Universidade Federal de Pernambuco
Centro de Informática
Bach. em Engenharia da Computação

IF672ec Algoritmos e Estr. de Dados 2018.1
Prof. Paulo Fonseca

Primeira Prova — 26 de Abril de 2018

QUESTÃO 1

Considere o algoritmo a seguir.

Algoritmo *rmd*
Entrada *head* ptr para lista encadeada *ordenada* com sentinela
Saída ptr para lista após modificação

```
 1 cur ← head
 2 v ← cur → next → val
 3 cur ← cur → next
 4 enquanto cur → next ≠ ⊥ faça
 5     w ← cur → next → val
 6     se v = w então
 7         list_delete(cur)
 8     senão
 9         v ← w
10         cur ← cur → next
11     fim se
12 fim faça
13 devolva head
fim
```

a) Ilustre a execução do algoritmo sobre a lista com os elementos $(1,2,2,3,3,3,4,5)$. ▷ *0,5 pt*

b) Descreva em poucas palavras (máx. 02 linhas) *o que* o algoritmo faz. ▷ *0,5 pt*

c) Qual a complexidade assintótica do algoritmo no *pior caso* (em função do tamanho n da lista de entrada)? Justifique brevemente sua resposta. (máx. 05 linhas) ▷ *1,0 pt*

QUESTÃO 2

Considere o algoritmo a seguir.

Algoritmo *algo*
Entrada $V = (v_0, \ldots, v_{n-1})$ array de inteiros
Saída ???

```
1 devolva func(0, n − 1, ⊥)
fim
```

Função *func*
Entrada $V = (v_0, \ldots, v_{n-1})$ array de inteiros
l, r inteiros tais que $0 \leq l \leq r \leq n-1$
root ponteiro para um nó
Saída ???

```
 1 se l > r então
 2     devolva root
 3 senão
 4     p ← partition(V, l, r)
 5     node ← BST_insert(root, V[p])
 6     node ← func(V, l, p − 1, node)
 7     node ← func(V, p + 1, r, node)
 8     devolva node
 9 fim se
fim
```

A função *partition* é a mesma usada no Algoritmo Quicksort visto em aula, na qual o pivô escolhido é o primeiro elemento do trecho a ser particionado ($V[l]$). A função *BST_insert* é a função vista em aula usada para inserir elementos numa Árvore de Busca Binária.

a) Ilustre a execução do algoritmo para a entrada $V = (50, 40, 10, 60, 20, 30, 70)$ indicando claramente a saída. Descreva em poucas palavras (máx. 02 linhas) *o que* o algoritmo produz como saída em geral. ▷ *0,5 pt*

b) Forneça uma permutação do vetor $(1,2,3,4,5,6,7)$ que corresponde ao *melhor caso* do algoritmo, exibindo a sua saída nesse caso. ▷ *1,5 pt*

c) Considerando a saída da letra (a) como uma AVL, ilustre a inserção do valor 35, exibindo a(s) rotação(ões) eventualmente necessária(s) e a árvore final. ▷ *1,5 pt*

QUESTÃO 3

Seja T uma *hashtable* fechada de tamanho m que utiliza a função de dispersão $h(k) = k \mod m$ e uma política de resolução de colisões denominada *LCFS (Last Come First Served) linear probing*. Nessa estratégia, se um elemento x é mapeado numa posição $i = h(x)$ já ocupada por um elemento y, então x é colocado na posição i e y é deslocado para a posição $j = (i+1) \mod m$. Caso essa posição também já esteja ocupada por um elemento z, então y é colocado na posição j e z é deslocado para a próxima posição $k = (j+1) \mod m$, e assim sucessivamente, até que uma posição vaga seja encontrada.

Ilustre as inserções em T dos valores

$$14,\ 54,\ 75,\ 22,\ 89$$

nessa ordem, exibindo a tabela após cada inserção. Considere T de tamanho $m = 7$, icialmente vazia. Caso o fator de carga α seja superior ou igual a 0.5 *imediatamente antes* a uma inserção, essa inserção deve ser precedida por um *rehashing* para um novo tamanho $2m + 1$, sendo os elementos reinseridos pela ordem em que aparecem em T. ▷ *2,0 pt*

QUESTÃO 4

Considere um T.A.D. D que armazena uma coleção dinâmica de inteiros com duas operações:

- *insert*(D, x) - insere o valor x
- *remove_med*(D) - remove a *mediana* dos valores em D. A mediana dos valores de $D = (d_0, \ldots, d_{n-1})$ é o elemento que terminaria na posição $\lfloor n/2 \rfloor$ caso os valores fossem ordenados. Por exemplo, a mediana de $D = (4, 5, 2, 1, 3)$ é 3.

Esse T.A.D. pode ser implementado com duas *heaps*: uma *max_heap* L com os $\lfloor n/2 \rfloor$ menores elementos, e uma *min_heap* U com os restantes elementos. Assim, devemos ter L e U com mesmo tamanho (se n é par), ou U com um elemento a mais do que L (se n é ímpar). No exemplo acima, L conteria os valores $(2, 1)$ e U conteria $(4, 5, 3)$.

As operações são implementadas da seguinte forma:

- *insert*(D, x) - Se U está vazia, insere x em U. Caso contrário, seja m o menor elemento de U. Se $x < m$, insere o valor x em L. Caso contrário, insere x em U. Após essa inserção, pode ser que L tenha um elemento a mais do que U, ou que U tenha dois elementos a mais do que L. No primeiro caso, remove um elemento de L e o insere em U e, no segundo caso, remove de U e insere em L.
- *remove_med*(D) - remove e retorna o menor elemento de U. Após essa remoção, pode ser que L tenha um elemento a mais do que U. Nesse caso, remove um elemento de L e o insere em U.

Considere D inicialmente composta pelas heaps $L = \langle 70, 70, 50, 40, 50 \rangle$ e $U = \langle 75, 80, 90, 100, 85 \rangle$.

a) Ilustre a operação *insert*$(D, 65)$ exibindo as heaps na forma de árvore depois de cada inserção/extração. ▷ *1,0 pt*

b) Ilustre a operação *remove_med*(D) sobre o D resultante do item (a), exibindo as heaps na forma de árvore depois de cada extração/inserção. ▷ *1,0 pt*

c) Indique a complexidade assintótica do pior caso das duas operações, *insert* e *remove_med*, em função de n. ▷ *0,5 pt*