Segundo teste de **Algoritmos e Estruturas de Dados**

12 de Novembro de 2018 **Duração máxima: 1 hora**

Justifique todas as suas respostas.

3.0 **1:** Explique como pode usar uma lista simplesmente ligada para implementar uma pilha. Que vantagens e desvantagens tem uma implementação deste tipo quando comparada com uma outra que usa um *array*?

3.0 **2:** Um *array* circular pode ser utilizado para implementar uma fila. Explique como. Que vantagens e desvantagens tem uma implementação deste tipo quando comparada com uma outra que usa uma lista ligada?

3.0 **3:** Explique como está organizado um *max-heap* e como se insere informação nesta estrutura de dados. (Para explicar a inserção, pode usar como exemplo inserir os números 1, 2, 3, 4 e 5, num *max-heap* inicialmente vazio.)

2.0 **4:** Explique como pode procurar informação numa lista biligada não ordenada, e indique qual a complexidade computacional do algoritmo que descreveu. O que é que pode fazer para tornar a procura mais eficiente quando alguns itens de informação são mais procurados que outros?

4.0 **5:** Numa aplicação que pretende contar o número de ocurrências de palavras num texto extenso usando uma *hash table* (tabela de dispersão, dicionário) um aluno, com pouca experiência nestas coisas, usou a seguinte *hash function*:

```
#define hash_table_size 1000003u

unsigned int hash_function(unsigned char *s)
{
  unsigned int sum;
  for(sum = 0;*s != '\0';s++)
    sum += (unsigned int)(*s);
  return sum % hash_table_size;
}
```

Sabe-se que o número de palavras distintas que existem no texto é inferior mas próximo de um milhão, e que o número de letras médio de uma palavra é inferior a 10.

2.0 **a)** Explique porque é que neste caso deve usar uma implementação da *hash table* que usa *separate chaining*, em vez de usar uma que usa *open addressing*.

2.0 **b)** A *hash function* apresentada acima é muito má. Porquê? Sugira uma outra que seja bem melhor.

Nas duas perguntas seguintes sobre árvores binárias, cada nó da árvore usa a seguinte estrutura de dados:

```
typedef struct tree_node
{
  struct tree_node *left;
  struct tree_node *right;
  long data;
}
tree_node;
```

2.0 **6:** A seguinte função é uma implementação **errada** de uma função recursiva que procura informação numa árvore binária **não ordenada**.

```
tree_node *tree_search(tree_node *n,long data)
{
  if(n == NULL || n->data == data)
    return NULL;
  return tree_search((data < n->data) ? n->left : n->right,data);
}
```

Explique o que está errado na função e corrija-a.

3.0 **7:** Escreva uma função que indique se uma árvore binária está balanceada ou não. Recorda-se que numa árvore binária balanceada as alturas das sub-árvores dos lados direito e esquerdo não podem diferir em mais de 1.

```
int is_balanced(tree_node *n)
{
  // put your code here
}
```

Considere que se o valor devolvido pela função for -1 então a árvore não está balanceada.