

TP2: Ranking de Times

1. Introdução

O trabalho prático propõe a criação de um ranking de times. O trabalho tem por objetivo simular o ranking de times em um torneio de futebol de pontos corridos. Para que a simulação seja feita, o programa deve realizar a ordenação do campeonato sempre que houver algum jogo e, portanto, exige o uso de um eficiente algoritmo de ordenação. Além de ordenar o campeonato, o programa deve se preparar para que possam ser feitas consultas instantâneas à pontuação ou à classificação de um time no campeonato. Para isso, o uso de um algoritmo de busca eficiente é de extrema importância para a ordem de complexidade do programa.

Para a realização da tarefa, a estrutura de dados é organizada em um Tipo Abstrato de Dados (TAD), que contém a declaração de estruturas para representação dos times e contém as funções que serão operadas sobre essas estruturas. Esse modelo de organização ajuda na clareza do código e permite ao programador que vê o código um maior entendimento do programa. São objetivos do trabalho ainda o desenvolvimento da prática de programação e a aplicação da análise de complexidade em um contexto real.

2. Implementação

A implementação do trabalho foi feita por meio do programa principal e de um Tipo Abstrato de Dados. O programa principal é representado pelo arquivo “ranking.c” e o TAD é composto por dois arquivos: “TAD_time.c” e “TAD_time.h”. O primeiro arquivo contém a implementação das funções utilizadas na função principal. O segundo, por sua vez, é composto pela declaração de tipos de dados e pela declaração das funções implementadas. Duas estruturas foram declaradas como tipos de dados no TAD:

Tipo entrada: É uma estrutura composta por 3 strings. Cada string tem por objetivo representar um dos campos de cada linha da entrada. Dessa forma, cada tipo “entrada” tem a capacidade de armazenar as informações contidas em uma linha do arquivo de entrada do programa.

Tipo time: É uma estrutura de dados que tem a finalidade de representar um time. Cada time deve conter importantes informações, como seu nome, sua pontuação e sua classificação no campeonato, por exemplo. Por isso, essa estrutura é composta por 4 atributos: uma string e três inteiros. A string armazena o nome do time. Dentre os valores inteiros, dois representam a pontuação e a posição do time no ranking. Já o terceiro é utilizado como uma *flag*, que demonstra se o time já realizou alguma partida no campeonato ou não.

A seguir, será detalhado o funcionamento do programa principal e, por fim, será especificado o funcionamento de cada uma das funções presentes no TAD.

2.1 Programa Principal

O programa principal do Trabalho Prático é composto apenas pela função “main”. Essa função recebe dois argumentos pelo terminal: arquivo de entrada e arquivo de saída dos dados. Caso o número de argumentos seja diferente de dois, o programa é abortado. Para a execução do programa pelo terminal, o seguinte comando deverá ser utilizado:

./programa [arg 1] [arg 2]

A função principal verifica o número de linhas do arquivo de entrada. Essa verificação é necessária para que os vetores de times “times_camp” e “campeonato” tenham memória alocada corretamente. O vetor “times_camp” é utilizado para armazenar, em ordem alfabética, os times citados na entrada, com a finalidade de que esse vetor seja usado para uma pesquisa binária. O outro vetor, “campeonato”, armazena os times que disputaram alguma partida, em ordem de pontuação. Para cada um dos vetores é alocada memória suficiente para o pior caso, onde cada linha do arquivo contém dois times novos, ou seja, dois times que não foram citados em linhas anteriores.

Inicialmente, o programa chama a função “verifica_entrada”, que armazena o conteúdo da entrada em um vetor. Logo após, é verificado o primeiro atributo de cada linha do arquivo. Caso seja “CONSULTA”, a função operacao_consulta é chamada. Essa função imprime a pontuação ou o ranking do time solicitado no arquivo de saída. Caso o primeiro campo da linha seja “VITORIA”, é chamada a função “armazena_vitoria” para modificar a pontuação dos times. Por fim, caso seja “EMPATE”, a função “armazena_empate” é utilizada para alterar a pontuação dos times que jogaram. Se o primeiro campo da linha tem qualquer outra informação, o programa é finalizado e uma informação de erro é impressa na tela do usuário.

Finalmente, o programa imprime no arquivo de saída dos dados o ranking final do campeonato. Após a impressão, a memória alocada dinamicamente é liberada e o programa é finalizado.

2.2 TAD Time

void verifica_entrada(FILE* ent, entrada* vetor)

A função tem por objetivo armazenar e organizar todo o conteúdo da entrada em um vetor. O vetor utilizado para isso é um vetor do tipo “entrada”, justamente para que cada posição do vetor contenha uma linha da entrada. Para realizar a leitura da entrada, é utilizada a função “fgets”, que armazena uma linha inteira do arquivo em uma string. Essa string deve conter duas vírgulas, que serão utilizadas para separar os campos em três linhas de uma matriz do tipo “char”. Por fim, cada linha da matriz é copiada para um elemento da posição *i* do vetor. O ciclo é repetido até que todas as linhas do arquivo sejam lidas. Dessa maneira, ao fim do processo, o vetor contém todo o conteúdo do arquivo de entrada, onde cada posição do vetor recebe uma linha do arquivo.

Além de realizar essa tarefa, a função realiza ainda a validação da entrada com relação ao número de atributos contidos em cada linha do arquivo de entrada e com relação ao tamanho do nome de cada time, que deve ter no máximo 63 caracteres.

void operacao_consulta(FILE* saida, entrada *vetor, time* campeonato, time* times_camp, int* i, int* quant_times, int* quant_times_jogaram)

A função tem a finalidade de realizar as operações relacionadas à consulta da pontuação ou da posição de um time no ranking. Primeiramente, a função verifica qual é o time a ser consultado. Para isso, é feita uma pesquisa binária em dois vetores, o que contém todos os times e o que contém apenas os times que já jogaram. Dessa maneira, a função determina a posição do time no vetor e

verifica se o time já jogou, se já foi ao menos citado anteriormente ou se o time é novo para o programa. Caso o time seja novo, ele é inserido no vetor que armazena todos os times em ordem alfabética, por meio da função “insere_time”. Caso contrário, a função imprime no arquivo de saída a pontuação ou o ranking do time, conforme a solicitação do usuário. Por fim, a função faz a reordenação do vetor que organiza os times por ordem alfabética, já que este vetor pode ter sido alterado com a inserção de um novo time. Nessa função é realizada ainda uma outra validação da entrada, que verifica se o segundo campo da linha é válido, caso o primeiro campo tenha sido “CONSULTA”.

**void armazena_vitoria(entrada* vetor, time* campeonato, time* times_camp,
int *i, int* quant_times, int* quant_times_jogaram)**

Esta função tem por objetivo alterar a pontuação dos dois times que realizaram a partida resultante em uma vitória. Inicialmente, são verificados quais são os times que disputaram a partida e são verificadas as posições dos times em dois vetores, o que é ordenado pela pontuação e o que está em ordem alfabética dos times. A partir dessa pesquisa nos vetores, é possível determinar, para cada um dos times, se o time é novo para o programa e se ele já jogou ou não. Com isso, a função verifica se é necessário ou não inserir o time no vetor. Caso seja necessário, é chamada a função “insere_time”. Depois de verificada a posição do time em cada um dos vetores, é feita a modificação na pontuação dos times. Por meio da função “modifica_pontuacao”, 3 pontos são acrescidos à pontuação do time vencedor. Ao final da função, os dois vetores são ordenados novamente, um por ordem alfabética e o outro pela pontuação dos times.

**void armazena_empate(entrada* vetor, time* campeonato, time* times_camp,
int *i, int* quant_times, int* quant_times_jogaram)**

A função tem a finalidade de modificar a pontuação de dois times que empataram um jogo e é praticamente idêntica à função “armazena_vitoria”. A única diferença entre as funções está na modificação da pontuação dos times. Especificamente nesta função, a alteração nos pontos de cada time é feita de forma que cada um dos times receba mais 1 ponto apenas, já que o resultado da partida foi um empate.

void insere_time(time time_novo, time* vetor, int* quant_times)

A função recebe um vetor e um time novo. O objetivo dessa função é adicionar o time novo ao vetor. A adição do time ao vetor é feita de forma que o time passe a ocupar a posição N+1 do vetor, onde N é o número de times que já estão no vetor.

int consulta_time(time time_procurado, time* times_camp, int* quant_times)

Esta função realiza uma pesquisa binária no vetor ordenado por ordem alfabética do nome dos times. Caso encontre o nome do time procurado, a função retorna a posição do time procurado no vetor. Caso contrário, a função retorna um valor negativo.

int verifica_ranking(time time_procurado, time* campeonato, int* quant_times)

De maneira semelhante à função “consulta_time”, essa função também realiza uma pesquisa binária no vetor ordenado pela pontuação dos times para encontrar a posição de um time no vetor. No entanto, alguns tratamentos especiais foram dados à implementação dessa função. A diferença é que nessa função a chave procurada (pontuação do time) pode ser a mesma em diversos times. Então, para encontrar um time por meio de sua pontuação, primeiramente a função realiza a pesquisa binária até encontrar um time com a pontuação idêntica à procurada. Após encontrar um

time com essa pontuação, são verificados os times com a mesma pontuação nas posições adjacentes. Dessa forma, a função realiza uma pequena pesquisa sequencial, comparando o nome das equipes, entre os times com a mesma pontuação. Assim, caso o time esteja no vetor, sua posição é retornada.

void modifica_pontuacao(time* vetor, int posicao1, int pont_a, int posicao2, int pont_b)

A função modifica a pontuação de dois times específicos passados por argumento. Além disso, é modificada a *flag* de cada um dos times, para mostrar ao programa que os times já realizaram algum jogo no campeonato.

void ordena_ranking(time* campeonato, int quant_times)

Esta função utiliza-se do algoritmo de Inserção para ordenar um vetor pela pontuação dos times. A escolha desse algoritmo se deve ao seu excelente comportamento em situações onde o vetor a ser ordenado está praticamente ordenado. No entanto, a função não utiliza somente o critério pontuação para ordenar o vetor. Para a ordenação, é verificada também a ordem alfabética do nome dos times, caso a pontuação das equipes seja igual.

void ordena_nome(time* camp_times, int quant_times, time* campeonato)

Assim como a função “ordena_nome”, o algoritmo utilizado é o algoritmo de Inserção. Esse algoritmo de ordenação destaca-se entre os demais por ser mais rápido quando o vetor a ser ordenado está quase ordenado. Dessa forma, o algoritmo é utilizado para ordenar um vetor de times pela ordem alfabética dos nomes.

3. Casos de teste

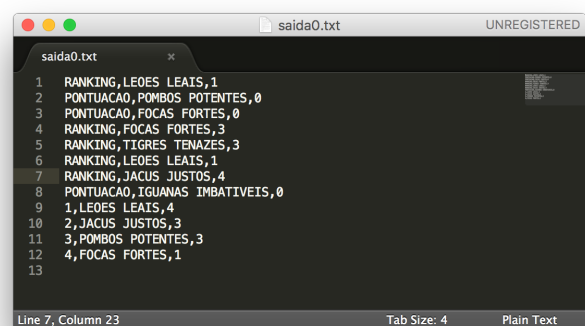
Foram executados onze testes para verificação do correto funcionamento do programa. Os testes foram divididos em teste 0, teste 1, teste 2 e assim por diante, até o teste 10. Os quatro primeiros testes estão detalhados a seguir. Os demais testes tem por objetivo testar a validação da entrada e foram anexados ao trabalho, no diretório */testes*.

3.1 Teste 0

O Teste 0 foi realizado com o exemplo presente na especificação do Trabalho Prático.



```
1 VITORIA, LEOES LEAIS, POMBOS POTENTES
2 CONSULTA, RANKING, LEOES LEAIS
3 CONSULTA, PONTUACAO, POMBOS POTENTES
4 CONSULTA, PONTUACAO, FOCAS FORTES
5 CONSULTA, RANKING, FOCAS FORTES
6 CONSULTA, RANKING, TIGRES TENAZES
7 EMPATE, LEOES LEAIS, FOCAS FORTES
8 VITORIA, POMBOS POTENTES, JACUS JUSTOS
9 CONSULTA, RANKING, LEOES LEAIS
10 CONSULTA, RANKING, JACUS JUSTOS
11 CONSULTA, PONTUACAO, IGUANAS IMBATIVEIS
12 VITORIA, JACUS JUSTOS, POMBOS POTENTES
```



```
1 RANKING, LEOES LEAIS, 1
2 PONTUACAO, POMBOS POTENTES, 0
3 PONTUACAO, FOCAS FORTES, 0
4 RANKING, FOCAS FORTES, 3
5 RANKING, TIGRES TENAZES, 3
6 RANKING, LEOES LEAIS, 1
7 RANKING, JACUS JUSTOS, 4
8 PONTUACAO, IGUANAS IMBATIVEIS, 0
9 1, LEOES LEAIS, 4
10 2, JACUS JUSTOS, 3
11 3, POMBOS POTENTES, 3
12 4, FOCAS FORTES, 1
13
```

3.2 Teste 1

O Teste 1 foi feito com o objetivo de simular as partidas do Grupo D da "Liga dos Campeões da Europa 2015/2016". O grupo é composto por quatro equipes, que devem se enfrentar duas vezes. No entanto, o teste foi feito utilizando apenas as 4 primeiras rodadas da Liga. Para tornar o teste mais eficaz, foram inseridas operações de consulta ao longo do teste.

MENU

ge

LIGA DOS CAMPEÕES

BUSCAR

4

Astana

2

4

0

2

2

2

8

-6

16,7

ATL

X

GAL

GRUPO D

| CLASSIFICAÇÃO | P | J | V | E | D | GP | GC | SG | % | ÚLT. JOGOS | |
|----------------------------|-----|---|---|---|---|----|----|----|------|------------|--|
| 1 Manchester City | 1 + | 9 | 4 | 3 | 0 | 1 | 8 | 5 | 3 | 75,0 | |
| 2 Juventus | 1 + | 8 | 4 | 2 | 2 | 0 | 5 | 2 | 3 | 66,7 | |
| 3 Sevilla | | 3 | 4 | 1 | 0 | 3 | 5 | 7 | -2 | 25,0 | |
| 4 Borussia Mönchengladbach | | 2 | 4 | 0 | 2 | 2 | 6 | -4 | 16,7 | | |

01ª RODADA

TER 15/09/2015 CITY OF MANCHESTER 15:45

MAC

1 × 2

JUV

VEJA COMO FOI

TER 15/09/2015 RAMÓN SÁNCHEZ PIZJUÁN 15:45

SEV

3 × 0

BMG

VEJA COMO FOI

```
teste1.csv
1 VITORIA, JUVENTUS, MANCHESTER CITY
2 VITORIA, SEVILLA, BORUSSIA
3 CONSULTA, RANKING, JUVENTUS
4 CONSULTA, PONTUACAO, JUVENTUS
5 VITORIA, MANCHESTER CITY, BORUSSIA
6 VITORIA, JUVENTUS, SEVILLA
7 CONSULTA, RANKING, SEVILLA
8 CONSULTA, PONTUACAO, SEVILLA
9 EMPATE, JUVENTUS, BORUSSIA
10 VITORIA, MANCHESTER CITY, SEVILLA
11 CONSULTA, RANKING, MANCHESTER CITY
12 CONSULTA, PONTUACAO, MANCHESTER CITY
13 EMPATE, BORUSSIA, JUVENTUS
14 VITORIA, MANCHESTER CITY, SEVILLA
```

```
saida1.txt
1 RANKING, JUVENTUS, 1
2 PONTUACAO, JUVENTUS, 3
3 RANKING, SEVILLA, 3
4 PONTUACAO, SEVILLA, 3
5 RANKING, MANCHESTER CITY, 2
6 PONTUACAO, MANCHESTER CITY, 6
7 1, MANCHESTER CITY, 9
8 2, JUVENTUS, 8
9 3, SEVILLA, 3
10 4, BORUSSIA, 2
11
```

3.3 Teste 2

O Teste 2 simula as rodadas do Grupo A da “Copa América 2015”. Ao todo, foram 3 rodadas, onde cada um dos quatro times enfrentou os outros 3 times uma única vez. Assim como no Teste 1, operações de consulta foram inseridas no teste.

MENU

ge

COPA AMÉRICA

BUSCAR

DESEMPENHO

PRIMEIRA FASE

GRUPO A

| CLASSIFICAÇÃO | | P | J | V | E | D | GP | GC | SG | % | ÚLT. JOGOS | | 01ª RODADA | |
|---------------|---------|-------|---|---|---|---|----|----|----|----|------------|-------|--|--|
| 1 | Chile | 0 0 0 | 7 | 3 | 2 | 1 | 0 | 10 | 3 | 7 | 77.8 | 0 0 0 | QUI 11/06/2015 SANTIAGO (NACIONAL) 20:30 | |
| 2 | Bolívia | 0 0 0 | 4 | 3 | 1 | 1 | 1 | 3 | 7 | -4 | 44.4 | 0 0 0 | CHI 2 x 0 EQU | |
| | | | | | | | | | | | | | VEJA COMO FOI | |
| 3 | Equador | 1 0 0 | 3 | 3 | 1 | 0 | 2 | 4 | 6 | -2 | 33.3 | 0 0 0 | SEX 12/06/2015 VÍÑA DEL MAR 20:30 | |
| 4 | México | 1 0 0 | 2 | 3 | 0 | 2 | 1 | 4 | 5 | -1 | 22.2 | 0 0 0 | MEX 0 x 0 BOL | |
| | | | | | | | | | | | | | VEJA COMO FOI | |

```

teste2.csv
1 VITORIA,CHILE,EQUADOR
2 EMPATE,MEXICO,BOLIVIA
3 CONSULTA,RANKING,BRASIL
4 CONSULTA,PONTUACAO,BRASIL
5 VITORIA,BOLIVIA,EQUADOR
6 EMPATE,CHILE,MEXICO
7 CONSULTA,RANKING,CHILE
8 CONSULTA,PONTUACAO,BOLIVIA
9 VITORIA,EQUADOR,MEXICO
10 VITORIA,CHILE,BOLIVIA

```

Line 10, Column 22 Tab Size: 4 Plain Text

```

saida2.txt
1 RANKING,BRASIL,5
2 PONTUACAO,BRASIL,0
3 RANKING,CHILE,2
4 PONTUACAO,BOLIVIA,4
5 1,CHILE,7
6 2,BOLIVIA,4
7 3,EQUADOR,3
8 4,MEXICO,2
9

```

Line 1, Column 1 Tab Size: 4 Plain Text

3.4 Teste 3

O Teste 3, por sua vez, simulou as partidas do Grupo A da “Copa do Mundo 2014”. Assim como no Teste 2, a competição é composta por 3 rodadas onde todos os times se enfrentam uma única vez.

classificação / jogos

Fase de grupos

| Grupo A - 3ª rodada | | | | | | | | | | |
|---------------------|---|---|---|---|---|----|----|----|------|--|
| CLASSIFICAÇÃO | P | J | V | E | D | GP | GC | SG | % | |
| 1 Brasil | 7 | 3 | 2 | 1 | 0 | 7 | 2 | 5 | 77.8 | |
| 2 México | 7 | 3 | 2 | 1 | 0 | 4 | 1 | 3 | 77.8 | |
| 3 Croácia | 3 | 3 | 1 | 0 | 2 | 6 | 6 | 0 | 33.3 | |
| 4 Camarões | 0 | 3 | 0 | 0 | 3 | 1 | 9 | -8 | 0.0 | |

Seg 23/06/2014 - 17h00 Arena Pernambuco

CRO 1 x 3 MEX

Seg 23/06/2014 - 17h00 Mané Garrincha

CAM 1 x 4 BRA

```

teste3.csv
1 VITORIA,BRASIL,CROACIA
2 VITORIA,MEXICO,CAMARÕES
3 CONSULTA,RANKING,BRASIL
4 CONSULTA,PONTUACAO,BRASIL
5 CONSULTA,PONTUACAO,MEXICO
6 EMPATE,BRASIL,MEXICO
7 VITORIA,CROACIA,CAMARÕES
8 CONSULTA,PONTUACAO,CAMARÕES
9 CONSULTA,RANKING,MEXICO
10 VITORIA,MEXICO,CROACIA
11 CONSULTA,RANKING,MEXICO
12 VITORIA,BRASIL,CAMARÕES

```

23 characters selected Tab Size: 4 Plain Text

```

saida3.txt
1 RANKING,BRASIL,1
2 PONTUACAO,BRASIL,3
3 PONTUACAO,MEXICO,3
4 PONTUACAO,CAMARÕES,0
5 RANKING,MEXICO,2
6 RANKING,MEXICO,1
7 1,BRASIL,7
8 2,MEXICO,7
9 3,CROACIA,3
10 4,CAMARÕES,0
11

```

Line 4, Column 21 Tab Size: 4 Plain Text

3.5 Outros testes

Os outros testes foram feitos com a finalidade de testar a validação da entrada. São eles:

- Teste 4** : Utiliza na entrada um campo em branco, onde existem duas vírgulas seguidas.
- Teste 5** : Utiliza uma linha que contém 4 campos, ao invés de 3 apenas.
- Teste 6** : A entrada contém uma linha em branco.
- Teste 7** : Utiliza na entrada um campo em branco, onde existem espaços vazios.

-**Teste 8** : Utiliza uma operação diferente de CONSULTA, VITORIA ou EMPATE.

-**Teste 9** : A entrada contém o nome de um time com 64 caracteres.

-**Teste 10** : Coloca a palavra PONTUACAO no segundo campo de uma linha que contém a operação CONSULTA.

4. Estudo de complexidade

Será realizada a análise de complexidade de cada função executada pelo programa e, por fim, será analisado o programa principal. A análise será feita com base no tempo de execução dos procedimentos implementados, em função do número de times e em função do número de consultas. Todo o estudo será feito utilizando a notação O .

Função verifica_entrada: A função é composta por um loop do tipo "while", que abrange todas as outras operações. Esse loop é executado n vezes, de acordo com o número de linhas do arquivo de entrada. As operações dentro do loop são, em maioria, $O(1)$. Há dois "for" aninhados dentro do loop, porém são eles são executados um numero fixo de vezes e, portanto, são $O(1)$. Há ainda dentro do loop outro "while", que é executado até que o $\backslash n$ ou o EOF seja encontrado em uma string, ou seja, n vezes. Logo, esse ultimo "while" é $O(n)$, ja que dentro dele existem apenas uma comparação e 2 atribuições no pior caso. As demais operações do loop são atribuições. Portanto, a função é da ordem $O(n^2)$ tanto para o número de consultas, quanto para o número de times.

Função operacao_consulta: A função executa apenas comparações. Porém, no pior caso são chamadas as funções "consulta_time"- $O(\log n)$, "insere_time"- $O(1)$, "verifica_ranking"- $O(\log n)$ e "ordena_nome"- $O(n^2)$. Logo, a função é $O(n^2)$.

Função armazena_vitoria: A função executa apenas comparações. Porém, no pior caso são chamadas as funções "consulta_time"- $O(\log n)$, "insere_time"- $O(1)$, "verifica_ranking"- $O(\log n)$, "modifica_pontuacao"- $O(1)$, "ordena_ranking"- $O(n^2)$ e "ordena_nome"- $O(n^2)$. Logo, a função é $O(n^2)$.

Função armazena_empate: A análise de complexidade da função é semelhante à função "armazena_vitoria". Portanto, a ordem de complexidade é $O(n^2)$.

Função insere_time: A função insere um time no fim do vetor. Portanto, a ordem de complexidade é $O(1)$.

Função consulta_time: A função realiza uma pesquisa binária e, portanto, tem complexidade $O(\log n)$ em relação ao número de consultas e em relação ao número de times.

Função verifica_ranking: A função realiza uma pesquisa binária. No pior caso, a pesquisa pode ser sequencial, caso os times tenham todos a mesma pontuação. e, portanto, tem complexidade $O(n)$, mas na prática acaba se aproximando mais de $O(\log n)$ para a maioria dos casos de teste, tanto em função do número de times quanto em função do número de consultas.

Função modifica_pontuacao: A função realiza quatro atribuições e, portanto, tem ordem de complexidade igual a $O(1)$.

Função ordena_ranking: A função realiza uma ordenação com o uso do algoritmo de Inserção. Portanto, a função tem ordem de complexidade $O(n^2)$ em função do número de times e é $O(1)$ com relação ao número de consultas realizadas.

Função ordena_nome: Essa função também realiza uma ordenação com o uso do algoritmo de Inserção. Portanto, a função tem ordem de complexidade $O(n^2)$ em função do número de times e é $O(1)$ com relação ao número de consultas realizadas.

Função Principal: A função principal realiza diversas atribuições. Além disso, ela contém um “while” que é executado de acordo com o número de linhas do arquivo de entrada, ou seja, é $O(n)$. Há também dois “for” aninhados, mas que são executados um número fixo de vezes e, portanto, são $O(1)$. Para inicializar um vetor, a função contém um for com complexidade $O(n)$, de acordo com o número de times da entrada. Outro loop existente na função é um “while”, que pode chamar a função “armazena_vitoria”, a função “armazena_empate” ou a função “operacao_consulta”, todas $O(n^2)$. Esse loop é executado n vezes, de acordo com o número de linhas do arquivo de entrada e, portanto, é $O(n^3)$. Por fim, um outro loop é executado n vezes, de acordo com o número de jogos realizados. Portanto, a complexidade do programa é da ordem de $O(n^3)$, em função do número de times e em função do número de consultas feitas.

5. Conclusão

A execução de um ranking requer bastante conhecimento com relação aos algoritmos de ordenação. Além disso, como a proposta do Trabalho Prático não visa somente a ordenação, mas também a pesquisa, a tarefa exige que o aluno saiba implementar um algoritmo de pesquisa também. A dificuldade do trabalho está relacionada, principalmente, à exigência de um método de pesquisa que tenha ordem de complexidade estritamente inferior a $O(n)$. Para seguir as exigências, a solução ideal é o uso da Pesquisa Binária. Porém, realizar uma pesquisa no vetor de times por meio do nome da equipe é fácil desde que o vetor esteja em ordem alfabética, o que não é o caso. Durante o programa, é necessário realizar consultas pelo nome do time em um vetor ordenado pela pontuação, ou seja, onde os nomes estão desordenados. A solução desse problema é modificar o algoritmo de Pesquisa Binária, de forma que ele cumpra a tarefa e sua complexidade continue menor que $O(n)$.

O Trabalho Prático 2 exige ainda o estudo dos algoritmos de ordenação, com o intuito de que o aluno tenha condições de identificar qual algoritmo melhor se aplica à situação em que o trabalho se encontra. No caso do TP2, as ordenações são realizadas em vetores que estão, na maioria das vezes, praticamente ordenados. Neste caso, o algoritmo que tem maior eficiência é o Insertion Sort (Inserção). Por fim, a execução do Trabalho Prático 2 tem grande importância para o entendimento e para a aplicação do conteúdo abordado no curso de AEDS II, no que tange o uso de algoritmos de ordenação e de algoritmos de pesquisa.

6. Referências

- [1] Ziviani, N., Projeto de Algoritmos com Implementações em Pascal e C, 2a Edição, Editora Thomson, 2004.
- [2] Stack Overflow - <http://stackoverflow.com>
- [3] C Plus Plus - <http://www.cplusplus.com>

7. Anexos

Listagem dos Programas:

- Ranking.c
- TAD_time.c e TAD_time.h