Estruturas de Dados e Algoritmos

Tipo Abstrato de Dado e Estruturas Elementares

Adalberto Cajueiro

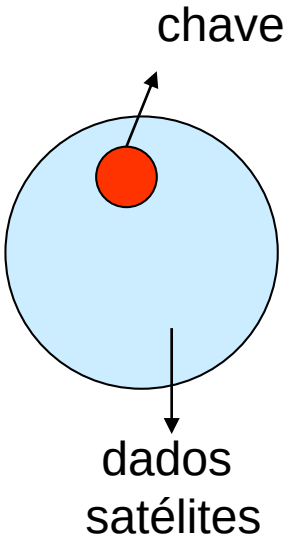
Departamento de Sistemas e Computação

Universidade Federal de Campina Grande

Introducao

- Conjuntos são fundamentais para matemática e ciência da computação
  - Matemática: imutável (naturais, inteiros)
  - Computação: dinâmico (modificam o tamanho)
- Manipulação de conjuntos em computacao envolve
  - Inserir, deletar, pertinência
  - Qualquer conjunto que suporta as operacoes acima é chamado de dicionário

Conjuntos Dinâmicos



- Elementos são objetos contendo uma chave e possivelmente dados satélite
- Alguns conjuntos dinamicos pressupoem que existe uma relação de ordem entre as chaves. Isso permite falar do próximo elemento ao invés de um elemento específico.
- Operações sobre um conjunto dinâmico S:
  - Consultas: search(k)
  - Modificações: insert(x), delete(x), minimum(S), maximum(S), successor(x), predecessor(x)
- Exemplos:
  - Vetor, pilha, fila, listas
  - Árvore binária, AVL, splay, B, preto-vermelho
  - Tabelas de dispersão (hash)

Tipo Abstrato de Dado (TAD)

- É um *contrato de serviço*
  - Estabelece como cliente e servidor devem se comportar para que tudo funcione
- Formalmente é uma especificacao abstrata de um tipo
  - Encapsula a estrutura de dados
  - Agrupa a estrutura de dados e operações. A escolha da representacao é influenciada pelas operações.
  - Abstrai de uma implementação específica (não diz como é implementado)
- É util para dizer em que consiste o serviço e como ele deve ser usado.

Tipo Abstrato de Dado (TAD)

- Como definir um TAD em Java?

- Como implementar um TAD em Java?

Tipo Abstrato de Dado (TAD)

- Como definir um TAD em Java?
  - Uso de interfaces
  - Interface é um contrato em aberto
  - Cumprimento dos aspectos sintáticos → garantidos pelo compilador
  - Cumprimento dos aspectos semanticos → depende da boa programação
- Como implementar um TAD em Java?
  - Seguir a assinatura da interface
  - Seguir a semantica especificada para os serviços da interface

Tipo Abstrato de Dado (TAD)

- Consequencias do uso de TAD
  - Bom uso de interfaces
    - Desacopla codigo do cliente do servidor (melhora a modularidade)
    - Facilita mudanças do tipo concreto em uma aplicação (aumenta estensibilidade)
    - Permite reuso de tipos
  - O bom uso requer que se identifique
    - Interfaces adequadas para o problema
    - Boas implementacoes de cada interface
- Programe para interfaces, não para implementações!

Estruturas de Dados Elementares

Vetor

- É a mais básica das estruturas de dados.
- Vetores são matrizes unidimensionais e também são chamados de arrays.
- Acesso direto ao elemento através de indexação
- Exemplo
  - A = [10, 5, -2, 0, 7]
  - Acesso: A[2];
  - Modificação: A[3] = 19;

Vetor

- Operações
  - Inserir x
    - O(1)
  - Remover x
    - O(n)
  - Pesquisar x
    - O(n)
  - Pesquisar k-ésimo elemento
    - O(1)

Vetor (API de Java)

- Vector
  - http://www.docjar.com/html/api/java/util/Vector.java.html
- Operações
  - indexOf
  - elementAt
  - addElement
  - removeElement
  - remove(int i)

Pilha (Stack) - Intuição

Pilha (Stack)

- Conjunto dinamico onde a operação de remoção segue uma política específica
- Definição
  - Estrutura de dados em que a inserção e a remoção de elementos de uma seqüência se faz pela mesma extremidade, geralmente designada por **topo** da pilha.
  - Política de acesso LIFO = Last-in First-out

Pilha (Stack)

- Operações (interface)
  - Criar uma pilha vazia (create)
  - Inserir/Empilhar (push)
  - Deletar/Desempilhar (pop)
  - Acessar o elemento do topo (top)
  - Verificar se está vazia (isEmpty)
  - Verificar se está cheia (isFull)

Pilha (Stack)

- Interface em Java
  - Como seria a interface de uma pilha genérica em Java?

Pilha (Stack)

- Interface em Java
  - Como descrever uma pilha genérica em Java?

```
public interface Stack<T> {
   public void push(T elem) throws StackOverflowException;
   public T pop() throws StackUnderflowException;
   public T top();
   public boolean isEmpty();
   public boolean isFull();
}

public class StackOverflowException extends Exception {
 public StackOverflowException() {
  super("Stack is full");
 }
}
```

Pilha (Stack)

- Interface em Java
  - Como dscrever uma pilha genérica em Java?

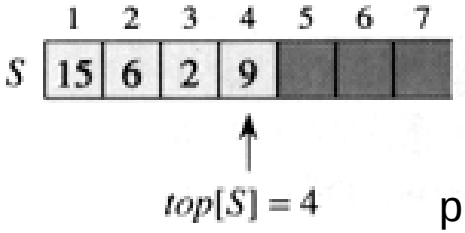
```
public interface Stack<T> {
   public void push(T elem) throws StackOverflowException;
   public T pop() throws StackUnderflowException;
   public T top();
   public boolean isEmpty();
   public boolean isFull();
}
```

E o create da pilha?

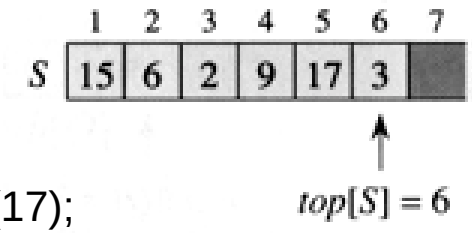
```
public class StackOverflowException extends Exception {
 public StackOverflowException() {
  super("Stack is full");
 }
}
```

Pilha (Stack)

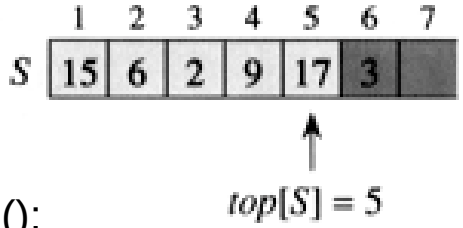
- Implementacao com array



push(17);
push(3);

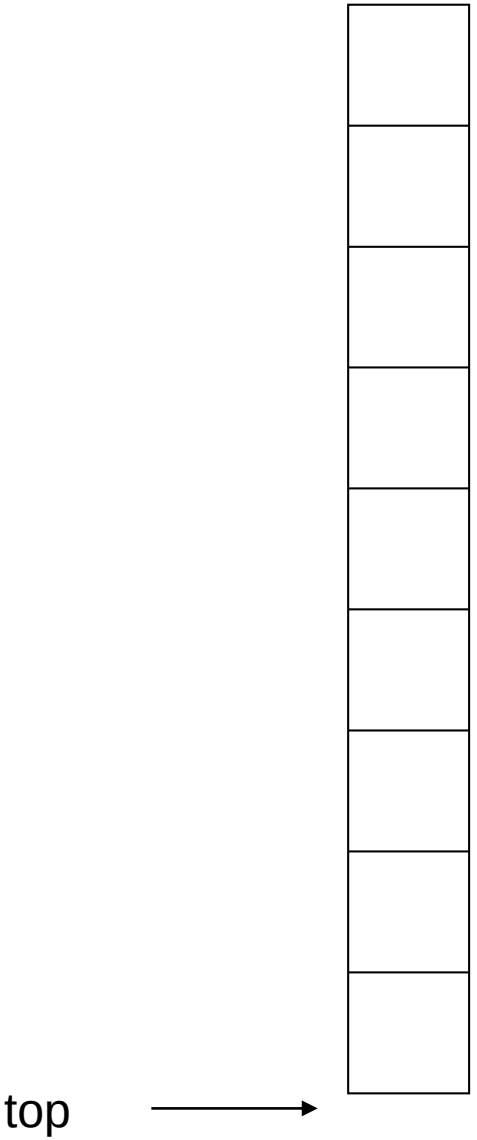


pop();



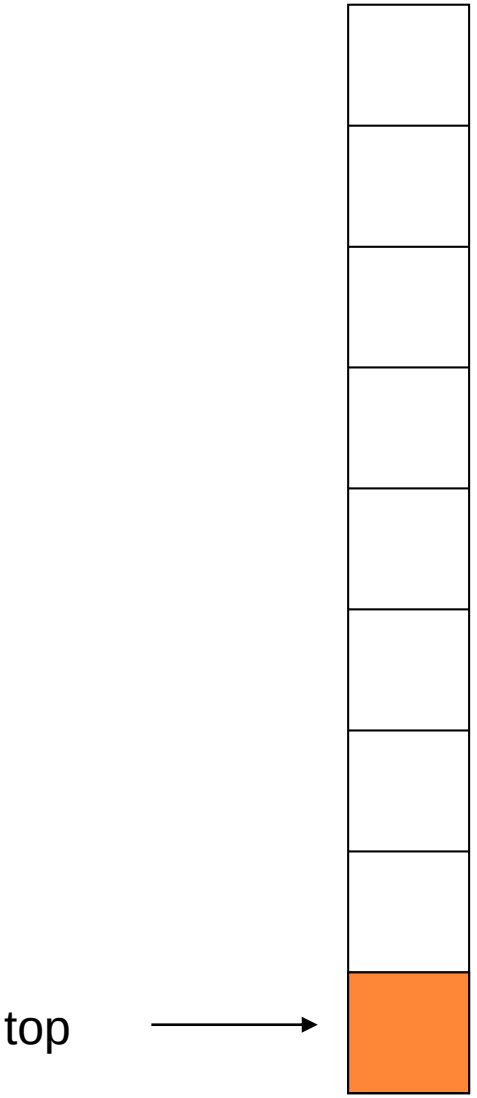
Stack (funcionamento)

inicializacao



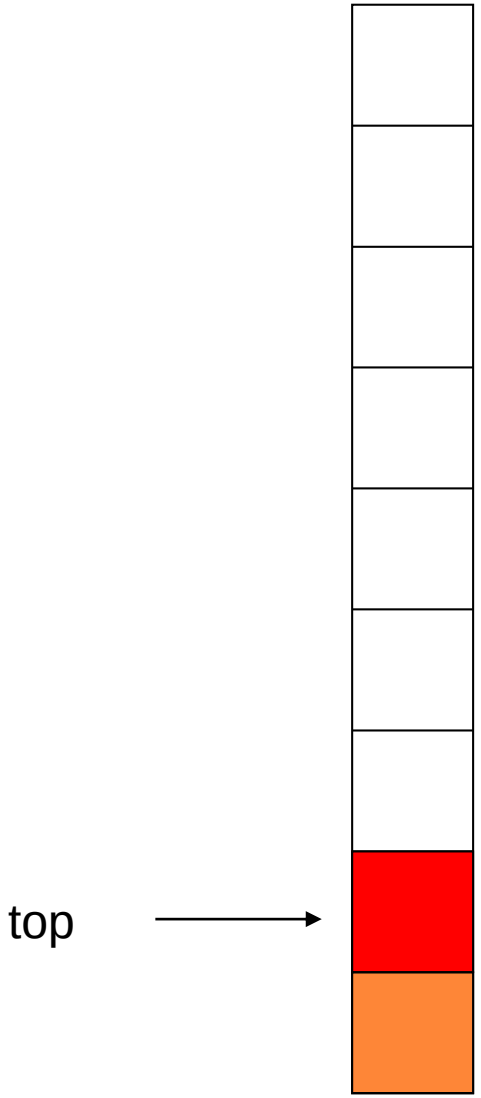
STACK(FUNCIONAMENTO)

push(elemento)



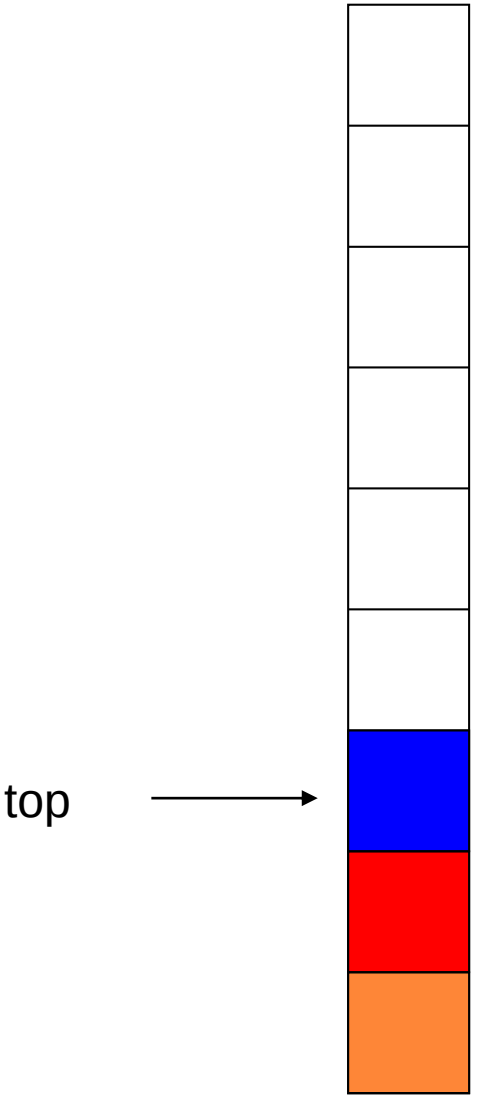
Stack (funcionamento)

push(elemento)
push(elemento)



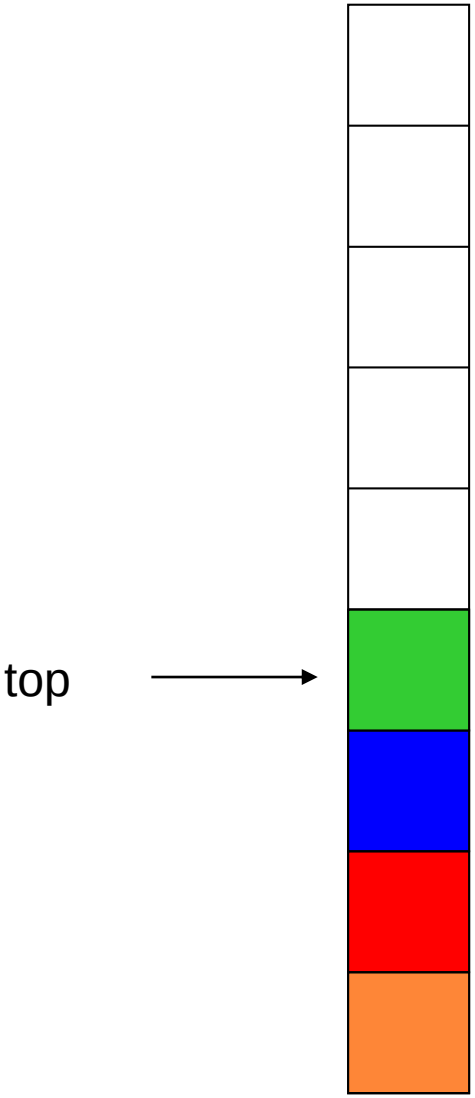
Stack (funcionamento)

push(elemento)
push(elemento)
push(elemento)



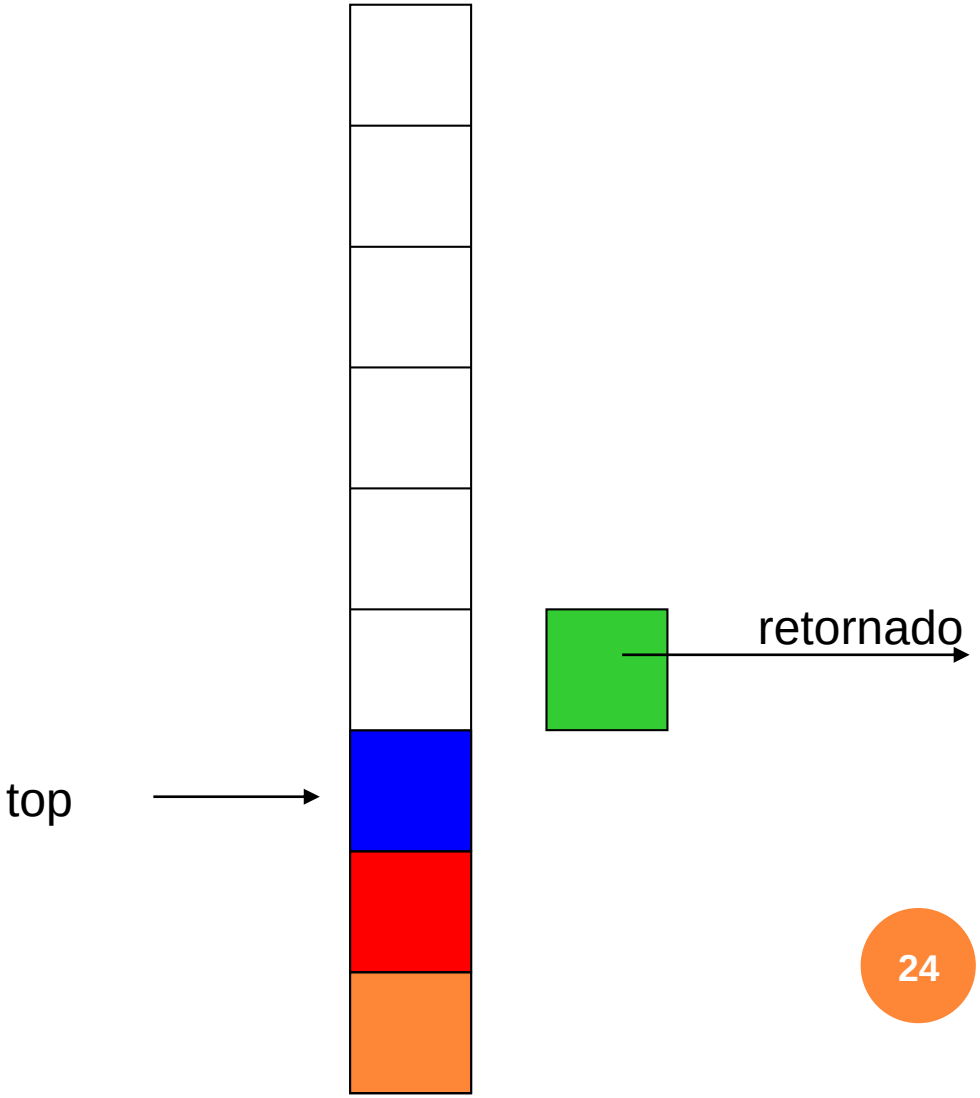
Stack (funcionamento)

push(elemento)
push(elemento)
push(elemento)
push(elemento)



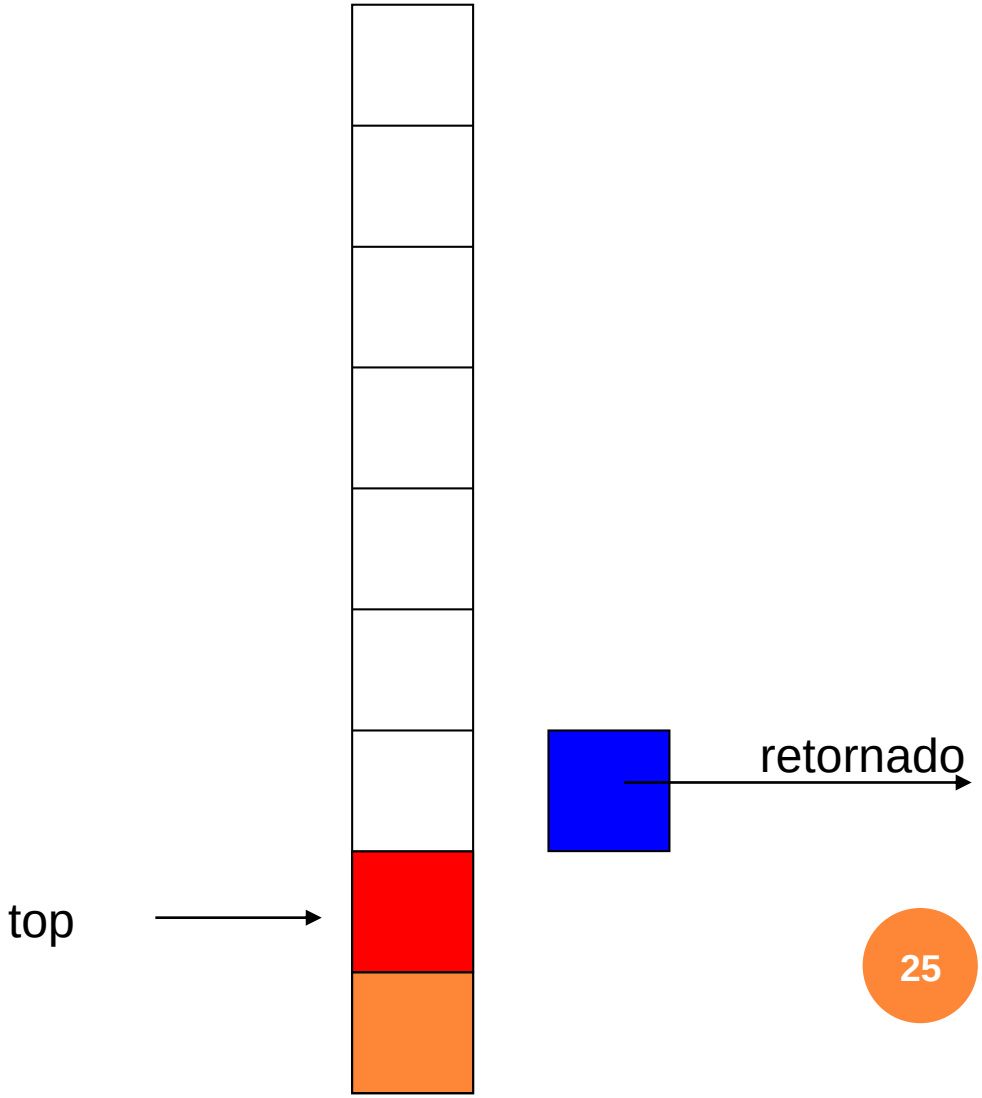
Stack (funcionamento)

push(elemento)
push(elemento)
push(elemento)
push(elemento)
pop()



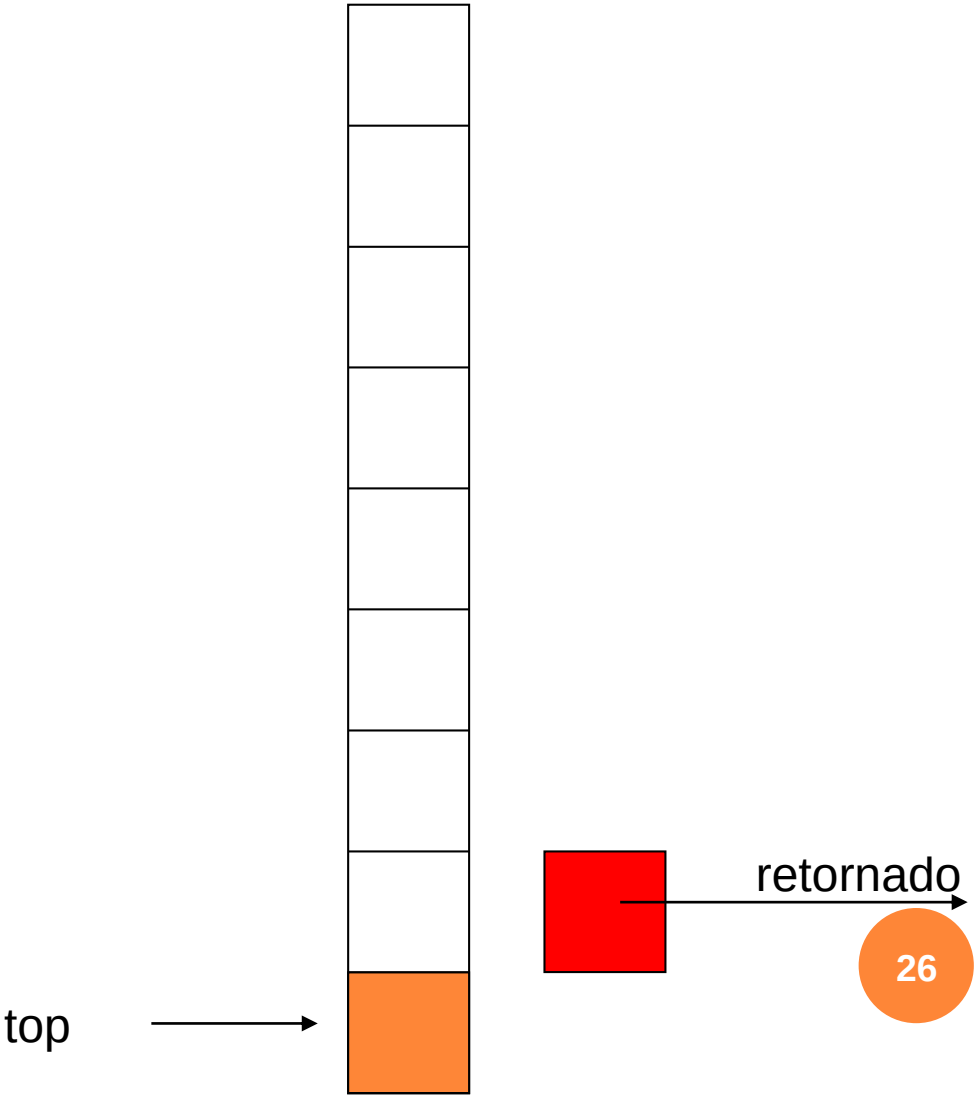
Stack (funcionamento)

push(elemento)
push(elemento)
push(elemento)
push(elemento)
pop()
pop()



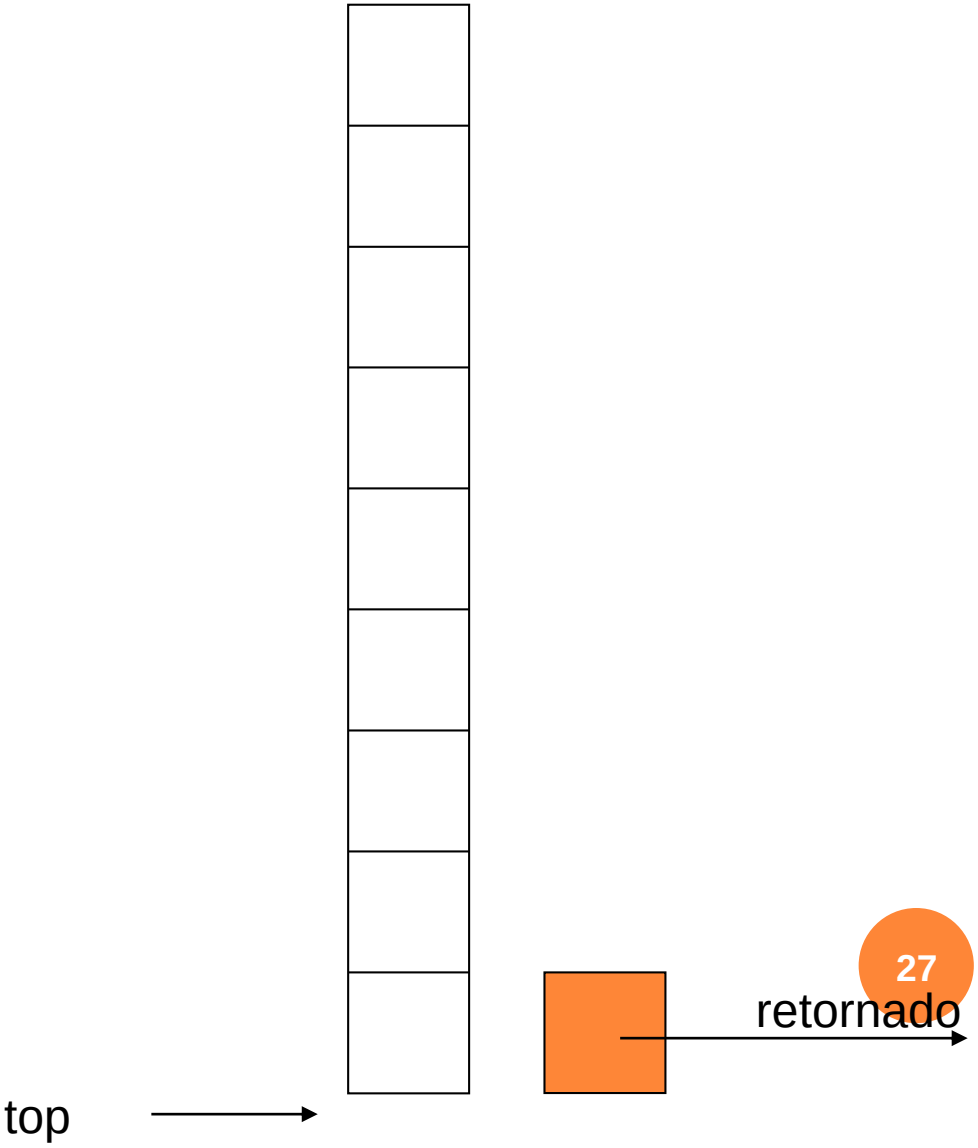
Stack (funcionamento)

push(elemento)
push(elemento)
push(elemento)
push(elemento)
pop()
pop()
pop()



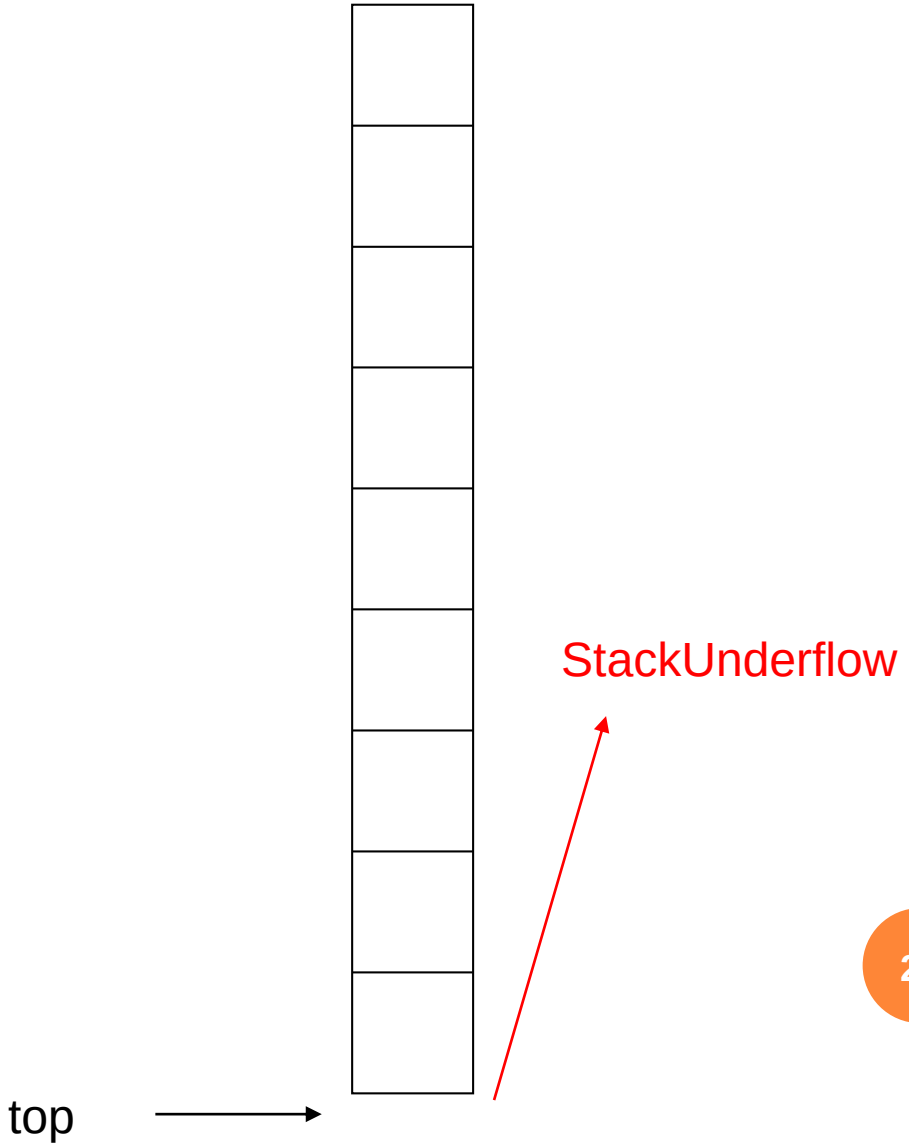
Stack (funcionamento)

push(elemento)
push(elemento)
push(elemento)
push(elemento)
pop()
pop()
pop()
pop()

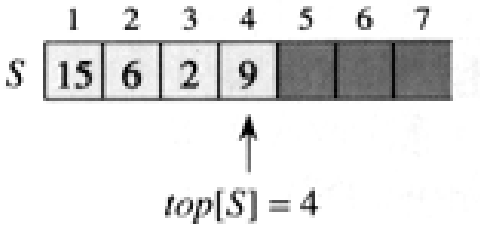


Stack (funcionamento)

push(elemento)
push(elemento)
push(elemento)
push(elemento)
pop()
pop()
pop()
pop()
pop()



IMPLEMENTAÇÃO

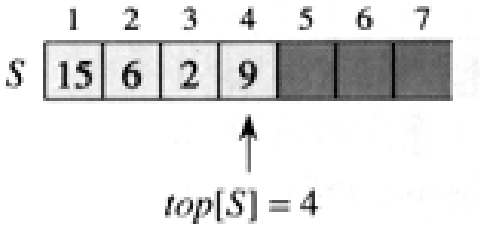



```
STACK-EMPTY(S)
1  if top[S] = 0
2      then return TRUE
3      else  return FALSE
```

```
PUSH(S, x)
1  top[S] ← top[S] + 1
2  S[top[S]] ← x
```

```
POP(S)
1  if STACK-EMPTY(S)
2      then error "underflow"
3      else  top[S] ← top[S] – 1
4            return S[top[S] + 1]
```

IMPLEMENTAÇÃO



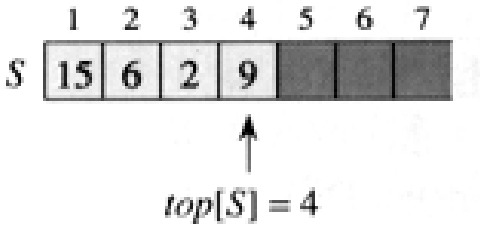

```
STACK-EMPTY(S)
1  if top[S] = 0
2      then return TRUE
3      else  return FALSE
```

```
PUSH(S, x)
1  top[S] ← top[S] + 1
2  S[top[S]] ← x
```

```
POP(S)
1  if STACK-EMPTY(S)
2      then error "underflow"
3      else  top[S] ← top[S] − 1
4            return S[top[S] + 1]
```

Qual a complexidade de cada serviço?

IMPLEMENTAÇÃO




```
STACK-EMPTY(S)
1  if top[S] = 0
2      then return TRUE
3      else return FALSE
```

$\Theta(1)$

```
PUSH(S, x)
1  top[S] ← top[S] + 1
2  S[top[S]] ← x
```

$\Theta(1)$

```
POP(S)
1  if STACK-EMPTY(S)
2      then error "underflow"
3      else top[S] ← top[S] − 1
4           return S[top[S] + 1]
```

$\Theta(1)$

Qual a complexidade de cada serviço?

Exercício

- Como fica o estado de uma pilha inicialmente vazia após a execução dos comandos:
  - push(10)
  - push(5)
  - pop()
  - push(7)
  - top()
  - pop()
  - isEmpty()

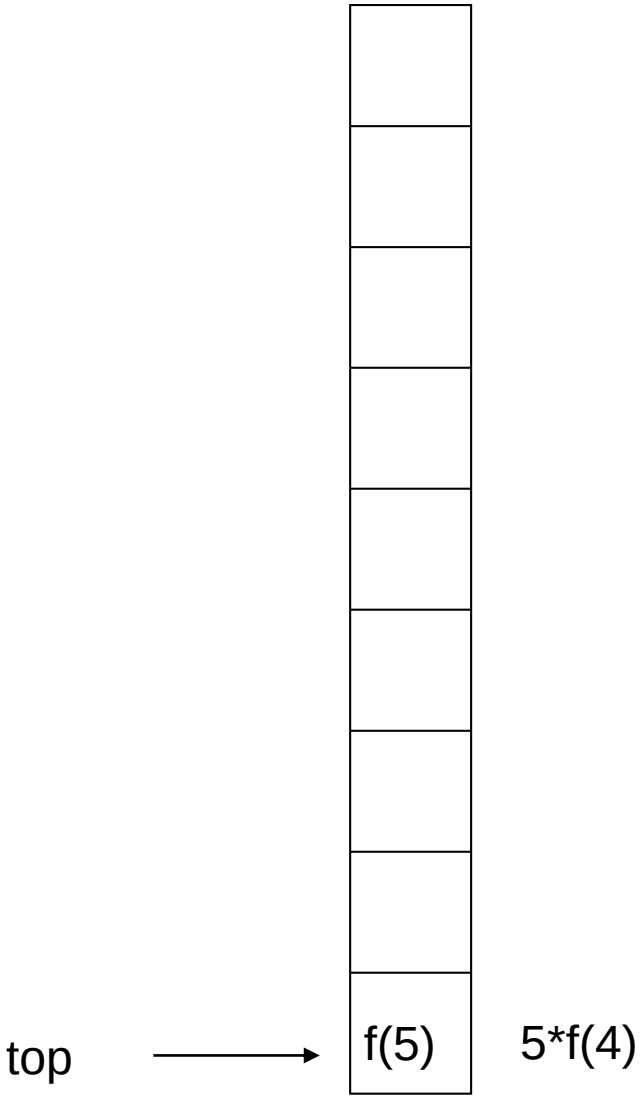
Stack (Aplicacao)

- Processamento de expressões aritméticas em calculadoras
  - (1+5)*7 = 1 5 + 7 * = 42
- Execução de programas
  - Quando uma rotina chama outra rotina, a primeira deve saber como prosseguir quando a segunda for concluída.
  - Exemplo: fatorial
- Browser (botão voltar)
- Undo (Ctrl + Z) do editores de texto

Pilha (aplicacao)

f(0) = 1
f(n) = n*f(n-1)

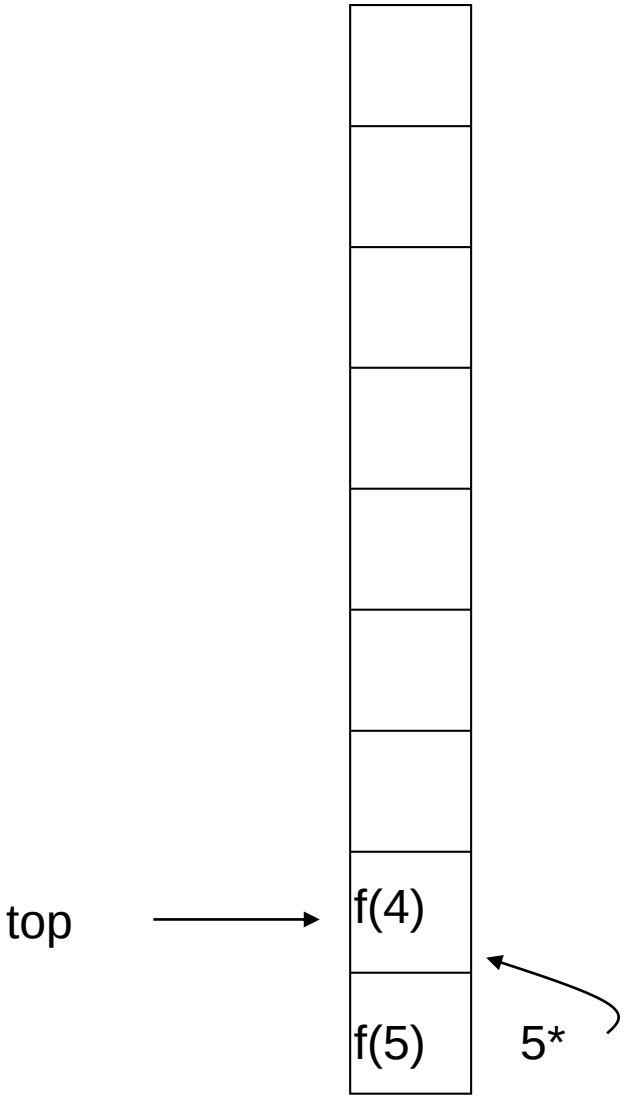
f(5) = 5*f(4)



Pilha (aplicacao)

f(0) = 1
f(n) = n*f(n-1)

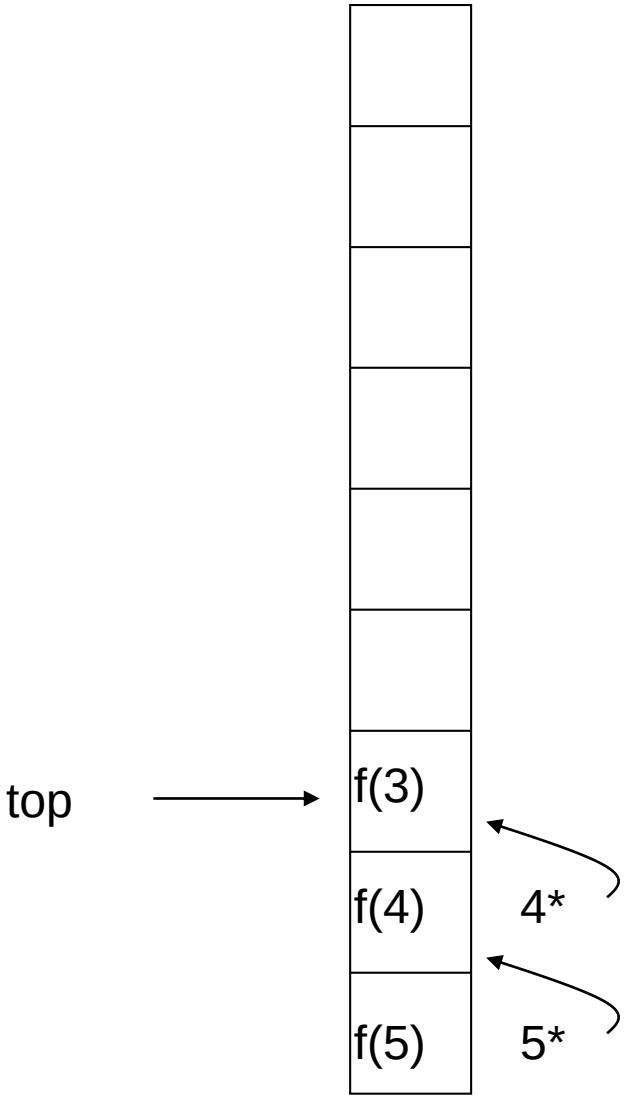
f(5) = 5*f(4)



PILHA (APLICACAO)

f(0) = 1
f(n) = n*f(n-1)

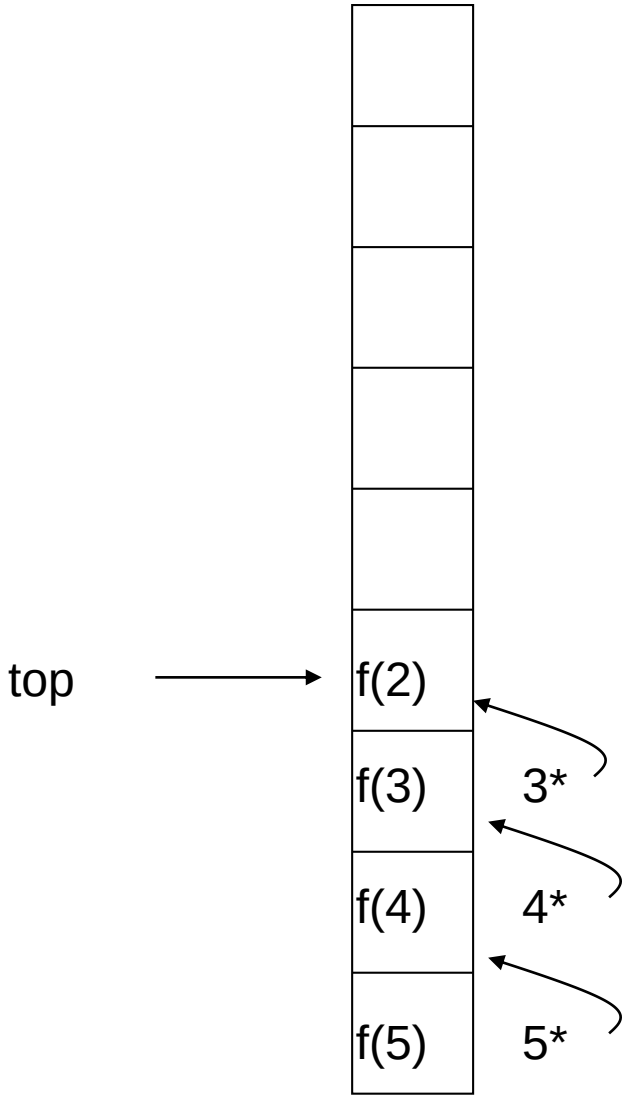
f(5) = 5*f(4)
f(4) = 4*f(3)



Pilha (aplicacao)

f(0) = 1
f(n) = n*f(n-1)

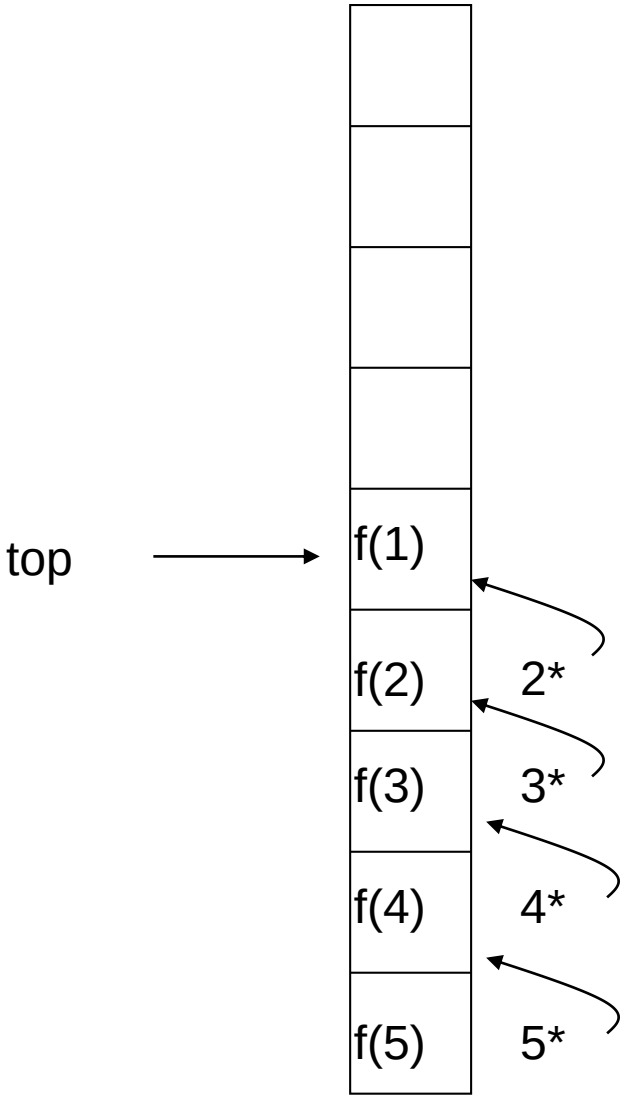
f(5) = 5*f(4)
f(4) = 4*f(3)
f(3) = 3*f(2)



Pilha (aplicacao)

f(0) = 1
f(n) = n*f(n-1)

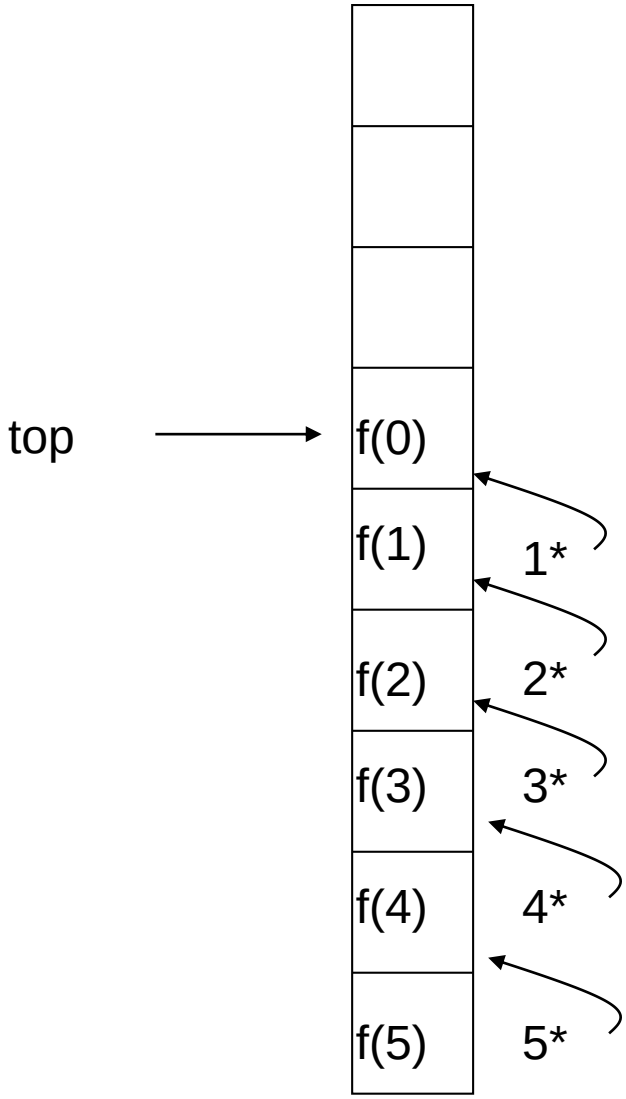
f(5) = 5*f(4)
f(4) = 4*f(3)
f(3) = 3*f(2)
f(2) = 2*f(1)



Pilha (aplicacao)

f(0) = 1
f(n) = n*f(n-1)

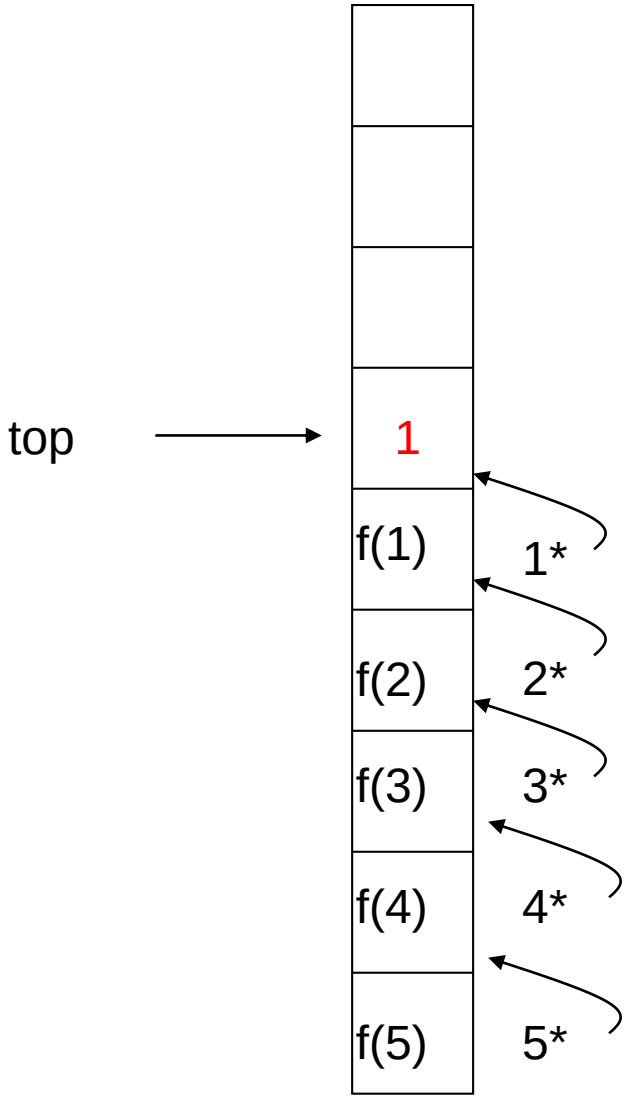
f(5) = 5*f(4)
f(4) = 4*f(3)
f(3) = 3*f(2)
f(2) = 2*f(1)
f(1) = 1*f(0)



PILHA (APLICACAO)

f(0) = 1
f(n) = n*f(n-1)

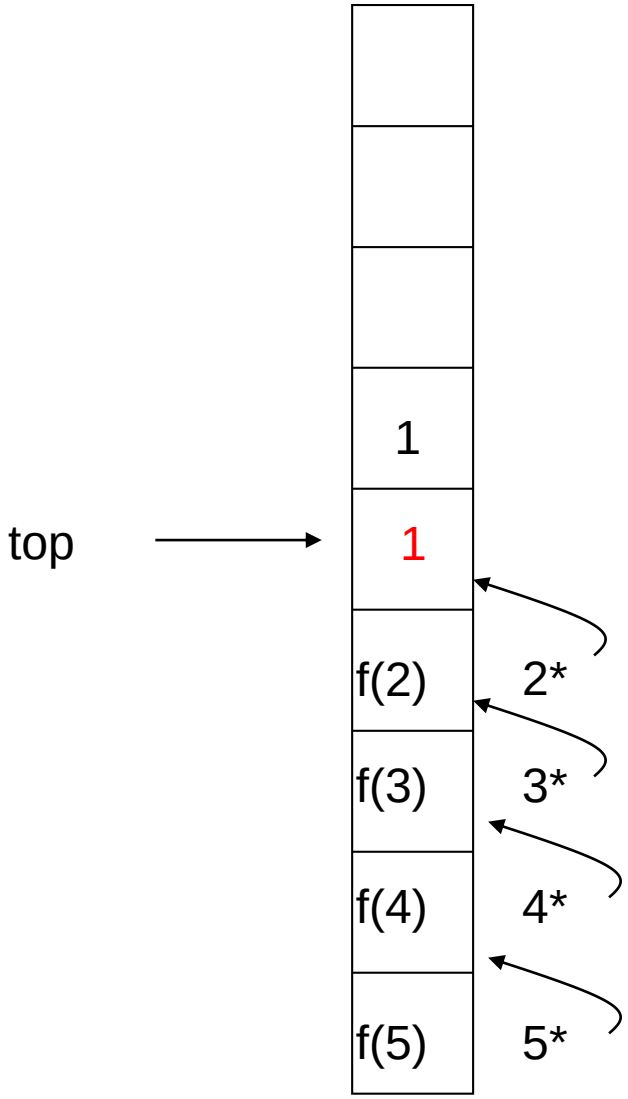
f(5) = 5*f(4)
f(4) = 4*f(3)
f(3) = 3*f(2)
f(2) = 2*f(1)
f(1) = 1*f(0)
f(0) = 1



PILHA (APLICACAO)

f(0) = 1
f(n) = n*f(n-1)

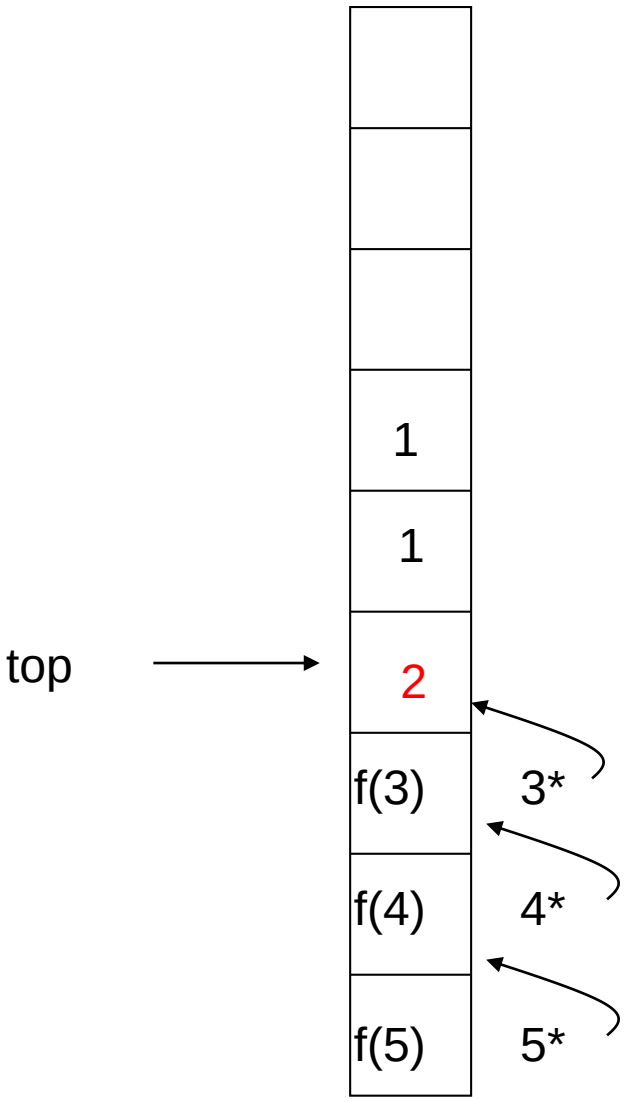
f(5) = 5*f(4)
f(4) = 4*f(3)
f(3) = 3*f(2)
f(2) = 2*f(1)
f(1) = 1*1



PILHA (APLICACAO)

f(0) = 1
f(n) = n*f(n-1)

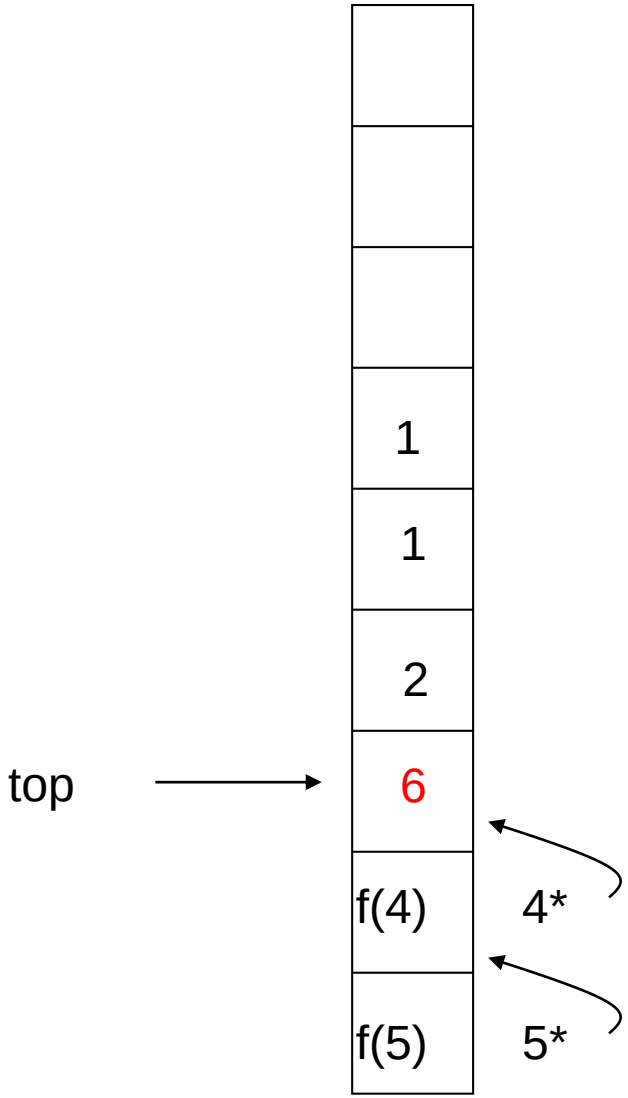
f(5) = 5*f(4)
f(4) = 4*f(3)
f(3) = 3*f(2)
f(2) = 2*1*1



PILHA (APLICACAO)

f(0) = 1
f(n) = n*f(n-1)

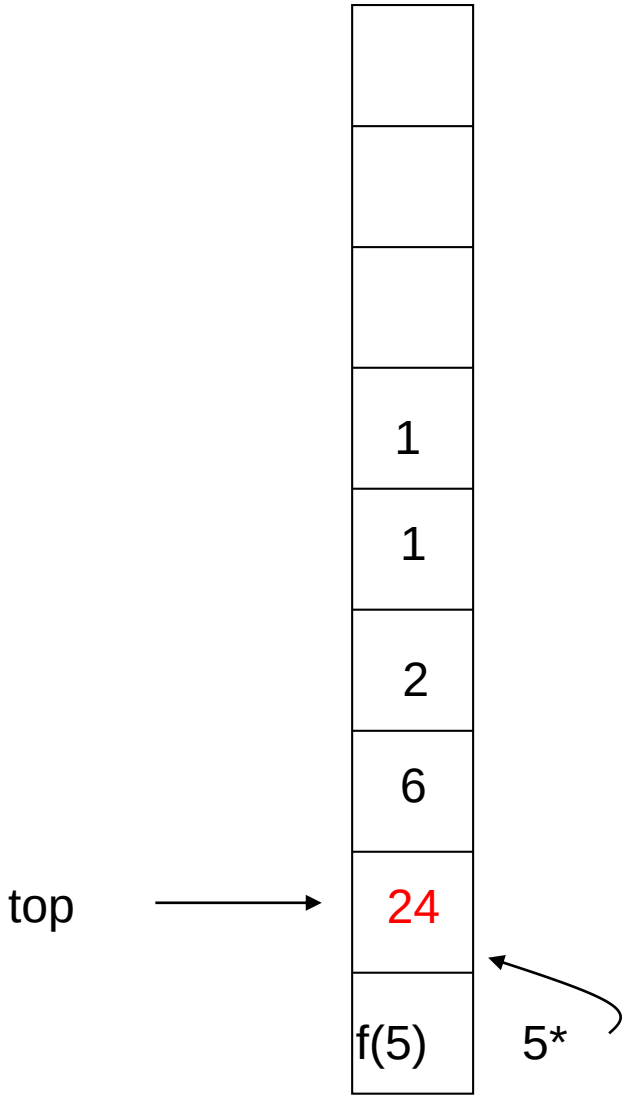
f(5) = 5*f(4)
f(4) = 4*f(3)
f(3) = 3*2*1*1



PILHA (APLICACAO)

f(0) = 1
f(n) = n*f(n-1)

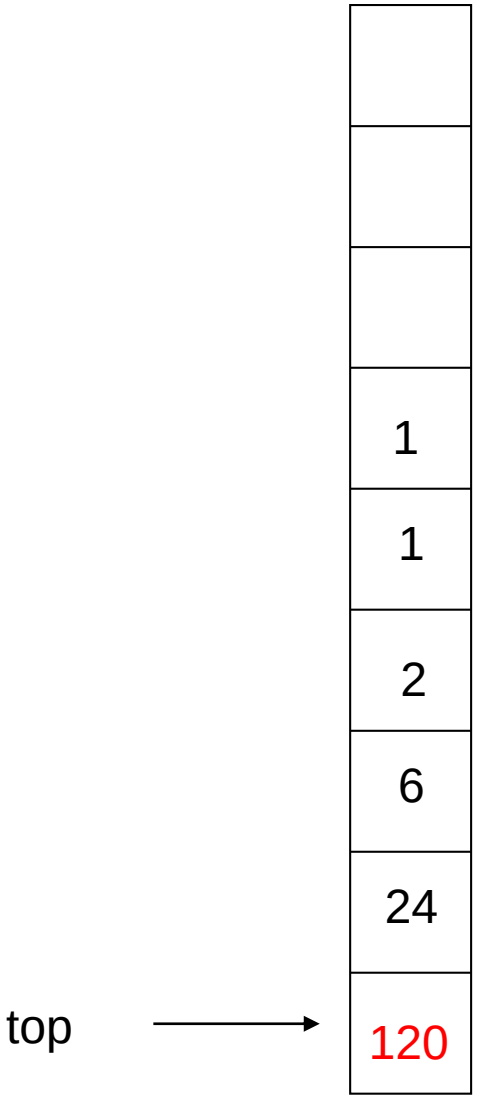
f(5) = 5*f(4)
f(4) = 4*3*2*1*1



PILHA (APLICACAO)

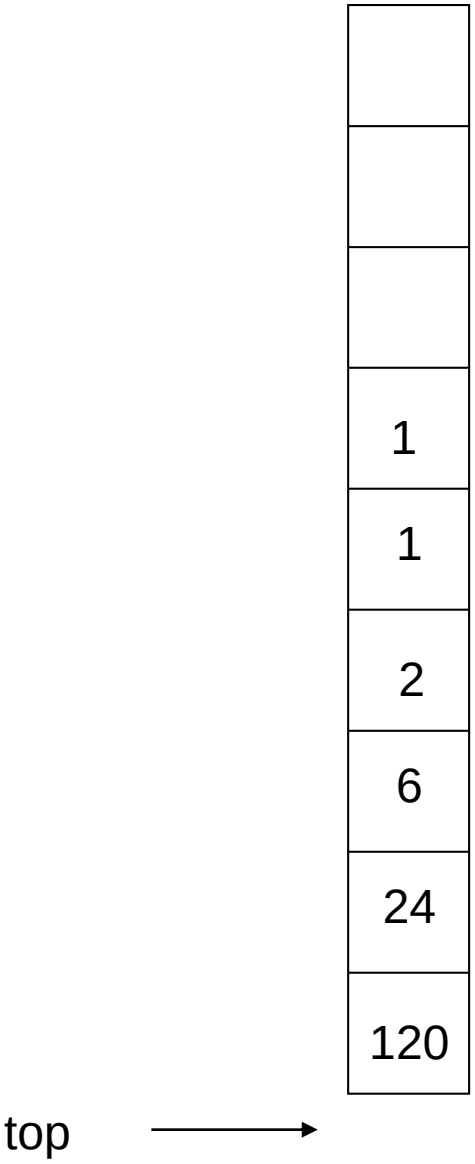
f(0) = 1
f(n) = n*f(n-1)

f(5) = 5*4*3*2*1*1



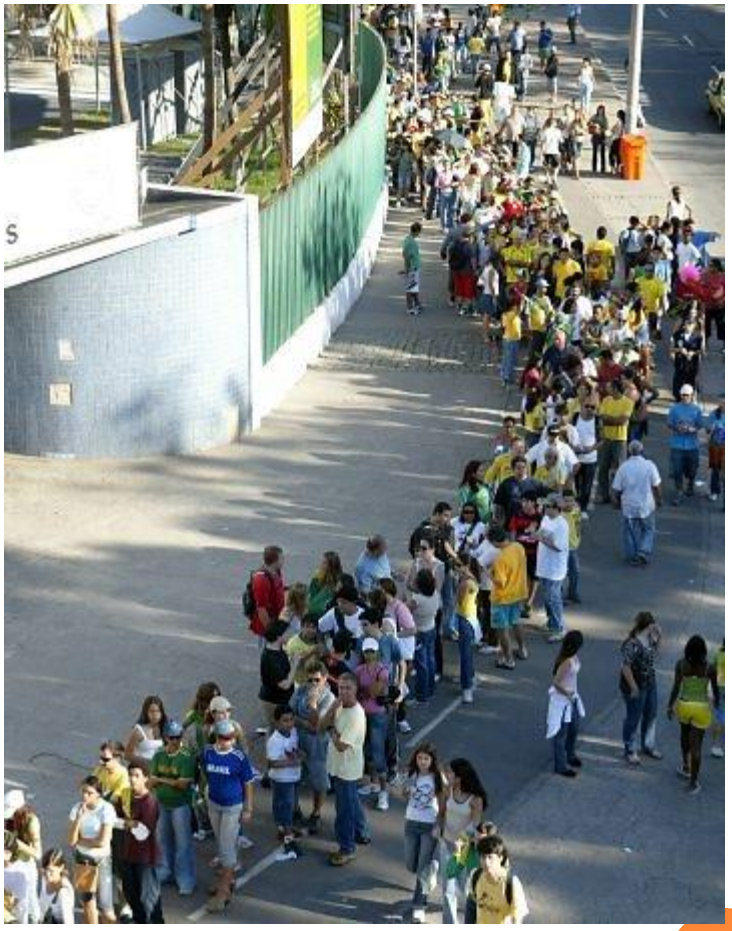
Pilha (aplicacao)

f(0) = 1
f(n) = n*f(n-1)



retornado
120

Fila (Queue) - Intuição



Fila (Queue)

- Conjunto dinamico com políticas de acesso específicas
- Definição
  - Estrutura de dados em que a respeita a ordem temporal dos elementos.
  - Os elementos são introduzidos na cauda
  - Os elementos são removidos da cabeça
  - **FIFO** = ***First In, First Out.***

Fila (Queue)

- Operações (interface)
  - Criar uma pilha vazia (create)
  - Inserir/Enfileirar (enqueue)
  - Deletar/Remover (dequeue)
  - Acessar o elemento da cabeça (head)
  - Verificar se está vazia (isEmpty)
  - Verificar se está cheia (isFull)

Fila (Queue)

- Interface em Java
  - Como descrever uma Fila genérica em Java?

```
public interface Queue<T> {
  public void enqueue(T elem) throws QueueOverflowException;
  public T dequeue() throws QueueUnderflowException;
  public T head();
  public boolean isEmpty();
  public boolean isFull();
}

public class QueueOverflowException extends Exception {
 public QueueOverflowException() {
  super("Fila cheia");
 }
}
```

Fila (Queue)

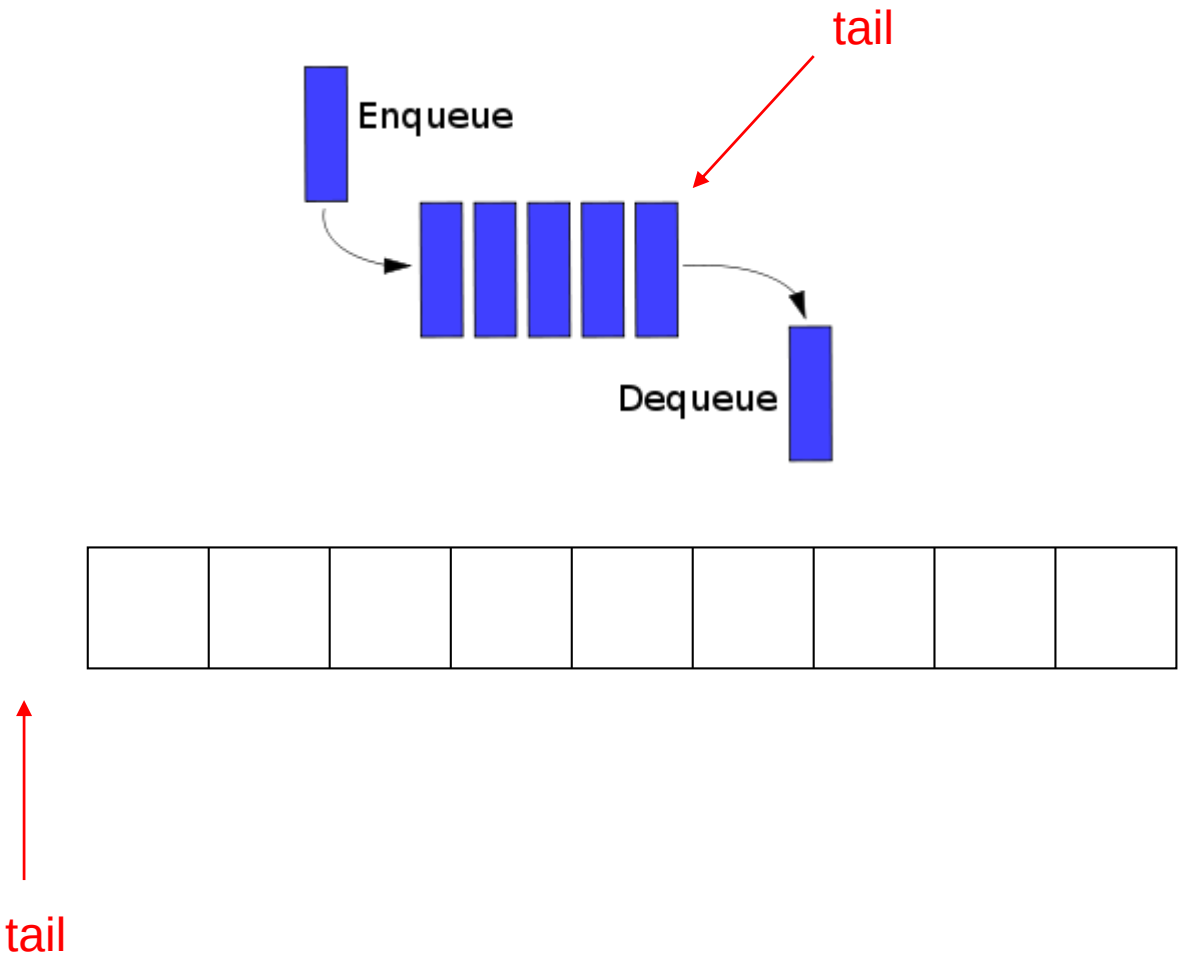
- Interface em Java
  - Como descrever uma Fila genérica em Java?

```
public interface Queue<T> {
  public void enqueue(T elem) throws QueueOverflowException;
  public T dequeue() throws QueueUnderflowException;
  public T head();
  public boolean isEmpty();
  public boolean isFull();
}
```

E o create da fila?

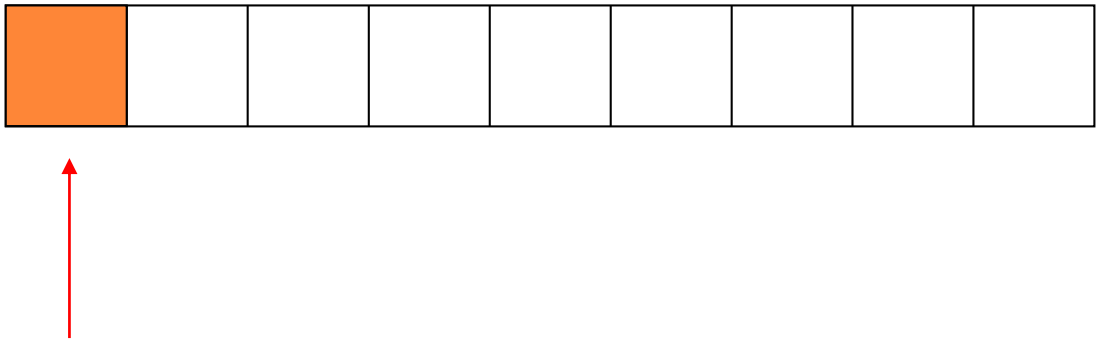
```
public class QueueOverflowException extends Exception {
 public QueueOverflowException() {
  super("Fila cheia");
 }
}
```

FILA (FUNCIONAMENTO)



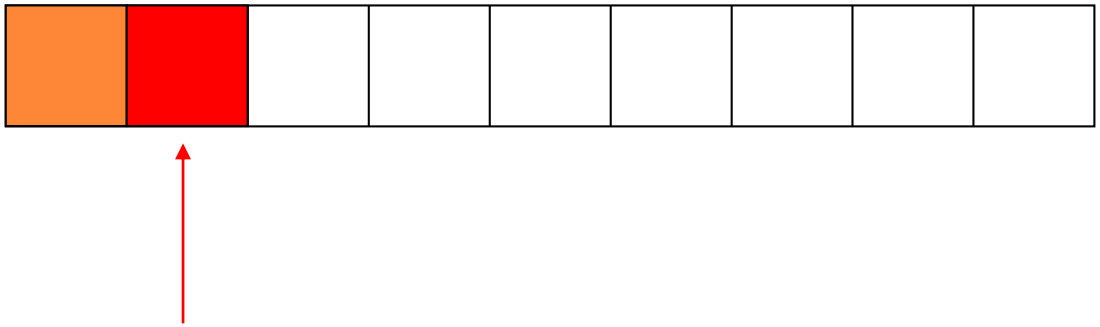
Fila (funcionamento)

enqueue



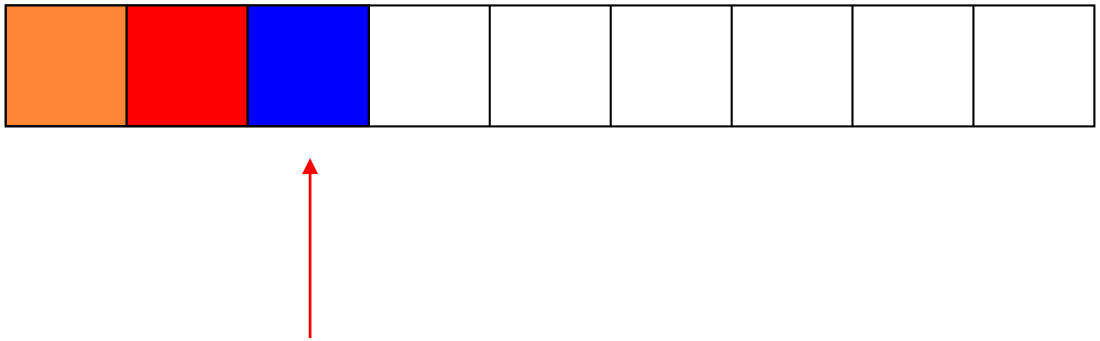
Fila (funcionamento)

enqueue
enqueue



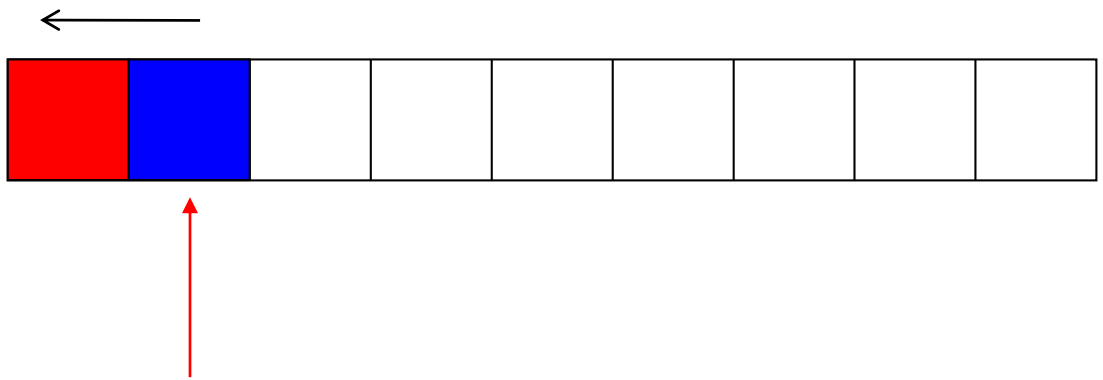
Fila (funcionamento)

enqueue
enqueue
enqueue



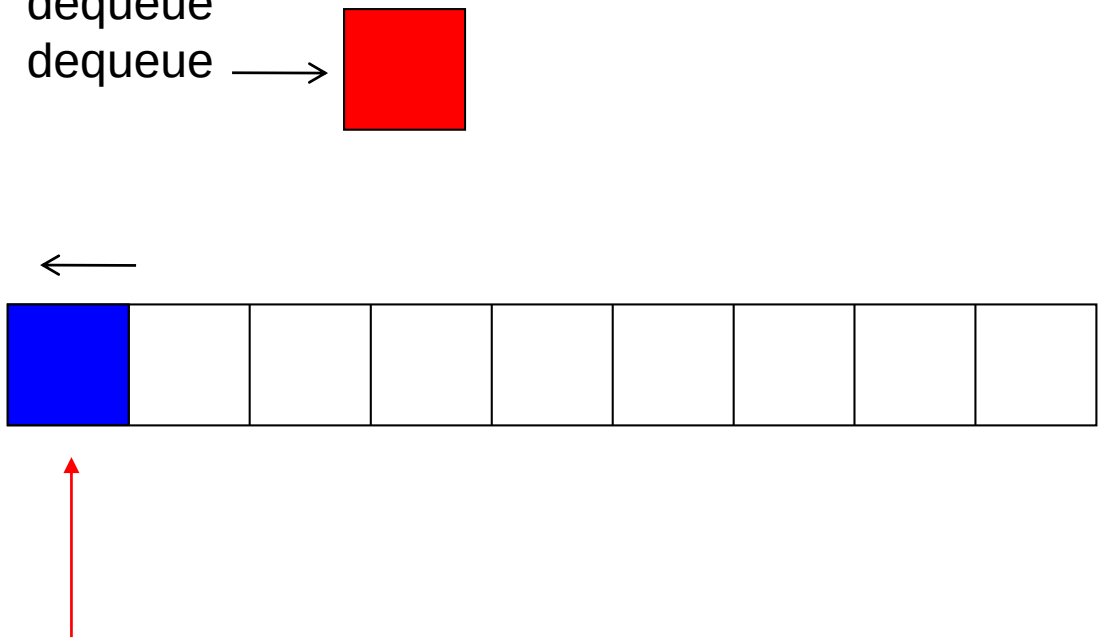
Fila (funcionamento)

enqueue
enqueue
enqueue
dequeue



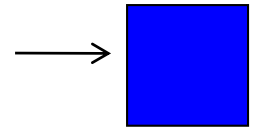
Fila (funcionamento)

enqueue
enqueue
enqueue
dequeue
dequeue



FILA (FUNCIONAMENTO)

enqueue
enqueue
enqueue
dequeue
dequeue
dequeue



tail

Fila (funcionamento)

enqueue
enqueue
enqueue
dequeue
dequeue
dequeue
dequeue

QueueUnderflowException

Fila (Implementação)

```
isEmpty(){
   return tail == -1
}
```

```
isFull(){
   return tail == A.length-1
}
```

```
enqueue(item){
   if (not isFull()){
      A[++tail]=item
   }else{
     //error:queue is full
   }
}
```

```
dequeue(){
   if (not isEmpty()){
      result = A[0]
      shiftLeft()
      tail--;
   }else{
     //error:queue is empty
   }
   return result;
}
```

Fila (Implementação)

```
isEmpty(){
   return tail == -1
}
```

```
isFull(){
   return tail == A.length-1
}
```

```
enqueue(item){
   if (not isFull()){
      A[++tail]=item
   }else{
     //error:queue is full
   }
}
```

```
dequeue(){
   if (not isEmpty()){
      result = A[0]
      shiftLeft()
      return result
   }else{
     //error:queue is empty
   }
   return result;
}
```

```
shiftLeft(){
   for i : 0 .. tail{
      A[i] = A[i+1]
   }
}
```

Fila (Implementação)

Qual a complexidade?

```
isEmpty(){
   return tail == -1
}
```

```
isFull(){
   return tail == A.length-1
}
```

```
enqueue(item){
   if (not isFull()){
      A[++tail]=item
   }else{
     //error:queue is full
   }
}
```

```
dequeue(){
   if (not isEmpty()){
      result = A[0]
      shiftLeft()
      tail--;
   }else{
     //error:queue is empty
   }
   return result;
}
```

Fila (Implementação)

Qual a complexidade?

```
isEmpty(){
   return tail == -1
}
```
$\Theta(1)$

```
isFull(){
   return tail == A.length-1
}
```
$\Theta(1)$

```
enqueue(item){
   if (not isFull()){
      A[++tail]=item
   }else{
     //error:queue is full
   }
}
```
$\Theta(1)$

```
dequeue(){
   if (not isEmpty()){
      result = A[0]
      shiftLeft()
      return result;
   }else{
     //error:queue is empty
   }
   return result;
}
```
$\Theta(n)$

Fila (Implementação)

- Como evitar a operação de shift em uma fila?

Fila (Implementação)

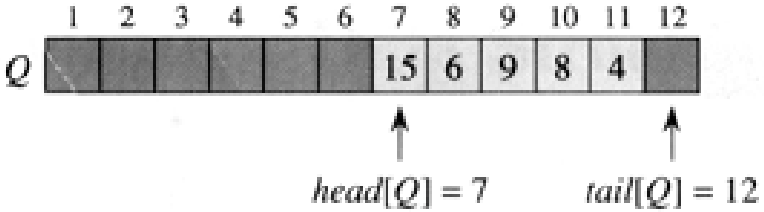
- Como evitar a operação de shift em uma fila?
  - Sabendo o início e o fim da fila!

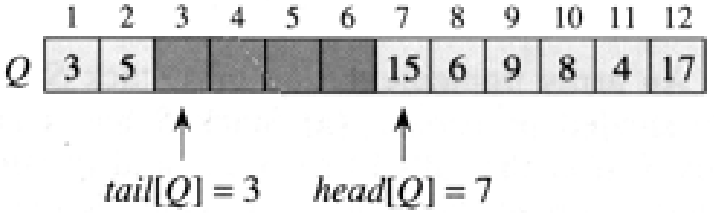
Exercício

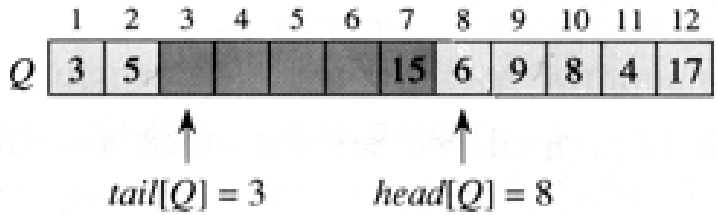
- Como fica o estado de uma fila inicialmente vazia após a execução dos comandos:
  - enqueue(10)
  - enqueue(5)
  - dequeue()
  - enqueue(7)
  - head()
  - dequeue()
  - isEmpty()

Fila (Implementação)

- Conhecendo o início e o fim da fila

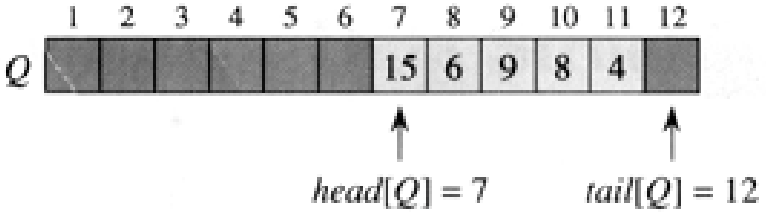






Fila (Implementação)

- Conhecendo o início e o fim da fila



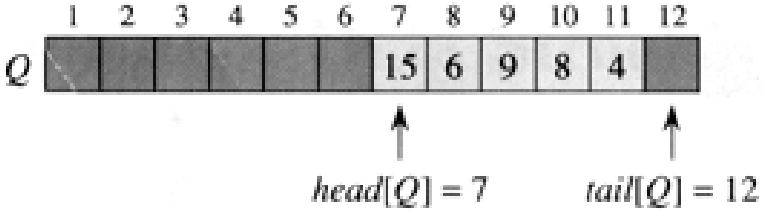

```
ENQUEUE(Q, x)
1  Q[tail[Q]] ← x
2  if tail[Q] = length[Q]
3      then tail[Q] ← 1
4      else tail[Q] ← tail[Q] + 1
```

```
DEQUEUE(Q)
1  x ← Q[head[Q]]
2  if head[Q] = length[Q]
3      then head[Q] ← 1
4      else head[Q] ← head[Q] + 1
5  return x
```


Qual o custo das operações?

Fila (Implementação)

- Conhecendo o início e o fim da fila



ENQUEUE(Q, x) $\Theta(1)$

```
1  Q[tail[Q]] ← x
2  if tail[Q] = length[Q]
3      then tail[Q] ← 1
4      else tail[Q] ← tail[Q] + 1
```

DEQUEUE(Q) $\Theta(1)$

```
1  x ← Q[head[Q]]
2  if head[Q] = length[Q]
3      then head[Q] ← 1
4      else head[Q] ← head[Q] + 1
5  return x
```


Qual o custo das operações?

Problemas Interessantes

- Implementar uma Fila utilizando Pilhas (ou Pilha utilizando Filas) sobrejacentes
- Implementar uma Pilha que lê uma sequência de parênteses e verifique se a sequência é bem formada. Ex:
  -)(-> false
  -)()(()) -> true
  -))((-> false
  - ()) -> false
- Implementar uma estrutura híbrida, que funcione como pilha e fila ao mesmo tempo. A pilha funciona em uma extremidade e a fila funciona na outra extremidade.

Referências

- Capítulo 11