

## Contents

<b>1 Compile</b>	<b>3</b>	3.4 Edit Distance . . . . .	23
1.1 Compile . . . . .	3	3.5 LCS . . . . .	23
1.2 Template . . . . .	3	3.6 Line Container . . . . .	24
<b>2 Data Structures</b>	<b>3</b>	3.7 Longest Increasing Subsequence . . . . .	24
2.1 BIT . . . . .	3	<b>4 Flow</b>	<b>24</b>
2.2 Bitset . . . . .	3	4.1 Dinic . . . . .	24
2.3 Bit Trie . . . . .	3	4.2 Hopcroft-Karp . . . . .	25
2.4 Disjoint Set Union Bipartite . . . . .	4	4.3 Hungarian . . . . .	26
2.5 Disjoint Set Union . . . . .	4	4.4 Max Flow Min Cost . . . . .	27
2.6 Dynamic Conectivity . . . . .	4	4.5 Max Flow . . . . .	29
2.7 Fenwick Tree . . . . .	6	4.6 Min Cost Max Flow . . . . .	29
2.8 Fenwick Tree 2D . . . . .	7	4.7 Push Relabel . . . . .	31
2.9 Merge Sort Tree . . . . .	7	<b>5 Geometry</b>	<b>31</b>
2.10 Minimum Cartesian Tree . . . . .	7	5.1 Point Struct . . . . .	31
2.11 Multi Ordered Set . . . . .	8	5.2 Sort Points . . . . .	32
2.12 Ordered Set . . . . .	8	<b>6 Graphs</b>	<b>32</b>
2.13 Palindromic Tree . . . . .	9	6.1 2Sat . . . . .	32
2.14 Persistent Array . . . . .	9	6.2 Articulation Points . . . . .	33
2.15 Persistent Segment Tree . . . . .	10	6.3 Bellman-Ford . . . . .	33
2.16 Segment Tree . . . . .	11	6.4 Bipartite Checker . . . . .	34
2.17 Segment Tree 2D . . . . .	11	6.5 Bipartite Maximum Matching . . . . .	34
2.18 Segment Tree Dynamic . . . . .	12	6.6 Block Cut Tree . . . . .	34
2.19 Segment Tree Lazy Types . . . . .	12	6.7 Blossom . . . . .	35
2.20 Segment Tree Lazy . . . . .	13	6.8 Bridges . . . . .	37
2.21 Segment Tree Lazy Range Set . . . . .	14	6.9 Bridges Online . . . . .	37
2.22 Segment Tree Max Subarray Sum . . . . .	15	6.10 Dijkstra . . . . .	39
2.23 Segment Tree Range Update . . . . .	16	6.11 Eulerian Path . . . . .	39
2.24 Segment Tree Struct Types . . . . .	16	6.12 Floyd-Warshall . . . . .	39
2.25 Segment Tree Struct . . . . .	16	6.13 Kruskal . . . . .	39
2.26 Segment Tree Walk . . . . .	17	6.14 Marriage . . . . .	39
2.27 Sparse Table . . . . .	18	6.15 SCC . . . . .	40
2.28 Square Root Decomposition . . . . .	18	<b>7 Linear Algebra</b>	<b>40</b>
2.29 Treap . . . . .	19	7.1 Simplex . . . . .	40
2.30 Treap 2 . . . . .	20	<b>8 Math</b>	<b>42</b>
2.31 Treap With Inversion . . . . .	21	8.1 BinPow . . . . .	42
<b>3 Dynamic Programming</b>	<b>22</b>	8.2 Diophantine . . . . .	42
3.1 CHT Deque . . . . .	22	8.3 Discrete Logarithm . . . . .	43
3.2 Digit DP . . . . .	22	8.4 Divisors . . . . .	44
3.3 Divide and Conquer DP . . . . .	22		

8.5	Euler Totient (Phi) . . . . .	44	12.3	Least Rotation String . . . . .	58
8.6	Fibonacci . . . . .	44	12.4	Manacher . . . . .	58
8.7	Matrix Exponentiation . . . . .	45	12.5	Suffix Array . . . . .	58
8.8	Miller Rabin Deterministic . . . . .	46	12.6	Suffix Automaton . . . . .	60
8.9	Mobius . . . . .	46	12.7	Trie Ahocorasick . . . . .	62
8.10	Prefix Sum Phi . . . . .	46	12.8	Z Function . . . . .	62
8.11	Sieve . . . . .	47			
<b>9</b>	<b>More Topics</b> . . . . .	<b>47</b>	<b>13</b>	<b>Trees</b> . . . . .	<b>63</b>
9.1	2D Prefix Sum . . . . .	47	13.1	Centroid Decomposition . . . . .	63
9.2	Custom Comparators . . . . .	48	13.2	Heavy Light Decomposition . . . . .	64
9.3	Day of the Week . . . . .	48	13.3	Lowest Common Ancestor (LCA) . . . . .	65
9.4	GCD Convolution . . . . .	48	13.4	Tree Diameter . . . . .	66
9.5	int128 . . . . .	48			
9.6	Iterating Over All Subsets . . . . .	49			
9.7	LCM Convolution . . . . .	49			
9.8	Manhattan MST . . . . .	49			
9.9	Mo . . . . .	50			
9.10	MOD INT . . . . .	51			
9.11	Next Permutation . . . . .	51			
9.12	Next and Previous Smaller/Greater Element . . . . .	51			
9.13	Parallel Binary Search . . . . .	52			
9.14	Random Number Generators . . . . .	52			
9.15	setprecision . . . . .	52			
9.16	Ternary Search . . . . .	52			
9.17	Ternary Search Int . . . . .	52			
9.18	XOR Convolution . . . . .	53			
9.19	XOR Basis . . . . .	53			
<b>10</b>	<b>Polynomials</b> . . . . .	<b>53</b>			
10.1	Berlekamp Massey . . . . .	53			
10.2	FFT . . . . .	54			
10.3	NTT . . . . .	55			
10.4	Roots NTT . . . . .	56			
<b>11</b>	<b>Scripts</b> . . . . .	<b>56</b>			
11.1	build.sh . . . . .	56			
11.2	stress.sh . . . . .	56			
11.3	validate.sh . . . . .	57			
<b>12</b>	<b>Strings</b> . . . . .	<b>57</b>			
12.1	Hashed String . . . . .	57			
12.2	KMP . . . . .	57			

# 1 Compile

## 1.1 Compile

```
1 g++-13 nombre.cpp -o nombre (compilar)
2 ./nombre (ejecutar)
```

## 1.2 Template

```
1 #include <bits/stdc++.h>
2 #pragma GCC optimize("O3,unroll-loops")
3 #pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")
4 using namespace std;
5 #define pb push_back
6 #define ll long long
7 #define s second
8 #define f first
9 #define MOD 1000000007
10 #define INF 1000000000000000
11
12 void solve(){
13
14 }
15
16 int main() {
17     ios_base::sync_with_stdio(false); cin.tie(0); cout.tie(0);
18     int t;cin>>t;for(int T=0;T<t;T++)
19         solve();
20 }
```

# 2 Data Structures

## 2.1 BIT

```
1 #define MAXN 10000
2 int bit[MAXN];
3 void update(int x, int val){
4     for(; x < MAXN; x+=x&-x)
5         bit[x] += val;
6 }
7 int get(int x){
8     int ans = 0;
9     for(; x; x-=x&-x)
```

```
10     ans += bit[x];
11     return ans;
12 }
```

## 2.2 Bitset

```
1 bitset<3001> b[3001];
2
3 //set() Set the bit value at the given index to 1.
4 //count() Count the number of set bits.
5 //any() Checks if any bit is set
6 //all() Check if all bit is set.
7 // count the number of set bits in an integer
8
9 #pragma GCC target("popcnt")
10 (int) __builtin_popcount(x);
11 (int) __builtin_popcountll(x);
12 __builtin_clz(x); // count leading zeros
13
14 // declare bitset
15 bitset<64> b;
16
17 // count set bits in bitset
18 b.count();
```

## 2.3 Bit Trie

```
1 const int K = 2;
2 struct Vertex {
3     int next[K];
4
5     Vertex() {
6         fill(begin(next), end(next), -1);
7     }
8 };
9
10
11 //insert
12 for(int j=30;j>=0;j--) {
13     int c = 1&(a[i]>>j);
14     if (trie[v].next[c] == -1) {
15         trie[v].next[c] = trie.size();
16         trie.emplace_back();
17         d.pb(-1);
```

```

18     }
19     v = trie[v].next[c];
20 }

```

## 2.4 Disjoint Set Union Bipartite

```

1 //dsu for checking parity of path length (can be used for checking
  bipartiteness)
2 void make_set(int v) {
3     parent[v] = make_pair(v, 0);
4     rank[v] = 0;
5     bipartite[v] = true;
6 }
7
8 pair<int, int> find_set(int v) {
9     if (v != parent[v].first) {
10         int parity = parent[v].second;
11         parent[v] = find_set(parent[v].first);
12         parent[v].second ^= parity;
13     }
14     return parent[v];
15 }
16
17 void add_edge(int a, int b) {
18     pair<int, int> pa = find_set(a);
19     a = pa.first;
20     int x = pa.second;
21
22     pair<int, int> pb = find_set(b);
23     b = pb.first;
24     int y = pb.second;
25
26     if (a == b) {
27         if (x == y)
28             bipartite[a] = false;
29     } else {
30         if (rank[a] < rank[b])
31             swap(a, b);
32         parent[b] = make_pair(a, x^y^1);
33         bipartite[a] ^= bipartite[b];
34         if (rank[a] == rank[b])
35             ++rank[a];
36     }

```

```

37 }
38
39 bool is_bipartite(int v) {
40     return bipartite[find_set(v).first];
41 }

```

## 2.5 Disjoint Set Union

```

1 struct DSU {
2     vector<int> e;
3     vector<pair<int, int>> st;
4
5     DSU(int N) : e(N, -1) {}
6
7     int get(int x) { return e[x] < 0 ? x : e[x] = get(e[x]); }
8
9     bool connected(int a, int b) { return get(a) == get(b); }
10
11     int size(int x) { return -e[get(x)]; }
12
13     bool unite(int x, int y) {
14         x = get(x), y = get(y);
15         if (x == y) { return false; }
16         if (e[x] > e[y]) { swap(x, y); }
17         st.push_back({x, e[x]});
18         st.push_back({y, e[y]});
19         e[x] += e[y];
20         e[y] = x;
21         return true;
22     }
23
24     //skip if no rollback
25     int time() {return (int)st.size();}
26
27     void rollback(int t) {
28         for (int i = time(); i --> t;)
29             e[st[i].first] = st[i].second;
30         st.resize(t);
31     }
32 };

```

## 2.6 Dynamic Conectivity

```

1 #include <bits/stdc++.h>

```

```

2 using namespace std;
3
4 typedef long long ll;
5
6 struct DSU {
7     vector<int> e;
8     vector<pair<int, int>> st;
9     int cnt;
10
11     DSU(){}
12
13     DSU(int N) : e(N, -1), cnt(N) {}
14
15     int get(int x) { return e[x] < 0 ? x : get(e[x]);}
16
17     bool connected(int a, int b) { return get(a) == get(b); }
18
19     int size(int x) { return -e[get(x)]; }
20
21     bool unite(int x, int y) {
22         x = get(x), y = get(y);
23         if (x == y) { return false; }
24         if (e[x] > e[y]) { swap(x, y); }
25         st.push_back({x, e[x]});
26         st.push_back({y, e[y]});
27         e[x] += e[y];
28         e[y] = x;
29         cnt--;
30         return true;
31     }
32
33     void rollback(){
34         auto [x, y]=st.back();
35         st.pop_back();
36         e[x] = y;
37         auto [a, b]=st.back();
38         st.pop_back();
39         e[a]=b;
40         cnt++;
41     }
42 };
43
44 struct query {

```

```

45     int v, u;
46     bool united;
47     query(int _v, int _u) : v(_v), u(_u) {}
48 };
49
50 struct QueryTree {
51     vector<vector<query>> t;
52     DSU dsu;
53     int T;
54
55     QueryTree(){}
56
57     QueryTree(int _T, int n) : T(_T) {
58         dsu = DSU(n);
59         t.resize(4 * T + 4);
60     }
61
62     void add(int v, int l, int r, int ul, int ur, query& q) {
63         if (ul > ur)
64             return;
65         if (l == ul && r == ur) {
66             t[v].push_back(q);
67             return;
68         }
69         int mid = (l + r) / 2;
70         add(2 * v, l, mid, ul, min(ur, mid), q);
71         add(2 * v + 1, mid + 1, r, max(ul, mid + 1), ur, q);
72     }
73
74     void add_query(query q, int l, int r) {
75         add(1, 0, T - 1, l, r, q);
76     }
77
78     void dfs(int v, int l, int r, vector<int>& ans) {
79         for (query& q : t[v]) {
80             q.united = dsu.unite(q.v, q.u);
81         }
82         if (l == r)
83             ans[l] = dsu.cnt;
84         else {
85             int mid = (l + r) / 2;
86             dfs(2 * v, l, mid, ans);
87             dfs(2 * v + 1, mid + 1, r, ans);

```

```

88     }
89     for (query q : t[v]) {
90         if (q.united)
91             dsu.rollback();
92     }
93 }
94 };
95
96
97 int main(){
98     ios_base::sync_with_stdio(false); cin.tie(NULL);
99     //freopen("connect.in", "r", stdin);
100    //freopen("connect.out", "w", stdout);
101    int n, k; cin >> n >> k;
102    if(k==0) return 0;
103    QueryTree st=QueryTree(k, n);
104    map<pair<int, int>, int> mp;
105    vector<int> ans(k), q;
106    for(int i=0;i<k;i++){
107        char c; cin >> c;
108        if(c=='?'){
109            q.push_back(i);
110            continue;
111        }
112        int u, v; cin >> u >> v;
113        u--; v--;
114        if(u>v) swap(u, v);
115        if(c=='+'){
116            mp[{u, v}]=i;
117        }
118        else{
119            st.add_query(query(u, v), mp[{u, v}], i);
120            mp[{u, v}]=-1;
121        }
122    }
123    for(auto [x, y]:mp){
124        if(y!=-1){
125            st.add_query(query(x.first, x.second), y, k-1);
126        }
127    }
128    st.dfs(1, 0, k-1, ans);
129    for(int x:q){
130        cout << ans[x] << endl;

```

```

131     }
132 }

```

## 2.7 Fenwick Tree

```

1  template <typename T>
2  struct Fenwick {
3      int n;
4      std::vector<T> a;
5
6      Fenwick(int n_ = 0) {
7          init(n_);
8      }
9
10     void init(int n_) {
11         n = n_;
12         a.assign(n, T{});
13     }
14
15     void add(int x, const T &v) {
16         for (int i = x + 1; i <= n; i += i & -i) {
17             a[i - 1] = a[i - 1] + v;
18         }
19     }
20
21     T sum(int x) {
22         T ans{};
23         for (int i = x; i > 0; i -= i & -i) {
24             ans = ans + a[i - 1];
25         }
26         return ans;
27     }
28
29     T rangeSum(int l, int r) {
30         return sum(r) - sum(l);
31     }
32
33     int select(const T &k) {
34         int x = 0;
35         T cur{};
36         for (int i = 1 << std::__lg(n); i; i /= 2) {
37             if (x + i <= n && cur + a[x + i - 1] <= k) {
38                 x += i;

```

```

39         cur = cur + a[x - 1];
40     }
41 }
42 return x;
43 }
44 };

```

## 2.8 Fenwick Tree 2D

```

1 struct Fenwick2D{
2     vector<vector<ll>> b;
3     int n;
4
5     Fenwick2D(int _n) : b(_n+5, vector<ll>(_n+5, 0)), n(_n) {}
6
7     void update(int x, int y, int val){
8         for(; x<=n; x+=(x&-x)){
9             for(int j=y; j<=n; j+=(j&-j)){
10                 b[x][j]+=val;
11             }
12         }
13     }
14
15     ll get(int x, int y){
16         ll ans=0;
17         for(; x; x-=x&-x){
18             for(int j=y; j; j-=j&-j){
19                 ans+=b[x][j];
20             }
21         }
22         return ans;
23     }
24
25     ll get1(int x1, int y1, int x2, int y2){
26         return get(x2, y2)-get(x1-1, y2)-get(x2, y1-1)+ get(x1-1, y1-1);
27     }
28 };
29

```

## 2.9 Merge Sort Tree

```

1 vector<int> t[200005];
2 int a[100005];
3 int n;

```

```

4
5 void build(){
6     for(int i=0;i<n;i++){
7         t[i+n].push_back(a[i]);
8     }
9     for(int i=n-1;i;i--){
10         auto b=t[2*i], c=t[2*i+1];
11         merge(b.begin(), b.end(), c.begin(), c.end(), back_inserter(t[i]));
12     }
13 }
14
15
16 int q(int l, int r, int mid) {
17     int res = 0;
18     for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
19         if (l&1){
20             res+=upper_bound(all(t[l]), mid)-t[l].begin();
21             l++;
22         }
23         if (r&1){
24             r--;
25             res+=upper_bound(all(t[r]), mid)-t[r].begin();
26         }
27     }
28     return res;
29 }

```

## 2.10 Minimum Cartesian Tree

```

1 struct min_cartesian_tree
2 {
3     vector<int> par;
4     vector<vector<int>> sons;
5     int root;
6     void init(int n, vector<int> &arr)
7     {
8         par.assign(n, -1);
9         sons.assign(n, vector<int>(2, -1));
10        stack<int> st;
11        for (int i = 0; i < n; i++)
12        {
13            int last = -1;
14            while (!st.empty() && arr[st.top()] < arr[i])

```

```

15     {
16         last = st.top();
17         st.pop();
18     }
19     if (!st.empty())
20     {
21         par[i] = st.top();
22         sons[st.top()][1] = i;
23     }
24     if (last != -1)
25     {
26         par[last] = i;
27         sons[i][0] = last;
28     }
29     st.push(i);
30 }
31 for (int i = 0; i < n; i++)
32 {
33     if (par[i] == -1)
34     {
35         root = i;
36     }
37 }
38 }
39 };

```

## 2.11 Multi Ordered Set

```

1 #include <ext/pb_ds/assoc_container.hpp>
2 #include <ext/pb_ds/tree_policy.hpp>
3 using namespace __gnu_pbds;
4 template <typename T> using oset = __gnu_pbds::tree<
5     T, __gnu_pbds::null_type, less<T>, __gnu_pbds::rb_tree_tag,
6     __gnu_pbds::tree_order_statistics_node_update
7 >;
8
9 //en main
10
11 oset<pair<int,int>> name;
12 map<int,int> cuenta;
13 function<void(int)> meter = [&] (int val) {
14     name.insert({val,++cuenta[val]});
15 };

```

```

16     auto quitar = [&] (int val) {
17         name.erase({val,cuenta[val]--});
18     };
19
20 meter(x);
21 quitar(y);
22 multiset.order_of_key({y+1,-1})-multiset.order_of_key({x,0})

```

## 2.12 Ordered Set

```

1 #include <ext/pb_ds/assoc_container.hpp>
2 #include <ext/pb_ds/tree_policy.hpp>
3 using namespace __gnu_pbds;
4 template <typename T> using oset = __gnu_pbds::tree<
5     T, __gnu_pbds::null_type, less<T>, __gnu_pbds::rb_tree_tag,
6     __gnu_pbds::tree_order_statistics_node_update
7 >;
8 // order_of_key() primero mayor o igual;
9 // find_by_order() apuntador al elemento k;
10 // oset<pair<int,int>> os;
11 // os.insert({1,2});
12 // os.insert({2,3});
13 // os.insert({5,6});
14 // ll k=os.order_of_key({2,0});
15 // cout<<k<<endl; // 1
16 // pair<int,int> p=os.find_by_order(k);
17 // cout<<p.f<<" "<<p.s<<endl; // 2 3
18 // os.erase(p);
19 // p=os.find_by_order(k);
20 // cout<<p.f<<" "<<p.s<<endl; // 5 6
21
22
23 // check if upperbound or lowerbound does what you want
24 // because they give better time.
25
26 // to allow repetitions
27 #define ordered_set tree<int, null_type,less_equal<int>, rb_tree_tag,
28     tree_order_statistics_node_update>
29
30 // to not allow repetitions
31 #define ordered_set tree<int, null_type,less<int>, rb_tree_tag,
32     tree_order_statistics_node_update>

```



```

32 //order_of_key(x): number of items are strictly smaller than x
33
34 //find_by_order(k) iterator to the kth element

```

## 2.13 Palindromic Tree

```

1  const int N = 3e5 + 9;
2
3  /*
4  -> cnt contains the number of palindromic suffixes of the node
5  */
6  struct PalindromicTree {
7      struct node {
8          int nxt[26], len, st, en, link, cnt, oc;
9      };
10     string s;
11     vector<node> t;
12     int sz, last;
13     PalindromicTree() {}
14     PalindromicTree(string _s) {
15         s = _s;
16         int n = s.size();
17         t.clear();
18         t.resize(n + 9);
19         sz = 2, last = 2;
20         t[1].len = -1, t[1].link = 1;
21         t[2].len = 0, t[2].link = 1;
22     }
23     int extend(int pos) { // returns 1 if it creates a new palindrome
24         int cur = last, curlen = 0;
25         int ch = s[pos] - 'a';
26         while (1) {
27             curlen = t[cur].len;
28             if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos]) break;
29             cur = t[cur].link;
30         }
31         if (t[cur].nxt[ch]) {
32             last = t[cur].nxt[ch];
33             t[last].oc++;
34             return 0;
35         }
36         sz++;
37         last = sz;

```

```

38         t[sz].oc = 1;
39         t[sz].len = t[cur].len + 2;
40         t[cur].nxt[ch] = sz;
41         t[sz].en = pos;
42         t[sz].st = pos - t[sz].len + 1;
43         if (t[sz].len == 1) {
44             t[sz].link = 2;
45             t[sz].cnt = 1;
46             return 1;
47         }
48         while (1) {
49             cur = t[cur].link;
50             curlen = t[cur].len;
51             if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos]) {
52                 t[sz].link = t[cur].nxt[ch];
53                 break;
54             }
55         }
56         t[sz].cnt = 1 + t[t[sz].link].cnt;
57         return 1;
58     }
59     void calc_occurrences() {
60         for (int i = sz; i >= 3; i--) t[t[i].link].oc += t[i].oc;
61     }
62 } t;
63
64 int main() {
65     ios_base::sync_with_stdio(0);
66     cin.tie(0);
67     string s;
68     cin >> s;
69     PalindromicTree t(s);
70     for (int i = 0; i < s.size(); i++) t.extend(i);
71     t.calc_occurrences();
72     long long ans = 0; // number of palindromes
73     for (int i = 3; i <= t.sz; i++) ans += t.t[i].oc;
74     cout << ans << '\n';
75     return 0;
76 }

```

## 2.14 Persistent Array

```

1  struct Node {

```

```

2   int val;
3   Node *l, *r;

4   Node(ll x) : val(x), l(nullptr), r(nullptr) {}
5   Node(Node *ll, Node *rr) : val(0), l(ll), r(rr) {}
6   };
7
8
9   int n, a[100001];    // The initial array and its size
10  Node *roots[100001]; // The persistent array's roots
11
12  Node *build(int l = 0, int r = n - 1) {
13      if (l == r) return new Node(a[l]);
14      int mid = (l + r) / 2;
15      return new Node(build(l, mid), build(mid + 1, r));
16  }
17
18  Node *update(Node *node, int val, int pos, int l = 0, int r = n - 1) {
19      if (l == r) return new Node(val);
20      int mid = (l + r) / 2;
21      if (pos > mid)
22          return new Node(node->l, update(node->r, val, pos, mid + 1, r));
23      else return new Node(update(node->l, val, pos, l, mid), node->r);
24  }
25
26  int query(Node *node, int pos, int l = 0, int r = n - 1) {
27      if (l == r) return node->val;
28      int mid = (l + r) / 2;
29      if (pos > mid) return query(node->r, pos, mid + 1, r);
30      return query(node->l, pos, l, mid);
31  }
32
33  int get_item(int index, int time) {
34      // Gets the array item at a given index and time
35      return query(roots[time], index);
36  }
37
38  void update_item(int index, int value, int prev_time, int curr_time) {
39      // Updates the array item at a given index and time
40      roots[curr_time] = update(roots[prev_time], index, value);
41  }
42
43  void init_arr(int nn, int *init) {
44      // Initializes the persistent array, given an input array

```

```

45   n = nn;
46   for (int i = 0; i < n; i++) a[i] = init[i];
47   roots[0] = build();
48   }

```

## 2.15 Persistent Segment Tree

```

1   struct Node {
2       ll val;
3       Node *l, *r;

4       Node(ll x) : val(x), l(nullptr), r(nullptr) {}
5       Node(Node *_l, Node *_r) {
6           l = _l, r = _r;
7           val = 0;
8           if (l) val += l->val;
9           if (r) val += r->val;
10      }
11      Node(Node *cp) : val(cp->val), l(cp->l), r(cp->r) {}
12  };
13
14
15  int n, sz = 1;
16  ll a[200001];
17  Node *t[200001];
18
19  Node *build(int l = 1, int r = n) {
20      if (l == r) return new Node(a[l]);
21      int mid = (l + r) / 2;
22      return new Node(build(l, mid), build(mid + 1, r));
23  }
24
25  Node *update(Node *node, int pos, int val, int l = 1, int r = n) {
26      if (l == r) return new Node(val);
27      int mid = (l + r) / 2;
28      if (pos > mid)
29          return new Node(node->l, update(node->r, pos, val, mid + 1, r));
30      else return new Node(update(node->l, pos, val, l, mid), node->r);
31  }
32
33  ll query(Node *node, int a, int b, int l = 1, int r = n) {
34      if (l > b || r < a) return 0;
35      if (l >= a && r <= b) return node->val;
36      int mid = (l + r) / 2;

```

```

37     return query(node->l, a, b, l, mid) + query(node->r, a, b, mid + 1, r)
38     ;
39 }
40 int main(){
41     ios_base::sync_with_stdio(false); cin.tie(NULL);
42     int q; cin >> n >> q;
43     for(int i=1;i<=n;i++){
44         cin >> a[i];
45     }
46     t[sz++]=build();
47     while(q--){
48         int ty; cin >> ty;
49         if(ty==1){
50             int k, pos, x; cin >> k >> pos >> x;
51             t[k]=update(t[k], pos, x);
52         }
53         else if(ty==2){
54             int k, l, r; cin >> k >> l >> r;
55             cout << query(t[k], l, r) << endl;
56         }
57         else{
58             int k; cin >> k;
59             t[sz++]=new Node(t[k]);
60         }
61     }
62 }

```

## 2.16 Segment Tree

```

1 struct SegmentTree {
2     vector<ll> a;
3     int n;
4
5     SegmentTree(int _n) : a(2 * _n, 1e18), n(_n) {}
6
7     void update(int pos, ll val) {
8         for (a[pos += n] = val; pos > 1; pos >>= 1) {
9             a[pos / 2] = min(a[pos], a[pos ^ 1]);
10        }
11    }
12
13    ll get(int l, int r) {

```

```

14        ll res = 1e18;
15        for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
16            if (l & 1) {
17                res = min(res, a[l++]);
18            }
19            if (r & 1) {
20                res = min(res, a[--r]);
21            }
22        }
23        return res;
24    }
25 };

```

## 2.17 Segment Tree 2D

```

1 void build_y(int vx, int lx, int rx, int vy, int ly, int ry) {
2     if (ly == ry) {
3         if (lx == rx)
4             t[vx][vy] = a[lx][ly];
5         else
6             t[vx][vy] = t[vx*2][vy] + t[vx*2+1][vy];
7     } else {
8         int my = (ly + ry) / 2;
9         build_y(vx, lx, rx, vy*2, ly, my);
10        build_y(vx, lx, rx, vy*2+1, my+1, ry);
11        t[vx][vy] = t[vx][vy*2] + t[vx][vy*2+1];
12    }
13 }
14
15 void build_x(int vx, int lx, int rx) {
16     if (lx != rx) {
17         int mx = (lx + rx) / 2;
18         build_x(vx*2, lx, mx);
19         build_x(vx*2+1, mx+1, rx);
20     }
21     build_y(vx, lx, rx, 1, 0, m-1);
22 }
23
24 int sum_y(int vx, int vy, int tly, int try_, int ly, int ry) {
25     if (ly > ry)
26         return 0;
27     if (ly == tly && try_ == ry)
28         return t[vx][vy];

```

```

29     int tmy = (tly + try_) / 2;
30     return sum_y(vx, vy*2, tly, tmy, ly, min(ry, tmy))
31         + sum_y(vx, vy*2+1, tmy+1, try_, max(ly, tmy+1), ry);
32 }
33
34 int sum_x(int vx, int tlx, int trx, int lx, int rx, int ly, int ry) {
35     if (lx > rx)
36         return 0;
37     if (lx == tlx && trx == rx)
38         return sum_y(vx, 1, 0, m-1, ly, ry);
39     int tmx = (tlx + trx) / 2;
40     return sum_x(vx*2, tlx, tmx, lx, min(rx, tmx), ly, ry)
41         + sum_x(vx*2+1, tmx+1, trx, max(lx, tmx+1), rx, ly, ry);
42 }
43
44
45 void update_y(int vx, int lx, int rx, int vy, int ly, int ry, int x, int
    y, int new_val) {
46     if (ly == ry) {
47         if (lx == rx)
48             t[vx][vy] = new_val;
49         else
50             t[vx][vy] = t[vx*2][vy] + t[vx*2+1][vy];
51     } else {
52         int my = (ly + ry) / 2;
53         if (y <= my)
54             update_y(vx, lx, rx, vy*2, ly, my, x, y, new_val);
55         else
56             update_y(vx, lx, rx, vy*2+1, my+1, ry, x, y, new_val);
57         t[vx][vy] = t[vx][vy*2] + t[vx][vy*2+1];
58     }
59 }
60
61 void update_x(int vx, int lx, int rx, int x, int y, int new_val) {
62     if (lx != rx) {
63         int mx = (lx + rx) / 2;
64         if (x <= mx)
65             update_x(vx*2, lx, mx, x, y, new_val);
66         else
67             update_x(vx*2+1, mx+1, rx, x, y, new_val);
68     }
69     update_y(vx, lx, rx, 1, 0, m-1, x, y, new_val);
70 }

```

## 2.18 Segment Tree Dynamic

```

1 struct Vertex {
2     int left, right;
3     int sum = 0;
4     Vertex *left_child = nullptr, *right_child = nullptr;
5
6     Vertex(int lb, int rb) {
7         left = lb;
8         right = rb;
9     }
10
11     void extend() {
12         if (!left_child && left + 1 < right) {
13             int t = (left + right) / 2;
14             left_child = new Vertex(left, t);
15             right_child = new Vertex(t, right);
16         }
17     }
18
19     void add(int k, int x) {
20         extend();
21         sum += x;
22         if (left_child) {
23             if (k < left_child->right)
24                 left_child->add(k, x);
25             else
26                 right_child->add(k, x);
27         }
28     }
29
30     int get_sum(int lq, int rq) {
31         if (lq <= left && right <= rq)
32             return sum;
33         if (max(left, lq) >= min(right, rq))
34             return 0;
35         extend();
36         return left_child->get_sum(lq, rq) + right_child->get_sum(lq, rq);
37     }
38 };

```

## 2.19 Segment Tree Lazy Types

```

1 struct max_t {
2     long long val;
3     static const long long null_v = -9223372036854775807LL;
4
5     max_t(): val(0) {}
6     max_t(long long v): val(v) {}
7
8     max_t op(max_t& other) {
9         return max_t(max(val, other.val));
10    }
11
12    max_t lazy_op(max_t& v, int size) {
13        return max_t(val + v.val);
14    }
15 };
16
17 struct min_t {
18     long long val;
19     static const long long null_v = 9223372036854775807LL;
20
21     min_t(): val(0) {}
22     min_t(long long v): val(v) {}
23
24     min_t op(min_t& other) {
25         return min_t(min(val, other.val));
26     }
27
28     min_t lazy_op(min_t& v, int size) {
29         return min_t(val + v.val);
30     }
31 }
32 };
33
34 struct sum_t {
35     long long val;
36     static const long long null_v = 0;
37
38     sum_t(): val(0) {}
39     sum_t(long long v): val(v) {}
40
41     sum_t op(sum_t& other) {
42         return sum_t(val + other.val);
43     }

```

```

44 }
45
46 sum_t lazy_op(sum_t& v, int size) {
47     return sum_t(val + v.val * size);
48 }
49 };

```

## 2.20 Segment Tree Lazy

```

1 template <typename num_t>
2 struct segtree {
3     int n, depth;
4     vector<num_t> tree, lazy;
5
6     void init(int s, long long* arr) {
7         n = s;
8         tree = vector<num_t>(4 * s, 0);
9         lazy = vector<num_t>(4 * s, 0);
10        init(0, 0, n - 1, arr);
11    }
12
13    num_t init(int i, int l, int r, long long* arr) {
14        if (l == r) return tree[i] = arr[l];
15
16        int mid = (l + r) / 2;
17        num_t a = init(2 * i + 1, l, mid, arr),
18              b = init(2 * i + 2, mid + 1, r, arr);
19        return tree[i] = a.op(b);
20    }
21
22    void update(int l, int r, num_t v) {
23        if (l > r) return;
24        update(0, 0, n - 1, l, r, v);
25    }
26
27    num_t update(int i, int tl, int tr, int ql, int qr, num_t v) {
28        eval_lazy(i, tl, tr);
29
30        if (tr < ql || qr < tl) return tree[i];
31        if (ql <= tl && tr <= qr) {
32            lazy[i] = lazy[i].val + v.val;
33            eval_lazy(i, tl, tr);
34            return tree[i];

```

```

35     }
36
37     int mid = (tl + tr) / 2;
38     num_t a = update(2 * i + 1, tl, mid, ql, qr, v),
39               b = update(2 * i + 2, mid + 1, tr, ql, qr, v);
40     return tree[i] = a.op(b);
41 }
42
43 num_t query(int l, int r) {
44     if (l > r) return num_t::null_v;
45     return query(0, 0, n-1, l, r);
46 }
47
48 // int get_first(int v, int tl, int tr, int l, int r, int x) {
49 //     eval_lazy(0, tl, tr);
50 //     if(tl > r || tr < l) return -1;
51 //     if(tree[v].val < x) return -1;
52
53 //     if (tl== tr) return tl;
54
55 //     int tm = tl + (tr-tl)/2;
56 //     int left = get_first(2*v+1, tl, tm, l, r, x);
57 //     if(left != -1) return left;
58 //     return get_first(2*v+2, tm+1, tr, l, r, x);
59 // }
60
61 num_t query(int i, int tl, int tr, int ql, int qr) {
62     eval_lazy(i, tl, tr);
63
64     if (ql <= tl && tr <= qr) return tree[i];
65     if (tr < ql || qr < tl) return num_t::null_v;
66
67     int mid = (tl + tr) / 2;
68     num_t a = query(2 * i + 1, tl, mid, ql, qr),
69             b = query(2 * i + 2, mid + 1, tr, ql, qr);
70     return a.op(b);
71 }
72
73 void eval_lazy(int i, int l, int r) {
74     tree[i] = tree[i].lazy_op(lazy[i], (r - l + 1));
75     if (l != r) {
76         lazy[i * 2 + 1] = lazy[i].val + lazy[i * 2 + 1].val;
77         lazy[i * 2 + 2] = lazy[i].val + lazy[i * 2 + 2].val;

```

```

78     }
79
80     lazy[i] = num_t();
81 }
82 };

```

## 2.21 Segment Tree Lazy Range Set

```

1
2 int N, Q;
3 int a[maxN];
4
5 struct node {
6     ll val;
7     ll lzAdd;
8     ll lzSet;
9     node(){};
10 } tree[maxN << 2];
11
12 #define lc p << 1
13 #define rc (p << 1) + 1
14
15 inline void pushup(int p) {
16     tree[p].val = tree[lc].val + tree[rc].val;
17     return;
18 }
19
20 void pushdown(int p, int l, int mid, int r) {
21     // lazy: range set
22     if (tree[p].lzSet != 0) {
23         tree[lc].lzSet = tree[rc].lzSet = tree[p].lzSet;
24         tree[lc].val = (mid - l + 1) * tree[p].lzSet;
25         tree[rc].val = (r - mid) * tree[p].lzSet;
26         tree[lc].lzAdd = tree[rc].lzAdd = 0;
27         tree[p].lzSet = 0;
28     } else if (tree[p].lzAdd != 0) { // lazy: range add
29         if (tree[lc].lzSet == 0) tree[lc].lzAdd += tree[p].lzAdd;
30         else {
31             tree[lc].lzSet += tree[p].lzAdd;
32             tree[lc].lzAdd = 0;
33         }
34         if (tree[rc].lzSet == 0) tree[rc].lzAdd += tree[p].lzAdd;
35         else {

```

```

36     tree[rc].lzSet += tree[p].lzAdd;
37     tree[rc].lzAdd = 0;
38 }
39 tree[lc].val += (mid - l + 1) * tree[p].lzAdd;
40 tree[rc].val += (r - mid) * tree[p].lzAdd;
41 tree[p].lzAdd = 0;
42 }
43 return;
44 }
45
46 void build(int p, int l, int r) {
47     tree[p].lzAdd = tree[p].lzSet = 0;
48     if (l == r) {
49         tree[p].val = a[l];
50         return;
51     }
52     int mid = (l + r) >> 1;
53     build(lc, l, mid);
54     build(rc, mid + 1, r);
55     pushup(p);
56     return;
57 }
58
59 void add(int p, int l, int r, int a, int b, ll val) {
60     if (a > r || b < l) return;
61     if (a <= l && r <= b) {
62         tree[p].val += (r - l + 1) * val;
63         if (tree[p].lzSet == 0) tree[p].lzAdd += val;
64         else tree[p].lzSet += val;
65         return;
66     }
67     int mid = (l + r) >> 1;
68     pushdown(p, l, mid, r);
69     add(lc, l, mid, a, b, val);
70     add(rc, mid + 1, r, a, b, val);
71     pushup(p);
72     return;
73 }
74
75 void set(int p, int l, int r, int a, int b, ll val) {
76     if (a > r || b < l) return;
77     if (a <= l && r <= b) {
78         tree[p].val = (r - l + 1) * val;

```

```

79     tree[p].lzAdd = 0;
80     tree[p].lzSet = val;
81     return;
82 }
83 int mid = (l + r) >> 1;
84 pushdown(p, l, mid, r);
85 set(lc, l, mid, a, b, val);
86 set(rc, mid + 1, r, a, b, val);
87 pushup(p);
88 return;
89 }
90
91 ll query(int p, int l, int r, int a, int b) {
92     if (a > r || b < l) return 0;
93     if (a <= l && r <= b) return tree[p].val;
94     int mid = (l + r) >> 1;
95     pushdown(p, l, mid, r);
96     return query(lc, l, mid, a, b) + query(rc, mid + 1, r, a, b);
97 }

```

## 2.22 Segment Tree Max Subarray Sum

```

1  const ll inf=1e18;
2
3  struct Node {
4      ll maxi, l_max, r_max, sum;
5
6      Node(ll _maxi, ll _l_max, ll _r_max, ll _sum){
7          maxi=_maxi;
8          l_max=_l_max;
9          r_max=_r_max;
10         sum=_sum;
11     }
12
13     Node operator+(Node b) {
14         return {max(maxi, b.maxi), r_max + b.l_max,
15                 max(l_max, sum + b.l_max), max(b.r_max, r_max + b.sum),
16                 sum + b.sum};
17     }
18
19 };
20
21 struct SegmentTreeMaxSubSum{

```

```

22 int n;
23 vector<Node> t;
24
25 SegmentTreeMaxSubSum(int _n) : n(_n), t(2 * _n, Node(-inf, -inf, -inf,
    -inf)) {}
26
27 void update(int pos, ll val) {
28     t[pos += n] = Node(val, val, val, val);
29     for (pos >>= 1; pos ; pos >>= 1) {
30         t[pos] = t[2*pos] + t[2*pos+1];
31     }
32 }
33
34 Node query(int l, int r) {
35     Node node_l = Node(-inf, -inf, -inf, -inf);
36     Node node_r = Node(-inf, -inf, -inf, -inf);
37     for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
38         if (l & 1) {
39             node_l = node_l + t[l++];
40         }
41         if (r & 1) {
42             node_r = t[--r] + node_r;
43         }
44     }
45     return node_l + node_r;
46 }
47 };

```

## 2.23 Segment Tree Range Update

```

1 struct SegmentTree {
2     vector<ll> a;
3     int n;
4
5     SegmentTree(int _n) : a(2 * _n, 1e18), n(_n) {}
6
7
8     ll get(int pos) {
9         ll res = 1e18;
10        for (pos += n; pos; pos >>= 1) {
11            res = min(res, a[pos]);
12        }
13        return res;

```

```

14    }
15
16    void update(int l, int r, ll val) {
17        for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
18            if (l & 1) {
19                a[l] = min(a[l], val);
20                l++;
21            }
22            if (r & 1) {
23                r--;
24                a[r] = min(a[r], val);
25            }
26        }
27    }
28 };

```

## 2.24 Segment Tree Struct Types

```

1 struct sum_t{
2     ll val;
3     static const long long null_v = 0;
4
5     sum_t(): val(null_v) {}
6     sum_t(long long v): val(v) {}
7
8     sum_t operator + (const sum_t &a) const {
9         sum_t ans;
10        ans.val = val + a.val;
11        return ans;
12    }
13 };
14 // agregar max subarray sum

```

## 2.25 Segment Tree Struct

```

1 // works as a 0-indexed segtree (not lazy)
2 template <typename num_t>
3 struct segtree
4 {
5     int n, k;
6     vector<num_t> tree;
7
8     void init(int s, vector<ll> arr)
9     {

```



```

10     n = s;
11     k = 0;
12     while ((1 << k) < n)
13         k++;
14     tree = vector<num_t>(2 * (1 << k) + 1);
15     for (int i = 0; i < n; i++)
16     {
17         tree[(1 << k) + i] = arr[i];
18     }
19     for (int i = (1 << k) - 1; i > 0; i--)
20     {
21         tree[i] = tree[i * 2] + tree[i * 2 + 1];
22     }
23 }
24
25 void update(int a, ll b)
26 {
27     a += (1 << k);
28     tree[a] = b;
29     for (a /= 2; a >= 1; a /= 2)
30     {
31         tree[a] = tree[a * 2] + tree[a * 2 + 1];
32     }
33 }
34 num_t find(int a, int b)
35 {
36     a += (1 << k);
37     b += (1 << k);
38     num_t s;
39     while (a <= b)
40     {
41         if (a % 2 == 1)
42             s = s + tree[a++];
43         if (b % 2 == 0)
44             s = s + tree[b--];
45         a /= 2;
46         b /= 2;
47     }
48     return s;
49 }
50 };

```

## 2.26 Segment Tree Walk

```

1 struct SegmentTreeWalk {
2     vector<ll> a, final_pos;
3     int n;
4
5     SegmentTreeWalk(int _n) : a(4 * _n, 1e18), final_pos(_n), n(_n) {}
6
7     // l = 0, r = n - 1
8     void build(int l, int r, int node, const vector<ll> &vals) {
9         if (l == r){
10             final_pos[l] = node;
11             a[node] = vals[l];
12         }
13         else {
14             int mid = (l + r) / 2;
15             build(l, mid, node * 2, vals);
16             build(mid + 1, r, node * 2 + 1, vals);
17             a[node] = min(a[node * 2], a[node * 2 + 1]);
18         }
19     }
20
21     void update(int pos, ll val){
22         pos = final_pos[pos];
23         a[pos] = val;
24         pos /= 2;
25         while(pos){
26             a[pos] = min(a[2 * pos], a[2 * pos + 1]);
27             pos /= 2;
28         }
29     }
30
31     //inclusive
32     ll get(int l, int r, int L, int R, int node) {
33         if (L > R)
34             return 1e18;
35         if (l == L && r == R) {
36             return a[node];
37         }
38         int mid = (l + r) / 2;
39         return min(get(l, mid, L, min(R, mid), 2 * node), get(mid + 1, r,
40             max(L, mid + 1), R, 2 * node + 1));
41     }
42 }

```

```

41 // l = 0, r = n - 1, L = query start, R = query end
42 // you can just do ll if you only care about value and not index or no
43 // update
44 pair<ll, ll> query(int l, int r, int L, int R, int node, int val){
45     //cout << l << " " << r << endl;
46     if(l > R || r < L) return {-1, 0};
47     if(a[node] < val) return {-1, 0};
48     if(l == r){
49         // depending on what you want to do
50         return {a[node], l};
51     }
52
53     int mid = (l + r) / 2;
54     auto left = query(l, mid, L, R, 2 * node, val);
55     if(left.first != -1) return left;
56     auto right = query(mid + 1, r, L, R, 2 * node + 1, val);
57     return right;
58 }
59 };

```

## 2.27 Sparse Table

```

1 const int MAXN=100005, K=30;
2 int lg[MAXN+1];
3 int st[K + 1][MAXN];
4
5 int mini(int L, int R){
6     int i = lg[R - L + 1];
7     int minimum = min(st[i][L], st[i][R - (1 << i) + 1]);
8     return minimum;
9 }
10
11 int main(){
12     lg[1]=0;
13     for (int i = 2; i <= MAXN; i++)
14         lg[i] = lg[i/2] + 1;
15     std::copy(a.begin(), a.end(), st[0]);
16
17     for (int i = 1; i <= K; i++)
18         for (int j = 0; j + (1 << i) <= n; j++)
19             st[i][j] = min(st[i - 1][j], st[i - 1][j + (1 << (i - 1))]);
20 }

```

## 2.28 Square Root Decomposition

```

1
2 int n, numBlocks;
3 string s;
4
5 struct Block{
6     int l, r;
7     int sz(){
8         return r-l;
9     }
10 };
11
12 Block blocks[2*MAXI];
13 Block newBlocks[2*MAXI];
14
15 void rebuildDecomp(){
16     string newS=s;
17     int k=0;
18     for(int i=0;i<numBlocks;i++){
19         for(int j=blocks[i].l;j<blocks[i].r;j++){
20             newS[k++]=s[j];
21         }
22     }
23     numBlocks=1;
24     blocks[0]={0, n};
25     s=newS;
26 }
27
28 void cut(int a, int b){
29     int pos=0, curBlock=0;
30     for(int i=0;i<numBlocks;i++){
31         Block B=blocks[i];
32         bool containsA = pos < a && pos + B.sz() > a;
33         bool containsB = pos < b && pos + B.sz() > b;
34         int cutA = B.l + a - pos;
35         int cutB = B.l + b - pos;
36         if(containsA && containsB){
37             newBlocks[curBlock++]={B.l, cutA};
38             newBlocks[curBlock++]={cutA, cutB};
39             newBlocks[curBlock++]={cutB, B.r};
40         }
41         else if(containsA){

```

```

42     newBlocks[curBlock++]={B.l, cutA};
43     newBlocks[curBlock++]={cutA, B.r};
44 }
45 else if(containsB){
46     newBlocks[curBlock++]={B.l, cutB};
47     newBlocks[curBlock++]={cutB, B.r};
48 }
49 else{
50     newBlocks[curBlock++]=B;
51 }
52 pos += B.sz();
53 }
54 pos=0;
55 numBlocks=0;
56 for(int i=0;i<curBlock;i++){
57     if(pos<a || pos>=b){
58         blocks[numBlocks++]=newBlocks[i];
59     }
60     pos+=newBlocks[i].sz();
61 }
62 pos=0;
63 for(int i=0;i<curBlock;i++){
64     if(pos>=a && pos<b){
65         blocks[numBlocks++]=newBlocks[i];
66     }
67     pos+=newBlocks[i].sz();
68 }
69 }
70
71 // while doing operations
72 if(numBlocks>MAXI){
73     rebuildDecomp();
74 }
75
76 // rebuild before final ans
77 rebuildDecomp();
78 cout << ans << endl;

```

## 2.29 Treap

```

1 struct Node {
2     Node *l = 0, *r = 0;
3     int val, y, c = 1;

```

```

4     Node(int val) : val(val), y(rand()) {}
5     void recalc();
6 };
7
8 int cnt(Node* n) { return n ? n->c : 0; }
9 void Node::recalc() { c = cnt(l) + cnt(r) + 1; }
10
11 template<class F> void each(Node* n, F f) {
12     if (n) { each(n->l, f); f(n->val); each(n->r, f); }
13 }
14
15 pair<Node*, Node*> split(Node* n, int k) {
16     if (!n) return {};
17     if (cnt(n->l) >= k) { // "n->val >= k" for lower_bound(k)
18         auto pa = split(n->l, k);
19         n->l = pa.second;
20         n->recalc();
21         return {pa.first, n};
22     } else {
23         auto pa = split(n->r, k - cnt(n->l) - 1); // and just "k"
24         n->r = pa.first;
25         n->recalc();
26         return {n, pa.second};
27     }
28 }
29
30 Node* merge(Node* l, Node* r) {
31     if (!l) return r;
32     if (!r) return l;
33     if (l->y > r->y) {
34         l->r = merge(l->r, r);
35         l->recalc();
36         return l;
37     } else {
38         r->l = merge(l, r->l);
39         r->recalc();
40         return r;
41     }
42 }
43
44 Node* ins(Node* t, Node* n, int pos) {
45     auto pa = split(t, pos);
46     return merge(merge(pa.first, n), pa.second);

```

```

47 }
48
49 // Example application: move the range [l, r) to index k
50 void move(Node*& t, int l, int r, int k) {
51     Node *a, *b, *c;
52     tie(a,b) = split(t, l); tie(b,c) = split(b, r - l);
53     if (k <= l) t = merge(ins(a, b, k), c);
54     else t = merge(a, ins(c, b, k - r));
55 }
56
57 // Usage
58 // create treap
59 // Node* name=nullptr;
60 // insert element
61 // name=ins(name, new Node(val), pos);
62 // Node* x = new Node(val);
63 // name = ins(name, x, pos);
64 // merge two treaps (name before x)
65 // name=merge(name, x);
66 // split treap (this will split treap in two treaps,
67 // first with elements [0, pos) and second with elements [pos, n))
68 // pa will be pair of two treaps
69 // auto pa = split(name, pos);
70 // move range [l, r) to index k
71 // move(name, l, r, k);
72 // iterate over treap
73 // each(name, [&](int val) {
74 //     cout << val << ' ';
75 // });

```

## 2.30 Treap 2

```

1 typedef struct item * pitem;
2 struct item {
3     int prior, value, cnt;
4     bool rev;
5     pitem l, r;
6 };
7
8 int cnt (pitem it) {
9     return it ? it->cnt : 0;
10 }
11

```

```

12 void upd_cnt (pitem it) {
13     if (it)
14         it->cnt = cnt(it->l) + cnt(it->r) + 1;
15 }
16
17 void push (pitem it) {
18     if (it && it->rev) {
19         it->rev = false;
20         swap (it->l, it->r);
21         if (it->l) it->l->rev ^= true;
22         if (it->r) it->r->rev ^= true;
23     }
24 }
25
26 void merge (pitem & t, pitem l, pitem r) {
27     push (l);
28     push (r);
29     if (!l || !r)
30         t = l ? l : r;
31     else if (l->prior > r->prior)
32         merge (l->r, l->r, r), t = l;
33     else
34         merge (r->l, l, r->l), t = r;
35     upd_cnt (t);
36 }
37
38 void split (pitem t, pitem & l, pitem & r, int key, int add = 0) {
39     if (!t)
40         return void( l = r = 0 );
41     push (t);
42     int cur_key = add + cnt(t->l);
43     if (key <= cur_key)
44         split (t->l, l, t->l, key, add), r = t;
45     else
46         split (t->r, t->r, r, key, add + 1 + cnt(t->l)), l = t;
47     upd_cnt (t);
48 }
49
50 void reverse (pitem t, int l, int r) {
51     pitem t1, t2, t3;
52     split (t, t1, t2, l);
53     split (t2, t2, t3, r-l+1);
54     t2->rev ^= true;

```

```

55     merge (t, t1, t2);
56     merge (t, t, t3);
57 }
58
59 void output (pitem t) {
60     if (!t) return;
61     push (t);
62     output (t->l);
63     printf ("%d_", t->value);
64     output (t->r);
65 }

```

## 2.31 Treap With Inversion

```

1 struct Node {
2     Node *l = 0, *r = 0;
3     int val, y, c = 1;
4     bool rev = 0;
5     Node(int val) : val(val), y(rand()) {}
6     void recalc();
7     void push();
8 };
9
10 int cnt(Node* n) { return n ? n->c : 0; }
11 void Node::recalc() { c = cnt(l) + cnt(r) + 1; }
12 void Node::push() {
13     if (rev) {
14         rev = 0;
15         swap(l, r);
16         if (l) l->rev ^= 1;
17         if (r) r->rev ^= 1;
18     }
19 }
20
21 template<class F> void each(Node* n, F f) {
22     if (n) { n->push(); each(n->l, f); f(n->val); each(n->r, f); }
23 }
24
25 pair<Node*, Node*> split(Node* n, int k) {
26     if (!n) return {};
27     n->push();
28     if (cnt(n->l) >= k) {
29         auto pa = split(n->l, k);

```

```

30     n->l = pa.second;
31     n->recalc();
32     return {pa.first, n};
33 } else {
34     auto pa = split(n->r, k - cnt(n->l) - 1);
35     n->r = pa.first;
36     n->recalc();
37     return {n, pa.second};
38 }
39 }
40
41 Node* merge(Node* l, Node* r) {
42     if (!l) return r;
43     if (!r) return l;
44     l->push();
45     r->push();
46     if (l->y > r->y) {
47         l->r = merge(l->r, r);
48         l->recalc();
49         return l;
50     } else {
51         r->l = merge(l, r->l);
52         r->recalc();
53         return r;
54     }
55 }
56
57 Node* ins(Node* t, Node* n, int pos) {
58     auto pa = split(t, pos);
59     return merge(merge(pa.first, n), pa.second);
60 }
61
62 // Example application: reverse the range [l, r]
63 void reverse(Node*& t, int l, int r) {
64     Node *a, *b, *c;
65     tie(a,b) = split(t, l);
66     tie(b,c) = split(b, r - l + 1);
67     b->rev ^= 1;
68     t = merge(merge(a, b), c);
69 }
70
71 void move(Node*& t, int l, int r, int k) {
72     Node *a, *b, *c;

```

```

73 tie(a,b) = split(t, l);
74 tie(b,c) = split(b, r - l);
75 if (k <= l) t = merge(ins(a, b, k), c);
76 else t = merge(a, ins(c, b, k - r));
77 }

```

## 3 Dynamic Programming

### 3.1 CHT Deque

```

1 // needs fixing
2
3 struct line {
4     ll a, b;
5     line(ll A, ll B) : a(A), b(B) {}
6     double intersect(const line &line1) const {
7         return 1.0 * (line1.b - b) / (a - line1.a);
8     }
9     ll eval(ll x) {
10         return a * x + b;
11     }
12 };
13
14 // this finds the minimum and slope in increasing
15 deque<line> l[p+1];
16 l[0].push_front({-1, -s[1]});
17 for(int i=1; i<=m; i++){
18     for(int j=p; j>0; j--){
19         if(j>i) continue;
20         while((int)l[j-1].size()>=2 && l[j-1].back().eval(a[i])>=l[j-1][1][(int)l[j-1].size()-2].eval(a[i])){
21             l[j-1].pop_back();
22         }
23         dp[i][j]=l[j-1].back().eval(a[i])+(a[i]*(i))+s[i];
24         line cur(-i-1, dp[i][j]-s[i+1]);
25         while((int)l[j].size()>=2 && cur.intersect(l[j][1])<=l[j][0].intersect(l[j][1])){
26             l[j].pop_front();
27         }
28         l[j].push_front(cur);
29     }
30 }

```

### 3.2 Digit DP

```

1 vector<int> num;
2 ll DP[20][20][2][2];
3
4 ll g(int pos, int last, int f, int z){
5
6     if(pos == num.size()){
7         return 1;
8     }
9
10    if(DP[pos][last][f][z] != -1) return DP[pos][last][f][z];
11    ll res = 0;
12
13    int l=(f ? 9 : num[pos]);
14
15    for(int dgt = 0; dgt<=l; dgt++){
16        if(dgt==last && !(dgt==0 && z==1)) continue;
17        int nf = f;
18        if(f == 0 && dgt < l) nf = 1;
19        if(z && !dgt) res+=g(pos+1, dgt, nf, 1);
20        else res += g(pos+1, dgt, nf, 0);
21    }
22    DP[pos][last][f][z]=res;
23    return res;
24 }
25
26 ll solve(ll x){
27     num.clear();
28     if(x==-1) return 0;
29     memset(DP, -1, sizeof(DP));
30     while(x>0){
31         num.pb(x%10);
32         x/=10;
33     }
34     reverse(all(num));
35     return g(0, 0, 0, 1);
36 }

```

### 3.3 Divide and Conquer DP

```

1 int m, n;
2 vector<long long> dp_before, dp_cur;

```

```

3
4 long long C(int i, int j);
5
6 // compute dp_cur[l], ... dp_cur[r] (inclusive)
7 void compute(int l, int r, int optl, int optr) {
8     if (l > r)
9         return;
10
11     int mid = (l + r) >> 1;
12     pair<long long, int> best = {LLONG_MAX, -1};
13
14     for (int k = optl; k <= min(mid, optr); k++) {
15         best = min(best, {(k ? dp_before[k - 1] : 0) + C(k, mid), k});
16     }
17
18     dp_cur[mid] = best.first;
19     int opt = best.second;
20
21     compute(l, mid - 1, optl, opt);
22     compute(mid + 1, r, opt, optr);
23 }
24
25 long long solve() {
26     dp_before.assign(n, 0);
27     dp_cur.assign(n, 0);
28
29     for (int i = 0; i < n; i++)
30         dp_before[i] = C(0, i);
31
32     for (int i = 1; i < m; i++) {
33         compute(0, n - 1, 0, n - 1);
34         dp_before = dp_cur;
35     }
36
37     return dp_before[n - 1];
38 }

```

### 3.4 Edit Distance

```

1 string s, t; cin >> s >> t;
2 int n=s.length(), m=t.length();
3 for (int i=0; i<=n; i++){
4     fill(dp[i], dp[i]+m+1, 1e9);

```

```

5 }
6 dp[0][0]=0;
7 for (int i=0; i<=n; i++){
8     for (int j=0; j<=m; j++){
9         if(j){
10             dp[i][j]=min(dp[i][j], dp[i][j-1]+1);
11         }
12         if(i){
13             dp[i][j]=min(dp[i][j], dp[i-1][j]+1);
14         }
15         if(i && j){
16             int a=(s[i-1]!=t[j-1] ? 1:0);
17             dp[i][j]=min(dp[i][j], dp[i-1][j-1]+a);
18         }
19     }
20 }

```

### 3.5 LCS

```

1 string s, t; cin >> s >> t;
2 int n=s.length(), m=t.length();
3 int dp[n+1][m+1];
4 memset(dp, 0, sizeof(dp));
5 for(int i=1; i<=n; i++){
6     for(int j=1; j<=m; j++){
7         dp[i][j]=max(dp[i-1][j], dp[i][j-1]);
8         if(s[i-1]==t[j-1]){
9             dp[i][j]=dp[i-1][j-1]+1;
10        }
11    }
12 }
13 int i=n, j=m;
14 string ans="";
15 while(i && j){
16     if(s[i-1]==t[j-1]){
17         ans+=s[i-1];
18         i--; j--;
19     }
20     else if(dp[i][j-1]>=dp[i-1][j]){
21         j--;
22     }
23     else{
24         i--;

```

```

25     }
26 }
27 reverse(all(ans));
28 cout << ans << endl;

```

### 3.6 Line Container

```

1 //Queries for maximum point x. To change this modify first comparator.
2 struct Line {
3     mutable ll k, m, p;
4     bool operator<(const Line& o) const { return k < o.k; }
5     bool operator<(ll x) const { return p < x; }
6 };
7
8 struct LineContainer : multiset<Line, less<>> {
9     // (for doubles, use inf = 1/.0, div(a,b) = a/b)
10    static const ll inf = LLONG_MAX;
11    ll div(ll a, ll b) { // floored division
12        return a / b - ((a ^ b) < 0 && a % b); }
13    bool isect(iterator x, iterator y) {
14        if (y == end()) return x->p = inf, 0;
15        if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
16        else x->p = div(y->m - x->m, x->k - y->k);
17        return x->p >= y->p;
18    }
19    void add(ll k, ll m) {
20        auto z = insert({k, m, 0}), y = z++, x = y;
21        while (isect(y, z)) z = erase(z);
22        if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
23        while ((y = x) != begin() && (--x)->p >= y->p)
24            isect(x, erase(y));
25    }
26    ll query(ll x) {
27        assert(!empty());
28        auto l = *lower_bound(x);
29        return l.k * x + l.m;
30    }
31 };

```

### 3.7 Longest Increasing Subsequence

```

1 vector<int> dp;
2 for (int i=0;i<n;i++){
3     auto it=lower_bound(dp.begin(), dp.end(), v[i]);

```

```

4     if(it==dp.end()){
5         dp.push_back(v[i]);
6     }
7     else{
8         int pos=it-dp.begin();
9         dp[pos]=v[i];
10    }
11 }
12 cout << dp.size() << endl;

```

## 4 Flow

### 4.1 Dinic

```

1 // Si en el grafo todos los vertices distintos
2 // de s y t cumplen que solo tienen una arista
3 // de entrada o una de salida la y dicha arista
4 // tiene capacidad 1 entonces la complejidad es
5 // O(E sqrt(v))
6
7 // si todas las aristas tienen capacidad 1
8 // el algoritmo tiene complejidad O(E sqrt(E))
9
10 struct FlowEdge {
11     int v, u;
12     long long cap, flow = 0;
13     FlowEdge(int v, int u, long long cap) : v(v), u(u), cap(cap) {}
14 };
15
16 struct Dinic {
17     const long long flow_inf = 1e18;
18     vector<FlowEdge> edges;
19     vector<vector<int>> adj;
20     int n, m = 0;
21     int s, t;
22     vector<int> level, ptr;
23     queue<int> q;
24
25     Dinic(int n, int s, int t) : n(n), s(s), t(t) {
26         adj.resize(n);
27         level.resize(n);
28         ptr.resize(n);
29     }

```



```

30
31 void add_edge(int v, int u, long long cap) {
32     edges.emplace_back(v, u, cap);
33     edges.emplace_back(u, v, 0);
34     adj[v].push_back(m);
35     adj[u].push_back(m + 1);
36     m += 2;
37 }
38
39 bool bfs() {
40     while (!q.empty()) {
41         int v = q.front();
42         q.pop();
43         for (int id : adj[v]) {
44             if (edges[id].cap - edges[id].flow < 1)
45                 continue;
46             if (level[edges[id].u] != -1)
47                 continue;
48             level[edges[id].u] = level[v] + 1;
49             q.push(edges[id].u);
50         }
51     }
52     return level[t] != -1;
53 }
54
55 long long dfs(int v, long long pushed) {
56     if (pushed == 0)
57         return 0;
58     if (v == t)
59         return pushed;
60     for (int& cid = ptr[v]; cid < (int)adj[v].size(); cid++) {
61         int id = adj[v][cid];
62         int u = edges[id].u;
63         if (level[v] + 1 != level[u] || edges[id].cap - edges[id].
64             flow < 1)
65             continue;
66         long long tr = dfs(u, min(pushed, edges[id].cap - edges[id].
67             flow));
68         if (tr == 0)
69             continue;
70         edges[id].flow += tr;
71         edges[id ^ 1].flow -= tr;
72         return tr;

```

```

71     }
72     return 0;
73 }
74
75 long long flow() {
76     long long f = 0;
77     while (true) {
78         fill(level.begin(), level.end(), -1);
79         level[s] = 0;
80         q.push(s);
81         if (!bfs())
82             break;
83         fill(ptr.begin(), ptr.end(), 0);
84         while (long long pushed = dfs(s, flow_inf)) {
85             f += pushed;
86         }
87     }
88     return f;
89 }
90 };

```

## 4.2 Hopcroft-Karp

```

1 // maximum matching in bipartite graph
2 vector<int> match, dist;
3 vector<vector<int>>> g;
4 int n, m, k;
5 bool bfs()
6 {
7     queue<int> q;
8     // The alternating path starts with unmatched nodes
9     for (int node = 1; node <= n; node++)
10     {
11         if (!match[node])
12         {
13             q.push(node);
14             dist[node] = 0;
15         }
16         else
17         {
18             dist[node] = INF;
19         }
20     }

```

```

21
22 dist[0] = INF;
23
24 while (!q.empty())
25 {
26     int node = q.front();
27     q.pop();
28     if (dist[node] >= dist[0])
29     {
30         continue;
31     }
32     for (int son : g[node])
33     {
34         // If the match of son is matched
35         if (dist[match[son]] == INF)
36         {
37             dist[match[son]] = dist[node] + 1;
38             q.push(match[son]);
39         }
40     }
41 }
42 // Returns true if an alternating path has been found
43 return dist[0] != INF;
44 }
45
46 // Returns true if an augmenting path has been found starting from
47 // vertex node
48 bool dfs(int node)
49 {
50     if (node == 0)
51     {
52         return true;
53     }
54     for (int son : g[node])
55     {
56         if (dist[match[son]] == dist[node] + 1 && dfs(match[son]))
57         {
58             match[node] = son;
59             match[son] = node;
60             return true;
61         }
62     }
63     dist[node] = INF;

```

```

63     return false;
64 }
65
66 int hopcroft_karp()
67 {
68     int cnt = 0;
69     // While there is an alternating path
70     while (bfs())
71     {
72         for (int node = 1; node <= n; node++)
73         {
74             // If node is unmatched but we can match it using an augmenting
75             // path
76             if (!match[node] && dfs(node))
77             {
78                 cnt++;
79             }
80         }
81     }
82     return cnt;
83 }
84 // usage
85 // n numero de puntos en la izquierda
86 // m numero de puntos en la derecha
87 // las aristas se guardan en g
88 // los puntos estan 1 indexados
89 // el punto 1 de m es el punto n+1 de g
90 // hopcroft_karp() devuelve el tamaño del máximo matching
91 // match contiene el match de cada punto
92 // si match de i es 0, entonces i no está matcheado
93 // https://judge.yosupo.jp/submission/247277

```

### 4.3 Hungarian

```

1 #define forn(i,n) for(int i=0;i<int(n);++i)
2 #define forsn(i,s,n) for(int i=s;i<int(n);++i)
3 #define forall(i,c) for(typeof(c).begin() i=c.begin();i!=c.end();++i)
4 #define DBG(X) cerr << #X << " = " << X << endl;
5 typedef vector<int> vint;
6 typedef vector<vint> vvint;
7
8 void showmt();

```

```

9
10 /* begin notebook */
11
12 #define MAXN 256
13 #define INFTO 0x7f7f7f7f
14 int n;
15 int mt[MAXN][MAXN]; // Matriz de costos (X * Y)
16 int xy[MAXN], yx[MAXN]; // Matching resultante (X->Y, Y->X)
17
18 int lx[MAXN], ly[MAXN], slk[MAXN], slkx[MAXN], prv[MAXN];
19 char S[MAXN], T[MAXN];
20
21 void updtree(int x) {
22     forn(y, n) if (lx[x] + ly[y] - mt[x][y] < slk[y]) {
23         slk[y] = lx[x] + ly[y] - mt[x][y];
24         slkx[y] = x;
25     }
26 }
27 int hungar() {
28     forn(i, n) {
29         ly[i] = 0;
30         lx[i] = *max_element(mt[i], mt[i]+n);
31     }
32     memset(xy, -1, sizeof(xy));
33     memset(yx, -1, sizeof(yx));
34
35     forn(m, n) {
36         memset(S, 0, sizeof(S));
37         memset(T, 0, sizeof(T));
38         memset(prv, -1, sizeof(prv));
39         memset(slk, 0x7f, sizeof(slk));
40         queue<int> q;
41         #define bpone(e, p) { q.push(e); prv[e] = p; S[e] = 1; updtree(e); }
42         forn(i, n) if (xy[i] == -1) { bpone(i, -2); break; }
43
44         int x=0, y=-1;
45         while (y== -1) {
46             while (!q.empty() && y== -1) {
47                 x = q.front(); q.pop();
48                 forn(j, n) if (mt[x][j] == lx[x] + ly[j] && !T[j]) {
49                     if (yx[j] == -1) { y = j; break; }
50                     T[j] = 1;
51                     bpone(yx[j], x);

```

```

52     }
53     }
54     if (y!= -1) break;
55     int dlt = INFTO;
56     forn(j, n) if (!T[j]) dlt = min(dlt, slk[j]);
57     forn(k, n) {
58         if (S[k]) lx[k] -= dlt;
59         if (T[k]) ly[k] += dlt;
60         if (!T[k]) slk[k] -= dlt;
61     }
62     // q = queue<int>();
63     forn(j, n) if (!T[j] && !slk[j]) {
64         if (yx[j] == -1) {
65             x = slkx[j]; y = j; break;
66         } else {
67             T[j] = 1;
68             if (!S[yx[j]]) bpone(yx[j], slkx[j]);
69         }
70     }
71 }
72 if (y!= -1) {
73     for(int p = x; p != -2; p = prv[p]) {
74         yx[y] = p;
75         int ty = xy[p]; xy[p] = y; y = ty;
76     }
77 } else break;
78 }
79 int res = 0;
80 forn(i, n) res += mt[i][xy[i]];
81 return res;
82 }

```

#### 4.4 Max Flow Min Cost

```

1 // dado un acomodo de flujos con costos
2 // devuelve el costo minimo para un flujo especificado
3
4 struct Edge
5 {
6     int from, to, capacity, cost;
7     Edge(int _from, int _to, int _capacity, int _cost)
8     {
9         from = _from;

```

```

10     to = _to;
11     capacity = _capacity;
12     cost = _cost;
13 }
14 };
15
16 vector<vector<int>> adj, cost, capacity;
17
18 const int INF = 1e9;
19
20 void shortest_paths(int n, int v0, vector<int> &d, vector<int> &p)
21 {
22     d.assign(n, INF);
23     d[v0] = 0;
24     vector<bool> inq(n, false);
25     queue<int> q;
26     q.push(v0);
27     p.assign(n, -1);
28
29     while (!q.empty())
30     {
31         int u = q.front();
32         q.pop();
33         inq[u] = false;
34         for (int v : adj[u])
35         {
36             if (capacity[u][v] > 0 && d[v] > d[u] + cost[u][v])
37             {
38                 d[v] = d[u] + cost[u][v];
39                 p[v] = u;
40                 if (!inq[v])
41                 {
42                     inq[v] = true;
43                     q.push(v);
44                 }
45             }
46         }
47     }
48 }
49
50 int min_cost_flow(int N, vector<Edge> edges, int K, int s, int t)
51 {
52     adj.assign(N, vector<int>());

```

```

53     cost.assign(N, vector<int>(N, 0));
54     capacity.assign(N, vector<int>(N, 0));
55     for (Edge e : edges)
56     {
57         adj[e.from].push_back(e.to);
58         adj[e.to].push_back(e.from);
59         cost[e.from][e.to] = e.cost;
60         cost[e.to][e.from] = -e.cost;
61         capacity[e.from][e.to] = e.capacity;
62     }
63
64     int flow = 0;
65     int cost = 0;
66     vector<int> d, p;
67     while (flow < K)
68     {
69         shortest_paths(N, s, d, p);
70         if (d[t] == INF)
71             break;
72
73         // find max flow on that path
74         int f = K - flow;
75         int cur = t;
76         while (cur != s)
77         {
78             f = min(f, capacity[p[cur]][cur]);
79             cur = p[cur];
80         }
81
82         // apply flow
83         flow += f;
84         cost += f * d[t];
85         cur = t;
86         while (cur != s)
87         {
88             capacity[p[cur]][cur] -= f;
89             capacity[cur][p[cur]] += f;
90             cur = p[cur];
91         }
92     }
93
94     if (flow < K)
95         return -1;

```

```

96     else
97         return cost;
98 }

```

## 4.5 Max Flow

```

1  long long max_flow(vector<vector<int>> adj, vector<vector<long long>>
    capacity,
2      int source, int sink)
3  {
4      int n = adj.size();
5      vector<int> parent(n, -1);
6      // Find a way from the source to sink on a path with non-negative
    capacities
7      auto reachable = [&]() -> bool
8      {
9          queue<int> q;
10         q.push(source);
11         while (!q.empty())
12         {
13             int node = q.front();
14             q.pop();
15             for (int son : adj[node])
16             {
17                 long long w = capacity[node][son];
18                 if (w <= 0 || parent[son] != -1)
19                     continue;
20                 parent[son] = node;
21                 q.push(son);
22             }
23         }
24         return parent[sink] != -1;
25     };
26
27     long long flow = 0;
28     // While there is a way from source to sink with non-negative
    capacities
29     while (reachable())
30     {
31         int node = sink;
32         // The minimum capacity on the path from source to sink
33         long long curr_flow = LLONG_MAX;
34         while (node != source)

```

```

35     {
36         curr_flow = min(curr_flow, capacity[parent[node]][node]);
37         node = parent[node];
38     }
39     node = sink;
40     while (node != source)
41     {
42         // Subtract the capacity from capacity edges
43         capacity[parent[node]][node] -= curr_flow;
44         // Add the current flow to flow backedges
45         capacity[node][parent[node]] += curr_flow;
46         node = parent[node];
47     }
48     flow += curr_flow;
49     fill(parent.begin(), parent.end(), -1);
50 }
51
52 return flow;
53 }
54
55
56
57 //vector<vector<long long>> capacity(n, vector<long long>(n));
58 //vector<vector<int>> adj(n);
59 //adj[a].push_back(b);
60 //adj[b].push_back(a);
61 //capacity[a][b] += c;

```

## 4.6 Min Cost Max Flow

```

1  /**
2   * If costs can be negative, call setpi before maxflow, but note that
    negative cost cycles are not supported.
3   * To obtain the actual flow, look at positive values only
4   * Time:  $O(F \cdot E \cdot \log(V))$  where F is max flow.  $O(VE)$  for setpi.
5   */
6  #include <bits/stdc++.h>
7  using namespace std;
8
9  #include <ext/pb_ds/priority_queue.hpp>
10 using namespace __gnu_pbds;
11
12 #define rep(i, a, b) for(int i = a; i < (b); ++i)

```

```

13 #define all(x) begin(x), end(x)
14 #define sz(x) (int)(x).size()
15 typedef long long ll;
16 typedef pair<int, int> pii;
17 typedef vector<int> vi;
18
19 #pragma once
20
21 // #include <bits/extc++.h> /// include-line, keep-include
22
23 const ll INF = numeric_limits<ll>::max() / 4;
24
25 struct MCMF {
26     struct edge {
27         int from, to, rev;
28         ll cap, cost, flow;
29     };
30     int N;
31     vector<vector<edge>> ed;
32     vi seen;
33     vector<ll> dist, pi;
34     vector<edge*> par;
35
36     MCMF(int N) : N(N), ed(N), seen(N), dist(N), pi(N), par(N) {}
37
38     void addEdge(int from, int to, ll cap, ll cost) {
39         if (from == to) return;
40         ed[from].push_back(edge{ from, to, sz(ed[to]), cap, cost, 0 });
41         ed[to].push_back(edge{ to, from, sz(ed[from])-1, 0, -cost, 0 });
42     }
43
44     void path(int s) {
45         fill(all(seen), 0);
46         fill(all(dist), INF);
47         dist[s] = 0; ll di;
48
49         __gnu_pbds::priority_queue<pair<ll, int>> q;
50         vector<decltype(q)::point_iterator> its(N);
51         q.push({ 0, s });
52
53         while (!q.empty()) {
54             s = q.top().second; q.pop();
55             seen[s] = 1; di = dist[s] + pi[s];

```

```

56         for (edge& e : ed[s]) if (!seen[e.to]) {
57             ll val = di - pi[e.to] + e.cost;
58             if (e.cap - e.flow > 0 && val < dist[e.to]) {
59                 dist[e.to] = val;
60                 par[e.to] = &e;
61                 if (its[e.to] == q.end())
62                     its[e.to] = q.push({ -dist[e.to], e.to });
63                 else
64                     q.modify(its[e.to], { -dist[e.to], e.to });
65             }
66         }
67     }
68     rep(i, 0, N) pi[i] = min(pi[i] + dist[i], INF);
69 }
70
71 pair<ll, ll> maxflow(int s, int t) {
72     ll totflow = 0, totcost = 0;
73     while (path(s), seen[t]) {
74         ll fl = INF;
75         for (edge* x = par[t]; x; x = par[x->from])
76             fl = min(fl, x->cap - x->flow);
77
78         totflow += fl;
79         for (edge* x = par[t]; x; x = par[x->from]) {
80             x->flow += fl;
81             ed[x->to][x->rev].flow -= fl;
82         }
83     }
84     rep(i, 0, N) for (edge& e : ed[i]) totcost += e.cost * e.flow;
85     return {totflow, totcost/2};
86 }
87
88 // If some costs can be negative, call this before maxflow:
89 void setpi(int s) { // (otherwise, leave this out)
90     fill(all(pi), INF); pi[s] = 0;
91     int it = N, ch = 1; ll v;
92     while (ch-- && it--)
93         rep(i, 0, N) if (pi[i] != INF)
94             for (edge& e : ed[i]) if (e.cap)
95                 if ((v = pi[i] + e.cost) < pi[e.to])
96                     pi[e.to] = v, ch = 1;
97     assert(it >= 0); // negative cost cycle
98 }

```

```
99 };
```

## 4.7 Push Relabel

```
1  const int inf = 1000000000;
2
3  int n;
4  vector<vector<int>> capacity, flow;
5  vector<int> height, excess, seen;
6  queue<int> excess_vertices;
7
8  void push(int u, int v) {
9      int d = min(excess[u], capacity[u][v] - flow[u][v]);
10     flow[u][v] += d;
11     flow[v][u] -= d;
12     excess[u] -= d;
13     excess[v] += d;
14     if (d && excess[v] == d)
15         excess_vertices.push(v);
16 }
17
18 void relabel(int u) {
19     int d = inf;
20     for (int i = 0; i < n; i++) {
21         if (capacity[u][i] - flow[u][i] > 0)
22             d = min(d, height[i]);
23     }
24     if (d < inf)
25         height[u] = d + 1;
26 }
27
28 void discharge(int u) {
29     while (excess[u] > 0) {
30         if (seen[u] < n) {
31             int v = seen[u];
32             if (capacity[u][v] - flow[u][v] > 0 && height[u] > height[v])
33                 push(u, v);
34             else
35                 seen[u]++;
36         } else {
37             relabel(u);
38             seen[u] = 0;
39         }
```

```
40     }
41 }
42
43 int max_flow(int s, int t) {
44     height.assign(n, 0);
45     height[s] = n;
46     flow.assign(n, vector<int>(n, 0));
47     excess.assign(n, 0);
48     excess[s] = inf;
49     for (int i = 0; i < n; i++) {
50         if (i != s)
51             push(s, i);
52     }
53     seen.assign(n, 0);
54
55     while (!excess_vertices.empty()) {
56         int u = excess_vertices.front();
57         excess_vertices.pop();
58         if (u != s && u != t)
59             discharge(u);
60     }
61
62     int max_flow = 0;
63     for (int i = 0; i < n; i++)
64         max_flow += flow[i][t];
65     return max_flow;
66 }
```

## 5 Geometry

### 5.1 Point Struct

```
1  typedef long long T;
2  struct pt {
3      T x,y;
4      pt operator+(pt p) {return {x+p.x, y+p.y};}
5      pt operator-(pt p) {return {x-p.x, y-p.y};}
6      pt operator*(T d) {return {x*d, y*d};}
7      pt operator/(T d) {return {x/d, y/d};}
8  };
9
10 // cross product
11 // positivo si el segundo esta en sentido antihorario
```

```

12 // 0 si el angulo es 180
13 // negativo si el segundo esta en sentido horario
14 T cross(pt v, pt w) {return v.x*w.y - v.y*w.x;}
15
16 // dot product
17 // positivo si el angulo entre los vectores es agudo
18 // 0 si son perpendiculares
19 // negativo si el angulo es obtuso
20 T dot(pt v, pt w) {return v.x*w.x + v.y*w.y;}
21
22 T orient(pt a, pt b, pt c) {return cross(b-a,c-a);}
23
24 T dist(pt a,pt b){
25     pt aux=b-a;
26     return sqrtl(aux.x*aux.x+aux.y*aux.y);
27 }

```

## 5.2 Sort Points

```

1 // This comparator sorts the points clockwise
2 // starting from the first quarter
3
4 bool getQ(Point a){
5     if(a.y!=0){
6         if(a.y>0)return 0;
7         return 1;
8     }
9     if(a.x>0)return 0;
10    return 1;
11 }
12 bool comp(Point a, Point b){
13     if(getQ(a)!=getQ(b))return getQ(a)<getQ(b);
14     return a*b>0;
15 }

```

## 6 Graphs

### 6.1 2Sat

```

1 struct TwoSatSolver {
2     int n_vars;
3     int n_vertices;
4     vector<vector<int>> adj, adj_t;

```

```

5     vector<bool> used;
6     vector<int> order, comp;
7     vector<bool> assignment;
8
9     TwoSatSolver(int _n_vars) : n_vars(_n_vars), n_vertices(2 * n_vars),
10         adj(n_vertices), adj_t(n_vertices), used(n_vertices), order(),
11         comp(n_vertices, -1), assignment(n_vars) {
12         order.reserve(n_vertices);
13     }
14     void dfs1(int v) {
15         used[v] = true;
16         for (int u : adj[v]) {
17             if (!used[u])
18                 dfs1(u);
19         }
20         order.push_back(v);
21     }
22     void dfs2(int v, int cl) {
23         comp[v] = cl;
24         for (int u : adj_t[v]) {
25             if (comp[u] == -1)
26                 dfs2(u, cl);
27         }
28     }
29     bool solve_2SAT() {
30         order.clear();
31         used.assign(n_vertices, false);
32         for (int i = 0; i < n_vertices; ++i) {
33             if (!used[i])
34                 dfs1(i);
35         }
36
37         comp.assign(n_vertices, -1);
38         for (int i = 0, j = 0; i < n_vertices; ++i) {
39             int v = order[n_vertices - i - 1];
40             if (comp[v] == -1)
41                 dfs2(v, j++);
42         }
43
44         assignment.assign(n_vars, false);
45         for (int i = 0; i < n_vertices; i += 2) {

```



```

46         if (comp[i] == comp[i + 1])
47             return false;
48         assignment[i / 2] = comp[i] > comp[i + 1];
49     }
50     return true;
51 }
52
53 void add_disjunction(int a, bool na, int b, bool nb) {
54     // na==1 means not a
55     a = 2 * a ^ na;
56     b = 2 * b ^ nb;
57     int neg_a = a ^ 1;
58     int neg_b = b ^ 1;
59     adj[neg_a].push_back(b);
60     adj[neg_b].push_back(a);
61     adj_t[b].push_back(neg_a);
62     adj_t[a].push_back(neg_b);
63 }
64 };

```

## 6.2 Articulation Points

```

1  int n; // number of nodes
2  vector<vector<int>> adj; // adjacency list of graph
3
4  vector<bool> visited;
5  vector<int> tin, low;
6  int timer;
7
8  void dfs(int v, int p = -1) {
9      visited[v] = true;
10     tin[v] = low[v] = timer++;
11     int children=0;
12     for (int to : adj[v]) {
13         if (to == p) continue;
14         if (visited[to]) {
15             low[v] = min(low[v], tin[to]);
16         } else {
17             dfs(to, v);
18             low[v] = min(low[v], low[to]);
19             if (low[to] >= tin[v] && p!=-1){
20                 // Articulation point
21             }

```

```

22         ++children;
23     }
24 }
25 if(p == -1 && children > 1){
26     // Articulation point
27 }
28 }
29
30 void find_cutpoints() {
31     timer = 0;
32     visited.assign(n, false);
33     tin.assign(n, -1);
34     low.assign(n, -1);
35     for (int i = 0; i < n; ++i) {
36         if (!visited[i])
37             dfs (i);
38     }
39 }

```

## 6.3 Bellman-Ford

```

1  const int INF = 1000000000;
2  vector<vector<pair<int, int>>> adj;
3
4  bool spfa(int s, vector<int>& d) {
5      int n = adj.size();
6      d.assign(n, INF);
7      vector<int> cnt(n, 0);
8      vector<bool> inqueue(n, false);
9      queue<int> q;
10
11      d[s] = 0;
12      q.push(s);
13      inqueue[s] = true;
14      while (!q.empty()) {
15         int v = q.front();
16         q.pop();
17         inqueue[v] = false;
18
19         for (auto edge : adj[v]) {
20             int to = edge.first;
21             int len = edge.second;
22

```

```

23         if (d[v] + len < d[to]) {
24             d[to] = d[v] + len;
25             if (!inqueue[to]) {
26                 q.push(to);
27                 inqueue[to] = true;
28                 cnt[to]++;
29                 if (cnt[to] > n)
30                     return false; // negative cycle
31             }
32         }
33     }
34 }
35 return true;
36 }

```

## 6.4 Bipartite Checker

```

1 int n;
2 vector<vector<int>> adj;
3
4 vector<int> side(n, -1);
5 bool is_bipartite = true;
6 queue<int> q;
7 for (int st = 0; st < n; ++st) {
8     if (side[st] == -1) {
9         q.push(st);
10        side[st] = 0;
11        while (!q.empty()) {
12            int v = q.front();
13            q.pop();
14            for (int u : adj[v]) {
15                if (side[u] == -1) {
16                    side[u] = side[v] ^ 1;
17                    q.push(u);
18                } else {
19                    is_bipartite &= side[u] != side[v];
20                }
21            }
22        }
23    }
24 }
25
26 cout << (is_bipartite ? "YES" : "NO") << endl;

```

## 6.5 Bipartite Maximum Matching

```

1
2 int n, k;
3 vector<vector<int>> g;
4 vector<int> mt;
5 vector<bool> used;
6
7 bool try_kuhn(int v) {
8     if (used[v])
9         return false;
10    used[v] = true;
11    for (int to : g[v]) {
12        if (mt[to] == -1 || try_kuhn(mt[to])) {
13            mt[to] = v;
14            return true;
15        }
16    }
17    return false;
18 }
19
20 int main() {
21     //... reading the graph ...
22
23     mt.assign(k, -1);
24     for (int v = 0; v < n; ++v) {
25         used.assign(n, false);
26         try_kuhn(v);
27     }
28
29     for (int i = 0; i < k; ++i)
30         if (mt[i] != -1)
31             printf("%d_ %d\n", mt[i] + 1, i + 1);
32 }

```

## 6.6 Block Cut Tree

```

1 vector<vector<int>> biconnected_components(vector<vector<int>> &g,
2
3                                     vector<bool> &is_cutpoint,
4                                     vector<int> &id) {
5
6     int n = (int)g.size();

```

```

6  vector<vector<int>> comps;
7  vector<int> stk;
8  vector<int> num(n);
9  vector<int> low(n);
10 is_cutpoint.resize(n);
11 // Finds the biconnected components
12 function<void(int, int, int &)> dfs = [&](int node, int parent, int &
    timer) {
13     num[node] = low[node] = ++timer;
14     stk.push_back(node);
15     for (int son : g[node]) {
16         if (son == parent) { continue; }
17         if (num[son]) {
18             low[node] = min(low[node], num[son]);
19         } else {
20             dfs(son, node, timer);
21             low[node] = min(low[node], low[son]);
22             if (low[son] >= num[node]) {
23                 is_cutpoint[node] = (num[node] > 1 || num[son] > 2);
24                 comps.push_back({node});
25                 while (comps.back().back() != son) {
26                     comps.back().push_back(stk.back());
27                     stk.pop_back();
28                 }
29             }
30         }
31     }
32 };
33
34 int timer = 0;
35 dfs(0, -1, timer);
36 id.resize(n);
37 // Build the block-cut tree
38
39 function<vector<vector<int>>>()> build_tree = [&]() {
40     vector<vector<int>> t(1);
41     int node_id = 0;
42     for (int node = 0; node < n; node++) {
43         if (is_cutpoint[node]) {
44             id[node] = node_id++;
45             t.push_back({});
46         }
47     }

```

```

48
49     for (auto &comp : comps) {
50         int node = node_id++;
51         t.push_back({});
52         for (int u : comp)
53             if (!is_cutpoint[u]) {
54                 id[u] = node;
55             } else {
56                 t[node].push_back(id[u]);
57                 t[id[u]].push_back(node);
58             }
59     }
60     return t;
61 };
62 return build_tree();
63
64 }

```

## 6.7 Blossom

```

1  const int N = 2e3 + 9;
2
3  mt19937 rnd(chrono::steady_clock::now().time_since_epoch().count());
4  struct Blossom {
5      int vis[N], par[N], orig[N], match[N], aux[N], t;
6      int n;
7      bool ad[N];
8      vector<int> g[N];
9      queue<int> Q;
10     Blossom() {}
11     Blossom(int _n) {
12         n = _n;
13         t = 0;
14         for (int i = 0; i <= _n; ++i) {
15             g[i].clear();
16             match[i] = aux[i] = par[i] = vis[i] = aux[i] = ad[i] = orig[i] =
                0;
17         }
18     }
19     void add_edge(int u, int v) {
20         g[u].push_back(v);
21         g[v].push_back(u);
22     }

```

```

23 void augment(int u, int v) {
24     int pv = v, nv;
25     do {
26         pv = par[v];
27         nv = match[pv];
28         match[v] = pv;
29         match[pv] = v;
30         v = nv;
31     } while (u != pv);
32 }
33 int lca(int v, int w) {
34     ++t;
35     while (true) {
36         if (v) {
37             if (aux[v] == t) return v;
38             aux[v] = t;
39             v = orig[par[match[v]]];
40         }
41         swap(v, w);
42     }
43 }
44 void blossom(int v, int w, int a) {
45     while (orig[v] != a) {
46         par[v] = w;
47         w = match[v];
48         ad[v] = true;
49         if (vis[w] == 1) Q.push(w), vis[w] = 0;
50         orig[v] = orig[w] = a;
51         v = par[w];
52     }
53 }
54 //it finds an augmented path starting from u
55 bool bfs(int u) {
56     fill(vis + 1, vis + n + 1, -1);
57     iota(orig + 1, orig + n + 1, 1);
58     Q = queue<int> ();
59     Q.push(u);
60     vis[u] = 0;
61     while (!Q.empty()) {
62         int v = Q.front();
63         Q.pop();
64         ad[v] = true;
65         for (int x : g[v]) {

```

```

66         if (vis[x] == -1) {
67             par[x] = v;
68             vis[x] = 1;
69             if (!match[x]) return augment(u, x), true;
70             Q.push(match[x]);
71             vis[match[x]] = 0;
72         } else if (vis[x] == 0 && orig[v] != orig[x]) {
73             int a = lca(orig[v], orig[x]);
74             blossom(x, v, a);
75             blossom(v, x, a);
76         }
77     }
78 }
79 return false;
80 }
81 int maximum_matching() {
82     int ans = 0;
83     vector<int> p(n - 1);
84     iota(p.begin(), p.end(), 1);
85     shuffle(p.begin(), p.end(), rnd);
86     for (int i = 1; i <= n; i++) shuffle(g[i].begin(), g[i].end(), rnd);
87     for (auto u : p) {
88         if (!match[u]) {
89             for(auto v : g[u]) {
90                 if (!match[v]) {
91                     match[u] = v, match[v] = u;
92                     ++ans;
93                     break;
94                 }
95             }
96         }
97     }
98     for(int i = 1; i <= n; ++i) if (!match[i] && bfs(i)) ++ans;
99     return ans;
100 }
101 } M;
102 int32_t main() {
103     ios_base::sync_with_stdio(0);
104     cin.tie(0);
105     int t;
106     cin >> t;
107     while (t--) {
108         int n, m;

```

```

109     cin >> n >> m;
110     M = Blossom (n);
111     while (m--) {
112         int u, v;
113         cin >> u >> v;
114         M.add_edge(u, v);
115     }
116     int ans = M.maximum_matching();
117     if (ans * 2 == n) cout << 0 << '\n';
118     else {
119         memset(M.ad, 0, sizeof M.ad);
120         for (int i = 1; i <= n; i++) if (M.match[i] == 0) M.bfs(i);
121         int ans = 0;
122         for (int i = 1; i <= n; i++) ans += M.ad[i]; //nodes which are
123             unmatched in some maximum matching
124         cout << ans << '\n';
125     }
126     return 0;
127 }

```

## 6.8 Bridges

```

1  int n;
2  vector<vector<int>> adj;
3
4  vector<bool> visited;
5  vector<int> tin, low;
6  int timer;
7
8  void dfs(int v, int p = -1) {
9      visited[v] = true;
10     tin[v] = low[v] = timer++;
11     for (int to : adj[v]) {
12         if (to == p) continue;
13         if (visited[to]) {
14             low[v] = min(low[v], tin[to]);
15         } else {
16             dfs(to, v);
17             low[v] = min(low[v], low[to]);
18             if (low[to] > tin[v]){
19                 // Bridge
20             }
21         }
22     }
23 }

```

```

21     }
22 }
23
24 }
25
26 void find_bridges() {
27     timer = 0;
28     visited.assign(n, false);
29     tin.assign(n, -1);
30     low.assign(n, -1);
31     for (int i = 0; i < n; ++i) {
32         if (!visited[i])
33             dfs(i);
34     }
35 }

```

## 6.9 Bridges Online

```

1  vector<int> par, dsu_2ecc, dsu_cc, dsu_cc_size;
2  int bridges;
3  int lca_iteration;
4  vector<int> last_visit;
5
6  void init(int n) {
7      par.resize(n);
8      dsu_2ecc.resize(n);
9      dsu_cc.resize(n);
10     dsu_cc_size.resize(n);
11     lca_iteration = 0;
12     last_visit.assign(n, 0);
13     for (int i=0; i<n; ++i) {
14         dsu_2ecc[i] = i;
15         dsu_cc[i] = i;
16         dsu_cc_size[i] = 1;
17         par[i] = -1;
18     }
19     bridges = 0;
20 }
21
22 int find_2ecc(int v) {
23     if (v == -1)
24         return -1;
25     return dsu_2ecc[v] == v ? v : dsu_2ecc[v] = find_2ecc(dsu_2ecc[v]);

```

```

26 }
27
28 int find_cc(int v) {
29     v = find_2ecc(v);
30     return dsu_cc[v] == v ? v : dsu_cc[v] = find_cc(dsu_cc[v]);
31 }
32
33 void make_root(int v) {
34     int root = v;
35     int child = -1;
36     while (v != -1) {
37         int p = find_2ecc(par[v]);
38         par[v] = child;
39         dsu_cc[v] = root;
40         child = v;
41         v = p;
42     }
43     dsu_cc_size[root] = dsu_cc_size[child];
44 }
45
46 void merge_path (int a, int b) {
47     ++lca_iteration;
48     vector<int> path_a, path_b;
49     int lca = -1;
50     while (lca == -1) {
51         if (a != -1) {
52             a = find_2ecc(a);
53             path_a.push_back(a);
54             if (last_visit[a] == lca_iteration){
55                 lca = a;
56                 break;
57             }
58             last_visit[a] = lca_iteration;
59             a = par[a];
60         }
61         if (b != -1) {
62             b = find_2ecc(b);
63             path_b.push_back(b);
64             if (last_visit[b] == lca_iteration){
65                 lca = b;
66                 break;
67             }
68             last_visit[b] = lca_iteration;

```

```

69         b = par[b];
70     }
71
72 }
73
74 for (int v : path_a) {
75     dsu_2ecc[v] = lca;
76     if (v == lca)
77         break;
78     --bridges;
79 }
80 for (int v : path_b) {
81     dsu_2ecc[v] = lca;
82     if (v == lca)
83         break;
84     --bridges;
85 }
86 }
87
88 void add_edge(int a, int b) {
89     a = find_2ecc(a);
90     b = find_2ecc(b);
91     if (a == b)
92         return;
93
94     int ca = find_cc(a);
95     int cb = find_cc(b);
96
97     if (ca != cb) {
98         ++bridges;
99         if (dsu_cc_size[ca] > dsu_cc_size[cb]) {
100             swap(a, b);
101             swap(ca, cb);
102         }
103         make_root(a);
104         par[a] = dsu_cc[a] = b;
105         dsu_cc_size[cb] += dsu_cc_size[a];
106     } else {
107         merge_path(a, b);
108     }
109 }

```

## 6.10 Dijkstra

```

1 priority_queue <ll> q;
2 dist[0]=0;
3 q.push({0, 0});
4 while((int)q.size()){
5     auto [w, v]=q.top(); q.pop();
6     w=-w;
7     if (w>=dist[v]) continue;
8     for (auto [u, p]:adj[v]){
9         if(dist[v]+p<dist[u]){
10             dist[u]=dist[v]+p;
11             q.push({-dist[p], u});
12         }
13     }
14 }

```

## 6.11 Eulerian Path

```

1 // check if all edges are visite
2 // directed
3 void dfs(int node) {
4     while (!g[node].empty()) {
5         auto [son, idx] = g[node].back();
6         g[node].pop_back();
7         if (seen[idx]) { continue; }
8         seen[idx] = true;
9         dfs(son);
10    }
11    path.push_back(node);
12 }
13 //undirected
14 void dfs(int node) {
15     while (!g[node].empty()) {
16         int son = g[node].back();
17         g[node].pop_back();
18         dfs(son);
19     }
20    path.push_back(node);
21 }

```

## 6.12 Floyd-Warshall

```

1 vector<vector<ll>> d(n, vector<ll>(n, 1e18));

```

```

2
3 for (int k = 0; k < n; k++) {
4     for (int i = 0; i < n; i++) {
5         for (int j = i + 1; j < n; j++) {
6             long long new_dist = d[i][k] + d[k][j];
7             if (new_dist < d[i][j]) {
8                 d[i][j] = d[j][i] = new_dist;
9             }
10        }
11    }
12 }

```

## 6.13 Kruskal

```

1 template <class T> T kruskal(int N, vector<pair<T, pair<int, int>>>
2     edges) {
3     sort(edges.begin(), edges.end());
4     T ans = 0;
5     DSU D(N + 1); // edges that unite are in MST
6     for (pair<T, pair<int, int>> &e : edges) {
7         if (D.unite(e.second.first, e.second.second)) { ans += e.first; }
8     }
9     // -1 if the graph is not connected, otherwise the sum of the edge
10    // lengths
11    return (D.size(1) == N ? ans : -1);
12 }

```

## 6.14 Marriage

```

1 // MALE OPTIMAL STABLE MARRIAGE PROBLEM O(N^2)
2 // gv[i][j] jth most preferred female for ith male
3 // om[i][j] jth most preferred male for ith female
4 #define MAXN 1000
5 int gv[MAXN][MAXN], om[MAXN][MAXN];
6 int pv[MAXN], pm[MAXN]; // ans
7 int pun[MAXN]; // Auxiliary
8
9 void stableMarriage(int n) {
10    fill_n(pv, n, -1); fill_n(pm, n, -1); fill_n(pun, n, 0);
11    int s = n, i = n-1;
12    #define engage pm[j] = i; pv[i] = j;
13    while (s) {
14        while (pv[i] == -1) {
15            int j = gv[i][pun[i]++];

```

```

16     if (pm[j] == -1) {
17         s--;
18         engage;
19     }
20     else if (om[j][i] < om[j][pm[j]]) {
21         int loser = pm[j];
22         pv[loser] = -1;
23         engage;
24         i = loser;
25     }
26 }
27 i--;
28 if (i < 0) i = n-1;
29 }
30 }

```

## 6.15 SCC

```

1 vector<vector<int>> adj,adjr;
2 vector<bool> vis;
3 vector<int> order,comp;
4 void dfs(int a){
5     vis[a]=1;
6     for(auto u:adj[a]){
7         if(!vis[u]){
8             dfs(u);
9         }
10    }
11    order.pb(a);
12 }
13 void dfsr(int a,int k){
14     vis[a]=1;
15     comp[a]=k;
16     for(auto u:adjr[a]){
17         if(!vis[u]){
18             dfsr(u,k);
19         }
20    }
21 }
22
23 void solve() {
24     int n,m;cin>>n>>m;
25     adj.assing(n,vector<int>());

```

```

26     adjr.assing(n,vector<int>());
27     comp.resize(n);
28     for(int i=0;i<m;i++){
29         int a,b;cin>>a>>b;a--;b--;
30         adj[a].pb(b);
31         adjr[b].pb(a);
32     }
33     vis.assign(n,0);
34     for(int i=0;i<n;i++){
35         if(!vis[i])dfs(i);
36     }
37     vis.assign(n,0);
38     int c=0;
39     for(int i=n-1;i>=0;i--){
40         if(!vis[order[i]]){
41             dfsr(order[i],c);
42             c++;
43         }
44     }
45 }
46 }

```

## 7 Linear Algebra

### 7.1 Simplex

```

1  /*
2  Parametric Self-Dual Simplex method
3  Solve a canonical LP:
4      min or max. c x
5      s.t. A x <= b
6           x >= 0
7  */
8  #include <bits/stdc++.h>
9  using namespace std;
10 const double eps = 1e-9, oo = numeric_limits<double>::infinity();
11
12 typedef vector<double> vec;
13 typedef vector<vec> mat;
14
15 pair<vec, double> simplexMethodPD(const mat &A, const vec &b, const vec
    &c, bool mini = true){
16     int n = c.size(), m = b.size();

```



```

17 mat T(m + 1, vec(n + m + 1));
18 vector<int> base(n + m), row(m);
19
20 for(int j = 0; j < m; ++j){
21     for(int i = 0; i < n; ++i)
22         T[j][i] = A[j][i];
23     row[j] = n + j;
24     T[j][n + j] = 1;
25     base[n + j] = 1;
26     T[j][n + m] = b[j];
27 }
28
29 for(int i = 0; i < n; ++i)
30     T[m][i] = c[i] * (mini ? 1 : -1);
31
32 while(true){
33     int p = 0, q = 0;
34     for(int i = 0; i < n + m; ++i)
35         if(T[m][i] <= T[m][p])
36             p = i;
37
38     for(int j = 0; j < m; ++j)
39         if(T[j][n + m] <= T[q][n + m])
40             q = j;
41
42     double t = min(T[m][p], T[q][n + m]);
43
44     if(t >= -eps){
45         vec x(n);
46         for(int i = 0; i < m; ++i)
47             if(row[i] < n) x[row[i]] = T[i][n + m];
48         return {x, T[m][n + m] * (mini ? -1 : 1)}; // optimal
49     }
50
51     if(t < T[q][n + m]){
52         // tight on c -> primal update
53         for(int j = 0; j < m; ++j)
54             if(T[j][p] >= eps)
55                 if(T[j][p] * (T[q][n + m] - t) >= T[q][p] * (T[j][n + m] - t))
56                     q = j;
57
58         if(T[q][p] <= eps)
59             return {vec(n), oo * (mini ? 1 : -1)}; // primal infeasible

```

```

60     }else{
61         // tight on b -> dual update
62         for(int i = 0; i < n + m + 1; ++i)
63             T[q][i] = -T[q][i];
64
65         for(int i = 0; i < n + m; ++i)
66             if(T[q][i] >= eps)
67                 if(T[q][i] * (T[m][p] - t) >= T[q][p] * (T[m][i] - t))
68                     p = i;
69
70         if(T[q][p] <= eps)
71             return {vec(n), oo * (mini ? -1 : 1)}; // dual infeasible
72     }
73
74     for(int i = 0; i < m + n + 1; ++i)
75         if(i != p) T[q][i] /= T[q][p];
76
77     T[q][p] = 1; // pivot(q, p)
78     base[p] = 1;
79     base[row[q]] = 0;
80     row[q] = p;
81
82     for(int j = 0; j < m + 1; ++j){
83         if(j != q){
84             double alpha = T[j][p];
85             for(int i = 0; i < n + m + 1; ++i)
86                 T[j][i] -= T[q][i] * alpha;
87         }
88     }
89 }
90
91 return {vec(n), oo};
92 }
93
94 int main(){
95     int m, n;
96     bool mini = true;
97     cout << "Numero_de_restricciones: ";
98     cin >> m;
99     cout << "Numero_de_incognitas: ";
100    cin >> n;
101    mat A(m, vec(n));
102    vec b(m), c(n);

```

```

103 for(int i = 0; i < m; ++i){
104     cout << "Restriccion_" << (i + 1) << ":\n";
105     for(int j = 0; j < n; ++j){
106         cin >> A[i][j];
107     }
108     cin >> b[i];
109 }
110 cout << "[0]Max_o_Min?:\n";
111 cin >> mini;
112 cout << "Coeficientes_de_" << (mini ? "min" : "max") << "_z:\n";
113 for(int i = 0; i < n; ++i){
114     cin >> c[i];
115 }
116 cout.precision(6);
117 auto ans = simplexMethodPD(A, b, c, mini);
118 cout << (mini ? "Min" : "Max") << "_z=" << ans.second << ", cuando:\n";
119 for(int i = 0; i < ans.first.size(); ++i){
120     cout << "x_" << (i + 1) << "= " << ans.first[i] << "\n";
121 }
122 return 0;
123 }

```

## 8 Math

### 8.1 BinPow

```

1 ll bnpow(ll a, ll b){
2     ll r=1;
3     while(b){
4         if(b%2)
5             r=(r*a)%MOD;
6         a=(a*a)%MOD;
7         b/=2;
8     }
9     return r;
10 }
11
12 ll divide(ll a, ll b){
13     return ((a%MOD)*bnpow(b, MOD-2))%MOD;
14 }
15 void inverses(long long p) {
16     inv[MAXN] = exp(fac[MAXN], p - 2, p);

```

```

17     for (int i = MAXN; i >= 1; i--) { inv[i - 1] = inv[i] * i % p; }
18 }

```

### 8.2 Diophantine

```

1 int gcd(int a, int b, int& x, int& y) {
2     if (b == 0) {
3         x = 1;
4         y = 0;
5         return a;
6     }
7     int x1, y1;
8     int d = gcd(b, a % b, x1, y1);
9     x = y1;
10    y = x1 - y1 * (a / b);
11    return d;
12 }
13
14 bool find_any_solution(int a, int b, int c, int &x0, int &y0, int &g) {
15     g = gcd(abs(a), abs(b), x0, y0);
16     if (c % g) {
17         return false;
18     }
19
20     x0 *= c / g;
21     y0 *= c / g;
22     if (a < 0) x0 = -x0;
23     if (b < 0) y0 = -y0;
24     return true;
25 }
26
27
28
29 //n variables
30 vector<ll> find_any_solution(vector<ll> a, ll c) {
31     int n = a.size();
32     vector<ll> x;
33     bool all_zero = true;
34     for (int i = 0; i < n; i++) {
35         all_zero &= a[i] == 0;
36     }
37     if (all_zero) {
38         if (c) return {};

```

```

39     x.assign(n, 0);
40     return x;
41 }
42 ll g = 0;
43 for (int i = 0; i < n; i++) {
44     g = __gcd(g, a[i]);
45 }
46 if (c % g != 0) return {};
47 if (n == 1) {
48     return {c / a[0]};
49 }
50 vector<ll> suf_gcd(n);
51 suf_gcd[n - 1] = a[n - 1];
52 for (int i = n - 2; i >= 0; i--) {
53     suf_gcd[i] = __gcd(suf_gcd[i + 1], a[i]);
54 }
55 ll cur = c;
56 for (int i = 0; i + 1 < n; i++) {
57     ll x0, y0, g;
58     // solve for a[i] * x + suf_gcd[i + 1] * (y / suf_gcd[i + 1]) = cur
59     bool ok = find_any_solution(a[i], suf_gcd[i + 1], cur, x0, y0, g);
60     assert(ok);
61     {
62         // trying to minimize x0 in case x0 becomes big
63         // it is needed for this problem, not needed in general
64         ll shift = abs(suf_gcd[i + 1] / g);
65         x0 = (x0 % shift + shift) % shift;
66     }
67     x.push_back(x0);
68
69     // now solve for the next suffix
70     cur -= a[i] * x0;
71 }
72 x.push_back(a[n - 1] == 0 ? 0 : cur / a[n - 1]);
73 return x;
74 }

```

### 8.3 Discrete Logarithm

```

1 // Returns minimum x for which a ^ x % m = b % m, a and m are coprime.
2 int solve(int a, int b, int m) {
3     a %= m, b %= m;
4     int n = sqrt(m) + 1;

```

```

5
6     int an = 1;
7     for (int i = 0; i < n; ++i)
8         an = (an * 1ll * a) % m;
9
10    unordered_map<int, int> vals;
11    for (int q = 0, cur = b; q <= n; ++q) {
12        vals[cur] = q;
13        cur = (cur * 1ll * a) % m;
14    }
15
16    for (int p = 1, cur = 1; p <= n; ++p) {
17        cur = (cur * 1ll * an) % m;
18        if (vals.count(cur)) {
19            int ans = n * p - vals[cur];
20            return ans;
21        }
22    }
23    return -1;
24 }
25
26 // Returns minimum x for which a ^ x % m = b % m.
27 int solve(int a, int b, int m) {
28     a %= m, b %= m;
29     int k = 1, add = 0, g;
30     while ((g = gcd(a, m)) > 1) {
31         if (b == k)
32             return add;
33         if (b % g)
34             return -1;
35         b /= g, m /= g, ++add;
36         k = (k * 1ll * a / g) % m;
37     }
38
39     int n = sqrt(m) + 1;
40     int an = 1;
41     for (int i = 0; i < n; ++i)
42         an = (an * 1ll * a) % m;
43
44     unordered_map<int, int> vals;
45     for (int q = 0, cur = b; q <= n; ++q) {
46         vals[cur] = q;
47         cur = (cur * 1ll * a) % m;

```

```

48     }
49
50     for (int p = 1, cur = k; p <= n; ++p) {
51         cur = (cur * 111 * an) % m;
52         if (vals.count(cur)) {
53             int ans = n * p - vals[cur] + add;
54             return ans;
55         }
56     }
57     return -1;
58 }

```

## 8.4 Divisors

```

1 long long numberOfDivisors(long long num)
2 {
3     long long total = 1;
4     for (int i = 2; (long long)i * i <= num; i++)
5     {
6         if (num % i == 0)
7         {
8             int e = 0;
9             do
10             {
11                 e++;
12                 num /= i;
13             } while (num % i == 0);
14             total *= e + 1;
15         }
16     }
17     if (num > 1)
18     {
19         total *= 2;
20     }
21     return total;
22 }
23
24 long long SumOfDivisors(long long num)
25 {
26     long long total = 1;
27
28     for (int i = 2; (long long)i * i <= num; i++)
29     {

```

```

30         if (num % i == 0)
31         {
32             int e = 0;
33             do
34             {
35                 e++;
36                 num /= i;
37             } while (num % i == 0);
38
39             long long sum = 0, pow = 1;
40             do
41             {
42                 sum += pow;
43                 pow *= i;
44             } while (e-- > 0);
45             total *= sum;
46         }
47     }
48     if (num > 1)
49     {
50         total *= (1 + num);
51     }
52     return total;
53 }

```

## 8.5 Euler Totient (Phi)

```

1 //counts coprimes to each number from 1 to n
2 vector<int> phi1(int n) {
3     vector<int> phi(n + 1);
4     for (int i = 0; i <= n; i++)
5         phi[i] = i;
6
7     for (int i = 2; i <= n; i++) {
8         if (phi[i] == i) {
9             for (int j = i; j <= n; j += i)
10                 phi[j] -= phi[j] / i;
11         }
12     }
13     return phi1;
14 }

```

## 8.6 Fibonacci

```

1 void fib(ll n, ll&x, ll&y){
2     if(n==0){
3         x = 0;
4         y = 1;
5         return ;
6     }
7
8     if(n&1){
9         fib(n-1, y, x);
10        y=(y+x)%MOD;
11    }else{
12        ll a, b;
13        fib(n>>1, a, b);
14        y = (a*a+b*b)%MOD;
15        x = (a*b + a*(b-a+MOD))%MOD;
16    }
17 }
18
19 // Usage
20 // ll x, y;
21 // fib(10, x, y);
22 // cout << x << " " << y << endl;
23 // This will output 55 89

```

## 8.7 Matrix Exponentiation

```

1 struct Mat {
2     int n, m;
3     vector<vector<int>> a;
4     Mat() { }
5     Mat(int _n, int _m) {n = _n; m = _m; a.assign(n, vector<int>(m, 0)); }
6     Mat(vector< vector<int> > v) { n = v.size(); m = n ? v[0].size() : 0;
7         a = v; }
8     inline void make_unit() {
9         assert(n == m);
10        for (int i = 0; i < n; i++) {
11            for (int j = 0; j < n; j++) a[i][j] = i == j;
12        }
13    }
14    inline Mat operator + (const Mat &b) {
15        assert(n == b.n && m == b.m);
16        Mat ans = Mat(n, m);
17        for(int i = 0; i < n; i++) {

```

```

17        for(int j = 0; j < m; j++) {
18            ans.a[i][j] = (a[i][j] + b.a[i][j]) % mod;
19        }
20    }
21    return ans;
22 }
23 inline Mat operator - (const Mat &b) {
24     assert(n == b.n && m == b.m);
25     Mat ans = Mat(n, m);
26     for(int i = 0; i < n; i++) {
27         for(int j = 0; j < m; j++) {
28             ans.a[i][j] = (a[i][j] - b.a[i][j] + mod) % mod;
29         }
30     }
31     return ans;
32 }
33 inline Mat operator * (const Mat &b) {
34     assert(m == b.n);
35     Mat ans = Mat(n, b.m);
36     for(int i = 0; i < n; i++) {
37         for(int j = 0; j < b.m; j++) {
38             for(int k = 0; k < m; k++) {
39                 ans.a[i][j] = (ans.a[i][j] + 1LL * a[i][k] * b.a[k][j] % mod)
40                     % mod;
41             }
42         }
43     }
44     return ans;
45 }
46 inline Mat pow(long long k) {
47     assert(n == m);
48     Mat ans(n, n), t = a; ans.make_unit();
49     while (k) {
50         if (k & 1) ans = ans * t;
51         t = t * t;
52         k >>= 1;
53     }
54     return ans;
55 }
56 inline Mat& operator += (const Mat& b) { return *this = (*this) + b; }
57 inline Mat& operator -= (const Mat& b) { return *this = (*this) - b; }
58 inline Mat& operator *= (const Mat& b) { return *this = (*this) * b; }
59 inline bool operator == (const Mat& b) { return a == b.a; }

```

```

59 inline bool operator != (const Mat& b) { return a != b.a; }
60 };
61
62 // Usage
63 // Mat a(n, n);
64 // Mat b(n, n);
65 // Mat c = a * b;
66 // Mat d = a + b;
67 // Mat e = a - b;
68 // Mat f = a.pow(k);
69 // a.a[i][j] = x;

```

## 8.8 Miller Rabin Deterministic

```

1 using u64 = uint64_t;
2 using u128 = __uint128_t;
3
4 u64 binpower(u64 base, u64 e, u64 mod) {
5     u64 result = 1;
6     base %= mod;
7     while (e) {
8         if (e & 1)
9             result = (u128)result * base % mod;
10        base = (u128)base * base % mod;
11        e >>= 1;
12    }
13    return result;
14 }
15
16 bool check_composite(u64 n, u64 a, u64 d, int s) {
17     u64 x = binpower(a, d, n);
18     if (x == 1 || x == n - 1)
19         return false;
20     for (int r = 1; r < s; r++) {
21         x = (u128)x * x % n;
22         if (x == n - 1)
23             return false;
24     }
25     return true;
26 };
27
28 bool MillerRabin(ll n) {
29

```

```

30     if (n < 2)
31         return false;
32
33     int r = 0;
34     ll d = n - 1;
35     while ((d & 1) == 0) {
36         d >>= 1;
37         r++;
38     }
39
40     for (int a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}) {
41         if (n == a)
42             return true;
43         if (check_composite(n, a, d, r))
44             return false;
45     }
46     return true;
47 }

```

## 8.9 Mobius

```

1 int mob[N];
2 void mobius() {
3     mob[1] = 1;
4     for (int i = 2; i < N; i++){
5         mob[i]--;
6         for (int j = i + i; j < N; j += i) {
7             mob[j] -= mob[i];
8         }
9     }
10 }

```

## 8.10 Prefix Sum Phi

```

1 vector<ll> sieve(kMaxV + 1, 0);
2 vector<ll> phi(kMaxV + 1, 0);
3
4 void primes()
5 {
6     phi[1] = 1;
7     vector<ll> pr;
8     for (int i = 2; i < kMaxV; i++) {
9         if (sieve[i] == 0) {
10             sieve[i] = i;

```

```

11     pr.pb(i);
12     phi[i]=i-1;
13 }
14 for(auto p:pr){
15     if(p>sieve[i]||i*p>=kMaxV)break;
16     sieve[i*p]=p;
17     phi[i*p]=(p==sieve[i]?p:p-1)*phi[i];
18 }
19 }
20 for(int i=1;i<kMaxV;i++){
21     phi[i]+=phi[i-1];
22     phi[i]%MOD;
23 }
24 }
25
26 map<ll,ll> m;
27 ll PHI(ll a){
28     if(a<kMaxV)return phi[a];
29     if(m.count(a))return m[a];
30     // if(a<3)return 1;
31     m[a]=((((a%MOD)*((a+1)%MOD))%MOD)*inverse(2));
32     m[a]%MOD;
33     long long i=2;
34     while(i<=a){
35         long long j=a/i;
36         j=a/j;
37         m[a]+=MOD;
38         m[a]-=((j-i+1)*PHI(a/i))%MOD;
39         m[a]%MOD;
40         i=j+1;
41     }
42     m[a]%MOD;
43     return m[a];
44 }

```

## 8.11 Sieve

```

1 const int kMaxV = 1e6;
2
3 int sieve[kMaxV + 1];
4
5 //stores some prime (not necessarily the minimum one)
6 void primes()

```

```

7 {
8     for (int i = 4; i <= kMaxV; i += 2)
9         sieve[i] = 2;
10    for (int i = 3; i <= kMaxV / i; i += 2)
11    {
12        if (sieve[i])
13            continue;
14        for (int j = i * i; j <= kMaxV; j += i)
15            sieve[j] = i;
16    }
17 }
18
19 vector<int> PrimeFactors(int x)
20 {
21     if (x == 1)
22         return {};
23
24     unordered_set<int> primes;
25     while (sieve[x])
26     {
27         primes.insert(sieve[x]);
28         x /= sieve[x];
29     }
30     primes.insert(x);
31     return {primes.begin(), primes.end()};
32 }

```

## 9 More Topics

### 9.1 2D Prefix Sum

```

1 int b[MAXN] [MAXN];
2 int a[MAXN] [MAXN];
3
4 for (int i = 1; i <= N; i++) {
5     for (int j = 1; j <= N; j++) {
6         b[i] [j] = a[i] [j] + b[i - 1] [j] +
7             b[i] [j - 1] - b[i - 1] [j - 1];
8     }
9 }
10
11 for (int q = 0; q < Q; q++) {
12     int from_row, to_row, from_col, to_col;

```

```

13 cin >> from_row >> from_col >> to_row >> to_col;
14 cout << b[to_row][to_col] - b[from_row - 1][to_col] -
15      b[to_row][from_col - 1] +
16      b[from_row - 1][from_col - 1]
17      << '\n';
18 }

```

## 9.2 Custom Comparators

```

1 bool cmp(const Edge &x, const Edge &y) { return x.w < y.w; }
2
3 sort(a.begin(), a.end(), cmp);
4
5 set<int, greater<int>> a;
6 map<int, string, greater<int>> b;
7 priority_queue<int, vector<int>, greater<int>> c;

```

## 9.3 Day of the Week

```

1 int dayOfWeek(int d, int m, lli y){
2     if(m == 1 || m == 2){
3         m += 12;
4         --y;
5     }
6     int k = y % 100;
7     lli j = y / 100;
8     return (d + 13*(m+1)/5 + k + k/4 + j/4 + 5*j) % 7;
9 }

```

## 9.4 GCD Convolution

```

1 vector<int> PrimeEnumerate(int n) {
2     vector<int> P; vector<bool> B(n + 1, 1);
3     for (int i = 2; i <= n; i++) {
4         if (B[i]) P.push_back(i);
5         for (int j : P) { if (i * j > n) break; B[i * j] = 0; if (i % j ==
6             0) break; }
7     }
8     return P;
9 }
10
11 template<typename T>
12 void MultipleZetaTransform(vector<T>& v) {

```

```

13     const int n = (int)v.size() - 1;
14     for (int p : PrimeEnumerate(n)) {
15         for (int i = n / p; i; i--)
16             v[i] += v[i * p];
17     }
18 }
19
20 template<typename T>
21 void MultipleMobiusTransform(vector<T>& v) {
22     const int n = (int)v.size() - 1;
23     for (int p : PrimeEnumerate(n)) {
24         for (int i = 1; i * p <= n; i++)
25             v[i] -= v[i * p];
26     }
27 }
28
29 template<typename T>
30 vector<T> GCDConvolution(vector<T> A, vector<T> B) {
31     MultipleZetaTransform(A);
32     MultipleZetaTransform(B);
33     for (int i = 0; i < A.size(); i++) A[i] *= B[i];
34     MultipleMobiusTransform(A);
35     return A;
36 }

```

## 9.5 int128

```

1 //cout for __int128
2 ostream &operator<<(ostream &os, const __int128 & value){
3     char buffer[64];
4     char *pos = end(buffer) - 1;
5     *pos = '\0';
6     __int128 tmp = value < 0 ? -value : value;
7     do{
8         --pos;
9         *pos = tmp % 10 + '0';
10        tmp /= 10;
11    }while(tmp != 0);
12    if(value < 0){
13        --pos;
14        *pos = '-';
15    }
16    return os << pos;

```



```

17 }
18
19 //cin for __int128
20 istream &operator>>(istream &is, __int128 & value){
21     char buffer[64];
22     is >> buffer;
23     char *pos = begin(buffer);
24     int sgn = 1;
25     value = 0;
26     if(*pos == '-'){
27         sgn = -1;
28         ++pos;
29     }else if(*pos == '+'){
30         ++pos;
31     }
32     while(*pos != '\0'){
33         value = (value << 3) + (value << 1) + (*pos - '0');
34         ++pos;
35     }
36     value *= sgn;
37     return is;
38 }
39
40
41 ll mult(__int128 a, __int128 b){ return ((a*1LL*b)%MOD + MOD)%MOD; }

```

## 9.6 Iterating Over All Subsets

```

1 for (int mk = 0; mk < (1 << k); mk++) {
2     Ap[mk] = 0;
3     for (int s = mk;; s = (s - 1) & mk) {
4         Ap[mk] += A[s];
5         if (!s)
6             break;
7     }
8 }

```

## 9.7 LCM Convolution

```

1 /* Linear Sieve, O(n) */
2 vector<int> PrimeEnumerate(int n) {
3     vector<int> P; vector<bool> B(n + 1, 1);
4     for (int i = 2; i <= n; i++) {
5         if (B[i]) P.push_back(i);

```

```

6         for (int j : P) { if (i * j > n) break; B[i * j] = 0; if (i % j ==
7             0) break; }
8     }
9     return P;
10 }
11
12 template<typename T>
13 void DivisorZetaTransform(vector<T>& v) {
14     const int n = (int)v.size() - 1;
15     for (int p : PrimeEnumerate(n)) {
16         for (int i = 1; i * p <= n; i++)
17             v[i * p] += v[i];
18     }
19 }
20
21 template<typename T>
22 void DivisorMobiusTransform(vector<T>& v) {
23     const int n = (int)v.size() - 1;
24     for (int p : PrimeEnumerate(n)) {
25         for (int i = n / p; i; i--)
26             v[i * p] -= v[i];
27     }
28 }
29
30 template<typename T>
31 vector<T> LCMConvolution(vector<T> A, vector<T> B) {
32     DivisorZetaTransform(A);
33     DivisorZetaTransform(B);
34     for (int i = 0; i < A.size(); i++) A[i] *= B[i];
35     DivisorMobiusTransform(A);
36     return A;
37 }

```

## 9.8 Manhattan MST

```

1 struct point {
2     long long x, y;
3 };
4
5 vector<tuple<long long, int, int>> manhattan_mst_edges(vector<point> ps)
6 {
7     vector<int> ids(ps.size());

```

```

7   iota(ids.begin(), ids.end(), 0);
8   vector<tuple<long long, int, int>> edges;
9   for (int rot = 0; rot < 4; rot++) { // for every rotation
10      sort(ids.begin(), ids.end(), [&](int i, int j){
11         return (ps[i].x + ps[i].y) < (ps[j].x + ps[j].y);
12      });
13      map<int, int, greater<int>> active; // (xs, id)
14      for (auto i : ids) {
15         for (auto it = active.lower_bound(ps[i].x); it != active.end();
16              active.erase(it++)) {
17             int j = it->second;
18             if (ps[i].x - ps[i].y > ps[j].x - ps[j].y) break;
19             assert(ps[i].x >= ps[j].x && ps[i].y >= ps[j].y);
20             edges.push_back({(ps[i].x - ps[j].x) + (ps[i].y - ps[j].y), i, j
21                             });
22         }
23         active[ps[i].x] = i;
24     }
25     for (auto &p : ps) { // rotate
26         if (rot & 1) p.x *= -1;
27         else swap(p.x, p.y);
28     }
29     return edges;
30 }

```

## 9.9 Mo

```

1   ll n, q;
2   ll cur=0;
3   ll cnt[1000005];
4   ll answers[200500];
5   ll BLOCK_SIZE;
6   ll arr[200500];
7
8   pair< pair<ll, ll>, ll> queries[200500];
9
10  inline bool cmp(const pair< pair<ll, ll>, ll> &x, const pair< pair<ll,
11                  ll>, ll> &y) {
12      ll block_x = x.first.first / BLOCK_SIZE;
13      ll block_y = y.first.first / BLOCK_SIZE;
14      if(block_x != block_y)
15          return block_x < block_y;

```

```

15      return x.first.second < y.first.second;
16  }
17
18  int main(){
19      cin >> n >> q;
20      BLOCK_SIZE =(ll)(sqrt(n));
21      for(int i = 0; i < n; i++)
22          cin >> arr[i];
23
24      for(int i = 0; i < q; i++) {
25          cin >> queries[i].first.first >> queries[i].first.second;
26          queries[i].second = i;
27      }
28
29      sort(queries, queries + q, cmp);
30
31      ll l = 0, r = -1;
32
33      for(int i = 0; i < q; i++) {
34          ll left = queries[i].first.first;
35          left--;
36          ll right = queries[i].first.second;
37          right--;
38
39          while(r < right) {
40              //operations
41              r++;
42          }
43          while(r > right) {
44              //operations
45              r--;
46          }
47
48          while(l < left) {
49              //operations
50              l++;
51          }
52          while(l > left) {
53              //operations
54              l--;
55          }
56          answers[queries[i].second] = cur;
57      }

```

58 } }

## 9.10 MOD INT

```

1  /**
2   * Description: Mod integer class for doing modular arithmetic.
3   * Source: https://github.com/jakobkogler/Algorithm-DataStructures/blob/master/Math/Modular.h
4   * Verification: https://open.kattis.com/problems/modulararithmetic
5   * Time: fast
6   */
7
8  template<int MOD>
9  struct ModInt {
10     long long v;
11     ModInt(long long _v = 0) {v = (-MOD < _v && _v < MOD) ? _v : _v %
        MOD; if (v < 0) v += MOD;}
12     ModInt& operator += (const ModInt &other) {v += other.v; if (v >=
        MOD) v -= MOD; return *this;}
13     ModInt& operator -= (const ModInt &other) {v -= other.v; if (v < 0)
        v += MOD; return *this;}
14     ModInt& operator *= (const ModInt &other) {v = v * other.v % MOD;
        return *this;}
15     ModInt& operator /= (const ModInt &other) {return *this *= inverse(
        other);}
16     bool operator == (const ModInt &other) const {return v == other.v;}
17     bool operator != (const ModInt &other) const {return v != other.v;}
18     friend ModInt operator + (ModInt a, const ModInt &b) {return a += b
        ;}
19     friend ModInt operator - (ModInt a, const ModInt &b) {return a -= b
        ;}
20     friend ModInt operator * (ModInt a, const ModInt &b) {return a *= b
        ;}
21     friend ModInt operator / (ModInt a, const ModInt &b) {return a /= b
        ;}
22     friend ModInt operator - (const ModInt &a) {return 0 - a;}
23     friend ModInt power(ModInt a, long long b) {ModInt ret(1); while (b
        > 0) {if (b & 1) ret *= a; a *= a; b >>= 1;} return ret;}
24     friend ModInt inverse(ModInt a) {return power(a, MOD - 2);}
25     friend istream& operator >> (istream &is, ModInt &m) {is >> m.v; m.v
        = (-MOD < m.v && m.v < MOD) ? m.v : m.v % MOD; if (m.v < 0) m.v
        += MOD; return is;}
26     friend ostream& operator << (ostream &os, const ModInt &m) {return

```

os &lt;&lt; m.v;}

27 };

## 9.11 Next Permutation

```

1  sort(v.begin(),v.end());
2  while(next_permutation(v.begin(),v.end())){
3      for(auto u:v){
4          cout<<u<<" ";
5      }
6      cout<<endl;
7  }
8
9  string s="asdfassd";
10 sort(s.begin(),s.end());
11 while(next_permutation(s.begin(),s.end())){
12     cout<<s<<endl;
13 }

```

## 9.12 Next and Previous Smaller/Greater Element

```

1  vector<int> nextSmaller(vector<int> a, int n){
2      stack<int> s;
3      vector<int> res(n, n);
4      for(int i=0;i<n;i++){
5          while(s.size() && a[s.top()]>a[i]){
6              res[s.top()]=i;
7              s.pop();
8          }
9          s.push(i);
10     }
11     return res;
12 }
13
14 vector<int> prevSmaller(vector<int> a, int n){
15     stack<int> s;
16     vector<int> res(n, -1);
17     for(int i=n-1;i>=0;i--){
18         while(s.size() && a[s.top()]>a[i]){
19             res[s.top()]=i;
20             s.pop();
21         }
22         s.push(i);
23     }

```

```

24 | return res;
25 | }

```

### 9.13 Parallel Binary Search

```

1 | int lo[maxn], hi[maxn];
2 | vector<int> tocheck[maxn];
3 |
4 | bool c=true;
5 | while(c){
6 |     c=false;
7 |     //initialize changes of structure to 0
8 |
9 |     for(int i=0;i<k;i++){
10 |         if(low[i]!=high[i]){
11 |             check[(low[i]+high[i])/2].pb(i);
12 |         }
13 |     }
14 |
15 |     for(int i=0;i<m;i++){
16 |         // apply change for ith query
17 |
18 |         while(check[i].size()){
19 |             c=true;
20 |             int x=check[i].back();
21 |             check[i].pop_back();
22 |
23 |             if(operationToCheck){
24 |                 high[x]=i;
25 |             }
26 |             else{
27 |                 low[x]=i+1;
28 |             }
29 |         }
30 |     }
31 | }

```

### 9.14 Random Number Generators

```

1 | //to avoid hacks
2 | mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
3 | mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
4 | //you can also just write seed_value if hacks are not an issue
5 |

```

```

6 | // rng() for generating random numbers between 0 and 2<<31-1
7 |
8 | // for generating numbers with uniform probability in range
9 | uniform_int_distribution<int>(0, n)(rng)
10 | std::normal_distribution<> normal_dist(mean, 2)
11 | exponential_distribution
12 |
13 |
14 | // for shuffling array
15 | shuffle(permutation.begin(), permutation.end(), rng);

```

### 9.15 setprecision

```

1 | cout<<fixed<<setprecision(10);

```

### 9.16 Ternary Search

```

1 | double ternary_search(double l, double r) {
2 |     double eps = 1e-9; //set the error limit here
3 |     while (r - l > eps) {
4 |         double m1 = l + (r - l) / 3;
5 |         double m2 = r - (r - l) / 3;
6 |         double f1 = f(m1); //evaluates the function at m1
7 |         double f2 = f(m2); //evaluates the function at m2
8 |         if (f1 < f2)
9 |             l = m1;
10 |        else
11 |            r = m2;
12 |    }
13 |    return f(l); //return the maximum of f(x) in [l,
14 |    r]
15 | }

```

### 9.17 Ternary Search Int

```

1 | int lo = -1, hi = n;
2 | while (hi - lo > 1){
3 |     int mid = (hi + lo)>>1;
4 |     if (f(mid) > f(mid + 1))
5 |         hi = mid;
6 |     else
7 |         lo = mid;
8 | }
9 | //lo + 1 is the answer

```

## 9.18 XOR Convolution

```

1 void FWHT (int A[], int k, int inv) {
2     for (int j = 0; j < k; j++)
3         for (int i = 0; i < (1 << k); i++)
4             if (~i & (1 << j)) {
5                 int p0 = A[i];
6                 int p1 = A[i | (1 << j)];
7
8                 A[i] = p0 + p1;
9                 A[i | (1 << j)] = p0 - p1;
10
11                 if (inv) {
12                     A[i] /= 2;
13                     A[i | (1 << j)] /= 2;
14                 }
15             }
16 }
17
18 void XOR_conv (int A[], int B[], int C[], int k) {
19     FWHT(A, k, false);
20     FWHT(B, k, false);
21
22     for (int i = 0; i < (1 << k); i++)
23         C[i] = A[i] * B[i];
24
25     FWHT(A, k, true);
26     FWHT(B, k, true);
27     FWHT(C, k, true);
28 }

```

## 9.19 XOR Basis

```

1
2 int basis[d]; // basis[i] keeps the mask of the vector whose f value is
   i
3
4 int sz; // Current size of the basis
5
6 void insertVector(int mask) {
7     //turn for around if u want max xor
8     for (int i = 0; i < d; i++) {
9         if ((mask & 1 << i) == 0) continue; // continue if i != f(mask)

```

```

10
11     if (!basis[i]) { // If there is no basis vector with the i'th bit
12         // set, then insert this vector into the basis
13         basis[i] = mask;
14         ++sz;
15
16         return;
17     }
18
19     mask ^= basis[i]; // Otherwise subtract the basis vector from this
20     // vector
21 }
22
23 // If you dont need the basis sorted.
24 vector<ll> basis;
25 void add(ll x)
26 {
27     for (int i = 0; i < basis.size(); i++)
28     {
29         x = min(x, x ^ basis[i]);
30     }
31     if (x != 0)
32     {
33         basis.pb(x);
34     }
35 }

```

## 10 Polynomials

### 10.1 Berlekamp Massey

```

1 template<typename T>
2 vector<T> berlekampMassey(const vector<T> &s) {
3     vector<T> c; // the linear recurrence sequence we are building
4     vector<T> oldC; // the best previous version of c to use (the one
5         // with the rightmost left endpoint)
6     int f = -1; // the index at which the best previous version of c
7         // failed on
8     for (int i=0; i<(int)s.size(); i++) {
9         // evaluate c(i)
10        // delta = s_i - \sum_{j=1}^n c_j s_{i-j}
11        // if delta == 0, c(i) is correct

```

```

10     T delta = s[i];
11     for (int j=1; j<=(int)c.size(); j++)
12         delta -= c[j-1] * s[i-j]; // c_j is one-indexed, so we
            actually need index j - 1 in the code
13     if (delta == 0)
14         continue; // c(i) is correct, keep going
15     // now at this point, delta != 0, so we need to adjust it
16     if (f == -1) {
17         // this is the first time we're updating c
18         // s_i was the first non-zero element we encountered
19         // we make c of length i + 1 so that s_i is part of the base
            case
20         c.resize(i + 1);
21         mt19937 rng(chrono::steady_clock::now().time_since_epoch().
            count());
22         for (T &x : c)
23             x = rng(); // just to prove that the initial values don
                't matter in the first step, I will set to random
                values
24         f = i;
25     } else {
26         // we need to use a previous version of c to improve on this
            one
27         // apply the 5 steps to build d
28         // 1. set d equal to our chosen sequence
29         vector<T> d = oldC;
30         // 2. multiply the sequence by -1
31         for (T &x : d)
32             x = -x;
33         // 3. insert a 1 on the left
34         d.insert(d.begin(), 1);
35         // 4. multiply the sequence by delta / d(f + 1)
36         T df1 = 0; // d(f + 1)
37         for (int j=1; j<=(int)d.size(); j++)
38             df1 += d[j-1] * s[f+1-j];
39         assert(df1 != 0);
40         T coef = delta / df1; // storing this in outer variable so
            it's O(n^2) instead of O(n^2 log MOD)
41         for (T &x : d)
42             x *= coef;
43         // 5. insert i - f - 1 zeros on the left
44         vector<T> zeros(i - f - 1);
45         zeros.insert(zeros.end(), d.begin(), d.end());

```

```

46         d = zeros;
47         // now we have our new recurrence: c + d
48         vector<T> temp = c; // save the last version of c because it
            might have a better left endpoint
49         c.resize(max(c.size(), d.size()));
50         for (int j=0; j<(int)d.size(); j++)
51             c[j] += d[j];
52         // finally, let's consider updating oldC
53         if (i - (int) temp.size() > f - (int) oldC.size()) {
54             // better left endpoint, let's update!
55             oldC = temp;
56             f = i;
57         }
58     }
59 }
60 return c;
61 }

```

## 10.2 FFT

```

1 using cd = complex<double>;
2 const double PI = acos(-1);
3 //declare size of vectors used like this
4 const int MAXN=2<<19;
5
6 void fft(vector<cd> & a, bool invert) {
7     int n = (int)a.size();
8
9     for (int i = 1, j = 0; i < n; i++) {
10         int bit = n >> 1;
11         for (; j & bit; bit >>= 1)
12             j ^= bit;
13         j ^= bit;
14
15         if (i < j)
16             swap(a[i], a[j]);
17     }
18
19     for (int len = 2; len <= n; len <<= 1) {
20         double ang = 2 * PI / len * (invert ? -1 : 1);
21         cd wlen(cos(ang), sin(ang));
22         for (int i = 0; i < n; i += len) {
23             cd w(1);

```

```

24     for (int j = 0; j < len / 2; j++) {
25         cd u = a[i+j], v = a[i+j+len/2] * w;
26         a[i+j] = u + v;
27         a[i+j+len/2] = u - v;
28         w *= wlen;
29     }
30 }
31 }
32
33 if (invert) {
34     for (cd & x : a)
35         x /= n;
36 }
37 }
38
39 vector<int> multiply(vector<int> const& a, vector<int> const& b) {
40     vector<cd> fa(a.begin(), a.end()), fb(b.begin(), b.end());
41     int n = 1;
42     while (n < a.size() + b.size())
43         n <<= 1;
44     fa.resize(n);
45     fb.resize(n);
46
47     fft(fa, false);
48     fft(fb, false);
49     for (int i = 0; i < n; i++)
50         fa[i] *= fb[i];
51     fft(fa, true);
52
53     vector<int> result(n);
54     for (int i = 0; i < n; i++)
55         result[i] = round(fa[i].real());
56     return result;
57 }
58
59 //normalizing for when mult is between 2 big numbers and not polynomials
60 int carry = 0;
61 for (int i = 0; i < n; i++){
62     result[i] += carry;
63     carry = result[i] / 10;
64     result[i] %= 10;
65 }

```

### 10.3 NTT

```

1 // number theory transform
2
3 const int MOD = 998244353, ROOT = 3;
4 // const int MOD = 7340033, ROOT = 5;
5 // const int MOD = 167772161, ROOT = 3;
6 // const int MOD = 469762049, ROOT = 3;
7
8 int power(int base, int exp) {
9     int res = 1;
10    while (exp) {
11        if (exp % 2) res = 1LL * res * base % MOD;
12        base = 1LL * base * base % MOD;
13        exp /= 2;
14    }
15    return res;
16 }
17
18 void ntt(vector<int>& a, bool invert) {
19     int n = a.size();
20     for (int i = 1, j = 0; i < n; i++) {
21         int bit = n >> 1;
22         for (; j & bit; bit >>= 1) j ^= bit;
23         j ^= bit;
24         if (i < j) swap(a[i], a[j]);
25     }
26     for (int len = 2; len <= n; len <<= 1) {
27         int wlen = power(ROOT, (MOD - 1) / len);
28         if (invert) wlen = power(wlen, MOD - 2);
29         for (int i = 0; i < n; i += len) {
30             int w = 1;
31             for (int j = 0; j < len / 2; j++) {
32                 int u = a[i + j], v = 1LL * a[i + j + len / 2] * w % MOD;
33                 a[i + j] = u + v < MOD ? u + v : u + v - MOD;
34                 a[i + j + len / 2] = u - v >= 0 ? u - v : u - v + MOD;
35                 w = 1LL * w * wlen % MOD;
36             }
37         }
38     }
39     if (invert) {
40         int n_inv = power(n, MOD - 2);
41         for (int& x : a) x = 1LL * x * n_inv % MOD;

```

```

42     }
43 }
44
45 vector<int> multiply(vector<int>& a, vector<int>& b) {
46     int n = 1;
47     while (n < a.size() + b.size()) n <= 1;
48     a.resize(n), b.resize(n);
49     ntt(a, false), ntt(b, false);
50     for (int i = 0; i < n; i++) a[i] = 1LL * a[i] * b[i] % MOD;
51     ntt(a, true);
52     return a;
53 }
54 // usage
55 // vector<int> a = {1, 2, 3}, b = {4, 5, 6};
56 // vector<int> c = multiply(a, b);
57 // for (int x : c) cout << x << " ";

```

## 10.4 Roots NTT

```

1 1*2^0 + 1 = 2, 1, 1
2 1*2^1 + 1 = 3, 2, 2
3 1*2^2 + 1 = 5, 2, 3
4 2*2^3 + 1 = 17, 2, 9
5 1*2^4 + 1 = 17, 3, 6
6 3*2^5 + 1 = 97, 19, 46
7 3*2^6 + 1 = 193, 11, 158
8 2*2^7 + 1 = 257, 9, 200
9 1*2^8 + 1 = 257, 3, 86
10 15*2^9 + 1 = 7681, 62, 1115
11 12*2^10 + 1 = 15361, 49, 1254
12 6*2^11 + 1 = 12289, 7, 8778
13 3*2^12 + 1 = 12289, 41, 4496
14 5*2^13 + 1 = 40961, 12, 23894
15 4*2^14 + 1 = 65537, 15, 30584
16 2*2^15 + 1 = 65537, 9, 7282
17 1*2^16 + 1 = 65537, 3, 21846
18 6*2^17 + 1 = 786433, 8, 688129
19 3*2^18 + 1 = 786433, 5, 471860
20 11*2^19 + 1 = 5767169, 12, 3364182
21 7*2^20 + 1 = 7340033, 5, 4404020
22 11*2^21 + 1 = 23068673, 38, 21247462
23 25*2^22 + 1 = 104857601, 21, 49932191
24 20*2^23 + 1 = 167772161, 4, 125829121

```

```

25 10*2^24 + 1 = 167772161, 2, 83886081
26 5*2^25 + 1 = 167772161, 17, 29606852
27 7*2^26 + 1 = 469762049, 30, 15658735
28 15*2^27 + 1 = 2013265921, 137, 749463956
29 12*2^28 + 1 = 3221225473, 8, 2818572289
30 6*2^29 + 1 = 3221225473, 14, 1150437669
31 3*2^30 + 1 = 3221225473, 13, 1734506024
32 35*2^31 + 1 = 75161927681, 93, 44450602392
33 18*2^32 + 1 = 77309411329, 106, 5105338484

```

## 11 Scripts

### 11.1 build.sh

This file should be called before stress.sh or validate.sh. build.sh name.cpp

```

1 g++ -static -DLOCAL -lm -s -x c++ -Wall -Wextra -O2 -std=c++17 -o $1 $1.
  cpp

```

### 11.2 stress.sh

Format is stress.sh Awrong Aslow Agen Numtests

```

1 #!/usr/bin/env bash
2
3 for ((testNum=0;testNum<$4;testNum++))
4 do
5     ./$3 > input
6     ./$2 < input > outSlow
7     ./$1 < input > outWrong
8     H1='md5sum outWrong'
9     H2='md5sum outSlow'
10    if !(cmp -s "outWrong" "outSlow")
11    then
12        echo "Error found!"
13        echo "Input:"
14        cat input
15        echo "Wrong Output:"
16        cat outWrong
17        echo "Slow Output:"
18        cat outSlow
19        exit
20    fi
21 done

```



```
22 | echo Passed $4 tests
```

## 11.3 validate.sh

Format is validate.sh Awrong Avalidator Agen NumTests

```
1 | #!/usr/bin/env bash
2 |
3 | for ((testNum=0;testNum<$4;testNum++))
4 | do
5 |     ./$3 > input
6 |     ./$1 < input > out
7 |     cat input out > data
8 |     ./$2 < data > res
9 |     result=$(cat res)
10 |    if [ "${result:0:2}" != "OK" ];
11 |    then
12 |        echo "Error_found!"
13 |        echo "Input:"
14 |        cat input
15 |        echo "Output:"
16 |        cat out
17 |        echo "Validator_Result:"
18 |        cat res
19 |        exit
20 |    fi
21 | done
22 | echo Passed $4 tests
```

## 12 Strings

### 12.1 Hashed String

```
1 | const ll N = 2e5+1;
2 | const ll MOD = 212345678987654321LL;
3 | const ll base = 33;
4 |
5 | // double hash or big mod values
6 |
7 | class HashedString {
8 |     private:
9 |         // change M and B if you want
```

```
10 |     static const long long M = 1e9 + 9;
11 |     static const long long B = 9973;
12 |
13 |     // pow[i] contains B^i % M
14 |     static vector<long long> pow;
15 |
16 |     // p_hash[i] is the hash of the first i characters of the given string
17 |     vector<long long> p_hash;
18 |
19 | public:
20 |     HashedString(const string &s) : p_hash(s.size() + 1) {
21 |         while (pow.size() < s.size()) { pow.push_back((pow.back() * B) % M);
22 |         }
23 |
24 |         p_hash[0] = 0;
25 |         for (int i = 0; i < s.size(); i++) {
26 |             p_hash[i + 1] = ((p_hash[i] * B) % M + s[i]) % M;
27 |         }
28 |
29 |     long long get_hash(int start, int end) {
30 |         long long raw_val =
31 |             (p_hash[end + 1] - (p_hash[start] * pow[end - start + 1]));
32 |         return (raw_val % M + M) % M;
33 |     }
34 | };
35 | vector<long long> HashedString::pow = {1};
```

### 12.2 KMP

```
1 | // Maximum length of substring that ends at position i and is prefix of
2 | string
3 | vector<int> KMP(string s){
4 |     int n=s.length();
5 |     vector<int> pf(n, 0);
6 |     for(int i=1;i<n;i++){
7 |         int j=pf[i-1];
8 |         while(j>0 && s[i]!=s[j]){
9 |             j=pf[j-1];
10 |         }
11 |         if(s[i]==s[j]){
12 |             pf[i]=j+1;
13 |         }
14 |     }
```

```

13 }
14 return pf;
15 }

```

## 12.3 Least Rotation String

```

1 string least_rotation(string s)
2 {
3     s += s;
4     vector<int> f(s.size(), -1);
5     int k = 0;
6     for(int j = 1; j < s.size(); j++)
7     {
8         char sj = s[j];
9         int i = f[j - k - 1];
10        while(i != -1 && sj != s[k + i + 1])
11        {
12            if(sj < s[k + i + 1]){
13                k = j - i - 1;
14            }
15            i = f[i];
16        }
17        if(sj != s[k + i + 1])
18        {
19            if(sj < s[k]){
20                k = j;
21            }
22            f[j - k] = -1;
23        }
24        else
25            f[j - k] = i + 1;
26    }
27    return s.substr(k, s.size() / 2);
28 }

```

## 12.4 Manacher

```

1 // Number of palindromes centered at each position
2
3 vector<int> manacher_odd(string s)
4 {
5     int n = s.size();
6     s = "$" + s + "^";
7     vector<int> p(n + 2);

```

```

8     int l = 1, r = 1;
9     for (int i = 1; i <= n; i++)
10    {
11        p[i] = max(0, min(r - i, p[l + (r - i)]));
12        while (s[i - p[i]] == s[i + p[i]])
13        {
14            p[i]++;
15        }
16        if (i + p[i] > r)
17        {
18            l = i - p[i], r = i + p[i];
19        }
20    }
21    return vector<int>(begin(p) + 1, end(p) - 1);
22 }
23 vector<int> manacher(string s)
24 {
25     string t;
26     for (auto c : s)
27     {
28         t += string("#") + c;
29     }
30     auto res = manacher_odd(t + "#");
31     return vector<int>(begin(res) + 1, end(res) - 1);
32 }
33
34 // usage
35 // vector<int> p = manacher("abacaba");
36 // this will return {2, 1, 4, 1, 2, 1, 8, 1, 2, 1, 4, 1, 2}
37 // vector<int> p = manacher("abaaba");
38 // this will return {2, 1, 4, 1, 2, 7, 2, 1, 4, 1, 2}

```

## 12.5 Suffix Array

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 using i64 = long long;
5
6 struct SuffixArray
7 {
8     int n;
9     string t;

```

```

10 vector<int> sa, rk, lc;
11 SuffixArray(const std::string &s)
12 {
13     n = s.length();
14     t = s;
15     sa.resize(n);
16     lc.resize(n - 1);
17     rk.resize(n);
18     std::iota(sa.begin(), sa.end(), 0);
19     std::sort(sa.begin(), sa.end(), [&](int a, int b)
20         { return s[a] < s[b]; });
21     rk[sa[0]] = 0;
22     for (int i = 1; i < n; ++i)
23         rk[sa[i]] = rk[sa[i - 1]] + (s[sa[i]] != s[sa[i - 1]]);
24     int k = 1;
25     std::vector<int> tmp, cnt(n);
26     tmp.reserve(n);
27     while (rk[sa[n - 1]] < n - 1)
28     {
29         tmp.clear();
30         for (int i = 0; i < k; ++i)
31             tmp.push_back(n - k + i);
32         for (auto i : sa)
33             if (i >= k)
34                 tmp.push_back(i - k);
35         std::fill(cnt.begin(), cnt.end(), 0);
36         for (int i = 0; i < n; ++i)
37             ++cnt[rk[i]];
38         for (int i = 1; i < n; ++i)
39             cnt[i] += cnt[i - 1];
40         for (int i = n - 1; i >= 0; --i)
41             sa[--cnt[rk[tmp[i]]]] = tmp[i];
42         std::swap(rk, tmp);
43         rk[sa[0]] = 0;
44         for (int i = 1; i < n; ++i)
45             rk[sa[i]] = rk[sa[i - 1]] + (tmp[sa[i - 1]] < tmp[sa[i]] || sa[i - 1] + k == n || tmp[sa[i - 1] + k] < tmp[sa[i] + k]);
46         k *= 2;
47     }
48     for (int i = 0, j = 0; i < n; ++i)
49     {
50         if (rk[i] == 0)
51             {

```

```

52         j = 0;
53     }
54     else
55     {
56         for (j -= j > 0; i + j < n && sa[rk[i] - 1] + j < n && s[i + j]
57             == s[sa[rk[i] - 1] + j];)
58             ++j;
59         lc[rk[i] - 1] = j;
60     }
61 }
62
63 int search(string &p){
64     int tam = p.size();
65     int l = 0, r = n;
66
67     string tmp = "";
68     //cout << p << endl;
69     while(r > l) {
70         int m = l + (r-l)/2;
71         tmp = t.substr(sa[m], min(n-sa[m], tam));
72         //cout << m << " " << sa[m] << " " << tmp << endl;
73         if(tmp >= p){
74             r = m;
75         } else {
76             l = m + 1;
77         }
78     }
79     if(l < n) {
80         tmp = t.substr(sa[l], min(n-sa[l], tam));
81     } else{
82         return -1;
83     }
84     if(tmp == p){
85         return l;
86     } else {
87         return -1;
88     }
89 }
90
91 int count(string &p) {
92     int x = search(p);
93     if(x == -1) return 0;

```

```

94     int cnt = 0;
95     int tam = p.size();
96     int maxx = 0;
97     while((1 << maxx) + x < n) maxx++;
98     int y = x;
99     for(int i = maxx-1; i >= 0; i--) {
100         if(x + (1 << i) >= n) continue;
101         string tmp = t.substr(sa[x + (1 << i)], min(n-sa[x + (1 << i)
102             ], tam));
103         if(tmp == p) x += (1 << i);
104     }
105     return x-y+1;
106 }
107 int main() {
108     cin.tie(0)->sync_with_stdio(0);
109     string s; cin >> s;
110     SuffixArray SA(s);
111
112     int q; cin >> q;
113     for(int t = 0; t < q; t++) {
114         string tmp; cin >> tmp;
115         cout << SA.count(tmp) << endl;
116     }
117
118     return 0;
119 }

```

## 12.6 Suffix Automaton

```

1 struct state {
2     int len, link, firstposition;
3     map<char, int> next;
4 };
5
6 const int MAXN = 100000;
7 state st[MAXN * 2];
8 ll cnt[MAXN*2], cntPaths[MAXN*2], cntSum[MAXN*2];
9 int sz, last;
10
11 void initSuffixAutomaton() {
12     st[0].len = 0;
13     st[0].link = -1;

```

```

14     sz++;
15     last = 0;
16 }
17
18 void insertChar(char c) {
19     int cur = sz++;
20     st[cur].len = st[last].len + 1;
21     st[cur].firstposition=st[last].len;
22     int p = last;
23     while (p != -1 && !st[p].next.count(c)) {
24         st[p].next[c] = cur;
25         p = st[p].link;
26     }
27     if (p == -1) {
28         st[cur].link = 0;
29     } else {
30         int q = st[p].next[c];
31         if (st[p].len + 1 == st[q].len) {
32             st[cur].link = q;
33         } else {
34             int clone = sz++;
35             st[clone].len = st[p].len + 1;
36             st[clone].next = st[q].next;
37             st[clone].link = st[q].link;
38             st[clone].firstposition=st[q].firstposition;
39             while (p != -1 && st[p].next[c] == q) {
40                 st[p].next[c] = clone;
41                 p = st[p].link;
42             }
43             st[q].link = st[cur].link = clone;
44         }
45     }
46     last = cur;
47     cnt[last]=1;
48 }
49
50 int search(string s){
51     int cur=0, i=0, n=(int)s.length();
52     while(i<n){
53         if(!st[cur].next.count(s[i])) return -1;
54         cur=st[cur].next[s[i]];
55         i++;
56     }

```

```

57 //sumar 2 si se quiere 1 indexado
58 return st[cur].firstposition-n+1;
59 }
60
61 void dfs(int cur){
62     cntPaths[cur]=1;
63     for(auto [x, y]:st[cur].next){
64         if(cntPaths[y]==0) dfs(y);
65         cntPaths[cur]+=cntPaths[y];
66     }
67 }
68
69 void countPaths(){
70     dfs(0);
71 }
72
73 string kthSmallestDistinct(ll k){
74     string s="";
75     int cur=0;
76     while(k>0){
77         for(auto [c, y]:st[cur].next){
78             if(k>cntPaths[y]) k-=cntPaths[y];
79             else{
80                 k--;
81                 s+=c;
82                 cur=y;
83                 break;
84             }
85         }
86     }
87     return s;
88 }
89
90 void countOccurrences(){
91     vector<pair<int, int>> a;
92     for(int i=sz-1;i>0;i--){
93         a.push_back({st[i].len, i});
94     }
95     sort(a.begin(), a.end());
96     for(int i=sz-2;i>=0;i--){
97         cnt[st[a[i].second].link]+=cnt[a[i].second];
98     }
99 }

```

```

100
101 void dfs1(int cur){
102     for(auto [x, y]:st[cur].next){
103         if(cntSum[y]==cnt[y]) dfs1(y);
104         cntSum[cur]+=cntSum[y];
105     }
106 }
107
108 void countSumOccurrences(){
109     for(int i=0;i<sz;i++){
110         cntSum[i]=cnt[i];
111     }
112     dfs1(0);
113 }
114
115 string kthSmallest(ll k){
116     string s="";
117     int cur=0;
118     while(k>0){
119         for(auto [c, y]:st[cur].next){
120             if(k>cntSum[y]) k-=cntSum[y];
121             else{
122                 k-=cnt[y];
123                 s+=c;
124                 cur=y;
125                 break;
126             }
127         }
128     }
129     return s;
130 }
131
132 //longest common substring
133 //build automaton for s first
134 string lcs (string S, string T) {
135     int v = 0, l = 0, best = 0, bestpos = 0;
136     for (int i = 0; i < T.size(); i++) {
137         while (v && !st[v].next.count(T[i])) {
138             v = st[v].link ;
139             l = st[v].len;
140         }
141         if (st[v].next.count(T[i])) {
142             v = st [v].next[T[i]];

```

```

143         l++;
144     }
145     if (l > best) {
146         best = l;
147         bestpos = i;
148     }
149 }
150 return T.substr(bestpos - best + 1, best);
151 }
152
153 int main(){
154     ios_base::sync_with_stdio(false); cin.tie(NULL);
155     string s; cin >> s;
156     initSuffixAutomaton();
157     for(char c:s){
158         insertChar(c);
159     }
160 }

```

## 12.7 Trie Ahocorasick

```

1  const int K = 26;
2
3  struct Vertex {
4      int next[K];
5      bool output = false;
6      int p = -1;
7      char pch;
8      int link = -1;
9      int go[K];
10
11      Vertex(int p=-1, char ch='$') : p(p), pch(ch) {
12          fill(begin(next), end(next), -1);
13          fill(begin(go), end(go), -1);
14      }
15 };
16
17 vector<Vertex> t(1);
18
19 void add_string(string const& s) {
20     int v = 0;
21     for (char ch : s) {
22         int c = ch - 'a';

```

```

23         if (t[v].next[c] == -1) {
24             t[v].next[c] = t.size();
25             t.emplace_back(v, ch);
26         }
27         v = t[v].next[c];
28     }
29     t[v].output = true;
30 }
31
32 int go(int v, char ch);
33
34 int get_link(int v) {
35     if (t[v].link == -1) {
36         if (v == 0 || t[v].p == 0)
37             t[v].link = 0;
38         else
39             t[v].link = go(get_link(t[v].p), t[v].pch);
40     }
41     return t[v].link;
42 }
43
44 int go(int v, char ch) {
45     int c = ch - 'a';
46     if (t[v].go[c] == -1) {
47         if (t[v].next[c] != -1)
48             t[v].go[c] = t[v].next[c];
49         else
50             t[v].go[c] = v == 0 ? 0 : go(get_link(v), ch);
51     }
52     return t[v].go[c];
53 }

```

## 12.8 Z Function

```

1  // Mayor x tal que el prefijo de s de tamaño x es igual al prefijo
2  //del sufijo que empieza en la posición i y tiene tamaño x
3
4  vector<int> z_function(string s) {
5      int n = s.size();
6      vector<int> z(n);
7      int l = 0, r = 0;
8      for(int i = 1; i < n; i++) {
9          if(i < r) {

```

```

10     z[i] = min(r - i, z[i - 1]);
11 }
12 while(i + z[i] < n && s[z[i]] == s[i + z[i]]) {
13     z[i]++;
14 }
15 if(i + z[i] > r) {
16     l = i;
17     r = i + z[i];
18 }
19 }
20 return z;
21 }
22
23 // usage
24 // vector<int> z = z_function("abacaba");
25 // this will return {0, 0, 1, 0, 3, 0, 1}
26 // vector<int> z = z_function("aaaaa");
27 // this will return {0, 4, 3, 2, 1}
28 // vector<int> z = z_function("aaabaab");
29 // this will return {0, 2, 1, 0, 2, 1, 0}

```

## 13 Trees

### 13.1 Centroid Decomposition

```

1 // code for xenia and tree
2 const int MAXN=200005;
3
4 vector<int> adj[MAXN];
5 vector<bool> is_removed(MAXN, false);
6 vector<int> subtree_size(MAXN, 0);
7 vector<int> dis(MAXN, 1e9);
8 vector<vector<pair<int, int>>> ancestor(MAXN);
9
10 int get_subtree_size(int node, int parent = -1) {
11     subtree_size[node] = 1;
12     for (int child : adj[node]) {
13         if (child == parent || is_removed[child]) { continue; }
14         subtree_size[node] += get_subtree_size(child, node);
15     }
16     return subtree_size[node];
17 }
18

```

```

19 int get_centroid(int node, int tree_size, int parent = -1) {
20     for (int child : adj[node]) {
21         if (child == parent || is_removed[child]) { continue; }
22         if (subtree_size[child] * 2 > tree_size) {
23             return get_centroid(child, tree_size, node);
24         }
25     }
26     return node;
27 }
28
29 void getDist(int cur, int centroid, int p=-1, int dist=1){
30     for (int child:adj[cur]){
31         if(child==p || is_removed[child])
32             continue;
33         dist++;
34         getDist(child, centroid, cur, dist);
35         dist--;
36     }
37     ancestor[cur].push_back(make_pair(centroid, dist));
38 }
39
40 void update(int cur){
41     for (int i=0;i<ancestor[cur].size();i++){
42         dis[ancestor[cur][i].first]=min(dis[ancestor[cur][i].first],
43             ancestor[cur][i].second);
44     }
45     dis[cur]=0;
46 }
47
48 int query(int cur){
49     int mini=dis[cur];
50     for (int i=0;i<ancestor[cur].size();i++){
51         mini=min(mini, ancestor[cur][i].second+dis[ancestor[cur][i].first]);
52     }
53     return mini;
54 }
55
56 void build_centroid_decomp(int node = 1) {
57     int centroid = get_centroid(node, get_subtree_size(node));
58     for (int child : adj[centroid]) {
59         if (is_removed[child]) { continue; }

```

```

60     getDist(child, centroid, centroid);
61 }
62
63 is_removed[centroid] = true;
64
65 for (int child : adj[centroid]) {
66     if (is_removed[child]) { continue; }
67     build_centroid_decomp(child);
68 }
69 }

```

## 13.2 Heavy Light Decomposition

```

1 //call dfs1 first
2 struct SegmentTree {
3     vector<ll> a;
4     int n;
5
6     SegmentTree(int _n) : a(2 * _n, 0), n(_n) {}
7
8     void update(int pos, ll val) {
9         for (a[pos += n] = val; pos > 1; pos >>= 1) {
10             a[pos / 2] = (a[pos] ^ a[pos ^ 1]);
11         }
12     }
13
14     ll get(int l, int r) {
15         ll res = 0;
16         for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
17             if (l & 1) {
18                 res ^= a[l++];
19             }
20             if (r & 1) {
21                 res ^= a[--r];
22             }
23         }
24         return res;
25     }
26 };
27
28 const int MAXN=500005;
29 vector<int> adj[MAXN];
30

```

```

31 SegmentTree st(MAXN);
32 int a[MAXN], sz[MAXN], to[MAXN], dpth[MAXN], s[MAXN], par[MAXN];
33 int cnt=0;
34
35 void dfs1(int cur, int p){
36     sz[cur]=1;
37     for(int x:adj[cur]){
38         if(x==p) continue;
39         dpth[x]=dpth[cur]+1;
40         par[x]=cur;
41         dfs1(x, cur);
42         sz[cur]+=sz[x];
43     }
44 }
45
46 void dfs(int cur, int p, int l){
47     st.update(cnt, a[cur]);
48     s[cur]=cnt++;
49     to[cur]=l;
50     int g=-1;
51     for(int x:adj[cur]){
52         if(x==p) continue;
53         if(g==-1 || sz[g]<sz[x]){
54             g=x;
55         }
56     }
57     if(g==-1) return;
58     dfs(g, cur, l);
59     for(int x:adj[cur]){
60         if(x==p || x==g) continue;
61         dfs(x, cur, x);
62     }
63 }
64
65 int query(int u, int v){
66     int res=0;
67     while(to[u]!=to[v]){
68         if(dpth[to[u]]<dpth[to[v]]) swap(u, v);
69         res^=st.get(s[to[u]], s[u]+1);
70         u=par[to[u]];
71     }
72     if(dpth[u]>dpth[v]) swap(u, v);
73     res^=st.get(s[u], s[v]+1);

```



```

74     return res;
75 }
76
77
78
79
80 //alternate implementation
81 vector<int> parent, depth, heavy, head, pos;
82 int cur_pos;
83
84 int dfs(int v, vector<vector<int>> const& adj) {
85     int size = 1;
86     int max_c_size = 0;
87     for (int c : adj[v]) {
88         if (c != parent[v]) {
89             parent[c] = v, depth[c] = depth[v] + 1;
90             int c_size = dfs(c, adj);
91             size += c_size;
92             if (c_size > max_c_size)
93                 max_c_size = c_size, heavy[v] = c;
94         }
95     }
96     return size;
97 }
98
99 void decompose(int v, int h, vector<vector<int>> const& adj) {
100     head[v] = h, pos[v] = cur_pos++;
101     if (heavy[v] != -1)
102         decompose(heavy[v], h, adj);
103     for (int c : adj[v]) {
104         if (c != parent[v] && c != heavy[v])
105             decompose(c, c, adj);
106     }
107 }
108
109 void init(vector<vector<int>> const& adj) {
110     int n = adj.size();
111     parent = vector<int>(n);
112     depth = vector<int>(n);
113     heavy = vector<int>(n, -1);
114     head = vector<int>(n);
115     pos = vector<int>(n);
116     cur_pos = 0;

```

```

117
118     dfs(0, adj);
119     decompose(0, 0, adj);
120 }
121
122 int query(int a, int b) {
123     int res = 0;
124     for (; head[a] != head[b]; b = parent[head[b]]) {
125         if (depth[head[a]] > depth[head[b]])
126             swap(a, b);
127         int cur_heavy_path_max = segment_tree_query(pos[head[b]], pos[b]);
128         res = max(res, cur_heavy_path_max);
129     }
130     if (depth[a] > depth[b])
131         swap(a, b);
132     int last_heavy_path_max = segment_tree_query(pos[a], pos[b]);
133     res = max(res, last_heavy_path_max);
134     return res;
135 }

```

### 13.3 Lowest Common Ancestor (LCA)

```

1  const int N=200005;
2  vector<int> adj[N];
3  vector<int> start(N), end1(N), depth(N);
4  vector<vector<int>> t(N, vi(32));
5  int timer=0;
6  int n, l;
7  // l=(int)ceil(log2(n))
8  // call dfs(1, 1, 0)
9  // 1 indexed, dont use 0 indexing
10
11
12 void dfs(int cur, int p, int cnt){
13     depth[cur]=cnt;
14     t[cur][0]=p;
15     start[cur]=timer++;
16     for(int i=1;i<=l;i++){
17         t[cur][i]=t[t[cur][i-1]][i-1];
18     }
19     for(int x:adj[cur]){
20         if(x==p) continue;

```

```
21     dfs(x, cur, cnt+1);
22 }
23 end1[cur]=++timer;
24 }
25
26 bool ancestor(int u, int v){
27     return start[u]<=start[v] && end1[u]>=end1[v];
28 }
29
30 int lca(int u, int v){
31     if(ancestor(u, v))
32         return u;
33     if (ancestor(v, u)){
34         return v;
35     }
36     for(int i=1;i>=0;i--){
37         if(!ancestor(t[u][i], v)){
38             u=t[u][i];
39         }
40     }
41     return t[u][0];
42 }
```

### 13.4 Tree Diameter

```
1 pair<int, int> dfs(const vector<vector<int>> &tree, int node = 1,
2   int previous = 0, int length = 0) {
3   pair<int, int> max_path = {node, length};
4   for (const int &i : tree[node]) {
5       if (i == previous) { continue; }
6       pair<int, int> other = dfs(tree, i, node, length + 1);
7       if (other.second > max_path.second) { max_path = other; }
8   }
9   return max_path;
10 }
```