

Содержание

1 Теория чисел	1
1.1 КТО	1
1.2 Алгоритм Миллера — Рабина	1
1.3 Алгоритм Берлекэмпа — Мессе	1
2 Графы	2
2.1 SCC и 2-SAT	2
2.2 Эйлеров цикл	2
2.3 Компоненты рёберной двусвязности	2
3 xor, and, or-свёртки	3
4 Структуры данных	3
4.1 Дерево Фенвика	3
4.2 Дерево отрезков	3
4.2.1 Примеры использования	4
4.3 Ordered set	4
5 Строковые алгоритмы	4
5.1 Префикс-функция	4
5.2 Z-функция	5
5.3 Алгоритм Манакера	5
5.4 Суфмассив	5
5.5 Алгоритм Ахо — Корасик	5
5.6 Дерево палиндромов	6
6 Потоки	6
6.1 Алгоритм Диница	6
6.2 Mincost k-flow	7
6.2.1 Строим граф	7
6.2.2 Запускаем Форда — Беллмана	7
6.2.3 Ищем кратчайший путь Дейкстрой с потенциалами	7
6.2.4 Восстанавливаем ответ	7
7 FFT & co	7
7.1 NTT & co	7
7.2 старое доброе FFT	8
8 Геометрия	9
8.1 Касательные	9
8.2 Примитивы	9
8.3 Точка нестрого внутри выпуклости	9
9 Разное	9
9.1 Флаги копирования	9
9.1.1 Сеточка в vim	10
9.2 Что сделать на пробном туре	10
9.3 Шаблон	10

1 Теория чисел

1.1 КТО

```

1 int gcd(int a, int b, int &x, int &y) {
2     if (b==0) { x = 1; y = 0; return a; }
3     int d = gcd(b, a%b, y, x);
4     y-=a/b*x;
5     return d;
6 }
7 int inv(int r, int m) {
8     int x, y;
9     gcd(r, m, x, y);
10    return (x+m)%m;
11 }
12 int crt(int r, int n, int c, int m) { return r + ((
    c - r) % m + m) * inv(n, m) % m * n; }
```

1.2 Алгоритм Миллера — Рабина

```

1 __int128 one=1;
2 int po(int a, int b, int p)
3 {
4     int res=1;
5     while(b) {if(b & 1) {res=(res*one*a)%p;--b;} else
        {a=(a*one*a)%p;b>>=1;}} return res;
6 }
7 bool chprime(int n) ///miller-rabin
8 {
9     if(n==2) return true;
10    if(n<=1 || n%2==0) return false;
11    int h=n-1; int d=0; while(h%2==0) {h/=2;++d;}
12    for(int a:{2, 3, 5, 7, 11, 13, 17, 19, 23, 29,
        31, 37})
13    {
14        if(a==n) return true;
15        int u=po(a, h, n); bool ok=0;
16        if(u%n==1) continue;
17        for(int c=0; c<d; ++c)
18        {
19            if((u+1)%n==0) {ok=1; break;}
20            u=(u*one*u)%n;
21        }
22        if(!ok) return false;
23    }
24    return true;
25 }
```

1.3 Алгоритм Берлекэмпа — Мессе

<https://mzhang2021.github.io/cp-blog/berlekamp-massey/>

```

1 template<typename T>
2 vector<T> berlekampMassey(const vector<T> &s) {
3     int n = s.size(), l = 0, m = 1;
4     vector<T> b(n), c(n);
5     T ld = b[0] = c[0] = 1;
6     for (int i=0; i<n; i++, m++) {
7         T d = s[i];
8         for (int j=1; j<=l; j++)
9             d += c[j] * s[i-j];
10        if (d == 0) continue;
11        vector<T> temp = c;
12        T coef = d / ld;
13        for (int j=m; j<n; j++) c[j] -= coef * b[j-m];
14        if (2 * l <= i) {
15            l = i + 1 - l;
16            b = temp;
17            ld = d;
18            m = 0;
19        }
20    }
21    c.resize(l + 1);
22    c.erase(c.begin());
23    for (T &x : c)
24        x = -x;
25    return c;
26 }
```

2 Графы

2.1 SCC и 2-SAT

Алгоритм ищет сильносвязные компоненты в графе g , если есть путь $i \rightarrow j$, то $scc[i] \leq scc[j]$

В случае 2-SAT рёбра $i \Rightarrow j$ и $(j \oplus 1) \Rightarrow (i \oplus 1)$ должны быть добавлены одновременно.

```
1 vector<vector<int>> g(2 * n);
2 vector<vector<int>> r(g.size());
3 for (int i = 0; i < g.size(); ++i) {
4     for (int j : g[i]) r[j].push_back(i);
5 }
6 vector<int> used(g.size()), tout(g.size());
7 int time = 0;
8 auto dfs = [&](auto dfs, int cur) -> void {
9     if (used[cur]) return;
10    used[cur] = 1;
11    for (int nxt : g[cur]) {
12        dfs(dfs, nxt);
13    }
14    // used[cur] = 2;
15    tout[cur] = time++;
16 };
17 for (int i = 0; i < g.size(); ++i) if (!used[i])
18     dfs(dfs, i);
19 vector<int> ind(g.size());
20 iota(ind.begin(), ind.end(), 0);
21 sort(all(ind), [&](int i, int j){return tout[i] >
22     tout[j];});
23 vector<int> scc(g.size(), -1);
24 auto go = [&](auto go, int cur, int color) -> void
25 {
26     if (scc[cur] != -1) return;
27     scc[cur] = color;
28     for (int nxt : r[cur]) {
29         go(go, nxt, color);
30     }
31 };
32 int color = 0;
33 for (int i : ind) {
34     if (scc[i] == -1) go(go, i, color++);
35 }
36 for (int i = 0; i < g.size() / 2; ++i) {
37     if (scc[2 * i] == scc[2 * i + 1]) "IMPOSSIBLE"
38     if (scc[2 * i] < scc[2 * i + 1]) {
39         // !i => i, assign i = true
40     } else {
41         // i => !i, assign i = false
42     }
43 }
```

2.2 Эйлеров цикл

```
1 vector<vector<pair<int, int>>> g(n); // pair{nxt,
2     idx}
3 vector<pair<int, int>> e(p.size());
4 // build graph
5 vector<int> in(n), out(n);
6 for (auto [u, v] : e) in[v]++, out[u]++;
7 vector<int> used(m), it(n), cycle;
8 auto dfs = [&](auto dfs, int cur) -> void {
9     while (true) {
10        while (it[cur] < g[cur].size() && used[g[cur][
11            it[cur]].second]) it[cur]++;
12        if (it[cur] == g[cur].size()) return;
13        auto [nxt, idx] = g[cur][it[cur]];
14        used[idx] = true;
15        dfs(dfs, nxt);
16        cycle.push_back(idx);
17    }
18 };
19 int cnt = 0, odd = -1;
20 for (int i = 0; i < n; ++i){
21     if (out[i] && odd == -1) odd = i;
22     if (in[i] != out[i]) {
23         if (in[i] + 1 == out[i]) odd = i;
24         if (abs(in[i] - out[i]) > 1) return {}; // must
25             hold
26     }
27 }
```

```
23     cnt++;
24 }
25 }
26 if (cnt != 0 && cnt != 2) return {}; // must hold
27 // for undirected find odd vertex (and count that #
28     of odd is 0 or 2)
29 dfs(dfs, odd);
30 reverse(cycle.begin(), cycle.end());
31 if (cycle.size() != m) return {};
```

2.3 Компоненты рёберной двусвязности

```
1 int n, m;
2 cin >> n >> m;
3 vector <vector <int> > g(n + 1);
4 map <pair <int, int>, int> comp, col;
5 for (int i = 0; i < m; ++i) {
6     int u, v, c; cin >> u >> v >> c; c--;
7     col[{u,v}] = col[{v,u}] = c;
8     g[u].push_back(v);
9     g[v].push_back(u);
10 }
11 vector <int> used(n + 1);
12 vector <int> newCompWithoutParent(n + 1), h(n + 1),
13     up(n + 1);
14 auto findCutPoints = [&](auto self, int u, int p)
15     -> void {
16     used[u] = 1;
17     up[u] = h[u];
18     for (int v : g[u]) {
19         if (!used[v]) {
20             h[v] = h[u] + 1;
21             self(self, v, u);
22             up[u] = min(up[u], up[v]);
23             if (up[v] >= h[u]) {
24                 newCompWithoutParent[v] = 1;
25             }
26         }
27         else {
28             up[u] = min(up[u], h[v]);
29         }
30     }
31 };
32 for (int u = 1; u <= n; ++u) {
33     if (!used[u]) {
34         findCutPoints(findCutPoints, u, u);
35     }
36 }
37 int ptr = 0;
38 vector <map <int, int> > colors(m);
39 auto markComponents = [&](auto self, int u, int
40     cur) -> void {
41     used[u] = 1;
42     for (int v : g[u]) {
43         if (!used[v]) {
44             if (newCompWithoutParent[v]) {
45                 ptr++;
46                 self(self, v, ptr - 1);
47             }
48             else {
49                 self(self, v, cur);
50             }
51         }
52         else if (h[v] < h[u]) {
53             comp[{u,v}] = comp[{v,u}] = cur;
54             int c = col[{u,v}];
55             colors[cur][u] |= 1 << c;
56             colors[cur][v] |= 1 << c;
57         }
58     }
59 };
60 used.assign(n + 1, 0);
61 for (int u = 1; u <= n; ++u) {
62     if (!used[u]) {
63         markComponents(markComponents, u, -1);
64     }
65 }
66 for (int comp = 0; comp < m; ++comp) {
67     vector <int> cnt(4);
68 }
```

```

65 int tot = 0;
66 for (auto [u, mask] : colors[comp]) {
67     tot |= mask;
68     cnt[bp(mask)]++;
69 }
70 if (bp(tot)<3) {
71     continue;
72 }
73 if (cnt[2] || cnt[3]>2) {
74     cout << "Yes" << endl;
75     return;
76 }
77 }
78 cout << "No" << endl;

```

3 xor, and, or-свёртки

```

1 const int p = 998244353;
2 vector<int> band(vector<int> a, vector<int> b)
3 {
4     int n=0; while((1<<n)<a.size()) ++n;
5     a.resize(1<<n); b.resize(1<<n);
6     for(int i=0; i<n; ++i) for(int mask=0; mask<(1<<n); ++mask) if(mask & (1<<i)) {a[mask-(1<<i)]+=a[mask]; a[mask-(1<<i)]%=p;}
7     for(int i=0; i<n; ++i) for(int mask=0; mask<(1<<n); ++mask) if(mask & (1<<i)) {b[mask-(1<<i)]+=b[mask]; b[mask-(1<<i)]%=p;}
8     vector<int> c(1<<n, 0);
9     for(int mask=0; mask<(1<<n); ++mask) {c[mask]=a[mask]*b[mask]; c[mask]%=p;}
10    for(int i=0; i<n; ++i) for(int mask=0; mask<(1<<n); ++mask) if(!(mask & (1<<i))) {c[mask]-=c[mask+(1<<i)]; c[mask]%=p;}
11    return c;
12 }
13 vector<int> bor(vector<int> a, vector<int> b)
14 {
15     int n=0; while((1<<n)<a.size()) ++n;
16     a.resize(1<<n); b.resize(1<<n);
17     for(int i=0; i<n; ++i) for(int mask=0; mask<(1<<n); ++mask) if(!(mask & (1<<i))) {a[mask+(1<<i)]+=a[mask]; a[mask+(1<<i)]%=p;}
18     for(int i=0; i<n; ++i) for(int mask=0; mask<(1<<n); ++mask) if(!(mask & (1<<i))) {b[mask+(1<<i)]+=b[mask]; b[mask+(1<<i)]%=p;}
19     vector<int> c(1<<n, 0);
20     for(int mask=0; mask<(1<<n); ++mask) {c[mask]=a[mask]*b[mask]; c[mask]%=p;}
21     for(int i=0; i<n; ++i) for(int mask=0; mask<(1<<n); ++mask) if(mask & (1<<i)) {c[mask]-=c[mask-(1<<i)]; c[mask]%=p;}
22     return c;
23 }
24 vector<int> bxor(vector<int> a, vector<int> b)
25 {
26     assert(p%2==1); int inv2=(p+1)/2;
27     int n=0; while((1<<n)<a.size()) ++n;
28     a.resize(1<<n); b.resize(1<<n);
29     for(int i=0; i<n; ++i) for(int mask=0; mask<(1<<n); ++mask) if(!(mask & (1<<i))) {int u=a[mask], v=a[mask+(1<<i)]; a[mask+(1<<i)]=(u+v)%p; a[mask]=(u-v)%p;}
30     for(int i=0; i<n; ++i) for(int mask=0; mask<(1<<n); ++mask) if(!(mask & (1<<i))) {int u=b[mask], v=b[mask+(1<<i)]; b[mask+(1<<i)]=(u+v)%p; b[mask]=(u-v)%p;}
31     vector<int> c(1<<n, 0);
32     for(int mask=0; mask<(1<<n); ++mask) {c[mask]=a[mask]*b[mask]; c[mask]%=p;}
33     for(int i=0; i<n; ++i) for(int mask=0; mask<(1<<n); ++mask) if(!(mask & (1<<i))) {int u=c[mask], v=c[mask+(1<<i)]; c[mask+(1<<i)]=((v-u)*inv2)%p; c[mask]=((u+v)*inv2)%p;}
34     return c;
35 }

```

4 Структуры данных

4.1 Дерево Фенвика

```

1 int fe[maxn]; /// fenwick tree
2 void pl(int pos, int val) {while(pos<maxn) {fe[pos]+=val; pos+=(pos+1);}}
3 int get(int pos) {int ans=0; while(pos>=0) {ans+=fe[pos]; pos&=(pos+1); --pos;} return ans;} /// [0, pos] - vkluchitelno!!!
4 int get(int l, int r) {return get(r-1)-get(l-1);} // / summa na [l,r)

```

4.2 Дерево отрезков

```

1 template<typename Data, typename Mod, typename
    UniteData, typename UniteMod, typename Apply>
2 struct MassSegmentTree {
3     int h, n;
4     Data zd;
5     Mod zm;
6     vector<Data> data;
7     vector<Mod> mod;
8
9     UniteData ud; // Data (Data, Data)
10    UniteMod um; // Mod (Mod, Mod);
11    Apply a; // Data (Data, Mod, int); last argument
    is the length of current segment (could be used
    for range += and sum counting, for instance)
12
13    template<typename I>
14    MassSegmentTree(int sz, Data zd, Mod zm,
        UniteData ud, UniteMod um, Apply a, I init) : h(
        __lg(sz > 1 ? sz - 1 : 1) + 1), n(1 << h), zm(zm),
        zd(zd), data(2 * n, zd), mod(n, zm), ud(ud), um(um), a(a) {
15        for (int i = 0; i < sz; ++i) data[i + n] = init(i);
16        for (int i = n - 1; i > 0; --i) data[i] = ud(
            data[2 * i], data[2 * i + 1]);
17    }
18
19    MassSegmentTree(int sz, Data zd, Mod zm,
        UniteData ud, UniteMod um, Apply a) : h(__lg(sz
        > 1 ? sz - 1 : 1) + 1), n(1 << h), zm(zm), zd(
        zd), data(2 * n, zd), mod(n, zm), ud(ud), um(um), a(a) {}
20
21    void push(int i) {
22        if (mod[i] == zm) return;
23        apply(2 * i, mod[i]);
24        apply(2 * i + 1, mod[i]);
25        mod[i] = zm;
26    }
27
28    // is used only for apply
29    int length(int i) { return 1 << (h - __lg(i)); }
30
31    // is used only for descent
32    int left(int i) {
33        int lvl = __lg(i);
34        return (i & ((1 << lvl) - 1)) * (1 << (h - lvl));
35    }
36
37    // is used only for descent
38    int right(int i) {
39        int lvl = __lg(i);
40        return ((i & ((1 << lvl) - 1)) + 1) * (1 << (h - lvl));
41    }
42
43    template<typename S>
44    void apply(int i, S x) {
45        data[i] = a(data[i], x, length(i));
46        if (i < n) mod[i] = um(mod[i], x);
47    }
48
49    void update(int i) {

```

```

50     if (mod[i] != zm) return;
51     data[i] = ud(data[2 * i], data[2 * i + 1]);
52 }
53
54 template<typename S>
55 void update(int l, int r, S x) { // [l; r)
56     l += n, r += n;
57     for (int shift = h; shift > 0; --shift) {
58         push(l >> shift);
59         push((r - 1) >> shift);
60     }
61     for (int lf = l, rg = r; lf < rg; lf /= 2, rg
62         /= 2) {
63         if (lf & 1) apply(lf++, x);
64         if (rg & 1) apply(--rg, x);
65     }
66     for (int shift = 1; shift <= h; ++shift) {
67         update(l >> shift);
68         update((r - 1) >> shift);
69     }
70 }
71
72 Data get(int l, int r) { // [l; r)
73     l += n, r += n;
74     for (int shift = h; shift > 0; --shift) {
75         push(l >> shift);
76         push((r - 1) >> shift);
77     }
78     Data leftRes = zd, rightRes = zd;
79     for (; l < r; l /= 2, r /= 2) {
80         if (l & 1) leftRes = ud(leftRes, data[l++]);
81         if (r & 1) rightRes = ud(data[--r], rightRes);
82     }
83     return ud(leftRes, rightRes);
84 }
85
86 // l \in [0; n) && ok(get(l, l), l);
87 // returns last r: ok(get(l, r), r)
88 template<typename C>
89 int lastTrue(int l, C ok) {
90     l += n;
91     for (int shift = h; shift > 0; --shift) push(l
92         >> shift);
93     Data cur = zd;
94     do {
95         l >>= __builtin_ctz(l);
96         Data with1;
97         with1 = ud(cur, data[l]);
98         if (ok(with1, right(l))) {
99             cur = with1;
100             ++l;
101         } else {
102             while (l < n) {
103                 push(l);
104                 Data with2;
105                 with2 = ud(cur, data[2 * l]);
106                 if (ok(with2, right(2 * l))) {
107                     cur = with2;
108                     l = 2 * l + 1;
109                 } else {
110                     l = 2 * l;
111                 }
112             }
113             return l - n;
114         }
115     } while (l & (l - 1));
116     return n;
117 }
118
119 // r \in [0; n) && ok(get(r, r), r);
120 // returns first l: ok(get(l, r), l)
121 template<typename C>
122 int firstTrue(int r, C ok) {
123     r += n;
124     for (int shift = h; shift > 0; --shift) push((r
125         - 1) >> shift);
126     Data cur = zd;
127     while (r & (r - 1)) {
128         r >>= __builtin_ctz(r);

```

```

126     Data with1;
127     with1 = ud(data[--r], cur);
128     if (ok(with1, left(r))) {
129         cur = with1;
130     } else {
131         while (r < n) {
132             push(r);
133             Data with2;
134             with2 = ud(data[2 * r + 1], cur);
135             if (ok(with2, right(2 * r))) {
136                 cur = with2;
137                 r = 2 * r;
138             } else {
139                 r = 2 * r + 1;
140             }
141         }
142         return r - n + 1;
143     }
144 }
145 return 0;
146 }
147 };

```

4.2.1 Примеры использования

- Взятие максимума и прибавление константы

```

1 MassSegmentTree segtree(n, 0LL, 0LL,
2 [](int x, int y) { return max(x, y); },
3 [](int x, int y) { return x + y; },
4 [](int x, int y, int len) { return x + y; });

```

- Взятие суммы и прибавление константы

```

1 MassSegmentTree segtree(n, 0LL, 0LL,
2 [](int x, int y) { return x + y; },
3 [](int x, int y) { return x + y; },
4 [](int x, int y, int len) { return x + y * len;
5     });

```

- Взятие суммы и присвоение

```

1 MassSegmentTree segtree(n, 0LL, -1LL,
2 [](int x, int y) { return x + y; },
3 [](int x, int y) { return y; },
4 [](int x, int y, int len) { return y * len; });

```

4.3 Ordered set

```

1 #include <ext/pb_ds/assoc_container.hpp>
2 #include <ext/pb_ds/tree_policy.hpp>
3
4 using namespace __gnu_pbds;
5 using namespace std;
6
7 using ordered_set = tree<int, null_type, less<>,
8     rb_tree_tag, tree_order_statistics_node_update
9     >;

```

5 Строковые алгоритмы

5.1 Префикс-функция

```

1 vector<int> prefix_function(string s) {
2     vector<int> p(s.size());
3     for (int i = 1; i < s.size(); ++i) {
4         p[i] = p[i - 1];
5         while (p[i] && s[p[i]] != s[i]) p[i] = p[p[i] -
6             1];
7         p[i] += s[i] == s[p[i]];
8     }
9     return p;

```

5.2 Z-функция

```

1 vector<int> z_function (string s) { // z[i] - lcp
    of s and s[i:]
2     int n = (int) s.length();
3     vector<int> z (n);
4     for (int i=1, l=0, r=0; i<n; ++i) {
5         if (i <= r)
6             z[i] = min (r-i+1, z[i-l]);
7         while (i+z[i] < n && s[z[i]] == s[i+z[i]])
8             ++z[i];
9         if (i+z[i]-1 > r)
10            l = i, r = i+z[i]-1;
11     }
12     return z;
13 }

```

5.3 Алгоритм Манакера

```

1 vector<int> manacher_odd(const string &s) {
2     vector<int> man(s.size(), 0);
3     int l = 0, r = 0;
4     int n = s.size();
5     for (int i = 1; i < n; i++) {
6         if (i <= r) {
7             man[i] = min(r - i, man[l + r - i]);
8         }
9         while (i + man[i] + 1 < n && i - man[i] - 1 >=
0 && s[i + man[i] + 1] == s[i - man[i] - 1]) {
10             man[i]++;
11         }
12         if (i + man[i] > r) {
13             l = i - man[i];
14             r = i + man[i];
15         }
16     }
17     return man;
18 }
19 // abacaba : (0 1 0 3 0 1 0)
20 // abbaa : (0 0 0 0 0)
21
22 vector<int> manacher_even(const string &s) {
23     assert(s.size());
24     string t;
25     for (int i = 0; i + 1 < s.size(); ++i) {
26         t += s[i];
27         t += '#';
28     }
29     t += s.back();
30     auto odd = manacher_odd(t);
31     vector<int> ans;
32     for (int i = 1; i < odd.size(); i += 2) {
33         ans.push_back((odd[i]+1)/2);
34     }
35     return ans;
36 }
37 // abacaba : (0 0 0 0 0 0)
38 // abbaa : (0 2 0 1)

```

5.4 Суфмассив

Переработанный китайский суфмассив

```

1 const int inf = 1e9;
2 struct rmq {
3     int n;
4     vector<int> a;
5     void build(const vector<int> &x) {
6         assert(x.size() == n);
7         for (int i = 0; i < n; ++i) a[n + i] = x[i];
8         for (int i = n - 1; i > 0; --i) a[i] = min(a[2
9         * i], a[2 * i + 1]);
10    }
11    rmq(int n) : n(n), a(2 * n, inf) {}
12    void put(int i, int x) {
13        a[i + n] = min(a[i + n], x);
14        for (i = (i + n) / 2; i > 0; i /= 2) {
15            a[i] = min(a[i * 2], a[i * 2 + 1]);
16        }
17    }

```

```

16 }
17 int getMin(int l, int r) { // [l;r)
18     assert(l < r);
19     int res = inf;
20     for (l += n, r += n; l < r; l /= 2, r /= 2) {
21         if (l & 1) res = min(res, a[l++]);
22         if (r & 1) res = min(res, a[--r]);
23     }
24     return res;
25 }
26 };
27 template <typename T>
28 struct suar {
29     vector<int> sa, lcp, rank; rmq t;
30     suar (T s, int lim=256) : t((int)s.size() - 1)
31     { // s must be nonempty, 0 < s[i] < lim!
32         int n = (int)s.size() + 1, k = 0, a, b; s.
33         app(0);
34         vector<int> x(s.begin(), s.end()), y(n),
35         ws(max(n, lim)); rank.resize(n);
36         sa = lcp = y, iota(sa.begin(), sa.end(), 0)
37         ;
38         for (int j = 0, p = 0; p < n; j = max(1ll,
39         j * 2), lim = p) {
40             p = j, iota(y.begin(), y.end(), n - j);
41             for (int i = 0; i < n; i++) if (sa[i]
42             >= j) y[p++] = sa[i] - j;
43             fill(ws.begin(), ws.end(), 0);
44             for (int i = 0; i < n; i++) ws[x[i]]++;
45             for (int i = 1; i < lim; i++) ws[i] +=
46             ws[i - 1];
47             for (int i = n; i--;) sa[--ws[x[y[i]
48             ]]] = y[i];
49             swap(x, y), p = 1, x[sa[0]] = 0;
50             for (int i = 1; i < n; i++) a = sa[i -
51             1], b = sa[i], x[b] = (y[a] == y[b] && y[a + j]
52             == y[b + j]) ? p - 1 : p++;
53             for (int i = 1; i < n; i++) rank[sa[i]] = i
54             ;
55             for (int i = 0, j; i < n - 1; lcp[rank[i]
56             ++]=k)
57                 for (k && k--, j = sa[rank[i] - 1];
58                 s[i + k] == s[j + k]; k++);
59             sa.erase(sa.begin()); lcp.erase(lcp.begin()
60             ); lcp.erase(lcp.begin());
61             t.build(lcp);
62             for (auto &e : rank) {
63                 e--;
64             }
65         }
66     }
67     int getLcp(int i, int j) {
68         i = rank[i]; j = rank[j];
69         if (j < i) {
70             swap(i, j);
71         }
72         if (i == j) {
73             return inf;
74         }
75         else {
76             return t.getMin(i, j);
77         }
78     }
79 }
80 };

```

5.5 Алгоритм Ахо — Корасик

```

1 struct node{
2     int next[alpha] = {}, link[alpha] = {};
3     int suf = 0;
4     ll visited = 0, ans = 0;
5     vector<int> term;
6     node() {}
7 };
8
9 vector<node> mem;
10
11 int get_next(int nd, char c) {

```

```

12 if (!mem[nd].next[c - a]) { mem[nd].next[c - a] =
    mem.size(); mem.emplace_back(); }
13 return mem[nd].next[c - a];
14 }
15
16 void find(string s, vector<string> t) {
17     mem.reserve(1e6 + 100); mem.clear();
18     mem.emplace_back(); mem.emplace_back();
19     // 0th element is nullptr, 1st is the root
20     int q = t.size();
21     for (int j = 0; j < q; ++j) {
22         int cur = 1;
23         for (char c : ts[j]) cur = get_next(cur, c);
24         mem[cur].term.push_back(j);
25     }
26     vector<int> bfs_order;
27     queue<int> bfs;
28     {
29         node &root = mem[1];
30         root.suf = 1;
31         for (char c = a; c < a + alpha; ++c) {
32             root.link[c - a] = (root.next[c - a] ?
33                 root.next[c - a] : 1);
34             bfs.push(1);
35         }
36         while (!bfs.empty()) {
37             int cur_idx = bfs.front();
38             bfs.pop();
39             node &cur = mem[cur_idx];
40             bfs_order.push_back(cur_idx);
41             for (char c = a; c < a + alpha; ++c) {
42                 int nxt_idx = cur.next[c - a];
43                 if (!nxt_idx) continue;
44                 node &nxt = mem[nxt_idx];
45                 nxt.suf = (cur_idx == 1 ? 1 : mem[cur.suf].
46                     link[c - a]);
47                 for (char c = a; c < a + alpha; ++c) {
48                     nxt.link[c - a] = (nxt.next[c - a] ? nxt.
49                         next[c - a] : mem[nxt.suf].link[c - a]);
50                 }
51                 bfs.push(nxt_idx);
52             }
53         }
54     }
55     // do something
56 }

```

5.6 Дерево палиндромов

```

1 struct palindromic{
2     int n;
3     vector<int> p, suf{0, 0}, len{-1, 0};
4     vector<array<int, alpha>> to{{}, {}};
5     int sz = 2;
6
7     palindromic(const string &s) : n(s.size()), p(n +
8         1, 1) {
9         suf.reserve(n);
10        len.reserve(n);
11        for (int i = 0; i < n; ++i) {
12            auto check = [&](int l) { return i > l && s[i]
13                == s[i - l - 1]; };
14            int par = p[i];
15            while (!check(len[par])) par = suf[par];
16            if (to[par][s[i] - a]) {
17                p[i + 1] = to[par][s[i] - a];
18                continue;
19            }
20            p[i + 1] = sz++;
21            to[par][s[i] - a] = p[i + 1];
22            to.emplace_back();
23            len.emplace_back(len[par] + 2);
24            do {
25                par = suf[par];
26            } while (!check(len[par]));
27            int link = to[par][s[i] - a];
28            if (link == p[i + 1]) link = 1;
29            suf.emplace_back(link);
30        }
31    }
32 }

```

6 Потоки

6.1 Алгоритм Диница

```

1 #define pb push_back
2 struct Dinic{
3     struct edge{
4         int to, flow, cap;
5     };
6
7     const static int N = 555; //count of vertices
8
9     vector<edge> e;
10    vector<int> g[N + 7];
11    int dp[N + 7];
12    int ptr[N + 7];
13
14    void clear(){
15        for (int i = 0; i < N + 7; i++) g[i].clear();
16        e.clear();
17    }
18
19    void addEdge(int a, int b, int cap){
20        g[a].pb(e.size());
21        e.pb({b, 0, cap});
22        g[b].pb(e.size());
23        e.pb({a, 0, 0});
24    }
25
26    int minFlow, start, finish;
27
28    bool bfs(){
29        for (int i = 0; i < N; i++) dp[i] = -1;
30        dp[start] = 0;
31        vector<int> st;
32        int uk = 0;
33        st.pb(start);
34        while(uk < st.size()){
35            int v = st[uk++];
36            for (int to : g[v]){
37                auto ed = e[to];
38                if (ed.cap - ed.flow >= minFlow && dp[ed.to]
39                    == -1){
40                    dp[ed.to] = dp[v] + 1;
41                    st.pb(ed.to);
42                }
43            }
44        }
45        return dp[finish] != -1;
46    }
47
48    int dfs(int v, int flow){
49        if (v == finish) return flow;
50        for (; ptr[v] < g[v].size(); ptr[v]++){
51            int to = g[v][ptr[v]];
52            edge ed = e[to];
53            if (ed.cap - ed.flow >= minFlow && dp[ed.to] ==
54                dp[v] + 1){
55                int add = dfs(ed.to, min(flow, ed.cap - ed.
56                    flow));
57                if (add){
58                    e[to].flow += add;
59                    e[to ^ 1].flow -= add;
60                    return add;
61                }
62            }
63        }
64        return 0;
65    }
66
67    int dinic(int start, int finish){
68        Dinic::start = start;
69        Dinic::finish = finish;
70        int flow = 0;
71        for (minFlow = (1 << 30); minFlow; minFlow >>= 1)
72            {
73                while(bfs()){
74                    for (int i = 0; i < N; i++) ptr[i] = 0;
75                    while(int now = dfs(start, (int)2e9 + 7))
76                        flow += now;
77                }
78            }
79    }

```

```

72     }
73 }
74 return flow;
75 }
76 } dinic;

```

6.2 Mincost k-flow

6.2.1 Строим граф

```

1 struct edge {
2     int next, capacity, cost, flow = 0;
3
4     edge() = default;
5
6     edge(int next, int capacity, int cost) : next(
7         next), capacity(capacity), cost(cost) {}
8
9     int rem() const { return capacity - flow; }
10
11     int operator+=(int f) { return flow += f; }
12
13     int operator-=(int f) { return flow -= f; }
14 };
15 auto addEdge = [&](auto from, auto next, auto
16     capacity, int cost) {
17     g[from].push_back(e.size());
18     e.emplace_back(next, capacity, cost);
19     g[next].push_back(e.size());
20     e.emplace_back(from, 0, -cost);
21 };

```

Если граф ориентированный, то addEdge вызываем один раз. Если неориентированный, то два, вот так:

```

1 addEdge(u, v, capacity, cost);
2 addEdge(v, u, capacity, cost);

```

6.2.2 Запускаем Форда — Беллмана

```

1 vector<ll> phi(n, 0);
2 auto fordBellman = [&](int s, int t) {
3     phi.assign(n, 0);
4     for (int iter = 0; iter < n; ++iter) {
5         bool changed = false;
6         for (int u = 0; u < n; ++u) {
7             for (auto index : g[u]) {
8                 auto edge = e[index];
9                 if (edge.rem() > 0 && phi[edge.next] > phi[
10                     u] + edge.cost) {
11                     phi[edge.next] = phi[u] + edge.cost;
12                     changed = true;
13                 }
14             }
15         }
16         if (!changed) break;
17     };
18     fordBellman(s, t);
19 };

```

6.2.3 Ищем кратчайший путь Дейкстры с потенциалами

```

1 vector<ll> dist;
2 vector<int> from;
3 vector<bool> cnt;
4 auto dijkstra = [&](int s, int t) {
5     dist.assign(n, 1e18);
6     from.assign(n, -1);
7     cnt.assign(n, false);
8     dist[s] = 0;
9     set<pair<int, int>> se;
10    se.insert({0, s});
11    while ((int)(se.size())) {
12        int cur = se.begin()->y;
13        se.erase(se.begin());
14        cnt[cur] = true;
15        for (int index : g[cur]) {
16            auto &edge = e[index];
17            if (edge.rem() == 0) continue;

```

```

18            ll weight = edge.cost + phi[cur] - phi[edge.
19                next];
20            if (dist[edge.next] > dist[cur] + weight) {
21                se.erase({dist[edge.next], edge.next});
22                dist[edge.next] = dist[cur] + weight;
23                se.insert({dist[edge.next], edge.next});
24                from[edge.next] = cur;
25            }
26        }
27    }
28    if (dist[t] == (ll) 1e18) return -1LL;
29    ll cost = 0;
30    for (int p = t; p != s; p = from[p]) {
31        for (auto index : g[from[p]]) {
32            auto &edge = e[index];
33            ll weight = edge.cost + phi[from[p]] - phi[
34                edge.next];
35            if (edge.rem() > 0 && edge.next == p && dist[
36                edge.next] == dist[from[p]] + weight) {
37                edge += 1;
38                e[index ^ 1] -= 1;
39                cost += edge.cost;
40                break;
41            }
42        }
43    }
44    return cost;
45 };
46 ll cost = 0;
47 for (int flow = 0; flow < k; ++flow) {
48     ll a = dijkstra(s, t);
49     if (a == -1) {
50         cout << "-1\n";
51         return;
52     }
53     cost += a;
54 }

```

6.2.4 Восстанавливаем ответ

```

1 auto findPath = [&](int s, int t) {
2     vector<int> ans;
3     int cur = s;
4     while (cur != t) {
5         for (auto index : g[cur]) {
6             auto &edge = e[index];
7             if (edge.flow <= 0) continue;
8             edge -= 1;
9             e[index ^ 1] += 1;
10            ans.push_back(index / 4);
11            // index / 4 because each edge has 4 copies
12            cur = edge.next;
13            break;
14        }
15    }
16    return ans;
17 };
18 for (int flow = 0; flow < k; ++flow) {
19     auto p = findPath(s, t);
20     cout << p.size() << ' ';
21     for (int x : p) cout << x + 1 << ' ';
22     cout << '\n';
23 }

```

7 FFT & co

7.1 NTT & co

```

1 #define int long long
2 using namespace std;
3 typedef long long ll;
4 const int p=998244353;
5 int po(int a,int b){if(b==0) return 1; if(b==1)
6     return a; if(b%2==0){int u=po(a,b/2);return (u
7     *1LL*u)%p;} else {int u=po(a,b-1);return (a*1LL
8     *u)%p;}}

```

```

6 int inv(int x) {return po(x,p-2);}
7 template<int M, int K, int G> struct Fft {
8     // 1, 1/4, 1/8, 3/8, 1/16, 5/16, 3/16, 7/16, ...
9     int g[1 << (K - 1)];
10    Fft() : g() { //if t1 constexpr...
11        static_assert(K >= 2, "Fft: K >= 2 must hold");
12        g[0] = 1;
13        g[1 << (K - 2)] = G;
14        for (int l = 1 << (K - 2); l >= 2; l >>= 1) {
15            g[l >> 1] = (g[l] * 1LL * g[l]) % M;
16        }
17        assert((g[1]*1LL * g[1]) % M == M - 1);
18        for (int l = 2; l <= 1 << (K - 2); l <<= 1) {
19            for (int i = 1; i < l; ++i) {
20                g[l + i] = (g[l] * 1LL * g[i]) % M;
21            }
22        }
23    }
24    void fft(vector<int> &x) const {
25        const int n = x.size();
26        assert(n <= 1 << K);
27        for (int h = __builtin_ctz(n); h--; ) {
28            const int l = (1 << h);
29            for (int i = 0; i < n >> (h+1); ++i) {
30                for (int j = i << (h+1); j < (((i << 1) +
31                    1) << h); ++j) {
32                    const int t = (g[i] * 1LL * x[j | l]) % M;
33                    x[j | l] = x[j] - t;
34                    if (x[j|l] < 0) x[j | l] += M;
35                    x[j] += t;
36                    if (x[j] >= M) x[j] -= M;
37                }
38            }
39            for (int i = 0, j = 0; i < n; ++i) {
40                if (i < j) std::swap(x[i], x[j]);
41                for (int l = n; (l >>= 1) && !((j ^= 1) & 1);
42                    ) {}
43            }
44            vector<int> convolution(const vector<int> &a,
45                const vector<int> &b) const {
46                if(a.empty() || b.empty()) return {};
47                const int na = a.size(), nb = b.size();
48                int n, invN = 1;
49                for (n = 1; n < na + nb - 1; n <<= 1) invN = ((
50                    invN & 1) ? (invN + M) : invN) >> 1;
51                vector<int> x(n, 0), y(n, 0);
52                std::copy(a.begin(), a.end(), x.begin());
53                std::copy(b.begin(), b.end(), y.begin());
54                fft(x);
55                fft(y);
56                for (int i = 0; i < n; ++i) x[i] = (((
57                    static_cast<long long>(x[i]) * y[i]) % M) *
58                    invN) % M;
59                std::reverse(x.begin() + 1, x.end());
60                fft(x);
61                x.resize(na + nb - 1);
62                return x;
63            }
64        };
65        Fft<998244353,23,31> muls;
66        vector<int> form(vector<int> v, int n)
67        {
68            while(v.size()<n) v.push_back(0);
69            while(v.size()>n) v.pop_back();
70            return v;
71        }
72        vector<int> operator *(vector<int> v1,vector<int>
73            v2)
74        {
75            return muls.convolution(v1,v2);
76        }
77        vector<int> operator +(vector<int> v1,vector<int>
78            v2)
79        {
80            while(v2.size()<v1.size()) v2.push_back(0); while
81                (v1.size()<v2.size()) v1.push_back(0);
82            for(int i=0;i<v1.size();++i) {v1[i]+=v2[i];if(v1[
83                i]>=p) v1[i]-=p; else if(v1[i]<0) v1[i]+=p;}
84        }
85    };
86    }
87    }
88    }
89    }
90    }
91    }
92    }
93    }
94    }
95    }
96    }
97    }
98    }
99    }
100    }
101    }
102    }
103    }
104    }
105    }
106    }
107    }
108    }
109    }
110    }
111    }
112    }
113    }
114    }
115    }
116    }
117    }
118    }
119    }
120    }
121    }
122    }
123    }
124    }
125    }
126    }
127    }
128    }
129    }
130    }
131    }
132    }
133    }
134    }
135    }
136    }
137    }
138    }
139    }
140    }
141    }
142    }
143    }
144    }
145    }
146    }
147    }
148    }
149    }
150    }
151    }
152    }
153    }
154    }
155    }
156    }
157    }
158    }
159    }
160    }
161    }
162    }
163    }
164    }
165    }
166    }
167    }
168    }
169    }
170    }
171    }
172    }
173    }
174    }
175    }
176    }
177    }
178    }
179    }
180    }
181    }
182    }
183    }
184    }
185    }
186    }
187    }
188    }
189    }
190    }
191    }
192    }
193    }
194    }
195    }
196    }
197    }
198    }
199    }
200    }
201    }
202    }
203    }
204    }
205    }
206    }
207    }
208    }
209    }
210    }
211    }
212    }
213    }
214    }
215    }
216    }
217    }
218    }
219    }
220    }
221    }
222    }
223    }
224    }
225    }
226    }
227    }
228    }
229    }
230    }
231    }
232    }
233    }
234    }
235    }
236    }
237    }
238    }
239    }
240    }
241    }
242    }
243    }
244    }
245    }
246    }
247    }
248    }
249    }
250    }
251    }
252    }
253    }
254    }
255    }
256    }
257    }
258    }
259    }
260    }
261    }
262    }
263    }
264    }
265    }
266    }
267    }
268    }
269    }
270    }
271    }
272    }
273    }
274    }
275    }
276    }
277    }
278    }
279    }
280    }
281    }
282    }
283    }
284    }
285    }
286    }
287    }
288    }
289    }
290    }
291    }
292    }
293    }
294    }
295    }
296    }
297    }
298    }
299    }
300    }
301    }
302    }
303    }
304    }
305    }
306    }
307    }
308    }
309    }
310    }
311    }
312    }
313    }
314    }
315    }
316    }
317    }
318    }
319    }
320    }
321    }
322    }
323    }
324    }
325    }
326    }
327    }
328    }
329    }
330    }
331    }
332    }
333    }
334    }
335    }
336    }
337    }
338    }
339    }
340    }
341    }
342    }
343    }
344    }
345    }
346    }
347    }
348    }
349    }
350    }
351    }
352    }
353    }
354    }
355    }
356    }
357    }
358    }
359    }
360    }
361    }
362    }
363    }
364    }
365    }
366    }
367    }
368    }
369    }
370    }
371    }
372    }
373    }
374    }
375    }
376    }
377    }
378    }
379    }
380    }
381    }
382    }
383    }
384    }
385    }
386    }
387    }
388    }
389    }
390    }
391    }
392    }
393    }
394    }
395    }
396    }
397    }
398    }
399    }
400    }
401    }
402    }
403    }
404    }
405    }
406    }
407    }
408    }
409    }
410    }
411    }
412    }
413    }
414    }
415    }
416    }
417    }
418    }
419    }
420    }
421    }
422    }
423    }
424    }
425    }
426    }
427    }
428    }
429    }
430    }
431    }
432    }
433    }
434    }
435    }
436    }
437    }
438    }
439    }
440    }
441    }
442    }
443    }
444    }
445    }
446    }
447    }
448    }
449    }
450    }
451    }
452    }
453    }
454    }
455    }
456    }
457    }
458    }
459    }
460    }
461    }
462    }
463    }
464    }
465    }
466    }
467    }
468    }
469    }
470    }
471    }
472    }
473    }
474    }
475    }
476    }
477    }
478    }
479    }
480    }
481    }
482    }
483    }
484    }
485    }
486    }
487    }
488    }
489    }
490    }
491    }
492    }
493    }
494    }
495    }
496    }
497    }
498    }
499    }
500    }
501    }
502    }
503    }
504    }
505    }
506    }
507    }
508    }
509    }
510    }
511    }
512    }
513    }
514    }
515    }
516    }
517    }
518    }
519    }
520    }
521    }
522    }
523    }
524    }
525    }
526    }
527    }
528    }
529    }
530    }
531    }
532    }
533    }
534    }
535    }
536    }
537    }
538    }
539    }
540    }
541    }
542    }
543    }
544    }
545    }
546    }
547    }
548    }
549    }
550    }
551    }
552    }
553    }
554    }
555    }
556    }
557    }
558    }
559    }
560    }
561    }
562    }
563    }
564    }
565    }
566    }
567    }
568    }
569    }
570    }
571    }
572    }
573    }
574    }
575    }
576    }
577    }
578    }
579    }
580    }
581    }
582    }
583    }
584    }
585    }
586    }
587    }
588    }
589    }
590    }
591    }
592    }
593    }
594    }
595    }
596    }
597    }
598    }
599    }
600    }
601    }
602    }
603    }
604    }
605    }
606    }
607    }
608    }
609    }
610    }
611    }
612    }
613    }
614    }
615    }
616    }
617    }
618    }
619    }
620    }
621    }
622    }
623    }
624    }
625    }
626    }
627    }
628    }
629    }
630    }
631    }
632    }
633    }
634    }
635    }
636    }
637    }
638    }
639    }
640    }
641    }
642    }
643    }
644    }
645    }
646    }
647    }
648    }
649    }
650    }
651    }
652    }
653    }
654    }
655    }
656    }
657    }
658    }
659    }
660    }
661    }
662    }
663    }
664    }
665    }
666    }
667    }
668    }
669    }
670    }
671    }
672    }
673    }
674    }
675    }
676    }
677    }
678    }
679    }
680    }
681    }
682    }
683    }
684    }
685    }
686    }
687    }
688    }
689    }
690    }
691    }
692    }
693    }
694    }
695    }
696    }
697    }
698    }
699    }
700    }
701    }
702    }
703    }
704    }
705    }
706    }
707    }
708    }
709    }
710    }
711    }
712    }
713    }
714    }
715    }
716    }
717    }
718    }
719    }
720    }
721    }
722    }
723    }
724    }
725    }
726    }
727    }
728    }
729    }
730    }
731    }
732    }
733    }
734    }
735    }
736    }
737    }
738    }
739    }
740    }
741    }
742    }
743    }
744    }
745    }
746    }
747    }
748    }
749    }
750    }
751    }
752    }
753    }
754    }
755    }
756    }
757    }
758    }
759    }
760    }
761    }
762    }
763    }
764    }
765    }
766    }
767    }
768    }
769    }
770    }
771    }
772    }
773    }
774    }
775    }
776    }
777    }
778    }
779    }
780    }
781    }
782    }
783    }
784    }
785    }
786    }
787    }
788    }
789    }
790    }
791    }
792    }
793    }
794    }
795    }
796    }
797    }
798    }
799    }
800    }
801    }
802    }
803    }
804    }
805    }
806    }
807    }
808    }
809    }
810    }
811    }
812    }
813    }
814    }
815    }
816    }
817    }
818    }
819    }
820    }
821    }
822    }
823    }
824    }
825    }
826    }
827    }
828    }
829    }
830    }
831    }
832    }
833    }
834    }
835    }
836    }
837    }
838    }
839    }
840    }
841    }
842    }
843    }
844    }
845    }
846    }
847    }
848    }
849    }
850    }
851    }
852    }
853    }
854    }
855    }
856    }
857    }
858    }
859    }
860    }
861    }
862    }
863    }
864    }
865    }
866    }
867    }
868    }
869    }
870    }
871    }
872    }
873    }
874    }
875    }
876    }
877    }
878    }
879    }
880    }
881    }
882    }
883    }
884    }
885    }
886    }
887    }
888    }
889    }
890    }
891    }
892    }
893    }
894    }
895    }
896    }
897    }
898    }
899    }
900    }
901    }
902    }
903    }
904    }
905    }
906    }
907    }
908    }
909    }
910    }
911    }
912    }
913    }
914    }
915    }
916    }
917    }
918    }
919    }
920    }
921    }
922    }
923    }
924    }
925    }
926    }
927    }
928    }
929    }
930    }
931    }
932    }
933    }
934    }
935    }
936    }
937    }
938    }
939    }
940    }
941    }
942    }
943    }
944    }
945    }
946    }
947    }
948    }
949    }
950    }
951    }
952    }
953    }
954    }
955    }
956    }
957    }
958    }
959    }
960    }
961    }
962    }
963    }
964    }
965    }
966    }
967    }
968    }
969    }
970    }
971    }
972    }
973    }
974    }
975    }
976    }
977    }
978    }
979    }
980    }
981    }
982    }
983    }
984    }
985    }
986    }
987    }
988    }
989    }
990    }
991    }
992    }
993    }
994    }
995    }
996    }
997    }
998    }
999    }
1000   }

```

```

76 return v1;
77 }
78 vector<int> operator -(vector<int> v1,vector<int>
79     v2)
80 {
81     int sz=max(v1.size(),v2.size());while(v1.size()<
82         sz) v1.push_back(0); while(v2.size()<sz) v2.
83         push_back(0);
84     for(int i=0;i<sz;++i) {v1[i]-=v2[i];if(v1[i]<0)
85         v1[i]+=p; else if(v1[i]>=p) v1[i]-=p;} return
86         v1;
87 }
88 vector<int> trmi(vector<int> v)
89 {
90     for(int i=1;i<v.size();i+=2) {if(v[i]>0) v[i]=p-v
91         [i]; else v[i]=(-v[i]);}
92     return v;
93 }
94 vector<int> deriv(vector<int> v)
95 {
96     if(v.empty()) return{};
97     vector<int> ans(v.size()-1);
98     for(int i=1;i<v.size();++i) ans[i-1]=(v[i]*1LL*i)
99         %p;
100    return ans;
101 }
102 vector<int> integ(vector<int> v)
103 {
104     vector<int> ans(v.size()+1);ans[0]=0;
105     for(int i=1;i<v.size();++i) ans[i-1]=(v[i]*1LL*i)
106         %p;
107    return ans;
108 }
109 vector<int> mul(vector<vector<int> > v)
110 {
111     if(v.size()==1) return v[0];
112     vector<vector<int> > v1,v2;for(int i=0;i<v.size()
113         /2;++i) v1.push_back(v[i]); for(int i=v.size()
114         /2;i<v.size();++i) v2.push_back(v[i]);
115     return muls.convolution(mul(v1),mul(v2));
116 }
117 vector<int> inv1(vector<int> v,int n)
118 {
119     assert(v[0]!=0);
120     int sz=1;v=form(v,n);vector<int> a={inv(v[0])};
121     while(sz<n)
122     {
123         vector<int> vsz;for(int i=0;i<min(n,2*sz);++i)
124             vsz.push_back(v[i]);
125         vector<int> b=((vector<int>) {1})-muls.
126             convolution(a,vsz);
127         for(int i=0;i<sz;++i) assert(b[i]==0);
128         b.erase(b.begin(),b.begin()+sz);
129         vector<int> c=muls.convolution(b,a);
130         for(int i=0;i<sz;++i) a.push_back(c[i]);
131         sz*=2;
132     }
133     return form(a,n);
134 }
135 }
136 }
137 }
138 }
139 }
140 }
141 }
142 }
143 }
144 }
145 }
146 }
147 }
148 }
149 }
150 }
151 }
152 }
153 }
154 }
155 }
156 }
157 }
158 }
159 }
160 }
161 }
162 }
163 }
164 }
165 }
166 }
167 }
168 }
169 }
170 }
171 }
172 }
173 }
174 }
175 }
176 }
177 }
178 }
179 }
180 }
181 }
182 }
183 }
184 }
185 }
186 }
187 }
188 }
189 }
190 }
191 }
192 }
193 }
194 }
195 }
196 }
197 }
198 }
199 }
200 }
201 }
202 }
203 }
204 }
205 }
206 }
207 }
208 }
209 }
210 }
211 }
212 }
213 }
214 }
215 }
216 }
217 }
218 }
219 }
220 }
221 }
222 }
223 }
224 }
225 }
226 }
227 }
228 }
229 }
230 }
231 }
232 }
233 }
234 }
235 }
236 }
237 }
238 }
239 }
240 }
241 }
242 }
243 }
244 }
245 }
246 }
247 }
248 }
249 }
250 }
251 }
252 }
253 }
254 }
255 }
256 }
257 }
258 }
259 }
260 }
261 }
262 }
263 }
264 }
265 }
266 }
267 }
268 }
269 }
270 }
271 }
272 }
273 }
274 }
275 }
276 }
277 }
278 }
279 }
280 }
281 }
282 }
283 }
284 }
285 }
286 }
287 }
288 }
289 }
290 }
291 }
292 }
293 }
294 }
295 }
296 }
297 }
298 }
299 }
300 }
301 }
302 }
303 }
304 }
305 }
306 }
307 }
308 }
309 }
310 }
311 }
312 }
313 }
314 }
315 }
316 }
317 }
318 }
319 }
320 }
321 }
322 }
323 }
324 }
325 }
326 }
327 }
328 }
329 }
330 }
331 }
332 }
333 }
334 }
335 }
336 }
337 }
338 }
339 }
340 }
341 }
342 }
343 }
344 }
345 }
346 }
347 }
348 }
349 }
350 }
351 }
352 }
353 }
354 }
355 }
356 }
357 }
358 }
359 }
360 }
361 }
362 }
363 }
364 }
365 }
366 }
367 }
368 }
369 }
370 }
371 }
372 }
373 }
374 }
375 }
376 }
377 }
378 }
379 }
380 }
381 }
382 }
383 }
384 }
385 }
386 }
387 }
388 }
389 }
390 }
391 }
392 }
393 }
394 }
395 }
396 }
397 }
398 }
399 }
400 }
401 }
402 }
403 }
404 }
405 }
406 }
407 }
408 }
409 }
410 }
411 }
412 }
413 }
414 }
415 }
416 }
417 }
418 }
419 }
420 }
421 }
422 }
423 }
424 }
425 }
426 }
427 }
428 }
429 }
430 }
431 }
432 }
433 }
434 }
435 }
436 }
437 }
438 }
439 }
440 }
441 }
442 }
443 }
444 }
445 }
446 }
447 }
448 }
449 }
450 }
451 }
452 }
453 }
454 }
455 }
456 }
457 }
458 }
459 }
460 }
461 }
462 }
463 }
464 }
465 }
466 }
467 }
468 }
469 }
470 }
471 }
472 }
473 }
474 }
475 }
476 }
477 }
478 }
479 }
480 }
481 }
482 }
483 }
484 }
485 }
486 }
487 }
488 }
489 }
490 }
491 }
492 }
493 }
494 }
495 }
496 }
497 }
498 }
499 }
500 }
501 }
502 }
503 }
504 }
505 }
506 }
507 }
508 }
509 }
510 }
511 }
512 }
513 }
514 }
515 }
516 }
517 }
518 }
519 }
520 }
521 }
522 }
523 }
524 }
525 }
526 }
527 }
528 }
529 }
530 }
531 }
532 }
533 }
534 }
535 }
536 }
537 }
538 }
539 }
540 }
541 }
542 }
543 }
544 }
545 }
546 }
547 }
548 }
549 }
550 }
551 }
552 }
553 }
554 }
555 }
556 }
557 }
558 }
559 }
560 }
561 }
562 }
563 }
564 }
565 }
566 }
567 }
568 }
569 }
570 }
571 }
572 }
573 }
574 }
575 }
576 }
577 }
578 }
579 }
580 }
581 }
582 }
583 }
584 }
585 }
586 }
587 }
588 }
589 }
590 }
591 }
592 }
593 }
594 }
595 }
596 }
597 }
598 }
599 }
600 }
601 }
602 }
603 }
604 }
605 }
606 }
607 }
608 }
609 }
610 }
611 }
612 }
613 }
614 }
615 }
616 }
617 }
618 }
619 }
620 }
621 }
622 }
623 }
624 }
625 }
626 }
627 }
628 }
629 }
630 }
631 }
632 }
633 }
634 }
635 }
636 }
637 }
638 }
639 }
640 }
641 }
642 }
643 }
644 }
645 }
646 }
647 }
648 }
649 }
650 }
651 }
652 }
653 }
654 }
655 }
656 }
657 }
658 }
659 }
660 }
661 }
662 }
663 }
664 }
665 }
666 }
667 }
668 }
669 }
670 }
671 }
672 }
673 }
674 }
675 }
676 }
677 }
678 }
679 }
680 }
681 }
682 }
683 }
684 }
685 }
686 }
687 }
688 }
689 }
690 }
691 }
692 }
693 }
694 }
695 }
696 }
697 }
698 }
699 }
700 }
701 }
702 }
703 }
704 }
705 }
706 }
707 }
708 }
709 }
710 }
711 }
712 }
713 }
714 }
715 }
716 }
717 }
718 }
719 }
720 }
721 }
722 }
723 }
724 }
725 }
726 }
727 }
728 }
729 }
730 }
731 }
732 }
733 }
734 }
735 }
736 }
737 }
738 }
739 }
740 }
741 }
742 }
743 }
744 }
745 }
746 }
747 }
748 }
749 }
750 }
751 }
752 }
753 }
754 }
755 }
756 }
757 }
758 }
759 }
760 }
761 }
762 }
763 }
764 }
765 }
766 }
767 }
768 }
769 }
770 }
771 }
772 }
773 }
774 }
775 }
776 }
777 }
778 }
779 }
780 }
781 }
782 }
783 }
784 }
785 }
786 }
787 }
788 }
789 }
790 }
791 }
792 }
793 }
794 }
795 }
796 }
797 }
798 }
799 }
800 }
801 }
802 }
803 }
804 }
805 }
806 }
807 }
808 }
809 }
810 }
811 }
812 }
813 }
814 }
815 }
816 }
817 }
818 }
819 }
820 }
821 }
822 }
823 }
824 }
825 }
826 }
827 }
828 }
829 }
830 }
831 }
832 }
833 }
834 }
835 }
836 }
837 }
838 }
839 }
840 }
841 }
842 }
843 }
844 }
845 }
846 }
847 }
848 }
849 }
850 }
851 }
852 }
853 }
854 }
855 }
856 }
857 }
858 }
859 }
860 }
861 }
862 }
863 }
864 }
865 }
866 }
867 }
868 }
869 }
870 }
871 }
872 }
873 }
874 }
875 }
876 }
877 }
878 }
879 }
880 }
881 }
882 }
883 }
884 }
885 }
886 }
887 }
888 }
889 }
890 }
891 }
892 }
893 }
894 }
895 }
896 }
897 }
898 }
899 }
900 }
901 }
902 }
903 }
904 }
905 }
906 }
907 }
908 }
909 }
910 }
911 }
912 }
913 }
914 }
915 }
916 }
917 }
918 }
919 }
920 }
921 }
922 }
923 }
924 }
925 }
926 }
927 }
928 }
929 }
930 }
931 }
932 }
933 }
934 }
935 }
936 }
937 }
938 }
939 }
940 }
941 }
942 }
943 }
944 }
945 }
946 }
947 }
948 }
949 }
950 }
951 }
952 }
953 }
954 }
955 }
956 }
957 }
958 }
959 }
960 }
961 }
962 }
963 }
964 }
965 }
966 }
967 }
968 }
969 }
970 }
971 }
972 }
973 }
974 }
975 }
976 }
977 }
978 }
979 }
980 }
981 }
982 }
983 }
984 }
985 }
986 }
987 }
988 }
989 }
990 }
991 }
992 }
993 }
994 }
995 }
996 }
997 }
998 }
999 }
1000 }

```

7.2 старое доброе FFT

```

1 using cd = complex<double>;
2 const double PI = acos(-1);
3
4 void fft(vector<cd> &a, bool invert) {
5     int n = a.size();

```



```

18     double ang = 2 * PI / len * (invert ? -1 : 1);
19     cd wlen(cos(ang), sin(ang));
20     for (int i = 0; i < n; i += len) {
21         cd w(1);
22         for (int j = 0; j < len / 2; j++) {
23             cd u = a[i+j], v = a[i+j+len/2] * w;
24             a[i+j] = u + v;
25             a[i+j+len/2] = u - v;
26             w *= wlen;
27         }
28     }
29 }
30
31 if (invert) {
32     for (cd & x : a)
33         x /= n;
34 }
35 }
36 vector<int> multiply(vector<int> const& a, vector<
37     int> const& b) {
38     vector<cd> fa(a.begin(), a.end()), fb(b.begin(),
39         b.end());
40     int n = 1;
41     while (n < a.size() + b.size())
42         n <<= 1;
43     fa.resize(n);
44     fb.resize(n);
45     fft(fa, false);
46     fft(fb, false);
47     for (int i = 0; i < n; i++)
48         fa[i] *= fb[i];
49     fft(fa, true);
50     vector<int> result(n);
51     for (int i = 0; i < n; i++)
52         result[i] = round(fa[i].real());
53     while(!result.empty() && !result.back()) result.
54         pop_back();
55     return result;
56 }

```

8 Геометрия

8.1 Касательные

```

1     auto max = [&] (auto cmp) {
2         int k = 0;
3         for (int lg = 18; lg >= 0; --lg) {
4             int i = k + (1 << lg), j = k - (1 << lg);
5
6             i = (i % n + n) % n;
7             j = (j % n + n) % n;
8             array<int, 3> ind{i, j, k};
9             sort(all(ind), cmp);
10            k = ind[2];
11        }
12    };
13    auto uppert = [&](Point p) { //last vertex in
14        counterclockwise order about p
15        auto cmp = [&] (int i, int j) {return (a[i]
16            - p) < (a[j] - p); };
17        return max(cmp);
18    };
19    auto lowert = [&](Point p) { //first vertex in
20        counterclockwise order about p
21        auto cmp = [&] (int i, int j) {return (a[i]
22            - p) > (a[j] - p); };
23        return max(cmp);
24    };
25    auto uppertinf = [&](Point p) { //upper tangent
26        line parallel to vector p
27        swap(p.x, p.y);
28        p.x = -p.x;
29        auto cmp = [&] (int i, int j) { return a[i]
30            % p < a[j] % p; };
31        return max(cmp);
32    };

```

```

27     auto lowertinf = [&](Point p) { //lower tangent
28         line parallel to vector p
29         swap(p.x, p.y);
30         p.x = -p.x;
31         auto cmp = [&] (int i, int j) { return a[i]
32             % p > a[j] % p; };
33         return max(cmp);
34     };

```

8.2 Примитивы

```

1 struct Point {
2     int x, y;
3     Point(){}
4     Point (int x_, int y_) {
5         x = x_; y = y_;
6     }
7     Point operator + (Point p) {
8         return Point(x+p.x,y+p.y);
9     }
10    Point operator - (Point p) {
11        return Point(x - p.x, y - p.y);
12    }
13    int operator * (Point p) {
14        return x * p.y - y * p.x;
15    }
16    int operator % (Point p) {
17        return x * p.x + y * p.y;
18    }
19    bool operator < (Point v) {
20        return (*this) * v > 0;
21    }
22    bool operator > (Point v) {
23        return v < (*this);
24    };
25    bool operator <= (Point v) {
26        return (*this) * v >= 0;
27    }
28 };
29 bool line(Point a, Point b, Point c) {
30     return (b-a)*(c-b)==0;
31 }
32 bool ord(Point a, Point p, Point b) {
33     return (p - a)%(p - b)<0;
34 }

```

8.3 Точка нестрого внутри выпуклости

```

1     auto inT = [&] (Point a, Point b, Point c,
2         Point p) {
3         a = a-p; b = b-p; c = c-p;
4         return abs(a*b)+abs(b*c)+abs(c*a) == abs(a*
5             b+b*c+c*a);
6     };
7     auto inP = [&] (Point p) { //a must be in
8         counterclockwise order!
9         int l = 1, r = n - 1;
10        while (l < r - 1) {
11            int m = (l + r) / 2;
12            if ((a[m] - a[0]) < (p - a[0])) {
13                l = m;
14            }
15            else {
16                r = m;
17            }
18        }
19        return inT(a[l], a[0], a[r], p);
20    };

```

9 Разное

9.1 Флаги компиляции

```

-DLOCAL -Wall -Wextra -pedantic -Wshadow -Wformat=2
-Wfloat-equal -Wconversion -Wlogical-op -Wshift-overflow=2
-Wduplicated-cond -Wcast-qual -Wcast-align -D_GLIBCXX_DEBUG
-D_GLIBCXX_DEBUG_PEDANTIC -D_FORTIFY_SOURCE=2
-fsanitize=address -fsanitize=undefined -fno-sanitize-recover
-fstack-protector -std=c++2a

```

9.1.1 Сеточка в vim

<https://codeforces.com/blog/entry/122540>

```
1 i|<esc>25A    |<esc>
2 o+<esc>25A---+<esc>
3 Vky35Pdd
```

9.2 Что сделать на пробном туре

- Убедиться, что работают все IDE. Разобраться, как настраивать в них LOCAL.
- В системе ML — это ML или RE?
- Максимальный размер файла
- Можно посмотреть на время работы серверов позапусков Флойда — Варшалла

9.3 Шаблон

```
1 #include<bits/stdc++.h>
2
3 using namespace std;
4
5 #define int long long
6 #define app push_back
7 #define all(x) x.begin(), x.end()
8 #ifdef LOCAL
9 #define debug(...) [](auto...a){ ((cout << a <<
    ' '), ...) << endl; }(#__VA_ARGS__, ":",
    __VA_ARGS__)
10 #else
11 #define debug(...)
12 #endif
13
14 int32_t main() {
15     cin.tie(0);ios_base::sync_with_stdio(0);
16 }
```