Trabalho prático 2 – Implementação de biblioteca de grafos e de filas de prioridade

1) Informação geral

O objetivo do trabalho prático 2 é avaliar a capacidade do estudante para analisar um problema algorítmico, utilizando estruturas derivadas das apresentadas na unidade curricular, e implementar em C uma solução correta e eficiente.

Este trabalho deverá ser feito de forma autónoma por cada grupo na 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 no Moodle de Estruturas de Dados e Algoritmos é **23 de maio às 13h30**.

2) Descrição

Estás encarregue de gerir a informação da Rede Ferroviária de Portugal. Deves desenvolver a lógica de pesquisa e manipulação de informação sobre as estações, as linhas e outras informações.

A estrutura de dados que guarda as informações da rede é um grafo direcionado (ver "grafo.h" para mais detalhes). Este deve ser implementado por um vetor de adjacências associado a cada nó, com arestas pesadas por valores distintos, consoante a pesquisa desejada.

1. Vais precisar de implementar a biblioteca referente ao grafo. Os registos essenciais para a construção do grafo são:

```
Estrutura grafo:

  int tamanho          /* número de posições válidas de 'nos' */
  no_grafo **nos       /* vetor de apontadores para 'no_grafo' */

Estrutura no_grafo ( estações ):

  char *estacao        /* string com nome da estação */
  int tamanho          /* número de posições válidas de 'arestas' */
  aresta_grafo **arestas    /* vetor de apontadores para 'arestas' */
  char distrito[30]    /* string com o nome do distrito */
  double p_acumulado; /* peso acumulado,a ser usado no Dijkstra */
  struct no_grafos *anterior;  /* nó anterior,a ser usado no Dijkstra */
```

Estrutura **aresta_grafo (troços):**

```
 char *codigo      /* código do troço */
 char *linha       /* nome da linha ferroviária */
 int distancia   /* distância deste troço de linha em metros*/
 int inspecao      /* ano da última inspeção */
 no_grafo *destino /* apontador para o nó de destino */
```

Nota importante: Cada troço, que liga duas estações, existe no vetor de apontadores para 'arestas' desses dois nós (estações). A função grafo_importa, **que já está implementada**, cria o grafo a partir de um ficheiro e trata de inserir cada troço nos dois nós.

Para a sua construção deverás implementar as seguintes funções, na biblioteca grafo:

`grafo *grafo_novo();`

Cria grafo de tamanho 0, retorna um apontador para o grafo criado ou NULL em caso de insucesso.

`no_grafo *no_novo(char *estacao, char *distrito);`

Cria um novo nó, retorna um apontador para o nó criado ou NULL em caso de insucesso.

`int no_insere(grafo *g, no_grafo *no)`

Insere um nó 'no', no grafo 'g', Retorno 0 se for bem-sucedido, 1 se for repetido (ou seja, já exista uma estação com o mesmo nome) e -1 em caso de erro.

`int cria_aresta(no_grafo *origem, no_grafo *destino, char *codigo, char *linha, int distancia, int ano);`

Cria uma ligação entre a estação de origem e a estação de destino. Retorno 0 se for bem-sucedido, 1 se for repetido (ou seja, já exista uma aresta entre origem e destino) e -1 em caso de erro.

`int aresta_apaga(aresta_grafo *aresta)`

Elimina o espaço de memória ocupado por 'aresta'. Retorna zero em caso de sucesso e -1 se insucesso.

`no_grafo *no_remove(grafo *g, char *estacao)`

Remove do grafo e retorna apontador para o nó referente à 'estacao'. Note-se que se devem remover também todas as arestas que têm esse nó como destino. Retorna NULL em caso de insucesso.

`int no_apaga(no_grafo *no)`

Elimina o espaço de memória ocupado por 'no', bem como pelas arestas com origem nele. Retorna zero em caso de sucesso e -1 se insucesso.

`int grafo_apaga(grafo *g)`

Apaga o grafo 'g', eliminando-o e libertando toda a memória associada. Retorna zero em caso de sucesso e -1 se insucesso.

2. Queres dar funcionalidades úteis ao serviço, permitindo que se pesquise o grafo em busca de certas propriedades. Implementa as seguintes funções:

```
no_grafo *encontra_no(grafo *g, char *estacao)
```

Pesquisa no grafo 'g' o nó referente a determinada 'estacao'. Retorna o apontador para o nó. Retorna NULL se o nó não for encontrado ou em caso de erro.

```
no_grafo *encontra_troco(grafo *g, char *codigo, int
    *aresta_pos)
```

Pesquisa no grafo 'g' o troço com determinado 'codigo', retornando por referência através do argumento 'aresta_pos' a posição da respetiva aresta nos vetores de arestas do nó retornado. Visto que o troço existe duas vezes, como consequência existem dois nós que têm esse troço. Devem escolher o nó que tiver o nome da estação mais baixo alfabeticamente.

Retorna ainda o apontador para o nó de origem do troço. Retorna NULL se o troço não for encontrado ou em caso de erro.

```
no_grafo **pesquisa_avancada(grafo *g, char *destino, int
    *n)
```

Retorna um vetor de apontadores para todos os nós do grafo 'g' que tenham troços diretos a chegar ao nó com o nome 'destino'. Retorna NULL se não forem encontrados troços ou em caso de erro. O tamanho do vetor deve ser retornado por referência através do argumento 'n'.

```
no_grafo **estacoes_distrito (grafo *g, char *distrito,
    int *n)
```

Retorna um vetor de apontadores para todos os nós do grafo 'g' que façam parte do distrito ' distrito' . Retorna NULL se não forem encontradas estações ou em caso de erro. O tamanho do vetor deve ser retornado por referência através do argumento 'n'.

```
no_grafo **estacoes_linha (grafo *g, char *estacao, char
    *linha, int *n)
```

Retorna um vetor de apontadores para todos os nós do grafo 'g' que façam parte da mesma 'linha' a partir de uma 'estacao'(um nó faz parte de uma linha se um troço dessa linha partir desse nó). Retorna NULL se não forem encontrados troços ou em caso de erro. O tamanho do vetor deve ser retornado por referência através do argumento 'n'.

```
int metros_linha (grafo *g, char *linha, char *origem)
```

Retorna o comprimento total em metros de uma determinada 'linha' que começa na estação 'origem'(ou seja, percorrendo a linha até que chegue a uma estação sem mais troços nessa linha . Retorna -1 em caso de insucesso.

```
no_grafo **trajeto_mais_curto(grafo *g, char *origem, char
    *destino, int *n)
```

Calcula o trajeto mais curto entre as estações de 'origem' e 'destino', usando o algoritmo de Dijkstra. Retorna um vetor de apontadores para todos os nós do grafo 'g' desse trajeto. Retorna NULL se não for encontrada nenhuma combinação de troços válida ou em caso

de erro. O tamanho do vetor deve ser retornado por referência através do argumento 'n'.

***Nota:** Para o algoritmo de Dijkstra podes utilizar as funções **já implementadas** no ficheiro heap.c, que contém uma heap adaptada a este problema (cada elemento da heap contém um 'no_grafo' e um 'aresta_grafo', para este caso só precisam do no_grafo, por isso colocam aresta_grafo a NULL).*

```
Estrutura elemento:

  int prioridade       /* prioridade do elemento (ano de construção)*/
  aresta_grafo *troco          /* apontador para o troço */
  no_grafo *estacao          /* apontador para o no*/


Estrutura heap:
  int tamanho         /* número de elementos no vetor */
  int capacidade      /* tamanho máximo de elementos no vetor */
  elemento ** elementos /* vetor de apontadores para elementos*/
```

3. Vais precisar de complementar a biblioteca referente à heap.

`heap *heap_carrega(grafo *g, int capacidade)`
Cria e preenche uma heap, com uma dada 'capacidade' e com o conteúdo do grafo 'g'. Cada elemento da heap deve ser composto pelos troços(arestas), o campo estação deve ficar a NULL e a prioridade é o ano da inspeção. Retorna o apontador para a heap criada se bem-sucedido ou NULL em caso de impossibilidade por falta de capacidade ou outro erro.

`aresta_grafo **inspecao_trocos (heap *h, int dias, int *n)`
Pretende-se saber quais são os próximos troços a serem inspecionados, nos próximos 'dias', sendo a ordem de inspeção definida pela antiguidade do ano da última inspeção(logo o primeiro a ser inspecionado é o que tem o ano mais baixo). Sabendo que por dia consegue-se inspecionar 10Km, o objetivo é descobrir a ordem que dessa inspeção. A heap 'h' que vem como parametro foi criada pela função anterior(heap_carrega).
Retorna um vetor de apontadores de troços (arestas_grafo). Esses troços devem estar pela ordem do mais antigo ao mais recente (ano de inspeção). Esses troços devem ser retirados da heap. Retorna NULL se não forem encontrados troços ou em caso de erro. O tamanho do vetor deve ser retornado por referência através do argumento 'n'.
Obs: Quando um troço tem por exemplo 32 Km, consegue-se inspecionar em teoria 3.2 dias, arredonda-se sempre para cima logo fica 4 dias. Se tivermos 4 dias para inspecionar e o troço mais antigo tiver 55 Km não se inspeciona nada. Ou seja, se o próximo troço a ser inspecionado não conseguir ser inspecionado completamente, não se faz mais nada.

Para todas as funções a implementar, uma descrição da própria, dos seus argumentos e retornos poderá ser consultada nos próprios *header files* "grafo.h" e "heap.h".

3) Avaliação

A classificação do trabalho é dada pela avaliação feita à implementação submetida pelos estudantes. A classificação final do trabalho (T2) é dada por:

T2 = 0.80 Implementação + 0.20 Memória

A classificação da implementação será essencialmente determinada por testes automáticos adicionais. Serão avaliados os outputs produzidos pelo código dos estudantes. No caso da implementação submetida não compilar, esta componente será de 0%.

A gestão de memória também será avaliada, tendo a respetiva cotação parcial a contemplar 3 patamares: 100% nenhum *memory leak*, 50% alguns, mas pouco significativos, 0% muitos *memory leaks.*

4) Teste em servidor

Em breve será disponibilizado um servidor para que possam testar o vosso código durante o desenvolvimento. O código submetido neste servidor NÃO SERÁ AVALIADO. Apenas a submissão via moodle é válida para efeitos de avaliação.

4) 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:

- ficheiros **grafo.c e heap.c**
- 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_G9999.zip

5) Exemplo de resultados esperados

A biblioteca pode ser testada executando o programa `linhas-teste.c`. Existe um teste por cada função a implementar e que determina se essa função tem o comportamento esperado. Note que os testes não são exaustivos. Por isso, os testes devem ser considerados apenas como um indicador de uma aparente correta implementação das funcionalidades esperadas.

Estando as funções corretamente implementadas, o programa `linhas-teste` quando executado deverá apresentar o seguinte resultado:

```
IINICIO DOS TESTES


TESTES DO GRAFO


...verifica_grafo_novo: grafo novo nao e' NULL (ok)

...verifica_grafo_novo: Tamanho do grafo novo coincide com o esperado (= 0) (ok)
```

...verifica_grafo_novo: O apontador para o vetor de apontadores de nos e´ o esperado (= NULL) (ok)

OK: verifica_grafo_novo passou

...verifica_no_novo: no novo nao e' NULL (ok)

...verifica_no_novo: Campo estacao coincide com o esperado (= melgaço) (ok)

...verifica_no_novo: Campo distrito do novo no coincide com o esperado (= Viana do Castelo) (ok)

...verifica_no_novo: Campo tamanho do novo no coincide com o esperado (= 0) (ok)

...verifica_no_novo: O apontador para o vetor de apontadores de arestas e´ o esperado (= NULL) (ok)

OK: verifica_no_novo passou

...verifica_no_insere: inseriu o 1º e 2º nos nas posicoes corretas (ok)

...verifica_no_insere: campo tamanho do grafo coincide com o esperado (= 2) (ok)

OK: verifica_no_insere passou

...verifica_cria_aresta: codigo da aresta no no' 0 na posicao 0 coincide com o esperado (=T01) (ok)

...verifica_cria_aresta: numero de aresta no no´ 0 coincide com o esperado (= 2) (ok)

...verifica_cria_aresta: aresta já existente (ok)

OK: verifica_cria_aresta passou

...verifica_no_remove (teste de estacao inexistente): não removeu nenhum nó (ok)

...verifica_no_remove (teste de estacao válida): primeiro no´ coincide com o esperado (=campanhã) (ok)

...verifica_no_insere: campo tamanho do grafo coincide com o esperado (= 1) (ok)

OK: verifica_no_remove passou

...verifica_no_apaga: apagou com sucesso (ok)

OK: verifica_no_apaga passou

...verifica_encontra_troco: encontrou com sucesso (OK)

OK: verifica_encontra_troco passou

...verifica_pesquisa_avancada: encontrou com sucesso (OK)

OK: verifica_pesquisa_avancada passou

...verifica_estacoes_distrito: encontrou com sucesso (=71) (OK)

OK: verifica_estacoes_distrito passou

...verifica_estacoes_linha: encontrou com sucesso (=9) (OK)

OK: verifica_estacoes_linha passou

...verifica_metros_linha: encontrou com sucesso (=135903) (OK)

OK: verifica_metros_linha passou

...verifica_trajeto_mais_rapido: encontrou com sucesso (OK)

OK: verifica_trajeto_mais_rapido passou

...verifica_heap_carrega (número de elementos): heap_carrega criou(=540) elementos (OK)

...verifica_heap_carrega (1º elemnto): 1º elemnto (='L003') (OK)

OK: verifica_heap_carrega passou

...verifica_inspecao_trocos (número de arestas): verifica_inspecao_trocos criou(=5) arestas (OK)

...verifica_inspecao_trocos (1º elemnto que ficou na heap): 1º aresta (='L005') (OK)

OK: verifica_inspecao_trocos passou

FIM DOS TESTES: Todos os testes passaram