ICPC Library | Ufes

Contents

1 C++

1.1 Template

```
#include <bits/stdc++.h>
using namespace std;
#define fi first
#define se second
#define forn(i, n) for (int i = 0; i < (int)n; ++i)
#define for 1(i, n) for (int i = 1; i \le (int)n; ++i)
#define fore(i, l, r) for (int i = (int)l; i \le (int)r; ++i)
#define ford(i, n) for (int i = (int)(n) - 1; i \ge 0; --i)
#define fored(i, l, r) for (int i = (int)r; i \ge (int)l; --i)
#define pb push_back
#define pf push_front
#define el '\n'
#define d(x) cout << #x << " " << x << el
#define ri(n) scanf("%d". &n)
#define sz(v) int(v.size())
#define all(v) v.begin(), v.end()
#define mset(x, y) memset(x, (y), sizeof(x));
typedef long long
typedef unsigned long long
                             ull;
typedef long double
                             ld:
typedef pair<int, int>
                             pii:
typedef pair<ll, ll>
                             pll;
typedef tuple<int, int, int> iii;
typedef vector<int>
                             vi;
typedef vector<pii>
                             vii;
typedef vector<ll>
                             vll;
typedef vector<ld>
                             vd;
const int INF = 0x3f3f3f3f;
const ull INFLL = 0x3f3f3f3f3f3f3f3f3f3f;
const int MAX = 1e5 + 200;
const ld PI = acos(-1);
const ld EPS = 1e-9:
int dr[] = \{1, -1, 0, 0, 1, -1, -1, 1\};
int dc[] = \{0, 0, 1, -1, 1, 1, -1, -1\};
ostream& operator<<(ostream& os, const pii& pa) {
    return os << "(" << pa.fi << ", " << pa.se << ")";
int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);
    cout << setprecision(20) << fixed;</pre>
```

```
return 0;
```

1.2 Complexity

Complexidade

```
Pior Algoritmo Aceito | ex.
<= 10
              0(n!), 0(n^6)
                                  Permutacao
<= 20
              0(2^n), 0(n^5)
                                  DP + Bitmask
<= 50
                  0(n^4)
                                  DP com 3d + O(n) loop
<= 100
                  0(n^3)
                                  Floyd Warshall's
<= 1K
                  0(n^2)
                                  Bubble/Selection/Insertion Sort
<= 100K
               0(nlog(n))
                                  Merge Sort, building Segment Tree
\leq 1M | O(n), O(log(n)), O(1) | Busca binaria
```

2 Divide and Conquer

2.1 Bisection Method

```
// Bisection Method
// Very useful for finding roots of a function
// With 100 repetitions the value already converges.
// This implementation only works if the function in the range [lo, hi] has
// zero.
//F(x)
        F(lo)
// |
//
//
//
                              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:
```

2.2 Ternary Search

```
// Ternary search
// Very useful for finding max/min values between interval
// The function on the interval must be unimodal (only 1 maximum)
//F(x)
// ^
                     Goal
//
//
//
//
//
//
//
                                  *F(r)
       *F(1)
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 - 1) / 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]
}
```

2.3 Binary Search

```
// Binary Search
int binary_search(vector<int> arr, int target) {
   int left = 0;
   int right = arr.size() - 1;
   while (left <= right) {
      int mid = left + (right - left) / 2;
      if (arr[mid] == target) return mid;
      if (arr[mid] < target)
            left = mid + 1;
      else
            right = mid - 1;
   }
   return -1;
}</pre>
```

3 Sorting

3.1 Merge Sort (Inversion Counter)

```
// 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];</pre>
    int i = lo;
    int j = mid + 1;
    for (int k = lo; k <= hi; k++) {</pre>
        if (i > mid)
            arr[k] = aux[j++];
        else if (j > hi)
            arr[k] = aux[i++];
        else if (aux[j] < aux[i]) {</pre>
            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;
```

4 Graph Algorithms

4.1 DFS

4.2 Strongly Connected Components

```
// Tarjan's Algorithm
// Finding strongly connected components (Directed Graph)
int     V;
vector<vi> adj;
vi     dfslow, dfsnum;
```

```
vis[VMAX];
bool
int
           SCC, TIME;
stack<int> aux;
void tarjan_dfs(int s) {
    dfslow[s] = dfsnum[s] = ++TIME;
    aux.push(s);
    vis[s] = true;
    for (auto a : adi[s]) {
        if (!dfsnum[a]) tarjan_dfs(a);
        if (vis[a]) dfslow[s] = min(dfslow[s], dfslow[a]);
    if (dfslow[s] == dfsnum[s]) {
        SCC += 1;
        while (1) {
            int v = aux.top();
            aux.pop();
            vis[v] = 0;
            if (s == v) break:
void scc() {
    aux = stack<int>();
    dfslow = vi(V, 0);
    dfsnum = vi(V, 0);
    memset(vis, false, sizeof(vis));
    TIME = SCC = 0;
    for (int i = 0; i < V; i++) {
        if (!dfsnum[i]) tarjan_dfs(i);
}
```

4.3 Edmonds-Karp

```
// Edmonds-Karp Algorithm
// Min-Cut/Max-Flow problem
int    V;
vector<vi> adj;
```

```
vector<vi> capacity;
int bfs(int s, int t, vi& parent) {
    fill(parent.begin(), parent.end(), -1);
    parent[s] = -2;
    queue<pair<int, int>> q;
    q.push({s, INF});
    while (!q.empty()) {
        int cur = q.front().first;
        int flow = q.front().second;
        q.pop();
        for (int next : adj[cur]) {
            if (parent[next] == -1 && capacity[cur][next]) {
                parent[next] = cur;
                int new_flow = min(flow, capacity[cur][next]);
                if (next == t) return new_flow;
                q.push({next, new_flow});
           }
       }
    return 0;
int maxflow(int s, int t) {
    int flow = 0;
    vi parent(V);
    int new_flow;
    while ((new_flow = bfs(s, t, parent))) {
        flow += new_flow;
        int cur = t;
        while (cur != s) {
            int prev = parent[cur];
            capacity[prev][cur] -= new_flow;
            capacity[cur][prev] += new_flow;
            cur = prev;
    return flow;
```