ESCOLA DE PRIMAVERA DA MARATONA DE PROGRAMAÇÃO

PROMOÇÃO:

CEFET-MG

UFMG

UFV

UNIFEI

USP Universidade de São Paulo

UFU

APOIO:

beecrowd

SBC Sociedade Brasileira de Computação

Grupo de Computação Competitiva

VETORES COM VECTOR

Por: *Gabriel Alves Theodoro*

CONTEÚDOS

01 - PROBLEMA MOTIVADOR

Dado uma fechadura com N pinos, quantos movimentos são necessários para desbloquear a fechadura?

A fechadura é desbloqueada quando todos os pinos tem o valor K.

O valor de cada pino é alterado de 2 em 2, ou seja, se você aumentar o pino n, você terá que alterar o pino n+1 também. Aumentando ambos ou diminuindo ambos.

02 - DEFINIÇÃO DO ALGORITMO

O que é um vetor?

Um vetor (ou array) em programação é uma estrutura de dados que armazena vários elementos do mesmo tipo em uma sequência contínua de memória.

Cada elemento em um vetor pode ser acessado por um índice, onde o primeiro elemento tem o índice 0.

Exemplo: int arr[5] = {1, 2, 3, 4, 5};

02 - DEFINIÇÃO DO ALGORITMO

Um vector, nada mais é do que um contêiner da biblioteca padrão que implementa um array dinâmico, armazenando e manipulando elementos de maneira sequencial.

Caracteristicas:

- Redimensionamento dinâmico: Ao contrário dos arrays tradicionais em C++, os vetores podem crescer ou encolher automaticamente conforme necessário.
- Acesso rápido: O acesso a elementos individuais pode ser feito de forma eficiente usando índices (como em arrays).
- Métodos úteis:

  push_back(): Adiciona um elemento ao final do vetor.
  size(): Retorna o número de elementos no vetor.
  empty(): Verifica se o vetor está vazio.
  clear(): Remove todos os elementos do vetor.

02 - DEFINIÇÃO DO ALGORITMO

Um vetor pode ser inicializado da seguinte forma:
int vetor[10];

Só que há certas vantagens em declarar o vetor como um std::vector. Algumas vantagens são:

1. **Tamanho dinâmico:** Redimensiona automaticamente.
2. **Gerenciamento automático de memória:** Evita erros manuais.
3. **Acesso seguro:** Verifica limites com at().
4. **Facilidade de uso:** Funções como push_back() e resize().
5. **Compatibilidade com a STL:** Funciona bem com algoritmos padrão.
6. **Iteradores:** Navegação e manipulação mais seguras.

03 - FUNCIONAMENTO DO ALGORITMO

1. push_back(const T& value)
Adiciona um elemento ao final do vetor.

```
numeros.push_back(10);
```

2. pop_back()
Remove o último elemento do vetor.

```
numeros.pop_back();
```

3. size()
Retorna o número de elementos no vetor.

```
int tam = numeros.size();
```

4. empty()
Verifica se o vetor está vazio.

```
if (numeros.empty()) { /* ... */ }
```

5. clear()
Remove todos os elementos do vetor.

```
numeros.clear();
```

6. at(size_type pos)
Retorna uma referência ao elemento na posição especificada, com verificação de limites.

```
int valor = numeros.at(1);
```

7. operator[]
Acessa o elemento no índice especificado sem verificação de limites.

```
int valor = numeros[1];
```

8. insert(iterator pos, const T& value)
Insere um elemento antes da posição especificada.

```
numeros.insert(numeros.begin() + 1, 15);
```

03 - FUNCIONAMENTO DO ALGORITMO

9. erase(iterator pos)
Remove o elemento na posição especificada.

```
numeros.erase(numeros.begin() + 1);
```

10. reserve(size_type new_cap)
Reserva capacidade no vetor, garantindo que ele possa armazenar pelo menos o número especificado de elementos sem realocação.

```
numeros.reserve(100);
```

11. capacity()
Retorna a capacidade atual do vetor, ou seja, o número de elementos que ele pode armazenar sem realocação.

```
int cap = numeros.capacity();
```

12. resize(size_type count)
Redimensiona o vetor para conter o número especificado de elementos. Se o tamanho aumentar, novos elementos são inicializados com o valor padrão.

```
numeros.resize(5);
```

13. front()
Retorna uma referência ao primeiro elemento do vetor.

```
int primeiro = numeros.front();
```

14. back()
Retorna uma referência ao último elemento do vetor.

```
int ultimo = numeros.back();
```

04 - ALGORITMO

```cpp
#include <iostream>
#include <vector>

using namespace std;

int main() {
    vector<int> numeros;

    if (numeros.empty()) {
        cout << "O vetor está vazio." << endl;
    }

    numeros.push_back(10);
    numeros.push_back(20);
    numeros.push_back(30);

    cout << "Tamanho do vetor: " << numeros.size() << endl;

    cout << "Elementos do vetor: ";
    for (int i = 0; i < numeros.size(); ++i) {
        cout << numeros[i] << " ";
    }
    cout << endl;

    numeros.clear();

    if (numeros.empty()) {
        cout << "O vetor está vazio após clear()." << endl;
    }

    return 0;
}
```

05 - RESOLUÇÃO DO PROBLEMA MOTIVADOR

Se temos que alterar os pinos de 2 em 2, basta percorrer o vetor e caso o pino não tenha o valor K, somamos ou diminuimos dele a diferença entre o valor dele e K. Exemplo:

Se o pino tem o valor 25 e K = 50, então a diferença entre os dois é 25. Logo, teremos que somar 25 para o valor do pino. Isso equivale a 25 movimentos na fechadura.

Lembre-se: Se mover o pino n, move também o n+1. O pino n+1 nesse caso também receberia 25.

05 - RESOLUÇÃO DO PROBLEMA MOTIVADOR

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

using namespace std;

int main() {
    int altura, n, i, movimentos = 0;

    cin >> n >> altura;
    vector<int> v;

    // Usando push_back para adicionar elementos ao vetor
    for(i = 0; i < n; i++) {
        int rótulo;
        cin >> rótulo;
        v.push_back(rótulo); // Adiciona rótulo ao vetor
    }

    for(i = 0; i < n - 1; i++) {
        if(v[i] < altura) {
            movimentos += (altura - v[i]);
            v[i + 1] += (altura - v[i]);
        }
        else if(v[i] > altura) {
            movimentos += (v[i] - altura);
            v[i + 1] -= (v[i] - altura);
        }
    }
    movimentos += abs(altura - v[n - 1]);

    cout << movimentos << endl;

    return 0;
}
```

06 - OUTRAS APLICAÇÕES

Vector para algoritmos como: Dijkstra, DFS, BFS. (também é necessário o conhecimento de matrizes com vector).

Vector para várias outras aplicações, como:
Iterador;
Saber o tamanho do vetor(size);
Limpar um vetor (clear);
Retirar um último elemento do vetor (pop_back).

06 - OUTRAS APLICAÇÕES

```cpp
#include <iostream>
#include <vector>

using namespace std;

int main() {
    vector<int>arr;

    arr.push_back(1);
    arr.push_back(2);
    arr.push_back(3);
    arr.push_back(4);
    arr.push_back(5);

    for(int i : arr){
        cout << i << endl;
    }

    return 0;
}
```

OBRIGADO PELA ATENÇÃO

Grupo de Computação Competitiva

GCC.CEFETMG

gcc-cefetmg.github.io/Site-GCC

@gcc.cefetmg