

Team Notebook
CodeWinter Genius

- 1) Source
 - 1.1)
- 2) Деревья
 - 2.1) HLD
 - 2.2) Link-cut
 - 2.3) ЕТТ
- 3) Графы
 - 3.1) Нахождение отрицательного цикла
 - 3.2) mincost maxflow
- 4) ТЧ
 - 4.1) FFT
 - 4.2) Поллард
 - 4.3) Решето Эратосфена за $O(n)$
- 5) Структуры данных
 - 6.1) ДО снизу
 - 6.2) Персистентное ДД
- 6) Строки
 - 7.1) Суфф. автомат
 - 7.2) Суфф. масс + lcp
- 7) Оптимизация Динамики
 - 9.1) Ли Чао
 - 9.2) Лямбда

1) Source

```
26 #define FAST_ALLOCATOR_MEMORY (5e8)
27
28 #ifdef FAST_ALLOCATOR_MEMORY
29     int allocator_pos = 0;
30     char allocator_memory[(int)FAST_ALLOCATOR_MEMORY];
31     inline void * operator new ( size_t n ) {
32         char *res = allocator_memory + allocator_pos;
33         allocator_pos += n;
34         assert(allocator_pos <= (int)FAST_ALLOCATOR_MEMORY);
35         return (void *)res;
36     }
37     inline void operator delete ( void * ) noexcept { }
38 #endif
```

2) Деревья

2.1) HLD

```
void prec(ll v, ll p = -1) {
    sz[v] = 1;
    tin[v] = ttime++;
    for (auto x : g[v]) {
        if(x == p)
            continue;
        prec(x, v);
        sz[v] += sz[x];
    }
    tout[v] = ttime++;
}

ll dfs(ll v, ll p = -1, ll l = -1) {
    par[v] = p;
    if(l == -1) {
        l = v;
    }
    gr[v].ft = l;
    pos[v] = euler.size();
    euler.pb(v);

    ll nxt = -1;
    for (auto x : g[v]) {
        if(x == p)
            continue;
        nxt = x;
        gr[v].sc = dfs(x, v, l);
        break;
    }
    if(nxt == -1) {
        gr[v].sc = v;
        rght[v] = euler.size();
        return v;
    }
    for (auto x : g[v]) {
        if(x == p || x == nxt)
            continue;
        dfs(x, v, -1);
    }
    rght[v] = euler.size();
    return gr[v].sc;
}

bool cmp(ll a, ll b) {
    return !(sz[a] < sz[b]);
}

bool isp(ll a, ll b) {
    if(tin[a] <= tin[b] && tout[b] <= tout[a])
        return 1;
    return 0;
}

void update(ll v) {
    if(v == 0)
        return;
    tree[v] = tree[v * 2] + tree[v * 2 + 1];
    update(v / 2);
}

ll get(ll v, ll l, ll r, ll vl, ll vr) {
    if(vr <= l || r <= vl)
        return 0;
    if(vl <= l && r <= vr) {
        return tree[v];
    }
    ll m = (l + r) / 2;
    return get(v * 2, l, m, vl, vr) + get(v * 2 + 1, m, r, vl, vr);
}

ll solve(ll v) {
    ll ans = 0;
    ll pr = -1;
    while(v != -1) {
        if(pr == -1) {
            ans += get_sz(v) * w[v];
            if(pos[gr[v].ft] < pos[v]) {
                ans += get(1, 0, N, pos[gr[v].ft], pos[v]);
            }
        }
        else {
            if(pos[gr[v].ft] < pos[v]) {
                ans += get(1, 0, N, pos[gr[v].ft], pos[v]);
            }
            ans += (get_sz(v) - get_sz(pr)) * w[v];
        }
        pr = gr[v].ft;
        v = par[gr[v].ft];
    }
    return ans;
}
```

2.2) Link-cut

```

1  #include <iostream>
2  #include <cstdio>
3  #include <cassert>
4
5  using namespace std;
6
7  // BEGIN ALGO
8
9  const int MAXN = 110000;
10
11 typedef struct _node{
12     _node *l, *r, *p, *pp;
13     int size; bool rev;
14     _node();
15     explicit _node(nullptr_t){
16         l = r = p = pp = this;
17         size = rev = 0;
18     }
19     void push(){
20         if (rev){
21             l->rev ^= 1; r->rev ^= 1;
22             rev = 0; swap(l,r);
23         }
24     }
25     void update();
26 }* node;
27 node None = new _node(nullptr);
28 node v2n[MAXN];
29 _node::_node(){
30     l = r = p = pp = None;
31     size = 1; rev = false;
32 }
33 void _node::update(){
34     size = (this != None) + l->size + r->size;
35     l->p = r->p = this;
36 }
37 void rotate(node v){
38     assert(v != None && v->p != None);
39     assert(!v->rev); assert(!v->p->rev);
40     node u = v->p;
41     if (v == u->l)
42         u->l = v->r, v->r = u;
43     else
44         u->r = v->l, v->l = u;
45     swap(u->p, v->p); swap(v->pp, u->pp);
46     if (v->p != None){
47         assert(v->p->l == u || v->p->r == u);
48         if (v->p->r == u) v->p->r = v;
49         else v->p->l = v;
50     }
51     u->update(); v->update();
52 }
53 void bigRotate(node v){
54     assert(v->p != None);
55     v->p->p->push();
56     v->p->push();
57     v->push();
58     if (v->p->p != None){
59         if ((v->p->l == v) ^ (v->p->p->r == v->p))
60             rotate(v->p);
61         else
62             rotate(v);
63     }
64     rotate(v);
65 }
66 inline void Splay(node v){
67     while (v->p != None) bigRotate(v);
68 }
69 inline void splitAfter(node v){
70     v->push();
71     Splay(v);
72     v->r->p = None;
73     v->r->pp = v;
74     v->r = None;
75     v->update();
76 }
77 void expose(int x){
78     node v = v2n[x];
79     splitAfter(v);
80     while (v->pp != None){
81         assert(v->p == None);
82         splitAfter(v->pp);
83         assert(v->pp->r == None);
84         assert(v->pp->p == None);
85         assert(!v->pp->rev);
86         v->pp->r = v;
87         v->pp->update();
88         v = v->pp;
89         v->r->pp = None;
90     }
91     assert(v->p == None);
92     Splay(v2n[x]);
93 }
94 inline void makeRoot(int x){
95     expose(x);
96     assert(v2n[x]->p == None);
97     assert(v2n[x]->pp == None);
98     assert(v2n[x]->r == None);
99     v2n[x]->rev ^= 1;
100 }
101 inline void link(int x,int y){
102     makeRoot(x); v2n[x]->pp = v2n[y];
103 }
104 inline void cut(int x,int y){
105     expose(x);
106     Splay(v2n[y]);
107     if (v2n[y]->pp != v2n[x]){
108         swap(x,y);
109         expose(x);
110         Splay(v2n[y]);
111         assert(v2n[y]->pp == v2n[x]);
112     }
113     v2n[y]->pp = None;
114 }
115 inline int get(int x,int y){
116     if (x == y) return 0;
117     makeRoot(x);
118     expose(y); expose(x);
119     Splay(v2n[y]);
120     if (v2n[y]->pp != v2n[x]) return -1;
121     return v2n[y]->size;
122 }
123 // END ALGO
124
125 _node mem[MAXN];
126
127
128 int main(){
129     freopen("linkcut.in","r",stdin);
130     freopen("linkcut.out","w",stdout);
131
132     int n,m;
133     scanf("%d %d",&n,&m);
134
135     for (int i = 0; i < n; i++){
136         v2n[i] = &mem[i];
137     }
138
139     for (int i = 0; i < m; i++){
140         int a,b;
141         if (scanf(" link %d %d",&a,&b) == 2)
142             link(a-1,b-1);
143         else if (scanf(" cut %d %d",&a,&b) == 2)
144             cut(a-1,b-1);
145         else if (scanf(" get %d %d",&a,&b) == 2)
146             printf("%d\n",get(a-1,b-1));
147         else
148             assert(false);
149     }

```

2.3) ETT

```

1  const int N = 3e5 + 228;
2  mt19937 rnd(228);
3
4  struct treap {
5      treap *l, *r, *par;
6      int sz;
7      int y;
8      treap() {
9          sz = 1;
10         y = rnd();
11         par = 0;
12         l = r = 0;
13     }
14     treap(treap *p) {
15         sz = 1;
16         y = rnd();
17         par = p;
18         l = r = 0;
19     }
20 };
21
22 int siz(treap *t) {
23     if (!t) return 0;
24     return t->sz;
25 }
26
27 void recalc(treap *t) {
28     if (!t) return;
29     t->sz = 1 + siz(t->l) + siz(t->r);
30 }
31
32 int get_id(treap *t) {
33     int sum = siz(t->l);
34     while (t->par) {
35         if (t->par->l == t) {
36             t = t->par;
37         } else {
38             sum += siz(t->par->l) + 1;
39             t = t->par;
40         }
41     }
42     return sum;
43 }
44
45 void set_l(treap *t, treap *x) {
46     if (x)
47         x->par = t;
48     t->l = x;
49     recalc(t);
50 }
51
52 void set_r(treap *t, treap *x) {
53     if (x)
54         x->par = t;
55     t->r = x;
56     recalc(t);
57 }
58

```

```

59 pair <treap *, treap *> split(treap *t, int k) {
60     if (!t) return {0, 0};
61     t->par = 0;
62     if (siz(t->l) >= k) {
63         auto a = split(t->l, k);
64         set_l(t, a.second);
65         return {a.first, t};
66     } else {
67         auto a = split(t->r, k - siz(t->l) - 1);
68         set_r(t, a.first);
69         return {t, a.second};
70     }
71 }
72
73 treap *merge(treap *a, treap *b) {
74     if (!a) return b;
75     if (!b) return a;
76     a->par = 0, b->par = 0;
77     if (a->y > b->y) {
78         set_r(a, merge(a->r, b));
79         return a;
80     } else {
81         set_l(b, merge(a, b->l));
82         return b;
83     }
84 }
85
86 treap *get_root(treap *t) {
87     while (t->par) t = t->par;
88     return t;
89 }
90
91 set <int> g[N];
92 map <pair <int, int>, treap*> st;
93
94 pair <int, int> any_edge(int v) {
95     assert(!g[v].empty());
96     return make_pair(v, *g[v].begin());
97 }
98
99 bool is_connected(int a, int b) {
100     if (a == b) {
101         return true;
102     }
103     if (g[a].empty() || g[b].empty()) return false;
104     return get_root(st[any_edge(a)]) == get_root(st[any_edge(b)]);
105 }
106
107 void make_first(treap *t) {
108     int id = get_id(t);
109     auto a = split(get_root(t), id);
110     merge(a.second, a.first);
111 }
112
113 void make_last(treap *t) {
114     int id = get_id(t);
115     auto a = split(get_root(t), id + 1);
116     merge(a.second, a.first);
117 }
118

```

```

119 void link(int u, int v) {
120
121     st[{u, v}] = new treap();
122     st[{v, u}] = new treap();
123     if (g[v].empty()) swap(u, v);
124     if (g[u].empty()) {
125         if (g[v].empty()) {
126             merge(st[{u, v}], st[{v, u}]);
127         } else {
128             auto x = any_edge(v);
129             make_first(st[x]);
130             merge(st[{u, v}], get_root(st[x]));
131             merge(get_root(st[x]), st[{v, u}]);
132         }
133     } else {
134         pair <int, int> x, y;
135         {
136             x = any_edge(u);
137             swap(x.first, x.second);
138             make_last(st[x]);
139         }
140         {
141             y = any_edge(v);
142             make_first(st[y]);
143         }
144         merge(get_root(st[x]), st[{u, v}]);
145         merge(get_root(st[x]), get_root(st[y]));
146         merge(get_root(st[x]), st[{v, u}]);
147     }
148     g[u].insert(v);
149     g[v].insert(u);
150 }
151
152 void cut(int u, int v) {
153     g[u].erase(v);
154     g[v].erase(u);
155     auto x = make_pair(u, v);
156     make_first(st[x]);
157     auto a = split(get_root(st[x]), 1);
158     auto y = make_pair(v, u);
159     int ret = get_id(st[y]);
160     auto b = split(a.second, ret);
161     split(b.second, 1);
162     delete st[x];
163     delete st[y];
164     st[x] = st[y] = 0;
165 }

```

3) Графы

3.1) Нахождение отрицательного цикла

```
struct edge {
    int a, b, cost;
};

int n, m;
vector<edge> e;
const int INF = 1000000000;

void solve() {
    vector<int> d (n);
    vector<int> p (n, -1);
    int x;
    for (int i=0; i<n; ++i) {
        x = -1;
        for (int j=0; j<m; ++j)
            if (d[e[j].b] > d[e[j].a] + e[j].cost) {
                d[e[j].b] = max (-INF, d[e[j].a] +
e[j].cost);

                p[e[j].b] = e[j].a;
                x = e[j].b;
            }
    }

    if (x == -1)
        cout << "No negative cycle found.";
    else {
        int y = x;
        for (int i=0; i<n; ++i)
            y = p[y];

        vector<int> path;
        for (int cur=y; ; cur=p[cur]) {
            path.push_back (cur);
            if (cur == y && path.size() > 1) break;
        }
        reverse (path.begin(), path.end());

        cout << "Negative cycle: ";
        for (size_t i=0; i<path.size(); ++i)
            cout << path[i] << ' ';
    }
}
```


3.2) mincost maxflow

```
8 struct edge {
9     int u, c, w, rev, f = 0;
10     edge() {}
11     edge(int u, int c, int w, int rev): u(u), c(c), w(w), rev(rev) {}
12 };
13
14 const int MAX_N = 1007, INF = 1e18;
15 vector<edge> g[MAX_N];
16 int dist[MAX_N];
17 int p[MAX_N];
18 pair<int, int> prevv[MAX_N];
19 bool visited[MAX_N];
20 int s, t;
21
22 void relax(int v, int pos, int f) {
23     g[v][pos].c -= f;
24     g[v][pos].f += f;
25     int u = g[v][pos].u, pos2 = g[v][pos].rev;
26     g[u][pos2].c += f;
27     g[u][pos2].f -= f;
28 }
29
30 void Ford_Bellman(int n) {
31     for (int i = 0; i <= n; i++)
32         dist[i] = INF;
33     dist[s] = 0;
34     prevv[s] = {-1, -1};
35     for (int i = 0; i <= n; i++) {
36         for (int v = 0; v <= n; v++) {
37             for (int j = 0; j < (int) g[v].size(); j++) {
38                 edge e = g[v][j];
39                 if (e.c <= 0)
40                     continue;
41                 dist[e.u] = min(dist[e.u], dist[v] + e.w);
42             }
43         }
44     }
45 }
46
47 bool Dijkstra(int n) {
48     for (int i = 0; i <= n; i++)
49         p[i] = dist[i] + p[i];
50     for (int i = 0; i <= n; i++)
51         dist[i] = INF;
52     memset(visited, 0, sizeof(visited));
53     dist[s] = 0;
54     prevv[s] = {-1, -1};
55     while (1) {
56         int mini = INF, v = -1;
57         for (int i = 0; i <= n; i++) {
58             if (dist[i] < mini && !visited[i]) {
59                 mini = dist[i];
60                 v = i;
61             }
62         }
63         if (v == -1)
64             break;
65         visited[v] = 1;
66         for (int i = 0; i < (int) g[v].size(); i++) {
67             edge e = g[v][i];
68             if (e.c <= 0 || visited[e.u])
69                 continue;
70             int w = e.w + p[v] - p[e.u];
71             if (dist[e.u] > dist[v] + w) {
72                 dist[e.u] = dist[v] + w;
73                 prevv[e.u] = {v, i};
74             }
75         }
76     }
77     return (dist[t] < INF);
78 }
79
```

```
80 int max_flow(int n) {
81     int f = 0;
82     Ford_Bellman(n);
83     while (Dijkstra(n)) {
84         int pos = t;
85         vector<pair<int, int>> path;
86         while (pos != -1) {
87             if (prevv[pos].first != -1)
88                 path.push_back(prevv[pos]);
89             pos = prevv[pos].first;
90         }
91         int cur_f = INF;
92         for (auto elem : path) {
93             cur_f = min(cur_f, g[elem.first][elem.second].c);
94         }
95         for (auto elem : path)
96             relax(elem.first, elem.second, cur_f);
97         f += cur_f;
98     }
99     return f;
100 }
101
102 signed main() {
103     int n;
104     cin >> n;
105     s = n + n;
106     t = s + 1;
107     for (int i = 0; i < n; i++) {
108         g[s].push_back(edge(i, 1, 0, g[i].size()));
109         g[i].push_back(edge(s, 0, 0, (int) g[s].size() - 1));
110         g[i + n].push_back(edge(t, 1, 0, g[t].size()));
111         g[t].push_back(edge(i + n, 0, 0, (int) g[i + n].size() - 1));
112     }
113     for (int i = 0; i < n; i++) {
114         for (int j = 0; j < n; j++) {
115             int x;
116             cin >> x;
117             g[i].push_back(edge(n + j, 1, x, g[n + j].size()));
118             g[n + j].push_back(edge(i, 0, -x, (int) g[i].size() - 1));
119         }
120     }
121     max_flow(t);
122     vector<pair<int, int>> ans;
123     int cost = 0;
124     for (int i = 0; i < n; i++) {
125         for (edge e : g[i]) {
126             if (e.f == 1) {
127                 cost += e.w;
128                 ans.push_back({i + 1, e.u - n + 1});
129             }
130         }
131     }
132     cout << cost << endl;
133     for (auto elem : ans) {
134         cout << elem.first << ' ' << elem.second << endl;
135     }
136     return 0;
137 }
```


4) ТЧ

4.1) FFT

```
27 int maxlog = 19;
28 int N = (1 << maxlog);
29 ld pi = acos(-1);
30
31 vector<complx> fft(vector<complx> a) {
32     vector<int> rev(N);
33     for (int i = 0; i < N; i++) {
34         rev[i] = (rev[i / 2] / 2) + ((i & 1) << (maxlog - 1));
35     }
36     for (int i = 0; i < N; i++) {
37         if(i < rev[i])
38             swap(a[i], a[rev[i]]);
39     }
40     for (int k = 1; k < N; k *= 2) {
41         complx w = {cos(2 * pi / (2 * k)), sin(2 * pi / (2 * k))};
42         for (int s = 0; s < N; s += 2 * k) {
43             complx now_w = {1, 0};
44             for (int j = 0; j < k; j++) {
45                 complx u = a[s + j];
46                 complx v = now_w * a[s + j + k];
47                 a[s + j] = u + v;
48                 a[s + j + k] = u - v;
49                 now_w *= w;
50             }
51         }
52     }
53     return a;
54 }
55
56 vector<int> get_ans(vector<complx> a) {
57     a = fft(a);
58     reverse(a.begin() + 1, a.end());
59     vector<int> ans(N, 0);
60     for (int i = 0; i < N; i++)
61         ans[i] = a[i].real() / N + 0.5;
62     return ans;
63 }
```

4.2) Поллард

```
20 typedef __int128 bigint;
21
22 vector<int> have;
23 vector<ll> anss;
24 int maxn = 1e4;
25 mt19937 rnd(228);
26
27 bigint pw(bigint a, bigint st, bigint mod) {
28     if(st == 0)
29         return 1;
30     if(st % 2 == 0) {
31         bigint y = pw(a, st / 2, mod);
32         return (y * y) % mod;
33     }
34     else {
35         bigint y = pw(a, st - 1, mod);
36         return (y * a) % mod;
37     }
38 }
39
40 bigint gcd(bigint a, bigint b) {
41     if(a == 0 || b == 0)
42         return a + b;
43     return gcd(b, a % b);
44 }
```

```

46 bool check(bigint p) {
47     if(p < 1000) {
48         for (int i = 2; i * i <= p; i++) {
49             if(p % i == 0)
50                 return 0;
51         }
52         return 1;
53     }
54
55     bigint f = (p - 1);
56     int cnt = 0;
57     while(f % 2 == 0) {
58         cnt++;
59         f /= 2;
60     }
61     for (auto a : have) {
62         if(pw(a, p - 1, p) != 1)
63             return 0;
64         if(gcd(a, p) != 1)
65             return 0;
66         vector<bigint> now;
67         bigint r = pw(a, f, p);
68         for (int i = 0; i <= cnt; i++) {
69             now.pb(r);
70             r = (r * r) % p;
71         }
72         for (int i = len(now) - 1; i >= 0; i--) {
73             if(now[i] != 1) {
74                 if(now[i] != (p - 1))
75                     return 0;
76                 break;
77             }
78         }
79     }
80     return 1;
81 }
82
83 bigint f(bigint a, bigint mod) {
84     bigint res = (a * a + 1) % mod;
85     return res;
86 }
87
88 bigint abss(bigint x) {
89     if(x < 0)
90         return -x;
91     return x;
92 }
93
94 void solve(bigint N) {
95     if(check(N)) {
96         anss.pb(N);
97         return;
98     }
99     bool f1 = 0;
100     for (int i = 2; i < maxn; i++) {
101         while(N % i == 0) {
102             anss.pb(i);
103             f1 = 1;
104             N /= i;
105         }
106     }
107     if(f1) {
108         solve(N);
109         return;
110     }
111
112     while(1) {
113         bigint a = rnd() % N;
114         vector<bigint> now;
115         int v1 = -1, v2 = -1;
116         ll ans = -1;
117         while(1) {
118             now.pb(a);
119             a = f(a, N);
120             now.pb(a);
121             v2 += 2;
122             v1 += 1;
123             bigint g = gcd(abss(now[v1] - now[v2]), N);
124             if(g == N)
125                 break;
126             if(g != 1) {
127                 ans = g;
128                 break;
129             }
130         }
131         if(ans != -1) {
132             solve(ans);
133             solve(N / ans);
134             return;
135         }
136     }
137 }
138
139 for (int i = 2; i < 1e3; i++) {
140     have.push_back(i);
141     for (int j = 2; j < i; j++) {
142         if(i % j == 0) {
143             have.pop_back();
144             break;
145         }
146     }
147     if(have.size() >= 30)
148         break;
149 }
150 ll N;
151 cin >> N;
152 solve(N);

```

4.3) Решето Эратосфена за $O(n)$

```
24     vector<int> primes;
25     for (ll i = 2; i < maxn; i++) {
26         if(p[i] == -1) {
27             p[i] = primes.size();
28             primes.push_back(i);
29             ll ic = i;
30         }
31         for (int j = 0; j <= p[i]; j++) {
32             ll x = primes[j];
33             if(x * i > maxn) break;
34             p[primes[j] * i] = j;
35         }
36     }
37 }
```

5) Структуры данных

5.1) ДО снизу

```
7  #define S (1 << 17) // 131072
8  #define INF (1e18 + 1)
9
10 struct node {
11     long long ls, rs, s, m;
12 } T[2 * S];
13
14 int 0;
15
16 void relax (int i) {
17     int l = 2 * i, r = l + 1;
18     T[i].ls = max (T[l].ls, T[l].s + T[r].ls);
19     T[i].rs = max (T[r].rs, T[l].rs + T[r].s);
20     T[i].s = T[l].s + T[r].s;
21     T[i].m = max (max (T[l].m, T[r].m), T[l].rs + T[r].ls);
22 }
23
24 void assign (int i, int v) {
25     T[0 + i].ls = T[0 + i].rs = T[0 + i].s = T[0 + i].m = v;
26 }
27
28
```

```
29 void assign_relax (int i, int v) {
30     assign (i, v);
31     i += 0;
32     while (i > 1) {
33         i >>= 1;
34         relax (i);
35     }
36 }
37
38 long long eval (int l, int r) {
39     l += 0, r += 0;
40     long long ls = -INF, rs = -INF, m = -INF;
41     while (l <= r) {
42         if (l & 1) {
43             m = max (m, max (T[l].m, ls + T[l].ls));
44             ls = max (T[l].rs, ls + T[l].s);
45             ++l;
46         }
47         if (!(r & 1)) {
48             m = max (m, max (T[r].m, rs + T[r].rs));
49             rs = max (T[r].ls, rs + T[r].s);
50             --r;
51         }
52         l >>= 1, r >>= 1;
53     }
54     return max (m, ls + rs);
55 }
56
```

5.2) Персистентное ДД

```
1. struct treap
2. {
3.     int val, sz;
4.     treap *l, *r;
5.     treap(int val, treap *l, treap *r): val(val), sz(1), l(l), r(r)
6.     {
7.     }
8.     treap()
9.     {
10.    }
11.};
12.
13.const int N = 10 * 10000 * 100 + 7;
14.const int MAX_SZ = N - 1e5;
15.
16.treap node[N];
17.int ptr = 0;
18.
19.treap *new_treap(int val, treap *l, treap *r)
20.{
21.    node[ptr].val = val;
22.    node[ptr].l = l;
23.    node[ptr].r = r;
24.    node[ptr].sz = 1;
25.    assert(ptr < N);
26.    return &node[ptr++];
27.}
28.
29.int sz(treap *t)
30.{
31.    if (!t)
32.    {
33.        return 0;
34.    }
35.    else
36.    {
37.        return t->sz;
38.    }
39.}
40.
41.void recalc(treap *t)
42.{
43.    if (!t)
44.    {
45.        return;
46.    }
47.    else
48.    {
49.        t->sz = sz(t->l) + 1 + sz(t->r);
50.    }
51.}
52.
53.void zhfs(treap *t, vector <int> &a)
54.{
55.    if (!t)
56.    {
57.        return;
58.    }
59.    zhfs(t->l, a);
60.    a.push_back(t->val);
61.    zhfs(t->r, a);
62.}
63.
64.pair <treap*, treap*> split(treap *t, int x)
65.{
66.    if (!t)
67.    {
68.        return {0, 0};
```

```

69. }
70. if (sz(t->l) >= x)
71. {
72.     auto a = split(t->l, x);
73.     treap *l = a.first;
74.     treap *r = new_treap(t->val, a.second, t->r);
75.     recalc(r);
76.     return {l, r};
77. }
78. else
79. {
80.     auto a = split(t->r, x - sz(t->l) - 1);
81.     treap *l = new_treap(t->val, t->l, a.first);
82.     treap *r = a.second;
83.     recalc(l);
84.     return {l, r};
85. }
86. }
87.
88. treap *merge(treap *a, treap *b)
89. {
90.     if (!a)
91.     {
92.         return b;
93.     }
94.     if (!b)
95.     {
96.         return a;
97.     }
98.     if (rnd() % (a->sz + b->sz) < a->sz)
99.     {
100.         treap *ret = new_treap(a->val, a->l, merge(a->r, b));
101.         recalc(ret);
102.         return ret;
103.     }
104.     else
105.     {
106.         treap *ret = new_treap(b->val, merge(a, b->l), b->r);
107.         recalc(ret);
108.         return ret;
109.     }
110. }
111.
112. struct q
113. {
114.     int len, from, to;
115.     q(int len, int from, int to): len(len), from(from), to(to)
116.     {
117.     }
118.     q()
119.     {
120.     }
121. };
122.
123. void print(treap *t)
124. {
125.     if (!t)
126.     {
127.         return;
128.     }
129.     print(t->l);
130.     printf("%d ", t->val);
131.     print(t->r);
132. }
133.
134. treap *build(const vector <int> &a)
135. {
136.     if (a.empty())
137.     {

```

```

138.         return 0;
139.     }
140.     int mid = (int) a.size() / 2;
141.     vector<int> l, r;
142.     for (int i = 0; i < mid; i++) l.push_back(a[i]);
143.     for (int i = mid + 1; i < (int) a.size(); i++) r.push_back(a[i]);
144.     treap *go = new_treap(a[mid], build(l), build(r));
145.     recalc(go);
146.     return go;
147. }
148.
149. treap *cur = 0, *rev = 0;
150.
151. treap *get_segm(treap *t, int l, int r)
152. {
153.     l++, r++;
154.     auto a = split(t, r);
155.     auto b = split(a.first, l - 1);
156.     return b.second;
157. }
158.
159. treap *to_copy(treap *t, int l, int r, treap *go)
160. {
161.     l++, r++;
162.     auto a = split(t, r);
163.     auto b = split(a.first, l - 1);
164.     return merge(b.first, merge(go, a.second));
165. }

```

6) Строки

6.1) Суфф. автомат

```
9  int last = 0, sz = 0;
10
11  struct state{
12
13      int len;
14      int link;
15      map < char, int > g;
16
17  };
18
19  vector < state > st(3e5);
20
21  void add(char c) {
22
23      int now = sz++;
24      st[now].len = st[last].len + 1;
25      int p = last;
26      last = now;
27      while (p != -1 && !st[p].g.count(c)) {
28          st[p].g[c] = now;
29          p = st[p].link;
30      }
31      if (p == -1) {
32          st[now].link = 0;
33          return;
34      }
35      int q = st[p].g[c];
36      if (st[p].len + 1 == st[q].len) {
37          st[now].link = q;
38          return;
39      }
40      int clone = sz++;
41      st[clone] = st[q];
42      st[clone].len = st[p].len + 1;
43      while (p != -1 && st[p].g[c] == q) {
44          st[p].g[c] = clone;
45          p = st[p].link;
46      }
47      st[q].link = st[now].link = clone;
48
49  }
50
51  bool check(string& s) {
52
53      int now = 0;
54      int n = s.size();
55      for (int i = 0; i < n; i++) {
56          if ('A' <= s[i] && s[i] <= 'Z') {
57              s[i] = 'a' + (s[i] - 'A');
58          }
59          if (!st[now].g.count(s[i])) {
60              return false;
61          }
62          now = st[now].g[s[i]];
63      }
64      return true;
65  }
```


6.2) Цыф. масс + lcp

```

94 vector<int> a(n);
95 vector<int> color(n);
96 vector<int> head(n);
97 vector<int> newa(n);
98 vector<int> newcolor(n);
99 vector<pair<char, int>> fs(n);
100 for (int i = 0; i < n; i++) {
101     fs[i] = {s[i], i};
102 }
103 sort(fs.begin(), fs.end());
104 head[0] = 0;
105 int cnt = 0;
106 char pr = fs[0].ft;
107 for (int i = 0; i < n; i++) {
108     a[i] = fs[i].sc;
109     if(fs[i].ft == pr) {
110         color[a[i]] = cnt;
111     }
112     else {
113         cnt++;
114         color[a[i]] = cnt;
115         head[cnt] = i;
116     }
117     pr = fs[i].ft;
118 }
119 int k = 1;
120 while(k < n) {
121     for (int i = 0; i < n; i++) {
122         int st = (a[i] - k + 2 * n) % n;
123         newa[head[color[st]]] = st;
124         head[color[st]]++;
125     }
126     head[0] = 0;
127     newcolor[newa[0]] = 0;
128     for (int i = 1; i < n; i++) {
129         int st1 = (newa[i - 1] + k) % n, st2 = (newa[i] + k) % n;
130         if(color[newa[i]] == color[newa[i - 1]] && color[st1] == color[st2]) {
131             newcolor[newa[i]] = newcolor[newa[i - 1]];
132         }
133         else {
134             newcolor[newa[i]] = newcolor[newa[i - 1]] + 1;
135             head[newcolor[newa[i]]] = i;
136         }
137     }
138     a = newa;
139     color = newcolor;
140     k *= 2;
141 }
142 vector<int> obr(n);
143 vector<int> lcp(n);
144 for (int i = 0; i < n; i++) {
145     obr[a[i]] = i;
146 }
147 int maxlcp = 0;
148 for (int i = 0; i < n; i++) {
149     int i1 = obr[i];
150     if(i1 == n - 1) {
151         maxlcp = 0;
152         lcp[i1] = 0;
153         continue;
154     }
155     int i2 = a[i1 + 1];
156     while(s[i + maxlcp] == s[i2 + maxlcp])
157         maxlcp++;
158     lcp[i1] = maxlcp;
159     maxlcp = max(0, maxlcp - 1);
160 }

```

7) Оптимизация динамики

7.1) Ли Чао

```
26 struct line {
27     int k, b;
28     int get(int x) {
29         return k * x + b;
30     }
31     bool operator < (const line& x) const {
32         return k < x.k;
33     }
34 };
35
36 struct node {
37     node* l;
38     node* r;
39     line now;
40     node(node* _l, node* _r, line _now) : l(_l), r(_r), now(_now) {}
41 };
42
43 int get(node* v, int l, int r, int x) {
44     int ans = v->now.get(x);
45     int m = (l + r) / 2;
46     if(m <= x) {
47         if(v->r == NULL)
48             return ans;
49         else
50             return min(ans, get(v->r, m, r, x));
51     }
52     else {
53         if(v->l == NULL)
54             return ans;
55         else
56             return min(ans, get(v->l, l, m, x));
57     }
58 }
59
60 void upd(node* v, int l, int r, line val) {
61     int m = (l + r) / 2;
62     bool lft = val.get(l) < v->now.get(l);
63     bool mid = val.get(m) < v->now.get(m);
64     if(mid)
65         swap(v->now, val);
66     if(r - l == 1)
67         return;
68     if(lft == mid) {
69         if(v->r == NULL) {
70             auto neww = new node(NULL, NULL, {0, infx});
71             v->r = neww;
72         }
73         upd(v->r, m, r, val);
74         return;
75     }
76     else {
77         if(v->l == NULL) {
78             auto neww = new node(NULL, NULL, {0, infx});
79             v->l = neww;
80         }
81         upd(v->l, l, m, val);
82         return;
83     }
84 }
85 }
```

7.2) Лямбда оптимизация

Рассмотрим следующую задачу: Дан массив a_1, a_2, \dots, a_n . Надо разбить его на k отрезков так, чтобы минимизировать сумму квадратов сумм отрезков, $k \leq n$.

Сведем задачу к другой — вместо того, чтобы набирать k отрезков, придумаем разбиение на сколько-нибудь отрезков, но за каждый отрезок будем добавлять к функции ответа штраф, равный λ . Пусть для λ_1 количество отрезков в разбиении равно k_1 . Тогда для $\lambda_2 < \lambda_1$ $k_2 \geq k_1$. Интуиция у этого факта такая — если мы за каждый отрезок платим меньше, то мы можем взять больше отрезков. Понятно, что если данное неравенство не выполняется, то лямбда-оптимизация не работает.

Поскольку зависимость между k и λ монотонная, то можно сделать [бинарный поиск](#) по λ и найти такую, при которой $k_\lambda = k$. Новую задачу можно решать при помощи [convex hull trick](#), потому что

$$dp_i = \min_{j=1}^{i-1} dp[j] + \lambda + pref_i^2 - (2 \cdot pref_j) \cdot pref_i + pref_j^2$$

где $pref$ — массив префиксных сумм.

```
1 pair<int, int> calc(int X) {
2     vector<pair<int, int>> dp(n + 1);
3     add(0, 0);
4     for (int i = 1; i <= n; i++) {
5         int argmin = get(pref[i]);
6         dp[i].first = dp[argmin].first + X + pref[i] * pref[i] - 2 * pref[argmin] * pref[i] + pref[argmin].first;
7         dp[i].second = dp[argmin].second + 1;
8         add(-2 * pref[i], dp[i] + pref[i] * pref[i])
9     }
10    return dp[n];
11 }
12 /*
13 .
14 .
15 .
16 */
17
18 int L = -INF;
19 int R = INF;
20 while (L + 1 < R) {
21     int mid = (L + R) / 2;
22     pair<int, int> X = calc(mid);
23     if (X.second > k) {
24         R = mid;
25     }
26     else {
27         L = mid;
28     }
29 }
30 pair<int, int> res = check(R);
31 cout << res.first - k * R;
```