Relatório de Estruturas de Dados e Algoritmos 2

Miguel Maia, nº 33456
Daniel Carvalho, nº 34250

12 de Junho de 2018

Resumo

O objetivo do trabalho de EDA2 é fazer um programa de troca de mensagens entre utilizadores. Tendo a informação em disco, pode-se encerrar o programa, mas mesmo assim a informação não se perde.

1 Estruturas de Dados

Depois de termos tentado utilizar tries para a realização deste trabalho, mas nao termos sido bem sucedidos, a única estrutura de dados utilizada neste trabalho foi uma hashtable. A razão desta escolha foi a fácil utilização, o tamanho que ocupa e a complexidade dos seus métodos que é maioritariamente O(1);. A hashtable funciona de forma semelhante a um array, mas tem associada a cada elemento, uma chave. É esta chave que indica onde é que o elemento fica na hashtable. O tamanho da hashtable é de 2500009 pois o tamanho tinha de ser pelo menos 2200000 e numero primo. Como não sabíamos o que estava errado no trabalho decidimos ter mais posições na hashtable do que precisavamos. Esta estrutura de dados é utilizada apenas em memória central. Cada elemento da nossa hashtable tem uma struct hashtable_item, que contem um short flag, para verificar se o elemento existe e uma struct data, que tem um char nick[6] e um int file_ptr que é o apontador para o ficheiro.

2 Ficheiros de Dados

- Os nossos dados são guardados em dois ficheiros distintos, o hashtable_data e dados_data. O ficheiro hashtable_data guarda as informações da nossa hashtable, ou seja, contém a posição do elemento, a sua flag e o nick correspondente. O ficheiro dados_data guarda structs users, que têem a informação dos utilizadores, incluindo as mensagens enviadas, os seguidores e os seguidos. Qualquer um dos ficheiros funciona como um array.

Dados em Memória Secundária

	id = 0	id = 1	...
Dados_data	char nick[6] char nome[26] int menagem int seguido_por short n_seguidores struct seguidores seguidor[100] char seguidor[].nick[6] int seguidor[].ult_mens	char nick[6] char nome[26] int menagem int seguido_por short n_seguidores struct seguidores seguidor[100] char seguidor[].nick[6] int seguidor[].ult_mens	...

Cada id contém:

- **char nick** onde está armazenado o nick do utilizador;
- **char nome** onde está armazenado o nome do utilizador;
- **int mensagem** onde está armazenado o número de mensagens que o utilizador já enviou;
- **int seguido_por** onde está armazenado o número de utilizadores que o seguem;
- **short n_seguidores** onde está armazenado o número de seguidores que o utilizador segue;
- **char seguidor.nick** que contém o nome dos utilizadores seguidos;
- **int seguidor.ult_mens** que tem a ultima mensagem lida dos seguidores.

As posições são adicionadas a partir do 0, e são acrescentados sequencialmente. Cada posição ocupa 6 + 26 + 4 + 4 +2 + (6+4)*100 = 1042 + 1(alinhamento) = 1044 *Bytes*. O tamanho deste ficheiro depende do número de utilizadores e é dado por: **n users x 1044 *Bytes***.

	id = 0	id = 1	...
Hashtable_data	int flag struct hashtable_item int file_ptr char nick[6]	int flag struct hashtable_item int file_ptr char nick[6]	...

Cada id contém:

- **int flag** onde está armazenado a condição para saber se está ativo ou não;
- **int file_ptr** onde está armazenado a zona do ficheiro onde está;
- **char nick** onde está armazenado o nick do utilizador.

As posições são adicionadas a partir do 0, e são acrescentados sequencialmente. Cada posição ocupa 4 + 4 + 6 = 14 + 2(alinhamento) = 16 *Bytes*. O tamanho deste ficheiro depende do número de utilizadores e é dado por: **n users x 16 *Bytes***

3 Operações

- **Main.c**
  - **Main :** A primeira coisa que fazemos é colocar o buffer stdout a NULL para que o output seja disponibilizado assim que possível. Em seguida verifica-se se o ficheiro "dados_data"se encontra disponível. Em caso negativo, ele é criado. De seguida é aberto. Alocamos memória á hashtable com o tamanho máximo pré-definido. Após isso, inicializamos todos os bytes da memoria alocada a zero. Verifica-se se o ficheiro "hashtable_data"existe. Caso exista, é carregado. Inicializamos as variáveis caracter, nick, nome e nick2, e colocamos tudo a 0. Enquanto o input for diferente de EOF ou se o caracter for X, o programa termina.Em caso de ser X, guarda-se a hashtable e fecha-se o ficheiro dados_data e por fim faz-se free da memoria. Consoante a letra do input, é chamada uma diferente função.
  - **Binsearch :** Primeiro inicializamos as variáveis que vamos utilizar no resto da função. Enquanto o inicio do array for menor que o final, se a posição tiver o elemento que procuramos, a função devolve a posição do elemento. Caso contrário a função devolve -1, porque não encontrou o elemento. A complexidade desta função é O(log n).
  - **Insert :** A primeira coisa a fazer é calcular as funções de hash do nick dado. O index passa a ser o valor dado na função de hash. Enquanto a flag da posição acedida for diferente de 0, ou seja, se for diferente de posição vazia e o nick encontrado for igual ao nick recebido, então o utilizador já foi utilizado. Se isto não aconteceu, a posição passa o resto da posição menos a segunda função de hash pelo tamanho máximo da hashtable. Se houver colisões na hashtable, a programa gera um ficheiro com essas colisões. Aloca-se espaço para a struct users. Coloca-se todos os valores da struct a 0. Depois associa-se a informação do utilizador à sua struct e os valores que não são dados são inicializados a 0. Pomos o apontador do ficheiro no fim do mesmo. Guarda-se essa posição no file_ptr na hashtable. Passa-se o nick para a hashtable e coloca-se a flag a 1, o que significa que esse espaço na hashtable está ocupado e ativo. Escreve-se no ficheiro a struct temporária new_user e incrementa-se o tamanho ocupado da hashtable. Liberta-se a memória previamente alocada e o programa indica que o utilizador foi criado. A complexidade temporal desta função é O(n), onde n é o tamanho da hashtable.
  - **Remove_element :** Tenta-se encontrar o nick na hashtable através da função find. Caso o resultado seja -1, significa que o elemento não foi encontrado e o programa indica o mesmo. Caso exista, a flag passa a 2, o que significa que o nick já não pode ser utilizado outra vez e o programa diz que o utilizador foi removido. A complexidade temporal é O(1).
  - **Seguir_nick :** Tenta-se encontrar o nick1 através da função find. Caso dê -1, então o utilizador não existe. Poe-se o apontador do ficheiro na posição deste utilizador e lê-se a informação que temos sobre ele. Depois faz-se o mesmo para o nick2. Tenta-se encontrar o nick2 no array de seguidores do nick1. Se o resultado nao for -1 então o nick1 já segue o nick2. Depois verifica-se se o numero de seguidores não está no limite. Se passar estas condições guarda-se o numero de seguidores e cria-se uma struct seguidores temporaria. Copia-se o nick2 para essa struct e o numero de mensagens do mesmo. Se o número de seguidores for diferente de zero, enquanto o nick na posição atual for diferente do nick2 e a posição atual não ultrapassar o numero de seguidores, incrementa um short. Se o short for igual ao numero de seguidores, então o utilizador é adicionado no fim do array seguidores. Se não, todos os elementos avançam uma posição e o utilizador é colocado na posição correta. Caso o numero de seguidores seja

0, coloca-se o utilizador a seguir no inicio do array. Por fim, incrementa-se o numero de seguidores do nick1 e o número dos seguidos do nick2. Se o nick1 e o nick2 forem iguais, o numero de seguidos aumenta, coloca-se o apontador do ficheiro na posição desse utilizador e escreve-se no ficheiro as novas informaçãos. Se forem diferentes, tem de se fazer o mesmo, mas para os dois nicks em questão. O programa indica que o nick1 passa a seguir o nick2. A função tem complexidade temporal de O(n^2), onde n é o numero de seguidores.

- **D_seguir_nick :** Tenta-se encontrar o nick1 através da função find. Caso dê -1, então o utilizador nao existe. Põe-se o apontador do ficheiro na posição deste utilizador e lê-se a informação que temos sobre ele. Depois faz-se o mesmo para o nick2. Vai-se procurar o nick2 no array de seguidores do nick1. Se o resultado der -1, então o nick2 não segue o nick1. Se seguir, todos os utilizadores que estiverem depois do nick a remover recuam uma posição no array. O numero de seguidores do nick1 e o número de seguidos do nick2 é decrementado. Se o nick1 e o nick2 forem iguais, o numero de seguidos diminui, coloca-se o apontador do ficheiro na posição desse utilizador e escreve-se no ficheiro as novas informaçãos. Se forem diferentes, tem de se fazer o mesmo, mas para os dois nicks em questão. No fim o programa indica que o nick1 deixou de seguir o nick2. A complexidade temporal é de O(n), onde n é o numero de seguidores.
- **Enviar_mens :** Tenta-se encontrar o nick através da função find. Se não for encontrado o programa mostra a mensagem correspondente e termina esta função. Se for encontrado, coloca o apontador de leitura/escrita na localização de dados do nick indicado e lê o conjunto de dados relativos ao mesmo. Incrementa o valor de mensagens enviadas e coloca o apontador de leitura/escrita na localização de dados do nick indicado e escreve o conjunto de dados relativos ao mesmo. Esta função tem complexidade temporal O(1).
- **Ler_mens :** Tenta-se encontrar o nick através da função find. Se não for encontrado o programa mostra a mensagem correspondente e termina esta função. Se for encontrado, coloca o apontador de leitura/escrita na localização de dados do nick indicado e lê o conjunto de dados relativos ao mesmo. Se o nick não tiver seguidos, o programa mostra a mensagem correspondente e termina a função. Se tiver, vai percorrer o array de seguidos e verificar a existência dos mesmos na hashtable. Se não for encontrado, mostra a mensagem correspondente, retira o mesmo dos seguidos do nick e, se existentes, move todos seguintes seguidos uma casa para trás no array. Se for encontrado, coloca o apontador de leitura/escrita na localização de dados do seguido indicado e lê o conjunto de dados relativos ao mesmo. De seguida, compara a ultima mensagem lida com a última mensagem envia do seguido, mostra a mensagem correspondente e atualiza o valor a ultima mensagem lida. Coloca o apontador de leitura/escrita na localização de dados do nick indicado e escreve o conjunto de dados relativos ao mesmo. Esta função tem complexidade temporal O(n^2), onde n é o número de seguidos do nick.
- **Info_nick :** Tenta-se encontrar o nick através da função find. Se não for encontrado o programa mostra a mensagem correspondente e termina esta função. Se for encontrado, coloca o apontador de leitura/escrita na localização de dados do nick indicado e lê o conjunto de dados relativos ao mesmo. O programa, de seguida, mostra o nick, o nome, mensagens enviadas, número de seguidores e seguidos. Entra-se num ciclo e é mostrado o nick e ultima mensagem lida de cada seguido. Esta função tem complexidade temporal O(n), onde n é o número de seguidos do nick.
- **Carregar_hashtable :** Abre-se o ficheiro de dados da hashtable. Inicializa-se variá-

veis temporais para a leitura do ficheiro. Lê-se a primeira linha onde indica o tamanho previamente ocupado na hashtable. Faz-se um ciclo onde vai-se ler cada linha contendo a posição na hashtable, a flag, o nick e o apontador para o ficheiro de dados do utilizador. No final, liberta-se a memória previamente alocada. Esta função tem complexidade temporal O(n), onde n é o tamanho previamente ocupado na hashtable.

- **Guardar_hashtable :** Abre-se o ficheiro de dados da hashtable. Guarda-se na primeira linha o número que utilizadores que estão guardados na hashtable na memória primária. Percorre-se a hashtable num ciclo e se a flag de cada posição for diferente de 0 (ou seja, tem dados guardados ativos ou não) guarda-se em cada linha a posição na hashtable, a respetiva flag, o nick, e o apontador para o ficheiro de dados do utilizador. No final, liberta-se a memória previamente alocada. Esta função tem complexidade temporal O(n), onde n é o tamanho máximo da hashtable.

- **HashTable.c**
  - **Hashcode1 :** Esta função devolve o resto da chave com o tamanho máximo da hashtable. A complexidade é O(1).
  - **Hashcode2 :** Semelhante á função hashcode1, mas devolve o resultado do numero primo menos o resto entre a chave e o primo. A complexidade é O(1).
  - **Ht_calc_hash :** Esta função faz o calculo da chave através do nick fornecido. Só acaba quando percorrer todas as letras no nick. Esta função foi retirada da Internet. A complexidade desta função é O(1), pois o nick tem sempre tamanho 5.
  - **Find :** Primeiro calcula-se as hashes do nick fornecido. Depois, dentro de um ciclo, verifica se o nick está na posição do primeiro hash. Caso esteja, retorna a posição. Caso contrário, vai calcular uma nova posição de pesquisa com o hash dois e tentar encontrar de novo.Se encontrar o nick desejado mas não estiver marcado como ativo, retorna -1. No final do ciclo, se não encontrar, retorna -1. A complexidade é O(n), sendo n o tamanho da hashtable.

4 Bibliografia

Os seguintes sites foram onde fomos buscar a implementação das hashtables, que depois adaptamos para este trabalho:

https://www.sanfoundry.com/c-program-implement-hash-tables-with-double-hashing/
http://www.cse.yorku.ca/ oz/hash.html

5 O código

— hashtable.h —

```
1  //
2  // Created by Daniel on 10/06/2018.
3  //
4
5  #ifndef TRABALHOV6_HASHTABLE_H
6  #define TRABALHOV6_HASHTABLE_H
7
8  struct seguidores{
9      char nick[6];
10     int ult_mens;
```

```
11 };
12
13 struct users{
14     char nick[6];
15     char nome[26];
16     int mensagem;
17     int seguido_por;
18     short n_seguidores;
19     struct seguidores seguidor[100];
20 };
21
22 struct data
23 {
24     long file_ptr;
25     char nick[6];
26 };
27
28 struct hashtable_item
29 {
30
31     int flag;
32     /*
33      * flag = 0 : data not present
34      * flag = 1 : some data already present
35      * flag = 2 : data was present,but deleted
36     */
37     struct data item;
38
39 };
40
41 const int max;
42 int size;
43 int prime;
44
45 struct hashtable_item *array;
46
47 int hashcode1(unsigned long long key);
48 int hashcode2(unsigned long long key);
49
50 unsigned long long ht_calc_hash (char* key);
51
52 int find(unsigned long long key, char nick[6]);
53
54 #endif //TRABALHOV6_HASHTABLE_H
```

— hashtable.c —

```
1 //
2 // Created by Daniel on 10/06/2018.
3 //
4
5 #include <memory.h>
6 #include "hashtable.h"
7
8 const int max = 2500009;
9 int size = 0;
10 int prime = 2499997;
```

```
11
12 struct hashtable_item *array;
13
14 /*
15     Tem como argumentos:     key inicial
16
17     calcula o 1 hashcode
18
19     Devolve:     1 hashcode
20 */
21 int hashcode1(unsigned long long key)
22 {
23     return (key % max);
24 }
25
26 /*
27     Tem como argumentos:     key inicial
28
29     calcula o 2 hashcode
30
31     Devolve:     2 hashcode
32 */
33 int hashcode2(unsigned long long key)
34 {
35     return (prime - (key % prime));
36 }
37
38 /*
39     Tem como argumentos:     nick de um user
40
41     calcula a key inicial
42
43     Devolve:     key inicial
44 */
45 unsigned long long ht_calc_hash (char* key) {
46     unsigned long long h = 5381;
47     while (*(key++)){
48         h = ((h << 5) + h) + (*key);
49     }
50     return h;
51 }
52
53 /*
54     Tem como argumentos:     key inicial
55
56     Pesquisa o nick na hashtable
57
58     Devolve:     index na hashtable
59 */
60 int find(unsigned long long key, char nick[6]) {
61     int hash1 = hashcode1(key);
62     int hash2 = hashcode2(key);
63
64     int index = hash1;
65
66     while (array[index].flag !=0) {
```

```
67          if (strcmp(array[index].item.nick, nick) == 0) {
68              if (array[index].flag==1) {
69                  return index;
70              }
71              else {
72                  index = -1;
73                  return index;
74              }
75          }
76          index = (index + hash2) % max;
77
78      }
79      index=-1;
80      return index;
81  }
```

— main.c —

```
1  #include<stdio.h>
2  //#include<math.h>
3  #include <stdlib.h>
4  #include <stdbool.h>
5  #include <memory.h>
6  #include <unistd.h>
7
8  #include "hashtable.h"
9
10 /*
11      Tem como argumentos:     array dos seguidores de um user
12                               numero de seguidores do user
13                               nick a ser encontrado
14
15      Faz a pesquisa binaria de um seguido por um user
16
17      Devolve:     posicao no array do nick encontrado
18                   menos 1 se nao for encontrado
19 */
20 int binsearch(struct seguidores seguidor[max], int max, char *value) {
21      int returntemp=-1;
22      int position=0;
23      int begin = 0;
24      int end = max - 1;
25      int cond = 0;
26
27      while(begin <= end) {
28          position = (begin + end) / 2;
29          if((cond = strcmp(seguidor[position].nick, value)) == 0) {
30              returntemp=position;
31              return returntemp;
32          }
33          else if(cond < 0)
34              begin = position + 1;
35          else
36              end = position - 1;
37      }
38
39      return returntemp;
```

```
40 }
41
42 /*
43     Tem como argumentos:     nick do user
44                              nome do user
45                              ficheiro de dados
46
47     Faz a insercao do user na hashtable e no ficheiro, se aplicavel
48
49     Devolve:     nada
50 */
51 void insert(char nick[6], char nome[26], FILE *ficheiro)
52 {
53     int hash1 = hashcode1(ht_calc_hash(nick));
54     int hash2 = hashcode2(ht_calc_hash(nick));
55
56     int index = hash1;
57
58     while (array[index].flag !=0) {
59         if (strcmp(array[index].item.nick, nick) == 0) {
60             printf("+ nick %s usado previamente\n", nick);
61             return;
62         }
63         index = (index + hash2) % max;
64         if (index == hash1) {
65             FILE *err = fopen("err", "w");
66             fprintf(err, "erro no index %d %s\n", index, nick);
67             fclose(err);
68             return;
69         }
70     }
71
72
73     struct users *new_user = malloc(sizeof(struct users));
74     memset(new_user, 0, sizeof(struct users));
75     strncpy(new_user->nick,nick,6);
76     strncpy(new_user->nome,nome,26);
77     new_user->n_seguidores=0;
78     new_user->mensagem=0;
79     new_user->seguido_por=0;
80
81
82     fseek(ficheiro,0,SEEK_END);
83     array[index].item.file_ptr=ftell(ficheiro);
84     strcpy(array[index].item.nick,nick);
85     array[index].flag = 1;
86     fwrite(new_user, sizeof(struct users),1,ficheiro);
87     size++;
88     free(new_user);
89     printf("+ utilizador %s criado\n", nick);
90
91 }
92
93 /*
94     Tem como argumentos:     nick do user
95
```

```
96      Faz a remocao de dados do user na hashtable, se aplicavel
97
98      Devolve:      nada
99  */
100 void remove_element(char nick[6])
101 {
102     int index = find(ht_calc_hash(nick),nick);
103
104     if(index==-1){
105         printf("+ utilizador %s inexistente\n",nick);
106         return;
107     }
108
109     array[index].flag = 2;
110     printf("+ utilizador %s removido\n", nick);
111 }
112
113 /*
114     Tem como argumentos:      nick do user a seguir
115                               nome do user a ser seguido
116                               ficheiro de dados
117
118     Coloca o user a ser seguido no array
119     de seguidores no user a seguir (se aplicavel)
120
121     Devolve:      nada
122 */
123 void seguir_nick(char nick1[6], char nick2[6], FILE *ficheiro){
124     int index1 = find(ht_calc_hash(nick1),nick1);
125
126     if(index1==-1){
127         printf("+ utilizador %s inexistente\n",nick1);
128         return;
129     }
130     fseek(ficheiro,array[index1].item.file_ptr,SEEK_SET);
131     struct users user1;
132     fread(&user1, sizeof(struct users),1,ficheiro);
133
134
135     int index2 = find(ht_calc_hash(nick2),nick2);
136
137     if(index2==-1){
138         printf("+ utilizador %s inexistente\n",nick2);
139         return;
140     }
141     fseek(ficheiro,array[index2].item.file_ptr,SEEK_SET);
142     struct users user2;
143     fread(&user2, sizeof(struct users),1,ficheiro);
144
145
146     if (binsearch(user1.seguidor,user1.n_seguidores,user2.nick)!=-1){
147         printf("+ utilizador %s segue %s\n",user1.nick,user2.nick);
148         return;
149     }
150
151     if (user1.n_seguidores==100){
```

```
152         printf("+ utilizador %s segue o limite\n", user1.nick);
153         return;
154     }
155 
156     short n = user1.n_seguidores;
157     struct seguidores temp;
158     strcpy(temp.nick,user2.nick);
159     temp.ult_mens=user2.mensagem;
160 
161     short l=0;
162 
163     if (n!=0) {
164         while (strcmp(user1.seguidor[l].nick, user2.nick) < 0 && l<n) {
165             l++;
166         }
167 
168         if (l==n){
169             strcpy(user1.seguidor[l].nick,temp.nick);
170             user1.seguidor[l].ult_mens = temp.ult_mens;
171         }
172         else {
173             struct seguidores temp2;
174             for (int i = l; i < n; ++i) {
175                 strcpy(temp2.nick, user1.seguidor[i].nick);
176                 temp2.ult_mens = user1.seguidor[i].ult_mens;
177                 strcpy(user1.seguidor[i].nick, temp.nick);
178                 user1.seguidor[i].ult_mens = temp.ult_mens;
179                 strcpy(temp.nick, temp2.nick);
180                 temp.ult_mens = temp2.ult_mens;
181             }
182             strcpy(user1.seguidor[n].nick, temp.nick);
183             user1.seguidor[n].ult_mens = temp.ult_mens;
184         }
185     }
186     else {
187         strcpy(user1.seguidor[0].nick,temp.nick);
188         user1.seguidor[0].ult_mens = temp.ult_mens;
189     }
190 
191 
192     user1.n_seguidores++;
193     user2.seguido_por++;
194 
195 
196     if (strcmp(user1.nick,user2.nick)==0){
197         user1.seguido_por++;
198         fseek(ficheiro, array[index1].item.file_ptr, SEEK_SET);
199         fwrite(&user1, sizeof(struct users),1,ficheiro);
200     }
201 
202     else {
203         fseek(ficheiro, array[index1].item.file_ptr, SEEK_SET);
204         fwrite(&user1, sizeof(struct users), 1, ficheiro);
205 
206         fseek(ficheiro, array[index2].item.file_ptr, SEEK_SET);
207         fwrite(&user2, sizeof(struct users), 1, ficheiro);
```

```
208     }
209
210     printf("+ %s passou a seguir %s\n",user1.nick,user2.nick);
211
212 }
213
214 /*
215     Tem como argumentos:     nick do user a deixar de seguir
216                              nome do user a deixar de ser seguido
217                              ficheiro de dados
218
219     Retira o user a deixar de ser seguido no array
220     de seguidores no user a deixar de seguir
221     emover todos os seguidores seguintes um passo
222     para tras no array (se aplicavel)
223
224     Devolve:     nada
225 */
226 void D_seguir_nick(char nick1[6], char nick2[6], FILE *ficheiro){
227     int index1 = find(ht_calc_hash(nick1),nick1);
228
229     if(index1==-1){
230         printf("+ utilizador %s inexistente\n",nick1);
231         return;
232     }
233     fseek(ficheiro,array[index1].item.file_ptr,SEEK_SET);
234     struct users user1;
235     fread(&user1, sizeof(struct users),1,ficheiro);
236
237
238     int index2 = find(ht_calc_hash(nick2),nick2);
239
240     if(index2==-1){
241         printf("+ utilizador %s inexistente\n",nick2);
242         return;
243     }
244     fseek(ficheiro,array[index2].item.file_ptr,SEEK_SET);
245     struct users user2;
246     fread(&user2, sizeof(struct users),1,ficheiro);
247
248     int indexseg=binsearch(user1.seguidor,user1.n_seguidores,user2.nick);
249     if (indexseg==-1){
250         printf("+ utilizador %s nao segue %s\n",user1.nick,user2.nick);
251         return;
252     }
253
254     short n = user1.n_seguidores;
255     for (int i = indexseg; i < n-1; ++i) {
256         strcpy(user1.seguidor[i].nick,user1.seguidor[i+1].nick);
257         user1.seguidor[i].ult_mens=user1.seguidor[i+1].ult_mens;
258     }
259
260     user1.n_seguidores--;
261
262     user2.seguido_por--;
263
```

```
264     if (strcmp(user1.nick,user2.nick)==0){
265         user1.seguido_por--;
266         fseek(ficheiro, array[index1].item.file_ptr, SEEK_SET);
267         fwrite(&user1, sizeof(struct users),1,ficheiro);
268     }
269 
270     else {
271         fseek(ficheiro, array[index1].item.file_ptr, SEEK_SET);
272         fwrite(&user1, sizeof(struct users), 1, ficheiro);
273 
274         fseek(ficheiro, array[index2].item.file_ptr, SEEK_SET);
275         fwrite(&user2, sizeof(struct users), 1, ficheiro);
276     }
277 
278     printf("+ %s deixou de seguir %s\n",user1.nick,user2.nick);
279 }
280 
281 /*
282     Tem como argumentos:     nick do user
283                              ficheiro de dados
284 
285     Incrementa o contador de mensagens enviadas do user (se aplicavel)
286 
287     Devolve:     nada
288 */
289 void enviar_mens(char nick[6], FILE *ficheiro){
290 
291     int index = find(ht_calc_hash(nick),nick);
292 
293     if(index==-1){
294         printf("+ utilizador %s inexistente\n",nick);
295         return;
296     }
297     fseek(ficheiro,array[index].item.file_ptr,SEEK_SET);
298     struct users user1;
299     fread(&user1, sizeof(struct users),1,ficheiro);
300 
301     user1.mensagem++;
302 
303     fseek(ficheiro,array[index].item.file_ptr,SEEK_SET);
304     fwrite(&user1, sizeof(struct users),1,ficheiro);
305 }
306 
307 /*
308     Tem como argumentos:     nick do user
309                              ficheiro de dados
310 
311     Percorre o array de seguidores de um user,
312     verifica se algum foi removido e remove do array
313     e indica as mensagens nao lidas(se aplicavel)
314 
315     Devolve:     nada
316 */
317 void ler_mens(char nick[6],FILE *ficheiro){
318     int index = find(ht_calc_hash(nick),nick);
319 
```

```
320     if(index==-1){
321         printf("+ utilizador %s inexistente\n",nick);
322         return;
323     }
324     fseek(ficheiro,array[index].item.file_ptr,SEEK_SET);
325     struct users user1;
326     fread(&user1, sizeof(struct users),1,ficheiro);
327 
328     if (user1.n_seguidores==0){
329         printf("+ utilizador %s sem seguidos\n",user1.nick);
330         return;
331     }
332 
333     int n_seg_temp=user1.n_seguidores;
334     for (int l = 0; l < n_seg_temp; ++l) {
335 
336         int indexseg = find(ht_calc_hash((user1.seguidor[l].nick)), user1.
                seguidor[l].nick);
337         if (indexseg == -1) {
338             printf("utilizador %s desactivado\n", user1.seguidor[l].nick);
339             for (int i = l; i < n_seg_temp - 1; ++i) {
340                 //*infonick1.seguidor_nick[i]=*infonick1.seguidor_nick[i
                        +1];
341                 strcpy(user1.seguidor[i].nick, user1.seguidor[i + 1].nick);
342                 user1.seguidor[i].ult_mens = user1.seguidor[i + 1].ult_mens
                        ;
343             }
344             strncpy(user1.seguidor[n_seg_temp-1].nick, "", 6);
345             user1.seguidor[n_seg_temp-1].ult_mens = 0;
346             user1.n_seguidores--;
347             l--;
348             n_seg_temp--;
349         } else {
350             fseek(ficheiro, array[indexseg].item.file_ptr, SEEK_SET);
351             struct users nickseguidor;
352             fread(&nickseguidor, sizeof(struct users), 1, ficheiro);
353 
354             if (nickseguidor.mensagem == user1.seguidor[l].ult_mens) {
355                 printf("sem mensagens novas de %s (%s)\n", nickseguidor.
                        nick, nickseguidor.nome);
356             }
357             if (nickseguidor.mensagem == user1.seguidor[l].ult_mens + 1) {
358                 printf("mensagem nova de %s (%s): %d\n", nickseguidor.nick,
                         nickseguidor.nome, nickseguidor.mensagem);
359                 user1.seguidor[l].ult_mens = nickseguidor.mensagem;
360             }
361             if (nickseguidor.mensagem > user1.seguidor[l].ult_mens + 1) {
362                 printf("mensagens novas de %s (%s): %d a %d\n",
                        nickseguidor.nick, nickseguidor.nome,
363                         user1.seguidor[l].ult_mens + 1, nickseguidor.
                            mensagem);
364                 user1.seguidor[l].ult_mens = nickseguidor.mensagem;
365             }
366         }
367     }
368 
```

```
369     fseek(ficheiro,array[index].item.file_ptr,SEEK_SET);
370     fwrite(&user1, sizeof(struct users),1,ficheiro);
371
372
373 }
374
375 /*
376     Tem como argumentos:     nick do user
377                              ficheiro de dados
378
379     Mostra as informacoes do nick indicado
380     (nick, nome, mensagens enviadas, numeros de seguidores e que segue)
381
382     Devolve:     nada
383 */
384 void info_nick(char nick[6], FILE *ficheiro){
385     int index = find(ht_calc_hash(nick), nick);
386
387     if(index==-1){
388         printf("+ utilizador %s inexistente\n",nick);
389         return;
390     }
391     fseek(ficheiro,array[index].item.file_ptr,SEEK_SET);
392     struct users user1;
393     fread(&user1, sizeof(struct users),1,ficheiro);
394
395     printf("utilizador %s (%s)\n",nick,user1.nome);
396     printf("%d mensagens, %d seguidores, segue %d utilizadores\n",user1.
        mensagem,user1.seguido_por,user1.n_seguidores);
397
398     for (int i = 0; i < user1.n_seguidores; ++i) {
399         printf("%s (%d lidas)\n", user1.seguidor[i].nick, user1.seguidor[i
            ].ult_mens);
400     }
401
402
403 }
404
405
406 /*void init_array()
407 {
408     for(int i = 0; i < max; i++)
409     {
410         *array[i].item.nick=NULL;
411         array[i].item.file_ptr=0;
412         array[i].flag = 0;
413     }
414
415 }*/
416
417 /*
418     Tem como argumentos:     nada
419
420     Carrega os dados da hashtable no inicio do programa
421
422     Devolve:     nada
```

```
423 */
424 void carregar_hashtable(){
425     FILE *hashtable = fopen("hashtable_data","r");
426     int i=0;
427     int flag=0;
428     char nick[6];
429     long file_ptr=0;
430     fscanf(hashtable,"%d\n",&size);
431     for (int j = 0; j < size; ++j) {
432     fscanf(hashtable,"%d %d %s %ld\n",&i,&flag,nick,&file_ptr);
433     array[i].flag=flag;
434     strncpy(array[i].item.nick,nick,6);
435     array[i].item.file_ptr=file_ptr;
436     }
437     fclose(hashtable);
438 }
439
440 /*
441     Tem como argumentos:     nada
442
443     Guarda os dados da hashtable no final do programa
444
445     Devolve:     nada
446 */
447 void guardar_hashtable(){
448     FILE *hashtable = fopen("hashtable_data","w");
449     fprintf(hashtable,"%d\n",size);
450     for (int i = 0; i < max; ++i) {
451         if (array[i].flag!=0){
452             fprintf(hashtable,"%d %d %s %ld\n",i,array[i].flag,array[i].
                    item.nick,array[i].item.file_ptr);
453         }
454     }
455
456     fclose(hashtable);
457 }
458
459 /*
460     Tem como argumentos:     nada
461
462     Verifica a existencia do ficheiro de dados e cria se necessario
463     Aloca e inicializa a 0 a hashtable
464     Chama todas as outras funcoes de acordo com o input
465
466     Devolve:     nada
467 */
468 int main() {
469     setbuf(stdout, NULL);
470
471     if (access("dados_data",F_OK)==-1){
472         FILE *temp = fopen("dados_data","a");
473         fclose(temp);
474     }
475     FILE *ficheiro = fopen("dados_data","r+b");
476
477     array = malloc(max * sizeof(struct hashtable_item));
```

```
478
479     memset(array,0, max * sizeof(struct hashtable_item));
480
481     if (access("hashtable_data",F_OK)!=-1){
482         carregar_hashtable();
483     }
484
485     char caracter;
486     char nick[6];
487     char nome[26];
488     char nick2[6];
489     memset(&caracter,'\0', sizeof(caracter));
490     memset(&nick,'\0', sizeof(nick));
491     memset(&nick2,'\0', sizeof(nick2));
492     memset(&nome,'\0', sizeof(nome));
493
494
495     while(scanf("␣%c",&caracter)!=EOF) {
496
497         if(caracter=='U')
498         {
499             scanf("␣%s␣%[^\n]", nick, nome);
500             insert(nick, nome,ficheiro);
501         }
502         else if(caracter=='R')
503         {
504             scanf("␣%s", nick);
505             remove_element(nick);
506         }
507         else if(caracter=='S')
508         {
509             scanf("␣%s␣%s",nick, nick2);
510             seguir_nick(nick, nick2, ficheiro);
511         }
512         else if(caracter=='D')
513         {
514             scanf("␣%s␣%s",nick, nick2);
515             D_seguir_nick(nick, nick2, ficheiro);
516         }
517         else if(caracter=='P')
518         {
519             scanf("␣%s", nick);
520             enviar_mens(nick, ficheiro);
521         }
522         else if(caracter=='L')
523         {
524             scanf("␣%s",nick);
525             ler_mens(nick, ficheiro);
526         }
527         else if(caracter=='I')
528         {
529             scanf("␣%s",nick);
530             info_nick(nick, ficheiro);
531         }
532         else if(caracter=='X')
533         {
```

```
534                 guardar_hashtable();
535                 fclose(ficheiro);
536                 free(array);
537                 return 0;
538             }
539         }
540 }
```