

Competitive Programming Notebook

Aguardando o PR adicionando HLD na QueryTree

Contents

| | | | |
|---|-----------|---|-----------|
| 1 DS | 2 | 3.10 Interactive | 20 |
| 1.1 Manhattan Mst | 2 | 3.11 Flags | 20 |
| 1.2 Treap Maletta | 2 | 3.12 Custom Unordered Map | 20 |
| 1.3 Bit2d | 3 | 3.13 Overflow | 21 |
| 1.4 Bigk | 3 | 3.14 Next Permutation | 21 |
| 1.5 Mex | 4 | 3.15 First True | 21 |
| 1.6 Psum2d | 4 | 3.16 Kosaraju | 21 |
| 1.7 Dsu | 5 | 3.17 Min Priority Queue | 22 |
| 1.8 Maxqueue | 5 | 4 Math | 22 |
| 1.9 Segtree | 5 | 4.1 Is Prime | 22 |
| 1.10 Seglazystuctnode | 6 | 4.2 Fft Quirino | 22 |
| 1.11 Seglazy | 7 | 4.3 Factorization | 22 |
| 1.12 Seghash | 8 | 4.4 Sieve | 23 |
| 1.13 Segtree Lazy Iterative | 9 | 4.5 Ceil | 23 |
| 1.14 Mergesorttree | 9 | 4.6 Log Any Base | 23 |
| 1.15 Treap Cp | 10 | 4.7 Ifac | 23 |
| 1.16 Ordered Set | 11 | 4.8 Division Trick | 23 |
| 1.17 Trie Old | 11 | 4.9 Fexp | 23 |
| 1.18 Range Color Update | 12 | 4.10 Number Sum Product Of Divisors | 23 |
| 1.19 Cht | 12 | 4.11 Divisors | 24 |
| 1.20 Bit | 12 | 5 Graph | 24 |
| 1.21 Triexor | 13 | 5.1 Floyd Warshall | 24 |
| 1.22 Querytree | 13 | 5.2 Lca | 24 |
| 1.23 Sparse | 15 | 5.3 Bfs | 25 |
| 1.24 Trie | 15 | 5.4 Dinic | 25 |
| 1.25 Kruskal | 15 | 5.5 2sat | 26 |
| 2 DP | 16 | 5.6 Min Cost Max Flow | 27 |
| 2.1 Lcs | 16 | 5.7 Ford Fulkerson | 28 |
| 2.2 Lis Binary Search | 16 | 5.8 Dijkstra | 28 |
| 2.3 Edit Distance | 16 | 5.9 Has Negative Cycle | 29 |
| 2.4 Digit Dp | 16 | 5.10 3sat | 29 |
| 2.5 Range Dp | 17 | 6 Geometry | 30 |
| 2.6 Lis Segtree | 17 | 6.1 Convex Hull | 30 |
| 2.7 Knapsack | 17 | 7 Primitives | 31 |
| 2.8 Digit Dp 2 | 18 | 7.1 Set Union Intersection | 31 |
| 3 General | 19 | 8 String | 31 |
| 3.1 Last True | 19 | 8.1 Split | 31 |
| 3.2 Input By File | 19 | 8.2 Hash | 31 |
| 3.3 Mix Hash | 19 | 8.3 Is Substring | 32 |
| 3.4 Random | 19 | 8.4 Trie Xor | 32 |
| 3.5 Template | 19 | | |
| 3.6 Get Subsets Sum Iterative | 19 | | |
| 3.7 Xor Basis | 20 | | |
| 3.8 Xor 1 To N | 20 | | |
| 3.9 Base Converter | 20 | | |

1 DS

1.1 Manhattan Mst

```

1 /**
2  * Author: chilli, Takanori MAEHARA
3  * Date: 2019-11-02
4  * License: CC0
5  * Source: https://github.com/spaghetti-source/
6  *   algorithm/blob/master/geometry/rectilinear_mst.cc
7  * Description: Given N points, returns up to 4*N
8  *   edges, which are guaranteed
9  *   to contain a minimum spanning tree for the graph
10  *   with edge weights  $w(p, q) =$ 
11  *    $|p.x - q.x| + |p.y - q.y|$ . Edges are in the form (
12  *   distance, src, dst). Use a
13  *   standard MST algorithm on the result to find the
14  *   final MST.
15  * Time:  $O(N \log N)$ 
16  * Status: Stress-tested
17  */
18 /**
19  * Author: Ulf Lundstrom
20  * Date: 2009-02-26
21  * License: CC0
22  * Source: My head with inspiration from tinyKACTL
23  * Description: Class to handle points in the plane.
24  *   T can be e.g. double or long long. (Avoid int.)
25  * Status: Works fine, used a lot
26  */
27 #pragma once
28
29 template <class T> int sgn(T x) { return (x > 0) - (x
30 < 0); }
31
32 template <class T>
33 struct Point {
34     typedef Point P;
35     T x, y;
36     explicit Point(T x=0, T y=0) : x(x), y(y) {}
37     bool operator<(P p) const { return tie(x,y) < tie
38 (p.x,p.y); }
39     bool operator==(P p) const { return tie(x,y)==tie
40 (p.x,p.y); }
41     P operator+(P p) const { return P(x+p.x, y+p.y); }
42     P operator-(P p) const { return P(x-p.x, y-p.y); }
43     P operator*(T d) const { return P(x*d, y*d); }
44     P operator/(T d) const { return P(x/d, y/d); }
45     T dot(P p) const { return x*p.x + y*p.y; }
46     T cross(P p) const { return x*p.y - y*p.x; }
47     T cross(P a, P b) const { return (a-*this).cross(
48 b-*this); }
49     T dist2() const { return x*x + y*y; }
50     double dist() const { return sqrt((double)dist2()
51 ); }
52     // angle to x-axis in interval [-pi, pi]
53     double angle() const { return atan2(y, x); }
54     P unit() const { return *this/dist(); } // makes
55 dist()==1
56     P perp() const { return P(-y, x); } // rotates
57 +90 degrees
58     P normal() const { return perp().unit(); }
59     // returns point rotated 'a' radians ccw around
60 the origin
61     P rotate(double a) const {
62         return P(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a))
63     };
64 }
65
66 friend ostream& operator<<(ostream& os, P p) {
67     return os << "(" << p.x << "," << p.y << ")";
68 }

```

```

52 };
53
54 typedef Point<int> P;
55 vector<array<int, 3>> manhattanMST(vector<P> ps) {
56     vi id(sz(ps));
57     iota(all(id), 0);
58     vector<array<int, 3>> edges;
59     rep(k,0,4) {
60         sort(all(id), [&](int i, int j) {
61             return (ps[i]-ps[j]).x < (ps[j]-ps[i]).y
62         });
63         map<int, int> sweep;
64         for (int i : id) {
65             for (auto it = sweep.lower_bound(-ps[i].y
66 );
67                 it != sweep.end(); sweep.
68 erase(it++)) {
69                 int j = it->second;
70                 P d = ps[i] - ps[j];
71                 if (d.y > d.x) break;
72                 edges.push_back({d.y + d.x, i, j});
73             }
74             sweep[-ps[i].y] = i;
75         }
76         for (P& p : ps) if (k & 1) p.x = -p.x; else
77 swap(p.x, p.y);
78     }
79     return edges;
80 }

```

1.2 Treap Maletta

```

1 // CÃşdigo do Bruno Maletta!!!!!!
2 // pra problemas mais simples, usar a treap do cp!
3 // essa aqui Ãl mais poderosa, mas por isso Ãl um
4 // pouco mais lenta
5
6 // Treap Implicita
7 //
8 // Todas as operacoes custam
9 //  $O(\log(n))$  com alta probabilidade
10
11 mt19937 rng((int) chrono::steady_clock::now().
12 time_since_epoch().count());
13
14 template<typename T> struct treap {
15     struct node {
16         node *l, *r;
17         int p, sz;
18         T val, sub, lazy;
19         bool rev;
20         node(T v) : l(NULL), r(NULL), p(rng()), sz(1)
21 , val(v), sub(v), lazy(0), rev(0) {}
22         void prop() {
23             if (lazy) {
24                 val += lazy, sub += lazy*sz;
25                 if (l) l->lazy += lazy;
26                 if (r) r->lazy += lazy;
27             }
28             if (rev) {
29                 swap(l, r);
30                 if (l) l->rev ^= 1;
31                 if (r) r->rev ^= 1;
32             }
33             lazy = 0, rev = 0;
34         }
35         void update() {
36             sz = 1, sub = val;
37             if (l) l->prop(), sz += l->sz, sub += l->
38 sub;
39             if (r) r->prop(), sz += r->sz, sub += r->
40 sub;

```

```

37     }
38 };
39
40 node* root;
41
42 treap() { root = NULL; }
43 treap(const treap& t) {
44     throw logic_error("Nao copiar a treap!");
45 }
46 ~treap() {
47     vector<node*> q = {root};
48     while (q.size()) {
49         node* x = q.back(); q.pop_back();
50         if (!x) continue;
51         q.push_back(x->l), q.push_back(x->r);
52         delete x;
53     }
54 }
55
56 int size(node* x) { return x ? x->sz : 0; }
57 int size() { return size(root); }
58 void join(node* l, node* r, node*& i) { // assume
59     que l < r
60     if (!l or !r) return void(i = l ? l : r);
61     l->prop(), r->prop();
62     if (l->p > r->p) join(l->r, r, l->r), i = l;
63     else join(l, r->l, r->l), i = r;
64     i->update();
65 }
66 void split(node* i, node*& l, node*& r, int v,
67 int key = 0) {
68     if (!i) return void(r = l = NULL);
69     i->prop();
70     if (key + size(i->l) < v) split(i->r, i->r, r,
71 , v, key+size(i->l)+1), l = i;
72     else split(i->l, l, i->l, v, key), r = i;
73     i->update();
74 }
75 void push_back(T v) {
76     node* i = new node(v);
77     join(root, i, root);
78 }
79 T query(int l, int r) {
80     node *L, *M, *R;
81     split(root, M, R, r+1), split(M, L, M, l);
82     T ans = M->sub;
83     join(L, M, M), join(M, R, root);
84     return ans;
85 }
86 void update(int l, int r, T s) {
87     node *L, *M, *R;
88     split(root, M, R, r+1), split(M, L, M, l);
89     M->lazy += s;
90     join(L, M, M), join(M, R, root);
91 }
92 void reverse(int l, int r) {
93     node *L, *M, *R;
94     split(root, M, R, r+1), split(M, L, M, l);
95     M->rev ^= 1;
96     join(L, M, M), join(M, R, root);
97 }
98 };
99
100 // https://cses.fi/problemset/task/2074/
101 // Nesse problema vc tem que printar a soma de l...r
102 // e tmb dar um reverse no range l...r
103 void solve() {
104     int n, q;
105     cin >> n >> q;
106
107     treap<ll> root;

```

```

107
108     for(int i = 0; i < n; i++) {
109         ll re; cin >> re;
110         // coloca esse vertice no final do array (que
111         tãa armazenado na treap)
112         root.push_back(re);
113     }
114
115     while(q--) {
116         int t, l, r;
117         cin >> t >> l >> r;
118         l--; r--;
119         if(t == 1) {
120             root.reverse(l, r);
121         } else {
122             cout << root.query(l, r) << "\n";
123         }
124     }

```

1.3 Bit2d

```

1 struct BIT2D {
2
3     int n, m;
4     vector<vector<int>>> bit;
5
6     BIT2D(int nn, int mm) {
7         //use as 0-indexed, but inside here I will
8         use 1-indexed positions
9         n = nn + 2;
10        m = mm + 2;
11        bit.assign(n, vector<int>(m));
12    }
13
14    void update(int x, int y, int p) {
15        x++; y++;
16        assert(x > 0 && y > 0 && x <= n && y <= m);
17        for(; x < n; x += (x&(-x)))
18            for(int j = y; j < m; j += (j&(-j)))
19                bit[x][j] += p;
20    }
21
22    int sum(int x, int y) {
23        int ans = 0;
24        for(; x > 0; x -= (x & (-x)))
25            for(int j = y; j > 0; j -= (j&(-j)))
26                ans += bit[x][j];
27        return ans;
28    }
29
30    int query(int x, int y, int p, int q) {
31        //x...p on line, y...q on column
32        //sum from [x][y] to [p][q];
33        x++; y++; p++; q++;
34        assert(x > 0 && y > 0 && x <= n && y <= m);
35        assert(p > 0 && q > 0 && p <= n && q <= m);
36        return sum(p, q) - sum(x - 1, q) - sum(p, y -
37        1) + sum(x - 1, y - 1);
38    }
39 };

```

1.4 Bigk

```

1 struct SetSum {
2     ll sum;
3     multiset<ll> ms;
4
5     SetSum() {}
6

```

```

7   void add(ll x) {
8       sum += x;
9       ms.insert(x);
10  }
11
12  int rem(ll x) {
13      auto it = ms.find(x);
14
15      if (it == ms.end()) {
16          return 0;
17      }
18
19      sum -= x;
20      ms.erase(it);
21      return 1;
22  }
23
24  ll getMin() { return *ms.begin(); }
25
26  ll getMax() { return *ms.rbegin(); }
27
28  ll getSum() { return sum; }
29
30  int size() { return (int)ms.size(); }
31 };
32
33 struct BigK {
34     int k;
35     SetSum gt, mt;
36
37     BigK(int k): k(k) {}
38
39     void balance() {
40         while (gt.size() > k) {
41             ll mn = gt.getMin();
42             gt.rem(mn);
43             mt.add(mn);
44         }
45
46         while (gt.size() < k && mt.size() > 0) {
47             ll mx = mt.getMax();
48             mt.rem(mx);
49             gt.add(mx);
50         }
51     }
52
53     void add(ll x) {
54         gt.add(x);
55         balance();
56     }
57
58     void rem(ll x) {
59         if (mt.rem(x) == 0) {
60             gt.rem(x);
61         }
62
63         balance();
64     }
65
66     // be careful, O(abs(oldK - newK) * log)
67     void setK(int _k) {
68         k = _k;
69         balance();
70     }
71
72     // O(log)
73     void incK() { setK(k + 1); }
74
75     // O(log)
76     void decK() { setK(k - 1); }
77 };

```

1.5 Mex

```

1  // Mex
2  //
3  // facilita queries de mex com update
4  //
5  // N eh o maior valor possivel do mex
6  // add(x) = adiciona x
7  // rem(x) = remove x
8  //
9  // O(log N) por insert
10 // O(1) por query
11
12 struct Mex {
13     map<int, int> cnt;
14     set<int> possible;
15
16     Mex(int n) {
17         for (int i = 0; i <= n + 1; i++) {
18             possible.insert(i);
19         }
20     }
21
22     void add(int x) {
23         cnt[x]++;
24         possible.erase(x);
25     }
26
27     void rem(int x) {
28         cnt[x]--;
29
30         if (cnt[x] == 0) {
31             possible.insert(x);
32         }
33     }
34
35     int query() {
36         return *(possible.begin());
37     }
38 };

```

1.6 Psum2d

```

1  struct PSum {
2
3      vector<vi> arr;
4      int n, m, initialized = 0;
5
6      PSum(int _n, int _m) {
7          n = _n;
8          m = _m;
9          arr.resize(n + 2);
10         arr.assign(n + 2, vector<int>(m + 2, 0));
11     }
12
13     void add(int a, int b, int c) {
14         //a and b are 0-indexed
15         arr[a + 1][b + 1] += c;
16     }
17
18     void init() {
19         for(int i = 1; i <= n; i++) {
20             for(int j = 1; j <= m; j++) {
21                 arr[i][j] += arr[i][j - 1];
22                 arr[i][j] += arr[i - 1][j];
23                 arr[i][j] -= arr[i - 1][j - 1];
24             }
25         }
26         initialized = 1;
27     }
28
29     int query(int a, int b, int c, int d) {

```

```

30 // sum of a...c and b...d
31 // a, b, c and d are 0-indexed
32 assert(initialized);
33 return arr[c + 1][d + 1] - arr[a][d + 1] -
arr[c + 1][b] + arr[a][b];
34 }
35
36 };

```

1.7 Dsu

```

1 // DSU
2 //
3 // https://judge.yosupo.jp/submission/126864
4
5 struct DSU {
6     int n = 0, components = 0;
7     vector<int> parent;
8     vector<int> size;
9
10    DSU(int nn){
11        n = nn;
12        components = n;
13        size.assign(n + 5, 1);
14        parent.assign(n + 5, 0);
15        iota(parent.begin(), parent.end(), 0);
16    }
17
18    int find(int x){
19        if(x == parent[x]) {
20            return x;
21        }
22        //path compression
23        return parent[x] = find(parent[x]);
24    }
25
26    void join(int a, int b){
27        a = find(a);
28        b = find(b);
29
30        if(a == b) {
31            return;
32        }
33
34        if(size[a] < size[b]) {
35            swap(a, b);
36        }
37
38        parent[b] = a;
39        size[a] += size[b];
40        components -= 1;
41    }
42
43    int sameSet(int a, int b) {
44        a = find(a);
45        b = find(b);
46        return a == b;
47    }
48 };

```

1.8 Maxqueue

```

1 struct MaxQueue {
2     stack< pair<ll,ll> > in, out;
3
4     void add(ll x){
5         if(in.size())
6             in.push( { x, max(x, in.top().ss) } );
7         else
8             in.push( {x, x} );
9     }
10

```

```

11 get_max(){
12     if(in.size() > 0 && out.size() > 0)
13         return max(in.top().ss, out.top().ss);
14     else if(in.size() > 0) return in.top().ss;
15     else if(out.size() > 0) return out.top().ss;
16     else return INF;
17 }
18
19
20
21 void rem(){
22
23     if(out.size() == 0){
24         while(in.size()){
25             ll temp = in.top().ff, ma;
26             if(out.size() == 0) ma = temp;
27             else ma = max(temp, out.top().ss);
28             out.push({temp, ma});
29             in.pop();
30         }
31     }
32     //removendo o topo de out
33     out.pop();
34 }
35
36 ll size(){
37     return in.size() + out.size();
38 }
39
40 };

```

1.9 Segtree

```

1 struct Segtree {
2
3     int n; //size do array que a seg vai ser criada
4     em cima
5     vector<ll> seg;
6
7     Segtree(vector<ll>& s){
8         n = (int)s.size();
9         seg.resize(n+n+n, 0);
10        seg_build(1,0,n-1,s);
11    }
12
13    ll merge(ll a, ll b){
14        //return a+b;
15        if(!a) a = 0;
16        if(!b) b = 0;
17        return min(a,b);
18    }
19
20    void seg_build(int x, int l, int r, vector<ll>& s)
21    ){
22        if(r < l) return;
23        if(l == r){
24            seg[x] = s[l];
25        } else {
26            int mid = l + (r-l)/2;
27            seg_build(x+x+1, mid+1, r, s);
28            seg[x] = merge(seg[x+x+1], seg[x+x+1]);
29        }
30    }
31
32    //Não atual, intervalo na Árvore e intervalo
33    pedido
34    ll q(int x, int l, int r, int i, int j){
35        if(r < i || l > j) return 0;
36        if(l >= i && r <= j) return seg[x];
37        int mid = l + (r-l)/2;
38        return merge(q(x+x+1,mid,i,j), q(x+x+1,mid+1,
39        r,i,j));
40    }

```

```

37     }
38
39     //att posi pra val
40     void att(int x, int l, int r, int posi, ll val){
41         if(l == r){
42             seg[x] = val;
43         } else {
44             int mid = l + (r-1)/2;
45             if(posi <= mid)att(x+x,l,mid,posi,val);
46             else att(x+x+1,mid+1,r,posi,val);
47             seg[x] = merge(seg[x+x], seg[x+x+1]);
48         }
49     }
50
51     int findkth(int x, int l, int r, int k){
52         if(l == r){
53             return l;
54         } else {
55             int mid = l + (r-1)/2;
56             if(seg[x+x] >= k){
57                 return findkth(x+x,l,mid,k);
58             } else {
59                 return findkth(x+x+1,mid+1, r, k -
60                     seg[x+x]);
61             }
62         }
63
64     ll query(int l, int r){
65         return q(1, 0, n-1, l, r);
66     }
67
68     void update(int posi, ll val){ //alterar em posi
69         att(1, 0, n-1, posi, val);
70     }
71
72     int findkth(int k){
73         //kth smallest, 0(logN)
74         //use position i to count how many times
75         //value 'i' appear
76         //merge must be the sum of nodes
77         return findkth(1,0,n-1,k);
78     }
79 };

```

1.10 Seglazystuctnode

```

1 struct Node {
2
3     int l, r;
4
5     int pref0, suf0, best0;
6     int pref1, suf1, best1;
7
8     Node(){
9         pref0 = 0; suf0 = 0; best0 = 0;
10        pref1 = 0; suf1 = 0; best1 = 0;
11        l = -1; r = -1;
12    };
13
14    void Init(int val_, int l_, int r_) {
15        best0 = !val_;
16        pref0 = !val_;
17        suf0 = !val_;
18
19        best1 = val_;
20        pref1 = val_;
21        suf1 = val_;
22
23        l = l_;
24        r = r_;

```

```

25     }
26
27
28     bool AllZero() {
29         return r - l + 1 == best0;
30     }
31
32     bool AllOne() {
33         return r - l + 1 == best1;
34     }
35
36     void Reverse() {
37         swap(pref0, pref1);
38         swap(suf0, suf1);
39         swap(best0, best1);
40     }
41
42 };
43
44 Node Merge(Node a, Node b) {
45
46     if(a.l == -1 || a.r == -1) {
47         return b;
48     }
49
50     if(b.l == -1 || b.r == -1) {
51         return a;
52     }
53
54     auto ans = Node();
55
56     ans.l = a.l;
57     ans.r = b.r;
58
59     // -----
60     //
61
62     if(a.AllZero()) {
63         ans.pref0 = a.pref0 + b.pref0;
64     } else {
65         ans.pref0 = a.pref0;
66     }
67
68     if(b.AllZero()) {
69         ans.suf0 = b.suf0 + a.suf0;
70     } else {
71         ans.suf0 = b.suf0;
72     }
73
74     ans.best0 = max({
75         a.best0,
76         b.best0,
77         a.suf0 + b.pref0
78     });
79
80     // -----
81     //
82
83     if(a.AllOne()) {
84         ans.pref1 = a.pref1 + b.pref1;
85     } else {
86         ans.pref1 = a.pref1;
87     }
88
89     if(b.AllOne()) {
90         ans.suf1 = b.suf1 + a.suf1;
91     } else {
92         ans.suf1 = b.suf1;
93     }
94
95     ans.best1 = max({

```

```

96         a.best1,
97         b.best1,
98         a.suf1 + b.pref1
99     });
100
101     // -----
102     //
103     return ans;
104 }
105
106 struct SegLazy {
107     private:
108
109         int n;
110         vector<Node> seg;
111         vector<bool> lazy; // precisa reverter ou nao
112
113         void build(ll x, int l, int r, string& s){
114             if(l == r){
115                 int val = s[l] - '0';
116                 seg[x].Init(val, l, r);
117             } else {
118                 int mid = l + (r-l)/2;
119                 build(x+x, l, mid, s);
120                 build(x+x+1, mid+1, r, s);
121                 seg[x] = Merge(seg[x+x], seg[x+x+1]);
122             }
123         }
124
125         void upd_lazy(ll node, ll l, ll r){
126
127             if(lazy[node]) {
128                 seg[node].Reverse();
129             }
130
131             ll esq = node + node, dir = esq + 1;
132
133             if(dir < (int)seg.size() && lazy[node]){
134                 lazy[esq] = !lazy[esq];
135                 lazy[dir] = !lazy[dir];
136             }
137
138             lazy[node] = 0;
139         }
140
141         Node q(ll x, int l, int r, int i, int j){
142             upd_lazy(x,l,r);
143
144             if(r < i || l > j)
145                 return Node();
146
147             if(l >= i && r <= j )
148                 return seg[x];
149
150             int mid = l + (r-l)/2;
151             return Merge(q(x+x,l,mid,i,j), q(x+x+1,
152 mid+1,r,i,j));
153         }
154
155         void upd(ll x, int l, int r, int i, int j){
156             upd_lazy(x,l,r);
157             if(r < i || l > j) return;
158             if(l >= i && r <= j){
159                 lazy[x] = !lazy[x];
160                 upd_lazy(x,l,r);
161             } else {
162                 int mid = l + (r-l)/2;
163                 upd(x+x,l,mid,i,j);
164                 upd(x+x+1,mid+1,r,i,j);
165             }
166
167             seg[x] = Merge(seg[x+x], seg[x+x+1]);
168         }
169     }
170
171     public:
172
173     SegLazy(string& s){
174         n = (int)s.size();
175         seg.assign(n+n+n+n, Node());
176         lazy.assign(n+n+n+n, 0);
177         build(1,0,n-1,s);
178     }
179
180     void update(int l){
181         upd(1,0,n-1,l,l);
182     }
183
184     void update_range(int l, int r){
185         upd(1,0,n-1,l,r);
186     }
187
188     Node query(int l){
189         return q(1, 0, n-1, l, l);
190     }
191
192     Node query(int l, int r){
193         return q(1, 0, n-1, l, r);
194     }
195
196 };
197
198 void solve() {
199     int n, q;
200     string s;
201
202     cin >> n >> q >> s;
203
204     SegLazy seg(s);
205
206     while(q--) {
207         int c, l, r;
208         cin >> c >> l >> r;
209
210         if(c == 1) {
211             // inverte l...r
212             seg.update_range(l - 1, r - 1);
213         } else {
214             // query l...r
215             auto node = seg.query(l - 1, r - 1);
216             cout << node.best1 << "\n";
217         }
218     }
219 }
220
221
222
223
224

```

1.11 Seglazy

```

1 struct SegLazy {
2
3     int n;
4     vector<ll> seg;
5     vector<ll> lazy;
6
7     SegLazy(vector<ll>& arr){
8         n = (int)arr.size();
9         seg.assign(n+n+n+n, 0);
10        lazy.assign(n+n+n+n, 0);
11        build(1,0,n-1,arr);
12    }

```

```

13     ll merge(ll a, ll b){
14         return a+b;
15     }
16
17     void build(ll x, int l, int r, vector<ll>& arr){
18         if(l == r){
19             seg[x] = 1LL * arr[l];
20         } else {
21             int mid = l + (r-1)/2;
22             build(x+x, l, mid, arr);
23             build(x+x+1, mid+1, r, arr);
24             seg[x] = merge(seg[x+x], seg[x+x+1]);
25         }
26     }
27
28     void upd_lazy(ll node, ll l, ll r){
29         seg[node] += (ll)(r-l+1) * lazy[node];
30         ll esq = node + node, dir = esq + 1;
31
32         if(dir < (int)seg.size()){
33             lazy[esq] += lazy[node];
34             lazy[dir] += lazy[node];
35         }
36
37         lazy[node] = 0;
38     }
39
40     ll q(ll x, int l, int r, int i, int j){
41         upd_lazy(x,l,r);
42
43         if(r < i || l > j)
44             return 0;
45
46         if(l >= i && r <= j )
47             return seg[x];
48
49         int mid = l + (r-1)/2;
50         return merge(q(x+x,l,mid,i,j), q(x+x+1,mid+1,
51 r,i,j));
52     }
53
54     ll query(int l, int r){ //valor em uma posi
55     espec fica -> query de [l,l];
56         return q(1, 0, n-1, l, r);
57     }
58
59     void upd(ll x, int l, int r, int i, int j, ll u){
60         upd_lazy(x,l,r);
61         if(r < i || l > j) return;
62         if(l >= i && r <= j){
63             lazy[x] += u;
64             upd_lazy(x,l,r);
65         } else {
66             int mid = l + (r-1)/2;
67             upd(x+x,l,mid,i,j,u);
68             upd(x+x+1,mid+1,r,i,j,u);
69             seg[x] = merge(seg[x+x], seg[x+x+1]);
70         }
71     }
72
73     void upd_range(int l, int r, ll u){ //intervalo e
74     valor
75         upd(1,0,n-1,l,r,u);
76     }
77
78     };
79
80     1.12 Seghash
81
82     template<typename T> //use as SegtreeHash<int> h or
83     SegtreeHash<char>
84     struct SegtreeHash {
85
86         int n; //size do array que a seg vai ser criada
87         em cima
88
89         // P = 31, 53, 59, 73 .... (prime > number of
90         different characters)
91         // M = 578398229, 895201859, 1e9 + 7, 1e9 + 9 (
92         big prime)
93         int p, m;
94
95         vector<ll> seg, pot;
96
97         ll minValue = 0; // menor valor poss vel que
98         pode estar na estrutura
99         // isso   pra evitar que a hash
100         de '0' seja igual a de '0000...'
101
102         SegtreeHash(vector<T>& s, ll P = 31, ll MOD = (ll
103 )1e9 + 7){
104             n = (int)s.size();
105             p = P; m = MOD;
106             seg.resize(4 * n, -1);
107             pot.resize(4 * n);
108             pot[0] = 1;
109             for(int i = 1; i < (int)pot.size(); i++) {
110                 pot[i] = (pot[i - 1] * P) % MOD;
111             }
112             seg_build(1, 0, n - 1, s);
113         }
114
115         ll merge(ll a, ll b, int tam){
116             if(a == -1) return b;
117             if(b == -1) return a;
118             return (a + b * pot[tam]) % m;
119         }
120
121         void seg_build(int x, int l, int r, vector<T>& s)
122         {
123             if(r < l) return;
124             if(l == r){
125                 seg[x] = (int)s[l] - minValue + 1;
126             } else {
127                 int mid = l + (r-1)/2;
128                 seg_build(x+x, l, mid, s);
129                 seg_build(x+x+1, mid+1, r, s);
130                 seg[x] = merge(seg[x+x], seg[x+x+1], mid
131 - l + 1);
132             }
133         }
134
135         //n s atual, intervalo na  rvore e intervalo
136         pedido
137         ll q(int x, int l, int r, int i, int j){
138             if(r < i || l > j ) return -1;
139             if(l >= i && r <= j ) return seg[x];
140             int mid = l + (r-1)/2;
141             return merge(q(x+x,l,mid,i,j), q(x+x+1,mid+1,
142 r,i,j), mid - max(i, l) + 1);
143         }
144
145         //att posi pra val
146         void att(int x, int l, int r, int posi, T val){
147             if(l == r){
148                 seg[x] = (int)val - minValue + 1;
149             } else {
150                 int mid = l + (r-1)/2;
151                 if(posi <= mid)att(x+x,l,mid,posi,val);
152                 else att(x+x+1,mid+1,r,posi,val);
153                 seg[x] = merge(seg[x+x], seg[x+x+1], mid
154 - l + 1);
155             }
156         }
157     };
158
159     8

```



```

65 ll query(int l, int r){
66     return q(1, 0, n-1, l, r);
67 }
68
69 void update(int posi, T val){ //alterar em posi
70     att(1, 0, n-1, posi, val);
71 }
72
73 };

```

1.13 Segtree Lazy Iterative

```

1 // Segtree iterativa com lazy
2 //
3 // https://codeforces.com/gym/103708/problem/C
4 //
5 // O(N * log(N)) build
6 // O(log(N)) update e query
7
8 const int MAX = 524288; // NEED TO BE POWER OF 2 !!!
9 const int LOG = 19; // LOG = ceil(log2(MAX))
10
11 namespace seg {
12     ll seg[2*MAX], lazy[2*MAX];
13     int n;
14
15     ll junta(ll a, ll b) {
16         return a+b;
17     }
18
19     // soma x na posicao p de tamanho tam
20     void poe(int p, ll x, int tam, bool prop=1) {
21         seg[p] += x*tam;
22         if (prop and p < n) lazy[p] += x;
23     }
24
25     // atualiza todos os pais da folha p
26     void sobe(int p) {
27         for (int tam = 2; p /= 2; tam *= 2) {
28             seg[p] = junta(seg[2*p], seg[2*p+1]);
29             poe(p, lazy[p], tam, 0);
30         }
31     }
32
33     void upd_lazy(int i, int tam) {
34         if (lazy[i] && (2 * i + 1) < 2 * MAX) {
35             poe(2*i, lazy[i], tam);
36             poe(2*i+1, lazy[i], tam);
37             lazy[i] = 0;
38         }
39     }
40
41     // propaga o caminho da raiz ate a folha p
42     void prop(int p) {
43         int tam = 1 << (LOG-1);
44         for (int s = LOG; s; s--, tam /= 2) {
45             int i = p >> s;
46             upd_lazy(i, tam);
47         }
48     }
49
50     void build(int n2) {
51         n = n2;
52         for (int i = 0; i < n; i++) seg[n+i] = 0;
53         for (int i = n-1; i; i--) seg[i] = junta(seg[2*i], seg[2*i+1]);
54         for (int i = 0; i < 2*n; i++) lazy[i] = 0;
55     }
56
57     ll query(int a, int b) {
58         ll ret = 0;

```

```

59         for (prop(a+=n), prop(b+=n); a <= b; ++a/=2,
60             --b/=2) {
61             if (a%2 == 1) ret = junta(ret, seg[a]);
62             if (b%2 == 0) ret = junta(ret, seg[b]);
63         }
64         return ret;
65     }
66
67     void update(int a, int b, int x) {
68         int a2 = a += n, b2 = b += n, tam = 1;
69         for (; a <= b; ++a/=2, --b/=2, tam *= 2) {
70             if (a%2 == 1) poe(a, x, tam);
71             if (b%2 == 0) poe(b, x, tam);
72         }
73         sobe(a2), sobe(b2);
74     }
75
76     int findkth(int x, int l, int r, ll k, int tam){
77         int esq = x + x;
78         int dir = x + x + 1;
79
80         upd_lazy(x, tam);
81         upd_lazy(esq, tam/2);
82         upd_lazy(dir, tam/2);
83
84         if(l == r){
85             return l;
86         } else {
87             int mid = l + (r-l)/2;
88
89             if(seg[esq] >= k){
90                 return findkth(esq,l,mid,k, tam/2);
91             } else {
92                 return findkth(dir,mid+1, r, k - seg[
93                     esq], tam/2);
94             }
95         }
96     }
97
98     int findkth(ll k){
99         // kth smallest, O(logN)
100         // use position i to count how many times
101         // value 'i' appear
102         // merge must be the sum of nodes
103         return findkth(1,0,n-1,k,(1 << (LOG-1)));
104     }
105 };

```

1.14 Mergesorttree

```

1 //const int MAXN = 3e5 + 10;
2 //vector<int> seg[ 4 * MAXN + 10];
3
4 struct MergeSortTree {
5
6     int n; //size do array que a seg vai ser criada
7     em cima
8     vector< vector<int> > seg;
9     //vector< vector<ll> > ps; //prefix sum
10
11     MergeSortTree(vector<int>& s){
12         //se o input for grande (ou o tempo mt puxado
13         ), coloca a seg com size
14         //maximo de forma global
15         n = (int)s.size();
16         seg.resize(4 * n + 10);
17         //ps.resize(4 * n + 10);
18         seg_build(1,0,n-1,s);
19     }
20
21     vector<int> merge(vi& a, vi& b){
22         int i = 0, j = 0, p = 0;
23         vi ans(a.size() + b.size());

```

```

22     while(i < (int)a.size() && j < (int)b.size()) {
23         if(a[i] < b[j]){
24             ans[p++] = a[i++];
25         } else {
26             ans[p++] = b[j++];
27         }
28     }
29     while(i < (int)a.size()){
30         ans[p++] = a[i++];
31     }
32     while(j < (int)b.size()){
33         ans[p++] = b[j++];
34     }
35     return ans;
36 }

37 vector<ll> calc(vi& s) {
38     ll sum = 0;
39     vector<ll> tmp;
40     for(auto &x : s) {
41         sum += x;
42         tmp.push_back(sum);
43     }
44     return tmp;
45 }

46 void seg_build(int x, int l, int r, vector<int>&
47 s){
48     if(r < l) return;
49     if(l == r){
50         seg[x].push_back(s[l]);
51         //ps[x] = {s[l]};
52     } else {
53         int mid = l + (r-l)/2;
54         seg_build(x+x, l, mid, s);
55         seg_build(x+x+1, mid+1, r, s);
56         seg[x] = merge(seg[x+x], seg[x+x+1]);
57         //ps[x] = calc(seg[x]);
58     }
59 }

60 //nÃs atual, intervalo na Ãrvore e intervalo
61 pedido
62 // retorna a quantidade de numeros <= val em [l,
63 r]
64
65 ll q(int x, int l, int r, int i, int j, int val){
66     if(r < i || l > j) return 0;
67     if(l >= i && r <= j){
68         return (lower_bound(seg[x].begin(), seg[x
69 ].end(), val) - seg[x].begin());
70     }
71     int mid = l + (r-l)/2;
72     return q(x+x,l,mid,i,j, val) + q(x+x+1,mid+1,
73 r,i,j, val);
74 }

75 // retorna a soma dos numeros <= val em [l, r]
76 // nÃs atual, intervalo na Ãrvore e intervalo
77 pedido
78 /*
79 ll q(int x, int l, int r, int i, int j, ll val){
80     if(r < i || l > j) return 0;
81     if(l >= i && r <= j){
82         auto it = upper_bound(seg[x].begin(), seg
83 [x].end(), val) - seg[x].begin();
84         if(val > seg[x].back()) {
85             return ps[x].back();
86         }
87     }
88     if(val < seg[x][0]) {
89         return 0;
90     }
91     return ps[x][it - 1];
92 }
93
94 int mid = l + (r-l)/2;
95 return q(x+x,l,mid,i,j, val) + q(x+x+1,mid+1,
96 r,i,j, val);
97 }
98 */
99
100 ll query(int l, int r, ll val){
101     return q(1, 0, n-1, l, r, val);
102 }
103
104 };

```

1.15 Treap Cp

```

1 mt19937 rng((int) chrono::steady_clock::now().
2 time_since_epoch().count());
3
4 typedef struct item * pitem;
5
6 struct item {
7     int prior, value, cnt;
8     bool rev;
9     pitem l, r;
10
11 // Construtor para inicializar um nÃs com um
12 valor dado
13 item(int _val) {
14     prior = rng();
15     value = _val;
16     cnt = 1; // Inicializa o contador como 1
17     rev = false; // Define o reverso como falso
18     por padrÃo
19     l = r = nullptr;
20 }
21
22 };
23
24 int cnt (pitem it) {
25     return it ? it->cnt : 0;
26 }
27
28 void upd_cnt (pitem it) {
29     if (it)
30         it->cnt = cnt(it->l) + cnt(it->r) + 1;
31 }
32
33 void push (pitem it) {
34     if (it && it->rev) {
35         it->rev = false;
36         swap (it->l, it->r);
37         if (it->l) it->l->rev ^= true;
38         if (it->r) it->r->rev ^= true;
39     }
40 }
41
42 void merge (pitem & t, pitem l, pitem r) {
43     push (l);
44     push (r);
45     if (!l || !r)
46         t = l ? l : r;
47     else if (l->prior > r->prior)
48         merge (l->r, l->r, r), t = l;
49     else
50         merge (r->l, l, r->l), t = r;
51     upd_cnt (t);
52 }

```

```

49 // essa func quebra um range baseado na key e salva
50 // as duas partes em l, r
51 void split (pitem t, pitem & l, pitem & r, int key,
52             int add = 0) {
53     if (!t)
54         return void( l = r = 0 );
55     push (t);
56     int cur_key = add + cnt(t->l);
57     if (key <= cur_key)
58         split (t->l, l, t->l, key, add), r = t;
59     else
60         split (t->r, t->r, r, key, add + 1 + cnt(t->l));
61     upd_cnt (t);
62 }
63 // essa inverte o range l, r do nó t
64 void reverse (pitem t, int l, int r) {
65     pitem t1, t2, t3;
66     split (t, t1, t2, l);
67     split (t2, t2, t3, r-l+1);
68     t2->rev ^= true;
69     merge (t, t1, t2);
70     merge (t, t, t3);
71 }
72 vector<int> ans;
73 void output (pitem t) {
74     if (!t) return;
75     push (t);
76     output (t->l);
77     // pode printar o valor direto aq tmb
78     ans.push_back(t->value);
79     output (t->r);
80 }
81 // https://cses.fi/problemset/task/2072/
82 // cortar o range [l, r] e cola no final
83 void cut_and_paste(pitem root, int l, int r) {
84     pitem A, B, C, D;
85     // separa a root em caras com indice < l r >= l
86     // e salva as partes em A, B
87     split(root, A, B, l);
88     // pega a parte B (indices i >= l) e pega
89     // exatamente o tamanho que vc quer
90     // salva as partes em C e D
91     split(B, C, D, r - l + 1);
92     // Da merge dos indices i < l com a parte i > r
93     merge(root, A, D);
94     // da merge do pedaço que vc queria final e
95     // deixa salvo em root
96     merge(root, root, C);
97 }
98 void solve() {
99     int n, q;
100     cin >> n >> q;
101     string s;
102     cin >> s;
103     pitem root = nullptr;
104     for(int i = 0; i < n; i++) {
105         pitem newNode = new item(i);
106         merge(root, root, newNode);
107     }
108     while(q--) {
109         int l, r;

```

```

118     cin >> l >> r;
119     cut_and_paste(root, l - 1, r - 1);
120 }
121 output(root);
122 for(int i = 0; i < n; i++) {
123     cout << s[ans[i]];
124 }
125 cout << "\n";

```

1.16 Ordered Set

```

1 // Ordered Set
2 //
3 // set roubado com mais operacoes
4 //
5 // para alterar para multiset
6 // trocar less para less_equal
7 //
8 // ordered_set<int> s
9 //
10 // order_of_key(k) // number of items strictly
11 // smaller than k -> int
12 // find_by_order(k) // k-th element in a set (
13 // counting from zero) -> iterator
14 //
15 // https://cses.fi/problemset/task/2169
16 // 0(log N) para insert, erase (com iterator),
17 // order_of_key, find_by_order
18 using namespace __gnu_pbds;
19 template <typename T>
20 using ordered_set = tree<T, null_type, less<T>,
21 rb_tree_tag, tree_order_statistics_node_update>;
22 void erase(ordered_set& a, int x){
23     int r = a.order_of_key(x);
24     auto it = a.find_by_order(r);
25     a.erase(it);

```

1.17 Trie Old

```

1 struct Trie {
2
3     int nxt = 1, sz, maxLet = 26; //tamanho do
4     alfabeto
5     vector< vector<int> > trie;
6     bitset<(int)1e7> finish; //modificar esse valor
7     pra ser >= n
8     //garantir que vai submeter em cpp 64
9
10     Trie(int n){
11         sz = n;
12         trie.assign(sz, vector<int>(maxLet,0));
13     }
14
15     void add(string &s){
16         int cur = 0;
17         for(auto c: s){
18             //alterar esse azinho dependendo da
19             entrada!!
20             if(trie[cur][c-'a'] == 0){
21                 trie[cur][c-'a'] = nxt++;
22                 cur = trie[cur][c-'a'];
23             } else {
24                 cur = trie[cur][c-'a'];

```

```

22     }
23 }
24 finish[cur] = 1;
25 }
26
27 int search(string& s){
28     int cur = 0;
29     for(auto c: s){
30         if(trie[cur][c - 'a'] == 0){
31             return 0;
32         }
33         cur = trie[cur][c - 'a'];
34     }
35     return finish[cur];
36 }
37
38 };

```

1.18 Range Color Update

```

1 // Range color update (brunomaletta)
2 //
3 // update(l, r, c) colore o range [l, r] com a cor c,
4 // e retorna os ranges que foram coloridos {l, r, cor}
5 // query(i) retorna a cor da posicao i
6 //
7 // Complexidades (para q operacoes):
8 // update - O(log(q)) amortizado
9 // query - O(log(q))
10
11 template<typename T> struct color {
12     set<tuple<int, int, T>> se;
13
14     vector<tuple<int, int, T>> update(int l, int r, T val) {
15         auto it = se.upper_bound({r, INF, val});
16         if (it != se.begin() and get<1>(*prev(it)) > r) {
17             auto [L, R, V] = *--it;
18             se.erase(it);
19             se.emplace(L, r, V), se.emplace(r+1, R, V);
20         }
21         it = se.lower_bound({l, -INF, val});
22         if (it != se.begin() and get<1>(*prev(it)) >= l) {
23             auto [L, R, V] = *--it;
24             se.erase(it);
25             se.emplace(L, l-1, V), it = se.emplace(l, R, V).first;
26         }
27         vector<tuple<int, int, T>> ret;
28         for (; it != se.end() and get<0>(*it) <= r; it = se.erase(it))
29             ret.push_back(*it);
30         se.emplace(l, r, val);
31         return ret;
32     }
33     T query(int i) {
34         auto it = se.upper_bound({i, INF, T{}});
35         if (it == se.begin() or get<1>(*--it) < i)
36             return -1; // nao tem
37         return get<2>(*it);
38     }
39 };

```

1.19 Cht

```

1 // CHT (tiagodfs)
2
3 const ll is_query = -LLINF;

```

```

4 struct Line{
5     ll m, b;
6     mutable function<const Line*> succ;
7     bool operator<(const Line& rhs) const{
8         if(rhs.b != is_query) return m < rhs.m;
9         const Line* s = succ();
10        if(!s) return 0;
11        ll x = rhs.m;
12        return b - s->b < (s->m - m) * x;
13    }
14 };
15 struct Cht : public multiset<Line>{ // maintain max m
16     *x+b
17     bool bad(iterator y){
18         auto z = next(y);
19         if(y == begin()){
20             if(z == end()) return 0;
21             return y->m == z->m && y->b <= z->b;
22         }
23         auto x = prev(y);
24         if(z == end()) return y->m == x->m && y->b <=
25             x->b;
26         return (ld)(x->b - y->b)*(z->m - y->m) >= (ld)
27             (y->b - z->b)*(y->m - x->m);
28     }
29     void insert_line(ll m, ll b){ // min -> insert (-
30         m, -b) -> -eval()
31         auto y = insert({ m, b });
32         y->succ = [=]{ return next(y) == end() ? 0 :
33             &*next(y); };
34         if(bad(y)){ erase(y); return; }
35         while(next(y) != end() && bad(next(y))) erase
36             (next(y));
37         while(y != begin() && bad(prev(y))) erase(
38             prev(y));
39     }
40     ll eval(ll x){
41         auto l = *lower_bound((Line) { x, is_query })
42         ;
43         return l.m * x + l.b;
44     }
45 };

```

1.20 Bit

```

1 struct BIT {
2     int n, LOGN = 0;
3     vector<ll> bit;
4
5     BIT(int nn){
6         n = nn + 10;
7         bit.resize(n + 10, 0);
8         while( (1LL << LOGN) <= n ) LOGN++;
9     }
10
11     ll query(int x){
12         x++;
13         ll ans = 0;
14         while(x > 0){
15             ans += bit[x];
16             x -= (x & (-x));
17         }
18         return ans;
19     }
20
21     void update(int x, ll val){
22         x++;
23         while(x < (int)bit.size()){
24             bit[x] += val;
25             x += (x & (-x));
26         }
27     }
28 }

```

```

29 int findkth(int k){
30     //kth smallest, O(logN)
31     //use position i to count how many times
    value 'i' appear
32     int sum = 0, pos = 0;
33     for(int i = LOGN; i >= 0; i--){
34         if(pos + (1LL << i) < n && sum + bit[pos
+ (1LL << i)] < k){
35             sum += bit[pos + (1LL << i)];
36             pos += (1LL << i);
37         }
38     }
39     return pos;
40 }
41 /*
42 int findkth(int k){
43     //kth smallest, O(log^2(N))
44     //use position i to count how many times
    value 'i' appear
45     int x = 0, mx = 200;
46     for(int b = n; b > 0 && mx > 0; b /= 2){
47         while( x+b < n && query(x+b) < k && mx--
> 0 ){
48             x += b;
49         }
50     }
51     return x+1;
52 }
53 */
54 };

```

1.21 Triexor

```

1 struct Trie {
2
3     int nxt = 1, sz, maxLet = 2;
4     vector< vector<int> > trie;
5     vector<int> finish, paths;
6
7     Trie(int n){
8         sz = n;
9         trie.assign(sz + 10, vector<int>(maxLet,0));
10        finish.resize(sz + 10);
11        paths.resize(sz+10);
12    }
13
14    void add(int x){
15        int cur = 0;
16        for(int i = 31; i >= 0; i--){
17            int b = ( (x & (1 << i)) > 0);
18            if(trie[cur][b] == 0)
19                trie[cur][b] = nxt++;
20            cur = trie[cur][b];
21            paths[cur]++;
22        }
23        paths[cur]++;
24    }
25
26    void rem(int x){
27        int cur = 0;
28        for(int i = 31; i >= 0; i--){
29            int b = ( (x & (1 << i)) > 0);
30            cur = trie[cur][b];
31            paths[cur]--;
32        }
33        finish[cur]--;
34        paths[cur]--;
35    }
36
37    int query(int x){ //return the max xor with x
38        int ans = 0, cur = 0;
39
40        for(int i = 31; i >= 0; i--){

```

```

41        int b = ( (x & (1 << i)) > 0);
42        int bz = trie[cur][0];
43        int bo = trie[cur][1];
44
45        if(bz > 0 && bo > 0 && paths[bz] > 0 &&
paths[bo] > 0){
46            //cout << "Optimal" << endl;
47            cur = trie[cur][b ^ 1];
48            ans += (1 << i);
49        } else if(bz > 0 && paths[bz] > 0){
50            //cout << "Zero" << endl;
51            cur = trie[cur][0];
52            if(b) ans += (1 << i);
53        } else if(bo > 0 && paths[bo] > 0){
54            //cout << "One" << endl;
55            cur = trie[cur][1];
56            if(!b) ans += (1 << i);
57        } else {
58            break;
59        }
60    }
61
62    return ans;
63 }
64
65 };

```

1.22 Querytree

```

1 struct QueryTree {
2     int n, t = 0, l = 3, build = 0, euler = 0;
3     vector<ll> dist;
4     vector<int> in, out, d;
5     vector<vector<int>> sobe;
6     vector<vector<pair<int,ll>>> arr;
7     vector<vector<ll>> table_max; //max edge
8     vector<vector<ll>> table_min; //min edge
9
10    QueryTree(int nn) {
11        n = nn + 5;
12        arr.resize(n);
13        in.resize(n);
14        out.resize(n);
15        d.resize(n);
16        dist.resize(n);
17        while( (1 << l) < n ) l++;
18        sobe.assign(n + 5, vector<int>(l+1));
19        table_max.assign(n + 5, vector<ll>(l));
20        table_min.assign(n + 5, vector<ll>(l));
21    }
22
23    void add_edge(int u, int v, ll w){ //
    bidirectional edge with weight w
24        arr[u].push_back({v, w});
25        arr[v].push_back({u, w});
26    }
27
28    //assert the root of tree is node 1 or change the
    'last' in the next function
29    void Euler_Tour(int u, int last = 1, ll we = 0,
    int depth = 0, ll sum = 0){ //euler tour
30        euler = 1; //remember to use this function
    before the queries
31        in[u] = t++;
32        d[u] = depth;
33        dist[u] = sum; //sum = sum of the values in
    edges from root to node u
34        sobe[u][0] = last; //parent of u. parent of 1
    is 1
35        table_max[u][0] = we;
36        table_min[u][0] = we;
37        for(auto v: arr[u]) if(v.ff != last){

```

```

38         Euler_Tour(v.ff, u, v.ss, depth + 1, sum
+ v.ss);
39     }
40     out[u] = t++;
41 }
42
43 void build_table(){ //binary lifting
44     assert(euler);
45     build = 1; //remeber use this function before
queries
46     for(int k = 1; k < l; k++){
47         for(int i = 1; i <= n; i++){
48             sobe[i][k] = sobe[sobe[i][k-1]][k-1];
49             table_max[i][k] = max(table_max[i][k
- 1], table_max[sobe[i][k-1]][k-1]);
50             table_min[i][k] = min(table_min[i][k
- 1], table_min[sobe[i][k-1]][k-1]);
51         }
52     }
53 }
54
55 int is_ancestor(int u, int v){ // return 1 if u
is ancestor of v
56     assert(euler);
57     return in[u] <= in[v] && out[u] >= out[v];
58 }
59
60 int lca(int u, int v){ //return lca of u and v
61     assert(build && euler);
62     if(is_ancestor(u,v)) return u;
63     if(is_ancestor(v,u)) return v;
64     int lca = u;
65     for(int k = l - 1; k >= 0; k--){
66         int tmp = sobe[lca][k];
67         if(!is_ancestor(tmp, v)){
68             lca = tmp;
69         }
70     }
71     return sobe[lca][0];
72 }
73
74 int lca(int u, int v, int root) { //return lca of
u and v when tree is rooted at 'root'
75     return lca(u, v) ^ lca(v, root) ^ lca(root, u
); //magic
76 }
77
78 int up_k(int u, int qt){ //return node k levels
higher starting from u
79     assert(build && euler);
80     for(int b = 0; b < l; b++){
81         if(qt%2) u = sobe[u][b];
82         qt >>= 1;
83     }
84     return u;
85 }
86
87 ll goUpMax(int u, int to){ //return the max
weigth of a edge going from u to 'to'
88     assert(build);
89     if(u == to) return 0;
90     ll mx = table_max[u][0];
91     for(int k = l - 1; k >= 0; k--){
92         int tmp = sobe[u][k];
93         if( !is_ancestor(tmp, to) ){
94             mx = max(mx, table_max[u][k]);
95             u = tmp;
96         }
97     }
98     return max(mx, table_max[u][0]);
99 }
100
101 ll max_edge(int u, int v){ //return the max
weight of a edge in the simple path from u to v
102     assert(build);
103     int ancestor = lca(u, v);
104     ll a = goUpMax(u, ancestor), b = goUpMax(v,
ancestor);
105     if(ancestor == u) return b;
106     else if(ancestor == v) return a;
107     return max(a,b);
108 }
109
110 ll goUpMin(int u, int to){ //return the min
weight of a edge going from u to 'to'
111     assert(build);
112     if(u == to) return oo;
113     ll mx = table_min[u][0];
114     for(int k = l - 1; k >= 0; k--){
115         int tmp = sobe[u][k];
116         if( !is_ancestor(tmp, to) ){
117             mx = min(mx, table_min[u][k]);
118             u = tmp;
119         }
120     }
121     return min(mx, table_min[u][0]);
122 }
123
124 ll min_edge(int u, int v){ //return the min
weight of a edge in the simple path from u to v
125     assert(build);
126     int ancestor = lca(u, v);
127     ll a = goUpMin(u, ancestor), b = goUpMin(v,
ancestor);
128     if(ancestor == u) return b;
129     else if(ancestor == v) return a;
130     return min(a,b);
131 }
132
133 ll query_dist(int u, int v){ //distance of nodes
u and v
134     int x = lca(u, v);
135     return dist[u] - dist[x] + dist[v] - dist[x];
136 }
137
138 int kth_between(int u, int v, int k){ //kth node
in the simple path from u to v; if k = 1, ans = u
139     k--;
140     int x = lca(u, v);
141     if( k > d[u] - d[x] ){
142         k -= (d[u] - d[x]);
143         return up_k(v, d[v]-d[x]-k);
144     }
145     return up_k(u, k);
146 }
147
148 };
149
150 int main() {
151     ios::sync_with_stdio(false);
152     cin.tie(NULL);
153
154     int t = 1, n, u, v, w, k;
155     string s;
156     cin >> t;
157     while(t--){
158         cin >> n;
159         QueryTree arr(n);
160         for(int i = 1; i < n; i++){
161             cin >> u >> v >> w;
162             arr.add_edge(u,v,w);
163         }
164         arr.Euler_Tour(1);
165         arr.build_table();
166         while(cin >> s, s != "DONE"){
167             cin >> u >> v;

```

```

168         if(s == "DIST") {
169             cout << arr.query_dist(u, v) << "\n";
170         } else {
171             cin >> k;
172             cout << arr.kth_between(u,v,k) << "\n";
173         }
174     }
175     cout << "\n";
176 }
177
178 }

```

1.23 Sparse

```

1 struct Sparse {
2
3     vector<vector<int>> arr;
4
5     int op(int& a, int& b){ //min, max, gcd, lcm, and
6         , or
7         return min(a,b);
8         //return __gcd(a,b);
9         //return max(a,b);
10    }
11
12    Sparse(vector<int>& v){ //Constrói a tabela
13        int n = v.size(), logn = 0;
14        while((1<<logn) <= n) logn++;
15        arr.assign(n, vector<int>(logn, 0));
16        for(int i = 0; i < n; i++){
17            arr[i][0] = v[i];
18            for(int k = 1; k < logn; k++){
19                for(int i = 0; i < n; i++){
20                    if(i + ( 1 << k) -1 >= n)
21                        break;
22                    int p = i+( 1 << (k-1) );
23                    arr[i][k] = op( arr[i][ k-1 ] , arr[p
24                        ][k-1] );
25                }
26            }
27        }
28
29        int query(int l, int r){
30            int pot = 31 - __builtin_clz(r-l+1); //r-l+1
31            sÃo INTEIROS, nÃo ll
32            int k = (1 << pot) ;
33            return op( arr[l][pot] , arr[ r - (k-1) ][
34                pot] );
35        }
36    };
37 }

```

1.24 Trie

```

1 struct Trie {
2
3     struct Node {
4         map<char, Node> adj; // dÃ pra trocar por
5         vector(26)
6         ll finishHere;
7
8         Node() {
9             finishHere = 0;
10        }
11
12        bool find(char c) {
13            return adj.find(c) != adj.end();
14        }
15    };
16 }

```

```

Node mainNode;

Trie(){
    mainNode = Node();
}

void add(string &s) {
    Node *curNode = &mainNode;

    for(auto &c : s) {
        if(!curNode->find(c)) {
            curNode->adj[c] = Node();
        }

        curNode = &curNode->adj[c];
    }

    curNode->finishHere += 1;
}

void dfs(Node& node) {
    for(auto &v : node.adj) {
        dfs(v.ss);
        // faz alguma coisa
    }
}

void dfs() {
    return dfs(mainNode);
}

bool search(string &s) {
    Node* curNode = &mainNode;

    for(auto &c : s) {
        if(!curNode->find(c))
            return false;

        curNode = &curNode->adj[c];
    }

    return curNode->finishHere > 0;
}

void debugRec(Node node, int depth) {
    for(auto &x : node.adj) {
        cout << string(3 * depth, ' ') << x.ff <<
        " " << x.ss.finishHere << "\n";
        debugRec(x.ss, depth + 1);
    }
}

void debug() {
    debugRec(mainNode, 0);
}

};

```

1.25 Kruskal

```

1 struct Edge {
2     int u, v;
3     ll weight;
4
5     Edge() {}
6
7     Edge(int u, int v, ll weight) : u(u), v(v),
8     weight(weight) {}
9
10    bool operator<(Edge const& other) {
11        return weight < other.weight;
12    }
13 }

```

```

12 };
13
14 vector<Edge> kruskal(vector<Edge> edges, int n) {
15     vector<Edge> result;
16     ll cost = 0;
17
18     sort(edges.begin(), edges.end());
19     DSU dsu(n);
20
21     for (auto e : edges) {
22         if (!dsu.same(e.u, e.v)) {
23             cost += e.weight;
24             result.push_back(e);
25             dsu.unite(e.u, e.v);
26         }
27     }
28
29     return result;
30 }

```

2 DP

2.1 Lcs

```

1 // LCS (Longest Common Subsequence)
2 //
3 // maior subsequencia comum entre duas strings
4 //
5 // tamanho da matriz da dp eh |a| x |b|
6 // lcs(a, b) = string da melhor resposta
7 // dp[a.size()][b.size()] = tamanho da melhor
  resposta
8 //
9 // https://atcoder.jp/contests/dp/tasks/dp_f
10 //
11 // O(n^2)
12
13 string lcs(string a, string b) {
14     int n = a.size();
15     int m = b.size();
16
17     int dp[n+1][m+1];
18     pair<int, int> p[n+1][m+1];
19
20     memset(dp, 0, sizeof(dp));
21     memset(p, -1, sizeof(p));
22
23     for (int i = 1; i <= n; i++) {
24         for (int j = 1; j <= m; j++) {
25             if (a[i-1] == b[j-1]) {
26                 dp[i][j] = dp[i-1][j-1] + 1;
27                 p[i][j] = {i-1, j-1};
28             } else {
29                 if (dp[i-1][j] > dp[i][j-1]) {
30                     dp[i][j] = dp[i-1][j];
31                     p[i][j] = {i-1, j};
32                 } else {
33                     dp[i][j] = dp[i][j-1];
34                     p[i][j] = {i, j-1};
35                 }
36             }
37         }
38     }
39
40     // recuperar resposta
41
42     string ans = "";
43     pair<int, int> curr = {n, m};
44
45     while (curr.first != 0 && curr.second != 0) {
46         auto [i, j] = curr;
47

```

```

48         if (a[i-1] == b[j-1]) {
49             ans += a[i-1];
50         }
51
52         curr = p[i][j];
53     }
54
55     reverse(ans.begin(), ans.end());
56
57     return ans;
58 }

```

2.2 Lis Binary Search

```

1 int lis(vector<int> arr) {
2     vector<int> dp;
3
4     for (auto e : arr) {
5         int pos = lower_bound(dp.begin(), dp.end(), e
6         ) - dp.begin();
7
8         if (pos == (int)dp.size()) {
9             dp.push_back(e);
10        } else {
11            dp[pos] = e;
12        }
13    }
14
15    return (int)dp.size();

```

2.3 Edit Distance

```

1 // Edit Distance / Levenshtein Distance
2 //
3 // numero minimo de operacoes
4 // para transformar
5 // uma string em outra
6 //
7 // tamanho da matriz da dp eh |a| x |b|
8 // edit_distance(a.size(), b.size(), a, b)
9 //
10 // https://cses.fi/problemset/task/1639
11 //
12 // O(n^2)
13
14 int tb[MAX][MAX];
15
16 int edit_distance(int i, int j, string &a, string &b)
17 {
18     if (i == 0) return j;
19     if (j == 0) return i;
20
21     int &ans = tb[i][j];
22
23     if (ans != -1) return ans;
24
25     ans = min({
26         edit_distance(i-1, j, a, b) + 1,
27         edit_distance(i, j-1, a, b) + 1,
28         edit_distance(i-1, j-1, a, b) + (a[i-1] != b[
29         j-1])
30     });
31
32     return ans;
33 }

```

2.4 Digit Dp

```

1 // Digit DP 1: https://atcoder.jp/contests/dp/tasks/
  dp_s
2 //

```



```

3 // find the number of integers between 1 and K (
  inclusive)
4 // where the sum of digits in base ten is a multiple
  of D
5
6 #include <bits/stdc++.h>
7
8 using namespace std;
9
10 const int MOD = 1e9+7;
11
12 string k;
13 int d;
14
15 int tb[10010][110][2];
16
17 int dp(int pos, int sum, bool under) {
18     if (pos >= k.size()) return sum == 0;
19
20     int& mem = tb[pos][sum][under];
21     if (mem != -1) return mem;
22     mem = 0;
23
24     int limit = 9;
25     if (!under) limit = k[pos] - '0';
26
27     for (int digit = 0; digit <= limit; digit++) {
28         mem += dp(pos+1, (sum + digit) % d, under | (
29             digit < limit));
30         mem %= MOD;
31     }
32
33     return mem;
34 }
35
36 int main() {
37     ios::sync_with_stdio(false);
38     cin.tie(NULL);
39
40     cin >> k >> d;
41
42     memset(tb, -1, sizeof(tb));
43
44     cout << (dp(0, 0, false) - 1 + MOD) % MOD << '\n';
45
46     return 0;

```

2.5 Range Dp

```

1 // Range DP 1: https://codeforces.com/problemset/
  problem/1132/F
2 //
3 // You may apply some operations to this string
4 // in one operation you can delete some contiguous
  substring of this string
5 // if all letters in the substring you delete are
  equal
6 // calculate the minimum number of operations to
  delete the whole string s
7
8 #include <bits/stdc++.h>
9
10 using namespace std;
11
12 const int MAX = 510;
13
14 int n, tb[MAX][MAX];
15 string s;
16
17 int dp(int left, int right) {
18     if (left > right) return 0;

```

```

19
20     int& mem = tb[left][right];
21     if (mem != -1) return mem;
22
23     mem = 1 + dp(left+1, right); // gastar uma
  operaçãõ arrumando sã o cara atual
24     for (int i = left+1; i <= right; i++) {
25         if (s[left] == s[i]) {
26             mem = min(mem, dp(left+1, i-1) + dp(i,
27                 right));
28         }
29     }
30
31     return mem;
32 }
33
34 int main() {
35     ios::sync_with_stdio(false);
36     cin.tie(NULL);
37
38     cin >> n >> s;
39     memset(tb, -1, sizeof(tb));
40     cout << dp(0, n-1) << '\n';
41
42     return 0;

```

2.6 Lis Segtree

```

1 int n, arr[MAX], aux[MAX]; cin >> n;
2 for (int i = 0; i < n; i++) {
3     cin >> arr[i];
4     aux[i] = arr[i];
5 }
6
7 sort(aux, aux+n);
8
9 Segtree st(n); // seg of maximum
10
11 int ans = 0;
12 for (int i = 0; i < n; i++) {
13     int it = lower_bound(aux, aux+n, arr[i]) - aux;
14     int lis = st.query(0, it) + 1;
15
16     st.update(it, lis);
17
18     ans = max(ans, lis);
19 }
20
21 cout << ans << '\n';

```

2.7 Knapsack

```

1 //Submeter em c++ 64bits otimiza o long long
2 ll knapsack(vector<ll>& weight, vector<ll>& value,
  int W) {
3     //Usar essa knapsack se sã precisar do resultado
  final.
4     //O(W) em memãria
5     vector<vector<ll>> table(2, vector<ll>(W + 1, 0))
  ;
6     int n = (int)value.size();
7
8     for(int k = 1; k <= n; k++) {
9         for(int i = 0; i <= W; i++) {
10             if(i - weight[k - 1] >= 0) {
11                 table[k % 2][i] = max(table[(k - 1)
12                     % 2][i],
13                     value[k - 1] + table[(k - 1) %
14                     2][i - weight[k - 1]]);
15             } else {
16                 table[k % 2][i] = max(table[(k - 1) %
17                     2][i], table[k % 2][i]);

```

```

15     }
16 }
17 }
18
19 return table[n % 2][W];
20 }
21
22 ll knapsack(vector<ll>& weight, vector<ll>& value,
23 int W) {
24     //Usar essa knapsack se, em algum momento,
25     //precisar recuperar os indices
26     //O(NW) em memÃria
27
28     int n = (int)value.size();
29     vector<vector<ll>> table(W + 1, vector<ll>(n + 1,
30     0));
31
32     for(int k = 1; k <= n; k++) {
33         for(int i = 0; i <= W; i++) {
34             if(i - weight[k - 1] >= 0) {
35                 table[i][k] = max(table[i][k - 1],
36                 value[k - 1] + table[i - weight[k - 1]][k - 1]);
37             } else {
38                 table[i][k] = max(table[i][k - 1],
39                 table[i][k]);
40             }
41         }
42     }
43
44     /*
45     int per = W;
46     vector<int> idx;
47     for(int k = n; k > 0; k--) {
48         if(table[per][k] == table[per][k - 1]){
49             continue;
50         } else {
51             idx.push_back(k - 1);
52             per -= weight[k - 1];
53         }
54     }
55     */
56
57     return table[W][n];
58 }
59
60 const int MOD = 998244353;
61
62 struct Knapsack {
63
64     int S; // max value
65     vector<ll> dp;
66
67     Knapsack(int S_) {
68         S = S_ + 5;
69         dp.assign(S, 0);
70         dp[0] = 1;
71     }
72
73     void Add(int val) {
74         if(val <= 0 || val >= S) return;
75         for(int i = S - 1; i >= val; i--) {
76             dp[i] += dp[i - val];
77             dp[i] %= MOD;
78         }
79     }
80
81     void Rem(int val) {
82         if(val <= 0 || val >= S) return;
83         for(int i = val; i < S; i++) {
84             dp[i] += MOD - dp[i - val];
85             dp[i] %= MOD;

```

```

83     }
84 }
85
86 int Query(int val) {
87     // # of ways to select a subset of numbers
88     with sum = val
89     if(val <= 0 || val >= S) return 0;
90     return dp[val];
91 }
92
93 };
94
95 void solve() {
96
97     int n, w;
98     cin >> n >> w;
99     vector<ll> weight(n), value(n);
100     for(int i = 0; i < n; i++) {
101         cin >> weight[i] >> value[i];
102     }
103     cout << knapsack(weight, value, w) << "\n";
104 }

```

2.8 Digit Dp 2

```

1 // Digit DP 2: https://cses.fi/problemset/task/2220
2 //
3 // Number of integers between a and b
4 // where no two adjacent digits are the same
5
6 #include <bits/stdc++.h>
7
8 using namespace std;
9 using ll = long long;
10
11 const int MAX = 20; // 10^18
12
13 ll tb[MAX][MAX][2][2];
14
15 ll dp(string& number, int pos, int last_digit, bool
16 under, bool started) {
17     if (pos >= (int)number.size()) {
18         return 1;
19     }
20
21     ll& mem = tb[pos][last_digit][under][started];
22     if (mem != -1) return mem;
23     mem = 0;
24
25     int limit = 9;
26     if (!under) limit = number[pos] - '0';
27
28     for (int digit = 0; digit <= limit; digit++) {
29         if (started && digit == last_digit) continue;
30
31         bool is_under = under || (digit < limit);
32         bool is_started = started || (digit != 0);
33
34         mem += dp(number, pos+1, digit, is_under,
35 is_started);
36     }
37
38     return mem;
39 }
40
41 ll solve(ll ubound) {
42     memset(tb, -1, sizeof(tb));
43     string number = to_string(ubound);
44     return dp(number, 0, 10, 0, 0);
45 }
46
47 int main() {

```

```

46 ios::sync_with_stdio(false);
47 cin.tie(NULL);
48
49 ll a, b; cin >> a >> b;
50 cout << solve(b) - solve(a-1) << '\n';
51
52 return 0;
53 }

```

3 General

3.1 Last True

```

1 // Binary Search (last_true)
2
3 // last_true(2, 10, [](int x) { return x * x <= 30;
4 // }); // outputs 5
5 //
6 // [1, r]
7 // if none of the values in the range work, return lo
8 // - 1
9 //
10 // f(1) = true
11 // f(2) = true
12 // f(3) = true
13 // f(4) = true
14 // f(5) = true
15 // f(6) = false
16 // f(7) = false
17 // f(8) = false
18 //
19 // last_true(1, 8, f) = 5
20 // last_true(7, 8, f) = 6
21
22 int last_true(int lo, int hi, function<bool(int)> f)
23 {
24     lo--;
25     while (lo < hi) {
26         int mid = lo + (hi - lo + 1) / 2;
27
28         if (f(mid)) {
29             lo = mid;
30         } else {
31             hi = mid - 1;
32         }
33     }
34     return lo;
35 }

```

3.2 Input By File

```

1 freopen("file.in", "r", stdin);
2 freopen("file.out", "w", stdout);

```

3.3 Mix Hash

```

1 // magic hash function using mix
2
3 using ull = unsigned long long;
4 ull mix(ull o){
5     o+=0x9e3779b97f4a7c15;
6     o=(o^(o>>30))*0xbf58476d1ce4e5b9;
7     o=(o^(o>>27))*0x94d049bb133111eb;
8     return o^(o>>31);
9 }
10 ull hash(pii a) {return mix(a.first ^ mix(a.second))
11 ;}

```

3.4 Random

```

1 int main() {
2     ios::sync_with_stdio(false);
3     cin.tie(NULL);
4
5     //mt19937 rng(chrono::steady_clock::now().
6     time_since_epoch().count()); //gerar int
7     mt19937_64 rng(chrono::steady_clock::now().
8     time_since_epoch().count()); //gerar ll
9
10    /*usar rng() pra gerar numeros aleatórios*/
11    /*usar rng() % x pra gerar numeros em [0, x-1]*/
12    for(int i = 0; i < 10; i++){
13        cout << rng() << endl;
14    }
15    vector<ll> arr = {1,2,3,4,5,6,7,8,9};
16    /*dã pra usar no shuffle de vector também*/
17    shuffle(arr.begin(), arr.end(),rng);
18    for(auto &x: arr)
19        cout << x << endl;
20 }

```

3.5 Template

```

1 #include <bits/stdc++.h>
2 #define ff first
3 #define ss second
4
5 using namespace std;
6 using ll = long long;
7 using ld = long double;
8 using pii = pair<int,int>;
9 using vi = vector<int>;
10
11 using tii = tuple<int,int,int>;
12 // auto [a,b,c] = ...
13 // .insert({a,b,c})
14
15 const int oo = (int)1e9 + 5; //INF to INT
16 const ll OO = 0x3f3f3f3f3f3f3fLL; //INF to LL
17
18 // g++ -std=c++17 -Wall -Wshadow -fsanitize = address
19 // -O2 -o cod a.cpp
20
21 int main() {
22     ios::sync_with_stdio(false);
23     cin.tie(NULL);
24
25
26     return 0;
27 }

```

3.6 Get Subsets Sum Iterative

```

1 vector<ll> get_subset_sums(int l, int r, vector<ll>&
2 arr) {
3     vector<ll> ans;
4
5     int len = r-l+1;
6     for (int i = 0; i < (1 << len); i++) {
7         ll sum = 0;
8
9         for (int j = 0; j < len; j++) {
10             if (i&(1 << j)) {
11                 sum += arr[l + j];
12             }
13         }
14         ans.push_back(sum);
15     }
16 }

```

```

17     return ans;
18 }

```

3.7 Xor Basis

```

1 // XOR Basis
2 // You are given a set of $N$ integer values. You
   should find the minimum number of values that you
   need to add to the set such that the following
   will hold true:
3 // For every two integers $A$ and $B$ in the set,
   their bitwise xor $A \oplus B$ is also in the set
   .
4
5 vector<ll> basis;
6
7 void add(ll x) {
8     for (int i = 0; i < (int)basis.size(); i++) {
9         // reduce x using the current basis vectors
10        x = min(x, x ^ basis[i]);
11    }
12
13    if (x != 0) { basis.push_back(x); }
14 }
15
16 ll res = (1LL << (int)basis.size()) - n;

```

3.8 Xor 1 To N

```

1 // XOR sum from 1 to N
2 ll xor_1_to_n(ll n) {
3     if (n % 4 == 0) {
4         return n;
5     } else if (n % 4 == 1) {
6         return 1;
7     } else if (n % 4 == 2) {
8         return n + 1;
9     }
10
11    return 0;
12 }

```

3.9 Base Converter

```

1 const string digits = "0123456789
   ABCDEFGHIJKLMNOPQRSTUVWXYZ";
2
3 ll tobase10(string number, int base) {
4     map<char, int> val;
5     for (int i = 0; i < digits.size(); i++) {
6         val[digits[i]] = i;
7     }
8
9     ll ans = 0, pot = 1;
10
11    for (int i = number.size() - 1; i >= 0; i--) {
12        ans += val[number[i]] * pot;
13        pot *= base;
14    }
15
16    return ans;
17 }
18
19 string frombase10(ll number, int base) {
20     if (number == 0) return "0";
21
22     string ans = "";
23
24     while (number > 0) {
25         ans += digits[number % base];
26         number /= base;
27     }

```

```

28
29     reverse(ans.begin(), ans.end());
30
31     return ans;
32 }
33
34 // verifica se um número está na base especificada
35 bool verify_base(string num, int base) {
36     map<char, int> val;
37     for (int i = 0; i < digits.size(); i++) {
38         val[digits[i]] = i;
39     }
40
41     for (auto digit : num) {
42         if (val[digit] >= base) {
43             return false;
44         }
45     }
46
47     return true;
48 }

```

3.10 Interactive

```

1 // you should use cout.flush() every cout
2 int query(int a) {
3     cout << "? " << a << '\n';
4     cout.flush();
5     char res; cin >> res;
6     return res;
7 }
8
9 // using endl you don't need
10 int query(int a) {
11     cout << "? " << a << endl;
12     char res; cin >> res;
13     return res;
14 }

```

3.11 Flags

```

1 // g++ -std=c++17 -Wall -Wshadow -fsanitize=address -
   02 -D -o cod a.cpp

```

3.12 Custom Unordered Map

```

1 // Source: Tiagosf00
2
3 struct custom_hash {
4     static uint64_t splitmix64(uint64_t x) {
5         // http://xorshift.di.unimi.it/splitmix64.c
6         x += 0x9e3779b97f4a7c15;
7         x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
8         x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
9         return x ^ (x >> 31);
10    }
11
12    size_t operator()(uint64_t x) const {
13        static const uint64_t FIXED_RANDOM = chrono::
14        steady_clock::now().time_since_epoch().count();
15        return splitmix64(x + FIXED_RANDOM);
16    }
17 };
18 unordered_map<long long, int, custom_hash> safe_map;
19
20 // when using pairs
21 struct custom_hash {
22     inline size_t operator()(const pii & a) const {
23         return (a.first << 6) ^ (a.first >> 2) ^
24        2038074743 ^ a.second;
25     }

```

```
25 };
```

3.13 Overflow

```
1 // Signatures of some built-in functions to perform
  arithmetic operations with overflow check
2 // Source: https://gcc.gnu.org/onlinedocs/gcc/Integer
  -Overflow-Builtins.html
3 //
4 // you can also check overflow by performing the
  operation with double
5 // and checking if the result it's greater than the
  maximum value supported by the variable
6
7 bool __builtin_add_overflow (type1 a, type2 b, type3
  *res)
8 bool __builtin_sadd_overflow (int a, int b, int *res)
9 bool __builtin_saddl_overflow (long int a, long int b
  , long int *res)
10 bool __builtin_saddll_overflow (long long int a, long
  long int b, long long int *res)
11 bool __builtin_uadd_overflow (unsigned int a,
  unsigned int b, unsigned int *res)
12 bool __builtin_uaddl_overflow (unsigned long int a,
  unsigned long int b, unsigned long int *res)
13 bool __builtin_uaddll_overflow (unsigned long long
  int a, unsigned long long int b, unsigned long
  long int *res)
14
15 bool __builtin_sub_overflow (type1 a, type2 b, type3
  *res)
16 bool __builtin_ssub_overflow (int a, int b, int *res)
17 bool __builtin_ssubl_overflow (long int a, long int b
  , long int *res)
18 bool __builtin_ssubll_overflow (long long int a, long
  long int b, long long int *res)
19 bool __builtin_usub_overflow (unsigned int a,
  unsigned int b, unsigned int *res)
20 bool __builtin_usubl_overflow (unsigned long int a,
  unsigned long int b, unsigned long int *res)
21 bool __builtin_usubll_overflow (unsigned long long
  int a, unsigned long long int b, unsigned long
  long int *res)
22
23 bool __builtin_mul_overflow (type1 a, type2 b, type3
  *res)
24 bool __builtin_smul_overflow (int a, int b, int *res)
25 bool __builtin_smull_overflow (long int a, long int b
  , long int *res)
26 bool __builtin_smulll_overflow (long long int a, long
  long int b, long long int *res)
27 bool __builtin_umul_overflow (unsigned int a,
  unsigned int b, unsigned int *res)
28 bool __builtin_umull_overflow (unsigned long int a,
  unsigned long int b, unsigned long int *res)
29 bool __builtin_umulll_overflow (unsigned long long
  int a, unsigned long long int b, unsigned long
  long int *res)
30
31 bool __builtin_add_overflow_p (type1 a, type2 b,
  type3 c)
32 bool __builtin_sub_overflow_p (type1 a, type2 b,
  type3 c)
33 bool __builtin_mul_overflow_p (type1 a, type2 b,
  type3 c)
```

3.14 Next Permutation

```
1 // output: 1,2,3; 1,3,2; 2,1,3; 2,3,1; 3,1,2; 3,2,1;
2
3 vector<int> arr = {1, 2, 3};
4 int n = arr.size();
```

```
5
6 do {
7     for (auto e : arr) {
8         cout << e << ' ';
9     }
10    cout << '\n';
11 } while (next_permutation(arr.begin(), arr.end()));
```

3.15 First True

```
1 // Binary Search (first_true)
2 //
3 // first_true(2, 10, [](int x) { return x * x >= 30;
  }); // outputs 6
4 //
5 // [l, r]
6 //
7 // if none of the values in the range work, return hi
  + 1
8 //
9 // f(4) = false
10 // f(5) = false
11 // f(6) = true
12 // f(7) = true
13
14 int first_true(int lo, int hi, function<bool(int)> f)
  {
15     hi++;
16     while (lo < hi) {
17         int mid = lo + (hi - lo) / 2;
18
19         if (f(mid)) {
20             hi = mid;
21         } else {
22             lo = mid + 1;
23         }
24     }
25     return lo;
26 }
```

3.16 Kosaraju

```
1 struct Kosaraju {
2
3     int N;
4     int cntComps;
5
6     vector<vector<int>> g;
7     vector<vector<int>> gi;
8
9     stack<int> S;
10    vector<int> vis;
11    vector<int> comp;
12
13    Kosaraju(vector<vector<int>>& arr) {
14        N = (int)arr.size();
15        cntComps = 0;
16
17        g.resize(N);
18        gi.resize(N);
19        vis.resize(N);
20        comp.resize(N);
21
22        for(int i = 0; i < (int)arr.size(); i++) {
23            for(auto &v : arr[i]) {
24                g[i].push_back(v);
25                gi[v].push_back(i);
26            }
27        }
28
29        run();
30    }
```

```

31
32 void dfs(int u) {
33     vis[u] = 1;
34     for(auto &v : g[u]) if(!vis[v]) {
35         dfs(v);
36     }
37     S.push(u);
38 }
39
40 void scc(int u, int c) {
41     vis[u] = 1;
42     comp[u] = c;
43     for(auto &v : gi[u]) if(!vis[v]) {
44         scc(v, c);
45     }
46 }
47
48 void run() {
49     vis.assign(N, 0);
50
51     for(int i = 0; i < N; i++) if(!vis[i]) {
52         dfs(i);
53     }
54
55     vis.assign(N, 0);
56
57     while((int)S.size()) {
58         int u = S.top();
59         S.pop();
60         if(!vis[u]) {
61             scc(u, cntComps++);
62         }
63     }
64 }
65 }
66
67 };

```

3.17 Min Priority Queue

```

1 template<class T> using min_priority_queue =
  priority_queue<T, vector<T>, greater<T>>;

```

4 Math

4.1 Is Prime

```

1 bool is_prime(ll n) {
2     if (n <= 1) return false;
3     if (n == 2) return true;
4
5     for (ll i = 2; i*i <= n; i++) {
6         if (n % i == 0)
7             return false;
8     }
9
10    return true;
11 }

```

4.2 Fft Quirino

```

1 // FFT
2 //
3 // boa em memÃria e ok em tempo
4 //
5 // https://codeforces.com/group/YgJmumGtHD/contest
  //528947/problem/H (maratona mineira)
6
7 using cd = complex<double>;
8 const double PI = acos(-1);
9

```

```

10 void fft(vector<cd> &A, bool invert) {
11     int N = size(A);
12
13     for (int i = 1, j = 0; i < N; i++) {
14         int bit = N >> 1;
15         for (; j & bit; bit >>= 1)
16             j ^= bit;
17         j ^= bit;
18
19         if (i < j)
20             swap(A[i], A[j]);
21     }
22
23     for (int len = 2; len <= N; len <= 1) {
24         double ang = 2 * PI / len * (invert ? -1 : 1);
25         cd wlen(cos(ang), sin(ang));
26         for (int i = 0; i < N; i += len) {
27             cd w(1);
28             for (int j = 0; j < len/2; j++) {
29                 cd u = A[i+j], v = A[i+j+len/2] * w;
30                 A[i+j] = u + v;
31                 A[i+j+len/2] = u - v;
32                 w *= wlen;
33             }
34         }
35     }
36
37     if (invert) {
38         for (auto &x : A)
39             x /= N;
40     }
41 }
42
43 vector<int> multiply(vector<int> const& A, vector<int>
  > const& B) {
44     vector<cd> fa(begin(A), end(A)), fb(begin(B), end(B));
45
46     int N = 1;
47     while (N < size(A) + size(B))
48         N <= 1;
49     fa.resize(N);
50     fb.resize(N);
51
52     fft(fa, false);
53     fft(fb, false);
54     for (int i = 0; i < N; i++)
55         fa[i] *= fb[i];
56     fft(fa, true);
57
58     vector<int> result(N);
59     for (int i = 0; i < N; i++)
60         result[i] = round(fa[i].real());
61     return result;
62 }

```

4.3 Factorization

```

1 // nson
2
3 using ll = long long;
4
5 vector<pair<ll, int>> factorization(ll n) {
6     vector<pair<ll, int>> ans;
7
8     for (ll p = 2; p*p <= n; p++) {
9         if (n%p == 0) {
10             int expoente = 0;
11
12             while (n%p == 0) {
13                 n /= p;
14                 expoente++;
15             }
16

```

```

17         ans.push_back({p, expoente});
18     }
19 }
20
21 if (n > 1) {
22     ans.push_back({n, 1});
23 }
24
25 return ans;
26 }

```

4.4 Sieve

```

1 vector<int> sieve(int MAXN){
2     //list of prime numbers up to MAXN
3     vector<int> primes;
4     bitset<(int)1e7> not_prime;
5     not_prime[0] = 1;
6     not_prime[1] = 1;
7     for(int i = 2; i <= MAXN; i++){
8         if(!not_prime[i]){
9             primes.push_back(i);
10            for(ll j = 1LL * i * i; j <= MAXN; j += i
11        ){
12            not_prime[(int)j] = 1;
13        }
14    }
15    return primes;
16 }

```

4.5 Ceil

```

1 using ll = long long;
2
3 // avoid overflow
4 ll division_ceil(ll a, ll b) {
5     return 1 + ((a - 1) / b); // if a != 0
6 }
7
8 int intceil(int a, int b) {
9     return (a+b-1)/b;
10 }

```

4.6 Log Any Base

```

1 int intlog(double base, double x) {
2     return (int)(log(x) / log(base));
3 }

```

4.7 Ifac

```

1 // inverse of factorial
2
3 mint fac[N], ifac[N];
4
5 void build_fac() {
6     fac[0] = 1;
7
8     for (int i = 1; i < N; i++) {
9         fac[i] = fac[i - 1] * i;
10    }
11
12    ifac[N - 1] = inv(fac[N - 1]);
13
14    for (int i = N - 2; i >= 0; i--) {
15        ifac[i] = ifac[i + 1] * (i + 1);
16    }
17 }

```

4.8 Division Trick

```

1 for(int l = 1, r; l <= n; l = r + 1) {
2     r = n / (n / l);
3     // n / x yields the same value for l <= x <= r
4 }
5 for(int l, r = n; r > 0; r = l - 1) {
6     int tmp = (n + r - 1) / r;
7     l = (n + tmp - 1) / tmp;
8     // (n+x-1) / x yields the same value for l <= x
9     <= r
10 }

```

4.9 Fexp

```

1 using ll = long long;
2
3 ll fexp(ll base, ll exp, ll m) {
4     ll ans = 1;
5     base %= m;
6
7     while (exp > 0) {
8         if (exp % 2 == 1) {
9             ans = (ans * base) % m;
10        }
11
12        base = (base * base) % m;
13        exp /= 2;
14    }
15
16    return ans;
17 }

```

4.10 Number Sum Product Of Divisors

```

1 // CSES - Divisor Analysis
2 // Print the number, sum and product of the divisors.
3 // Since the input number may be large, it is given
4 // as a prime factorization.
5 // Input:
6 // The first line has an integer n: the number of
7 // parts in the prime factorization.
8 // After this, there are n lines that describe the
9 // factorization. Each line has two numbers x and k
10 // where x is a prime and k is its power.
11
12 // Output:
13 // Print three integers modulo 10^9+7: the number,
14 // sum and product of the divisors.
15
16 // Constraints:
17 // (1 <= n <= 1e5) ; (2 <= x <= 1e6) ; (1 <= k <= 1e9
18 // ) ; each x is a distinct prime
19
20 #include <bits/stdc++.h>
21 typedef long long ll;
22 using namespace std;
23
24 const ll MOD = 1e9 + 7;
25
26 ll expo(ll base, ll pow) {
27     ll ans = 1;
28     while (pow) {
29         if (pow & 1) ans = ans * base % MOD;
30         base = base * base % MOD;
31         pow >>= 1;
32     }
33     return ans;
34 }
35
36 ll p[100001], k[100001];
37
38 int main() {
39     cin.tie(0) -> sync_with_stdio(0);

```

```

35 int n;
36 cin >> n;
37 for (int i = 0; i < n; i++) cin >> p[i] >> k[i];
38 ll div_cnt = 1, div_sum = 1, div_prod = 1,
div_cnt2 = 1;
39 for (int i = 0; i < n; i++) {
40     div_cnt = div_cnt * (k[i] + 1) % MOD;
41     div_sum = div_sum * (expo(p[i], k[i] + 1) -
1) % MOD *
42     expo(p[i] - 1, MOD - 2) % MOD;
43     div_prod = expo(div_prod, k[i] + 1) *
44     expo(expo(p[i], (k[i] * (k[i] + 1)
/ 2)), div_cnt2) % MOD;
45     div_cnt2 = div_cnt2 * (k[i] + 1) % (MOD - 1);
46 }
47 cout << div_cnt << ' ' << div_sum << ' ' <<
div_prod;
48 return 0;
49 }

```

4.11 Divisors

```

1 vector<ll> divisors(ll n) {
2     vector<ll> ans;
3
4     for (ll i = 1; i*i <= n; i++) {
5         if (n%i == 0) {
6             ll value = n/i;
7
8             ans.push_back(i);
9             if (value != i) {
10                 ans.push_back(value);
11             }
12         }
13     }
14
15     return ans;
16 }

```

5 Graph

5.1 Floyd Warshall

```

1 const long long LLINF = 0x3f3f3f3f3f3f3f3fLL;
2
3 for (int i = 0; i < n; i++) {
4     for (int j = 0; j < n; j++) {
5         adj[i][j] = 0;
6     }
7 }
8
9 long long dist[MAX][MAX];
10 for (int i = 0; i < n; i++) {
11     for (int j = 0; j < n; j++) {
12         if (i == j)
13             dist[i][j] = 0;
14         else if (adj[i][j])
15             dist[i][j] = adj[i][j];
16         else
17             dist[i][j] = LLINF;
18     }
19 }
20
21 for (int k = 0; k < n; k++) {
22     for (int i = 0; i < n; i++) {
23         for (int j = 0; j < n; j++) {
24             dist[i][j] = min(dist[i][j], dist[i][k] +
25 dist[k][j]);
26         }
27 }

```

5.2 Lca

```

1 // LCA
2 //
3 // lowest common ancestor between two nodes
4 //
5 // edit_distance(n, adj, root)
6 //
7 // https://cses.fi/problemset/task/1688
8 //
9 // O(log N)
10
11 struct LCA {
12     const int MAXE = 31;
13     vector<vector<int>> up;
14     vector<int> dep;
15
16     LCA(int n, vector<vector<int>>& adj, int root =
1) {
17         up.assign(n+1, vector<int>(MAXE, -1));
18         dep.assign(n+1, 0);
19
20         dep[root] = 1;
21         dfs(root, -1, adj);
22
23         for (int j = 1; j < MAXE; j++) {
24             for (int i = 1; i <= n; i++) {
25                 if (up[i][j-1] != -1)
26                     up[i][j] = up[ up[i][j-1] ][j-1];
27             }
28         }
29     }
30
31     void dfs(int x, int p, vector<vector<int>>& adj)
32     {
33         up[x][0] = p;
34         for (auto e : adj[x]) {
35             if (e != p) {
36                 dep[e] = dep[x] + 1;
37                 dfs(e, x, adj);
38             }
39         }
40     }
41
42     int jump(int x, int k) { // jump from node x k
43         times
44         for (int i = 0; i < MAXE; i++) {
45             if (k && (1 << i) && x != -1) x = up[x][i];
46         }
47         return x;
48     }
49
50     int lca(int a, int b) {
51         if (dep[a] > dep[b]) swap(a, b);
52         b = jump(b, dep[b] - dep[a]);
53
54         if (a == b) return a;
55
56         for (int i = MAXE-1; i >= 0; i--) {
57             if (up[a][i] != up[b][i]) {
58                 a = up[a][i];
59                 b = up[b][i];
60             }
61         }
62         return up[a][0];
63     }
64
65     int dist(int a, int b) {
66         return dep[a] + dep[b] - 2 * dep[lca(a, b)];
67     }
68 };

```


5.3 Bfs

```

1 vector<vector<int>> adj; // adjacency list
  representation
2 int n; // number of nodes
3 int s; // source vertex
4
5 queue<int> q;
6 vector<bool> used(n + 1);
7 vector<int> d(n + 1), p(n + 1);
8
9 q.push(s);
10 used[s] = true;
11 p[s] = -1;
12 while (!q.empty()) {
13     int v = q.front();
14     q.pop();
15     for (int u : adj[v]) {
16         if (!used[u]) {
17             used[u] = true;
18             q.push(u);
19             d[u] = d[v] + 1;
20             p[u] = v;
21         }
22     }
23 }
24
25 // restore path
26 if (!used[u]) {
27     cout << "No path!";
28 } else {
29     vector<int> path;
30
31     for (int v = u; v != -1; v = p[v])
32         path.push_back(v);
33
34     reverse(path.begin(), path.end());
35
36     cout << "Path: ";
37     for (int v : path)
38         cout << v << " ";
39 }

```

5.4 Dinic

```

1 // Dinic / Dinitz
2 //
3 // max-flow / min-cut
4 //
5 // https://cses.fi/problemset/task/1694/
6 //
7 // O(E * V^2)
8
9 using ll = long long;
10 const ll FLOW_INF = 1e18 + 7;
11
12 struct Edge {
13     int from, to;
14     ll cap, flow;
15     Edge* residual; // a inversa da minha aresta
16
17     Edge() {};
```

```

18
19     Edge(int from, int to, ll cap) : from(from), to(to), cap(cap), flow(0) {};
```

```

20
21     ll remaining_cap() {
22         return cap - flow;
23     }
24
25     void augment(ll bottle_neck) {
26         flow += bottle_neck;

```

```

27         residual->flow -= bottle_neck;
28     }
29
30     bool is_residual() {
31         return cap == 0;
32     }
33 };
34
35 struct Dinic {
36     int n;
37     vector<vector<Edge*>> adj;
38     vector<int> level, next;
39
40     Dinic(int n): n(n) {
41         adj.assign(n+1, vector<Edge*>());
42         level.assign(n+1, -1);
43         next.assign(n+1, 0);
44     }
45
46     void add_edge(int from, int to, ll cap) {
47         auto e1 = new Edge(from, to, cap);
48         auto e2 = new Edge(to, from, 0);
49
50         e1->residual = e2;
51         e2->residual = e1;
52
53         adj[from].push_back(e1);
54         adj[to].push_back(e2);
55     }
56
57     bool bfs(int s, int t) {
58         fill(level.begin(), level.end(), -1);
59         queue<int> q;
60
61         q.push(s);
62         level[s] = 1;
63
64         while (q.size()) {
65             int curr = q.front();
66             q.pop();
67
68             for (auto edge : adj[curr]) {
69                 if (edge->remaining_cap() > 0 &&
57 level[edge->to] == -1) {
70                     level[edge->to] = level[curr] +
71 1;
72                     q.push(edge->to);
73                 }
74             }
75         }
76
77         return level[t] != -1;
78     }
79
80     ll dfs(int x, int t, ll flow) {
81         if (x == t) return flow;
82
83         for (int& cid = next[x]; cid < (int)adj[x].
size(); cid++) {
84             auto& edge = adj[x][cid];
85             ll cap = edge->remaining_cap();
86
87             if (cap > 0 && level[edge->to] == level[x]
+ 1) {
88                 ll sent = dfs(edge->to, t, min(flow,
cap)); // bottle neck
89                 if (sent > 0) {
90                     edge->augment(sent);
91                     return sent;
92                 }
93             }
94         }

```

```

95     return 0;
96 }
97
98 ll solve(int s, int t) {
99     ll max_flow = 0;
100
101     while (bfs(s, t)) {
102         fill(next.begin(), next.end(), 0);
103
104         while (ll sent = dfs(s, t, FLOW_INF)) {
105             max_flow += sent;
106         }
107     }
108
109     return max_flow;
110 }
111
112 // path recover
113 vector<bool> vis;
114 vector<int> curr;
115
116 bool dfs2(int x, int& t) {
117     vis[x] = true;
118     bool arrived = false;
119
120     if (x == t) {
121         curr.push_back(x);
122         return true;
123     }
124
125     for (auto e : adj[x]) {
126         if (e->flow > 0 && !vis[e->to]) { // !e->
127             is_residual() &&
128             bool aux = dfs2(e->to, t);
129
130             if (aux) {
131                 arrived = true;
132                 e->flow--;
133             }
134         }
135     }
136
137     if (arrived) curr.push_back(x);
138
139     return arrived;
140 }
141
142 vector<vector<int>> get_paths(int s, int t) {
143     vector<vector<int>> ans;
144
145     while (true) {
146         curr.clear();
147         vis.assign(n+1, false);
148
149         if (!dfs2(s, t)) break;
150
151         reverse(curr.begin(), curr.end());
152         ans.push_back(curr);
153     }
154
155     return ans;
156 };

```

5.5 2sat

```

1 // 2SAT
2 //
3 // verifica se existe e encontra soluÃ§Ã£o
4 // para fÃ³rmulas booleanas da forma
5 // (a or b) and (!a or c) and (...)
6 //
7 // indexado em 0

```

```

8 // n(a) = 2*x e n(~a) = 2*x+1
9 // a = 2 ; n(a) = 4 ; n(~a) = 5 ; n(a)^1 = 5 ; n(~a)
10 // ^1 = 4
11 // https://cses.fi/problemset/task/1684/
12 // https://codeforces.com/gym/104120/problem/E
13 // (add_eq, add_true, add_false e at_most_one nÃ£o
14 // foram testadas)
15 // 0(n + m)
16
17 struct sat {
18     int n, tot;
19     vector<vector<int>> adj, adjt; // grafo original,
20     // grafo transposto
21     vector<int> vis, comp, ans;
22     stack<int> topo; // ordem topolÃ³gica
23
24     sat() {}
25     sat(int n_) : n(n_), tot(n), adj(2*n), adjt(2*n)
26     {}
27
28     void dfs(int x) {
29         vis[x] = true;
30
31         for (auto e : adj[x]) {
32             if (!vis[e]) dfs(e);
33         }
34
35         topo.push(x);
36     }
37
38     void dfst(int x, int& id) {
39         vis[x] = true;
40         comp[x] = id;
41
42         for (auto e : adjt[x]) {
43             if (!vis[e]) dfst(e, id);
44         }
45     }
46
47     void add_impl(int a, int b) { // a -> b = (!a or
48     b)
49         a = (a >= 0 ? 2*a : -2*a-1);
50         b = (b >= 0 ? 2*b : -2*b-1);
51
52         adj[a].push_back(b);
53         adj[b^1].push_back(a^1);
54
55         adjt[b].push_back(a);
56         adjt[a^1].push_back(b^1);
57     }
58
59     void add_or(int a, int b) { // a or b
60         add_impl(~a, b);
61     }
62
63     void add_nor(int a, int b) { // a nor b = !(a or
64     b)
65         add_or(~a, b), add_or(a, ~b), add_or(~a, ~b);
66     }
67
68     void add_and(int a, int b) { // a and b
69         add_or(a, b), add_or(~a, b), add_or(a, ~b);
70     }
71
72     void add_nand(int a, int b) { // a nand b = !(a
73     and b)
74         add_or(~a, ~b);
75     }
76
77     void add_xor(int a, int b) { // a xor b = (a != b)
78     }
79 }

```

```

73     add_or(a, b), add_or(~a, ~b);
74 }
75
76 void add_xnor(int a, int b) { // a xnor b = !(a
xor b) = (a == b)
77     add_xor(~a, b);
78 }
79
80 void add_true(int a) { // a = T
81     add_or(a, ~a);
82 }
83
84 void add_false(int a) { // a = F
85     add_and(a, ~a);
86 }
87
88 // magia - brunomaletta
89 void add_true_old(int a) { // a = T (n sei se
funciona)
90     add_impl(~a, a);
91 }
92
93 void at_most_one(vector<int> v) { // no max um
verdadeiro
94     adj.resize(2*(tot+v.size()));
95     for (int i = 0; i < v.size(); i++) {
96         add_impl(tot+i, ~v[i]);
97         if (i) {
98             add_impl(tot+i, tot+i-1);
99             add_impl(v[i], tot+i-1);
100         }
101     }
102     tot += v.size();
103 }
104
105 pair<bool, vector<int>> solve() {
106     ans.assign(n, -1);
107     comp.assign(2*tot, -1);
108     vis.assign(2*tot, 0);
109     int id = 1;
110
111     for (int i = 0; i < 2*tot; i++) if (!vis[i])
dfs(i);
112
113     vis.assign(2*tot, 0);
114     while (topo.size()) {
115         auto x = topo.top();
116         topo.pop();
117
118         if (!vis[x]) {
119             dfst(x, id);
120             id++;
121         }
122     }
123
124     for (int i = 0; i < tot; i++) {
125         if (comp[2*i] == comp[2*i+1]) return {
false, {} };
126         ans[i] = (comp[2*i] > comp[2*i+1]);
127     }
128
129     return {true, ans};
130 }
131 };

```

5.6 Min Cost Max Flow

```

1 // Min Cost Max Flow (brunomaletta)
2 //
3 // min_cost_flow(s, t, f) computa o par (fluxo, custo
)
4 // com max(fluxo) <= f que tenha min(custo)

```

```

5 // min_cost_flow(s, t) -> Fluxo maximo de custo
minimo de s pra t
6 // Se for um dag, da pra substituir o SPFA por uma DP
pra nao
7 // pagar O(nm) no comeco
8 // Se nao tiver aresta com custo negativo, nao
precisa do SPFA
9 //
10 // O(nm + f * m log n)
11
12 template<typename T> struct mcmf {
13     struct edge {
14         int to, rev, flow, cap; // para, id da
reversa, fluxo, capacidade
15         bool res; // se eh reversa
16         T cost; // custo da unidade de fluxo
17         edge() : to(0), rev(0), flow(0), cap(0), cost
(0), res(false) {}
18         edge(int to_, int rev_, int flow_, int cap_,
T cost_, bool res_)
19             : to(to_), rev(rev_), flow(flow_), cap(
cap_), res(res_), cost(cost_) {}
20     };
21
22     vector<vector<edge>> g;
23     vector<int> par_idx, par;
24     T inf;
25     vector<T> dist;
26
27     mcmf(int n) : g(n), par_idx(n), par(n), inf(
numeric_limits<T>::max()/3) {}
28
29     void add(int u, int v, int w, T cost) { // de u
pra v com cap w e custo cost
30         edge a = edge(v, g[v].size(), 0, w, cost,
false);
31         edge b = edge(u, g[u].size(), 0, 0, -cost,
true);
32
33         g[u].push_back(a);
34         g[v].push_back(b);
35     }
36
37     vector<T> spfa(int s) { // nao precisa se nao
tiver custo negativo
38         deque<int> q;
39         vector<bool> is_inside(g.size(), 0);
40         dist = vector<T>(g.size(), inf);
41
42         dist[s] = 0;
43         q.push_back(s);
44         is_inside[s] = true;
45
46         while (!q.empty()) {
47             int v = q.front();
48             q.pop_front();
49             is_inside[v] = false;
50
51             for (int i = 0; i < g[v].size(); i++) {
52                 auto [to, rev, flow, cap, res, cost]
= g[v][i];
53                 if (flow < cap and dist[v] + cost <
dist[to]) {
54                     dist[to] = dist[v] + cost;
55
56                     if (is_inside[to]) continue;
57                     if (!q.empty() and dist[to] >
dist[q.front()]) q.push_back(to);
58                     else q.push_front(to);
59                     is_inside[to] = true;
60                 }
61             }
62         }

```

```

63     return dist;
64 }
65 bool dijkstra(int s, int t, vector<T>& pot) {
66     priority_queue<pair<T, int>, vector<pair<T,
67 int>>, greater<>> q;
68     dist = vector<T>(g.size(), inf);
69     dist[s] = 0;
70     q.emplace(0, s);
71     while (q.size()) {
72         auto [d, v] = q.top();
73         q.pop();
74         if (dist[v] < d) continue;
75         for (int i = 0; i < g[v].size(); i++) {
76             auto [to, rev, flow, cap, res, cost]
77 = g[v][i];
78             cost += pot[v] - pot[to];
79             if (flow < cap and dist[v] + cost <
80 dist[to]) {
81                 dist[to] = dist[v] + cost;
82                 q.emplace(dist[to], to);
83                 par_idx[to] = i, par[to] = v;
84             }
85         }
86         return dist[t] < inf;
87     }
88     pair<int, T> min_cost_flow(int s, int t, int flow
89 = INF) {
90         vector<T> pot(g.size(), 0);
91         pot = spfa(s); // mudar algoritmo de caminho
92         minimo aqui
93         int f = 0;
94         T ret = 0;
95         while (f < flow and dijkstra(s, t, pot)) {
96             for (int i = 0; i < g.size(); i++)
97                 if (dist[i] < inf) pot[i] += dist[i];
98             int mn_flow = flow - f, u = t;
99             while (u != s) {
100                 mn_flow = min(mn_flow,
101 g[par[u]][par_idx[u]].cap - g[par
102 [u]][par_idx[u]].flow);
103                 u = par[u];
104             }
105             ret += pot[t] * mn_flow;
106             u = t;
107             while (u != s) {
108                 g[par[u]][par_idx[u]].flow += mn_flow
109 ;
110                 g[u][g[par[u]][par_idx[u]].rev].flow
111 -= mn_flow;
112                 u = par[u];
113             }
114             f += mn_flow;
115         }
116         return make_pair(f, ret);
117     }
118 // Opcional: retorna as arestas originais por
119 onde passa flow = cap
120 vector<pair<int, int>> recover() {
121     vector<pair<int, int>> used;
122     for (int i = 0; i < g.size(); i++) for (edge
123 e : g[i])
124         if (e.flow == e.cap && !e.res) used.
125         push_back({i, e.to});
126     return used;

```

```

125     }
126 };

```

5.7 Ford Fulkerson

```

1 // Ford-Fulkerson
2 //
3 // max-flow / min-cut
4 //
5 // MAX nÃ§s
6 //
7 // https://cses.fi/problemset/task/1694/
8 //
9 // O(m * max_flow)
10
11 using ll = long long;
12 const int MAX = 510;
13
14 struct Flow {
15     int n;
16     ll adj[MAX][MAX];
17     bool used[MAX];
18
19     Flow(int n) : n(n) {};
20
21     void add_edge(int u, int v, ll c) {
22         adj[u][v] += c;
23         adj[v][u] = 0; // cuidado com isso
24     }
25
26     ll dfs(int x, int t, ll amount) {
27         used[x] = true;
28
29         if (x == t) return amount;
30
31         for (int i = 1; i <= n; i++) {
32             if (adj[x][i] > 0 && !used[i]) {
33                 ll sent = dfs(i, t, min(amount, adj[x
34 ][i]));
35
36                 if (sent > 0) {
37                     adj[x][i] -= sent;
38                     adj[i][x] += sent;
39
40                     return sent;
41                 }
42             }
43         }
44         return 0;
45     }
46
47     ll max_flow(int s, int t) { // source and sink
48         ll total = 0;
49         ll sent = -1;
50
51         while (sent != 0) {
52             memset(used, 0, sizeof(used));
53             sent = dfs(s, t, INT_MAX);
54             total += sent;
55         }
56
57         return total;
58     }
59 };

```

5.8 Dijkstra

```

1 const int INF = 1e9+17;
2 vector<vector<pair<int, int>>> adj; // {neighbor,
3     weight}

```

```

4 void dijkstra(int s, vector<int> & d, vector<int> & p
    ) {
5     int n = adj.size();
6     d.assign(n, INF);
7     p.assign(n, -1);
8
9     d[s] = 0;
10    set<pair<int, int>> q;
11    q.insert({0, s});
12    while (!q.empty()) {
13        int v = q.begin()->second;
14        q.erase(q.begin());
15
16        for (auto edge : adj[v]) {
17            int to = edge.first;
18            int len = edge.second;
19
20            if (d[v] + len < d[to]) {
21                q.erase({d[to], to});
22                d[to] = d[v] + len;
23                p[to] = v;
24                q.insert({d[to], to});
25            }
26        }
27    }
28 }

```

5.9 Has Negative Cycle

```

1 // Edson
2
3 using edge = tuple<int, int, int>;
4
5 bool has_negative_cycle(int s, int N, const vector<
    edge>& edges)
6 {
7     const int INF { 1e9+17 };
8
9     vector<int> dist(N + 1, INF);
10    dist[s] = 0;
11
12    for (int i = 1; i <= N - 1; i++) {
13        for (auto [u, v, w] : edges) {
14            if (dist[u] < INF && dist[v] > dist[u] +
15                w) {
16                dist[v] = dist[u] + w;
17            }
18        }
19    }
20
21    for (auto [u, v, w] : edges) {
22        if (dist[u] < INF && dist[v] > dist[u] + w) {
23            return true;
24        }
25    }
26
27    return false;
28 }

```

5.10 3sat

```

1 // We are given a CNF, e.g. phi(x) = (x_1 or ~x_2)
2 // and (x_3 or ~x_4 or ~x_5) and ...
3 // SAT finds an assignment x for phi(x) = true.
4 // Davis-Putnam-Logemann-Loveland Algorithm (
5 // youknowwho code)
6 // Complexity: O(2^n) in worst case.
7 // This implementation is practical for n <= 1000 or
8 // more. lmao.
9
10 #include<bits/stdc++.h>
11 using namespace std;

```

```

10 const int N = 3e5 + 9;
11
12 // positive literal x in [0,n),
13 // negative literal ~x in [-n,0)
14 // 0 indexed
15 struct SAT_GOD {
16     int n;
17     vector<int> occ, pos, neg;
18     vector<vector<int>> g, lit;
19     SAT_GOD(int n) : n(n), g(2*n), occ(2*n) {}
20     void add_clause(const vector<int> &c) {
21         for(auto u: c) {
22             g[u+n].push_back(lit.size());
23             occ[u+n] += 1;
24         }
25         lit.push_back(c);
26     }
27     /*(!u | v | !w) -> (u, 0, v, 1, w, 0)
28     void add(int u, int af, int v = 1e9, int bf = 0,
29         int w = 1e9, int cf = 0) {
30         vector<int> a;
31         if(!af) u = ~u;
32         a.push_back(u);
33         if(v != 1e9) {
34             if(!bf) v = ~v;
35             a.push_back(v);
36         }
37         if(w != 1e9) {
38             if(!cf) w = ~w;
39             a.push_back(w);
40         }
41         add_clause(a);
42     }
43     vector<bool> x;
44     vector<vector<int>> decision_stack;
45     vector<int> unit_stack, pure_stack;
46     void push(int u) {
47         x[u + n] = 1;
48         decision_stack.back().push_back(u);
49         for (auto i: g[u + n]) if (pos[i]++ == 0) {
50             for (auto u: lit[i]) --occ[u+n];
51         }
52         for (auto i: g[~u + n]) {
53             ++neg[i];
54             if (pos[i] == 0) unit_stack.push_back(i);
55         }
56     }
57     void pop() {
58         int u = decision_stack.back().back();
59         decision_stack.back().pop_back();
60         x[u + n] = 0;
61         for (auto i: g[u + n]) if (--pos[i] == 0) {
62             for (auto u: lit[i]) ++occ[u + n];
63         }
64         for (auto i: g[~u+n]) --neg[i];
65     }
66     bool reduction() {
67         while(!unit_stack.empty() || !pure_stack.empty())
68         {
69             if(!pure_stack.empty()) { // pure literal
70                 elimination
71                 int u = pure_stack.back();
72                 pure_stack.pop_back();
73                 if (occ[u + n] == 1 && occ[~u + n] == 0) push
74                     (u);
75             } else { // unit propagation
76                 int i = unit_stack.back();
77                 unit_stack.pop_back();
78                 if(pos[i] > 0) continue;
79                 if(neg[i] == lit[i].size()) return false;
80                 if(neg[i] + 1 == lit[i].size()) {
81                     int w = n;

```

```

78     for (int u: lit[i]) if (!x[u + n] && !x[~u + n]) w = u;
79     if (x[~w + n]) return false;
80     push(w);
81 }
82 }
83 }
84 return true;
85 }
86 bool ok() {
87     x.assign(2*n, 0);
88     pos = neg = vector<int>(lit.size());
89     decision_stack.assign(1, {});
90     while(1) {
91         if(reduction()) {
92             int s = 0;
93             for(int u = 0; u < n; ++u) if(occ[s + n] +
94             occ[~s + n] < occ[u + n] + occ[~u + n]) s = u;
95             if(occ[s + n] + occ[~s + n] == 0) return true;
96             ;
97             decision_stack.push_back({});
98             push(s);
99         } else {
100             int s = decision_stack.back()[0];
101             while(!decision_stack.back().empty()) pop();
102             decision_stack.pop_back();
103             if (decision_stack.empty()) return false;
104             push(~s);
105         }
106     }
107 }
108 int32_t main() {
109     int n = 9;
110     SAT_GOD t(n);
111     t.add(0, 0, 1, 1);
112     t.add(1, 0);
113     t.add(1, 0, 3, 1, 5, 1);
114     cout << t.ok() << endl;
115 }

```

```

25     return x == o.x && y == o.y;
26 }
27 };
28
29 ftype cross(Point a, Point b, Point c) {
30     // v: a -> c
31     // w: a -> b
32
33     // v: c.x - a.x, c.y - a.y
34     // w: b.x - a.x, b.y - a.y
35
36     return (c.x - a.x) * (b.y - a.y) - (c.y - a.y) *
37     (b.x - a.x);
38 }
39
40 ftype dir(Point a, Point b, Point c) {
41     // 0 -> colineares
42     // -1 -> esquerda
43     // 1 -> direita
44
45     ftype cp = cross(a, b, c);
46
47     if (cp == 0) return 0;
48     else if (cp < 0) return -1;
49     else return 1;
50 }
51
52 vector<Point> convex_hull(vector<Point> points) {
53     sort(points.begin(), points.end());
54     points.erase( unique(points.begin(), points.end())
55     ), points.end()); // somente pontos distintos
56     int n = points.size();
57
58     if (n == 1) return { points[0] };
59
60     vector<Point> upper_hull = {points[0], points
61     [1]};
62     for (int i = 2; i < n; i++) {
63         upper_hull.push_back(points[i]);
64
65         int sz = upper_hull.size();
66
67         while (sz >= 3 && dir(upper_hull[sz-3],
68         upper_hull[sz-2], upper_hull[sz-1]) == -1) {
69             upper_hull.pop_back();
70             upper_hull.pop_back();
71             upper_hull.push_back(points[i]);
72             sz--;
73         }
74     }
75
76     vector<Point> lower_hull = {points[n-1], points[n
77     -2]};
78     for (int i = n-3; i >= 0; i--) {
79         lower_hull.push_back(points[i]);
80
81         int sz = lower_hull.size();
82
83         while (sz >= 3 && dir(lower_hull[sz-3],
84         lower_hull[sz-2], lower_hull[sz-1]) == -1) {
85             lower_hull.pop_back();
86             lower_hull.pop_back();
87             lower_hull.push_back(points[i]);
88             sz--;
89         }
90     }
91
92     // reverse(lower_hull.begin(), lower_hull.end());
93     // counterclockwise
94
95     for (int i = (int)lower_hull.size() - 2; i > 0; i
96     --) {
97         upper_hull.push_back(lower_hull[i]);
98     }
99 }

```

6 Geometry

6.1 Convex Hull

```

1 // Convex Hull - Monotone Chain
2 //
3 // Convex Hull is the subset of points that forms the
4 // smallest convex polygon
5 // which encloses all points in the set.
6 //
7 // https://cses.fi/problemset/task/2195/
8 // https://open.kattis.com/problems/convexhull (
9 // counterclockwise)
10 //
11 // 0(n log(n))
12
13 typedef long long ftype;
14
15 struct Point {
16     ftype x, y;
17
18     Point() {}
19     Point(ftype x, ftype y) : x(x), y(y) {}
20
21     bool operator<(Point o) {
22         if (x == o.x) return y < o.y;
23         return x < o.x;
24     }
25
26     bool operator==(Point o) {

```

```

90     }
91
92     return upper_hull;
93 }

```

7 Primitives

7.1 Set Union Intersection

```

1 // Template pra fazer união e intercessão de sets
  de forma fácil
2 // Usar + para uniao e * para intercessão
3 // Source: https://stackoverflow.com/questions
  /13448064/how-to-find-the-intersection-of-two-stl
  -sets
4
5 template <class T, class CMP = std::less<T>, class
  ALLOC = std::allocator<T> >
6 std::set<T, CMP, ALLOC> operator * (
7     const std::set<T, CMP, ALLOC> &s1, const std::set<T
  , CMP, ALLOC> &s2)
8 {
9     std::set<T, CMP, ALLOC> s;
10    std::set_intersection(s1.begin(), s1.end(), s2.
  begin(), s2.end(),
11        std::inserter(s, s.begin()));
12    return s;
13 }
14
15 template <class T, class CMP = std::less<T>, class
  ALLOC = std::allocator<T> >
16 std::set<T, CMP, ALLOC> operator + (
17     const std::set<T, CMP, ALLOC> &s1, const std::set<T
  , CMP, ALLOC> &s2)
18 {
19     std::set<T, CMP, ALLOC> s;
20    std::set_union(s1.begin(), s1.end(), s2.begin(), s2
  .end(),
21        std::inserter(s, s.begin()));
22    return s;
23 }

```

8 String

8.1 Split

```

1 vector<string> split(string s, char key=' ') {
2     vector<string> ans;
3     string aux = "";
4
5     for (int i = 0; i < (int)s.size(); i++) {
6         if (s[i] == key) {
7             if (aux.size() > 0) {
8                 ans.push_back(aux);
9                 aux = "";
10            }
11        } else {
12            aux += s[i];
13        }
14    }
15
16    if ((int)aux.size() > 0) {
17        ans.push_back(aux);
18    }
19
20    return ans;
21 }

```

8.2 Hash

```

1 struct Hash {
2     ll MOD, P;
3     int n; string s;
4     vector<ll> h, hi, p;
5     Hash() {}
6     Hash(string s, ll MOD, ll P = 31): s(s), MOD(MOD)
  , P(P), n(s.size()), h(n), hi(n), p(n) {
7         for (int i=0;i<n;i++) p[i] = (i ? P*p[i-1]:1)
  % MOD;
8         for (int i=0;i<n;i++)
9             h[i] = (s[i] + (i ? h[i-1]:0) * P) % MOD;
10        for (int i=n-1;i>=0;i--)
11            hi[i] = (s[i] + (i+1<n ? hi[i+1]:0) * P)
  % MOD;
12    }
13    int query(int l, int r) {
14        ll hash = (h[r] - (l ? h[l-1]*p[r-l+1]%MOD :
  0));
15        return hash < 0 ? hash + MOD : hash;
16    }
17    int query_inv(int l, int r) {
18        ll hash = (hi[l] - (r+1 < n ? hi[r+1]*p[r-1
  +1] % MOD : 0));
19        return hash < 0 ? hash + MOD : hash;
20    }
21 };
22
23 struct DoubleHash {
24     const ll MOD1 = 90264469;
25     const ll MOD2 = 25699183;
26
27     Hash hash1, hash2;
28
29     DoubleHash();
30
31     DoubleHash(string s) : hash1(s, MOD1), hash2(s,
  MOD2) {}
32
33     pair<int, int> query(int l, int r) {
34         return { hash1.query(l, r), hash2.query(l, r)
  };
35     }
36
37     pair<int, int> query_inv(int l, int r) {
38         return { hash1.query_inv(l, r), hash2.
  query_inv(l, r) };
39     }
40 };
41
42 struct TripleHash {
43     const ll MOD1 = 90264469;
44     const ll MOD2 = 25699183;
45     const ll MOD3 = 81249169;
46
47     Hash hash1, hash2, hash3;
48
49     TripleHash();
50
51     TripleHash(string s) : hash1(s, MOD1), hash2(s,
  MOD2), hash3(s, MOD3) {}
52
53     tuple<int, int, int> query(int l, int r) {
54         return { hash1.query(l, r), hash2.query(l, r)
  , hash3.query(l, r) };
55     }
56
57     tuple<int, int, int> query_inv(int l, int r) {
58         return { hash1.query_inv(l, r), hash2.
  query_inv(l, r), hash3.query_inv(l, r) };
59     }
60 };
61
62 struct HashK {

```

```

63 vector<ll> primes; // more primes = more hashes
64 vector<Hash> hash;
65
66 HashK();
67
68 HashK(string s, vector<ll> primes): primes(primes)
69 {
70     for (auto p : primes) {
71         hash.push_back(Hash(s, p));
72     }
73 }
74
75 vector<int> query(int l, int r) {
76     vector<int> ans;
77
78     for (auto h : hash) {
79         ans.push_back(h.query(l, r));
80     }
81
82     return ans;
83 }
84
85 vector<int> query_inv(int l, int r) {
86     vector<int> ans;
87
88     for (auto h : hash) {
89         ans.push_back(h.query_inv(l, r));
90     }
91
92     return ans;
93 };

```

8.3 Is Substring

```

1 // equivalente ao in do python
2
3 bool is_substring(string a, string b){ // verifica se
4     a Ã substring de b
5     for(int i = 0; i < b.size(); i++){
6         int it = i, jt = 0; // b[it], a[jt]
7
8         while(it < b.size() && jt < a.size()){
9             if(b[it] != a[jt])
10                break;
11
12             it++;
13             jt++;
14
15             if(jt == a.size())
16                 return true;
17         }
18
19         return false;
20 }

```

8.4 Trie Xor

```

1 // TrieXOR
2 //
3 // adiciona, remove e verifica se existe strings
4 // binarias
5 // max_xor(x) = maximiza o xor de x com algum valor
6 // da trie
7 //
8 // raiz = 0
9 //
10 // https://codeforces.com/problemset/problem/706/D
11 // 0(|s|) adicionar, remover e buscar

```

```

12 struct TrieXOR {
13     int n, alph_sz, nxt;
14     vector<vector<int>> trie;
15     vector<int> finish, paths;
16
17     TrieXOR() {}
18
19     TrieXOR(int n, int alph_sz = 2) : n(n), alph_sz(
20         alph_sz) {
21         nxt = 1;
22         trie.assign(n, vector<int>(alph_sz));
23         finish.assign(n * alph_sz, 0);
24         paths.assign(n * alph_sz, 0);
25     }
26
27     void add(int x) {
28         int curr = 0;
29
30         for (int i = 31; i >= 0; i--) {
31             int b = ((x >> i) > 0);
32
33             if (trie[curr][b] == 0)
34                 trie[curr][b] = nxt++;
35
36             paths[curr]++;
37             curr = trie[curr][b];
38         }
39
40         paths[curr]++;
41         finish[curr]++;
42     }
43
44     void rem(int x) {
45         int curr = 0;
46
47         for (int i = 31; i >= 0; i--) {
48             int b = ((x >> i) > 0);
49
50             paths[curr]--;
51             curr = trie[curr][b];
52         }
53
54         paths[curr]--;
55         finish[curr]--;
56     }
57
58     int search(int x) {
59         int curr = 0;
60
61         for (int i = 31; i >= 0; i--) {
62             int b = ((x >> i) > 0);
63
64             if (trie[curr][b] == 0) return false;
65
66             curr = trie[curr][b];
67         }
68
69         return (finish[curr] > 0);
70     }
71
72     int max_xor(int x) { // maximum xor with x and
73         any number of trie
74         int curr = 0, ans = 0;
75
76         for (int i = 31; i >= 0; i--) {
77             int b = ((x >> i) > 0);
78             int want = b^1;
79
80             if (trie[curr][want] == 0 || paths[trie[
81                 curr][want]] == 0) want ^= 1;
82             if (trie[curr][want] == 0 || paths[trie[
83                 curr][want]] == 0) break;
84             if (want != b) ans |= (1 << i);

```



```
81         curr = trie[curr][want];
82     }
83 }
84
85     return ans;
86 }
87 };
```