PILHAS

Tipos abstratos de dados (TADs)

- Um TAD é uma abstração de uma estrutura de dados
- Um TAD especifica:
  - Dados armazenados
  - Operações sobre os dados
  - Condições de erros associadas à opers
- Exemplo: TAD que modela um sistema de controle de estoque
  - Os dados são os pedidos de compra/venda
  - As operações suportadas são:
    - comprar(produto, preço)
    - vender(produto, preço)
    - cancelar(pedido)
  - Condições de erro:
    - Comprar/vender um produto não existente
    - Cancelar um pedido não existente

O TAD Pilha

- O TAD Pilha armazena *objetos* arbitrários
- Inserções e remoções segue o esquema *LIFO*
- Exemplo: uma pilha de pratos
- Principais operações:
  - push(object): insere um elemento
  - object pop(): remove e returna o último elemento inserido
- Operações auxiliares:
  - object top(): retorna o último elemento inserido sem removê-lo
  - integer size(): retorna o número de lementos armazenados
  - boolean isEmpty(): indica se há ou não elementos na Pilha

Exceções

- Ao executar uma operação em um TAD, pode-se causar uma condição de erro, que chamamos exceção
- Execções podem ser *levantadas* (*thrown*) por uma operação que não pode ser executada
- No TAD Pilha, as operações *pop* e *top* não podem ser realizadas se a pilha está vazia
- Executar *pop* ou *top* numa pilha vazia causa a exceção EPilhaVazia

Aplicações de pilhas

- Aplicações diretas
  - Histórico de páginas visitadas num navegador
  - Sequência de desfazer em um editor de textos
  - Cadeia de chamada de métodos num programa
- Aplicações indiretas
  - Estrutura de dados auxiliares para algoritmos
  - Componentes de outras estruturas de dados

Pilhas baseadas em *Arrays*

- Uma forma simples de implementar uma pilha usa *arrays*
- Adicionamos elementos da esquerda para a direita
- Uma variável mantém o índice do elemento no topo da pilha

```
Algoritmo size()
  retorne t + 1

Algoritmo pop()
  Se (estaVazia())
    throw EPilhaVazia
  senão
    t ← t - 1
    retorne S[t + 1]
```

Pilhas baseadas em *Arrays*

- O *array* pode ficar cheio
- A operação *push* pode então levantar a exceção EPilhaCheia
  - Está é uma limitação da implementgação baseada em *arrays*
  - Não é intrínsico do TAD Pilha

Algoritmo ***push(o)***
  Se ($t = S.length - 1$ **)**
    throw ***EPilhaCheia***
  senão
    $t \leftarrow t + 1$
    $S[t] \leftarrow o$

Desempenho e limitações

- Desempenho
  - Seja n o número de elemento na pilha
  - O espaço usado é $O(n)$
  - Cada operação roda em tempo $O(1)$
- Limitações
  - O tamanho máximo deve ser definido a priori e não pode ser mudado
  - Tentando colocar um novo elemento numa pilha cheia causa uma exceção específica da implementação (*array*)

Pilha crescente baseada em *array*

- Em uma operação *push()*, quando o *array* está cheio, ao invés de levantar uma exceção, substituímos o *array* por um maior
- Qual o tamanho do *array*?
  - Estratégia incremental: aumentar o *array* usando uma constante c
  - Estratégia de duplicação: duplicar o tamanho do *array*

```
Algoritmo colocar(o)
  Se (t = S.length-1)
                então
    A ← novo array
    para i ← 0 até t faça
       A[i] ← S[i]
       S ← A
  t ← t + 1
  S[t] ← o
```

Comparação de estratégias

- Comparamos a estratégia incremental e de duplicação analisando o tempo total $T(n)$ necessário para realizar uma série de n operações *push*
- Assumimos que começamos com uma pilha vazia representada por um *array* de tamanho 1
- Chamamos tempo de amortização de uma operação *push* o tempo médio de uma operação sobre uma série de operações-$T(n)/n$

Análise da estratégia incremental

- Substituímos o *array* $k = n/c$ vezes
- O número toral $T(n)$ de uma série de n operações *push* é proporcional a

$$n + c + 2c + 3c + 4c + \ldots + kc =$$

$$n + c(1 + 2 + 3 + \ldots + k) =$$

$$n + ck(k + 1)/2$$

- Como c é uma constante, $T(n)$ é $O(n + k^2)$, i.e., $O(n^2)$
- O tempo amortizado de uma operação *push* é $O(n)$

Análise da estratégia de duplicação

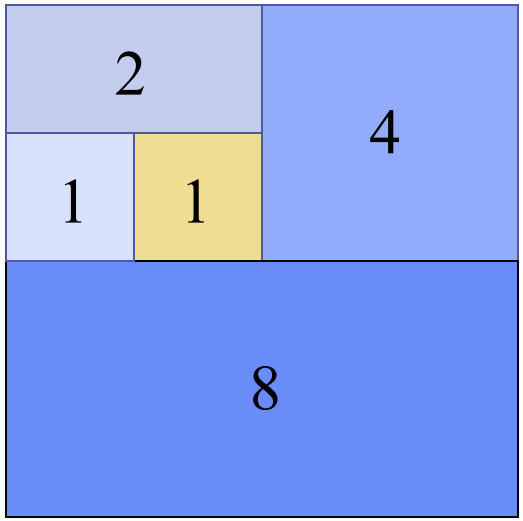
- Substituímos o *array* $k = \log_2 n$ vezes
- O tempo total $T(n)$ de uma série n de operações *push* é proporcional a

$$n + 1 + 2 + 4 + 8 + \ldots + 2^k =$$

$$n + 2^{k+1} - 2 = 3n - 2$$

- $T(n)$ é $O(n)$
- O tempo amortizado de uma operação *push* é $O(1)$

Série geométrica



Pilha baseada nas estrategias de crescimento e duplicação

- Interface JAVA correspondente ao nosso TAD Pilha
- Requer a definição da classe EPilhaVazia
- Diferente da classe interna JAVA java.util.Stack

```
public interface Pilha {
    public int size();
    public boolean isEmpty();
    public Object top() throws pilhaVaziaExcecao;
    public void push(Object o);
    public Object pop() throws PilhaVaziaExcecao;
```

PilhaArray – classe PilhaVaziaExcecao

```
public class PilhaVaziaExcecao extends
RuntimeException {
      public PilhaVaziaExcecao(String err){
         super(err);
      }
}
```

PilhaArray – atributos e construtor

```
public class PilhaArray implements Pilha {
    private int capacidade;
    private Object[] a;
    private int t;
    private int FC;
    public PilhaArray(int capacidade, int
                                  crescimento){
        this.capacidade=capacidade;
        t=-1;
        FC=crescimento;
         if (crescimento<=0)
             FC=0;
        a=new Object[capacidade];
    }
```

PilhaArray - Push

```
public void push(Object o) {
        if(t>=capacidade-1){
          if(FC==0)
            capacidade*=2;
          else
            capacidade+=FC;
          Object b[]=new Object[capacidade];
          for(int f=0;f<a.length;f++)
            b[f]=a[f];
          a=b;
        }
        a[++t]=o;
    }
```

PilhaArray - Pop

```
 public Object pop()throws PilhaVaziaExcecao {
        if(isEmpty())
          throw new PilhaVaziaExcecao("A Pilha está
vazia");
        Object r=a[t--];
        return r;
    }
```

PilhaArray - Top

```
 public Object top()throws PilhaVaziaExcecao {
        if(isEmpty())
          throw new PilhaVaziaExcecao("A Pilha
está vazia");
        return a[t];
    }
```

PilhaArray – isEmpty, size

```
 public boolean isEmpty(){
      return t==-1;
    }

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

Dúvidas