Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e de Computadores

Algoritmos e Estruturas de Dados

Semestre de Verão 2007/08

Teste final, época especial (2h30m)

Observações: No contexto deste teste, o triplo `(v,l,r)` representa o *sub-array* do *array* `v`, compreendido entre os índices `l` e `r` inclusivé.

I (6 v)

1. (2) Comente a afirmação

   A inserção num amontoado binário tem custo $O(\log(n))$, contudo a remoção tem custo $O(n)$.

2. (2) Considere o seguinte método contendo uma implementação recursiva do algoritmo de ordenação *Insertion Sort*.

```
public static void sortR(int[] v, int l, int r){
   if(r<l) return;
   sortR(v,l,r-1);
   insertSortedR(v, l, r-1, v[r]);
}
```

   O método `sortR(int[] v, int l, int r)` ordena o sub-*array* `(v,l,r)`.

   O método `insertSortedR(int[] v, int l, int r, int a)` insere `a` no sub-*array* `(v,l,r)`.

   2.1. Represente a recorrência que define o número de vezes que `insertSortedR` é chamado em função da dimensão do sub-*array* a ordenar.

   2.2. Resolva a recorrência anterior.

3. (2) Considere o amontoado binário definido pelo *array* $[10; 5; 9; 3; 4; 1; 8; 2; 1]$. Apresente a sequência de alterações realizadas sobre o amontoado durante a operação de inserção do inteiro 7.

II (14 v)

Nota: a resolução das questões deste grupo pode utilizar métodos ou classes auxiliares. Contudo, o seu código tem de ser completamente apresentado.

1. (3,5) Realize o método estático

   ```
   public static int countLowerOrEqual(int[] v, int l, int r, int a)
   ```

   que retorna o número de elementos do sub-*array* `(v,l,r)` menores ou iguais que `a`. O custo assimptótico do algoritmo deve ser $O(\log(n))$, onde n é a dimensão do sub-*array*.

2. (3,5) Realize o método estático

   ```
   public static <E> Node<E> rotateRight(Node<E> head, int count)
   ```

   que *roda* a lista simplesmente ligada sem sentinela e *não circular*, referida por `head`, de `count` nós para a direita. Rodar `count` nós para a direita significa que os últimos `count` nós são colocados no início da lista, mantendo a ordem relativa. Por exemplo, a rotação de 3 elementos da lista $[3; 4; 5; 6; 7; 8; 9; 0; 1; 2]$ resulta na lista $[0; 1; 2; 3; 4; 5; 6; 7; 8; 9]$.

   Os nós da lista são representados pela classe `Node<E>`, que possui dois campos públicos: `value`, com o elemento presente no nó; e `next` com a referência para o próximo nó (quando existe). Assuma que `count` é menor do que a dimensão da lista.

   O método retorna a referência para o primeiro nó da lista resultante.

3. (3,5) Realize o seguinte método de instância da classe `HashTable<E>`, que representa uma tabela de dispersão com encadeamento externo.

   ```
   boolean hasDuplicates()
   ```

   Este método retorna `true` se e só se a tabela de dispersão tiver elementos duplicados. Assuma que a classe `HashTable<E>` possui os seguintes campos:

   - `Node<E>[] table` - `array` com a tabela de dispersão.

4. (3,5) Realize o método estático

   ```
   public static <E> Node<E> reverse(Node<E> t)}
   ```

   que *inverte* a árvore binária de pesquisa referenciada por `t`. Duas árvores não vazias `t1` e `t2` dizem-se *inversas* se

   - Os valores presentes nas raizes forem iguais.
   - A sub-árvore esquerda de `t1` for inversa da sub-árvore direita de `t2`.
   - A sub-árvore direita de `t1` for inversa da sub-árvore esquerda de `t2`.

   Duas árvores vazias consideram-se inversas uma da outra.

   Assuma que cada nó da árvore tem três campos: um `E val`, e duas referências, `left` e `right`, para os descendentes respectivos.

ISEL, 13 de Outubro de 2008.