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

IF672ec Algoritmos e Estr. de Dados 2015.2
Prof. Paulo Fonseca

Primeira Prova — 13 de Outubro de 2015

- Esta prova tem 05 questões.
- A duração da prova é de 02h00min.

QUESTÃO 1 (2,0pt)

Uma lista *duplamente* encadeada é uma lista na qual cada nó possui um valor *val* e dois ponteiros para os nós anterior e posterior, respectivamente. O professor de Algoritmos pediu à classe que desenvolvessem um programa pra o seguinte problema: dada uma lista duplamente encadeada com n valores binários, determinar se a lista consiste de uma sequência de `0`'s seguida por uma sequência com a mesma quantidade de `1`'s. Para tornar a questão mais interessante, o professor estipulou que não poderiam ser usados contadores. Dois alunos então forneceram as seguintes respostas:

Algoritmo A: Se a lista começar por `0`, marque o primeiro nó (por ex., sobrescrevendo seu valor para 'X') e percorra a lista até encontrar o primeiro `1` correspondente; marque-o e volte até encontrar o primeiro `0` ainda não marcado; marque este `0` e percorra a lista até encontrar o próximo `1` ainda não marcado. Repita o procedimento percorrendo a lista em vai-e-vém e marcando sucessivamente cada `0` e o seu correspondente `1`. Se, ao percorrer a lista, o cursor encontrar um `0` depois de já ter passado por um `1` (mesmo marcado), retorne `FALSE`. Da mesma forma, se não houver um `1` não-marcado que corresponda a um `0` recém-marcado, retorne igualmente `FALSE`. Do contrário, o algoritmo pára assim que todos os `0`'s em sequência no início da lista e seus correspondentes `1`'s tenham sido marcados. Se, então, todos os nós estiverem marcados, retorne `TRUE`; caso ainda restem nós não marcados no final da lista, retorne `FALSE`. (vide ilustração a seguir)

Algoritmo B: Percorra a lista do início ao fim marcando alternadamente um `0` sim e outro não e, depois, um `1` sim e outro não; se, ao percorrer a lista, o cursor passar por um `0` depois de já ter passado por um `1`, retorne `FALSE`; senão volte ao início da lista e percorra-a novamente, marcando alternadamente os `0`'s *não-marcados* e, em seguida, os `1`'s *não-marcados*; repita o processo, parando assim que todos os `0`'s estejam marcados *ou* (não-exclusivamente) todos os `1`'s estejam marcados. Se todos os `0`'s *e* `1`'s estão marcados, retorne `TRUE`, do contrário, retorne `FALSE`. (vide ilustração a seguir)

Algoritmo A:

```
0000000000000000111111111111111
X000000000000000111111111111111
---------------->
X000000000000000X111111111111111
 <----------------
XX00000000000000X111111111111111
 ---------------->
XX00000000000000XX11111111111111
  <----------------
              ...
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX1
                ---------------->
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
====> OK!
```

Algoritmo B:

```
0000000000000000111111111111111
X0X0X0X0X0X0X0X0X1X1X1X1X1X1X1
------------------------------->
<-------------------------------
XXX0XXX0XXX0XXX0XXX1XXX1XXX1XXX1
------------------------------->
<-------------------------------
               ...

XXXXXXXXXXXXXXX0XXXXXXXXXXXXXXX1
------------------------------->
<-------------------------------
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
====> OK!
```

Responda, justificando adequadamente:

a) Qual o custo assintótico do Algoritmo A no pior caso?

b) Qual o custo assintótico do Algoritmo B no pior caso?

QUESTÃO 2 (2,0pt)

Considere as operações de inserção e remoção de uma chave k, realizadas sobre uma BST, e representadas respectivamente por $+k$ e $-k$.

a) Represente a árvore resultante da sequência de operações

$$+70, +30, +100, +50, +90, +20, +80, +60, +30, +40, -100, -30$$

sobre uma BST inicialmente vazia.

b) Tratando a árvore da resposta do item (a) como uma AVL, represente a inserção $+45$, ilustrando as rotações que se fizerem necessárias.

QUESTÃO 3 (2,0pt)

Considere uma tabela de dispersão (*hash table*) de capacidade $m = 7$ e com a função de dispersão $h(k) = k \mod m$. Represente a tabela após a inserção das chaves

$$19, 26, 13, 48, 17,$$

nesta ordem, em cada um dos cenários a seguir.

a) Quando as colisões são tratadas por encadeamento (hashing aberto).

b) Quando as colisões são tratadas por sondagem linear.

c) Quando as colisões são tratadas por hashing duplo com função secundária $h'(k) = 5 - (k \mod 5)$

Obs.: Em nenhum dos casos acima, deve ser feito *rehashing*.

■ **QUESTÃO 4** (2,0pt)
Considere a estrutura *max-heap* binária $H = (10, 9, 8, 7, 6, 5, 4, 3, 2, 1)$ aumentada por uma estrutura auxiliar P_H e uma operação $pos(P_H, k)$ que retorna a posição da chave k na heap H em tempo constante (ex. $pos(P_H, 7) = 4$). Nesse caso, podemos implementar uma operação $update(H, k, k')$ que atualiza a chave k para k'. Essa operação primeiro localiza a posição i da chave k (com o auxílio de P_H), depois altera o valor dessa posição para k', e depois faz uso de uma das funções $bubble\_up(H, i)$ ou $max\_heapify(H, i)$ (similares às vistas em aula, porém a partir da posição i) para reposicionar a chave k' adequadamente. Ilustre a execução de cada uma das operações a seguir sobre a *max-heap* H original.

a) $update(H, 7, 13)$

b) $update(H, 9, 0)$

Responda também:

c) Qual o custo assintótico de m updates sobre uma heap com n elementos no pior caso? Justifique adequadamente (máx. 4 linhas — não precisa ser uma prova formal).

■ **QUESTÃO 5** (2,0pt)
Considere a representação por florestas da estrutura *union-find* vista em aula com a huerística da união ponderada. Suponha que tenhamos uma estrutura representando incialmente conjuntos unitários contendo os inteiros de 1 a 8, cada um na sua própria classe de equivalência. Ilustre, em cada um dos casos a seguir, uma sequência *mínima* de uniões da forma $Union(a, b)$, sobre a estrutura incial, onde a e b são números no intervalo $1, \ldots, 8$. Desenhe a árvore correspondente.

a) A estrutura resultante consiste em apenas uma árvore com a *maior* altura possível.

b) A estrutura resultante consiste em apenas uma árvore com a *menor* altura possível.

■ ■ ■