

Содержание

1	Теория чисел	1
1.1	КТО	1
1.2	Алгоритм Миллера — Рабина	1
2	Графы	1
2.1	<i>SCC</i> и 2- <i>SAT</i>	1
2.2	Эйлеров цикл	2
3	xor, and, or-свёртки	2
3.1	and-свёртка	2
3.2	or-свёртка	2
3.3	xor-свёртка	2
4	Структуры данных	2
4.1	Дерево Фенвика	2
4.2	Ordered set	2
4.3	Дерево отрезков	3
4.3.1	Примеры:	4
5	Строковые алгоритмы	4
5.1	Префикс-функция	4
5.2	Z-функция	4
5.3	Алгоритм Манакера	4
5.4	Суфмассив	4
6	Потоки	5
6.1	Алгоритм Диница	5
6.2	Mincost k-flow	5
6.2.1	Строим граф	5
6.2.2	Запускаем Форда — Беллмана	6
6.2.3	Ищем кратчайший путь Дейкстрой с потенциалами	6
6.2.4	Восстанавливаем ответ	6
7	FFT & co	6
7.1	NTT & co	6

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
        {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 }
```

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])
    dfs(dfs, i);
18 vector<int> ind(g.size());
19 iota(ind.begin(), ind.end(), 0);
```

```

20 sort(all(ind), [&](int i, int j){return tout[i] >
    tout[j];});
21 vector<int> scc(g.size(), -1);
22 auto go = [&](auto go, int cur, int color) -> void
    {
23     if (scc[cur] != -1) return;
24     scc[cur] = color;
25     for (int nxt : r[cur]) {
26         go(go, nxt, color);
27     }
28 };
29 int color = 0;
30 for (int i : ind) {
31     if (scc[i] == -1) go(go, i, color++);
32 }
33 for (int i = 0; i < g.size() / 2; ++i) {
34     if (scc[2 * i] == scc[2 * i + 1]) "IMPOSSIBLE"
35     if (scc[2 * i] < scc[2 * i + 1]) {
36         // !i => i, assign i = true
37     } else {
38         // i => !i, assign i = false
39     }
40 }

```

2.2 Эйлеров цикл

```

1 vector<vector<pair<int, int>>> g(n); // pair{nxt,
    idx}
2 vector<pair<int, int>> e(p.size());
3 // build graph
4 vector<int> in(n), out(n);
5 for (auto [u, v] : e) in[v]++, out[u]++;
6 vector<int> used(m), it(n), cycle;
7 auto dfs = [&](auto dfs, int cur) -> void {
8     while (true) {
9         while (it[cur] < g[cur].size() && used[g[cur][
            it[cur]].second]) it[cur]++;
10        if (it[cur] == g[cur].size()) return;
11        auto [nxt, idx] = g[cur][it[cur]];
12        used[idx] = true;
13        dfs(dfs, nxt);
14        cycle.push_back(idx);
15    }
16 };
17 int cnt = 0, odd = -1;
18 for (int i = 0; i < n; ++i){
19     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
            hold
23         cnt++;
24     }
25 }
26 if (cnt != 0 && cnt != 2) return {}; // must hold
27 // for undirected find odd vertex (and count that #
    of odd is 0 or 2)
28 dfs(dfs, odd);
29 reverse(cycle.begin(), cycle.end());
30 if (cycle.size() != m) return {};

```

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
            - 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(l))) {
97             cur = with1;
98             ++l;
99         } else {
100             while (l < n) {
101                 push(l);
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
        - 1) >> shift);
123     Data cur = zd;
124     while (r & (r - 1)) {
125         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 };

```

4.3.1 Примеры:

- 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 присоединение

```

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] -
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]) {

```

```

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 Суффикс-массив

Китайский суффикс-массив

```

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         }
24         for (int i = 1; i < n; i++) rank[sa[i]] = i;
25         for (int i = 0, j; i < n - 1; lcp[rank[i++]] = k)
26             for (k && k--, j = sa[rank[i] - 1];
27                  s[i + k] == s[j + k]; k++);
28     }
29 };
30 struct Rmq {
31     const int INF = 1e9;
32     int n;
33     vector<int> rmq;
34     Rmq() {}
35     void build(const vector<int> &x) {
36         assert(x.size() == n);
37         for (int i = 0; i < n; ++i) rmq[n + i] = x[i];
38         for (int i = n - 1; i > 0; --i) rmq[i] = min(
39           rmq[2 * i], rmq[2 * i + 1]);
40     }
41     Rmq(int n) : n(n), rmq(2 * n, INF) {}
42
43     void put(int i, int x) {
44         rmq[i + n] = min(rmq[i + n], x);
45         for (i = (i + n) / 2; i > 0; i /= 2) {
46             rmq[i] = min(rmq[i * 2], rmq[i * 2 + 1]);

```

```

39     }
40 }
41 int getMin(int l, int r) { //[l;r)
42     assert(l < r);
43     int res = INF;
44     for (l += n, r += n; l < r; l /= 2, r /= 2) {
45         if (l & 1) res = min(res, rmq[l++]);
46         if (r & 1) res = min(res, rmq[--r]);
47     }
48     return res;
49 }
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
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 };

```

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 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] ==
53             dp[v] + 1){
54             int add = dfs(ed.to, min(flow, ed.cap - ed.
55                 flow));
56             if (add){
57                 e[to].flow += add;
58                 e[to ^ 1].flow -= add;
59                 return add;
60             }
61         }
62     }
63     return 0;
64 }
65 int dinic(int start, int finish){
66     Dinic::start = start;
67     Dinic::finish = finish;
68     int flow = 0;
69     for (minFlow = (1 << 30); minFlow; minFlow >>= 1)
70     {
71         while(bfs()){
72             for (int i = 0; i < N; i++) ptr[i] = 0;
73             while(int now = dfs(start, (int)2e9 + 7))
74                 flow += now;
75         }
76     }
77     return flow;
78 }
79 }
80 } 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);
```

6.2.3 Ищем кратчайший путь Дейкстрой 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) -
11             cnt.begin();
12         for (int j = 0; j < n; ++j) {
13             if (!cnt[j] && dist[j] < dist[cur]) cur = j;
14         }
15         cnt[cur] = true;
16         for (int index : g[cur]) {
17             auto &edge = e[index];
18             if (edge.rem() == 0) continue;
19             ll weight = edge.cost + phi[cur] - phi[edge.
20 next];
21             if (dist[edge.next] > dist[cur] + weight) {
22                 dist[edge.next] = dist[cur] + weight;
23                 from[edge.next] = cur;
24             }
25         }
26         if (dist[t] == (ll) 1e18) return -1LL;
27         ll cost = 0;
28         for (int p = t; p != s; p = from[p]) {
29             for (auto index : g[from[p]]) {
30                 auto &edge = e[index];
31                 ll weight = edge.cost + phi[from[p]] - phi[
32 edge.next];
33                 if (edge.rem() > 0 && edge.next == p && dist[
34 edge.next] == dist[from[p]] + weight) {
35                     edge += 1;
36                     e[index ^ 1] -= 1;
37                     cost += edge.cost;
38                     break;
39                 }
40             }
41         }
42         for (int i = 0; i < n; ++i) {
43             phi[i] += dist[i];
44         }
45         return cost;
46     };
47     ll cost = 0;
48     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 typedef long long ll;
2 const int p=998244353;
3 int po(int a,int b) {if(b==0) return 1; if(b==1)
4     return a; if(b%2==0) {int u=po(a,b/2);return (u
5     *1LL*u)%p;} else {int u=po(a,b-1);return (a*1LL
6     *u)%p;}}
7 int inv(int x) {return po(x,p-2);}
8 template<int M, int K, int G> struct Fft {
9     // 1, 1/4, 1/8, 3/8, 1/16, 5/16, 3/16, 7/16, ...
10    int g[1 << (K - 1)];
11    Fft() : g() { //if t1 constexpr...
12        static_assert(K >= 2, "Fft: K >= 2 must hold");
13        g[0] = 1;
14        for (int l = 1 << (K - 2); l >= 2; l >= 1) {
15            g[l >> 1] = (static_cast<long long>(g[l]) * g
16            [1]) % M;
17        }
18        assert((static_cast<long long>(g[1]) * g[1]) %
19        M == M - 1);
20        for (int l = 2; l <= 1 << (K - 2); l <= 1) {
21            for (int i = 1; i < l; ++i) {
22                g[l + i] = (static_cast<long long>(g[l]) *
23                g[i]) % M;
24            }
25        }
26    }
27    void fft(vector<int> &x) const {
28        const int n = x.size();
29        assert(!(n & (n - 1)) && n <= 1 << K);
30        for (int h = __builtin_ctz(n); h--;) {
31            const int l = 1 << h;
32            for (int i = 0; i < n >> 1 >> h; ++i) {
33                for (int j = i << 1 << h; j < ((i << 1) +
34                1) << h; ++j) {
35                    const int t = (static_cast<long long>(g[i
36                    ]) * x[j | l]) % M;
37                    if ((x[j | l] = x[j] - t) < 0) x[j | l]
38                    += M;
39                    if ((x[j] += t) >= M) x[j] -= M;
```

```

32     }
33 }
34 }
35 for (int i = 0, j = 0; i < n; ++i) {
36     if (i < j) std::swap(x[i], x[j]);
37     for (int l = n; (l >= 1) && !((j ^= 1) & 1);
38         ) {}
39 }
40 vector<int> convolution(const vector<int> &a,
41     const vector<int> &b) const {
42     if(a.empty() || b.empty()) return {};
43     const int na = a.size(), nb = b.size();
44     int n, invN = 1;
45     for (n = 1; n < na + nb - 1; n <= 1) invN = ((
46         invN & 1) ? (invN + M) : invN) >> 1;
47     vector<int> x(n, 0), y(n, 0);
48     std::copy(a.begin(), a.end(), x.begin());
49     std::copy(b.begin(), b.end(), y.begin());
50     fft(x);
51     fft(y);
52     for (int i = 0; i < n; ++i) x[i] = (((
53         static_cast<long long>(x[i]) * y[i]) % M) *
54         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>
69     v2)
70 {
71     return muls.convolution(v1, v2);
72 }
73 vector<int> operator +(vector<int> v1, vector<int>
74     v2)
75 {
76     while(v2.size()<v1.size()) v2.push_back(0); while
77     (v1.size()<v2.size()) v1.push_back(0);
78     for(int i=0; i<v1.size(); ++i) {v1[i]+=v2[i]; if(v1[
79     i]>=p) v1[i]-=p; else if(v1[i]<0) v1[i]+=p;}
80     return v1;
81 }
82 vector<int> operator -(vector<int> v1, vector<int>
83     v2)
84 {
85     int sz=max(v1.size(), v2.size()); while(v1.size()<
86     sz) v1.push_back(0); while(v2.size()<sz) v2.
87     push_back(0);
88     for(int i=0; i<sz; ++i) {v1[i]-=v2[i]; if(v1[i]<0)
89     v1[i]+=p; else if(v1[i]>=p) v1[i]-=p;} return
90     v1;
91 }
92 vector<int> trmi(vector<int> v)
93 {
94     for(int i=1; i<v.size(); i+=2) {if(v[i]>0) v[i]=p-v
95     [i]; else v[i]=(-v[i]);}
96     return v;
97 }
98 vector<int> deriv(vector<int> v)
99 {
100     if(v.empty()) return {};
101     vector<int> ans(v.size()-1);
102     for(int i=1; i<v.size(); ++i) ans[i-1]=(v[i]*1LL*i)
103     %p;
104     return ans;
105 }
106 vector<int> integ(vector<int> v)
107 {
108     vector<int> ans(v.size()+1); ans[0]=0;
109     for(int i=1; i<v.size(); ++i) ans[i-1]=(v[i]*1LL*i)
110     %p;
111 }
112 return ans;
113 }
114 vector<int> mul(vector<vector<int> > v)
115 {
116     if(v.size()==1) return v[0];
117     vector<vector<int> > v1, v2; for(int i=0; i<v.size()
118     /2; ++i) v1.push_back(v[i]); for(int i=v.size()
119     /2; i<v.size(); ++i) v2.push_back(v[i]);
120     return muls.convolution(mul(v1), mul(v2));
121 }
122 vector<int> inv1(vector<int> v, int n)
123 {
124     assert(v[0]!=0);
125     int sz=1; v=form(v, n); vector<int> a={inv(v[0])};
126     while(sz<n)
127     {
128         vector<int> vsz; for(int i=0; i<min(n, 2*sz); ++i)
129         vsz.push_back(v[i]);
130         vector<int> b=((vector<int>) {1})-muls.
131         convolution(a, vsz);
132         for(int i=0; i<sz; ++i) assert(b[i]==0);
133         b.erase(b.begin(), b.begin()+sz);
134         vector<int> c=muls.convolution(b, a);
135         for(int i=0; i<sz; ++i) a.push_back(c[i]);
136         sz*=2;
137     }
138     return form(a, n);
139 }
140 vector<int> inv(vector<int> v, int n)
141 {
142     v=form(v, n); assert(v[0]!=0); if(v.size()==1) {
143         return {inv(v[0])}; } vector<int> v1=trmi(v);
144     vector<int> a=v1*v; a=form(a, 2*n);
145     vector<int> b((n+1)/2); for(int i=0; i<b.size(); ++i
146     ) b[i]=a[2*i];
147     vector<int> ans1=inv(b, b.size()); vector<int> ans2
148     (n); for(int i=0; i<n; ++i) {if(i%2==0) ans2[i]=
149     ans1[i/2]; else ans2[i]=0;}
150     return form(v1*ans2, n);
151 }
152 vector<int> operator/(vector<int> a, vector<int> b)
153 {
154     while(!a.empty() && a.back()==0) a.pop_back();
155     while(!b.empty() && b.back()==0) b.pop_back();
156     int n=a.size(); int m=b.size(); if(n<m) return {};
157     reverse(a.begin(), a.end()); reverse(b.begin(), b.
158     end()); vector<int> ans=a*inv(b, n-m+1); while(ans
159     .size()>n-m+1) ans.pop_back();
160     reverse(ans.begin(), ans.end()); while(!ans.empty()
161     && ans.back()==0) ans.pop_back(); return ans;
162 }
163 vector<int> operator%(vector<int> a, vector<int> b)
164 {
165     vector<int> ans=a-b*(a/b); while(!ans.empty() &&
166     ans.back()==0) ans.pop_back(); return ans;
167 }

```