

# Contents

|   |           |
|---|-----------|
| <b>1 Basic</b>                              | <b>1</b>  |
| 1.1 Default Code . . . . .                  | 1         |
| 1.2 .vimrc . . . . .                        | 2         |
| 1.3 PBDS . . . . .                          | 2         |
| 1.4 Random . . . . .                        | 2         |
| 1.5 Clock . . . . .                         | 2         |
| <b>2 Data Structure</b>                     | <b>2</b>  |
| 2.1 Binary Indexed Tree . . . . .           | 2         |
| 2.2 Disjoint Set Union-Find . . . . .       | 2         |
| 2.3 Segment Tree . . . . .                  | 2         |
| 2.4 Dynamic Segment Tree . . . . .          | 3         |
| 2.5 Treap . . . . .                         | 4         |
| <b>3 Graph</b>                              | <b>5</b>  |
| 3.1 Dijkstra . . . . .                      | 5         |
| 3.2 Floyd-Warshall . . . . .                | 6         |
| 3.3 Kruskal . . . . .                       | 6         |
| 3.4 Tarjan SCC . . . . .                    | 6         |
| 3.5 SPFA . . . . .                          | 6         |
| 3.6 Block-cut Tree . . . . .                | 7         |
| <b>4 String</b>                             | <b>7</b>  |
| 4.1 KMP . . . . .                           | 7         |
| 4.2 Z Value . . . . .                       | 8         |
| 4.3 Longest Palindromic Substring . . . . . | 8         |
| 4.4 Suffix Array . . . . .                  | 8         |
| <b>5 Math and Geometry</b>                  | <b>9</b>  |
| 5.1 Vector Operations . . . . .             | 9         |
| 5.2 Convex Hull . . . . .                   | 9         |
| 5.3 Prime Sieve . . . . .                   | 9         |
| 5.4 XOR Basis . . . . .                     | 10        |
| <b>6 DP Trick</b>                           | <b>10</b> |
| 6.1 Dynamic Convex Hull . . . . .           | 10        |
| <b>7 Numbers and Math Formulae</b>          | <b>10</b> |
| 7.1 Fibonacci . . . . .                     | 10        |
| 7.2 Catalan . . . . .                       | 11        |
| 7.3 Geometry . . . . .                      | 11        |
| 7.4 Prime Numbers . . . . .                 | 11        |
| 7.5 Number Theory . . . . .                 | 11        |
| 7.6 Combinatorics . . . . .                 | 11        |

## 1 Basic

### 1.1 Default Code

```
#include <bits/stdc++.h>
#include <bits/extc++.h>

#define StarBurstStream ios_base::
    sync_with_stdio(false); cin.tie(0);
    cout.tie(0);
```

```
#define iter(a) a.begin(), a.end()
#define riter(a) a.rbegin(), a.rend()
#define lsort(a) sort(iter(a))
#define gsort(a) sort(riter(a))
#define pb(a) push_back(a)
#define eb(a) emplace_back(a)
#define pf(a) push_front(a)
#define ef(a) emplace_front(a)
#define pob pop_back()
#define pof pop_front()
#define mp(a, b) make_pair(a, b)
#define F first
#define S second
#define mt make_tuple
#define gt(t, i) get<i>(t)
#define iceil(a, b) (((a) + (b) - 1) / (
    b))
#define tomax(a, b) ((a) = max((a), (b))
    )
#define tomin(a, b) ((a) = min((a), (b))
    )
#define topos(a) ((a) = (((a) % MOD +
    MOD) % MOD))
#define uni(a) a.resize(unique(iter(a))
    - a.begin())
#define printv(a, b) {bool pvaspace=
    false; \
    for(auto pva : a){ \
        if(pvaspace) b << " "; pvaspace=true;\
        b << pva;\
    }\
    b << "\n";}

using namespace std;
using namespace __gnu_pbds;

typedef long long ll;
typedef unsigned long long ull;
typedef long double ld;

using pii = pair<int, int>;
using pll = pair<ll, ll>;
using pdd = pair<ld, ld>;
using tiii = tuple<int, int, int>;

const ll MOD = 1000000007;
const ll MAX = 2147483647;
```

```
template<typename A, typename B>
ostream& operator<<(ostream& o, pair<A,
    B> p){
    return o << '(' << p.F << ',' << p.S
        << ')';
}

int main(){
```

```

    StarBurstStream

    return 0;
}

```

## 1.2 .vimrc

```

:set nu
:set ai
:set cursorline
:set tabstop=4
:set shiftwidth=4
:set mouse=a
:set expandtab
hi CursorLine cterm=none ctermbg=
    DarkMagenta

```

## 1.3 PBDS

```

tree<int, int, less<>, rb_tree_tag,
    tree_order_statistics_node_update> tr
;
tr.order_of_key(123);
tr.find_by_order(123);

```

## 1.4 Random

```

mt19937 rnd(chrono::steady_clock::now().
    time_since_epoch().count());
uniform_int_distribution<int> dis(1,
    100);
cout << dis(rnd) << "\n";

```

## 1.5 Clock

```

int st = clock();

int ed = clock();
if(ed - st >= CLOCKS_PER_SEC * 1);

```

# 2 Data Structure

## 2.1 Binary Indexed Tree

```

template<typename T>
struct BIT{

private:
    vector<T> bit;
    int lowbit(int x){
        return x & (-x);
    }

public:
    explicit BIT(int sz){
        bit.resize(sz + 1);
    }
}

```

```

void modify(int x, T v){
    for(; x < bit.size(); x += lowbit(x))
        bit[x] += v;
}

```

```

T get(int x){
    T ans = T();
    for(; x; x -= lowbit(x)) ans += bit[
        x];
    return ans;
}
};

```

## 2.2 Disjoint Set Union-Find

```

vector<int> dsu, rk;

void initDSU(int n){
    dsu.resize(n);
    rk.resize(n);
    for(int i = 0; i < n; i++) dsu[i] = i,
        rk[i] = 1;
}

int findDSU(int x){
    if(dsu[x] == x) return x;
    dsu[x] = findDSU(dsu[x]);
    return dsu[x];
}

void unionDSU(int a, int b){
    int pa = findDSU(a), pb = findDSU(b);
    if(rk[pa] > rk[pb]) swap(pa, pb);
    if(rk[pa] == rk[pb]) rk[pb]++;
    dsu[pa] = pb;
}

```

## 2.3 Segment Tree

```

template<typename T>
struct Node{
    T v = 0, tag = 0;
    int sz = 1, l = -1, r = -1;
    T rv(){
        return v + tag * sz;
    }
    void addTag(T t){
        tag += t;
    }
};

template<typename T>
T pullValue(T b, T c){
    return b + c;
}

template<typename T>

```

```

void pull(Node<T> &a, Node<T> &l, Node<T>
    > &r){
    a.v = pullValue(l.rv(), r.rv());
    a.sz = l.sz + r.sz;
}

template<typename T>
void push(Node<T> &a, Node<T> &l, Node<T>
    > &r){
    l.addTag(a.tag);
    r.addTag(a.tag);
    a.v = a.rv();
    a.tag = 0;
}

template<typename T>
struct SegmentTree{
    vector<Node<T>> st;
    int cnt = 0;

    explicit SegmentTree(int sz){
        st.resize(4 * sz);
    }

    int build(int l, int r, vector<T>& o){
        int id = cnt++;
        if(l == r){
            st[id].v = o[l];
            return id;
        }
        int m = (l + r) / 2;
        st[id].l = build(l, m, o);
        st[id].r = build(m + 1, r, o);
        pull(st[id], st[st[id].l], st[st[id]
            ].r]);
        return id;
    }

    void modify(int l, int r, int v, int L
        , int R, int id){
        if(l == L && r == R){
            st[id].addTag(v);
            return;
        }
        int M = (L + R) / 2;
        if(r <= M) modify(l, r, v, L, M, st[
            id].l);
        else if(l > M) modify(l, r, v, M +
            1, R, st[id].r);
        else{
            modify(l, M, v, L, M, st[id].l);
            modify(M + 1, r, v, M + 1, R, st[
                id].r);
        }
        pull(st[id], st[st[id].l], st[st[id]
            ].r]);
    }
}

```

```

    }

    T query(int l, int r, int L, int R,
        int id){
        if(l == L && r == R) return st[id].
            rv();
        push(st[id], st[st[id].l], st[st[id]
            ].r]);
        int M = (L + R) / 2;
        if(r <= M) return query(l, r, L, M,
            st[id].l);
        else if(l > M) return query(l, r, M
            + 1, R, st[id].r);
        else{
            return pullValue(query(l, M, L, M,
                st[id].l), query(M + 1, r, M + 1, R,
                    st[id].r));
        }
    }
};

```

## 2.4 Dynamic Segment Tree

```

template<typename T>
struct Node{
    T v = T(), tag = T();
    int l = -1, r = -1;
    int lr = -1, rr = -1;
    T rv(){
        return v + tag * (rr - lr + 1);
    }
    void addTag(T t){
        tag += t;
    }
};

template<typename T>
T pullValue(T b, T c){
    return b + c;
}

template<typename T>
struct SegmentTree{
    vector<Node<T>> st;
    int cnt = 0;

    explicit SegmentTree(int sz){
        st.resize(sz);
    }

    int node(int l, int r){
        int id = cnt++;
        st[id].lr = l;
        st[id].rr = r;
        return id;
    }
}

```

```

void pull(int id){
    st[id].v = pullValue(st[id].l == -1
        ? T() : st[st[id].l].rv(), st[id].r
        == -1 ? T() : st[st[id].r].rv());
}

void push(int id, int L, int R){
    int M = (L + R) / 2;
    if(st[id].l == -1) st[id].l = node(L,
        M);
    st[st[id].l].addTag(st[id].tag);
    if(st[id].r == -1) st[id].r = node(M
        + 1, R);
    st[st[id].r].addTag(st[id].tag);
    st[id].v = st[id].rv();
    st[id].tag = T();
}

int modify(int l, int r, T v, int L,
    int R, int id){
    if(id == -1) id = node(L, R);
    if(l == L && r == R){
        st[id].addTag(v);
        return id;
    }
    int M = (L + R) / 2;
    if(r <= M) st[id].l = modify(l, r, v,
        L, M, st[id].l);
    else if(l > M) st[id].r = modify(l,
        r, v, M + 1, R, st[id].r);
    else{
        st[id].l = modify(l, M, v, L, M,
            st[id].l);
        st[id].r = modify(M + 1, r, v, M +
            1, R, st[id].r);
    }
    pull(id);
    return id;
}

T query(int l, int r, int L, int R,
    int id){
    if(id == -1) return T();
    if(l == L && r == R) return st[id].
        rv();
    push(id, L, R);
    int M = (L + R) / 2;
    if(r <= M) return query(l, r, L, M,
        st[id].l);
    else if(l > M) return query(l, r, M
        + 1, R, st[id].r);
    else{
        return pullValue(query(l, M, L, M,
            st[id].l), query(M + 1, r, M + 1, R,
            st[id].r));
    }
}

```

```

    }
}
};

```

## 2.5 Treap

```

mt19937 rnd(chrono::steady_clock::now().
    time_since_epoch().count());

template<typename T>
struct Node{
    int l = -1, r = -1, pri = rnd(), sz =
        1;
    T v, sum, tag;
    T rsum(){
        return sum + tag * sz;
    }
};

template<typename T>
struct Treap{
    vector<Node<T>> tr;
    int ts = 0;
    explicit Treap(int sz){
        tr.resize(sz);
    }

    int node(T v){
        int r = ts++;
        tr[r].v = v;
        tr[r].sum = 0;
        tr[r].tag = 0;
        return r;
    }

    void pull(int r){
        if(r != -1){
            tr[r].sz = 1;
            tr[r].sum = tr[r].v;
            if(tr[r].l != -1){
                tr[r].sum += tr[tr[r].l].rsum();
                tr[r].sz += tr[tr[r].l].sz;
            }
            if(tr[r].r != -1){
                tr[r].sum += tr[tr[r].r].rsum();
                tr[r].sz += tr[tr[r].r].sz;
            }
        }
    }

    void push(int r){
        if(r == -1) return;
        if(tr[r].l != -1){
            tr[tr[r].l].tag += tr[r].tag;
        }
        if(tr[r].r != -1){

```

```

        tr[tr[r].r].tag += tr[r].tag;
    }
    tr[r].sum = tr[r].rsum();
    tr[r].v += tr[r].tag;
    tr[r].tag = 0;
}

void merge(int a, int b, int& r){
    push(a);
    push(b);
    if(a == -1 && b == -1) r = -1;
    else if(a == -1) r = b;
    else if(b == -1) r = a;
    else{
        if(tr[a].pri > tr[b].pri){
            r = a;
            merge(tr[a].r, b, tr[a].r);
        }
        else{
            r = b;
            merge(a, tr[b].l, tr[b].l);
        }
    }
    pull(r);
}

void split1(int a, T k, int& r1, int&
r2){
    if(a == -1){
        r1 = r2 = -1;
        return;
    }
    push(a);
    if(tr[a].v < k){
        r1 = a;
        split1(tr[a].r, k, tr[a].r, r2);
    }
    else{
        r2 = a;
        split1(tr[a].l, k, r1, tr[a].l);
    }
    pull(a);
}

void split2(int a, int k, int& r1, int
& r2){
    if(a == -1){
        r1 = r2 = -1;
        return;
    }
    push(a);
    if(k == 0){
        r1 = -1;
        r2 = a;
        return;
    }

```

```

        if(tr[a].l == -1 || tr[tr[a].l].sz <
k){
            r1 = a;
            if(tr[a].l != -1) split2(tr[a].r,
k - tr[tr[a].l].sz - 1, tr[a].r, r2);
            else split2(tr[a].r, k - 1, tr[a].
r, r2);
        }
        else{
            r2 = a;
            split2(tr[a].l, k, r1, tr[a].l);
        }
        pull(a);
    }

    void printtr(int now){
        if(now == -1) return;
        printtr(tr[now].l);
        cerr << now << "," << tr[now].v + tr
[now].tag << "," << tr[now].rsum() <<
", " << tr[now].tag << " ";
        printtr(tr[now].r);
    }

    void print(int r){
        printtr(r);
        cerr << "\n";
    }

};

```

### 3 Graph

#### 3.1 Dijkstra

```

//The first element in pair should be
    edge weight, and the second should be
    vertex
vector<vector<pii>> g;
int n;

int dijkstra(int start, int end){
    priority_queue<pii, vector<pii>,
greater<pii>> q;
    for(pii p : g[start]){
        q.push(p);
    }
    q.push(mp(0, start));
    vector<int> dis(n, -1);
    dis[start] = 0;
    vector<int> visit(n);
    while(q.size()){
        int v = q.top().S;
        int d = q.top().F;
        if(v == end) break;
        q.pop();
        if(visit[v]) continue;
    }
}

```

```

    visit[v] = true;
    for(pii p : g[v]){
        if(visit[p.S]) continue;
        if(dis[p.S] == -1 || d + p.F < dis
[p.S]){
            dis[p.S] = d + p.F;
            q.push(mp(dis[p.S], p.S));
        }
    }
}
return dis[end];
}

```

### 3.2 Floyd-Warshall

```

vector<vector<int>> g;
int n;

void floydwarshall(){
    for(int k = 0; k < n; k++)
        for(int i = 0; i < n; i++)
            for(int j = 0; j < n; j++)
                if(g[i][k] != -MAX && g[k][j] !=
-MAX && (g[i][j] == -MAX || g[i][k]
+ g[k][j] < g[i][j]))
                    g[i][j] = g[i][k] + g[k][j];
}

```

### 3.3 Kruskal

```

int kruskal(){
    int ans = 0;
    lsort(e);
    initDSU();
    for(auto& i : e){
        int a = i.S.F, b = i.S.S;
        if(findDSU(a) == findDSU(b))
            continue;
        ans += i.F;
        unionDSU(a, b);
    }
    return ans;
}

```

### 3.4 Tarjan SCC

```

vector<vector<int>> g;
vector<int> st;
vector<bool> inst;
vector<int> scc;
vector<int> ts, low;
int tmp = 0;
int sccid = 0;

void initSCC(int n){
    tmp = 0;
    sccid = 0;
    st.clear();
}

```

```

g.clear();
g.resize(2 * n + 1);
inst.clear();
inst.resize(2 * n + 1);
scc.clear();
scc.resize(2 * n + 1);
ts.clear();
ts.resize(2 * n + 1, -1);
low.clear();
low.resize(2 * n + 1);
}

```

```
void dfs(int now){
```

```

    st.pb(now);
    inst[now] = true;
    ts[now] = ++tmp;
    low[now] = ts[now];

    for(int i : g[now]){
        if(ts[i] == -1){
            dfs(i);
            low[now] = min(low[now], low[i]);
        }
        else if(inst[i]) low[now] = min(low[
now], ts[i]);
    }
}

```

```

if(low[now] == ts[now]){
    sccid++;
    int t;
    do{
        t = st.back();
        st.pop();
        inst[t] = false;
        scc[t] = sccid;
    }
    while(t != now);
}
}

```

### 3.5 SPFA

```

const ll INFINITE = 2147483647;

int n;
vector<vector<pii>> g;

int spfa(int start, int end){

    vector<int> dis(n, INFINITE);
    int start;
    cin >> start;
    dis[start] = 0;

    queue<int> q;
}

```

```

q.push(start);
vector<bool> inq(n);
inq[start] = true;
vector<int> cnt(n);

while(!q.empty()){
    int v = q.front();
    q.pop();
    inq[v] = false;
    for(pii p : g[v]){
        if(!(dis[p.F] == INFINITE || dis[v]
] + p.S < dis[p.F])) continue;
        cnt[p.F]++;
        if(cnt[p.F] >= n) return -INFINITE
; //negative cycle
        dis[p.F] = dis[v] + p.S;
        if(!inq[p.F]){
            inq[p.F] = true;
            q.push(p.F);
        }
    }
}

return dis[end];
}

```

### 3.6 Block-cut Tree

```

#include <bits/stdc++.h>

#define eb(a) emplace_back(a)

using namespace std;

// tg is the origin graph, g is the
// result
vector<vector<int>> tg, g;
int bcc; // = n+1, initially
vector<int> low, in;
int tts = 1;
stack<int> st;
vector<vector<int>> c;
vector<bool> iscut;

void dfsbcc(int now, int p){ //
    calculate low
    low[now] = in[now] = tts++;
    for(int i : tg[now]){
        if(i == p) continue;
        if(in[i]) low[now] = min(low[now],
in[i]);
        else{
            dfsbcc(i, now);
            low[now] = min(low[now], low[i]);
            c[now].eb(i);
        }
    }
    if(low[i] >= in[now] && now != 1)

```

```

        iscut[now] = true;
    }
    if(now == 1 && c[now].size() > 1)
        iscut[now] = true;
}

void dfsbcc2(int now, int p){ // build
    block-cut tree
    st.push(now);
    for(int i : c[now]){
        dfsbcc2(i, now);
    }
    if(now == 1){
        if(st.size() > 1){
            while(!st.empty()){
                g[st.top()].eb(bcc);
                g[bcc].eb(st.top());
                st.pop();
            }
            bcc++;
        }
    }
    else if((p != 1 && low[now] >= in[p])
|| (p == 1 && c[p].size() > 1)){
        while(!st.empty()){
            int t = st.top();
            g[st.top()].eb(bcc);
            g[bcc].eb(st.top());
            st.pop();
            if(t == now) break;
        }
        g[bcc].eb(p);
        g[p].eb(bcc);
        bcc++;
    }
}

```

## 4 String

### 4.1 KMP

```

vector<int> f;
void build(string& t){
    f.clear();
    f.resize(t.size());
    int p = -1;
    f[0] = -1;
    for(int i = 1; i < t.size(); i++){
        while(p != -1 && t[p + 1] != t[i]) p
= f[p];
        if(t[p + 1] == t[i]) f[i] = p + 1;
        else f[i] = -1;
        p = f[i];
    }
}

int kmp(string& s, string& t){

```

```

int ans = 0;
int p = -1;
for(int i = 0; i < s.size(); i++){
    while(p != -1 && t[p + 1] != s[i]) p
    = f[p];
    if(t[p + 1] == s[i]) p++;
    if(p + 1 == t.size()){
        ans++;
        p = f[p];
    }
}
return ans;
}

```

## 4.2 Z Value

```

vector<int> z;

void build(string s, int n){
    z.clear();
    z.resize(n);
    int l = 0;
    for(int i = 1; i < n; i++){
        if(l + z[l] >= i) z[i] = min(z[l] +
        l - i, z[i - 1]);
        while(i + z[i] < n && s[z[i]] == s[i
        + z[i]]) z[i]++;
        if(i + z[i] > l + z[l]) l = i;
    }
}

```

## 4.3 Longest Palindromic Substring

```

#define T(x) ((x) % 2 ? s[(x) / 2] : '.'
    )

string s;
int L;

int ex(int l, int r){
    int i = 0;
    while(l - i >= 0 && r + i < L && T(l -
    i) == T(r + i)) i++;
    return i;
}

int lps(string ss){
    s = ss;
    L = 2 * s.size() + 1;

    int mx = 0;
    int center = 0;
    vector<int> r(L);
    int ans = 1;
    r[0] = 1;
    for(int i = 1; i < L; i++){
        int ii = center - (i - center);

```

```

int len = mx - i + 1;
if(i > mx){
    r[i] = ex(i, i);
    center = i;
    mx = i + r[i] - 1;
}
else if(r[ii] == len){
    r[i] = len + ex(i - len, i + len);
    center = i;
    mx = i + r[i] - 1;
}
else r[i] = min(r[ii], len);
ans = max(ans, r[i]);
}

return ans - 1;
}

```

## 4.4 Suffix Array

```

#include <bits/stdc++.h>

#define eb(a) emplace_back(a)

using namespace std;

vector<int> sa(string s){
    s += '$';
    int n = s.size();
    int t = __lg(n) + 1;

    vector<vector<int>>> rk(t + 1, vector<
    int>(n)), b;

    vector<vector<int>>> c1(27);
    for(int i = 0; i < n; i++) c1[s[i] ==
    '$' ? 0 : s[i] - 'a' + 1].eb(i);
    for(int i = 0; i < 27; i++){
        if(!c1[i].empty()) b.eb(c1[i]);
    }
    b.resize(n);
    for(int i = 0; i < n; i++){
        for(int k : b[i]) rk[0][k] = i;
    }

    for(int i = 1; i <= t; i++){
        vector<vector<int>>> tb(n);
        for(int j = 0; j < n; j++){
            for(int k : b[j]){
                int tmp = ((k - (1 << (i - 1)))
                % n + n) % n;
                int now = rk[i - 1][tmp];
                tb[now].eb(tmp);
            }
        }
        b = tb;
        int cnt = -1;

```



```

    for(int j = 0; j < n; j++){
        int lst = -1;
        for(int k : b[j]){
            int now = rk[i - 1][(k + (1 << (
i - 1))) % n];
            if(now != lst) cnt++;
            rk[i][k] = cnt;
            lst = now;
        }
    }

    return rk[t];
}

```

## 5 Math and Geometry

### 5.1 Vector Operations

```

template<typename T>
pair<T, T> operator+(pair<T, T> a, pair<
T, T> b){
    return mp(a.F + b.F, a.S + b.S);
}

template<typename T>
pair<T, T> operator-(pair<T, T> a, pair<
T, T> b){
    return mp(a.F - b.F, a.S - b.S);
}

template<typename T>
pair<T, T> operator*(pair<T, T> a, T b){
    return mp(a.F * b, a.S * b);
}

template<typename T>
pair<T, T> operator/(pair<T, T> a, T b){
    return mp(a.F / b, a.S / b);
}

template<typename T>
T dot(pair<T, T> a, pair<T, T> b){
    return a.F * b.F + a.S * b.S;
}

template<typename T>
T cross(pair<T, T> a, pair<T, T> b){
    return a.F * b.S - a.S * b.F;
}

template<typename T>
T abs2(pair<T, T> a){
    return a.F * a.F + a.S * a.S;
}

```

### 5.2 Convex Hull

```

template<typename T>
pair<T, T> operator-(pair<T, T> a, pair<
T, T> b){
    return mp(a.F - b.F, a.S - b.S);
}

template<typename T>
T cross(pair<T, T> a, pair<T, T> b){
    return a.F * b.S - a.S * b.F;
}

template<typename T>
vector<pair<T, T>> getConvexHull(vector<
pair<T, T>>& pnts){

    int n = pnts.size();
    lsort(pnts);

    vector<pair<T, T>> hull;
    hull.reserve(n);

    for(int i = 0; i < 2; i++){
        int t = hull.size();
        for(pair<T, T> pnt : pnts){
            while(hull.size() - t >= 2 &&
cross(hull.back() - hull[hull.size()
- 2], pnt - hull[hull.size() - 2]) <=
0){
                hull.pop_back();
            }
            hull.pb(pnt);
        }
        hull.pop_back();
        reverse(iter(pnts));
    }

    return hull;
}

```

### 5.3 Prime Sieve

```

vector<int> prime;
vector<int> p;
void sieve(int n){
    prime.resize(n + 1, 1);
    for(int i = 2; i <= n; i++){
        if(prime[i] == 1){
            p.push_back(i);
            prime[i] = 0;
        }
        for(int j : p){
            if((ll)i * j > n || j > prime[i])
                break;
            prime[i * j] = 0;
        }
    }
}

```

## 5.4 XOR Basis

```
const int mxdigit = 50;

vector<ll> b(mxdigit + 1);

void add(ll t){
    for(int i = mxdigit; i >= 0; i--){
        if(!(1LL << i & t)) continue;
        if(b[i] != 0){
            t ^= b[i];
            continue;
        }
        for(int j = 0; j < i; j++){
            if(1LL << j & t) t ^= b[j];
        }
        for(int j = i + 1; j <= mxdigit; j++){
            if(1LL << i & b[j]) b[j] ^= t;
        }
        b[i] = t;
        break;
    }
}
```

## 6 DP Trick

### 6.1 Dynamic Convex Hull

```
const ll INF = 1LL << 60;

template<typename T>
struct Line{
    mutable T a, b, r = 0;

    Line(T a, T b) : a(a), b(b){}

    bool operator<(Line<T> l) const{
        return a < l.a;
    }

    bool operator<(T v) const{
        return r < v;
    }
};

template<typename T>
T divfloor(T a, T b){
    return a / b - ((a ^ b) < 0 && a % b);
}

template<typename T>
struct DynamicHull{
    multiset<Line<T>, less<>> s;

    int size(){
        return s.size();
    }
};
```

```
}

bool intersect(typename set<Line<T>>::
    iterator a, typename set<Line<T>>::
    iterator &b){
    if(b == s.end()){
        a->r = INF;
        return false;
    }
    if(a->a == b->a){
        if(a->b > b->b) a->r = INF;
        else a->r = -INF;
    }
    else{
        a->r = divfloor(b->b - a->b, a->a
            - b->a);
    }
    return a->r >= b->r;
}

void insert(T a, T b){
    Line<T> l(a, b);
    auto it = s.insert(l), after = next(
        it), before = it;
    while(intersect(it, after)) after =
        s.erase(after);
    if(before != s.begin() && intersect(
        (--before, it)){
        it = s.erase(it);
        intersect(before, it);
    }
    while((it = before) != s.begin() &&
        (--before)->r >= it->r) intersect(
        before, it = s.erase(it));
}

T query(T v){
    Line<T> l = *s.lower_bound(v);
    return l.a * v + l.b;
}
};
```

## 7 Numbers and Math Formulae

### 7.1 Fibonacci

$$f(n) = f(n-1) + f(n-2)$$

$$\begin{bmatrix} f(n) \\ f(n-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

|    |         |         |         |          |
|----|---------|---------|---------|----------|
| 1  | 1       | 1       | 2       | 3        |
| 5  | 5       | 8       | 13      | 21       |
| 9  | 34      | 55      | 89      | 144      |
| 13 | 233     | 377     | 610     | 987      |
| 17 | 1597    | 2584    | 4181    | 6765     |
| 21 | 10946   | 17711   | 28657   | 46368    |
| 25 | 75025   | 121393  | 196418  | 317811   |
| 29 | 514229  | 832040  | 1346269 | 2178309  |
| 33 | 3524578 | 5702887 | 9227465 | 14930352 |

$$f(45) \approx 10^9$$

$$f(88) \approx 10^{18}$$

## 7.2 Catalan

$$C_0 = 1, C_n = \sum_{i=0}^{n-1} C_i C_{n-1-i}$$

$$C_n = C_n^{2n} - C_{n-1}^{2n}$$

|    |        |        |         |         |
|----|--------|--------|---------|---------|
| 0  | 1      | 1      | 2       | 5       |
| 4  | 14     | 42     | 132     | 429     |
| 8  | 1430   | 4862   | 16796   | 58786   |
| 12 | 208012 | 742900 | 2674440 | 9694845 |

## 7.3 Geometry

- Heron's formula:  
The area of a triangle whose lengths of sides is  $a, b, c$  and  $s = (a + b + c)/2$  is  $\sqrt{s(s-a)(s-b)(s-c)}$ .
- Vector cross product:  
 $v_1 \times v_2 = |v_1||v_2| \sin \theta = (x_1 \times y_2) - (x_2 \times y_1)$ .
- Vector dot product:  
 $v_1 \cdot v_2 = |v_1||v_2| \cos \theta = (x_1 \times y_1) + (x_2 \times y_2)$ .

## 7.4 Prime Numbers

First 50 prime numbers:

|    |     |     |     |     |     |
|----|-----|-----|-----|-----|-----|
| 1  | 2   | 3   | 5   | 7   | 11  |
| 6  | 13  | 17  | 19  | 23  | 29  |
| 11 | 31  | 37  | 41  | 43  | 47  |
| 16 | 53  | 59  | 61  | 67  | 71  |
| 21 | 73  | 79  | 83  | 89  | 97  |
| 26 | 101 | 103 | 107 | 109 | 113 |
| 31 | 127 | 131 | 137 | 139 | 149 |
| 36 | 151 | 157 | 163 | 167 | 173 |
| 41 | 179 | 181 | 191 | 193 | 197 |
| 46 | 199 | 211 | 223 | 227 | 229 |

Very large prime numbers:

|            |            |            |
|------------|------------|------------|
| 1000001333 | 1000500889 | 2500001909 |
| 2000000659 | 900004151  | 850001359  |

## 7.5 Number Theory

- Inversion:  
 $aa^{-1} \equiv 1 \pmod{m}$ .  $a^{-1}$  exists iff  $\gcd(a, m) = 1$ .
- Linear inversion:  
 $a^{-1} \equiv (m - \lfloor \frac{m}{a} \rfloor) \times (m \bmod a)^{-1} \pmod{m}$
- Fermat's little theorem:  
 $a^p \equiv a \pmod{p}$  if  $p$  is prime.
- Euler function:  
 $\phi(n) = n \prod_{p|n} \frac{p-1}{p}$
- Euler theorem:  
 $a^{\phi(n)} \equiv 1 \pmod{n}$  if  $\gcd(a, n) = 1$ .
- Extended Euclidean algorithm:  
 $ax + by = \gcd(a, b) = \gcd(b, a \bmod b) = \gcd(b, a - \lfloor \frac{a}{b} \rfloor b) = bx_1 + (a - \lfloor \frac{a}{b} \rfloor b)y_1 = ay_1 + b(x_1 - \lfloor \frac{a}{b} \rfloor y_1)$
- Divisor function:  
 $\sigma_x(n) = \sum_{d|n} d^x$ .  $n = \prod_{i=1}^r p_i^{a_i}$ .  
 $\sigma_x(n) = \prod_{i=1}^r \frac{p_i^{(a_i+1)x} - 1}{p_i^x - 1}$  if  $x \neq 0$ .  $\sigma_0(n) = \prod_{i=1}^r (a_i + 1)$ .
- Chinese remainder theorem:  
 $x \equiv a_i \pmod{m_i}$ .  
 $M = \prod m_i$ .  $M_i = M/m_i$ .  $t_i = M_i^{-1}$ .  
 $x = kM + \sum a_i t_i M_i$ ,  $k \in \mathbb{Z}$ .

## 7.6 Combinatorics

- $P_k^n = \frac{n!}{(n-k)!}$
- $C_k^n = \frac{n!}{(n-k)!k!}$
- $H_k^n = C_k^{n+k-1} = \frac{(n+k-1)!}{k!(n-1)!}$