Algoritmos e Estruturas de Dados I

Pilhas e Filas

versão 2.9.2

Fabiano Oliveira
fabiano.oliveira@ime.uerj.br

Pilhas e Filas

- Listas Lineares agrupam diversos elementos e oferecem operações de inserção, busca e remoção gerais
  - Caso não seja necessário fornecer todas estas operações, ou não seja necessário que cada operação seja oferecida de forma tão geral, há a possibilidade de oferecer mais eficientemente um conjunto menor (e ainda útil!) de operações

Pilhas e Filas

- **Deque**: inserções/remoções/buscas apenas em pontas da lista
  - **Pilha**:
    - inserções/remoções/buscas na mesma ponta
    - o último a entrar é o primeiro a sair (LIFO - "last in, first out")
  - **Fila**:
    - inserções numa ponta; remoções/buscas na outra
    - o primeiro a entrar é o primeiro a sair (FIFO - "first in, first out")

Pilha<TElem>

```
função Topo(ref P: Pilha): <TElem>
```
Obtém o último elemento inserido em P, ou NULO se não há
```
procedimento Empilha(ref P: Pilha, x: <TElem>)
```
Insere x em P
```
função Desempilha(ref P: Pilha): <TElem>
```
Remove e retorna o topo de P
```
função Tamanho(ref P: Pilha): Inteiro
```
Obtém o número de elementos na pilha

Pilhas e Filas

- **Exemplo 1:**

```
var P: Pilha <Inteiro>
Constroi(P)
Empilha(P, 12)
Empilha(P, 25)
escrever (Topo(P))  //saída: 25
Empilha(P, 1)
escrever (Desempilha(P))   //saída: 1
Empilha(P, 7)
escrever (Desempilha(P))   //saída: 7
escrever (Desempilha(P))   //saída: 25
escrever (Desempilha(P))   //saída: 12
Destroi(P)
```

Pilhas e Filas

- **Exemplo 2:**

```
var P1, P2: Pilha <Inteiro>
Constroi(P1), Constroi(P2)
Empilha(P1, 12)
Empilha(P1, 25)
Empilha(P2, Topo(P1))
Empilha(P2, 7)
Empilha(P2, Desempilha(P1))
Empilha(P2, 30)
Empilha(P1, Topo(P2))
enquanto Tamanho(P1) > 0 faça
    escrever (Desempilha(P1))   //saídas: 30, 12
enquanto Tamanho(P2) > 0 faça
    escrever (Desempilha(P2))   //saídas: 30, 25, 7, 25
Destroi(P1), Destroi(P2)
```

Pilhas e Filas

- **Exemplo 3:**

```
função OqueEuFaco(ref B[]: Inteiro,
                  N: Inteiro): Lógico
   var P: Pilha<Inteiro>
   Constroi(P)
   para i ← 1 até N faça
      Empilha(P, B[i])
   i ← 1
   enquanto i ≤ N E B[i] = Desempilha(P) faça
      i ← i + 1
   Destroi(P)
   retornar (i > N)
```

Pilhas e Filas

- **Exemplo 4:** na Notação Polonesa Reversa (NPR), operandos vêm antes de operação, que é aplicada imediatamente

```
(a) 3 x 7 x 4 - 1 - 5 + 1
(b) 3 x (7 x (4 - 1) - 5) + 1
```

em NPR:

```
(a) 3 7 x 4 x 1 - 5 - 1 +
(b) 3 7 4 1 - x 5 - x 1 +
```

Pilhas e Filas

```
programa NPR()
   var Expr: ListaLinear<Inteiro, Cadeia>
   var Termo: Cadeia
   var i ← 1
   ler (Termo)
   enquanto Termo ≠ "FIM" faça
      InsereEm(Expr, i, i, Termo)
      i ← i+1
      ler (Termo)
   escrever (AvaliaNPR(Expr))
```

Pilhas e Filas

```
função AvaliaNPR(Expr: ListaLinear): Real
        //Ex: Expr = ["3","7","4","1","-","x","5","-","x","1","+"]
        var P: Pilha<Real>
        para i ← 1 até Tamanho(Expr) faça
            Termo ← BuscaEm(Expr, i)
            se Termo = "+" então
                v2←Desempilha(P); v1←Desempilha(P)
                Empilha(P, v1+v2)
            senão se Termo = "*" então
                v2←Desempilha(P); v1←Desempilha(P)
                Empilha(P, v1*v2)
            senão se Termo = ... então
                //tratar cada possível operação
            senão //trata-se de número
                Empilha(P, de_cadeia(Termo))
        retornar Desempilha(P)
```

Pilhas e Filas

- Com Alocação Sequencial:

```
var MAX_N: Inteiro ← <NÚMERO MÁXIMO DE ELEMENTOS>

estrutura Pilha<TElem>:
    Elem[1..MAX_N]: <TElem>
    N: Inteiro //número corrente de elementos na pilha

procedimento Constroi(ref P: Pilha)
    P.N ← 0

procedimento Destroi(ref P: Pilha)
    //nada precisa ser feito
```

Espaço:
θ(MAX_N)

Pilhas e Filas

- Com Alocação Sequencial:

```
função Topo(ref P: Pilha): <TElem>
   retornar (P.Elem[P.N])
```

Tempo:
θ(1)

Pilhas e Filas

- Com Alocação Sequencial:

```
procedimento Empilha(ref P: Pilha, x: <TElem>)
   se P.N < MAX_N então
      P.N ← P.N + 1
      P.Elem[P.N] ← x
   senão
      Exceção("Overflow")
```

Tempo:
θ(1)

Pilhas e Filas

- Com Alocação Sequencial:

```
função Desempilha(ref P: Pilha): <TElem>
   se P.N > 0 então
      P.N ← P.N - 1
      retornar (P.Elem[P.N+1])
   senão
      Exceção("Underflow")
```

Tempo:
θ(1)

Pilhas e Filas

- Com Alocação Sequencial:

```
função Tamanho(ref P: Pilha): Inteiro
   retornar (P.N)
```

Tempo:
$\theta(1)$

Pilhas e Filas

- Com Alocação Encadeada:

```
estrutura No <TElem>:
    Elem: <TElem>
    Prox: ^No

estrutura Pilha <TElem>:
    Inicio: ^No <TElem>
    N: Inteiro

procedimento Constroi(ref P: Pilha)
    P.Inicio, P.N ← NULO, 0

procedimento Destroi(ref P: Pilha)
    // mesma destruição de listas encadeadas
```

Espaço:
θ(N)

Pilhas e Filas

- Com Alocação Encadeada:

```
função Topo(ref P: Pilha): <TElem>
   retornar (P.Inicio^.Elem)
```

Tempo:
θ(1)

Pilhas e Filas

- Com Alocação Encadeada:

```
procedimento Empilha(ref P: Pilha, x: <TElem>)
    var novoNo: ^No
    alocar(novoNo)
    novoNo^.Elem, novoNo^.Prox ← x, P.Inicio
    P.Inicio, P.N ← novoNo, P.N+1
```

Tempo:
θ(1)

Pilhas e Filas

- Com Alocação Encadeada:

```
função Desempilha(ref P: Pilha): <TElem>
   se P.Inicio ≠ NULO então
      var NoARemover: ^No, x: <TElem>
      NoARemover, x, P.Inicio, P.N ←
         P.Inicio, P.Inicio^.Elem,
         P.Inicio^.Prox, P.N-1
      desalocar(NoARemover)
      retornar (x)
   senão
      Exceção("Underflow")
```

Tempo:
θ(1)

Pilhas e Filas

- Com Alocação Encadeada:

```
função Tamanho(ref P: Pilha): Inteiro
   retornar (P.N)
```

Tempo:
$\theta(1)$

Fila<TElem>

```
função Proximo(ref F: Fila): <TElem>
```
Obtém o primeiro elemento inserido em F, ou NULO se não há
```
procedimento Enfileira(ref F: Fila, x: <TElem>)
```
Insere x em F
```
função Desenfileira(ref F: Fila): <TElem>
```
Remove e retorna o próximo de F
```
função Tamanho(ref F: Fila): Inteiro
```
Obtém o número de elementos na fila

Pilhas e Filas

- **Exemplo 1:**

```
var F: Fila <Inteiro>
Constroi(F)
Enfileira(F, 12)
Enfileira(F, 25)
escrever (Proximo(F))  //saída: 12
Enfileira(F, 1)
escrever (Desenfileira(F))   //saída: 12
Enfileira(F, 7)
escrever (Desenfileira(F))   //saída: 25
escrever (Desenfileira(F))   //saída: 1
escrever (Desenfileira(F))   //saída: 7
Destroi(F)
```

Pilhas e Filas

- **Exemplo 2:**

```
var F1, F2: Fila <Inteiro>
Constroi(F1), Constroi(F2)
Enfileira(F1, 12)
Enfileira(F1, 25)
Enfileira(F2, Proximo(F1))
Enfileira(F2, 7)
Enfileira(F2, Desenfileira(F1))
Enfileira(F2, 30)
Enfileira(F1, Proximo(F2))
enquanto Tamanho(F1) > 0 faça
    escrever (Desenfileira(F1))   //saídas: 25, 12
enquanto Tamanho(F2) > 0 faça
    escrever (Desenfileira(F2))   //saídas: 12, 7, 12, 30
Destroi(F1), Destroi(F2)
```

Pilhas e Filas

- **Exemplo 3:**

```
procedimento OqueEuFaco(ref B[]: Inteiro,
                                N: Inteiro)
   var F1, F2: Fila<Inteiro>
   Constroi(F1), Constroi(F2)
   para i ← 1 até N faça
      Enfileira(F1, B[i])
   para i ← 1 até N faça
      para j ← 1 até Tamanho(F1)-1 faça
         Enfileira(F2, Desenfileira(F1))
      B[i] ← Desenfileira(F1)
      F1, F2 ← F2, F1
   Destroi(F1), Destroi(F2)
```

Pilhas e Filas

- **Exemplo 4:**

```
programa NPR()
   var Expr: Fila<Cadeia>
   var Termo: Cadeia
   ler (Termo)
   enquanto Termo ≠ "FIM" faça
      Enfileira(Expr, Termo)
      ler (Termo)
   escrever (AvaliaNPR(Expr))
```

Pilhas e Filas

```
função AvaliaNPR(Expr: Fila): Real
        //Ex: Expr = ["3","7","4","1","-","x","5","-","x","1","+"]
        var P: Pilha<Real>
        enquanto Tamanho(Expr) > 0 faça
            Termo ← Desenfileira(Expr)
            se Termo = "+" então
                v2←Desempilha(P); v1←Desempilha(P)
                Empilha(P, v1+v2)
            senão se Termo = "*" então
                v2←Desempilha(P); v1←Desempilha(P)
                Empilha(P, v1*v2)
            senão se Termo = ... então
                //tratar cada possível operação
            senão //trata-se de número
                Empilha(P, de_cadeia(Termo))
        retornar Desempilha(P)
```

Pilhas e Filas

- Com Alocação Sequencial:

```
var MAX_N: Inteiro ← <MÁXIMO NÚMERO DE ELEMENTOS>

estrutura Fila<TElem>:
    Elem[1..MAX_N]: <TElem>
    Inicio, Fim: Inteiro //início e fim da fila

procedimento Constroi(ref F: Fila)
    F.Inicio, F.Fim ← 0, 0

procedimento Destroi(ref F: Fila)
    //nada precisa ser feito
```

Espaço:
θ(MAX_N)

Pilhas e Filas

- Com Alocação Sequencial:

```
função Proximo(ref F: Fila): <TElem>
   retornar (F.Elem[F.Inicio])
```

Tempo:
θ(1)

Pilhas e Filas

- Com Alocação Sequencial:

```
procedimento Enfileira(ref F: Fila, x: <TElem>)
    var i: Inteiro
    i ← F.Fim mod MAX_N + 1
    se  i ≠ F.Inicio então
        F.Fim ← i
        F.Elem[F.Fim] ← x
        se F.Inicio = 0 então F.Inicio ← 1
    senão
        Exceção("Overflow")
```

Tempo:
θ(1)

Pilhas e Filas

- Com Alocação Sequencial:

```
função Desenfileira(ref F: Fila): <TElem>
    se F.Inicio ≠ 0 então
        var x: <TElem> ← F.Elem[F.Inicio]
        se F.Inicio = F.Fim então
            F.Inicio, F.Fim ← 0, 0
        senão
            F.Inicio ← F.Inicio mod MAX_N + 1
        retornar (x)
    senão
        Exceção("Underflow")
```

Tempo:
θ(1)

Pilhas e Filas

- Com Alocação Sequencial:

```
função Tamanho(ref F: Fila): Inteiro
   se F.Inicio = 0 então
      retornar (0)
   senão se F.Inicio ≤ F.Fim então
      retornar (F.Fim - F.Inicio + 1)
   senão
      retornar MAX_N-(F.Inicio-F.Fim+1)+2
```

Tempo:
$\theta(1)$

Pilhas e Filas

- Com Alocação Encadeada:

```
estrutura No <TElem>:
    Elem: <TElem>
    Prox: ^No

estrutura Fila <TElem>:
    Inicio, Fim: ^No <TElem>
    N: Inteiro

procedimento Constroi(ref F: Fila)
    F.Inicio, F.Fim, F.N ← NULO, NULO, 0

procedimento Destroi(ref F: Fila)
    // mesma destruição de listas encadeadas
```

Espaço:
θ(N)

Pilhas e Filas

- Com Alocação Encadeada:

```
função Proximo(ref F: Fila): <TElem>
   retornar (F.Inicio^.Elem)
```

Tempo:
θ(1)

Pilhas e Filas

- Com Alocação Encadeada:

```
procedimento Enfileira(ref F: Fila, x: <TElem>)
    var novoNo: ^No
    alocar(novoNo)
    novoNo^.Elem, novoNo^.Prox, F.N ← x, NULO, F.N+1
    se F.Inicio = NULO então
        F.Inicio, F.Fim ← novoNo, novoNo
    senão
        F.Fim^.Prox, F.Fim ← novoNo, novoNo
```

Tempo:
$\theta(1)$

Pilhas e Filas

- Com Alocação Encadeada:

```
função Desenfileira(ref F: Fila): <TElem>
    se F.Inicio ≠ NULO então
        var NoARemover: ^No, x: <TElem> ←
                                        F.Inicio, F.Inicio^.Elem
        se F.Inicio = F.Fim então
            F.Inicio, F.Fim ← NULO, NULO
        senão
            F.Inicio ← F.Inicio^.Prox
        desalocar(NoARemover)
        F.N ← F.N - 1
        retornar (x)
    senão
        Exceção("Underflow")
```

Tempo:
θ(1)

Pilhas e Filas

- Com Alocação Encadeada:

```
função Tamanho(ref F: Fila): Inteiro
   retornar (F.N)
```

Tempo:
$\theta(1)$

Exercícios

Exercícios

1. Um deque geral deve prover as seguintes operações:
   a. ***procedimento*** *insereInicio(**ref** D: Deque, x: <TElem>)*
   b. ***procedimento*** *insereFim(**ref** D: Deque, x: <TElem>)*
   c. ***função*** *removeInicio(**ref** D: Deque): <TElem>*
   d. ***função*** *removeFim(**ref** D: Deque): <TElem>*
   e. ***função*** *buscaInicio(**ref** D: Deque): <TElem>*
   f. ***função*** *buscaFim(**ref** D: Deque): <TElem>*

   Defina o estrutura Deque e escreva os procedimentos e funções acima de forma análoga ao realizado com Pilhas e Filas. Considere o uso da alocação tanto sequencial quanto encadeada.

2. Reescreva as operações de Pilha utilizando duas Filas como estrutura de dados auxiliar para guardar os elementos. Manipule as filas por suas interfaces padrão.

3. Reescreva as operações de Fila utilizando duas Pilhas como estrutura de dados auxiliar para guardar os elementos. Manipule as pilhas por suas interfaces padrão.

Exercícios

4. Escreva um algoritmo que dada uma pilha P, inverta a ordem dos elementos de P. Seu algoritmo deve usar espaço auxiliar constante e:
   a. uma fila
   b. duas pilhas
   c. uma pilha

   Seu algoritmo deve manipular as pilhas e filas por suas interfaces padrão.

5. Escreva um algoritmo que dada uma fila F, inverta a ordem dos elementos de F. Seu algoritmo deve manipular F por sua interface padrão, usar espaço auxiliar constante e:
   a. uma pilha
   b. duas filas

   Seu algoritmo deve manipular as pilhas e filas por suas interfaces padrão.

Exercícios

6. Seja X um arquivo em disco que guarda uma sequência de N naturais. Sabendo-se que o conteúdo de X é muito grande para ser carregado todo em memória, faça um algoritmo que escreva os últimos 1000 naturais de X em tempo θ(N). Suponha que o arquivo deve ser acessado sequencialmente, da seguinte maneira: ele deve ser primeiramente aberto, depois cada os naturais são lidos um a um até que um -1 seja lido (indicando final de arquivo), situação em que o arquivo deve ser fechado. Como acesso a disco é uma operação custosa, deseja-se fazer tal impressão numa única passagem pelos dados do arquivo.

7. Criar uma variação de pilha, chamada de PilhaMin, que, além de fornecer as operações de pilha em tempo constante, define a operação
   função *obterMinimo(**ref** P: PilhaMin): <TElem>*
   que retorna o elemento de P com a menor chave em tempo constante.

Exercícios

8. Escrever um algoritmo que converta uma expressão aritmética parentizada usando as 4 operações para a expressão correspondente em notação polonesa reversa. Ex: **Entrada:** ((A+B)*(C-(F/D))) **Saída:** AB+CFD/-*

9. Uma pilha é ordenada se os elementos removidos de P são e_1, …, e_N (nesta ordem) e tais que $e_i \leq e_{i+1}$, para todo $1 \leq i < N$. Dada uma pilha P, escreva um algoritmo que remova e reinsira elementos de P (através das funções empilha e desempilha) até que P se torne ordenada. Seu algoritmo deverá utilizar espaço auxiliar constante acrescido de outra pilha.

Exercícios

10. Escrever um algoritmo que verifique se uma expressão está corretamente parentizada (isto é, sem que um parênteses abra (respectivamente feche) sem fechar (respectivamente sem abrir) entre um par de abre-e-fecha de parênteses). Você deve atender as seguintes especificações:
    a. não há limite de tamanho para a entrada (exceto pela memória disponível)
    b. existem vários tipos de parênteses. Cada fechamento de parênteses deve corresponder a uma abertura do mesmo tipo. O tipo é especificado pelo nome logo após cada parêntese.

    Exemplo de entrada:

    (p(p)p(p)p(p(p)p)p)p ⇒ OK (só um tipo de parênteses, p)

    (p1(p2)p1)p2 ⇒ incorreto (dois tipos de parênteses, p1 e p2)

    (p1(p2)p2)p1 ⇒ OK (dois tipos de parênteses, p1 e p2)

Exercícios

11. Deseja-se por razões técnicas quebrar uma pilha P em várias pilhas menores P_1, P_2, …, P_T. A ideia é que sejam inseridos em P_1 do primeiro ao k-ésimo elemento de P, em P_2 do (k+1)-ésimo ao (2k)-ésimo elemento de P, e assim por diante. Para tanto, projetou-se a estrutura de dados chamada PilhaDividida como se segue:

```
var T: Inteiro ← <NÚMERO MÁXIMO DE PILHAS>

estrutura PilhaDividida:
    P[1..T]: Pilha
    k, N: Inteiro

procedimento Constroi(ref P: PilhaDividida, k: Inteiro)
    P.k, P.N ← k, 0
    para i ← 1 até T faça
        Constroi(P.P[i])

var P: PilhaDividida

Constroi(P, <NÚMERO MÁXIMO DE ELEMENTOS NUMA PILHA>)
```

Termine o trabalho, escrevendo as operações de pilha para a PilhaDividida, acrescida da seguinte operação:

```
função Desempilha(ref P: PilhaDividida, i: Inteiro): <TElem>
```

que desempilha um elemento da pilha P_i e o retorna

Exercícios

12. Projete um programa para monitorar entradas e saídas de pacotes em um roteador. O programa deve ler uma sequência de N pares X Y onde X = "E" representando se o roteador deve receber em seu buffer um novo dado cujo valor é Y, ou X="S" representando que ele deve enviar o dado mais antigo no buffer pelo canal Y. Como entrada, haverá o valor de N seguido dos N pares X Y. Como saída, o programa deve imprimir, tão logo um par X Y com X = "S" é lido, o par R S representando que o dado R será enviado pelo canal S. O algoritmo deve ter complexidade de tempo O(N).
    Exemplo:

Entrada:	**Saída:**
13	
E 10 E 2 S 4	10 4
E 3 E 7 S 5	2 5
S 6	3 6
S 7	7 7
E 6 E 1 E 4 E 5 S 1	6 1

Exercícios

13. Dada uma sequência de N teclas pressionadas com o cursor focado em um editor de texto inicialmente sem texto algum, onde cada tecla pode ser uma letra, espaço em branco, número, [(código para tecla Home) ou] (código para tecla End), determine qual será o texto final produzido na tela do editor. Lembre-se que Home envia o cursor ao início do texto e a tecla End o envia para o final. O algoritmo deve ter complexidade de tempo θ (N).
    Exemplo:
    Entrada: Text[o ba]gun[ça[d]o; **Saída:** dçao baTextguno

Exercícios

14. Elabore um algoritmo que leia N inteiros e imprima-os alternando números positivos com negativos, sendo que tanto a ordem de impressão entre os números positivos quanto aquela entre os números negativos devem ser a mesma daquela de leitura. Não há qualquer garantia sobre como se dará a aparição de positivos e negativos durante a leitura, exceto de que haverão ao todo N/2 números positivos e N/2 números negativos. Seu algoritmo deve ter complexidade de tempo O(N). Exemplo:
Entrada: 3 2 1 -2 -1 4 6 -3 -4 -5 -6 -7 -8 -9 5 7 8 9
Saída: 3 -2 2 -1 1 -3 4 -4 6 -5 5 -6 7 -7 8 -8 9 -9