Relatório

Algoritmos e Estruturas de Dados II

Aluno/os: André Freitas nº21112
Luis Silva nº 21113

Professor/es: Alberto Simões

Óscar Ribeiro

Licenciatura em Engenharia de Sistemas Informáticos

Barcelos, maio, 2021

Resumo:

Neste trabalho de teor prático são abordados 3 ficheiros de dados contendo todos os conjuntos de peças LEGO até à data.

A implementação em C usada serve para cumprir diversos objetivos propostos desde a manipulação dos dados até ao cálculo de peças de um determinado tema.

Índice

1. Introdução

1.1. Complexidade dos algoritmos

Apesar de todos os algoritmos serem dados no decorrer das aulas, não podemos negar que mesmo assim os algoritmos usados nesta implementação foram bastante difíceis perante os algoritmos dados no semestre anterior.

Mas depois de habituar às listas usadas neste projeto, vai ficando razoavelmente mais fácil a sua utilização e manipulação para os objetivos propostos.

O uso de listas quase circulares facilita e acelera bastante o processo de percorrer todos os ficheiros de cima a baixo, dado que uma lista simplesmente ligada no meu computador, por exemplo, demorava à volta de meia hora para ler e assim apenas demora 10 segundos.

Por fim, fica a nota que pesquisas na internet e esclarecimento de dúvidas é uma obrigação, dado que algumas funções necessárias para a implementação deste projeto, não são faladas nas aulas, pelo menos a nosso ver.

1.2. Contextualização

Trabalho prático que consiste na consolidação de conhecimentos aprendidos ao longo do primeiro e segundo semestre, bem como promover a autoaprendizagem através da investigação em grupo ou individual.

2. Enunciado

2.1.1. Interpretação e decisões tomadas

No enunciado é nos dito que existem 3 ficheiros contendo dados, logo a primeira coisa a fazer é criar 3 structures (uma para cada ficheiro) que contenham os dados indicados no ficheiro.

1. Quais os conjuntos de determinado tema (ordenados pelo ano);

Com este objetivo o que nos é pedido, é perguntar ao utilizador que tema ele gostaria de pesquisar e depois com esse tema selecionado mostramos todos os conjuntos dentro desse tema por ordem cronológica do mais antigo ao mais recente.

2. As peças de determinado tipo em determinado conjunto;

Aqui é nos requisitado perguntar ao utilizador o nome de um conjunto e dentro desse conjunto questionar o tipo de peça que este quer. Algo que é possível com o ficheiro *parts_sets.tsv* que mostra a identificação de os conjuntos indicando a identificação do tipo de peças que o conjunto tem.

3. Quais as peças necessárias para construir um dado conjunto, indicando os dados de cada peça e respetiva quantidade;

Neste objetivo a melhor solução é perguntar ao utilizador o nome do conjunto e com a sua respetiva identificação procurar no ficheiro *parts_sets.tsv* todas as peças associadas a essa identificação, demonstrando o seu tipo e a sua quantidade.

4. O total de peças em stock;

Este é muito simples, apenas que primeiro verificar se a peça está em stock ou não e depois somar todas as peças em stock e apresentar esse valor.

5. O total de peças incluídas num determinado conjunto;

Perguntamos ao utilizador o nome de um conjunto e depois com a sua identificação verificar no ficheiro *parts_sets.tsv* todas as peças incluídas no conjunto e somá-las.

6. A peça que é utilizada em mais conjuntos diferentes, independentemente da quantidade em cada um deles;

Para esta situação temos que aceder ao ficheiro *parts_sets.tsv* e verificar a identificação de tipo de peça que se repete mais vezes ao longo do ficheiro.

7. A lista dos conjuntos que se conseguem construir com o stock existente.

Neste objetivo temos que aceder ao ficheiro *sets.tsv*, pegar na identificação dos conjuntos e com essa identificação comparar com a identificação dos conjuntos no ficheiro *parts_sets.tsv*. Depois dessa comparação, devemos verificar se a quantidade de peças necessárias para a construção de um conjunto é maior que o stock dessas mesmas peças.

Alterar o número de peças em stock;

Esta proposta é bastante fácil apenas pegamos na quantidade de uma peça e acrescentamos ou retiramos stock.

A adição de stock com base no identificador de um conjunto (adicionar as peças que esse conjunto tem ao stock total de peças).

Aqui é para escolher um conjunto e depois no ficheiro *parts_sets.tsv* com a identificação desse conjunto ver todas as peças nesse conjunto, somar toda a quantidade e depois com essa soma tens que adicionar no stock total que já tinhas visto

Remover todas as peças de determinada classe (tipo de peça);

Temos que perguntar ao utilizador para escolher uma classe e de seguida selecionar todas as peças incluídas nessa classe e removê-las dos ficheiros

Remover todos os sets de determinado tema

Este é bastante parecido com o anterior só que em vez de classe são temas que o utilizador deve escolher para que depois possamos retirar todos os sets dos ficheiros.

3. Implementação

3.1.1. Estruturas

```
typedef struct _sets {
    char * num;
    char * name;
    int year;
    char * theme;

    struct _sets *prev, *next, *last, *first;
} Sets;
```

Esta estrutura representa todos os dados referentes ao ficheiro *sets.tsv*. Esses dados são: o número da coleção, o nome, o ano na qual foi comercializado e o tema na qual está relacionado.

```
typedef struct _parts {
    char * num;
    char * name;
    char * class;
    int stock;

    struct _parts *prev, *next, *last, *first;
} Parts;
```

Esta estrutura representa todos os dados referentes ao ficheiro *parts.tsv*. Esses dados são: o número da peça, o nome, o tema na qual está relacionado e a quantidade em stock.

```
typedef struct part_sets {
    char * set_num;
    int quantity;
    char * part_num;

    struct part_sets *prev, *next, *last, *first;
} PS;
```

Esta estrutura representa todos os dados referentes ao ficheiro *parts_sets.tsv*. Esses dados são: o número da coleção, a quantidade em stock e o número da peça.

3.1.2. Funções

```c
Sets * createSetsList (char * num, char * name, int year, char * theme)
{
    Sets * list = (Sets*) malloc(sizeof(Sets));
    list->num = strdup(num);
    list->name = strdup(name);
    list->year = year;
    list->theme = strdup(theme);
    list->next = NULL;
    list->prev = NULL;
    list->last = list;
    list->first = list;

    return list;
}

Sets * Sets_insert (Sets * list, char * num, char * name, int year, char * theme)
{
    assert(list);

    list->last->next = createSetsList(num, name, year, theme);
    list->last->next->prev = list->last;
    list->last->next->first = list;
    list->last = list->last->next;

    return list;
}
```

Estas duas funções são bastante importantes para a leitura dos ficheiros dados. Sendo que a primeira é a criação de uma lista quase circular e a segunda, a inserção da mesma.

Estas duas listas são relativas ao ficheiro *sets.tsv.*

```c
Parts * createPartsList (char * num, char * name, char * class, int stock)
{
    Parts * list = (Parts*) malloc(sizeof(Parts));
    list->num = strdup(num);
    list->name = strdup(name);
    list->class = strdup(class);
    list->stock = stock;
    list->next = NULL;
    list->prev = NULL;
    list->last = list;
    list->first = list;

    return list;
}

Parts * Parts_insert (Parts * list, char * num, char * name, char * class, int stock)
{
    assert(list);

    list->last->next = createPartsList(num, name, class, stock);
    list->last->next->prev = list->last;
    list->last->next->first = list;
    list->last = list->last->next;

    return list;
}
```

Exatamente a mesma coisa que na figura anterior, mas desta vez estas duas listas são relativas ao ficheiro *parts.tsv.*

```c
PS * createPSList (char * set_num, int quantity, char * part_num)
{
    PS * list = (PS*) malloc(sizeof(PS));
    list->set_num = strdup(set_num);
    list->quantity = quantity;
    list->part_num = strdup(part_num);
    list->next = NULL;
    list->prev = NULL;
    list->last = list;
    list->first = list;

    return list;
}

PS * PS_insert (PS * list, char * set_num, int quantity, char * part_num)
{
    assert(list);

    list->last->next = createPSList(set_num, quantity, part_num);
    list->last->next->prev = list->last;
    list->last->next->first = list;
    list->last = list->last->next;

    return list;
}
```

O mesmo é feito aqui, apenas mudando o facto de que estas duas funções são relativas ao ficheiro parts_sets.tsv

```c
Sets *loadSets(char *PATH)
```

```c
Parts *loadParts(char *PATH)
```

```c
PS *loadPS(char *PATH)
```

Estas funções são o nosso meio de ler todas as variáveis contidas nos ficheiros sejam elas int, char ou double.

```
266    void PrintSetsT(Sets *SetsList, Parts *PartsList, PS *PSList)
```

Esta função serve para responder ao objetivo 1 onde é pedido para listar conjuntos de um determinado tema.

```
313    void PrintPartsTC(Sets * SetsList, Parts * PartsList, PS * PSList){
```

Esta função serve para cumprir o objetivo 2 que pede para listarmos as pecas de um determinado tipo num determinado conjunto.

```
395    void PrintPartsC(Sets * SetsList, Parts * PartsList, PS * PSList)
```

Com esta função somos capazes de cumprir o objetivo 3 onde é necessário listar todos as informações de todas as peças necessárias para construir um determinado conjunto.

```
455    void PrintPartsTotalS(Sets * SetsList, Parts * PartsList, PS * PSList)
```

Aqui, procuramos responder ao objetivo 4, um dos mais fáceis, objetivo este que pede para simplesmente calcular a quantidade total de peças em stock.

```
474    void PrintPartsTotalC(Sets * SetsList, Parts * PartsList, PS * PSList)
```

Esta função recolhe o conjunto que o utilizador quer para a seguir poder listar todas as peças necessárias, incluindo todas as informações dessas mesmas peças e também calcular o número de peças necessárias para construir o conjunto, isto tudo para cumprir o objetivo 5.

```
537    void PrintPartsMU(Sets * SetsList, Parts * PartsList, PS * PSList)
```

Esta função tem um certo grau de dificuldade, pois para responder ao objetivo 6 é necessário percorrer pelo ficheiro *parts_sets.tsv* e recolher quantas vezes cada uma das peças se repete nesse ficheiro. Cada vez que alguma peça se repete mais vezes que as anteriores torna-se a mais repetida, isto vai acontecendo até ao final do ficheiro onde apenas uma peça será a mais usada.

```
583    void PrintSetsWC(Sets * SetsList, Parts * PartsList, PS * PSList)
```

Provavelmente o objetivo mais complexo de todos, o objetivo 7 requere que nós acedemos ao ficheiro *parts_sets.tsv* recolher as quantidades de todas as peças de cada conjunto, verificando ao mesmo tempo no ficheiro *parts.tsv* se essa quantidade é igual ou menor ao stock dessa mesma peça. Se num conjunto inteiro verificar-se a possibilidade de stock para todas as peças esse conjunto irá ser listado.

```
643    void ChangeSP(Sets * SetsList, Parts * PartsList, PS * PSList)
```

Esta função cumpre o pedido de poder mudar o stock de uma peça. Pedimos para o utilizador escolher uma peça e depois deixamo-lo mudar a variável *stock* consoante ele quer. Depois disto listamos a peça outra vez com o seu novo stock.

```
711    void ChangeSPC(Sets * SetsList, Parts * PartsList, PS * PSList)
```

Pedimos ao utilizador um stock, depois de escolhido calculamos o número de peças necessárias para construí-lo. Com esse número adicionamos ao stock total já calculado numa função anterior.

```
789    void DeletePC(Sets * SetsList, Parts * PartsList, PS * PSList)
```

O utilizador escolhe um tipo de peça e ao ser escolhido esse tipo, todas as peças relativas a esse tipo não serão listadas na próxima vez que o utilizador quiser ver as peças.

```
863    void DeleteST(Sets * SetsList, Parts * PartsList, PS * PSList)
```

Exatamente o mesmo pensamento que a função anterior só que aplicado aos temas de conjuntos.

```
937    void MenuL(Sets * SetsList, Parts * PartsList, PS * PSList)
```

Esta função é o menu onde o utilizador pode escolher todas as opções de listagem de dados.

```
1006    void MenuE(Sets * SetsList, Parts * PartsList, PS * PSList)
```

Já este menu contém as opções de troca e mudança de dados.

```
1056    void MenuP(Sets * SetsList, Parts * PartsList, PS * PSList)
```

Aqui deixamos o utilizador escolher se quer editar os dados dos ficheiros ou listá-los.

4. Bibliografia

Os dados utilizados para este trabalho prático foram obtidos a partir de uma base de dados pública, disponível em https://rebrickable.com/.

Eventuais dúvidas que apareceram ao longo do projeto forma resolvidas junto do professor ou com que recurso ao site Stack Overflow.