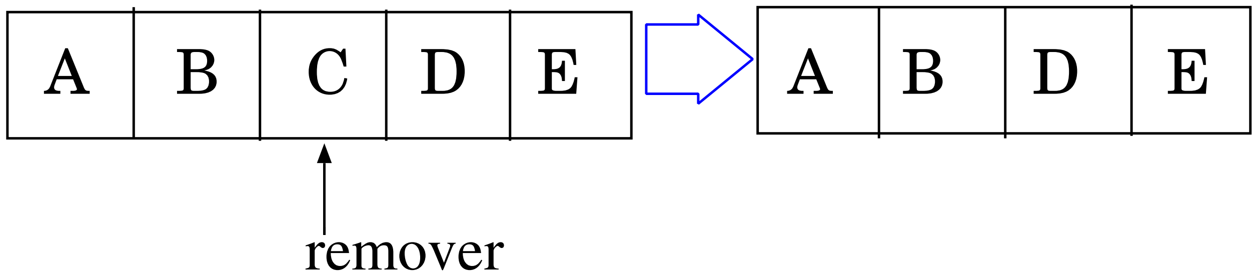
Aula 10: Pilhas

- O conceito de pilha
- Algoritmos de inserção e remoção
- Exemplo: Notação Polonesa

Pilhas

- Uso eficiente de listas:
  inserções e remoções não devem acarretar movimentação de nós.



- Listas, pilhas e deques: inserções e remoções ocorrem somente nas extremidades.

Pilhas

- Uso eficiente de listas:
  inserções e remoções não devem acarretar movimentação de nós.

A	B	D	E

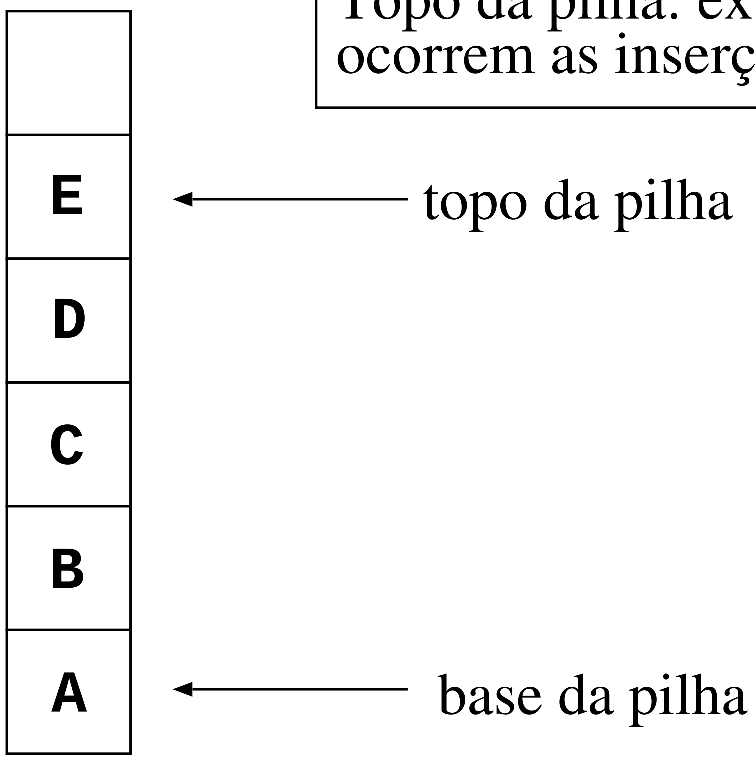
Remover Voltar

- Listas, pilhas e deques: inserções e remoções ocorrem somente nas extremidades.

Pilhas

- Pilhas: inserções e remoções ocorrem em uma <u>mesma</u> extremidade

Topo da pilha: extremidade na qual ocorrem as inserções e remoções.



Pilhas

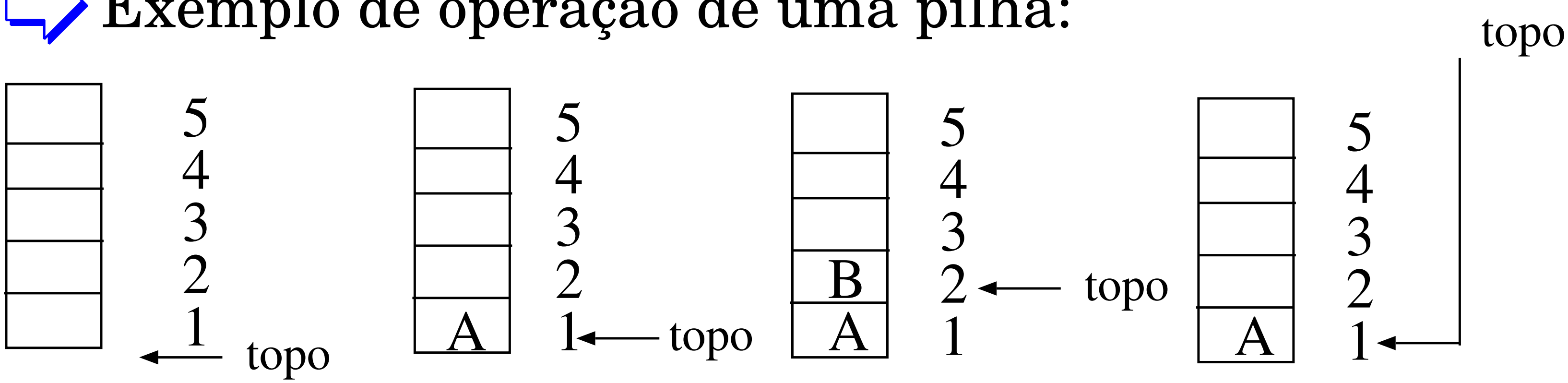
- Formas de armazenamento:
  - Em alocação sequencial:
    - usa vetores;
    - um ponteiro indica a posição do topo da pilha.
- Operações básicas:
  - inserção;
  - remoção.

Pilhas

- Situações extremas:
  - pilha vazia;
  - pilha cheia.
- Operações inválidas:
  - inserção em pilha cheia (<u>overflow</u>)
  - remoção em pilha vazia (<u>underflow</u>)
- Forma de operação:
  - último a entrar, primeiro a sair (LIFO)

Exemplo

Exemplo de operação de uma pilha:

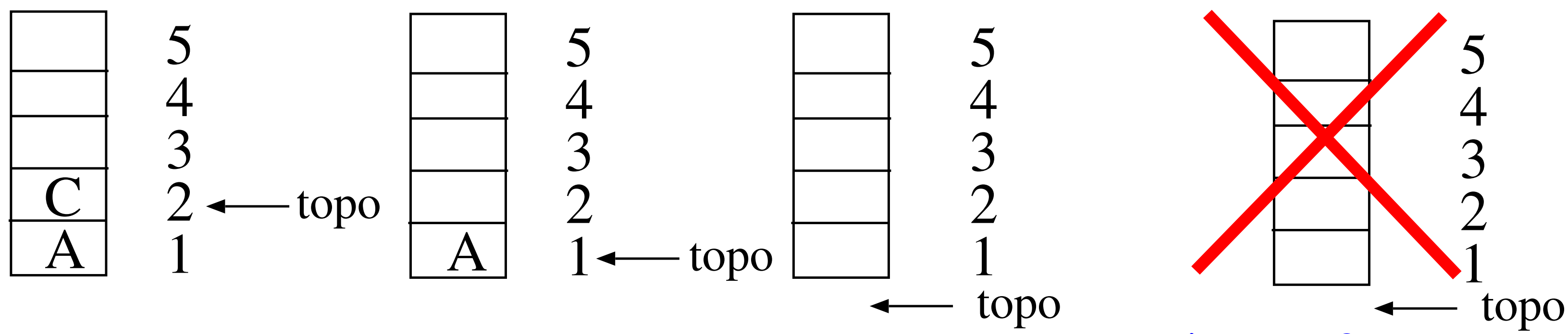


situação 1: inicial (pilha vazia)

situação 2: inserir A

situação 3: inserir B

situação 4: remover



situação 5: inserir C

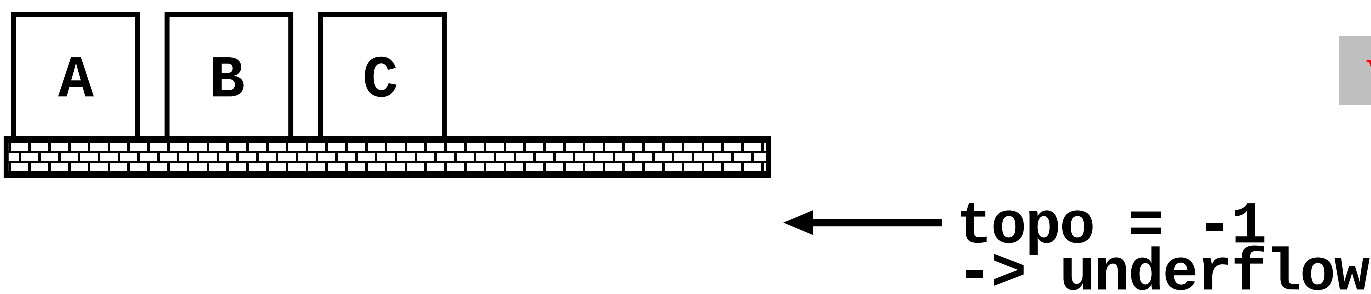
situação 6: remover

situação 7: remover

situação 8: remover (underflow)

Exemplo

- Exemplo de operação de uma pilha



Animar

Voltar

Algoritmo de Inserção

- A pilha se encontra armazenada no vetor `P`
- A variável <u>`topo`</u> indica o topo da pilha
- O vetor `P` possui `M` posições

<u>Algoritmo</u>: inserção na pilha P

```
se topo ≠ M então
     topo := topo + 1
     P[ topo ] := novo_valor
senão overflow
```

- Pilha vazia: `topo` = 0
- Overflow: inserção com `topo` = `M`
- A informação a ser inserida é `novo_valor`.

Algoritmo de Remoção

- A pilha se encontra armazenada no vetor P
- A variável topo indica o topo da pilha
- A situação de pilha vazia é indicada por topo = 0

Algoritmo: remoção na pilha P

```
se topo ≠ 0 então
     valor_recuperado := P[ topo ]
     topo := topo - 1
senão underflow
```

- Pilha vazia: topo = 0
- Overflow: remoção com topo = 0
- A informação a ser removida é transferida para valor_recuperado.

Exercício

Determinar a sequência de nós correspondentes às remoções de uma pilha, nos seguintes casos.

- I 1, I 2, I 3, R, I 4, I 5, R, R, I 6, R, R, R, I 7
- I 1, I 2, I 3, R, R, I 4, R, R, I 5, R, I 6, R, R, I 7

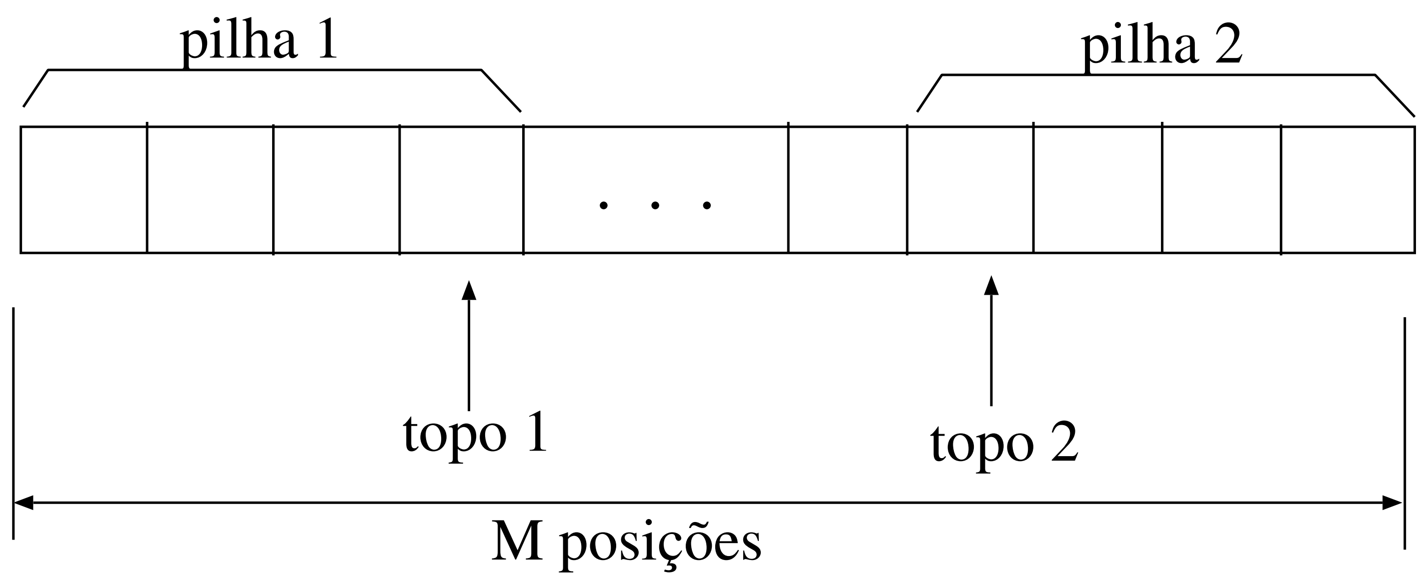
Tempo: 3 minutos

Exercício (solução)

- 3 5 4 6 2 1
- 3 2 4 1 5 6 (underflow)

Exercício

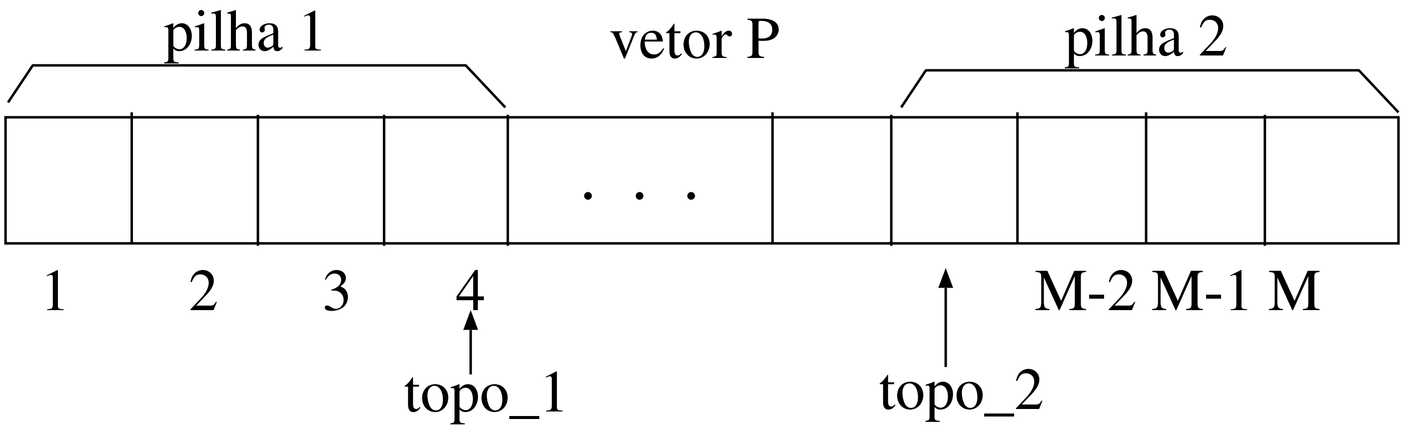
Elaborar algoritmos de inserção e remoção para um conjunto de duas pilhas armazenadas em alocação sequencial, que compartilham a memória de dimensão M.



Tempo: 15 minutos

Exercício (solução)

- 2 pilhas, P_1 e P_2
- vetor P: armazena as pilhas P_1 e P_2



Pilha P_1

- se desenvolve da esquerda para a direita
- posição `P[1]` é a base de P_1
- `topo_1` indica o topo da pilha 1

Pilha P_2

- se desenvolve da direita para a esquerda
- posição `P[M]` é a base de P_2
- `topo_2` indica o topo da pilha 2

Situação inicial: `topo_1` = 0 ; `topo_2` = `M` + 1

Exercício (solução)

- Algoritmo: inserção em pilhas compartilhadas

```
se topo_2 ≠ topo_1 + 1 então
     se b então
          topo_1 := topo_1 + 1
          P[ topo_1 ] := novo_valor
     senão
          topo_2 := topo_2 - 1
          P[ topo_2 ] := novo_valor
senão overflow
```

- Situação inicial (pilha vazia): topo_1 = 0
  topo_2 = M + 1

- Variável booleana b:
  - b = verdadeiro, inserção em P1
  - b = falso, inserção em P2

Exercício (solução)

- Algoritmo: remoção de pilhas compartilhadas

```
se b então
     se topo_1 ≠ 0 então
          valor_recuperado := P[ topo_1 ]
          topo_1  := topo_1 - 1
     senão underflow em P1
senão  se topo_2 ≠ M + 1 então
                valor_recuperado := P[ topo_2 ]
                topo_2  := topo_2 + 1
       senão underflow em P2
```

- Situação inicial (pilha vazia): topo_1 = 0
  topo_2 = M + 1
- Variável booleana b:
  b = verdadeiro, remoção em P1
  b = falso, remoção em P2

cederj

Aplicação: Notação Polonesa

- Notação tradicional de expressões aritméticas: ambiguidade obriga uso de parênteses
- Exemplo: A + B x C
  A + (B x C)
  (A + B) x C
- Notação tradicional completamente parentizada: um par de parênteses, para cada operação
- Notação tradicional: A x B - C / D
- Notação completamente parentizada:
  ((A x B) - (C / D))
- Número total de pares de parênteses = número total de operações

Aplicação: Notação Polonesa

- Notação polonesa:
  - operadores antes dos operandos, em cada operação;
  - a notação explicita quais operadores, e em que ordem, devem ser calculados;
  - dispensa o uso de parênteses, sem ambiguidades.
- Exemplos:
  - Notação tradicional: A x B - C / D
    Notação polonesa: - x A B / C D
  - Notação tradicional: A x ((B - C) / D)
    Notação polonesa: x A / - B C D

Notação Polonesa Reversa

- Notação polonesa reversa:
  - operadores aparecem após os operandos;
  - utilizada em vários equipamentos eletrônicos, como calculadoras e computadores;
  - a ordem dos operandos na notação tradicional e na notação polonesa (reversa ou não) é idêntica;
  - os operadores aparecem na ordem em que devem ser calculados.
- Exemplos:
  - Notação tradicional: A x B - C / D
    Notação polonesa reversa: A B x C D / -
  - Notação tradicional: A x ((B - C) / D)
    Notação polonesa reversa: A B C - D / x

Exercício

- Escrever um algoritmo para converter uma expressão na notação tradicional completamente parentizada para a notação polonesa reversa.
  - o algoritmo deve determinar a ordem e posição dos operadores;
  - uma pilha é utilizada;
  - os operadores devem ser armazenados na pilha até que um ")" seja encontrado. O último operador armazenado corresponde a essa operação;
  - o algoritmo deve supor que a expressão tradicional dada, completamente parentizada, esteja correta;
  - a expressão dada encontra-se no vetor `exp`;
  - a expressão convertida deve ser escrita no vetor `pol`.

Tempo: 15 minutos

Solução

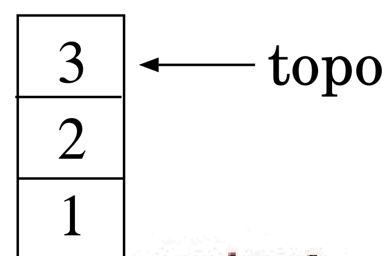
Algoritmo: conversão de notações

```
indexp := 1; indpol := 0; topo := 0
enquanto indexp ≤ fim faça
   se exp[ indexp ] é operando então
        indpol := indpol + 1
        pol[ indpol ] := exp[ indexp ]
   senão se exp[ indexp ] é operador então
                  topo := topo + 1
                  pilha[ topo ] := exp[ indexp ]
              senão se exp[ indexp ] = ")" então
                             se topo ≠ 0 então
                                  operador := pilha[ topo ]
                                  topo := topo - 1
                                  indpol := indpol + 1
                                   pol[ indpol ] := operador
                             senão "expressão errada"
  indexp := indexp + 1
```

- `indexp` = índice para o vetor `exp`
- `indpol` = índice para o vetor `pol`
- `fim` = tamanho do vetor `exp`
- `pilha` = vetor usado para armazenar a pilha

Exercícios Finais

- Seja S a sequência fornecida pelos inteiros 1, 2, ..., n, cujos elementos são inseridos em uma pilha P, em ordem crescente. As remoções de P podem ocorrer de forma qualquer, respeitando a condição de underflow. Uma permutação admissível é uma sequência formada pelos elementos de S, que corresponde a alguma ordem válida de remoção de P.
  - Exemplo: se S = 1, 2, 3 então
    - 3 2 1 é admissível, pois
      I 1, I 2, I 3, R, R, R => 3 2 1
    - 2 3 1 é admissível, pois
      I 1, I 2, R, I 3, R, R => 2 3 1
    - 3 1 2 não é admissível, pois



Exercícios Finais

- Resolver as questões abaixo:
  - Escrever todas as permutações de 1, 2, 3, 4.
  - Mostrar que uma permutação $p_1\ p_2\ p_3\ \ldots\ p_n$ é admissível se e somente se não existirem índices i, j, k satisfazendo

    $$i < j < k \text{ e } p_j < p_k < p_i$$

    Exemplo: $p_1\ p_2\ p_3 = 3\ 1\ 2$ não é admissível, pois $p_2 < p_3 < p_1$