Estruturas de Dados - Pilha

Tipos abstratos de dados

- Descrevem um conjunto de dados segundo as suas funcionalidades
- Devemos especificar como os dados são manipulados
- Em geral, um tipo abstrato de dados pode usar diferentes estruturas de dados para sua implementação
  - utilizaremos as estruturas de dados já vistas: vetor e lista de nós
- Tipos abstratos:
  - pilha
  - fila
  - heap
  - árvore
  - ⋮

Tipo abstrato de dados: pilha

- Conjunto que permite inclusão e exclusão de elementos com as seguintes propriedades:
  - inclusão de um elemento: o elemento é adicionado ao topo da pilha
  - exclusão de um elemento: o elemento excluído é o que está no topo da pilha
- *last in, first out*
- Exemplos de aplicações:
  - mecanismos de voltar/avançar (ou desfazer/refazer)
  - conversão de decimal para binário
  - análise de expressões matemáticas
  - auxilia em versões iterativas para algumas recorrências

Pilha sequencial

- Estrutura de dados utilizada: vetor
  - implementação eficiente: preencher o vetor "da esquerda para a direita"
  - elementos "a esquerda" estão na base da pilha
  - elementos "a direita" estão no topo da pilha
- Informações de uma pilha sequencial:
  - *tam*: número máximo de elementos na pilha
  - *topo*: índice para o topo da pilha
- Operações a serem analisadas:
  - CriaPilha: criação de uma pilha sequencial
  - Top: consulta o topo da pilha sequencial
  - Empilha: inclusão de um elemento em uma pilha sequencial
  - Desempilha: exclusão de um elemento em uma pilha sequencial

Pilha sequencial: criar

Algoritmo: CriaPilhaSequencial(n)

Entrada: tamanho n da pilha

Saída: pilha sequencial P

1 criar nova pilha sequencial P com um vetor de n posições

2 $P.tam = n$

3 $P.topo = 0$

4 **retorne** P

Complexidade: $O(1)$

Pilha sequencial: checar o topo da pilha

Algoritmo: TopSequencial(P)

Entrada: pilha sequencial P

Saída: valor no topo da pilha, ou um erro se a pilha estiver vazia

```
1 se P.topo == 0 então
2 |   retorne "Erro: pilha vazia!"
3 retorne P[P.topo − 1]
```

Complexidade: $O(1)$

Pilha sequencial: empilhar

Algoritmo: EmpilhaSequencial(P, x)

Entrada: pilha sequencial P, valor x

```
1 se P.topo == P.tam então
2 |   retorne "Erro: pilha cheia!"
3 P[P.topo] = x
4 P.topo = P.topo + 1
```

Complexidade: $O(1)$

Pilha sequencial: desempilhar

Algoritmo: DesempilhaSequencial(P)

Entrada: pilha sequencial P

Saída: valor removido do topo da pilha, ou um erro se a pilha estiver vazia

```
1 se P.topo == 0 então
2 |   retorne "Erro: pilha vazia!"
3 P.topo = P.topo − 1
4 retorne P[P.topo]
```

Complexidade: $O(1)$

Pilha encadeada

- Estrutura de dados utilizada: lista de nós encadeados
- Para acessar a pilha, basta conhecermos o primeiro nó da pilha
  - primeiro nó da pilha: depende da implementação
  - implementação mais eficiente: primeiro nó está no topo da pilha
  - justificativa: não precisamos acessar a base da pilha
- Informações de um nó:
  - *chave*: guarda o elemento
  - *prox*: indica a localização do nó que o sucede na pilha
- Operações a serem analisadas:
  - CriaPilha: criação de uma pilha encadeada
  - Top: consulta o topo da pilha encadeada
  - Empilha: inclusão de um elemento em uma pilha encadeada
  - Desempilha: exclusão de um elemento em uma pilha encadeada

Pilha encadeada: criar

Algoritmo: CriaPilhaEncadeada()

Saída: nó inicial v da pilha encadeada

1 criar novo nó v
2 $v \rightarrow prox = \lambda$
3 **retorne** v

Complexidade: $O(1)$

Pilha encadeada: checar o topo da pilha

Algoritmo: TopEncadeada(v)

Entrada: nó inicial v da pilha encadeada

Saída: nó no topo da pilha, ou λ se a pilha estiver vazia

1 **retorne** $v \rightarrow prox$

Complexidade: $O(1)$

Pilha encadeada: empilhar

Algoritmo: EmpilhaEncadeada(v, x)

Entrada: nó inicial v da pilha encadeada, valor x

1 criar novo nó u

2 $u \rightarrow chave = x$

3 $u \rightarrow prox = v \rightarrow prox$

4 $v \rightarrow prox = u$

Complexidade: $O(1)$

Pilha encadeada: desempilhar

Algoritmo: DesempilhaEncadeada(v)

Entrada: nó inicial v da pilha encadeada

Saída: nó removido do topo da pilha, ou λ se a pilha estiver vazia

```
1 se v→prox == λ então
2 |   retorne λ
3 r = v→prox
4 v→prox = v→prox→prox
5 retorne r
```

Complexidade: $O(1)$

Pilha: aplicações

Algoritmo: BinarioPilha(n)

Entrada: número decimal n

Saída: representação binária de n

```
1 criar nova pilha P vazia
2 enquanto n ≥ 1 faça
3 |   Empilha(P, n mod 2)
4 |   n = n div 2
5 enquanto P não estiver vazia faça
6 |   imprima Desempilha(P)
```

Pilha: aplicações

Algoritmo: HanoiPilha(n, O, D, T)

Entrada: quantidade n de discos, torres O, D e T

Saída: movimentos para a Torre de Hanoi com n discos

```
1  criar nova pilha P vazia
2  Empilha(P, "Hanoi(n, O, D, T)")
3  enquanto P não estiver vazia faça
4  |  x = Desempilha(P)
5  |  se x == "Hanoi(k, A, B, C)" então
6  |  |  se k == 1 então
7  |  |  |  imprima "mover o disco k de A para B"
8  |  |  senão
9  |  |  |  Empilha(P, "Hanoi(k − 1, C, B, A)")
10 |  |  |  Empilha(P, "Mover(k, A, B)")
11 |  |  |  Empilha(P, "Hanoi(k − 1, A, C, B)")
12 |  se x == "Mover(k, A, B)" então
13 |  |  imprima "mover o disco k de A para B"
```