CES-11

Algoritmos e Estruturas de Dados

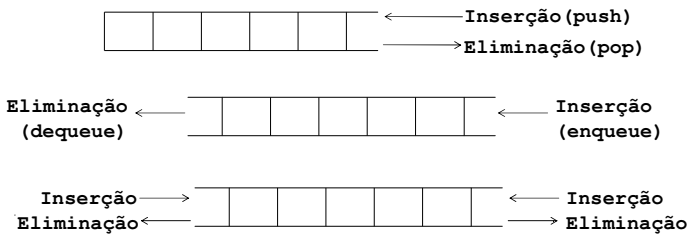
Carlos Alberto Alonso Sanches
Juliana de Melo Bezerra

CES-11

- Pilhas
- Filas
- *Deques*

Pilhas, filas e *deques*

- As listas lineares admitem inserção e eliminação em qualquer posição.
- Pilhas, filas e *deques* (filas com duplo-fim) só admitem acessos em suas extremidades:

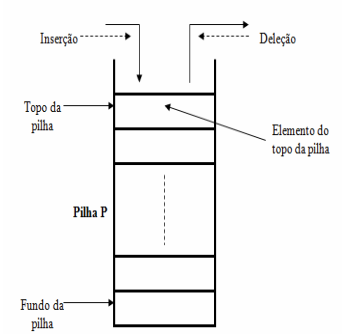


CES-11

- Pilhas
- Filas
- *Deques*

Pilhas (*stacks*)

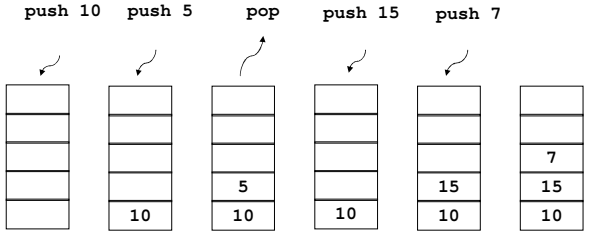
- *Pilha* é uma estrutura linear que permite acesso em somente uma extremidade.
- Por essa razão, a pilha é chamada de estrutura *LIFO* (*last in / first out*).



Pilhas (*stacks*)

- Operações:
  - push(x): insere x no topo
  - pop(): retira o elemento topo
  - top(): retorna o topo sem desempilhá-lo
  - size(): retorna o tamanho atual da pilha
  - isEmpty(): verifica se a pilha está vazia

Exemplos de operações com pilhas



Uma aplicação típica

- Pilhas são utilizadas, por exemplo, para verificar o emparelhamento de delimitadores.

```
Leia o próximo caractere ch;
while não é fim de arquivo {
     if ch é ´(´,´[´ ou ´{´
        push(ch);
     else if ch é ´)´,´]´ ou ´}´ {
             x = top();
             pop();
             if ch e x não se casam
                 erro;
     }
     Leia o próximo caractere ch;
}
if isEmpty()
   sucesso;
else falha;
```

Implementação com vetor


```
size() {
return t+1; }

isEmpty() {
return (t<0); }

top() {
if (isEmpty())
   return Erro;
return S[t]; }
```

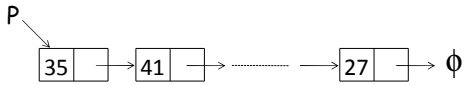
```
push(x) {
if (size() == N)
   return Erro;
S[++t]=x; }

pop() {
if (isEmpty())
   return Erro;
S[t--]=null; }
```

Eficiência dessa implementação

- Na implementação com vetor, todas as operações anteriores podem ser executadas em tempo $O(1)$.
- Caso fosse necessário implementar uma operação de busca na pilha, a solução gastaria tempo proporcional ao tamanho da pilha, ou seja, seria ineficiente...

Implementação com nós encadeados

- Os elementos da pilha podem estar conectados através de uma cadeia de nós:

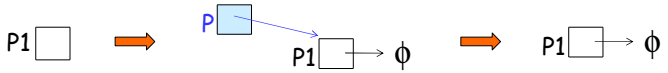



```
struct node {int elem; node *prox;};
typedef node *pilha;
pilha P;
```

Implementação com nós encadeados

- `inicPilha(&P1)`: inicializa a pilha P1

```
void inicPilha (pilha *P) {
     *P = NULL;
}
```

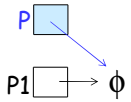


Implementação com nós encadeados

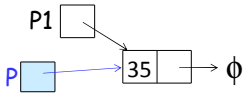
- `isEmpty(P1)`: verifica se a pilha P1 está vazia

```
bool isEmpty (pilha P) {
     if (P == NULL) return TRUE;
     else return FALSE;
}
```

Pilha vazia



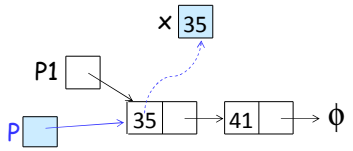
Pilha não vazia



Implementação com nós encadeados

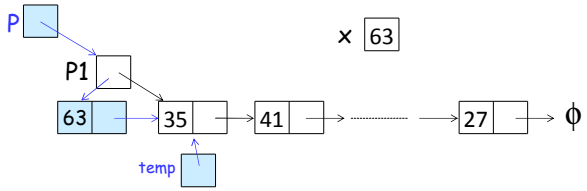
- `x = top(P1)`: x recebe o topo da pilha P1

```
int top (pilha P) {
     if (!isEmpty (P))
          return P->elem;
     else
          Erro ("Pilha vazia");
}
```



Implementação com nós encadeados

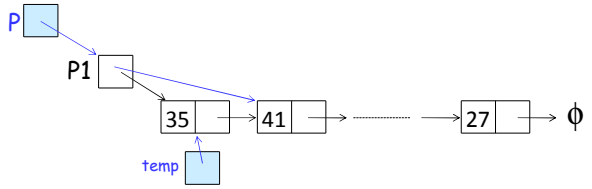
- `push (x,&P1)`: empilha o conteúdo de x na pilha P1




```
void push (char x, pilha *P) {
      node *temp;
      temp = *P;
      *P = (node *) malloc (sizeof (node));
      (*P)->elem = x;
      (*P)->prox = temp;
}
```

Implementação com nós encadeados

- `pop (&P1)`: desempilha o topo da pilha P1




```
void pop (pilha *P) {
      node *temp;
      if (!isEmpty (*P)) {
            temp = *P;
            *P = (*P)->prox;
            free (temp);  }
      else Erro ("Pilha vazia"); }
```

Eficiência dessa implementação

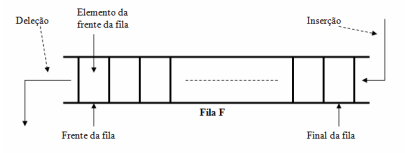
- Na implementação com nós encadeados, todas as operações anteriores podem ser executadas em tempo O(1).
- Qual a vantagem em relação à implementação com vetor?
  - A memória é alocada gradativamente, apenas quando necessário.

CES-11

- Pilhas
- Filas
- *Deques*

Filas (*queues*)

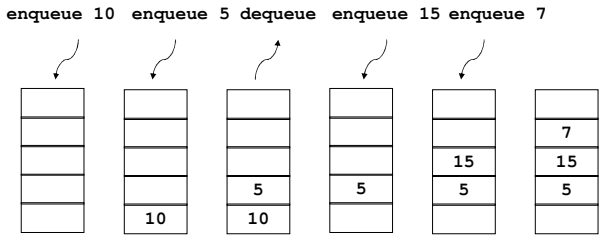
- *Filas* são simples sequências de espera: crescem através do acréscimo de novos elementos no final e diminuem com a saída dos elementos da frente.
- Os elementos são acrescentados em uma extremidade e removidos da outra.
- Em relação à pilha, a principal diferença é que a fila é uma estrutura *FIFO (first in/first out).*



Filas

- Operações:
  - size(): retorna o tamanho atual da fila
  - isEmpty(): verifica se a fila está vazia
  - first(): retorna o primeiro elemento da fila sem removê-lo
  - dequeue(): retira o primeiro elemento da fila
  - enqueue(x): coloca x no final da fila

Exemplos de operações com filas



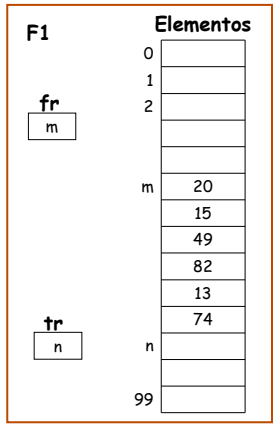
Implementação com vetor circular

- O vetor é utilizado de maneira circular

```
const int max = 100;
typedef int vetor[max];
struct fila {
      vetor Elementos;
      int fr, tr;
};
fila F1, F2, F3;
```

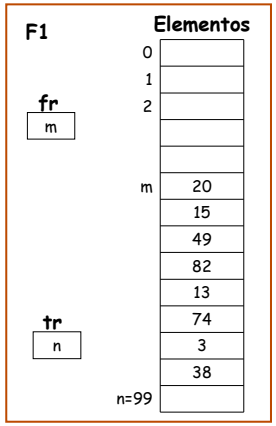
Próxima posição disponível

Posição do primeiro elemento



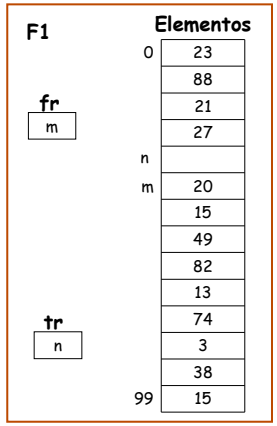
Implementação com vetor circular

- É circular: quando **tr=99**, uma inserção levará **tr** para o índice **0** (zero).
- Desse modo, não haverá necessidade de deslocar todos os elementos para cima...



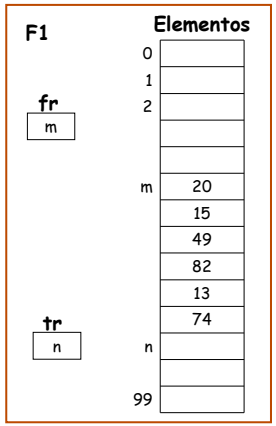
Implementação com vetor circular

- Quando max=100, é possível admitir 100 elementos na fila?
  - Não: fila cheia seria confundida com fila vazia...
- Caso geral desta implementação: em uma fila com max posições, podemos ter no máximo max-1 elementos.



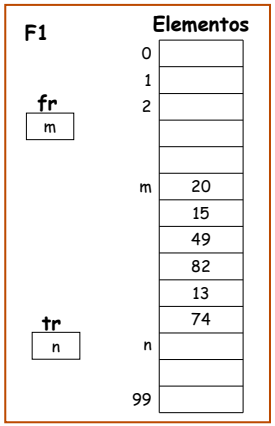
Implementação com vetor circular

```
void inicFila (fila *F) {
     F->fr = 0;
     F->tr = 0;
}


 int prox (int i)  {
      return (i+1) % max;
 }
```



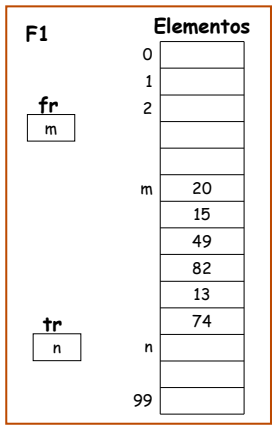
Implementação com vetor circular

```
bool isEmpty (fila *F) {
       if (F->tr == F->fr)
              return TRUE;
       else
              return FALSE;
}


int first (fila *F) {
   if (isEmpty (F))
      Erro ("Fila vazia");
   else
      return F->Elementos[F->fr];
}
```



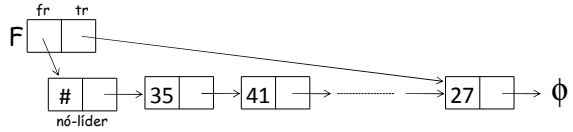
Implementação com vetor circular

```
void enqueue (int x, fila *F)
{
    if ( prox(F->tr) == F->fr )
       Erro ("Fila cheia");
    else {
       F->Elementos[F->tr] = x;
       F->tr = prox(F->tr);
    }
}


 void dequeue (fila *F)
     if (isEmpty (F))
          Erro ("Fila vazia");
     else
          F->fr = prox(F->fr);
 }
```



Implementação com nós encadeados

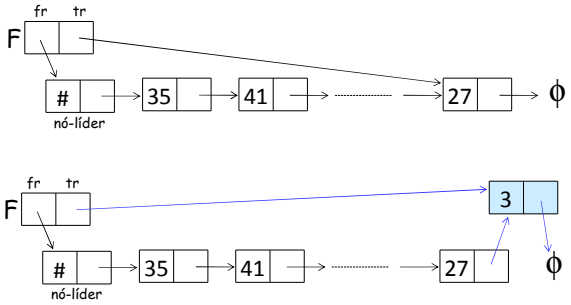
- Os elementos da fila podem estar conectados através de uma cadeia de nós:




```
struct node {int elem; node *prox;};
struct fila {node *fr, *tr;};
fila F;
```

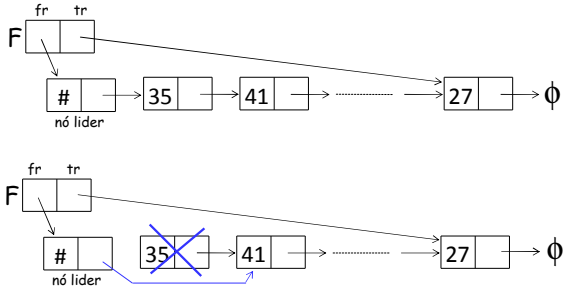
Implementação com nós encadeados

- **enqueue(x,F)**:
  - Inserir um nó com valor x após o último elemento
  - Atualizar o ponteiro F.tr



Implementação com nós encadeados

- **dequeue(F)**:
  - Apagar o primeiro nó da cadeia
  - Se esse nó fosse o único, F.tr deverá apontar para o nó-líder



Implementação com nós encadeados

- Primeiro elemento da fila F
  - `F.fr->prox->elem`
- Verificação de fila F vazia
  - `F.fr == F.tr` OU
  - `F.fr->prox == NULL`
- Esvaziamento da fila F
  - Liberar os nós de F, exceto o líder
  - `F.fr->prox = NULL;`
  - `F.tr = F.fr`

CES-11

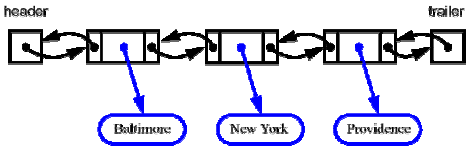
- Pilhas
- Filas
- *Deques*

Deques (filas com duplo-fim)

- Filas com duplo-fim (*doubled-ended queue*) permitem inserção e remoção, em tempo constante, tanto no início como no fim.
- Uma *deque* pode simular tanto as operações de uma pilha como as de uma fila.
- Operações:
  - insertFirst(x): insere x no início
  - insertLast(x): insere x no final
  - removeFirst(): remove o primeiro elemento
  - removeLast(): remove o último elemento
  - first(): retorna o primeiro elemento
  - last(): retorna o último elemento
  - size(): retorna o tamanho atual
  - isEmpty(): verifica se está vazia

Implementação com encadeamento duplo

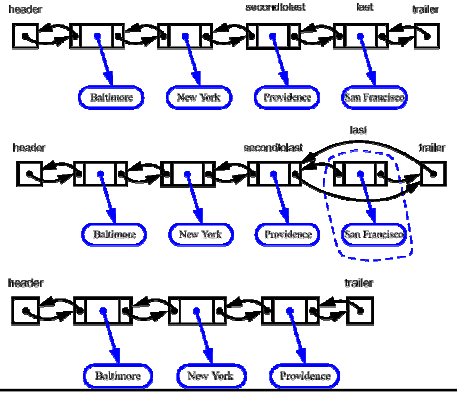
- Podemos implementar uma *deque* através de uma lista duplamente ligada:



- Cada nó tem dois ponteiros: para o próximo e para o anterior.
- *header* e *trailer* são nós especiais, chamados de sentinelas. Não são necessários, mas facilitam a codificação.

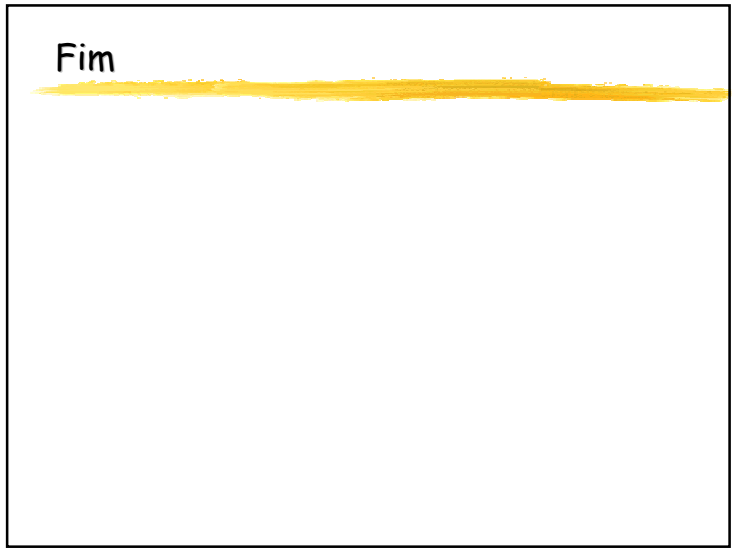
Implementação com encadeamento duplo

- Remoção do último elemento:



Implementação com vetor

- Seria possível implementar uma *deque* através de um vetor?
- Quais as vantagens e as desvantagens em relação à implementação com listas duplamente ligadas?


Fim