Concurso de Arqueologia

# Algoritmos e Estruturas de Dados

P7

62413 Diogo Jorge Cachão Soares Ventura

62695 Francisco Luís Ribeiro

# 1 – Introdução

Neste trabalho realizámos todos os comandos pedidos: riqueza, terreno, estrela, escavação, reforço, equipa e sair. Todos os comandos funcionam como é pedido.

Fizemos a fase 1 a 100% e a fase 2 a 100%. Na fase 2 tínhamos pensado usar uma Tabela de Dispersão Aberta, mas enquanto fazíamos a classificação deparámo-nos com alguns problemas a ordenar o vetor com as equipas, o que fez com que demorássemos mais e não tivemos tempo para fazer a Tabela. O algoritmo que usámos para organizar o nosso vetor foi o Quick Sort, com vetor do iterador como parâmetro na função.

Entre a fase 1 e fase 2 adicionámos o reforço, a classificação, comentámos todas as funções, diminuímos a complexidade e aperfeiçoamos a qualidade do código.

Deixámos por fazer a Tabela de Dispersão Aberta, que iria tornar o código mais eficiente e organizado.

## 2 - TADs do Problema

* talhao.h
* terreno.h
* participante.h
* equipa.h
* jogo.h

## 2.1 – talhao.h

**Objetivo**: O talhão é uma estrutura que serve simplesmente para guardar o tesouro e o número de vezes que esse tesouro foi escavado.

struct \_talhao {

int num\_escavados;

int valor\_tesouro;

};

**int num\_escavados;** Este inteiro serve para guardar o número de vezes que um talhão foi escavado.

**int valor\_tesouro**; Este inteiro serve para guardar o valor do tesouro de cada talhão, que é lido no ficheiro.

## 2.2 – terreno.h

**Objetivo**: O terreno é uma estrutura que guarda os vários talhões numa matriz do tamanho dado pelo utilizador e que guarda também várias informações como o tesouro total.

struct \_terreno {

talhao \*\*matriz\_terreno;

int tesouro\_total;

int tesouro\_atual;

int linhas;

int colunas;

};

**talhao \*\*matriz\_terreno;** Esta variável é uma matriz dinâmica de talhões.

**int tesouro\_total**; Este inteiro serve para guardar o valor do tesouro total, ou seja, a soma de todos os tesouros de cada talhão.

**int tesouro\_atual**; Este inteiro vai sendo atualizada ao longo do programa, sempre que ocorre uma escavação retira-se o valor do tesouro ao tesouro atual.

**int linhas**; Este inteiro guarda o número de linhas da matriz de talhões.

**int colunas**; Este inteiro guarda o número de colunas da matriz de talhões.

## 2.3 – participante.h

**Objetivo**: O participante guarda toda a informação de cada arqueólogo do jogo, o seu nome, número de pontos, número de penalizações, etc.

struct \_participante {

char nome[40];

int pontos;

int penalizacoes;

int posição\_x;

int posição\_y;

int descf;

int numEscavacoes;

int numPenalizacoes;

};

**char nome[40];** Esta string de caracteres guarda o nome do arqueólogo.

**int pontos**; Este inteiro serve para guardar o valor de tesouros que o participante já escavou, ou seja, os seus pontos positivos.

**int penalizacoes**; Este inteiro serve para guardar o valor de penalizações, ou seja, nas vezes que um participante escavou um talhão, mas não havia tesouro, ele sofre uma penalização.

**int posição\_x**; Este inteiro guarda a linha onde o arqueólogo se situa, que é alterado quando ocorre uma escavação.

**int posição\_y**; Este inteiro guarda a coluna onde o arqueólogo se situa, que é alterado quando ocorre uma escavação.

**int descf**; Este inteiro indica se este arqueólogo está desclassificado ou não, fica a 1 quando este salta para uma posição fora do terreno.

**int numEscavacoes**; Este inteiro indica o número de vezes que o arqueólogo fez escavações, esta informação é importante, por exemplo, para comparar arqueólogos.

**int numPenalizacoes**; Este inteiro o número de vezes que o arqueólogo foi penalizado, esta informação também é importante para comparar arqueólogos.

## 2.4 – equipa.h

**Objetivo**: A equipa guarda uma sequência de participantes (arqueólogos) e informação da equipa, por exemplo, nome da equipa, número de membros, etc.

struct \_equipa {

sequencia participante;

char nome[40];

int pessoa\_escavar;

int num\_membros;

int equipa\_descf;

};

**sequencia partcipante**; Esta sequência guarda os apontadores dos participantes na equipa.

**char nome[40];** Esta string de caracteres guarda o nome da equipa.

**int pessoa\_escavar;** Este inteiro vai incrementando sempre que há uma escavação e guarda a posição do próximo arqueólogo da equipa que vai escavar.

**int num\_membros;** Este inteiro guarda o número de membros da equipa.

**int equipa\_descf;** Este inteiro indica se a equipa está desclassificada ou não. É ativado quando todos os participantes da equipa estiverem desclassificados.

## 2.4 – jogo.h

**Objetivo**: O jogo é composto por um vetor de apontadores para as estruturas equipas, que contem todas as equipas do ficheiro. Contém também um dicionário com apontadores para as equipas que estão a participar no jogo e o número total de equipas do ficheiro.

struct \_jogo {

equipa \*equipas;

dicionario equipas\_jogo;

int numTotalEquipas;

};

**equipa \*equipas**; Este vetor de apontadores para as equipas guarda todas as equipas que são lidas no ficheiro.

**dicionario equipas\_jogo;** Este dicionário guarda as equipas que estão em jogo. Esta separação por dicionário e vetor ajuda-nos a fazer várias comparações importantes entre equipas.

**int numTotalEquipas;** Este inteiro guarda o número total de equipas no ficheiro.

# 3 - Complexidade Final

## 3.1 - Riqueza

A função só irá dar return do valor da riqueza atual do terreno, fazendo com que não exista nenhuma complexidade nenhuma.

Começar cada subsecção com uma frase descrevendo brevemente o algoritmo implementado (2 a 3 linhas).

**Complexidade Caso Médio:**

TAD Terreno - daRiquezaAtual = O(1)

main - Riqueza = O(1)

**Final**

O(1) => O(1), complexidade constante.

## 3.2 - TERRENO

Esta função irá verificar cada talhão e o seu valor, se este já tiver sido escavado, fará printf de ‘-’, senão ‘\*’. Usa-se dois ciclos for’s para verificar cada talhão.

**Complexidade Caso Médio:**

TAD talhao - TesouroTalhao = O(1)

TAD terreno – escreverMatrizComSinais = O(1)

Main – terreno = O(c\*l) + O(1) = O(c\*l), sendo c as colunas do terreno e l as linhas do terreno

**Final**

O(c\*l) => O(c\*l), complexidade quadrada

## 3.3 – ESTRELA

A função começa por verificar os pré-requisitos para a efetuar. Após isso irá verificar a cada participante da equipa se foi desclassificado e se sim, não o incluir na estrela. Vai fazendo uma comparação dos participantes um a um, até encontrar a estrela.

TAD participante = (tamanho\_equipa\*2\*5+3)\*O(1) => O(1),

sendo tamanho\_equipa o número de participantes numa equipa

TAD equipa - estrelaEquipa = O(1) + O(2\*tamanho\_equipa) => O(tamanho\_equipa)

, sendo tamanho\_equipa o número de participantes presentes numa equipa.

main - estrela = O(numero\_equipas\*3) \* O(tamanho\_equipa) => O(numero\_equipas\*tamanho\_equipa),

Sendo numero equipas, o número de equipas presentes no jogo,

sendo tamanho\_equipa o número de participantes numa equipa

**Final**

O(numero\_equipas\*tamanho\_equipa)=> O(numero\_equipas\*tamanho\_equipa), complexidade quadrada.

## 3.4 – ESCAVAÇÃO

# Esta função vai procurar na equipa qual o participante a jogar, se a pessoa que for escavar estiver desclassificada incrementa-se para a próxima até encontrar uma com licença. Se não houver mais ninguém na equipa que tenha licença a equipa é desclassificada. Quando se encontra um participante, faz-se o salto, se este for para fora do terreno, esse participante é desclassificado e incrementa-se a posição da próxima pessoa a escavar. Após isso é só os cálculos para a pontuação do participante.

TAD participante = (tamanho\_equipa+2)\*O(1) => O(1)

TAD equipa = (3+tamanho\_equipa)\*O(1)+O(tamanho\_equipa)+O(numero\_equipas) => O(numero\_equipas)

TAD jogo – escavacaoEquipa = O(numero\_equipas)\*O(tamanho\_equipa) + 4\*O(1)=>O(numero\_equipas\*tamanho\_equipas)

Sendo numero\_equipas, o numero de equipas lidas no jogo e tamanho\_equipa, o numero de participantes que estão numa equipa.

**Final**

O(numero\_equipas\*tamanho\_equipa)=> O(numero\_equipas\*tamanho\_equipa), complexidade quadrada.

## 3.5 – REFORÇO

A função verifica os requisitos necessários para adicionar um participante a uma equipa e após isso irá adicionar à sequência de participantes na equipa indicada.

TAD participante – 7\*O(1) =>O(1)

TAD equipa – O(1) + 5\*O(tamanho\_equipa) =>O(tamanho\_equipa)

Main – reforço = O(3\*numero\_equipas) \* O(tamanho\_equipa) => O(tamanho\_equipa\*numero\_equipas)

**Final**

O(numero\_equipas\*tamanho\_equipa)=> O(numero\_equipas\*tamanho\_equipa), complexidade quadrada.

## 3.6 – EQUIPA

A função após fazer os requisitos necessários, guarda no dicionário do jogo.

TAD equipa = O(numero\_equipas) => O(numero\_equipas)

Main – equipa = O(numero\_equipas\*3) + O(1) =>O(numero\_equipas)

**Final**

O(numero\_equipas)=> O(numero\_equipas), complexidade linear.

## 3.7 – SAIR

A função irá acabar o jogo, dando ou não a classificação das equipas conforme a existência de tesouros no terreno ou se as equipas tiverem sido todas desclassificadas.

TAD equipa – O(numero\_equipas) + O(1)=>O(numero\_equipas)

TAD jogo – organizaEquipas = O(numero\_equipas) + O(1) + O(numero\_equipas\*log2(numero\_equipas)) => O(numero\_equipas\*log2(numero\_equipas))

main – sair = O(C\*L) + O(2\*numero\_equipas\*Tamanho\_equipas) + O(numero\_equipas\*2)+ 4\*O(1) => O(C\*L)

Onde C = colunas do terreno

L = linhas do terreno

numero\_equipas = numero de equipas dentro do jogo

tamanho\_equipas = numero de participantes numa equipa.

**Final**

O(C\*L) => O(C\*L), complexidade quadrática.

# 4 - Conclusão

Neste trabalho conseguimos realizar a fase 1 e a fase 2 a 100%, ultrapassámos várias dificuldades como, a forma de gerir a memória de todas as TADs, a função escavação e, principalmente, a função classificação, que requer o uso de algoritmos para a ordenação de vetores segundo várias comparações. Outra dificuldade foi a simplificação do código de forma a diminuir a complexidade e aumentar a qualidade, no final de já ter obtido os resultados necessários.

Tínhamos o objetivo de cumprir todos os requerimentos dados, mas não conseguimos realizar a parte da Tabela de Dispersão Aberta. Com isso, achamos que a nossa qualificação deveria ser 17 valores.