# Vector e Matrizes

**Kaio Christaldo**
**Fabricio Matsunaga**

# Apresentação Problema Motivador

## Fechadura

Por OBI - Olimpíada Brasileira de Informática 2014 🇧🇷 Brazil

**Timelimit: 1**

---

Joãozinho estava um dia chegando em casa quando percebeu que havia perdido a chave da porta. Desesperado, ele resolveu pedir ajuda a seu amigo Roberto, que em poucos segundos conseguiu abrir a porta usando suas ferramentas.

Admirado com a velocidade em que seu amigo conseguiu abrir a porta de sua casa sem a chave, ele decidiu perguntar como ele tinha conseguido aquilo. Roberto explicou que a fechadura da casa de Joãozinho é baseada em um sistema de pinos de tamanhos diferentes que, uma vez alinhados na mesma altura M, possibilitam a abertura da porta.

Uma fechadura é um conjunto de N pinos dispostos horizontalmente que podem ser movimentados para cima ou para baixo com o auxílio de uma chave de metal que, ao ser inserida dentro da fechadura, pode aumentar ou diminuir em 1mm, simultaneamente, a altura de quaisquer dois pinos consecutivos.

Joãozinho como um exemplar perfeccionista decidiu desbloquear sua fechadura na menor quantidade de movimentos, onde cada movimento consiste em escolher dois pinos consecutivos da fechadura e aumentar ou diminuir a altura dos dois pinos em 1mm. Após todos os pinos possuírem altura exatamente igual a M, a fechadura é desbloqueada.

### Entrada

A primeira linha da entrada contém dois inteiros **N** ($1 \leq \mathbf{N} \leq 1000$) e **M** ($1 \leq \mathbf{M} \leq 100$) representando, respectivamente, a quantidade de pinos da fechadura e a altura em que eles devem ficar para a fechadura ser desbloqueada.

A segunda linha da entrada contém **N** ($1 \leq \mathbf{N_i} \leq 100$) inteiros, representando as alturas dos pinos da fechadura.

### Saída

Seu programa deve imprimir uma linha contendo um inteiro representando a quantidade mínima de movimentos para desbloquear a fechadura.

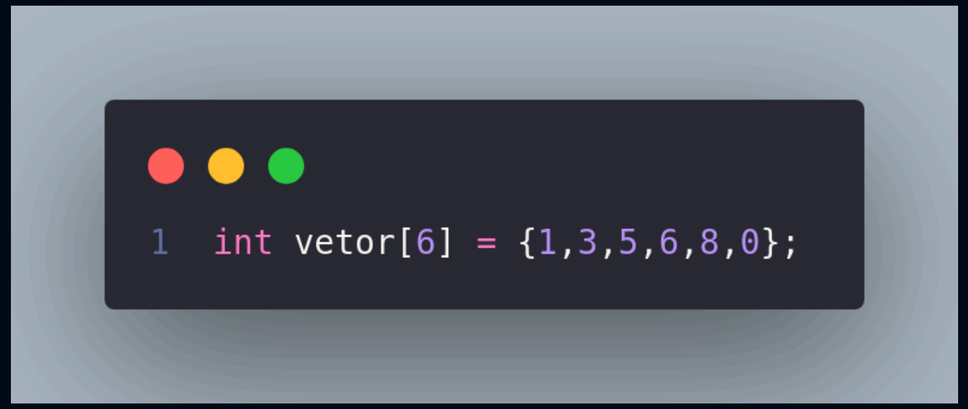
**2449 – Fechadura**

# O que é um Vetor?

Um vetor ou Array na programação é uma estrutura de dado onde armazenamos um conjunto de dados de um mesmo tipo, em uma sequência de memória contínua.

Cada elemento pode ser acessado diretamente através de um índice, onde o primeiro elemento é o 0.

## Declaração de um vetor em c.



# Template <Vector>

**Vector em C++ é como um array dinamicamente alocado.**

**Ambos são estruturas de dados usadas para armazenar múltiplos elementos de mesmo tipo de dados.**

**A diferença entre eles é que um vetor não pode modificar o seu tamanho, você não pode adicionar ou remover elementos de um vetor.**

# Template <Vector>

```
vector<string> cars = {"Volvo", "BMW", "Ford", "Mazda"};

    // Print vector elements
    for (string car : cars) {
        cout << car << "\n";
    }
```

# Template <Vector>

```
// Get the first element
    cout << cars[0];

    // Get the third element
    cout << cars.at(2);

    // Get the first element
    cout << cars.front();

    // Get the last element
    cout << cars.back();
```

# Template <Vector>

```
cars[0] = "Opel";

cars.at(0) = "Opel";
```

```
vector<string> cars = {"Volvo", "BMW", "Ford", "Mazda"};

cars.push_back("Tesla");

vector<string> cars = {"Volvo", "BMW", "Ford", "Mazda"};

cars.pop_back();
```

# Template <Vector>

```cpp
vector<string> cars = {"Volvo", "BMW", "Ford", "Mazda"};

cout << cars.size();
```

```cpp
vector<string> cars;
cout << cars.empty(); // Outputs 1 (The vector is empty)

vector<string> cars = {"Volvo", "BMW", "Ford", "Mazda"};
cout << cars.empty(); // Outputs 0 (not empty)
```

# Template <Vector>

```cpp
vector<string> cars = {"Volvo", "BMW", "Ford", "Mazda"};
for (int i = 0; i < cars.size(); i++) {
    cout << cars[i] << "\n";
}

for (string car : cars) {
    cout << car << "\n";
}
```

# Resolução do Problema Motivador

# Resolução do Problema Motivador

```cpp
#include <iostream>
#include <vector>
#include <stdlib.h>

using namespace std;

int main(int argc, char const *argv[])
{
    int n, altura;
    cin >> n >> altura;
    //scanf("%d %d", &n, &altura);

    vector<int> v;

    for (int i = 0; i < n; i++)
    {
        int rotulo;
        cin >> rotulo;
        v.push_back(rotulo);
    }
    int movimentos = 0;
```

```cpp
for (int i = 0; i < n-1 ; i++)
{
    if (v.at(i) > altura){
        movimentos += v.at(i) - altura;
        v.at(i + 1) -= v.at(i) - altura;

    }else if (v.at(i) < altura){
        movimentos += altura - v.at(i);
        v.at(i + 1) += altura - v.at(i);
    }
}
movimentos += abs(altura - v.at(n-1));

cout << movimentos << endl;
```

# Matrizes

# Apresentação Problema Motivador

beecrowd | 1383

## Sudoku

Maratona de Programação IME-USP 🇧🇷 Brasil

**Timelimit: 1**

---

O jogo de Sudoku espalhou-se rapidamente por todo o mundo, tornando-se hoje o passatempo mais popular em todo o planeta. Muitas pessoas, entretanto, preenchem a matriz de forma incorreta, desrespeitando as restrições do jogo. Sua tarefa neste problema é escrever um programa que verifica se uma matriz preenchida é ou não uma solução para o problema.

A matriz do jogo é uma matriz de inteiros 9 x 9 . Para ser uma solução do problema, cada linha e coluna deve conter todos os números de 1 a 9. Além disso, se dividirmos a matriz em 9 regiões 3 x 3, cada uma destas regiões também deve conter os números de 1 a 9. O exemplo abaixo mostra uma matriz que é uma solução do problema.

$$\left(\begin{array}{ccc|ccc|ccc} 1 & 3 & 2 & 5 & 7 & 9 & 4 & 6 & 8 \\ 4 & 9 & 8 & 2 & 6 & 1 & 3 & 7 & 5 \\ 7 & 5 & 6 & 3 & 8 & 4 & 2 & 1 & 9 \\ \hline 6 & 4 & 3 & 1 & 5 & 8 & 7 & 9 & 2 \\ 5 & 2 & 1 & 7 & 9 & 3 & 8 & 4 & 6 \\ 9 & 8 & 7 & 4 & 2 & 6 & 5 & 3 & 1 \\ \hline 2 & 1 & 4 & 9 & 3 & 5 & 6 & 8 & 7 \\ 3 & 6 & 5 & 8 & 1 & 7 & 9 & 2 & 4 \\ 8 & 7 & 9 & 6 & 4 & 2 & 1 & 5 & 3 \end{array}\right)$$

[1383 – Sudoku]

# Onde vou usar?

- Busca em matriz (DFS/BFS) para problemas de grafos.
- Multiplicação de matrizes em problemas de exponenciação de matrizes.
- Uso de vector para simular tabuleiros de jogo.
- Entre outros

# Declaração e Inicialização

```cpp
int n; cin >> n;

vector<vector<int>> matriz;

vector<vector<int>> matriz(n); // Necessita definir tamanho das colunas

vector<vector<int>> matriz = {{1,2,3}, {1,2,3}};

vector<vector<int>> matriz(n, vector<int>(n));

vector<vector<int>> matriz(n, vector<int>(n, -1));
```

# Declaração e Inicialização

```cpp
vector<vector <int>> matriz;

matriz.resize(N, vector<int>(M, 0)); // M e N são constantes definidas no #define
```

- **matriz.resize() tem custo O(n)**

# Entrada e Saída de Matrizes

## Leitura das Entradas:

```cpp
vector<vector <int>> matriz;
matriz.resize(N, vector<int>(M));

for (int i = 0; i < N; ++i)
    for (int j = 0; j < M; ++j)
        cin >> matriz[i][j];
```

# Entrada e Saída de Matrizes

## Imprimindo Saidas:

```cpp
vector<vector <int>> matriz(3, vector<int>(3, -1));

for (auto& linha : matriz) {
    for (auto& elemento : linha)
        cout << elemento << " ";
    cout << endl;
}
```

# Operações Básicas

## Soma de duas matrizes:

```cpp
vector<vector<int>> C(N, vector<int>(M, 0));

for (int i = 0; i < N; i++)
    for (int j = 0; j < M; j++)
        C[i][j] = A[i][j] + B[i][j]; // A e B são matrizes preenchidas
```

# Operações Básicas

## Transposição de matriz:

```cpp
vector<vector<int>> transposta(M, vector<int>(N));

for (int i = 0; i < N; i++)
    for (int j = 0; j < M; j++)
        transposta[j][i] = matriz[i][j];
```

# Vetor Unidimensional

## Preenchendo  o vector

```cpp
vector<int> matriz(N * M, 0);  // Inicializa com zeros

// Preenchendo a matriz
for (int i = 0; i < N; i++)
    for (int j = 0; j < M; j++)
        cin >> matriz[i * M + j]; // Exemplo: valores baseados em i e j
```

# Vetor Unidimensional

## Imprimindo vector

```cpp
vector<int> matriz(N * M, 0);  // Inicializa com zeros

// Preenchendo a matriz
for (int i = 0; i < N; i++) {
    for (int j = 0; j < M; j++)
        cout << matriz[i * M + j] << " "; // Indice para acesso: i * M + j
    cout << '\n';
}
```

# Vetor Unidimensional

## Vantagens

- Acesso sequencial melhora a performace
- Menos Overhead – vector<vector<int>> tem ponteiros extras
- Facilidade para Copiar e Ordenar – vector<int> pode ser copiado e manipulado diretamente

## Desvantagens

- Erro comum: esquecer M no cálculo do índice pode gerar acessos incorretos e segfaults.
- redimensionar a matriz dinamicamente (exemplo: aumentar número de colunas)
- Difícil Leitura e Depuração

```
1  int valor = matriz[i * M + j];  // Acesso correto
2
```

```
1  cout << matriz[i][j]; // Com matriz bidimensional
2  cout << matriz[i * M + j]; // Com vetor unidimensional (menos legível)
```

# Resolução do Problema Motivador

## 1383 - Sudoku

- A resolução estará disponível no Drive. Tente resolver por conta própria e, se precisar, compare com a solução! 😊

# Lista de Exercícios

1177 - Prenchimento de Vetor II

1175 - Troca em Vetor I

1174 - Seleção em Vetor I

1259 - Pares e Ímpares

1566 - Altura

1435 - Matriz Quadrada I

- Se tiver alguma dúvida ou dificuldade na resolução de algum exercício, sinta-se à vontade para perguntar! 😊