

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

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;}
6         else {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 }
```

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);
```

```

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;
  });

```

- 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[
6             i] - 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] -
11    1]) {
12         man[i]++;
13     }
14     if (i + man[i] > r) {
15         l = i - man[i];
16         r = i + man[i];
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)
8         ;
9         for (int j = 0, p = 0; p < n; j = max(1ll,
10        j * 2), lim = p) {
11             p = j, iota(y.begin(), y.end(), n - j);
12             for (int i = 0; i < n; i++) if (sa[i]
13             >= j) y[p++] = sa[i] - j;
14             fill(ws.begin(), ws.end(), 0);
15             for (int i = 0; i < n; i++) ws[x[i]]++;
16             for (int i = 1; i < lim; i++) ws[i] +=
17             ws[i - 1];
18             for (int i = n; i--;) sa[--ws[x[y[i]
19             ]]] = y[i];
20             swap(x, y), p = 1, x[sa[0]] = 0;
21             for (int i = 1; i < n; i++) a = sa[i -
22             1], b = sa[i], x[b] = (y[a] == y[b] && y[a + j]
23             == y[b + j]) ? p - 1 : p++;
24         }
25         for (int i = 1; i < n; i++) rank[sa[i]] = i
26         ;
27         for (int i = 0, j; i < n - 1; lcp[rank[i]
28             ++]=k)
29             for (k && k--, j = sa[rank[i] - 1];
30                 s[i + k] == s[j + k]; k++);
31     }
32 };
33 struct Rmq {
34     const int INF = 1e9;
35     vi rmq;
36     int sz;
37     Rmq(){}
38     void build(int n) {
39         sz = 1;
40         while (sz < n) sz *= 2;
41         rmq.assign(sz * 2, INF);

```

```

32     }
33     Rmq(int n) {
34         sz = 1;
35         while (sz < n) sz *= 2;
36         rmq.assign(sz * 2, INF);
37     }
38     void put(int i, int x) {
39         i += sz;
40         ckmin(rmq[i], x);
41         for (i /= 2; i; i /= 2) {
42             rmq[i] = min(rmq[i * 2], rmq[i * 2 +
43 1]);
44         }
45     }
46     int getMin(int l, int r) { //[l;r)
47         assert(l < r);
48         r--;
49         l += sz;
50         r += sz;
51         int res = INF;
52         while(l < r) {
53             if (l%2 == 1) res = min(res, rmq[l]);
54             if (r%2 == 0) res = min(res, rmq[r]);
55             l = (l + 1)/2;
56             r = (r - 1) / 2;
57         }
58         if (l == r) res = min(res, rmq[l]);
59         return res;
60     };
61
62     struct Lc {
63         vi pos;
64         Rmq rmq;
65         void build(string s) {
66             SuffixArray sa(s);
67             auto ss = sa.sa;
68             ss.erase(ss.begin());
69
70             auto lcp = sa.lcp;
71             lcp.erase(lcp.begin());
72             lcp.erase(lcp.begin());
73
74             pos.resize(s.size());
75             assert(s.size() == ss.size());
76             FOR (i, ss.size()) {
77                 pos[ss[i]] = i;
78             }
79             int n = s.size();
80             assert(lcp.size() == n - 1);
81             rmq.build(n - 1);
82             FOR (i, n - 1) {
83                 rmq.put(i, lcp[i]);
84             }
85         }
86         int getLcp(int i, int j) {
87             i = pos[i]; j = pos[j];
88             if (j < i) {
89                 swap(i, j);
90             }
91             if (i == j) {
92                 return 1e18;
93             }
94             else {
95                 return rmq.getMin(i, j);
96             }
97         }
98     };

```