

Содержание

1 Setup & Scripts	1
1.1 CMake	1
1.2 wipe.sh	1
2 Bugs	1
3 Geometry	2
3.1 Пересечение прямых	2
4 Numbers	2
5 Graphs	3
5.1 Weighted matroid intersection	3
6 Push-free segment tree	5
7 Number theory	6
7.1 Chinese remainder theorem without overflows	6
7.2 Integer points under a rational line	7
8 Suffix Automaton	7

1 Setup & Scripts

1.1 CMake

```

1 cmake_minimum_required(VERSION 3.14)
2 project(olymp)
3
4 set(CMAKE_CXX_STANDARD 17)
5 add_compile_definitions(LOCAL)
6 #set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -fsanitize=undefined -fno-
   ↪ sanitize-recover")
7 #sanitizers: address, leak, thread, undefined, memory
8
9 add_executable(olymp f.cpp)

```

1.2 wipe.sh

```

1 touch {a..l}.cpp
2
3 for file in *.cpp ; do
4     cat template.cpp > $file ;
5 done

```

2 Bugs

- powmod :)
- Всегда чекать Куна дважды, особенно на количество итераций

- `uniform_int_distribution` от одного параметра
- `for (char c : "NEWS")`
- Порядок верхних и нижних границ в случае, когда задача двумерна $t - b \neq b - t$
- `static` с мультитестами

3 Geometry

3.1 Пересечение прямых

$$AB = A - B; CD = C - D$$

$$(A \times B \cdot CD.x - C \times D \cdot AB.x : A \times B \cdot CD.y - C \times D \cdot AB.y : AB \times CD)$$

4 Numbers

- A lot of divisors
 - $\leq 20 : d(12) = 6$
 - $\leq 50 : d(48) = 10$
 - $\leq 100 : d(60) = 12$
 - $\leq 1000 : d(840) = 32$
 - $\leq 10^4 : d(9240) = 64$
 - $\leq 10^5 : d(83160) = 128$
 - $\leq 10^6 : d(720720) = 240$
 - $\leq 10^7 : d(8648640) = 448$
 - $\leq 10^8 : d(91891800) = 768$
 - $\leq 10^9 : d(931170240) = 1344$
 - $\leq 10^{11} : d(97772875200) = 4032$
 - $\leq 10^{12} : d(963761198400) = 6720$
 - $\leq 10^{15} : d(866421317361600) = 26880$
 - $\leq 10^{18} : d(897612484786617600) = 103680$
- Numeric integration
 - simple: $F(0)$
 - simpson: $\frac{F(-1)+4 \cdot F(0)+F(1)}{6}$
 - runge2: $\frac{F(-\sqrt{\frac{1}{3}})+F(\sqrt{\frac{1}{3}})}{2}$
 - runge3: $\frac{F(-\sqrt{\frac{3}{5}}) \cdot 5 + F(0) \cdot 8 + F(\sqrt{\frac{3}{5}}) \cdot 5}{18}$

5 Graphs

5.1 Weighted matroid intersection

```

1 // here we use T = __int128 to store the independent set
2 // calling expand k times to an empty set finds the maximum
3 // cost of the set with size exactly k,
4 // that is independent in blue and red matroids
5 // ver is the number of the elements in the matroid,
6 // e[i].w is the cost of the i-th element
7 // first return value is new independent set
8 // second return value is difference between
9 // new and old costs
10 // oracle(set, red) and oracle(set, blue) check whether
11 // or not the set lies in red or blue matroid respectively
12
13 auto expand = [&] (T cur_set) → pair<T, int>
14 {
15     vector<int> in(ver);
16     for (int i = 0; i < ver; i++)
17         in[i] = ((cur_set >> i) & 1);
18
19     const int red = 1;
20     const int blue = 2;
21
22     vector<vector<int>> g(ver);
23     for (int i = 0; i < ver; i++)
24     for (int j = 0; j < ver; j++)
25     {
26         T swp_mask = (cur_set ^ (T(1) << i) ^ (T(1) << j));
27         if (!in[i] && in[j])
28         {
29             if (oracle(swp_mask, red))
30                 g[i].push_back(j);
31             if (oracle(swp_mask, blue))
32                 g[j].push_back(i);
33         }
34     }
35
36     vector<int> from, to;
37     for (int i = 0; i < ver; i++) if (!in[i])
38     {
39         T add_mask = cur_set ^ (T(1) << i);
40         if (oracle(add_mask, blue))
41             from.push_back(i);
42         if (oracle(add_mask, red))
43             to.push_back(i);
44     }
45
46     auto get_cost = [&] (int x)

```

```
47     {
48         const int cost = (!in[x] ? e[x].w : -e[x].w);
49         return (ver + 1) * cost - 1;
50     };
51
52     const int inf = int(1e9);
53     vector<int> dist(ver, -inf), prev(ver, -1);
54     for (int x : from)
55         dist[x] = get_cost(x);
56
57     queue<int> q;
58
59     vector<int> used(ver);
60     for (int x : from)
61     {
62         q.push(x);
63         used[x] = 1;
64     }
65
66     while (!q.empty())
67     {
68         int cur = q.front(); used[cur] = 0; q.pop();
69
70         for (int to : g[cur])
71         {
72             int cost = get_cost(to);
73             if (dist[to] < dist[cur] + cost)
74             {
75                 dist[to] = dist[cur] + cost;
76                 prev[to] = cur;
77                 if (!used[to])
78                 {
79                     used[to] = 1;
80                     q.push(to);
81                 }
82             }
83         }
84     }
85
86     int best = -inf, where = -1;
87     for (int x : to)
88     {
89         if (dist[x] > best)
90         {
91             best = dist[x];
92             where = x;
93         }
94     }
95
96     if (best == -inf)
```

```

97         return pair<T, int>(cur_set, best);
98
99     while (where  $\neq$  -1)
100     {
101         cur_set ^= (T(1) << where);
102         where = prev[where];
103     }
104
105     while (best % (ver + 1))
106         best++;
107     best /= (ver + 1);
108
109     assert(oracle(cur_set, red) && oracle(cur_set, blue));
110     return pair<T, int>(cur_set, best);
111 };

```

6 Push-free segment tree

```

1 class pushfreesegtree
2 {
3     vector<modulo◇> pushed, unpushed;
4
5     modulo◇ add(int l, int r, int cl, int cr, int v, const modulo
6         ↪ ◇ &x)
7     {
8         if (r ≤ cl || cr ≤ l)
9             return 0;
10        if (l ≤ cl && cr ≤ r)
11        {
12            unpushed[v] += x;
13
14            return x * (cr - cl);
15        }
16
17        int ct = (cl + cr) / 2;
18
19        auto tmp = add(l, r, cl, ct, 2 * v, x) + add(l, r, ct, cr,
20            ↪ 2 * v + 1, x);
21
22        pushed[v] += tmp;
23
24        return tmp;
25    }
26
27    modulo◇ sum(int l, int r, int cl, int cr, int v)
28    {
29        if (r ≤ cl || cr ≤ l)
30            return 0;
31        if (l ≤ cl && cr ≤ r)

```

```

31         return pushed[v] + unpushed[v] * (cr - cl);
32
33     int ct = (cl + cr) / 2;
34
35     return sum(l, r, cl, ct, 2 * v) + unpushed[v] * (min(r, cr)
36         ↪ - max(l, cl)) + sum(l, r, ct, cr, 2 * v + 1);
37 }
38 public:
39     pushfreesegetree(int n) : pushed(2 * up(n)), unpushed(2 * up(n))
40     {}
41
42     modulo◇ sum(int l, int r)
43     {
44         return sum(l, r, 0, pushed.size() / 2, 1);
45     }
46
47     void add(int l, int r, const modulo◇ &x)
48     {
49         add(l, r, 0, pushed.size() / 2, 1, x);
50     }
51 };
```

7 Number theory

7.1 Chinese remainder theorem without overflows

```

1 // Replace T with an appropriate type!
2 using T = long long;
3
4 // Finds x, y such that ax + by = gcd(a, b).
5 T gcdext (T a, T b, T &x, T &y)
6 {
7     if (b == 0)
8     {
9         x = 1, y = 0;
10        return a;
11    }
12
13    T res = gcdext (b, a % b, y, x);
14    y -= x * (a / b);
15    return res;
16 }
17
18 // Returns true if system x = r1 (mod m1), x = r2 (mod m2) has
19 ↪ solutions
20 // false otherwise. In first case we know exactly that x = r (mod m
21 ↪ )
```

```

20
21 bool crt (T r1, T m1, T r2, T m2, T &r, T &m)
22 {
23     if (m2 > m1)
24     {
25         swap(r1, r2);
26         swap(m1, m2);
27     }
28
29     T g = __gcd(m1, m2);
30     if ((r2 - r1) % g  $\neq$  0)
31         return false;
32
33     T c1, c2;
34     auto nrem = gcdext(m1 / g, m2 / g, c1, c2);
35     assert(nrem == 1);
36     assert(c1 * (m1 / g) + c2 * (m2 / g) == 1);
37     T a = c1;
38     a *= (r2 - r1) / g;
39     a %= (m2 / g);
40     m = m1 / g * m2;
41     r = a * m1 + r1;
42     r = r % m;
43     if (r < 0)
44         r += m;
45
46     assert(r % m1 == r1 && r % m2 == r2);
47     return true;
48 }

```

7.2 Integer points under a rational line

```

1 // integer (x, y) :  $0 \leq x < n$ ,  $0 < y \leq (kx + b) / d$ 
2 // (real division)
3 // In other words,  $\sum_{x=0}^{n-1} [(kx+b)/d]$ 
4 ll trapezoid (ll n, ll k, ll b, ll d)
5 {
6     if (k == 0)
7         return (b / d) * n;
8     if (k  $\geq$  d || b  $\geq$  d)
9         return (k / d) * n * (n - 1) / 2 + (b / d) * n + trapezoid(
10              $\rightarrow$  n, k % d, b % d, d);
11     return trapezoid((k * n + b) / d, d, (k * n + b) % d, k);

```

8 Suffix Automaton

```

1 struct Vx{
2     static const int AL = 26;
3     int len, suf;
4     int next[AL];

```

```
5     Vx(){}
6     Vx(int l, int s):len(l), suf(s){}
7 };
8
9 struct SA{
10     static const int MAX_LEN = 1e5 + 100, MAX_V = 2 * MAX_LEN;
11     int last, vcnt;
12     Vx v[MAX_V];
13
14     SA(){
15         vcnt = 1;
16         last = newV(0, 0); // root = vertex with number 1
17     }
18     int newV(int len, int suf){
19         v[vcnt] = Vx(len, suf);
20         return vcnt++;
21     }
22
23     int add(char ch){
24         int p = last, c = ch - 'a';
25         last = newV(v[last].len + 1, 0);
26         while(p && !v[p].next[c]) //added p &&
27             v[p].next[c] = last, p = v[p].suf;
28         if(!p)
29             v[last].suf = 1;
30         else{
31             int q = v[p].next[c];
32             if (v[q].len == v[p].len + 1)
33                 v[last].suf = q;
34             else{
35                 int r = newV(v[p].len + 1, v[q].suf);
36                 v[last].suf = v[q].suf = r;
37                 memcpy(v[r].next, v[q].next, sizeof(v[r].next));
38                 while(p && v[p].next[c] == q)
39                     v[p].next[c] = r, p = v[p].suf;
40             }
41         }
42         return last;
43     }
44 };
```
































