IPS Instituto Politécnico de Setúbal
Escola Superior de Tecnologia de Setúbal

ATAD 2019/20

Algoritmos e Tipos Abstratos de Dados

Enunciado de Laboratório
AVALIADO

2

Algoritmos, Pseudo-código e Complexidade Algorítmica

Objetivos

- Interpretar um algoritmo expresso em pseudo-código.
- Traduzir um algoritmo em pseudo-código para a linguagem C e vice-versa.
- Identificar a complexidade algorítmica.
- Strings e tipos compostos.
- Documentação Doxygen.

Nível 1

1. Crie um novo projeto VS Code (uma pasta no sistema de ficheiros). Todo o pseudo-código solicitado deverá ser inserido no ficheiro `pseudocode.txt`.
2. Crie um algoritmo `strlength` em pseudo-código que receba uma array de carateres (string) e calcule o seu tamanho, i.e., até encontrar um caractere `\0`.
3. Identifique a complexidade algorítmica deste algoritmo na notação *Big-O* e justifique-a. Insira a resposta no ficheiro `pseudocode.txt`.

Nível 2

1. Crie um algoritmo `swapStrings` em pseudo-código que recebe dois arrays de carateres (strings) e um número natural `by` e permuta caracteres entre as duas strings:
   - Se o comprimento das duas strings for diferente, retorna `false`;
   - Se `by` não for válido para o tamanho das strings, retorna `false`;
   - Caso contrário, permuta os caracteres de ambas as strings no intervalo dado por `by` e retorna `true`.
   - Deverá fazer uso da invocação do algoritmo `strlength`.

Exemplo entrada:

```
s1 = "abcdefghij"
s2 = "klmnopqrst"
by = 3
```

Estado final:

```
s1 = "abmdepghsj"
s2 = "klcnofqrit"
```

No limite, se `by = 1` permuta todos os caracteres.

2. Identifique a complexidade algorítmica deste algoritmo na notação *Big-O* e justifique-a. Insira a resposta no ficheiro `pseudocode.txt`.

Nível 3

1. Traduza o código anterior para a *linguagem C* no ficheiro `main.c`.

2. Crie um *makefile* com as diretivas `release` (default), `debug` e `clear`; os nomes deverão ser auto-explicativos.

3. Codifique a função `main()` por forma a testar a função `swapStrings`. Pode utilizar os exemplos fornecidos, mas teste com vários valores para `by`, incluindo inválidos.

   Exemplos:

   ```
   by = 3 | s1 = "abmdepghsj" | s2 = "klcnofqrit"
   by = 1 | s1 = "klmnopqrst" | s2 = "abcdefghij"
   by = 10 | s1 = "abcdefghit" | s2 = "klmnopqrsj"
   by = -5 | s1 = "abcdefghij" | s2 = "klmnopqrst"
   ```

4. Forneça a documentação *doxygen* para as funções desenvolvidas e gere a documentação.
   - Utilize o *doxyfile* disponível no Moodle;
   - Verifique a documentação gerada na pasta `html`.

Nível 4

1. Crie o módulo `vector` nos ficheiros apropriados.
   - Defina a estrutura `Vector` contendo as projeções escalares (inteiras) `x` e `y`.

2. **Defina**, **implemente** e **comente** as funções:
   - `Vector vectorCreate(int x, int y)` - cria e inicializa uma nova instância de *Vector*.
   - `void vectorPrint(Vector v)` - imprime na consola a instância `v`.
   - `double vectorLength(Vector v)` - calcula a magnitude de `v`;
   - `int vectorDotProduct(Vector v1, Vector v2)` - calcula o *produto escalar* entre `v1` e `v2`.

3. Crie um ficheiro `main_vector.c` contendo uma função `main()` e teste as funções desenvolvidas no módulo `vector`.
   - No *makefile* adicione uma diretiva `vectors` para compilar este ficheiro e o módulo `vector`; utilize a *flag* `-lm` para poder compilar a biblioteca `<math.h>` se a utilizou.

Exemplos de resultados:

```
a = <4, 6>
b = <-3, 2>
c = <1, 3>
|a| = 7.211103
a . b = 0
a . c = 22
```

Nível 5

1. No módulo `vector` crie e **comente** a função `bool existOrthogonals(Vector list[], int listSize)` que verifica se existe algum par de vectores *ortogonais*; **dois vectores são ortogonais se o seu produto escalar for *zero* (0)**.

2. Teste na função `main()` anterior, dando exemplos para ambos os resultados possíveis.

3. Qual a complexidade algorítmica da função `existOrthogonals`? Insira as respostas em comentário ao código da função (interno, não doxygen).

   - Sem ter em conta a complexidade de `vectorDotProduct`;
   - Levando em conta a complexidade de `vectorDotProduct`;

Exemplos de resultados:

```
vList1 = [ <4, 6>, <-3, 2>, <1, 3> ]
vList1 exist orthogonals? true
vList2 = [ <4, 6>, <1, 3>, <0, 2>, <-1, 5> ]
vList2 exist orthogonals? false
```

(fim de enunciado)