

Pilhas

Estruturas de Dados e Algoritmos



**INSTITUTO
FEDERAL**
Brasília

Prof. Daniel Saad Nogueira
Nunes

IFB – Instituto Federal de Brasília,
Campus Taguatinga



Sumário

- 1 Introdução
- 2 Pilhas
- 3 Vetores dinâmicos
- 4 Listas
- 5 Exemplo



Sumário

1 Introdução



Introdução

Pilhas

- Pilhas são um TAD em qual os elementos são mantidos em uma ordem específica. Esta ordem é a ordem LIFO (Last-in-First-Out)
- A ordem LIFO se caracteriza pelo fato dos últimos elementos a fazerem parte da estrutura, também serão os primeiros elementos a deixarem a estrutura.
- Seu uso é interessante quando é necessário acessar elementos na ordem inversa a de inserção.



Pilhas





Operações sobre Pilhas

- Algumas das operações suportadas por uma pilha devem ser:
 - ▶ Empilhar elementos;
 - ▶ Desempilhar elementos;
 - ▶ Acessar o topo da pilha;
 - ▶ Verificar o tamanho da pilha.
 - ▶ Verificar se a pilha está vazia.



Sumário

2 Pilhas



Implementação de pilhas

- Pilhas podem ser implementadas através de vetores dinâmicos ou de listas.



Sumário

3 Vetores dinâmicos

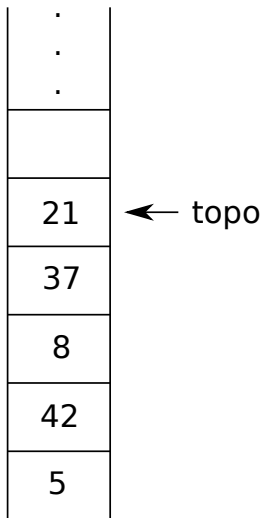


Implementação de Pilhas

- Pilhas podem ser implementadas através de vetores dinâmicos.



Implementação de Pilhas sobre Vetores





Implementação de Pilhas sobre Vetores

- Empilhar é equivalente a inserir ao final do vetor dinâmico.
- Desempilhar é equivalente a remover o último elemento do vetor dinâmico.
- Acessar o topo é equivalente a acessar o último elemento do vetor dinâmico.
- Verificar o tamanho da pilha é equivalente a verificar o tamanho do vetor dinâmico.
- Verificar se a pilha é vazia é equivalente a verificar se o tamanho do vetor dinâmico é zero.



Representação de Pilhas sobre Vetores

- Através da nossa implementação de vetores dinâmicos, implementar uma pilha é trivial.



Sumário

3 Vetores dinâmicos

- Definição
- Inicialização
- Funções auxiliares
- Empilhar
- Desempilhar
- Acessar o topo
- Limpeza
- Análise



Pilhas: definição

```
6 typedef struct stack_t {  
7     int* stack;  
8     size_t capacity;  
9     size_t size;  
10 } stack_t;
```



Sumário

3 Vetores dinâmicos

- Definição
- **Inicialização**
- Funções auxiliares
- Empilhar
- Desempilhar
- Acessar o topo
- Limpeza
- Análise



Pilhas: Inicialização

```
8 void stack_initialize(stack_t **s) {
9     *s = mallocx(sizeof(stack_t));
10    (*s)->capacity = 4;
11    (*s)->stack = mallocx(sizeof(int) * (*s)->capacity);
12    (*s)->size = 0;
13 }
```



Sumário

- 3 Vetores dinâmicos
 - Definição
 - Inicialização
 - Funções auxiliares
 - Empilhar
 - Desempilhar
 - Acessar o topo
 - Limpeza
 - Análise



Pilhas: verificar o tamanho

```
54  size_t stack_size(stack_t *s) {  
55      return s->size;  
56  }
```



Pilhas: verificar se é vazia

```
50 bool stack_empty(stack_t *s) {  
51     return stack_size(s) == 0;  
52 }
```



Sumário

3 Vetores dinâmicos

- Definição
- Inicialização
- Funções auxiliares
- **Empilhar**
- Desempilhar
- Acessar o topo
- Limpeza
- Análise



Pilhas: empilhar

```
31 void stack_push(stack_t *s, int data) {
32     if (s->size == s->capacity) {
33         stack_expand(s);
34     }
35     s->stack[s->size++] = data;
36 }
```



Pilhas: empilhar

```
15 static void stack_expand(stack_t *s) {  
16     s->capacity *= 2;  
17     s->stack = reallocx(s->stack, sizeof(int) * s->capacity);  
18 }
```



Sumário

3 Vetores dinâmicos

- Definição
- Inicialização
- Funções auxiliares
- Empilhar
- **Desempilhar**
- Acessar o topo
- Limpeza
- Análise



Pilhas: desempilhar

```
43 void stack_pop(stack_t *s) {  
44     if (s->size == s->capacity / 4 && s->capacity > 4) {  
45         stack_shrink(s);  
46     }  
47     s->size--;  
48 }
```



Pilhas: desempilhar

```
20 static void stack_shrink(stack_t *s) {  
21     s->capacity /= 2;  
22     s->stack = reallocx(s->stack, sizeof(int) * s->capacity);  
23 }
```



Sumário

3 Vetores dinâmicos

- Definição
- Inicialização
- Funções auxiliares
- Empilhar
- Desempilhar
- Acessar o topo
- Limpeza
- Análise



Pilhas: acessar o topo

```
38 int stack_top(stack_t *s) {  
39     assert(!stack_empty(s));  
40     return s->stack[s->size - 1];  
41 }
```



Sumário

3 Vetores dinâmicos

- Definição
- Inicialização
- Funções auxiliares
- Empilhar
- Desempilhar
- Acessar o topo
- **Limpeza**
- Análise



Pilhas: limpeza

```
25 void stack_delete(stack_t **s) {  
26     free((*s)->stack);  
27     free(*s);  
28     *s = NULL;  
29 }
```



Sumário

3 Vetores dinâmicos

- Definição
- Inicialização
- Funções auxiliares
- Empilhar
- Desempilhar
- Acessar o topo
- Limpeza
- Análise



Pilhas: análise

Complexidade das Operações

Operação	Complexidade
Empilhar	$\Theta(1)$ amortizado
Desempilhar	$\Theta(1)$ amortizado
Verificar topo	$\Theta(1)$ amortizado



Sumário

4 Listas



Implementação de Pilhas

- Pilhas podem ser implementadas por meio de estruturas auto-referenciadas.
- Uma das estruturas que podem prover as funcionalidades de uma pilha é uma lista ligada.
- Basta utilizar a lista de uma maneira muito específica para simular uma pilha.



Implementação de Pilhas sobre Listas

- Utilizando listas, a operação de verificar se a pilha está vazia equivale à verificar se a lista está vazia.
- Para empilhar um elemento, insere-se um elemento na cabeça.
- Para desempilhar um elemento, retira-se da cabeça.
- Para acessar o topo da pilha, a cabeça deve ser acessada.



Sumário

5 Exemplo



Exemplo

```
1  #include "stack.h"
2  #include <stdio.h>
3  #include <string.h>
4
5  int main(void) {
6      stack_t *stack;
7      stack_initialize(&stack);
8      for (int i = 0; i < 1000000; i++) {
9          stack_push(stack, i);
10     }
11     while (!stack_empty(stack)) {
12         printf("%d\n", stack_top(stack));
13         stack_pop(stack);
14     }
15     stack_delete(&stack);
16     return 0;
17 }
```