

Universidade Federal de Sergipe

Contents

Greedy

1 STL	1
1.1 Sort (628fd6)	1
2 Binary Search and Ternary Search	1
2.1 BinarySearch (36e7bb)	1
2.2 BinarySearchDouble (758367)	1
2.3 LowerUpperBound (4d5e64)	1
3 Dynamic Programming	1
3.1 LIS BS (78d63a)	1
3.2 LIS DS (12492d)	2
4 Geometry	3
4.1 ConvexHull (e8ad2a)	3
4.2 SweepLine (9d88c7)	3
5 Graph	4
5.1 BFS (ecec15)	4
5.2 Cycles (b74aae)	4
5.3 DFS (bf2c67)	5
5.4 Dijkstra (df96ee)	5
5.5 FindingConnectComponents (7a0e22)	5
5.6 FloydWarshall (f8ebbc)	5
5.7 SizeOfAllSubtrees (a2922a)	6
6 Math	6
6.1 Divisores (de19ab)	6
7 Strings	6
7.1 KMP (082a41)	6
8 Data Structures	6
8.1 FenwickTree (27422d)	6
8.2 OrderStatisticSet (0b671e)	7
8.3 SegmentTree (b19a9b)	7
8.4 SegmentTreeLazy (e2aa98)	8
9 Techniques	8
9.1 CoordinateCompression (750997)	8
9.2 Grid (070c56)	8
10 Utils	9
10.1 Makefile (dacce6)	9
10.2 hash (d78ff6)	9
10.3 template (c25d80)	9
10.4 vimrc (54a8c1)	9

1 STL

1.1 Sort (628fd6)

```
#include <bits/stdc++.h>
using namespace std;
vector<int> v = {1, 2, 3, 4};
signed main() {
    // Ordenacao CRESCENTE (Padrao)
    sort(v.begin(), v.end());
    // Ordenacao DECRESCENTE
```

```
sort(v.rbegin(), v.rend());
// Ordenacao CRESCENTE usando lambda
sort(v.begin(), v.end(), [](int a, int b) {
    return a < b;
});
// Ordenacao DECRESCENTE usando lambda
sort(v.begin(), v.end(), [](int a, int b) {
    return a > b;
});
// Ordenacao CRESCENTE usando funcao pronta
sort(v.begin(), v.end(), less<int>());
// Ordenacao DECRESCENTE usando funcao pronta
sort(v.begin(), v.end(), greater<int>());
// Verificar se v[] esta ordenado em ordem CRESCENTE
if (is_sorted(v.begin(), v.end())) {
    cout << "v[] esta ordenado em ordem CRESCENTE" << endl;
}
// Verificar se v[] esta ordenado em ordem DECRESCENTE
if (is_sorted(v.rbegin(), v.rend())) {
    cout << "v[] esta ordenado em ordem DECRESCENTE" << endl;
}
}
```

2 Binary Search and Ternary Search

2.1 BinarySearch (36e7bb)

```
#include <bits/stdc++.h>
using namespace std;
int n, x;
bool possible(int m) { return true; }; // O(M)
// Retorna o primeiro elemento que valida nossa propriedade
// lower_bound -> primeiro valor >= x (Primeiro Verdadeiro (MINIMIZAR))
// O nosso f(x) precisa ser:
// - f(x) = 1, entao f(y) = 1 para todo y >= x
// - f(x) = 0, entao f(y) = 0 para todo y <= x.
int bs(int x) { // O(M*log(n))
    int l = 0, r = n; // r = MAX
    while (l < r) {
        int m = l + (r-l)/2; // Piso
        if (possible(m)) r = m;
        else l = m + 1;
    }
    if (l == n) return -1;
    return l;
}
// Retorna o ultimo elemento que valida nossa propriedade
// O nosso f(x) precisa ser:
// - f(x) = 1, entao f(y) = 1 para todo y <= x
// - f(x) = 0, entao f(y) = 0 para todo y >= x.
int bs_last(int x) {
    int l = -1, r = n; // r = MAX
    while (l < r) {
        int m = l + (r-l+1)/2; // Teto (Ultimo Verdadeiro (MAXIMIZAR))
        if (possible(m)) l=m;
        else r=m-1;
    }
    return l; // Caso nao encontre, retorna -1
}
```

2.2 BinarySearchDouble (758367)

```
#include <bits/stdc++.h>
using namespace std;
const double EPS = 1e-6; // Preciso -> defino as casas no setprecision
const int N_ITERATIONS = 100;
bool is_possible(long double m) {return true;}
long double bs(int x) {
    long double l = 0, r = 1e7;
    // for (int i = 0; i < N_ITERATIONS; i++) { -> Mais seguro
```

```

while ((r - l) > EPS) {
    // Abaixo voce pode ajustar para o problema
    long double m = l + (r-l)/2;
    if (is_possible(m)) l = m;
    else r = m;
}
cout << fixed << setprecision(6) << l << endl;
}

```

2.3 LowerUpperBound (4d5e64)

```

#include <bits/stdc++.h>
using namespace std;
signed main() {
    int x; vector<int> v;
    // lower_bound: retorna um it para o PRIMEIRO elemento >= x
    auto it = lower_bound(v.begin(), v.end(), x);
    cout << "pos: " << it - v.begin() << endl;
    cout << "valor: " << *it << endl;
    // upper_bound: retorna um it para o PRIMEIRO elemento > x
    auto it = upper_bound(v.begin(), v.end(), x);
    // Quantidade de Elementos Validos = UPPER - LOWER
}

```

3 Dynamic Programming

3.1 LIS BS (78d63a)

```

#include <bits/stdc++.h>
using namespace std;

#define _ ios_base::sync_with_stdio(0);cin.tie(0);
#define endl '\n'
#define all(x) (x).begin(), (x).end()

// O(n*log(n)) --> Solucao TOP
// dp[l] --> e o menor a[i] que a maior LIS de tamanho l termina
// solucao e maior l t.q d[l] nao e INF
// Obs.: Essa solucao nao serve para contar o numero de LIS da a[], precisa usar
// lis-ds.

// int lis(vector<int> const& a) {
vector<int> lis(vector<int> const& a) {
    int n = a.size();
    const int INF = 1e9;
    vector<int> dp(n + 1, INF);
    vector<int> id_dp(n + 1, -1); // Armazena o indice i do valor dp[l] =
    // a[i]
    vector<int> previous(n + 1, -1); // Armazena o indice do dp[l - 1]

    dp[0] = -INF;

    for (int i = 0; i < n; i++) {
        // Solucao trivial
        // for (int l = 1; l <= i; l++) {
        //     if (dp[l-1] < a[i] && a[i] < dp[l]) {
        //         dp[l] = a[i];
        //     }
        // }

        // dp e estritamente crescente e a[i] atualiza apenas um valor de dp[l]
        // dp[l - 1] < a[i] < dp[l] --> Podemos encontrar o l a partir da Busca
        // Binaria
        int l = upper_bound(all(dp), a[i]) - dp.begin();
        if (dp[l-1] < a[i] && a[i] < dp[l]) {
            dp[l] = a[i];
            id_dp[l] = i;
            previous[i] = id_dp[l-1];
        }
    }
}

```

```

int ans = 0;
for (int l = 0; l <= n; l++) {
    if (dp[l] < INF) ans = l;
}

// Apenas retornar a resposta
// return ans;

// Reconstruir a subsequencia
vector<int> subseq;

int pos = id_dp[ans];
while (pos != -1) {
    subseq.push_back(a[pos]);
    pos = previous[pos];
}

reverse(all(subseq));
return subseq;
}

```

```

signed main(){
    int n; cin >> n;
    vector<int> a(n);
    for (auto &x : a) cin >> x;

    // int ans = lis(a);
    // cout << ans << endl;

    vector<int> ans = lis(a);
    for (auto x : ans) {
        cout << x << " ";
    }
    cout << endl;
}

```

/* https://cp-algorithms.com/sequences/longest_increasing_subsequence.html
Considere uma array $a[n]$.

Problema: Nosso objetivo e encontrar a maior subsequencia estritamente crescente de a .

Subproblema: Vamos considerar que $dp[l]$ e menor valor que um subsequencia estritamente crescente de tamanho l termina.

- $dp[0] = -INF$ (Caso base)

Vamos iniciar $dp[l] = INF$ e vamos processar cada $a[i]$ com $0 \leq i \leq n$

Exemplo: $a = \{8, 3, 4, 6, 5, 2, 0, 7, 9, 1\}$

prefix = {}	--> d = {-INF, INF, INF, ..., INF}
prefix = {8}	--> d = {-INF, 8, INF, ..., INF}
prefix = {8, 3}	--> d = {-INF, 3, INF, ..., INF}
prefix = {8, 3, 4}	--> d = {-INF, 3, 4, ..., INF}
prefix = {8, 3, 4, 6}	--> d = {-INF, 3, 4, 6, ..., INF}
prefix = {8, 3, 4, 6, 5}	--> d = {-INF, 3, 4, 5, ..., INF}
prefix = {8, 3, 4, 6, 5, 2}	--> d = {-INF, 2, 4, 5, ..., INF}
prefix = {8, 3, 4, 6, 5, 2, 0}	--> d = {-INF, 0, 4, 5, ..., INF}
prefix = {8, 3, 4, 6, 5, 2, 0, 7}	--> d = {-INF, 0, 4, 5, 7, ..., INF}
prefix = {8, 3, 4, 6, 5, 2, 0, 7, 9}	--> d = {-INF, 0, 4, 5, 7, 9, ..., INF}
prefix = {8, 3, 4, 6, 5, 2, 0, 7, 9, 1}	--> d = {-INF, 0, 1, 5, 7, 9, ..., INF}

Quando vamos fazer $d[l] = a[i]$?
 - Quando tiver nenhuma lis de tamanho l que termine em $a[i]$
 - E, se nao tiver nenhuma outra lis de tamanho l que termina em algum numero menor que $a[i]$

Note que ha uma sub-estrutura otima do problema, a solucao para l pode ser construida a partir de $l - 1$

```

dp[l] = -INF, se l = 0
        min(a[i], d[l]), se a[i] > d[l - 1]
        +INF, caso contrario.
*/

```

3.2 LIS DS (12492d)

```
#include <bits/stdc++.h>
using namespace std;

#define _ ios_base::sync_with_stdio(0);cin.tie(0);
#define endl '\n'
#define all(x) (x).begin(), (x).end()

typedef long long ll;

const ll MOD = 1e9 + 7;

// Coordinate Compression
void coordinate_compression(vector<int> &a) {
    set<int> s; // Conjunto para armazenar todos os numeros unicos
    for (auto x : a) s.insert(x);

    int index = 0;
    map<int, int> mp; // Map para armazenar os novos elementos

    set<int>::iterator itr;
    for (itr = s.begin(); itr != s.end(); itr++) {
        index++;
        mp[*itr] = index;
    }

    // Alterando o valor de a
    for (int i = 0; i < a.size(); i++) {
        a[i] = mp[a[i]];
    }
}

// BIT ou SegTree
struct fenw {
    int n;
    vector<int> bit;

    fenw() {}
    fenw(int size) {
        n = size;
        bit.assign(size + 1, 0);
    }
    // query do maior prefixo a[0...r]
    int qry(int r) {
        int ans = 0;
        for (int i = r + 1; i > 0; i -= i & -i) // i & -i retorna os bits menos
            // signativos de i
            ans = max(ans, bit[i]);
        return ans;
    }

    // atualiza o valor a[r] = x
    void upd(int r, int x) {
        for (int i = r + 1; i <= n; i += i & -i)
            bit[i] = max(bit[i], x);
    }
};

// O(nlog(n))
int lis(vector<int> &a) {
    coordinate_compression(a);

    int n = a.size();

    fenw tree(n + 1);

    int ans = 0;
    for (int i = 0; i < n; i++) {
        int best = tree.qry(a[i] - 1);
        tree.upd(a[i], best + 1);
        ans = max(ans, best + 1);
    }

    return ans;
}

signed main() { _
```

```
int n; cin >> n;
vector<int> a(n);
for (auto &x : a) cin >> x;

cout << lis(a) << endl;
}

// https://cp-algorithms.com/sequences/longest_increasing_subsequence.html#
// solution-in-on-log-n-with-data-structures
// Considere t[a[i]] = dp[i] # dp[i] e o maior l de LIS t.q termina com a[i]
/* Para calcular dp[], precisamos fazer:
ans = -INF;
Para i = 0 ate 0:
    dp[i] = max(t[0...a[i]] + 1) // Maior prefixo ate a[i]
    ans = max(ans, dp[i])
# O problema e buscar o maior prefixo de a[0..i] com a[k] mudando,
sendo 0 <= k < i. --> SegTree ou BIT
# Pontos Negativos da Solucao
- Implementacao Complexa
- a[i] for mt grande --> Coordinate Compression ou Dynamic SegTree
*/
```

4 Geometry

4.1 ConvexHull (e8ad2a)

```
#include <bits/stdc++.h>
using namespace std;

#define int long long int

struct pt {
    int x, y, id;
    pt(int x, int y, int id) : x(x), y(y), id(id) {}
    pt() {}
    pt operator-(pt const &o) const {
        return {x - o.x, y - o.y, -1};
    }
    bool operator<(pt const &o) const {
        if (x == o.x) {
            return y < o.y;
        }
        return x < o.x;
    }
    int operator^(pt const &o) const {
        return (int)x * o.y - (int)y * o.x;
    }
};

int ccw(pt const &a, pt const &b, pt const &x) {
    auto p = (b - a) ^ (x - a);
    return (p > 0) - (p < 0);
}

vector<pt> convex_hull(vector<pt> P, bool include_collinear) {
    // include_collinear: define se vai retornar os pontos colineares as
    // bordas
    // ou seja, pontos da aresta do poligono
    sort(P.begin(), P.end());
    vector<pt> L, U;
    int pop_threshold = include_collinear ? -1 : 0;
    for (auto p : P) {
        while (L.size() >= 2 && ccw(L.end()[-2], L.end()[-1], p) <=
            pop_threshold) {
            L.pop_back();
        }
        L.push_back(p);
    }
    reverse(P.begin(), P.end());
    for (auto p : P) {
        while (U.size() >= 2 && ccw(U.end()[-2], U.end()[-1], p) <=
            pop_threshold) {
            U.pop_back();
        }
    }
}
```

```

        U.push_back(p);
    }
    L.insert(L.end(), U.begin() + 1, U.end() - 1);
    return L;
}
signed main() {
    int n; cin >> n;
    vector<pt> arr;
    for (int i = 0; i < n; i++) {
        int x, y; cin >> x >> y;
        arr.emplace_back(x, y, i+1);
    }
    vector<pt> ans = convex_hull(arr, true);
    for (auto [x, y, id] : ans) {
        cout << x << " " << y << " " << id << '\n';
    }
}

```

4.2 SweepLine (9d88c7)

```

#include <bits/stdc++.h>
using namespace std;

#define endl '\n'
#define ENTRADA 0
#define SAIDA 1

typedef struct Evento {
    int tempo, tipo, id;
    Evento(int _tempo, int _tipo, int _id) : tempo(_tempo), tipo(_tipo), id(_id) {}
    // O elemento id e util quando temos que responder queries
    bool operator < (Evento outro) {
        if (tempo == outro.tempo) return tipo < outro.tempo; // Definir o
        // criterio de desempate
        return tempo < outro.tempo;
    }
} evento_t;

int main() {
    int n; cin >> n;
    vector<evento_t> ev;
    // Leitura dos eventos
    for (int i = 0; i < n; i++) {
        int e, s; cin >> e >> s;
        ev.push_back({e, ENTRADA, i}); // Evento de Entrada
        ev.push_back({s, SAIDA, i}); // Evento de Saída
    }
    // Pre-ordenacao para Varredura
    sort(ev.begin(), ev.end());
    // Varredura --> Verificar qual o maximo de pessoas
    int acc = 0, ans = -1;
    for (auto [tempo, tipo, id] : ev) {
        if (tipo == ENTRADA) acc++;
        else acc--;
        ans = max(ans, acc);
    }
    cout << ans << endl;
    return 0;
}

/*
Problemas de sweep line sao descritas por duas variaveis:
- Localizacao temporal ou espacial;
- Tipo do evento

O problema esta em saber qual a melhor estrutura para armazenar a resposta (um
inteiro, um set, uma Segment Tree, ...)

https://noic.com.br/materiais-informatica/curso/line-sweep/
*/

```

5 Graph

5.1 BFS (ecec15)

```

#include <bits/stdc++.h>
using namespace std;
#define pb push_back
const int MAXN = 1e5 + 10;
int n, m;
vector<int> adj[MAXN], dist(MAXN, -1), parent(MAXN, -1);
bool vis[MAXN];
// BFS Multisource -> bfs(vector<pi> ms): permite fazer BFS em varias fontes
// Encontrar o menor ciclo de um grafo ou menor ciclo que possui um vertice
// Encontrar a menor distancia de s para todos os outros vertices
void bfs(int s) { // O(V + E)
    queue<int> q;
    q.push(s); vis[s] = true;
    dist[s] = 0;
    parent[s] = -1;
    while (!q.empty()) {
        int v = q.front(); q.pop();
        for (auto u : adj[v]) if (!vis[u]) {
            q.push(u); vis[u] = true;
            dist[u] = dist[v] + 1;
            parent[u] = v;
        }
    }
}
// https://cp-algorithms.com/graph/breadth-first-search.html
signed main() {
    cin >> n >> m;
    for (int i = 0; i < m; i++) {
        int u, v; cin >> u >> v;
        adj[u].pb(v);
        adj[v].pb(u);
    }
    bfs(0);
    // Mostrar o caminho de s ate u
    int u; cin >> u;
    if (!vis[u]) {
        cout << "No path" << endl;
    } else {
        stack<int> path;
        for (int v = u; v != -1; v = parent[v]) {
            path.push(v);
        }
        // Exibir caminho
        cout << "Path: ";
        while (!path.empty()) {
            cout << path.top() + 1 << " ";
        }
        cout << endl;
    }
}

```

5.2 Cycles (b74aae)

```

#include <bits/stdc++.h>
using namespace std;
#define pb push_back
const int MAX = 1e5 + 10;
int n, m;
vector<int> adj[MAX];
int parent[MAX], cycle_start, cycle_end;
bool vis[MAX]; // Grafo nao-direcionado
int color[MAX]; // Grafo direcionado
// Grafo nao-direcionado
bool dfs(int v) {
    vis[v] = true;
    for (auto u : adj[v]) {
        if (u == parent[v]) continue;
        if (vis[u]) {
            cycle_end = v;
        }
    }
}

```

```

        cycle_start = u;
        return true;
    }
    parent[u] = v;
    if (dfs(u)) return true;
}
return false;
}
// Grafo direcionado
bool dfs(int v) {
    color[v] = 1;
    for (int u : adj[v]) {
        if (color[u] == 0) {
            parent[u] = v;
            if (dfs(u))
                return true;
        } else if (color[u] == 1) {
            cycle_end = v;
            cycle_start = u;
            return true;
        }
    }
    color[v] = 2;
    return false;
}

signed main() {
    cycle_start = -1;
    memset(parent, -1, sizeof(parent));
    memset(color, 0, sizeof(color));
    // Grafo nao-direcionado
    for (int v = 0; v < n; v++) {
        // break -> encontrou um ciclo
        if (!vis[v] and dfs(v)) break;
    }
    // Grafo direcionado
    for (int v = 0; v < n; v++) {
        // break -> encontrou um ciclo
        if ((color[v]==0) and dfs(v)) break;
    }
    // Exibindo o ciclo
    if (cycle_start == -1) {
        cout << "Acyclic" << endl;
    } else {
        vector<int> cycle;
        cycle.pb(cycle_start);
        for (int v = cycle_end; v != cycle_start; v = parent[v])
            cycle.pb(v);
        cycle.pb(cycle_start);
    }
}
// https://cp-algorithms.com/graph/finding-cycle.html

```

5.3 DFS (bf2c67)

```

#include <bits/stdc++.h>
using namespace std;
#define pb push_back
#define MAXN 100005
vector<int> adj[MAXN];
bool visited[MAXN];
int n, m;
// O(V + E)
void dfs(int v) { // dfs(int v, int p=-1) -> Arvores
    visited[v] = true;
    // for (auto u : adj[v]) if (u != p) { -> Arvores
    for (auto u : adj[v]) if (!visited[u]) {
        dfs(u);
    }
}
// https://cp-algorithms.com/graph/depth-first-search.html

```

5.4 Dijkstra (df96ee)

```

#include <bits/stdc++.h>
using namespace std;
#define int long long
typedef pair<int, int> pi;
const int INF = 0x3f3f3f3f3f3f3f3f;
const int MAX = 1e5 + 10;
int n, m;
vector<pi> adj[MAX]; // {vizinho, peso}
bool vis[MAX]; int dist[MAX], parent[MAX];
void dijkstra(int s) {
    fill(vis, vis+MAX, false);
    fill(dist, dist+MAX, INF);
    // priority_queue de {distancia, vertice}
    priority_queue<pi, vector<pi>, greater<pi>> pq;
    pq.push({0, s}); dist[s] = 0; parent[s] = -1;
    while (!pq.empty()) {
        auto [d, v] = pq.top(); pq.pop();
        if (vis[v]) continue;
        vis[v] = true;
        for (auto [u, w] : adj[v]) if (dist[u] > dist[v] + w) {
            dist[u] = dist[v] + w;
            parent[u] = v;
            pq.push({dist[u], u});
        }
    }
}

signed main() {
    ios_base::sync_with_stdio(0); cin.tie(0);
    cin >> n >> m;
    for (int i = 0; i < m; i++) {
        int a, b, c; cin >> a >> b >> c; a--, b--;
        adj[a].push_back({b, c});
    }
}
// https://cp-algorithms.com/graph/dijkstra.html

```

5.5 FindingConnectComponents (7a0e22)

```

#include <bits/stdc++.h>
using namespace std;
#define pb push_back
const int MAXN = 1e5 + 10;
vector<int> adj[MAXN], comp;
bool vis[MAXN];
int n, m;

void dfs(int v) {
    vis[v] = true;
    comp.pb(v);
    for (auto u : adj[v]) if (!vis[u]) {
        dfs(u);
    }
}

signed main() {
    cin >> n >> m;
    for (int i = 0; i < m; i++) {
        int u, v; cin >> u >> v;
        adj[u].pb(v);
        adj[v].pb(u);
    }
    // Encontrar os componentes conexos (O(V + E))
    for (int v = 0; v < n; v++) {
        if (!vis[v]) {
            comp.clear();
            dfs(v);
            // Percorrendo no componente conexo
            cout << "Component: ";
            for (auto u : comp) {
                cout << u << " ";
            }
            cout << endl;
        }
    }
}
// https://cp-algorithms.com/graph/search-for-connected-components.html

```

5.6 FloydWarshall (f8ebbc)

```
#include <bits/stdc++.h>
using namespace std;

const int INF = 0x3f3f3f3f;

#define MAXN 10000

// n -> numero de vertices, m -> numero de arestas
int n, m;
// w[i][j] -> matriz com o peso da aresta (i, j)
int w[MAXN][MAXN];
// dist[i][j] -> matriz com a menor distancia entre os vertices i e j
int dist[MAXN][MAXN];

void initialize() {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (i == j) {
                dist[i][j] = 0;
            } else {
                dist[i][j] = INF;
            }
        }
    }
}

void floyd_warshall() {
    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]);
            }
        }
    }
}

signed main() {
    cin >> n >> m;
    initialize();
    for (int i = 0; i < m; i++) {
        // a, b -> vertices e c -> custo da aresta (a, b)
        int a, b, c; cin >> a >> b >> c;
        w[a][b] = c;
        dist[a][b] = min(dist[a][b], c);
        // Comente as linhas abaixo, caso o Grafo seja direcionado
        w[b][a] = c;
        dist[b][a] = min(dist[b][a], c);
    }
    floyd_warshall();
}
```

5.7 SizeOfAllSubtrees (a2922a)

```
#include <bits/stdc++.h>
using namespace std;
#define pb push_back

const int MAXN = 1e5 + 10;

int n, m, subtree_size[MAXN];
vector<int> adj[MAXN];
bool vis[MAXN];

void dfs(int v) {
    vis[v] = true;
    subtree_size[v] = 1;
    for (auto u : adj[v]) if (!vis[u]) {
        dfs(u);
        subtree_size[v] += subtree_size[u];
    }
}
```

```
signed main() {
    cin >> n >> m;
    for (int i = 0; i < m; i++) {
        int u, v; cin >> u >> v;
        adj[u].pb(v);
        adj[v].pb(u);
    }
    dfs(0);
}
```

6 Math

6.1 Divisores (de19ab)

```
#include <bits/stdc++.h>
using namespace std;

#define pb push_back
#define MAXN 100005

vector<int> divisores[MAXN];

signed main() {
    int n; cin >> n;
    for (int i = 1; i <= n; i++) { // O(nlog(n))
        for (int j = i; j <= n; j += i) { // j -> multiplo de i
            divisores[j].pb(i);
        }
    }
}
```

7 Strings

7.1 KMP (082a41)

```
#include <bits/stdc++.h>
using namespace std;

// n = |padrao| e m = |texto|
// pi: O(n) match: O(n + m) automato: O(|sigma| * n)

// Funcao de prefixo pi(i):
// Para todo prefixo s' de s, a funcao calcula o tamanho do maior
// do prefixo proprio de s' que tambem e sufixo.
vector<int> pi(string s) {
    vector<int> p(s.size());
    for (int i = 1, j = 0; i < s.size(); i++) {
        while (j > 0 and s[i] != s[j]) j = p[j-1];
        if (s[i] == s[j]) j++;
        p[i] = j;
    }
    return p;
}

// t -> texto, s -> padrao
// A funcao retorna a quantidade de matchings
vector<int> matching(string &t, string& s) {
    // '$' e um caracter impossivel no padrao,
    // serve para nao tratar o caso de quando acaba o matching
    vector<int> p = pi(s+'$'), match;
    for (int i = 0, j = 0; i < t.size(); i++) {
        while (j > 0 and t[i] != s[j]) j = p[j-1];
        if (t[i] == s[j]) j++;
        if (j == s.size()) match.push_back(i-j+1);
    }
    return match;
}
```

```
vector<int> autKMP(string &s) {
}

vector<int> matchingAut(string &t, string &s) {
    vector<int> match;
    return match;
}
```

8 Data Structures

8.1 FenwickTree (27422d)

```
// Binary Indexed Tree ou Fenwick Tree
#include <bits/stdc++.h>
using namespace std;

// 0-INDEXXED
struct fenw {
    int n;
    vector<int> bit;
    fenw() {}
    fenw(int size) {
        n = size;
        bit.assign(size + 1, 0);
    }
    // query do prefixo a[0] + a[1] + ... + a[r]
    int qry(int r) {
        int ans = 0;
        for (int i = r + 1; i > 0; i -= i & -i) // i & -i retorna os bits menos
            // signativos de i
            ans += bit[i];
        return ans;
    }
    // atualiza o valor a[r] = x
    void upd(int r, int x) {
        for (int i = r + 1; i <= n; i += i & -i) bit[i] += x;
    }
    // busca binaria para o maior indice i (i < n) tal que qry(i) < x
    int bs(int x) {
        int i = 0, k = 0;
        while (1 <= (k + 1) <= n) k++;
        while (k >= 0) {
            int nxt_i = i + (1 <= k);
            if (nxt_i <= n && bit[nxt_i] < x) {
                i = nxt_i;
                x -= bit[i];
            }
            k--;
        }
        return i - 1;
    }
};
```

8.2 OrderStatisticSet (0b671e)

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

// Consiste em uma Arvore Binaria Balanceada (std::set + superpoderes)
// Armazena elementos unicos e de forma ordenada
// Operacoes:
// 1 - Todas de um std::set
// 2 - find_by_order(k): retorna um iterator para o k-esimo valor (0-INDEXXED) (0
    (log(n)))
// 3 - order_of_key(k): retorna o numero de elementos estritamente:
// - menores, se less<T>
```

```
// - maiores, se greater<T>
// que k. (ou seja, o NUMERO DE INVERSOES)
template <class T>
using ord_set = tree<
    T,
    null_type,
    less<T>,
    rb_tree_tag,
    tree_order_statistics_node_update
>;
// Voce pode fazer um multiset com ord_set<pair<T, int>>, o segundo elemento e
    um indice por exemplo.
signed main() {
    ord_set<int> s;
    // Insercao
    for (int i = 0; i < 10; i++) s.insert(i);
    // Leitura
    for (auto i : s) cout << i << endl;
    int k = 0;
    // Posicao do valor k
    cout << *s.find_by_order(k) << endl;
    // Quantos sao menores do que k O(log|s|):
    cout << s.order_of_key(k) << endl;
    // Remocao - Set
    s.erase(5); // apaga o valor 5 (se existir)
    // apaga o elemento pelo iterator
    auto it = s.find(7);
    if (it != s.end()) s.erase(it);
    // remove todos os menores < 5
    auto it_end = s.lower_bound(5);
    while (s.begin() != it_end) {
        s.erase(s.begin());
    }
}
// https://codeforces.com/blog/entry/123624
```

8.3 SegmentTree (b19a9b)

```
#include <bits/stdc++.h>
using namespace std;
#define endl '\n'
const int INF = 0x3f3f3f3f;
#define MAXN 1000000
int n, v[MAXN], seg[4*MAXN]; // a SegTree esta 0-INDEXXED
// Funcoes de Apoio
int single(int x) {return x;}
int neutral() {return -INF;}
int merge(int a, int b) {return max(a, b);}
// p -> indice na segtree, [l, r] -> intervalo da subarray
int build(int p=1, int l=0, int r=n-1) { // O(n)
    if (l == r) return seg[p] = single(v[l]);
    int m = (l+r)/2;
    return seg[p] = merge(build(2*p, l, m), build(2*p+1, m+1, r));
}
// query no intervalo [a, b]
int qry(int a, int b, int p=1, int l=0, int r=n-1) { // O(log(n))
    if (b < l or a > r) return neutral();
    if (a <= l and r <= b) return seg[p];
    int m = (l+r)/2;
    return merge(qry(a, b, 2*p, l, m), qry(a, b, 2*p+1, m+1, r));
}
// update -> v[i] = x
int upd(int i, int x, int p=1, int l=0, int r=n-1) { // O(log(n))
    if (i < l or r < i) return seg[p];
    if (l == r) return seg[p] = single(x);
    int m = (l+r)/2;
    return seg[p] = merge(upd(i, x, 2*p, l, m), upd(i, x, 2*p+1, m+1, r));
}
// primeira posicao >= x em [a, b] (ou -1, caso nao exista)
// Obs.: So funciona na SegTree de Maximos
int first_above(int x, int a, int b, int p=1, int l=0, int r=n-1) { // O(log(n))
    if (b < l or r < a or seg[p] < x) return -1;
    if (l == r) return l;
    int m = (l+r)/2;
    int left = first_above(x, a, b, 2*p, l, m);
```

```

    if (left != -1) return left;
    return first_above(x, a, b, 2*p+1, m+1, r);
}
// ultima posicao >= x em [a, b] (ou -1, caso nao exista)
// Obs.: So funciona na SegTree de Maximos
int last_above(int x, int a, int b, int p=1, int l=0, int r=n-1) { // O(log(n))
    if (b < l or r < a or seg[p] < x) return -1;
    if (l == r) return l;
    int m = (l+r)/2;
    int right = last_above(x, a, b, 2*p+1, m+1, r);
    if (right != -1) return right;
    return last_above(x, a, b, 2*p, l, m);
}
// retorna a posicao i do vetor do Kth l
// Obs.: So funciona na SegTree de Soma
// k e 1-INDEXED e o RETORNO e 0-INDEXED
int find_kth(int k, int p=1, int l=0, int r=n-1) {
    if (k > seg[p]) return -1;
    if (l == r) return l;
    int m = (l + r) / 2;
    if (seg[p*2] >= k)
        return find_kth(k, p*2, l, m);
    else
        return find_kth(k - seg[p*2], p*2+1, m+1, r);
}
signed main() {
    ios_base::sync_with_stdio(0); cin.tie(0);
    cin >> n;
    // Voce pode inicializar v[] com um valor neutro
    // fill(v, v+MAXN, netrual());
    // Leitura de v[]
    for(int i = 0; i < n; i++) cin >> v[i];
    // Construir a SegTree
    build(1, 0, n - 1);
    // Query do intervalo a, b
    int a, b; cin >> a >> b;
    cout << qry(a, b, 1, 0, n - 1) << endl;
    // Update de v[i] = x
    int i, x; cin >> i >> x;
    cout << upd(i, x, 1, 0, n - 1) << endl;
}
/* Segment Tree ou SegTree ou Arvore de Segmentos
# Altura da arvore -> O(log(n))
# Qtd de nos -> 2n - 1 -> O(n)
-> Problemas de RMQ: Range Minimun Query
    -> Descobrir o minimo de um intervalo [l, r]

-> Soma, Minimo, Maximo, Produto, ...
-> SegTree para qualquer operacao ASSOCIATIVA: (A ? B) ? C = A ? (B ? C)
*/

```

8.4 SegmentTreeLazy (e2aa98)

```

#include <bits/stdc++.h>
using namespace std;
#define endl '\n'
#define MAXN 100005
int n, q, v[MAXN], seg[MAXN*4], lazy[MAXN*4], leafs[MAXN];
// Funcoes de Apoio
int single(int x) { return x; }
int neutral() { return 0; }
int merge(int a, int b) { return a + b; }
// Funcao de Propagacao
void prop(int p, int l, int r) { // O(1)
    seg[p] += lazy[p]*(r-l+1);
    if (l != r) {
        lazy[2*p] += lazy[p];
        lazy[2*p+1] += lazy[p];
    }
    lazy[p] = 0;
}
// p -> indice na segtree, [l, r] -> intervalo da subarray
int build(int p=1, int l=0, int r=n-1) { // O(n)
    lazy[p] = 0;
    if (l == r) return seg[p] = single(v[l]);

```

```

    int m = (l+r)/2;
    return seg[p] = merge(build(2*p, l, m), build(2*p+1, m+1, r));
}
// query no intervalo [a, b]
int qry(int a, int b, int p=1, int l=0, int r=n-1) {
    prop(p, l, r);
    if (b < l or a > r) return neutral();
    if (a <= l and r <= b) return seg[p];
    int m = (l+r)/2;
    return merge(qry(a, b, 2*p, l, m), qry(a, b, 2*p+1, m+1, r));
}
// update -> Para todo v[i] += x, com a <= i <= b.
int upd(int a, int b, int x, int p=1, int l=0, int r=n-1) {
    prop(p, l, r);
    if (a <= l and r <= b) {
        lazy[p] += x;
        prop(p, l, r);
        return seg[p];
    }
    if (b < l or r < a) return seg[p];
    int m = (l+r)/2;
    return seg[p] = merge(upd(a, b, x, 2*p, l, m), upd(a, b, x, 2*p+1, m+1, r));
}
// Funcao para visitar as folhas, ou seja, o v[]
void get_leafs(int p = 1, int l = 0, int r = n - 1) { // O(n)
    prop(p, l, r);
    if (l == r) {
        leafs[l] = seg[p];
        return;
    }
    int m = (l+r)/2;
    get_leafs(2*p, l, m);
    get_leafs(2*p+1, m+1, r);
}
// primeira posicao >= x em [a, b] (ou -1, caso nao exista)
// Obs.: So funciona na SegTree de Maximos
int first_above(int x, int a, int b, int p=1, int l=0, int r=n-1) { // O(log(n))
    if (b < l or r < a or seg[p] < x) return -1;
    if (l == r) return l;
    int m = (l+r)/2;
    int left = first_above(x, a, b, 2*p, l, m);
    if (left != -1) return left;
    return first_above(x, a, b, 2*p+1, m+1, r);
}
// ultima posicao >= x em [a, b] (ou -1, caso nao exista)
// Obs.: So funciona na SegTree de Maximos
int last_above(int x, int a, int b, int p=1, int l=0, int r=n-1) { // O(log(n))
    if (b < l or r < a or seg[p] < x) return -1;
    if (l == r) return l;
    int m = (l+r)/2;
    int right = last_above(x, a, b, 2*p+1, m+1, r);
    if (right != -1) return right;
    return last_above(x, a, b, 2*p, l, m);
}
}

```

9 Techniques

9.1 CoordinateCompression (750997)

```

#include <bits/stdc++.h>
using namespace std;
// Considere que 1 <= ai <= 10^9 e 1 <= |a| <= 10^5.
// Vamos comprimir os valores de ai para [0, 10^5-1]. -> 0-INDEXED
void coordinate_compression(vector<int> &a) {
    set<int> s; // Conjunto para armazenar todos os numeros unicos
    for (auto x : a) s.insert(x);

    int index = 0;
    map<int, int> mp; // Map para armazenar os novos elementos

    set<int>::iterator it;
    for (it = s.begin(); it != s.end(); it++) {

```



```

    mp[*it] = index;
    index++;
}

// Alterando o valor de a
for (int i = 0; i < a.size(); i++) {
    a[i] = mp[a[i]];
}
}

```

9.2 Grid (070c56)

```

#include <bits/stdc++.h>
using namespace std;
#define endl '\n'
#define ff first
#define ss second
typedef pair<int, int> pi;
const int MAX = 1e3;
int n, m;
char grid[MAX][MAX];
bool vis[MAX][MAX];
// Movimentos: cima, baixo, esquerda, direita
pi mov[4] = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
// Funcao para validar de new_i, new_j sao validos
bool valid(int i, int j) {
    // Podemos adicionar outras CONDICAOES...
    // Por exemplo, "Ja foi visitado", "movimento nao e obstruido"
    return i >= 0 and j >= 0 and i < n and j < m and !vis[i][j];
}
signed main() {
    cin >> n >> m;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cin >> grid[i][j];
        }
    }
    int i = 0, j = 0; // Posicao Inicial
    for (auto [di, dj] : mov) {
        int ni = i + di, nj = j + dj;
        if (valid(ni, nj)) {
            vis[ni][nj] = true;
            // Faca algo!
        }
    }
}

```

10 Utils

10.1 Makefile (dacce6)

```

CXX = g++
CXXFLAGS = -fsanitize=address,undefined -fno-omit-frame-pointer -g -Wall -
Wshadow -std=gnu++20 -Wno-unused-result -Wno-sign-compare -Wno-char-
subscripts #-fuse-lto=gold

```

10.2 hash (d78ff6)

```

# Para usar (hash das linhas [11, 12]):
# bash hash.sh arquivo.cpp 11 12
sed -n $2', '$3' p' $1 | sed '/^#/d' | cpp -dD -P -fpreprocessed | tr -d '[:
space:]' | md5sum | cut -c-6

```

10.3 template (c25d80)

```

#include <bits/stdc++.h>
using namespace std;

#define endl '\n'
#define int long long
#define pb push_back
#define eb emplace_back
#define ff first
#define ss second
#define all(x) (x).begin(), (x).end()
#define rall(x) (x).rbegin(), (x).rend()
#define dbg(x) cout << #x << " = " << x << endl

signed main(){
    ios_base::sync_with_stdio(0);cin.tie(0);
}

```

10.4 vimrc (54a8c1)

```

set ts=4 sw=4 sts=4 expandtab mouse=a nu ai si undofile

```