Recursão Generativa

Fundamentos de Algoritmos

INF05008

Recursão Estrutural

- Até aqui vimos funções que decompõem um dado de entrada em seus componentes
- Se um desses componentes pertence à **própria classe** do dado de entrada, a função é **recursiva**
- Este tipo de recursão é chamado de **recursão estrutural**
- É um processo **disciplinado**, baseado na **estrutura dos dados de entrada**

Recursão Generativa

- **Entrada** para um algoritmo **representa um problema**
- **Decomposição** do problema em **subproblemas**
- Se **um dos subproblemas for da mesma classe** do problema inicial, temos **recursão generativa**
- Processo **não é tão disciplinado** como na recursão estrutural
- Parte **criativa** pode exigir **mais conhecimento do domínio e inspiração** por parte do programador

Exemplo: Movendo uma bola sobre uma mesa

- *"Desenvolver a função* `move-até-cair` *que, dada uma bola, a movimenta a uma velocidade constante sobre uma mesa até que ela caia"*

- **Elementos a representar?**

Exemplo: Movendo uma bola sobre uma mesa

- *"Desenvolver a função* `move-até-cair` *que, dada uma* ***bola*** *, a* ***movimenta*** *a uma velocidade constante sobre uma* ***mesa*** *até que ela caia"*

- Elementos a representar:
  - A mesa
  - A bola
  - O movimento da bola

Exemplo: Movendo uma bola sobre uma mesa

- Como representar a **mesa** ?

Exemplo: Movendo uma bola sobre uma mesa

- Como representar a **mesa** ?
- A mesa pode ser modelada por **uma tela de tamanho fixo**

Exemplo: Movendo uma bola sobre uma mesa

- Como representar a **bola** ?

Exemplo: Movendo uma bola sobre uma mesa

- Como representar a **bola** ?
- A bola pode ser modelada por **um disco que cruza a tela**

Exemplo: Movendo uma bola sobre uma mesa

- Como representar o **movimento da bola** ?

Exemplo: Movendo uma bola sobre uma mesa

- Como representar o **movimento da bola** ?

- O movimento da bola pode ser representado da seguinte forma:

1. **Desenhar disco** em uma dada posição da tela,
2. **Aguardar** por um tempo,
3. **Apagar disco**, e
4. **Redesenhar disco em outra posição**,
5. **Repetir os passos 1 a 4** até que disco fique fora da tela

Exemplo: Movendo uma bola sobre uma mesa - completo

- A **mesa** pode ser modelada por **uma tela de tamanho fixo**
- A **bola** pode ser modelada por **um disco que cruza a tela**
- O **movimento da bola** pode ser representado da seguinte forma:

1. **Desenhar disco** em uma dada posição da tela,
2. **Aguardar** por um tempo,
3. **Apagar disco**, e
4. **Redesenhar disco em outra posição**,
5. **Repetir os passos 1 a 4** até que disco fique fora da tela

Definição de Dados

```
(define-struct bola (x y incr-x incr-y))
;; Uma bola é uma estrutura
;;    (make-bola x y incr-x incr-y)
;; onde x, y, incr-x e incr-y são números.
```

- Uma bola é uma estrutura com informações sobre a **posição corrente** e o **incremento ao deslocamento** em cada direção

- Ou seja, os primeiros dois números são as coordenadas na tela e os dois últimos são os valores de deslocamento em cada direção em que a bola é redesenhada

Função de Redesenho

```
;; desenha-e-apaga : bola -> true
;; Desenha, espera e limpa disco da tela
;; -> Projeto estrutural
(define (desenha-e-apaga uma-bola)
    (and
        (draw-solid-disk (make-posn (bola-x uma-bola)
                                    (bola-y uma-bola))
                         5 'red)
        (sleep-for-a-while TEMPO-DE-ESPERA)
        (clear-solid-disk (make-posn (bola-x uma-bola)
                                     (bola-y uma-bola)) 5 'red)))
```

- Essa função desenha um disco sólido (bola), espera por um período TEMPO-DE-ESPERA e depois apaga a bola da tela

Função que Move a Bola

```
;; move-bola : bola -> bola
;; Cria uma nova bola, modelando movimento
;; -> Conhecimento de Física

(define (move-bola uma-bola)
   (make-bola (+ (bola-x uma-bola) (bola-incr-x uma-bola))
              (+ (bola-y uma-bola) (bola-incr-y uma-bola))
              (bola-incr-x uma-bola)
              (bola-incr-y uma-bola)))
```

- **Consome** uma bola e **cria** uma outra, modelando um passo do movimento

Efetuando Movimentos da Bola

Para mover a bola algumas vezes, poderíamos escrever:

```
(define uma-bola (make-bola 150 20 -5 +17))
...
(and
(desenha-e-apaga uma-bola)
(and
(desenha-e-apaga (move-bola uma-bola))
   ....))
```

Efetuando Movimentos da Bola

Para mover a bola algumas vezes, poderíamos escrever:

```
(define uma-bola (make-bola 150 20 -5 +17))
...
(and
(desenha-e-apaga uma-bola)
(and
(desenha-e-apaga (move-bola uma-bola))
   ....))
```

- **Melhor escrever uma função `move-até-cair` que move uma bola até que ela saia dos limites da tela**

Bola Fora dos Limites da Tela

- A parte fácil: função `fora-dos-limites?` que, dada uma bola, determina se ela está fora dos limites da tela

```
;; fora-dos-limites? : bola -> boolean
;; Determina se uma bola está fora dos limites da tela
;; -> Conhecimento de Geometria

(define (fora-dos-limites? uma-bola)
  (not
   (and
     (<= 0 (bola-x uma-bola) LARGURA)
     (<= 0 (bola-y uma-bola) ALTURA))))
```

Move Bola Enquanto Estiver na Tela

- Função que, dada uma bola, a move até que ela fique fora dos limites da tela **não é definida com base na estrutura do dado de entrada**

```
;; move-até-cair : bola -> true
;; Modela o movimento de uma bola até que ela fique fora dos
;; limites da tela

(define (move-até-cair uma-bola)
 (cond
   [(fora-dos-limites? uma-bola) true]
   [else (and (desenha-e-apaga uma-bola)
              (move-até-cair (move-bola uma-bola)))]))
```

Move Bola Enquanto Estiver na Tela

- Corpo da função tem um `cond`, mas **as duas cláusulas não tem a ver com a definição do tipo de dado de entrada** (como ocorre em recursão estrutural)

- **Parte recursiva da função não consome parte do dado**, mas tem como entrada uma nova bola (que representa a original, mas movida para uma nova posição)

- As receitas de projeto vistas até aqui não levam a uma definição de função como essa

- Temos, portanto, **uma nova forma de programar**

Testando a Função

```
;; Dimensão da tela
 (define LARGURA 500)
 (define ALTURA 500)

;; Dimensão e cor da bola
 (define RAIO 10)
 (define COR 'red)

;; Duração da espera
 (define TEMPO-DE-ESPERA 0.9)

;; Cria a tela, movimenta bola até ela cair e pára
 (start LARGURA ALTURA)
 (move-até-cair (make-bola 150 20 -5 +17))
 (stop)
```

Exercícios

1. Descreva o quê acontece se executarmos o seguinte código:

```
(start LARGURA ALTURA)
(move-até-cair (make-bola 150 20 0 0))
(stop)
```

2. Escreva a função `move-bolas` que, dada uma lista de bolas, move cada uma até que todas estejam fora do limite da mesa

QuickSort

- Exemplo clássico de recursão generativa
- Como o `insertion sort` (visto anteriormente), `quick sort` recebe uma **lista de números** como entrada e produz uma versão dessa lista com os números **ordenados em ordem crescente**
- O `insertion sort` é baseado em **recursão estrutural** enquanto que o `quick sort`, em **recursão generativa**

QuickSort

- Estratégia de **divisão-e-conquista**
- **Dividimos instância não-trivial** do problema em dois problemas menores relacionados
- Resolvemos esses problemas e **combinamos as soluções** em uma solução para o problema original

QuickSort

- No caso do `quick sort`, a lista de entrada é dividida em duas partes:
  - Uma que contém todos os itens **estritamente menores que o primeiro item da lista**
  - Outra que contém todos os itens **estritamente maiores do que o primeiro elemento**
- **Ambas as listas são ordenadas usando o mesmo procedimento**
- Quando as duas listas estiverem ordenadas, fazemos a **justaposição de ambas**
- O primeiro elemento é chamado de **pivô**

- O `quick sort` distingue dois casos
  - Se entrada é `empty`, produz `empty`
  - Caso contrário: **recursão generativa sobre dois problemas menores**

```
;; quick-sort : lista-de-números -> lista-de-números
;; Cria lista de nros com os mesmos números da lista de
;; entrada, mas ordenada em ordem crescente

(define (quick-sort ldn)
 (cond
   [(empty? ldn) empty]
   [else ...]))
```

- **Usa-se o primeiro elemento para particionar a lista** em duas listas menores:
  - Lista com todos os elementos estritamente **menores** do que o pivô
  - Lista com todos os elementos estritamente **maiores** do que o pivô

- A tarefa da partição fica a cargo de duas **funções auxiliares** (definidas usando **recursão estrutural**)

Elementos Maiores

```
;; maiores : lista-de-números número -> lista-de-números
;; Cria lista com todos os números da lista de entrada
;; que são maiores do que o nro passado como argumento

(define (maiores ldn n)
 (cond
   [(empty? ldn) empty]
   [else
     (cond
       [(> (first ldn) n) (cons (first ldn)
                                (maiores (rest ldn) n))]
       [else (maiores (rest ldn) n)])]))
```

Elementos Menores

```
;; menores : lista-de-números número -> lista-de-números
;; Cria lista com todos os números da lista de entrada
;; que são menores do que o nro passado como argumento

(define (menores ldn n)
 (cond
   [(empty? ldn) empty]
   [else
     (cond
       [(< (first ldn) n) (cons (first ldn)
                                (menores (rest ldn) n))]
       [else (menores (rest ldn) n)])]))
```

Função `quick-sort`

```
;; quick-sort : lista-de-números -> lista-de-números
;; Cria uma lista de nros com os mesmos nros da lista de
;; entrada, mas ordenados em ordem crescente
;; Assume que todos os nros são diferentes

 (define (quick-sort ldn)
  (cond
    [(empty? ldn) empty]
    [else
      (append (quick-sort (menores ldn (first ldn)))
              (list (first ldn))
              (quick-sort (maiores ldn (first ldn))))]))
```