Competitive programming Notebook 🎈

Pablo Arruda Araujo

Sumário

1 Ds

1.1 sparse-table

```
// Sparse-Table
// O(log n)
const int logn = 22; // max log

int logv[MAX];
// Pre comp log values
void make_log(){
    logv[1] = 0;
    for(int i = 2; i <= MAX; i++)
        logv[i] = logv[i/2]+1;
}

struct Sparse {
    vector<vector<int> > st;

    Sparse(vector<int>& v) {
        int n = v.size();
        st.assign(n, vector<int>(logn, 0));
        // Unitary values st[i][0] = v[i, i+2^0] = v[i]
        for(int i = 0; i < n; i++){
            st[i][0] = v[i];
        }
        // Constructing Sparse Table in O(log n)
        for(int k = 1; k < logn; k++){
            for(int i = 0; i < n; i++){
                if(i + (1 << k)-1 >= n)
                    continue;
                int prox = i + (1 << (k-1));
                st[i][k] = min(st[i][k-1], st[prox][k-1]);
            }
        }
    }

    int f(int a, int b){
        // Can be: min, max, gcd
        // f must have idempotent property
        return min(a, b);
    }
    // Queries in O(1)
    int query(int l, int r){
        int size = r-l+1;
        int k = logv[size];
        // cat jump for queries in O(1)
        int res = f(st[l][k], st[r - ((1 << k)-1)][k]);
        return res;
    }
};
```

1.2 DSU

```
// Disjoint union set
// Operation ~ O(1)
int r[MAXN];
vector<int> qtd(MAXN, 1);

int get(int x) {
  return p[x] = (p[x] == x ? x : get(p[x]));
}

void unite(int a, int b){
  a = get(a);
  b = get(b);

  if(r[a] == r[b]){
    p[a] = b;
    r[b]++;
    qtd[b]+=qtd[a];
  }else if(r[a] > r[b]){
    p[b] = a;
    qtd[a]+=qtd[b];
  }else{
    p[a] = b;
    qtd[b]+=qtd[a];
  }
}

// Initializing values in main()
for(int i = 1; i <= n; i++) p[i]=i;
```

1.3 prefix-sum-array

```
// Preffix sum 1D
// O(n)
int v[MAXN];
int psum[MAXN];

int create_psum(){
    int acc = 0;
    for(int i = 0; i < v.size(); i++){
        acc+=v[i];
        psum[i] = acc;
    }
}

int query(int l, int r){
    return l == 0 ? psum[r] : psum[r]-psum[l-1];
}
```

1.4 delta-encoding

```
// Delta encoding
// O (n)

for(int i = 0; i < queries; i++){
    int l, r, x;
    cin >> l >> r >> x;
    delta[l]+=x;
    delta[r+1]-=x;
}
int acc = 0;
for(int i = 0; i < v.size(); i++){
    acc+=delta[i];
    v[i]+=acc;
}
```

1.5 Segtree

```
// Segtree MAX
// O(log n) operations

// DESCRIPTION:
// sti: id do nodo que estamos na segment tree
// stl: limite inferior do intervalo que aquele nodo representa(inclusivo)
// str: limite superior do intervalo que aquele nodo representa(inclusivo)
// l : limite inferior do intervalo que queremos fazer a consulta
// r : limite superior do intervalo que queremos fazer a consulta
// i : indice do vetor que queremos atualizar
// amm: novo valor daquele indice no vetor

class SegTree{
    vector<int> st;
    vector<int> lazy;
    vector<bool> has;
int size;

int el_neutro = -(1e9 + 7);

int f(int a, int b){
    return max(a,b);
}

void propagate(int sti, int stl, int str){
    if(has[sti]){
        st[sti] = lazy[sti]*(str-stl+1);
        if(stl!=str){
            lazy[sti*2+1] = lazy[sti];
            lazy[sti*2+2] = lazy[sti];

            has[sti*2+1] = true;
            has[sti*2+2] = true;
        }
        has[sti] = false;
    }
}

int query(int sti, int stl, int str, int l, int r){
    if(str < l || stl > r) return el_neutro;

    if(stl >= l && str <= r)
        return st[sti];

    // intervalo parcialmente incluido em l-r
    int mid = (stl+str)/2;

    return f(query(2*sti+1, stl, mid, l, r), query(2*sti+2, mid+1, str, l, r));
}

void update(int sti, int stl, int str, int i, int amm){
    if(stl == i && str == i){
        st[sti] += amm;
        return;
    }

    if(stl > i || str < i) return;

    int mid = (stl+str)/2;

    // Processo de atualizacao dos nos filhos
    update(sti*2+1, stl, mid, i, amm);
    update(sti*2+2, mid+1, str, i, amm);

    st[sti] = f(st[sti*2+1], st[sti*2+2]);
}

void update_range(int sti, int stl, int str, int l, int r, int amm){
    if(stl >= l && str <= r){
        lazy[sti] = amm;
        has[sti] = true;
        propagate(sti, stl, str);
        return;
    }

    if(stl > r || str < l) return;

    int mid = (stl+str)/2;
    update_range(sti*2+1, stl, mid, l, r, amm);
    update_range(sti*2+2, mid+1, str, l, r, amm);

    st[sti] = f(st[sti*2+1],st[sti*2+2]);
}

public:
        SegTree(int n): st(4*n, 0){size=n;}
        int query(int l, int r){return query(0,0,size-1,l,r);}
        void update(int i, int amm){update(0,0,size-1,i,amm);}
        void update_range(int l, int r, int amm){update_range(0,0,size-1,l,r,amm);}
};

// In main()

SegTree st(v.size());

for(int i = 0; i < n; i++){
    st.update(i, v[i]);
}
```

2 Graph

2.1 Dijkstra

```
// Dijkstra
// O(n + m log m)
#define INF 1e9+10
vector<pair<int, int>> adj[MAXN];
vector<int> dist;
vector<bool> visited;
priority_queue<pair<int,int>> q;

void Dijkstra(int n, int start){
    for(int i = 0; i <= n; i++){
        dist.push_back(INF);
        visited.push_back(false);
    }
    dist[start] = 0;
    q.push(make_pair(0, start));
    while(!q.empty()){
        int a = q.top().second; q.pop();
        if(visited[a]) continue;
        visited[a] = true;
        for(auto u : adj[a]){
            int b = u.first, w = u.second;
            if(dist[a]+w < dist[b]){
                dist[b] = dist[a]+w;
                q.push({-dist[b], b});
            }
        }
    }
}
```

2.2 DSU-MST

```
// Minimum Spanning tree
// w/ DSU structure

typedef struct{
    int a, b;
    int w;
} edge;

/* ----- DSU Structure ------*/
int get(int x) {
  return p[x] = (p[x] == x ? x : get(p[x]));
}

void unite(int a, int b){
  a = get(a);
  b = get(b);

  if(r[a] == r[b]) r[a]++;
  if(r[a] > r[b]) p[b] = a;
  else p[a] = b;
}

// Initializing values in main()
for(int i = 1; i <= n; i++) p[i]=i;

/* -------------------------*/

vector<edge> edges;
int total_weight;

void mst(){
    // sort edges
    for(auto e : edges){
        if(get(e.a) != get(e.b)){
            unite(e.a, e.b);
            total_weight+=e.w;
        }
    }
}
```

2.3 BFS

```
// BFS
// O(n+m)
vector<vector<int> > g(MAX_NODES);
vector<bool> visited(MAX_NODES);
vector<int> dist(MAX_NODES, oo);
queue<int> q;

void bfs(int s){
    q.push(s);
    dist[s] = 0;
    visited[s] = true;

    while(!q.empty()){
        int u = q.front(); q.pop();

        for(auto v : g[u]){
            if(not visited[v]){
                dist[v] = dist[u]+1;
                visited[v] = true;
                q.push(v);
            }
        }
    }
}
```

2.4 DFS

```
// DFS
// O(n+m)
vector<vector<int> > graph(MAX_NODES);
vector<bool> visited(MAX_NODES);

void dfs(int s){
    if(visited[s]) return;
    visited[s] = true;
    for(auto v : graph[s]){
        dfs(v);
    }
}
```

2.5 Warshall

```
// Floyd - Warshall
// O(n^3)
#define INF 1e9+10

int adj[MAXN][MAXN];
int distances[MAXN][MAXN];

void Warshall(int n, int start){
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            if (i == j) distances[i][j] = 0;
            else if (adj[i][j]) distances[i][j] = adj[i][j];
            else distances[i][j] = INF;
        }
    }
    for (int z = 1; z <= n; z++) {
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                distances[i][j] = min(distances[i][j],distances[i][z] + distances[z][j]);
            }
        }
    }
}
```

3 Algorithm

3.1 merge-sort

```
// Merge Sort
// O(n log n)
void merge_sort(vector<int>& v){
    if(v.size() == 1) return;

    vector<int> l, r;

    for(int i = 0; i < v.size()/2; i++)
        l.push_back(v[i]);
    for(int i = v.size()/2; i < v.size(); i++)
        r.push_back(v[i]);

    merge_sort(l);
    merge_sort(r);

    l.push_back(INF);
    r.push_back(INF);

    int inil = 0, inir = 0;

    for(int i = 0; i < v.size(); i++){
        if(l[inil] < r[inir]) v[i] = l[inil++];
        else v[i] = r[inir++];
    }

    return;
}
```

3.2 bsearch-iterative

```
// Binary search in iterative questions
// O(log n)
bool query(int mid, int x){
    cout << mid << endl;
    cout.flush();

    int ans;
    cin >> ans;
    return ans == x;
}

int solve(int x){
    int l = 1, r = n;
    int res = -1;

    while(l <= r){
        int mid = (l+r)/2;
        if(query(mid, x)){
            res = mid;
            l = mid+1;
        }else{
            r = m-1;
        }
    }

    return res;
}
```

3.3 counting-inversions

```
// Counting inversions in Array
// O(n log n)
int merge_sort(vector<int>& v){
    if(v.size() == 1) return 0;

    vector<int> l, r;

    for(int i = 0; i < v.size()/2; i++)
        l.push_back(v[i]);
    for(int i = v.size()/2; i < v.size(); i++)
        r.push_back(v[i]);
    int ans = 0;
    ans += merge_sort(l);
    ans += merge_sort(r);

    l.push_back(1e9);
    r.push_back(1e9);

    int inil = 0, inir = 0;

    for(int i = 0; i < v.size(); i++){
        if(l[inil] <= r[inir]) v[i] = l[inil++];
        else{
            v[i] = r[inir++];
            ans+=l.size()-inil-1;
        }
    }

    return ans;
}
```

3.4 kadane

```
// Maximum possible sum in Array
// O(n)
int array[MAXN];

int kadane(){
    int sum = 0, best = 0;
    for(int i = 0; i < n; i++){
        sum = max(array[i], sum+array[i]);
        best = max(sum, best);
    }

    return best;
}
```

4 Math

4.1 floor-log

```
// Find floor(log(x))
// O(n)
int logv[MAXN];
void make_log(){
    logv[1] = 0;
    for(int i = 2; i <= MAXN; i++)
        logv[i] = logv[i/2]+1;
}
```

4.2 fast-exponentiation

```
// Fast Exponentiation
// O(log n)
ll fexp(ll b, ll e){
    if(e == 0){
        return 1;
    }
    ll resp = fexp(b, e/2)%MOD;
    resp = (resp*resp)%MOD;
    if(e%2) resp = (b*resp)%MOD;

    return resp;
}
```

4.3 matrix-exponentiation

```
// Matrix Exponentiation
// O(log n)
#define ll long long int
#define vl vector<ll>
struct Matrix {
    vector<vl> m;
    int r, c;

    Matrix(vector<vl> mat) {
        m = mat;
        r = mat.size();
        c = mat[0].size();
    }

    Matrix(int row, int col, bool ident=false) {
        r = row; c = col;
        m = vector<vl>(r, vl(c, 0));
        if(ident)
            for(int i = 0; i < min(r, c); i++)
                m[i][i] = 1;
    }

    Matrix operator*(const Matrix &o) const {
        assert(c == o.r); // garantir que da pra multiplicar
        vector<vl> res(r, vl(o.c, 0));

        for(int i = 0; i < r; i++)
            for(int j = 0; j < o.c; j++)
                for(int k = 0; k < c; k++)
                    res[i][j] = (res[i][j] + m[i][k]*o.m[k][j]) % 1000000007;

        return Matrix(res);
    }

    void printMatrix(){
        for(int i = 0; i < r; i++)
            for(int j = 0; j < c; j++)
                cout << m[i][j] << " \n"[j == (c-1)];
    }
};

Matrix fexp(Matrix b, ll e, int n) {
    if(e == 0) return Matrix(n, n, true); // identidade
    Matrix res = fexp(b, e/2LL, n);
    res = (res * res);
    if(e%2) res = (res * b);

    return res;
}

// Fibonacci Example O(log n)
/*  Fibonacci
    |1 1|*|Fn  | = |Fn+1|
    |1 0| |Fn-1|   |Fn  |

    Generic
    |a1 a2 ... an| ** K *  |Fn-1| = |Fk+n-1|
    |1 0   ...  0|         |Fn-2|   |Fk+n-2|
    |0 1 0 ...  0|         |Fn-3|   |Fk+n-3|
           ...              ...      ...
    |0 0 0 ...1 0|         |F0  |   |Fk    |
*/

int main() {
    ll n;
    cin >> n; // Fibonacci(n)

    if(n == 0) {
        cout << 0 << endl;
        return 0;
    }

    vector<vl> m = {{1LL, 1LL}, {1LL, 0LL}};
    vector<vl> b = {{1LL}, {0LL}};

    Matrix mat = Matrix(m);
    Matrix base = Matrix(b);

    mat = fexp(mat, n-1, 2);
    mat = mat*base;

    cout << mat.m[0][0] << endl;


    return 0;
}
```

5 Dp

5.1 knapsack

```
// Knapsack problem
// O(n.w)
int valor[MAXN], peso[MAXN], memo[MAXN];

ll solve(int i, int w){ // Recursive version
    if(i <= 0 || w <= 0) return 0;
    if(memo[i][w] != -1) return memo[i][w];
    ll pegar=-1e9;

    if(peso[i] <= w){
        pegar = solve(i-1,w-peso[i])+valor[i];
    }

    ll naopegar = solve(i-1,w);

    memo[i][w] = max(pegar,naopegar);

    return memo[i][w];
}

int dp[MAXN][MAXN], valor[MAXN], peso[MAXN];
int solve(int n, w){  // Iterative version
// n objects | max weight
    for(int i = 0; i <= n; i++)
        for(int j=0; j <= w;j++)
            dp[i][j] = 0;

    for(int i = 0; i <= n; i++){
        for(int j = 0; j <= w; j++){
            if(i == 0 || j == 0) return dp[i][j];
            else if(peso[i-1] <= j)
                dp[i][j] = max(dp[i-1][j-peso[i-1]]+valor[i-1],dp[i-1][j]);
            else
                dp[i][j] = dp[i-1][j];
        }
    }
    return dp[n][w];
}

int val[MAX], wt[MAX], dp[MAX]; // Optimization for space
int solve(int n, int W){
    for(int i=0; i < n; i++)
        for(int j=W; j>=wt[i]; j--)
            dp[j] = max(dp[j],dp[j-wt[i]]+val[i]);
    return dp[W];
}
```

5.2 LCS

```
// LCS  maior subs comum
// ** usar s[1 - n]
#define MAXN 1010

int s1[MAXN], s2[MAXN], tab[MAXN][MAXN];

int lcs(int a, int b){

    if(a == 0 || b == 0) return tab[a][b] = 0;

    if(tab[a][b] != -1) return tab[a][b];

    if(s1[a] == s2[b]) return lcs(a-1,b-1)+1;

    return tab[a][b] = max(lcs(a-1, b), lcs(a, b-1));
}
```

5.3 coin-change

```
// You have n coins {c1, ..., cn}
// Find min quantity of coins to sum K
// O(n.c)
int dp(int acc){ // Recursive version
    if(acc < 0) return oo;
    if(acc == 0) return 0;

    if(memo[acc] != -1) return memo[acc];

    int best = oo;

    for(auto c : coins){
        best = min(best, dp(acc-c)+1);
    }

    return memo[acc] = best;
}

int dp(){ // Iterative version
    memo[0] = 0
    for(int i = 1; i <= n; i++){
        memo[i] = oo;
        for(auto c : coins){
            if(i-c >= 0)
                memo[i] = min(memo[i], memo[i-c]+1);
        }
    }
}
```

5.4 unbouded-knapsack

```
// Knapsack (unlimited objects)
// O(n.w)

int w, n;
int c[MAXN], v[MAXN], dp[MAXN];

int unbounded_knapsack(){

    for(int i=0;i<=w;i++)
        for(int j=0;j<n;j++)
            if(c[j] <= i)
                dp[i] = max(dp[i], dp[i-c[j]] + v[j]);

    return dp[w];
}
```