Trabalho prático 2 – Plataforma FlightDreams

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 Programação 2 é 4 de junho às 21:00.

2) Descrição

Estás encarregue de tratar do back-end de uma plataforma online de reserva de voos - Flight Dreams. Deves desenvolver a lógica de pesquisa e manipulação de informação sobre os horários de voo, origens, destinos e outras informações, de forma a ficarem disponíveis para o site as usar e apresentar. Um colaborador anterior já iniciou a tarefa.

A estrutura de dados que guarda as informações dos voos está já implementada, e tem por base um grafo direcionado (ver "grafo.h" para mais detalhes). Este deve ser implementado por um vetor de adjacências, com arestas pesadas por valores distintos, consoante a pesquisa desejada.

Deves implementar:

1. Vais precisar de completar 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:

  char *cidade         /* string com nome da cidade */
  int tamanho          /* número de posições válidas de 'arestas' */
  aresta_grafo **arestas    /* vetor de apontadores para 'arestas' */
  double p_acumulado  /* peso acumulado, Dijkstra */
  no_grafo *anterior  /* apontador para nó anterior, Dijkstra */
  data *dataatualizada;/* apontador para a data, Dijkstra */
```

```
Estrutura aresta_grafo:

  char *codigo      /* código do voo */
  char *companhia   /* nome da companhia que opera o voo */
  data partida      /* dia e hora da partida da cidade de origem  */
  data chegada      /* dia e hora da chegada na cidade de destino  */
  double preco      /* preço do voo */
  int lugares       /* número de lugares disponíveis para a ligação */
  no_grafo *destino /* apontador para o nó de destino do voo */
```

Para a sua construção deverás implementar as seguintes funções:

`no_grafo *no_remove(grafo *g, char *cidade)`
Remove e retorna apontador para o nó do grafo 'g', referente à 'cidade', corrigindo o restante registo de forma adequada. Retorna NULL em caso de insucesso.

`int aresta_apaga(aresta_grafo *aresta)`
Elimina o espaço de memória ocupado por 'aresta'. Retorna zero em caso de sucesso e -1 se insucesso.

`int no_apaga(no_grafo *no)`
Elimina o espaço de memória ocupado por 'no'. Retorna zero em caso de sucesso e -1 se insucesso.

`void grafo_apaga(grafo *g)`
Apaga o grafo 'g', eliminando-o e libertando toda a memória associada.

***Nota:** As variáveis do tipo 'data' são instâncias de um registo de inteiros, originalmente definido na biblioteca "time.h". Sugerimos que se familiarize com elas antes de desenvolver o seu código.*

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

`no_grafo *encontra_voo(grafo g*, char *codigo, int *aresta_pos)`
Pesquisa no grafo 'g' o voo com determinado 'codigo', retornando por referência através do argumento 'aresta_pos' a posição da respetiva aresta no vetor de arestas do nó retornado. Retorna ainda o apontador para o nó de origem do voo. Retorna NULL se o voo não for encontrado ou em caso de erro.

`no_grafo **pesquisa_avancada(grafo *g, char *destino, data chegada, double preco_max, int *n)`
Retorna um vetor de apontadores para todos os nós do grafo 'g' que tenham voos directos a chegar ao nó com o nome 'destino', no dia indicado em 'chegada' e com o preço máximo de 'preco_max', inclusive. No vetor retornado não deve haver duplicados.

Retorna NULL se não forem encontrados voos ou em caso de erro. O tamanho do vetor deve ser retornado por referência através do argumento 'n'.

Nota: *Sugerimos que usufruas das funções "mktime" e "difftime", da biblioteca "time.h" para comparar mais facilmente as datas.*

3. Permite a pesquisa em transportes mais complexos, com voos que envolvem transbordos:

```
no_grafo **trajeto_mais_rapido(grafo *g, char *origem,
    char *destino, data partida, int *n)
```

Calcula o trajeto que permite chegar o mais cedo possível da cidade 'origem' para a 'destino', considerando os voos a partir do dia 'partida', 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 voos válida ou em caso de erro. O tamanho do vetor deve ser retornado por referência através do argumento 'n'.

```
no_grafo **menos_transbordos(grafo *g, char *origem,
    char *destino, data partida, int *n )
```

Calcula o trajeto com menor número de transbordos entre as cidades de 'origem' e 'destino', considerando os voos a partir do dia 'partida', usando o algoritmo de Dijkstra. Considera que todos os transbordos são válidos (não deves verificar se a partida de um dado voo ocorre depois da chegada do voo anterior). Retorna um vetor de apontadores para todos os nós do grafo 'g' desse trajeto. Retorna NULL se não for encontrada nenhuma combinação de voos válida ou em caso de erro. O tamanho do vetor deve ser retornado por referência através do argumento 'n'.

Nota: *De novo, sugerimos que usufruas das funções "mktime" e "difftime", da biblioteca "time.h" para comparar mais facilmente as datas. 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').*

4. Periodicamente o sistema deve atualizar o grafo mediante a alteração de lugares disponíveis nos voos. Implementa a função seguinte para ler do ficheiro atualizado e remover os voos que ficaram sem lotação disponível.

```
aresta_grafo **atualiza_lugares(char *ficheiro, grafo
    *g, int *n )
```

Adquire as informações atualizadas e corrige o grafo. Remove do grafo e retorna as arestas/voos cujos lugares desceram para zero por via de um vetor de apontadores para aresta_grafo. O tamanho do vetor deve ser retornado por referência através do argumento 'n'. Retorna NULL em caso de erro.

Nota: *Os dados relativos à actualização dos lugares disponíveis estão disponíveis no ficheiro* ***"flightPlanUpdate.txt"*** *e estão no formato: <**código de voo**>, <**lugares disponíveis**>.*

5. Pretendes tornar a pesquisa no grafo um pouco mais rápida. Como tal, resolves implementar uma tabela de dispersão para alojar os nós e facilitar o seu acesso, sem comprometer o grafo original. A tabela de dispersão deve ser de encadeamento aberto e inclui o registo '`tabela_dispersao`' na sua implementação. Use o nome da cidade como chave da tabela.

```
Estrutura tabela_dispersao:

  hash_func *hfunc    /* apontador para função de dispersão */
  sond_func *sfunc     /* apontador para função de sondagem */
  int capacidade       /* número de posições alocadas de 'nos' */
  int tamanho          /* número de posições preenchidas de 'nos' */
  no_grafo **nos       /* vetor de apontadores para nó */
  int *estado_celulas  /* vetor de indicadores de estado 0:vazio,
1:válido, -1:removido */
```

Adicionalmente, deves implementar a biblioteca que permite a utilização da tabela:

`tabela_dispersao *tabela_nova(int capacidade, hash_func *hfunc, sond_func *sfunc)`

Cria uma instância de 'tabela_dispersao' de certa 'capacidade', que utilize a função de dispersão 'hfunc' e a função de sondagem 'sfunc'. Retorna o apontador para a tabela criada ou NULL em caso de erro.

`int tabela_adiciona(tabela_dispersao *td, no_grafo *entrada)`

Adiciona a 'entrada' à tabela 'td'. Retorna o índice do nó adicionado se foi bem sucedido ou -1 em caso contrário.

`int tabela_remove(tabela_dispersao *td, no_grafo *saida)`

Remove o nó 'saida' da tabela 'td', não desalocando a memória do mesmo. Retorna 0 se tiver sucesso e -1 se ocorrer algum erro.

`void tabela_apaga(tabela_dispersao *td)`

Retira todos os valores da tabela 'td', ficando a tabela vazia e apaga o registo 'td', libertando toda a memória.

`int tabela_existe(tabela_dispersao *td, const char *cidade)`

Verifica se 'cidade' existe na tabela. Retorna o índice do nó encontrado na tabela se bem sucedido ou -1 em contrário.

`tabela_dispersao *tabela_carrega(grafo *g, int capacidade)`

Cria e preenche uma tabela de dispersão, com a dada 'capacidade' e o conteúdo do grafo 'g'. Retorna o apontador para a tabela criada se bem sucedido ou NULL em caso de impossibilidade por falta de capacidade ou outro erro.

***Nota:** Considere as funções `hash_krm` e `sond_rh` como implementações para as funções de dispersão e sondagem, respetivamente.*

`unsigned long hash_krm(const char* chave, int tamanho)`

Função de dispersão. Retorna o índice correspondente à 'chave'.

`unsigned long sond_rh(int pos, int tentativas, int tamanho)`

Função de resolução de colisões, implementada por sondagem quadrática em que o passo é o número de 'tentativas', partindo do índice 'pos'. Calcula e retorna um índice alternativo, correspondente à 'tentativa'. Deve usar esta função sempre que se verificar uma colisão.

6. O teu conhecimento de estruturas de dados evoluiu com o desenvolver do protótipo anterior. Desenvolve uma estrutura ou combinação de estruturas que permita obter um melhor desempenho global de pesquisa e remoção de voos. A tua estrutura deverá lidar com um rácio de 1 criação da estrutura para 100000 pedidos. As pesquisas concentrar-se-ão em voos directos (i.e. sem transbordos), baseadas no critério preço.

   Desenvolve a biblioteca para o que precisas, podendo criar as funções e estruturas que achares adequadas, mas usando o seguinte conjunto de interfaces para interagir com o código de teste:

`estrutura *st_nova()`

Cria e inicializa a estrutura que possa alojar as instâncias pretendidas. Retorna o apontador para a estrutura se bem-sucedido ou NULL em caso contrário.

`int st_importa_grafo(estrutura *st, grafo *g)`

Importa todo o conteúdo do grafo 'g' para o novo formato de acesso em 'st'. A estrutura 'st' deverá ser preenchida com elementos novos, deixando o conteúdo do grafo inalterado. Retorna 0 se bem-sucedido ou -1 caso contrário. Esta função será avaliada pelo tempo de execução.

`char *st_pesquisa(estrutura *st, char *origem, char *destino)`

Obtém o código de voo do par origem-destino com menor preço. A instância retornada deverá ser mantida, i.e., deverá ficar uma cópia dela no respetivo elemento de 'st'. Retorna o código do voo ou em caso de insucesso, retorna NULL. Esta função será avaliada pelo tempo de execução.

`int st_apaga(estrutura *st)`

Retorna 0 se bem-sucedido e -1 se ocorrer algum erro. Elimina todas as instâncias presentes na estrutura 'st' e desaloca toda a memória da mesma. Esta função será avaliada pelo tempo de execução.

***Nota**: Aconselhamos o desenvolvimento de duas funções adicionais: `st_insere` e `st_remove`, para facilitar o desenvolvimento da sua biblioteca.*

Para todas as funções a implementar, uma descrição da própria, dos seus argumentos e retornos poderá ser consultada nos próprio *header files* "grafo.h", "tabdispersao.h" e "stnova.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 = 0.60 Implementação + 0.25 Eficiência + 0.15 Memória

A classificação da implementação e da performance serão essencialmente determinadas por testes automáticos adicionais. Serão avaliados os outputs produzidos pelo código dos estudantes e o tempo de execução face a uma implementação de referência. No caso da implementação submetida não compilar, esta componente será de 0%. Haverá uma diferenciação da classificação, de acordo com a eficiência, como explícito na fórmula, essencialmente focada no exercício 6. Programas que demorarem mais do que 8 minutos a executar serão forçadamente terminados e só serão considerados os testes completados até esse ponto.

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) 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, tabdispersao.c, stnova.c e stnova.h**
- **funções adicionais** que desejem usar devem ser implementadas nos ficheiros **stnova.c/.h**
- 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

5) Exemplo de resultados esperados

```
INICIO DOS TESTES

TESTES DO GRAFO

...verifica_encontra_voo: encontrou com sucesso (OK)
OK: verifica_encontra_voo passou

...verifica_pesquisa_avancada: encontrou com sucesso (OK)
OK: verifica_pesquisa_avancada passou

...verifica_trajeto_mais_rapido: encontrou com sucesso (OK)
OK: verifica_trajeto_mais_rapido passou

...verifica_menos_transbordos: encontrou com sucesso (OK)
OK: verifica_menos_transbordos passou

...verifica_atualiza_lugares: encontrou com sucesso (OK)
OK: verifica_atualiza_lugares passou

...verifica_no_remove (teste de cidade inexistente): não removeu nenhum
nó (OK)
...verifica_no_remove (teste de cidade válida): removeu com sucesso (OK)
OK: verifica_no_remove passou

...verifica_no_apaga: apagou com sucesso (OK)
```

```
OK: verifica_no_apaga passou


TESTES DA TABELA DE DISPERSAO

...verifica_tabela_nova: tabela_nova criou a tabela corretamente (OK)
OK: verifica_tabela_nova passou

...verifica_tabela_adiciona (sem colisão): adicionou corretamente todos
os nós (OK)
...verifica_tabela_adiciona (com colisão): adicionou corretamente o nó
(OK)
OK: verifica_tabela_adiciona passou

...verifica_tabela_remove (remoção sem colisões): tabela_remove removeu
corretamente (OK)
...verifica_tabela_remove (remoção com colisões): tabela_remove removeu
corretamente (OK)
OK: verifica_tabela_remove passou

...verifica_tabela_existe (não existe): tabela_existe não encontrou (OK)
...verifica_tabela_existe (existe): tabela_existe encontrou corretamente
(OK)
OK: verifica_tabela_existe passou

...verifica_tabela_carrega (número de espaços ocupados): tabela_carrega
deu 22 espaços ocupados (OK)
OK: verifica_tabela_carrega passou



TESTES DA ST NOVA

      Tempo a criar a nova estrutura: 0.02957300
...verifica_st  (100000  pesquisas):  st_pesquisa  pesquisou  os  voos
corretos (OK)
      Tempo a pesquisar (100000 pesquisas): 0.39388600
      Tempo a apagar a nova estrutura: 0.00342500
OK: verifica_st passou

FIM DOS TESTES: Todos os testes passaram
```