ICPC Library | Ufes

Contents

1 Divide and Conquer

1.1 Bisection Method

```
// Bisection Method
// Very useful for finding roots of a function

// F(x)
// ^     F(lo)
// |   *
// |   | *
// |   |  *
// |   |   *  Goal
// |--------O-------------------------> x
// |         *              |
// |           *            *
// |             *       *   F(hi)
// |               * * *

double bisection(double lo, double hi) {
    for (int i = 0; i < 100; i++) {
        double mid = (lo + hi) / 2;
        double F   = f(mid);  // Declare a function
        if (F > 0)
            lo = mid;
        else
            hi = mid;
    }
    return lo;
}
```

1.2 Ternary Search

```
// Ternary search
// Very useful for finding max/min values between interval

// F(x)
// ^                 Goal
// |                o
// |             *     *
// |           *         *
// |         *             *
// |--------*----------------*----------> x
// |   |   *                   *  |
// |   | *                        * F(r)
// |   * F(l)

double ternary_search(double l, double r) {
    double eps = 1e-9;
    while (r - l > eps) {
        double m1 = l + (r - l) / 3;
        double m2 = r - (r - l) / 3;
        double f1 = f(m1);
        double f2 = f(m2);
        if (f1 < f2)
            l = m1;
        else
            r = m2;
    }
    return f(l);  // Return the maximum of f(x) in [l, r]
}
```

2 Sorting

2.1 Merge Sort

```
// Merge sort with inversion counter

int merge(int *arr, int *aux, int lo, int hi, int mid) {
    int inv = 0;
    for (int k = lo; k <= hi; k++) aux[k] = arr[k];
    int i = lo;
    int j = mid + 1;
    for (int k = lo; k <= hi; k++) {
        if (i > mid)
            arr[k] = aux[j++];
        else if (j > hi)
            arr[k] = aux[i++];
        else if (aux[j] < aux[i]) {
            arr[k] = aux[j++];
            inv += mid + 1 - i;
        } else
            arr[k] = aux[i++];
    }
    return inv;
}

int mergesort(int *arr, int *aux, int lo, int hi) {
    int inv = 0;
    if (lo >= hi) return inv;
    int mid = lo + (hi - lo) / 2;
    inv += mergesort(arr, aux, lo, mid);
    inv += mergesort(arr, aux, mid + 1, hi);
    inv += merge(arr, aux, lo, hi, mid);
    return inv;
}
```

3 Graph Algorithms

3.1 DFS

```
// Depth first search

int        V;
vector<vi> adj;
bool       vis[VMAX];
vi         topsort;  // Topological Sort.
                     // Only works in directed acyclic graph.

void dfs(int s) {
    vis[s] = true;
    for (auto a : adj[s]) {
        if (!vis[a]) {
            dfs(a);
        }
    }
    topsort.push_back(s);  // Only works in DAG.
}
```