Team Reference

SPb SU LOUD Enough

2019–2020

1 Setup & Scripts

1.1 CMake

```
columns
cmake_minimum_required(VERSION 3.14)
project(olymp)

set(CMAKE_CXX_STANDARD 17)
add_compile_definitions(LOCAL)
#set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -fsanitize=
   ↪ undefined -fno-sanitize-recover")
#sanitizers: address, leak, thread, undefined, memory

add_executable(olymp f.cpp)
```

1.2 wipe.sh

```
columns
touch {a..l}.cpp

for file in ?.cpp ; do
    cat template.cpp > $file ;
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

```
columns
// here we use T = __int128 to store the independent
   ↪ set
// calling expand k times to an empty set finds the
   ↪ maximum
// cost of the set with size exactly k,
// that is independent in blue and red matroids
// ver is the number of the elements in the matroid,
// e[i].w is the cost of the i-th element
// first return value is new independent set
// second return value is difference between
// new and old costs
// oracle(set, red) and oracle(set, blue) check
   ↪ whether
// or not the set lies in red or blue matroid
   ↪ respectively

auto expand = [&] (T cur_set) -> pair<T, int>
{
    vector<int> in(ver);
    for (int i = 0; i < ver; i++)
        in[i] = ((cur_set  »  i) & 1);

    const int red = 1;
    const int blue = 2;

    vector<vector<int » g(ver);
```

```
for (int i = 0; i < ver; i++)
for (int j = 0; j < ver; j++)
{
    T swp_mask = (cur_set ^ (T(1) << i) ^ (T(1) <<
       ↪ j));
    if (!in[i] && in[j])
    {
        if (oracle(swp_mask, red))
            g[i].push_back(j);
        if (oracle(swp_mask, blue))
            g[j].push_back(i);
    }
}

vector<int> from, to;
for (int i = 0; i < ver; i++) if (!in[i])
{
    T add_mask = cur_set ^ (T(1) << i);
    if (oracle(add_mask, blue))
        from.push_back(i);
    if (oracle(add_mask, red))
        to.push_back(i);
}

auto get_cost = [&] (int x)
{
    const int cost = (!in[x] ? e[x].w : -e[x].w);
    return (ver + 1) * cost - 1;
};

const int inf = int(1e9);
vector<int> dist(ver, -inf), prev(ver, -1);
for (int x : from)
    dist[x] = get_cost(x);

queue<int> q;

vector<int> used(ver);
for (int x : from)
{
    q.push(x);
```

```
    used[x] = 1;
}

while (!q.empty())
{
    int cur = q.front(); used[cur] = 0; q.pop();

    for (int to : g[cur])
    {
        int cost = get_cost(to);
        if (dist[to] < dist[cur] + cost)
        {
            dist[to] = dist[cur] + cost;
            prev[to] = cur;
            if (!used[to])
            {
                used[to] = 1;
                q.push(to);
            }
        }
    }
}

int best = -inf, where = -1;
for (int x : to)
{
    if (dist[x] > best)
    {
        best = dist[x];
        where = x;
    }
}

if (best == -inf)
    return pair<T, int>(cur_set, best);

while (where != -1)
{
    cur_set ^= (T(1) « where);
    where = prev[where];
}
```

```
    while (best % (ver + 1))
        best++;
    best /= (ver + 1);

    assert(oracle(cur_set, red) && oracle(cur_set,
       ↪ blue));
    return pair<T, int>(cur_set, best);
};
```

6 Push-free segment tree

```
columns
class pushfreesegtree
{
    vector<modulo<>> pushed, unpushed;

    modulo<> add(int l, int r, int cl, int cr, int v,
       ↪ const modulo<> &x)
    {
        if (r <= cl || cr <= l)
            return 0;
        if (l <= cl && cr <= r)
        {
            unpushed[v] += x;

            return x * (cr - cl);
        }

        int ct = (cl + cr) / 2;

        auto tmp = add(l, r, cl, ct, 2 * v, x) + add(l
           ↪ , r, ct, cr, 2 * v + 1, x);

        pushed[v] += tmp;

        return tmp;
    }
```

```
    modulo<> sum(int l, int r, int cl, int cr, int v)
    {
        if (r <= cl || cr <= l)
            return 0;
        if (l <= cl && cr <= r)
            return pushed[v] + unpushed[v] * (cr - cl)
               ↪ ;

        int ct = (cl + cr) / 2;

        return sum(l, r, cl, ct, 2 * v) + unpushed[v]
           ↪ * (min(r, cr) - max(l, cl)) + sum(l, r,
           ↪ ct, cr, 2 * v + 1);
    }

public:
    pushfreesegtree(int n) : pushed(2 * up(n)),
       ↪ unpushed(2 * up(n))
    {}


    modulo<> sum(int l, int r)
    {
        return sum(l, r, 0, pushed.size() / 2, 1);
    }


    void add(int l, int r, const modulo<> &x)
    {
        add(l, r, 0, pushed.size() / 2, 1, x);
    }
};
```