

Contents

| | | | |
|--|-----------|--|-----------|
| 1 Basic | 2 | 9 Problems | 21 |
| 1.1 vimrc | 2 | 9.1 Manhattan distance minimum spanning tree | 21 |
| 1.2 Compilation Argument | 2 | 9.2 "Dynamic" kth element (parallel binary search) | 22 |
| 1.3 Checker | 2 | 9.3 Dynamic kth element (persistent segment tree) | 22 |
| 1.4 Fast Integer Input | 2 | 9.4 Hilbert's curve (faster MO's algorithm) | 23 |
| 1.5 IncStack | 2 | | |
| 1.6 Pragma optimization | 2 | | |
| 2 Flow | 2 | | |
| 2.1 Dinic | 2 | | |
| 2.2 ISAP | 2 | | |
| 2.3 Minimum-cost flow | 3 | | |
| 2.4 Gomory-Hu Tree | 3 | | |
| 2.5 Stoer-Wagner minimum cut | 3 | | |
| 2.6 Hungarian ($O(n^3)$) | 4 | | |
| 2.7 Hungarian ($O(n^4)$) | 4 | | |
| 2.8 Flow Model | 4 | | |
| 3 Data Structure | 5 | | |
| 3.1 Disjoint Set | 5 | | |
| 3.2 <ext/pbds> | 5 | | |
| 3.3 Li Chao Tree | 5 | | |
| 4 Graph | 6 | | |
| 4.1 Link-Cut Tree | 6 | | |
| 4.2 Heavy-Light Decomposition | 6 | | |
| 4.3 Centroid Decomposition | 7 | | |
| 4.4 Minimum mean cycle | 7 | | |
| 4.5 Minimum Steiner Tree | 7 | | |
| 4.6 Maximum Clique | 8 | | |
| 4.7 Tarjan's articulation point | 8 | | |
| 4.8 Tarjan's bridge | 8 | | |
| 5 String | 8 | | |
| 5.1 KMP | 8 | | |
| 5.2 Z algorithm | 9 | | |
| 5.3 Manacher's | 9 | | |
| 5.4 Aho-Corasick Automaton | 9 | | |
| 5.5 Suffix Automaton | 9 | | |
| 5.6 Suffix Array | 10 | | |
| 5.7 SAIS | 10 | | |
| 5.8 DC3 | 11 | | |
| 5.9 Smallest Rotation | 11 | | |
| 6 Math | 11 | | |
| 6.1 Fast Fourier transform | 11 | | |
| 6.2 Number theoretic transform | 12 | | |
| 6.2.1 NTT Prime List | 13 | | |
| 6.3 Fast Walsh-Hadamard transform | 13 | | |
| 6.4 Lagrange Interpolation | 13 | | |
| 6.5 Miller Rabin | 13 | | |
| 6.6 Pollard's rho | 14 | | |
| 6.7 Prime counting | 14 | | |
| 6.8 Gaussian Elimination | 14 | | |
| 6.9 Linear Equations (full pivoting) | 14 | | |
| 6.10 μ function | 14 | | |
| 6.11 $\lfloor \frac{n}{i} \rfloor$ Enumeration | 15 | | |
| 6.12 Extended GCD | 15 | | |
| 6.13 Chinese remainder theorem | 15 | | |
| 6.14 Lucas's theorem | 15 | | |
| 6.15 Kirchhoff's theorem | 15 | | |
| 6.16 Tutte matrix | 15 | | |
| 6.17 Primes | 15 | | |
| 7 Dynamic Programming | 15 | | |
| 7.1 Convex Hull (monotone) | 15 | | |
| 7.2 Convex Hull (non-monotone) | 15 | | |
| 7.3 1D/1D Convex Optimization | 16 | | |
| 7.4 Conditon | 16 | | |
| 7.4.1 totally monotone (concave/convex) | 16 | | |
| 7.4.2 monge condition (concave/convex) | 16 | | |
| 8 Geometry | 16 | | |
| 8.1 Basic | 16 | | |
| 8.2 KD Tree | 16 | | |
| 8.3 Delaunay triangulation | 17 | | |
| 8.4 Sector Area | 18 | | |
| 8.5 Polygon Area | 18 | | |
| 8.6 Half Plane Intersection | 18 | | |
| 8.7 Rotating Sweep Line | 18 | | |
| 8.8 Triangle Center | 18 | | |
| 8.9 Polygon Center | 18 | | |
| 8.10 Maximum Triangle | 19 | | |
| 8.11 Point in Polygon | 19 | | |
| 8.12 Circle-Line Intersection | 19 | | |
| 8.13 Circle-Triangle Intersection | 19 | | |
| 8.14 Polygon Diameter | 20 | | |
| 8.15 Minimum Distance of 2 Polygons | 20 | | |
| 8.16 Convex Hull | 20 | | |
| 8.17 Rotating Caliper | 20 | | |
| 8.18 Minimum Enclosing Circle | 21 | | |
| 8.19 Closest Pair | 21 | | |

1 Basic

1.1 vimrc

```
set number relativenumber
syn on
colo desert
se ai nu ru mouse=a
se cin et ts=4 sw=4 sts=4
set backspace=indent,eol,start
set number relativenumber
inoremap {<ENTER> {<ENTER>}<UP><END><ENTER>
```

1.2 Compilation Argument

```
g++ -W -Wall -Wextra -O2 -std=c++14 -fsanitize=address
      -fsanitize=undefined -fsanitize=leak
```

1.3 Checker

```
for ((i = 0; i < 100; i++))
do
    ./gen > in
    ./ac < in > out1
    ./tle < in > out2
    diff out1 out2 || break
done
```

1.4 Fast Integer Input

```
#define getchar gtx
inline int gtx() {
    const int N = 4096;
    static char buffer[N];
    static char *p = buffer, *end = buffer;
    if (p == end) {
        if ((end = buffer + fread(buffer, 1, N, stdin)) ==
            buffer) return EOF;
        p = buffer;
    }
    return *p++;
}
template <typename T>
inline bool rit(T& x) {
    char c = 0; bool flag = false;
    while (c = getchar(), (c < '0' && c != '-') || c > '9'
        || c == '-') if (c == -1) return false;
    c == '-' ? (flag = true, x = 0) : (x = c - '0');
    while (c = getchar(), c >= '0' && c <= '9') x = x *
        10 + c - '0';
    if (flag) x = -x;
    return true;
}
template <typename T, typename ...Args>
inline bool rit(T& x, Args& ...args) { return rit(x) &&
    rit(args...); }
```

1.5 IncStack

```
const int size = 256 << 20;
register long rsp asm("rsp");
char *p = (char*)malloc(size) + size, *bak = (char*)rsp
;
__asm__("movq %0, %%rsp\n::"r"(p));
// main
__asm__("movq %0, %%rsp\n::"r"(bak));
```

1.6 Pragma optimization

```
#pragma GCC optimize("Ofast", "no-stack-protector", "no
-math-errno", "unroll-loops")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,sse4.2,
popcnt,abm,mmx,avx,tune=native,arch=core-avx2,tune=
core-avx2")
#pragma warning(disable:4996)
#pragma GCC ivdep
```

2 Flow

2.1 Dinic

```
struct dinic {
    static const int inf = 1e9;
    struct edge {
        int dest, cap, rev;
        edge(int d, int c, int r): dest(d), cap(c), rev(r)
        {}
    };
    vector<edge> g[maxn];
    int qu[maxn], ql, qr;
    int lev[maxn];
    void init() {
        for (int i = 0; i < maxn; ++i)
            g[i].clear();
    }
    void add_edge(int a, int b, int c) {
        g[a].emplace_back(b, c, g[b].size() - 0);
        g[b].emplace_back(a, 0, g[a].size() - 1);
    }
    bool bfs(int s, int t) {
        memset(lev, -1, sizeof(lev));
        lev[s] = 0;
        ql = qr = 0;
        qu[qr++] = s;
        while (ql < qr) {
            int x = qu[ql++];
            for (edge &e : g[x]) if (lev[e.dest] == -1 && e.
                cap > 0) {
                lev[e.dest] = lev[x] + 1;
                qu[qr++] = e.dest;
            }
        }
        return lev[t] != -1;
    }
    int dfs(int x, int t, int flow) {
        if (x == t) return flow;
        int res = 0;
        for (edge &e : g[x]) if (e.cap > 0 && lev[e.dest]
            == lev[x] + 1) {
            int f = dfs(e.dest, t, min(e.cap, flow - res));
            res += f;
            e.cap -= f;
            g[e.dest][e.rev].cap += f;
        }
        if (res == 0) lev[x] = -1;
        return res;
    }
    int operator()(int s, int t) {
        int flow = 0;
        for (; bfs(s, t); flow += dfs(s, t, inf));
        return flow;
    }
};
```

2.2 ISAP

```
struct isap {
    static const int inf = 1e9;
    struct edge {
        int dest, cap, rev;
        edge(int a, int b, int c): dest(a), cap(b), rev(c)
        {}
    };
};
```

```

vector<edge> g[maxn];
int it[maxn], gap[maxn], d[maxn];
void add_edge(int a, int b, int c) {
    g[a].emplace_back(b, c, g[b].size() - 0);
    g[b].emplace_back(a, 0, g[a].size() - 1);
}
int dfs(int x, int t, int tot, int flow) {
    if (x == t) return flow;
    for (int &i = it[x]; i < g[x].size(); ++i) {
        edge &e = g[x][i];
        if (e.cap > 0 && d[e.dest] == d[x] - 1) {
            int f = dfs(e.dest, t, tot, min(flow, e.cap));
            if (f) {
                e.cap -= f;
                g[e.dest][e.rev].cap += f;
                return f;
            }
        }
    }
    if (--gap[d[x]] == 0) d[x] = tot;
    else d[x]++, it[x] = 0, ++gap[d[x]];
    return 0;
}
int operator()(int s, int t, int tot) {
    memset(it, 0, sizeof(it));
    memset(gap, 0, sizeof(gap));
    memset(d, 0, sizeof(d));
    int r = 0;
    gap[0] = tot;
    for (; d[s] < tot; r += dfs(s, t, tot, inf));
    return r;
}
};

```

2.3 Minimum-cost flow

```

struct mincost {
    struct edge {
        int dest, cap, w, rev;
        edge(int a, int b, int c, int d): dest(a), cap(b),
        w(c), rev(d) {}
    };
    vector<edge> g[maxn];
    int d[maxn], p[maxn], ed[maxn];
    bool inq[maxn];
    void init() {
        for (int i = 0; i < maxn; ++i) g[i].clear();
    }
    void add_edge(int a, int b, int c, int d) {
        g[a].emplace_back(b, c, +d, g[b].size() - 0);
        g[b].emplace_back(a, 0, -d, g[a].size() - 1);
    }
    bool spfa(int s, int t, int &f, int &c) {
        for (int i = 0; i < maxn; ++i) {
            d[i] = inf;
            p[i] = ed[i] = -1;
            inq[i] = false;
        }
        d[s] = 0;
        queue<int> q;
        q.push(s);
        while (q.size()) {
            int x = q.front(); q.pop();
            inq[x] = false;
            for (int i = 0; i < g[x].size(); ++i) {
                edge &e = g[x][i];
                if (e.cap > 0 && d[e.dest] > d[x] + e.w) {
                    d[e.dest] = d[x] + e.w;
                    p[e.dest] = x;
                    ed[e.dest] = i;
                    if (!inq[e.dest]) q.push(e.dest), inq[e.dest] = true;
                }
            }
        }
        if (d[t] == inf) return false;
        int dlt = inf;
        for (int x = t; x != s; x = p[x]) dlt = min(dlt, g[p[x]][ed[x]].cap);
        for (int x = t; x != s; x = p[x]) {

```

```

            edge &e = g[p[x]][ed[x]];
            e.cap -= dlt;
            g[e.dest][e.rev].cap += dlt;
        }
        f += dlt; c += d[t] * dlt;
        return true;
    }
    pair<int, int> operator()(int s, int t) {
        int f = 0, c = 0;
        while (spfa(s, t, f, c));
        return make_pair(f, c);
    }
};

```

2.4 Gomory-Hu Tree

```

int g[maxn];
vector<edge> GomoryHu(int n){
    vector<edge> rt;
    for(int i=1;i<=n;++i)g[i]=1;
    for(int i=2;i<=n;++i){
        int t=g[i];
        flow.reset(); // clear flows on all edge
        rt.push_back({i,t,flow(i,t)});
        flow.walk(i); // bfs points that connected to i (
        use edges not fully flow)
        for(int j=i+1;j<=n;++j){
            if(g[j]==t && flow.connect(j))g[j]=i; // check if
            i can reach j
        }
    }
    return rt;
}

```

2.5 Stoer-Wagner minimum cut

```

const int maxn = 500 + 5;
int w[maxn][maxn], g[maxn];
bool v[maxn], del[maxn];

void add_edge(int x, int y, int c) {
    w[x][y] += c;
    w[y][x] += c;
}

pair<int, int> phase(int n) {
    memset(v, false, sizeof(v));
    memset(g, 0, sizeof(g));
    int s = -1, t = -1;
    while (true) {
        int c = -1;
        for (int i = 0; i < n; ++i) {
            if (del[i] || v[i]) continue;
            if (c == -1 || g[i] > g[c]) c = i;
        }
        if (c == -1) break;
        v[c] = true;
        s = t, t = c;
        for (int i = 0; i < n; ++i) {
            if (del[i] || v[i]) continue;
            g[i] += w[c][i];
        }
    }
    return make_pair(s, t);
}

int mincut(int n) {
    int cut = 1e9;
    memset(del, false, sizeof(del));
    for (int i = 0; i < n - 1; ++i) {
        int s, t; tie(s, t) = phase(n);
        del[t] = true;
        cut = min(cut, g[t]);
        for (int j = 0; j < n; ++j) {
            w[s][j] += w[t][j];
            w[j][s] += w[j][t];
        }
    }
}

```

```

    return cut;
}

```

2.6 Hungarian ($O(n^3)$)

```

struct Hungarian {
    vector<vector<int>> w;
    bitset<maxn> s, t;
    vector<int> lx, ly, mx, my, slack, prv;
    int n, matched;
    Hungarian() {}
    Hungarian(int _n): n(_n) {
        w = vector<vector<int>>(n, vector<int>(n));
        lx.resize(n); ly.resize(n); mx.assign(n, -1); my.
            assign(n, -1);
        slack.resize(n); prv.resize(n);
    }
    void add_edge(int a, int b, int c) {
        w[a][b] = c;
    }
    void add(int x) {
        s[x] = true;
        for (int i = 0; i < n; ++i) {
            if (lx[x] + ly[i] - w[x][i] < slack[i]) {
                slack[i] = lx[x] + ly[i] - w[x][i];
                prv[i] = x;
            }
        }
    }
    void augment(int now) {
        int x = prv[now], y = now;
        ++matched;
        while (true) {
            int tmp = mx[x]; mx[x] = y; my[y] = x; y = tmp;
            if (y == -1) return;
            x = prv[y];
        }
    }
    void relabel() {
        int delta = inf;
        for (int i = 0; i < n; ++i) if (!t[i]) delta = min(
            delta, slack[i]);
        for (int i = 0; i < n; ++i) if (s[i]) lx[i] -=
            delta;
        for (int i = 0; i < n; ++i) {
            if (t[i]) ly[i] += delta;
            else slack[i] -= delta;
        }
    }
    void go() {
        s.reset(); t.reset();
        fill(slack.begin(), slack.end(), inf);
        int root = 0;
        for (; root < n && mx[root] != -1; ++root);
        add(root);
        while (true) {
            relabel();
            int y = 0;
            for (; y < n; ++y) if (!t[y] && slack[y] == 0)
                break;
            if (my[y] == -1) return augment(y), void();
            add(my[y]); t[y] = true;
        }
    }
    int matching() {
        int ret = 0;
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) lx[i] = max(lx[i], w[
                i][j]);
        }
        for (int i = 0; i < n; ++i) go();
        for (int i = 0; i < n; ++i) ret += w[i][mx[i]];
        return ret;
    }
};

```

2.7 Hungarian ($O(n^4)$)

```

struct hungarian {
    static const int inf = 1e9;
    int lx[maxn], ly[maxn], w[maxn][maxn];
    int match[maxn];
    bool vx[maxn], vy[maxn];
    void init() {
        for (int i = 0; i < maxn; ++i) for (int j = 0; j <
            maxn; ++j) w[i][j] = -inf;
        for (int i = 0; i < maxn; ++i) w[i][i] = 0;
    }
    void add_edge(int a, int b, int c) {
        w[a][b] = max(w[a][b], c);
    }
    bool dfs(int now) {
        vx[now] = true;
        for (int i = 0; i < maxn; ++i) if (lx[now] + ly[i]
            == w[now][i] && !vy[i]) {
                vy[i] = true;
                if (!match[i] || dfs(match[i])) {
                    match[i] = now;
                    return true;
                }
            }
        return false;
    }
    void relabel() {
        int dlt = inf;
        for (int i = 0; i < maxn; ++i) if (vx[i]) {
            for (int j = 0; j < maxn; ++j) if (!vy[j]) dlt =
                min(dlt, lx[i] + ly[j] - w[i][j]);
        }
        for (int i = 0; i < maxn; ++i) if (vx[i]) lx[i] -=
            dlt;
        for (int i = 0; i < maxn; ++i) if (vy[i]) ly[i] +=
            dlt;
    }
    int operator()() {
        fill(lx, lx + maxn, -inf); fill(ly, ly + maxn, 0);
        for (int i = 0; i < maxn; ++i) {
            for (int j = 0; j < maxn; ++j) lx[i] = max(lx[i],
                w[i][j]);
        }
        memset(match, 0, sizeof(match));
        for (int i = 0; i < maxn; ++i) {
            while (true) {
                memset(vx, false, sizeof(vx));
                memset(vy, false, sizeof(vy));
                if (dfs(i)) break;
                relabel();
            }
        }
        int r = 0;
        for (int i = 0; i < maxn; ++i) if (w[match[i]][i] >
            0) r += w[match[i]][i];
        return r;
    }
};

```

2.8 Flow Model

- Maximum flow with lower/upper bound from s to t
 - Construct super source S and sink T
 - For each edge (x, y, l, u) , connect $x \rightarrow y$ with capacity $u - l$
 - For each vertex v , denote $in(v)$ as the difference between the sum of incoming lower bounds and the sum of outgoing lower bounds
 - If $in(v) > 0$, connect $S \rightarrow v$ with capacity $in(v)$, otherwise, connect $v \rightarrow T$ with capacity $-in(v)$
 - Denote f as the maximum flow of the current graph from S to T
 - Connect $t \rightarrow s$ with capacity ∞ , increment f by the maximum flow from S to T
 - If $f \neq \sum_{v \in V, in(v) > 0} in(v)$, there's no solution

8. Otherwise, the solution of each edge e is $l_e + f_e$, where f_e corresponds to the flow on the graph
- Construct minimum vertex cover from maximum matching on bipartite graph (X, Y)
 1. DFS from unmatched vertex in X using unused edges
 2. $x \in X$ is chosen iff x is unvisited
 3. $y \in Y$ is chosen iff y is visited
 - Minimum cost cyclic flow
 1. Construct super source S and sink T
 2. For each edge (x, y, c) , connect $x \rightarrow y$ with $(cost, cap) = (c, 1)$ if $c > 0$, otherwise connect $y \rightarrow x$ with $(cost, cap) = (-c, 1)$
 3. For each edge with $c < 0$, sum these cost as K , then increase $d(y)$ by 1, decrease $d(x)$ by 1
 4. For each vertex v with $d(v) > 0$, connect $S \rightarrow v$ with $(cost, cap) = (0, d(v))$
 5. For each vertex v with $d(v) < 0$, connect $v \rightarrow T$ with $(cost, cap) = (0, -d(v))$
 6. Flow from S to T , the answer is the cost of the flow $C + K$
 - Maximum density induced subgraph
 1. Binary search on answer, suppose we're checking answer T
 2. Construct a max flow model, let K be the sum of all weights
 3. Connect source $s \rightarrow v, v \in G$ with capacity K
 4. For each edge (u, v, w) in G , connect $u \rightarrow v$ and $v \rightarrow u$ with capacity w
 5. For $v \in G$, connect it with sink $v \rightarrow t$ with capacity $K + 2T - (\sum_{e \in E(v)} w(e)) - 2w(v)$
 6. T is a valid answer if the maximum flow $f < T|V|$

```
int find(int x) {
    if (x == p[x]) return x;
    return find(p[x]);
}
void merge(int x, int y) {
    x = find(x); y = find(y);
    if (x == y) return;
    if (sz[x] > sz[y]) swap(x, y);
    assign(&sz[y], sz[x] + sz[y]);
    assign(&p[x], y);
    assign(&cc, cc - 1);
}
} dsu;
```

3.2 <ext/pbds>

```
#include <bits/stdc++.h>
#include <bits/extc++.h>
#include <ext/rope>
using namespace __gnu_pbds;
using namespace __gnu_cxx;
#include <ext/pb_ds/assoc_container.hpp>
typedef tree<int, null_type, std::less<int>,
    rb_tree_tag, tree_order_statistics_node_update>
    tree_set;
typedef cc_hash_table<int, int> umap;
typedef priority_queue<int> heap;

int main() {
    // rb tree
    tree_set s;
    s.insert(71); s.insert(22);
    assert(*s.find_by_order(0) == 22); assert(*s.
        find_by_order(1) == 71);
    assert(s.order_of_key(22) == 0); assert(s.
        order_of_key(71) == 1);
    s.erase(22);
    assert(*s.find_by_order(0) == 71); assert(s.
        order_of_key(71) == 0);
    // mergable heap
    heap a, b; a.join(b);
    // persistent
    rope<char> r[2];
    r[1] = r[0];
    std::string st = "abc";
    r[1].insert(0, st.c_str());
    r[1].erase(1, 1);
    std::cout << r[1].substr(0, 2) << std::endl;
    return 0;
}
```

3 Data Structure

3.1 Disjoint Set

```
struct DisjointSet {
    int p[maxn], sz[maxn], n, cc;
    vector<pair<int*, int*>> his;
    vector<int> sh;
    void init(int _n) {
        n = _n; cc = n;
        for (int i = 0; i < n; ++i) sz[i] = 1, p[i] = i;
        sh.clear(); his.clear();
    }
    void assign(int *k, int v) {
        his.emplace_back(k, *k);
        *k = v;
    }
    void save() {
        sh.push_back((int)his.size());
    }
    void undo() {
        int last = sh.back(); sh.pop_back();
        while (his.size() != last) {
            int *k, v;
            tie(k, v) = his.back(); his.pop_back();
            *k = v;
        }
    }
}
```

3.3 Li Chao Tree

```
namespace lichao {
    struct line {
        long long a, b;
        line(): a(0), b(0) {}
        line(long long a, long long b): a(a), b(b) {}
        long long operator()(int x) const { return a * x + b; }
    };
    line st[maxc * 4];
    int sz, lc[maxc * 4], rc[maxc * 4];
    int gnode() {
        st[sz] = line(1e9, 1e9);
        lc[sz] = -1, rc[sz] = -1;
        return sz++;
    }
    void init() {
        sz = 0;
    }
    void add(int l, int r, line tl, int o) {
        bool lcp = st[o](l) > tl(l);
        bool mcp = st[o>((l + r) / 2) > tl((l + r) / 2);
        if (mcp) swap(st[o], tl);
        if (r - l == 1) return;
        if (lcp != mcp) {
            if (lc[o] == -1) lc[o] = gnode();
            if (rc[o] == -1) rc[o] = gnode();
        }
    }
}
```

```

    add(l, (l + r) / 2, tl, lc[o]);
  } else {
    if (rc[o] == -1) rc[o] = gnode();
    add((l + r) / 2, r, tl, rc[o]);
  }
}
long long query(int l, int r, int x, int o) {
  if (r - l == 1) return st[o](x);
  if (x < (l + r) / 2) {
    if (lc[o] == -1) return st[o](x);
    return min(st[o](x), query(l, (l + r) / 2, x, lc[o]));
  } else {
    if (rc[o] == -1) return st[o](x);
    return min(st[o](x), query((l + r) / 2, r, x, rc[o]));
  }
}
}

```

4 Graph

4.1 Link-Cut Tree

```

struct node {
  node *ch[2], *fa, *pfa;
  int sum, v, rev;
  node(int s): v(s), sum(s), rev(0), fa(nullptr), pfa(
    nullptr) {
    ch[0] = nullptr;
    ch[1] = nullptr;
  }
  int relation() {
    return this == fa->ch[0] ? 0 : 1;
  }
  void push() {
    if (!rev) return;
    swap(ch[0], ch[1]);
    if (ch[0]) ch[0]->rev ^= 1;
    if (ch[1]) ch[1]->rev ^= 1;
    rev = 0;
  }
  void pull() {
    sum = v;
    if (ch[0]) sum += ch[0]->sum;
    if (ch[1]) sum += ch[1]->sum;
  }
  void rotate() {
    if (fa->fa) fa->fa->push();
    fa->push(), push();
    swap(pfa, fa->pfa);
    int d = relation();
    node *t = fa;
    if (t->fa) t->fa->ch[t->relation()] = this;
    fa = t->fa;
    t->ch[d] = ch[d ^ 1];
    if (ch[d ^ 1]) ch[d ^ 1]->fa = t;
    ch[d ^ 1] = t;
    t->fa = this;
    t->pull(), pull();
  }
  void splay() {
    while (fa) {
      if (!fa->fa) {
        rotate();
        continue;
      }
      fa->fa->push(), fa->push();
      if (relation() == fa->relation()) fa->rotate(),
        rotate();
      else rotate(), rotate();
    }
  }
  void evert() {
    access();
    splay();
    rev ^= 1;
  }
}

```

```

void expose() {
  splay(), push();
  if (ch[1]) {
    ch[1]->fa = nullptr;
    ch[1]->pfa = this;
    ch[1] = nullptr;
    pull();
  }
}
bool splice() {
  splay();
  if (!pfa) return false;
  pfa->expose();
  pfa->ch[1] = this;
  fa = pfa;
  pfa = nullptr;
  fa->pull();
  return true;
}
void access() {
  expose();
  while (splice());
}
int query() {
  return sum;
}
};

namespace lct {
  node *sp[maxn];
  void make(int u, int v) {
    // create node with id u and value v
    sp[u] = new node(v, u);
  }
  void link(int u, int v) {
    // u become v's parent
    sp[v]->evert();
    sp[v]->pfa = sp[u];
  }
  void cut(int u, int v) {
    // u was v's parent
    sp[u]->evert();
    sp[v]->access(), sp[v]->splay(), sp[v]->push();
    sp[v]->ch[0]->fa = nullptr;
    sp[v]->ch[0] = nullptr;
    sp[v]->pull();
  }
  void modify(int u, int v) {
    sp[u]->splay();
    sp[u]->v = v;
    sp[u]->pull();
  }
  int query(int u, int v) {
    sp[u]->evert(), sp[v]->access(), sp[v]->splay();
    return sp[v]->query();
  }
}

```

4.2 Heavy-Light Decomposition

```

struct HeavyLightDecomp {
  vector<int> G[maxn];
  int tin[maxn], top[maxn], dep[maxn], maxson[maxn], sz
    [maxn], p[maxn], n, clk;
  void dfs(int now, int fa, int d) {
    dep[now] = d;
    maxson[now] = -1;
    sz[now] = 1;
    p[now] = fa;
    for (int u : G[now]) if (u != fa) {
      dfs(u, now, d + 1);
      sz[now] += sz[u];
      if (maxson[now] == -1 || sz[u] > sz[maxson[now]])
        maxson[now] = u;
    }
  }
  void link(int now, int t) {
    top[now] = t;
    tin[now] = ++clk;
    if (maxson[now] == -1) return;
  }
}

```

```

    link(maxson[now], t);
    for (int u : G[now]) if (u != p[now]) {
        if (u == maxson[now]) continue;
        link(u, u);
    }
}
HeavyLightDecomp(int n): n(n) {
    clk = 0;
    memset(tin, 0, sizeof(tin)); memset(top, 0, sizeof(
top)); memset(dep, 0, sizeof(dep));
    memset(maxson, 0, sizeof(maxson)); memset(sz, 0,
sizeof(sz)); memset(p, 0, sizeof(p));
}
void add_edge(int a, int b) {
    G[a].push_back(b);
    G[b].push_back(a);
}
void solve() {
    dfs(0, -1, 0);
    link(0, 0);
}
int lca(int a, int b) {
    int ta = top[a], tb = top[b];
    while (ta != tb) {
        if (dep[ta] < dep[tb]) {
            swap(ta, tb); swap(a, b);
        }
        a = p[ta]; ta = top[a];
    }
    if (a == b) return a;
    return dep[a] < dep[b] ? a : b;
}
vector<pair<int, int>> get_path(int a, int b) {
    int ta = top[a], tb = top[b];
    vector<pair<int, int>> ret;
    while (ta != tb) {
        if (dep[ta] < dep[tb]) {
            swap(ta, tb); swap(a, b);
        }
        ret.push_back(make_pair(tin[ta], tin[a]));
        a = p[ta]; ta = top[a];
    }
    ret.push_back(make_pair(min(tin[a], tin[b]), max(
tin[a], tin[b])));
    return ret;
}
};

```

4.3 Centroid Decomposition

```

vector<pair<int, int>> G[maxn];
int sz[maxn], mx[maxn];
bool v[maxn];
vector<int> vtx;

void get_center(int now) {
    v[now] = true; vtx.push_back(now);
    sz[now] = 1; mx[now] = 0;
    for (int u : G[now]) if (!v[u]) {
        get_center(u);
        mx[now] = max(mx[now], sz[u]);
        sz[now] += sz[u];
    }
}

void get_dis(int now, int d, int len) {
    dis[d][now] = cnt;
    v[now] = true;
    for (auto u : G[now]) if (!v[u.first]) {
        get_dis(u, d, len + u.second);
    }
}

void dfs(int now, int fa, int d) {
    get_center(now);
    int c = -1;
    for (int i : vtx) {
        if (max(mx[i], (int)vtx.size() - sz[i]) <= (int)vtx
.size() / 2) c = i;
        v[i] = false;
    }
}

```

```

}
get_dis(c, d, 0);
for (int i : vtx) v[i] = false;
v[c] = true; vtx.clear();
dep[c] = d; p[c] = fa;
for (auto u : G[c]) if (u.first != fa && !v[u.first])
{
    dfs(u.first, c, d + 1);
}
}
}

```

4.4 Minimum mean cycle

```

// d[i][j] == 0 if {i,j} !in E
long long d[1003][1003], dp[1003][1003];

pair<long long, long long> MMWC() {
    memset(dp, 0x3f, sizeof(dp));
    for (int i = 1; i <= n; ++i) dp[0][i] = 0;
    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= n; ++j) {
            for (int k = 1; k <= n; ++k) {
                dp[i][k] = min(dp[i-1][j] + d[j][k], dp[i][k]);
            }
        }
    }
    long long au = 1ll << 31, ad = 1;
    for (int i = 1; i <= n; ++i) {
        if (dp[n][i] == 0x3f3f3f3f3f3f3f3f) continue;
        long long u = 0, d = 1;
        for (int j = n-1; j >= 0; --j) {
            if ((dp[n][i] - dp[j][i]) * d > u * (n-j)) {
                u = dp[n][i] - dp[j][i];
                d = n-j;
            }
        }
        if (u * ad < au * d) au = u, ad = d;
    }
    long long g = __gcd(au, ad);
    return make_pair(au/g, ad/g);
}

```

4.5 Minimum Steiner Tree

```

namespace steiner {
    const int maxn = 64, maxk = 10;
    const int inf = 1e9;
    int w[maxn][maxn], dp[1 << maxk][maxn], off[maxn];
    void init(int n) {
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) w[i][j] = inf;
            w[i][i] = 0;
        }
    }
    void add_edge(int x, int y, int d) {
        w[x][y] = min(w[x][y], d);
        w[y][x] = min(w[y][x], d);
    }
    int solve(int n, vector<int> mark) {
        for (int k = 0; k < n; ++k) {
            for (int i = 0; i < n; ++i) {
                for (int j = 0; j < n; ++j) w[i][j] = min(w[i][
j], w[i][k] + w[k][j]);
            }
        }
        int k = (int)mark.size();
        assert(k < maxk);
        for (int s = 0; s < (1 << k); ++s) {
            for (int i = 0; i < n; ++i) dp[s][i] = inf;
        }
        for (int i = 0; i < n; ++i) dp[0][i] = 0;
        for (int s = 1; s < (1 << k); ++s) {
            if (__builtin_popcount(s) == 1) {
                int x = __builtin_ctz(s);
                for (int i = 0; i < n; ++i) dp[s][i] = w[mark[x
]][i];
                continue;
            }
        }
    }
}

```



```

    for (int i = 0; i < n; ++i) {
        for (int sub = s & (s - 1); sub; sub = s & (sub - 1)) {
            dp[s][i] = min(dp[s][i], dp[sub][i] + dp[s ^ sub][i]);
        }
    }
    for (int i = 0; i < n; ++i) {
        off[i] = inf;
        for (int j = 0; j < n; ++j) off[i] = min(off[i], dp[s][j] + w[j][i]);
    }
    for (int i = 0; i < n; ++i) dp[s][i] = min(dp[s][i], off[i]);
    int res = inf;
    for (int i = 0; i < n; ++i) res = min(res, dp[(1 << k) - 1][i]);
    return res;
}
}

```

4.6 Maximum Clique

```

struct MaxClique {
    int n, deg[maxn], ans;
    bitset<maxn> adj[maxn];
    vector<pair<int, int>> edge;
    void init(int _n) {
        n = _n;
        for (int i = 0; i < n; ++i) adj[i].reset();
        for (int i = 0; i < n; ++i) deg[i] = 0;
        edge.clear();
    }
    void add_edge(int a, int b) {
        edge.emplace_back(a, b);
        ++deg[a]; ++deg[b];
    }
    int solve() {
        vector<int> ord;
        for (int i = 0; i < n; ++i) ord.push_back(i);
        sort(ord.begin(), ord.end(), [&](const int &a, const int &b) { return deg[a] < deg[b]; });
        vector<int> id(n);
        for (int i = 0; i < n; ++i) id[ord[i]] = i;
        for (auto e : edge) {
            int u = id[e.first], v = id[e.second];
            adj[u][v] = adj[v][u] = true;
        }
        bitset<maxn> r, p;
        for (int i = 0; i < n; ++i) p[i] = true;
        ans = 0;
        dfs(r, p);
        return ans;
    }
    void dfs(bitset<maxn> r, bitset<maxn> p) {
        if (p.count() == 0) return ans = max(ans, (int)r.count());
        if ((r | p).count() <= ans) return;
        int now = p._Find_first();
        bitset<maxn> cur = p & ~adj[now];
        for (now = cur._Find_first(); now < n; now = cur._Find_next(now)) {
            r[now] = true;
            dfs(r, p & adj[now]);
            r[now] = false;
            p[now] = false;
        }
    }
};

```

4.7 Tarjan's articulation point

```

vector<pair<int, int>> g[maxn];
int low[maxn], tin[maxn], t;
int bcc[maxn], sz;
int a[maxn], b[maxn], deg[maxn];

```

```

bool cut[maxn], ins[maxn];

vector<int> ed[maxn];

stack<int> st;

void dfs(int x, int p) {
    tin[x] = low[x] = ++t;
    int ch = 0;
    for (auto u : g[x]) if (u.first != p) {
        if (!ins[u.second]) st.push(u.second), ins[u.second] = true;
        if (tin[u.first]) {
            low[x] = min(low[x], tin[u.first]);
            continue;
        }
        ++ch;
        dfs(u.first, x);
        low[x] = min(low[x], low[u.first]);
        if (low[u.first] >= tin[x]) {
            cut[x] = true;
            ++sz;
            while (true) {
                int e = st.top(); st.pop();
                bcc[e] = sz;
                if (e == u.second) break;
            }
        }
    }
    if (ch == 1 && p == -1) cut[x] = false;
}

```

4.8 Tarjan's bridge

```

vector<pair<int, int>> g[maxn];
int tin[maxn], low[maxn], t;
int a[maxn], b[maxn];
int bcc[maxn], sz;
bool br[maxn];

stack<int> st;

void dfs(int x, int p) {
    tin[x] = low[x] = ++t;
    st.push(x);
    for (auto u : g[x]) if (u.first != p) {
        if (tin[u.first]) {
            low[x] = min(low[x], tin[u.first]);
            continue;
        }
        dfs(u.first, x);
        low[x] = min(low[x], low[u.first]);
        if (low[u.first] == tin[u.first]) br[u.second] = true;
    }
    if (tin[x] == low[x]) {
        ++sz;
        while (st.size()) {
            int u = st.top(); st.pop();
            bcc[u] = sz;
            if (u == x) break;
        }
    }
}

```

5 String

5.1 KMP

```

int f[maxn];

int kmp(const string& a, const string& b) {
    f[0] = -1; f[1] = 0;
    for (int i = 1, j = 0; i < b.size() - 1; f[++i] = ++j) {
        if (b[i] == b[j]) f[i] = f[j];
    }
}

```



```

    while (j != -1 && b[i] != b[j]) j = f[j];
}
for (int i = 0, j = 0; i - j + b.size() <= a.size();
    ++i, ++j) {
    while (j != -1 && a[i] != b[j]) j = f[j];
    if (j == b.size() - 1) return i - j;
}
return -1;
}

```

5.2 Z algorithm

```

int z[maxn];
// z[i] = longest common prefix of suffix i and suffix
// 0

void z_function(const string& s) {
    memset(z, 0, sizeof(z));
    z[0] = (int)s.length();
    int l = 0, r = 0;
    for (int i = 1; i < s.length(); ++i) {
        z[i] = max(0, min(z[i - l], r - i + 1));
        while (i + z[i] < s.length() && s[z[i]] == s[i + z[i]]) {
            ++z[i];
            l = i; r = i + z[i];
        }
    }
}

```

5.3 Manacher's

```

int z[maxn];

int manacher(const string& s) {
    string t = ".";
    for (int i = 0; i < s.length(); ++i) t += s[i], t +=
        '.';
    int l = 0, r = 0;
    for (int i = 1; i < t.length(); ++i) {
        z[i] = (r > i ? min(z[2 * l - i], r - i) : 1);
        while (i - z[i] >= 0 && i + z[i] < t.length() && t[i - z[i]] == t[i + z[i]]) ++z[i];
        if (i + z[i] > r) r = i + z[i], l = i;
    }
    int ans = 0;
    for (int i = 1; i < t.length(); ++i) ans = max(ans, z[i] - 1);
    return ans;
}

```

5.4 Aho-Corasick Automaton

```

struct AC {
    static const int maxn = 1e5 + 5;
    int sz, ql, qr, root;
    int cnt[maxn], q[maxn], ed[maxn], el[maxn], ch[maxn][26], f[maxn];
    int gnode() {
        for (int i = 0; i < 26; ++i) ch[sz][i] = -1;
        f[sz] = -1;
        ed[sz] = 0;
        cnt[sz] = 0;
        return sz++;
    }
    void init() {
        sz = 0;
        root = gnode();
    }
    int add(const string &s) {
        int now = root;
        for (int i = 0; i < s.length(); ++i) {
            if (ch[now][s[i] - 'a'] == -1) ch[now][s[i] - 'a'] = gnode();
            now = ch[now][s[i] - 'a'];
        }
    }
}

```

```

    ed[now] = 1;
    return now;
}
void build_fail() {
    ql = qr = 0; q[qr++] = root;
    while (ql < qr) {
        int now = q[ql++];
        for (int i = 0; i < 26; ++i) if (ch[now][i] != -1) {
            int p = ch[now][i], fp = f[now];
            while (fp != -1 && ch[fp][i] == -1) fp = f[fp];
            int pd = fp != -1 ? ch[fp][i] : root;
            f[p] = pd;
            el[p] = ed[pd] ? pd : el[pd];
            q[qr++] = p;
        }
    }
}
void build(const string &s) {
    build_fail();
    int now = root;
    for (int i = 0; i < s.length(); ++i) {
        while (now != -1 && ch[now][s[i] - 'a'] == -1)
            now = f[now];
        now = now != -1 ? ch[now][s[i] - 'a'] : root;
        ++cnt[now];
    }
    for (int i = qr - 1; i >= 0; --i) cnt[f[q[i]]] += cnt[q[i]];
}
};

```

5.5 Suffix Automaton

```

struct SAM {
    static const int maxn = 5e5 + 5;
    int nxt[maxn][26], to[maxn], len[maxn];
    int root, last, sz;
    int gnode(int x) {
        for (int i = 0; i < 26; ++i) nxt[sz][i] = -1;
        to[sz] = -1;
        len[sz] = x;
        return sz++;
    }
    void init() {
        sz = 0;
        root = gnode(0);
        last = root;
    }
    void push(int c) {
        int cur = last;
        last = gnode(len[last] + 1);
        for (; ~cur && nxt[cur][c] == -1; cur = to[cur])
            nxt[cur][c] = last;
        if (cur == -1) return to[last] = root, void();
        int link = nxt[cur][c];
        if (len[link] == len[cur] + 1) return to[last] = link, void();
        int tlink = gnode(len[cur] + 1);
        for (; ~cur && nxt[cur][c] == link; cur = to[cur])
            nxt[cur][c] = tlink;
        for (int i = 0; i < 26; ++i) nxt[tlink][i] = nxt[link][i];
        to[tlink] = to[link];
        to[link] = tlink;
        to[last] = tlink;
    }
    void add(const string &s) {
        for (int i = 0; i < s.size(); ++i) push(s[i] - 'a');
    }
    bool find(const string &s) {
        int cur = root;
        for (int i = 0; i < s.size(); ++i) {
            cur = nxt[cur][s[i] - 'a'];
            if (cur == -1) return false;
        }
        return true;
    }
    int solve(const string &t) {
    }
}

```

```

int res = 0, cnt = 0;
int cur = root;
for (int i = 0; i < t.size(); ++i) {
    if (~nxt[cur][t[i] - 'a']) {
        ++cnt;
        cur = nxt[cur][t[i] - 'a'];
    } else {
        for (; ~cur && nxt[cur][t[i] - 'a'] == -1; cur
= to[cur]);
        if (~cur) cnt = len[cur] + 1, cur = nxt[cur][t[
i] - 'a'];
        else cnt = 0, cur = root;
    }
    res = max(res, cnt);
}
return res;
}
};

```

5.6 Suffix Array

```

int sa[maxn], tmp[2][maxn], c[maxn], hi[maxn], r[maxn];
// sa[i]: sa[i]-th suffix is the i-th lexicographically
// smallest suffix.
// hi[i]: longest common prefix of suffix sa[i] and
// suffix sa[i - 1].
void build(const string &s) {
    int *rnk = tmp[0], *rkn = tmp[1];
    for (int i = 0; i < 256; ++i) c[i] = 0;
    for (int i = 0; i < s.size(); ++i) c[rnk[i] = s[i]
]]++;
    for (int i = 1; i < 256; ++i) c[i] += c[i - 1];
    for (int i = s.size() - 1; i >= 0; --i) sa[--c[s[i]]]
= i;
    int sigma = 256;
    for (int n = 1; n < s.size(); n *= 2) {
        for (int i = 0; i < sigma; ++i) c[i] = 0;
        for (int i = 0; i < s.size(); ++i) c[rnk[i]]++;
        for (int i = 1; i < sigma; ++i) c[i] += c[i - 1];
        int *sa2 = rnk;
        int r = 0;
        for (int i = s.size() - n; i < s.size(); ++i) sa2[r
++] = i;
        for (int i = 0; i < s.size(); ++i) {
            if (sa[i] >= n) sa2[r++] = sa[i] - n;
        }
        for (int i = s.size() - 1; i >= 0; --i) sa[--c[rnk[
sa2[i]]]] = sa2[i];
        rnk[sa[0]] = r = 0;
        for (int i = 1; i < s.size(); ++i) {
            if (!(rnk[sa[i - 1]] == rnk[sa[i]] && sa[i - 1] +
n < s.size() && rnk[sa[i - 1] + n] == rnk[sa[i] +
n])) r++;
            rnk[sa[i]] = r;
        }
        swap(rnk, rnk);
        if (r == s.size() - 1) break;
        sigma = r + 1;
    }
    for (int i = 0; i < s.size(); ++i) r[sa[i]] = i;
    int ind = 0; hi[0] = 0;
    for (int i = 0; i < s.size(); ++i) {
        if (!r[i]) { ind = 0; continue; }
        while (i + ind < s.size() && s[i + ind] == s[sa[r[i]
] - 1] + ind) ++ind;
        hi[r[i]] = ind ? ind-- : 0;
    }
}
}

```

5.7 SAIS

```

namespace SAIS {
    enum type { L, S, LMS };
    const int maxn = 1e5 + 5;
    int bkt[maxn], cnt[maxn], lp[ptr][maxn], rp[ptr][maxn],
    tp[ptr][maxn];
    int rev[maxn];
    void pre(const vector<int> &s, int sigma) {

```

```

        fill(bkt, bkt + s.size(), -1);
        fill(cnt, cnt + sigma, 0);
        for (int i = 0; i < s.size(); ++i) ++cnt[s[i]];
        int last = 0;
        for (int i = 0; i < sigma; ++i) {
            lp[ptr][i] = last;
            last += cnt[i];
            rp[ptr][i] = tp[ptr][i] = last - 1;
        }
    }
    void induce(const vector<int> &s, const vector<type>
&v) {
        for (int i = 0; i < s.size(); ++i) if (bkt[i] > 0)
        {
            if (v[bkt[i] - 1] == L) bkt[lp[ptr][s[bkt[i] -
1]]++] = bkt[i] - 1;
        }
        for (int i = s.size() - 1; i >= 0; --i) if (bkt[i]
> 0) {
            if (v[bkt[i] - 1] != L) bkt[rp[ptr][s[bkt[i] -
1]]--] = bkt[i] - 1;
        }
    }
    bool equal(int l, int r, const vector<int> &s, const
vector<type> &v) {
        do { if (s[l] != s[r]) return false; ++l, ++r; }
        while (v[l] != LMS && v[r] != LMS);
        return s[l] == s[r];
    }
    vector<int> radix_sort(const vector<int> &lms, const
vector<int> &s, const vector<type> &v, int sigma) {
        pre(s, sigma);
        for (int i = 0; i < lms.size(); ++i) bkt[tp[ptr][s[lms
[i]]]]-- = lms[i];
        induce(s, v);
        vector<int> rt(lms.size());
        for (int i = 0; i < lms.size(); ++i) rev[lms[i]] =
i;
        int prv = -1, rnk = 0;
        for (int i = 0; i < s.size(); ++i) {
            int x = bkt[i];
            if (v[x] != LMS) continue;
            if (prv == -1) {
                rt[rev[x]] = rnk;
                prv = x;
                continue;
            }
            if (!equal(prv, x, s, v)) ++rnk;
            rt[rev[x]] = rnk;
            prv = x;
        }
        return rt;
    }
    vector<int> counting_sort(const vector<int> &s) {
        vector<int> o(s.size());
        for (int i = 0; i < s.size(); ++i) o[s[i]] = i;
        return o;
    }
    vector<int> reconstruct(const vector<int> &sa, const
vector<int> &s, const vector<type> &v) {
        vector<int> pos;
        for (int i = 0; i < s.size(); ++i) if (v[i] == LMS)
pos.push_back(i);
        vector<int> rev(sa.size());
        for (int i = 0; i < sa.size(); ++i) rev[i] = pos[sa
[i]];
        return rev;
    }
    vector<int> sais(const vector<int> &s, int sigma) {
        vector<type> v(s.size());
        v[s.size() - 1] = S;
        for (int i = s.size() - 2; i >= 0; --i) {
            if (s[i] < s[i + 1] || s[i] == s[i + 1] && v[i +
1] == S) v[i] = S;
            else v[i] = L;
        }
        for (int i = s.size() - 1; i >= 1; --i) {
            if (v[i] == S && v[i - 1] == L) v[i] = LMS;
        }
        vector<int> lms;
        for (int i = 0; i < s.size(); ++i) if (v[i] == LMS)
lms.push_back(i);

```

```

vector<int> r = radix_sort(lms, s, v, sigma);
vector<int> sa;
if (*max_element(r.begin(), r.end()) == r.size() - 1) sa = counting_sort(r);
else sa = sais(r, *max_element(r.begin(), r.end()) + 1);
sa = reconstruct(sa, s, v);
pre(s, sigma);
for (int i = sa.size() - 1; i >= 0; --i) bkt[tptr[s[sa[i]]]--] = sa[i];
induce(s, v);
return vector<int>(bkt, bkt + s.size());
}
vector<int> build(const string &s) {
vector<int> v(s.size() + 1);
for (int i = 0; i < s.size(); ++i) v[i] = s[i];
v[v.size() - 1] = 0;
vector<int> sa = sais(v, 256);
return vector<int>(sa.begin() + 1, sa.end());
}
}

```

5.8 DC3

```

namespace DC3{
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wsign-compare"
#define SG(v,i) ((i)>=int(v.size())?0:v[i])
inline bool smaller(int a, int b, vector<int> &r){
if(SG(r,a+0) != SG(r,b+0)) return SG(r,a+0)<SG(r,b+0);
if(SG(r,a+1) != SG(r,b+1)) return SG(r,a+1)<SG(r,b+1);
return SG(r,a+2)<SG(r,b+2);
}
int cc[100005];
inline vector<int> sort(vector<int> &r, int o, vector<int> &ix, int m){
vector<int> rt(ix.size());
for(int z=0;z<o;++z) r.push_back(0);
for(int i=0;i<=m;++i) cc[i] = 0;
for(int i=0;i<ix.size();++i) ++cc[r[ix[i]+o]];
for(int i=0;i<=m;++i) cc[i+1] += cc[i];
for(int i=ix.size()-1;i>=0;--i) rt[--cc[r[ix[i]+o]]] = ix[i];
for(int z=0;z<o;++z) r.pop_back();
return rt;
}
vector<int> dc3(vector<int> &v, int n, int m){
int c1 = (n+1)/3;
vector<int> i12;
for(int i=0;i<n;++i){
if(i%3==0)continue;
i12.push_back(i);
}
i12 = sort(v, 2, i12, m);
i12 = sort(v, 1, i12, m);
i12 = sort(v, 0, i12, m);

int nr = 1;
vector<int> r12(i12.size());
#define GRI(x) ((x)/3 + ((x)%3==2?c1:0))
r12[GRI(i12[0])] = 1;
for(int i=1;i<i12.size();++i){
if(smaller(i12[i-1], i12[i], v)) r12[GRI(i12[i])] = ++nr;
else r12[GRI(i12[i])] = nr;
}
#define GEI(x) ((x)<c1?(x)*3+1:(x-c1)*3+2)
if(nr != i12.size()){
i12 = dc3(r12, i12.size(), nr);

for(int i=0;i<i12.size();++i) r12[i12[i]] = i+1;
for(int &i: i12) i = GEI(i);
}
}

```

```

vector<int> i0;
if(n%3==1) i0.push_back(n-1);
for(int i=0;i<i12.size();++i) if(i12[i]%3 == 1) i0.push_back(i12[i]-1);
i0 = sort(v, 0, i0, m);

vector<int> ret(v.size());
int ptr12=0, ptr0=0, ptr=0;
while(ptr12<i12.size() && ptr0<i0.size()){
if(i12[ptr12]%3 == 1){
if([&](int i, int j) -> bool{
if(SG(v,i) != SG(v,j)) return SG(v,i)<SG(v,j)
};
return SG(r12,GRI(i+1))<SG(r12,GRI(j+1));
}(i12[ptr12], i0[ptr0]))ret[ptr++] = i12[ptr12++];
else ret[ptr++] = i0[ptr0++];
}
else{
if([&](int i, int j) -> bool{
if(SG(v,i+0) != SG(v,j+0)) return SG(v,i+0)<SG(v,j+0);
if(SG(v,i+1) != SG(v,j+1)) return SG(v,i+1)<SG(v,j+1);
return SG(r12,GRI(i+2))<SG(r12,GRI(j+2));
}(i12[ptr12], i0[ptr0]))ret[ptr++] = i12[ptr12++];
else ret[ptr++] = i0[ptr0++];
}
}
while(ptr12<i12.size()) ret[ptr++] = i12[ptr12++];
while(ptr0<i0.size()) ret[ptr++] = i0[ptr0++];

return ret;
}
vector<int> build(string str){
vector<int> val(str.size()+1, 0);
for(int i=0;i<str.size();++i) val[i] = str[i];
return dc3(val, val.size(), 255);
}
#pragma GCC diagnostic pop
}

```

5.9 Smallest Rotation

```

string rotate(const string &s) {
int n = s.length();
string t = s + s;
int i = 0, j = 1;
while (i < n && j < n) {
int k = 0;
while (k < n && t[i + k] == t[j + k]) ++k;
if (t[i + k] <= t[j + k]) j += k + 1;
else i += k + 1;
if (i == j) ++j;
}
int pos = (i < n ? i : j);
return t.substr(pos, n);
}

```

6 Math

6.1 Fast Fourier transform

```

struct cplx {
double re, im;
cplx(): re(0), im(0) {}
cplx(double r, double i): re(r), im(i) {}
cplx operator+(const cplx &rhs) const { return cplx(re + rhs.re, im + rhs.im); }
cplx operator-(const cplx &rhs) const { return cplx(re - rhs.re, im - rhs.im); }
cplx operator*(const cplx &rhs) const { return cplx(re * rhs.re - im * rhs.im, re * rhs.im + im * rhs.re); }
cplx conj() const { return cplx(re, -im); }
}

```

```

};

const int maxn = 262144;
const double pi = acos(-1);
cplx omega[maxn + 1];

void prefft() {
    for (int i = 0; i <= maxn; ++i)
        omega[i] = cplx(cos(2 * pi * i / maxn), sin(2 * pi * i / maxn));
}

void bitrev(vector<cplx> &v, int n) {
    int z = __builtin_ctz(n) - 1;
    for (int i = 0; i < n; ++i) {
        int x = 0;
        for (int j = 0; (1 << j) < n; ++j) x ^= (((i >> j & 1)) << (z - j));
        if (x > i) swap(v[x], v[i]);
    }
}

void fft(vector<cplx> &v, int n) {
    bitrev(v, n);
    for (int s = 2; s <= n; s <= 1) {
        int z = s >> 1;
        for (int i = 0; i < n; i += s) {
            for (int k = 0; k < z; ++k) {
                cplx x = v[i + z + k] * omega[maxn / s * k];
                v[i + z + k] = v[i + k] - x;
                v[i + k] = v[i + k] + x;
            }
        }
    }
}

void ifft(vector<cplx> &v, int n) {
    fft(v, n);
    reverse(v.begin() + 1, v.end());
    for (int i = 0; i < n; ++i) v[i] = v[i] * cplx(1. / n, 0);
}

vector<int> conv(const vector<int> &a, const vector<int> &b) {
    int sz = 1;
    while (sz < a.size() + b.size() - 1) sz <= 1;
    vector<cplx> v(sz);
    for (int i = 0; i < sz; ++i) {
        double re = i < a.size() ? a[i] : 0;
        double im = i < b.size() ? b[i] : 0;
        v[i] = cplx(re, im);
    }
    fft(v, sz);
    for (int i = 0; i <= sz / 2; ++i) {
        int j = (sz - i) & (sz - 1);
        cplx x = (v[i] + v[j].conj()) * (v[i] - v[j].conj()) * cplx(0, -0.25);
        if (j != i) v[j] = (v[j] + v[i].conj()) * (v[j] - v[i].conj()) * cplx(0, -0.25);
        v[i] = x;
    }
    ifft(v, sz);
    vector<int> c(sz);
    for (int i = 0; i < sz; ++i) c[i] = round(v[i].re);
    while (c.size() && c.back() == 0) c.pop_back();
    return c;
}

    (a *= a) %= mod;
}
return r;
}

void prentt() {
    long long x = fpow(root, (mod - 1) / maxn);
    omega[0] = 1;
    for (int i = 1; i <= maxn; ++i)
        omega[i] = omega[i - 1] * x % mod;
}

void bitrev(vector<long long> &v, int n) {
    int z = __builtin_ctz(n) - 1;
    for (int i = 0; i < n; ++i) {
        int x = 0;
        for (int j = 0; j <= z; ++j) x ^= ((i >> j & 1) << (z - j));
        if (x > i) swap(v[x], v[i]);
    }
}

void ntt(vector<long long> &v, int n) {
    bitrev(v, n);
    for (int s = 2; s <= n; s <= 1) {
        int z = s >> 1;
        for (int i = 0; i < n; i += s) {
            for (int k = 0; k < z; ++k) {
                long long x = v[i + k + z] * omega[maxn / s * k] % mod;
                v[i + k + z] = (v[i + k] + mod - x) % mod;
                (v[i + k] += x) %= mod;
            }
        }
    }
}

void intt(vector<long long> &v, int n) {
    ntt(v, n);
    reverse(v.begin() + 1, v.end());
    long long inv = fpow(n, mod - 2);
    for (int i = 0; i < n; ++i) (v[i] *= inv) %= mod;
}

vector<long long> conv(vector<long long> a, vector<long long> b) {
    int sz = 1;
    while (sz < a.size() + b.size() - 1) sz <= 1;
    vector<long long> c(sz);
    while (a.size() < sz) a.push_back(0);
    while (b.size() < sz) b.push_back(0);
    ntt(a, sz), ntt(b, sz);
    for (int i = 0; i < sz; ++i) c[i] = a[i] * b[i] % mod;
    intt(c, sz);
    while (c.size() && c.back() == 0) c.pop_back();
    return c;
}

```

6.2 Number theoretic transform

```

const int maxn = 262144;
const long long mod = 2013265921, root = 31;
long long omega[maxn + 1];

long long fpow(long long a, long long n) {
    (n += mod - 1) %= mod - 1;
    long long r = 1;
    for (; n; n >>= 1) {
        if (n & 1) (r *= a) %= mod;
    }
}

```

6.2.1 NTT Prime List

| Prime | Root |
|------------|------|
| 97 | 5 |
| 193 | 5 |
| 257 | 3 |
| 7681 | 17 |
| 12289 | 11 |
| 40961 | 3 |
| 65537 | 3 |
| 786433 | 10 |
| 5767169 | 3 |
| 7340033 | 3 |
| 23068673 | 3 |
| 104857601 | 3 |
| 167772161 | 3 |
| 469762049 | 3 |
| 605028353 | 3 |
| 1107296257 | 10 |
| 2013265921 | 31 |
| 2810183681 | 11 |
| 2885681153 | 3 |

6.3 Fast Walsh-Hadamard transform

```

void xorfwft(int v[], int l, int r) {
    if (r - l == 1) return;
    int m = l + r >> 1;
    xorfwft(v, l, m), xorfwft(v, m, r);
    for (int i = l, j = m; i < m; ++i, ++j) {
        int x = v[i] + v[j];
        v[j] = v[i] - v[j], v[i] = x;
    }
}

void xorifwt(int v[], int l, int r) {
    if (r - l == 1) return;
    int m = l + r >> 1;
    for (int i = l, j = m; i < m; ++i, ++j) {
        int x = (v[i] + v[j]) / 2;
        v[j] = (v[i] - v[j]) / 2, v[i] = x;
    }
    xorifwt(v, l, m), xorifwt(v, m, r);
}

void andfwft(int v[], int l, int r) {
    if (r - l == 1) return;
    int m = l + r >> 1;
    andfwft(v, l, m), andfwft(v, m, r);
    for (int i = l, j = m; i < m; ++i, ++j) v[i] += v[j];
}

void andifwt(int v[], int l, int r) {
    if (r - l == 1) return;
    int m = l + r >> 1;
    andifwt(v, l, m), andifwt(v, m, r);
    for (int i = l, j = m; i < m; ++i, ++j) v[i] -= v[j];
}

void orfwft(int v[], int l, int r) {
    if (r - l == 1) return;
    int m = l + r >> 1;
    orfwft(v, l, m), orfwft(v, m, r);
    for (int i = l, j = m; i < m; ++i, ++j) v[j] += v[i];
}

void orifwt(int v[], int l, int r) {
    if (r - l == 1) return;
    int m = l + r >> 1;
    orifwt(v, l, m), orifwt(v, m, r);
    for (int i = l, j = m; i < m; ++i, ++j) v[j] -= v[i];
}

```

6.4 Lagrange Interpolation

```

namespace lagrange {
    long long pf[maxn], nf[maxn];
    void init() {
        pf[0] = nf[0] = 1;
        for (int i = 1; i < maxn; ++i) {
            pf[i] = pf[i - 1] * i % mod;
            nf[i] = nf[i - 1] * (mod - i) % mod;
        }
    }
    // given y: value of f(a), a = [0, n], find f(x)
    long long solve(int n, vector<long long> y, long long x) {
        if (x <= n) return y[x];
        long long all = 1;
        for (int i = 0; i <= n; ++i) (all *= (x - i + mod)) %= mod;
        long long ans = 0;
        for (int i = 0; i <= n; ++i) {
            long long z = all * fpow(x - i, -1) % mod;
            long long l = pf[i], r = nf[n - i];
            (ans += y[i] * z % mod * fpow(l * r, -1)) %= mod;
        }
        return ans;
    }
}

```

6.5 Miller Rabin

```

// n < 4759123141   chk = [2, 7, 61]
// n < 1122004669633   chk = [2, 13, 23, 1662803]
// n < 2^64   chk = [2, 325, 9375, 28178, 450775, 9780504, 1795265022]
//
vector<long long> chk = { 2, 325, 9375, 28178, 450775, 9780504, 1795265022 };

long long fmul(long long a, long long n, long long mod) {
    long long ret = 0;
    for (; n; n >>= 1) {
        if (n & 1) (ret += a) %= mod;
        (a += a) %= mod;
    }
    return ret;
}

long long fpow(long long a, long long n, long long mod) {
    long long ret = 1LL;
    for (; n; n >>= 1) {
        if (n & 1) ret = fmul(ret, a, mod);
        a = fmul(a, a, mod);
    }
    return ret;
}

bool check(long long a, long long u, long long n, int t) {
    a = fpow(a, u, n);
    if (a == 0) return true;
    if (a == 1 || a == n - 1) return true;
    for (int i = 0; i < t; ++i) {
        a = fmul(a, a, n);
        if (a == 1) return false;
        if (a == n - 1) return true;
    }
    return false;
}

bool is_prime(long long n) {
    if (n < 2) return false;
    if (n % 2 == 0) return n == 2;
    long long u = n - 1; int t = 0;
    for (; u & 1; u >>= 1, ++t);
    for (long long i : chk) {
        if (!check(i, u, n, t)) return false;
    }
    return true;
}

```

6.6 Pollard's rho

```
long long f(long long x, long long n, int p) { return (
    fmul(x, x, n) + p) % n; }

map<long long, int> cnt;

void pollard_rho(long long n) {
    if (n == 1) return;
    if (prime(n)) return ++cnt[n], void();
    if (n % 2 == 0) return pollard_rho(n / 2), ++cnt[2],
        void();
    long long x = 2, y = 2, d = 1, p = 1;
    while (true) {
        if (d != n && d != 1) {
            pollard_rho(n / d);
            pollard_rho(d);
            return;
        }
        if (d == n) ++p;
        x = f(x, n, p); y = f(f(y, n, p), n, p);
        d = __gcd(abs(x - y), n);
    }
}
```

6.7 Prime counting

```
int prc[maxn];
long long phic[msz][nsz];

void sieve() {
    bitset<maxn> v;
    pr.push_back(0);
    for (int i = 2; i < maxn; ++i) {
        if (!v[i]) pr.push_back(i);
        for (int j = 1; i * pr[j] < maxn; ++j) {
            v[i * pr[j]] = true;
            if (i % pr[j] == 0) break;
        }
    }
    for (int i = 1; i < pr.size(); ++i) prc[pr[i]] = 1;
    for (int i = 1; i < maxn; ++i) prc[i] += prc[i - 1];
}

long long p2(long long, long long);

long long phi(long long m, long long n) {
    if (m < msz && n < nsz && phic[m][n] != -1) return
        phic[m][n];
    if (n == 0) return m;
    if (pr[n] >= m) return 1;
    long long ret = phi(m, n - 1) - phi(m / pr[n], n - 1);
    if (m < msz && n < nsz) phic[m][n] = ret;
    return ret;
}

long long pi(long long m) {
    if (m < maxn) return prc[m];
    long long n = pi(cbrt(m));
    return phi(m, n) + n - 1 - p2(m, n);
}

long long p2(long long m, long long n) {
    long long ret = 0;
    long long lim = sqrt(m);
    for (int i = n + 1; pr[i] <= lim; ++i) ret += pi(m /
        pr[i]) - pi(pr[i]) + 1;
    return ret;
}
```

6.8 Gaussian Elimination

```
void gauss(vector<vector<double>> &d) {
    int n = d.size(), m = d[0].size();
    for (int i = 0; i < m; ++i) {
        int p = -1;
```

```
        for (int j = i; j < n; ++j) {
            if (fabs(d[j][i]) < eps) continue;
            if (p == -1 || fabs(d[j][i]) > fabs(d[p][i])) p =
                j;
        }
        if (p == -1) continue;
        for (int j = 0; j < m; ++j) swap(d[p][j], d[i][j]);
        for (int j = 0; j < n; ++j) {
            if (i == j) continue;
            double z = d[j][i] / d[i][i];
            for (int k = 0; k < m; ++k) d[j][k] -= z * d[i][k];
        }
    }
}
```

6.9 Linear Equations (full pivoting)

```
void linear_equation(vector<vector<double>> &d, vector<
    double> &aug, vector<double> &sol) {
    int n = d.size(), m = d[0].size();
    vector<int> r(n), c(m);
    iota(r.begin(), r.end(), 0);
    iota(c.begin(), c.end(), 0);
    for (int i = 0; i < m; ++i) {
        int p = -1, z = -1;
        for (int j = i; j < n; ++j) {
            for (int k = i; k < m; ++k) {
                if (fabs(d[r[j]][c[k]]) < eps) continue;
                if (p == -1 || fabs(d[r[j]][c[k]]) > fabs(d[r[p]
                    ][c[z]])) p = j, z = k;
            }
        }
        if (p == -1) continue;
        swap(r[p], r[i]), swap(c[z], c[i]);
        for (int j = 0; j < n; ++j) {
            if (i == j) continue;
            double z = d[r[j]][c[i]] / d[r[i]][c[i]];
            for (int k = 0; k < m; ++k) d[r[j]][c[k]] -= z *
                d[r[i]][c[k]];
            aug[r[j]] -= z * aug[r[i]];
        }
    }
    vector<vector<double>> fd(n, vector<double>(m));
    vector<double> faug(n), x(n);
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) fd[i][j] = d[r[i]][c[j]
            ];
        faug[i] = aug[r[i]];
    }
    d = fd, aug = faug;
    for (int i = n - 1; i >= 0; --i) {
        double p = 0.0;
        for (int j = i + 1; j < n; ++j) p += d[i][j] * x[j];
        x[i] = (aug[i] - p) / d[i][i];
    }
    for (int i = 0; i < n; ++i) sol[c[i]] = x[i];
}
```

6.10 μ function

```
int mu[maxn], pi[maxn];
vector<int> prime;

void sieve() {
    mu[1] = pi[1] = 1;
    for (int i = 2; i < maxn; ++i) {
        if (!pi[i]) {
            pi[i] = i;
            prime.push_back(i);
            mu[i] = -1;
        }
        for (int j = 0; i * prime[j] < maxn; ++j) {
            pi[i * prime[j]] = prime[j];
            mu[i * prime[j]] = -mu[i];
            if (i % prime[j] == 0) {
                mu[i * prime[j]] = 0;
            }
        }
    }
}
```



```

        break;
    }
}
}
}

```

6.11 $\lfloor \frac{n}{i} \rfloor$ Enumeration

```

vector<int> solve(int n) {
    vector<int> vec;
    for (int t = 1; t < n; t = (n / (n / (t + 1)))) vec.
        push_back(t);
    vec.push_back(n);
    vec.resize(unique(vec.begin(), vec.end()) - vec.begin()
    );
    return vec;
}

```

6.12 Extended GCD

```

template <typename T> tuple<T, T, T> extgcd(T a, T b) {
    if (!b) return make_tuple(a, 1, 0);
    T d, x, y;
    tie(d, x, y) = extgcd(b, a % b);
    return make_tuple(d, y, x - (a / b) * y);
}

```

6.13 Chinese remainder theorem

Given $x \equiv a_i \pmod{n_i} \forall 1 \leq i \leq k$, where n_i are pairwise co-prime, find x .

Let $N = \prod_{i=1}^k n_i$ and $N_i = N/n_i$, there exist integer M_i and m_i such that $M_i N_i + m_i n_i = 1$.

A solution to the system of congruence is $x = \sum_{i=1}^k a_i M_i N_i$.

6.14 Lucas's theorem

For non-negative integers m and n and prime p ,

$$\binom{m}{n} = \prod_{i=0}^k \binom{m_i}{n_i} \pmod{p}$$

where

$$m = m_k p^k + m_{k-1} p^{k-1} + \dots + m_1 p + m_0,$$

$$n = n_k p^k + n_{k-1} p^{k-1} + \dots + n_1 p + n_0.$$

6.15 Kirchhoff's theorem

Denote L be a $n \times n$ matrix as the Laplacian matrix of graph G , where $L_{ii} = d(i)$, $L_{ij} = -c$ where c is the number of edge (i, j) in G .

- The number of undirected spanning in G is $|\det(L^*)|$, where L^* is the $(n-1) \times (n-1)$ matrix by removing row x and column x for some arbitrary x in L
- The number of directed spanning tree rooted at r in G is $|\det(L_r)|$, where L_r is the $(n-1) \times (n-1)$ matrix by removing row r and column r in L

6.16 Tutte matrix

Let D be a $n \times n$ matrix, where $d_{ij} = x_{ij}$ (x_{ij} is chosen uniform randomly) if $i < j$ and $(i, j) \in E$, otherwise $d_{ij} = -d_{ji}$. $\frac{\text{rank}(D)}{2}$ is the maximum matching on G .

6.17 Primes

97, 101, 131, 487, 593, 877, 1087, 1187, 1487, 1787, 3187, 12721, 13331, 14341, 75577, 123457, 222557, 556679, 999983, 1097774749, 1076767633, 100102021, 999997771, 1001010013, 1000512343, 987654361, 999991231, 999888733, 98789101, 987777733, 999991921, 1000000007, 1000000087, 1000000123, 1010101333, 1010102101, 1000000000039, 1000000000000037, 2305843009213693951, 4611686018427387847, 9223372036854775783, 18446744073709551557

7 Dynamic Programming

7.1 Convex Hull (monotone)

```

struct line {
    double a, b;
    inline double operator()(const double &x) const {
        return a * x + b; }
    inline bool checkfront(const line &l, const double &x)
        const { return (*this)(x) < l(x); }
    inline double intersect(const line &l) const { return
        (l.b - b) / (a - l.a); }
    inline bool checkback(const line &l, const line &
        pivot) const { return pivot.intersect((*this)) <=
        pivot.intersect(l); }
};

void solve() {
    for (int i = 1; i < maxn; ++i) dp[0][i] = inf;
    for (int i = 1; i <= k; ++i) {
        deque<line> dq; dq.push_back((line){ 0.0, dp[i -
        1][0] });
        for (int j = 1; j <= n; ++j) {
            while (dq.size() >= 2 && dq[1].checkfront(dq[0],
            invt[j])) dq.pop_front();
            dp[i][j] = st[j] + dq.front()(invt[j]);
            line nl = (line){ -s[j], dp[i - 1][j] - st[j] + s
            [j] * invt[j] };
            while (dq.size() >= 2 && nl.checkback(dq[dq.size
            () - 1], dq[dq.size() - 2])) dq.pop_back();
            dq.push_back(nl);
        }
    }
}

```

7.2 Convex Hull (non-monotone)

```

struct line {
    int m, y;
    int l, r;
    line(int m = 0, int y = 0, int l = -5, int r =
    1000000009): m(m), y(y), l(l), r(r) {}
    int get(int x) const { return m * x + y; }
    int useful(line le) const {
        return (int)(get(l) >= le.get(l)) + (int)(get(r) >=
        le.get(r));
    }
};

int magic;
bool operator < (const line &a, const line &b) {
    if (magic) return a.m < b.m;
    return a.l < b.l;
}

set<line> st;

void addline(line l) {
    magic = 1;
    auto it = st.lower_bound(l);
    if (it != st.end() && it->useful(l) == 2) return;
    while (it != st.end() && it->useful(l) == 0) it = st.
    erase(it);
}

```



```

if (it != st.end() && it->useful(l) == 1) {
    int L = it->l, R = it->r, M;
    while (R > L) {
        M = (L + R + 1) >> 1;
        if (it->get(M) >= l.get(M)) R = M - 1;
        else L = M;
    }
    line cp = *it;
    st.erase(it);
    cp.l = L + 1;
    if (cp.l <= cp.r) st.insert(cp);
    l.r = L;
}
else if (it != st.end()) l.r = it->l - 1;
it = st.lower_bound(l);
while (it != st.begin() && prev(it)->useful(l) == 0)
    it = st.erase(prev(it));
if (it != st.begin() && prev(it)->useful(l) == 1) {
    --it;
    int L = it->l, R = it->r, M;
    while (R > L) {
        M = (L + R) >> 1;
        if (it->get(M) >= l.get(M)) L = M + 1;
        else R = M;
    }
    line cp = *it;
    st.erase(it);
    cp.r = L - 1;
    if (cp.l <= cp.r) st.insert(cp);
    l.l = L;
}
else if (it != st.begin()) l.l = prev(it)->r + 1;
if (l.l <= l.r) st.insert(l);
}

int getval(int d) {
    magic = 0;
    return (--st.upper_bound(line(0, 0, d, 0)))->get(d);
}

```

7.3 1D/1D Convex Optimization

```

struct segment {
    int i, l, r;
    segment() {}
    segment(int a, int b, int c): i(a), l(b), r(c) {}
};

inline long long f(int l, int r) {
    return dp[l] + w(l + 1, r);
}

void solve() {
    dp[0] = 0ll;
    deque<segment> deq; deq.push_back(segment(0, 1, n));
    for (int i = 1; i <= n; ++i) {
        dp[i] = f(deq.front().i, i);
        while (deq.size() && deq.front().r < i + 1) deq.pop_front();
        deq.front().l = i + 1;
        segment seg = segment(i, i + 1, n);
        while (deq.size() && f(i, deq.back().l) < f(deq.back().i, deq.back().l)) deq.pop_back();
        if (deq.size()) {
            int d = 1048576, c = deq.back().l;
            while (d >= 1) if (c + d <= deq.back().r) {
                if (f(i, c + d) > f(deq.back().i, c + d)) c += d;
            }
            deq.back().r = c; seg.l = c + 1;
        }
        if (seg.l <= n) deq.push_back(seg);
    }
}

```

7.4 Conditon

7.4.1 totally monotone (concave/convex)

$$\forall i < i', j < j', B[i][j] \leq B[i'][j] \implies B[i][j'] \leq B[i'][j']$$

$$\forall i < i', j < j', B[i][j] \geq B[i'][j] \implies B[i][j'] \geq B[i'][j']$$

7.4.2 monge condition (concave/convex)

$$\forall i < i', j < j', B[i][j] + B[i'][j'] \geq B[i][j'] + B[i'][j]$$

$$\forall i < i', j < j', B[i][j] + B[i'][j'] \leq B[i][j'] + B[i'][j]$$

8 Geometry

8.1 Basic

```

bool same(const double a, const double b) { return abs(a - b) < 1e-9; }

struct Point {
    double x, y;
    Point(): x(0), y(0) {}
    Point(double x, double y): x(x), y(y) {}
};

Point operator+(const Point a, const Point b) { return Point(a.x + b.x, a.y + b.y); }
Point operator-(const Point a, const Point b) { return Point(a.x - b.x, a.y - b.y); }
Point operator*(const Point a, const double b) { return Point(a.x * b, a.y * b); }
Point operator/(const Point a, const double b) { return Point(a.x / b, a.y / b); }
double operator^(const Point a, const Point b) { return a.x * b.y - a.y * b.x; }
double abs(const Point a) { return sqrt(a.x * a.x + a.y * a.y); }

```

```

struct Line {
    // ax + by + c = 0
    double a, b, c;
    double angle;
    Point pa, pb;
    Line(): a(0), b(0), c(0), angle(0), pa(), pb() {}
    Line(Point pa, Point pb): a(pa.y - pb.y), b(pb.x - pa.x), c(pa.x * pb.y - pa.y * pb.x), angle(atan2(-a, b)), pa(pa), pb(pb) {}
};

Point intersect(Line la, Line lb) {
    if (same(la.a * lb.b, la.b * lb.a)) return Point(7122, 7122);
    double bot = -la.a * lb.b + la.b * lb.a;
    return Point(-la.b * lb.c + la.c * lb.b, la.a * lb.c - la.c * lb.a) / bot;
}

```

8.2 KD Tree

```

namespace kdt {
    int root, lc[maxn], rc[maxn], xl[maxn], xr[maxn], yl[maxn], yr[maxn];
    point p[maxn];
    int build(int l, int r, int dep = 0) {
        if (l == r) return -1;
        function<bool(const point &, const point &)> f = [dep](const point &a, const point &b) {
            if (dep & 1) return a.x < b.x;
            else return a.y < b.y;
        };
        int m = (l + r) >> 1;
        nth_element(p + l, p + m, p + r, f);
        xl[m] = xr[m] = p[m].x;
        yl[m] = yr[m] = p[m].y;
        lc[m] = build(l, m, dep + 1);
        if (~lc[m]) {
            xl[m] = min(xl[m], xl[lc[m]]);
            xr[m] = max(xr[m], xr[lc[m]]);
        }
    }
}

```

```

    yl[m] = min(yl[m], yl[lc[m]]);
    yr[m] = max(yr[m], yr[lc[m]]);
}
rc[m] = build(m + 1, r, dep + 1);
if (~rc[m]) {
    xl[m] = min(xl[m], xl[rc[m]]);
    xr[m] = max(xr[m], xr[rc[m]]);
    yl[m] = min(yl[m], yl[rc[m]]);
    yr[m] = max(yr[m], yr[rc[m]]);
}
return m;
}
bool bound(const point &q, int o, long long d) {
    double ds = sqrt(d + 1.0);
    if (q.x < xl[o] - ds || q.x > xr[o] + ds ||
        q.y < yl[o] - ds || q.y > yr[o] + ds) return
        false;
    return true;
}
long long dist(const point &a, const point &b) {
    return (a.x - b.x) * 1ll * (a.x - b.x) +
        (a.y - b.y) * 1ll * (a.y - b.y);
}
void dfs(const point &q, long long &d, int o, int dep
    = 0) {
    if (!bound(q, o, d)) return;
    long long cd = dist(p[o], q);
    if (cd != 0) d = min(d, cd);
    if ((dep & 1) && q.x < p[o].x || !(dep & 1) && q.y
        < p[o].y) {
        if (~lc[o]) dfs(q, d, lc[o], dep + 1);
        if (~rc[o]) dfs(q, d, rc[o], dep + 1);
    } else {
        if (~rc[o]) dfs(q, d, rc[o], dep + 1);
        if (~lc[o]) dfs(q, d, lc[o], dep + 1);
    }
}
void init(const vector<point> &v) {
    for (int i = 0; i < v.size(); ++i) p[i] = v[i];
    root = build(0, v.size());
}
long long nearest(const point &q) {
    long long res = 1e18;
    dfs(q, res, root);
    return res;
}
}

```

8.3 Delaunay triangulation

```

namespace triangulation {
    static const int maxn = 1e5 + 5;
    vector<point> p;
    set<int> g[maxn];
    int o[maxn];
    set<int> s;
    void add_edge(int x, int y) {
        s.insert(x), s.insert(y);
        g[x].insert(y);
        g[y].insert(x);
    }
    bool inside(point a, point b, point c, point p) {
        if (((b - a) ^ (c - a)) < 0) swap(b, c);
        function<long long(int)> sqr = [](int x) { return x
            * 1ll * x; };
        long long k11 = a.x - p.x, k12 = a.y - p.y, k13 =
            sqr(a.x) - sqr(p.x) + sqr(a.y) - sqr(p.y);
        long long k21 = b.x - p.x, k22 = b.y - p.y, k23 =
            sqr(b.x) - sqr(p.x) + sqr(b.y) - sqr(p.y);
        long long k31 = c.x - p.x, k32 = c.y - p.y, k33 =
            sqr(c.x) - sqr(p.x) + sqr(c.y) - sqr(p.y);
        long long det = k11 * (k22 * k33 - k23 * k32) - k12
            * (k21 * k33 - k23 * k31) + k13 * (k21 * k32 - k22
            * k31);
        return det > 0;
    }
    bool intersect(const point &a, const point &b, const
        point &c, const point &d) {
        return ((b - a) ^ (c - a)) * ((b - a) ^ (d - a)) <
            0 &&

```

```

        ((d - c) ^ (a - c)) * ((d - c) ^ (b - c)) < 0;
    }
    void dfs(int l, int r) {
        if (r - l <= 3) {
            for (int i = l; i < r; ++i) {
                for (int j = i + 1; j < r; ++j) add_edge(i, j);
            }
            return;
        }
        int m = (l + r) >> 1;
        dfs(l, m), dfs(m, r);
        int pl = l, pr = r - 1;
        while (true) {
            int z = -1;
            for (int u : g[pl]) {
                long long c = ((p[pl] - p[pr]) ^ (p[u] - p[pr]))
            );
            if (c > 0 || c == 0 && abs(p[u] - p[pr]) < abs(
                p[pl] - p[pr])) {
                z = u;
                break;
            }
        }
        if (z != -1) {
            pl = z;
            continue;
        }
        for (int u : g[pr]) {
            long long c = ((p[pr] - p[pl]) ^ (p[u] - p[pl]))
        );
        if (c < 0 || c == 0 && abs(p[u] - p[pl]) < abs(
            p[pr] - p[pl])) {
            z = u;
            break;
        }
    }
    if (z != -1) {
        pr = z;
        continue;
    }
    break;
}
add_edge(pl, pr);
while (true) {
    int z = -1;
    bool b = false;
    for (int u : g[pl]) {
        long long c = ((p[pl] - p[pr]) ^ (p[u] - p[pr]))
    );
    if (c < 0 && (z == -1 || inside(p[pl], p[pr], p
        [z], p[u]))) z = u;
    }
    for (int u : g[pr]) {
        long long c = ((p[pr] - p[pl]) ^ (p[u] - p[pl]))
    );
    if (c > 0 && (z == -1 || inside(p[pl], p[pr], p
        [z], p[u]))) z = u, b = true;
    }
    if (z == -1) break;
    int x = pl, y = pr;
    if (b) swap(x, y);
    for (auto it = g[x].begin(); it != g[x].end(); )
    {
        int u = *it;
        if (intersect(p[x], p[u], p[y], p[z])) {
            it = g[x].erase(it);
            g[u].erase(x);
        } else {
            ++it;
        }
    }
    if (b) add_edge(pl, z), pr = z;
    else add_edge(pr, z), pl = z;
}
}
vector<vector<int>> solve(vector<point> v) {
    int n = v.size();
    for (int i = 0; i < n; ++i) g[i].clear();
    for (int i = 0; i < n; ++i) o[i] = i;
    sort(o, o + n, [&](int i, int j) { return v[i] < v[
        j]; });
    p.resize(n);

```

```

    for (int i = 0; i < n; ++i) p[i] = v[o[i]];
    dfs(0, n);
    vector<vector<int>> res(n);
    for (int i = 0; i < n; ++i) {
        for (int j : g[i]) res[o[i]].push_back(o[j]);
    }
    return res;
}

```

8.4 Sector Area

```

// calc area of sector which include a, b
double SectorArea(Point a, Point b, double r) {
    double theta = atan2(a.y, a.x) - atan2(b.y, b.x);
    while (theta <= 0) theta += 2 * pi;
    while (theta >= 2 * pi) theta -= 2 * pi;
    theta = min(theta, 2 * pi - theta);
    return r * r * theta / 2;
}

```

8.5 Polygon Area

```

// point sort in counterclockwise
double ConvexPolygonArea(vector<Point> &p, int n) {
    double area = 0;
    for (int i = 1; i < p.size() - 1; ++i) area += Cross(
        p[i] - p[0], p[i + 1] - p[0]);
    return area / 2;
}

```

8.6 Half Plane Intersection

```

bool jizz(Line l1, Line l2, Line l3) {
    Point p = intersect(l2, l3);
    return ((l1.pb - l1.pa) ^ (p - l1.pa)) < -eps;
}

bool cmp(const Line &a, const Line &b) {
    return same(a.angle, b.angle) ? ((b.pb - b.pa) ^ (a.pb - b.pa)) > eps : a.angle < b.angle;
}

// availble area for Line l is (l.pb - l.pa) ^ (p - l.pa) > 0
vector<Point> HPI(vector<Line> &ls) {
    sort(ls.begin(), ls.end(), cmp);
    vector<Line> pls(1, ls[0]);
    for (unsigned int i = 0; i < ls.size(); ++i) if (!same(ls[i].angle, pls.back().angle)) pls.push_back(ls[i]);
    deque<int> dq; dq.push_back(0); dq.push_back(1);
    for (unsigned int i = 2; i < pls.size(); ++i) {
        while (dq.size() > 1u && jizz(pls[i], pls[dq.back()], pls[dq[dq.size() - 2]])) dq.pop_back();
        while (dq.size() > 1u && jizz(pls[i], pls[dq[0]], pls[dq[1]])) dq.pop_front();
        dq.push_back(i);
    }
    while (dq.size() > 1u && jizz(pls[dq.front()], pls[dq.back()], pls[dq[dq.size() - 2]])) dq.pop_back();
    while (dq.size() > 1u && jizz(pls[dq.back()], pls[dq[0]], pls[dq[1]])) dq.pop_front();
    if (dq.size() < 3u) return vector<Point>(); // no solution or solution is not a convex
    vector<Point> rt;
    for (unsigned int i = 0; i < dq.size(); ++i) rt.push_back(intersect(pls[dq[i]], pls[dq[(i + 1) % dq.size()]]));
    return rt;
}

```

8.7 Rotating Sweep Line

```

void rotatingSweepLine(vector<pair<int, int>> &ps) {
    int n = int(ps.size());
    vector<int> id(n), pos(n);
    vector<pair<int, int>> line(n * (n - 1) / 2);
    int m = -1;
    for (int i = 0; i < n; ++i) for (int j = i + 1; j < n; ++j) line[++m] = make_pair(i, j); ++m;
    sort(line.begin(), line.end(), [&](const pair<int, int> &a, const pair<int, int> &b) -> bool {
        if (ps[a.first].first == ps[a.second].first) return 0;
        if (ps[b.first].first == ps[b.second].first) return 1;
        return (double)(ps[a.first].second - ps[a.second].second) / (ps[a.first].first - ps[a.second].first) < (double)(ps[b.first].second - ps[b.second].second) / (ps[b.first].first - ps[b.second].first);
    });
    for (int i = 0; i < n; ++i) id[i] = i;
    sort(id.begin(), id.end(), [&](const int &a, const int &b) { return ps[a] < ps[b]; });
    for (int i = 0; i < n; ++i) pos[id[i]] = i;

    for (int i = 0; i < m; ++i) {
        auto l = line[i];
        // meow
        tie(pos[l.first], pos[l.second], id[pos[l.first]], id[pos[l.second]]) = make_tuple(pos[l.second], pos[l.first], l.second, l.first);
    }
}

```

8.8 Triangle Center

```

Point TriangleCircumCenter(Point a, Point b, Point c) {
    Point res;
    double a1 = atan2(b.y - a.y, b.x - a.x) + pi / 2;
    double a2 = atan2(c.y - b.y, c.x - b.x) + pi / 2;
    double ax = (a.x + b.x) / 2;
    double ay = (a.y + b.y) / 2;
    double bx = (c.x + b.x) / 2;
    double by = (c.y + b.y) / 2;
    double r1 = (sin(a2) * (ax - bx) + cos(a2) * (by - ay)) / (sin(a1) * cos(a2) - sin(a2) * cos(a1));
    return Point(ax + r1 * cos(a1), ay + r1 * sin(a1));
}

Point TriangleMassCenter(Point a, Point b, Point c) {
    return (a + b + c) / 3.0;
}

Point TriangleOrthoCenter(Point a, Point b, Point c) {
    return TriangleMassCenter(a, b, c) * 3.0 - TriangleCircumCenter(a, b, c) * 2.0;
}

Point TriangleInnerCenter(Point a, Point b, Point c) {
    Point res;
    double la = len(b - c);
    double lb = len(a - c);
    double lc = len(a - b);
    res.x = (la * a.x + lb * b.x + lc * c.x) / (la + lb + lc);
    res.y = (la * a.y + lb * b.y + lc * c.y) / (la + lb + lc);
    return res;
}

```

8.9 Polygon Center

```

Point BaryCenter(vector<Point> &p, int n) {
    Point res(0, 0);
    double s = 0.0, t;
    for (int i = 1; i < p.size() - 1; ++i) {
        t = Cross(p[i] - p[0], p[i + 1] - p[0]) / 2;
        s += t;
        res.x += (p[0].x + p[i].x + p[i + 1].x) * t;
        res.y += (p[0].y + p[i].y + p[i + 1].y) * t;
    }
}

```

```

res.x /= (3 * s);
res.y /= (3 * s);
return res;
}

```

8.10 Maximum Triangle

```

double ConvexHullMaxTriangleArea(Point p[], int res[],
    int chnum) {
    double area = 0, tmp;
    res[chnum] = res[0];
    for (int i = 0, j = 1, k = 2; i < chnum; i++) {
        while (fabs(Cross(p[res[j]] - p[res[i]], p[res[(k + 1) % chnum]] - p[res[i]])) > fabs(Cross(p[res[j]] - p[res[i]], p[res[k]] - p[res[i]]))) k = (k + 1) % chnum;
        tmp = fabs(Cross(p[res[j]] - p[res[i]], p[res[k]] - p[res[i]]));
        if (tmp > area) area = tmp;
        while (fabs(Cross(p[res[(j + 1) % chnum]] - p[res[i]], p[res[k]] - p[res[i]])) > fabs(Cross(p[res[j]] - p[res[i]], p[res[k]] - p[res[i]]))) j = (j + 1) % chnum;
        tmp = fabs(Cross(p[res[j]] - p[res[i]], p[res[k]] - p[res[i]]));
        if (tmp > area) area = tmp;
    }
    return area / 2;
}

```

8.11 Point in Polygon

```

bool on(point a, point b, point c) {
    if (a.x == b.x) {
        if (c.x != a.x) return false;
        if (c.y >= min(a.y, b.y) && c.y <= max(a.y, b.y))
            return true;
        return false;
    }
    if (((a - c) ^ (b - c)) != 0) return false;
    if (a.x > b.x) swap(a, b);
    if (c.x < min(a.x, b.x) || c.x > max(a.x, b.x))
        return false;
    return ((a - b) ^ (a - c)) == 0;
}

int sgn(long long x) {
    if (x > 0) return 1;
    if (x < 0) return -1;
    return 0;
}

bool in(const vector<point> &c, point p) {
    int last = -2;
    int n = c.size();
    for (int i = 0; i < c.size(); ++i) {
        if (on(c[i], c[(i + 1) % n], p)) return true;
        int g = sgn((c[i] - p) ^ (c[(i + 1) % n] - p));
        if (last == -2) last = g;
        else if (last != g) return false;
    }
    return true;
}

bool in(point a, point b, point c, point p) {
    return in({ a, b, c }, p);
}

bool inside(const vector<point> &ch, point t) {
    point p = ch[1] - ch[0];
    point q = t - ch[0];
    if ((p ^ q) < 0) return false;
    if ((p ^ q) == 0) {
        if (p * q < 0) return false;
        if (q.len() > p.len()) return false;
        return true;
    }
    p = ch[ch.size() - 1] - ch[0];

```

```

    if ((p ^ q) > 0) return false;
    if ((p ^ q) == 0) {
        if (p * q < 0) return false;
        if (q.len() > p.len()) return false;
        return true;
    }
    p = ch[1] - ch[0];
    double ang = acos(1.0 * (p * q) / p.len() / q.len());
    int d = 20, z = ch.size() - 1;
    while (d-- > 0) {
        if (z - (1 << d) < 1) continue;
        point p1 = ch[1] - ch[0];
        point p2 = ch[z - (1 << d)] - ch[0];
        double tang = acos(1.0 * (p1 * p2) / p1.len() / p2.len());
        if (tang >= ang) z -= (1 << d);
    }
    return in(ch[0], ch[z - 1], ch[z], t);
}

```

8.12 Circle-Line Intersection

```

// remove second level if to get points for line (
// default: segment)
void CircleCrossLine(Point a, Point b, Point o, double
    r, Point ret[], int &num) {
    double x0 = o.x, y0 = o.y;
    double x1 = a.x, y1 = a.y;
    double x2 = b.x, y2 = b.y;
    double dx = x2 - x1, dy = y2 - y1;
    double A = dx * dx + dy * dy;
    double B = 2 * dx * (x1 - x0) + 2 * dy * (y1 - y0);
    double C = (x1 - x0) * (x1 - x0) + (y1 - y0) * (y1 - y0) - r * r;
    double delta = B * B - 4 * A * C;
    num = 0;
    if (epssgn(delta) >= 0) {
        double t1 = (-B - sqrt(fabs(delta))) / (2 * A);
        double t2 = (-B + sqrt(fabs(delta))) / (2 * A);
        if (epssgn(t1 - 1.0) <= 0 && epssgn(t1) >= 0) ret[num++] = Point(x1 + t1 * dx, y1 + t1 * dy);
        if (epssgn(t2 - 1.0) <= 0 && epssgn(t2) >= 0) ret[num++] = Point(x1 + t2 * dx, y1 + t2 * dy);
    }
}

vector<Point> CircleCrossLine(Point a, Point b, Point o
    , double r) {
    double x0 = o.x, y0 = o.y;
    double x1 = a.x, y1 = a.y;
    double x2 = b.x, y2 = b.y;
    double dx = x2 - x1, dy = y2 - y1;
    double A = dx * dx + dy * dy;
    double B = 2 * dx * (x1 - x0) + 2 * dy * (y1 - y0);
    double C = (x1 - x0) * (x1 - x0) + (y1 - y0) * (y1 - y0) - r * r;
    double delta = B * B - 4 * A * C;
    vector<Point> ret;
    if (epssgn(delta) >= 0) {
        double t1 = (-B - sqrt(fabs(delta))) / (2 * A);
        double t2 = (-B + sqrt(fabs(delta))) / (2 * A);
        if (epssgn(t1 - 1.0) <= 0 && epssgn(t1) >= 0) ret.emplace_back(x1 + t1 * dx, y1 + t1 * dy);
        if (epssgn(t2 - 1.0) <= 0 && epssgn(t2) >= 0) ret.emplace_back(x1 + t2 * dx, y1 + t2 * dy);
    }
    return ret;
}

```

8.13 Circle-Triangle Intersection

```

// calc area intersect by circle with radius r and
// triangle OAB
double Calc(Point a, Point b, double r) {
    Point p[2];
    int num = 0;
    bool ina = epssgn(len(a) - r) < 0, inb = epssgn(len(b) - r) < 0;

```

```

if (ina) {
    if (inb) return fabs(Cross(a, b)) / 2.0; //
    triangle in circle
    else { // a point inside and another outside: calc
        sector and triangle area
        CircleCrossLine(a, b, Point(0, 0), r, p, num);
        return SectorArea(b, p[0], r) + fabs(Cross(a, p
[0])) / 2.0;
    }
} else {
    CircleCrossLine(a, b, Point(0, 0), r, p, num);
    if (inb) return SectorArea(p[0], a, r) + fabs(Cross
(p[0], b)) / 2.0;
    else {
        if (num == 2) return SectorArea(a, p[0], r) +
SectorArea(p[1], b, r) + fabs(Cross(p[0], p[1])) /
2.0; // segment ab has 2 point intersect with
circle
        else return SectorArea(a, b, r); // segment has
no intersect point with circle
    }
}
}
}

```

8.14 Polygon Diameter

```

// get diameter of p[res[]] store opposite points in
app
double Diameter(Point p[], int res[], int chnum, int
app[][2], int &appnum) {
    double ret = 0, nowlen;
    res[chnum] = res[0];
    appnum = 0;
    for (int i = 0, j = 1; i < chnum; ++i) {
        while (Cross(p[res[i]] - p[res[i + 1]], p[res[j] +
1]) - p[res[i + 1]] < Cross(p[res[i]] - p[res[i +
1]], p[res[j]] - p[res[i + 1]])) {
            ++j;
            j %= chnum;
        }
        app[appnum][0] = res[i];
        app[appnum][1] = res[j];
        ++appnum;
        nowlen = dis(p[res[i]], p[res[j]]);
        if (nowlen > ret) ret = nowlen;
        nowlen = dis(p[res[i + 1]], p[res[j + 1]]);
        if (nowlen > ret) ret = nowlen;
    }
    return ret;
}

```

8.15 Minimun Distance of 2 Polygons

```

// p, q is convex
double TwoConvexHullMinDist(Point P[], Point Q[], int n
, int m) {
    int YMinP = 0, YMaxQ = 0;
    double tmp, ans = 999999999;
    for (i = 0; i < n; ++i) if (P[i].y < P[YMinP].y) YMinP
= i;
    for (i = 0; i < m; ++i) if (Q[i].y > Q[YMaxQ].y) YMaxQ
= i;
    P[n] = P[0], Q[m] = Q[0];
    for (int i = 0; i < n; ++i) {
        while (tmp = Cross(Q[YMaxQ + 1] - P[YMinP + 1], P[
YMinP] - P[YMinP + 1]) > Cross(Q[YMaxQ] - P[YMinP +
1], P[YMinP] - P[YMinP + 1])) YMaxQ = (YMaxQ + 1)
% m;
        if (tmp < 0) ans = min(ans, PointToSegDist(P[YMinP
], P[YMinP + 1], Q[YMaxQ]));
        else ans = min(ans, TwoSegMinDist(P[YMinP], P[YMinP
+ 1], Q[YMaxQ], Q[YMaxQ + 1]));
        YMinP = (YMinP + 1) % n;
    }
    return ans;
}

```

8.16 Convex Hull

```

vector<point> convex(vector<point> p) {
    sort(p.begin(), p.end());
    vector<point> ch;
    for (int i = 0; i < n; ++i) {
        while (ch.size() >= 2 && ((p[i] - ch[ch.size() -
2]) ^ (ch[ch.size() - 1] - ch[ch.size() - 2])) >=
0) ch.pop_back();
        ch.push_back(p[i]);
    }
    int t = ch.size();
    for (int i = n - 2; i >= 0; --i) {
        while (ch.size() > t && ((p[i] - ch[ch.size() - 2])
^ (ch[ch.size() - 1] - ch[ch.size() - 2])) >= 0)
            ch.pop_back();
        ch.push_back(p[i]);
    }
    ch.pop_back();
    return ch;
}

```

8.17 Rotating Caliper

```

struct pnt {
    int x, y;
    pnt(): x(0), y(0) {};
    pnt(int xx, int yy): x(xx), y(yy) {};
} p[maxn];

pnt operator-(const pnt &a, const pnt &b) { return pnt(
b.x - a.x, b.y - a.y); }
int operator^(const pnt &a, const pnt &b) { return a.x
* b.y - a.y * b.x; } //cross
int operator*(const pnt &a, const pnt &b) { return (a -
b).x * (a - b).x + (a - b).y * (a - b).y; } //
distance
int tb[maxn], tbz, rsd;

int dist(int n1, int n2){
    return p[n1] * p[n2];
}
int cross(int t1, int t2, int n1){
    return (p[t2] - p[t1]) ^ (p[n1] - p[t1]);
}
bool cmpx(const pnt &a, const pnt &b) { return a.x == b
.x ? a.y < b.y : a.x < b.x; }

void RotatingCaliper() {
    sort(p, p + n, cmpx);
    for (int i = 0; i < n; ++i) {
        while (tbz > 1 && cross(tb[tbz - 2], tb[tbz - 1], i
) <= 0) --tbz;
        tb[tbz++] = i;
    }
    rsd = tbz - 1;
    for (int i = n - 2; i >= 0; --i) {
        while (tbz > rsd + 1 && cross(tb[tbz - 2], tb[tbz -
1], i) <= 0) --tbz;
        tb[tbz++] = i;
    }
    --tbz;
    int lpr = 0, rpr = rsd;
    // tb[lpr], tb[rpr]
    while (lpr < rsd || rpr < tbz - 1) {
        if (lpr < rsd && rpr < tbz - 1) {
            pnt rvt = p[tb[rpr + 1]] - p[tb[rpr]];
            pnt lvt = p[tb[lpr + 1]] - p[tb[lpr]];
            if ((lvt ^ rvt) < 0) ++lpr;
            else ++rpr;
        }
        else if (lpr == rsd) ++rpr;
        else ++lpr;
        // tb[lpr], tb[rpr]
    }
}

```


8.18 Minimum Enclosing Circle

```

pt center(const pt &a, const pt &b, const pt &c) {
    pt p0 = b - a, p1 = c - a;
    double c1 = norm2(p0) * 0.5, c2 = norm2(p1) * 0.5;
    double d = p0 ^ p1;
    double x = a.x + (c1 * p1.y - c2 * p0.y) / d;
    double y = a.y + (c2 * p0.x - c1 * p1.x) / d;
    return pt(x, y);
}

circle min_enclosing(vector<pt> &p) {
    random_shuffle(p.begin(), p.end());
    double r = 0.0;
    pt cent;
    for (int i = 0; i < p.size(); ++i) {
        if (norm2(cent - p[i]) <= r) continue;
        cent = p[i];
        r = 0.0;
        for (int j = 0; j < i; ++j) {
            if (norm2(cent - p[j]) <= r) continue;
            cent = (p[i] + p[j]) / 2;
            r = norm2(p[j] - cent);
            for (int k = 0; k < j; ++k) {
                if (norm2(cent - p[k]) <= r) continue;
                cent = center(p[i], p[j], p[k]);
                r = norm2(p[k] - cent);
            }
        }
    }
    return circle(cent, sqrt(r));
}

```

8.19 Closest Pair

```

pt p[maxn];

double dis(const pt& a, const pt& b) {
    return sqrt((a - b) * (a - b));
}

double closest_pair(int l, int r) {
    if (l == r) return inf;
    if (r - l == 1) return dis(p[l], p[r]);
    int m = (l + r) >> 1;
    double d = min(closest_pair(l, m), closest_pair(m + 1, r));
    vector<int> vec;
    for (int i = m; i >= l && fabs(p[m].x - p[i].x) < d; --i) vec.push_back(i);
    for (int i = m + 1; i <= r && fabs(p[m].x - p[i].x) < d; ++i) vec.push_back(i);
    sort(vec.begin(), vec.end(), [=](const int& a, const int& b) { return p[a].y < p[b].y; });
    for (int i = 0; i < vec.size(); ++i) {
        for (int j = i + 1; j < vec.size() && fabs(p[vec[i]].y - p[vec[j]].y) < d; ++j) {
            d = min(d, dis(p[vec[i]], p[vec[j]]));
        }
    }
    return d;
}

```

9 Problems

9.1 Manhattan distance minimum spanning tree

```

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

const int maxn = 1e5 + 5;
int x[maxn], y[maxn], fa[maxn];
pair<int, int> bit[maxn];
vector<tuple<int, int, int>> ed;

```

```

void init() {
    for (int i = 0; i < maxn; ++i)
        bit[i] = make_pair(1e9, -1);
}

void add(int p, pair<int, int> v) {
    for (; p < maxn; p += p & -p)
        bit[p] = min(bit[p], v);
}

pair<int, int> query(int p) {
    pair<int, int> res = make_pair(1e9, -1);
    for (; p; p -= p & -p)
        res = min(res, bit[p]);
    return res;
}

void add_edge(int u, int v) {
    ed.emplace_back(u, v, abs(x[u] - x[v]) + abs(y[u] - y[v]));
}

void solve(int n) {
    init();
    vector<int> v(n), ds;
    for (int i = 0; i < n; ++i) {
        v[i] = i;
        ds.push_back(x[i] - y[i]);
    }
    sort(ds.begin(), ds.end());
    ds.resize(unique(ds.begin(), ds.end()) - ds.begin());
    sort(v.begin(), v.end(), [&](int i, int j) { return x[i] == x[j] ? y[i] > y[j] : x[i] > x[j]; });
    int j = 0;
    for (int i = 0; i < n; ++i) {
        int p = lower_bound(ds.begin(), ds.end(), x[v[i]] - y[v[i]]) - ds.begin() + 1;
        pair<int, int> q = query(p);
        if (~q.second) add_edge(v[i], q.second);
        add(p, make_pair(x[v[i]] + y[v[i]], v[i]));
    }

    int find(int x) {
        if (x == fa[x]) return x;
        return fa[x] = find(fa[x]);
    }

    void merge(int x, int y) {
        fa[find(x)] = find(y);
    }

    int main() {
        int n; scanf("%d", &n);
        for (int i = 0; i < n; ++i) scanf("%d %d", &x[i], &y[i]);
        solve(n);
        for (int i = 0; i < n; ++i) swap(x[i], y[i]);
        solve(n);
        for (int i = 0; i < n; ++i) x[i] = -x[i];
        solve(n);
        for (int i = 0; i < n; ++i) swap(x[i], y[i]);
        solve(n);
        sort(ed.begin(), ed.end(), [](const tuple<int, int, int> &a, const tuple<int, int, int> &b) { return get<2>(a) < get<2>(b); });
        for (int i = 0; i < n; ++i) fa[i] = i;
        long long ans = 0;
        for (int i = 0; i < ed.size(); ++i) {
            int x, y, w; tie(x, y, w) = ed[i];
            if (find(x) == find(y)) continue;
            merge(x, y);
            ans += w;
        }
        printf("%lld\n", ans);
        return 0;
    }
}

```

9.2 "Dynamic" kth element (parallel binary search)

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

const int maxn = 1e5 + 5;
int a[maxn], ans[maxn], tmp[maxn];

struct query { int op, l, r, k, qid; };

struct fenwick {
    int dat[maxn];
    void init() { memset(dat, 0, sizeof(dat)); }
    void add(int p, int v) { for (; p < maxn; p += p & -p) dat[p] += v; }
    int qry(int p, int v = 0) { for (; p; p -= p & -p) v += dat[p]; return v; }
} bit;

void bs(vector<query> &qry, int l, int r) {
    if (l == r) {
        for (int i = 0; i < qry.size(); ++i) {
            if (qry[i].op == 3) ans[qry[i].qid] = l;
        }
        return;
    }
    if (qry.size() == 0) return;
    int m = l + r >> 1;
    for (int i = 0; i < qry.size(); ++i) {
        if (qry[i].op == 1 && qry[i].r <= m) bit.add(qry[i].l, 1);
        else if (qry[i].op == 2 && qry[i].r <= m) bit.add(qry[i].l, -1);
        else if (qry[i].op == 3) tmp[qry[i].qid] += bit.qry(qry[i].r) - bit.qry(qry[i].l - 1);
    }
    vector<query> ql, qr;
    for (int i = 0; i < qry.size(); ++i) {
        if (qry[i].op == 3) {
            if (qry[i].k - tmp[qry[i].qid] > 0) qry[i].k -= tmp[qry[i].qid], qr.push_back(qry[i]);
            else ql.push_back(qry[i]);
            tmp[qry[i].qid] = 0;
            continue;
        }
        if (qry[i].r <= m) ql.push_back(qry[i]);
        else qr.push_back(qry[i]);
    }
    for (int i = 0; i < qry.size(); ++i) {
        if (qry[i].op == 1 && qry[i].r <= m) bit.add(qry[i].l, -1);
        else if (qry[i].op == 2 && qry[i].r <= m) bit.add(qry[i].l, 1);
    }
    bs(ql, l, m), bs(qr, m + 1, r);
}

int main() {
    int t; scanf("%d", &t);
    while (t--) {
        int n, q; scanf("%d %d", &n, &q);
        vector<query> qry;
        vector<int> ds;
        bit.init();
        for (int i = 1; i <= n; ++i) {
            scanf("%d", &a[i]); ds.push_back(a[i]);
            qry.push_back({ 1, i, a[i], -1, -1 });
        }
        int qid = 0;
        for (int i = 0; i < q; ++i) {
            int t; scanf("%d", &t);
            if (t == 1) {
                int l, r, k; scanf("%d %d %d", &l, &r, &k);
                qry.push_back({ 3, l, r, k, qid }); ++qid;
            }
            if (t == 2) {
                int c, v; scanf("%d %d", &c, &v);
                ds.push_back(v);
                qry.push_back({ 2, c, a[c], -1, -1 });
                qry.push_back({ 1, c, v, -1, -1 });
            }
        }
    }
}
```

```
        a[c] = v;
    }
    if (t == 3) {
        int x, v; scanf("%d %d", &x, &v);
        ans[qid] = -1, ++qid;
    }
}
sort(ds.begin(), ds.end()); ds.resize(unique(ds.begin(), ds.end()) - ds.begin());
for (int i = 0; i < qry.size(); ++i) {
    if (qry[i].op == 3) continue;
    qry[i].r = lower_bound(ds.begin(), ds.end(), qry[i].r) - ds.begin();
}
bs(qry, 0, ds.size() - 1);
for (int i = 0; i < qid; ++i) {
    if (ans[i] == -1) puts("7122");
    else assert(ans[i] < ds.size()), printf("%d\n", ds[ans[i]]);
}
return 0;
}
```

9.3 Dynamic kth element (persistent segment tree)

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

const int maxn = 1e5 + 5;
int a[maxn], bit[maxn];
vector<int> ds;
vector<vector<int>> &qr;

namespace segtree {
    int st[maxn * 97], lc[maxn * 97], rc[maxn * 97], sz;
    int gnode() {
        st[sz] = 0;
        lc[sz] = rc[sz] = 0;
        return sz++;
    }
    int gnode(int z) {
        st[sz] = st[z];
        lc[sz] = lc[z], rc[sz] = rc[z];
        return sz++;
    }
    int build(int l, int r) {
        int z = gnode();
        if (r - l == 1) return z;
        lc[z] = build(l, (l + r) / 2), rc[z] = build((l + r) / 2, r);
        return z;
    }
    int modify(int l, int r, int p, int v, int o) {
        int z = gnode(o);
        if (r - l == 1) return st[z] += v, z;
        if (p < (l + r) / 2) lc[z] = modify(l, (l + r) / 2, p, v, lc[o]);
        else rc[z] = modify((l + r) / 2, r, p, v, rc[o]);
        st[z] = st[lc[z]] + st[rc[z]];
        return z;
    }
    int query(int l, int r, int ql, int qr, int o) {
        if (l >= qr || ql >= r) return 0;
        if (l >= ql && r <= qr) return st[o];
        return query(l, (l + r) / 2, ql, qr, lc[o]) + query((l + r) / 2, r, ql, qr, rc[o]);
    }
}

void init(int n) {
    segtree::sz = 0;
    bit[0] = segtree::build(0, ds.size());
    for (int i = 1; i <= n; ++i) bit[i] = bit[0];
}

void add(int p, int n, int x, int v) {
    for (; p <= n; p += p & -p)
```



```

    bit[p] = segtree::modify(0, ds.size(), x, v, bit[p]);
}

vector<int> query(int p) {
    vector<int> z;
    for (; p; p -= p & -p)
        z.push_back(bit[p]);
    return z;
}

int dfs(int l, int r, vector<int> lz, vector<int> rz,
        int k) {
    if (r - l == 1) return l;
    int ls = 0, rs = 0;
    for (int i = 0; i < lz.size(); ++i) ls += segtree::st
[segtree::lc[lz[i]]];
    for (int i = 0; i < rz.size(); ++i) rs += segtree::st
[segtree::lc[rz[i]]];
    if (rs - ls >= k) {
        for (int i = 0; i < lz.size(); ++i) lz[i] = segtree
::lc[lz[i]];
        for (int i = 0; i < rz.size(); ++i) rz[i] = segtree
::lc[rz[i]];
        return dfs(l, (l + r) / 2, lz, rz, k);
    } else {
        for (int i = 0; i < lz.size(); ++i) lz[i] = segtree
::rc[lz[i]];
        for (int i = 0; i < rz.size(); ++i) rz[i] = segtree
::rc[rz[i]];
        return dfs((l + r) / 2, r, lz, rz, k - (rs - ls));
    }
}

int main() {
    int t; scanf("%d", &t);
    while (t--) {
        int n, q; scanf("%d %d", &n, &q);
        for (int i = 1; i <= n; ++i) scanf("%d", &a[i]), ds
.push_back(a[i]);
        for (int i = 0; i < q; ++i) {
            int a, b, c; scanf("%d %d %d", &a, &b, &c);
            vector<int> v = { a, b, c };
            if (a == 1) {
                int d; scanf("%d", &d);
                v.push_back(d);
            }
            qr.push_back(v);
        }
        for (int i = 0; i < q; ++i) if (qr[i][0] == 2) ds.
push_back(qr[i][2]);
        sort(ds.begin(), ds.end()), ds.resize(unique(ds.
begin(), ds.end()) - ds.begin());
        for (int i = 1; i <= n; ++i) a[i] = lower_bound(ds.
begin(), ds.end(), a[i]) - ds.begin();
        for (int i = 0; i < q; ++i) if (qr[i][0] == 2) qr[i
][2] = lower_bound(ds.begin(), ds.end(), qr[i][2])
- ds.begin();
        init(n);
        for (int i = 1; i <= n; ++i) add(i, n, a[i], 1);
        for (int i = 0; i < q; ++i) {
            if (qr[i][0] == 3) {
                puts("7122");
                continue;
            }
            if (qr[i][0] == 1) {
                vector<int> lz = query(qr[i][1] - 1);
                vector<int> rz = query(qr[i][2]);
                int ans = dfs(0, ds.size(), lz, rz, qr[i][3]);
                printf("%d\n", ds[ans]);
            } else {
                add(qr[i][1], n, a[qr[i][1]], -1);
                add(qr[i][1], n, qr[i][2], 1);
                a[qr[i][1]] = qr[i][2];
            }
        }
        ds.clear(), qr.clear();
    }
    return 0;
}

```

9.4 Hilbert's curve (faster MO's algorithm)

```

long long hilbert(int n, int x, int y) {
    long long res = 0;
    for (int s = n / 2; s; s >>= 1) {
        int rx = (x & s) > 0;
        int ry = (y & s) > 0;
        res += s * 1ll * s * ((3 * rx) ^ ry);
        if (ry == 0) {
            if (rx == 1) {
                x = s - 1 - x;
                y = s - 1 - y;
            }
            swap(x, y);
        }
    }
    return res;
}

```