

# Notebook de Programação Competitiva

October 1, 2025

# Contents

<b>1</b>	<b>Estrutura De Dados</b>	<b>2</b>
1.1	Fila . . . . .	2
1.2	Pilha . . . . .	3
1.3	Vector . . . . .	5
<b>2</b>	<b>Outros</b>	<b>5</b>
2.1	Soma De Prefixos . . . . .	5

# 1 Estrutura De Dados

## 1.1 Fila

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int main()
5 {
6     queue<int> q;
7     deque<int> dq;
8
9     q.push(3); // Adiciona elemento na fila
10    q.pop(); // Remove elemento no inicio
11    q.empty(); // Verifica se a fila está vazia
12    q.size(); // Retorna tamanho da fila
13    q.front(); // Retorna valor no inicio da fila
14    q.back(); // Retorna valor no fim da fila
15    dq.push_front(3); // Adiciona elemento no inicio
16    dq.pop_front(); // Remove elemento no inicio
17    dq.push_back(3); // Adiciona elemento no fim
18    dq.pop_back(); // Remove elemento no fim
19
20 }
```

Listing 1: fila.cpp

### Fila monótona

Seja  $F$  uma fila de elementos do tipo  $T$ . A fila  $F$  é dita **monótona** se, quando extraídos todos os elementos de  $F$ , eles formam uma sequência  $x_1, x_2, \dots, x_N$ , onde  $x_i$  é o elemento obtido na  $i$ -ésima extração, tais que a função  $F: \mathbb{N} \rightarrow T$ , com  $f(i) = x_i$ , é monótona. A fila  $F$  será **não-decrescente** se  $f$  for **não-decrescente**; caso contrário,  $F$  será **não-decrescente**.

- Em filas monótonas é necessário manter a invariante da monotonicidade a cada inserção.
- Seja  $F$  uma fila não-decrescente e  $x$  um elemento a ser inserido em  $F$ .
- Se  $F$  estiver vazia, basta inserir  $x$  em  $F$ : o invariante estará preservado.
- Se  $F$  não estiver vazia, o mesmo acontece se  $x \leq y$ , onde  $y$  é o último de  $F$ .
- Contudo, se  $x > y$ , é preciso remover  $y$  antes da inserção de  $x$ .
- Após a remoção de  $y$ , é preciso confrontar  $x$  com o novo elemento que ocupará a última posição até que  $x$  possa ser inserido na última posição de  $F$ .

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 template <typename T>
5 class MonoQueue {
6 public:
7     void push(const T& x) {
8         while (not st.empty() and st.back() > x)
9             st.pop_back();
10        st.emplace_back(x);
11    }
12
13    void pop() {
14        st.pop_front();
15    }
16
17    auto back() const {
18        return st.back();
19    }
20
21    auto front() const {
22        return st.front();
23    }
24 }
```

```

25     bool empty() const {
26         return st.empty();
27     }
28
29 private:
30     deque<T> st;
31 };
32
33 template <typename T>
34 ostream& operator<<(ostream& os, const MonoQueue<T>& ms) {
35     auto temp(ms);
36     while (not temp.empty()) {
37         cout << temp.front() << ' ';
38         temp.pop();
39     }
40     return os;
41 }
42
43 int main() {
44     vector<int> as{1, 4, 3, 4, 2, 1, 3};
45     MonoQueue<int> mq;
46
47     for (auto& a : as) {
48         mq.push(a);
49         cout << mq << '\n';
50     }
51
52     return 0;
53 }

```

Listing 2: fila monotona.cpp

## 1.2 Pilha

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int main()
5 {
6     stack<int> st;
7
8     st.push(3); // Adiciona elemento na pilha
9     st.pop();  // Remove elemento no topo
10    st.empty(); // Verifica se a pilha está vazia
11    st.size();  // Retorna tamanho da pilha
12    st.top();   // Retorna valor no topo da pilha
13
14    // Exemplo de delimitador com pilha
15    string n, resultado = "";
16    stack<char> st;
17    cin >> n;
18    for(size_t i = 0; i < n.length(); i++)
19    {
20        if(n[i] == '(')
21        {
22            st.push(n[i]);
23            resultado += '(';
24        }
25        else if(n[i] == ')' && st.size() > 0)
26        {
27            st.pop();
28            resultado += ')';
29        }
30        else if(n[i] == ')' && st.size() == 0)
31        {
32            resultado = '(' + resultado + ')';
33        }
34    }
35    while(st.size() > 0)
36    {
37        resultado += ')';
38        st.pop();

```

```

39     }
40     cout << resultado << '\n';
41 }

```

Listing 3: pilha.cpp

## Pilha monótona

Seja  $P$  uma pilha de elementos do tipo  $T$ . A pilha  $P$  é dita **monótona** se, quando extraídos todos os elementos de  $P$ , eles formam uma sequência  $x_1, x_2, \dots, x_N$ , onde  $x_i$  é o elemento obtido na  $i$ -ésima extração, tais que a função  $F : \mathbb{N} \rightarrow T$ , com  $f(i) = x_i$ , é monótona. A pilha  $P$  será **não-decrescente** se  $f$  for **não-crescente**; caso contrário,  $P$  será **não-decrescente**.

- É possível determinar o maior elemento à esquerda para todos os elementos de uma sequência  $a_1, a_2, \dots, a_N$  em  $O(N^2)$  por meio de uma busca completa.
- Para cada índice  $i$ , é preciso avaliar todos os elementos  $a_j$ , com  $j = 1, 2, \dots, i - 1$ .
- Contudo, é possível determinar estes valores em  $O(N)$  com uma modificação no método de inserção de uma pilha não-crescente.
- A inserção em uma pilha não-crescente ocorre em duas etapas: manutenção do invariante e inserção do novo elemento.
- Finalizada a manutenção do invariante, os elementos que restam na pilha são todos maiores do que  $a_i$  e o elemento do topo será o maior elemento à esquerda de  $a_i$ .
- Em algumas implementações são mantidos os índices e não os valores da sequência propriamente ditos (ou pares com ambas informações).

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  template <typename T>
5  class MonoStack {
6  public:
7      void push(const T& x) {
8          while (not st.empty() and st.top() > x)
9              st.pop();
10         st.emplace(x);
11     }
12
13     void pop() {
14         st.pop();
15     }
16
17     auto top() const {
18         return st.top();
19     }
20
21     bool empty() const {
22         return st.empty();
23     }
24
25 private:
26     stack<T> st;
27 };
28
29 template <typename T>
30 ostream& operator<< (ostream& os, const MonoStack<T>& ms) {
31     auto temp(ms);
32     while (not temp.empty()) {
33         cout << temp.top() << ' ';
34         temp.pop();
35     }
36     return os;
37 }
38
39 int main() {
40     vector<int> as{1, 4, 3, 4, 2, 1, 3};

```

```

41 MonoStack<int> ms;
42
43 for (auto& a : as) {
44     ms.push(a);
45     cout << ms << '\n';
46 }
47
48 return 0;
49 }

```

Listing 4: pilha monotona.cpp

## 1.3 Vector

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int main()
5 {
6     int n = 5;
7     vector<int> vet(n, 0); // Cria vetor de tamanho 5, com todos valores = 0
8
9     vet.push_back(5); // Adiciona valor no fim do vetor
10    vet.pop_back(); // Remove ultimo valor do vetor
11    vet.size(); // Retorna tamanho do vetor
12    vet.clear(); // Remove todos os elementos mas não libera a memória alocada
13
14    vector<vector<int>> matriz(n, vector<int>(n, -2)); // Declara Matriz de tamanho NxN
15    for (int i = 0; i < n; i++) { // percorre as linhas
16        for (int j = 0; j < n; j++) { // percorre as colunas
17            cout << matriz[i][j]; // acessa elemento [i][j]
18        }
19    }
20 }

```

Listing 5: vector.cpp

## 2 Outros

### 2.1 Soma De Prefixos

Listing 6: soma de prefixos.cpp