Programação Funcional

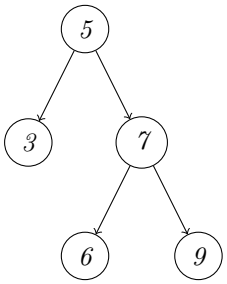
Resolução da Frequência

Universidade da Beira Interior

2021-2022

Exercício 1 *Vamos considerar aqui uma forma alternativa para ordenar uma sequência de valores. Esta utiliza como recurso auxiliar uma* ***árvore binária de pesquisa*** *(BST). Primeiro constrói-se uma tal árvore com os elementos da sequência original e devolve-se a sequência que resulta de um percurso* depth-first search *(DFS).*

Por exemplo dado a sequência `[5;3;7;6;9]` *a árvore binária de pesquisa que lhe corresponde é*



e a listagem DFS resultante é `[3;5;6;7;9]`

1. *Qual é o pior caso (pior cenário) neste método de ordenação?*
2. *Qual é a ordem de complexidade do método sugerido neste pior caso?*
3. *Considere-se para esta alínea uma sequência por ordenar de tamanho N. Em termos do seu consumo de memória, e para além da sequência por ordenar em si, qual é a ordem de grandeza da memória extra para executar este método de ordenação?*
4. *Considere o tipo das árvores binárias de pesquisa seguinte:*

   ```
   type 'a bst = E | N of 'a bst * 'a * 'a bst
   ```

   Dê uma implementação OCaml da função `add_bst comp x t` *que devolve a árvore que resulta da inserção do valor* `x` *na arvore* `t` *conforme o* ***critério de ordenação*** `comp` *(como é habitual, de tipo* `'a -> 'a -> int`*) e seguinte o* ***critério de formação de uma árvore binária de pesquisa****.*

5. *Dê uma implementação OCaml da função* `tolist_dfs t` *que devolve a listagem dos valores contidos na árvore* `t` *seguindo a estratégia "*em profundidade primeiro, da esquerda para a direita*". Para informação, a função* `tolist_dfs t` *tem por tipo* `'a bst -> 'a list`*.*

6. *Dê uma implementação OCaml da função* `bst_sort comp l` *que ordena a lista em parâmetro* `l` *conformo o critério de comparação* `comp`*, utilizando árvores binárias de pesquisa e em particular as funçoes das alíneas anteriores. Para além da justeza da solução, será valorizado o uso de combinadores sobre listas (como, a título de exemplo,* `List.map`*)*

7. *Há um elefante na sala. Este método como sugerido aqui não lida adequadamente com todas as sequências por ordenar. Indica com precisão qual é o problema.*

Resposta:

1. O pior cenário é o de uma árvore de tipo pente. Isso acontece quando a sequência dos elementos por juntar já se encontra ordenada (no sentido crescente ou no sentido decrescente).

2. Neste caso construir a árvore obriga a um percurso "dispendioso" de todos elementos presentes na árvore. Na primeira vez, custa 0. Na segunda vez, custa 1. Na terceira custa 2 etc... Assim temos uma construção da árvore que tem um custo quadrático ($0 + 1 + 2 + 3... + N - 1$, ordem N^2, sendo N o tamanho da sequência).

3. Em termos espaciais (i.e. de memória), os custos são lineares. Precisamos de N nodos.

4. ```
   type 'a bst = E | N of 'a bst * 'a * 'a bst

   let rec add_bst comp x t =
     match t with
     | E          -> N (E,x,E)
     | N (l,v,r) ->
       let c = comp x v in
       if c<0       then N (add_bst comp x l,v,r)
       else if c>0 then N (l,v,add_bst comp x r)
       else t
   ```

5. Na sua versão simples, não recursiva terminal:

   ```
   let rec tolist_dfs = function
     | E          -> []
     | N (l,v,r) -> tolist_dfs l @ (v::tolist_dfs r)
   ```

6. Ordenação:

   ```
   let bst_sort comp l =
       tolist_dfs (List.fold_left (fun a e -> add_bst comp e a) E l)
   ```

   ou

   ```
   let bst_sort comp l =
      l |> List.fold_left (fun a e -> add_bst comp e a) E |> tolist_dfs
   ```
   ```

7. A inserção (i.e. pela função add_bst) de um elemento x numa árvore que já contempla o elemento x não tem efeito nenhum: Não há duplicados numa BST. É esse o elefante na sala! Uma sequência com duplicados processada por este método resultará numa sequência ordenada mas que não será uma permutação da sequência original. Em particular terá um comprimento diferente. E a isso chamamos um bug. Ou uma "feature", se isso não vos chocar.

```
# bst_sort compare  [4;7;2;9;4;6;2;1;7;12] ;;
- : int list = [1; 2; 4; 6; 7; 9; 12]

(* deveria ser [1; 2; 2; 4; 4; 6; 7; 7; 9; 12] *)
```

□