Team unimi

Team Reference Document

24/01/2020

Contents

2. Trees

2.1. Lca.

```
// LCA with binary lifting ------------------------------//d7
// just add node with pointer to father ----------------//83
// be careful as memory is MAXN * MAXJ -----------------//21
// MAXJ should be >= log2(N) ---------------------------//40
const int MAXN = 1e5; -------------------------------------//3e
const int MAXJ = 20; --------------------------------------//d0
struct node { ---------------------------------------------//cd
- node *p, *jmp[MAXJ]; ------------------------------------//cd
- int depth; ----------------------------------------------//07
- node(node *_p = NULL) : p(_p) { -------------------------//14
--- depth = p ? 1 + p->depth : 0; -------------------------//92
--- memset(jmp, 0, sizeof(jmp)); --------------------------//f3
--- jmp[0] = p; -------------------------------------------//40
--- for (int i = 1; (1<<i) <= depth; i++) -----------------//b1
----- jmp[i] = jmp[i-1]->jmp[i-1]; } }; -------------------//c0
node* st[MAXN]; -------------------------------------------//88
node* lca(node *a, node *b) { -----------------------------//a7
- if (!a || !b) return NULL; ------------------------------//90
- if (a->depth < b->depth) swap(a,b); ---------------------//ac
- for (int j = 19; j >= 0; j--) ---------------------------//5b
--- while (a->depth - (1<<j) >= b->depth) a = a->jmp[j]; --//47
- if (a == b) return a; -----------------------------------//dc
- for (int j = 19; j >= 0; j--) ---------------------------//1c
--- while (a->depth >= (1<<j) && a->jmp[j] != b->jmp[j]) --//bf
----- a = a->jmp[j], b = b->jmp[j]; ---------------------//b9
- return a->p; } -------------------------------------------//3d
```

2.2. Hld.

```
vector<int> g[MN]; //graph --------------------------------//78
vector<int> parent(n),hldpos(n),chain(n),chain_d(n),dim(n);//2b
//computes subtree dimensions, stores parent of each node -//d6
void hlddfs0(int p){ --------------------------------------//9c
--- dim[p]=1; ---------------------------------------------//b2
--- for(auto i:g[p]){ -------------------------------------//e8
------- if(i!=parent[p]){ -------------------------------//29
---------- parent[i]=p; -----------------------------------//d6
---------- dfs1(i); ---------------------------------------//f8
---------- dim[p]+=dim[i]; --------------------------------//75
------- } -----------------------------------------------//01
--- } -----------------------------------------------------//41
} ---------------------------------------------------------//94
//heavy light decomhldposition ------------------------//df
void hlddfs1(int p, int chain_id, int cd){ ----------------//1e
--- hldpos[p]=cnt++; //hldposition in base array for rt queries
--- chain[p]=chain_id; // highest node in chain ----------//4b
--- chain_d[p]=cd; // chain depth -----------------------//db
--- int hldson=-1; ----------------------------------------//86
--- for(auto i:g[p])if(i!=parent[p]&&(hldson==-1||dim[i]>dim[hldso
--- if(hldson!=-1){ //not a leaf node --------------------//fe
------- buildhld(hldson,chain_id,cd); //heavy child -------//d7
------- for(auto i:G[p]){ --------------------------------//75
---------- if(i!=parent[p] && i!=hldson[p]) //light edge -//bf
--------------- buildhld(i,i,cd+1); ----------------------//76
------- } -----------------------------------------------//4d
--- } -----------------------------------------------------//12
} ---------------------------------------------------------//16
parent[0]=-1; ---------------------------------------------//23
hlddfs0(0); -----------------------------------------------//45
hlddfs1(0,0,0); -------------------------------------------//d3
// build ds with new indeces (in hldpos) ----------------//e4
int query(int x, int y){ ----------------------------------//a8
--- int ans=0; --------------------------------------------//c7
--- if(chain_d[chain[x]]>chain_d[chain[y]]) ---------------//96
------- swap(x,y); --------------------------------------//f2
--- for(int dif = chain_d[chain[y]]-chain_d[chain[x]]; dif; --dif)
------- ans=max(ans,rmq(hldpos[chain[y]],hldpos[y]); ------//6c
------- y=chain[y]; y=parent[y]; ------------------------//30
--- } -----------------------------------------------------//12
--- while(chain[x]!=chain[y]){ ----------------------------//9e
------- ans=max(ans,rmq(hldpos[chain[y]],hldpos[y]); ------//f1
------- y=chain[y]; y=parent[y]; ------------------------//18
------- ans=max(ans,rmq(hldpos[chain[x]],hldpos[x]); ------//dc
------- x=chain[x]; x=parent[x]; ------------------------//e9
--- } -----------------------------------------------------//81
--- if(hldpos[x]<hldpos[y]) -------------------------------//73
------- ans=max(ans,rmq(hldpos[x],hldpos[y]); -------------//78
--- else --------------------------------------------------//69
------- ans=max(ans,rmq(hldpos[y],hldpos[x]); -------------//ed
--- return ans; -------------------------------------------//8f
} ---------------------------------------------------------//c9
 ----------------------------------------------------------//f0
```

2.3. Centroid Decomp.

```
/* Centroid decomposition, O(NlogN) memory/time ----------//59
- Given a 0-rooted tree, build a tree of height logN -----//7d
- then process it in order to answer A to B (path) queries //e7
- Can also be used in other situations -----------------//6c
*/ ----------------------------------------------------------//e3
struct centroid_decomposition{ ----------------------------//7f
- vvi &g, c; ----------------------------------------------//f2
- vi size; ------------------------------------------------//8e
- vi parent; ----------------------------------------------//11
- int n, root; --------------------------------------------//d7
- vvi memo; -----------------------------------------------//5d
- int height; ---------------------------------------------//97
- vector<char> level; -------------------------------------//a2
- int dfs(int i){ ------------------------------------------//f9
--- size[i] = 1; -------------------------------------------//46
```

```
--- for(int x : g[i]) ------------------------------------//b5
----- dfs(x), size[i] += size[x]; ------------------------//85
--- return size[i]; --------------------------------------//18
- } ---------------------------------------------------//05
- bool is_centroid(int i, int r){ ---------------------//02
--- for(int x : g[i]) ------------------------------------//ba
----- if(size[x] > size[r]/2) ------------------------------//aa
------- return 0; ------------------------------------------//3f
--- return 1; } ------------------------------------------//ed
- int prefered(int i){ --------------------------------//bd
--- if(g[i].size() == 0) ---------------------------------//e8
----- return i; --------------------------------------------//72
--- int x = g[i].front(); --------------------------------//02
--- for(int e : g[i]) ------------------------------------//75
----- if(size[e] > size[x]) --------------------------------//6f
------- x = e; ---------------------------------------------//a9
--- return x; } ------------------------------------------//14
- int centroidify(int i, int p){ ----------------------//ff
--- int curr = i; ----------------------------------------//e5
--- vi path; ---------------------------------------------//51
--- while(!is_centroid(curr, i)){ ------------------------//59
----- path.push_back(curr); --------------------------------//84
----- curr = prefered(curr); -------------------------------//24
--- } ----------------------------------------------------//1d
--- for(int p : path) ------------------------------------//a3
----- size[p] -= size[curr]; -------------------------------//79
--- size[curr] = 0; --------------------------------------//e0
--- parent[curr] = p; ------------------------------------//9f
--- if(i != curr) ----------------------------------------//5c
----- c[curr].push_back(centroidify(i, curr)); ------------//00
--- for(int x : g[curr]) if(size[x]) ---------------------//70
----- c[curr].push_back(centroidify(x, curr)); ------------//f3
--- return curr; } ---------------------------------------//4c
- // i think this should be done in centroidify -----------//8e
- void level_dfs(int i){ -------------------------------//07
--- for(int x : c[i]){ -----------------------------------//40
----- level[x] = level[i]+1; -------------------------------//cd
----- level_dfs(x); ----------------------------------------//66
--- } } --------------------------------------------------//c5
 ----------------------------------------------------------//9b
- /* IMPORTANT: -------------------------------------------//b0
--- if you need to answer query, -------------------------//90
--- you can just precompute PATHS ------------------------//0e
--- from/to each centroid to all his ---------------------//26
--- childs. Then you can answer A->B queries using -------//3d
--- LCA and this precomputed paths */ --------------------//a1
 ----------------------------------------------------------//a1
- void decompose(){ ------------------------------------//51
--- dfs(0); ----------------------------------------------//5f
--- int root = centroidify(0, -1); -----------------------//cb
--- level[root] = 0; -------------------------------------//96
--- level_dfs(root); -------------------------------------//d8
--- height = *max_element(level.begin(), level.end()) + 1; //83
--- fill(size.begin(), size.end() , 1); ------------------//40
- } ----------------------------------------------------//f8
- centroid_decomposition(vvi &source) : g(source), n(source.size()),
--- c(source.size()), size(source.size()), level(source.size()),
--- parent(source.size()) { ------------------------------//40
--- decompose(); -----------------------------------------//6c
- } ----------------------------------------------------//50
}; ---------------------------------------------------//d7
```

3. Strings

3.1. Suffix Tree2.

```
#include <bits/stdc++.h> -------------------------------//84
using namespace std; ---------------------------------------//16
 ----------------------------------------------------------//11
const int MAXN = 2e5; --------------------------------------//19
 ----------------------------------------------------------//a3
string s; --------------------------------------------------//d0
int n; -----------------------------------------------------//42
 ----------------------------------------------------------//2c
struct node { ----------------------------------------------//8b
--- int l, r, par, link; ---------------------------------//d3
--- map<char,int> next; ----------------------------------//b2
 ----------------------------------------------------------//86
--- node (int l=0, int r=0, int par=-1) ------------------//9a
------- : l(l), r(r), par(par), link(-1) {} ---------------//d3
--- int len()  {  return r - l;  } -----------------------//b1
--- int &get (char c) { ----------------------------------//01
------- if (!next.count(c))  next[c] = -1; ---------------//0d
------- return next[c]; --------------------------------//dc
--- } ----------------------------------------------------//cd
--- void print(){ ----------------------------------------//fa
----- cout << l << ' ' << r << ' ' << par << ' ' << link << '\n';
--- } ----------------------------------------------------//8c
}; ---------------------------------------------------//fb
node t[MAXN]; ----------------------------------------------//00
int sz; ----------------------------------------------------//5f
 ----------------------------------------------------------//1a
struct state { ---------------------------------------------//b9
--- int v, pos; ------------------------------------------//0a
--- state (int v, int pos) : v(v), pos(pos)  {} ----------//fe
}; ---------------------------------------------------//0e
state ptr (0, 0); ------------------------------------------//4b
 ----------------------------------------------------------//a0
state go (state st, int l, int r) { ------------------//91
--- while (l < r) ----------------------------------------//01
------- if (st.pos == t[st.v].len()) { -------------------//2f
----------- st = state (t[st.v].get( s[l] ), 0); ----------//f1
----------- if (st.v == -1)  return st; -------------------//c4
------- } ------------------------------------------------//fa
------- else { -------------------------------------------//de
----------- if (s[ t[st.v].l + st.pos ] != s[l]) ----------//51
--------------- return state (-1, -1); -------------------//28
----------- if (r-l < t[st.v].len() - st.pos) -------------//ec
--------------- return state (st.v, st.pos + r-l); --------//0f
----------- l += t[st.v].len() - st.pos; -------------------//11
----------- st.pos = t[st.v].len(); ----------------------//e9
------- } ------------------------------------------------//55
--- return st; -------------------------------------------//c2
} ----------------------------------------------------//45
 ----------------------------------------------------------//d9
int split (state st) { -------------------------------//d1
--- if (st.pos == t[st.v].len()) -------------------------//59
------- return st.v; ---------------------------------------//9c
--- if (st.pos == 0) -------------------------------------//b3
------- return t[st.v].par; --------------------------------//b7
--- node v = t[st.v]; ------------------------------------//2a
--- int id = sz++; ---------------------------------------//99
--- t[id] = node (v.l, v.l+st.pos, v.par); ----------------//27
--- t[v.par].get( s[v.l] ) = id; -------------------------//67
--- t[id].get( s[v.l+st.pos] ) = st.v; -------------------//bf
--- t[st.v].par = id; ------------------------------------//da
--- t[st.v].l += st.pos; ---------------------------------//0a
--- return id; -------------------------------------------//1c
} ----------------------------------------------------//cc
 ----------------------------------------------------------//fa
int get_link (int v) { -------------------------------//ce
--- if (t[v].link != -1)  return t[v].link; --------------//07
--- if (t[v].par == -1)  return 0; -----------------------//68
--- int to = get_link (t[v].par); ------------------------//0f
--- return t[v].link = split (go (state(to,t[to].len()), t[v].l +
} ----------------------------------------------------//33
 ----------------------------------------------------------//d6
void tree_extend (int pos) { -------------------------//75
--- for(;;) { --------------------------------------------//18
------- state nptr = go (ptr, pos, pos+1); -------------//0a
------- if (nptr.v != -1) { ----------------------------//ce
----------- ptr = nptr; ------------------------------------//5d
----------- return; ----------------------------------------//6d
------- } ----------------------------------------------//70
 ----------------------------------------------------------//fb
------- int mid = split (ptr); -------------------------//7f
------- int leaf = sz++; -------------------------------//0f
------- t[leaf] = node (pos, n, mid); ------------------//be
------- t[mid].get( s[pos] ) = leaf; -------------------//a4
 ----------------------------------------------------------//24
------- ptr.v = get_link (mid); ------------------------//7c
------- ptr.pos = t[ptr.v].len(); ----------------------//76
------- if (!mid)  break; ------------------------------//76
--- } ----------------------------------------------------//8f
} ----------------------------------------------------//d7
 ----------------------------------------------------------//45
void build_tree() { ----------------------------------//e1
--- sz = 1; ----------------------------------------------//71
--- for (int i=0; i<n; ++i) ------------------------------//75
------- tree_extend (i); -------------------------------//96
} ----------------------------------------------------//fa
 ----------------------------------------------------------//f6
 ----------------------------------------------------------//69
int main() { -----------------------------------------//3a
- cin >> s; ------------------------------------------//c9
- n = s.size(); --------------------------------------//79
- build_tree(); --------------------------------------//f6
 ----------------------------------------------------------//f4
- node root = t[0]; ----------------------------------//60
 ----------------------------------------------------------//82
```

```
- vector<bool> exists(1000); ----------//0c
- for(char c : s) ----------//af
--- exists[c] = 1; ----------//c5
 ----------//08
 ----------//ff
- while(){ ----------//f1
--- string x; cin >> x; ----------//e4
--- bool ok = 1; ----------//11
--- for(char c : x) ----------//50
----- if(exists[c] == 0){ ----------//8b
------- ok = 0; ----------//74
------- break; ----------//ea
----- } ----------//18
--- if(!ok) { ----------//5b
----- cout << -1 << '\n', 0; ----------//0f
----- continue; ----------//85
--- } ----------//67
--- node curr = root; ----------//78
--- int l = 0, r = 0; ----------//b7
--- int blocks = 1; ----------//90
--- for(int i = 0; i < x.size(); i++) { ----------//2d
----- if(l >= r) { ----------//62
------- int next = curr.get(x[i]); ----------//a5
------- // cout << "block is over, next is " << next << '\n';
------- if(next == -1) { ----------//6b
--------- blocks++; ----------//5b
--------- // cout << "finished matching, new block\n"; ----//02
--------- curr = root; ----------//71
--------- // repeat this letter, but from root ----------//01
--------- i--; ----------//99
--------- continue; ----------//5d
------- } ----------//7e
------- curr = t[next]; ----------//08
------- l = curr.l, r = curr.r; ----------//61
------- // cout << "could match " << l << " to " << r << '\n';
----- } ----------//e9
----- // match next one ----------//f7
----- if(s[l] != x[i]) { ----------//c6
------- blocks++; ----------//a7
------- // cout << "new block in " << i << '\n'; ----------//f0
------- curr = root; ----------//8f
------- l = curr.l, r = curr.r; ----------//aa
------- i--; ----------//60
----- } else{ ----------//da
------- // cout << "match " << i << " in " << l << '\n'; --//b2
------- l++; ----------//d3
----- } ----------//7b
--- } ----------//ad
--- // cout << "\nsolved in " << blocks << "\n\n"; --------//be
--- cout << blocks << '\n'; ----------//79
- } ----------//79
 ----------//c1
} ----------//0e
```

3.2. Suffix Tree.

```
// O(N * logAlfa) Suffix Tree construction ----------//70
 ----------//f3
const int inf = 1e9; ----------//ef
const int maxn = 1e4; ----------//74
char s[maxn]; ----------//47
map<int, int> to[maxn]; ----------//0d
int len[maxn], f_pos[maxn], link[maxn]; ----------//52
int node, pos; ----------//20
int sz = 1, n = 0; ----------//4b
int make_node(int _pos, int _len){ ----------//ad
- f_pos[sz] = _pos; ----------//c5
- len [sz] = _len; ----------//3c
- return sz++; ----------//33
} ----------//05
void go_edge(){ ----------//43
- while(pos > len[to[node][s[n - pos]]]){ ----------//28
--- node = to[node][s[n - pos]]; ----------//76
--- pos -= len[node]; ----------//f5
- } ----------//27
} ----------//df
void add_letter(int c){ ----------//b2
- s[n++] = c; pos++; ----------//70
- int last = 0; ----------//8f
- while(pos > 0){ ----------//75
--- go_edge(); ----------//ce
--- int edge = s[n - pos]; ----------//8e
--- int &v = to[node][edge]; ----------//05
--- int t = s[f_pos[v] + pos - 1]; ----------//83
--- if(v == 0){ ----------//b6
----- v = make_node(n - pos, inf); ----------//c6
----- link[last] = node; ----------//2d
----- last = 0; ----------//65
--- } else if(t == c){ ----------//2d
----- link[last] = node; ----------//fc
----- return; ----------//3e
--- } else{ ----------//39
----- int u = make_node(f_pos[v], pos - 1); ----------//a2
----- to[u][c] = make_node(n - 1, inf); ----------//1e
----- to[u][t] = v; ----------//37
----- f_pos[v] += pos - 1; ----------//b6
----- len [v] -= pos - 1; ----------//3f
----- v = u; ----------//94
----- link[last] = u; ----------//3e
----- last = u; ----------//27
--- } ----------//ce
--- if(node == 0) pos--; ----------//93
--- else  node = link[node]; ----------//9a
- } ----------//3f
} ----------//3f
int main(){ ----------//b0
- len[0] = inf; ----------//8b
- string s; ----------//f6
- cin >> s; ----------//ba
- int ans = 0; ----------//92
- for(int i = 0; i < s.size(); i++) ----------//a2
--- add_letter(s[i]); ----------//4d
- for(int i = 1; i < sz; i++) ----------//53
--- ans += min((int)s.size() - f_pos[i], len[i]); ---------//51
- cout << ans << "\n"; ----------//2d
} ----------//99
```

3.3. Suffix Array.

```
//O(NlogN) suffixarray - add a $ char at the end ----------//36
 ----------//53
struct suffarray{ ----------//76
- vi sa; ----------//94
- string &s; ----------//52
- void radix_sort(vi &rank, int k){ ----------//fd
--- vi count(max(256, (int)sa.size())); ----------//54
--- for(int i : sa) count[((i + k) < rank.size() ? rank[i+k]+1 : 1
--- int last = 0; ----------//23
--- for(int &i : count) i = last += i; ----------//00
--- vi temp(sa.size()); ----------//4d
--- for(int i : sa) temp[count[(i + k < rank.size() ? rank[i + k]
--- fill(count.begin(), count.end(), 0); ----------//2c
--- for(int &i : rank) count[i+1]++; ----------//78
--- last = 0; ----------//96
--- for(int &i : count) i = last += i; ----------//e6
--- for(int i : temp) sa[count[rank[i]]++] = i; ----------//d9
- } ----------//46
- void update_ranks(vi &rank, int k){ ----------//e4
--- vi old(rank); ----------//73
--- int r = rank[sa[0]] = 0; ----------//2c
--- for(int i = 1; i < rank.size(); i++) ----------//23
----- rank[sa[i]] = (old[sa[i]] == old[sa[i-1]] && (sa[i] + k < ol
----- old[sa[i]+k] : 0) <= (sa[i-1]+k < old.size() ? old[sa[i-1]+k
- } ----------//1d
- suffarray(string &s) : sa(s.size()), s(s){ ----------//dd
--- int n = s.size(); ----------//0c
--- for(int i = 0; i < n; i++) ----------//1c
----- sa[i] = i; ----------//ad
--- vi rank(s.size()); ----------//c8
--- for(int i = 0; i < s.size(); i++) ----------//44
----- rank[i] = s[i]; ----------//11
--- for(int k = 1; k <= n; k<<=1){ ----------//2b
----- radix_sort(rank, k); ----------//44
----- update_ranks(rank, k); ----------//fc
----- if(rank[sa.back()] == s.size()-1) break; ----------//e3
--- } ----------//e2
- } ----------//cf
}; ----------//df
// O(N) to compute LCP array ----------//ff
//ignore the $ sign? ----------//63
vi lcp_kasai(suffarray &x){ ----------//7d
- vi &sa = x.sa; ----------//95
- string &s = x.s; ----------//34
- int n = sa.size(); ----------//06
- vi rev(n), lcp(n); ----------//ad
- for(int i = 1; i < n; i++) ----------//87
--- rev[sa[i]] = i; ----------//f6
- for(int i = 1, k = 0; i < n; i++){ ----------//9d
--- if(rev[i] == n-1) continue; ----------//4d
```

```
--- int next = sa[rev[i]+1]; ----------------------------//8a
--- while(i+k < n && next+k < n && s[i+k] == s[next+k]) ---//6c
----- k++; ----------------------------------------------//dc
--- lcp[rev[i]] = k; ------------------------------------//ae
--- if(k) k--; ------------------------------------------//9e
- } ---------------------------------------------------//5c
- return lcp; ----------------------------------------//ee
} ---------------------------------------------------//e5
```

3.4. Polynomial Hashing.

```
pair<ll,ll> diofante(ll a, ll b){ -------------------------//e2
        if(!a)return {0,1}; ---------------------------------//
        auto t = diofante(b%a,a); ---------------------------//
        return {t.second-(b/a)*t.first,t.first}; ------------//
} ---------------------------------------------------------//75
ll modinv(ll x, ll m){ ------------------------------------//fe
        return (diofante(x,m).first%m+m)%m; -----------------//
} ---------------------------------------------------------//93
template<class T, ll a, ll mod> //use int sized stuff, does not handle overflow over 64 bit
struct prefix_hash{ ---------------------------------------//80
        int n; ----------------------------------------------//
        vector<ll> a_inv; -----------------------------------//
        vector<ll> psum; ------------------------------------//
        prefix_hash(T& arr, int n):n(n),a_inv(n),psum(n+1){ -//
                a_inv[0]=1; -----------------------------------//
                a_inv[1]=modinv(a,mod); -----------------------//
                for(int i=2; i<n; ++i){ -----------------------//
                        a_inv[i]=(a_inv[1]*a_inv[i-1])%mod; ---//
                } ---------------------------------------------//
                psum[n]=0; ------------------------------------//
                psum[n-1]=arr[n-1]; ---------------------------//
                ll powa=a; ------------------------------------//
                for(int i=n-2; i>=0; --i){ --------------------//
                        psum[i]=(psum[i+1]+powa*arr[i]%mod)%mod; //
                        powa=(powa*a)%mod; ----------------------//
                } ---------------------------------------------//
        } ---------------------------------------------------//
        ll operator()(int l, int r){ // returns hash of range [l,r]
                return (((psum[l]-psum[r+1]+mod)%mod)*a_inv[n-r-1])%mod;
        } ---------------------------------------------------//
}; ----------------------------------------------------------//2e
 ----------------------------------------------------------//9a
```

3.5. Manacher.

```
// O(N) algorithm to find longest  ----------------------//36
// palindromic substring with center --------------------//08
// in each index ----------------------------------------//c4
// first[i] = longest ODD size pal. in i ----------------//71
// second[i] = longest EVEN size pal. in i --------------//0e
pair<vi, vi> manacher (string s) { -------------------------//1f
- vi d1(s.size()), d2(s.size()); ---------------------------//9b
- int n = s.size(); ----------------------------------------//ca
- int l = 0, r = -1; ---------------------------------------//fa
- for (int i = 0; i < n; ++i) { ----------------------------//21
--- int k = (i > r ? 0 : min (d1[l + r - i], r - i)) + 1; -//0c
--- while (i + k < n && i - k >= 0 && s[i + k] == s[i - k]) ++k;
--- d1[i] = k--; ----------------------------------------//2d
--- if (i + k > r) --------------------------------------//75
----- l = i - k,  r = i + k; ----------------------------//09
- } ---------------------------------------------------//77
 ----------------------------------------------------------//c7
- l = 0, r = -1; -------------------------------------------//64
- for (int i = 0; i < n; ++i) { ----------------------------//d9
--- int k = (i > r ? 0 : min (d2[l + r - i + 1 ], r - i + 1)) + 1;
--- while (i + k - 1 < n && i - k >= 0 && s[i + k - 1] == s[i - k]) ++k;
--- d2[i] = --k; ----------------------------------------//16
--- if (i + k - 1 > r) ----------------------------------//aa
----- l = i - k,  r = i + k - 1; ------------------------//ec
- } ---------------------------------------------------//2d
- return {d1, d2}; -----------------------------------//9f
} ---------------------------------------------------//8b
```

3.6. Kmp.

```
// cc(|S| + |T|) string matchin with KMP ----------------//1b
// Computes occurences of S in T ------------------------//01
// Amortized O(|S|) prefix function --------------------//2c
vi pref_func(const string &s){ ----------------------------//65
- vi p(s.size()); -----------------------------------------//a4
- for(int i = 1, match = 0; i < s.size(); i++){ -----------//8e
--- while(match && s[i] != s[match]) -----------------------//62
----- match = p[match - 1]; ------------------------------//d9
--- match += s[i] == s[match]; -------------------------//96
--- p[i] = match; --------------------------------------//f9
- } ---------------------------------------------------//d3
- return p; ------------------------------------------//e6
} ---------------------------------------------------//e1
int count_matches(string &s, string &t){ ------------------//8f
- vi p = pref_func(s); -------------------------------------//dc
- int ans = 0, match = 0; ---------------------------------//16
- for(char c : t){ -----------------------------------------//cb
--- while(match && c != s[match]) ----------------------//19
----- match = p[match - 1]; ------------------------------//c2
--- match += c == s[match]; ----------------------------//ea
--- if(match == s.size()){ -----------------------------//e1
----- ans++; ---------------------------------------------//01
----- match = p.back();} ---------------------------------//0d
- } ---------------------------------------------------//47
- return ans; ----------------------------------------//f4
} ---------------------------------------------------//23
```

3.7. Aho Corasick2.

```
// Aho corasick for -------------------------------------//21
// "Count pattern occurencies" queries ------------------//ad
// O(N) time/memory complexity --------------------------//cd
struct node{ ----------------------------------------------//b2
- node *parent, *fall, *output; ---------------------------//71
- char c = '$'; -------------------------------------------//a6
- int ending = -1, taken = 0; -----------------------------//b8
- unordered_map<char, node*> f; ---------------------------//bf
- node(node *p, char c) : parent(p), c(c){} ---------------//d3
- node(){} ------------------------------------------------//4b
- vector<node*> rev_out; ----------------------------------//43
}; --------------------------------------------------------//db
struct aho_trie{ ------------------------------------------//fe
- node *root = new node(); --------------------------------//27
- int l = 0;  ----------------------------------------------//e6
- void insert(string s){ ----------------------------------//1c
--- node *curr = root; ---------------------------------------//cb
--- for(char c : s) ------------------------------------------//f8
----- curr = curr->f.count(c) ? curr->f[c] : (curr->f[c] = new node
----- curr->ending = l++; --------------------------------//1b
- } ---------------------------------------------------//6d
- void set_falls(){ ----------------------------------------//ac
--- queue<node*> q; ------------------------------------------//3c
--- q.push(root); --------------------------------------------//31
--- while(!q.empty()){ ---------------------------------------//8f
----- node *curr = q.front(); q.pop(); -------------------//52
----- for(auto &x : curr->f) -----------------------------//1d
------- q.push(x.second); ----------------------------------//d6
----- if(curr == root) continue; -------------------------//70
----- // compute suffix link ---------------------------//2d
----- curr->fall = curr->parent->fall; -------------------//05
----- while(curr->fall->f.count(curr->c) == 0 && curr->fall != roo
------- curr->fall = curr->fall->fall; ---------------------//a3
----- if(curr->fall != curr->parent && curr->fall->f.count(curr->c
------- curr->fall = curr->fall->f[curr->c]; ---------------//e0
----- curr->output = curr->fall->ending != -1 ? curr->fall : curr-
----- curr->output->rev_out.push_back(curr); -------------//20
--- } ---------------------------------------------------//07
- } ---------------------------------------------------//e1
- vi query(string s){ --------------------------------------//b5
--- node *stato = root; --------------------------------------//34
--- for(char c : s){ -----------------------------------------//6a
----- while(stato != root && stato->f.count(c) == 0) ------//ec
------- stato = stato->fall; -------------------------------//9b
----- if(stato->f.count(c)) ------------------------------//3e
------- stato = stato->f[c]; -------------------------------//52
----- stato->taken++; ------------------------------------//c6
--- } ---------------------------------------------------//1e
--- vi ans(l); -----------------------------------------------//de
--- function<int(node*)> dfs = [&](node *i){ -----------//38
----- int taken = 0; -------------------------------------//37
----- for(auto x : i->rev_out) ---------------------------//a5
------- taken += dfs(x); -----------------------------------//cd
----- taken += i->taken; ---------------------------------//54
----- if(i->ending != -1) ans[i->ending] = taken; ---------//73
----- i->taken = 0; --------------------------------------//49
----- return taken; --------------------------------------//c9
--- }; --------------------------------------------------//85
--- dfs(root); -----------------------------------------------//84
--- return ans; ----------------------------------------------//d9
- } ---------------------------------------------------//ec
- aho_trie(){ ----------------------------------------------//33
--- root->parent = root->fall = root->output = root; ------//23
- } ---------------------------------------------------//37
}; --------------------------------------------------------//2f
```

3.8. Aho Corasick.

```
// Aho corasick which allows ----------------------------//e5
// to get full automata (go) ----------------------------//f6
// O(N*alfa) memory/time complexity ---------------------//80
struct aho{ ------------------------------------------------//14
- const int root = 0; --------------------------------------//3b
- vvi go; --------------------------------------------------//31
- vi parent, ch, suff, super, end; -------------------------//dd
- int label = 0; -------------------------------------------//52
- int size = 1; --------------------------------------------//b5
- int new_node(char c, int p){ -----------------------------//2c
--- ch[size] = c; ---------------------------------------------//ab
--- parent[size] = p; -----------------------------------------//47
--- return size++; --------------------------------------------//df
- } ---------------------------------------------------------//ec
- aho(int total) : go(total + 5, vi(26, -1)), --------------//14
--- parent(total + 5, 0), ch(total + 5, -1), ----------------//d8
--- suff(total + 5, 0), super(total + 5, 0), ----------------//62
--- end(total + 5, -1) {} ------------------------------------//cb
 ----------------------------------------------------------//44
- void insert(string &s){ ----------------------------------//18
--- int i = root; ---------------------------------------------//33
--- for(char c : s){ ------------------------------------------//88
----- c -= 'A'; -------------------------------------------------//18
----- if(go[i][c] == -1) ----------------------------------------//e9
------- go[i][c] = new_node(c, i); ------------------------------//11
----- i = go[i][c]; ---------------------------------------------//d2
--- } ---------------------------------------------------------//f5
--- end[i] = label++; -----------------------------------------//a8
- } ---------------------------------------------------------//6d
- void build(){ --------------------------------------------//23
--- queue<int> q; q.push(root); -------------------------------//47
--- while(!q.empty()){ ----------------------------------------//cd
----- int i = q.front(); q.pop(); -------------------------------//a2
----- if(i != root){ --------------------------------------------//bf
------- if(parent[i] == root) -------------------------------//36
--------- suff[i] = root; -----------------------------------//c3
------- else ------------------------------------------------//98
--------- suff[i] = go[suff[parent[i]]][ch[i]]; -------------//b0
----- } ---------------------------------------------------//fb
----- for(int c = 0; c < 26; c++){ ------------------------//84
------- if(go[i][c] == -1){ ---------------------------------//a6
--------- go[i][c] = i == root ? root : go[suff[i]][c]; ---//5d
------- }else q.push(go[i][c]); -----------------------------//fc
----- } ---------------------------------------------------//38
----- if(i != root) { -------------------------------------//50
------- super[i] = end[suff[i]] != -1 ? suff[i] : super[suff[i]];
----- } ---------------------------------------------------//61
--- } ---------------------------------------------------//1d
- } ---------------------------------------------------//a9
}; ---------------------------------------------------//d3
```

4. Maths

4.1. Simplex.

```
// Two-phase simplex algorithm for solving linear programs of the form
// ----------------------------------------------------//f0
//     maximize     c^T x ------------------------------//e0
//     subject to   Ax <= b ---------------------------//46
//                  x >= 0 ----------------------------//31
// ----------------------------------------------------//c5
// INPUT: A -- an m x n matrix ------------------------//b7
//        b -- an m-dimensional vector ------------------//13
//        c -- an n-dimensional vector ------------------//48
//        x -- a vector where the optimal solution will be stored
// ----------------------------------------------------//7f
// OUTPUT: value of the optimal solution (infinity if unbounded
//         above, nan if infeasible) --------------------//4d
// ----------------------------------------------------//cc
// To use this code, create an LPSolver object with A, b, and c as
// arguments.  Then, call Solve(x). -------------------//9e
 ----------------------------------------------------//b8
#include <iostream> ------------------------------------//4c
#include <iomanip> -------------------------------------//20
#include <vector> --------------------------------------//86
#include <cmath> ---------------------------------------//05
#include <limits> --------------------------------------//d1
 ----------------------------------------------------//06
using namespace std; ---------------------------------//d3
 ----------------------------------------------------//fb
typedef long double DOUBLE; --------------------------//27
typedef vector<DOUBLE> VD; ---------------------------//8f
typedef vector<VD> VVD; ------------------------------//a5
typedef vector<int> VI; ------------------------------//83
 ----------------------------------------------------//d1
const DOUBLE EPS = 1e-9; -----------------------------//51
 ----------------------------------------------------//d2
struct LPSolver { ------------------------------------//f7
- int m, n; ------------------------------------------//51
- VI B, N; -------------------------------------------//34
- VVD D; ---------------------------------------------//06
 ----------------------------------------------------//7e
- LPSolver(const VVD &A, const VD &b, const VD &c) : ------//6f
--- m(b.size()), n(c.size()), N(n + 1), B(m), D(m + 2, VD(n + 2))
--- for (int i = 0; i < m; i++) for (int j = 0; j < n; j++)//05
----- D[i][j] = A[i][j]; ----------------------------------//80
--- for (int i = 0; i < m; i++) { ----------------------//d2
----- B[i] = n + i; D[i][n] = -1; D[i][n + 1] = b[i]; } ---//54
--- for (int j = 0; j < n; j++) { N[j] = j; D[m][j] = -c[j]; }
--- N[n] = -1; D[m + 1][n] = 1; ------------------------//f9
- } ---------------------------------------------------//10
 ----------------------------------------------------//a0
- void Pivot(int r, int s) { ---------------------------//91
--- double inv = 1.0 / D[r][s]; ------------------------//9b
--- for (int i = 0; i < m + 2; i++) if (i != r) --------//bb
----- for (int j = 0; j < n + 2; j++) if (j != s) ------//91
------- D[i][j] -= D[r][j] * D[i][s] * inv; ------------//fe
--- for (int j = 0; j < n + 2; j++) if (j != s) D[r][j] *= inv;
--- for (int i = 0; i < m + 2; i++) if (i != r) D[i][s] *= -inv;
--- D[r][s] = inv; -------------------------------------//b4
--- swap(B[r], N[s]); ----------------------------------//ec
- } ---------------------------------------------------//af
 ----------------------------------------------------//a3
- bool Simplex(int phase) { ----------------------------//ef
--- int x = phase == 1 ? m + 1 : m; ----------------------//7c
--- while (true) { ---------------------------------------//8c
----- int s = -1; -----------------------------------------//ba
----- for (int j = 0; j <= n; j++) { ----------------------//d8
------- if (phase == 2 && N[j] == -1) continue; ----------//46
------- if (s == -1 || D[x][j] < D[x][s] || D[x][j] == D[x][s] &&
--------- s = j; ---------------------------------------//17
----- } ---------------------------------------------------//57
----- if (D[x][s] > -EPS) return true; --------------------//b4
----- int r = -1; -----------------------------------------//5d
----- for (int i = 0; i < m; i++) { ----------------------//f7
------- if (D[i][s] < EPS) continue; ----------------------//46
------- if (r == -1 || D[i][n + 1] / D[i][s] < D[r][n + 1] / D[r]
--------- (D[i][n + 1] / D[i][s]) == (D[r][n + 1] / D[r][s]) && B
--------- r = i; ---------------------------------------//c0
----- } ---------------------------------------------------//ca
----- if (r == -1) return false; --------------------------//a6
----- Pivot(r, s); ----------------------------------------//51
--- } ---------------------------------------------------//b3
- } ---------------------------------------------------//a5
 ----------------------------------------------------//27
- DOUBLE Solve(VD &x) { -------------------------------//e9
--- int r = 0; ------------------------------------------//e7
--- for (int i = 1; i < m; i++) if (D[i][n + 1] < D[r][n + 1]) r =
--- if (D[r][n + 1] < -EPS) { ---------------------------//f9
----- Pivot(r, n); ----------------------------------------//08
----- if (!Simplex(1) || D[m + 1][n + 1] < -EPS) ----------//1c
------- return -numeric_limits<DOUBLE>::infinity(); -------//43
----- for (int i = 0; i < m; i++) if (B[i] == -1) { -------//04
------- int s = -1; ---------------------------------------//f6
------- for (int j = 0; j <= n; j++) ----------------------//17
--------- if (s == -1 || D[i][j] < D[i][s] || D[i][j] == D[i][s] &
----------- s = j; -------------------------------------//4b
------- Pivot(i, s); ----------------------------------//e9
----- } ---------------------------------------------------//64
--- } ---------------------------------------------------//76
--- if (!Simplex(2)) return numeric_limits<DOUBLE>::infinity();
--- x = VD(n); ------------------------------------------//c7
--- for (int i = 0; i < m; i++) if (B[i] < n) x[B[i]] = D[i][n + 1
--- return D[m][n + 1]; ---------------------------------//a1
- } ---------------------------------------------------//5d
}; ---------------------------------------------------//4c
 ----------------------------------------------------//3e
int main() { -----------------------------------------//d2
 ----------------------------------------------------//4f
- const int m = 4; -----------------------------------//24
- const int n = 3; -----------------------------------//66
- DOUBLE _A[m][n] = { ---------------------------------//7a
--- { 6, -1, 0 }, ---------------------------------------//31
--- { -1, -5, 0 }, --------------------------------------//f0
--- { 1, 5, 1 }, ----------------------------------------//cb
--- { -1, -5, -1 } --------------------------------------//23
- }; --------------------------------------------------//01
- DOUBLE _b[m] = { 10, -4, 5, -5 }; --------------------//e3
- DOUBLE _c[n] = { 1, -1, 0 }; -------------------------//24
 ----------------------------------------------------//01
```

```
- VVD A(m); ----------//10
- VD b(_b, _b + m); ----------//de
- VD c(_c, _c + n); ----------//3c
- for (int i = 0; i < m; i++) A[i] = VD(_A[i], _A[i] + n); //c7
 ----------//bf
- LPSolver solver(A, b, c); ----------//e2
- VD x; ----------//d8
- DOUBLE value = solver.Solve(x); ----------//1d
 ----------//09
- cerr << "VALUE: " << value << endl; // VALUE: 1.29032 ---//e3
- cerr << "SOLUTION:"; // SOLUTION: 1.74194 0.451613 1 ----//ae
- for (size_t i = 0; i < x.size(); i++) cerr << " " << x[i];
- cerr << endl; ----------//b0
- return 0; ----------//5b
} ----------//a1
```

4.2. Sieve 2.

```
// Linear sieve variant:\ ----------//75
- also computes euler totient (coprimes <= i),\ ----------//ef
- number of distinct divisors,\ ----------//7d
- sum of divisors,\ ----------//91
- for all i in [1, N] ----------//df
 ----------//41
struct sieve{ ----------//7c
- vi d, s0, fi; // smallest div, number of divs, euler fi -//bc
- vector<ll> s1; // sum of divisors ----------//18
- sieve(int n) : d(n+1), s0(n+1), s1(n+1), fi(n+1){ -------//1d
--- // just standard linear sieve ----------//f5
--- vi primes; ----------//4b
--- fi[1] = 1; ----------//e7
--- d[1] = 0; iota(d.begin() + 2, d.end(), 2); ----------//81
--- s0[1] = 1; ----------//17
--- s1[1] = 1; ----------//81
--- for(int k = 2; k <= n; k++){ ----------//69
----- if(d[k] == k) primes.push_back(k); ----------//98
----- for(int p : primes){ ----------//41
------- if(p > d[k]) break; ----------//25
------- if(p * (ll)k <= n) d[p * k] = p; ----------//68
------- else break; ----------//d1
----- } ----------//22
----- // smallest div ----------//ac
----- ll p1 = d[k]; ----------//85
----- // fi function ----------//b8
----- if(d[k/p1] != p1) fi[k] = fi[k/p1] * (p1-1); --------//ac
----- else fi[k] = fi[k/p1] * p1; ----------//7e
----- // distinct divisors ----------//4c
----- int times = 0; ----------//f6
----- for(int i = k / p1; d[i] == p1; i /= p1, times++); --//01
----- s0[k] = s0[k / p1] / (times+1) * (times+2); ---------//65
----- // sum of divisors ----------//e3
----- ll sum_prev = s1[k / bin_exp(p1, times + 1)]; -------//85
----- s1[k] = s1[k/p1] + sum_prev * bin_exp(p1, times+1); -//b0
--- } ----------//da
- } ----------//0d
}; ----------//4f
```

4.3. Sieve.

```
// Linear sieve: returns the smallest divisor ----------//dc
// for all numbers in [0, N] ----------//fa
vi divs(int n){ ----------//6f
- vi ans(n+1), primes; ----------//73
- iota(ans.begin(), ans.end(), 0); ----------//9f
- for(int k = 2; k <= n; k++){ ----------//cc
--- if(ans[k] == k) ----------//60
----- primes.push_back(k); ----------//48
--- for(int p : primes){ ----------//ea
----- if(p > ans[k]) break; ----------//46
----- if(p * k <= n) ans[p * k] = p; ----------//f6
----- else break; ----------//f3
--- } ----------//f5
- } ----------//ad
- return move(ans); ----------//0a
} ----------//cf
```

4.4. Segmented Sieve.

```
/* Returns primality for each number ----------//c0
- in range [l, r], given a sieve ----------//46
- up to sqrt(r), shifted by l ----------//aa
- O(max(sqrt(r), r - l) * log(r-l)) */ ----------//a7
 ----------//e1
vb seg_sieve(vi &sieve, int l, int r){ ----------//13
- vb prime(r - l + 1); ----------//b7
- for(int i = 2;i<sieve.size();i++){ ----------//95
--- if(sieve[i] != i) continue; ----------//d1
--- for(int j = i*(l/i); j<=r; j+=i){ ----------//0b
----- if(j < l) continue; ----------//6d
----- int tmp = j; ----------//69
----- while(tmp && tmp % i == 0){ ----------//21
------- tmp /= i; ----------//64
------- prime[j - l] = 1; ----------//71
----- } ----------//11
--- } ----------//ee
- } ----------//25
- return prime; ----------//1d
} ----------//3a
```

4.5. Rabin Miller.

```
using u64 = uint64_t; ----------//fe
using u128 = __uint128_t; ----------//27
 ----------//6f
u64 binpower(u64 base, u64 e, u64 mod) { ----------//f2
--- u64 result = 1; ----------//f3
--- base %= mod; ----------//0c
--- while (e) { ----------//64
------- if (e & 1) ----------//1d
----------- result = (u128)result * base % mod; ----------//2c
------- base = (u128)base * base % mod; ----------//23
------- e >>= 1; ----------//41
--- } ----------//f9
--- return result; ----------//57
} ----------//78
 ----------//bc
bool check_composite(u64 n, u64 a, u64 d, int s) { --------//b0
--- u64 x = binpower(a, d, n); ----------//24
--- if (x == 1 || x == n - 1) ----------//9f
------- return 0; ----------//fd
--- for (int r = 1; r < s; r++) { ----------//6e
------- x = (u128)x * x % n; ----------//ad
------- if (x == n - 1) ----------//41
----------- return 0; ----------//36
--- } ----------//40
--- return 1; ----------//64
}; ----------//bf
 ----------//bf
bool MillerRabin(u64 n) { ----------//23
--- if (n < 2) ----------//63
------- return 0; ----------//87
--- int r = 0; ----------//78
--- u64 d = n - 1; ----------//8e
--- while ((d & 1) == 0) { ----------//2c
------- d >>= 1; ----------//b9
------- r++; ----------//53
--- } ----------//8e
--- for(int a:{2, 3, 5, 13, 19, 73, 193, 407521, 299210837})if(n==
--- for (int a : {2, 325, 9375, 28178, 450775, 9780504, 1795265022
------- if (check_composite(n, a, d, r)) ----------//5b
----------- return 0; ----------//66
--- return 1; ----------//a9
} ----------//b8
 ----------//34
```

4.6. Fft.

```
// FFT, it multiplies two polynomials in O(NlogN) --------//23
// Extremely powerful, but I've no idea how it works ------//2f
// Call mul and give the polynomials as ----------//9d
// arrays of their coefficients :) ----------//17
 ----------//06
namespace FFT { ----------//16
- struct complex { ----------//ea
--- double a, b; ----------//6a
--- complex() { a = b = 0; } ----------//bd
--- complex(double _a, double _b) { a = _a; b = _b; } -----//86
--- complex(double _a) { a = _a; b = 0; } ----------//9c
--- complex operator + (complex o) { return complex(a + o.a, b + o
--- complex operator - (complex o) { return complex(a - o.a, b - o
--- complex operator * (complex o) { ----------//04
----- return complex(a * o.a - b * o.b,  a * o.b + b * o.a); }
--- void operator *= (complex o) {  ----------//1a
----- double na = a * o.a - b * o.b; ----------//e9
----- double nb = a * o.b + b * o.a; ----------//01
----- a = na, b = nb; ----------//86
--- } ----------//3e
--- void operator /= (double x) { a /= x, b /= x;} --------//1f
--- double real() { return a; } ----------//e0
--- double imag() { return b; } ----------//08
- }; ----------//53
- const int M = 21; ----------//36
- const int N = (1 << M); ----------//1d
- int pre[N][M]; ----------//69
- int reverse(int num, int lg_n) { ----------//1a
```

```
    if(pre[num][lg_n]) return pre[num][lg_n] - 1; //c6
    int res = 0; //b1
    for (int i = 0; i < lg_n; i++) { //5d
      if (num & (1 << i)) //ef
        res |= 1 << (lg_n - 1 - i); //95
    } //fc
    pre[num][lg_n] = res + 1; //9e
    return res; //8b
}; //a5
void fft(vector<complex> & a, bool invert) { //57
    int n = a.size(); //39
    int lg_n = 0; //fe
    while ((1 << lg_n) < n) lg_n++; //bd
    for (int i = 0; i < n; i++) { //7b
      if (i < reverse(i, lg_n)) swap(a[i], a[reverse(i, lg_n)]); //3d
    } //0e
    for (int len = 2; len <= n; len <<= 1) { //82
      double ang = 2 * PI / len * (invert ? -1 : 1); //6b
      complex wlen(cos(ang), sin(ang)); //67
      for (int i = 0; i < n; i += len) { //47
        complex w(1); //23
        for (int j = 0; j < len / 2; j++) { //1f
          complex u = a[i+j], v = a[i+j+len/2] * w; //36
          a[i+j] = u + v; //a7
          a[i+j+len/2] = u - v; //3b
          w *= wlen; //44
        } //39
      } //83
    } //5c
    if (invert) { //32
      for (complex & x : a) x /= n; //9c
    } //9f
}; //ba
inline vi mul(vi const& a, vi const& b) { //74
    vector<complex> fa(a.begin(), a.end()), fb(b.begin(), b.end()); //74
    int n = 1; //aa
    while (n < a.size() + b.size()) n <<= 1; //dc
    fa.resize(n); //2a
    fb.resize(n); //bd
    fft(fa, false); //e2
    fft(fb, false); //54
    for (int i = 0; i < n; i++) fa[i] *= fb[i]; //6e
    fft(fa, true); //b2
    vi result(n); //6b
    for (int i = 0; i < n; i++) { //6e
      ll x = fa[i].real() + 0.5; //cc
      result[i] = x; //6a
    } //91
    while(result.size() && !result[result.size() - 1]) result.pop_back(); //64
    return result; //35
}; //05
inline vi moduloMul(vi const& a, vi const& b, ll MOD) { //58
    vector<complex> fa(a.begin(), a.end()), fb(b.begin(), b.end()); //63
    int n = 1; //5b
    while (n < a.size() + b.size()) n <<= 1; //89
    fa.resize(n); //90
    fb.resize(n); //bb
    fft(fa, false); fft(fb, false); //67
    for (int i = 0; i < n; i++) fa[i] *= fb[i]; //96
    fft(fa, true); //38
    vi result(n); //fd
    for (int i = 0; i < n; i++) { //52
      ll x = fa[i].real() + 0.5; //70
      if(x >= MOD) x %= MOD; //be
      result[i] = x; //
    } //
    while(result.size() && !result[result.size() - 1]) result.pop_back(); //
    return result; //
}; //
```

4.7. Extended Gcd.

```
// Extended GCD //70
 //78
/* Solves diophantine equation //89
-- a*x + b*y = gcd(a, b) //7d
-- and return {x, y, gcd(a, b)} //da
-- if ll, might require __i128 */ //e3
vector<ll> exgcd (ll a, ll b){ //85
        if(!b) return {1, 0, a};
        auto x = exgcd(b, a % b);
        return {x[1], x[0] - x[1] * (a/b), x[2]};
} //33
```

4.8. Crt.

```
// Chinese remainder theorem //e0
 //d2
/* Call gcrt(a, b) to find smallest {x, y} such //77
- that (x + k * y) % a[i] = b[i] for all i, k */ //79
 //38
#define rep(i,a,b) for (__typeof(a) i=(a); i<(b); ++i) //39
 //8f
#define iter(it,c) for (__typeof((c).begin()) \ //71
- it = (c).begin(); it != (c).end(); ++it) //36
 //a0
ll egcd(ll a, ll b, ll& x, ll& y) { //e8
- if (b == 0) { x = 1; y = 0; return a; } //43
- ll d = egcd(b, a % b, x, y); //0c
- x -= a / b * y; swap(x, y); return d; } //44
 //2e
template <class T> int size(const T &x) { return x.size(); }
template <class T> T smod(T a, T b) { return (a % b + b) % b; }
 //ec
ll crt(vector<ll> &as, vector<ll> &ns) { //14
- ll cnt = size(as), N = 1, x = 0, r, s, l; //33
- rep(i,0,cnt) N *= ns[i]; //94
- rep(i,0,cnt) egcd(ns[i], l = N/ns[i], r, s), x += as[i]*s*l;
- return smod(x, N); } //26
 //b5
pair<ll,ll> gcrt(vector<ll> &as, vector<ll> &ns) { //db
- map<ll,pair<ll,ll> > ms; //de
- rep(at,0,size(as)) { //5e
--- ll n = ns[at]; //ef
--- for (ll i = 2; i*i <= n; i = i == 2 ? 3 : i + 2) { //3a
----- ll cur = 1; //98
----- while (n % i == 0) n /= i, cur *= i; //df
----- if (cur > 1 && cur > ms[i].first) //55
------- ms[i] = make_pair(cur, as[at] % cur); } //af
--- if (n > 1 && n > ms[n].first) //73
----- ms[n] = make_pair(n, as[at] % n); } //43
- vector<ll> as2, ns2; ll n = 1; //22
- iter(it,ms) { //03
--- as2.push_back(it->second.second); //71
--- ns2.push_back(it->second.first); //2a
--- n *= it->second.first; } //eb
- ll x = crt(as2,ns2); //a5
- rep(i,0,size(as)) if (smod(x,ns[i]) != smod(as[i],ns[i])) //a9
----- return pii(0,0); //b3
- return make_pair(x,n); } //c1
```

4.9. Bin Exp.

```
// Binary exponentiation //d8
// in case needed, add % mod whenever //6d
// you see some computations going on //34
// can be modified into bin_mul easily, //a9
// just replace * with + //b7
 //fe
ll bin_exp(ll b, ll p){ //50
- if(p == 0) return 1; //2b
- if(p == 1) return b; //f9
- ll ans = 1; //02
- if(p % 2 == 1){ //1f
--- ans *= b; //24
--- p--; //d8
- } //f8
- ll x = bin_exp(b, p/2); //b0
- return ans * x * x; //6d
} //49
```

4.10. Big Integers.

```
// Library from some japanese coder //87
// Supposedly a good implementation, but doesn't //e9
// use FFT for multiplications, so not giga fast //c5
 //d8
 //ba
const ll base=100000000; //c5
struct bignum{ //d1
- int len; //dc
- ll data[bignumlen]; //23
- ll &operator [](int x){ return(data[x]);} //6a
- const ll &operator [](int x)const { return(data[x]);} //cc
- bignum (){ //25
--- memset(data,0,sizeof(data)); //01
--- len=0; //0b
- } //a5
- void clear(){ //f4
--- for(int i=len;i>=1;--i)data[i]=0; //ad
--- len=0; //fd
- } //ba
- int check (const bignum &a,const bignum &b){ //70
--- if(a.len>b.len)return(0); //56
```

```
    if(b.len>a.len)return(1); //f3
    for(int i=a.len;i>=1;--i){ //b1
      if(a.data[i]<b.data[i])return(1); //39
      if(b.data[i]<a.data[i])return(0); //e6
    } //fb
    return 2; //5a
- } //b9
- bool operator < (const bignum &b){ return(check(*this,b)==1);}
- bool operator > (const bignum &b){ return(check(*this,b)==0);}
- bool operator <=(const bignum &b){ return(check(*this,b)>=1);}
- bool operator >=(const bignum &b){ return(check(*this,b)%2==0);}
- bool operator !=(const bignum &b){ return(check(*this,b)!=2);}
- bool operator ==(const bignum &b){ return(check(*this,b)==2);}
-  //3b
- bignum operator=(const bignum &x){ //4a
    for(int i=x.len+1;i<=len;++i)data[i]=0; //60
    for(int i=1;i<=x.len;++i)data[i]=x.data[i]; //07
    len=x.len; //3c
    return *this; //03
- } //fe
- bignum operator=(ll x){ //a9
    for(int i=len;i>=0;--i)data[i]=0; //b4
    len=0; //64
    while(x){ //13
      data[++len]=x%base; //40
      x/=base; //7c
    } //ab
    return *this; //c6
- } //4e
- bignum(ll x){ //b1
    memset(data,0,sizeof(data)); //c0
    len=0; //1d
    (*this)=x; //93
- } //5c
- bignum operator *(const bignum &b){ //ac
    int i,j; //c1
    bignum tmp; //69
    for(i=1;i<=len;++i)if(data[i]!=0) //48
      for(j=1;j<=b.len;++j)if(b.data[j]!=0){ //28
        tmp.data[i+j-1]+=data[i]*b.data[j]; //8f
        tmp.data[i+j]+=tmp.data[i+j-1]/base; //99
        tmp.data[i+j-1]%=base; //74
      } //22
    tmp.len=len+b.len-1; //a1
    while(tmp.data[tmp.len+1])tmp.len++; //79
    return tmp; //2a
- } //b8
- bignum operator *(ll x){ //e5
    int i; //03
    bignum tmp; //9b
    for(i=1;i<=len;++i)tmp[i]=data[i]*x; //e7
    tmp.len=len; //78
    for(i=1;i<=len;++i){ //74
      tmp[i+1]+=tmp[i]/base,tmp[i]%=base; //06
      if(tmp[i+1]&&i+1>tmp.len)tmp.len++; //f6
    } //56
    return tmp; //fa
- } //ae
- bignum operator /(ll x){ //f9
    int i; //72
    bignum tmp; //ac
    ll y=0; //d3
    for(i=len;i>=1;--i){ //7e
      y=y*base+data[i]; //80
      tmp[i]=y/x; //0c
      y%=x; //99
    } //81
    tmp.len=len; //c3
    while(tmp[tmp.len]==0&&tmp.len>1)tmp.len--; //94
    return tmp; //fa
- } //49
- bignum operator /(const bignum &b){ //b6
    if(b.len<=1 && b[1]==0){ //48
      printf("error!"); //58
      for(;;); //91
    } //5c
    int i,l1=(len-1)*Blen,l2=(b.len-1)*Blen; //5d
    ll x=data[len],y=b[b.len]; //06
    while(x)x/=10,l1++; //17
    while(y)y/=10,l2++; //3b
    bignum tmp,chu,B; //aa
    chu=*this; B=b; //30
    //7a
    for(i=1;i*Blen<=l1-l2;++i)B*=base; //33
    for(i=1;i<=(l1-l2)%Blen;++i)B*=10; //68
    for(i=l1-l2;i>=0;--i){ //85
      x=0; //ce
      while(chu>=B)chu-=B,x++; //6d
      tmp[i/Blen+1]=tmp[i/Blen+1]*10+x; //c0
      B/=10; //f4
    } //8c
    tmp.len=(l1-l2)/Blen+1; //33
    while(tmp.len>=1 && !tmp[tmp.len])tmp.len--; //32
    return tmp; //22
- } //52
- bignum operator +(const bignum &b){ //27
    bignum tmp; //50
    int i,l=max(len,b.len); //c7
    for(i=1;i<=l;++i)tmp[i]=data[i]+b[i]; //33
    for(i=1;i<=l;++i)tmp[i+1]+=tmp[i]/base,tmp[i]%=base; //00
    tmp.len=l; //4a
    if(tmp[tmp.len+1])tmp.len++; //3b
    return tmp; //a4
- } //98
- bignum operator +(ll x){ //27
    bignum tmp; tmp=*this; //0a
    tmp[1]+=x; //8c
    for(int i=1;i<=len&&tmp[i]>=base;++i) //2a
      tmp[i+1]+=tmp[i]/base, tmp[i]%=base; //3a
    while(tmp[tmp.len+1])tmp.len++; //2d
    return tmp; //83
- } //10
- bignum operator -(const bignum &b){ //db
    int i; //e0
    bignum tmp; //57
    for(i=1;i<=len;++i)tmp.data[i]=data[i]-b.data[i]; //cd
    for(i=1;i<=len;++i){ //3c
      if(tmp[i]<0)tmp.data[i]+=base,tmp.data[i+1]--; //58
    } //3b
    tmp.len=len; //73
    while(tmp[tmp.len]==0&&tmp.len>1)tmp.len--; //4c
    return tmp; //83
- } //25
- bignum operator -(ll x){ //72
    bignum tmp; tmp=*this; //bd
    tmp[1]-=x; //f3
    for(int i=1;i<=len&&tmp[i]<0;++i){ //73
      tmp[i+1]+=(tmp[i]+1)/base-1; //71
      tmp[i]=(tmp[i]+1)%base+base-1; //22
    } //31
    while(!tmp[tmp.len]&&tmp.len>1)tmp.len--; //b2
    return tmp; //34
- } //55
- ll operator %(ll x){ //9e
    int i; //d3
    ll y=0; //49
    for(i=len;i>=1;--i)y=(y*base+data[i])%x; //1a
    return y; //87
- } //6b
- bignum operator %(const bignum &b){ //57
    if(b.len<=1 && b[1]==0){ //ea
      printf("error! 0 mod!"); //b0
      for(;;); //3d
    } //4f
    int i,l1=(len-1)*Blen,l2=(b.len-1)*Blen; //20
    ll x=data[len],y=b[b.len]; //18
    while(x)x/=10,l1++; //c0
    while(y)y/=10,l2++; //f2
    bignum chu,B; //c1
    chu=*this; B=b; //52
    //c7
    for(i=1;i*Blen<=l1-l2;++i)B*=base; //cf
    for(i=1;i<=(l1-l2)%Blen;++i)B*=10; //9c
    for(i=l1-l2;i>=0;--i){ //ab
      while(chu>=B)chu-=B; //48
      B/=10; //44
    } //b2
    return chu; //d3
- } //1d
-  //0f
- bignum operator +=(const bignum &b){return *this=(*this+b);}
- bignum operator *=(const bignum &b){return *this=(*this*b);}
- bignum operator -=(const bignum &b){return *this=(*this -b);}
- bignum operator /=(const bignum &b){return *this=(*this/b);}
- bignum operator %=(const bignum &b){return *this=(*this%b);}
- bignum operator *=(ll x) {return( *this=(*this *x));} //25
- bignum operator +=(ll x) {return( *this=(*this +x));} //fa
- bignum operator -=(ll x) {return( *this=(*this -x));} //ad
```

```
- bignum operator /=(ll x) {return( *this=(*this /x));} ---//6d
- void read(){ ----------------------------------------------//df
--- char c[bignumlen*Blen+10]; ---------------------------------//48
--- scanf("%s",c+1); -------------------------------------------//01
--- int l=strlen(c+1); -----------------------------------------//bb
--- (*this).clear(); -------------------------------------------//7c
--- ll x; ------------------------------------------------------//a0
--- for(int i=1;i<=(l-1)/Blen+1;++i){ --------------------------//73
----- x=0; -------------------------------------------------------//41
----- for(int j=l-Blen*i+1;j<=l-Blen*i+Blen;++j) ----------------//3a
------- if(j>=1)x=x*10+c[j]-48; -------------------------------//79
----- data[++len]=x; ---------------------------------------------//39
--- } ----------------------------------------------------------//25
- } ----------------------------------------------------------//43
- void write(){ ---------------------------------------------//94
--- printf("%I64d",data[len]); -----------------------------//25
--- for(int i=len-1;i>=1;--i)printf("%0*I64d",Blen,data[i]);
- } ----------------------------------------------------------//8c
}p,q,pp,qq; ------------------------------------------------//c6
bignum gcd(const bignum &A,const bignum &B){ ---------------//14
- bignum a=A,b=B,res=1; -------------------------------------//1c
- while(!(a[1]&1)  && !(b[1]&1))a/=2,b/=2,res*=2; ---------//8b
- for(;;){ ---------------------------------------------------//83
--- if(a.len==1 && a[1]==0)return b*res; ------------------//e0
--- if(b.len==1 && b[1]==0)return a*res; ------------------//bb
--- while(!(a[1]&1))a/=2; ---------------------------------//7e
--- while(!(b[1]&1))b/=2; ---------------------------------//74
--- if(a>b)a-=b; ------------------------------------------//4a
--- else b-=a; --------------------------------------------//ba
- } ----------------------------------------------------------//24
} ----------------------------------------------------------//43
```

5. Graphs

5.1. Topological Sort.

```
// O(N) Topological sort for DAG -------------------------//3e
// Returns array v, indexes in topo-sorted order ---------//0d
// (for i < j, v[i] it NOT reachable by v[j]) ------------//58
// Can be used (often reversed) instead of DP -------------//90
// for DAG problems --------------------------------------//c3
 ------------------------------------------------------------//09
vi toposort(vvi &g){ ---------------------------------------//2f
- const int n = g.size(); ----------------------------------//b2
- vector<bool> vis(n); --------------------------------------//c6
- vi topo; topo.reserve(n); ---------------------------------//15
- function<void(int)> dfs = [](int i){ ----------------------//b4
--- vis[i] = 1; ------------------------------------------//80
--- for(int e : g[i]) if(!vis[e]) dfs(e); ----------------//95
----- topo.push_back(i); -----------------------------------//a5
- }; --------------------------------------------------------//2d
- for(int i = 0; i < n; i++) ---------------------------------//bd
--- if(!vis[i]) ------------------------------------------//9c
----- dfs(i); ----------------------------------------------//81
- reverse(topo.begin(), topo.end()); ------------------------//fc
- return topo; ----------------------------------------------//27
} ----------------------------------------------------------//49
```

5.2. Min Cost Max Flow.

```
// O(V^3M) (estimated) algorithm for MCMF ----------------//3d
// Uses an implementation of Pape-Levit ------------------//8b
// for shortest path (better in practice, ----------------//d2
// but not as good as Dijkstra asymptotically) ------------//23
// Hope it's good enough :P ------------------------------//48
 ------------------------------------------------------------//72
struct edge { ----------------------------------------------//ea
- int b, u; ------------------------------------------------//7d
- ll c, f; -------------------------------------------------//8f
- size_t back; ---------------------------------------------//3a
}; ---------------------------------------------------------//1d
 ------------------------------------------------------------//06
using graph = vector<vector<edge>>; ----------------------//b6
 ------------------------------------------------------------//25
// a to b, capacity u, cost c ----------------------------//80
void add_edge (graph &g, int a, int b, ll u, ll c) { ------//4f
- edge r1 = {b, u, c, 0, g[b].size ()}; --------------------//d0
- edge r2 = {a, 0, -c, 0, g[a].size ()}; -------------------//ba
- g[a].push_back (r1); -------------------------------------//ab
- g[b].push_back (r2); -------------------------------------//79
} ----------------------------------------------------------//f5
 ------------------------------------------------------------//4e
// graph, source, sink -----------------------------------//c9
// returns {flow, cost} ----------------------------------//2f
pair<ll, ll> mcmf(graph &g, int s, int t){ ----------------//32
- ll flow = 0, cost = 0; -----------------------------------//8f
- ll k = MAX_FLOW; // use Dinic :) -------------------------//46
- while(flow < k){ -----------------------------------------//2c
--- vi id(n), d(n, INF), q(n), p(n); ---------------------//f0
--- vector<size_t> p_edge(n); ----------------------------//83
--- int qh = 0, qt = 0; ----------------------------------//fd
--- q[qt++] = s; -----------------------------------------//2a
--- d[s] = 0; --------------------------------------------//1b
--- while (qh != qt) { -----------------------------------//11
----- int v = q[qh++]; -------------------------------------//ab
----- id[v] = 2; -------------------------------------------//af
----- if (qh == n) qh = 0; ---------------------------------//c2
----- for (size_t i = 0; i < g[v].size(); ++i) { ----------//18
------- edge & r = g[v][i]; ----------------------------//80
------- if (rf <ru && d[v] + rc < d[rb]) { --------------//a5
--------- d[rb] = d[v] + rc; ---------------------------//90
--------- if (id[rb] == 0) { ---------------------------//bc
----------- q[qt++] = rb; ------------------------------//0d
----------- if (qt == n) qt = 0; -----------------------//1a
--------- } ---------------------------------------------//b9
--------- else if (id[rb] == 2) { ----------------------//78
----------- if (--qh == -1) qh = n-1; ------------------//56
----------- q[qh] = rb; --------------------------------//e1
--------- } ---------------------------------------------//07
--------- id[rb] = 1; ----------------------------------//45
--------- p[rb] = v; -----------------------------------//47
--------- p_edge[rb] = i; ------------------------------//2f
------- } ----------------------------------------------//4d
----- } ----------------------------------------------------//84
--- } ------------------------------------------------------//5c
--- if (d[t] == INF) break; ------------------------------//16
--- ll addflow = k - flow; -------------------------------//39
--- for (int v = t; v! = s; v = p[v]) { ------------------//06
----- int pv = p [v]; size_t pr = p_edge[v]; -------------//41
----- addflow = min(addflow, g[pv][pr].u - g[pv][pr].f); --//45
--- } ----------------------------------------------------//d6
--- for (int v = t; v! = s; v = p [v]) { -----------------//54
----- int pv = p [v]; --------------------------------------//ee
----- size_t pr = p_edge[v], r = g[pv][pr].back; ----------//00
----- g[pv][pr].f += addflow; ------------------------------//7b
----- g[v][r].f -= addflow; --------------------------------//51
----- cost += g[pv][pr].c * addflow; ----------------------//0b
--- } ----------------------------------------------------//49
--- flow += addflow; -------------------------------------//8d
- } ---------------------------------------------------------//71
- return {flow, cost}; --------------------------------------//91
} ----------------------------------------------------------//14
```

5.3. Kosaraju.

```
// O(N) algo to find SCC ---------------------------------//fb
 ------------------------------------------------------------//d5
void dfs_order(vvi &g, int i, vec<bool> &vis, vi &dfso){ --//2b
- vis[i] = 1; -----------------------------------------------//21
- for(int &p : g[i]) ----------------------------------------//e5
--- if(!vis[p]) ------------------------------------------//5a
----- dfs_order(g, p, vis, dfso); --------------------------//34
- dfso.push_back(i); ----------------------------------------//e6
} ----------------------------------------------------------//fb
 ------------------------------------------------------------//92
void l_dfs(vvi &g, int i, vec<bool> &vis, vi &labels, int &l){
- if(!vis[i]){ ----------------------------------------------//35
--- vis[i] = 1; ------------------------------------------//eb
--- ++l; -------------------------------------------------//12
- } ----------------------------------------------------------//c8
- labels[i] = l; --------------------------------------------//41
- for(int &x : g[i]) ----------------------------------------//2e
--- if(!vis[x]){ -----------------------------------------//8f
----- vis[x] = 1; ------------------------------------------//bc
--- l_dfs(g, x, vis, labels, l); -------------------------//2e
- } ----------------------------------------------------------//b8
} ----------------------------------------------------------//ad
 ------------------------------------------------------------//8a
// Returns a graph (DAG) of SCC --------------------------//c2
// g is the graph to build on ----------------------------//88
// labels[i] = SCC to which g[i] belongs -----------------//42
// dfs = order in which nodes were visited in the first ---//27
 ------------------------------------------------------------//4d
vvi kosaraju(vvi &g, vi &labels, vi &dfso){ ----------------//26
- //we make a stack based on dfs order (in a way that no node is\
--- before someone reachable by him - except for SCC case! //9e
- dfso.reserve(g.size()); ----------------------------------//08
- vec<bool> vis(g.size()); ----------------------------------//10
- for(int i = 0; i < g.size(); i++) -------------------------//d2
--- if(!vis[i]) ------------------------------------------//4f
----- dfs_order(g, i, vis, dfso); --------------------------//5f
- //we need the reverse graph ----------------------------//0c
```

```
- vvi rg(g.size()); ----------//63
- for(int i = 0; i < g.size(); i++) ----------//fb
--- for(int &p : g[i]) ----------//10
----- rg[p].push_back(i); ----------//70
- //now, we just go backwards from last node visited and build the scc
- fill(vis.begin(), vis.end(), 0); ----------//c4
- int l = -1; ----------//7b
- for(int i = g.size() - 1; i >= 0; i--) ----------//9e
--- if(!vis[dfso[i]]) ----------//22
----- l_dfs(rg, dfso[i], vis, labels, l); ----------//64
- //labels are set, we construct the graph ----------//76
- vvi scc(l+1);//l started as -1, so we add 1 ----------//3e
- set<pii> edges; ----------//e4
- for(int i = 0; i < g.size(); i++){ ----------//9f
--- for(int &p : g[i]) ----------//3d
----- if(labels[i] != labels[p]){ ----------//94
------- if(edges.count({labels[i], labels[p]}) == 0){ -----//37
--------- scc[labels[i]].push_back(labels[p]); ----------//a0
--------- edges.insert({labels[i], labels[p]}); ----------//9f
------- } ----------//79
----- } ----------//a1
- } ----------//e8
- return scc; ----------//73
} ----------//e7
```

5.4. Hungarian.

```
// O(N^3) Hungarian algorithm for the assignment problem --//42
// Min Cost Bipartite Matching ----------//49
// cost[i][j] = cost to match i with j ----------//07
// Returns {minCost, match[]}, ----------//06
// where match[i] = who we matched i with ----------//b4
 ----------//fb
// Use 0-based, n <= m ----------//70
template <typename T> ----------//d3
pair<T, vi> min_cost_matching(const vector<vector<T>> &cost) {
        // internally, this will be 1-based, ----------//70
        // but you must call 0-based ! ----------//9d
        const T T_INF = numeric_limits<T>::max() / 2.0; ----------//aa
        int n = cost.size(); ----------//94
        int m = cost[0].size(); ----------//29
        vector<T> u(n + 1); ----------//ea
        vector<T> v(m + 1); ----------//a7
        vector<T> p(m + 1); ----------//4d
        vector<T> way(m + 1); ----------//40
        for (int i = 1; i <= n; ++i) { ----------//3b
                p[0] = i; ----------//ed
                int j0 = 0; ----------//49
                vector<T> minv (m + 1, T_INF); ----------//97
                vector<bool> used (m + 1, false); ----------//3c
                do { ----------//84
                        used[j0] = true; ----------//6a
                        int i0 = p[j0],  j1; ----------//17
                        T delta = T_INF; ----------//0e
                        for (int j = 1; j <= m; ++j) ----------//6f
                                if (!used[j]) { ----------//34
                                        int cur = cost[i0-1][j-1] - u[i0] - v[j]; ----------//4d
                                        if (cur < minv[j]) ----------//07
                                                minv[j] = cur,  way[j] = j0; ----------//39
                                        if (minv[j] < delta) ----------//9b
                                                delta = minv[j],  j1 = j; ----------//b0
                                } ----------//a3
                        for (int j=0; j<=m; ++j) ----------//82
                                if (used[j]) ----------//bf
                                        u[p[j]] += delta,  v[j] -= delta; ----------//76
                                else ----------//b3
                                        minv[j] -= delta; ----------//4f
                        j0 = j1; ----------//86
                } while (p[j0] != 0); ----------//48
                do { ----------//65
                        int j1 = way[j0]; ----------//a9
                        p[j0] = p[j1]; ----------//ac
                        j0 = j1; ----------//b6
                } while (j0); ----------//4d
        } ----------//6d
        // returns both cost and matchings ----------//71
        T sol = -v[0]; ----------//c9
        vector<int> mates(n); ----------//c9
        for(int i = 1; i <= m; i++) if(p[i] != 0) ----------//45
                mates[p[i]-1] = i-1; ----------//2c
        // build matchings ----------//2c
        return {sol, mates}; ----------//73
} ----------//5d
```

5.5. Floyd Warshall.

```
// O(N^3) Floyd Warshall to find APSP ----------//90
// Takes adj matrix (a[i][j] = INF means no edge!) --------//e0
// Return dist matrix. careful for overflows! ----------//a7
 ----------//d8
vvi floyd_warshall(vvi &adj){ ----------//6b
- vvi d(adj); ----------//6e
- int n = adj.size(); ----------//4c
- for(int k = 0; k < n; k++) ----------//8c
-       for(int i = 0; i < n; i++) ----------//88
-               for(int j = 0; j < n; j++) ----------//1e
-                       if(max(d[i][k], d[k][j]) != INF) ----------//b7
-                               d[i][j] = min(d[i][j], d[i][k] + d[k][j]); ------//21
-       return d; ----------//1e
} ----------//29
```

5.6. Dinic.

```
// O(VE^2) Dinic algorithm for MaxFlow ----------//bb
// Diff with Scaling Dinic (asyntotically faster) ---------//b2
// commented above Dinic's code ----------//e6
 ----------//49
// We represent the graph as an adiacency list ----------//97
// of edges' list indexes !!! ----------//3c
 ----------//3e
struct edge{ //97 ----------//6a
        int from, to; //to all the connected nodes ----------//17
        int f; // the flow we can still push ----------//0e
}; ----------//3c
 ----------//34
using graph = vvi; ----------//4d
 ----------//07
using edge_list = vector<edge>; ----------//39
 ----------//02
// Also, we use i^1 to get back/last edge of i-th edge ---------//9b
constexpr int back_edge(int i){ return i ^ 1;} ----------//b0
 ----------//0a
// Assumes directed edges ----------//82
const bool IS_DIRECTED = true; ----------//bf
 ----------//76
void add_edge(graph &g, edge_list &e, int a, ----------//b3
        int b, int c, bool directed = IS_DIRECTED){ ----------//4f
        int index = e.size(); ----------//86
        g[a].push_back(index); ----------//48
        g[b].push_back(back_edge(index)); ----------//65
        e.push_back({a, b, c}); ----------//a9
        e.push_back({b, a, directed ? 0 : c}); ----------//ac
} ----------//dc
 ----------//73
//int level_dfs(graph &g, edge_list &el, vi &level, ------//bb
//vi &pointer, int source, int sink, int lim){ ---------//c9
int level_dfs(graph &g, edge_list &el, vi &level, ---------//f7
vi &pointer, int source, int sink, int flow = INF){ ---//5e
// if(source == sink) return lim; ----------//19
        if(source == sink) return flow; ----------//5c
        for(int &idx = pointer[source]; idx < g[source].size(); -//c4
        idx++){ ----------//3c
                int i = g[source][idx]; ----------//68
                edge &e = el[i]; ----------//97
                // if(level[e.to] == level[source] + 1 && e.f >= lim){ //80
                if(level[e.to] == level[source] + 1 && e.f > 0){ ------//ee
                        // int pushed = level_dfs(g, el, level, pointer, ----//cf
                        //   e.to, sink, lim); ----------//60
                        int pushed = level_dfs(g, el, level, pointer, -------//2e
                                e.to, sink, min(flow, e.f)); ----------//c4
                        if(pushed > 0){ ----------//18
                                el[i].f -= pushed; ----------//52
                                el[back_edge(i)].f += pushed; ----------//53
                                return pushed; ----------//f1
                        } ----------//3d
                } ----------//e9
        } ----------//ca
        return 0; // couldn't push flow ----------//1a
} ----------//f5
 ----------//66
// bool level_bfs(graph &g, edge_list &el, vi &level, -----//93
//   int source, int sink, int lim){ ----------//b4
bool level_bfs(graph &g, edge_list &el, vi &level, --------//53
        int source, int sink){ ----------//44
        fill(level.begin(), level.end(), INF); ----------//64
        level[source] = 0; ----------//f8
        queue<int> q; ----------//fe
        q.push(source); ----------//2c
        while(!q.empty() && level[sink] == INF){ ----------//51
                int curr = q.front(); q.pop(); ----------//1a
                for(int &i : g[curr]){ ----------//e0
                        edge &e = el[i]; ----------//3c
                        // if(e.f >= lim && level[e.to] == INF){ ----------//cc
```

```
----- if(e.f > 0 && level[e.to] == INF){ ------------------//44
------- level[e.to] = level[curr] + 1; --------------------//a3
------- q.push(e.to); -------------------------------------//64
----- } -------------------------------------------------//25
--- } ---------------------------------------------------//a4
- } -----------------------------------------------------//53
- return level[sink] != INF; ----------------------------//fc
} -------------------------------------------------------//c8
 -------------------------------------------------------//7c
ll dinic(graph &g, edge_list &el, int source, int sink){ --//6b
- vi level(g.size()); -----------------------------------//22
- ll flow = 0; ------------------------------------------//7f
- // TODO: check for long long ----------------------------//d4
- // int lim = 1<<30; -------------------------------------//99
- // while(lim > 0){ --------------------------------------//04
- while(level_bfs(g, el, level, source, sink)){ ---------//7e
--- // core optimization trick, for chained DFSs ----------//ba
--- vi pointer(g.size()); ----------------------------------//90
--- // if we can't send enough flow ----------------------//b3
--- // if(!level_bfs(g, el, level, source, sink, lim)){ ---//2b
--- //   lim /= 2; ----------------------------------------//75
--- //   continue; ----------------------------------------//dc
--- // } ---------------------------------------------------//ea
--- // while(level_dfs(g, el, level, pointer, source, sink, lim)){
--- //    flow += lim; ------------------------------------//de
--- // } ---------------------------------------------------//8c
--- int tmp; -----------------------------------------------//a7
--- while((tmp = level_dfs(g, el, level, pointer, source, sink))){
----- flow += tmp; ------------------------------------------//68
--- } ---------------------------------------------------//6e
- } -----------------------------------------------------//36
- return flow; -------------------------------------------//b6
} -------------------------------------------------------//d0
```

5.7. Dijkstra.

```
// (untested) O(ElogV) Dijkstra with priority queue -------//2c
// takes a vector<vector<pii>> and return vector of -------//f6
// distances. watchout for int overflow -------------------//35
vi dijkstra(graph &g, int source){ ---------------------//a1
- vi dist(g.size(), INF); ------------------------------//e7
- priority_queue<pii, vector<pii>, greater<pii>> pq; ------//39
- dist[source] = 0; ------------------------------------//a2
- pq.push({0, source}); --------------------------------//90
- while(!pq.empty()){ ----------------------------------//5c
--- pii curr = pq.top(); pq.pop(); -------------------//70
--- if(dist[curr].second < curr.first) ---------------//50
----- continue; ---------------------------------------//8d
--- for(pii &e : g[curr.second]) ---------------------//3f
----- if(curr.first + e.second < dist[e.first]){ ----------//27
------- dist[e.first] = curr.first + e.second; ------------//c1
------- pq.push({dist[e.first], e.first}); ----------------//4a
----- } -----------------------------------------------//2c
- } -----------------------------------------------------//65
- return dist; -------------------------------------------//27
} -------------------------------------------------------//8c
```

5.8. Articulation Points.

```
vector<int> dfs_low(n),dfs_num(n); ---------------------//42
int cnt=0; ---------------------------------------------//a8
void abdfs(int p){ -------------------------------------//d1
--- dfs_low[p]=dfs_num[p]=++cnt; ---------------------//a3
--- for(auto i:g[p]){ --------------------------------//03
------- if(!dfs_num[i]){ ---------------------------//29
----------- if(p==root)rootChildren++;//for if root is articulation point
----------- dfs_parent[i]=p; ---------------------------//26
----------- abdfs(i); ----------------------------------//d2
----------- if(dfs_low[i]>=dfs_num[p]){}//p is articulation point
----------- if(dfs_low[i]>dfs_num[p]){} //p-i is bridge ---//fc
----------- dfs_low[p]=min(dfs_low[p],dfs_low[i]); --------//8c
------- } else if(i!=dfs_parent[p]) dfs_low[p]=min(dfs_low[p],dfs_num[i]);
--- } ------------------------------------------------//81
} -------------------------------------------------------//8a
 -------------------------------------------------------//f7
```

5.9. 2sat.

```
// O(N) algorithm to solve 2SAT ---------------------------//d6
 -------------------------------------------------------//e5
// 1) make a graph of size 2 * N --------------------------//e6
// 2) add all implications (c1, i1) && (c2, i2)... --------//02
// 3) run Kosaraju on the obtained graph ------------------//d6
// 4) if label[x] = label[x^1], then we have no solution --//97
// 5) we have a solution. (and then? :O) ------------------//83
 -------------------------------------------------------//9a
// if (x = a) then (y = b)  -------------------------------//59
void add_implication(int x, bool a, int y, bool b, vvi &sat){
- x *= 2, y *= 2; --------------------------------------//7c
- //nego se necessario -----------------------------------//49
- if(!a) x ^= 1; ---------------------------------------//e6
- if(!b) y ^= 1; ---------------------------------------//db
- //aggiunge x -> y ---------------------------------------//22
- sat[x].push_back(y); ----------------------------------//c0
- //aggiunge !y -> !x -------------------------------------//65
- sat[y ^ 1].push_back(x ^ 1); --------------------------//1b
} -------------------------------------------------------//f7
```

6. Geom

6.1. Segment Point Distance.

```
// Magic formula to know point to line distance -----------//fd
// Very accurate :P ---------------------------------------//15
 -------------------------------------------------------//9b
// s.a.x <= s.b.x -----------------------------------------//53
double ps_distance(point &p, segment &s){ --------------//91
- //magic dot product / square len ratio ----------------//8f
- vector<double> v{p.x - s.a.x, p.y - s.a.y}; -----------//a2
- vector<double> u{s.b.x - s.a.x, s.b.y - s.a.y}; -------//31
- auto dot = v[0] * u[0] + v[1] * u[1]; ----------------//d2
- auto len = square(u[0]) + square(u[1]); --------------//23
- if(len == 0) return p.distance(s.a); ------------------//df
- auto ratio = dot / len; -------------------------------//08
- //cout << dot << ' ' << len << " -> " << ratio << '\n'; -//f4
- if(ratio <= 0) return p.distance(s.a); ----------------//e0
- if(ratio >= 1) return p.distance(s.b); ----------------//09
- point proj = {s.a.x + u[0] * ratio, s.a.y + u[1] * ratio};
- return p.distance(proj); ------------------------------//0c
} -------------------------------------------------------//21
```

6.2. Rotating Calipers.

```
/* ------------------------------------------------------//b6
- Rotating calipers: ------------------------------------//5c
-         A pair of parallel lines that can be rotated ----------
-         around the polygon and "holds" it as tight as possible
 -------------------------------------------------------//00
- I hope you won't need this. ---------------------------//3d
- But as you are here, let's see what we have: ----------//8b
-         (assuming we want farthest points) ---------------------
-    1) build convex hull ----------------------------------//59
- 2) pick any normal vector ? and find calipers ----------//83
-         that match this vector: --------------------------------
-                 (I assume this means that you pick any two hull-
        3) Test the points facing the calipers -----------------
- 4) find minimum angle to get next point ----------------//e7
- 5) rotate calipers by that angle ----------------------//ea
- 6) if state is already visited, stop the calipers -------//3f
 -------------------------------------------------------//a0
        The above runs in O(N), assuming you can code it. --------
*/ -----------------------------------------------------//8d
```

6.3. Graham Scan.

```
long long cross(P a, P b, P c){ ----------------------//64
        return (b.x-a.x)*(c.y-a.y)-(b.y-a.y)*(c.x-a.x); ----------
} -------------------------------------------------------//89
long long dist2(P a, P b){ ---------------------------//04
        return (a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y); ----------
} -------------------------------------------------------//44
vector<P> graham_scan(vector<P>& a){ -----------------//78
        sort(a.begin()+1,a.end(), ----------------------------------
        [&](P b, P c){ ---------------------------------------------
                if(cross(a[0],b,c)==0) -----------------------------
                        return dist2(a[0],b)<dist2(a[0],c); ------
                return cross(a[0],b,c)>0;          ----------------
        }); --------------------------------------------------------
        vector<P> ch; ----------------------------------------------
        ch.push_back(a[0]); ----------------------------------------
        ch.push_back(a[1]); ----------------------------------------
        for(auto &p:a){ --------------------------------------------
                while(ch.size()>=2 && cross(ch[ch.size()-2],ch.bac
                        ch.pop_back(); ---------------------------
                ch.push_back(p); -------------------------------
        } ----------------------------------------------------------
        return ch; -------------------------------------------------
} -------------------------------------------------------//87
 -------------------------------------------------------//34
```

6.4. Geom Lib.

```
//Blaise1 geometry lib, in its shrinked version ----------//50
 -------------------------------------------------------//5c
using T = long double; T EPS = 1e-12; T INF = 1e18; -------//25
struct point { -----------------------------------------//bd
- T x, y; ----------------------------------------------//01
- point(T x, T y) : x(x), y(y) { } ----------------------//04
```

```
- bool operator==(point &o) { return (fabs(x-o.x)<=EPS) ?  //99
--- ((fabs(y-o.y)<=EPS) ? true : false) : false; } --------//ee
- bool operator!=(point &o) { return !(*this == o); } -----//cd
- point operator+(point &o) { return {x+o.x,y+o.y}; } -----//ed
- point operator-() { return {-x,-y}; } -----------------//7b
- point operator-(point &o) { return *this+(-o); } ------//31
- point operator*(T &d) { return {x*d,y*d}; } -----------//94
- point operator*(point &o) { return {x*o.x,y*o.y}; } ---//6d
- T operator%(point &o) { return (*this*o).magnitude(); } -//5f
- T operator^(point &o) { return x*o.y-y*o.x; } ---------//db
- T distance(point &o) { --------------------------------//90
--- return hypotl(fabs(x - o.x), fabs(y - o.y)); } -------//d1
- T sq_distance(point &o) { -----------------------------//d7
--- return fabs(x-o.x)*fabs(x-o.x) +fabs(y-o.y)*fabs(y-o.y); }
- point normalized() { ----------------------------------//5b
--- point tmp(x, y); tmp.normalize(); return tmp; } -------//46
- void normalize(){ T hy = hypotl(x, y); x /= hy; y /= hy; }
- T magnitude() { return x + y; } ------------------------//bb
- point rotated_rad(T rad) { ----------------------------//44
--- point tmp(x, y); tmp.rotate_rad(rad); return tmp; } ---//0f
- void rotate_rad(T rad){ -------------------------------//29
--- T cos = cosl(rad); T sin = sinl(rad); ----------------//45
--- cos = fabs(cos) < EPS ? 0 : cos; ---------------------//41
--- sin = fabs(sin) < EPS ? 0 : sin; ---------------------//c3
--- x = x * cos - y * sin; y = x * sin + y * cos; } -------//2b
- T angle(point& o) { -----------------------------------//ef
--- point self = this->normalized(); ---------------------//25
--- point oth = o.normalized(); --------------------------//bf
--- if(self == oth) return 0; ----------------------------//a6
--- return fabs(acosl(self % oth)); } }; -----------------//b8
 ------------------------------------------------------------//7c
struct line { ------------------------------------------------//ba
- point dir, perp; -------------------------------------------//7a
- T a,b,c; ---------------------------------------------------//79
- line(T a, T b, T c) : perp(a,b), dir(0,0) { -------------//c5
--- assert(!(a == 0 && b == 0)); -------------------------//18
--- perp.normalize(); dir = perp.rotated_rad(-M_PI/2); ---//05
--- this->c = c / hypotl(a,b); ---------------------------//82
--- this->a = perp.x; this->b = perp.y; } ----------------//f7
- line(point p1, point p2) : perp(0,0), dir(p2-p1) { -----//e2
--- assert(p1 != p2); ------------------------------------//8f
--- dir.normalize(); perp = dir.rotated_rad(M_PI/2); -----//6a
--- a = perp.x; b = perp.y; c = -(perp % p1); } ----------//55
- bool inLine(point &p) { -------------------------------//c5
--- return fabs(dir.x * p.x + dir.y * p.y + c) <= EPS; } --//fd
- bool operator==(line &o) { ----------------------------//a0
--- if(fabs(a - o.a) > EPS) return false; ----------------//cd
--- line inv(-o.a,-o.b,-o.c); ----------------------------//60
--- return fabs(a-inv.a)<=EPS && fabs(b-inv.b)<=EPS && ---//57
----- fabs(c-inv.c)<=EPS; } --------------------------------//4f
- T signed_distance(point &p) { return perp % p + c; } ---//c2
- point project(point &p) { -----------------------------//5c
--- return p - (perp * signed_distance(p)); } ------------//91
- line parallel(point &p) { -----------------------------//a6
--- line res = *this; res.c -= signed_distance(p); -------//2d
--- return res; } ----------------------------------------//6a
- point intersect(line &s) { ----------------------------//22
--- line l = *this; T d = l.a * s.b - s.a * l.b; ---------//55
--- if(fabs(d) <= EPS) return {-INF, -INF}; --------------//61
--- return { (s.c*l.b-l.c*s.b)/d,(s.c*l.a-l.c*s.a)/(-d)}; } };
struct segment { ---------------------------------------------//60
- point start, end; -------------------------------------------//6d
- segment(point p1, point p2) : start(p1), end(p2) { } ----//e9
- bool lies(point &a) { ---------------------------------------//f1
--- point b = start; point c = end; line l(b,c); ---------//b9
--- T dist = l.signed_distance(a); -----------------------//79
--- if(!(fabs(dist) <= EPS)) return false; ---------------//2c
--- point ba = a-b; point bc = c-b; ----------------------//a6
--- point ca = a-c; point cb = b-c; ----------------------//56
--- return !(ba % bc < 0 || ca % cb < 0); } --------------//7e
- point intersect(segment &s2) { -------------------------//58
--- segment s1 = *this; line l1(s1.start, s1.end); -------//1f
--- line l2(s2.start, s2.end); ---------------------------//94
--- point inter = l1.intersect(l2); point inf(-INF, -INF); //1c
--- if(inter == inf){ ------------------------------------//7e
----- if(l1 == l2){ ----------------------------------------//ba
------- if(s1.lies(s2.start)) return s2.start; ------------//3d
------- if(s1.lies(s2.end)) return s2.end; ----------------//f4
------- if(s2.lies(s1.start)) return s1.start; ------------//b0
------- if(s2.lies(s1.end)) return s1.end; } --------------//1e
----- return inf; } ----------------------------------------//1c
--- if(s1.lies(inter) && s2.lies(inter)) return inter; ----//9e
--- return inf; } ----------------------------------------//ed
- T sq_distance(point &p) { -----------------------------//1e
--- point v(p.x - start.x, p.y - start.y); ---------------//ff
--- point u(end.x - start.x, end.y - start.y); -----------//9d
--- T dot = v % u; T len = (u * u).magnitude(); ----------//29
--- if(fabs(len) <= EPS) return p.sq_distance(start); ----//26
--- T ratio = dot / len; ---------------------------------//e1
--- if(ratio <= EPS) return p.sq_distance(start); --------//49
--- if(ratio >= 1 - EPS) return p.sq_distance(end); ------//a3
--- point proj = (u * ratio) + start; --------------------//a9
--- return p.sq_distance(proj); } ------------------------//15
- T distance(point &p) { return sqrtl(sq_distance(p)); } };//e1
```

6.5. Closest Pair.

```
// Return the closest pair of points in O(NlogN) ----------//20
using point = complex; ---------------------------------------//7b
struct cmpx { ------------------------------------------------//64
- bool operator ()(const point &a, const point &b) { ------//f7
--- return abs(real(a) - real(b)) > EPS ? -----------------//5e
----- real(a) < real(b) : imag(a) < imag(b); } }; ---------//d1
 ------------------------------------------------------------//d2
struct cmpy { ------------------------------------------------//86
- bool operator ()(const point &a, const point &b) { ------//67
- return abs(imag(a) - imag(b)) > EPS ? -----------------//64
----- imag(a) < imag(b) : real(a) < real(b); } }; ---------//06
 ------------------------------------------------------------//a1
pair<point, point> closest_pair(vector<point> pts) { ------//ff
- pair<point, point> sol; ------------------------------------//64
- sort(pts.begin(), pts.end(), cmpx()); ----------------------//3b
- set<point, cmpy> cur; --------------------------------------//ad
- set<point, cmpy>::const_iterator it, jt; -------------------//44
- double mn = INFINITY; --------------------------------------//41
- for (int i = 0, l = 0; i < size(pts); i++) { ---------------//bb
--- while (real(pts[i]) - real(pts[l]) > mn) --------------//eb
----- cur.erase(pts[l++]); ---------------------------------//e2
--- it = cur.lower_bound(point(-INFINITY, imag(pts[i]) - mn));
--- jt = cur.upper_bound(point(INFINITY, imag(pts[i]) + mn));
--- while (it != jt) { -----------------------------------//b4
----- double x = abs(*it - pts[i]); ------------------------//2f
----- if(x < mn){ ------------------------------------------//66
------- mn = x; ----------------------------------------//c7
------- sol = {*it, pts[i]}; ---------------------------//f0
----- } ----------------------------------------------------//08
----- it++; ------------------------------------------------//2e
--- cur.insert(pts[i]); } --------------------------------//b4
- return sol; } ----------------------------------------------//eb
```

6.6. Andrew Mc.

```
// Convex Hull with Andrew Monotone chain algorithm -------//a1
// Also supports queries to check if point ----------------//76
// is contained in convex hull ----------------------------//2a
 ------------------------------------------------------------//44
struct point { -----------------------------------------------//da
- int x, y; --------------------------------------------------//bc
- bool operator ==(point &other){ ----------------------------//a1
--- return x == other.x && y == other.y; ------------------//fa
- } ----------------------------------------------------------//b6
- bool operator<(point &other){ ------------------------------//26
--- return tie(x, y) < tie(other.x, other.y); -------------//0f
- } ----------------------------------------------------------//94
}; -----------------------------------------------------------//a4
 ------------------------------------------------------------//1b
// -1 ccw, 0 coll, 1 cw -----------------------------------//b7
int orientation(point &a, point &b, point &c){ -------------//8e
- ll val = (b.y - a.y) * (ll)(c.x - b.x) - ----------------//79
----- (b.x - a.x) * (ll)(c.y - b.y); -----------------------//3a
- return val < 0 ? -1 : !!val; ------------------------------//5c
} ------------------------------------------------------------//77
 ------------------------------------------------------------//c0
struct segment { ---------------------------------------------//78
- point a, b; ------------------------------------------------//61
- bool contains(point &p){ -----------------------------------//a4
--- if(orientation(a, b, p) != 0) ------------------------//9b
----- return 0; --------------------------------------------//7a
--- bool x = p.x >= a.x && p.x <= b.x; --------------------//0e
--- bool y = p.y >= min(a.y, b.y) && p.y <= max(a.y, b.y); //8d
--- return x && y; ---------------------------------------//9e
- } ----------------------------------------------------------//84
- bool operator& (segment &s){ --------------------------------//f7
--- return intersect(s); ----------------------------------//eb
- } ----------------------------------------------------------//99
- bool intersect(segment &s){ ---------------------------------//a3
--- ll o1 = orientation(a, b, s.a); -----------------------//c7
--- ll o2 = orientation(a, b, s.b); -----------------------//91
--- ll o3 = orientation(s.a, s.b, a); ---------------------//9c
--- ll o4 = orientation(s.a, s.b, b); ---------------------//24
```

```
--- if(o1 != o2 && o3 != o4) return 1; --------------------//95
--- return contains(s.a) || contains(s.b) || --------------//75
----- s.contains(a) || s.contains(b); ---------------------//88
- } -------------------------------------------------------//45
- segment(point p1, point p2) : a(p1), b(p2){ -------------//a5
--- if(a.x > b.x) -----------------------------------------//68
----- swap(a, b); -----------------------------------------//99
- } -------------------------------------------------------//52
}; --------------------------------------------------------//a4
struct convex_hull : vector<point>{ -----------------------//a2
- vector<point> &h = *this; -------------------------------//49
- vector&lt;segment&gt; bottom, top; ----------------------------//07
- convex_hull (vector<point> &v){ -------------------------//0a
--- sort(v.begin(), v.end()); -----------------------------//c1
--- for(int i = 0; i < v.size(); i++){ --------------------//d6
----- while(h.size() >= 2 && ------------------------------//e7
------- orientation(h[h.size()-2], h.back(), v[i]) <= 0) --//d0
------- h.pop_back(); -------------------------------------//f8
----- h.push_back(v[i]); ----------------------------------//d1
--- } -----------------------------------------------------//fd
--- int s = h.size() + 1, tops = h.size(); ----------------//da
--- for(int i = v.size() - 2; i >= 0; i--){ ---------------//dd
----- while(h.size() >= s && ------------------------------//02
------- orientation(h[h.size()-2], h.back(), v[i]) <= 0) --//a3
------- h.pop_back(); -------------------------------------//3b
----- h.push_back(v[i]); ----------------------------------//87
--- } -----------------------------------------------------//f8
--- //reminder : we have an extra point! ------------------//92
--- // we build top/bottom segments -----------------------//ac
--- top.reserve(tops - 1), bottom.reserve(h.size() - tops + 1);
--- for(int i = 0; i + 1 < tops; i++) ---------------------//cb
----- top.push_back(segment(h[i], h[i+1])); ---------------//38
--- for(int i = h.size() - 1; i - 1 >= tops - 1; i--) -----//69
----- bottom.push_back(segment(h[i], h[i-1])); ------------//89
- } -------------------------------------------------------//12
- //we also count point on the borders --------------------//c4
- bool contains(point &p){ --------------------------------//a2
--- //we create a line to the sky -------------------------//4a
--- segment line(p, {p.x, INF}); --------------------------//11
--- //if p lays in a segment, it's in ---------------------//b9
--- //if line doesn't hit top, it's out -------------------//32
--- //if line hits bot, it's out --------------------------//c3
--- //if none occurs, it's in -----------------------------//a4
-----------------------------------------------------------//7f
--- //we search top hull ----------------------------------//a3
--- //we find first segment with b.x >= p.x, --------------//b9
--- //so that intersection is there or eventually --------//9c
--- int l = 0, r = top.size(); ----------------------------//33
--- while(l < r){ -----------------------------------------//91
----- int m = (l + r)/2; ----------------------------------//df
----- if(top[m].b.x >= p.x) -------------------------------//eb
------- r = m; --------------------------------------------//25
----- else ------------------------------------------------//82
------- l = m +1; -----------------------------------------//64
--- } -----------------------------------------------------//fb
--- if(r == top.size()) return 0; -------------------------//b4
--- segment &t = top[l]; ----------------------------------//d7
--- if(t.contains(p)) return 1; ---------------------------//89
--- if(!t.intersect(line)) return 0; ----------------------//14
--- //we repeat for bottom ---------------------------------//25
--- l = 0, r = bottom.size(); -----------------------------//a1
--- while(l < r){ -----------------------------------------//aa
----- int m = (l + r)/2; ----------------------------------//07
----- if(bottom[m].b.x >= p.x) r = m; ---------------------//ed
----- else l = m+1; ---------------------------------------//4a
--- } -----------------------------------------------------//cf
--- //see above ---------------------------------------------//22
--- if(r == bottom.size()) return 0; ----------------------//57
--- segment &b = bottom[l]; -------------------------------//50
--- if(b.contains(p)) return 1; ---------------------------//cc
--- if(b.intersect(line)) return 0; -----------------------//1d
--- //only 1 intersection so it's good --------------------//cb
--- return 1; ---------------------------------------------//ee
- } -------------------------------------------------------//4c
}; --------------------------------------------------------//fb
```

7. Ds

7.1. Wavelet Tree.

```
// Wavelet tree data structure ----------------------------//5d
// can handle some 2D queries -----------------------------//95
// O(NlogA) preprocess ------------------------------------//dd
// O(logA) queries ----------------------------------------//ad
-----------------------------------------------------------//fe
// 1-indexed! ---------------------------------------------//61
struct wavelet { ------------------------------------------//55
- int lo, hi; ---------------------------------------------//64
- vi b; ---------------------------------------------------//0f
- wavelet *l = 0, *r = 0; ---------------------------------//f9
- // begin, end, min val, max val ------------------------//d8
- wavelet(vi::iterator pl, vi::iterator pr, ---------------//e9
----- int x, int y) { -------------------------------------//ab
--- lo = x, hi = y; ---------------------------------------//eb
--- if (lo == hi || pl >= pr) return; ---------------------//37
--- int mid = lo + (hi - lo) / 2; -------------------------//37
--- auto f = [mid](int x) { -------------------------------//12
----- return x <= mid; }; ---------------------------------//cc
--- b.reserve(pr - pl + 1); -------------------------------//76
--- b.push_back(0); ---------------------------------------//44
--- for (auto it = pl; it != pr; it++) --------------------//9a
----- b.push_back(b.back() + f(*it)); ---------------------//07
--- auto it = stable_partition(pl, pr, f); ----------------//5a
--- l = new wavelet(pl, it, lo, mid); ---------------------//a9
--- r = new wavelet(it, pr, mid + 1, hi); -----------------//24
- } -------------------------------------------------------//c4
-----------------------------------------------------------//83
- // retrieve the k_th minimal value in range [l, r] ------//c3
- int kth(int l, int r, int k) { --------------------------//b3
--- if (l > r) return 0; ----------------------------------//de
--- if (lo == hi) return lo; ------------------------------//3d
--- int le = b[l - 1], ri = b[r]; -------------------------//81
--- if (ri - le >= k) --------------------------------------//61
----- return this->l->kth(le+1,ri,k); ---------------------//b5
--- return this->r->kth(l-le,r-ri,k-(ri-le)); -------------//4c
- } -------------------------------------------------------//cb
-----------------------------------------------------------//7f
- // count values less than k in range [l, r] -------------//99
- int clt(int l, int r, int k) { --------------------------//d8
--- if (l > r || k <= lo) return 0; -----------------------//05
--- if (k > hi) return r - l + 1; -------------------------//f9
--- int le = b[l - 1], ri = b[r]; -------------------------//26
--- return this->l->clt(le+1,ri,k) + ----------------------//f2
---- this->r->clt(l-le, r-ri, k); -------------------------//20
- } -------------------------------------------------------//2e
-----------------------------------------------------------//30
- // count equal to k in range [l, r] ---------------------//67
- int ce(int l, int r, int k) { ---------------------------//ba
--- if (k < lo || k > hi || l > r) ------------------------//f8
----- return 0; -------------------------------------------//9b
--- if (lo == hi) return r - l + 1; -----------------------//82
--- int le = b[l - 1], ri = b[r]; -------------------------//75
--- return this->l->ce(le + 1, ri, k) + -------------------//6e
----- this->r->ce(l - le, r - ri, k); ---------------------//3f
- } -------------------------------------------------------//30
}; --------------------------------------------------------//b5
```

7.2. Treap.

```
/* Implicit Treap (BBST) ----------------------------------//23
- can be augmented or used as map/set ---------------------//f3
- O(logN) expected queries */ -----------------------------//5d
-----------------------------------------------------------//89
// srand(time(0)); ----------------------------------------//82
struct treap{ ---------------------------------------------//39
private: --------------------------------------------------//f7
- struct node{ --------------------------------------------//4a
--- int val; ----------------------------------------------//10
--- int p = rand(); ---------------------------------------//0f
--- // int h = 1; -----------------------------------------//00
--- int size = 1; -----------------------------------------//4c
--- node *l = nullptr; ------------------------------------//f3
--- node *r = nullptr; ------------------------------------//fd
--- node (int a){ -----------------------------------------//ba
----- val = a; --------------------------------------------//6f
--- } -----------------------------------------------------//5d
--- void update(){ ----------------------------------------//34
----- // int lh = l != nullptr ? l->h : 0; ---------------//f7
----- // int rh = r != nullptr ? r->h : 0; ---------------//ae
----- // h = 1 + max(lh, rh); ---------------------------//a0
----- int ls = l != nullptr ? l->size : 0; ----------------//42
----- int rs = r != nullptr ? r->size : 0; ----------------//c0
----- size = 1 + ls + rs; ---------------------------------//ae
--- } -----------------------------------------------------//96
- }; ------------------------------------------------------//31
- node *merge(node *a, node *b){ --------------------------//28
--- if(a == nullptr) return b; ----------------------------//2a
--- if(b == nullptr) return a; ----------------------------//8c
--- if(a->p < b->p){ --------------------------------------//51
----- a->r = merge(a->r, b); ------------------------------//92
----- a->update(); ----------------------------------------//02
```

```
----- return a; ----------------------------------------------//0c
--- } else{ ------------------------------------------------//0c
----- b->l = merge(a, b->l); ---------------------------------//ce
----- b->update(); -------------------------------------------//d2
----- return b; ----------------------------------------------//ff
--- } ------------------------------------------------------//4a
- } ------------------------------------------------------//68
- int pos(node *a){ ------------------------------------------//b7
--- if(a->l == nullptr) return 0; --------------------------//c0
--- return a->l->size; -------------------------------------//42
- } ------------------------------------------------------//8b
- pair<node*, node*> split(node *a, int k){ ----------------//7b
--- if(a == nullptr) return {nullptr, nullptr}; ------------//a4
--- int p = pos(a); ----------------------------------------//1b
--- // if a->key < k ----------------------------------------//f7
--- if(p < k){ ---------------------------------------------//ba
----- auto x = split(a->r, k - (p + 1)); -------------------//2d
----- a->r = x.first; ----------------------------------------//c6
----- a->update(); -------------------------------------------//2a
----- return {a, x.second}; ----------------------------------//51
--- } else{ ------------------------------------------------//ec
----- auto x = split(a->l, k); -------------------------------//50
----- a->l = x.second; ---------------------------------------//20
----- a->update(); -------------------------------------------//4d
----- return {x.first, a}; -----------------------------------//07
--- } ------------------------------------------------------//c0
- } ------------------------------------------------------//ae
- node *insert(node* a, int index, int val){ ----------------//c7
--- auto x = split(a, index); ------------------------------//f0
--- return merge(x.first, merge(new node(val), x.second)); //b0
- } ------------------------------------------------------//f0
- node *erase(node *a, int k){ -------------------------------//57
--- if(a == nullptr) return nullptr; ----------------------//f4
--- int p = pos(a); ----------------------------------------//f1
--- // if a->key == k ----------------------------------------//e1
--- if(p == k){ --------------------------------------------//96
----- node* ans = merge(a->l, a->r); -----------------------//89
----- free(a); -----------------------------------------------//b8
----- return ans; --------------------------------------------//5c
--- } ------------------------------------------------------//b3
--- if(k < p) ----------------------------------------------//6d
----- a->l = erase(a->l, k); ---------------------------------//c6
--- else ---------------------------------------------------//8c
----- a->r = erase(a->r, k - (p + 1)); ---------------------//97
--- a->update(); -------------------------------------------//8e
--- return a; ----------------------------------------------//8f
- } ------------------------------------------------------//87
- //0-based ----------------------------------------------//26
- int get_kth(node *a, int k){ -------------------------------//18
--- int p = pos(a); ----------------------------------------//de
--- if(p == k) ---------------------------------------------//e3
----- return a->val; -----------------------------------------//be
--- if(k < p) ----------------------------------------------//9d
----- return get_kth(a->l, k); -------------------------------//d7
--- else ---------------------------------------------------//c4
----- return get_kth(a->r, k - (p + 1)); -------------------//25
- } ------------------------------------------------------//69
- node *root = nullptr; ----------------------------------//25
public: ------------------------------------------------------//a0
- void insert(int i, int val){ ------------------------------//bf
--- root = insert(root, i, val); ---------------------------//32
- } ------------------------------------------------------//ae
- void erase(int k){ -----------------------------------------//83
--- root = erase(root, k); ---------------------------------//55
- } ------------------------------------------------------//35
- // int height(){ ----------------------------------------//7a
- //  return root != nullptr ? root->h : 0; ----------------//80
- // } ----------------------------------------------------//3b
- int operator[](int i){ -------------------------------------//1d
--- return get_kth(root, i); -------------------------------//15
- } ------------------------------------------------------//13
}; -----------------------------------------------------------//f1
```

7.3. Sparse Table.

```
// Sparse Table ------------------------------------------//61
// O(NlogN) memory, O(1) RMQ -----------------------------//80
 ------------------------------------------------------//bf
struct sparse_table { vvi m; ---------------------------------//f3
- sparse_table(vi arr) { -------------------------------------//8f
--- m.push_back(arr); --------------------------------------//a1
--- for (int k = 0; (1<<(++k)) <= arr.size(); ) { ----------//41
----- m.push_back(vi(arr.size()-(1<<k)+1)); ----------------//38
----- for(int i = 0; i < arr.size() - (1<<k) + 1; i++) ----//2f
------- m[k][i] = min(m[k-1][i], m[k-1][i+(1<<(k-1))]); } }//54
- int query(int l, int r) { ----------------------------------//01
--- int k = 0; while (1<<(k+1) <= r-l+1) k++; --------------//0e
--- return min(m[k][l], m[k][r-(1<<k)+1]); } }; ------------//dc
```

7.4. Segment Tree.

```
/* SegmentTree for Range Sum Queries ---------------------//9c
- and lazy propagation ---------------------------------//f2
- O(N) mermory, O(logN) queries */ ---------------------//ee
 ------------------------------------------------------//b8
struct segtree { ---------------------------------------------//46
- vector<int> v, u; ------------------------------------------//02
- int n; -----------------------------------------------------//ce
- const int NULL_VALUE = 0; ----------------------------------//f2
- segtree(int num) { -----------------------------------------//ea
--- v.resize((1 << (int)(ceil(log2(num))+1))-1, NULL_VALUE);
--- u.resize(v.size(), NULL_VALUE); ------------------------//99
--- n = v.size() / 2; --------------------------------------//49
- } ------------------------------------------------------//05
- segtree(vector<int> &source) : segtree(source.size()) { -//e6
--- for (int i = 0; i < source.size(); i++) ---------------//0b
----- v[i + n] = source[i]; ----------------------------------//e3
--- for (int i = n - 1; i >= 0; i--) ----------------------//a3
----- updateNode(i); -----------------------------------------//29
- } ------------------------------------------------------//db
- int left(int i) { return i*2+1; } ---------------------//56
- int right(int i) { return i*2+2; } --------------------//32
- int parent(int i) { return (i - 1) / 2; } -------------//ee
 ------------------------------------------------------//34
- int mergeValues(int v1, int v2) { ---------------------//76
--- return v1 + v2; ----------------------------------------//1c
- } ------------------------------------------------------//ff
 ------------------------------------------------------//dc
- void updateNode(int i) { -----------------------------------//a0
--- v[i] = mergeValues(v[left(i)], v[right(i)]); ----------//e9
- } ------------------------------------------------------//61
 ------------------------------------------------------//cc
- void lazyUpdate(int i) { -----------------------------------//d5
--- if (u[i] != 0) { ---------------------------------------//26
----- v[i] += u[i]; ------------------------------------------//20
----- if (i < v.size() / 2) { --------------------------------//05
------- u[left(i)] += u[i] / 2; ------------------------------//b7
------- u[right(i)] += u[i] / 2; } ---------------------------//21
----- u[i] = 0; ----------------------------------------------//32
--- } ------------------------------------------------------//79
- } ------------------------------------------------------//72
- int rangeQuery(int i, int l, int r, int a, int b) { -----//5e
--- lazyUpdate(i); -----------------------------------------//7f
--- if (l >= a && r <= b) ----------------------------------//84
----- return v[i]; -------------------------------------------//59
--- if (r < a || l > b) ------------------------------------//b4
----- return NULL_VALUE; -------------------------------------//c6
--- int m = (l + r) / 2; -----------------------------------//d6
--- return mergeValues(rangeQuery(left(i), l, m, a, b), ---//3d
----- rangeQuery(right(i), m + 1, r, a, b)); ---------------//04
- } ------------------------------------------------------//3a
- void updateToRoot(int i) { ----------------------------------//a9
--- updateNode(i); -----------------------------------------//c4
--- if (i != 0) updateToRoot(parent(i)); } ----------------//a3
- void rangeAdd(int i, int a, int b, int l, int r, int off) {
--- lazyUpdate(i); -----------------------------------------//2c
--- if (l >= a && r <= b) { --------------------------------//89
----- u[i] += off * (r - l + 1); -----------------------------//3f
----- lazyUpdate(i); -----------------------------------------//4d
--- } else if (!(r < a || l > b)) { ------------------------//d3
----- int m = (l + r) / 2; -----------------------------------//9b
----- rangeAdd(left(i), a, b, l , m, off); -------------------//c0
----- rangeAdd(right(i), a, b, m + 1, r, off); ---------------//91
----- updateNode(i); -----------------------------------------//e3
--- } ------------------------------------------------------//bd
- } ------------------------------------------------------//2b
- int rangeSum(int a, int b) { --------------------------------//cb
--- return rangeQuery(0, 0, n, a, b); } --------------------//4b
- void update(int index, int value) { ------------------------//ed
--- int i = n + index; -------------------------------------//09
--- v[i] = value; ------------------------------------------//36
--- updateToRoot(parent(i)); } -----------------------------//12
- void rangeUpdate(int a, int b, int offset) { ----------------//16
--- rangeAdd(0, a, b, 0, n, offset); } ---------------------//ed
}; -----------------------------------------------------------//50
```

7.5. Mergesort Tree.

```
/* ----------------------------------------------------//b6
- MergeSort Tree ---------------------------------------//eb
- O(NlogN) memory data structure  ----------------------//c2
- for various 2D queries -------------------------------//58
```

```
- to build: mergetree mt(vector...); ----------------------//8e
*/ ---------------------------------------------------------//0a
vi merge(vi &a, vi &b){ ------------------------------------//61
- vi v; ----------------------------------------------------//31
- int l = 0, r = 0; ----------------------------------------//cd
- v.reserve(a.size() + b.size()); --------------------------//d6
- while(l + r < a.size() + b.size()){ ----------------------//aa
--- if(l == a.size()) ------------------------------------//fb
----- v.push_back(b[r++]); ---------------------------------//d1
--- else if(r == b.size() || a[l] < b[r]) ----------------//0e
----- v.push_back(a[l++]); ---------------------------------//b6
--- else -------------------------------------------------//63
----- v.push_back(b[r++]); ---------------------------------//6e
- } ---------------------------------------------------------//65
- return move(v); -------------------------------------------//66
} ----------------------------------------------------------//a5
 ---------------------------------------------------------//c6
constexpr int right(int i){ --------------------------------//7d
- return i*2 + 2; -------------------------------------------//dc
} ----------------------------------------------------------//74
 ---------------------------------------------------------//fc
constexpr int left(int i){ ---------------------------------//f2
- return i*2 + 1; -------------------------------------------//59
} ----------------------------------------------------------//34
 ---------------------------------------------------------//f7
constexpr int dad(int i){ ----------------------------------//af
- return (i-1)/2; -------------------------------------------//7b
} ----------------------------------------------------------//a0
 ---------------------------------------------------------//11
struct mt_node{ --------------------------------------------//21
- vi v; ----------------------------------------------------//eb
- mt_node(){} ----------------------------------------------//69
- void build(int i, vi &source, vector<mt_node> &tree){ ---//f6
--- int pos = i - source.size() + 1; ---------------------//34
--- if(pos >= 0) -----------------------------------------//59
----- v.push_back(source[pos]); ----------------------------//18
--- else -------------------------------------------------//44
----- v = merge(tree[left(i)].v, tree[right(i)].v); -------//5f
- } ---------------------------------------------------------//18
- // count values = k ------------------------------------//4a
- int count(int k){ ------------------------------------//05
--- // values less than x ----------------------------//20
--- int l = lower_bound(v.begin(), v.end(), k) -----------//f9
----- - v.begin(); ------------------------------------//15
--- // less or equal than x -------------------------//42
--- // > x = v.size() - r ---------------------------//a3
--- int r = upper_bound(v.begin(), v.end(), k) -----------//ba
----- - v.begin(); ------------------------------------//59
--- return r-l; ------------------------------------------//fe
- } ---------------------------------------------------------//ed
- //  ---------------------------------------------------//d5
}; ---------------------------------------------------------//d3
 ---------------------------------------------------------//a6
struct mergetree{ ------------------------------------------//d7
- vector<mt_node> v; ----------------------------------//e0
- mergetree(vi &source){ ------------------------------//38
--- v.resize((1<<(int)(ceil(log2(source.size())+1)))-1); --//26
--- for(int i = source.size()*2 - 2; i >= 0; i--) --------//e7
----- v[i].build(i, source, v); ----------------------------//26
- } ---------------------------------------------------------//c0
- const int nullQuery = 0; ----------------------------//f5
- int qa, qb, qx; --------------------------------------//dc
- int count(int i, int l, int r){ ----------------------//9f
--- if(l >= qa && r <= qb) -------------------------------//ce
----- return v[i].count(qx); -------------------------------//a7
--- if(l > qb || r < qa) ---------------------------------//27
----- return nullQuery; ------------------------------------//08
--- int m = (l+r)/2; -------------------------------------//fd
--- return count(left(i), l, m) + count(right(i), m+1, r); //c8
- } ---------------------------------------------------------//fd
- // count value = x in [a, b] ---------------------------//0d
- int query(int a, int b, int x){ ----------------------//8e
--- qa = a, qb = b, qx = x; ------------------------------//e6
--- int l = 0, r = v.size()/2; ---------------------------//83
--- return count(0, l, r); -------------------------------//de
- } ---------------------------------------------------------//02
}; ---------------------------------------------------------//a8
```

7.6. Max Queue.

```
// Queue with get_max() in O(1) ---------------------------//0b
 ---------------------------------------------------------//09
struct maxqueue{ -------------------------------------------//75
- deque<pair<ll,int>> q; ------------------------------//4c
- int l = 0, r = 0; ----------------------------------------//79
- int size(){ return r - l; } ------------------------------//5a
- void push(ll val){ ----------------------------------------//95
--- while(q.size() && q.back().first <= val) -------------//7a
----- q.pop_back(); ----------------------------------------//6d
--- q.push_back({val, r++}); } ---------------------------//62
- void pop(){ -----------------------------------------------//b5
--- if(q.front().second == l) -----------------------------//d5
----- q.pop_front(); ---------------------------------------//e0
--- l++; } -----------------------------------------------//ec
- ll get_max(){ ---------------------------------------------//90
--- if(!size()) return -1; -------------------------------//11
--- return q.front().first;} -----------------------------//b1
}; ---------------------------------------------------------//72
```

7.7. Kd Tree.

```
// k-D Tree -----------------------------------------------//86
// Insert k dimensional points in ~O(log^k N) -------------//10
// Closest Neighbour in ~O(log^kN) ------------------------//cd
// CAREFUL - can degenerate to O(N) for -------------------//05
// pathological inputs ------------------------------------//e0
 ---------------------------------------------------------//71
#define INC(c) ((c) == K - 1 ? 0 : (c) + 1) ---------------//fc
template <int K> struct kd_tree { --------------------------//e2
- struct pt { -----------------------------------------------//bd
--- double coord[K]; -------------------------------------//99
--- pt() {} ----------------------------------------------//2a
--- pt(double c[K]) { rep(i,0,K) coord[i] = c[i]; } -------//b5
--- double dist(const pt &other) const { -----------------//ff
----- double sum = 0.0; ------------------------------------//78
----- rep(i,0,K) sum += pow(coord[i] - other.coord[i], 2.0);
----- return sqrt(sum); } }; -------------------------------//5f
- struct cmp { ----------------------------------------------//7d
--- int c; -----------------------------------------------//5c
--- cmp(int _c) : c(_c) {} -------------------------------//25
--- bool operator ()(const pt &a, const pt &b) { ---------//db
----- for (int i = 0, cc; i <= K; i++) { -------------------//3f
------- cc = i == 0 ? c : i - 1; -----------------------//4b
------- if (abs(a.coord[cc] - b.coord[cc]) > EPS) -------//5d
--------- return a.coord[cc] < b.coord[cc]; ---------------//1e
----- } ----------------------------------------------------//e0
----- return false; } }; -----------------------------------//be
- struct bb { -----------------------------------------------//b5
--- pt from, to; -----------------------------------------//8e
--- bb(pt _from, pt _to) : from(_from), to(_to) {} -------//9f
--- double dist(const pt &p) { ---------------------------//45
----- double sum = 0.0; ------------------------------------//3a
----- rep(i,0,K) { -----------------------------------------//50
------- if (p.coord[i] < from.coord[i]) -----------------//7e
--------- sum += pow(from.coord[i] - p.coord[i], 2.0); ----//17
------- else if (p.coord[i] > to.coord[i]) ---------------//51
--------- sum += pow(p.coord[i] - to.coord[i], 2.0); ------//c7
----- } ----------------------------------------------------//e9
----- return sqrt(sum); } ----------------------------------//5c
--- bb bound(double l, int c, bool left) { ---------------//c2
----- pt nf(from.coord), nt(to.coord); ---------------------//1c
----- if (left) nt.coord[c] = min(nt.coord[c], l); --------//9a
----- else nf.coord[c] = max(nf.coord[c], l); -------------//91
----- return bb(nf, nt); } }; ------------------------------//41
- struct node { ---------------------------------------------//8e
--- pt p; node *l, *r; -----------------------------------//bc
--- node(pt _p, node *_l, node *_r) ----------------------//1e
----- : p(_p), l(_l), r(_r) {  } }; ------------------------//b0
- node *root; -----------------------------------------------//db
- kd_tree(vector<pt> pts) { ----------------------------//d1
--- root = construct(pts, 0, size(pts) - 1, 0); } --------//46
- node* construct(vector<pt> &pts, int from, int to, int c) {
--- if (from > to) return NULL; --------------------------//7a
--- int mid = from + (to - from) / 2; --------------------//89
--- nth_element(pts.begin() + from, pts.begin() + mid, ----//25
--------- pts.begin() + to + 1, cmp(c)); -----------------//ab
--- return new node(pts[mid], ----------------------------//89
----------- construct(pts, from, mid - 1, INC(c)), --------//dc
----------- construct(pts, mid + 1, to, INC(c))); } -------//d6
- bool contains(const pt &p) { return _con(p, root, 0); } -//7b
- bool _con(const pt &p, node *n, int c) { ------------------//93
--- if (!n) return false; --------------------------------//2c
--- if (cmp(c)(p, n->p)) return _con(p, n->l, INC(c)); ----//8f
--- if (cmp(c)(n->p, p)) return _con(p, n->r, INC(c)); ----//63
--- return true; } ---------------------------------------//5c
- void insert(const pt &p) { _ins(p, root, 0); } ----------//8d
- void _ins(const pt &p, node* &n, int c) { ------------------//1b
--- if (!n) n = new node(p, NULL, NULL); -----------------//5b
--- else if (cmp(c)(p, n->p)) _ins(p, n->l, INC(c)); ------//ec
--- else if (cmp(c)(n->p, p)) _ins(p, n->r, INC(c)); } ----//a4
- void clear() { _clr(root); root = NULL; } ---------------//9e
```

```
- void _clr(node *n) { ------------------------------------//2f
--- if (n) _clr(n->l), _clr(n->r), delete n; } ------------//e8
- pair<pt, bool> nearest_neighbour(const pt &p, ----------//7f
----- bool allow_same=true) { ----------------------------//e0
--- double mn = INFINITY, cs[K]; -----------------------//73
--- rep(i,0,K) cs[i] = -INFINITY; ----------------------//62
--- pt from(cs); ---------------------------------------//0b
--- rep(i,0,K) cs[i] = INFINITY; -----------------------//a4
--- pt to(cs), resp; -----------------------------------//d0
--- _nn(p, root, bb(from, to), mn, resp, 0, allow_same); --//bd
--- return make_pair(resp, !std::isinf(mn)); } ---------//40
- void _nn(const pt &p, node *n, bb b, ---------------//09
----- double &mn, pt &resp, int c, bool same) { ----------//e8
--- if (!n || b.dist(p) > mn) return; ------------------//23
--- bool l1 = true, l2 = false; ------------------------//e7
--- if ((same || p.dist(n->p) > EPS) && p.dist(n->p) < mn) //ff
------- mn = p.dist(resp = n->p); --------------------------//29
--- node *n1 = n->l, *n2 = n->r; -----------------------//12
--- rep(i,0,2) { ---------------------------------------//95
----- if (i == 1 || cmp(c)(n->p, p)) swap(n1,n2),swap(l1,l2);
----- _nn(p, n1, b.bound(n->p.coord[c], c, l1), mn, -------//06
--------- resp, INC(c), same); } } }; ------------------//7e
```

7.8. Fenwick Tree.

```
/* Fenwick Tree (up to 3D) ----------------------------------//9e
- For each dimension: -------------------------------------//77
--- O(N) memory ----------------------------------------//43
--- prefix query in O(logN) ----------------------------//69
--- point update in O(logN) ----------------------------//96
*/ -------------------------------------------------------//15
constexpr int lsb(int n){ --------------------------------//e1
- return n&(-n); -----------------------------------------//d7
} --------------------------------------------------------//3f
// 1-based !!! ---------------------------------------------//80
struct fenwick{ ------------------------------------------//8b
- vi v; --------------------------------------------------//16
- fenwick(int n) : v(n+1){} ------------------------------//62
- fenwick(vi &source) { ----------------------------------//fd
--- v.resize(source.size() + 1); ---------------------------//ed
--- for(int i = 0; i < source.size(); i++){ ----------------//52
----- v[i+1] += source[i]; ------------------------------//7a
----- if(i+1 + lsb(i+1) < v.size()) --------------------//7e
------- v[i+1 + lsb(i+1)] += v[i+1]; ---------------------//41
--- } ------------------------------------------------//a2
- } ------------------------------------------------------//96
- int pf(int i){ ------------------------------------------//09
--- int ans = 0; -------------------------------------------//f6
--- for(; i; i-=lsb(i)) ------------------------------------//7f
----- ans += v[i]; --------------------------------------//8a
--- return ans; --------------------------------------------//89
- } ------------------------------------------------------//24
- int rsq(int l, int r){ ----------------------------------//11
--- return pf(r) - pf(l-1); --------------------------------//b5
- } ------------------------------------------------------//76
- void update(int i, int k){ ------------------------------//da
--- for(; i < v.size(); i += lsb(i)) -----------------------//9c
----- v[i] += k; ----------------------------------------//0a
- } ------------------------------------------------------//6b
}; -------------------------------------------------------//b6
struct fenwick2d{ ----------------------------------------//0c
- vector<fenwick> v; -------------------------------------//15
- fenwick2d(int n, int m) : v(n+1, fenwick(m)){} ---------//c5
- int pf(int r, int c){ ----------------------------------//ba
--- int ans = 0; -------------------------------------------//d5
--- for(int i = r; i > 0; i -= lsb(i)) ---------------------//3f
----- ans += v[i].pf(c); --------------------------------//46
--- return ans; --------------------------------------------//b2
- } ------------------------------------------------------//24
- int rsq(int r1, int c1, int r2, int c2){ ----------------//88
--- return pf(r2, c2) - pf(r1-1, c2) -----------------------//09
----- - pf(r2, c1-1) + pf(c1-1, r1-1); ------------------//16
- } ------------------------------------------------------//28
- void update(int r, int c, int k){ ----------------------//4a
--- for(int i = r; i < v.size(); i += lsb(i)) --------------//65
----- v[i].update(c, k); --------------------------------//2c
- } ------------------------------------------------------//8c
}; -------------------------------------------------------//b9
struct fenwick3d{ ----------------------------------------//aa
- vector<fenwick2d> v; -----------------------------------//a9
- fenwick3d(int x, int y, int z){ ------------------------//8d
--- v.resize(x+1, fenwick2d(y, z)); ------------------------//4b
- } ------------------------------------------------------//9c
- int pf(int x, int y, int z){ ---------------------------//86
--- int ans = 0; -------------------------------------------//79
--- for(int i = x; i > 0; i -= lsb(i)) ---------------------//91
----- ans += v[i].pf(y, z); -----------------------------//33
--- return ans; --------------------------------------------//26
- } ------------------------------------------------------//63
- int rsq(int x1,int y1,int z1,int x2,int y2,int z2){ -----//2d
--- return pf(x2, y2, z2) - pf(x1-1, y2, z2) - -------------//cb
----- pf(x2, y1-1, z2) - pf(x2, y2, z1-1) + ---------------//c8
----- pf(x2, y1-1, z1-1) + pf(x1-1, y2, z1-1) + ----------//2c
----- pf(x1-1, y1-1, z2) - pf(x1-1, y1-1, z1-1); ---------//da
- } ------------------------------------------------------//77
- void update(int x, int y, int z, int k){ ---------------//f5
--- for(int i = x; i < v.size(); i+=lsb(i)) ----------------//80
----- v[i].update(y, z, k); -----------------------------//25
- } ------------------------------------------------------//3d
}; -------------------------------------------------------//f2
```

7.9. Dsu.

```
// UnionFind Data structure ---------------------------------//27
// can be used for Kruskal's MST ----------------------------//f6
struct union_find { --------------------------------------//7a
- vi p; union_find(int n) : p(n, -1) { } -----------------//51
- int find(int x) { return p[x] < 0 ? x : p[x] = find(p[x]); }
- bool unite(int x, int y) { ----------------------------//bd
--- int xp = find(x), yp = find(y); ------------------------//38
--- if (xp == yp) return false; ----------------------------//0f
--- if (p[xp] > p[yp]) swap(xp,yp); ------------------------//18
--- p[xp] += p[yp], p[yp] = xp; ----------------------------//32
--- return true; } -----------------------------------------//14
- int size(int x) { return -p[find(x)]; } }; --------------//d2
```

8. Dp

8.1. Knuth Opt.

```
/* --------------------------------------------------------//b6
- Knuth's DP Optimization ---------------------------------//fc
- Implementation example ----------------------------------//c1
- Recurrence: ---------------------------------------------//7c
-- dp[i][j] = min(i < k < j){dp[i][k] + dp[k][j]} + C[i][j]}
- From O(n^3) to O(n^2) -----------------------------------//bd
- Conditions: ---------------------------------------------//36
--- opt[i][j-1] <= opt[i][j] <= opt[i+1][j], --------------//da
--- where opt[i][j] is smallest optimal k for dp[i][j] ----//62
- What to do: ---------------------------------------------//14
--- we solve smaller subproblems first -------------------//5f
--- then we use the fact that we don't need to try --------//a5
--- all k but that it's enough to try those between -------//9b
--- opt[i][j-1] and opt[i+1][j] ---------------------------//84
--- we magically drop a N in complexity :) ----------------//ca
*/ -------------------------------------------------------//67
/* input..*/ ------------------------------------------------//fe
 --------------------------------------------------------//5d
vi sum(n); -----------------------------------------------//33
partial_sum(v.begin(), v.end(), sum.begin()); ------------//87
 --------------------------------------------------------//e9
vector<vector<ll>> dp(n, vector<ll>(n)); -----------------//4a
// smallest index of optimal solution for dp[l, r] --------//1a
vvi opt(n, vi(n)); ---------------------------------------//bb
for(int s = 1; s <= n; s++){ -----------------------------//b1
- for(int l = 0; l + s - 1 < n; l++){ --------------------//4c
--- int r = l + s - 1; -------------------------------------//42
--- // base cases ------------------------------------------//a2
--- if(s <= 2){ ----------------------------------------//c7
----- dp[l][r] = s == 2 ? sum[r] - (l ? sum[l-1] : 0) : 0; //52
----- opt[l][r] = l; ------------------------------------//85
--- } else{ --------------------------------------------//fc
----- int low = opt[l][r-1]; ----------------------------//ff
----- int hi = opt[l+1][r]; -----------------------------//52
----- int curr = sum[r] - (l ? sum[l-1] : 0); -----------//b0
----- dp[l][r] = LLINF; ---------------------------------//2d
----- // solution must be in this range as\ --------------//05
------- opt[l-1][r] <= opt[l][r] <= opt[l][r+1] ----------//f9
----- for(int m = low; m <= hi; m++){ -------------------//b6
------- ll tmp = curr + dp[l][m] + dp[m+1][r]; ----------//47
------- if(tmp < dp[l][r]){ -------------------------------//86
--------- dp[l][r] = tmp; ---------------------------------//7f
--------- opt[l][r] = m; ----------------------------------//58
------- } -------------------------------------------------//f8
----- } // dp[l][r] is found ------------------------------//e7
--- } ------------------------------------------------------//96
- } --------------------------------------------------------//d4
} ---------------------------------------------------------//02
```

8.2. Dc Opt.

```
/* ---------------------------------------------------//b6
- Divide and Conquer DP Optimization ----------------//ad
- Sadly, no implementation examples -----------------//1b
- Recurrence:  --------------------------------------//92
-- dp[i][j] = min[k < j]{dp[i-1][k] + C[k][j]} -------//f9
- From O(n^2k) to O(nklogn) -------------------------//c8
- Conditions: ---------------------------------------//75
--- opt[i][j] <= opt[i][j+1], -------------------------//99
--- where opt[i][j] is smallest optimal k for dp[i][j] ----//3f
- What to do: ---------------------------------------//c0
--- we compute all dp[i] in [i][l, r] knowing that --------//4e
--- opt value is in k between [optL, optR] ----------//0b
- solve(i, l, r, optL, optR) =  ---------------------//74
--- check l == r ---------------------------------------//77
--- m = (l+r) / 2; -------------------------------------//bf
--- dp(i, m) -------------------------------------------//87
--- opt(i, m) ------------------------------------------//fc
--- // costs optR - optL ---------------------------------//1a
--- // use newly found opt to solve the problem ----------//65
--- solve(i, l, m-1, optL, opt[i][m]) ----------------//74
--- solve(i, m+1, r, opt[i][m], optR) ----------------//52
*/ ---------------------------------------------------//26
```

8.3. Cht.

```
struct cht{ ----------------------------------------------//3d
	struct line{ -----------------------------------------//4b
		mutable ll x1; -----------------------------------//40
		ll m; --------------------------------------------//41
		ll q; --------------------------------------------//cd
		bool query; --------------------------------------//23
		bool operator<(const line& o)const{ --------------//97
			return (query||o.query)?x1<o.x1: -------------//92
				m>o.m; //min cht ---------------------------//1b
				//m<o.m; //max cht ---------------------------//04
		} ------------------------------------------------//ef
	}; ---------------------------------------------------//f2
	ll intersect(const line& a, const line& b){ ----------//35
		if(a.m==b.m)return inf; --------------------------//16
		return (a.q-b.q)/(b.m-a.m); ----------------------//55
	} ----------------------------------------------------//78
	set<line> lines; -------------------------------------//73
	bool useless(set<line>::iterator it){ ----------------//1d
		if(it==lines.begin()||next(it)==lines.end())return 0; ---//e2
		return (intersect(*prev(it),*it)>=intersect(*it,*next(it)));
	} ----------------------------------------------------//96
	void recalc(set<line>::iterator it){ -----------------//3a
		it->x1 = (next(it)==lines.end()?inf:intersect(*it,*next(it)));
		if(it!=lines.begin())prev(it)->x1 = intersect(*prev(it),*it);
	} ----------------------------------------------------//69
	void insert(ll m, ll q){ -----------------------------//96
		line nl = line{-1,m,q,0}; ------------------------//75
		auto it = lines.insert(nl).first; ----------------//80
		if(useless(it)){ ---------------------------------//3b
			lines.erase(it); -----------------------------//96
			return; --------------------------------------//1f
		} ------------------------------------------------//98
		while(next(it)!=lines.end() && useless(next(it)))it=prev(lines.erase(next(it)));
		while(it!=lines.begin() && useless(prev(it)))it=lines.erase(prev(it));
		recalc(it); --------------------------------------//cb
	} ----------------------------------------------------//23
	set<line>::iterator find(ll x){ ----------------------//54
		line query = line{x,0,0,1}; ----------------------//b4
		return lines.lower_bound(query); -----------------//42
	} ----------------------------------------------------//26
	ll at(ll x){ -----------------------------------------//73
		auto t = find(x); --------------------------------//77
		return (t->m)*x+t->q; ----------------------------//b2
	} ----------------------------------------------------//83
}; ---------------------------------------------------------//4e
 ---------------------------------------------------------//bb
```