Trabalho prático 2 – FlightDreams Platform

1) General Information

The goal of trabalho prático 2 is to evaluate the students' capacity to analyse an algorithmic problem, using structures developed from those shown during the course, and implementing in C a correct and efficient solution.

This work must be done in an autonomous way by each group during practical classes until the established date. Consultation of diverse available information sources is acceptable. However, the code submitted must only be authored by the group elements and any plagiarism detected will be severely penalized. The inability to explain the submitted code by any of the group's elements will also incur in penalization.

The deadline for submitting on the Programação 2's Moodle is the 4th of June, by 21:00.

2) Description

You are in charge of dealing with the back-end of an online platform that books flights. - Flight Dreams. You are required to develop the search logic and manipulate the information related to flight timetables, origins, destinations, among others, in order to be ready for the site itself to show. A previous collaborator already started the task.

The data structure that keeps the flight information is already implemented and is based on a directed graph (check "grafo.h" for further details). This must be implemented using an adjacency vector, with edges weighted by several different variables, depending on the desired search.

You must implement:

1. You need to complete the library concerning the graph. The essential structs for the graph construction are:

```
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 */
```

For its construction you need to implement the following functions:

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

Remove and return the pointer to the node of graph 'g', concerning 'cidade', correcting the structure adequately. Return NULL in case of failure.

`int aresta_apaga(aresta_grafo *aresta)`

Eliminates the memory space used by 'aresta'. Return zero in case of success and -1 otherwise.

`int no_apaga(no_grafo *no)`

Eliminate the memory space used by the node 'no'. Return zero in case of success and -1 otherwise.

`void grafo_apaga(grafo *g)`

Delete graph 'g', freeing all its memory.

Note: *The variables of type 'data' are instances of a structure of integers, originally defined on the "time.h" library. We suggest that you get used to it before developing your code.*

2. You want to give useful functionalities to the service, allowing the user to search the graph by certain properties. Implement the following functions:

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

Search in graph 'g' the flight with the specified code - 'codigo'. Return the pointer of the node corresponding to the origin city of the searched flight. Return NULL if the flight is not found or an error occurs. Also, return by reference, through the input argument 'aresta_pos', the position of the respective flight edge on the array of edges of the returned node.

`no_grafo **pesquisa_avancada(grafo *g, char *destino, data chegada, double preco_max, int *n)`

Return the pointer array for all the nodes in the graph 'g' that has a direct flight to the node with name 'destino', on the day specified by 'chegada' and with maximum price given by, and including, 'preco_max'. The array should not include duplicates. Return NULL if no flights are found or in case of error. The size of the array should be returned by reference through the argument 'n'.

3. Allow the search of more complex voyages, with trips that include transhipment:

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

Calculate the path that allows for the earliest arrival from the city 'origem' to city 'destino', only taking into account flights not prior to the day 'partida', using Dijkstra's algorithm. Return an array of pointers to every node of graph 'g' along that path. Return NULL if no valid flight chaining exists or in case of error. The length of the node array should be returned by reference through the argument 'n'.

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

Calculate the path with least amount of transshipments (hops) between the cities 'origem' and 'destino', only considering the flights starting from the day 'partida', using the Dijkstra's algorithm. Assume that all transshipments are valid (you should not verify if the departure of a flight occurs after the arrival of the previous flight). Return an array of pointers to every node of graph 'g' along that path. Return NULL if no valid flight chaining exists or in case of error. The length of the node array should be returned by reference through the argument 'n'.

Note: *We suggest that you use the functions "mktime" and "difftime", of the "time.h" library to more easily compare the dates. For Dijkstra's algorithm you may use the functions already implemented in the file 'heap.c', that includes a heap, already adapted to this problem (each heap element contains a 'no_grafo' element).* Periodically the system should update the graph with alterations to the flights seating arrangements. Implement the following function to read the up-to-date file and remove all flights that ceased to have available seats.

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

Obtain the updated information and correct the graph. Remove from the graph and return the edges/flights whose seats have decreased to zero, via a pointer array of type 'aresta_grafo'. The size of the array should be returned by reference through the argument 'n'. Return NULL in case of error.

Note: The updated available flight seat data can be found in the file ***"flightPlanUpdate.txt"*** *and is in the format: <**flight code>, <available seats>**.*

4. You want to search the graph in a faster way. As such, you decide to implement a hash table to host the nodes and ease the access, without compromising the original graph. The hash table should be of open addressing and include the struct 'tabela_dispersao' in its implementation. Use the city name as the key for the table.

```
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 */
```

Additionally, you should implement the library that allows the use of the table:

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

Create an instance of 'tabela_dispersao' of a certain capacity - 'capacidade' - that uses the hashing function 'hfunc' and the probing function 'sfunc'. Return the pointer to the created table or NULL in case of an error.

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

Add the entry 'entrada' to the table 'td'. Return the index of the added node if it was successful or -1 otherwise.

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

Remove the node 'saida' from the table 'td', not de-allocating its memory. Return zero if successful or -1 otherwise.

`void tabela_apaga(tabela_dispersao *td)`

Remove all the values from the table 'td', leaving it empty, and delete the structure 'td', freeing its memory.

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

Verify if 'cidade' exists in the table. Return the node table index Retorna if the search was successful or -1 otherwise.

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

Create and fill the hash table with a certain capacity 'capacidade' and with the content of graph 'g'. Return the pointer to the created hash table if the process was successful or NULL if that would be impossible by lack of capacity or in case of another error.

***Note:** Consider the available functions `hash_krm` and `sond_rh` as the implementations for the hash and the probing function, respectively.*

```
unsigned long hash_krm(const char* chave, int tamanho)
```
Hashing function. Returns the index related to the table key 'chave'.

```
unsigned long sond_rh(int pos, int tentativas, int
    tamanho)
```
Collision resolution function, implemented with quadratic probing with a step of 'tentativas', starting on the index 'pos'. Calculate and return the alternative index corresponding to the try 'tentativa'. You should use this functions whenever you detect a collision with the previously used index.

5. Your knowledge of data structures has evolved by developing the previous prototype. Develop a structure or combination of structures that permits the best global performance concerning search and removal of flights. Your structure must deal witha ratio of 1 construction and loading for each 100000 search requests. The searches will focus on direct flights (i.e. no transshipping), based on the price criterion.

   Develop the library for what you need, creating the functions and structures that you find useful but always using the following set of interfaces to interact with the testing code:

```
estrutura *st_nova()
```
Create and initialize a structure that can host the desired entries. Return the pointer for the created structure if successful or NULL otherwise.

```
int st_importa_grafo(estrutura *st, grafo *g)
```
Import all the content of graph 'g' to the new access format of 'st'. Structure 'st' must be filled with new elements, leaving the contents of the graph unchanged. return zero if successful or -1 otherwise. <u>This function will be evaluated for execution time.</u>

```
char *st_pesquisa(estrutura *st, char *origem, char
    *destino)
```
Get the flight code of the pair 'origem'-'destino' with the smallest price. The instance returned should be kept, i.e., a copy should stay at the respective element of 'st'. Return the flight code in case of success or NULL otherwise. <u>This function will be evaluated for execution time.</u>

```
int st_apaga(estrutura *st)
```
Eliminate all entries present in the structure 'st' and free all of its memory. Return zero if successful or -1 otherwise. <u>This function will be evaluated for execution time.</u>

***Note**: We suggest, but not require, the development of two additional functions: `st_insere` and `st_remove`, to facilitate the development of the library.*

For all functions to implement a short description, its arguments and return values can be found on the respective *header files* "grafo.h", "tabdispersao.h" and "stnova.h".

3) Evaluation

The grading of this work is performed over the students' code submission. The final work grade for trabalho 2 (TP2) is given by:

T2 = 0.60 Implementation + 0.25 Efficiency + 0.15 Memory

The implementation and performance grading will be essentially determined by additional automatic tests. The outputs and execution time given by the submitted code will be evaluated versus a reference implementation. In the event of the submission not compiling, the implementation component will be graded with 0%. There will be grading differentiation according to efficiency, as explicit by the formula, essentially focusing on exercise 6. Programs that take more than 8 minutes to execute will be forcefully terminated and only the completed test by then will be considered for evaluation.

Memory management will also be evaluated, having the partial grade referring to 3 tiers: 100% no *memory leak*, 50% a few, but not very significant, 0% considerable amount or large *memory leaks.*

4) Submitting the work

The submission is only possible through the Moodle platform and until the date specified on the top of this document. A *zip* file should be uploaded including the following files:

- files **grafo.c, tabdispersao.c, stnova.c e stnova.h**
- **additional functions** that you feel should be implemented in the files **stnova.c/.h**
- a file **autores.txt**, stating the names and number of the group elements

Important Note: only the submission with the following name format will be accepted: T2_G<group_number>.zip. An example, T2_G999.zip

5) Expected output with test script

```
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
```