CONCURSO DE ARQUEOLOGIA

ALGORITMOS E ESTRUTURAS DE DADOS 21/22

Turno P6
62964 – Martim Duarte Agostinho
63355 – Tiago Daniel Santos Gouveia

Índice

1 - Introdução

1.1 - Fase 1:

Na fase 1, todos os comandos (nomeadamente Riqueza, Terreno, Estrela, Escavação, Equipa) foram realizados e testados com sucesso.

A princípio, para definir a posição do arqueólogo, criámos uma TAD Posição onde a estrutura tinha como elementos dois inteiros x e y, mas acabámos por mudar para um vetor de dimensão 2.

Entregámos a 1ª Fase com o TAD Talhão, que continha 2 unsigned ints, um para o tesouro e outro para a quantidade de escavações.

1.2 - Fase 2:

Na fase 2, todos os comandos (nomeadamente Riqueza, Terreno, Estrela, Escavação, Reforço, Equipa, Sair) foram realizados e testados com sucesso.

Removemos o a TAD talhao e substituímos por inteiros onde, caso o inteiro seja positivo é o tesouro do talhão, caso contrário é as vezes que foi escavado.

O dicionário vetorial foi substituído pelo dicionário tabela de dispersão aberta.

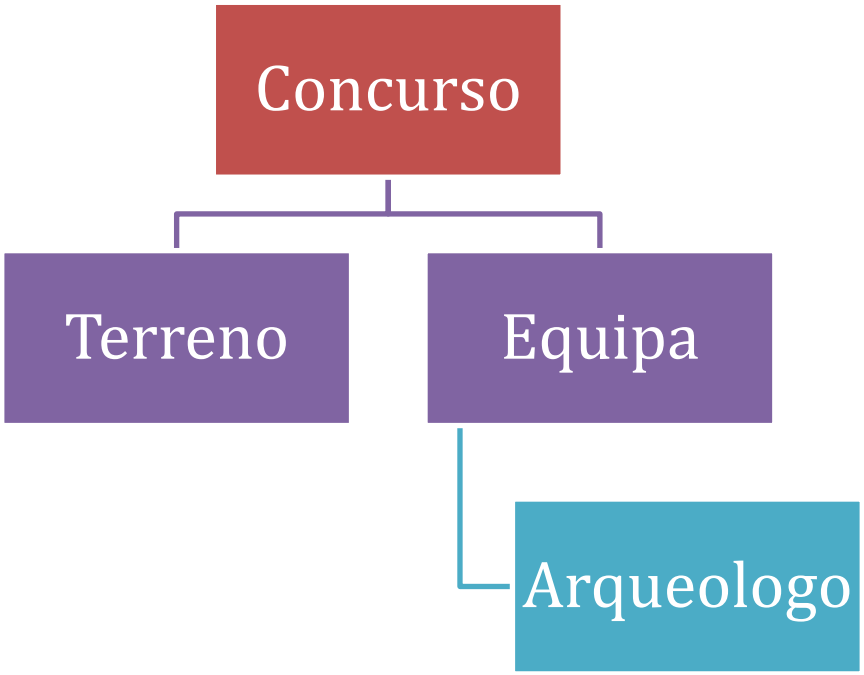
Implementámos a função primo da maneira mais eficaz que encontrámos, procuramos o número num vetor previamente definido caso ele seja menor que o maior número do vetor (no nosso caso 73), caso seja maior que 73 e menor que 73^2, verificamos de 2 em 2, se ele é divisível por algum primo de 3 até $\sqrt{n}$,em último caso quando o número é maior que 73^2, verificamos se ele é divisível por algum número de 73^2 até $\sqrt{n}$.

Para organizar as equipas começámos por tentar implementar um quick sort mas acabámos por simplesmente implementar um bubble sort que compara uma equipa com a sua adjacente até que estejam todas em ordem.

Dentro desta organização, quando é pedido para as equipas serem colocadas por ordem alfabética, antes de fazer string compare da biblioteca string.h, efetuamos uma copia de char * e colocamos todos os caracteres para minúscula, isto foi criado para, por exemplo a comparar o nome Zélia com armindo, o string compare dar prioridade ao armindo e não o contrário. Esta mesma função criou um erro quando comparado aos valores do mooshak, ao comparar a equipa Team 15 à team 12, isto porque o mooshak dizia que quem teria prioridade seria a Team 15, e o nosso programa dizia que era a team 12.

2 - TADs do Problema

No seguinte diagrama podemos ver as TADs implementadas no nosso trabalho:



2.1 - TAD Concurso

Objetivo: Gerenciar a comunicação entre TADs assim como guardar as equipas não desclassificadas e o terreno do jogo.

```
struct _concurso{
    sequencia equipasFicheiro;
    dicionario equipas;
    terreno campo;
};
```

`sequencia equipasFicheiro;` Sequência onde se vão encontrar todas as equipas presentes no ficheiro teams.txt.
`dicionario arqsDescl;` Dicionário de todos os arqueólogos que já foram desclassificados desde o início do concurso.
`dicionario equipas;` Dicionário das equipas que estão em jogo.
`terreno campo;` TAD Terreno onde decorre o jogo.

2.2 - TAD Terreno

Objetivo: Conter um terreno onde existe uma matriz de inteiros, que representam os diferentes talhões, e três unsigned int, dois para as dimensões e outro para a riqueza do terreno.

```
struct _terreno{
    int ** terr;
    unsigned int dim_x;
    unsigned int dim_y;
    unsigned int riqueza;
};
```

`int ** terr;` Matriz de inteiros que representarão os talhões, caso estes sejam positivos, representarão o tesouro, caso contrário a quantidade de escavações.
`unsigned int dim_x;` Quantidade de colunas do terreno (Unsigned int porque nunca vai ser negativo).
`unsigned int dim_y;` Quantidade de linhas do terreno (Unsigned int porque nunca vai ser negativo).
`unsigned int riqueza;` Soma de todos os tesouros dos talhões (Unsigned int porque nunca vai ser negativo).

2.3 - TAD Equipa

Objetivo: Conter principalmente uma lista ligada de arqueologos (explicada em baixo).

```
struct _equipa{
    noSimples cabeca;
    noSimples cauda;
    arqueologo estrela;
    iteraeq arqueologocorrente;
    int meritoeq;
    char desqualificada;
    char * nome;
    dicionario arqsDescl;
};
```

`noSimples cabeca;` No Simples que aponta para o primeiro arqueólogo da lista ligada
`noSimples cauda;` No Simples que aponta para o último arqueólogo da lista ligada
`arqueologo estrela;` TAD Arqueólogo que é a estrela da equipa.
`iteraeq arqueologocorrente;` Iterador que corre a lista ligada
`int meritoeq;` Soma dos méritos de todos os arqueólogos da equipa.
`char desqualificada;` Char para definir se a equipa está ou não desqualificada (1 se desqualificada, 0 se qualificada).

`char * nome;` Nome da Equipa.
`dicionario arqsDescl;` Dicionário de todos os arqueólogos que já foram desclassificados desta equipa.

A equipa tem uma lista ligada onde a cauda aponta para a cabeça porque usamos os arqueólogos de forma sequencial e desta forma para encontrar o arqueólogo corrente não temos de percorrer nada e temos apenas de apontar para o próximo arqueólogo depois do usar.

2.4 - TAD Arqueologo

Objetivo: Conter informações sobre cada jogador tais como a pontuação, nome, numero de penalizações e posição.

```
struct _arqueologo{
    int merito;
    char* nomeArqueologo;
    int penalizacoesCount;
    int pos[NDIM];
};
```

`int merito;` Mérito do arqueólogo
`char* nomeArqueologo;` Nome do arqueólogo
`int penalizacoesCount;` Penalizações do arqueólogo
`int pos[NDIM];` Posição do arqueólogo

3 - Complexidade Final

3.1 - Riqueza

No comando riqueza, mostra-se a riqueza do terreno (soma dos tesouros de todos os talhões), que, no nosso caso, é um elemento da estrutura “Terreno”.

Complexidade Caso Médio:

riquezaFunc = O(1) => O(1)	(main)
mostrarRiqueza = O(1) => O(1)	(TAD Concurso)
get_riqueza = O(1) => O(1)	(TAD Terreno)

Final
O(1) => O(1)
Todas as funções serão chamadas uma só vez.

3.2 – Terreno

No comando terreno pretende-se mostrar um esboço do terreno onde ‘*’ representam talhões com tesouro e ‘-‘ representam talhões sem tesouro

L – Número de linhas do terreno
C – Número de colunas do terreno

terrenoFunc = O(L * C) => O(L * C)	(main)
mostrar_talhao = O(L * C + 1) => O(L * C)	(TAD Concurso)
gett_dimx = O(1) => O(1)	(TAD Terreno)
gett_dimy = O(1) => O(1)	(TAD Terreno)
gett_val = O(1) => O(1)	(TAD Terreno)

Final
O(L * C) => O(L * C)
Complexidade proporcional à área do terreno.

3.3 – Estrela

N – Número de Equipas numa colisão no dicionário

estrelaFunc = O(N) => O(N)	(main)
existe_equipa = O(N) => O(N)	(TAD Concurso)
existeElemDicionario = O(1) => O(1)	(TAD Dicionario)
elementoDicionario = O(N) => O(N)	(TAD Dicionario)
dispersao = O(1) => O(1)	(TAD Chaves)
elemNoSimples = O(1) => O(1)	(TAD NoSimples)
priTuplo = O(1) => O(1)	(TAD Tuplo)
igualChaves = O(1) => O(1)	(TAD Chaves)
segTuplo = O(1) => O(1)	(TAD Tuplo)
segNoSimples = O(1) => O(1)	(TAD NoSimples)
equipa_desqualificada = O(1) => O(1)	(TAD Terreno)

Final
O(N) => O(N)
A complexidade será proporcional ao número de equipas numa colisão no dicionário

3.4 – Escavacao

O comando escavação pretende atribuir o tesouro do talhão selecionado ao arqueólogo e consequentemente à equipa.
N – Número de Equipas numa colisão no dicionário
A – Número de arqueólogos presentes na equipa

executar_escavacao = O(2N + 3A) => O(N + A)	(TAD Concurso)
existeElemDicionario = O(N) => O(N)	(TAD Dicionario)
elementoDicionario = O(N) => O(N)	(TAD Dicionario)
equipa_desqualificada = O(1) => O(1)	(TAD Equipa)
prox_arq = O(1) => O(1)	(TAD Equipa)
getit_arq = O(1) => O(1)	(TAD Equipa)
getPosicaoArqueologo = O(1) => O(1)	(TAD Arqueologo)
gett_dimx = O(1) => O(1)	(TAD Terreno)
gett_dimy = O(1) => O(1)	(TAD Terreno)
desclassificaArqueologo = O(1) => O(1)	(TAD Arqueologo)
getMeritoArqueologo = O(1) => O(1)	(TAD Arqueologo)
add_meritoeq = O(1) => O(1)	(TAD Equipa)
getNomeArqueologo = O(1) => O(1)	(TAD Arqueologo)
rm_elemeq = O(A) => O(A)	(TAD Equipa)
add_desq_eq = O(1) => O(1)	(TAD Equipa)
load_estrela = O(A) => O(A)	(TAD Equipa)
equipa_desqualificada = O(1) => O(1)	(TAD Equipa)

moverArqueologo = O(1) => O(1) (TAD Arqueologo)
escavar = O(1) => O(1) (TAD Terreno)
incrPenalizacoesArqueologo = O(1) => O(1) (TAD Arqueologo)
addMeritoArqueologo = O(1) => O(1) (TAD Arqueologo)
add_meritoeq = O(1) => O(1) (TAD Equipa)
get_equipa_estrela = O(1) => O(1) (TAD Equipa)
comparar_estrela = O(1) => O(1) (TAD Arqueologo)
muda_estrela = O(1) => O(1) (TAD Equipa)
load_estrela = O(A) => O(A) (TAD Equipa)

Final
O(N + A) => O(N + A)
A complexidade será proporcional à soma do número de equipas numa colisão no dicionário com o número de arqueólogos presentes na maior equipa.

3.5 – Reforco

O comando reforço adiciona um arqueólogo novo à equipa
N – Número de Equipas numa colisão no dicionário
A – Número de arqueólogos presentes na equipa

reforcoFunc = O(N + A) => O(N + A) (main)
comandoReforcoConcurso = O(2N + A) => O(N + A) (TAD Concurso)
existeElemDicionario = O(N) => O(N) (TAD Dicionario)
elementoDicionario = O(N) => O(N) (TAD Dicionario)
existe_arqueologo = O(A) => O(A) (TAD Equipa)
arq_desq_eq = O(1) => O(1) (TAD Equipa)
criaArqueologo = O(1) => O(1) (TAD Arqueologo
add_elemeq = O(1) => O(1) (TAD Equipa)
get_equipa_estrela = O(1) => O(1) (TAD Equipa)
comparar_estrela = O(1) => O(1) (TAD Arqueologo)
muda_estrela = O(1) => O(1) (TAD Equipa)

Final
O(N + A) => O(N + A)
A complexidade será proporcional à soma do número de equipas numa colisão no dicionário com o número de arqueólogos presentes na maior equipa.

3.6 - Equipa

O comando equipa pretende adicionar uma equipa ao jogo.
N – Número de Equipas numa colisão no dicionário
Complexidade Caso Médio:

equipaFunc = O(N) => O(N)	(main)
comandoEquipa = O(N) => O(N)	(TAD Concurso)
tamanhoSequencia = O(1) => O(1)	(TAD Sequencia)
elementoPosSequencia = O(1) => O(1)	(TAD Sequencia)
get_equipa_nome = O(1) => O(1)	(TAD Equipa)
existeElemDicionario = O(N) => O(N)	(TAD Dicionario)
add_equipa = O(1) => O(1)	(TAD Concurso)
elementoPosSequencia = O(1) => O(1)	(TAD Sequencia)
get_equipa_nome = O(1) => O(1)	(TAD Equipa)
adicionaElemDicionario = O(1) => O(1)	(TAD Dicionario)

Final
O(N) => O(N)
A complexidade será proporcional ao número de equipas numa colisão no dicionário

3.7 – Sair

A função sair pretende destruir todos os mallocs ainda não destruídos e mostrar as equipas ainda em jogo por uma certa ordem.

N – Número de Equipas no dicionário.
A – Número de arqueólogos presentes na equipa

Complexidade Caso Médio:

presair = O(N) => O(N)	(TAD Concurso)
classificacao = O(N * N + N + A) => O(N * N + A)	(main)
sort_equipas = O(N * N) =>O(N * N)	(TAD Terreno)
get_nequipas = O(1) => O(1)	(TAD Concurso)
returnEquipa = O(A) => O(A)	(TAD Concurso)

Final
O(N * N + 2N + A) => O(N * N + A)
A Complexidade será proporcional à soma do número de equipas no dicionário com o número de arqueólogos presentes na maior equipa.

3.8 – Programa

Como a maior complexidade obtida é O(N * N + A) chegamos à conclusão que essa é a complexidade final do nosso código.

4 - Conclusão

O trabalho no geral correu de forma formidável, conseguimos atender ao que era pedido no relatório enquanto nos preocupávamos por exemplo com a complexidade do código na função primo ou na função que organiza as equipas, obtendo complexidade de $\frac{n}{2}$ e n^2 respetivamente.