Ex-usuários de Python — CIn - UFPE

Contents

1 Graphs

1.1 Breadth First Search

```cpp
// Time Complexity: O(V + E)
void bfs(vector<vector<int>>& adj, vector<bool>& visited, int start) {
    queue<int> operation_order;

    visited[start] = true;
    operation_order.push(start);

    while (!operation_order.empty()) {
        auto top = operation_order.front();
        operation_order.pop();

        for (auto e : adj[top]) {
            if (!visited[e]) {
                visited[e] = true;
                operation_order.push(e);
            }
        }
    }
}
```

1.2 Depth First Search

```cpp
// Time Complexity: O(V + E)
void dfs(vector<vector<int>>& adj, vector<bool>& visited, int v) {
    visited[v] = true;
    // pre-visited

    for (auto e: adj[v]) {
        if (!visited[e]) {
            dfs(adj, visited, e);
        }
    }

    // post-visited
}
```

1.3 TopoSort

```cpp
// Time Complexity: O(V + E)
void toposort(vector<vector<int>>& adj, stack<int>& topo, vector<bool>&
     visited, int v) {
    visited[v] = true;
    for (auto e: adj[v]) {
        if (!visited[e]) {
            toposort(adj, topo, visited, e);
        }
    }
    topo.push(v);
}

// Time Complexity: O(V + E)
void toposort(vector<vector<int>>& adj, vector<int>& indegree, int n) {
    queue<int> q; // Use a min heap for lexicographically smallest toposort
    for (int i = 0; i < n; i++) {
        if (indegree[i] == 0) {
            q.push(i);
        }
    }

    while (!q.empty()) {
        int v = q.front();
        q.pop();
        cout << v << " ";
        for (auto e: adj[v]) {
            indegree[e]--;
            if (indegree[e] == 0) {
                q.push(e);
            }
        }
    }
}
```

1.4 Is Bicolorable

```cpp
// Time Complexity: O(V + E)
bool bicolorable(vector<vector<int>>&adj, vector<bool>& visited, vector<
     bool>& color, int v) {
    visited[v] = true;

    for (auto e: adj[v]) {
        if (!visited[e]) {
            color[e] = !color[v];
            if (!bicolorable(adj, visited, color, e)) {
                return false;
            }
        } else if (color[e] == color[v]) {
            return false;
        }
    }

    return true;
}
```

1.5 Dijkstra

```cpp
// Time Complexity: O((V + E) * log(V))
void dijkstra(vector<vector<pair<int, int>>>& adj, vector<int>& dist, int s
     ) {
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int
         , int>>> pq;
    pq.push({0, s});
```

```cpp
    dist[s] = 0;

    while (!pq.empty()) {
        int u = pq.top().second;
        pq.pop();

        for (auto e: adj[u]) {
            int v = e.first;
            int w = e.second;

            if (dist[v] > dist[u] + w) {
                dist[v] = dist[u] + w;
                pq.push({dist[v], v});
            }
        }
    }
}
```

1.6 Floyd Warshall

```cpp
// Time Complexity: O(V^3)
void FloydWashall(vector<vector<int>>& dist, int n) {
    for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
            }
        }
    }
}
```

2 Number Theory

2.1 Digit Sum

```cpp
int digit_sum(int n) {
    while(n>=10) {
            int temp = 0;
            while (n > 0) {
                temp += n % 10;
                n /= 10;
            }
            n = temp;
        }
    return n;
}
```

2.2 Binary Search

```cpp
// Time Complexity: O(log(n))
int binarySearch(int l, int r, int* arr, int target) {
    int answ;
    while (l <= r) {
        int m = l + (r - l) / 2;

        // If NOT SOLVE, ignore left half
        if (! (arr[m] > target) )
            l = m + 1;

        // If SOLVE, ignore right half
        else {
            answ = m;
            r = m - 1;
            }
        }

        return answ-1;
    }
```

2.3 Fast Exponentiation

```cpp
const ll MOD = 1e9+7;

class Matrix{
        public:
        vector<vector<ll>> mat;
        int m;
        Matrix(int m): m(m){
                mat.resize(m);
                for(int i = 0; i < m; i++) mat[i].resize(m,0);
        }
        Matrix operator * (const Matrix& rhs){
                Matrix ans = Matrix(m);
                for(int i = 0; i < m; i++)
                        for(int j = 0; j < m; j++)
                                for(int k = 0; k < m; k++)
                                        ans.mat[i][j] = (ans.mat[i][j] + (
                                            mat[i][k] * rhs.mat[k][j]) %
                                            MOD) % MOD;
                return ans;
        }
};

Matrix fexp(Matrix a, ll n){
        int m = a.m;
        Matrix ans = Matrix(m);
        for(int i = 0; i < m; i++) ans.mat[i][i] = 1;
        while(n){
                if(n & 1) ans = ans * a;
                a = a * a;
                n >>= 1;
        }
        return ans;
}

// Time complexity: O(log(n))
ll fexpll(ll a, ll n){
        ll ans = 1;
        while(n){
                if(n & 1) ans = (ans * a) % MOD;
                a = (a * a) % MOD;
                n >>= 1;
        }
        return ans;
}
```

2.4 GCD and LCM

```cpp
// Time Complexity: O(log(min(m, n)))
ll gcd(ll a, ll b) { return b ? gcd(b, a % b) : a; }

// Time Complexity: O(log(min(m, n)))
ll lcm(ll a, ll b) { return a / gcd(a, b) * b; }
```

2.5 Sieve of Eratosthenes

```cpp
vector<ll> prime_list;
```

```cpp
// Time Complexity: O(n log log n)
void EratosthenesSieve(ll n) {
    vector<bool> prime(n + 1, true);

    for (ll p = 2; p <= n; p++) {
        if (prime[p] == true) {
            prime_list.push_back(p);

            for (ll i = p; i <= n; i += p)
                prime[i] = false;
        }
    }
}
```

3 Data Structures

3.1 Segment Tree

```cpp
const int INF = INT_MAX;
const int max_size = 2e5 + 5;
vector<ll> seg(4 * max_size);
vector<ll> arr(max_size);

int n, q;

ll operation(ll a, ll b) { return a + b; }

// Time complexity: O(n)                  // build()
void build(int l = 0, int r = n - 1, int index = 0) {
    if (l == r) {
        seg[index] = arr[l];
        return;
    }
    int mid = l + (r - l) / 2;
    int left = 2 * index + 1;
    int right = 2 * index + 2;
    build(l, mid, left);
    build(mid + 1, r, right);
    seg[index] = operation(seg[left], seg[right]);
}

// Time complexity: O(log(n))                  // query(L-1, R-1)
ll query(int L, int R, int l = 0, int r = n - 1, int index = 0) {
    if (R < l || L > r) return 0;  // Neutral element of the operation
    if (L <= l && r <= R) return seg[index];

    int mid = l + (r - l) / 2;
    int left = 2 * index + 1;
    int right = 2 * index + 2;
    ll ql = query(L, R, l, mid, left);
    ll qr = query(L, R, mid + 1, r, right);
    return operation(ql, qr);
}

// Time complexity: O(log(n))                  // update(pos-1, value)
void update(int pos, int num, int l = 0, int r = n - 1, int index = 0) {
    if (l == r) {
        seg[index] = num;
        return;
    }
    int mid = l + (r - l) / 2;
    int left = 2 * index + 1;
    int right = 2 * index + 2;
    if (pos <= mid) {
        update(pos, num, l, mid, left);
    } else
        update(pos, num, mid + 1, r, right);
    seg[index] = operation(seg[left], seg[right]);
}
```

3.2 Binary Indexed Tree (BIT)

```cpp
const int max_size = 2e5+5;
vector<ll> arr(max_size+1,0);
vector<ll> bit(max_size+1,0);

int n, q;

// Time complexity: O(log(n))
ll query(int i){ // [1,i]
        ll ret = 0;
        for(; i > 0; i -= i & -i){
                ret += bit[i];
        }
        return ret;
}

// Time complexity: O(log(n))
ll queryRange(int l, int r){ // [l,r]
        ll qr = query(r);
        ll ql = query(l-1);
        return qr-ql;
}

// Time complexity: O(log(n))
void increment(ll index, ll value){
        for(; index <= n; index += index & -index){
                bit[index] += value;
        }
}

// Time complexity: O(n * log(n))
void build(const vector<ll>& nums){
        for(int i = 0; i < nums.size(); i++){
                increment(i+1,nums[i]);
        }
}
```