Team Notebook

February 12, 2024

Contents

1 01- Template

```cpp
#define _CRT_SECURE_NO_WARNINGS
#include <bits/stdc++.h>
#include <unordered_map>

#define ll long long
#define ld long double
#define pl pair<ll, ll>
#define vi vector<ll>
#define vii vector<vi>
#define vc vector<char>
#define vcc vector<vc>
#define vp vector<pl>
#define mi map<ll,ll>
#define mc map<char,int>
#define sortx(X) sort(X.begin(),X.end());
#define all(X) X.begin(),X.end()
#define ln '\n'
#define YES {cout << "YES\n"; return;}
#define NO {cout << "NO\n"; return;}

const int MODE = 1e9 + 7;

using namespace std;

void solve(int tc) {

}

int main()
{
    ios_base::sync_with_stdio(false), cin.tie(nullptr), cout.tie(nullptr);
    int size = 1;

    cin >> size;
    for (int i = 1; i <= size; i++)
        solve(i);
}
```

2 DataStructre

2.1 Graph

2.1.1 0-tmp

```cpp
class Graph {
public:
    vi vis;
    vii adj;

    void addEdge(int u, int v)
    {
        adj[u].push_back(v);
    }



    Graph(ll n) {
        vis.assign(n + 1, 0);
        adj.resize(n + 1);
    }
};
```

2.1.2 1-DFS

```cpp
void DFS(int s)
{
    stack<int> stack;

    stack.push(s);

    while (!stack.empty())

    {
        int m = stack.top();
        stack.pop();

        vis[m] = 1;

        for (auto a : adj[m]) {
            if (!vis[a]) {
                stack.push(a);
            }
        }
    }
}
```

2.1.3 Disjoint Set (dsu)

2.1.4 1- Construction

```cpp
/**
 * usage:-
```

```
 * creat dsu element.
 * DSU du;
 *
 * Functions you can use:
 * @val: get value of set.
 * @get: get the parent of element.
 * @add: marge two elements.
 * @build: build graph with given size.
 *
 * make sure to look at item typedef.
 * you can change marge function to change it's oppration.
*/

typedef long long item;
/*
struct item
{
 int val;

 item(){
  val = 0;
 }
};
*/

class DSU
{
public:

 item val(int n) {
  return(op[get(n)]);
 }

 int get(int n) {
  if (P[n] == n) return (n);
  return (P[n] = get(P[n]));
 }

 void add(int l, int r) {
  int a, b;
  a = get(l), b = get(r);

  if (a == b) return;

  if (R[a] == R[b])
   R[a]++;
  else if (R[a] < R[b])
   swap(a, b);

  P[b] = a;
  marge(op[a], op[b]);
 }

 void build(int n) {
  P.assign(n + 1, 0);
  R.assign(n + 1, 0);
  op.assign(n + 1, item());

  for (int i = 0; i <= n; i++)
   P[i] = i;
 }
private:
 vector<int> P, R;
 vector<item> op;

 void marge(item &a, item &b) {
  /*any oppration you want*/
 }
};
```

2.1.5 2- Kruskal's$_{M}inimum_{S}panning_{T}ree$

```
/**
 * -Sort all the edges in non-decreasing order of their weight.
 *
 * -Pick the smallest edge.
 * Check if it forms a cycle with the spanning tree formed so far.
 * If the cycle is not formed, include this edge. Else, discard it.
 *
 * -Repeat step#2 until there are (V-1) edges in the spanning tree.
*/

//Function to find sum of weights of edges of the Minimum Spanning Tree.
int spanningTree(int V, vector<vector<int>> adj[])
{
    int a, b, summ = 0;
    vector<vector<int>> X;

    //take all edges in array.
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < adj[i].size(); j++) {
            X.push_back(adj[i][j]);
            reverse(X.back().begin(), X.back().end());
            X.back().push_back(i);
        }
    }

    sort(X.begin(), X.end());
```

```
    DSU ds;
    ds.build(V);

    for (int i = 0; i < X.size(); i++)
    {
        a = ds.get(X[i][1]);
        b = ds.get(X[i][2]);

        if (a != b)
        {
            ds.add(a, b);
            summ += X[i][0];
        }
    }

    return (summ);
}
```

2.2 Trees

2.2.1 Segment$_{T}ree$

2.2.2 1- Construction

```
    /**
     * usage:-
     * creat tree element.
     * SegmentTree sg;
     *
     * Functions you can use:
     * @set: set index or range to value.
     * @geteange: get value of given range.
     * @build: build tree with given vector or size.
     *
     * make sure to look at item typedef.
     * you can change merge function to change it's oppration.
     * it you want to make change to segment work in checkLazy().
    */

    typedef long long item;
    /*
    struct item
    {
        int val;

        item(){
            val = 0;
        }
};
*/

class SegmentTree
{
public:

    void set(int index, int value) {
        set(0, 0, size - 1, index, value);
    }
    void set(int l, int r, int value) {
        set(0, 0, size - 1, l, r, value);
    }

    item getrange(int l, int r) {
        return (getrange(0, 0, size - 1, l, r));
    }

    void build(int n) {
        size = 1;
        while (size < n)
            size *= 2;
        tree.assign(size * 2, item());
        lazy.assign(size * 2, 0);
    }

    void build(vector<item>& X) {
        size = 1;
        while (size < X.size())
            size *= 2;
        tree.assign(size * 2, item());
        lazy.assign(size * 2, 0);

        build(X, 0, 0, size - 1);
    }

private:
    int size;
    vector<item> tree;
    vector<long long> lazy;

    item merge(item a, item b) {
        item res;
        return (res);
    }

    void checkLazy(int m, int lx, int rx) {
        if (!lazy[m]) return;
        tree[m] += lazy[m];
```

```
    if (lx != rx) {
        lazy[2 * m + 1] += lazy[m];
        lazy[2 * m + 2] += lazy[m];
    }

    lazy[m] = 0;
}

void set(int m, int lx, int rx, int pos, int val) {
    checkLazy(m, lx, rx);
    if (pos < lx || rx < pos) return;
    if (lx == rx && lx == pos)
    {
        tree[m] = val;
        return;
    }

    int mid = (lx + rx) / 2;
    item s1, s2;

    set(m * 2 + 1, lx, mid, pos, val);
    set(m * 2 + 2, mid + 1, rx, pos, val);
    s1 = tree[m * 2 + 1], s2 = tree[m * 2 + 2];

    tree[m] = merge(s1, s2);
}

void set(int m, int lx, int rx, int l, int r, int val) {
    checkLazy(m, lx, rx);
    if (rx < l || r < lx) return;
    if (l <= lx && rx <= r)
    {
        lazy[m] = val;
        checkLazy(m, lx, rx);
        return;
    }

    int mid = (lx + rx) / 2;
    item s1, s2;

    set(m * 2 + 1, lx, mid, l, r, val);
    set(m * 2 + 2, mid + 1, rx, l, r, val);
    s1 = tree[m * 2 + 1], s2 = tree[m * 2 + 2];

    tree[m] = merge(s1, s2);
}

item getrange(int m, int lx, int rx, int l, int r) {
    checkLazy(m, lx, rx);
    if (rx < l || r < lx) return (0);
    if (l <= lx && rx <= r) return (tree[m]);

    int mid = (lx + rx) / 2;
    item s1, s2;

    s1 = getrange(m * 2 + 1, lx, mid, l, r);
    s2 = getrange(m * 2 + 2, mid + 1, rx, l, r);

    return merge(s1, s2);
}

void build(vector<item>& X, int m, int lx, int rx) {
    if (lx == rx) {
        if (lx < X.size()) tree[m] = X[lx];
        return;
    }

    int mid = (lx + rx) / 2;
    item s1, s2;

    build(X, m * 2 + 1, lx, mid);
    build(X, m * 2 + 2, mid + 1, rx);
    s1 = tree[m * 2 + 1], s2 = tree[m * 2 + 2];

    tree[m] = merge(s1, s2);
}
};
```

2.2.3 2- get$_K th_e lement$

```
int get(int k) {
    return get(0, 0, size - 1, k);
}

int get(int m, int lx, int rx, int k) {
    if (lx == rx)
        return (lx);

    int mid = (lx + rx) / 2, s1, s2;
    s1 = tree[m * 2 + 1];
    s2 = tree[m * 2 + 2];

    if (s1 >= k)
        return (get(m * 2 + 1, lx, mid, k));
    return (get(m * 2 + 2, mid + 1, rx, k - s1));
}

/*I can use it to get first element greater than or equal k*/
int get(int m, int lx, int rx, int k) {
```

```
    if (tree[m] < k) return (-1);
    if (lx == rx)
        return (lx);

    int mid = (lx + rx) / 2, s1, s2;
    s1 = tree[m * 2 + 1];
    s2 = tree[m * 2 + 2];

    if (s1 >= k)
        return (get(m * 2 + 1, lx, mid, k));
    return (get(m * 2 + 2, mid + 1, rx, k));
}

/*I can use it to get first element from l to n greater than or equal k*/
ll get(int m, int lx, int rx, int k, int re) {
    if (tree[m] < k || rx < re)
        return (-1);
    if (lx == rx)
        return (lx);

    ll mid = (lx + rx) / 2, s1, s2;
    s1 = tree[m * 2 + 1];
    s2 = tree[m * 2 + 2];

    ll res = get(m * 2 + 1, lx, mid, k, re);

    if (res != -1)
        return (res);

    return (get(m * 2 + 2, mid + 1, rx, k, re));
}
```

2.2.4 3- work$_a t_h igh_r ange$

```
#include <iostream>
#include <vector>

using namespace std;

int n;

struct node {

 int sum, greater, begin, end, lazy, mid, qtd, on;

 vector<node> sons;

 node(int begin_ = 1, int end_ = n, int lazy_ = 0, int on_ = 0, int sum_ = 0, int greater_
     = 0) {
 sum = sum_, greater = greater_, begin = begin_, end = end_, lazy = lazy_, on = on_;

 mid = (begin + end) / 2, qtd = end - begin + 1;
}

void flush() {

 if (sons.empty() && end > begin) {

  sons.push_back(node(begin, mid));
  sons.push_back(node(mid + 1, end));
 }

 if (on) {

  if (end > begin) {

   sons[0].lazy = sons[1].lazy = lazy;
   sons[0].on = sons[1].on = on;
  }

  sum = greater = qtd * lazy;
 }

 lazy = on = 0;
}

int query(int h) {

 flush();

 if (begin == end) {

  if (greater > h) return begin - 1;
  return begin;
 }

 sons[0].flush();

 if (sons[0].greater > h) return sons[0].query(h);
 return sons[1].query(h - sons[0].sum);
}

void update(int a, int b, int d) {

 flush();

 if (a > end || b < begin) return;
```

```
  if (a <= begin && b >= end) {

   lazy = d;
   on = 1;
   flush();
   return;
  }

  sons[0].update(a, b, d);
  sons[1].update(a, b, d);

  sum = sons[0].sum + sons[1].sum;
  greater = max(sons[0].greater, sons[0].sum + sons[1].greater);
 }
};

int main() {

 ios_base::sync_with_stdio(false), cin.tie(nullptr), cout.tie(nullptr);

 cin >> n;

 node segtree;

 char op;

 while (cin >> op && op != 'E') {

  if (op == 'Q') {

   int h;
   cin >> h;

   cout << segtree.query(h) << '\n';
  }

  if (op == 'I') {

   int a, b, d;
   cin >> a >> b >> d;

   segtree.update(a, b, d);
  }
 }

 return 0;
}
```

2.2.5 TrieTree

2.2.6 2- int$_s earching$

```
/**
 * Search for int in it's binary represenaion.
 * it search for last digit.
 * you can change SIZE higher or lower range.
 *
*/

struct trie_node
{
    ll val;
    vector<int> next;

    trie_node() {
        val = -1;
        next.resize(2, -1);
    }
};

class Trie
{
public:
    ll get(ll n) {
        return (get(0, n, SIZE));
    }

    void add(ll n) {
        add(0, n, SIZE);
    }

    Trie() {
        tree.resize(1, trie_node());
    }
private:
    int SIZE = 32;
    vector<trie_node> tree;

    ll get(ll at, ll n, ll k) {
        if (k == -1) return (tree[at].val);
        int re = (n >> k) & 1;
        if (tree[at].next[re] == -1) return (-1);
        return get(tree[at].next[re], n, k - 1);
    }

    void add(ll at, ll n, ll k) {
        if (k == -1) {
            tree[at].val = 1;
```

```
            return;
        }
        int re = (n >> k) & 1;

        if (tree[at].next[re] == -1) {
            tree[at].next[re] = tree.size();
            tree.push_back(trie_node());
        }

        add(tree[at].next[re], n, k - 1);
    }
};
```

3 algoritms

3.1 DP

3.1.1 Longest$_i$ncreasing$_s$ubseq

```
ll Longest_Increasing_SubSeq(vi X) {
    vi Z;
    for (int i = 0; i < X.size(); i++) {
        ll re = upper_bound(Z.begin(), Z.end(), X[i]) - Z.begin();
        if (re == Z.size())Z.push_back(X[i]);
        else Z[re] = X[i];
    }
    return Z.size();
}
```

4 math

4.1 Combinatorics

4.1.1 combinatorics

```
const int SIZE = 1e6 + 1;
const int MODE = 998244353;
vi fac(SIZE, 1);

ll gcdExtended(ll a, ll b, ll* x, ll* y)
{
    if (a == 0) {
        *x = 0, *y = 1;
        return b;
    }
    ll x1, y1;
    ll gcd = gcdExtended(b % a, a, &x1, &y1);
    *x = y1 - (b / a) * x1;
    *y = x1;
    return gcd;
}
ll modeenv(ll n) {
    ll x, y;
    gcdExtended(n, MODE, &x, &y);
    return (x + MODE) % MODE;
}

// nCr = fac(n)/fec(r)*fac(n-r)
ll nCr(ll n, ll r) {
    ll res = fac[n];
    res *= modeenv((fac[r] * fac[n - r]) % MODE);
    return (res) % MODE;
}

//nPr = fac(n) / fac(n - r)
ll nPr(ll n, ll r) {
    ll res = fac[n];
    res *= modeenv(fac[n - r]);
    return (res) % MODE;
}

void INIT() {
    for (int i = 2; i < SIZE; i++)
        fac[i] = (i * fac[i - 1]) % MODE;
}
```