# Team Note of PS:Endgame

playsworld16, plast, Serendipity__

Compiled on June 1, 2024

## Contents

# 1 Have you...

## 1.1 tried...

- **Reading the problem once more?**
- doubting "obvious" things?
- writing obivous things?
- radical greedy approach?
- thinking in reverse direction?
- a greedy algorithm?
- network flow when your greedy algorithms stuck?
- a dynamic programming?
- checking the range of answer?
- random algorithm?
- graph modeling using states?
- inverting state only on odd indexes?
- square root decomposition?
- calculating error bound on a real number usage?

## 1.2 checked...

- **you have read the statement correctly?**
- typo copying the team note?
- initialization on multiple test case problem?
- additional information from the problem?
- undefined behavior?
- order of if-else statement?
- order of recursion inside a function?
- overflow?
- function without return value?
- real number error?
- implicit conversion?
- comparison between signed and unsigned integer?

## 1.3 Template

```
#include <bits/stdc++.h>
using namespace std;
mt19937_64
rng(chrono::high_resolution_clock::now().time_since_epoch().count());
// random int64 generator
typedef long long ll;
typedef unsigned long ul;
typedef unsigned long long ull;
typedef __int128 ll128;
typedef long double ld;
typedef pair<int, int> pi;
typedef pair<ll, ll> pii;
typedef complex<double> inum;
const double PI = acos(-1);
const int INF = 0x3f3f3f3f;
const ll LLINF = 1000000000000000000LL;
// Macros from KACTL pdf
#define rep(i, a, b) for(int i = a; i < (b); ++i)
#define all(x) begin(x), end(x)
#define sz(x) (int)(x).size()
typedef vector<int> vi;
typedef vector<double> vd;
void solve() {

}
int main() {
    std::ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);
    int tc = 1;
    // cin >> tc;
    while (tc--) {
        solve();
    }
    return 0;
}
```

# 2 DataStructures

## 2.1 UnionFind

**Usage:** path compression, 0-based
**Time Complexity:** $\mathcal{O}(1)$ amortized

```
struct DSU {
    int N;
    vector<int> parent, member;
    DSU() { N = 0; }
    DSU(int n) {
        N = n;
        parent.resize(n);
        member.resize(n);
        for (int i = 0; i < n; i++) { parent[i] = -1;
        member[i] = 1; }
    }
    int root(int n) {
        if (parent[n] == -1) { return n; }
        return parent[n] = root(parent[n]);
    }
    void merge(int r1, int r2) {
        r1 = root(r1);
        r2 = root(r2);
        if (r1 == r2) { return; }
        if (member[r1] < member[r2]) { swap(r1, r2); }
        parent[r2] = r1;
        member[r1] += member[r2];
    }
};
```

https://www.overleaf.com/project/6546695714fd2924210b16cc

## 2.2 Segment Tree

**Usage:** Non-recursive, 0-based

```
struct sumseg{
    int N;
    vector<ll> a,seg;
    sumseg(){N=0;}
    sumseg(int n){
        N=n;
        a.resize(N);
        seg.resize(2*N);
    }
    void init(){
        for (int i=0;i<N;i++){seg[i+N]=a[i];}
        for (int i=N-1;i>=1;i--){
            seg[i] = seg[i<<1]+seg[i<<1|1];
        }
    }
    void update(int i, ll val){
        a[i] = val;
        seg[i+N]=val;
        int p = (i+N)>>1;
        while(p>0){
            seg[p] = seg[p<<1]+seg[p<<1|1];
            p>>=1;
        }
    }
    ll query(int l, int r){
        if (l>r){return 0LL;}
        ll ret=0;
        int lp,rp;
        lp = l+N; rp=r+N+1;
        while(lp<rp){
            if (lp&1){ret+=seg[lp++];}
            if (rp&1){ret+=seg[--rp];}
            lp>>=1; rp>>=1;
        }
        return ret;
    }
};
```

## 2.3 Lazy Segment Tree

**Usage:** 0-based
**Time Complexity:** $\mathcal{O}(\log N)$

```
struct lazy_sumseg {
    int N;
    vector<ll> a, seg, lazy;
    lazy_sumseg() { N = 0; }
```

```
    lazy_sumseg(int n) {
        N = n;
        int M=1;
        while(M<N){M<<=1;}
        a.resize(n);
        seg.resize(M<<1);
        lazy.resize(M<<1);
    }

    void maketree(int l, int r, int node) {
        if (l == r) { seg[node] = a[l]; return; }
        int mid = (l + r) >> 1;
        maketree(l, mid, node * 2);
        maketree(mid + 1, r, node * 2 + 1);
        seg[node] = seg[node * 2] + seg[node * 2 + 1];
    }
    void propagate(int l, int r, int node) {
        if (l == r) { seg[node] = seg[node] + lazy[node];
        lazy[node] = 0; return; }
        seg[node] = seg[node] + lazy[node] * (r - l + 1);
        lazy[node * 2] = lazy[node*2] + lazy[node];
        lazy[node * 2 + 1] = lazy[node*2+1] + lazy[node];
        lazy[node] = 0;
    }
    void update(int l, int r, int from, int to, ll inc, int
    node) {
        propagate(l, r, node);
        if (from <= l && r <= to) { lazy[node] = lazy[node] +
        inc; propagate(l, r, node); return; }
        if (from > r || to < l) { return; }
        int mid = (l + r) >> 1;
        update(l, mid, from, to, inc, node * 2);
        update(mid + 1, r, from, to, inc, node * 2 + 1);
        seg[node] = seg[node * 2] + seg[node * 2 + 1];
    }
    ll query(int l, int r, int from, int to, int node) {
        propagate(l, r, node);
        if (from <= l && r <= to) { return seg[node]; }
        if (from > r || to < l) { return 0; }
        int mid = (l + r) >> 1;
        return query(l, mid, from, to, node * 2) + query(mid +
        1, r, from, to, node * 2 + 1);
    }
    void maketree() { maketree(0,N-1,1); }
    void update(int from, int to, ll inc) { update(0,N-1,
    from, to, inc, 1); }
    ll query(int from, int to) { return query(0,N-1, from, to,
    1); }
};
```

## 2.4 2D Segment Tree

**Usage:** Dynamic 2D Segment Tree, fixed array implmentation may help memory
**Time Complexity:** $\mathcal{O}(logN * logM)$ update, query

```
struct seg2d{
    struct Node{
        int v,l,r;
        Node(){v=l=r=0;}
    };
    int M,N,sz=2;
    vector<int> L,R;
    vector<Node> t;
    vector<vector<Node>> seg;
    seg2d(int N, int M):N(N),M(M){
        L.resize(2); R.resize(2); t.resize(2);
        seg.resize(2,t);
    }
    void upd1(int x, int val, int l, int r, int ny, int nx){
        if (l==r){
            seg[ny][nx].v = val; return;
        }
        seg[ny][nx].v = max(seg[ny][nx].v, val);
        int mid = (l+r)>>1;
        if (x<=mid){
            if (seg[ny][nx].l==0){
                seg[ny].push_back(Node());
                seg[ny][nx].l = seg[ny].size()-1;
            }
            upd1(x,val,l,mid,ny,seg[ny][nx].l);
        }else{
            if (seg[ny][nx].r==0){
                seg[ny].push_back(Node());
                seg[ny][nx].r = seg[ny].size()-1;
            }
            upd1(x,val,mid+1,r,ny,seg[ny][nx].r);
        }
    }
    void upd2(int y, int x, int val, int l, int r, int ny){
        upd1(x,val,0,N,ny,1);
        if (l==r){return;}

        int mid = (l+r)>>1;
        if (y<=mid){
            if (L[ny]==0){
                seg.push_back(t); L.push_back(0);
                R.push_back(0); ++sz;
                L[ny] = sz-1;
            }
            upd2(y,x,val,l,mid,L[ny]);
        }else{
            if (R[ny]==0){
                seg.push_back(t); L.push_back(0);
                R.push_back(0); ++sz;
                R[ny] = sz-1;
            }
            upd2(y,x,val,mid+1,r,R[ny]);
        }
    }
    int query1(int x1, int x2, int l, int r, int ny, int nx){
        if (ny==0 || nx==0){return 0;}
        if (x2<l || r<x1){return 0;}
        if (x1<=l && r<=x2){return seg[ny][nx].v;}
        int mid = (l+r)>>1;
        return max(query1(x1,x2,l,mid,ny,
        seg[ny][nx].l),query1(x1,x2,mid+1,r,ny,
        seg[ny][nx].r));
    }
    int query2(int y1, int y2, int x1, int x2, int l, int r,
    int ny){
        if (ny==0){return 0;}
        if (y2<l || r<y1){return 0;}
        if (y1<=l && r<=y2){return query1(x1,x2,0,N,ny,1);}
        int mid = (l+r)>>1;
        return max(query2(y1,y2,x1,x2,l,mid,L[ny]),
        query2(y1,y2,x1,x2,mid+1,r,R[ny]));
    }
    void update(int y, int x, int val){
        upd2(y,x,val,0,N,1);
    }
    int query(int y1, int x1, int y2, int x2){
        return query2(y1,y2,x1,x2,0,N,1);
    }
};
```

## 2.5 Persistent Segment Tree

**Usage:** 0-based, fixed array implmentation may help memory
**Time Complexity:** $\mathcal{O}(\log N)$

```
struct pst{
    struct Node{
        int l,r,c;
        ll v;
        Node(){l=r=c=v=0;}
    };
    int sz = 1;
    Node seg[20*MAX];
    void update(int i, int inc, int l, int r, int node, int
    prev){
        if (l==r){
            seg[node].c = seg[prev].c + 1;
            seg[node].v = seg[prev].v + inc;
            return;
        }
        int mid = (l+r)>>1;
        seg[node].c = seg[prev].c +1;
        seg[node].v = seg[prev].v + inc;
        ++sz;
        if (i<=mid){
            seg[node].l = sz-1;
```

```
            seg[node].r = seg[prev].r;
            update(i,inc,l,mid,seg[node].l, seg[prev].l);
        }else{
            seg[node].r = sz-1;
            seg[node].l = seg[prev].l;
            update(i,inc,mid+1,r,seg[node].r, seg[prev].r);
        }
    }
    ll vquery(int from, int to, int l, int r, int node){
        if (from>to || node==0 || to<l || r<from){return 0;}
        if (from<=l && r<=to){return seg[node].v;}
        int mid = (l+r)>>1;
        return vquery(from,to,l,mid,seg[node].l) +
        vquery(from,to,mid+1,r,seg[node].r);
    }
    int cquery(int from, int to, int l, int r, int node){
        if (from>to || node==0 || to<l || r<from){return 0;}
        if (from<=l && r<=to){return seg[node].c;}
        int mid = (l+r)>>1;
        return cquery(from,to,l,mid,seg[node].l) +
        cquery(from,to,mid+1,r,seg[node].r);
    }
    // find kth largest element in [L,R]
    // begin with calling rt[R] = hr, hr[L-1] = hl
    int kth(int k, int hl, int hr, int l, int r){
        if (l==r){
            return l;
        }
        int mid = (l+r)>>1;
        int x = seg[seg[hr].r].c - seg[seg[hl].r].c;
        if (x >= k){
            return kth(k,seg[hl].r, seg[hr].r, mid+1,r);
        }else{
            return kth(k-x,seg[hl].l, seg[hr].l, l,mid);
        }
    }
};
```

## 2.6 Merge Sort Tree

**Usage:** Update is unable, 0-based
**Time Complexity:** $\mathcal{O}(N \log N)$ space complexity

```
struct mergetree{
    int N;
    vector<ll> a;
    vector<vector<ll>> seg;

    mergetree(){N=0;}
    mergetree(int n){
        N=n;
        a.resize(N);
        seg.resize(2*N);
    }
    void init(){
        for (int i=0;i<N;i++){seg[i+N].push_back(a[i]);}
        for (int i=N-1;i>=1;i--){
            seg[i].resize(seg[i<<1].size() +
            seg[i<<1|1].size());
            merge(seg[i<<1].begin(), seg[i<<1].end(),
            seg[i<<1|1].begin(), seg[i<<1|1].end(),
            seg[i].begin());
        }
    }
    ll query(int l, int r){
        if (l>r){return 0LL;}
        ll ret=0;
        int lp,rp;
        lp = l+N; rp=r+N+1;
        while(lp<rp){
            if (lp&1){
                // define operation here

                ++lp;
            }
            if (rp&1){
                --rp;
                //define operation here

            }
            lp>>=1; rp>>=1;
        }
        return ret;
    }
};
```

## 2.7 Fenwick Tree

**Usage:** 0-based, only restricted operation
**Time Complexity:** $\mathcal{O}(\log N)$

```
struct fenwick {
    int n;
    vector<int> seg;
    fenwick(int n) {
        this->n = n;
        seg.resize(n);
    }
    int query(int r) {
        if (r<0){return 0;}
        int ret = 0;
        for (; r >= 0; r = (r & (r + 1)) - 1){
            ret += seg[r];
        }
        return ret;
    }
    void update(int i, int inc) {
        for (; i < n; i = i | (i + 1)){
            seg[i] += inc;
        }
    }
    int query(int l, int r){
        if (l>r){return 0;}
        return query(r) - query(l-1);
    }
};
```

## 2.8 Min-Max queue

**Usage:** same operation (`push,empty,pop,front`) with ordinary queue. Able to get min,max from `get_min`, `get_max`
**Time Complexity:** $\mathcal{O}(1)$

```
template <typename T>
struct minmaxqueue{
    int N;
    stack<T> s1,s2,ms1,ms2,Ms1,Ms2;
    minmaxqueue<T>(){N=0;}
    int size(){return N;}
    bool empty(){return (N==0)? true:false;}
    void push(T x){
        if (s2.empty()){ms2.push(x); Ms2.push(x);}
        else{
            if (ms2.top()>x){ms2.push(x);}
            else{ms2.push(ms2.top());}
            if (Ms2.top()<x){Ms2.push(x);}
            else{Ms2.push(Ms2.top());}
        }
        s2.push(x);
        ++N;
    };
    T front(){
        if (s1.empty()){
            while(!ms2.empty()){ms2.pop();}
            while(!Ms2.empty()){Ms2.pop();}
            while(!s2.empty()){
                T tmp = s2.top(); s2.pop();
                if (s1.empty()){ms1.push(tmp); Ms1.push(tmp);}
                else{
                    if (ms1.top()>tmp){ms1.push(tmp);}
                    else{ms1.push(ms1.top());}
                    if (Ms1.top()<tmp){Ms1.push(tmp);}
                    else{Ms1.push(Ms1.top());}
                }
                s1.push(tmp);
            }
        }
        T ret = s1.top();
        return ret;
    };
    void pop(){
        T tmp = front();
        s1.pop(); ms1.pop(); Ms1.pop();
```

```
        --N;
    };
    T getmin(){
        T ret;
        if (!ms1.empty() && !ms2.empty()){ret =
        min(ms1.top(),ms2.top());}
        else if (!ms1.empty()){ret=ms1.top();}
        else{ret=ms2.top();}
        return ret;
    };
    T getmax(){
        T ret;
        if (!Ms1.empty() &&
        !Ms2.empty()){ret=max(Ms1.top(),Ms2.top());}
        else if (!Ms1.empty()){ret=Ms1.top();}
        else{ret=Ms2.top();}
        return ret;
    };
};
```

## 2.9 Sparse Table

**Usage:** 0-based, only restricted operation
**Time Complexity:** $\mathcal{O}(N \log N)$ to build, $\mathcal{O}(1)$ per query

```
struct minsps{
    int N,M;
    vector<ll> a;
    vector<vector<ll>> v;
    minsps(int n){
        N = n;
        M = 31 - __builtin_clz(N);
        a.resize(N);
        v.resize(M+1);
        for (int j=0;j<=M;j++){v[j].resize(N);}
    }
    void init(){
        for(int i=0;i<N;i++){v[0][i] = a[i];}
        for (int j=1;j<=M;j++){
            for(int i=0;i<N;i++){
                if (i+(1<<j-1)<N){
                    v[j][i] = min(v[j-1][i],
                    v[j-1][i+(1<<j-1)]);
                }
            }
        }
    }
    ll query(int l, int r){
        int d = r-l+1;
        int m = 31 - __builtin_clz(d);
        return min(v[m][l], v[m][r+1-(1<<m)]);
    }
};
```

## 2.10 PBDS

**Usage:** Able to query k-th element in set. Beware of multiset erase!!
**Time Complexity:** $\mathcal{O}(\log N)$, but heavy constant

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

// SET PBDS
// #define pbds tree<int, null_type, less<int>,
rb_tree_tag,tree_order_statistics_node_update>
// st.insert(x) : insert x
// st.erase(x) : erase x (if exist)
// st.find_by_order(x) : returns pointer to x-th element
// st.order_of_key(x) : returns number of element smaller than
x
pbds st;
st.insert(3); st.insert(4); // {3,4}
st.insert(5); st.insert(3); // {3,4,5}
st.insert(7); st.insert(3); // {3,4,5,7}
cout<< *st.find_by_order(3); // 7 (3th element)
cout<< st.order_of_key(2); // 0 (element smaller than 2)
cout<< st.order_of_key(5); // 2 (element smaller than 5)
st.erase(3);    // {4,5,7}
st.erase(3);    // {4,5,7}

// MULTISET PBDS
// #define pbds tree<int, null_type, less_equal<int>,
rb_tree_tag,tree_order_statistics_node_update>
// m_erase(st, x) : erase one occurence x (if exist)
void m_erase(pbds& st, int x){
    int p = st.order_of_key(x);
    if
    (*st.find_by_order(p)==x){st.erase(st.find_by_order(p));}
}
pbds st;
st.insert(3); st.insert(4); // {3,4}
st.insert(5); st.insert(3); // {3,3,4,5}
st.insert(7); st.insert(3); // {3,3,3,4,5,7}
cout<< *st.find_by_order(3); // 4 (3th element)
cout<< st.order_of_key(2); // 0 (element smaller than 2)
cout<< st.order_of_key(5); // 4 (element smaller than 5)
m_erase(st,3);  // {3,3,4,5,7};
m_erase(st,2);  // {3,3,4,5,7};
st.erase(3);    // {3,3,4,5,7}; -> wrong usage
```

## 2.11 Splay Tree

**Usage:** Array management; shift, flip, range/point query & update, etc..
**Time Complexity:** $\mathcal{O}(\log N)$

```
struct Node {
    Node* l, * r, * p;
    ll v, sz, sum, mx, mn;
    bool dummy, flip;

    Node(ll _v, Node* p) : v(_v), p(p) {
        l = r = nullptr;
        sum = mx = mn = _v;
        sz = 1, dummy = 0, flip = 0;
    }
    Node(int _v) : Node(_v, nullptr) {}
    Node() : Node(0) {}
    ~Node() {
        if (l) delete l;
        if (r) delete r;
    }
};

struct SplayTree {
    Node* root;
    Node* ptr[1010101]; //node pointer

    void update(Node* x) {
        x->sz = 1;
        x->sum = x->mn = x->mx = x->v;
        if (x->l) {
            x->sz += x->l->sz;
            x->sum += x->l->sum;
            x->mn = min(x->mn, x->l->mn);
            x->mx = max(x->mx, x->l->mx);
        }
        if (x->r) {
            x->sz += x->r->sz;
            x->sum += x->r->sum;
            x->mn = min(x->mn, x->r->mn);
            x->mx = max(x->mx, x->r->mx);
        }
    }

    void push(Node* x) { // lazy propagate
        if (!x->flip) return;
        swap(x->l, x->r);
        if (x->l) x->l->flip = !x->l->flip;
        if (x->r) x->r->flip = !x->r->flip;
        x->flip = 0;
    }

    Node* gather(int s, int e) { //gather [s, e]
        kth(e + 1);
        auto tmp = root;
        kth(s - 1);
        splay(tmp, root);
        return root->r->l;
    }

    void rotate(Node* x) {
```

```
    auto p = x->p;
    Node* y;
    push(p); push(x);
    if (x == p->l) {
        p->l = y = x->r;
        x->r = p;
    }
    else {
        p->r = y = x->l;
        x->l = p;
    }
    x->p = p->p; p->p = x;
    if (y) y->p = p;
    if (x->p) {
        if (p == x->p->l) x->p->l = x;
        else x->p->r = x;
    }
    else root = x;
    update(p); update(x);
}

void splay(Node* x, Node* g = nullptr) {
    Node* y;
    while (x->p != g) {
        Node* p = x->p;
        if (p->p == g) {
            rotate(x); break;
        }
        auto pp = p->p;
        if ((p->l == x) == (pp->l == p)) {
            rotate(p); rotate(x);
        }
        else {
            rotate(x); rotate(x);
        }
    }
    if (!g) root = x;
}

SplayTree() : root() {
    memset(ptr, 0, sizeof ptr);
}
~SplayTree() {
    if (root) delete root;
}

void init(int n) {
    if (root) delete root;
    root = new Node(-inf); //left dummy node
    auto now = root;
    for (int i = 1; i <= n; i++) {
        ptr[i] = now->r = new Node(i, now);
        now = now->r;
    }
    now->r = new Node(inf, now); //right dummy node
    root->dummy = now->r->dummy = 1;
    for (int i = n; i >= 1; i--) update(ptr[i]);
    splay(ptr[n / 2]);
}

void flip(int s, int e) {
    Node* x = gather(s, e);
    x->flip = !x->flip;
}

void shift(int s, int e, int k) {
    Node* x = gather(s, e);
    cout << x->mn << " " << x->mx << " " << x->sum <<
    "\n";
    if (k >= 0) {
        k %= (e - s + 1);
        if (!k) return;
        flip(s, e); flip(s, s + k - 1); flip(s + k, e);
    }
    else {
        k *= -1;
        k %= (e - s + 1);
        if (!k) return;
        flip(s, e); flip(s, e - k); flip(e - k + 1, e);
    }
}

void getidx(int k) {
    splay(ptr[k]);
    cout << root->l->sz << "\n";
}

void kth(int k) { //1-based
    auto now = root;
    push(now);
    while (1) {
        while (now->l && now->l->sz > k) {
            now = now->l; push(now);
        }
        if (now->l) k -= now->l->sz;
        if (!k) break; k--;
        now = now->r;
        push(now);
    }
    splay(now);
}

void print(Node* x) {
    push(x);
    if (x->l) print(x->l);
    if (!x->dummy) cout << x->v << " ";
    if (x->r) print(x->r);
}
};
```

# 3 Math

## 3.1 Equations

$$\begin{aligned} ax + by &= e \\ cx + dy &= f \end{aligned} \Rightarrow \begin{aligned} x &= \frac{ed - bf}{ad - bc} \\ y &= \frac{af - ec}{ad - bc} \end{aligned}$$

In general, given an equation $Ax = b$, the solution to a variable $x_i$ is given by

$$x_i = \frac{\det A_i'}{\det A}$$

where $A_i'$ is $A$ with the $i$'th column replaced by $b$.

## 3.2 Recurrences

If $a_n = c_1 a_{n-1} + \cdots + c_k a_{n-k}$, and $r_1, \ldots, r_k$ are distinct roots of $x^k - c_1 x^{k-1} - \cdots - c_k$, there are $d_1, \ldots, d_k$ s.t.

$$a_n = d_1 r_1^n + \cdots + d_k r_k^n.$$

Non-distinct roots $r$ become polynomial factors, e.g. $a_n = (d_1 n + d_2) r^n$.

## 3.3 Trigonometry

$$\sin(v + w) = \sin v \cos w + \cos v \sin w$$
$$\cos(v + w) = \cos v \cos w - \sin v \sin w$$

$$\tan(v + w) = \frac{\tan v + \tan w}{1 - \tan v \tan w}$$
$$\sin v + \sin w = 2 \sin \frac{v + w}{2} \cos \frac{v - w}{2}$$
$$\cos v + \cos w = 2 \cos \frac{v + w}{2} \cos \frac{v - w}{2}$$
$$(V + W) \tan(v - w)/2 = (V - W) \tan(v + w)/2$$

where $V, W$ are lengths of sides opposite angles $v, w$.

$$a \cos x + b \sin x = r \cos(x - \phi)$$
$$a \sin x + b \cos x = r \sin(x + \phi)$$

where $r = \sqrt{a^2 + b^2}, \phi = \operatorname{atan2}(b, a)$.

## 3.4 Derivatives/Integrals

$$\frac{d}{dx}\arcsin x = \frac{1}{\sqrt{1-x^2}} \qquad \frac{d}{dx}\arccos x = -\frac{1}{\sqrt{1-x^2}}$$
$$\frac{d}{dx}\tan x = 1+\tan^2 x \qquad \frac{d}{dx}\arctan x = \frac{1}{1+x^2}$$
$$\int \tan ax = -\frac{\ln|\cos ax|}{a} \qquad \int x\sin ax = \frac{\sin ax - ax\cos ax}{a^2}$$
$$\int e^{-x^2} = \frac{\sqrt{\pi}}{2}\operatorname{erf}(x) \qquad \int xe^{ax}dx = \frac{e^{ax}}{a^2}(ax-1)$$

## 3.5 Sums

$$c^a + c^{a+1} + \cdots + c^b = \frac{c^{b+1}-c^a}{c-1}, c \neq 1$$

$$1+2+3+\cdots+n = \frac{n(n+1)}{2}$$
$$1^2+2^2+3^2+\cdots+n^2 = \frac{n(2n+1)(n+1)}{6}$$
$$1^3+2^3+3^3+\cdots+n^3 = \frac{n^2(n+1)^2}{4}$$
$$1^4+2^4+3^4+\cdots+n^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$$

## 3.6 Series

$$e^x = 1+x+\frac{x^2}{2!}+\frac{x^3}{3!}+\ldots, (-\infty < x < \infty)$$
$$\ln(1+x) = x-\frac{x^2}{2}+\frac{x^3}{3}-\frac{x^4}{4}+\ldots, (-1 < x \leq 1)$$
$$\sqrt{1+x} = 1+\frac{x}{2}-\frac{x^2}{8}+\frac{2x^3}{32}-\frac{5x^4}{128}+\ldots, (-1 \leq x \leq 1)$$
$$\sin x = x-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}+\ldots, (-\infty < x < \infty)$$
$$\cos x = 1-\frac{x^2}{2!}+\frac{x^4}{4!}-\frac{x^6}{6!}+\ldots, (-\infty < x < \infty)$$

## 3.7 Linear Sieve

**Usage:** prime generator
**Time Complexity:** $\mathcal{O}(n)$

```
// generate prime
vector<int> gen(int N){
    vector<int> mp(N+1);
    vector<int> pr;
    for (int i=2;i<=N;i++){
        if (mp[i]==0){mp[i]=i; pr.push_back(i);}
        for (int j=0;j<pr.size() && i*pr[j]<=N; j++){
            mp[i*pr[j]] = pr[j];
            if (mp[i]==pr[j]){break;}
        }
    }
    return pr;
}
```

## 3.8 Euler Phi

**Usage:** number of coprime numbers with n
**Time Complexity:** $\mathcal{O}(n)$

```
// check phi(n), O(sqrt(n))
ll get(ll n){
    ll phi = n;
    for (ll i=2;i*i<=n;i++){
        if (n%i==0){
            while(n%i==0){n/=i;}
            phi -= phi/i;
        }
    }
    if (n>1){phi-=phi/n;}
    return phi;
}
```

```
// generate every phi, O(n)
vector<int> gen(int n){
    vector<int> ret(n+1),vis(n+1);
    vector<int> prime;
    ret[1] = 1;
    for (int i=2;i<=n;i++){
        if (!vis[i]){
            prime.push_back(i);
            ret[i] = i-1;
        }
        for (auto& p:prime){
            if (i*p>n){break;}
            vis[i*p] = 1;
            if (i%p==0){
                ret[i*p] = ret[i]*p;
                break;
            }else{
                ret[i*p] = ret[i]*ret[p];
            }
        }
    }
    return ret;
}
```

## 3.9 Miller Rabin Primarily Test + Pollad Rho Factorization

**Usage:** `check64` For primarily test(64bit), `factorize` for factorization $(n < 2^{62})$
**Time Complexity:** $\mathcal{O}(B\log n)$ $(B \sim 7)$, $\mathcal{O}(n^{1/4})$

```
typedef unsigned long long ull;
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;
    ull A[] = {2, 325, 9375, 28178, 450775, 9780504,
    1795265022},
        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) { return modmul(x, x, n) + 1; };
    ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;
    while (t++ % 40 || gcd(prd, n) == 1) {
        if (x == y) x = ++i, y = f(x);
        if ((q = modmul(prd, max(x,y) - min(x,y), n))) prd =
        q;
        x = f(x), y = f(f(y));
    }
    return gcd(prd, n);
}
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;
}
```

## 3.10 Extended Euclidean Algorithm

**Usage:** return $g = gcd(a,b)$ and corresponding $(x,y)$ s.t. $ax+by = g$
**Time Complexity:** $\mathcal{O}(\log n)$

```
ll ex_gcd(ll a, ll b, ll& x, ll& y) {
    x = 1, y = 0;
```

```
    ll x1 = 0, y1 = 1, a1 = a, b1 = b;
    while (b1) {
        ll q = a1 / b1;
        tie(x, x1) = make_tuple(x1, x - q * x1);
        tie(y, y1) = make_tuple(y1, y - q * y1);
        tie(a1, b1) = make_tuple(b1, a1 - q * b1);
    }
    return a1;
}
```

### 3.11 Modular Inverse + FastPow

**Usage:** Modular Inverse Division + Exponential by Squaring
**Time Complexity:** $\mathcal{O}(\log n)$

```
ll divide(ll a, ll b, ll mod){
    a%=mod; b%=mod;
    ll x,y;
    ll g = ex_gcd(b,mod,x,y);
    x = (x%mod+mod)%mod;
    return a*x%mod;
}
ll power(ll b, ll e, ll mod) {
  ll ans = 1;
  for (; e; b = b * b % mod, e /= 2)
  if (e & 1) ans = ans * b % mod;
  return ans;
}
```

### 3.12 Chinese Remainder Theorem

**Usage:** return $N$ s.t. $N = a_i \pmod{m_i}$
**Time Complexity:** $\log(n)$

```
ll CRT(vector<ll>& a, vector<ll>& m){
    int n = a.size();
    ll M = 1;
    for (int i=0;i<n;i++){M*=m[i];}
    vector<ll> x(n);
    vector<vector<ll>> r(n,vector<ll>(n));
    for (int j=0;j<n;j++){
        for (int i=0;i<n;i++){
            r[j][i] = divide(1,m[j],m[i]);
        }
    }
    for (int j=0;j<n;j++){
        x[j]=a[j];
        for (int i=0;i<j;i++){
            x[j] = (x[j]-x[i])*r[i][j]%m[j];
            if (x[j]<0){x[j]+=m[j];}
        }
    }
    ll ans,pre; ans=0; pre=1;
    for (int i=0;i<n;i++){
        ans+=x[i]*pre%M; ans%=M;
        pre*=m[i]; pre%=M;
    }
    return ans;
};
```

### 3.13 Chinese Remainder Theorem-2

**Usage:** `crt(a, m, b, n)` computes $x$ such that $x \equiv a \pmod{m}$, $x \equiv b \pmod{n}$. If $|a| < m$ and $|b| < n$, $x$ will obey $0 \le x < \mathrm{lcm}(m, n)$. Assumes $mn < 2^{62}$
**Time Complexity:** $\mathcal{O}(\log n)$

```
ll crt(ll a, ll m, ll b, ll n) {
  if (n > m) swap(a, b), swap(m, n);
  ll x, y, g = ex_gcd(m, n, x, y);
  assert((a - b) % g == 0); // else no solution
  x = (b - a) % n * x % n / g * m + a;
  return x < 0 ? x + m*n/g : x;
}
```

### 3.14 z2 Basis

**Usage:** calculate z2 Basis for set of numbers (a)
**Time Complexity:** $\mathcal{O}(Nd)$

```
vector<ll> get_basis(ll d, vector<ll>& a){
    vector<ll> basis;
    vector<ll> mask(d);
    function<void(ll)> insert=[&](ll x){
        ll tx=x;
        for (ll b=d-1;b>=0;b--){
            if (tx==0){return;}
            if (tx>>b&1){
                if (mask[b]==0){
                    mask[b]=tx;
                    basis.push_back(tx);
                    return;
                }else{
                    tx^=mask[b];
                }
            }
        }
    };

    for (int i=0;i<a.size();i++){insert(a[i]);}
    sort(basis.rbegin(),basis.rend());
    for (int j=1;j<basis.size();j++){
        for (int i=0;i<j;i++){
            if ((basis[i]^basis[j]) <
            basis[i]){basis[i]^=basis[j];}
        }
    }
    return basis;
}
```

### 3.15 Graycode

**Usage:** Generate n-th Graycode + Inverse generate n from graycode
**Time Complexity:** $\mathcal{O}(\log n)$

```
namespace graycode{
    int gen(int n){
        return n^(n>>1);
    }
    int inv_gen(int g){
        int ret=0;
        while(g){
            ret^=g;
            g>>=1;
        }
        return ret;
    }
}
```

### 3.16 Discrete Logarithm

**Usage:** Calculate smallest $n$ s.t. $X^n = Y$
**Time Complexity:** $\mathcal{O}(\sqrt{n})$

```
ll BSGS(ll x, ll y, ll MOD){
    ll k = sqrt(MOD);
    map<ll,ll> mp;
    for (ll i=0;i<k;i++){
        ll div = divide(y,power(x,i,MOD),MOD);
        // mp[x]=i;
        if (mp.find(div)==mp.end()){mp[div]=i;}
    }
    for (ll i=0;i*k<MOD;i++){
        ll div = power(x,i*k,MOD);
        if (mp.find(div)!=mp.end()){
            return mp[div]+i*k;
        }
    }
    return -1;
}
```

### 3.17 Primitive Root

**Usage:** can be found only $n$=2,4,p,2p
**Time Complexity:** $\mathcal{O}(\log n^6)$

```
int generator (int p) {
    vector<int> fact;
    int phi = p-1,  n = phi;
    for (int i=2; i*i<=n; ++i)
        if (n % i == 0) {
            fact.push_back (i);
            while (n % i == 0)
```

```
            n /= i;
        }
    if (n > 1)
        fact.push_back (n);

    for (int res=2; res<=p; ++res) {
        bool ok = true;
        for (size_t i=0; i<fact.size() && ok; ++i)
            ok &= power (res, phi / fact[i], p) != 1;
        if (ok)  return res;
    }
    return -1;
}
```

### 3.18 Floor Sum

**Usage:** $\Sigma\frac{Ai+B}{M}$ from $0 \le i \le n$
**Time Complexity:** $\mathcal{O}(\log\min(a,b))$

```
ll floor_sum(ll a, ll b, ll m, ll n){
    if (a==0){return b/m*(n+1);}
    if (a>=m || b>=m){
        return a/m*n*(n+1)/2 + b/m*(n+1) +
        floor_sum(a%m,b%m,m,n);
    }
    ll x = (a*n+b)/m;
    return x*n - floor_sum(m,m-b-1,a,x-1);
}
```

### 3.19 Mobius Inversion

$$\mu(n) = \begin{cases} 0 & n \text{ is not square free} \\ 1 & n \text{ has even number of prime factors} \\ -1 & n \text{ has odd number of prime factors} \end{cases}$$

Mobius Inversion:

$$g(n) = \sum_{d|n} f(d) \Leftrightarrow f(n) = \sum_{d|n} \mu(d) g(n/d)$$

Other useful formulas/forms:
$\sum_{d|n}\mu(d) = [n=1]$ (very useful)
$g(n) = \sum_{n|d} f(d) \Leftrightarrow f(n) = \sum_{n|d} \mu(d/n) g(d)$
$g(n) = \sum_{1\le m\le n} f(\lfloor\frac{n}{m}\rfloor) \Leftrightarrow f(n) = \sum_{1\le m\le n} \mu(m) g(\lfloor\frac{n}{m}\rfloor)$

**Example 1.** Find out the number of co-prime pairs of integers $(x,y)$ in range $[1,n]$.

**Solution 1.** Notice that the question is the same as asking you the value of

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

Now apply the Möbius inversion on $[gcd(i,j)] = 1$, we have

$$f(n) = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{d|gcd(i,j)}\mu(d)$$

which is the same as

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

Notice that $[d|gcd(i,j)] = [d|i][d|j]$. Therefore

$$f(n) = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{d=1}^{n}[d|i][d|j]\mu(d)$$

We can change the order of summing things up, so

$$f(n) = \sum_{d=1}^{n}\mu(d)\left(\sum_{i=1}^{n}[d|i]\right)\left(\sum_{j=1}^{n}[d|j]\right)$$

We know that $\sum_{i=1}^{n}[d|i] = \sum_{j=1}^{n}[d|j] = \lfloor\frac{n}{d}\rfloor$. Thus

$$f(n) = \sum_{d=1}^{n}\mu(d)\lfloor\frac{n}{d}\rfloor^2$$

### 3.20 Mobius Function

**Usage:** Mobius Function Linear Sieve
**Time Complexity:** $\mathcal{O}(n)$

```
vector<int> get(int n){
    vector<int> ret(n+1),vis(n+1);
    vector<int> prime;
    ret[1] = 1;
    for (int i=2;i<=n;i++){
        if (!vis[i]){
            prime.push_back(i);
            ret[i] = -1;
        }
        for (auto& p:prime){
            if (p*i>n){break;}
            vis[p*i] = 1;
            if (i%p==0){
                ret[p*i] = 0;
                break;
            }else{
                ret[p*i] = -ret[i];
            }
        }
    }
    return ret;
}
```

## 4 Combinatorics

### 4.1 Cycles

Let $g_S(n)$ be the number of $n$-permutations whose cycle lengths all belong to the set $S$. Then

$$\sum_{n=0}^{\infty} g_S(n)\frac{x^n}{n!} = \exp\left(\sum_{n\in S}\frac{x^n}{n}\right)$$

### 4.2 Derangements

Permutations of a set such that none of the elements appear in their original position.

$$D(n) = (n-1)(D(n-1)+D(n-2)) = nD(n-1)+(-1)^n = \left\lfloor\frac{n!}{e}\right\rceil$$

### 4.3 Partition function

Number of ways of writing $n$ as a sum of positive integers, disregarding the order of the summands.

$$p(0)=1,\ p(n) = \sum_{k\in\mathbb{Z}\setminus\{0\}} (-1)^{k+1} p(n-k(3k-1)/2)$$

$$p(n) \sim 0.145/n \cdot \exp(2.56\sqrt{n})$$

| $n$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 20 | 50 | 100 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $p(n)$ | 1 | 1 | 2 | 3 | 5 | 7 | 11 | 15 | 22 | 30 | 627 | ~2e5 | ~2e8 |

### 4.4 Lucas' Theorem

Let $n,m$ be non-negative integers and $p$ a prime. Write $n = n_k p^k + ... + n_1 p + n_0$ and $m = m_k p^k + ... + m_1 p + m_0$. Then $\binom{n}{m} \equiv \prod_{i=0}^{k}\binom{n_i}{m_i}$ (mod $p$).

### 4.5 Stirling numbers of the first kind

Number of permutations on $n$ items with $k$ cycles.

$$c(n,k) = c(n-1,k-1) + (n-1)c(n-1,k),\ c(0,0)=1$$
$$\textstyle\sum_{k=0}^{n} c(n,k)x^k = x(x+1)\dots(x+n-1)$$

$c(8,k) = 8, 0, 5040, 13068, 13132, 6769, 1960, 322, 28, 1$
$c(n,2) = 0, 0, 1, 3, 11, 50, 274, 1764, 13068, 109584, \dots$

### 4.6 Stirling numbers of the second kind

Partitions of $n$ distinct elements into exactly $k$ groups.

$$S(n,k) = S(n-1,k-1) + kS(n-1,k)$$

$$S(n,1) = S(n,n) = 1$$

$$S(n,k) = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j}j^n$$

## 4.7 Eulerian numbers

Number of permutations $\pi \in S_n$ in which exactly $k$ elements are greater than the previous element. $k$ $j$:s s.t. $\pi(j) > \pi(j+1)$, $k+1$ $j$:s s.t. $\pi(j) \geq j$, $k$ $j$:s s.t. $\pi(j) > j$.

$$E(n,k) = (n-k)E(n-1,k-1) + (k+1)E(n-1,k)$$

$$E(n,0) = E(n,n-1) = 1$$

$$E(n,k) = \sum_{j=0}^{k}(-1)^j \binom{n+1}{j}(k+1-j)^n$$

## 4.8 Bell numbers

Total number of partitions of $n$ distinct elements. $B(n) = 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, \ldots$. For $p$ prime,

$$B(p^m + n) \equiv mB(n) + B(n+1) \pmod{p}$$

## 4.9 Labeled unrooted trees

\# on $n$ vertices: $n^{n-2}$
\# on $k$ existing trees of size $n_i$: $n_1 n_2 \cdots n_k n^{k-2}$
\# with degrees $d_i$: $(n-2)!/((d_1-1)!\cdots(d_n-1)!)$

## 4.10 Catalan numbers

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1} = \frac{(2n)!}{(n+1)!n!}$$

$$C_0 = 1,\ C_{n+1} = \frac{2(2n+1)}{n+2}C_n,\ C_{n+1} = \sum C_i C_{n-i}$$

$C_n = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, \ldots$

- sub-diagonal monotone paths in an $n \times n$ grid.
- strings with $n$ pairs of parenthesis, correctly nested.
- binary trees with with $n+1$ leaves (0 or 2 children).
- ordered trees with $n+1$ vertices.
- ways a convex polygon with $n+2$ sides can be cut into triangles by connecting vertices with straight lines.
- permutations of $[n]$ with no 3-term increasing subseq.

# 5 Polynomials and recurrences

## 5.1 Polynomials

```
struct Poly {
  vector<double> a;
  double operator()(double x) const {
    double val = 0;
    for (int i = sz(a); i--;) (val *= x) += a[i];
    return val;
  }
  void diff() {
    rep(i,1,sz(a)) a[i-1] = i*a[i];
    a.pop_back();
  }
  void divroot(double x0) {
    double b = a.back(), c; a.back() = 0;
    for(int i=sz(a)-1; i--;) c = a[i], a[i] = a[i+1]*x0+b,
    b=c;
    a.pop_back();
  }
};
```

## 5.2 PolyRoots

**Usage:** Finds the real roots to a polynomial. Usage: polyRoots(2,-3,1,-1e9,1e9) // solve $x^2 - 3x + 2 = 0$
**Time Complexity:** $O(n^2 \log(1/\epsilon))$

```
vector<double> polyRoots(Poly p, double xmin, double xmax) {
  if (sz(p.a) == 2) { return {-p.a[0]/p.a[1]}; }
  vector<double> ret;
  Poly der = p;
  der.diff();
  auto dr = polyRoots(der, xmin, xmax);
  dr.push_back(xmin-1);
  dr.push_back(xmax+1);
  sort(all(dr));
  rep(i,0,sz(dr)-1) {
    double l = dr[i], h = dr[i+1];
    bool sign = p(l) > 0;
    if (sign ^ (p(h) > 0)) {
      rep(it,0,60) { // while (h - l > 1e-8)
        double m = (l + h) / 2, f = p(m);
        if ((f <= 0) ^ sign) l = m;
        else h = m;
      }
      ret.push_back((l + h) / 2);
    }
  }
  return ret;
}
```

## 5.3 PolyInterpolate

**Usage:** Given $n$ points (x[i], y[i]), computes an n-1-degree polynomial $p$ that passes through them: $p(x) = a[0] * x^0 + \ldots + a[n-1] * x^{n-1}$. For numerical precision, pick $x[k] = c * \cos(k/(n-1) * \pi), k = 0 \ldots n-1$.
**Time Complexity:** $O(n^2)$

```
typedef vector<double> vd;
vd interpolate(vd x, vd y, int n) {
  vd res(n), temp(n);
  rep(k,0,n-1) rep(i,k+1,n)
    y[i] = (y[i] - y[k]) / (x[i] - x[k]);
  double last = 0; temp[0] = 1;
  rep(k,0,n) rep(i,0,n) {
    res[i] += y[k] * temp[i];
    swap(last, temp[i]);
    temp[i] -= last * x[k];
  }
  return res;
}
```

## 5.4 Berlekamp-Massey

**Usage:** Recovers any $n$-order linear recurrence relation from the first $2n$ terms of the recurrence. Useful for guessing linear recurrences after brute-forcing the first terms. Should work on any field, but numerical stability for floats is not guaranteed. Output will have size $\leq n$. berlekampMassey(0, 1, 1, 3, 5, 11) // 1, 2
**Time Complexity:** $O(N^2)$

```
vector<ll> berlekampMassey(vector<ll> s) {
  int n = sz(s), L = 0, m = 0;
  vector<ll> C(n), B(n), T;
  C[0] = B[0] = 1;

  ll b = 1;
  rep(i,0,n) { ++m;
    ll d = s[i] % mod;
    rep(j,1,L+1) d = (d + C[j] * s[i - j]) % mod;
    if (!d) continue;
    T = C; ll coef = d * power(b, mod-2, mod) % mod;
    rep(j,m,n) C[j] = (C[j] - coef * B[j - m]) % mod;
    if (2 * L > i) continue;
    L = i + 1 - L; B = T; b = d; m = 0;
  }
  C.resize(L + 1); C.erase(C.begin());
  for (ll& x : C) x = (mod - x) % mod;
  return C;
}
```

# 6 Matrices

## 6.1 Determinant

**Usage:** Calculates determinant of a matrix.
**Time Complexity:** $\mathcal{O}(n^3)$

```
double det(vector<vector<double>> a) {
  int n = sz(a); double res = 1;
  rep(i,0,n) {
    int b = i;
    rep(j,i+1,n) if (fabs(a[j][i]) > fabs(a[b][i])) b = j;
    if (i != b) swap(a[i], a[b]), res *= -1;
    res *= a[i][i];
    if (res == 0) return 0;
    rep(j,i+1,n) {
      double v = a[j][i] / a[i][i];
      if (v != 0) rep(k,i+1,n) a[j][k] -= v * a[i][k];
    }
```

```
  }
  return res;
}
ll det(vector<vector<ll>> a) {
  int n = sz(a); ll ans = 1;
  rep(i,0,n) {
    rep(j,i+1,n) {
      while (a[j][i] != 0) { // gcd step
        ll t = a[i][i] / a[j][i];
        if (t) rep(k,i,n)
          a[i][k] = (a[i][k] - a[j][k] * t) % mod;
        swap(a[i], a[j]);
        ans *= -1;
      }
    }
    ans = ans * a[i][i] % mod;
    if (!ans) return 0;
  }
  return (ans + mod) % mod;
}
```

## 6.2 Inverse

**Usage:** Invert matrix $A$. Returns rank; result is stored in $A$ unless singular (rank < n)

**Time Complexity:** $\mathcal{O}(n^3)$

```
int matInv(vector<vector<double>>& A) {
  int n = sz(A); vi col(n);
  vector<vector<double>> tmp(n, vector<double>(n));
  rep(i,0,n) tmp[i][i] = 1, col[i] = i;

  rep(i,0,n) {
    int r = i, c = i;
    rep(j,i,n) rep(k,i,n)
      if (fabs(A[j][k]) > fabs(A[r][c]))
        r = j, c = k;
    if (fabs(A[r][c]) < 1e-12) return i;
    A[i].swap(A[r]); tmp[i].swap(tmp[r]);
    rep(j,0,n)
      swap(A[j][i], A[j][c]), swap(tmp[j][i], tmp[j][c]);
    swap(col[i], col[c]);
    double v = A[i][i];
    rep(j,i+1,n) {
      double f = A[j][i] / v;
      A[j][i] = 0;
      rep(k,i+1,n) A[j][k] -= f*A[i][k];
      rep(k,0,n) tmp[j][k] -= f*tmp[i][k];
    }
    rep(j,i+1,n) A[i][j] /= v;
    rep(j,0,n) tmp[i][j] /= v;
    A[i][i] = 1;
  }

  /// forget A at this point, just eliminate tmp backward
  for (int i = n-1; i > 0; --i) rep(j,0,i) {
    double v = A[j][i];
    rep(k,0,n) tmp[j][k] -= v*tmp[i][k];
  }

  rep(i,0,n) rep(j,0,n) A[col[i]][col[j]] = tmp[i][j];
  return n;
}
// requires Math::power(a,b,mod)
int matInv(vector<vector<ll>>& A) {
  int n = sz(A); vi col(n);
  vector<vector<ll>> tmp(n, vector<ll>(n));
  rep(i,0,n) tmp[i][i] = 1, col[i] = i;

  rep(i,0,n) {
    int r = i, c = i;
    rep(j,i,n) rep(k,i,n) if (A[j][k]) {
      r = j; c = k; goto found;
    }
    return i;
found:
    A[i].swap(A[r]); tmp[i].swap(tmp[r]);
    rep(j,0,n) swap(A[j][i], A[j][c]), swap(tmp[j][i],
    tmp[j][c]);
    swap(col[i], col[c]);
    ll v = power(A[i][i], mod - 2, mod);
    rep(j,i+1,n) {
      ll f = A[j][i] * v % mod;
      A[j][i] = 0;
      rep(k,i+1,n) A[j][k] = (A[j][k] - f*A[i][k]) % mod;
      rep(k,0,n) tmp[j][k] = (tmp[j][k] - f*tmp[i][k]) % mod;
    }
    rep(j,i+1,n) A[i][j] = A[i][j] * v % mod;
    rep(j,0,n) tmp[i][j] = tmp[i][j] * v % mod;
    A[i][i] = 1;
  }

  for (int i = n-1; i > 0; --i) rep(j,0,i) {
    ll v = A[j][i];
    rep(k,0,n) tmp[j][k] = (tmp[j][k] - v*tmp[i][k]) % mod;
  }

  rep(i,0,n) rep(j,0,n)
    A[col[i]][col[j]] = tmp[i][j] % mod + (tmp[i][j] < 0 ? mod
    : 0);
  return n;
}
```

## 6.3 Gaussian Elimination + Solve Linear System

**Usage:** Solves $A * x = b$. If there are multiple solutions, an arbitrary one is returned. Returns rank, or -1 if no solutions.

**Time Complexity:** $\mathcal{O}(n^2m)$

```
typedef vector<double> vd;
const double eps = 1e-12;
int solveLinear(vector<vd> A, vd b, vd& x) {
  int n = sz(A), m = sz(x), rank = 0, br, bc;
  if (n) assert(sz(A[0]) == m);
  vi col(m); iota(all(col), 0);

  rep(i,0,n) {
    double v, bv = 0;
    rep(r,i,n) rep(c,i,m)
      if ((v = fabs(A[r][c])) > bv)
        br = r, bc = c, bv = v;
    if (bv <= eps) {
      rep(j,i,n) if (fabs(b[j]) > eps) return -1;
      break;
    }
    swap(A[i], A[br]);
    swap(b[i], b[br]);
    swap(col[i], col[bc]);
    rep(j,0,n) swap(A[j][i], A[j][bc]);
    bv = 1/A[i][i];
    rep(j,i+1,n) {
      double fac = A[j][i] * bv;
      b[j] -= fac * b[i];
      rep(k,i+1,m) A[j][k] -= fac*A[i][k];
    }
    rank++;
  }
  x.assign(m, 0);
  for (int i = rank; i--;) {
    b[i] /= A[i][i];
    x[col[i]] = b[i];
    rep(j,0,i) b[j] -= A[j][i] * b[i];
  }
  return rank; // (multiple solutions if rank < m)
}

typedef bitset<1000> bs;
int solveLinear(vector<bs> A, vi b, bs& x, int m) {
  int n = sz(A), rank = 0, br;
  assert(m <= sz(x));
  vi col(m); iota(all(col), 0);
  rep(i,0,n) {
    for (br=i; br<n; ++br) if (A[br].any()) break;
    if (br == n) {
      rep(j,i,n) if(b[j]) return -1;
      break;
    }
    int bc = (int)A[br]._Find_next(i-1);
    swap(A[i], A[br]);
    swap(b[i], b[br]);
    swap(col[i], col[bc]);
    rep(j,0,n) if (A[j][i] != A[j][bc]) {
```

```
      A[j].flip(i); A[j].flip(bc);
    }
    rep(j,i+1,n) if (A[j][i]) {
      b[j] ^= b[i];
      A[j] ^= A[i];
    }
    rank++;
  }

  x = bs();
  for (int i = rank; i--;) {
    if (!b[i]) continue;
    x[col[i]] = 1;
    rep(j,0,i) b[j] ^= A[j][i];
  }
  return rank; // (multiple solutions if rank < m)
}
```

# 7 String Algorithm

## 7.1 Suffix Array with LCP

**Time Complexity:** $\mathcal{O}(N \log N)$ for SA, $\mathcal{O}(N)$ for lcp_array

```
// get_sa(a,0) -> return suffix array (sa)
// get_sa(a,1) -> return suffix order (c)
vector<int> get_sa(const vector<int>& A, bool order=false){
    vector<int> a = A;
    a.push_back(-1);
    int N = a.size();
    vector<int> sa(N),c(N);
    vector<pi> tmp(N),new_tmp(N);
    for (int i=0;i<N;i++){
        tmp[i]={a[i],i};
    }
    sort(tmp.begin(),tmp.end());
    for (int i = 0; i < N; i++) {
        sa[i] = tmp[i].second;
    }
    c[sa[0]] = 0;
    for (int i = 1; i < N; i++) {
        if (tmp[i].first == tmp[i - 1].first) { c[sa[i]] =
        c[sa[i - 1]]; }
        else { c[sa[i]] = c[sa[i - 1]] + 1; }
    }

    for (ll k = 1; k < N; k <<= 1) {
        vector<pair<pair<int, int>, int>> tmp(N), new_tmp(N);
        for (int i = 0; i < N; i++) {
            tmp[i].first = { c[(sa[i] - k + N) % N],c[sa[i]]
            };
            tmp[i].second = (sa[i] - k + N) % N;
        }
        vector<int> cnt(N), p(N);
        for (int i = 0; i < N; i++) {
            cnt[tmp[i].first.first]++;
        }
        for (int i = 1; i < N; i++) {
            p[i] = p[i - 1] + cnt[i - 1];
            if (p[i]==N) {break;}
        }
        for (int i = 0; i < N; i++) {
            new_tmp[p[tmp[i].first.first]] = tmp[i];
            p[tmp[i].first.first]++;
        }
        swap(tmp, new_tmp);

        for (int i = 0; i < N; i++) { sa[i] = tmp[i].second; }
        c[sa[0]] = 0;
        for (int i = 1; i < N; i++) {
            if (tmp[i].first == tmp[i - 1].first) {
                c[sa[i]] = c[sa[i - 1]];
            }
            else {
                c[sa[i]] = c[sa[i - 1]] + 1;
            }
        }
        if (c[sa[N - 1]] == N - 1) { break; }
    }
    if (order){return c;}
    else{return sa;}
}
vector<int> get_lcp(const vector<int>& A){
    vector<int> sa = get_sa(A);
    vector<int> c = get_sa(A,1);
    vector<int> a = A;
    a.push_back(-1);
    int N = a.size();
    vector<int> lcp(N);

    int d = 0;
    for (int j = 0; j < N - 1; j++) {
        int x = c[j]; int i = sa[x - 1];
        while (a[j + d] == a[i + d]) { ++d; }
        lcp[x] = d;
        d = max(d - 1, 0);
    }
    return lcp;
}
```

## 7.2 KMP + Z

**Usage:** Z : $i$-th element is common prefix of $S$ and $S_{i...|S|}$
**Time Complexity:** $\mathcal{O}(N)$

```
vector<int> getfail(const vector<int>& A){
    int N = A.size();
    vector<int> fail(N);
    fail[0]=0;
    for (int i=1;i<N;i++){
        int j = fail[i-1];
        while(j>0 && A[j]!=A[i]){j=fail[j-1];}
        if (A[j]==A[i]){fail[i]=j+1;}
    }
    return fail;
}
vector<int> match(const vector<int>& T, const vector<int>& P){
    vector<int> ret;
    int N = T.size();
    int M = P.size();
    if (N<M){return ret;}
    vector<int> fail = getfail(P);
    for (int i=0,j=0; i<N;i++){
        while(j>0 && T[i]!=P[j]){
            j = fail[j-1];
        }
        if (T[i]==P[j]){
            if (j==M-1){
                ret.push_back(i-M+1);
                j=fail[j];
            }else{++j;}
        }
    }
    return ret;
}
vector<int> getz(const vector<int>& A){
    int N = A.size();
    vector<int> z(N);
    int l,r; l=r=0;
    for (int i=1;i<N;i++){
        if (i<=r){z[i]=min(r-i+1,z[i-l]);}
        while(i+z[i]<N && A[z[i]]==A[i+z[i]]){++z[i];}
        if (i+z[i]-1>r){l=i; r=i+z[i]-1;}
    }
    return z;
}
```

## 7.3 Manacher

**Usage:** Returns palindromic radius of $S$.
**Time Complexity:** $\mathcal{O}(N)$

```
vector<int> get(string &s){
    // string -> $s#t#r#i#n#g@
    int n = s.length();
    vector<char> a(2*n+1,'#');
    vector<int> d(2*n+1);
    a[0] = '$';
    a.back() = '@';
    for (int i=0;i<n;i++){
        a[2*i+1] = s[i];
    }
```

```
    int l=0,r=1;
    for (int i=0;i<=2*n;i++){
        if (i<=r){
            d[i] = min(r-i, d[l+r-i]);
        }

        while(a[i+d[i]]==a[i-d[i]]){++d[i];}
        if (i+d[i]>r){
            l = i-d[i];
            r = i+d[i];
        }
    }
    for (int i=0;i<=2*n;i++){
        if (i%2){
            if (d[i]%2==0){--d[i];}
        }else{
            if (d[i]%2){--d[i];}
        }
    }
    return d;
}
```

## 7.4 Trie

```
struct Trie{
    int sz;
    vector<int> tmp,leaf;
    vector<vector<int>> next;
    Trie(int a){
        sz=1;
        leaf.push_back(0);
        tmp.resize(a,-1);
        next.push_back(tmp);
    }
    void add(int v, int c){
        next[v][c]=sz;
        leaf.push_back(0);
        next.push_back(tmp);
        ++sz;
    }
    void insert(string& s){
        int n = s.length();
        int cur=0;
        for (int i=0;i<n;i++){
            int c = s[i]-'a';
            if (next[cur][c]==-1){
                add(cur,c);
            }
            cur=next[cur][c];
        }
        leaf[cur]++;
    }
};
```

## 7.5 Aho-Corasick

```
struct ahocorasick{
    int sz;
    vector<int> par,pch,cache1,leaf,cnt,tmp;
    vector<vector<int>> next,cache2;
    ahocorasick(int a){
        sz=1;
        par.push_back(0); pch.push_back(0);
        cache1.push_back(0);
        leaf.push_back(0); cnt.push_back(0);
        tmp.resize(a,-1);
        next.push_back(tmp); cache2.push_back(tmp);
    }
    void add(int v, int c){
        next[v][c]=sz;
        par.push_back(v); pch.push_back(c);
        cache1.push_back(-1);
        leaf.push_back(0); cnt.push_back(-1);
        next.push_back(tmp); cache2.push_back(tmp);
        ++sz;
    }
    void insert(string& s){
        int n = s.length(); int cur=0;
        for (int i=0;i<n;i++){
            int c = s[i]-'a';
            if (next[cur][c]==-1){
                add(cur,c);
            }
            cur = next[cur][c];
        }
        leaf[cur]++;
    }
    int link(int v){
        if (v==0 || par[v]==0){return 0;}
        int& ret = cache1[v];
        if (ret!=-1){return ret;}
        ret = go(link(par[v]),pch[v]);
        return ret;
    }
    int go(int v, int c){
        int& ret = cache2[v][c];
        if (ret!=-1){return ret;}
        if (next[v][c]!=-1){ret = next[v][c];}
        else{
            if (v==0){ret=0;}
            else{ret = go(link(v),c);}
        }
        return ret;
    }
    int count(int v){
        if (v==0){return 0;}
        int& ret = cnt[v];
        if (ret!=-1){return ret;}
        ret = leaf[v]+count(link(v));
        return ret;
    }
};
```

# 8 Flow

## 8.1 Dinic Algorithm

**Usage:** query(s,t) : start from s, end at t
**Time Complexity:** $\mathcal{O}(V^2 * E)$

```
struct maxflow{
    struct edge{
        int u,v;
        ll c,f=0;
        edge(int u, int v, ll c):u(u),v(v),c(c){}
    };

    int N,M=0,s,t;
    vector<edge> E;
    vector<int> level,see;
    vector<vector<int>> adj;
    queue<int> que;

    maxflow(int N):N(N){
        level.resize(N); see.resize(N); adj.resize(N);
    }
    void addEdge(int u, int v, ll c, bool undirected=0){
        if (undirected){
            E.emplace_back(u,v,c);
            E.emplace_back(v,u,c);
        }else{
            E.emplace_back(u,v,c);
            E.emplace_back(v,u,0);
        }
        adj[u].push_back(M);
        adj[v].push_back(M+1);
        M+=2;
    }
    bool bfs(){
        fill(level.begin(),level.end(),-1);
        que.push(s); level[s]=0;
        while(!que.empty()){
            int u = que.front(); que.pop();
            for (int& e:adj[u]){
                int v = E[e].v;
                if (E[e].c-E[e].f>0 && level[v]==-1){
                    level[v] = level[u]+1; que.push(v);
                }
            }
        }
```

```
        return (level[t]!=-1);
    }
    ll dfs(int u, ll flow){
        if (u==t){return flow;}
        if (flow==0){return 0;}

        for (int& i=see[u]; i<(int)adj[u].size();i++){
            int e = adj[u][i];
            ll sp = E[e].c - E[e].f;
            int v = E[e].v;
            if (sp>0 && level[v]==level[u]+1){
                ll f = dfs(v,min(flow,sp));
                if (f>0){
                    E[e].f+=f;
                    E[e^1].f-=f;
                    return f;
                }
            }
        }
        return 0;
    }
    ll query(int s, int t){
        ll ans=0;
        this->s = s; this->t = t;
        while(bfs()){
            fill(see.begin(),see.end(),0);
            while(ll flow = dfs(s,LLINF)){
                ans+=flow;
            }
        }
        return ans;
    }
};
```

## 8.2 MCMF Algorithm

**Usage:** query(s,t) : start from s, end at t.
**Time Complexity:** $\mathcal{O}(VEF)$ - $O(EF)$

```
struct mcmf{
    struct edge{
        int u,v;
        ll c,w,f=0;
        edge(int u, int v, ll c, ll w):u(u),v(v),c(c),w(w){}
    };

    int N,s,t,M=0;
    vector<edge> E;
    vector<int> back;
    vector<ll> dis,flow;
    vector<vector<int>> adj;
    queue<int> que;

    mcmf(int N):N(N){
        adj.resize(N);
        back.resize(N);
        flow.resize(N);
        dis.resize(N);
    }
    void addEdge(int u, int v, ll c, ll w){
        E.emplace_back(u,v,c,w);
        E.emplace_back(v,u,0,-w);
        adj[u].push_back(M);
        adj[v].push_back(M+1);
        M+=2;
    }
    void spfa(){
        fill(back.begin(),back.end(),-1);
        fill(flow.begin(),flow.end(),0);
        fill(dis.begin(),dis.end(),LLINF);

        dis[s]=0; que.push(s); flow[s]=LLINF;
        while(!que.empty()){
            int u = que.front(); que.pop();
            for (int& e:adj[u]){
                int v = E[e].v;
                ll sp = E[e].c - E[e].f;
                ll w = E[e].w;
                if (sp>0 && dis[v]>dis[u]+w){
                    back[v] = e;
                    flow[v] = min(flow[u],sp);
                    dis[v] = dis[u]+w;
                    que.push(v);
                }
            }
        }
    }
    pii query(int s, int t){
        pii ans={0,0};
        this->s = s; this->t = t;
        while(true){
            spfa();
            if (flow[t]==0){return ans;}

            int u = t;
            ll f = flow[t];
            ans.first+=f;
            while(u!=s){
                int e = back[u];
                E[e].f+=f;
                E[e^1].f-=f;
                ll w = E[e].w;
                ans.second+=w*f;
                u = E[e].u;
            }
        }
    }
};
```

## 8.3 Circulation

**Usage:** (u,v,l,r) : u->v with [l,r] edge constraint. If (maxflow > graph.sum) then no solution. First add edge by `addEdge` and set graph by `init`. If there is a feasible solution answer can be constructed by E[e].l + E[e].f
**Time Complexity:** $\mathcal{O}(V^2 * E)$

```
struct circulation{
    struct edge{
        int u,v;
        ll c,l,f=0;
        edge(int u, int v, ll c, ll l):u(u),v(v),c(c),l(l){}
    };

    int N,s,t,M=0;
    ll sum=0;
    vector<ll> d;
    vector<int> level,see;
    vector<vector<int>> adj;
    vector<edge> E;
    queue<int> que;

    circulation(int N): N(N){
        level.resize(N); see.resize(N); d.resize(N);
        adj.resize(N);
    }
    void addEdge(int u, int v, ll l, ll r){
        d[u]+=l;
        d[v]-=l;
        E.emplace_back(u,v,r-l,l);
        E.emplace_back(v,u,0,0);
        adj[u].push_back(M);
        adj[v].push_back(M+1);
        M+=2;
    }
    void init(int new_s, int new_t){
        for (int i=0;i<N;i++){
            if (i==new_s || i==new_t){continue;}
            if (d[i]>0){
                addEdge(i,new_t,0,d[i]);
                sum+=d[i];
            }else if (d[i]<0){
                addEdge(new_s,i,0,-d[i]);
            }
        }
    }
    bool bfs(){
        fill(level.begin(),level.end(),-1);
        level[s]=0; que.push(s);
        while(!que.empty()){
            int u = que.front(); que.pop();
            for (int e:adj[u]){
```

```
                int v = E[e].v;
                ll sp = E[e].c - E[e].f;
                if (sp>0 && level[v]==-1){
                    level[v]=level[u]+1;
                    que.push(v);
                }
            }
        }
        return (level[t]!=-1);
    }
    ll dfs(int u ,ll flow){
        if (u==t){return flow;}
        if (flow==0){return 0;}

        for (int& i=see[u]; i<(int)adj[u].size();i++){
            int e = adj[u][i];
            ll sp = E[e].c - E[e].f;
            int v = E[e].v;
            if (sp>0 && level[v]==level[u]+1){
                ll f = dfs(v,min(flow,sp));
                if (f>0){
                    E[e].f+=f;
                    E[e^1].f-=f;
                    return f;
                }
            }
        }
        return 0;
    }
    ll query(int s, int t){
        this->s=s; this->t=t;
        ll ans = 0;
        while(bfs()){
            fill(see.begin(),see.end(),0);
            while(ll f = dfs(s,LLINF)){
                ans+=f;
            }
        }
        return ans;
    }
};
```

## 8.4 General Matching

**Usage:** mate has matching mate for each vertices
**Time Complexity:** $\mathcal{O}(VE)$

```
vector<int> Blossom(vector<vector<int>>& graph) {
    int n = graph.size(), timer = -1;
    vector<int> mate(n, -1), label(n), parent(n),
                orig(n), aux(n, -1), q;
    auto lca = [&](int x, int y) {
        for (timer++; ; swap(x, y)) {
        if (x == -1) continue;
        if (aux[x] == timer) return x;
        aux[x] = timer;
        x = (mate[x] == -1 ? -1 : orig[parent[mate[x]]]);
        }
    };
    auto blossom = [&](int v, int w, int a) {
        while (orig[v] != a) {
        parent[v] = w; w = mate[v];
        if (label[w] == 1) label[w] = 0, q.push_back(w);
        orig[v] = orig[w] = a; v = parent[w];
        }
    };
    auto augment = [&](int v) {
        while (v != -1) {
        int pv = parent[v], nv = mate[pv];
        mate[v] = pv; mate[pv] = v; v = nv;
        }
    };
    auto bfs = [&](int root) {
        fill(label.begin(), label.end(), -1);
        iota(orig.begin(), orig.end(), 0);
        q.clear();
        label[root] = 0; q.push_back(root);
        for (int i = 0; i < (int)q.size(); ++i) {
        int v = q[i];
        for (auto x : graph[v]) {
            if (label[x] == -1) {
            label[x] = 1; parent[x] = v;
            if (mate[x] == -1)
                return augment(x), 1;
            label[mate[x]] = 0; q.push_back(mate[x]);
            } else if (label[x] == 0 && orig[v] != orig[x]) {
            int a = lca(orig[v], orig[x]);
            blossom(x, v, a); blossom(v, x, a);
            }
        }
        }
        return 0;
    };
    // Time halves if you start with (any) maximal matching.
    for (int i = 0; i < n; i++)
        if (mate[i] == -1)
        bfs(i);
    return mate;
}
```

## 8.5 Hungarian Algorithm

**Usage:** minimum cost matching algorithm. returns pair (cost,assignment[n])
**Time Complexity:** $\mathcal{O}(N^3)$

```
pair<int, vi> hungarian(const vector<vi> &a) {
  if (a.empty()) return {0, {}};
  int n = sz(a) + 1, m = sz(a[0]) + 1;
  vi u(n), v(m), p(m), ans(n - 1);
  rep(i,1,n) {
    p[0] = i;
    int j0 = 0; // add "dummy" worker 0
    vi dist(m, INT_MAX), pre(m, -1);
    vector<bool> done(m + 1);
    do { // dijkstra
      done[j0] = true;
      int i0 = p[j0], j1, delta = INT_MAX;
      rep(j,1,m) if (!done[j]) {
        auto cur = a[i0 - 1][j - 1] - u[i0] - v[j];
        if (cur < dist[j]) dist[j] = cur, pre[j] = j0;
        if (dist[j] < delta) delta = dist[j], j1 = j;
      }
      rep(j,0,m) {
        if (done[j]) u[p[j]] += delta, v[j] -= delta;
        else dist[j] -= delta;
      }
      j0 = j1;
    } while (p[j0]);
    while (j0) { // update alternating path
      int j1 = pre[j0];
      p[j0] = p[j1], j0 = j1;
    }
  }
  rep(j,1,m) if (p[j]) ans[p[j] - 1] = j - 1;
  return {-v[0], ans}; // min cost
}
```

## 8.6 Konig's Theorem

`vertex cover` $S$ is a set of vertices s.t. for every $(u, v)$, $u \in S$ or $v \in S$
`independent set` $S$ is a set of vertices s.t. there is no edge $(u, v)$ s.t. $u \in S$ and $v \in S$
`matching` $E$ is a set of edges s.t. that are not adjacent.
CLAIM : In bigraph, |minimum vertex cover| = |maximum matching|
Construction of minimum vertex cover : Find maximum matching in bigraph, then (unreachable in $A$ + reachable in $B$) = vertex cover. (unreachable iff $level[u] = -1$)
Construction of independent set : $U$ - vertex cover

# 9 Graph/Tree

## 9.1 Strongly Connected Component

**Usage:** 0-based

```
struct SCC{
    int N,cnt,scnt;
    vector<int> dfsn,scc;
    stack<int> stk;
    SCC(vector<vector<int>>& g){
        N = g.size();
        cnt=scnt=0;
```

```
        dfsn.resize(N,-1);
        scc.resize(N,-1);
        for (int i=0;i<N;i++){
            if (dfsn[i]==-1){dfs(g,i);}
        }
    }
    int dfs(vector<vector<int>>& g, int n){
        dfsn[n]=cnt; ++cnt; stk.push(n);
        int ret = dfsn[n];
        for (auto& it:g[n]){
            if (dfsn[it]==-1){ret=min(ret,dfs(g,it));}
            else{
                if (scc[it]==-1){ret=min(ret,dfsn[it]);}
            }
        }
        if (ret==dfsn[n]){
            while(!stk.empty()){
                int tmp = stk.top(); stk.pop();
                scc[tmp]=scnt;
                if (tmp==n){break;}
            }
            scnt++;
        }
        return ret;
    }
};
```

## 9.2 Biconnected Component

**Usage:** 0-based, one bcc is set of edges

```
struct BCC{
  int N,cnt,bcnt;
  vector<int> in;
  stack<pi> stk;
  vector<vector<pi>> bcc;
  BCC(vector<vector<int>>& g){
    N = g.size();
    cnt=bcnt=0;
    in.resize(N,-1);
    for (int i=0;i<N;i++){
      if (in[i]==-1){dfs(g, i,-1);}
    }
  }
  int dfs(vector<vector<int>>& g, int u, int p){
        in[u]=cnt; ++cnt;
        int ret = in[u];
        for (auto& v:g[u]){
            if (p==v){continue;}
            if (in[u]>in[v]){stk.push({u,v});}

            if (in[v]>=0){
                ret = min(ret, in[v]);
            }else{
                int x = dfs(g,v,u);
                if (x>=in[u]){
                    ++bcnt;
                    bcc.emplace_back();
                    while(!stk.empty()){
            pi e = stk.top(); stk.pop();
            bcc[bcnt-1].push_back(e);
            if (e==pi(u,v)){break;}
                    }
                }
                ret = min(ret,x);
            }
        }
        return ret;
  }
};
```

## 9.3 Eulerian Path

**Usage:** 0-based, ans contains Eulerian path starting from vertex 0. g contains edge index. Not vertices

**Time Complexity:** $O(M)$

```
vector<int> e(M);
vector<bool> vis(M);
vector<int> ans; // answer contains tour
stack<int> stk;
vector<vector<int>> g(N);
// g(u) contains adjacent edge index
// !! not adjacent vertex index

stk.push(0); // start from 0
while(!stk.empty()){
  int u = stk.top();
  while(!g[u].empty() && vis[g[u].back()]){g[u].pop_back();}

  if (g[u].empty()){
    stk.pop();
    ans.push_back(u);
    continue;
  }
  int ee = g[u].back(); g[u].pop_back();
  vis[ee]=vis[ee^1]=1;
  int v = e[ee];
  stk.push(v);
}
```

## 9.4 Heavy Light Decomposition

**Usage:** 0-based, Operations should be defined. Type of query depends on VERTEX or EDGE. Useful in path queries

**Time Complexity:** $\mathcal{O}(\log N)\times$ (operation cost)

```
struct HLD{
    int N,cnt;
    vector<int> h,par,pos,head,heavy;
    vector<vector<int>> g;
    HLD(int n){
        N=n; cnt=0;
        h.resize(n,-1);
        par.resize(n,-1);
        pos.resize(n,-1);
        head.resize(n,-1);
        heavy.resize(n,-1);
        g.resize(n);
    }
    void init(int r){
        h[r]=0;
        dfs(r);
        decompose(r,r);
    }
    int dfs(int u){
        int ret = 1;
        int m,c; m=c=-1;
        for (int& v:g[u]){
            if (par[u]==v){continue;}
            h[v]=h[u]+1;
            par[v]=u;
            int sub = dfs(v);
            ret+=sub;
            if (sub>m){
                m=sub; heavy[u]=v;
            }
        }
        return ret;
    }
    void decompose(int u, int h){
        pos[u]=cnt; ++cnt; head[u]=h;
        if (heavy[u]!=-1){
            decompose(heavy[u],h);
        }
        for (int& v:g[u]){
            if (par[u]==v){continue;}
            if (v==heavy[u]){continue;}
            decompose(v,v);
        }
    }
    int query(int u, int v){
        int ret = 0;
        while(head[u]!=head[v]){
            if (h[head[u]]>h[head[v]]){swap(u,v);}
            // ret = seg.query(pos[head[v]],pos[v]);
            // define ds + operation
            v = par[head[v]];
        }
        if (h[u]>h[v]){swap(u,v);}
        // ret = seg.query(pos[u]+1,pos[v]); (edge query)
        // ret = seg.query(pos[u],pos[v])l (vertex query)
        // Above line depends on type of query (edge ? vertex)
```

```
        return ret;
    }
};
```

## 9.5 Centroid Decomposition

**Usage:** Centroid of a Tree is a vertex s.t. every subtrees of centroid is smaller than $\frac{N}{2}$. Height of Centroid tree is bounded by $\log N$. Moreover, path (u,v) of original tree can be thought as path(u,P) + path(P,v) in the centroid tree. (P = LCA(u,v)). Thus, fix one vertex u, every vertices in tree can be partitioned by ancestors of u, and such number is bounded by $\log N$.

**Time Complexity:** $\mathcal{O}(\log N)$

```
struct centroid{
    vector<int> sub,vis,par;
    vector<vector<int>> g;
    int count(int u, int p){
        sub[u] = 1;
        for (int& v:g[u]){
            if (v==p || vis[v]){continue;}
            sub[u] += count(v,u);
        }
        return sub[u];
    }
    int cent(int u, int p, int sz){
        for (int& v:g[u]){
            if (v==p || vis[v]){continue;}
            if (2*sub[v] > sz){return cent(v,u,sz);}
        }
        return u;
    }
    void init(int u, int p){
        int sz = count(u,-1);
        int cen = cent(u,p,sz);
        vis[cen] = 1;
        par[cen] = p;
        for (int& v:g[cen]){
            if (!vis[v]){
                init(v,cen);
            }
        }
    }
};
```

# 10 Tricks

## 10.1 Fast Fourier Transform

**Usage:** fft(a) computes $\hat{f}(k) = \sum_x a[x]\exp(2\pi i \cdot kx/N)$ for all $k$. N must be a power of 2. Useful for convolution: `conv(a, b) = c`, where $c[x] = \sum a[i]b[x-i]$. For convolution of complex numbers or more than two vectors: FFT, multiply pointwise, divide by n, reverse(start+1, end), FFT back. Rounding is safe if $(\sum a_i^2 + \sum b_i^2)\log_2 N < 9 \cdot 10^{14}$ (in practice $10^{16}$; higher for random inputs). Otherwise, use NTT/FFTMod.

**Time Complexity:** $O(N \log N)$ with $N = |A|+|B|$ (1s for $N = 2^{22}$)

```
typedef vector<int> vi;
typedef complex<double> C;
typedef vector<double> vd;
void fft(vector<C>& a) {
    int n = sz(a), L = 31 - __builtin_clz(n);
    static vector<complex<long double>> R(2, 1);
    static vector<C> rt(2, 1);  // (^ 10% faster if double)
    for (static int k = 2; k < n; k *= 2) {
        R.resize(n); rt.resize(n);
        auto x = polar(1.0L, acos(-1.0L) / k);
        rep(i,k,2*k) rt[i] = R[i] = i&1 ? R[i/2] * x : R[i/2];
    }
    vi rev(n);
    rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
    rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k *= 2)
        for (int i = 0; i < n; i += 2 * k) rep(j,0,k) {
            // C z = rt[j+k] * a[i+j+k]; // (25% faster if
            hand-rolled)  /// include-line
            auto x = (double *)&rt[j+k], y = (double
            *)&a[i+j+k];        /// exclude-line
            C z(x[0]*y[0] - x[1]*y[1], x[0]*y[1] + x[1]*y[0]);
            /// exclude-line
            a[i + j + k] = a[i + j] - z;
            a[i + j] += z;
        }
}
vd conv(const vd& a, const vd& b) {
    if (a.empty() || b.empty()) return {};
    vd res(sz(a) + sz(b) - 1);
    int L = 32 - __builtin_clz(sz(res)), n = 1 << L;
    vector<C> in(n), out(n);
    copy(all(a), begin(in));
    rep(i,0,sz(b)) in[i].imag(b[i]);
    fft(in);
    for (C& x : in) x *= x;
    rep(i,0,n) out[i] = in[-i & (n - 1)] - conj(in[i]);
    fft(out);
    rep(i,0,sz(res)) res[i] = imag(out[i]) / (4 * n);
    return res;
}
```

## 10.2 Fast Fourier Transform MOD

**Usage:** Description: Higher precision FFT, can be used for convolutions modulo arbitrary integers as long as $N\log_2 N \cdot \text{mod} < 8.6 \cdot 10^{14}$ (in practice $10^{16}$ or higher). Inputs must be in $[0, \text{mod})$.

**Time Complexity:** $O(N \log N)$, where $N = |A|+|B|$ (twice as slow as NTT or FFT)

```
// Higher Precision FFT (Modular)
typedef vector<ll> vl;
template<int M> vl convMod(const vl &a, const vl &b) {
    if (a.empty() || b.empty()) return {};
    vl res(sz(a) + sz(b) - 1);
    int B=32-__builtin_clz(sz(res)), n=1<<B, cut=int(sqrt(M));
    vector<C> L(n), R(n), outs(n), outl(n);
    rep(i,0,sz(a)) L[i] = C((int)a[i] / cut, (int)a[i] % cut);
    rep(i,0,sz(b)) R[i] = C((int)b[i] / cut, (int)b[i] % cut);
    fft(L), fft(R);
    rep(i,0,n) {
        int j = -i & (n - 1);
        outl[j] = (L[i] + conj(L[j])) * R[i] / (2.0 * n);
        outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n) / 1i;
    }
    fft(outl), fft(outs);
    rep(i,0,sz(res)) {
        ll av = ll(real(outl[i])+.5), cv =
        ll(imag(outs[i])+.5);
        ll bv = ll(imag(outl[i])+.5) + ll(real(outs[i])+.5);
        res[i] = ((av % M * cut + bv) % M * cut + cv) % M;
    }
    return res;
}
```

## 10.3 Number Theoretic Transform

**Usage:** Description: ntt(a) computes $\hat{f}(k) = \sum_x a[x]g^{xk}$ for all $k$, where $g = \text{root}^{(mod-1)/N}$. N must be a power of 2. Useful for convolution modulo specific nice primes of the form $2^a b + 1$, where the convolution result has size at most $2^a$. For arbitrary modulo, see FFTMod. `conv(a, b) = c`, where $c[x] = \sum a[i]b[x-i]$. For manual convolution: NTT the inputs, multiply pointwise, divide by n, reverse(start+1, end), NTT back. Inputs must be in $[0, \text{mod})$.

**Time Complexity:** $O(N \log N)$

```
typedef vector<int> vi;
typedef vector<ll> vl;
const ll mod = (119 << 23) + 1, root = 62; // = 998244353
// For p < 2^30 there is also e.g. 5 << 25, 7 << 26, 479 << 21
// and 483 << 21 (same root). The last two are > 10^9.
void ntt(vl &a) {
  int n = sz(a), L = 31 - __builtin_clz(n);
  static vl rt(2, 1);
  for (static int k = 2, s = 2; k < n; k *= 2, s++) {
    rt.resize(n);
    ll z[] = {1, power(root, mod >> s, mod)};
    rep(i,k,2*k) rt[i] = rt[i / 2] * z[i & 1] % mod;
  }
  vi rev(n);
  rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
  rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
  for (int k = 1; k < n; k *= 2)
    for (int i = 0; i < n; i += 2 * k) rep(j,0,k) {
      ll z = rt[j + k] * a[i + j + k] % mod, &ai = a[i + j];
      a[i + j + k] = ai - z + (z > ai ? mod : 0);
```

```
      ai += (ai + z >= mod ? z - mod : z);
    }
}
vl conv(const vl &a, const vl &b) {
  if (a.empty() || b.empty()) return {};
  int s = sz(a) + sz(b) - 1, B = 32 - __builtin_clz(s), n = 1
  << B;
  int inv = power(n, mod - 2, mod);
  vl L(a), R(b), out(n);
  L.resize(n), R.resize(n);
  ntt(L), ntt(R);
  rep(i,0,n) out[-i & (n - 1)] = (ll)L[i] * R[i] % mod * inv %
  mod;
  ntt(out);
  return {out.begin(), out.begin() + s};
}
```

## 10.4 Monotone Convex Hull Trick

**Usage:** Able to add $mx + n$ lines, query max or minimum. Slopes must be inserted monotone, query x can be monotone

**Time Complexity:** $\mathcal{O}(N)$

```
struct linear{
    ll m,n;
    ld s;
    linear(){;}
    linear(ll _m, ll _n, ld _s):m(_m),n(_n),s(_s){}
    ll val(ll x){return m*x+n;}
    ld intersect(const linear& l){
        return (ld)(l.n-n)/(m-l.m);
    }
};
struct maxhull{
    int r=0;
    vector<linear> line;
    maxhull(){r=0;}
    void add(ll m, ll n){
        if (line.empty()){
            line.push_back({m,n,-LLINF});
        }else{
            bool push=true;
            linear new_l = linear(m, n,-LLINF);
            while(line.size()>0){
                if (line.back().m==new_l.m){
                    if
                    (line.back().n<new_l.n){line.pop_back();}
                    else{return;}
                }else{
                    if
                    (line.back().intersect(new_l)<=line.back().s)
                    {line.pop_back();}
                    else{break;}
                }
            }
            if (line.empty()){new_l.s = -LLINF;}
            else{new_l.s = line.back().intersect(new_l);}
            line.push_back(new_l);
        }
    }
    ll query(ll x){
        r=min(r,(int)line.size()-1);
        while(r+1<line.size() && line[r+1].s<=x){++r;}
        return line[r].val(x);
    }
};
```

## 10.5 Dynamic Convex Hull Trick

**Usage:** Able to add $mx + n$ lines, query max or minimum. Slopes can be inserted dynamic.

**Time Complexity:** $\mathcal{O}(\log N)$

```
struct Line {
  mutable ll k, m, p;
  bool operator<(const Line& o) const { return k < o.k; }
  bool operator<(ll x) const { return p < x; }
};
struct LineContainer : multiset<Line, less<>> {
  // (for doubles, use inf = 1/.0, div(a,b) = a/b)
  static const ll inf = LLONG_MAX;
  ll div(ll a, ll b) { // floored division
    return a / b - ((a ^ b) < 0 && a % b);
  }
  bool isect(iterator x, iterator y) {
    if (y == end()) return x->p = inf, 0;
    if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
    else x->p = div(y->m - x->m, x->k - y->k);
    return x->p >= y->p;
  }
  void add(ll k, ll m) {
    auto z = insert({ k, m, 0 }), y = z++, x = y;
    while (isect(y, z)) z = erase(z);
    if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
    while ((y = x) != begin() && (--x)->p >= y->p)
      isect(x, erase(y));
  }
  ll query(ll x) {
    assert(!empty());
    auto l = *lower_bound(x);
    return l.k * x + l.m;
  }
};
struct minhull{
    LineContainer hull;
    minhull(){;}
    void add(ll m, ll n){
        hull.add(-m,-n);
    }
    ll query(ll x){
        return -hull.query(x);
    }
};
struct maxhull{
    LineContainer hull;
    maxhull(){;}
    void add(ll m, ll n){
        hull.add(m,n);
    }
    ll query(ll x){
        return hull.query(x);
    }
};
```

## 10.6 Divide and Conquer Optimization

**Usage:** optimizes $dp(k, i) = \min dp(k-1, j-1) + c(j, i)$ formula. ex) k-partition dp. Let $opt(i)$ be optimal index of $dp(k, i)$, e.g. $dp(k-1, opt) + c(opt, i)$ is minimum. If $opt(i)$ obeys following rule, able to change calculation order of k-th row.

- $opt(i) \le opt(i + 1)$
- cost function $c(j, i)$ is Monge Array

**Time Complexity:** $\mathcal{O}(KN \log N)$

```
void calc(int l, int r, int optl, int optr){
    if (l>r){return;}
    int mid = (l+r)>>1;
    pair<ll,int> ret = {LLINF,-1};
    for (int i=optl;i<=min(mid,optr);i++){
        ret = min(ret,{dp[i-1]+c(i,mid),i});
    }
    new_dp[mid] = ret.first;
    int opt = ret.second;
    calc(l,mid-1,optl,opt);
    calc(mid+1,r,opt,optr);
}
```

## 10.7 Knuth Optimization

**Usage:** optimizes $dp(s, e) = \min dp(s, i) + dp(i+1, e) + c(s, e)$ formula. ex)range dp. Let $opt(s, e)$ be optimal index of $dp(s, e)$, e.g. $dp(s, opt) + dp(opt + 1, e) + c(s, e)$ is minimum. If $opt(s, e)$ obeys following rule, Knuth optimization is able.

- $opt(s, e - 1) \le opt(s, e) \le opt(s + 1, e)$
- cost function $c(j, i)$ is Monge Array

**Time Complexity:** $\mathcal{O}(N^2)$

```
for (int j=N-1;j>=0;j--){
    for (int i=j+1;i<N;i++){
        pair<ll,int> ret = {LLINF,-1};
        for (int k=opt[j][i-1]; k<=min(opt[j+1][i],i-1);k++){
            ret = min(ret,{dp[j][k] + dp[k+1][i] + c(j,i),k});
```

```
        }
        dp[j][i] = ret.first;
        opt[j][i] = ret.second;
    }
}
```

### 10.8 Monge Array

Let array c is Monge Array iff following inequality holds:
$c(a, c) + c(b, d) \le c(a, d) + c(b, c)$ , $(a < b < c < d)$
Intuition: It is beneficial to operate on smaller ranges than larger ranges.

## 11 Geometry

### 11.1 Triangles

Side lengths: $a, b, c$
Semiperimeter: $p = \dfrac{a+b+c}{2}$
Area: $A = \sqrt{p(p-a)(p-b)(p-c)}$
Circumradius: $R = \dfrac{abc}{4A}$
Inradius: $r = \dfrac{A}{p}$
Length of median (divides triangle into two equal-area triangles): $m_a = \frac{1}{2}\sqrt{2b^2 + 2c^2 - a^2}$

Length of bisector (divides angles in two): $s_a = \sqrt{bc\left[1 - \left(\dfrac{a}{b+c}\right)^2\right]}$

Law of sines: $\dfrac{\sin\alpha}{a} = \dfrac{\sin\beta}{b} = \dfrac{\sin\gamma}{c} = \dfrac{1}{2R}$
Law of cosines: $a^2 = b^2 + c^2 - 2bc\cos\alpha$
Law of tangents: $\dfrac{a+b}{a-b} = \dfrac{\tan\dfrac{\alpha+\beta}{2}}{\tan\dfrac{\alpha-\beta}{2}}$

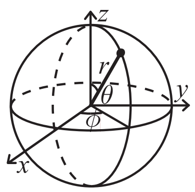
### 11.2 Quadrilaterals

With side lengths $a, b, c, d$, diagonals $e, f$, diagonals angle $\theta$, area $A$ and magic flux $F = b^2 + d^2 - a^2 - c^2$:

$$4A = 2ef \cdot \sin\theta = F\tan\theta = \sqrt{4e^2f^2 - F^2}$$

For cyclic quadrilaterals the sum of opposite angles is $180°$, $ef = ac + bd$, and $A = \sqrt{(p-a)(p-b)(p-c)(p-d)}$.

### 11.3 Spherical coordinates




$$x = r\sin\theta\cos\phi \qquad r = \sqrt{x^2 + y^2 + z^2}$$
$$y = r\sin\theta\sin\phi \qquad \theta = \mathrm{acos}(z/\sqrt{x^2 + y^2 + z^2})$$
$$z = r\cos\theta \qquad \phi = \mathrm{atan2}(y, x)$$

### 11.4 Point

**Usage:** Class to handle points in the plane. T can be e.g. double or long long. (Avoid int.)

```
template <class T> int sgn(T x) { return (x > 0) - (x < 0); }
template<class T>
struct Point {
  typedef Point P;
  T x, y;
  explicit Point(T x=0, T y=0) : x(x), y(y) {}
  bool operator<(P p) const { return tie(x,y) < tie(p.x,p.y);
  }
  bool operator==(P p) const { return tie(x,y)==tie(p.x,p.y);
  }
  P operator+(P p) const { return P(x+p.x, y+p.y); }
  P operator-(P p) const { return P(x-p.x, y-p.y); }
  P operator*(T d) const { return P(x*d, y*d); }
  P operator/(T d) const { return P(x/d, y/d); }
  T dot(P p) const { return x*p.x + y*p.y; }
  T cross(P p) const { return x*p.y - y*p.x; }
  T cross(P a, P b) const { return (a-*this).cross(b-*this); }
  T dist2() const { return x*x + y*y; }
  double dist() const { return sqrt((double)dist2()); }
  // angle to x-axis in interval [-pi, pi]
  double angle() const { return atan2(y, x); }
  P unit() const { return *this/dist(); } // makes dist()=1
  P perp() const { return P(-y, x); } // rotates +90 degrees
  P normal() const { return perp().unit(); }
  // returns point rotated 'a' radians ccw around the origin
  P rotate(double a) const {
    return P(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a)); }
  friend ostream& operator<<(ostream& os, P p) {
    return os << "(" << p.x << "," << p.y << ")"; }
};
```

### 11.5 LineDistance

**Usage:** Returns the signed distance between point p and the line containing points a and b. Positive value on left side and negative on right as seen from a towards b.

```
template<class P>
double lineDist(const P& a, const P& b, const P& p) {
  return (double)(b-a).cross(p-a)/(b-a).dist();
}
```

### 11.6 SegmentDistance

**Usage:** Returns the shortest distance between point p and the line segment from point s to e.

```
typedef Point<double> P;
double segDist(P& s, P& e, P& p) {
  if (s==e) return (p-s).dist();
  auto d = (e-s).dist2(), t = min(d,max(.0,(p-s).dot(e-s)));
  return ((p-s)*d-(e-s)*t).dist()/d;
}
```

### 11.7 SegmentIntersection

**Usage:** Returns intersection point. If none, returns empty vector. If infinitely many, returns a vector with 2 elements.

```
#include "OnSegment.h"

template<class P> vector<P> segInter(P a, P b, P c, P d) {
  auto oa = c.cross(d, a), ob = c.cross(d, b),
       oc = a.cross(b, c), od = a.cross(b, d);
  // Checks if intersection is single non-endpoint point.
  if (sgn(oa) * sgn(ob) < 0 && sgn(oc) * sgn(od) < 0)
    return {(a * ob - b * oa) / (ob - oa)};
  set<P> s;
  if (onSegment(c, d, a)) s.insert(a);
  if (onSegment(c, d, b)) s.insert(b);
  if (onSegment(a, b, c)) s.insert(c);
  if (onSegment(a, b, d)) s.insert(d);
  return {all(s)};
}
```

### 11.8 LineIntersection

**Usage:** Returns 1, point. If none, returns {0, (0, 0)}. If infinitely many, returns {-1, (0, 0)}.

```
template<class P>
pair<int, P> lineInter(P s1, P e1, P s2, P e2) {
  auto d = (e1 - s1).cross(e2 - s2);
  if (d == 0) // if parallel
    return {-(s1.cross(e1, s2) == 0), P(0, 0)};
  auto p = s2.cross(e1, e2), q = s2.cross(e2, s1);
  return {1, (s1 * p + e1 * q) / d};
}
```

### 11.9 SideOf

**Usage:** Returns where p is as seen from s towards e. 1/0/-1 left/on line/right. If the optional argument eps is given 0 is returned if p is within distance eps from the line.

```
template<class P>
int sideOf(P s, P e, P p) { return sgn(s.cross(e, p)); }

template<class P>
int sideOf(const P& s, const P& e, const P& p, double eps) {
  auto a = (e-s).cross(p-s);
  double l = (e-s).dist()*eps;
  return (a > l) - (a < -l);
}
```

## 11.10 OnSegment

**Usage:** Returns true iff p lies on the line segment from s to e. Use (segDist(s,e,p)<=epsilon) instead when using Point<double>.

```
template<class P> bool onSegment(P s, P e, P p) {
  return p.cross(s, e) == 0 && (s - p).dot(e - p) <= 0;
}
```

## 11.11 Angle

**Usage:** A class for ordering angles

```
struct Angle {
  int x, y;
  int t;
  Angle(int x, int y, int t=0) : x(x), y(y), t(t) {}
  Angle operator-(Angle b) const { return {x-b.x, y-b.y, t}; }
  int half() const {
    assert(x || y);
    return y < 0 || (y == 0 && x < 0);
  }
  Angle t90() const { return {-y, x, t + (half() && x >= 0)};
  }
  Angle t180() const { return {-x, -y, t + half()}; }
  Angle t360() const { return {x, y, t + 1}; }
};
bool operator<(Angle a, Angle b) {
  // add a.dist2() and b.dist2() to also compare distances
  return make_tuple(a.t, a.half(), a.y * (ll)b.x) <
         make_tuple(b.t, b.half(), a.x * (ll)b.y);
}

// Given two points, this calculates the smallest angle
between
// them, i.e., the angle that covers the defined line segment.
pair<Angle, Angle> segmentAngles(Angle a, Angle b) {
  if (b < a) swap(a, b);
  return (b < a.t180() ?
          make_pair(a, b) : make_pair(b, a.t360()));
}
Angle operator+(Angle a, Angle b) { // point a + vector b
  Angle r(a.x + b.x, a.y + b.y, a.t);
  if (a.t180() < r) r.t--;
  return r.t180() < a ? r.t360() : r;
}
Angle angleDiff(Angle a, Angle b) { // angle b - angle a
  int tu = b.t - a.t; a.t = b.t;
  return {a.x*b.x + a.y*b.y, a.x*b.y - a.y*b.x, tu - (b < a)};
}
```

## 11.12 CircleIntersection

**Usage:** Computes the pair of points at which two circles intersect. Returns false in case of no intersection.

```
typedef Point<double> P;
bool circleInter(P a,P b,double r1,double r2,pair<P, P>* out)
{
  if (a == b) { assert(r1 != r2); return false; }
  P vec = b - a;
  double d2 = vec.dist2(), sum = r1+r2, dif = r1-r2,
         p = (d2 + r1*r1 - r2*r2)/(d2*2), h2 = r1*r1 - p*p*d2;
  if (sum*sum < d2 || dif*dif > d2) return false;
  P mid = a + vec*p, per = vec.perp() * sqrt(fmax(0, h2) /
  d2);
  *out = {mid + per, mid - per};
  return true;
}
```

## 11.13 CircleTangents

**Usage:** Finds the external tangents of two circles, or internal if r2 is negated. Can return 0, 1, or 2 tangents – 0 if one circle contains the other (or overlaps it, in the internal case, or if the circles are the same); 1 if the circles are tangent to each other (in which case .first = .second and the tangent line is perpendicular to the line between the centers). .first and .second give the tangency points at circle 1 and 2 respectively. To find the tangents of a circle with a point set r2 to 0.

```
template<class P>
vector<pair<P, P>> tangents(P c1, double r1, P c2, double r2)
{
  P d = c2 - c1;
  double dr = r1 - r2, d2 = d.dist2(), h2 = d2 - dr * dr;
  if (d2 == 0 || h2 < 0)  return {};
  vector<pair<P, P>> out;
  for (double sign : {-1, 1}) {
    P v = (d * dr + d.perp() * sqrt(h2) * sign) / d2;
    out.push_back({c1 + v * r1, c2 + v * r2});
  }
  if (h2 == 0) out.pop_back();
  return out;
}
```

## 11.14 CirclePolygonIntersection

**Usage:** Returns the area of the intersection of a circle with a ccw polygon.
**Time Complexity:** $\mathcal{O}(N)$

```
typedef Point<double> P;
#define arg(p, q) atan2(p.cross(q), p.dot(q))
double circlePoly(P c, double r, vector<P> ps) {
  auto tri = [&](P p, P q) {
    auto r2 = r * r / 2;
    P d = q - p;
    auto a = d.dot(p)/d.dist2(), b =
    (p.dist2()-r*r)/d.dist2();
    auto det = a * a - b;
    if (det <= 0) return arg(p, q) * r2;
    auto s = max(0., -a-sqrt(det)), t = min(1., -a+sqrt(det));
    if (t < 0 || 1 <= s) return arg(p, q) * r2;
    P u = p + d * s, v = p + d * t;
    return arg(p,u) * r2 + u.cross(v)/2 + arg(v,q) * r2;
  };
  auto sum = 0.0;
  rep(i,0,sz(ps))
    sum += tri(ps[i] - c, ps[(i + 1) % sz(ps)] - c);
  return sum;
}
```

## 11.15 Circumcircle

**Usage:** ccRadius returns the radius of the circle going through points A, B and C and ccCenter returns the center of the same circle.

```
typedef Point<double> P;
double ccRadius(const P& A, const P& B, const P& C) {
  return (B-A).dist()*(C-B).dist()*(A-C).dist()/
      abs((B-A).cross(C-A))/2;
}
P ccCenter(const P& A, const P& B, const P& C) {
  P b = C-A, c = B-A;
  return A + (b*c.dist2()-c*b.dist2()).perp()/b.cross(c)/2;
}
```

## 11.16 MinimumEnclosingCircle

**Usage:** Computes the minimum circle that encloses a set of points.
**Time Complexity:** $\mathcal{O}(N)$

```
#include "circumcircle.h"

pair<P, double> mec(vector<P> ps) {
  shuffle(all(ps), mt19937(time(0)));
  P o = ps[0];
  double r = 0, EPS = 1 + 1e-8;
  rep(i,0,sz(ps)) if ((o - ps[i]).dist() > r * EPS) {
    o = ps[i], r = 0;
    rep(j,0,i) if ((o - ps[j]).dist() > r * EPS) {
      o = (ps[i] + ps[j]) / 2;
      r = (o - ps[i]).dist();
      rep(k,0,j) if ((o - ps[k]).dist() > r * EPS) {
        o = ccCenter(ps[i], ps[j], ps[k]);
        r = (o - ps[i]).dist();
      }
    }
  }
  return {o, r};
}
```

## 11.17 InsidePolygon

**Usage:** Returns true if p lies within the polygon. If strict is true, it returns false for points on the boundary. The algorithm uses products in intermediate steps so watch out for overflow
**Time Complexity:** $\mathcal{O}(N)$

```
#include "OnSegment.h"
#include "SegmentDistance.h"

template<class P>
bool inPolygon(vector<P> &p, P a, bool strict = true) {
  int cnt = 0, n = sz(p);
  rep(i,0,n) {
    P q = p[(i + 1) % n];
    if (onSegment(p[i], q, a)) return !strict;
    //or: if (segDist(p[i], q, a) <= eps) return !strict;
    cnt ^= ((a.y<p[i].y) - (a.y<q.y)) * a.cross(p[i], q) > 0;
  }
  return cnt;
}
```

## 11.18 PolygonCenter

**Usage:** Returns the center of mass for a polygon.
**Time Complexity:** $\mathcal{O}(N)$

```
typedef Point<double> P;
P polygonCenter(const vector<P>& v) {
  P res(0, 0); double A = 0;
  for (int i = 0, j = sz(v) - 1; i < sz(v); j = i++) {
    res = res + (v[i] + v[j]) * v[j].cross(v[i]);
    A += v[j].cross(v[i]);
  }
  return res / A / 3;
}
```

## 11.19 PolygonCut

**Usage:** Returns a vector with the vertices of a polygon with everything to the left of the line going from s to e cut away.

```
#include "lineIntersection.h"

typedef Point<double> P;
vector<P> polygonCut(const vector<P>& poly, P s, P e) {
  vector<P> res;
  rep(i,0,sz(poly)) {
    P cur = poly[i], prev = i ? poly[i-1] : poly.back();
    bool side = s.cross(e, cur) < 0;
    if (side != (s.cross(e, prev) < 0))
      res.push_back(lineInter(s, e, cur, prev).second);
    if (side)
      res.push_back(cur);
  }
  return res;
}
```

## 11.20 ConvexHull

**Usage:** Returns a vector of the points of the convex hull in counter-clockwise order. Points on the edge of the hull between two other points are not considered part of the hull.
**Time Complexity:** $\mathcal{O}(N \log N)$

```
typedef Point<ll> P;
vector<P> convexHull(vector<P> pts) {
  if (sz(pts) <= 1) return pts;
  sort(all(pts));
  vector<P> h(sz(pts)+1);
  int s = 0, t = 0;
  for (int it = 2; it--; s = --t, reverse(all(pts)))
    for (P p : pts) {
      while (t >= s + 2 && h[t-2].cross(h[t-1], p) <= 0) t--;
      h[t++] = p;
    }
  return {h.begin(), h.begin() + t - (t == 2 && h[0] ==
  h[1])};
}
```

## 11.21 Minkowski Sum

**Usage:** Returns a vector of the points of the Minkowski in counter-clockwise order. Input polygons must be Convex Hull of given set.
**Time Complexity:** $\mathcal{O}(A + B)$

```
typedef Point<ll> P;
void reorder_polygon(vector<P> & pts){
    size_t pos = 0;
    for(size_t i = 1; i < pts.size(); i++){
        if(pts[i].y < pts[pos].y || (pts[i].y == pts[pos].y &&
        pts[i].x < pts[pos].x))
            pos = i;
    }
    rotate(pts.begin(), pts.begin() + pos, pts.end());
}
vector<P> minkowski(vector<P> A, vector<P> B){
    reorder_polygon(A);
    reorder_polygon(B);
    A.push_back(A[0]);
    A.push_back(A[1]);
    B.push_back(B[0]);
    B.push_back(B[1]);
    vector<P> result;
    size_t i = 0, j = 0;
    while(i < A.size() - 2 || j < B.size() - 2){
        result.push_back(A[i] + B[j]);
        auto cross = (A[i + 1] - A[i]).cross(B[j + 1] - B[j]);
        if(cross >= 0 && i < A.size() - 2)
            ++i;
        if(cross <= 0 && j < B.size() - 2)
            ++j;
    }
    return result;
}
```

## 11.22 HullDiameter

**Usage:** Returns the two points with max distance on a convex hull (ccw, no duplicate/collinear points).

```
typedef Point<ll> P;
array<P, 2> hullDiameter(vector<P> S) {
  int n = sz(S), j = n < 2 ? 0 : 1;
  pair<ll, array<P, 2>> res({0, {S[0], S[0]}});
  rep(i,0,j)
    for (;; j = (j + 1) % n) {
      res = max(res, {(S[i] - S[j]).dist2(), {S[i], S[j]}});
      if ((S[(j + 1) % n] - S[j]).cross(S[i + 1] - S[i]) >= 0)
        break;
    }
  return res.second;
}
```

## 11.23 PointInsideHull

**Usage:** Determine whether a point t lies inside a convex hull (CCW order, with no collinear points). Returns true if point lies within the hull. If strict is true, points on the boundary aren't included.
**Time Complexity:** $\mathcal{O}(\log N)$

```
#include "sideOf.h"
#include "OnSegment.h"

typedef Point<ll> P;

bool inHull(const vector<P>& l, P p, bool strict = true) {
  int a = 1, b = sz(l) - 1, r = !strict;
  if (sz(l) < 3) return r && onSegment(l[0], l.back(), p);
  if (sideOf(l[0], l[a], l[b]) > 0) swap(a, b);
  if (sideOf(l[0], l[a], p) >= r || sideOf(l[0], l[b], p)<=
  -r)
    return false;
  while (abs(a - b) > 1) {
    int c = (a + b) / 2;
    (sideOf(l[0], l[c], p) > 0 ? b : a) = c;
  }
  return sgn(l[a].cross(l[b], p)) < r;
}
```

## 11.24 LineHullIntersection

**Usage:** Line-convex polygon intersection. The polygon must be ccw and have no collinear points. lineHull(line, poly) returns a pair describing the intersection of a line with the polygon: • (1, 1) if no collision, • (i, 1) if touching the corner i, • (i, i) if along side (i, i + 1), • (i, j) if crossing sides (i, i + 1) and (j, j + 1). In the last case, if a corner i is crossed, this is treated as happening on side (i, i + 1). The points are returned in the same order as the line hits the polygon. extrVertex returns the point of a hull with the max projection onto a line.
**Time Complexity:** $\mathcal{O}(\log N)$

```
#define cmp(i,j)
sgn(dir.perp().cross(poly[(i)%n]-poly[(j)%n]))
#define extr(i) cmp(i + 1, i) >= 0 && cmp(i, i - 1 + n) < 0
```

```cpp
template <class P> int extrVertex(vector<P>& poly, P dir) {
  int n = sz(poly), lo = 0, hi = n;
  if (extr(0)) return 0;
  while (lo + 1 < hi) {
    int m = (lo + hi) / 2;
    if (extr(m)) return m;
    int ls = cmp(lo + 1, lo), ms = cmp(m + 1, m);
    (ls < ms || (ls == ms && ls == cmp(lo, m)) ? hi : lo) = m;
  }
  return lo;
}

#define cmpL(i) sgn(a.cross(poly[i], b))
template <class P>
array<int, 2> lineHull(P a, P b, vector<P>& poly) {
  int endA = extrVertex(poly, (a - b).perp());
  int endB = extrVertex(poly, (b - a).perp());
  if (cmpL(endA) < 0 || cmpL(endB) > 0)
    return {-1, -1};
  array<int, 2> res;
  rep(i,0,2) {
    int lo = endB, hi = endA, n = sz(poly);
    while ((lo + 1) % n != hi) {
      int m = ((lo + hi + (lo < hi ? 0 : n)) / 2) % n;
      (cmpL(m) == cmpL(endB) ? lo : hi) = m;
    }
    res[i] = (lo + !cmpL(hi)) % n;
    swap(endA, endB);
  }
  if (res[0] == res[1]) return {res[0], -1};
  if (!cmpL(res[0]) && !cmpL(res[1]))
    switch ((res[0] - res[1] + sz(poly) + 1) % sz(poly)) {
      case 0: return {res[0], res[0]};
      case 2: return {res[1], res[1]};
    }
  return res;
}
```

## 11.25 PolygonUnion

**Usage:** Calculates the area of the union of $n$ polygons (not necessarily convex). The points within each polygon must be given in CCW order. (Epsilon checks may optionally be added to sideOf/sgn, but shouldn't be needed.)

**Time Complexity:** $\mathcal{O}(N^2)$

```cpp
#include "sideOf.h"

typedef Point<double> P;
double rat(P a, P b) { return sgn(b.x) ? a.x/b.x : a.y/b.y; }
double polyUnion(vector<vector<P>>& poly) {
  double ret = 0;
  rep(i,0,sz(poly)) rep(v,0,sz(poly[i])) {
    P A = poly[i][v], B = poly[i][(v + 1) % sz(poly[i])];
    vector<pair<double, int>> segs = {{0, 0}, {1, 0}};
    rep(j,0,sz(poly)) if (i != j) {
      rep(u,0,sz(poly[j])) {
        P C = poly[j][u], D = poly[j][(u + 1) % sz(poly[j])];
        int sc = sideOf(A, B, C), sd = sideOf(A, B, D);
        if (sc != sd) {
          double sa = C.cross(D, A), sb = C.cross(D, B);
          if (min(sc, sd) < 0)
            segs.emplace_back(sa / (sa - sb), sgn(sc - sd));
        } else if (!sc && !sd && j<i &&
        sgn((B-A).dot(D-C))>0){
          segs.emplace_back(rat(C - A, B - A), 1);
          segs.emplace_back(rat(D - A, B - A), -1);
        }
      }
    }
    sort(all(segs));
    for (auto& s : segs) s.first = min(max(s.first, 0.0),
    1.0);
    double sum = 0;
    int cnt = segs[0].second;
    rep(j,1,sz(segs)) {
      if (!cnt) sum += segs[j].first - segs[j - 1].first;
      cnt += segs[j].second;
    }
    ret += A.cross(B) * sum;
  }
  return ret / 2;
}
```

## 11.26 ClosestPair

**Usage:** Finds the closest pair of points

**Time Complexity:** $\mathcal{O}(N \log N)$

```cpp
typedef Point<ll> P;
pair<P, P> closest(vector<P> v) {
  assert(sz(v) > 1);
  set<P> S;
  sort(all(v), [](P a, P b) { return a.y < b.y; });
  pair<ll, pair<P, P>> ret{LLONG_MAX, {P(), P()}};
  int j = 0;
  for (P p : v) {
    P d{1 + (ll)sqrt(ret.first), 0};
    while (v[j].y <= p.y - d.x) S.erase(v[j++]);
    auto lo = S.lower_bound(p - d), hi = S.upper_bound(p + d);
    for (; lo != hi; ++lo)
      ret = min(ret, {(*lo - p).dist2(), {*lo, p}});
    S.insert(p);
  }
  return ret.second;
}
```

## 11.27 KdTree

**Usage:** KD-tree (2d, can be extended to 3d)

```cpp
typedef long long T;
typedef Point<T> P;
const T INF = numeric_limits<T>::max();

bool on_x(const P& a, const P& b) { return a.x < b.x; }
bool on_y(const P& a, const P& b) { return a.y < b.y; }

struct Node {
  P pt; // if this is a leaf, the single point in it
  T x0 = INF, x1 = -INF, y0 = INF, y1 = -INF; // bounds
  Node *first = 0, *second = 0;

  T distance(const P& p) { // min squared distance to a point
    T x = (p.x < x0 ? x0 : p.x > x1 ? x1 : p.x);
    T y = (p.y < y0 ? y0 : p.y > y1 ? y1 : p.y);
    return (P(x,y) - p).dist2();
  }

  Node(vector<P>&& vp) : pt(vp[0]) {
    for (P p : vp) {
      x0 = min(x0, p.x); x1 = max(x1, p.x);
      y0 = min(y0, p.y); y1 = max(y1, p.y);
    }
    if (vp.size() > 1) {
      // split on x if width >= height (not ideal...)
      sort(all(vp), x1 - x0 >= y1 - y0 ? on_x : on_y);
      // divide by taking half the array for each child (not
      // best performance with many duplicates in the middle)
      int half = sz(vp)/2;
      first = new Node({vp.begin(), vp.begin() + half});
      second = new Node({vp.begin() + half, vp.end()});
    }
  }
};

struct KDTree {
  Node* root;
  KDTree(const vector<P>& vp) : root(new Node({all(vp)})) {}

  pair<T, P> search(Node *node, const P& p) {
    if (!node->first) {
      // uncomment if we should not find the point itself:
      // if (p == node->pt) return {INF, P()};
      return make_pair((p - node->pt).dist2(), node->pt);
    }

    Node *f = node->first, *s = node->second;
    T bfirst = f->distance(p), bsec = s->distance(p);
    if (bfirst > bsec) swap(bsec, bfirst), swap(f, s);

    // search closest side first, other side if needed
    auto best = search(f, p);
```

```
    if (bsec < best.first)
      best = min(best, search(s, p));
    return best;
  }

  // find nearest point to a point, and its squared distance
  // (requires an arbitrary operator< for Point)
  pair<T, P> nearest(const P& p) {
    return search(root, p);
  }
};
```

## 11.28 FastDelaunay

**Usage:** Fast Delaunay triangulation. Each circumcircle contains none of the input points. There must be no duplicate points. If all points are on a line, no triangles will be returned. Should work for doubles as well, though there may be precision issues in 'circ'. Returns triangles in order {t[0][0], t[0][1], t[0][2], t[1][0], . . . }, all counter-clockwise.

**Time Complexity:** $\mathcal{O}(N \log N)$

```
typedef Point<ll> P;
typedef struct Quad* Q;
typedef __int128_t lll; // (can be ll if coords are < 2e4)
P arb(LLONG_MAX,LLONG_MAX); // not equal to any other point

struct Quad {
  Q rot, o; P p = arb; bool mark;
  P& F() { return r()->p; }
  Q& r() { return rot->rot; }
  Q prev() { return rot->o->rot; }
  Q next() { return r()->prev(); }
} *H;

bool circ(P p, P a, P b, P c) { // is p in the circumcircle?
  lll p2 = p.dist2(), A = a.dist2()-p2,
      B = b.dist2()-p2, C = c.dist2()-p2;
  return p.cross(a,b)*C + p.cross(b,c)*A + p.cross(c,a)*B > 0;
}
Q makeEdge(P orig, P dest) {
  Q r = H ? H : new Quad{new Quad{new Quad{new Quad{0}}}};
  H = r->o; r->r()->r() = r;
  rep(i,0,4) r = r->rot, r->p = arb, r->o = i & 1 ? r :
  r->r();
  r->p = orig; r->F() = dest;
  return r;
}
void splice(Q a, Q b) {
  swap(a->o->rot->o, b->o->rot->o); swap(a->o, b->o);
}
Q connect(Q a, Q b) {
  Q q = makeEdge(a->F(), b->p);
  splice(q, a->next());
  splice(q->r(), b);
  return q;
}

pair<Q,Q> rec(const vector<P>& s) {
  if (sz(s) <= 3) {
    Q a = makeEdge(s[0], s[1]), b = makeEdge(s[1], s.back());
    if (sz(s) == 2) return { a, a->r() };
    splice(a->r(), b);
    auto side = s[0].cross(s[1], s[2]);
    Q c = side ? connect(b, a) : 0;
    return {side < 0 ? c->r() : a, side < 0 ? c : b->r() };
  }

#define H(e) e->F(), e->p
#define valid(e) (e->F().cross(H(base)) > 0)
  Q A, B, ra, rb;
  int half = sz(s) / 2;
  tie(ra, A) = rec({all(s) - half});
  tie(B, rb) = rec({sz(s) - half + all(s)});
  while ((B->p.cross(H(A)) < 0 && (A = A->next())) ||
         (A->p.cross(H(B)) > 0 && (B = B->r()->o)));
  Q base = connect(B->r(), A);
  if (A->p == ra->p) ra = base->r();
  if (B->p == rb->p) rb = base;

#define DEL(e, init, dir) Q e = init->dir; if (valid(e)) \
    while (circ(e->dir->F(), H(base), e->F())) { \
      Q t = e->dir; \
      splice(e, e->prev()); \
      splice(e->r(), e->r()->prev()); \
      e->o = H; H = e; e = t; \
    }
  for (;;) {
    DEL(LC, base->r(), o);  DEL(RC, base, prev());
    if (!valid(LC) && !valid(RC)) break;
    if (!valid(LC) || (valid(RC) && circ(H(RC), H(LC))))
      base = connect(RC, base->r());
    else
      base = connect(base->r(), LC->r());
  }
  return { ra, rb };
}

vector<P> triangulate(vector<P> pts) {
  sort(all(pts));  assert(unique(all(pts)) == pts.end());
  if (sz(pts) < 2) return {};
  Q e = rec(pts).first;
  vector<Q> q = {e};
  int qi = 0;
  while (e->o->F().cross(e->F(), e->p) < 0) e = e->o;
#define ADD { Q c = e; do { c->mark = 1; pts.push_back(c->p);
\
  q.push_back(c->r()); c = c->next(); } while (c != e); }
  ADD; pts.clear();
  while (qi < sz(q)) if (!(e = q[qi++])->mark) ADD;
  return pts;
}
```

## 11.29 HalfplaneIntersection

```
typedef Point<long double> P;

// Redefine epsilon and infinity as necessary. Be mindful of
precision errors.
const long double eps = 1e-9, inf = 1e9;

// Basic half-plane struct.
struct Halfplane {
    // 'p' is a passing point of the line and 'pq' is the
    direction vector of the line.
    P p, pq;
    long double angle;

    Halfplane() {}
    Halfplane(const P& a, const P& b) : p(a), pq(b - a) {
    angle = atan2l(pq.y, pq.x); }

    // Check if point 'r' is outside this half-plane.
    // Every half-plane allows the region to the LEFT of its
    line.
    bool out(const P& r) { return P().cross(pq, r - p) < -eps;
    }

    // Comparator for sorting.
    bool operator < (const Halfplane& e) const { return angle
    < e.angle; }

    // Intersection point of the lines of two half-planes. It
    is assumed they're never parallel.
    friend P inter(const Halfplane& s, const Halfplane& t) {
        long double alpha = P().cross((t.p - s.p), t.pq) /
        P().cross(s.pq, t.pq);
        return s.p + (s.pq * alpha);
    }
};

// Actual algorithm
vector<P> hp_intersect(vector<Halfplane>& H) {

    P box[4] = {  // Bounding box in CCW order
        P(inf, inf),
        P(-inf, inf),
        P(-inf, -inf),
        P(inf, -inf)
    };
```

```
    for (int i = 0; i < 4; i++) { // Add bounding box
    half-planes.
        Halfplane aux(box[i], box[(i + 1) % 4]);
        H.push_back(aux);
    }

    // Sort by angle and start algorithm
    sort(H.begin(), H.end());
    deque<Halfplane> dq;
    int len = 0;
    for (int i = 0; i < int(H.size()); i++) {
        // Remove from the back of the deque while last
        half-plane is redundant
        while (len > 1 && H[i].out(inter(dq[len - 1], dq[len -
        2]))) {
            dq.pop_back();
            --len;
        }
        // Remove from the front of the deque while first
        half-plane is redundant
        while (len > 1 && H[i].out(inter(dq[0], dq[1]))) {
            dq.pop_front();
            --len;
        }
        // Special case check: Parallel half-planes
        if (len > 0 && fabsl(P().cross(H[i].pq, dq[len -
        1].pq)) < eps) {
            // Opposite parallel half-planes that ended up
            checked against each other.
            if (H[i].pq.dot(dq[len - 1].pq) < 0.0)
                return vector<P>();
            // Same direction half-plane: keep only the
            leftmost half-plane.
            if (H[i].out(dq[len - 1].p)) {
                dq.pop_back();
                --len;
            }
            else continue;
        }
        // Add new half-plane
        dq.push_back(H[i]);
        ++len;
    }

    // Final cleanup: Check half-planes at the front against
    the back and vice-versa
    while (len > 2 && dq[0].out(inter(dq[len - 1], dq[len -
    2]))) {
        dq.pop_back();
        --len;
    }
    while (len > 2 && dq[len - 1].out(inter(dq[0], dq[1]))) {
        dq.pop_front();
        --len;
    }

    // Report empty intersection if necessary
    if (len < 3) return vector<P>();

    // Reconstruct the convex polygon from the remaining
    half-planes.
    vector<P> ret(len);
    for (int i = 0; i + 1 < len; i++) {
        ret[i] = inter(dq[i], dq[i + 1]);
    }
    ret.back() = inter(dq[len - 1], dq[0]);
    return ret;
}
```

## 11.30 Point3D

**Usage:** Class to handle points in 3D space. T can be e.g. double or long long.

```
template<class T> struct Point3D {
  typedef Point3D P;
  typedef const P& R;
  T x, y, z;
  explicit Point3D(T x=0, T y=0, T z=0) : x(x), y(y), z(z) {}
  bool operator<(R p) const {
    return tie(x, y, z) < tie(p.x, p.y, p.z); }
  bool operator==(R p) const {
    return tie(x, y, z) == tie(p.x, p.y, p.z); }
  P operator+(R p) const { return P(x+p.x, y+p.y, z+p.z); }
  P operator-(R p) const { return P(x-p.x, y-p.y, z-p.z); }
  P operator*(T d) const { return P(x*d, y*d, z*d); }
  P operator/(T d) const { return P(x/d, y/d, z/d); }
  T dot(R p) const { return x*p.x + y*p.y + z*p.z; }
  P cross(R p) const {
    return P(y*p.z - z*p.y, z*p.x - x*p.z, x*p.y - y*p.x);
  }
  T dist2() const { return x*x + y*y + z*z; }
  double dist() const { return sqrt((double)dist2()); }
  //Azimuthal angle (longitude) to x-axis in interval [-pi,
  pi]
  double phi() const { return atan2(y, x); }
  //Zenith angle (latitude) to the z-axis in interval [0, pi]
  double theta() const { return atan2(sqrt(x*x+y*y),z); }
  P unit() const { return *this/(T)dist(); } //makes dist()=1
  //returns unit vector normal to *this and p
  P normal(P p) const { return cross(p).unit(); }
  //returns point rotated 'angle' radians ccw around axis
  P rotate(double angle, P axis) const {
    double s = sin(angle), c = cos(angle); P u = axis.unit();
    return u*dot(u)*(1-c) + (*this)*c - cross(u)*s;
  }
};
```

## 11.31 3dHull

**Usage:** Computes all faces of the 3-dimension hull of a point set. *No four points must be coplanar*, or else random results will be returned. All faces will point outwards

**Time Complexity:** $\mathcal{O}(N^2)$

```
typedef Point3D<double> P3;

struct PR {
  void ins(int x) { (a == -1 ? a : b) = x; }
  void rem(int x) { (a == x ? a : b) = -1; }
  int cnt() { return (a != -1) + (b != -1); }
  int a, b;
};

struct F { P3 q; int a, b, c; };

vector<F> hull3d(const vector<P3>& A) {
  assert(sz(A) >= 4);
  vector<vector<PR>> E(sz(A), vector<PR>(sz(A), {-1, -1}));
#define E(x,y) E[f.x][f.y]
  vector<F> FS;
  auto mf = [&](int i, int j, int k, int l) {
    P3 q = (A[j] - A[i]).cross((A[k] - A[i]));
    if (q.dot(A[l]) > q.dot(A[i]))
      q = q * -1;
    F f{q, i, j, k};
    E(a,b).ins(k); E(a,c).ins(j); E(b,c).ins(i);
    FS.push_back(f);
  };
  rep(i,0,4) rep(j,i+1,4) rep(k,j+1,4)
    mf(i, j, k, 6 - i - j - k);

  rep(i,4,sz(A)) {
    rep(j,0,sz(FS)) {
      F f = FS[j];
      if(f.q.dot(A[i]) > f.q.dot(A[f.a])) {
        E(a,b).rem(f.c);
        E(a,c).rem(f.b);
        E(b,c).rem(f.a);
        swap(FS[j--], FS.back());
        FS.pop_back();
      }
    }
    int nw = sz(FS);
    rep(j,0,nw) {
      F f = FS[j];
#define C(a, b, c) if (E(a,b).cnt() != 2) mf(f.a, f.b, i,
f.c);
      C(a, b, c); C(a, c, b); C(b, c, a);
    }
  }
  for (F& it : FS) if ((A[it.b] - A[it.a]).cross(
```

```
    A[it.c] - A[it.a]).dot(it.q) <= 0) swap(it.c, it.b);
  return FS;
};
```

# 12 Misc

## 12.1 FASTIO

```
#pragma once

inline char gc() { // like getchar()
    static char buf[1 << 16];
    static size_t bc, be;
    if (bc >= be) {
        buf[0] = 0, bc = 0;
        be = fread(buf, 1, sizeof(buf), stdin);
    }
    return buf[bc++]; // returns 0 on EOF
}

int readInt() {
    int a, c;
    while ((a = gc()) < 40);
    if (a == '-') return -readInt();
    while ((c = gc()) >= 48) a = a * 10 + c - 480;
    return a - 48;
}
```