International Institute Of Information Technology - Hyderabad

FLogic

Aditya, Amul, Arjo

ICPC Regionals 2021-22

October 2022

Contest (1)

.bashrc

3 lines

```
alias c='g++ -Wall -Wconversion -Wfatal-errors -g -std=c++14 \
  -fsanitize=undefined,address'
xmodmap -e 'clear lock' -e 'keycode 66=less greater' #caps = <>
```

template.cpp

17 lines

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

#define rep(i, a, b) for(int i = a; i < (b); ++i)
#define all(x) begin(x), end(x)
#define sz(x) (int)(x).size()
#define pb push_back
typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;

int main() {
  // freopen("sample.in", "r", stdin);
  // freopen("sample.out", "w", stdout);
  cin.tie(0)->sync_with_stdio(0);
  cin.exceptions(cin.failbit);
}
```

troubleshoot.txt

52 lines

```
Pre-submit:
Write a few simple test cases if sample is not enough.
Are time limits close? If so, generate max cases.
Is the memory usage fine?
Could anything overflow?
Make sure to submit the right file.

Wrong answer:
Print your solution! Print debug output, as well.
Are you clearing all data structures between test cases?
Can your algorithm handle the whole range of input?
Read the full problem statement again.
Do you handle all corner cases correctly?
Have you understood the problem correctly?
Any uninitialized variables?
Any overflows?
Confusing N and M, i and j, etc.?
Are you sure your algorithm works?
What special cases have you not thought of?
Are you sure the STL functions you use work as you think?
Add some assertions, maybe resubmit.
Create some testcases to run your algorithm on.
Go through the algorithm for a simple case.
Go through this list again.
Explain your algorithm to a teammate.
Ask the teammate to look at your code.
Go for a small walk, e.g. to the toilet.
Is your output format correct? (including whitespace)
Rewrite your solution from the start or let a teammate do it.

Runtime error:
Have you tested all corner cases locally?
Any uninitialized variables?
Are you reading or writing outside the range of any vector?
Any assertions that might fail?
Any possible division by 0? (mod 0 for example)
Any possible infinite recursion?
Invalidated pointers or iterators?
Are you using too much memory?
Debug with resubmits (e.g. remapped signals, see Various).

Time limit exceeded:
Do you have any possible infinite loops?
What is the complexity of your algorithm?
Are you copying a lot of unnecessary data? (References)
How big is the input and output? (consider scanf)
Avoid vector, map. (use arrays/unordered_map)
What do your teammates think about your algorithm?

Memory limit exceeded:
What is the max amount of memory your algorithm should need?
Are you clearing all data structures between test cases?
```

Data structures (2)

OrderedSet.h

Description: A set (not multiset!) with support for finding the n'th element, and finding the index of an element. To get a map, change `null_type`.
Time: $\mathcal{O}(\log N)$

782797, 16 lines

```
#include <bits/extc++.h>
using namespace __gnu_pbds;

template<class T>
using Tree = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

void example() {
  Tree<int> t, t2; t.insert(8);
  auto it = t.insert(10).first;
  assert(it == t.lower_bound(9));
  assert(t.order_of_key(10) == 1);
  assert(t.order_of_key(11) == 2);
  assert(*t.find_by_order(0) == 8);
  t.join(t2); // assuming T < T2 or T > T2, merge t2 into t
}
```

Dsu.h

Description: DSU with rollback
Time: $\mathcal{O}(\alpha(N))$

29a3e8, 18 lines

```
struct DSU {
  int sets; vi p, s;
  stack<pii> ss, sp;
  DSU(int n) : p(n, -1), s(n, 1), sets(n) {}
  bool IsSameSet(int a, int b) { return find(a) == find(b); }
  int find(int x) {return p[x] == -1 ? x : p[x] = find(p[x]);}
  void join(int a, int b) {
    a = find(a), b = find(b);
    if (a == b) return;
    if (s[a] < s[b]) swap(a, b);
    ss.push({a, s[a]}); sp.push({b, p[b]});
    sets--; s[a] += s[b]; p[b] = a;
  }
  void rollback() {
    p[sp.top().first] = sp.top().second; sp.pop();
    s[ss.top().first] = ss.top().second; ss.pop();
  }
};
```

DsuBp.h

Description: Graph, adding edges, checking bp color
Time: $\mathcal{O}(\alpha(N))$

8e325a, 18 lines

```
struct DSU {
  int sets; vi p, s, l;
  DSU(int n) : p(n, -1), s(n, 1), l(n, 0), sets(n) {}
  bool IsSameColor(int a,int b) {
    find(a); find(b); return l[a] == l[b];
  }
  bool IsSameSet(int a, int b) { return find(a) == find(b); }
  int find(int x) {
    if(p[x] == -1) return x;
    int y = find(p[x]); l[x] ^= l[p[x]]; return p[x] = y;
  }
  void join(int a, int b) {
    int ca = a, cb = b; a = find(a), b = find(b);
    if (a == b) return;
    if (s[a] < s[b]) swap(a, b);
    sets--; s[a] += s[b]; l[b] = 1 ^ l[ca] ^ l[cb]; p[b] = a;
  }
};
```

MinQueue.h

Description: DSU with rollback
Time: $\mathcal{O}(\alpha(N))$

68dd56, 24 lines

```
template<class T>
struct MinQueue {
  deque<pair<T, T>> q;
  int ca = 0, cr = 0, plus = 0, sze = 0;
  void push(T x) {
    x -= plus;
    // change '>' to '<' and you get max-queue
    while (!q.empty() && q.back().first > x)
      q.pop_back();
    q.push_back({x, ca}); ca++; sze++;
  }
  T pop() {
    T re = 0;
    if (!q.empty() && q.front().second == cr) {
      re = q.front().first; q.pop_front();
    }
    cr++; sze--; return re + plus;
  }
  // Returns minimum in the queue
  T min() { return q.front().first + plus; }
  int size() { return sze; }
  // Adds x to every element in the queue
  void add(int x) { plus += x; }
};
```

Matrix.h

Description: Basic operations on square matrices.
Usage: `Matrix<int, 3> A;`
`A.d = {{{{1,2,3}}, {{4,5,6}}, {{7,8,9}}}};`
`vector<int> vec = {1,2,3};`
`vec = (A^N) * vec;`

c43c7d, 26 lines

```
template<class T, int N> struct Matrix {
  typedef Matrix M;
  array<array<T, N>, N> d{};
  M operator*(const M& m) const {
    M a;
    rep(i,0,N) rep(j,0,N)
      rep(k,0,N) a.d[i][j] += d[i][k]*m.d[k][j];
    return a;
  }
  vector<T> operator*(const vector<T>& vec) const {
    vector<T> ret(N);
```

```
    rep(i,0,N) rep(j,0,N) ret[i] += d[i][j] * vec[j];
    return ret;
  }
  M operator^(ll p) const {
    assert(p >= 0);
    M a, b(*this);
    rep(i,0,N) a.d[i][i] = 1;
    while (p) {
      if (p&1) a = a*b;
      b = b*b;
      p >>= 1;
    }
    return a;
  }
};
```

SparseTable.h

Description: Range Minimum Queries on an array. Returns min(V[a], V[a + 1], ... V[b - 1]) in constant time.
Usage: `RMQ rmq(values);`
`rmq.query(inclusive, exclusive);`
Time: $\mathcal{O}(|V| \log |V| + Q)$

4a61f2, 21 lines

```
template<class T>
struct SparseTable {
    T (*op)(T, T);
    vi log2s; vector<vector<T>> st;
    SparseTable (const vector<T>& arr, T (*op)(T, T))
      : op(op), log2s(sz(arr)+1), st(sz(arr)) {
        rep(i,2,sz(log2s)) { log2s[i] = log2s[i/2] + 1; }
        rep(i,0,sz(arr)) {
          st[i].assign(log2s[sz(arr) - i] + 1);
          st[i][0] = arr[i];
        }
        rep(p, 1, log2s[sz(arr)] + 1) rep(i,0,sz(arr))
          if(i+(1<<p) <= sz(arr)) {
            st[i][p] = op(st[i][p-1], st[i+(1<<(p-1))][p-1]);
        }
    }
    T query (int l, int r) {
        int p = log2s[r-l+1];
        return op(st[l][p], st[r-(1<<p)+1][p]);
    }
};
```

FenwickTree.h

Description: Computes partial sums a[0] + a[1] + ... + a[pos - 1], and updates single elements a[i], taking the difference between the old and new value.
Time: Both operations are $\mathcal{O}(\log N)$.

e62fac, 22 lines

```
struct FT {
  vector<ll> s;
  FT(int n) : s(n) {}
  void update(int pos, ll dif) { // a[pos] += dif
    for (; pos < sz(s); pos |= pos + 1) s[pos] += dif;
  }
  ll query(int pos) { // sum of values in [0, pos)
    ll res = 0;
    for (; pos > 0; pos &= pos - 1) res += s[pos-1];
    return res;
  }
  int lower_bound(ll sum) {// min pos st sum of [0, pos] >= sum
    // Returns n if no sum is >= sum, or -1 if empty sum is.
    if (sum <= 0) return -1;
    int pos = 0;
    for (int pw = 1 << 25; pw; pw >>= 1) {
      if (pos + pw <= sz(s) && s[pos + pw-1] < sum)
        pos += pw, sum -= s[pos-1];
    }
    return pos;
  }
};
```

FenwickTree2D.h

Description: Computes sums a[i,j] for all i<I, j<J, and increases single elements a[i,j]. Requires that the elements to be updated are known in advance (call fakeUpdate() before init()).
Time: $\mathcal{O}\left(\log^2 N\right)$. (Use persistent segment trees for $\mathcal{O}(\log N)$.)

"FenwickTree.h" 157f07, 22 lines

```
struct FT2 {
  vector<vi> ys; vector<FT> ft;
  FT2(int limx) : ys(limx) {}
  void fakeUpdate(int x, int y) {
    for (; x < sz(ys); x |= x + 1) ys[x].push_back(y);
  }
  void init() {
    for (vi& v : ys) sort(all(v)), ft.emplace_back(sz(v));
  }
  int ind(int x, int y) {
    return (int)(lower_bound(all(ys[x]), y) - ys[x].begin()); }
  void update(int x, int y, ll dif) {
    for (; x < sz(ys); x |= x + 1)
      ft[x].update(ind(x, y), dif);
  }
  ll query(int x, int y) {
    ll sum = 0;
    for (; x; x &= x - 1)
      sum += ft[x-1].query(ind(x-1, y));
    return sum;
  }
};
```

SegmentTree.h

Description: RMQ SegTree
Time: $\mathcal{O}(\alpha(N))$

f19f05, 42 lines

```
const ll INF = 1e18;
struct node {
  ll x;
};

template<class T>
struct SegmentTrees {
  vector<node> st, lazy;
  node def;
  SegmentTrees(int n) : st(4*n, {INF}), lazy(4*n, {INF}), def({
       INF}) {}
  inline node combine(node a, node b) {
    node ret; ret.x = min(a.x, b.x); return ret;
  }
  void push(int pos) {
    if(lazy[pos].x != INF) {
      st[pos*2] = lazy[pos]; st[pos*2 + 1] = lazy[pos];
      lazy[pos*2] = lazy[pos]; lazy[pos*2+1] = lazy[pos];
      lazy[pos] = def;
    }
  }
  void update(int l,int r,T val,int left,int right,int pos=1) {
    if(l > r) return;
    if(l==left && r==right) {
      st[pos].x = val; lazy[pos] = {val};
    } else {
      push(pos);
      int mid = (left + right)/2;
      update(l, min(r,mid), val, left, mid, pos*2);
      update(max(l,mid+1), r, val, mid+1, right, pos*2+1);
      st[pos] = combine(st[pos*2], st[pos*2+1]);
    }
  }
  node query(int l,int r,int left,int right,int pos=1) {
    if(l>r) return def;
    if(l==left && r==right) return st[pos];
    else {
      push(pos); int mid = (left + right)/2;
      return combine(query(l, min(r,mid), left, mid, pos*2),
        query(max(l,mid+1), r, mid+1, right, pos*2+1));
    }
  }
};
```

Strings (3)

Hashing.h

Description: Various self-explanatory methods for string hashing. Use on Codeforces, which lacks 64-bit support and where solutions can be hacked.

<sys/time.h> 4b7e28, 36 lines

```
typedef uint64_t ull;
static int C; // initialized below

// Arithmetic mod two primes and 2^32 simultaneously.
// "typedef uint64_t H;" instead if Thue-Morse does not apply.
template<int M, class B>
struct A {
  int x; B b; A(int x=0) : x(x), b(x) {}
  A(int x, B b) : x(x), b(b) {}
  A operator+(A o){int y = x+o.x; return{y - (y>=M)*M, b+o.b};}
  A operator-(A o){int y = x-o.x; return{y + (y< 0)*M, b-o.b};}
  A operator*(A o) { return {(int)(1LL*x*o.x % M), b*o.b}; }
  explicit operator ull() { return x ^ (ull) b << 21; }
};
typedef A<1000000007, A<1000000009, unsigned>> H;

struct HashInterval {
  vector<H> ha, pw;
  HashInterval(string& str) : ha(SZ(str)+1), pw(ha) {
    pw[0] = 1;
    REP(i,0,SZ(str))
      ha[i+1] = ha[i] * C + str[i],
      pw[i+1] = pw[i] * C;
  }
  H hashInterval(int a, int b) { // hash [a, b)
    return ha[b] - ha[a] * pw[b - a];
  }
};

int main() {
  timeval tp;
  gettimeofday(&tp, 0);
  C = (int)tp.tv_usec; // (less than modulo)
  assert((ull)(H(1)*2+1-3) == 0);
  // ...
}
```

Kmp.h

Description: pi[x] computes the length of the longest prefix of s that ends at x, other than s[0...x] itself (abacaba -> 0010123). Can be used to find all occurrences of a string.
Time: $\mathcal{O}(n)$

d4375c, 16 lines

```
vi pi(const string& s) {
  vi p(sz(s));
  rep(i,1,sz(s)) {
    int g = p[i-1];
    while (g && s[i] != s[g]) g = p[g-1];
```

```
    p[i] = g + (s[i] == s[g]);
  }
  return p;
}

vi match(const string& s, const string& pat) {
  vi p = pi(pat + '\0' + s), res;
  rep(i,sz(p)-sz(s),sz(p))
    if (p[i] == sz(pat)) res.push_back(i - 2 * sz(pat));
  return res;
}
```

Manacher.h

Description: For each position in a string, computes p[0][i] = half length of longest even palindrome around pos i, p[1][i] = longest odd (half rounded down).
Time: $\mathcal{O}(N)$ e7ad79, 13 lines

```
array<vi, 2> manacher(const string& s) {
  int n = sz(s);
  array<vi,2> p = {vi(n+1), vi(n)};
  rep(z,0,2) for (int i=0,l=0,r=0; i < n; i++) {
    int t = r-i+!z;
    if (i<r) p[z][i] = min(t, p[z][l+t]);
    int L = i-p[z][i], R = i+p[z][i]-!z;
    while (L>=1 && R+1<n && s[L-1] == s[R+1])
      p[z][i]++, L--, R++;
    if (R>r) l=L, r=R;
  }
  return p;
}
```

MinRotation.h

Description: Finds the lexicographically smallest rotation of a string.
Usage: `rotate(v.begin(), v.begin()+minRotation(v), v.end());`
Time: $\mathcal{O}(N)$ d07a42, 8 lines

```
int minRotation(string s) {
  int a=0, N=sz(s); s += s;
  rep(b,0,N) rep(k,0,N) {
    if (a+k == b || s[a+k] < s[b+k]) {b += max(0, k-1); break;}
    if (s[a+k] > s[b+k]) { a = b; break; }
  }
  return a;
}
```

SuffixArray.h

Description: Builds suffix array for a string. `sa[i]` is the starting index of the suffix which is i'th in the sorted suffix array. The returned vector is of size $n+1$, and `sa[0] = n`. The `lcp` array contains longest common prefixes for neighbouring strings in the suffix array: `lcp[i] = lcp(sa[i], sa[i-1])`, `lcp[0] = 0`. The input string must not contain any zero bytes.
Time: $\mathcal{O}(n \log n)$ 38db9f, 23 lines

```
struct SuffixArray {
  vi sa, lcp;
  SuffixArray(string& s, int lim=256) { // or basic_string<int>
    int n = sz(s) + 1, k = 0, a, b;
    vi x(all(s)+1), y(n), ws(max(n, lim)), rank(n);
    sa = lcp = y, iota(all(sa), 0);
    for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim = p) {
      p = j, iota(all(y), n - j);
      rep(i,0,n) if (sa[i] >= j) y[p++] = sa[i] - j;
      fill(all(ws), 0);
      rep(i,0,n) ws[x[i]]++;
      rep(i,1,lim) ws[i] += ws[i - 1];
      for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
      swap(x, y), p = 1, x[sa[0]] = 0;
      rep(i,1,n) a = sa[i - 1], b = sa[i], x[b] =
        (y[a] == y[b] && y[a + j] == y[b + j]) ? p - 1 : p++;
    }
    rep(i,1,n) rank[sa[i]] = i;
    for (int i = 0, j; i < n - 1; lcp[rank[i++]] = k)
      for (k && k--, j = sa[rank[i] - 1];
          s[i + k] == s[j + k]; k++);
  }
};
```

Z.h

Description: z[x] computes the length of the longest common prefix of s[i:] and s, except z[0] = 0. (abacaba -> 0010301)
Time: $\mathcal{O}(n)$ ee09e2, 12 lines

```
vi Z(const string& S) {
  vi z(sz(S));
  int l = -1, r = -1;
  rep(i,1,sz(S)) {
    z[i] = i >= r ? 0 : min(r - i, z[i - l]);
    while (i + z[i] < sz(S) && S[i + z[i]] == S[z[i]])
      z[i]++;
    if (i + z[i] > r)
      l = i, r = i + z[i];
  }
  return z;
}
```