Trabalho de Programação

Pablo Freitas Santos
Engenharia Mecatrônica
Univiversidade Federal de Santa Catarina
Joinville, Brasil
pablo.santos1999@gmail.com

***Resumo*—Esse trabalho Foi feito com intuito de implementar um algoritmo de uma telemetria partindo de um arquivo tabelado que lista dados de forma sequencial.**

***Index Terms*—TDA, arquivos, estruturas**

I. Introdução

O cenário do esporte atual necessita de análise detalhada de cada informação captada durante uma partida, para isso foi preciso pensar em uma forma de listar todas as informações de cada jogador em um arquivo de acesso sequencial.

Nesse projeto foi utilizados tipos de dados abstratos, definido como um conjunto de valores e uma sequência de operações sobre estes valores [1] para modularizar o código e aumentar a eficiência do sistema.

Com a utilização principalmente das TDAs, listas duplamente encadeadas e estrutura, foram feitas operações com os dados obtidos, como por exemplo velocidade máxima, distância percorrida por cada jogador e tempo efetivo em campo.

II. Desenvolvimento

A organização do trabalho foi feita em 3 partes: leitura do arquivo sequencial bruto e armazenamento dessa informação em uma lista duplamente encadear, seleção dos dados importantes a fim de fazer as operações e posteriormente a realização das operações previamente definidas com os dados convertidos

Primeiro, foi preciso utilizar meios de captação de linhas como fgets, na leitura do arquivo e colocar essa linha em forma de vetor de caracteres em uma lista sempre inserindo no próximo da cauda, mantendo todas as informações em ordem até encontrar o final do arquivo e retornar essa lista.

Em seguida, cada célula da lista retornada passou a conter uma linha do arquivo, mantendo uma ordem que facilite o uso de seus dados para os cálculos solicitados, como por exemplo, tempo local e velocidade.

Para armazenar esse dado, foi criado uma lista cujos dados são outras listas, o que significa que cada célula da lista principal era uma sublista contendo as informações dos jogadores. Assim, foi possível acessar a informação de outro jogador independente da ordem que esse dado foi armazenado.

Além disso, foram consideradas informações após serem tratadas em uma estrutura chamada informação dos jogadores, possuindo a velocidade, identidade do jogador e o tempo separado em horas, minutos, segundos.

Antes de ir para a parte dos cálculos, foi pensado em uma forma de avaliar por período escolhido pelo usuário.

Assim foi criado uma entrada para definir previamente qual o tempo de cada parte do jogo que será tomado como parâmetro nas operações a serem avaliadas.

Nas operações, houve a utilização de um molde em todas as funções criadas , o qual foi feito na forma que a função recebe como parâmetro uma célula, que é referente da cabeça da lista do jogador em questão que foi armazenada as informações dos jogadores e foi utilizada para percorrer todos os dados que possuíam no seu interior.

Figura 1. Exemplo do molde

```
float velo_maxima(celula_t *aux){
    if(aux==NULL){
        return(0);
    }
    float velo=0;
    informacao_jogadores *info_temp = (informacao_jogadores*)lista_dado(aux);
    while(aux!=NULL){
        info_temp = (informacao_jogadores*)lista_dado(aux);
        if(info_temp->vel >= velo){
            velo = info_temp->vel;
        }
        aux = lista_proximo(aux);
    }
    return(velo);
}
```

Com isso, dependendo da informação requerida, foi utilizado um parâmetro de comparação para verificar a velocidade média atingida, por exemplo.

Um caso especial, foi a função que encontra quantas vezes o jogador ultrapassou a velocidade dada como parâmetro, foi implementada de forma recursiva, para exemplificar que pode ter outras formas sem ser a interativa de se revolver o mesmo problema e de forma mais compacta.

Figura 2. Ocorrências da velocidade dada como parâmetro

```
int run_forest_run(celula_t *info_jogador_calc_vel, int vel_compara){
    if(info_jogador_calc_vel == NULL)
        return(0);
    int i=0;
    informacao_jogadores *aux = (informacao_jogadores*)info_jogador_calc_vel->dado;
    if((int)aux->vel >= vel_compara){
        i=1;
    }
    else{
        i=0;
    }
    return(i + run_forest_run(info_jogador_calc_vel->prox, vel_compara));
}
```

Foi utilizado a lógica de que quando a função retornaria quantas vezes ele encontrasse o valor igual ou maior ao parâmetro, caso ele achasse algum que se enquadrava a

essa condição era retornado a mesma função, mas dava um acréscimo de mais um e por fim quando chegasse no final dos dados ele retornava o valor final que era o objetivo da função.

Na parte de liberar memória alocada foi criado 3 funções de destruição: a de destruir lista por meio de recursão, destruir lista principal e destruir lista secundária.

A divisão em etapas da destruição é importante porque a ordem em que o dado é liberado interfere no funcionamento correto do programa.

Além disso, como foi trabalhado listas que possuem em seu conteúdo outras sublistas então é uma solução modularizar a parte da destruição.

III. Discussão

O cenário do esporte atual necessita de análise detalhada de cada informação captada durante uma partida. Para isso foi de fundamental importância a utilização das TDAs para evitar erros e facilitar a identificar-los durante a construção do código, pois ao modularizar o código foi possível reutilização o mesmo e testar especificamente cada função.

Outro importante ponto foi utilização de listas duplamente encadeadas para acessar as informações guardadas nas estrutura. Como são listas pode-se acessar em qualquer lugar, mas nesse caso foi feito a utilização sempre em sequência do dado listado.

Chegou-se a resultados claros e definitivos apartir das informações dada pelo arquivo sequencial.

Figura 3. Resultados

```
+-----------------------------------------------------------+
| Dados referentes ao jogador [01]:                         |
| Quantidade de vezes que alcancou 18 (km/h) ou mais: 08    |
| Quantidade de vezes que alcancou 20 (km/h) ou mais: 05    |
| Velocidade Media: 4.717 (km/h) | 1.310 (m/s)              |
| Velocidade Media Round 1: 4.926 (km/h) | 1.368 (m/s)      |
| Velocidade Media Round 2: 4.512 (km/h) | 1.253 (m/s)      |
| Velocidade Maxima: 25.438 (km/h) | 7.066 (m/s)            |
| Distancia Percorrida: 7.619 km                            |
| Tempo 1 [H:M:S]: 00:48:00                                 |
| Tempo 2 [H:M:S]: 00:49:00                                 |
| Tempo Total [H:M:S]: 01:37:00                             |
+-----------------------------------------------------------+
| Dados referentes ao jogador [02]:                         |
| Quantidade de vezes que alcancou 18 (km/h) ou mais: 06    |
| Quantidade de vezes que alcancou 20 (km/h) ou mais: 02    |
| Velocidade Media: 3.015 (km/h) | 0.837 (m/s)              |
| Velocidade Media Round 1: 3.060 (km/h) | 0.850 (m/s)      |
| Velocidade Media Round 2: 2.970 (km/h) | 0.825 (m/s)      |
| Velocidade Maxima: 21.788 (km/h) | 6.052 (m/s)            |
| Distancia Percorrida: 4.873 km                            |
| Tempo 1 [H:M:S]: 00:48:00                                 |
| Tempo 2 [H:M:S]: 00:49:00                                 |
| Tempo Total [H:M:S]: 01:37:00                             |
+-----------------------------------------------------------+
```

No entanto, houve problema com vazamento de memória devido à complexidade da manipulação dos dados. Para não comprometer as informações obtidas foram deixados alguns vazamentos no código.

Figura 4. Vazamento

```
==26== 2,000 bytes in 1 blocks are definitely lost in loss record 7 of 8
==26==    at 0x4C2BBAF: malloc (vg_replace_malloc.c:299)
==26==    by 0x109B4A: le_bruto (seleciona.c:16)
==26==    by 0x109109: main (main.c:10)
==26==
==26== 78,552 bytes in 3,273 blocks are definitely lost in loss record 8 of 8
==26==    at 0x4C2BBAF: malloc (vg_replace_malloc.c:299)
==26==    by 0x109CBB: seleciona_dados (seleciona.c:48)
==26==    by 0x109F13: trata_dados (seleciona.c:96)
==26==    by 0x109E6D: listas_listas (seleciona.c:77)
==26==    by 0x109120: main (main.c:11)
==26==
```

No entanto, a maioria dos vazamentos apresentados possuem cerca de 24 bytes por bloco vazado, somente 1 não conseguiu ser destruído, pois é uma estrutura que armazena outra estrutura e para conciliar a integridade da informação e a memória não foi possível.

IV. Conclusão

Para construção de um algoritmo de captação de dados, é preciso se planejar na forma de organização desses dados e buscar separar-los por módulos de funções genéricas.

O planejamento desse código foi feito em função da necessidade da fácil mudança do algoritmo para buscar outras informações, listar todas as operações a serem feitas, para no momento de selecionar os dados coletar somente o essencial e deixar o código otimizado.

Referências

[1] TENENBAUM, Aaron M.; AUGENSTEIN, Moshe J.; LANGSAM, Yedidyah; SOUZA, Tereza Cristina Félix de. Estruturas de dados usando C. São Paulo: Makron Books, 2007. 426 p. (Programa do Livro Texto, 2007 ; 39). ISBN 8576051370