() != m) return {}; // check that all
    edges are present in the cycle, fail otherwise
    return cycle;
}
```

2.3 Компоненты рёберной двусвязности

```
int n, m;
cin >> n >> m;
vector<vector<int>> g(n + 1);
map<pair<int, int>, int> comp, col;
for (int i = 0; i < m; ++i) {
    int u, v, c; cin >> u >> v >> c; c--;
    col[{u,v}] = col[{v,u}] = c;
    g[u].push_back(v);
    g[v].push_back(u);
}
vector<int> used(n + 1);
vector<int> newCompWithoutParent(n + 1), h(n + 1), up
(n + 1);
auto findCutPoints = [&](auto self, int u, int p) ->
void {
    used[u] = 1;
    up[u] = h[u];
    for (int v : g[u]) {
        if (!used[v]) {
            h[v] = h[u] + 1;
            self(self, v, u);
            up[u] = min(up[u], up[v]);
            if (up[v] >= h[u]) {
                newCompWithoutParent[v] = 1;
            }
        }
        else {
            up[u] = min(up[u], h[v]);
        }
    }
};
for (int u = 1; u <= n; ++u) {
    if (!used[u]) {
        findCutPoints(findCutPoints, u, u);
    }
}
int ptr = 0;
vector<map<int, int>> colors(m);
auto markComponents = [&](auto self, int u, int cur)
-> void {
    used[u] = 1;
    for (int v : g[u]) {
        if (!used[v]) {
            if (newCompWithoutParent[v]) {
                ptr++;
                self(self, v, ptr - 1);
            }
            else {
                self(self, v, cur);
            }
        }
        else if (h[v] < h[u]) {
            comp[{u,v}] = comp[{v,u}] = cur;
            int c = col[{u,v}];
            colors[cur][u] |= 1 << c;
            colors[cur][v] |= 1 << c;
        }
    }
};
used.assign(n + 1, 0);
for (int u = 1; u <= n; ++u) {
    if (!used[u]) {
        markComponents(markComponents, u, -1);
    }
}
for (int comp = 0; comp < m; ++comp) {
    vector<int> cnt(4);
    int tot = 0;
    for (auto [u, mask] : colors[comp]) {
        tot |= mask;
        cnt[bp(mask)]++;
    }
    if (bp(tot) < 3) {
        continue;
    }
    if (cnt[2] || cnt[3] > 2) {
        cout << "Yes" << endl;
        return;
    }
}
```

```
cout << "No" << endl;
```

2.4 DCP offline

```
struct Dsu {
    int n;
    vector<pair<int &, int>> s;
    vector<int> p, sz;
    // other info

    Dsu(int n) : n(n), p(n), sz(n, 1) {
        iota(all(p), 0);
    }

    int get(int u) {
        while (u != p[u]) u = p[u];
        return u;
    }

    bool merge(int u, int v) {
        u = get(u), v = get(v);
        if (u == v) return false;
        if (sz[v] < sz[u]) swap(u, v);
        s.app({p[u], p[u]});
        s.app({sz[v], sz[v]});
        // app other info like s.app({comp, comp});
        p[u] = v;
        sz[v] += sz[u];
        return true;
    }

    void rollback(int sz) {
        while (s.size() != sz) {
            s.back().first = s.back().second;
            s.pop_back();
        }
    }
};

struct DcpOffline {
    int n;
    vector<vector<pair<int, int>>> d;

    void addEdgeOnSegment(int l, int r, int a, int b)
    {
        for (l += n, r += n; l < r; l /= 2, r /= 2) {
            if (l & 1) d[l++].app({a, b});
            if (r & 1) d[--r].app({a, b});
        }
    }

    template<typename T>
    void dfs(Dsu &dsu, T act) {
        dfs(1, 0, n, dsu, act);
    }

    template<typename T>
    void dfs(int v, int l, int r, Dsu &dsu, T act) {
        int sz = dsu.s.size();
        for (auto [u, v]: d[v]) {
            dsu.merge(u, v);
        }
        if (l + 1 == r) {
            act(l, dsu);
        } else {
            int m = (l + r) / 2;
            dfs(v * 2, l, m, dsu, act);
            dfs(v * 2 + 1, m, r, dsu, act);
        }
        dsu.rollback(sz);
    }

    DcpOffline(int maxt) : n(2 << __lg(maxt + 1)), d(2
        * n) {}
};
```

3 Свёртки

3.1 AND, OR, XOR свёртки

```
const int p = 998244353;
vector<int> band(vector<int> a, vector<int> b)
{
    int n=0; while((1<<n)<a.size()) ++n;
    a.resize(1<<n); b.resize(1<<n);
    for(int i=0; i<n; ++i) for(int mask=0; mask<(1<<n); ++
        mask) if(mask & (1<<i)) {a[mask-(1<<i)]+=a[mask]; a
        [mask-(1<<i)]%=p;}
    for(int i=0; i<n; ++i) for(int mask=0; mask<(1<<n); ++
        mask) if(mask & (1<<i)) {b[mask-(1<<i)]+=b[mask]; b
        [mask-(1<<i)]%=p;}
    vector<int> c(1<<n, 0);
    for(int mask=0; mask<(1<<n); ++mask) {c[mask]=a[mask]*
        b[mask]; c[mask]%=p;}
    for(int i=0; i<n; ++i) for(int mask=0; mask<(1<<n); ++
        mask) if(!(mask & (1<<i))) {c[mask]-=c[mask+(1<<i)
        ]; c[mask]%=p;}
    return c;
}

vector<int> bor(vector<int> a, vector<int> b)
{
    int n=0; while((1<<n)<a.size()) ++n;
    a.resize(1<<n); b.resize(1<<n);
    for(int i=0; i<n; ++i) for(int mask=0; mask<(1<<n); ++
        mask) if(!(mask & (1<<i))) {a[mask+(1<<i)]+=a[mask
        ]; a[mask+(1<<i)]%=p;}
    for(int i=0; i<n; ++i) for(int mask=0; mask<(1<<n); ++
        mask) if(!(mask & (1<<i))) {b[mask+(1<<i)]+=b[mask
        ]; b[mask+(1<<i)]%=p;}
    vector<int> c(1<<n, 0);
    for(int mask=0; mask<(1<<n); ++mask) {c[mask]=a[mask]*
        b[mask]; c[mask]%=p;}
    for(int i=0; i<n; ++i) for(int mask=0; mask<(1<<n); ++
        mask) if(mask & (1<<i)) {c[mask]-=c[mask-(1<<i)]; c
        [mask]%=p;}
    return c;
}

vector<int> bxor(vector<int> a, vector<int> b)
{
    assert(p%2==1); int inv2=(p+1)/2;
    int n=0; while((1<<n)<a.size()) ++n;
    a.resize(1<<n); b.resize(1<<n);
    for(int i=0; i<n; ++i) for(int mask=0; mask<(1<<n); ++
        mask) if(!(mask & (1<<i))) {int u=a[mask], v=a[mask
        +(1<<i)]; a[mask+(1<<i)]=(u+v)%p; a[mask]=(u-v)%p;}
    for(int i=0; i<n; ++i) for(int mask=0; mask<(1<<n); ++
        mask) if(!(mask & (1<<i))) {int u=b[mask], v=b[mask
        +(1<<i)]; b[mask+(1<<i)]=(u+v)%p; b[mask]=(u-v)%p;}
    vector<int> c(1<<n, 0);
    for(int mask=0; mask<(1<<n); ++mask) {c[mask]=a[mask]*
        b[mask]; c[mask]%=p;}
    for(int i=0; i<n; ++i) for(int mask=0; mask<(1<<n); ++
        mask) if(!(mask & (1<<i))) {int u=c[mask], v=c[mask
        +(1<<i)]; c[mask+(1<<i)]=((v-u)*inv2)%p; c[mask]=((u
        +v)*inv2)%p;}
    return c;
}
```

3.2 NTT & co

```
typedef long long ll;
const int p=998244353;
int po(int a, int b) {if(b==0) return 1; if(b==1)
    return a; if(b%2==0) {int u=po(a, b/2); return (u*1
    LL*u)%p;} else {int u=po(a, b-1); return (a*1LL*u)%p
    ;}}
int inv(int x) {return po(x, p-2);}
template<int M, int K, int G> struct Fft {
    // 1, 1/4, 1/8, 3/8, 1/16, 5/16, 3/16, 7/16, ...
    int g[1 << (K - 1)];
    Fft() : g() { //if t1 constexpr...
        static_assert(K >= 2, "Fft: K >= 2 must hold");
        g[0] = 1;
        g[1 << (K - 2)] = G;
        for (int l = 1 << (K - 2); l >= 2; l >= 1) {
            g[l >> 1] = (g[l] * 1LL * g[l]) % M;
        }
        assert((g[1]*1LL * g[1]) % M == M - 1);
        for (int l = 2; l <= 1 << (K - 2); l <= 1) {
            for (int i = 1; i < l; ++i) {
                g[l + i] = (g[l] * 1LL * g[i]) % M;
            }
        }
    }
};
```

```

    }
}
}
void fft(vector<int> &x) const {
    const int n = x.size();
    assert(n <= 1 << K);
    for (int h = __builtin_ctz(n); h--; ) {
        const int l = (1 << h);
        for (int i = 0; i < n >> (h+1); ++i) {
            for (int j = i << (h+1); j < (((i << 1) + 1) << h); ++j) {
                const int t = (g[i] * 1LL * x[j | l]) % M;
                x[j | l] = x[j] - t;
                if (x[j|l] < 0) x[j | l] += M;
                x[j] += t;
                if (x[j] >= M) x[j] -= M;
            }
        }
    }
    for (int i = 0, j = 0; i < n; ++i) {
        if (i < j) std::swap(x[i], x[j]);
        for (int l = n; (l >>= 1) && !((j ^ 1) & 1); )
        {}
    }
}
vector<int> convolution(vector<int> a, vector<int> b) const {
    if(a.empty() || b.empty()) return {};
    for(int& x:a) {x%=p;if(x>=p) x-=p; if(x<0) x+=p;}
    for(int& x:b) {x%=p;if(x>=p) x-=p; if(x<0) x+=p;}
    const int na = a.size(), nb = b.size();
    int n, invN = 1;
    for (n = 1; n < na + nb - 1; n <= 1) invN = ((invN & 1) ? (invN + M) : invN) >> 1;
    vector<int> x(n, 0), y(n, 0);
    std::copy(a.begin(), a.end(), x.begin());
    std::copy(b.begin(), b.end(), y.begin());
    fft(x);
    fft(y);
    for (int i = 0; i < n; ++i) x[i] = (((static_cast<long long>(x[i]) * y[i]) % M) * invN) % M;
    std::reverse(x.begin() + 1, x.end());
    fft(x);
    x.resize(na + nb - 1);
    return x;
}
};
Fft<998244353,23,31> muls;

vector<int> form(vector<int> v,int n)
{
    while(v.size()<n) v.push_back(0);
    while(v.size()>n) v.pop_back();
    return v;
}
vector<int> operator *(vector<int> v1,vector<int> v2)
{
    return muls.convolution(v1,v2);
}
vector<int> operator +(vector<int> v1,vector<int> v2)
{
    while(v2.size()<v1.size()) v2.push_back(0); while(v1.size()<v2.size()) v1.push_back(0);
    for(int i=0;i<v1.size();++i) {v1[i]+=v2[i];if(v1[i]>=p) v1[i]-=p; else if(v1[i]<0) v1[i]+=p;}
    return v1;
}
vector<int> operator -(vector<int> v1,vector<int> v2)
{
    int sz=max(v1.size(),v2.size());while(v1.size()<sz) v1.push_back(0); while(v2.size()<sz) v2.push_back(0);
    for(int i=0;i<sz;++i) {v1[i]-=v2[i];if(v1[i]<0) v1[i]+=p; else if(v1[i]>=p) v1[i]-=p;} return v1;
}
vector<int> trmi(vector<int> v)
{
    for(int i=1;i<v.size();i+=2) {if(v[i]>0) v[i]=p-v[i]; else v[i]=(-v[i]);}
    return v;
}

```

```

}
vector<int> deriv(vector<int> v)
{
    if(v.empty()) return{};
    vector<int> ans(v.size()-1);
    for(int i=1;i<v.size();++i) ans[i-1]=(v[i]*1LL*i)%p;
    return ans;
}
vector<int> integ(vector<int> v)
{
    vector<int> ans(v.size()+1);ans[0]=0;
    for(int i=1;i<v.size();++i) ans[i-1]=(v[i]*1LL*i)%p;
    return ans;
}
vector<int> mul(vector<vector<int> > v)
{
    if(v.size()==1) return v[0];
    vector<vector<int> > v1,v2;for(int i=0;i<v.size()/2;++i) v1.push_back(v[i]); for(int i=v.size()/2;i<v.size();++i) v2.push_back(v[i]);
    return muls.convolution(mul(v1),mul(v2));
}
vector<int> invl(vector<int> v,int n)
{
    assert(v[0]!=0);
    int sz=1;v=form(v,n);vector<int> a={inv(v[0])};
    while(sz<n)
    {
        vector<int> vsz;for(int i=0;i<min(n,2*sz);++i) vsz.push_back(v[i]);
        vector<int> b=((vector<int>) {1})-muls.convolution(a,vsz);
        for(int i=0;i<sz;++i) assert(b[i]==0);
        b.erase(b.begin(),b.begin()+sz);
        vector<int> c=muls.convolution(b,a);
        for(int i=0;i<sz;++i) a.push_back(c[i]);
        sz*=2;
    }
    return form(a,n);
}

```

3.3 старое доброе FFT

```

using cd = complex<double>;
const double PI = acos(-1);

void fft(vector<cd> &a, bool invert) {
    int n = a.size();

    for (int i = 1, j = 0; i < n; i++) {
        int bit = n >> 1;
        for (; j & bit; bit >>= 1)
            j ^= bit;
        j ^= bit;

        if (i < j)
            swap(a[i], a[j]);
    }

    for (int len = 2; len <= n; len <= 1) {
        double ang = 2 * PI / len * (invert ? -1 : 1);
        cd wlen(cos(ang), sin(ang));
        for (int i = 0; i < n; i += len) {
            cd w(1);
            for (int j = 0; j < len / 2; j++) {
                cd u = a[i+j], v = a[i+j+len/2] * w;
                a[i+j] = u + v;
                a[i+j+len/2] = u - v;
                w *= wlen;
            }
        }
    }

    if (invert) {
        for (cd &x : a)
            x /= n;
    }
}
vector<int> multiply(vector<int> const& a, vector<int> const& b) {
}

```

```

vector<cd> fa(a.begin(), a.end()), fb(b.begin(), b.
    end());
int n = 1;
while (n < a.size() + b.size())
    n <= 1;
fa.resize(n);
fb.resize(n);

fft(fa, false);
fft(fb, false);
for (int i = 0; i < n; i++)
    fa[i] *= fb[i];
fft(fa, true);

vector<int> result(n);
for (int i = 0; i < n; i++)
    result[i] = round(fa[i].real());
while(!result.empty() && !result.back()) result.
    pop_back();
return result;
}

```

4 Структуры данных

4.1 Дерево Фенвика

```

int fe[maxn];
void pl(int pos, int val) {while(pos<maxn) {fe[pos]+=
    val;pos+=(pos+1);}}
int get(int pos) {int ans=0;while(pos>=0) {ans+=fe[pos
    ];pos+=(pos+1);--pos;} return ans;} /// [0,pos] -
    vkluchitelno!!!
int get(int l, int r) {return get(r-1)-get(l-1);} ///
    sum of [l,r)

```

4.2 Дерево отрезков

```

#ifdef LOCAL
int __lg(int x) { return 63 - __builtin_clzll(x); }
#endif

template<typename Data, typename Mod, typename
    UniteData, typename UniteMod, typename Apply>
struct MassSegmentTree {
    int h, n;
    Data zd;
    Mod zm;
    vector<Data> data;
    vector<Mod> mod;

    UniteData ud; // Data (Data, Data)
    UniteMod um; // Mod (Mod, Mod);
    Apply a; // Data (Data, Mod, int); last argument is
        the length of current segment (could be used for
        range += and sum counting, for instance)

    template<typename I>
    MassSegmentTree(int sz, Data zd, Mod zm, UniteData
        ud, UniteMod um, Apply a, I init) : h(__lg(sz > 1
        ? sz - 1 : 1) + 1), n(1 << h), zm(zm), zd(zd),
        data(2 * n, zd), mod(n, zm), ud(ud), um(um), a(a)
        {
            for (int i = 0; i < sz; ++i) data[i + n] = init(i)
            ;
            for (int i = n - 1; i > 0; --i) data[i] = ud(data
                [2 * i], data[2 * i + 1]);
        }

    MassSegmentTree(int sz, Data zd, Mod zm, UniteData
        ud, UniteMod um, Apply a) : h(__lg(sz > 1 ? sz - 1
        : 1) + 1), n(1 << h), zm(zm), zd(zd), data(2 * n,
        zd), mod(n, zm), ud(ud), um(um), a(a) {}

    void push(int i) {
        if (mod[i] == zm) return;
        apply(2 * i, mod[i]);
        apply(2 * i + 1, mod[i]);
        mod[i] = zm;
    }
}

```

```

// is used only for apply
int length(int i) { return 1 << (h - __lg(i)); }

// is used only for descent
int left(int i) {
    int lvl = __lg(i);
    return (i & ((1 << lvl) - 1)) * (1 << (h - lvl));
}

// is used only for descent
int right(int i) {
    int lvl = __lg(i);
    return ((i & ((1 << lvl) - 1)) + 1) * (1 << (h -
        lvl));
}

template<typename S>
void apply(int i, S x) {
    data[i] = a(data[i], x, length(i));
    if (i < n) mod[i] = um(mod[i], x);
}

void update(int i) {
    if (mod[i] != zm) return;
    data[i] = ud(data[2 * i], data[2 * i + 1]);
}

template<typename S>
void update(int l, int r, S x) { // [l; r)
    l += n, r += n;
    for (int shift = h; shift > 0; --shift) {
        push(l >> shift);
        push((r - 1) >> shift);
    }
    for (int lf = l, rg = r; lf < rg; lf /= 2, rg /=
        2) {
        if (lf & 1) apply(lf++, x);
        if (rg & 1) apply(--rg, x);
    }
    for (int shift = 1; shift <= h; ++shift) {
        update(l >> shift);
        update((r - 1) >> shift);
    }
}

Data get(int l, int r) { // [l; r)
    l += n, r += n;
    for (int shift = h; shift > 0; --shift) {
        push(l >> shift);
        push((r - 1) >> shift);
    }
    Data leftRes = zd, rightRes = zd;
    for (; l < r; l /= 2, r /= 2) {
        if (l & 1) leftRes = ud(leftRes, data[l++]);
        if (r & 1) rightRes = ud(data[--r], rightRes);
    }
    return ud(leftRes, rightRes);
}

// l \in [0; n) && ok(get(l, l), l);
// returns last r: ok(get(l, r), r)
template<typename C>
int lastTrue(int l, C ok) {
    l += n;
    for (int shift = h; shift > 0; --shift) push(l >>
        shift);
    Data cur = zd;
    do {
        l >>= __builtin_ctz(l);
        Data withl;
        withl = ud(cur, data[l]);
        if (ok(withl, right(l))) {
            cur = withl;
            ++l;
        } else {
            while (l < n) {
                push(l);
                Data with2;
                with2 = ud(cur, data[2 * l]);
                if (ok(with2, right(2 * l))) {

```

```

        cur = with2;
        l = 2 * l + 1;
    } else {
        l = 2 * l;
    }
}
return l - n;
}
while (l & (l - 1));
return n;
}

// r \in [0; n) && ok(get(r, r), r);
// returns first l: ok(get(l, r), l)
template<typename C>
int firstTrue(int r, C ok) {
    r += n;
    for (int shift = h; shift > 0; --shift) push((r -
1) >> shift);
    Data cur = zd;
    while (r & (r - 1)) {
        r >>= __builtin_ctz(r);
        Data with1;
        with1 = ud(data[--r], cur);
        if (ok(with1, left(r))) {
            cur = with1;
        } else {
            while (r < n) {
                push(r);
                Data with2;
                with2 = ud(data[2 * r + 1], cur);
                if (ok(with2, right(2 * r))) {
                    cur = with2;
                    r = 2 * r;
                } else {
                    r = 2 * r + 1;
                }
            }
            return r - n + 1;
        }
    }
    return 0;
}
};

void example () {
    // max and +=
    MassSegmentTree segtree(n, 0LL, 0LL,
    [](int x, int y) { return max(x, y); },
    [](int x, int y) { return x + y; },
    [](int x, int y, int len) { return x + y; });

    // sum and +=
    MassSegmentTree segtree(n, 0LL, 0LL,
    [](int x, int y) { return x + y; },
    [](int x, int y) { return x + y; },
    [](int x, int y, int len) { return x + y * len; });

    // sum and assignment
    MassSegmentTree segtree(n, 0LL, -1LL,
    [](int x, int y) { return x + y; },
    [](int x, int y) { return y; },
    [](int x, int y, int len) { return y * len; });
}

```

4.3 Битовый бор

```

template<unsigned int sz, typename T=int>
struct binarytrie{
    using Bit=typename conditional<sz<=32,unsigned int,
    unsigned long long>::type;
    struct node{
        T cnt;
        array<int,2>nxt;
        node():cnt(0),nxt({-1,-1}){}
    };
    vector<node>v;
    binarytrie(){v.emplace_back();}
    void insert(Bit x){add(x,1);}
    void erase(Bit x){add(x,-1);}
    void add(Bit x,T k)

```

```

{
    assert(0<=x&&(x>>sz)==0);
    int p=0;
    v[p].cnt+=k;
    for(int i=sz;i--;)
    {
        int j=x>>i&1;
        if(v[p].nxt[j]==-1)
        {
            v[p].nxt[j]=v.size();
            v.emplace_back();
        }
        p=v[p].nxt[j];
        v[p].cnt+=k;
    }
}
T count(Bit x,Bit xor_val=0)const//[0,x)
{
    assert(0<=xor_val&&(xor_val>>sz)==0);
    if(x<0)return 0;
    else if(x>>sz)return v[0].cnt;
    T ret=0;
    int p=0;
    for(int i=sz;i--;)
    {
        int j=x>>i&1,k=xor_val>>i&1;
        if(j==0)p=v[p].nxt[k];
        else
        {
            if(v[p].nxt[k]>=0)ret+=v[v[p].nxt[k]].cnt;
            p=v[p].nxt[!k];
        }
        if(p==-1)break;
    }
    return ret;
}
Bit max(Bit xor_val=0)const
{
    assert(0<=xor_val&&(xor_val>>sz)==0);
    int p=0;
    Bit ret=0;
    if(v[p].cnt==0)return ret;
    for(int i=sz;i--;)
    {
        ret<=1;
        int k=xor_val>>i&1;
        if(v[p].nxt[!k]>=0&&v[v[p].nxt[!k]].cnt>0)
        {
            p=v[p].nxt[!k];
            ret|=1;
        }
        else p=v[p].nxt[k];
    }
    return ret;
}
Bit min(Bit xor_val=0)const
{
    assert(0<=xor_val&&(xor_val>>sz)==0);
    int p=0;
    Bit ret=0;
    for(int i=sz;i--;)
    {
        ret<=1;
        int k=xor_val>>i&1;
        if(v[p].nxt[k]>=0&&v[v[p].nxt[k]].cnt>0)p=v[p].
nxt[k];
        else
        {
            p=v[p].nxt[!k];
            ret|=1;
        }
    }
    return ret;
}
Bit find_by_order(T ord,Bit xor_val=0)const
{
    assert(0<=xor_val&&(xor_val>>sz)==0);
    assert(0<=ord&&ord<v[0].cnt);
    int p=0;
    Bit ret=0;

```

```

for(int i=sz;i--;)
{
    ret<=1;
    int k=xor_val>>i&1;
    if(v[p].nxt[k]>=0)
    {
        if(ord>=v[v[p].nxt[k]].cnt)
        {
            ord=v[v[p].nxt[k]].cnt;
            p=v[p].nxt[k];
            ret|=1;
        }
        else p=v[p].nxt[k];
    }
    else
    {
        p=v[p].nxt[!k];
        ret|=1;
    }
}
return ret;
}
T order_of_key(Bit x, Bit xor_val=0) const { return
    count(x, xor_val); }
};
binarytrie<32> bt;

```

4.4 Ordered set

```

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

using namespace __gnu_pbds;
using namespace std;

using ordered_set = tree<int, null_type, less<>,
    rb_tree_tag, tree_order_statistics_node_update>;

```

4.5 Convex hull trick

```

int div_up(int a, int b) { return a/b+((a^b)>0&&a%b);
} // divide a by b rounded up
const int LQ = ..., RQ = ...; //leftmost query,
rightmost query
int in(ii L, int x) {
    return L.x * x + L.y;
}
struct Hull {
    vector <pair <int, int> > lines;
    vector <int> borders;
    void push(ii L) {
        while (lines.size() && in(L, borders.back()) < in(
            lines.back(), borders.back())) {
            lines.pop_back();
            borders.pop_back();
        }
        if (lines.empty()) {
            lines = {L};
            borders = {LQ};
        }
        else if (lines.back().x > L.x) {
            int x = div_up(L.y - lines.back().y, lines.
                back().x - L.x);
            if (x <= RQ) {
                lines.app(L);
                borders.app(x);
            }
        }
    }
}
Hull (){}
Hull (vector <ii> a) {
    auto comp = [&] (ii u, ii v) {
        return u.x > v.x || (u.x == v.x && u.y < v.y);
    };
    sort(all(a), comp);
    for (auto L : a) {
        push(L);
    }
}
int get(int x) {

```

```

    int pos = upper_bound(all(borders), x) - borders.
        begin();
    assert(pos>0);
    pos--;
    return in(lines[pos], x);
}
};

```

4.6 Центроиды

```

vector<int> sz(n), lvl(n, -1);
auto dfs = [&](auto dfs, int cur, int prev) -> int {
    if (lvl[cur] != -1) return 0;
    sz[cur] = 1;
    for (auto [nxt, w] : g[cur]) {
        if (nxt != prev) sz[cur] += dfs(dfs, nxt, cur);
    }
    return sz[cur];
};
auto find = [&](auto find, int cur, int prev, int tot)
    -> int {
    int bch = -1, bsz = 0;
    for (auto [nxt, w] : g[cur]) {
        if (nxt == prev || lvl[nxt] != -1) continue;
        if (sz[nxt] > bsz) {
            bch = nxt;
            bsz = sz[nxt];
        }
    }
    if (bsz + bsz <= tot) return cur;
    return find(find, bch, cur, tot);
};
dfs(dfs, 0, 0);
auto c = find(find, 0, 0, sz[0]);
vector<pair<int, int>> stack{{c, 0}};
int ans = INF;
while (!stack.empty()) {
    auto [centroid, l] = stack.back();
    stack.pop_back();
    lvl[centroid] = 1;
    for (auto [nxt, w] : g[centroid]) {
        if (lvl[nxt] != -1) continue;
        dfs(dfs, nxt, centroid);
        int new_centroid = find(find, nxt, centroid,
            sz[nxt]);
        stack.push_back({new_centroid, lvl[centroid] +
            1});
    }
}

```

5 Строковые алгоритмы

5.1 Префикс-функция

```

vector<int> prefix_function(string s) {
    vector<int> p(s.size());
    for (int i = 1; i < s.size(); ++i) {
        p[i] = p[i - 1];
        while (p[i] && s[p[i]] != s[i]) p[i] = p[p[i] -
            1];
        p[i] += s[i] == s[p[i]];
    }
    return p;
}

```

5.2 Z-функция

```

vector<int> z_function (string s) { // z[i] - lcp of s
    and s[i:]
    int n = (int) s.length();
    vector<int> z (n);
    for (int i=1, l=0, r=0; i<n; ++i) {
        if (i <= r)
            z[i] = min (r-i+1, z[i-l]);
        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;
}

```

5.3 Алгоритм Манакера

```

vector<int> manacher_odd(const string &s) {
    vector<int> man(s.size(), 0);
    int l = 0, r = 0;
    int n = s.size();
    for (int i = 1; i < n; i++) {
        if (i <= r) {
            man[i] = min(r - i, man[l + r - i]);
        }
        while (i + man[i] + 1 < n && i - man[i] - 1 >= 0
            && s[i + man[i] + 1] == s[i - man[i] - 1]) {
            man[i]++;
        }
        if (i + man[i] > r) {
            l = i - man[i];
            r = i + man[i];
        }
    }
    return man;
}
// abacaba : (0 1 0 3 0 1 0)
// abbaa : (0 0 0 0 0)

vector<int> manacher_even(const string &s) {
    assert(s.size());
    string t;
    for (int i = 0; i + 1 < s.size(); ++i) {
        t += s[i];
        t += '#';
    }
    t += s.back();
    auto odd = manacher_odd(t);
    vector<int> ans;
    for (int i = 1; i < odd.size(); i += 2) {
        ans.push_back((odd[i] + 1) / 2);
    }
    return ans;
}
// abacaba : (0 0 0 0 0 0)
// abbaa : (0 2 0 1)

```

5.4 Суфмассив

Переработанный китайский суффмассив

```

const int inf = 1e9;
struct rmq {
    int n;
    vector<int> a;
    void build(const vector<int> &x) {
        assert(x.size() == n);
        for (int i = 0; i < n; ++i) a[n + i] = x[i];
        for (int i = n - 1; i > 0; --i) a[i] = min(a[2 * i], a[2 * i + 1]);
    }
    rmq(int n) : n(n), a(2 * n, inf) {}
    void put(int i, int x) {
        a[i + n] = min(a[i + n], x);
        for (i = (i + n) / 2; i > 0; i /= 2) {
            a[i] = min(a[i * 2], a[i * 2 + 1]);
        }
    }
    int getMin(int l, int r) { // [l; r)
        assert(l < r);
        int res = inf;
        for (l += n, r += n; l < r; l /= 2, r /= 2) {
            if (l & 1) res = min(res, a[l++]);
            if (r & 1) res = min(res, a[--r]);
        }
        return res;
    }
};
template <typename T>
vector<int> SA(const T &a) {
    int m = *max_element(all(a)) + 1, n = a.size();
    vector<int> sa(n), nsa(n), pre(max(n, m)), x(a.begin(), a.end()), y(n);
    for (int e : x) pre[e]++;

```

```

    for (int i = 1; i < m; ++i) pre[i] += pre[i - 1];
    for (int i = 0; i < n; ++i) sa[--pre[x[i]]] = i;
    int dif = 1;
    y[sa.front()] = 0;
    for (int i = 1; i < n; ++i) {
        dif += x[sa[i]] != x[sa[i - 1]];
        y[sa[i]] = dif - 1;
    }
    x = y;
    for (int h = 1; dif < n; h *= 2) {
        fill(all(pre), 0);
        for (int e : x) pre[e]++;
        for (int i = 1; i < dif; ++i) pre[i] += pre[i - 1];
        for (int t = n; t--;) {
            int i = sa[t];
            if (i >= h) {
                nsa[--pre[x[i - h]]] = i - h;
            }
            else if (i + 1 != h) {
                nsa[--pre[x[i - h + n + 1]]] = i - h + n + 1;
            }
        }
        nsa[--pre[x[n - h]]] = n - h;
        sa = nsa;
        auto getr = [&](int i) {
            if (i + h < n) {
                return x[i + h];
            }
            else {
                return x[i + h - n - 1];
            }
        };
        dif = 1;
        y[sa.front()] = 0;
        for (int i = 1; i < n; ++i) {
            if (x[sa[i]] != x[sa[i - 1]] || sa[i - 1] + h == n) {
                dif++;
            }
            else {
                dif += getr(sa[i]) != getr(sa[i - 1]);
            }
            y[sa[i]] = dif - 1;
        }
        x = y;
    }
    return sa;
}

```

```

template <typename T>
struct suar {
    vector<int> sa, lcp, pos;
    suar(const T &a) : t((int)a.size() - 1) {
        sa = SA(a);
        int n = (int)a.size(), k = 0;
        lcp.resize(n - 1);
        pos.resize(n);
        for (int i = 0; i < n; ++i) pos[sa[i]] = i;
        for (int i = 0; i < n; ++i) {
            if (pos[i] + 1 < n) {
                int j = sa[pos[i] + 1];
                while (i + k < n && j + k < n && a[i + k] == a[j + k]) k++;
                lcp[pos[i]] = k;
            }
            if (k) {
                k--;
            }
        }
        t.build(lcp);
    }
    int getLcp(int i, int j) {
        i = pos[i]; j = pos[j];
        if (j < i) {
            swap(i, j);
        }
        if (i == j) {
            return inf;
        }
    }
}

```



```

        else {
            return t.getMin(i, j);
        }
    }
};

```

5.5 Алгоритм Ахо — Корасик

5.6 Алгоритм Ахо Корасик

```

struct node{
    int next[alpha] = {}, link[alpha] = {};
    int suf = 0;
    ll visited = 0, ans = 0;
    vector<int> term;
    node() {}
};

vector<node> mem;

int get_next(int nd, char c) {
    if (!mem[nd].next[c - a]) { mem[nd].next[c - a] =
        mem.size(); mem.emplace_back(); }
    return mem[nd].next[c - a];
}

void find(string s, vector<string> t) {
    mem.reserve(1e6 + 100); mem.clear();
    mem.emplace_back(); mem.emplace_back();
    // 0th element is nullptr, 1st is the root
    int q = t.size();
    for (int j = 0; j < q; ++j) {
        int cur = 1;
        for (char c : ts[j]) cur = get_next(cur, c);
        mem[cur].term.push_back(j);
    }
    vector<int> bfs_order;
    queue<int> bfs;
    {
        node &root = mem[1];
        root.suf = 1;
        for (char c = a; c < a + alpha; ++c) {
            root.link[c - a] = (root.next[c - a] ? root.
next[c - a] : 1);
        }
        bfs.push(1);
    }
    while (!bfs.empty()) {
        int cur_idx = bfs.front();
        bfs.pop();
        node &cur = mem[cur_idx];
        bfs_order.push_back(cur_idx);
        for (char c = a; c < a + alpha; ++c) {
            int nxt_idx = cur.next[c - a];
            if (!nxt_idx) continue;
            node &nxt = mem[nxt_idx];
            nxt.suf = (cur_idx == 1 ? 1 : mem[cur.suf].link[
c - a]);
            for (char c = a; c < a + alpha; ++c) {
                nxt.link[c - a] = (nxt.next[c - a] ? nxt.next[
c - a] : mem[nxt.suf].link[c - a]);
            }
            bfs.push(nxt_idx);
        }
    }
    // do something
}

```

5.7 Дерево палиндромов

5.8 Дерево палиндромов

```

struct palindromic{
    int n;
    vector<int> p, suf{0, 0}, len{-1, 0};
    vector<array<int, alpha>> to{{}, {}};
    int sz = 2;

    palindromic(const string &s) : n(s.size()), p(n + 1,
1) {

```

```

        suf.reserve(n);
        len.reserve(n);
        for (int i = 0; i < n; ++i) {
            auto check = [&](int l) { return i > l && s[i]
== s[i - l - 1]; };
            int par = p[i];
            while (!check(len[par])) par = suf[par];
            if (to[par][s[i] - a]) {
                p[i + 1] = to[par][s[i] - a];
                continue;
            }
            p[i + 1] = sz++;
            to[par][s[i] - a] = p[i + 1];
            to.emplace_back();
            len.emplace_back(len[par] + 2);
            do {
                par = suf[par];
            } while (!check(len[par]));
            int link = to[par][s[i] - a];
            if (link == p[i + 1]) link = 1;
            suf.emplace_back(link);
        }
    }
};

```

6 Потоки

6.1 Алгоритм Диница

```

#define pb push_back
struct Dinic{
    struct edge{
        int to, flow, cap;
    };

    const static int N = 555; //count of vertices

    vector<edge> e;
    vector<int> g[N + 7];
    int dp[N + 7];
    int ptr[N + 7];

    void clear(){
        for (int i = 0; i < N + 7; i++) g[i].clear();
        e.clear();
    }

    void addEdge(int a, int b, int cap){
        g[a].pb(e.size());
        e.pb({b, 0, cap});
        g[b].pb(e.size());
        e.pb({a, 0, 0});
    }

    int minFlow, start, finish;

    bool bfs(){
        for (int i = 0; i < N; i++) dp[i] = -1;
        dp[start] = 0;
        vector<int> st;
        int uk = 0;
        st.pb(start);
        while(uk < st.size()){
            int v = st[uk++];
            for (int to : g[v]){
                auto ed = e[to];
                if (ed.cap - ed.flow >= minFlow && dp[ed.to] ==
-1){
                    dp[ed.to] = dp[v] + 1;
                    st.pb(ed.to);
                }
            }
        }
        return dp[finish] != -1;
    }

    int dfs(int v, int flow){
        if (v == finish) return flow;
        for (; ptr[v] < g[v].size(); ptr[v]++){
            int to = g[v][ptr[v]];

```

```

edge ed = e[to];
if (ed.cap - ed.flow >= minFlow && dp[ed.to] == dp
[v] + 1){
    int add = dfs(ed.to, min(flow, ed.cap - ed.flow)
);
    if (add){
        e[to].flow += add;
        e[to ^ 1].flow -= add;
        return add;
    }
}
return 0;
}

int dinic(int start, int finish){
    Dinic::start = start;
    Dinic::finish = finish;
    int flow = 0;
    for (minFlow = (1 << 30); minFlow; minFlow >>= 1){
        while(bfs()){
            for (int i = 0; i < N; i++) ptr[i] = 0;
            while(int now = dfs(start, (int)2e9 + 7)) flow
+= now;
        }
    }
    return flow;
}
} dinic;

```

6.2 Mincost k-flow

```

struct edge {
    int next, capacity, cost, flow = 0;

    edge() = default;

    edge(int next, int capacity, int cost) : next(next),
        capacity(capacity), cost(cost) {}

    int rem() const { return capacity - flow; }

    int operator+=(int f) { return flow += f; }

    int operator-=(int f) { return flow -= f; }
};

auto addEdge = [&](auto from, auto next, auto capacity
, int cost) {
    g[from].push_back(e.size());
    e.emplace_back(next, capacity, cost);
    g[next].push_back(e.size());
    e.emplace_back(from, 0, -cost);
};

/* in case of undirected graph use this:
addEdge(u, v, capacity, cost);
addEdge(v, u, capacity, cost);
*/

vector<ll> phi(n, 0);
auto fordBellman = [&](int s, int t) {
    phi.assign(n, 0);
    for (int iter = 0; iter < n; ++iter) {
        bool changed = false;
        for (int u = 0; u < n; ++u) {
            for (auto index : g[u]) {
                auto edge = e[index];
                if (edge.rem() > 0 && phi[edge.next] > phi[u]
+ edge.cost) {
                    phi[edge.next] = phi[u] + edge.cost;
                    changed = true;
                }
            }
        }
        if (!changed) break;
    }
};

fordBellman(s, t);
// now shortest path using dijkstra with potentials
vector<ll> dist;
vector<int> from;
vector<bool> cnt;
auto dijkstra = [&](int s, int t) {

```

```

    dist.assign(n, 1e18);
    from.assign(n, -1);
    cnt.assign(n, false);
    dist[s] = 0;
    set<pair<int, int>> se;
    se.insert({0, s});
    while ((int)(se.size())) {
        int cur = se.begin()->y;
        se.erase(se.begin());
        cnt[cur] = true;
        for (int index : g[cur]) {
            auto &edge = e[index];
            if (edge.rem() == 0) continue;
            ll weight = edge.cost + phi[cur] - phi[edge.next
];
            if (dist[edge.next] > dist[cur] + weight) {
                se.erase({dist[edge.next], edge.next});
                dist[edge.next] = dist[cur] + weight;
                se.insert({dist[edge.next], edge.next});
                from[edge.next] = cur;
            }
        }
    }
    if (dist[t] == (ll) 1e18) return -1LL;
    ll cost = 0;
    for (int p = t; p != s; p = from[p]) {
        for (auto index : g[from[p]]) {
            auto &edge = e[index];
            ll weight = edge.cost + phi[from[p]] - phi[edge.
next];
            if (edge.rem() > 0 && edge.next == p && dist[
edge.next] == dist[from[p]] + weight) {
                edge += 1;
                e[index ^ 1] -= 1;
                cost += edge.cost;
                break;
            }
        }
    }
    for (int i = 0; i < n; ++i) {
        phi[i] += dist[i];
    }
    return cost;
};

ll cost = 0;
for (int flow = 0; flow < k; ++flow) {
    ll a = dijkstra(s, t);
    if (a == -1) {
        cout << "-1\n";
        return;
    }
    cost += a;
}

// now recover answer
auto findPath = [&](int s, int t) {
    vector<int> ans;
    int cur = s;
    while (cur != t) {
        for (auto index : g[cur]) {
            auto &edge = e[index];
            if (edge.flow <= 0) continue;
            edge -= 1;
            e[index ^ 1] += 1;
            ans.push_back(index / 4);
            // index / 4 because each edge has 4 copies
            cur = edge.next;
            break;
        }
    }
    return ans;
};

for (int flow = 0; flow < k; ++flow) {
    auto p = findPath(s, t);
    cout << p.size() << ' ';
    for (int x : p) cout << x + 1 << ' ';
    cout << '\n';
}

```

7 Геометрия

7.1 Примитивы

```
struct Point {
    int x, y;
    Point(){}
    Point (int x_, int y_) {
        x = x_; y = y_;
    }
    Point operator + (Point p) {
        return Point(x+p.x,y+p.y);
    }
    Point operator - (Point p) {
        return Point(x - p.x, y - p.y);
    }
    int operator * (Point p) {
        return x * p.y - y * p.x;
    }
    int operator % (Point p) {
        return x * p.x + y * p.y;
    }
    bool operator < (Point v) {
        return (*this) * v > 0;
    }
    bool operator > (Point v) {
        return v < (*this);
    };
    bool operator <= (Point v) {
        return (*this) * v >= 0;
    }
};

bool line(Point a, Point b, Point c) {
    return (b-a)*(c-b)==0;
}

bool ord(Point a, Point p, Point b) {
    return (p - a)%(p - b)<0;
}
}
```

7.2 Выпуклая оболочка

```
using pt = pair<int, int>;
#define x first
#define y second

int cross(pt p, pt q) {
    return p.x * q.y - p.y * q.x;
}

int scalar(pt p, pt q) {
    return p.x * q.x + p.y * q.y;
}

pt operator-(pt a, pt b) { return {a.x - b.x, a.y - b.y}; }

vector<pt> convex(vector<pt> a) {
    sort(all(a));
    if (a.size() == 2 && a[0] == a[1]) return {a[0]};
    if (a.size() <= 1) return a;
    vector<pt> h;
    for (int t = 0; t < 2; ++t) {
        int sz = h.size() - t;
        for (auto p: a) {
            while (h.size() >= sz + 2 && cross(p - h.
                end()[-1], h.end()[-2] - h.end()[-1]) <= 0) h.
                pop_back();
            h.push_back(p);
        }
        reverse(all(a));
    }
    return h; // h is circular: h.front() == h.back()
}
}
```

7.3 Точка внутри многоугольника

```
auto inT = [&] (Point a, Point b, Point c, Point p
) {
    a = a-p; b = b-p; c = c-p;
    return abs(a*b)+abs(b*c)+abs(c*a) == abs(a*b+b
        *c+c*a);
};

auto inP = [&] (Point p) { //a must be in
    counterclockwise order!
```

```
int l = 1, r = n - 1;
while (l < r - 1) {
    int m = (l + r) / 2;
    if ((a[m] - a[0]) < (p - a[0])) {
        l = m;
    }
    else {
        r = m;
    }
}
return inT(a[l], a[0], a[r], p);
};
```

7.4 Касательные

```
auto max = [&] (auto cmp) {
    int k = 0;
    for (int lg = 18; lg >= 0; --lg) {
        int i = k + (1 << lg), j = k - (1 << lg);
        i = (i % n + n) % n;
        j = (j % n + n) % n;
        array<int, 3> ind{i, j, k};
        sort(all(ind), cmp);
        k = ind[2];
    }
    return k;
};

auto uppert = [&] (Point p) { //last vertex in
    counterclockwise order about p
    auto cmp = [&] (int i, int j) {return (a[i] -
        p) < (a[j] - p); };
    return max(cmp);
};

auto lowert = [&] (Point p) { //first vertex in
    counterclockwise order about p
    auto cmp = [&] (int i, int j) {return (a[i] -
        p) > (a[j] - p); };
    return max(cmp);
};

auto uppertinf = [&] (Point p) { //upper tangent
    line parallel to vector p
    swap(p.x, p.y);
    p.x = -p.x;
    auto cmp = [&] (int i, int j) { return a[i] %
        p < a[j] % p; };
    return max(cmp);
};

auto lowertinf = [&] (Point p) { //lower tangent
    line parallel to vector p
    swap(p.x, p.y);
    p.x = -p.x;
    auto cmp = [&] (int i, int j) { return a[i] %
        p > a[j] % p; };
    return max(cmp);
};
};
```

8 Разное

8.1 Флаги компиляции

```
-DLOCAL -Wall -Wextra -pedantic -Wshadow -Wformat=2
-Wfloat-equal -Wconversion -Wlogical-op -Wshift-
overflow=2 -Wduplicated-cond -Wcast-qual -Wcast-
align -D_GLIBCXX_DEBUG -D_GLIBCXX_DEBUG_PEDANTIC
-D_FORTIFY_SOURCE=2 -fsanitize=address -
fsanitize=undefined -fno-sanitize-recover -fstack-
protector -std=c++2a
```

8.1.1 Сетка в vim

<https://codeforces.com/blog/entry/122540>

```
i|<esc>25A |<esc>
o+<esc>25A---+<esc>
Vky35Pdd
```

8.2 Что сделать на пробном туре

- Убедиться, что работают все IDE. Разобраться, как настраивать в них LOCAL.

- В системе ML — это ML или RE?
- Максимальный размер файла
- Можно посмотреть на время работы серверов позапусков Флойда — Варшалла

8.3 Шаблон

```

#ifdef LOCAL
#define _GLIBCXX_DEBUG
#endif
#include<bits/stdc++.h>

using namespace std;

#define int long long
#define app push_back
#define all(x) x.begin(), x.end()
#ifdef LOCAL
#define debug(...) [](auto...a){ ((cout << a << ' '
    ), ...) << endl; }(__VA_ARGS__, ":",
    __VA_ARGS__)
#define debugv(v) do { cout << #v << ": "; for (
    auto x : v) cout << x << ' '; cout << endl; }
    while(0)
#else
#define debug(...)
#define debugv(v)
#endif

int32_t main() {
    cin.tie(0);ios_base::sync_with_stdio(0);
}

```