

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 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	4
5.3 Алгоритм Манакера	4
5.4 Суфмассив	5
5.5 Алгоритм Ахо — Корасик	5
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	6
6.2.3 Ищем кратчайший путь Дейкстрой с потенциалами	6
6.2.4 Восстанавливаем ответ	7
7 FFT & co	7
7.1 NTT & co	7
7.2 старое доброе FFT	8
8 Геометрия	8
8.1 Касательные	8
8.2 Примитивы	9
8.3 Точка нестрого внутри выпуклости	9
9 Разное	9
9.1 Флаги компиляции	9
9.1.1 Сетка в vim	9
9.2 Что сделать на пробном туре	9
9.3 Шаблон	9

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
6         {a=(a*one*a)%p;b>>=1;}} return res;
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,
13        31, 37})
14    {
15        if(a==n) return true;
16        int u=po(a, h, n); bool ok=0;
17        if(u%n==1) continue;
18        for(int c=0; c<d; ++c)
19        {
20            if((u+1)%n==0) {ok=1; break;}
21            u=(u*one*u)%n;
22        }
23        if(!ok) return false;
24    }
25    return true;
26 }
```

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{next,
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

```

```

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);

```

```

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

3.2 or-свёртка

3.3 xor-свёртка

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) {}

```

```

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
        /= 2) {
62         if (lf & 1) apply(lf++, x);
63         if (rg & 1) apply(--rg, x);
64     }
65     for (int shift = 1; shift <= h; ++shift) {
66         update(l >> shift);
67         update((r - 1) >> shift);
68     }
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
        >> shift);
91     Data cur = zd;
92     do {
93         l >>= __builtin_ctz(l);
94         Data with1;
95         with1 = ud(cur, data[l]);

```

```

96     if (ok(with1, right(1))) {
97         cur = with1;
98         ++l;
99     } else {
100         while (l < n) {
101             push(1);
102             Data with2;
103             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 };

```

4.2.1 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
2     of s and s[i:]
3     int n = (int) s.length();
4     vector<int> z (n);
5     for (int i=1, l=0, r=0; i<n; ++i) {
6         if (i <= r)
7             z[i] = min (r-i+1, z[i-l]);
8         while (i+z[i] < n && s[z[i]] == s[i+z[i]])
9             ++z[i];
10        if (i+z[i]-1 > r)
11            l = i, r = i+z[i]-1;
12    }
13    return z;

```

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
23 vector<int> manacher_even(const string &s) {
24     assert(s.size());
25     string t;
26     for (int i = 0; i + 1 < s.size(); ++i) {
27         t += s[i];
28         t += '#';
29     }
30     t += s.back();
31     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);
8         for (int j = 0, p = 0; p < n; j = max(1ll, j *
9             2), lim = p) {
10             p = j, iota(y.begin(), y.end(), n - j);
11             for (int i = 0; i < n; i++) if (sa[i] >= j) y
12                 [p++] = sa[i] - j;
13             fill(ws.begin(), ws.end(), 0);
14             for (int i = 0; i < n; i++) ws[x[i]]++;
15             for (int i = 1; i < lim; i++) ws[i] += ws[i -
16                 1];
17             for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[
18                 i];
19             swap(x, y), p = 1, x[sa[0]] = 0;
20             for (int i = 1; i < n; i++) a = sa[i - 1], b
21                 = sa[i], x[b] = (y[a] == y[b] && y[a + j] == y[
22                     b + j]) ? p - 1 : p++;
23             for (int i = 1; i < n; i++) rank[sa[i]] = i;
24             for (int i = 0, j; i < n - 1; lcp[rank[i++]] = k)
25                 for (k && k--, j = sa[rank[i] - 1];
26                     s[i + k] == s[j + k]; k++);
27         }
28 };
29 struct Rmq {
30     const int INF = 1e9;
31     int n;
32     vector<int> rmq;
33     Rmq() {}
34     void build(const vector<int> &x) {
35         assert(x.size() == n);
36         for (int i = 0; i < n; ++i) rmq[n + i] = x[i];
37         for (int i = n - 1; i > 0; --i) rmq[i] = min(
38             rmq[2 * i], rmq[2 * i + 1]);
39     }
40     Rmq(int n) : n(n), rmq(2 * n, INF) {}
41     void put(int i, int x) {
42         rmq[i + n] = min(rmq[i + n], x);
43         for (i = (i + n) / 2; i > 0; i /= 2) {
44             rmq[i] = min(rmq[i * 2], rmq[i * 2 + 1]);
45         }
46     }
47     int getMin(int l, int r) { // [l;r)
48         assert(l < r);
49         int res = INF;
50         for (l += n, r += n; l < r; l /= 2, r /= 2) {
51             if (l & 1) res = min(res, rmq[l++]);
52             if (r & 1) res = min(res, rmq[--r]);
53         }
54         return res;
55     }
56 };
57 struct Lc {
58     vector<int> pos;
59     Rmq rmq;
60     Lc(string s) : rmq(s.size()) {
61         SuffixArray sa(s);
62         auto ss = sa.sa;
63         ss.erase(ss.begin());

```

```

60 auto lcp = sa.lcp;
61 lcp.erase(lcp.begin());
62 lcp.erase(lcp.begin());
63
64 pos.resize(s.size());
65 assert(s.size() == ss.size());
66 for (int i = 0; i < ss.size(); ++i) {
67     pos[ss[i]] = i;
68 }
69 int n = s.size();
70 assert(lcp.size() == n - 1);
71 rmq.build(lcp);
72 }
73 int getLcp(int i, int j) {
74     i = pos[i]; j = pos[j];
75     if (j < i) {
76         swap(i, j);
77     }
78     if (i == j) {
79         return 1e18;
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    node *root = new node();
13    for (int i = 0; i < s.size(); ++i) {
14        node *cur = root;
15        for (char c : s[i]) cur = cur->get_next(c);
16        cur->term.push_back(i);
17    }
18    vector<node *> bfs_order;
19    queue<node *> bfs;
20    root->suf = root;
21    for (char c = 'a'; c <= 'z'; ++c) root->link[c - 'a']
22        = (root->next[c - 'a'] ? root->next[c - 'a']
23            : root);
24    bfs.push(root);
25    while (!bfs.empty()) {
26        node *cur = bfs.front();
27        bfs_order.push_back(cur);
28        bfs.pop();
29        for (char c = 'a'; c <= 'z'; ++c) {
30            node *nxt = cur->next[c - 'a'];
31            if (!nxt) continue;
32            nxt->suf = (cur == root ? cur : cur->suf->link[
33                c - 'a']);
34            for (char c = 'a'; c <= 'z'; ++c) nxt->link[c -
35                'a'] = (nxt->next[c - 'a'] ? nxt->next[c - 'a']
36                    : nxt->suf->link[c - 'a']);
37            bfs.push(nxt);
38        }
39    }
40    node *cur = root;
41    for (char c : t) {
42        cur = cur->link[c - 'a'];
43        cur->visited++;
44    }
45    vector<int> count(n);
46    for (int i = bfs_order.size() - 1; i >= 0; --i) {
47        node *cur = bfs_order[i];

```

```

43   for (int idx : cur->term) count[idx] = cur->
        visited;
44   cur->suf->visited += cur->visited;
45 }

```

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]
== -1){
39                 dp[ed.to] = dp[v] + 1;
40                 st.pb(ed.to);
41             }
42         }
43     }
44     return dp[finish] != -1;
45 }
46
47 int dfs(int v, int flow){
48     if (v == finish) return flow;
49     for (; ptr[v] < g[v].size(); ptr[v]++){
50         int to = g[v][ptr[v]];
51         edge ed = e[to];
52         if (ed.cap - ed.flow >= minFlow && dp[ed.to] ==
dp[v] + 1){
53             int add = dfs(ed.to, min(flow, ed.cap - ed.
flow));
54             if (add){
55                 e[to].flow += add;
56                 e[to ^ 1].flow -= add;
57                 return add;
58             }
59         }
60     }
61     return 0;
62 }
63
64 int dinic(int start, int finish){
65     Dinic::start = start;
66     Dinic::finish = finish;
67     int flow = 0;

```

```

68     for (minFlow = (1 << 30); minFlow; minFlow >= 1)
        {
69         while(bfs()){
70             for (int i = 0; i < N; i++) ptr[i] = 0;
71             while(int now = dfs(start, (int)2e9 + 7))
                flow += now;
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(
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      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.
next];
19     if (dist[edge.next] > dist[cur] + weight) {
20         se.erase({dist[edge.next], edge.next});
21         dist[edge.next] = dist[cur] + weight;
22         se.insert({dist[edge.next], edge.next});
23         from[edge.next] = cur;
24     }
25 }
26 }
27 if (dist[t] == (ll) 1e18) return -1LL;
28 ll cost = 0;
29 for (int p = t; p != s; p = from[p]) {
30     for (auto index : g[from[p]]) {
31         auto &edge = e[index];
32         ll weight = edge.cost + phi[from[p]] - phi[
edge.next];
33         if (edge.rem() > 0 && edge.next == p && dist[
edge.next] == dist[from[p]] + weight) {
34             edge += 1;
35             e[index ^ 1] -= 1;
36             cost += edge.cost;
37             break;
38         }
39     }
40 }
41 for (int i = 0; i < n; ++i) {
42     phi[i] += dist[i];
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)
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) +
1) << h); ++j) {
31                    const int t = (g[i] * 1LL* x[j | l]) % M;
32                    x[j | l] = x[j] - t;
33                    if (x[j|l] < 0) x[j | l] += M;
34                    x[j]+=t;
35                    if (x[j] >= M) x[j] -= M;
36                }
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,
const vector<int> &b) const {
45        if(a.empty() || b.empty()) return {};
46        const int na = a.size(), nb = b.size();
47        int n, invN = 1;
48        for (n = 1; n < na + nb - 1; n <= 1) invN = ((
invN & 1) ? (invN + M) : invN) >> 1;
49        vector<int> x(n, 0), y(n, 0);
50        std::copy(a.begin(), a.end(), x.begin());
51        std::copy(b.begin(), b.end(), y.begin());
52        fft(x);
53        fft(y);
54        for (int i = 0; i < n; ++i) x[i] = (((
static_cast<long long>(x[i]) * y[i]) % M) *
invN) % M;
55        std::reverse(x.begin() + 1, x.end());
56        fft(x);
57        x.resize(na + nb - 1);
58        return x;
59    }
60 };
61 Fft<998244353,23,31> muls;
62 vector<int> form(vector<int> v,int n)
63 {
64     while(v.size()<n) v.push_back(0);
65     while(v.size()>n) v.pop_back();
66     return v;
67 }
68 vector<int> operator *(vector<int> v1,vector<int>
v2)
69 {
70     return muls.convolution(v1,v2);
71 }

```

```

72 vector<int> operator +(vector<int> v1,vector<int>
    v2)
73 {
74     while(v2.size()<v1.size()) v2.push_back(0); while
        (v1.size()<v2.size()) v1.push_back(0);
75     for(int i=0;i<v1.size();++i) {v1[i]+=v2[i];if(v1[
        i]>=p) v1[i]-=p; else if(v1[i]<0) v1[i]+=p;}
76     return v1;
77 }
78 vector<int> operator -(vector<int> v1,vector<int>
    v2)
79 {
80     int sz=max(v1.size(),v2.size());while(v1.size()<
        sz) v1.push_back(0); while(v2.size()<sz) v2.
        push_back(0);
81     for(int i=0;i<sz;++i) {v1[i]-=v2[i];if(v1[i]<0)
        v1[i]+=p; else if(v1[i]>=p) v1[i]-=p;} return
        v1;
82 }
83 vector<int> trmi(vector<int> v)
84 {
85     for(int i=1;i<v.size();i+=2) {if(v[i]>0) v[i]=p-v
        [i]; else v[i]=(-v[i]);}
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
        %p;
93     return ans;
94 }
95 vector<int> integ(vector<int> v)
96 {
97     vector<int> ans(v.size()+1);ans[0]=0;
98     for(int i=1;i<v.size();++i) ans[i-1]=(v[i]*1LL*i
        %p;
99     return ans;
100 }
101 vector<int> mul(vector<vector<int> > v)
102 {
103     if(v.size()==1) return v[0];
104     vector<vector<int> > v1,v2;for(int i=0;i<v.size()
        /2;++i) v1.push_back(v[i]); for(int i=v.size()
        /2;i<v.size();++i) v2.push_back(v[i]);
105     return muls.convolution(mul(v1),mul(v2));
106 }
107 vector<int> inv1(vector<int> v,int n)
108 {
109     assert(v[0]!=0);
110     int sz=1;v=form(v,n);vector<int> a={inv(v[0])};
111     while(sz<n)
112     {
113         vector<int> vsz;for(int i=0;i<min(n,2*sz);++i)
            vsz.push_back(v[i]);
114         vector<int> b=((vector<int>) {1})-muls.
            convolution(a,vsz);
115         for(int i=0;i<sz;++i) assert(b[i]==0);
116         b.erase(b.begin(),b.begin()+sz);
117         vector<int> c=muls.convolution(b,a);
118         for(int i=0;i<sz;++i) a.push_back(c[i]);
119         sz*=2;
120     }
121     return form(a,n);
122 }

```

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
7     for (int i = 1, j = 0; i < n; i++) {
8         int bit = n >> 1;
9         for (; j & bit; bit >>= 1)
10             j ^= bit;

```

```

11         j ^= bit;
12
13         if (i < j)
14             swap(a[i], a[j]);
15     }
16
17     for (int len = 2; len <= n; len <= 1) {
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<
    int> const& b) {
37     vector<cd> fa(a.begin(), a.end()), fb(b.begin(),
        b.end());
38     int n = 1;
39     while (n < a.size() + b.size())
40         n <= 1;
41     fa.resize(n);
42     fb.resize(n);
43
44     fft(fa, false);
45     fft(fb, false);
46     for (int i = 0; i < n; i++)
47         fa[i] *= fb[i];
48     fft(fa, true);
49
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.
        pop_back();
54     return result;
55 }

```

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         i = (i % n + n) % n;
6         j = (j % n + n) % n;
7         array<int, 3> ind{i, j, k};
8         sort(all(ind), cmp);
9         k = ind[2];
10    }
11    return k;
12 };
13 auto upper = [&] (Point p) { //last vertex in
    counterclockwise order about p
14     auto cmp = [&] (int i, int j) {return (a[i]
        - p) < (a[j] - p); };
15     return max(cmp);
16 };
17 auto lower = [&] (Point p) { //first vertex in
    counterclockwise order about p
18     auto cmp = [&] (int i, int j) {return (a[i]
        - p) > (a[j] - p); };
19     return max(cmp);
20 };

```



```

21 auto uppertinf = [&](Point p) { //upper tangent
    line parallel to vector p
22     swap(p.x, p.y);
23     p.x = -p.x;
24     auto cmp = [&](int i, int j) { return a[i]
    % p < a[j] % p; };
25     return max(cmp);
26 };
27 auto lowertinf = [&](Point p) { //lower tangent
    line parallel to vector p
28     swap(p.x, p.y);
29     p.x = -p.x;
30     auto cmp = [&](int i, int j) { return a[i]
    % p > a[j] % p; };
31     return max(cmp);
32 };

```

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

```

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 }

```