UNIVERSIDADE DE SÃO PAULO – USP

SCC0223 - ESTRUTURAS DE DADOS I

EXERCÍCIO 3 - LISTA

Lucas Viana Vilela
10748409

São Carlos
2020

Sumário

1 Introdução

O trabalho envolveu a implementação do TAD Lista Não Ordenada, com abordagem Encadeada Dinâmica, bem como de uma função principal para gerir a interação com o usuário. Este desenvolvimento foi feito a partir da interface fornecida no arquivo lista.h, enviado em anexo.

Assim, este relatório inclui uma explicação geral acerca do código, bem como dos casos de teste desevolvidos para aperfeiçoá-lo.

2 Desenvolvimento

2.1 Estruturas da lista e de seus nós

```
struct NODE{
   ITEM movie;
   struct NODE *next;
}; typedef struct NODE node;

struct lista{
   node *start;
};
```

Primeiro, definiu-se que um nó da lista deve conter dois membros:

- Um filme - elemento do tipo ITEM, definido no lista.h, denominado *movie*.
- Uma variável do tipo ponteiro de nó (*struct NODE* = *node*), denominada *next*. Ao apontar para o nó seguinte, ela é responsável pelo encadeamento da lista.

Para simplificação e maior facilidade de leitura do código, também definiu-se o tipo *struct NODE* recém criado como *node*.

O tipo struct lista (ou LISTA), então, foi implementado como contendo apenas a variável *start*, ponteiro que apontará para o primeiro nó da lista (correspondente ao seu primeiro elemento). Apenas isto é suficiente para que se acesse toda a lista, pois, como supracitado, cada nó possui um membro que aponta para o seguinte.

2.2 Função para criação da lista

```
LISTA* lista_criar(void){
   LISTA *newList = (LISTA *)malloc(sizeof(LISTA));

   if(newList != NULL){
      newList->start = NULL; // Cria uma lista vazia
      return newList;
   }
```

```
   else { return NULL; } // Se no conseguir alocar espao para uma
      lista

}
```

Esta função vai tentar alocar memória para uma lista e, caso consiga (*newList != NULL*), vai inicializar a sua variável *start* como nula - algo que será mudado assim que o primeiro elemento for inserido na lista.
Por fim, a função vai retornar um ponteiro para a lista alocada.

2.3 Função para inserção de elementos na lista

```
boolean lista_inserir_pos(LISTA *l, ITEM filme, int pos){
   if(l == NULL || lista_cheia(l) || pos < 1 || pos >
      lista_tamanho(l)+1){ return FALSE; }

   else{
      node *newNode = (node *)malloc(sizeof(node));

      newNode->movie = filme;

      if(lista_vazia(l) || pos == 1){ // Caso a lista esteja vazia
         ou a posio desejada seja a primeira
         // Insere o novo n no incio da lista
         newNode->next = l->start;
         l->start = newNode;
      }

      else{
         node *tmp = l->start; // Varivel temporria que comea
            apontando para o primeiro n da lista

         // Varre a lista at encontrar o n na posio pos-1
         for(int i = 1; i < pos-1; i++){ tmp = tmp->next; }

         // Insere o n entre os das posies pos-1 e pos, passando a
            ser o n em pos
         newNode->next = tmp->next;
         tmp->next = newNode;
```

```
        }

        return TRUE;
    }

}
```

Esta função vai receber como parâmetros um filme (tipo ITEM), a posição em que ele deve ser inserido e a lista na qual ele deve ser inserido e retornar um booleano (*TRUE*, caso a inserção seja bem sucedida ou *FALSE*, caso seja mal-sucedida).

As condições para que a inserção seja inválida são:

- A lista recebida não existe (é nula);
- A lista passada está cheia e, portanto, não é possível inserir mais elementos;
- A posição recebida é inválida - o que pode significa duas coisas:
  - A posição é nula ou negativa;
  - A posição não existe (maior do que o tamanho da lista) e não é possível criá-la (maior do que o tamanho da lista + 1).

É importante ressaltar que as checagens de algumas dessas condições são feitas utilizando as funções lista_cheia() e lista_tamanho(), que serão explicadas posteriormente.

Se nenhuma dessas condições for satisfeita, a inserção será possível (o retorno da função será *TRUE*) e função começará por alocar a memória necessária para a criação de um novo nó, cujo membro *movie* irá igualar ao filme recebido como argumento.
Caso a lista esteja vazia ou a posição desejada seja a primeira, basta fazer a variável que indica o início da lista (o ponteiro *start*) apontar para o nó recém-criado e o membro *next* deste apontar para onde aquele antes o fazia (*NULL*, se a lista estava vazia, ou o agora segundo nó, se não).
Caso contrário, a função irá, com auxílio do ponteiro auxiliar *tmp*, percorrer toda a lista até encontrar a posição desejada, que chamaremos de "*pos*- ou, mais precisamente, a posição imediatamente anterior a ela, que chamaremos de "*pos-1*". Quando isso acontecer, o precesso feito será análogo ao descrito no parágrafo anterior: o *next* do nó em *pos-1* passará a apontar para o nó recém-inserido e o deste, para o nó que anteriormente ocupada a *pos* e agora passa a ocupar *pos+1*.

O processo de inserção de um novo nó causará a criação de uma nova posição (correspondente a tamanho da lista + 1) e subsequente incremento deste tamanho.

2.4 Funções para checar se a lista está vazia ou cheia e contar seu tamanho

```
boolean lista_vazia(LISTA *l){
   if(l == NULL || l->start != NULL){ return FALSE; }

   else{ return TRUE; }
}

boolean lista_cheia(LISTA *l){
   if(l == NULL || lista_vazia(l)){ return FALSE; } // Se a lista
      estiver vazia, ela no est cheia;

   else{
      // Como  uma lista com alocao dinmica, no h um mximo inerente
         de elmentos, apenas um ditado
      // pela memria disponvel. Se no for possvel alocar mais
         memria, significa que a lista estaria cheia.
      node *newNode = (node *)malloc(sizeof(node));

      if(newNode == NULL){
         free(newNode);
         return TRUE;
      }
      else{
         free(newNode);
         return FALSE;
      }
   }
}

int lista_tamanho(LISTA *l){
   if(l == NULL){ return ERRO; } // Se a lista fornecida for nula,
      retorna erro.

   else if(lista_vazia(l)){ return 0; } // Se a lista estiver vazia,
      o tamanho  0.

   else{
```

```
        node *tmp = l->start; // Varivel temporria que comea apontando
            para o primeiro n da lista
        int size = 1; // Contagem de ns, que comea em 1 (lista no est
            vazia)

        while(tmp->next != NULL){ // Varre toda a lista, at no haver
            um prximo n
            tmp = tmp->next; // Passa para o prximo n
            size++; // Incrementa a contagem de ns
        }

        return size; // Retorna o tamanho da lista
    }
}
```

Devido à forma como foram implementadas as duas funções anteriores, para checar se a lista está vazia basta verificar se seu *start* aponta para NULL. Se sim, ela está.

A lista_cheia(), por sua vez, é muito mais teórica/hipotética.
Devido à abordagem encadeada dinâmica que foi utilizada para a implementação do TAD, a única situação em que uma lista estaria cheia seria caso não houvesse mais memória disponível para criar novos nós no sistema que está rodando o programa - algo bem remoto no contexto dos computadores atuais e que seria mais palpável, por exemplo, em se tratando de um sistema embarcado. Por isso, esta função não será testada e a situação da lista cheia sequer será considerada nos casos de teste elaborados e descritos no Capítulo 3.
A checagem dessa condição é feita através da tentativa de alocação de um novo nó - caso mal-sucessida (*newNode == NULL*) a lista está cheia; do contrário, não o está.

Por fim, a função lista_tamanho() vai, caso a lista exista e não esteja vazia e com auxílio do ponteiro auxiliar *tmp*, varrer todos os seus elementos, do *start* até o nó cujo *next* aponta para *NULL*. Durante a varredura, é feita a contagem do número de elementos, que será o retorno da função.

2.5 Funções para busca e remoção de elementos da lista

```
int lista_buscar(LISTA *l, int chave){
    if(l == NULL){ return ERRO; }
```

```
    else if(lista_vazia(l)){ return ERRO; } // Se a lista estiver
        vazia,  impossvel conter qualquer chave.

    else{ // Se a lista no estiver vazia:
        node *tmp = l->start; // Varivel temporria que comea apontando
            para o primeiro n da lista
        int position = 1; // Posio do elemento atual apontado por tmp

        while(tmp->movie.chave != chave && tmp->next != NULL){ //
            Varre a lista at encontrar a chave ou chegar no fim
            tmp = tmp->next; // Passa para o prximo elemento
            position++; // Incrementa a posio

        }
        // Se encontrar a chave, retorna a posio dela
        if(tmp->movie.chave == chave){ return position; }

        // Se chegar no fim sem encontrar a chave, retorna erro
        else{ return ERRO; }
    }
}

boolean lista_remover(LISTA *l, int chave){
    int position = lista_buscar(l, chave); // Posio do filme com a
        chave desejada
    if(position == ERRO){ return FALSE; } // Se a funo buscar
        retornar erro, a chave no est contida na lista

    else{
        node *target = NULL; // Varivel temporria que vai apontar para
            o n que contem a chave desejada

        if(position == 1){
            target = l->start; // Faz target apontar para o primeiro n
            l->start = target->next; // Transforma o segundo n no
                primeiro
        }

        else{
```

```
            node *tmp = l->start; // Varivel temporria que comea
               apontando para o primeiro n da lista

            for(int i = 1; i < position-1; i++){ tmp = tmp->next; } //
               Percorre a lista at o anterior ao n em questo
            target = tmp->next; // Faz target apontar para o n em questo

            tmp->next = target->next; // Faz o n em position-1 apontar
               para o n em position+1
        }

        free(target); // Desaloca o n em questo

        return TRUE;
    }

}
```

A função lista_buscar() consiste em uma varredura semelhante às já explicadas, porém esta ocorre até que se encontre o elemento com a chave desejada - retornando, então, a posição do nó correspondente - ou o fim da lista - retornando, então, o código de erro, pois significa que o elemento buscado não está contido na lista.

A de remoção, por sua vez, utiliza a lista_busca() para encontrar a posição do nó com o elemento a ser removido na lista - ou verificar que ele não está nela - e então varre a lista até encontrar a posição anterior à do nó-alvo. O processo feito então é oposto ao desenvolvido na função lista_inserir_pos(): o *next* do nó na posição anterior (ou o *start* da lista, se o nó a ser removido for o primeiro) passará a apontar para onde o *next* do nó-alvo aponta e depois este terá sua memória desalocada - essencialmente, excluindo-o da lista.

2.6 Função para impressão da lista

```
void lista_imprimir(LISTA *l){
    if(l != NULL){
        node *tmp = l->start; // Varivel temporria que comea apontando
            para o possvel primeiro n

        while(tmp != NULL){ // Caso o n atual apontado por tmp exista
```

```
            printf("%d %s\n", tmp->movie.chave, tmp->movie.titulo); //
               Printa a chave e o nome do filme
            tmp = tmp->next; // E passa para o prximo n
         }
      }

      return;
}
```

A função de impressão é simples, ela apenas faz a varredura por todos os elementos da lista enquanto imprime suas chaves e nomes na ordem, de um por um, um por linha, na saída padrão.

2.7 Funções para exclusão da lista

```
void deallocate(node **n){ // Funo local para recursivamente
   desalocar os ns de uma lista
   // Se este n no for o ltimo n da lista, chama a si mesma
      recursivamente no prximo n
   if(*n != NULL && (*n)->next != NULL){ deallocate(&((*n)->next)); }

   // Indo do fim da lista at o incio
   free(*n); // Desaloca a memria do n
   *n = NULL; // Faz o ponteiro daquele n apontar para NULL

   return;
}

boolean lista_apagar(LISTA **l){
   if(*l == NULL){ return FALSE; }

   else{
      deallocate(&((*l)->start)); // Desaloca todos os ns da lista

      free(*l); // Desaloca a lista
      *l = NULL; // Faz o ponteiro da lista apontar para NULL

      return TRUE;
   }
}
```

Para a exclusão da lista foram necessárias duas funções, a principal, lista_apagar(), para excluir a lista em si, e a função local *deallocate*(), para excluir cada um dos nós.

A lista_apagar() não tem uma lógica complicada. Ela recebe um ponteiro que aponta para o ponteiro da lista (a necessidade disso será explicada em breve) e, caso a lista exista, vai chamar a *deallocate*() no *start*. Quanto a exclusão dos nós for concluída, a função desaloca a memória da lista e faz o seu ponteiro apontar para *NULL* - algo que, caso o parâmetro fosse *l, em vez de **l, não seria possível de ser feito dentro da função e justifica o ponteiro duplo.

A *deallocate*(), enfim, é muito semelhante e com a mesma justificativa da outra, recebe um ponteiro duplo para o nó como argumento. Primeiro, ela checa se o nó em que está operando atualmente é o último da lista e, se não for, faz uma chamada recursiva no próximo nó. Ao chegar no último, ela começa a desacolar todos os nós na ordem reversa (do último para o primeiro) e anular seus ponteiros.

2.8 main.c

```
// Constantes para representar os valores que indicam cada uma das
   funes possveis
#define DELETE 2
#define INSERT 3
#define REMOVE 4
#define SEARCH 5
#define PRINT 6
#define COUNT 7
#define IS_FULL 8
#define IS_EMPTY 9
#define EXIT 0

int main(){
   float auxOp, auxNoOp, auxPos, auxKey; // Variveis auxiliar que
      sero utilizadas para receber os inputs (podendo o input ser
      float ou int)
   int operation; // Identifica a operao solicitada pela entrada
   int noOperations; // Identifica quantas operaes do tipo
      'operation' sero realizadas
   int position; // Posio em que ser inserido determinado filme
   LISTA *database = lista_criar(); // Lista para armazenar os filmes
   ITEM movie; // Varivel auxiliar para armazenar nomes e chaves
```

```
while(1){
    scanf("%f", &auxOp);
    operation = (int)auxOp; // Esta linha (e todas as outra
        similares ao longo do cdigo) serve para truncar o valor
        recebido (seja int ou float) em um int

    switch (operation){ // De acordo com o valor de 'operations':
        case DELETE: // Apaga a lista
            printf("%d\n", lista_apagar(&database)); // Apaga a
                lista e printa o resultado

            database = lista_criar(); // Cria uma nova lista

            break;

        case INSERT: // Insere 'noOperations' itens na lista
            scanf("%f", &auxNoOp);
            noOperations = (int)auxNoOp;

            for(int i = 0; i < noOperations; i++){ // Para cada item
                a ser inserido
                scanf("%f %s %f", &auxKey, movie.titulo, &auxPos); //
                    Recebe as informaes
                movie.chave = (int)auxKey;
                position = (int)auxPos;

                printf("%d\n", lista_inserir_pos(database, movie,
                    position)); // Insere na lista e printa o resultado
            }

            break;

        case REMOVE: // Remove 'noOperations' itens da lsita
            scanf("%f", &auxNoOp);
            noOperations = (int)auxNoOp;

            for(int i = 0; i < noOperations; i++){ // Para cada item
                a ser removido
                scanf("%f", &auxKey); // Recebe a chave
```

```
            movie.chave = (int)auxKey;

            printf("%d\n",lista_remover(database,movie.chave));
               // Remove da lista e printa o resultado
         }

         break;

      case SEARCH: // Busca a posio de 'noOperations' itens na
         lista
         scanf("%f", &auxNoOp);
         noOperations = (int)auxNoOp;

         for(int i = 0; i < noOperations; i++){ // Para cada item
            a ser buscado
            scanf("%f", &auxKey); // Recebe a chave
            movie.chave = (int)auxKey;

            printf("%d\n", lista_buscar(database, movie.chave));
               // Printa sua posio na lista
         }

         break;

      case PRINT: // Imprime toda a lista
         lista_imprimir(database);

         break;

      case COUNT: // Conta quantos itens a lista contm atualmente
         printf("%d\n", lista_tamanho(database)); // Printa o
            tamanho da lista

         break;

      case IS_FULL: // Checa se a lista est cheia
         printf("%d\n", lista_cheia(database)); // Printa o
            resultado

         break;
```

```
            case IS_EMPTY: // Checa se a lista est vazia
                printf("%d\n", lista_vazia(database)); // Printa o
                    resultado

                break;

            case EXIT: // Encerra a execuo do programa
                lista_apagar(&database); // Apaga a lista

                return 0; // Encerra o programa
        }

    }

}
```

A *main*() possui uma lógica simples e, para facilitar o desenvolvimento, a leitura e o entendimento do código conta com a definição de constantes relacionadas aos códigos de cada uma das operações enunciadas.

Uma lista é criada no início da execução do programa, durante a qual sempre haverá exatamente uma lista - mesmo que aquela seja excluída por solicitação do usuário, outra será imediatamente criada.
Toda a interação com o usuário se dá dentro de um *loop* infinito, que será quebrado apenas se assim solicitado por aquele, encerrando a execução do programa. A primeira coisa feita é a leitura do input do usuário a indicar qual operação deve ser feita, interpretação que será feta através do *switch...case...* subsequente.
Em geral, será feita a identificação de qual operação deve ser realizada, bem como a coleta de mais informações, se necessário, como o número de vezes que determinada operação deve ser executada ou os dados do elemento a ser inserido, removido ou buscado na lista. Então, a função correspondente é chamada e o seu retorno é impresso na saída padrão.

O único tratamento do input desenvolvido na *main*() é o truncamento de todo e qualquer número decimal recebido em um inteiro - para isso, todo *input* numérico é armazenado em uma variável *float* e então convertido para uma variável *int* para finalmente ser passado como parâmetro para as funções da lista.

3 Casos de Teste

3.1 Descrição

Ao todo, foram elaborados 25 casos de teste (enumerados abaixo) com objetivo de verificar como o programa se comportaria em condições adversas e tentar prever problemas que poderiam ocorrer e, então, validar seu funcionamento.

1. Testar o exemplo do enunciado;
2. Apagar a lista e começar uma nova;
3. Apagar uma lista vazia;
4. Imprimir uma lista vazia;
5. Checar se a lista está cheia;
6. Checar se a lista está vazia;
7. Checar o tamanho de uma lista vazia;
8. Inserir um item numa posição inválida;
9. Inserir um item na última posição;
10. Inserir um item em uma posição qualquer no meio da lista;
11. Inserir um filme cujo nome possui mais de uma palavra separadas por hífens e *underscores*;
12. Remover um item que não está na lista;
13. Remover um item que está na lista;
14. Remover todos os itens de um por um;
15. Buscar um item que não está na lista;
16. Buscar o primeiro item da lista;
17. Buscar o último item da lista;
18. Buscar um item em uma posição qualquer no meio da lista;
19. Testar input de números não inteiros (*floats*);

20. Testar as operações para listas com algumas centenas de elementos (350);
21. Testar operaçõesp ara listas com muitos elementos (50000 elementos) e cujos nomes contém caracteres especiais;
22. Inserir um item repetido *[*]*;
23. Inserir um item com apenas nome repetido *[*]*;
24. Inserir um item com apenas chave repetida *[*]*;
25. Inserir um item cujo nome possui mais de uma palavra separadas por espaços *[*]*;

[*]: Estes últimos 4 casos de teste foram cancelados e não implementados, pois foi-se decidido que fugiam ao escopo do trabalho e suas situações-problema não deveriam ser consideradas. Portanto, totalizaram-se 21 casos de teste úteis.

É importante ressaltar que, devido à grande quantidade elementos que tratam, os casos 20 e 21 foram desenvolvidos com o auxílio de um *script* simples em *Python*, para gerar os *inputs* e *outputs* com números e strings aleatórias e que não se repetem para funcionarem como as chaves e nomes dos filmes, respectivamente.

Na seção 3.2 estão transcritos os casos de teste utilizados, na íntegra.

3.2 Transcrição

01.in

```
3 4
135 Titanic 1
55 Aladdin 1
689 Avatar 1
86 Matrix 1
6
5 2
689
13
4 1
86
7
6
0
```

01.out

```
1
1
1
1
86 Matrix
689 Avatar
55 Aladdin
135 Titanic
2
-32000
1
3
689 Avatar
55 Aladdin
135 Titanic
```

02.in

```
3 4
135 Titanic 1
55 Aladdin 1
689 Avatar 1
86 Matrix 1
6
2
6
3 3
65 Gravity 1
98 Irishman 1
77 Hackers 1
6
0
```

02.out

```
1
1
1
1
86 Matrix
689 Avatar
55 Aladdin
135 Titanic
1
1
1
1
77 Hackers
98 Irishman
65 Gravity
```

03.in

```
3 10
2301 clAgPd 1
4478 JkNLLrsWlSKyGYiN 2
246 vRfuWIBnrzMAXn 3
1073 tbdAffKbobs 4
1846 HbiXFuNWA 5
4760 UbMxOuiiRZPqbrQuQG 6
2370 XGqbSyxKwHcINk 7
3774 QmWad 8
4491 rpHIUlvYkF 9
196 uCIPnAgBhvMdoQ 10
7
9
2
7
9
2
0
```

03.out

```
1
1
1
1
1
1
1
1
1
1
10
0
1
0
1
1
```

04.in

```
3 4
135 Titanic 1
55 Aladdin 1
689 Avatar 1
86 Matrix 1
6
2
6
0
```

04.out

```
1
1
1
1
86 Matrix
689 Avatar
55 Aladdin
135 Titanic
1
```

05.in

```
8
3 20
135 Titanic 1
55 Aladdin 1
689 Avatar 1
86 Matrix 1
65 Gravity 1
98 Irishman 1
77 Hackers 1
339 Spiderverse 1
243 REC 1
76 Clue 1
356 Godfather 1
45 Drive 1
61 Shame 1
399 Millennium 1
294 Parasite 1
412 Inception 1
259 Midsommar 1
331 Tangled 1
289 Persona 1
380 Suspiria 1
8
6
0
```

05.out

```
0
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
0
380 Suspiria
289 Persona
331 Tangled
259 Midsommar
412 Inception
294 Parasite
399 Millennium
61 Shame
45 Drive
356 Godfather
76 Clue
243 REC
339 Spiderverse
77 Hackers
98 Irishman
65 Gravity
86 Matrix
689 Avatar
```

55 Aladdin
135 Titanic

06.in

9
3 20
135 Titanic 1
55 Aladdin 1
689 Avatar 1
86 Matrix 1
65 Gravity 1
98 Irishman 1
77 Hackers 1
339 Spiderverse 1
243 REC 1
76 Clue 1
356 Godfather 1
45 Drive 1
61 Shame 1
399 Millennium 1
294 Parasite 1
412 Inception 1
259 Midsommar 1
331 Tangled 1
289 Persona 1
380 Suspiria 1
9
2
6
9
8
0

06.out

1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
0
1
1
0

07.in

7
0

07.out

0

08.in

```
3 6
65 Gravity 2
98 Irishman 1
77 Hackers 0
77 Hackers -1
77 Hackers 2
339 Spiderverse 7
6
5 4
65
98
77
339
0
```

08.out

```
0
1
0
0
1
0
98 Irishman
77 Hackers
-32000
1
2
-32000
```

09.in

```
3 4
65 Gravity 1
98 Irishman 2
77 Hackers 2
339 Spiderverse 4
6
0
```

09.out

```
1
1
1
1
65 Gravity
77 Hackers
98 Irishman
339 Spiderverse
```

10.in

```
3 4
65 Gravity 1
98 Irishman 2
77 Hackers 2
339 Spiderverse 3
6
0
```

10.out

```
1
1
1
1
65 Gravity
77 Hackers
339 Spiderverse
98 Irishman
```

11.in

3 20
135 Titanic 1
55 Aladdin 1
689 Avatar1 1
86 Matrix-2 1
65 Gravity 1
98 The_Irishman 1
77 Hackers 1
339 Spider-man 1
243 REC 1
76 Clue 1
356 The_Godfather-II 1
45 Baby-Driver 1
61 Shame 1
399 Millennium 1
294 Parasite 1
412 Inception 1
259 Midsommar 1
331 Tangled 1
289 Persona 1
380 Suspiria 1
6
0

11.out

1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
380 Suspiria
289 Persona
331 Tangled
259 Midsommar
412 Inception
294 Parasite
399 Millennium
61 Shame
45 Baby-Driver
356 The_Godfather-II
76 Clue
243 REC
339 Spider-man
77 Hackers
98 The_Irishman
65 Gravity
86 Matrix-2
689 Avatar1
55 Aladdin
135 Titanic

12.in

```
3 4
65 Gravity 1
98 Irishman 2
77 Hackers 2
339 Spiderverse 3
6
4 3
65
65
12
0
```

12.out

```
1
1
1
1
65 Gravity
77 Hackers
339 Spiderverse
98 Irishman
1
0
0
```

13.in

```
3 4
65 Gravity 1
98 Irishman 2
77 Hackers 2
339 Spiderverse 3
6
4 3
65
98
468432
5 1
65
6
0
```

13.out

```
1
1
1
1
65 Gravity
77 Hackers
339 Spiderverse
98 Irishman
1
1
0
-32000
77 Hackers
339 Spiderverse
```

14.in	**14.out**
3 20	1
135 Titanic 1	1
55 Aladdin 1	1
689 Avatar 1	1
86 Matrix 1	1
65 Gravity 1	1
98 Irishman 1	1
77 Hackers 1	1
339 Spiderverse 1	1
243 REC 1	1
76 Clue 1	1
356 Godfather 1	1
45 Drive 1	1
61 Shame 1	1
399 Millennium 1	1
294 Parasite 1	1
412 Inception 1	1
259 Midsommar 1	1
331 Tangled 1	1
289 Persona 1	1
380 Suspiria 1	380 Suspiria
6	289 Persona
7	331 Tangled
4 10	259 Midsommar
135	412 Inception
55	294 Parasite
689	399 Millennium
86	61 Shame
65	45 Drive
98	356 Godfather
77	76 Clue
339	243 REC
243	339 Spiderverse
76	77 Hackers
7	98 Irishman
4 10	65 Gravity
356	86 Matrix
45	689 Avatar
61	55 Aladdin
399	135 Titanic

294
412
259
331
289
380
6
7
3 1
135 Titanic 1
6
7
0

20
1
1
1
1
1
1
1
1
1
1
10
1
1
1
1
1
1
1
1
1
0
1
135 Titanic
1

15.in

```
3 4
65 Gravity 1
98 Irishman 2
77 Hackers 2
339 Spiderverse 3
6
4 2
65
98
5 4
65
98
4745
77
6
2
7
5 4
65
98
4745
77
3 1
65 Gravity 1
5 1
65
0
```

15.out

```
1
1
1
1
65 Gravity
77 Hackers
339 Spiderverse
98 Irishman
1
1
-32000
-32000
-32000
1
77 Hackers
339 Spiderverse
1
0
-32000
-32000
-32000
-32000
1
1
```

16.in

```
3 4
65 Gravity 1
98 Irishman 2
77 Hackers 2
339 Spiderverse 3
6
5 1
65
5 4
65
77
339
98
5 4
98
339
77
65
5 4
98
77
339
65
0
```

16.out

```
1
1
1
1
65 Gravity
77 Hackers
339 Spiderverse
98 Irishman
1
1
2
3
4
4
3
2
1
4
2
3
1
```

17.in

```
3 4
65 Gravity 1
98 Irishman 2
77 Hackers 2
339 Spiderverse 3
6
7
5 1
98
4 1
98
7
5 1
339
4 1
339
7
5 1
77
4 1
77
7
5 1
65
4 1
65
7
6
0
```

17.out

```
1
1
1
1
65 Gravity
77 Hackers
339 Spiderverse
98 Irishman
4
4
1
3
3
1
2
2
1
1
1
1
0
```

18.in

```
3 20
135 Titanic 1
55 Aladdin 1
689 Avatar 1
86 Matrix 1
65 Gravity 1
98 Irishman 1
77 Hackers 1
339 Spiderverse 1
243 REC 1
76 Clue 1
356 Godfather 1
45 Drive 1
61 Shame 1
399 Millennium 1
294 Parasite 1
412 Inception 1
259 Midsommar 1
331 Tangled 1
289 Persona 1
380 Suspiria 1
6
5 7
356
45
98
86
412
689
135
0
```

18.out

```
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
380 Suspiria
289 Persona
331 Tangled
259 Midsommar
412 Inception
294 Parasite
399 Millennium
61 Shame
45 Drive
356 Godfather
76 Clue
243 REC
339 Spiderverse
77 Hackers
98 Irishman
65 Gravity
86 Matrix
689 Avatar
55 Aladdin
135 Titanic
```

	10 9 15 17 5 18 20
19.in	**19.out**
3.341 4.765	1
135.4654 Titanic 1.12	1
55.51 Aladdin 1.9	1
689.653 Avatar 1.543	1
86.52 Matrix 1.54	86 Matrix
6.54	689 Avatar
5.423 2.234	55 Aladdin
689.54	135 Titanic
13.4523	2
4.54 1.542	-32000
86.2	1
7.8	3
6.43	689 Avatar
0.9	55 Aladdin
	135 Titanic

Por conta da exorbitante quantidade de linhas nos arquivos 20.in, 20.out, 21.in e 21.out, eles serão omitidos deste relatório. Em vez deles, segue o script em Python que foi utilizado para gerá-los.

```
# Script to generate random "movies"

import random
import string

size = 50000 # Nmero de filmes
intList = [] # Lista com as chaves dos filmes
strList = [] # Lista com os nomes dos filmes

def randInt(): # Funo que vai criar uma chave nica aleatria para o
```

```
    filme
    while True: # Vai repetir o processo at encontrar uma chave nica
        e = random.randint(0,size*1.5) # Cria um int aleatrio
        if intList.count(e) == 0: # Caso a chave ainda no exista na
            lista (seja nica)
            intList.append(e) # Adiciona-a a lista
            break

    return e

def randStr(): # Funo que vaicriar uma string nica aleatria para ser
    o nome do filme
    while True: # Vai repetir o processo at encontrar uma string nica
        e = ''.join(random.choices(string.ascii_letters +
            string.punctuation, k = random.randint(3,19))) # Cria uma
            string aleatria contendo entre 3 e 19 caracteres, incluindo
            letras minsculas e maisculas e caracteres especiais
        if strList.count(e) == 0: # Caso o nome ainda no exista na
            lista (seja nico)
            strList.append(e) # Adiciona-o na lista
            break

    return e

fIn = open('./test-cases/29.in','w') # Arquivo de input do programa
    em C
fOut = open('./test-cases/29.out','w') # Arquivo do output esperado
    do programa em C

fIn.write("3 %d\n" % size) # Inserir 'size' elementos
for i in range(size):
    fIn.write("%d %s %d\n" % (randInt(), randStr(), (i+1)))
    fOut.write("1\n")

fIn.write("6\n9\n8\n") # Printar todos os elementos e checar se a
    lista est vazia ou cheia
for i in range(size): fOut.write("%d %s\n" % (intList[i],
    strList[i]))
fOut.write("0\n0\n")
```

```
fIn.write("5 %d\n" % size) # Buscar todos os elementos de um por um,
   na ordem
for i in range(size):
   fIn.write("%d\n" % intList[i])
   fOut.write("%d\n" % (i+1))

fIn.write("5 %d\n" % round(size*0.75)) # Buscar aleatoriamente 3/4
   dos elementos da lista
for i in range(round(size*0.75)):
   index = i+random.randint(-i,size-1-i)
   fIn.write("%d\n" % intList[index])
   fOut.write("%d\n" % (index+1))

fIn.write("7\n") # Checar o tamanho da lista
fOut.write("%d\n" % size)

fIn.write("4 %d\n" % round(size*0.25)) # Remover aleatoriamente 1/4
   dos elementos da lista
indexList = [] # Lista que vai conter os ndices na intList das
   chaves dos elementos a serem removidos
for i in range(round(size*0.25)):
   while True: # Repete o processo at encontrar um elemento que
      ainda no foi removido
      index = i+random.randint(-i,size-1-i) # Decide aleatoriamente
         qual elemento remover
      if indexList.count(index) == 0: # Verifica se o elemento j foi
         removido (se est na indexList)
         indexList.append(index) # Caso no, adiciona-o na indexList
         break

   fIn.write("%d\n" % intList[index])
   fOut.write("1\n")

newSize = size - round(size*0.25) # Novo tamanho da lista

fIn.write("7\n") # Checa o tamanho da lista
fOut.write("%d\n" % newSize)

fIn.write("5 %d\n" % round(size*0.25)) # Tenta remover de novo todos
   os elementos que acabaram de ser removidos
```

```
for i in range(round(size*0.25)):
   fIn.write("%d\n" % intList[indexList[i]])
   fOut.write("-32000\n")

fIn.write("2\n") # Apaga a lista
fOut.write("1\n")

fIn.write("7\n8\n9\n0\n") # Checa o tamanho da lista, se ela est
   cheia ou vazia e encerra a execuo do programa
fOut.write("0\n0\n1\n")

fIn.close() # Fecha os arquivos
fOut.close()
```

4 Conclusão

Constatou-se que execução do programa foi bem sucedida em todos os 21 casos de teste utilizados e apresentados no Capítulo 3 - o *output* foi idêntico ao esperado. Por conseguinte, conclui-se que o programa está operando corretamente.