Third written examination of
Algoritmos e Estruturas de Dados

November 9, 2015 Duration: no more than 30 minutes

Name:

Student number:

4.0 **1:** **Explain how a binary search is performed.**

Answer:

A pesquisa binária é um algoritmo de pesquisa para um *array* ordenado que divide repetidamente o intervalo de pesquisa ao meio. O intervalo de pesquisa inicial é, como é claro, o *array* inteiro. Se o valor que se pretende encontrar no *array* for menor que o item no meio do intervalo de pesquisa, restringe-se o intervalo de pesquisa à sua metade inferior. Caso contrário, restringe-se o intervalo de pesquisa à sua metade superior. Este procedimento é repetido até se encontrar o valor pretendido ou o intervalo de pesquisa ser vazio.

3.0 **2:** **The binary search algorithm is normaly used to find an exact match. Explain how the algorithm can be modified so that if returns the index of an exact match, or, if no match exists, so that it returns the index of the largest array element smaller than the value to be matched (or -1 if the value to be matched is smaller than the first array element).**

Answer:

```
int binary_search(T *data,int data_size,T value)
{
        int i_low = 0;
        int i_high = data_size - 1;
        int i_middle;
        if (value < data[i_low] ) { return -1; }
        while(i_low <= i_high)
        {
                i_middle = (i_low + i_high) / 2;
                if(value == data[i_middle])
                        return i_middle;
                if(value > data[i_middle])
                        if (i_low == i_high) { return i_middle; }
                        i_low = i_middle + 1;
                else
                        i_high = i_middle - 1;
        }
}
```

Antes do ciclo *while* que efetua as restriçóes sucessivas do intervalo de pesquisa, testamos se o valor que queremos encontrar é menor que *data[i_low]* (porque, neste momento, *i_low* é 0, apontando para o primeiro elemento do *array*). Se isto se verificar, retornamos -1.

Já dentro do ciclo *while*, sempre que o valor que procuramos é maior que *data[i_middle]*, verificamos se *data[i_middle]* é o último elemento *do array* com o qual poderíamos possivelmente comparar o valor procurado. Isto acontece quando *i_low == i_high*, ou seja, quando o intervalo de pesquisa já só tem um elemento. Se esse for o caso, e sabendo que os valores comparados não são iguais (porque *data[i_middle]* é maior), então retornamos *i_middle*, que se trata do índice do maior elemento do array que é menor que o valor procurado.

3.0 3: How is the information organized in a min-heap?

Answer:

Numa *min-heap* (árvore binária completa), o valor de cada nó interno é menor ou igual ao valor de qualquer nó que descenda de si próprio.

3.0 4: What operations are supported in a priority queue?

Answer:

Uma *priority queue* suporta as seguintes operações:
- *insert(item, priority)*: insere um item na *queue* com uma determinada prioridade
- *getHighestPriority( )*: retorna o item de maior prioridade na *queue*
- *deteleHighestPriority( )*: remove o item de maior prioridade da *queue*

Nota
Uma *priority queue* é uma extensão da *queue* e tem as seguintes propriedades:
1. cada elemento tem uma prioridade associada a si próprio
2. um elemento com elevada prioridade é retirado da *queue* antes de um elemento com baixa prioridade
3. se dois elementos tiverem a mesma prioridade, o que é retirado primeiro é o que foi colocado há mais tempo (como numa *queue* normal)

7.0 5: Explain what a hash table is and explain the main differences between its two variants (open addressing and chaining).

Answer:

Uma *hash table* é uma estrutura de dados que implementa um *array* associativo (estrutura de dados abstrata que mapeia chaves para valores). A *hash table* usa uma *hash function* para calcular um índice, também denominado *hash code*, correspondente a um espaço no *array*. Durante uma pesquisa, a chave é passada como argumento à *hash function* e o retorno da mesma indica onde, no *array* de valores, o valor pretendido está guardado.
As *hash tables* têm dois métodos diferentes para lidar com colisões:
- *open addressing*: todos os elementos devem ser armazenados na *hash table* em si; o tamanho da *hash table* deve sempre ser maior ou igual ao número total de chaves
- *separate chaining*: esta técnica transforma cada célula da *hash table* numa lista ligada que armazena os registos que têm o mesmo *hash code*