Trabalho prático 2 – Filas de prioridade

1) Informação geral

O trabalho prático 2 consiste na implementação de uma parte de uma biblioteca de funções para manipulação de **árvores AVL** e **filas de prioridade** em C.

Este trabalho deverá ser feito de forma autónoma por cada grupo durante a aula prática e completado fora das aulas até à data limite estabelecida. A consulta de informação nas diversas fontes disponíveis é aceitável. No entanto, o código submetido deverá ser apenas da autoria dos elementos do grupo e quaisquer cópias detetadas serão devidamente penalizadas. A incapacidade de explicar o código submetido por parte de algum elemento do grupo implicará também numa penalização.

O prazo de submissão na página de Programação 2 do Moodle é 6 de Maio às 21:00.

2) Objetivo

Este trabalho compreende a implementação de um armazém para guardar um conjunto de artigos por categorias. Pretende-se utilizar este armazém e respetivos stocks para elaborar montras de supermercados.

Para tal, espera-se que complete implementações das bibliotecas de arvores AVL e Heaps, fornecidas no moodle, com protótipos de utilização presentes nos ficheiros disponibilizados com o trabalho.

3) Implementação disponibilizada

O ficheiro *zip* PROG2_1819_T2 contém os ficheiros necessários para a realização deste trabalho, nomeadamente:

- `market.h` inclui as declarações das funções a implementar - **não deve ser alterado**
- `market.c` ficheiro onde deverão ser implementadas as funções da biblioteca
- `market-teste.c` inclui os testes feitos à biblioteca - **não deve ser alterado**

A estrutura de dados `elemento_t` compreende o registo base da biblioteca e tem a seguinte declaração:

```
typedef struct elemento_
{
    char nameItem[100];
    char expirationDate[11];
    int qty;
    int sellRate;
    float priorityVal;
} elemento_t;
```

Nesta estrutura é guardada a representação de um artigo que deverá ser incluído em cada posição de cada Heap, para a gestão de artigos de cada categoria. Uma variável do tipo `elemento_t` incorpora um *string* de designação do produto (`nameItem`), uma *string* com a data de expiração (`expirationDate) em formato YYYY-MM-DD`, a quantidade existente em unidades desse artigo (`qty`), o número médio de unidades do artigo vendidas por dia (`sellRate`), e o valor de prioridade associado ao artigo específico (`priorityVal`).

Os apontadores para os artigos deverão estar guardados em Max-Heaps para rápido acesso. Por sua vez, a Heap tem a seguinte definição:

```
typedef struct
{
      int tamanho;
      int capacidade;
      elemento_t** elementos;
} heap;
```

Pretende-se ainda guardar cada conjunto de diferentes artigos (categorias) de forma eficiente. Para tal deverá usar uma árvore AVL que guardará em cada nó uma categoria de artigos. Adicionalmente, cada categoria deverá incluir a representação de uma árvore Heap com os respetivos artigos dessa categoria.

Os registos associados a estas componentes têm a seguinte descrição:

```
typedef struct no_avl_
{
    category_t * categ;
    struct no_avl_ * esquerda;
    struct no_avl_ * direita;
    int altura;
} no_avl;


typedef struct category_
{
    char* categName;
    heap* itemTree;
} category_t;
```

Cada `no_avl` inclui campos de apontadores para os nós filhos da `esquerda` e `direita`, contudo deverá manter também a altura actual do nó e um registo da categoria (`categ`) para que aponta. A ordenação dos nós deverá ser feita por ordem alfabética.

Um registo do tipo `category_t`, contém a sua designação `(str)` e o apontador para a Heap `(itemTree)` com os artigos da respetiva categoria.

Além do mais, existe uma estrutura (`arvore_avl`) contendo unicamente o apontador para o nó `raiz` da árvore.

Ambas as estruturas de ordenação contêm algumas das funções implementadas, deixando as restantes assinaladas para o desenvolvimento/adaptação durante este trabalho.

4) Recursos disponibilizados

O ficheiro armazem.txt contém a listagem das várias encomendas que foram recebidas pelo armazém, que poderão ser utilizadas para a elaboração das montras. Cada amostra presente no ficheiro segue o seguinte formato:

<itenName_chars> <itemCategory_chars> <Qty_integer> <itensPerDay_int>
<ExpirationDate_charYYYY-MM-DD>

Exemplos:
Alface_Titanic_InterFrutasLda Frescos 400 31 2019-05-15
Motoserra_Gasoleo_LumberRUs Ferramentas 40 1 2050-12-30

5) Exercícios / Implementação do trabalho

O trabalho consiste em implementar um conjunto de funções que permitem a realização de diversas operações sobre os uma árvore AVL e Heaps, juntamente com tarefas específicas do problema descrito. Estas funções estão declaradas no ficheiro `market.h` e deverão ser implementadas no ficheiro `market.c`.

5.1) Criação de novo elemento e métrica de prioridade

Implemente a função *elemento_novo* que deverá receber as informações necessárias para gerar uma nova instância do tipo *elemento_t*, seguindo o protótipo:

elemento_t* elemento_novo (const char* nameItem, const char* expDate, int qty, int sellRate)

Os argumentos de entrada compreendem *strings* de nome e data de expiração do artigo (*nameItem*), e inteiros correspondendo à quantidade e taxa de venda de um artigo.

Cada artigo, no entanto, deverá ser introduzido no sistema com uma prioridade de venda. A métrica de prioridade de despacho dos artigos deverá ser calculada pela equação seguinte:

$$m = \frac{1}{expirationDate - CURDATE + \frac{1000}{sellRate}}$$

A função correspondente que deverá implementar segue o protótipo:

float calcMetrica(elemento_t* elem);

Pode-se calcular uma métrica de prioridade baseada no número de dias entre a data de expiração e o dia atual (fixado no código), contabilizando número de artigos vendidos num dia.

Notas:

1. Pode usar as funções e tipos de variável da biblioteca <time.h> para manipulação de datas, nomeadamente: difftime(), mktime(), o registo tm, variáveis time_t, entre outras.
2. Considere também que o exercício se passa na data definida como string em CURDATE.
3. Para os efeitos do exercício pode ignorar os instantes de tempo inferiores ao dia, i.e., pode considerar que o dia atual e datas de expiração ocorrem sempre às 0:00 horas.

5.2) Implementação da fila de prioridade (Heap)

Implemente o código de Heaps de forma a garantir o seu funcionamento para o registo `elemento_t`. As funções que deverão implementar têm os seguintes protótipos:

1. **heap * heap_nova** (int capacidade);
   cria uma heap nova, vazia, de tamanho máximo `capacidade`*, e retorna o apontador para a heap ou Null em caso de erro.*

2. **elemento_t * heap_remove** (heap* h);
   remove o elemento corresponde de prioridade máxima, retornando-o se possível ou retornando Null se não o for.
3. **void heap_apaga** (heap *h)
   elimina todos os elementos da heap e remove-a da memória.

5.3) Implementação árvores AVL

Adapte o código de árvores AVL de forma a garantir o seu funcionamento para o registo `category_t`. As funções que deverá implementar têm os seguintes protótipos:

1. **no_avl * avl_insere** (no_avl* no, category_t * categ);
   adiciona um elemento category_t no nó AVL indicado, retornando o apontador para o nó ou Null em caso de erro
2. **no_ avl * avl_pesquisa** (no_ avl * no, const char* categStr)
   procura o nó da categoria pedida pedido, retornando o seu apontador ou Null caso não esteja presente

5.4 Adição de artigo

Implemente a função que adiciona um artigo ao armazém com o seguinte protótipo:

int artigo_adiciona (arvore_avl *avl, elemento_t * elem, const char* categName, int capCateg);

Onde se deve:

- adicionar uma categoria (*categName*) se esta não existir;
- adicionar um artigo com a específica data de validade (*elem*) se não existir nessa categoria;
- deverá verificar se a tentativa de adição ultrapassar a capacidade máxima da heap (*capCateg*), e, como tal, considerar um insucesso se a adição for cancelada;
- retornar 1 em caso de sucesso e 0 se insucesso.

Notas:

1. Tome em atenção que o elemento a introduzir deverá estar completamente preenchido antes da função *artigo_adiciona* ser chamada (i.e., a prioridade deverá ser calculada anteriormente).
2. Tem a garantia de que as encomendas presentes no ficheiro de entrada nunca serão iguais (i.e., nunca serão o mesmo artigo, contendo a mesma prioridade).

5.5) Montagem de Montra

Implemente a função que crie uma seleção de artigos para dispor numa montra de determinada categoria. A função deve seguir o protótipo:

elemento_t criar_montra** (arvore_avl* avl, char* categName, int numPorItem, int totalItens, int* tamanho_array);

Onde se deve:

- requisitar cada artigo de cada prazo de validade (elemento) até ao valor de *numPorItem*, para criar uma montra da categoria *categName*, perfazendo uma contagem final de *totalItens* unidades;
- proceder à requisição de artigos por ordem de valor de prioridade, descontando o número de unidades requisitadas por artigo no armazém (heap respetiva).
- descontar só o número de artigos retirados do armazém para a respetiva validade e fazer isso refletir no total de artigos que faltam requisitar;
- retornar o tamanho do *array*, através do argumento de entrada *tamanho_array*;
- cancelar a requisição de todos os artigos para essa montra, caso não se consiga perfazer o número de unidades *totalItens*.
- Exemplo:
    - considere o seguinte estado do armazém na categoria dos frescos:
      - 4 x cebola 2019-05-15
      - 6 x alho 2019-05-18
      - 8 x alface 2019-05-22
      - assuma que têm a mesma sellRate
    - Considere o pedido: *criar_montra*(avltree, "frescos", 5, 10);
    - O resultado deverá ser um array de *elemento_t* contendo 4 cebolas, 5 alhos e 1 alface.
    - O armazém *avltree* deverá manter 1 alho e 7 alfaces, daquelas datas e o artigo cebolas deverá ser eliminado do armazém.

Em caso de sucesso, a função deverá descontar os artigos requisitados e retornar um array com os artigos descontados. Em caso de insucesso, o retorno deverá ser NULL, mantendo o armazém com todos os elementos.

6) Teste da biblioteca de funções

É fornecido um ficheiro `market-teste.c` que permite realizar um conjunto de testes à biblioteca desenvolvida. Note que os testes não são exaustivos, não verificando, por exemplo, *memory leaks* e por isso os testes devem ser considerados apenas como um indicador de uma aparente correta implementação das funcionalidades esperadas.

Inicialmente o programa `market-teste` quando executado apresentará o seguinte resultado:

```
verifica_elemento_novo: Erro na função elemento_novo (retornou NULL)
ERRO: 1 erros encontrados em teste_elemento_novo_e_metrica

verifica_heap_nova: Erro na função heap_nova (retornou NULL)
ERRO: 1 erros encontrados em teste_heap

ERRO: heap_insere retornou NULL
Os testes nao podem prosseguir devido a ERROS fatais.
```

Note que é fortemente aconselhável que as funções `descritas` sejam implementadas pela ordem indicada neste enunciado.

Depois de todas as funções corretamente implementadas o resultado do programa apresentará o seguinte resultado:

```
OK: teste_elemento_novo_e_metrica passou

OK: teste_heap passou
```

```
OK: teste_avl passou

OK: teste_artigo_adiciona_e_criar_montra passou

FIM DOS TESTES: Todos os testes passaram
```

7) Ferramenta de desenvolvimento

A utilização de um IDE ou do Visual Studio Code é aconselhável no desenvolvimento deste trabalho uma vez que permite fazer depuração de uma forma mais eficaz. Poderá encontrar informações sobre a utilização do Visual Studio Code num breve tutorial disponibilizado no Moodle.

8) Avaliação

A classificação do trabalho é dada pela avaliação feita à implementação submetida pelos estudantes, mas também pelo desempenho dos estudantes na aula dedicada a este trabalho. A classificação final do trabalho (T2) é dada por:

T2 = 0.7 Implementação + 0.1 Memória + 0.2 Desempenho

A classificação da implementação é essencialmente determinada por testes automáticos adicionais. No caso da implementação submetida não compilar, esta componente será de 0%.

A gestão de memória também será avaliada, sendo considerados 3 patamares: 100% nenhum *memory leak*, 50% alguns *memory leaks* mas pouco significativos, 0% muitos *memory leaks.*

O desempenho será avaliado durante a aula e está dependente da entrega do formulário "Preparação do trabalho" que se encontra disponível no Moodle. A classificação de desempenho poderá ser diferente para cada elemento do grupo.

9) Submissão da resolução

A submissão é apenas possível através do Moodle e até à data indicada no início do documento. Deverá ser submetido um ficheiro *zip* contendo:

- o ficheiro **market.c** com as funções implementadas
- um ficheiro **autores.txt** indicando o nome e número dos elementos do grupo

Nota importante: apenas as submissões com o seguinte nome serão aceites: `T2_G<numero_do_grupo>.zip`. Por exemplo, `T2_G999.zip`
Caso não se recorde do número do seu grupo a listagem encontra-se na secção introdutória da página do Moodle.

Bom trabalho!