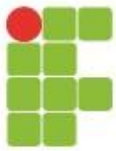


Licenciatura em Computação

Algoritmos e Estrutura de Dados

Pilhas e Filas

Prof. Walteno Martins Parreira Júnior

www.waltenomartins.com.br

waltenomartins@iftm.edu.br

2016

SUMÁRIO

1. PILHAS

1.1. Introdução

São estruturas de dados do tipo LIFO (last-in first-out), onde o último elemento a ser inserido, será o primeiro a ser retirado. Assim, uma pilha permite acesso a apenas um item de dados - o último inserido. Para processar o penúltimo item inserido, deve-se remover o último.

São exemplos de uso de pilha em um sistema:

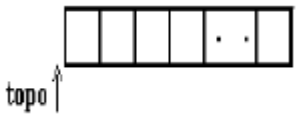
- Funções recursivas em compiladores;
- Mecanismo de desfazer/refazer dos editores de texto;
- Navegação entre páginas Web;
- etc.

A implementação de pilhas pode ser realizada através de vetor (alocação do espaço de memória para os elementos é contígua) ou através de listas encadeadas. Numa pilha, a manipulação dos elementos é realizada em apenas uma das extremidades, chamada de topo, em oposição a outra extremidade, chamada de base.

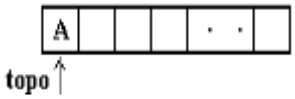
1.2. Conceito de Pilhas

Pilhas são listas onde a inserção de um novo item ou a remoção de um item já existente se dá em uma única extremidade, no topo.

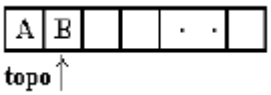
Pilha vazia



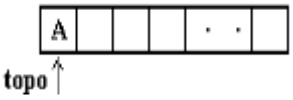
Insere(A)



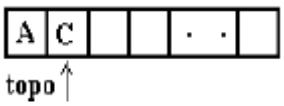
Insere(B)



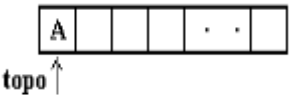
Retira(B)



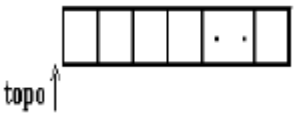
Insere(C)

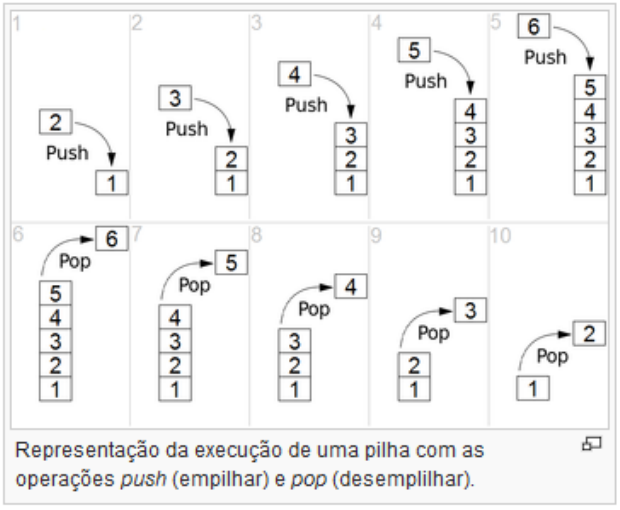


Retira(C)



Retira(A)





Representação da execução de uma pilha com as operações *push* (empilhar) e *pop* (desemplilhar).

Definição:

Dada uma pilha P=(a(1), a(2), ..., a(n)), dizemos que a(1) é o elemento da base da pilha; a(n) é o elemento topo da pilha; e a(i+1) está acima de a(i).

Pilhas são também conhecidas como listas **LIFO** (last in first out).

Operações Associadas:

1. **criar (P)** - criar uma pilha P vazia
2. **inserir (x, P)** - insere x no topo de P (empilha): push(x,P)
3. **vazia (P)** - testa se P está vazia
4. **topo (P)** - acessa o elemento do topo da pilha (sem eliminar)
5. **elimina (P)** - elimina o elemento do topo de P (desempilha): pop(P)

1.3. Implementação de Pilhas

Como lista Seqüencial ou Encadeada?

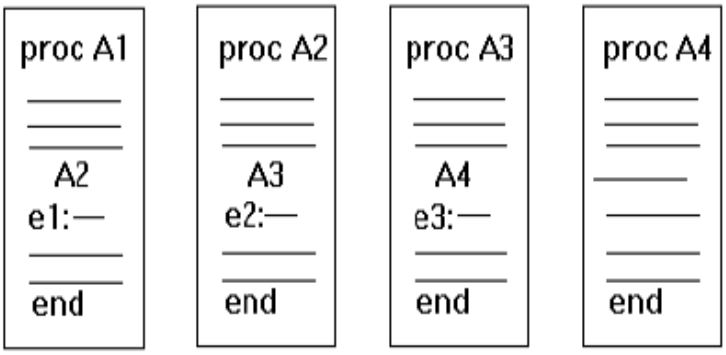
No caso geral de listas ordenadas, a maior vantagem da alocação encadeada sobre a seqüencial - se a memória não for problema - é a eliminação de deslocamentos na inserção ou eliminação dos elementos. No caso das pilhas, essas operações de deslocamento não ocorrem.

Portanto, podemos dizer que a alocação seqüencial é mais vantajosa na maioria das vezes.

1.4. Exemplos do Uso de Pilhas

a) Chamadas de procedimentos

Suponha a seguinte situação:



Quando o procedimento A1 é executado, ele efetua uma chamada a A2, que deve carregar consigo o endereço de retorno e1. Ao término de A2, o processamento deve retornar ao A1, no devido endereço. Situação idêntica ocorre em A2 e A3.

Assim, quando um procedimento termina, é o seu endereço de retorno que deve ser consultado. Portanto, há uma lista implícita de endereços (e0, e1, e2, e3) que deve ser manipulada como uma pilha pelo sistema, onde e0 é o endereço de retorno de A1.

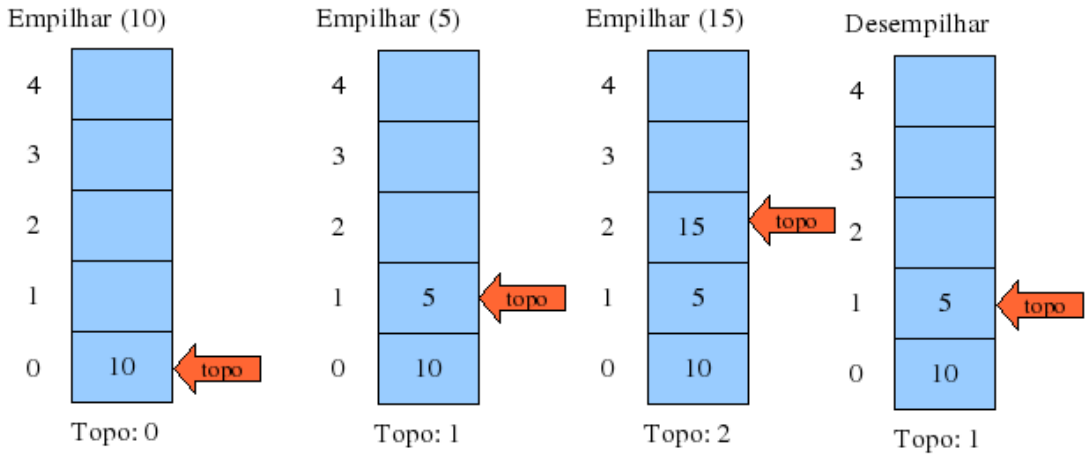
No caso de processamento recursivo, por exemplo em uma chamada a A2 dentro de A4, o gerenciamento da lista como uma pilha resolve automaticamente a obtenção dos endereços de retorno na ordem apropriada (e0, e1, e2, e3, e4).

b) Operações com Pilha:

Todas as operações em uma pilha podem ser imaginadas como as que ocorre numa pilha de pratos em um restaurante ou como num jogo com as cartas de um baralho:

- criação da pilha (informar a capacidade no caso de implementação sequencial - vetor);
- empilhar (push) - o elemento é o parâmetro nesta operação;
- desempilhar (pop);
- mostrar o topo;
- verificar se a pilha está vazia (isEmpty);
- verificar se a pilha está cheia (isFull - implementação sequencial - vetor).

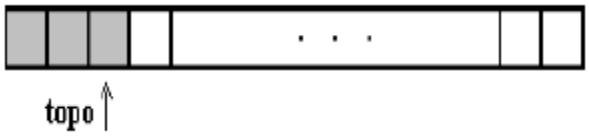
Supondo uma pilha com capacidade para 5 elementos (5 nós).



1.5. Alocação Sequencial de Pilhas

Definição da Estrutura de Dados:

```
Type
  pilha = array [1..maxp] of TipoElem;
  índice = 0 .. maxp;
Var
  P: pilha;
  topo: indice;
```



Operações:

1. criar (P) - criar uma pilha P vazia

```
procedure criar (Var topo:indice);
begin
  topo := 0;
end;
```

2. inserir (x, P) - insere x no topo de P(empilha): push (x, P).

```
procedure push (x:TipoElem; var P: pilha; Var topo: indice);
begin
 if topo = maxp
   then
     "PILHA CHEIA"
   else begin
     topo := topo + 1;
     P[topo] := x;
   end
end;
```

3. vazia (P) - testa se P está vazia

```
function vazia (topo: indice): boolean;
begin
 vazia := (topo = 0);
end;
```

4. topo (P) - acessa o elemento do topo da pilha (sem eliminar)

```
procedure top (Var topo_p: TipoElem; P:pilha; topo:indice);
begin
 if topo = 0
   then "PILHA VAZIA"
   else topo_p := P[topo];
end;
```

5. elimina (P) - elimina o elemento do topo de P (desempilha): pop (P)

```
procedure pop (var P:pilha; Var topo:indice);
begin
  if topo = 0
    then "PILHA VAZIA"
    else topo := topo-1;
end;
```

6. Devolve elemento eliminado

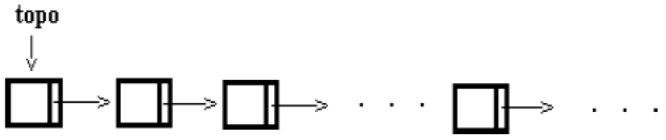
```
procedure pop_up (var topo_p:TipoElem; var P:pilha;
var topo:indice);
begin
  if topo = 0
    then "PILHA VAZIA"
    else begin
      topo_p := P[topo];  {no caso de acesso ao elemento}
      topo := topo-1;
    end;
end;
```

1.6. Alocação Encadeada de Pilhas

Definição da Estrutura de Dados:

```
Type
  tpont = ^reg_pilha;
  pilha = tpont;
  reg_pilha = record
      info: TipoElem;
      lig: tpont;
  End;
```

Var p: pilha;



Operações:

1. criar (P) - criar uma pilha P vazia

```
procedure criar (var p: pilha);
begin
  p := nil;
end;
```

2. inserir (x, P) - insere x no topo de P (empilha): push(x,P)

```
procedure push (x:TipoElem; var p: pilha);
var pont: pilha;
begin
  new(pont);
  pont^.info := x;
  pont^.lig := p;
  p := pont;
end;
```

3. vazia (P) - testa se P está vazia

```
function vazia (var p: pilha): boolean;
begin
  vazia := (p = nil);
end;
```

4. topo (P) - acessa o elemento do topo da pilha (sem eliminar)

```
function top (var p: pilha): TipoElem;
begin
  if (p <> nil) then
     top := p^.info
end;
```

5. elimina (P) - elimina o elemento do topo de P (desempilha) : pop(P)

```
procedure pop (var p: pilha);
 var aux:pilha;
 begin
   if (p <> nil)
      then begin
         aux := p;
```

```
            p:= p^.lig;
            dispose (aux);
          end
      end.
```

1.7. Aplicação de Pilha: Notação Polonesa

Uma representação para expressões aritméticas que seja conveniente do ponto de vista computacional é assunto de interesse, por exemplo, na área de compiladores. A notação tradicional é ambígua e, portanto, obriga o pré-estabelecimento de regras de prioridade. Isso torna a tarefa computacional menos simples. Outras notações são apresentadas a seguir, considerando-se apenas operações binárias (com dois operandos):

Notação completamente Parentizada: acrescenta-se sempre um parêntese a cada par de operandos e seu operador.

Exemplo:

tradicional: A * B - C / D

parentizada: ((A*B)-(C/D))

Notação Polonesa: os operadores aparecem imediatamente antes dos operandos. Esta notação especifica quais operadores, e em que ordem, devem ser calculados. Por esse motivo dispensa o uso de parênteses, sem ambigüidades.

Exemplo:

tradicional: A * B - C / D

polonesa: - * A B / C D

Notação Polonesa Reversa (ou posfix): é como a polonesa na qual os operandos aparecem após os operadores.

Exemplo:

tradicional: A * B - C / D

polonesa reversa: A B * C D / -

Avaliação de expressões aritméticas:

programa fonte - notação infix: x := A / B + D * E - A

objetivo - notação posfix: x := A B / D E * + A -

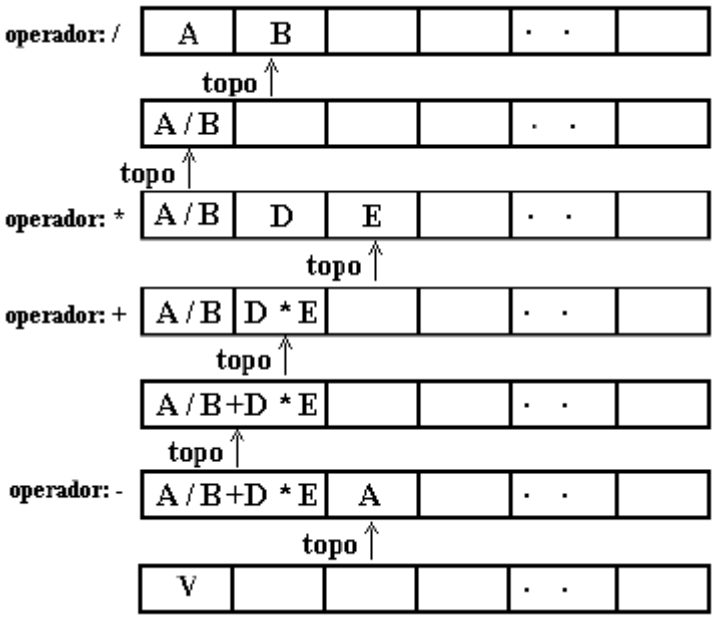
Um algoritmo para a avaliação de Expressões PosFix:

empilha operandos até encontrar um operador

retira o número de operandos; calcula e empilha o valor resultante

até que chegue ao final da expressão

Exemplo: A B / D E * + A –

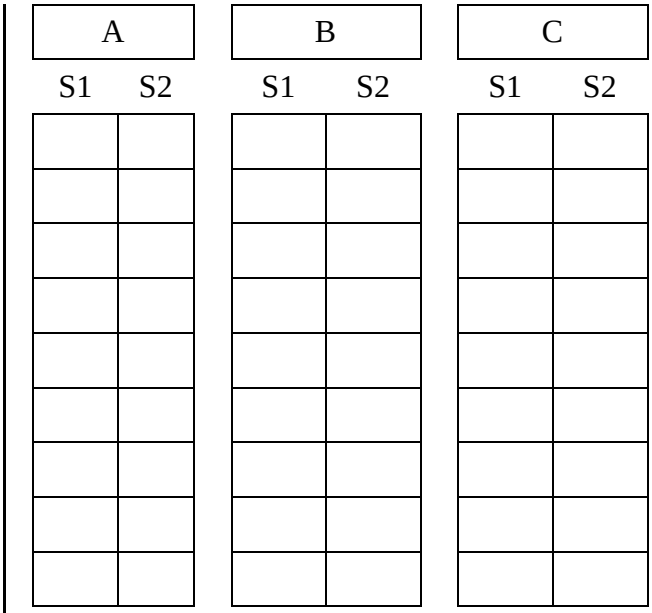



```
function valor ( E: expressão): TipoValor;
 var x : TipoOperador;
 begin
   topo := 0;
   while not acabou(E) do
     begin
       x := proxsimb(E);
       if  x é operando
         then push(x,pilha)
         else begin
           remove o número de operandos(2) para o operador x da pilha;
           calcule o resultado da operação;
           empilhe resultado;
         end;
    valor := P[topo];
 end;
```

1.8. Exercícios

1) Considere que existam duas pilhas vazias denominadas S1 e S2, preencha as ilustrações de cada uma das pilhas e desenvolva as instruções presentes no trecho de algoritmo abaixo. Considere uma rotina chamada Par, que recebe um valor e devolve o valor lógico Verdade se o número é par e falso se é impar. Preencha as ilustrações de cada pilha nos pontos definidos: A, B e C. Deixe as representações das pilhas como elas estão no ponto determinado.

```
A = 0                                   (A)
Para X = 12 ate 19 faça
        empilha(S1,X)
fim-para                                (B)
imprima (´passo dois´)
enquanto(não( pilha_vazia(S1)))
faça
   x ← desempilha(S1)
   Se (Par( x))
        então  empilha(S2,x)
   fim-se
fim-enquanto
Imprima(´final´)                        (C)
```



Para os próximos exercícios, considere as seguintes rotinas como disponíveis.
Pilha_Vazia(p) – retorna T se a pilha P estiver vazia e F se estiver cheia;
Pilha_Cheia(p) – retorna T se a pilha P estiver cheia e f se estiver vazia;
Empilha(p,x) – insere o elemento X no inicio da pilha P;
Desempilha(p) – remove o 1º elemento da pilha P e retorna o conteúdo como valor da função;
Imprime(p) – imprime a pilha P.

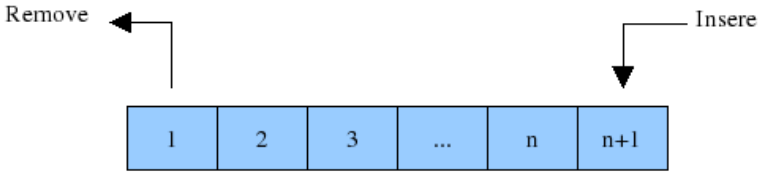
2) Desenhe a evolução da pilha (de tamanho máximo de 8 elementos) e mostre o que será impresso no vídeo, considerando a execução da sequência de instruções (lendo por linha) abaixo:
   Empilha(p,'b'), pilha_cheia(p), Empilha(p,'s'); Empilha(p,'a'), Empilha(p,'c'), Desempilha(p), Desempilha(p), Empilha(p,'h'), Empilha(p,'d'), Empilha(p,'f'); Desempilha(p), Empilha(p,'n'), Empilha(p,'b'), pilha_cheia(p), Desempilha(p), Empilha(p,'m'), Desempilha(p), Desempilha(p), Desempilha(p), Imprime(p), pilha_vazia(p)

3) Desenhe a evolução da pilha (de tamanho máximo de 6 elementos) e mostre o que será impresso no vídeo, considerando a execução da sequência de instruções abaixo: Empilha(p,'b'), pilha_cheia(p), Empilha(p,'s'); Empilha(p,'a'), Empilha(p,'c'), Empilha(p,'h'), Empilha(p,'d'), pilha_cheia(p), Desempilha(p), Empilha(p,'n'), Empilha(p,'b'), pilha_cheia(p), Desempilha(p), Empilha(p,'m'), Desempilha(p), Desempilha(p), Desempilha(p), Imprime(p), pilha_vazia(p)

2. FILAS

2.1. Introdução

São estruturas de dados do tipo FIFO (first-in first-out), onde o primeiro elemento a ser inserido, será o primeiro a ser retirado, ou seja, adiciona-se itens no fim e remove-se do início.



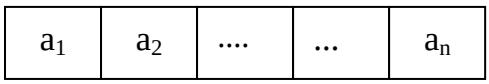
São exemplos de uso de fila em um sistema:

- Controle de documentos para impressão;
- Troca de mensagem entre computadores numa rede;
- etc.

A implementação de filas pode ser realizada através de vetor (alocação do espaço de memória para os elementos é contígua) ou através de listas encadeadas

2.2. Conceito de Filas

É uma lista linear em que a inserção é feita numa extremidade e a eliminação na outra. (FIFO: first in, first out).



Eliminações no início Inserções no final

Exemplo:

Escalonamento de "Jobs": fila de processos aguardando os recursos do sistema operacional.

Definição:

Uma fila é uma estrutura de dados dinâmica que admite remoção de elementos e inserção de novos objetos. Mais especificamente, uma *fila* (= *queue*) é uma estrutura sujeita à seguinte regra de operação: sempre que houver uma remoção, o elemento removido é o que está na estrutura há mais tempo.

Operações associadas:

1. **Criar (F)** - criar uma fila F vazia
2. **Inserir (x, F)** - insere x no fim de F
3. **Vazia (F)** - testa se F está vazia
4. **Primeiro (F)** - acessa o elemento do início da fila
5. **Elimina (F)** - elimina o elemento do início da fila

Implementação de filas como lista Seqüencial ou Encadeada? Dinâmica ou Estática?

Só tem sentido falarmos em fila seqüencial ou encadeada dinâmica, uma vez que não existe movimentação de elementos. As filas encadeadas são usadas quando não há previsão do tamanho máximo da fila.

2.3. Implementação Sequencial de Fila

Definição da Estrutura de Dados


```
Type índice = 0..maxfila;
     fila   = array[1..maxfila] of TipoElem;
Var
  F: fila;
  Começo,        {posição anterior ao primeiro elemento}
  Fim: índice;   {posição do último elemento}
```

Operações com Filas:

1. Criar (F) - criar uma fila F vazia

```
Procedure CriaFila (Var Começo, Fim: indice);
Begin
   Começo := 0;
   Fim := 0;
End;
```

2. Inserir (x, F) - insere x no fim de F

```
Procedure Inserir (x: TipoElem; Var F: fila; Var Fim: indice);
Begin
  If Fim < maxfila
    Then Begin
      Fim := Fim + 1;
      F[Fim] := x;
    End
    Else
      { OVERFLOW }
End;
```

3. Vazia (F) - testa se F está vazia

```
Function Vazia (Começo, Fim: indice): Boolean;
Begin
  If Começo = Fim
    Then
      {FILA VAZIA}
End;
```

4. Primeiro (F) - acessa o elemento do início da fila

```
Function Primeiro (F: fila; Começo: indice): TipoElem;
Begin
   x := F[Começo + 1];
End;
```

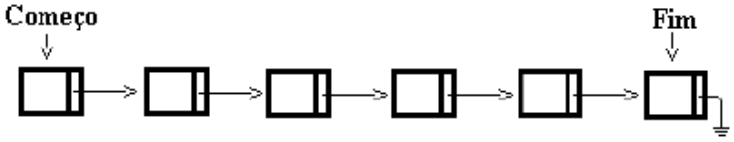
5. Elimina (F) - elimina o elemento do início da fila

```
Procedure Eliminar (Var Começo: indice, Fim: indice);
Begin
  If (Começo = Fim)
    Then
      {FILA VAZIA}
    Else
      Começo := Começo + 1;
End;
```

2.4. Implementação Encadeada de Fila

Definição da Estrutura de Dados

```
Type tpont    = ^reg_fila;
    fila     = tpont;
    reg_fila = record
                 info: TipoElem;
                 lig: tpont;
               End;
Var Começo, Fim: fila;
```



Operações com Filas:

1. Criar (F) - criar uma fila F vazia

```
Procedure CriaFila (Var Começo, Fim: fila);
 Begin
   Começo := nil;
   Fim := nil;
 End;
```

2. Inserir (x, F) - insere x no fim de F, {só se lista for não vazia}

```
Procedure Inserir (x: TipoElem;  Var Fim: fila);
 Begin
   new(Fim^.lig);
   Fim := Fim^.lig;
   Fim^.info := x;
   Fim^.lig := nil;
 End;
```

3. Vazia (F) - testa se F está vazia

```
Function Vazia (Começo, Fim: fila): boolean;
 Begin
   Vazia := ( Começo = nil) and (Fim = nil);
 End;
```

4. Primeiro (F) - acessa o elemento do início da fila

```
Function Primeiro (Var Começo: fila): TipoElem;
 Begin
   Primeiro := Começo^.info;
 End;
```

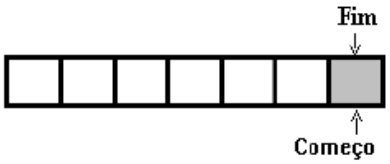
5. Elimina (F) - elimina o elemento do início da fila

```
Procedure Elimina (Var Começo, Fim: fila;);
Var p: fila;
Begin
  if (começo <> nil)
     then begin
          p := Começo;
          Começo := Começo^.lig;      {válido se lista não vazia}
          If ( Começo = Fim)
             Then Begin
                  Começo := nil;
                  Fim  := nil;
                 End;
          dispose(p);
       end
End;
```

2.5. Problema na Implementação com Fila

O que acontece com a fila considerando a seguinte seqüência de operações sobre um fila IEIEIEIEIE. (I - inserção e E - eliminação)

A fila terá sempre 1 ou 0 elementos, no entanto num certo instante:



Ou seja, apenas um elemento na fila, o qual ocupa a última posição do array! Na próxima inserção, teremos uma condição de overflow e a fila está vazia!

Alternativa: no algoritmo de eliminação após a atualização de Começo, verificar se a fila ficou vazia, i.e, Começo = Fim; se este for o caso, reinicializar Começo = Fim := 0;

Portanto, ficaria:

```
Procedure Eliminar (Var Começo, Fim: indice);
 Begin
   If ( Começo = Fim)
      Then
        {FILA VAZIA}
      Else Begin
            Começo := Começo + 1;
            If Começo = Fim
              Then Begin
                    Começo := 0;
                    Fim := 0;
                  End;
          End;
End;
```

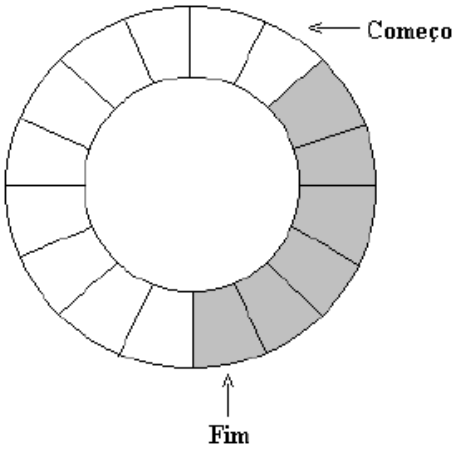
O que aconteceria se a sequência fosse IIEIEIEIEI?

A lista estaria com no máximo dois elementos, mas ainda ocorreria overflow com a lista quase vazia.

Alternativa: Forçar Fim a usar o espaço liberado por Começo (Fila Circular)

2.6. Fila Circular

Fila Circular Implementada como Anel



1. Índices do array: 0 .. m-1
2. Ponteiros de controle: Começo e Fim
3. Inicialmente Começo = Fim = 0
4. Quando Fim = m-1, o próximo elemento será inserido na posição 0 (se essa posição estiver vazia!)
5. Condição de fila vazia Começo = Fim
6. Para inserir:

```
Procedure Inserir (Var Fim: indice);
Begin
  If Fim = m-1
     Then
       Fim := 0
     Else
       Fim := Fim + 1;  {ou  Fim := ( Fim + 1 ) mod m}
End;
```

7.Para eliminar um elemento

```
Procedure Eliminar (Var Começo: indice);
Begin
   Começo := (Começo + 1) mod m;
End;
```

Problema: Nesta representação, como teremos fila cheia???

Começo = Fim

Para resolver esse problema, utilizaremos apenas m-1 posições do anel. Deste modo, existe sempre um elemento vazio e, portanto, Fim não coincide com Começo.

O algoritmo para inserção em anel fica:

```
Procedure Inserir (Var Fim: indice; Começo: indice; F: fila);
Begin
  If (Fim + 1) mod m = Começo
    Then
     {FILA CHEIA}
    Else Begin
          Fim := (Fim + 1) mod m;     {um registro fica vazio}
          F[Fim] := x;
        End;
End;
```

O algoritmo para eliminação em anel fica:

```
Procedure Eliminar (Var Comeco: indice; Fim: indice);
Begin
   If Começo = Fim
      Then
       {FILA VAZIA}
     Else
       Começo := (Começo + 1) mod m;
End;
```

2.7. Exercício

Para os próximos exercícios, considere as seguintes rotinas como disponíveis.
Criar_fila(f) – Cria uma fila vazia;
Fila_Vazia(f) – retorna T se a fila F estiver vazia e F se estiver elemento;
fila_Cheia(f) – retorna T se a fila F estiver cheia e F se estiver vazia;
Insere(f,x) – insere o elemento X no final da fila F;
Remove(f) – remove o 1º elemento da fila F;
Imprime(f) – imprime a Fila F.

1) Considere que existam duas filas vazias denominadas F1 e F2. Execute as instruções do trecho de algoritmo abaixo. Deixe as representações das filas como elas estão no ponto determinado. Preencha as ilustrações de cada fila nos pontos definidos: A, B e C. Considere que os dados a serem lidos no algoritmo são: { 3, 6, 8, 2, 5, 1, 7, 4, 0, 5}.

```
leia(x)                                   A
enquanto x <> 0 faça
        Insere(F1,x)
        leia(x)
fim-enquanto                                  B
leia(y)
enquanto(não( Fila_vazia(F1))) faça
          x ← Primeiro(F1)
          Remove(F1)
          Se x < y
            então Insere(F2,x)
          fim-se
fim-enquanto                                      C
imprima(´final´)
```

Preencher com as situações solicitadas:

A	F1								
	F2								
B	F1								
	F2								
C	F1								
	F2								

2) Desenhe a evolução da Fila (de tamanho máximo de 8 elementos) e mostre o que será impresso no vídeo, considerando a execução da sequência de instruções (lendo por linha) abaixo:
Insere(f,'b'), Fila_cheia(f), Insere(f,'s'); Insere(f,'a'), Insere(f,'c'), Remove(f), Remove(f), Insere(f,'h'), Insere(f,'d'), Insere(f,'f'); Remove(f), Insere(f,'n'), Insere(f,'b'), Fila_cheia(f), Imprime(f), Remove(f), Insere(f,'m'), Remove(f), Remove(f), Imprime(f), Remove(f), Imprime(f), Fila _vazia(f)

3) Desenhe a evolução da Fila (de tamanho máximo de 6 elementos) e mostre o que será impresso no vídeo, considerando a execução da sequência de instruções abaixo:
Insere(f,'b'), Fila_cheia(f), Insere(f,'s'); Insere(f,'a'), Insere(f,'c'), Remove(f), Insere(f,'h'), Insere(f,'d'), Insere(f,'f'); Fila_cheia(f), Imprime(f), Remove(f), Remove(f), Insere(f,'n'), Insere(f,'b'), Fila_cheia(f), Remove(f), Insere(f,'m'), Remove(f), Remove(f), Remove(f), Imprime(f), Remove(f), Remove(f), Remove(f), Fila _vazia(f)

REFERENCIAS

CORMEN, T. H. *Algoritmos:* Teoria e Prática. Rio de Janeiro: Campus. 2001.

FARIAS, R. *Estrutura de Dados e Algoritmos*. 2009. Disponível em <http://www.cos.ufrj.br/~rfarias/cos121>, acesso em 10 out. 2016.

KNUTH, D. E. *The Art of Computer Programming.* Massachusetts: Addison-Wesley Longman, 1997. v. 1 e 2.

SALVETTI, D. D.; BARBOSA L M. *Algoritmos.* São Paulo: Makron Books, 1998.

TANENBAUM, A. M. *Estruturas de dados Usando C*. São Paulo: Makron Books, 1995.

TERADA, R. *Desenvolvimento de Algoritmo e Estruturas de Dados*. São Paulo: Makron Books, 1991.

ZIVIANI, N. *Projeto de Algoritmos* - Com Implementações em PASCAL e C. São Paulo: Editora Pioneira, 1999.

Nota do Professor:

Este trabalho é um resumo do conteúdo da disciplina, para facilitar o desenvolvimento das aulas, devendo sempre ser complementado com estudos nos livros recomendados e o desenvolvimento dos exercícios indicados em sala de aula e a resolução das listas de exercícios propostas.

Parte deste material foi recolhida na internet ao longo dos anos de magistério e nem sempre foi possível relembrar a origem.