Ivan Franko National University of Lviv

Turtles

Ivan Manchur, Oleksandr Zhenchenko, Volodymyr Dudchak

50th Annual World Championship of the International Collegiate Programming Contest

Month ??, 202?

Contest (1)

template.hpp 28 lines

```
// hash = 9fd99b

#include <bits/stdc++.h>
using namespace std;

#define FOR(i, a, b) for(int i = (a); i < (b); i++)
#define RFOR(i, a, b) for(int i = (a) - 1; i >= (b); i--)
#define SZ(a) int(a.size())
#define ALL(a) a.begin(), a.end()
#define PB push_back
#define MP make_pair
#define F first
#define S second

typedef long long LL;
typedef vector<int> VI;
typedef vector<LL> VL;
typedef pair<int, int> PII;
typedef pair<LL, LL> PLL;
typedef double db;

int main()
{
  ios::sync_with_stdio(0);
  cin.tie(0);

  return 0;
}
```

compilation.txt 2 lines

```
g++ -O2 -std=c++20 -Wno-unused-result -Wshadow -Wall -o %e %e.
    cpp
g++ -std=c++20 -Wshadow -Wall -o %e %e.cpp -fsanitize=address -
    fsanitize=undefined -D_GLIBCXX_DEBUG -g
```

s.sh 6 lines

```
for((i = 0; ; i++)) do
  echo $i
  ./gen $i > in
  diff -w <(./a < in) <(./brute < in) || break
  [ $? == 0 ] || break
done
```

hash.sh 1 lines

```
cpp -dD -P -fpreprocessed $1 | tr -d '[:space:]'| md5sum |cut -
    c-6
```

Rules

Reject incorrect solutions from your teammates. Try to find counterexamples.

Discuss implementation and try to simplify the solution.

Avoid getting stuck on the problem.

Regularly discuss how many problems need to be solved and what steps to take, starting from the middle of the contest.

At the end of the contest, try to find a problem with an easy implementation.

Troubleshoot

Pre-submit

F9. Create a few manual test cases. Calculate time and memory complexity. Check the limits. Be careful with overflows, constants, clearing mutitestcases, uninitialized variables.

Wrong answer

F9. Print your solution! Read your code. Check pre-submit. Are you sure your algorithm works? Think about precision errors and hash collisions. Have you understood the problem correctly? Write the brute and the generator.

Runtime error

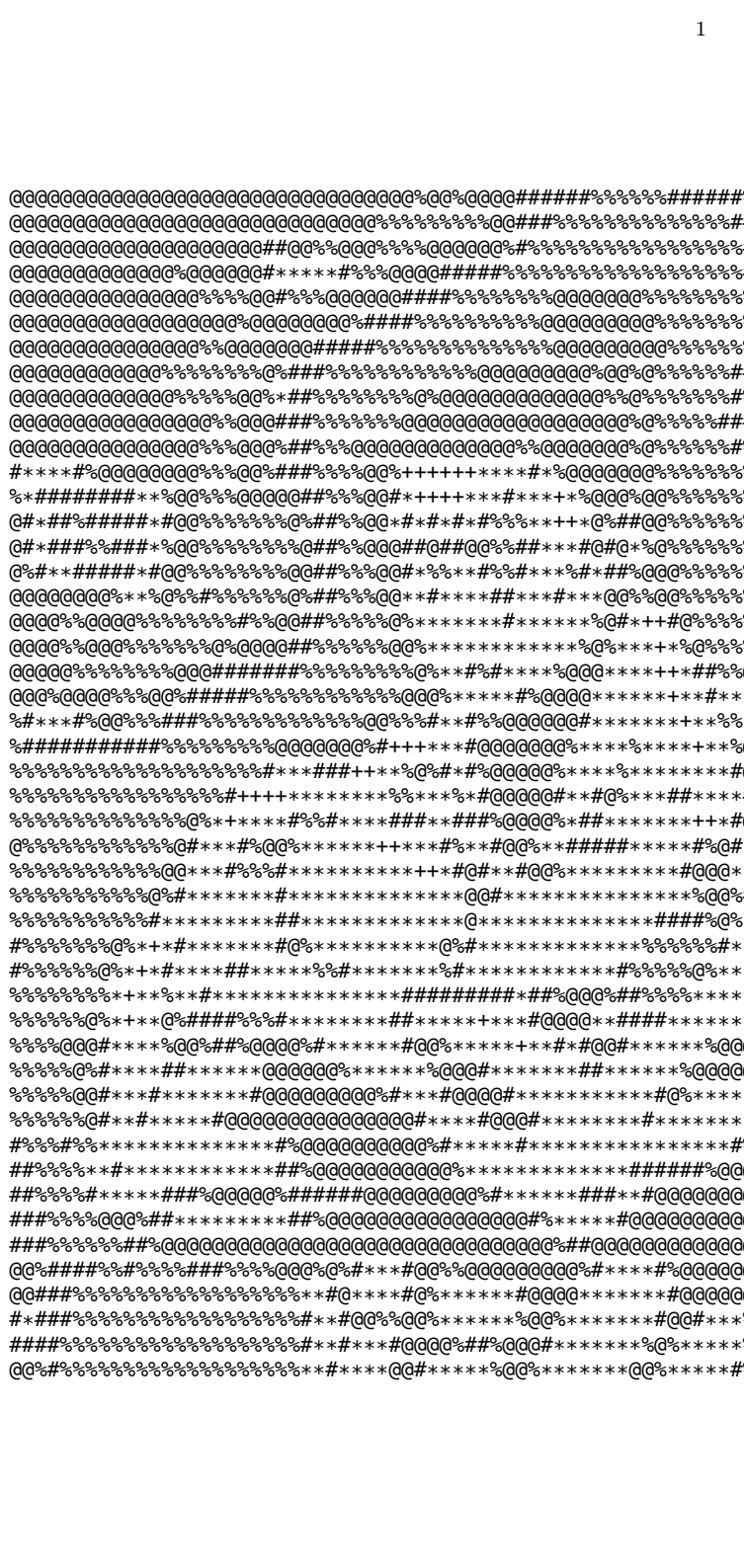
F9. Print your solution! Read your code. F9 with generator. Memory limit exceeded.

Time limit exceeded

What is the complexity of your algorithm? Are you copying a lot of unnecessary data? (References) Do you have any infinite loops? Use arrays, unordered maps instead of vectors and maps.

Pragmas

- **#pragma** GCC optimize ("Ofast") will make GCC auto-vectorize loops and optimizes floating points better. It is not unexpected to see your floating-point error analysis go to waste.
- **#pragma** GCC target ("avx2") can double performance of vectorized code, but causes crashes on old machines.
- **#pragma** GCC optimize("unroll-loops") enables aggressive loop unrolling, which reduces the number of branches and optimizes parallel computation.



Data Structures (2)

dsu.hpp

f097c2, 31 lines

```
struct DSU
{
  int n;
  VI p, sz;

  DSU(int _n = 0)
  {
    n = _n;
    p.resize(n);
    iota(ALL(p), 0);
    sz.assign(n, 1);
  }
  int find(int v)
  {
    if (v == p[v])
      return v;
    return p[v] = find(p[v]);
  }
  bool unite(int u, int v)
  {
    u = find(u);
    v = find(v);
    if (u == v)
      return false;
    if (sz[u] > sz[v])
      swap(u, v);
    p[u] = v;
    sz[v] += sz[u];
    return true;
  }
};
```

fenwick.hpp

6a7a21, 20 lines

```
struct Fenwick
{
  int n;
  VL t;

  Fenwick(int _n = 0): n(_n), t(n) {}

  void upd(int i, LL x)
  {
    for (; i < n; i |= i + 1)
      t[i] += x;
  }
  LL query(int i)
  {
    LL ans = 0;
    for (; i >= 0; i = (i & (i + 1)) - 1)
      ans += t[i];
    return ans;
  }
};
```

fenwick-lower-bound.hpp

730a06, 17 lines

```
int lowerBound(LL x)
{
  LL sum = 0;
  int i = -1;
  int lg = 31 - __builtin_clz(n);
  while (lg >= 0)
  {
    int j = i + (1 << lg);
    if (j < n && sum + t[j] < x)
    {
      sum += t[j];
      i = j;
    }
    lg--;
  }
  return i + 1;
}
```

Minimum on a Segment

Maintain two Fenwick trees with $n = 2^k$ — one for the original array and the other for the reversed array. If $n > 1$, you can use: `n = 1 << (32 - __builtin_clz(n - 1))`.

When querying for the minimum on the segment, only consider segments $[(i\&(i+1)), i]$ that are completely inside $[l, r]$.

Add on a Segment

Maintain two Fenwick trees: tMult and tAdd.

To add x on the segment $[l, r]$, tMult.upd(l, x), tMult.upd(r, $-x$), tAdd.upd(l, $-x \cdot (l-1)$), tAdd.upd(r, $x \cdot r$).

$r\cdot$ tMult.query(r) + tAdd.query(r) is the sum on $[0, r]$.

min= and sum with Segment Tree

Store in each node: max, cntMax, max2, sum.

In update check l, r conditions and:

- if (val $\geq$ max) return;
- else if (val $>$ max2) update this node;
- else go to left and right

You can do max= and += on segment at the same time. Time: $O(\log n)$. Each extra descent decreases number of diferent elements in segment.

treap.hpp

Description: uncomment in split for explicit key or in merge for implicit priority. Minimum and reverse queries.

5af66f, 144 lines

```
mt19937 rng;

struct Node
{
  int l, r;
  int x, y;
  int cnt, par;
  int rev, mn;

  Node(int value)
  {
    l = r = -1;
    x = value;
    y = rng();
    cnt = 1;
    par = -1;
    rev = 0;
    mn = value;
  }
};

struct Treap
{
  vector<Node> t;

  int getCnt(int v)
  {
    if (v == -1)
      return 0;
    return t[v].cnt;
  }
  int getMn(int v)
  {
    if (v == -1)
      return INF;
    return t[v].mn;
  }
  int newNode(int val)
  {
    t.PB({val});
    return SZ(t) - 1;
  }
  void upd(int v)
  {
    if (v == -1)
      return;
    // important!
    t[v].cnt = getCnt(t[v].l) +
    getCnt(t[v].r) + 1;

    t[v].mn = min(t[v].x, min(getMn(t[v].l), getMn(t[v].r)));
  }
  void reverse(int v)
  {
    if (v == -1)
      return;
    t[v].rev ^= 1;
  }
  void push(int v)
  {
    if (v == -1 || t[v].rev == 0)
      return;
    reverse(t[v].l);
    reverse(t[v].r);
    swap(t[v].l, t[v].r);
    t[v].rev = 0;
  }
  PII split(int v, int cnt)
  {
    if (v == -1)
      return {-1, -1};
    push(v);
    int left = getCnt(t[v].l);
    PII res;
    // elements a[v].x == val will be in right part
    // if (val <= a[v].x)
    if (cnt <= left)
    {
      if (t[v].l != -1)
        t[t[v].l].par = -1;
      // res = split(a[v].l, val);
      res = split(t[v].l, cnt);
      t[v].l = res.S;
      if (res.S != -1)
        t[res.S].par = v;
      res.S = v;
    }
    else
    {
      if (t[v].r != -1)
        t[t[v].r].par = -1;
      // res = split(a[v].r, val);
      res = split(t[v].r, cnt - left - 1);
      t[v].r = res.F;
```

```
      if (res.F != -1)
        t[res.F].par = v;
      res.F = v;
    }
    upd(v);
    return res;
  }
  int merge(int v, int u)
  {
    if (v == -1) return u;
    if (u == -1) return v;
    int res;
    // if ((int)(rng() % (getCnt(v) + getCnt(u))) < getCnt(v))
    if (t[v].y > t[u].y)
    {
      push(v);
      if (t[v].r != -1)
        t[t[v].r].par = -1;
      res = merge(t[v].r, u);
      t[v].r = res;
      if (res != -1)
        t[res].par = v;
      res = v;
    }
    else
    {
      push(u);
      if(t[u].l != -1)
        t[t[u].l].par = -1;
      res = merge(v, t[u].l);
      t[u].l = res;
      if (res != -1)
        t[res].par = u;
      res = u;
    }
    upd(res);
    return res;
  }
  // returns index of element [0, n)
  int getIdx(int v, int from = -1)
  {
    if (v == -1)
      return 0;
    int x = getIdx(t[v].par, v);
    push(v);
    if (from == -1 || t[v].r == from)
      x += getCnt(t[v].l) + (from != -1);
    return x;
  }
};
```

lct.hpp

Description: Link-Cut Tree. Calculate any path queries. Change `upd` to maintain what you need. Don't use `upd` in `push`:). Calculate non commutative functions in both ways and swap them in push. `cnt` – number of nodes in current splay tree. Don't touch `rev`, `sub`, `vsub`. `v->access()` brings `v` to the top and pushes it; its left subtree will be the path from `v` to the root and its right subtree will be empty. Only then `sub` will be the number of nodes in the connected component of `v` and `vsub` will be the number of nodes under `v`. Change upd to calc sum in subtree of other functions. Use `makeRoot` for arbitrary path queries.
Usage: `FOR (i, 0, n) LCT[i] = new snode(i); link(LCT[u], LCT[v]);`
Time: $\mathcal{O}(\log n)$
788027, 159 lines

```
typedef struct Snode* sn;
struct Snode
{
  sn p, c[2]; // parent, children
  bool rev = false; // subtree reversed or not (internal usage)
  int val, cnt; // value in node, # nodes in splay subtree
  int sub, vsub = 0; // vsub stores sum of virtual children
  Snode(int _val): val(_val)
  {
    p = c[0] = c[1] = 0;
    upd();
  }
  friend int getCnt(sn v)
  {
    return v ? v->cnt : 0;
  }
  friend int getSub(sn v)
  {
    return v ? v->sub : 0;
  }
  void push()
  {
    if (!rev)
      return;
    swap(c[0], c[1]);
    rev = false;
    FOR (i, 0, 2)
      if (c[i])
        c[i]->rev ^= 1;
  }
  void upd()
  {
    FOR (i, 0, 2)
      if (c[i])
        c[i]->push();
    cnt = 1 + getCnt(c[0]) + getCnt(c[1]);
    sub = 1 + getSub(c[0]) + getSub(c[1]) + vsub;
  }
  int dir()
  {
    if (!p) return -2;
    FOR (i, 0, 2)
      if (p->c[i] == this)
        return i;
    // p is path-parent pointer
    // -> not in current splay tree
    return -1;
  }

  // checks if root of current splay tree
  bool isRoot()
  {
    return dir() < 0;
  }
  friend void setLink(sn p, sn v, int d)
  {
    if (v)
      v->p = p;
    if (d >= 0)
      p->c[d] = v;
  }
  void rot()
  {
    assert(!isRoot());
    int d = dir();
    sn pa = p;
    setLink(pa->p, this, pa->dir());
    setLink(pa, c[d ^ 1], d);
    setLink(this, pa, d ^ 1);
    pa->upd();
  }
  void splay()
  {
    while (!isRoot() && !p->isRoot())
    {
      p->p->push();
      p->push();
      push();
      dir() == p->dir() ? p->rot() : rot();
      rot();
    }
    if (!isRoot())
      p->push(), push(), rot();
    push();
    upd();
  }
  // bring this to top of tree, propagate
  void access()
  {
    for (sn v = this, pre = 0; v; v = v->p)
    {
      v->splay();
      if (pre)
        v->vsub -= pre->sub;
      if (v->c[1])
        v->vsub += v->c[1]->sub;
      v->c[1] = pre;
      v->upd();
      pre = v;
    }
    splay();
    assert(!c[1]);
  }
  void makeRoot()
  {
    access();
    rev ^= 1;
    access();
    assert(!c[0] && !c[1]);
  }
  friend sn lca(sn u, sn v)
  {
    if (u == v)
      return u;
    u->access();
    v->access();
    if (!u->p)
      return 0;
    u->splay();
    return u->p ? u->p : u;
  }
  friend bool connected(sn u, sn v)
  {
    return lca(u, v);
  }
  void set(int v)
  {
    access();
    val = v;
    upd();
  }
  friend void link(sn u, sn v)
  {
    assert(!connected(u, v));
    v->makeRoot();
    u->access();
    setLink(v, u, 0);
    v->upd();
  }
  // cut v from it's parent in LCT
  // make sure about root or better use next function
  friend void cut(sn v)
  {
    v->access();
    assert(v->c[0]); // assert if not a root
```

```
    v->c[0]->p = 0;
    v->c[0] = 0;
    v->upd();
  }
  // u, v should be adjacent in tree
  friend void cut(sn u, sn v)
  {
    u->makeRoot();
    v->access();
    assert(v->c[0] == u && !u->c[0] && !u->c[1]);
    cut(v);
  }
};
```

ordered-set.hpp

16 lines

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
using namespace std;
typedef tree<int, null_type, less<int>, rb_tree_tag,
     tree_order_statistics_node_update> ordered_set;

ordered_set s;
s.insert(47);
// Returns the number of elements less then k
s.order_of_key(k);
// Returns iterator to the k-th element or s.end()
s.find_by_order(k);
// Does not exist
s.count();
// Doesn't trigger RE. Returns 0 if compiled using F8
*s.end();
```

sparse-table.hpp

Description: Sparse table for minimum on the range $[l, r), l < r$. You can push back an element in $O(\text{LOG})$ and query anytime.

3cfa53, 19 lines

```
struct SparseTable
{
  VI t[LOG];

  void push_back(int v)
  {
    int i = SZ(t[0]);
    t[0].PB(v);
    FOR (j, 0, LOG - 1)
      t[j + 1].PB(min(t[j][i], t[j][max(0, i - (1 << j))]));
  }
  // [l, r)
  int query(int l, int r)
  {
    assert(l < r && r <= SZ(t[0]));
    int i = 31 - __builtin_clz(r - l);
    return min(t[i][r - 1], t[i][l + (1 << i) - 1]);
  }
};
```

convex-hull-trick.hpp

Description: add(a, b) adds a straight line $y = ax + b$. `getMaxY(p)` finds the maximum y at $x = p$.

bb0dd6, 72 lines

```
struct Line
{
  LL a, b, xLast;
  Line() {}
  Line(LL _a, LL _b): a(_a), b(_b) {}
  bool operator<(const Line& l) const
  {
    return MP(a, b) < MP(l.a, l.b);
  }
  bool operator<(int x) const
  {
    return xLast < x;
  }
  __int128 getY(__int128 x) const
  {
    return a * x + b;
  }
  LL intersect(const Line& l) const
  {
    assert(a < l.a);
    LL dA = l.a - a, dB = b - l.b, x = dB / dA;
    if (dB < 0 && dB % dA != 0)
      x--;
    return x;
  }
};

struct ConvexHull: set<Line, less<>>
{
  bool needErase(iterator it, const Line& l)
  {
    LL x = it->xLast;
    if (it->getY(x) > l.getY(x))
      return false;
    if (it == begin())
      return it->a >= l.a;
    x = prev(it)->xLast + 1;
    return it->getY(x) < l.getY(x);
  }
  void add(LL a, LL b)
  {
    Line l(a, b);
    auto it = lower_bound(l);
    if (it != end())
    {
      LL x = it == begin() ? -LINF :
          prev(it)->xLast;
      if ((it == begin()
        || prev(it)->getY(x) >= l.getY(x))
        && it->getY(x + 1) >= l.getY(x + 1))
        return;
    }
    while (it != end() && needErase(it, l))
      it = erase(it);
    while (it != begin() && needErase(prev(it), l))
      erase(prev(it));
    if (it != begin())
    {
      auto itP = prev(it);
      Line itL = *itP;
      itL.xLast = itP->intersect(l);
      erase(itP);
      insert(itL);
    }
    l.xLast = it == end() ? LINF : l.intersect(*it);
    insert(l);
  }
  LL getMaxY(LL p)
  {
    return lower_bound(p)->getY(p);
  }
};
```