

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
3.1 and-свёртка	3
3.2 or-свёртка	3
3.3 xor-свёртка	3
4 Структуры данных	3
4.1 Дерево Фенвика	3
4.2 Ordered set	3
4.3 Дерево отрезков	3
4.3.1 Примеры использования	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	6
6 Потоки	6
6.1 Алгоритм Диница	6
6.2 Mincost k-flow	6
6.2.1 Строим граф	6
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

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, x, y);
4     swap(x, y);
5     y -= a/b*x;
6     return d;
7 }
8 int inv(int r, int m) {
9     int x, y;
10    gcd(r, m, x, y);
11    return (x+m)%m;
12 }
13 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
6         {a=(a*one*a)%p; b>>=1;}} return res;
7 }
8 bool chprime(int n) //miller-rabin
9 {
10    if(n==2) return true;
11    if(n<=1 || n%2==0) return false;
12    int h=n-1; int d=0; while(h%2==0) {h/=2; ++d;}
13    for(int a:{2, 3, 5, 7, 11, 13, 17, 19, 23, 29,
14        31, 37})
15    {
16        if(a==n) return true;
17        int u=po(a, h, n); bool ok=0;
18        if(u%n==1) continue;
19        for(int c=0; c<d; ++c)
20        {
21            if((u+1)%n==0) {ok=1; break;}
22            u=(u*one*u)%n;
23        }
24        if(!ok) return false;
25    }
26    return true;
27 }
```

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 = (int) 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)
11            continue;
12        vector<T> temp = c;
13        T coef = d / ld;
14        for (int j=m; j<n; j++)
15            c[j] -= coef * b[j-m];
16        if (2 * l <= i) {
17            l = i + 1 - l;
18            b = temp;
19            ld = d;
20            m = 0;
21        }
22    }
23    c.resize(l + 1);
24    c.erase(c.begin());
25    for (T &x : c)
26        x = -x;
```

```
27 return c;
28 }
```

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;
```

```
20 if (in[i] != out[i]) {
21     if (in[i] + 1 == out[i]) odd = i;
22     if (abs(in[i] - out[i]) > 1) return {}; // must
23         hold
24     cnt++;
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 function <void(int,int)> findCutPoints = [&](int u
15     , int p) {
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             findCutPoints(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(u, u);
35     }
36 }
37 int ptr = 0;
38 vector <map <int, int> > colors(m);
39 function <void(int, int)> markComponents = [&](int
40     u, int cur) {
41     used[u] = 1;
42     for (int v : g[u]) {
43         if (!used[v]) {
44             if (newCompWithoutParent[v]) {
45                 ptr++;
46                 markComponents(v, ptr - 1);
47             }
48             else {
49                 markComponents(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(u, -1);
64     }
65 }
```

```

61 }
62 }
63 for (int comp = 0; comp < m; ++comp) {
64     vector<int> cnt(4);
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

3.1 and-свёртка

```

1 vector<int> band(vector<int> a, vector<int> b)
2 {
3     int n=0; while((1<<n)<a.size()) ++n;
4     a.resize(1<<n); b.resize(1<<n);
5     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;}
6     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;}
7     vector<int> c(1<<n, 0);
8     for(int mask=0; mask<(1<<n); ++mask) {c[mask]=a[
            mask]*b[mask]; c[mask]%=p;}
9     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;}
10    return c;
11 }

```

3.2 or-свёртка

```

1 vector<int> bor(vector<int> a, vector<int> b)
2 {
3     int n=0; while((1<<n)<a.size()) ++n;
4     a.resize(1<<n); b.resize(1<<n);
5     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;}
6     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;}
7     vector<int> c(1<<n, 0);
8     for(int mask=0; mask<(1<<n); ++mask) {c[mask]=a[
            mask]*b[mask]; c[mask]%=p;}
9     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;}
10    return c;
11 }

```

3.3 xor-свёртка

```

1 vector<int> bxor(vector<int> a, vector<int> b)
2 {
3     assert(p%2==1); int inv2=(p+1)/2;
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))) {int u=a[mask], v=
            a[mask+(1<<i)]; a[mask+(1<<i)]=(u+v)%p; a[mask]=
            (u-v)%p;}
7     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;}

```

```

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))) {int u=c[mask], v=
        c[mask+(1<<i)]; c[mask+(1<<i)]=(v-u)*inv2%p; c[
        mask]=((u+v)*inv2)%p;}
11    return c;

```

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 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<>,
    rb_tree_tag, tree_order_statistics_node_update
    >;

```

4.3 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
41 - lvl));
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 Data get(int l, int r) { // [l; r)
72     l += n, r += n;
73     for (int shift = h; shift > 0; --shift) {
74         push(l >> shift);
75         push((r - 1) >> shift);
76     }
77     Data leftRes = zd, rightRes = zd;
78     for (; l < r; l /= 2, r /= 2) {
79         if (l & 1) leftRes = ud(leftRes, data[l++]);
80         if (r & 1) rightRes = ud(data[--r], rightRes);
81     }
82     return ud(leftRes, rightRes);
83 }
84
85 // l \in [0; n) && ok(get(l, l), l);
86 // returns last r: ok(get(l, r), r)
87 template<typename C>
88 int lastTrue(int l, C ok) {
89     l += n;
90     for (int shift = h; shift > 0; --shift) push(l
91 >> shift);
92     Data cur = zd;
93     do {
94         l >>= __builtin_ctz(l);
95         Data with1;
96         with1 = ud(cur, data[l]);
97         if (ok(with1, right(l))) {
98             cur = with1;
99             ++l;
100         } else {
101             while (l < n) {
102                 push(l);
103                 Data with2;
104                 with2 = ud(cur, data[2 * l]);

```

```

104         if (ok(with2, right(2 * l))) {
105             cur = with2;
106             l = 2 * l + 1;
107         } else {
108             l = 2 * l;
109         }
110     }
111     return l - n;
112 }
113 } while (l & (l - 1));
114 return n;
115 }
116
117 // r \in [0; n) && ok(get(r, r), r);
118 // returns first l: ok(get(l, r), l)
119 template<typename C>
120 int firstTrue(int r, C ok) {
121     r += n;
122     for (int shift = h; shift > 0; --shift) push((r
123 - 1) >> shift);
124     Data cur = zd;
125     while (r & (r - 1)) {
126         r >>= __builtin_ctz(r);
127         Data with1;
128         with1 = ud(data[--r], cur);
129         if (ok(with1, left(r))) {
130             cur = with1;
131         } else {
132             while (r < n) {
133                 push(r);
134                 Data with2;
135                 with2 = ud(data[2 * r + 1], cur);
136                 if (ok(with2, right(2 * r))) {
137                     cur = with2;
138                     r = 2 * r;
139                 } else {
140                     r = 2 * r + 1;
141                 }
142             }
143             return r - n + 1;
144         }
145     }
146     return 0;
147 };

```

4.3.1 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; });

```

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] -
      1];
6     p[i] += s[i] == s[p[i]];
7 }
8 return p;
9 }

```

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 >=
10            0 && s[i + man[i] + 1] == s[i - man[i] - 1]) {
11             man[i]++;
12         }
13         if (i + man[i] > r) {
14             l = i - man[i];
15             r = i + man[i];
16         }
17     }
18     return man;
19 }
20 // abacaba : (0 1 0 3 0 1 0)
21 // abbaa : (0 0 0 0 0)
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

Китайский суфмассив

```

1 struct SuffixArray {
2     vector<int> sa, lcp;
3     SuffixArray (string &s, int lim=256) {
4         int n = (int)s.size() + 1, k = 0, a, b;
5         vector<int> x(s.begin(), s.end() + 1), y(n),
6             ws(max(n, lim)), rank(n);
7         sa = lcp = y, iota(sa.begin(), sa.end(), 0);

```

```

7         for (int j = 0, p = 0; p < n; j = max(1ll, j *
8             2), lim = p) {
9             p = j, iota(y.begin(), y.end(), n - j);
10            for (int i = 0; i < n; i++) if (sa[i] >= j) y
11                [p++] = sa[i] - j;
12            fill(ws.begin(), ws.end(), 0);
13            for (int i = 0; i < n; i++) ws[x[i]]++;
14            for (int i = 1; i < lim; i++) ws[i] += ws[i -
15                1];
16            for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[
17                i];
18            swap(x, y), p = 1, x[sa[0]] = 0;
19            for (int i = 1; i < n; i++) a = sa[i - 1], b
20                = sa[i], x[b] = (y[a] == y[b] && y[a + j] == y[
21                    b + j]) ? p - 1 : p++;
22        }
23    }
24    struct Rmq {
25        const int INF = 1e9;
26        int n;
27        vector<int> rmq;
28        Rmq() {}
29        void build(const vector<int> &x) {
30            assert(x.size() == n);
31            for (int i = 0; i < n; ++i) rmq[n + i] = x[i];
32            for (int i = n - 1; i > 0; --i) rmq[i] = min(
33                rmq[2 * i], rmq[2 * i + 1]);
34        }
35        Rmq(int n) : n(n), rmq(2 * n, INF) {}
36        void put(int i, int x) {
37            rmq[i + n] = min(rmq[i + n], x);
38            for (i = (i + n) / 2; i > 0; i /= 2) {
39                rmq[i] = min(rmq[i * 2], rmq[i * 2 + 1]);
40            }
41        }
42        int getMin(int l, int r) { // [l;r)
43            assert(l < r);
44            int res = INF;
45            for (l += n, r += n; l < r; l /= 2, r /= 2) {
46                if (l & 1) res = min(res, rmq[l++]);
47                if (r & 1) res = min(res, rmq[--r]);
48            }
49            return res;
50        }
51    };
52    struct Lc {
53        vector<int> pos;
54        Rmq rmq;
55        Lc(string s) : rmq(s.size()) {
56            SuffixArray sa(s);
57            auto ss = sa.sa;
58            ss.erase(ss.begin());
59        }
60        auto lcp = sa.lcp;
61        lcp.erase(lcp.begin());
62        lcp.erase(lcp.begin());
63        pos.resize(s.size());
64        assert(s.size() == ss.size());
65        for (int i = 0; i < ss.size(); ++i) {
66            pos[ss[i]] = i;
67        }
68        int n = s.size();
69        assert(lcp.size() == n - 1);
70        rmq.build(lcp);
71    }
72    int getLcp(int i, int j) {
73        i = pos[i]; j = pos[j];
74        if (j < i) {
75            swap(i, j);
76        }
77        if (i == j) {
78            return 1e18;
79        }

```

```

80     }
81     else {
82         return rmq.getMin(i, j);
83     }
84 }
85 };

```

5.5 Алгоритм Ахо — Корасик

```

1 struct node{
2     node *next[26] = {}, *link[26] = {};
3     node *suf = nullptr;
4     vector<int> term;
5     int visited = 0;
6     node() {}
7     node *get_next(char c) {
8         if (next[c - 'a'] == nullptr) next[c - 'a'] =
9             new node();
10        return next[c - 'a'];
11    }
12 };
13 node *root = new node();
14 for (int i = 0; i < s.size(); ++i) {
15     node *cur = root;
16     for (char c : s[i]) cur = cur->get_next(c);
17     cur->term.push_back(i);
18 }
19 vector<node *> bfs_order;
20 queue<node *> bfs;
21 root->suf = root;
22 for (char c = 'a'; c <= 'z'; ++c) root->link[c - 'a']
23     = (root->next[c - 'a'] ? root->next[c - 'a']
24       : root);
25 bfs.push(root);
26 while (!bfs.empty()) {
27     node *cur = bfs.front();
28     bfs_order.push_back(cur);
29     bfs.pop();
30     for (char c = 'a'; c <= 'z'; ++c) {
31         node *nxt = cur->next[c - 'a'];
32         if (!nxt) continue;
33         nxt->suf = (cur == root ? cur : cur->suf->link[
34             c - 'a']);
35         for (char c = 'a'; c <= 'z'; ++c) nxt->link[c -
36             'a'] = (nxt->next[c - 'a'] ? nxt->next[c - 'a']
37                   : nxt->suf->link[c - 'a']);
38         bfs.push(nxt);
39     }
40 }
41 node *cur = root;
42 for (char c : t) {
43     cur = cur->link[c - 'a'];
44     cur->visited++;
45 }
46 vector<int> count(n);
47 for (int i = bfs_order.size() - 1; i >= 0; --i) {
48     node *cur = bfs_order[i];
49     for (int idx : cur->term) count[idx] = cur->
50         visited;
51     cur->suf->visited += cur->visited;
52 }

```

6 Поток

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     return flow;
80 }
81 }
82 } 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(
    next), capacity(capacity), cost(cost) {}
7
8   int rem() const { return capacity - flow; }
9
10  int operator+=(int f) { return flow += f; }
11
12  int operator-=(int f) { return flow -= f; }
13 };
14 auto addEdge = [&](auto from, auto next, auto
    capacity, int cost) {
15     g[from].push_back(e.size());
16     e.emplace_back(next, capacity, cost);
17     g[next].push_back(e.size());
18     e.emplace_back(from, 0, -cost);
19 };

```

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[
u] + edge.cost) {
10                     phi[edge.next] = phi[u] + edge.cost;
11                     changed = true;
12                 }
13             }
14         }
15         if (!changed) break;
16     }
17 };
18 fordBellman(s, t);

```

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     for (int i = 1; i < n; ++i) {
10         int cur = find(cnt.begin(), cnt.end(), false) -
            cnt.begin();
11         for (int j = 0; j < n; ++j) {
12             if (!cnt[j] && dist[j] < dist[cur]) cur = j;
13         }
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.
next];
19             if (dist[edge.next] > dist[cur] + weight) {
20                 dist[edge.next] = dist[cur] + weight;
21                 from[edge.next] = cur;
22             }
23         }
24     }
25     if (dist[t] == (ll) 1e18) return -1LL;
26     ll cost = 0;
27     for (int p = t; p != s; p = from[p]) {
28         for (auto index : g[from[p]]) {
29             auto &edge = e[index];
30             ll weight = edge.cost + phi[from[p]] - phi[
edge.next];

```

```

31         if (edge.rem() > 0 && edge.next == p && dist[
edge.next] == dist[from[p]] + weight) {
32             edge += 1;
33             e[index ^ 1] -= 1;
34             cost += edge.cost;
35             break;
36         }
37     }
38 }
39 for (int i = 0; i < n; ++i) {
40     phi[i] += dist[i];
41 }
42 return cost;
43 };
44 ll cost = 0;
45 for (int flow = 0; flow < k; ++flow) {
46     ll a = dijkstra(s, t);
47     if (a == -1) {
48         cout << "-1\n";
49         return;
50     }
51     cost += a;
52 }

```

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)
    return a; if(b%2==0){int u=po(a,b/2);return (u
    *1LL*u)%p;} else {int u=po(a,b-1);return (a*1LL
    *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         return v1;
85     }
86     vector<int> operator -(vector<int> v1, vector<int>
87 v2)
88     {
89         int sz=max(v1.size(), v2.size()); while(v1.size()<
90 sz) v1.push_back(0); while(v2.size()<sz) v2.
91 push_back(0);
92         for(int i=0; i<sz; ++i) {v1[i]-=v2[i]; if(v1[i]<0)
93 v1[i]+=p; else if(v1[i]>=p) v1[i]-=p;} return
94 v1;
95     }
96     vector<int> trmi(vector<int> v)
97     {
98         for(int i=1; i<v.size(); i+=2) {if(v[i]>0) v[i]=p-v
99 [i]; else v[i]=(-v[i]);}
100     }

```

```

86     return v;
87 }
88 vector<int> deriv(vector<int> v)
89 {
90     if(v.empty()) return {};
91     vector<int> ans(v.size()-1);
92     for(int i=1; i<v.size(); ++i) ans[i-1]=(v[i]*1LL*i)
93 %p;
94     return ans;
95 }
96 vector<int> integ(vector<int> v)
97 {
98     vector<int> ans(v.size()+1); ans[0]=0;
99     for(int i=1; i<v.size(); ++i) ans[i-1]=(v[i]*1LL*i)
100 %p;
101     return ans;
102 }
103 vector<int> mul(vector<vector<int> > v)
104 {
105     if(v.size()==1) return v[0];
106     vector<vector<int> > v1, v2; for(int i=0; i<v.size()
107 /2; ++i) v1.push_back(v[i]); for(int i=v.size()
108 /2; i<v.size(); ++i) v2.push_back(v[i]);
109     return muls.convolution(mul(v1), mul(v2));
110 }
111 vector<int> inv1(vector<int> v, int n)
112 {
113     assert(v[0]!=0);
114     int sz=1; v=form(v, n); vector<int> a={inv(v[0])};
115     while(sz<n)
116     {
117         vector<int> vsz; for(int i=0; i<min(n, 2*sz); ++i)
118 vsz.push_back(v[i]);
119         vector<int> b=((vector<int>) {1})-muls.
120 convolution(a, vsz);
121         for(int i=0; i<sz; ++i) assert(b[i]==0);
122         b.erase(b.begin(), b.begin()+sz);
123         vector<int> c=muls.convolution(b, a);
124         for(int i=0; i<sz; ++i) a.push_back(c[i]);
125         sz*=2;
126     }
127     return form(a, n);
128 }

```

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();
6     if (n == 1)
7         return;
8
9     vector<cd> a0(n / 2), a1(n / 2);
10    for (int i = 0; 2 * i < n; i++) {
11        a0[i] = a[2*i];
12        a1[i] = a[2*i+1];
13    }
14    fft(a0, invert);
15    fft(a1, invert);
16
17    double ang = 2 * PI / n * (invert ? -1 : 1);
18    cd w(1), wn(cos(ang), sin(ang));
19    for (int i = 0; 2 * i < n; i++) {
20        a[i] = a0[i] + w * a1[i];
21        a[i + n/2] = a0[i] - w * a1[i];
22        if (invert) {
23            a[i] /= 2;
24            a[i + n/2] /= 2;
25        }
26        w *= wn;
27    }
28 }
29 vector<int> multiply(vector<int> const& a, vector<
30 int> const& b) {
31     vector<cd> fa(a.begin(), a.end()), fb(b.begin(),
32 b.end());
33     int n = 1;

```



```
32 while (n < a.size() + b.size())
33     n <= 1;
34 fa.resize(n);
35 fb.resize(n);
36
37 fft(fa, false);
38 fft(fb, false);
39 for (int i = 0; i < n; i++)
40     fa[i] *= fb[i];
41 fft(fa, true);
42
43 vector<int> result(n);
44 for (int i = 0; i < n; i++)
45     result[i] = round(fa[i].real());
46 while(!result.empty() && !result.back()) result.
47     pop_back();
48 return result;
49 }
```