Содержание

1 Setup & Scripts

1.1 CMake

```
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

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

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
- set со своим компаратором склеивает элементы

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

```
// 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

```
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;
    }

```

```
25
26      modulo<> sum(int l, int r, int cl, int cr, int v)
27      {
28          if (r <= cl || cr <= l)
29              return 0;
30          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)
              ↪  - max(l, cl)) + sum(l, r, ct, cr, 2 * v + 1);
36      }
37
38  public:
39      pushfreesegtree(int n) : pushed(2 * up(n)), unpushed(2 * up(n))
40      {}
41
42
43      modulo<> sum(int l, int r)
44      {
45          return sum(l, r, 0, pushed.size() / 2, 1);
46      }
47
48
49      void add(int l, int r, const modulo<> &x)
50      {
51          add(l, r, 0, pushed.size() / 2, 1, x);
52      }
53  };
```

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;
```

```
}

// Returns true if system x = r1 (mod m1), x = r2 (mod m2) has solutions
// false otherwise. In first case we know exactly that x = r (mod m)

bool crt (T r1, T m1, T r2, T m2, T &r, T &m)
{
    if (m2 > m1)
    {
        swap(r1, r2);
        swap(m1, m2);
    }

    T g = __gcd(m1, m2);
    if ((r2 - r1) % g != 0)
        return false;

    T c1, c2;
    auto nrem = gcdext(m1 / g, m2 / g, c1, c2);
    assert(nrem == 1);
    assert(c1 * (m1 / g) + c2 * (m2 / g) == 1);
    T a = c1;
    a *= (r2 - r1) / g;
    a %= (m2 / g);
    m = m1 / g * m2;
    r = a * m1 + r1;
    r = r % m;
    if (r < 0)
        r += m;

    assert(r % m1 == r1 && r % m2 == r2);
    return true;
}
```

7.2 Integer points under a rational line

```
// integer (x, y) : 0 ⩽ x < n, 0 < y ⩽ (kx + b) / d
// (real division)
// In other words, sum_{x=0}^{n-1} [(kx+b)/d]
ll trapezoid (ll n, ll k, ll b, ll d)
{
    if (k == 0)
        return (b / d) * n;
    if (k ⩾ d || b ⩾ d)
        return (k / d) * n * (n - 1) / 2 + (b / d) * n + trapezoid(n, k % d, b % d, d);
    return trapezoid((k * n + b) / d, d, (k * n + b) % d, k);
}
```

8 Suffix Automaton

```
struct Vx{
    static const int AL = 26;
    int len, suf;
    int next[AL];
    Vx(){}
    Vx(int l, int s):len(l), suf(s){}
};

struct SA{
    static const int MAX_LEN = 1e5 + 100, MAX_V = 2 * MAX_LEN;
    int last, vcnt;
    Vx v[MAX_V];

    SA(){
        vcnt = 1;
        last = newV(0, 0); // root = vertex with number 1
    }
    int newV(int len, int suf){
        v[vcnt] = Vx(len, suf);
        return vcnt++;
    }

    int add(char ch){
        int p = last, c = ch - 'a';
        last = newV(v[last].len + 1, 0);
        while(p && !v[p].next[c]) //added p &&
            v[p].next[c] = last, p = v[p].suf;
        if(!p)
            v[last].suf = 1;
        else{
            int q = v[p].next[c];
            if (v[q].len == v[p].len + 1)
                v[last].suf = q;
            else{
                int r = newV(v[p].len + 1, v[q].suf);
                v[last].suf = v[q].suf = r;
                memcpy(v[r].next, v[q].next, sizeof(v[r].next));
                while(p && v[p].next[c] == q)
                    v[p].next[c] = r, p = v[p].suf;
            }
        }
        return last;
    }
};
```

9 Smth added at last moment

9.1 Dominator Tree

```
struct dom_tree {
  vvi g, rg, tree, bucket;
  vi sdom, par, dom, dsu, label, in, order, tin, tout;
  int T = 0, root = 0, n = 0;

  void dfs_tm (int x) {
    in[x] = T;
    order[T] = x;
    label[T] = T, sdom[T] = T, dsu[T] = T, dom[T] = T;
    T++;
    for (int to : g[x]) {
      if (in[to] == -1) {
        dfs_tm(to);
        par[in[to]] = in[x];
      }
      rg[in[to]].pb(in[x]);
    }
  }

  void dfs_tree (int v, int p) {
    tin[v] = T++;
    for (int dest : tree[v]) {
      if (dest != p) {
        dfs_tree(dest, v);
      }
    }
    tout[v] = T;
  }

  dom_tree (const vvi &g_, int root_) {
    g = g_;
    n = sz(g);
    assert(0 <= root && root < n);
    in.assign(n, -1);
    rg.resize(n);
    order = sdom = par = dom = dsu = label = vi(n);
    root = root_;
    bucket.resize(n);
    tree.resize(n);

    dfs_tm(root);

    for (int i = n - 1; i >= 0; i--) {
      for (int j : rg[i])
        sdom[i] = min(sdom[i], sdom[find(j)]);
      if (i > 0)
        bucket[sdom[i]].pb(i);

      for (int w : bucket[i]) {
        int v = find(w);
```

```
51          dom[w] = (sdom[v] == sdom[w] ? sdom[w] : v);
52        }
53
54        if (i > 0)
55          unite(par[i], i);
56      }
57
58      for (int i = 1; i < n; i++) {
59        if (dom[i] != sdom[i])
60          dom[i] = dom[dom[i]];
61        tree[order[i]].pb(order[dom[i]]);
62        tree[order[dom[i]]].pb(order[i]);
63      }
64
65      T = 0;
66      tin = tout = vi(n);
67      dfs_tree(root, -1);
68    }
69
70    void unite (int u, int v) {
71      dsu[v] = u;
72    }
73
74    int find (int u, int x = 0) {
75      if (u == dsu[u])
76        return (x ? -1 : u);
77      int v = find(dsu[u], x + 1);
78      if (v == -1)
79        return u;
80      if (sdom[label[dsu[u]]] < sdom[label[u]])
81        label[u] = label[dsu[u]];
82      dsu[u] = v;
83      return (x ? v : label[u]);
84    }
85
86    bool dominated_by (int v, int by_what) {
87      return tin[by_what] <= tin[v] && tout[v] <= tout[by_what];
88    }
89  };
```

9.2 Suffix Array

```
1  namespace suff_arr {
2
3  const int MAXN = 2e5 + 10;
4
5  string s;
6  int n;
7  int p[MAXN];
8  int lcp[MAXN];
9  int pos[MAXN];
```

```
10 int c[MAXN];
11
12 void print() {
13 #ifndef LOCAL
14     return;
15 #endif
16     eprintf("p:\n");
17     forn(i, sz(s)) {
18         eprintf("i=%d -- %d: %s, lcp=%d, c=%d\n", i, p[i], s.substr
               ↪ (p[i], sz(s) - p[i]).data(), lcp[i], c[p[i]]);
19     }
20     eprintf("\n");
21 }
22
23 void build(const string& s_) {
24     static int cnt[MAXN];
25     static int np[MAXN];
26     static int nc[MAXN];
27
28     s = s_;
29     n = sz(s);
30
31     memset (cnt, 0, sizeof cnt);
32     for (char ch : s) {
33         ++cnt[int(ch)];
34     }
35     forn(i, 256) {
36         cnt[i + 1] += cnt[i];
37     }
38     forn(i, sz(s)) {
39         p[--cnt[int(s[i])]] = i;
40     }
41
42     int cls = 1;
43     c[p[0]] = cls - 1;
44     for (int i = 1; i < n; ++i) {
45         if  (s[p[i]] ≠ s[p[i - 1]]) {
46             ++cls;
47         }
48         c[p[i]] = cls - 1;
49     }
50
51     for (int len = 1; len ⩽ n; len *= 2) {
52         memset (cnt, 0, sizeof(int) * cls);
53         forn(i, n) {
54             ++cnt[c[i]];
55         }
56         forn(i, cls - 1) {
57             cnt[i + 1] += cnt[i];
58         }
```

```
59          ford(i, n) {
60              const int j = p[i];
61              int j2 = (j - len + n) % n;
62              np[--cnt[c[j2]]] = j2;
63          }
64          memcpy(p, np, sizeof(int) * n);
65
66          cls = 1;
67          nc[p[0]] = cls - 1;
68          for (int i = 1; i < n; ++i) {
69              if  (c[p[i]] != c[p[i - 1]] || c[(p[i] + len) % n] != c
                ↪ [(p[i - 1] + len) % n]) {
70                  ++cls;
71              }
72              nc[p[i]] = cls - 1;
73          }
74          memcpy(c, nc, sizeof(int) * n);
75      }
76
77      forn(i, n) {
78          pos[p[i]] = i;
79      }
80
81      int pref = 0;
82      forn(i, n) {
83          int pi = pos[i];
84          if (pi == n - 1) {
85              continue;
86          }
87          int j = p[pi + 1];
88          while (i + pref < n && j + pref < n && s[i + pref] == s[j +
                ↪  pref]) {
89              ++pref;
90          }
91          lcp[pi] = pref;
92          pref = max(0, pref - 1);
93      }
94
95  //        print();
96  }
97
98  };
```

9.3 Fast LCS

```
1  // assumes that strings consist of lowercase latin letters
2  const int M = ((int)1e5 + 64) / 32 * 32;
3  // maximum value of m
4  using bs = bitset<M>;
5  using uint = unsigned int;
6  const ll bnd = (1LL << 32);
```

```
7
8  // WARNING: invokes undefined behaviour of modifying ans through
       ↪ pointer to another data type (uint)
9  // seems to work, but be wary
10 bs sum (const bs &bl, const bs &br)
11 {
12     const int steps = M / 32;
13     const uint* l = (uint*)&bl;
14     const uint* r = (uint*)&br;
15
16     bs ans;
17     uint* res = (uint*)&ans;
18
19     int carry = 0;
20     forn (i, steps)
21     {
22         ll cur = ll(*l++) + ll(*r++) + carry;
23         carry = (cur ⩾ bnd);
24         cur = (cur ⩾ bnd ? cur - bnd : cur);
25         *res++ = uint(cur);
26     }
27
28     return ans;
29 }
30
31 int fast_lcs (const string &s, const string &t)
32 {
33     const int m = sz(t);
34     const int let = 26;
35
36     vector<bs> has(let);
37     vector<bs> rev = has;
38
39     forn (i, m)
40     {
41         const int pos = t[i] - 'a';
42         has[pos].set(i);
43         forn (j, let) if (j ≠ pos)
44             rev[j].set(i);
45     }
46
47     bs row;
48     forn (i, m)
49         row.set(i);
50
51     int cnt = 0;
52     for (char ch : s)
53     {
54         const int pos = ch - 'a';
55
```

```
56          bs next = sum(row, row & has[pos]) | (row & rev[pos]);
57          cnt += next[m];
58          next[m] = 0;
59
60          row = next;
61      }
62
63      return cnt;
64  }
```

9.4 Fast Subset Convolution

```
1   // algorithm itself starts here
2   void mobius (int* a, int n, int sign)
3   {
4       forn (i, n)
5       {
6           int free = ((1 << n) - 1) ^ (1 << i);
7           for (int mask = free; mask > 0; mask = ((mask - 1) & free))
8               (sign == +1 ? add : sub)(a[mask ^ (1 << i)], a[mask]);
9           add(a[1 << i], a[0]);
10      }
11  }
12
13  // maximum number of bits allowed
14  const int B = 20;
15
16  vi fast_conv (vi a, vi b)
17  {
18      assert(!a.empty());
19      const int bits = __builtin_ctz(sz(a));
20      assert(sz(a) == (1 << bits) && sz(a) == sz(b));
21
22      static int trans_a[B + 1][1 << B];
23      static int trans_b[B + 1][1 << B];
24      static int trans_res[B + 1][1 << B];
25
26      forn (cnt, bits + 1)
27      {
28          for (auto cur : {trans_a, trans_b, trans_res})
29              fill(cur[cnt], cur[cnt] + (1 << bits), 0);
30      }
31
32      forn (mask, 1 << bits)
33      {
34          const int cnt = __builtin_popcount(mask);
35          trans_a[cnt][mask] = a[mask];
36          trans_b[cnt][mask] = b[mask];
37      }
38
39      forn (cnt, bits + 1)
```

```
40      {
41          mobius(trans_a[cnt], bits, +1);
42          mobius(trans_b[cnt], bits, +1);
43      }
44
45      // Not really a valid ranked mobius transform! But algorithm
           ↪ works anyway
46
47      forn (i, bits + 1) forn (j, bits - i + 1) forn (mask, 1 << bits
           ↪ )
48          add(trans_res[i + j][mask], mult(trans_a[i][mask], trans_b[
               ↪ j][mask]));
49
50      forn (cnt, bits + 1)
51          mobius(trans_res[cnt], bits, -1);
52
53      forn (mask, 1 << bits)
54      {
55          const int cnt = __builtin_popcount(mask);
56          a[mask] = trans_res[cnt][mask];
57      }
58
59      return a;
60  }
```

10 Karatsuba

```
1   // functon Karatsuba (and stupid as well) computes c += a * b, not
       ↪ c = a * b
2
3   using hvect = vector<modulo<>>::iterator;
4   using hcvect = vector<modulo<>>::const_iterator;
5
6
7   void add(hcvect abegin, hcvect aend, hvect ans)
8   {
9       for (auto it = abegin; it != aend; ++it, ++ans)
10          *ans += *it;
11  }
12
13
14  void sub(hcvect abegin, hcvect aend, hvect ans)
15  {
16      for (auto it = abegin; it != aend; ++it, ++ans)
17          *ans -= *it;
18  }
19
20
21  void stupid(int siz, hcvect abegin, hcvect bbegin, hvect ans)
22  {
```

```
23      for (auto a = abegin; a =/= abegin + siz; ++a, ans -= (siz - 1))
24          for (auto b = bbegin; b =/= bbegin + siz; ++b, ++ans)
25              *ans += *a * *b;
26  }
27
28
29  void Karatsuba(size_t siz, hcvect abegin, hcvect bbegin, hvect ans,
        ↪  hvect small, hvect big, hvect sum)
30  {
31      assert((siz & (siz - 1)) == 0);
32
33      if (siz <= 32)
34      {
35          stupid(siz, abegin, bbegin, ans);
36
37          return;
38      }
39
40      auto amid = abegin + siz / 2, aend = abegin + siz;
41      auto bmid = bbegin + siz / 2, bend = bbegin + siz;
42      auto smid = sum + siz / 2, send = sum + siz;
43
44      fill(small, small + siz, 0);
45      Karatsuba(siz / 2, abegin, bbegin, small, small + siz, big +
            ↪ siz, sum);
46      fill(big, big + siz, 0);
47      Karatsuba(siz / 2, amid, bmid, big, small + siz, big + siz, sum
            ↪ );
48
49      copy(abegin, amid, sum);
50      add(amid, aend, sum);
51      copy(bbegin, bmid, sum + siz / 2);
52      add(bmid, bend, sum + siz / 2);
53
54      Karatsuba(siz / 2, sum, smid, ans + siz / 2, small + siz, big +
            ↪  siz, send);
55
56      add(small, small + siz, ans);
57      sub(small, small + siz, ans + siz / 2);
58      add(big, big + siz, ans + siz);
59      sub(big, big + siz, ans + siz / 2);
60  }
61
62
63  void mult(vector<modulo<>> a, vector<modulo<>> b, vector<modulo<>>
        ↪ &c)
64  {
65      a.resize(up(max(a.size(), b.size())), 0);
66      b.resize(a.size(), 0);
67
```

```
68      c.assign(a.size() * 2, 0);
69
70      auto small = c;
71      auto big = c;
72      auto sum = c;
73
74      Karatsuba(a.size(), a.begin(), b.begin(), c.begin(), small.
           ↪ begin(), big.begin(), sum.begin());
75  }
```