

# Vetores Dinâmicos

## Estrutura de Dados e Algoritmos



Prof. Daniel Saad Nogueira  
Nunes

IFB – Instituto Federal de Brasília,  
Campus Taguatinga



# Sumário

---

- 1 Introdução
- 2 Vetores dinâmicos
- 3 Exemplos



# Sumário

---

## 1 Introdução



# Introdução

---

- Com nosso conhecimento sobre alocação dinâmica de memória, podemos projetar um vetor dinâmico.
- A ideia é que o vetor cresça e diminua quando necessário.



# Operações

---

As seguintes operações devem ser suportadas pelo vetor dinâmico:

- `push_back(x)`: insere um elemento  $x$  ao final do vetor.
- `pop_back()`: remove o último elemento.
- `front()`: retorna o primeiro elemento.
- `back()`: retorna o último elemento do vetor.
- `size()`: retorna o número de elementos do vetor.



# Sumário

---

## 2 Vetores dinâmicos



# Sumário

---

## 2 Vetores dinâmicos

- Definição
- Inicialização
- Funções auxiliares
- Inserção
- Remoção
- Acesso
- Limpeza
- Análise



# Definição

---

```
typedef struct {  
    int *v;           // vetor dinâmico  
    size_t capacity; // capacidade do vetor dinâmico  
    size_t size;      // tamanho do vetor dinâmico  
} dynamic_array_t;
```

- `v`: contém os dados propriamente ditos.
- `capacity`: capacidade máxima do vetor.
- `size`: número de elementos do vetor, sempre menor ou igual à capacidade.





# Definição

---

- A ideia é que, conforme o vetor cresça ou diminua, a capacidade seja modificada de acordo.
- O tamanho do vetor ( `size` ) indica o número de elementos válidos presentes no vetor. O tamanho pode ser menor ou igual à capacidade, visto que uma capacidade maior que o tamanho aponta a possibilidade de inserir mais elementos no vetor.
- Para redimensionar o vetor, utilizaremos a função `realloc`.



# Sumário

---

## 2 Vetores dinâmicos

- Definição
- Inicialização
- Funções auxiliares
- Inserção
- Remoção
- Acesso
- Limpeza
- Análise



# Inicialização

---

- A função de inicialização receberá um vetor dinâmico não inicializado e o iniciará.
- Inicialmente, ele alocará um espaço de 4 inteiros para o vetor e preencherá as variáveis `size` e `capacity` de acordo.



# Inicialização

---

```
4 void dynamic_array_initialize(dynamic_array_t **arr) {
5     /** Utilizamos uma capacidade de 4 por padrão na criação
6      * do vetor dinâmico
7      */
8     (*arr) = mallocx(sizeof(dynamic_array_t));
9     (*arr)->capacity = 4;
10    (*arr)->size = 0;
11    (*arr)->v = mallocx(sizeof(int) * 4);
12 }
```



# Inicialização

---

- Utilizamos um ponteiro de ponteiro `arr`.
- Motivo: modificar um ponteiro por referência.
- A função de inicialização recebe um endereço de ponteiro para que a variável original possa ser modificada.



# Sumário

---

## 2 Vetores dinâmicos

- Definição
- Inicialização
- Funções auxiliares
- Inserção
- Remoção
- Acesso
- Limpeza
- Análise



## Funções auxiliares

---

Para recuperar o tamanho de um vetor dinâmico, basta acessar sua variável `size`.

```
69  size_t dynamic_array_size(dynamic_array_t *arr) {  
70      return arr->size;  
71  }
```



## Funções auxiliares

---

Para recuperar a capacidade de um vetor dinâmico, basta acessar sua variável `capacity`.

```
73  size_t dynamic_array_capacity(dynamic_array_t *arr) {  
74      return arr->capacity;  
75  }
```





# Sumário

---

## 2 Vetores dinâmicos

- Definição
- Inicialização
- Funções auxiliares
- **Inserção**
- Remoção
- Acesso
- Limpeza
- Análise



# Inserção

---

## Inserção ao final

- Para inserir um elemento ao final do vetor, primeiro verificamos se o tamanho é igual à capacidade.
- Em caso afirmativo, aumentamos a capacidade: ela será duplicada. Duplicar a capacidade evita termos que chamar `realloc` frequentemente.
- Em seguida, basta colocar o elemento na posição indicada por `size` e incrementar essa variável.



# Inserção

```
24 void dynamic_array_push_back(dynamic_array_t *arr, int x) {
25     /** o tamanho do vetor dinâmico seja igual a sua capacidade,
26     * duplicamos a capacidade e realocamos o tamanho do vetor.
27     /**/
28     if (arr->size == arr->capacity) {
29         dynamic_array_double_capacity(arr);
30     }
31     /**
32     * O elemento é inserido ao final.
33     */
34     arr->v[arr->size] = x;
35     arr->size++;
36 }
```



# Inserção

---

```
14 void dynamic_array_double_capacity(dynamic_array_t *arr) {  
15     arr->capacity *= 2;  
16     arr->v = reallocx(arr->v, sizeof(int) * arr->capacity);  
17 }
```



# Sumário

---

## 2 Vetores dinâmicos

- Definição
- Inicialização
- Funções auxiliares
- Inserção
- **Remoção**
- Acesso
- Limpeza
- Análise



# Remoção

---

## Remoção do final

- Para remover do final, basta decrementar a variável `size`.
- Caso o tamanho do vetor dinâmico seja muito menor do que a sua capacidade ( $\frac{1}{4}$ ), reduzimos a capacidade pela metade.
- Nunca a capacidade é redimensionada para abaixo de 4.



# Inserção

---

```
38 void dynamic_array_pop_back(dynamic_array_t *arr) {
39     /** Removemos o último elemento do vetor. Para isto só é necessário
40     * decrementar o seu tamanho.
41     * Caso o tamanho atual esteja à 1/4 da capacidade máxima do vetor,
42     * o vetor é redimensionado para metade do seu tamanho.
43     * Nunca redimensionamos a capacidade para abaixo de 4.
44     */
45     if (arr->size == arr->capacity / 4 && arr->capacity > 4) {
46         dynamic_array_halve_capacity(arr);
47     }
48     arr->size--;
49 }
```



# Inserção

---

```
19 void dynamic_array_half_capacity(dynamic_array_t *arr) {  
20     arr->capacity /= 2;  
21     arr->v = reallocx(arr->v, sizeof(int) * arr->capacity);  
22 }
```





# Sumário

---

## 2 Vetores dinâmicos

- Definição
- Inicialização
- Funções auxiliares
- Inserção
- Remoção
- **Acesso**
- Limpeza
- Análise



# Acesso

---

- O acesso ao primeiro elemento é simples: `arr->v[0];` .
- O acesso ao último também é simples: `arr->v[arr->size-1];`
- Qualquer outra posição  $i$  também pode ser acessada e modificada através de: `arr->v[i];`



## Acesso ao primeiro elemento

---

```
65  int dynamic_array_front(dynamic_array_t *arr) {  
66      return arr->v[0];  
67  }
```



## Acesso ao último elemento

---

```
61  int dynamic_array_back(dynamic_array_t *arr) {  
62      return arr->v[arr->size - 1];  
63  }
```



# Sumário

---

## 2 Vetores dinâmicos

- Definição
- Inicialização
- Funções auxiliares
- Inserção
- Remoção
- Acesso
- **Limpeza**
- Análise



# Limpeza

---

- Para deletar completamente o vetor dinâmico da memória, as desalocações devem ser feitas de maneira inversa ao da inicialização.
- Primeiro liberamos o espaço pontado por `v`.
- Em seguida, liberamos o espaço apontado por `arr`.
- Como o objetivo é modificar o ponteiro `arr`, este é passado por referência (ponteiro de ponteiro).



# Limpeza

---

```
51 void dynamic_array_delete(dynamic_array_t **arr) {  
52     /**  
53      * Para deletar completamente o vetor dinâmico, basta remover  
54      * a área apontada por v e a área apontada pelo próprio vetor dinâmico  
55      */  
56     free((*arr)->v);  
57     free(*arr);  
58     *arr = NULL;  
59 }
```



# Sumário

---

## 2 Vetores dinâmicos

- Definição
- Inicialização
- Funções auxiliares
- Inserção
- Remoção
- Acesso
- Limpeza
- **Análise**





# Vetores dinâmicos

---

Operação	Complexidade
Inserção ao final	$\Theta(1)$ amortizado
Remoção do final	$\Theta(1)$ amortizado
Acesso	$\Theta(1)$



# Vetores dinâmicos

---

- O custo da inserção ao final e remoção ao final é constante amortizado.
- Em algum momento, o vetor deve ser redimensionado, onde crescerá ou diminuirá por um fator de 2. Mas isso só ocorre uma vez a cada  $n$  operações consecutivas de inserção (ou de remoção), em que  $n$  é a capacidade do vetor.
- Este custo é diluído entre as chamadas, fazendo com que o custo, considerando todas as chamadas seja  $\Theta(1)$  amortizado.



# Sumário

---

## 3 Exemplos



# Exemplo

```
1  #include "dynamic_array.h"
2  #include <stdio.h>
3
4  static const int N = 1000000;
5
6  int main(void) {
7      dynamic_array_t *arr;
8      dynamic_array_initialize(&arr);
9      for (int i = 0; i < N; i++) {
10         int value = rand();
11         dynamic_array_push_back(arr, value);
12         printf("v[%d] = %d\n", i, arr->v[i]);
13     }
14     while (dynamic_array_size(arr) > 0) {
15         printf("v[%zu] = %d\n", dynamic_array_size(arr) - 1,
16             dynamic_array_back(arr));
17         dynamic_array_pop_back(arr);
18     }
19     dynamic_array_delete(&arr);
20     return 0;
21 }
```