

## Lista 12 de Laboratório de Programação II

Nome : Messias Feres Curi Melo

Matrícula : 2022003764

### Problema 1 - Algoritmos de Pesquisa

#### 1.1

main.c

```
C main.c > main()
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  // Dados Gerais
6  int comp;
7
8  void troca(int* a, int *b){
9      int aux = *a;
10     *a = *b;
11     *b = aux;
12 }
13
14 int* copiaVetor(int* v, int n){
15     int i;
16     int *v2;
17     v2 = (int*) malloc (n*sizeof(int));
18     for(i=0; i<n; i++) v2[i] = v[i];
19     return v2;
20 }
21
22 void imprimeVetor(int* v, int n){
23     int i, prim = 1;
24     printf("[");
25     for(i=0; i<n; i++)
26         if(prim){ printf("%d", v[i]); prim = 0; }
27         else printf(", %d", v[i]);
28     printf("]\n");
29 }
30
31 void preencheAleatorio(int* v, int n, int ini, int fim){
32     int i;
33     for(i=0; i<n; i++)
34         v[i] = ini + rand() % (fim-ini + 1);
35 }
36
37 // Pesquisa Binaria
38 int particao(int *v, int ini, int fim){
39     int i = ini, j = fim;
40     int pivo = v[(ini+fim)/2];
41     while (1) {
42         while(v[i] < pivo){ i++; } //procura algum >= pivo do lado esquerdo
43         while(v[j] > pivo){ j--; } //procura algum <= pivo do lado direito
44
45         if(i<j){
46             troca(&v[i], &v[j]); //troca os elementos encontrados
47             i++;

```

```

48         j--;
49     }else
50         return j; //retorna o local onde foi feita a particao
51     }
52 }
53
54 void QuickSort(int *v, int ini, int fim){
55     if(ini < fim ){
56         int q = particao(v, ini, fim);
57         QuickSort(v, ini, q);
58         QuickSort(v, q+1, fim);
59     }
60 }
61
62 int rec_buscaBinaria(int *v, int ini, int fim, int elem){
63     if(ini > fim) return -1;
64     int meio = (ini + fim)/2;
65     comp++;
66     if(v[meio] == elem)
67         return meio;
68     else
69         if(elem < v[meio])
70             return rec_buscaBinaria(v, ini, meio-1, elem);
71         else
72             return rec_buscaBinaria(v, meio+1, fim, elem);
73 }
74
75 int it_buscaBinaria(int *v, int ini, int fim, int elem){
76     int meio;
77     while(ini <= fim){
78         meio = (ini + fim)/2;
79         comp++;
80         if(elem == v[meio]) return meio;
81         else
82             if(elem < v[meio])
83                 fim = meio-1;
84             else
85                 ini = meio+1;
86     }
87     return -1;
88 }
89
90 // Pesquisa Sequencial
91 int buscaSequencial(int *v, int n, int elem){
92     int i;

```

```

93     for(i=0; i<n; i++){
94         comp++;
95         if(v[i] == elem)
96             return i; //Elemento encontrado
97     }
98     return -1; //Elemento encontrado
99 }
100
101 int main(){
102     srand(time(NULL));
103     comp = 0;
104     clock_t t;
105
106     int *v;
107     int n, x;
108     printf("-> Digite o tamanho do vetor: ");
109     scanf("%d", &n);
110     v = (int*) malloc (n*sizeof(int));
111
112     preencheAleatorio(v, n, 1, n);
113     QuickSort(v, 0, n-1);
114
115     printf("\n-> Digite um elemento para busca: ");
116     scanf("%d", &x);
117
118     int ind;
119
120     t = clock();
121     ind = buscaSequencial(v, n, x);
122     t = clock() - t;
123     printf("\n-> Informacoes Busca Sequencial:\n");
124     printf("- Tempo Execucao: %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
125     printf("- Comparacoes: %d\n", comp);
126
127     if(ind != -1)
128         printf("\n-> O elemento %d foi encontrado na pos %d.\n", x, ind);
129     else
130         printf("\n-> O elemento %d NAO foi encontrado!\n", x);
131
132     comp = 0;
133     t = clock();
134     ind = rec_buscaBinaria(v, 0, n-1, x);
135     t = clock() - t;
136     printf("\n-> Informacoes Busca Binaria Recursiva:\n");

```

```

138     printf("- Tempo Execucao: %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
139     printf("- Comparacoes: %d\n", comp);
140
141     if(ind != -1)
142         printf("\n-> O elemento %d foi encontrado na pos %d.\n", x, ind);
143     else
144         printf("\n-> O elemento %d NAO foi encontrado!\n", x);
145
146     comp = 0;
147     t = clock();
148     ind = it_buscaBinaria(v, 0, n-1, x);
149     t = clock() - t;
150     printf("\n-> Informacoes Busca Binaria Iterativa:\n");
151     printf("- Tempo Execucao: %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
152     printf("- Comparacoes: %d\n", comp);
153
154     if(ind != -1)
155         printf("\n-> O elemento %d foi encontrado na pos %d.\n", x, ind);
156     else
157         printf("\n-> O elemento %d NAO foi encontrado!\n", x);
158
159     free(v);
160     return 0;
161 }

```

## Console

```
messiasfcm@MessiasFCM:/mnt/c/Users/Messi/OneDrive/Área de Trabalho/roteiro12/atv11$ gcc main.c -o tp1
messiasfcm@MessiasFCM:/mnt/c/Users/Messi/OneDrive/Área de Trabalho/roteiro12/atv11$ ./tp1
-> Digite o tamanho do vetor: 100
-> Digite um elemento para busca: 35

-> Informacoes Busca Sequencial:
- Tempo Execucao: 0.000001 seconds.
- Comparacoes: 34

-> O elemento 35 foi encontrado na pos 33.

-> Informacoes Busca Binaria Recursiva:
- Tempo Execucao: 0.000000 seconds.
- Comparacoes: 5

-> O elemento 35 foi encontrado na pos 33.

-> Informacoes Busca Binaria Iterativa:
- Tempo Execucao: 0.000000 seconds.
- Comparacoes: 5

-> O elemento 35 foi encontrado na pos 33.
messiasfcm@MessiasFCM:/mnt/c/Users/Messi/OneDrive/Área de Trabalho/roteiro12/atv11$
```

### 1.2

Para alterar de crescente para decrescente, basta alterar o sinal em algumas circunstâncias, como mostrado a seguir.

#### Códigos alterados em relação à questão anterior

```
int rec_buscaBinaria(int *v, int ini, int fim, int elem){
    if(ini > fim) return -1;
    int meio = (ini + fim)/2;
    comp++;
    if(v[meio] == elem)
        return meio;
    else
        if(elem > v[meio]) // Alteração de sinal AQUI
            return rec_buscaBinaria(v, ini, meio-1, elem);
        else
            return rec_buscaBinaria(v, meio+1, fim, elem);
}
```

```

int it_buscaBinaria(int *v, int ini, int fim, int elem){
    int meio;
    while(ini <= fim){
        meio = (ini + fim)/2;
        comp++;
        if(elem == v[meio]) return meio;
        else
            if(elem > meio) // Alteração de sinal AQUI
                fim = meio-1;
            else
                ini = meio+1;
    }
    return -1;
}

```

## Console

```

messiasfcn@MessiasFCM:/mnt/c/Users/Messi/OneDrive/Área de Trabalho/roteiro12/atv12$ gcc main.c -o tp2
messiasfcn@MessiasFCM:/mnt/c/Users/Messi/OneDrive/Área de Trabalho/roteiro12/atv12$ ./tp2
-> Digite o tamanho do vetor: 100
-> Digite um elemento para busca: 35

-> Informacoes Busca Binaria Recursiva:
- Tempo Execucao: 0.000001 seconds.
- Comparacoes: 7

-> O elemento 35 NAO foi encontrado!

-> Informacoes Busca Binaria Iterativa:
- Tempo Execucao: 0.000000 seconds.
- Comparacoes: 7

-> O elemento 35 NAO foi encontrado!
messiasfcn@MessiasFCM:/mnt/c/Users/Messi/OneDrive/Área de Trabalho/roteiro12/atv12$

```

## 1.3

## main.c

```

atv13 > C main.c > @imprimeVetor(Aluno *,int)
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4  #include <string.h>
5
6  // Dados do Aluno
7  typedef struct {
8      char nome[50];
9      int matricula;
10     float notas[3];
11 } Aluno;
12
13 // Dados Gerais
14 int comp;
15
16 void troca(Aluno *a, Aluno *b){
17     Aluno aux = *a;
18     *a = *b;
19     *b = aux;
20 }
21
22 Aluno* copiaVetor(Aluno *v, int n){
23     int i;
24     Aluno *v2;
25     v2 = (Aluno*)malloc(n * sizeof(Aluno));
26     for(i = 0; i < n; i++){
27         v2[i] = v[i];
28     }
29     return v2;
30 }
31
32 void imprimeVetor(Aluno *v, int n){
33     int i, prim = 1;
34     printf("\n");
35     for(i = 0; i < n; i++){
36         if(prim) {
37             printf("-> Nome: %s - Matricula: %d - Notas: [%f, %f, %f]", v[i].nome, v[i].matricula, v[i].notas[0], v[i].notas[1], v[i].notas[2]);
38             prim = 0;
39         }
40         printf("\n-> Nome: %s - Matricula: %d - Notas: [%f, %f, %f]", v[i].nome, v[i].matricula, v[i].notas[0], v[i].notas[1], v[i].notas[2]);
41     }
42     printf("\n\n");
43 }
44
45 void preencheAleatorio(Aluno *v, int n){
46     int i, j;
47     for(i = 0; i < n; i++){
48         v[i].matricula = 20230000 + i;
49         for (j = 0; j < 3; j++){
50             v[i].notas[j] = 0 + (rand() % 101) / 10.0;
51         }
52     }
53 }
54
55 // Pesquisa Binária
56 int particao(Aluno *v, int ini, int fim){
57     int i = ini, j = fim;
58     int pivo = v[(ini + fim) / 2].matricula;
59     while(1){
60         while(v[i].matricula < pivo){
61             i++;
62         }
63         while(v[j].matricula > pivo){
64             j--;
65         }
66         if(i < j){
67             troca(&v[i], &v[j]);
68             i++;
69             j--;
70         }
71         else
72             return j;
73     }
74 }
75
76 void QuickSort(Aluno *v, int ini, int fim){
77     if(ini < fim){
78         int q = particao(v, ini, fim);
79         QuickSort(v, ini, q);
80         QuickSort(v, q + 1, fim);
81     }
82 }
83
84 int rec_buscaBinariaNome(Aluno *v, int ini, int fim, char *nome){
85     if(ini > fim)
86         return -1;
87     int meio = (ini + fim) / 2;
88     comp++;
89     if(strcmp(v[meio].nome, nome) == 0)
90         return meio;
91     else if(strcmp(nome, v[meio].nome) < 0)
92         return rec_buscaBinariaNome(v, ini, meio - 1, nome);
93     else

```

```

93     return rec_buscaBinariaNome(v, meio + 1, fim, nome);
94 }
95
96 int rec_buscaBinariaMatricula(Aluno *v, int ini, int fim, int matricula){
97     if(ini > fim)
98         return -1;
99     int meio = (ini + fim) / 2;
100     comp++;
101     if(v[meio].matricula == matricula)
102         return meio;
103     else if(matricula < v[meio].matricula)
104         return rec_buscaBinariaMatricula(v, ini, meio - 1, matricula);
105     else
106         return rec_buscaBinariaMatricula(v, meio + 1, fim, matricula);
107 }
108
109 int main(){
110     srand(time(NULL));
111     comp = 0;
112     clock_t t;
113
114     Aluno *v;
115     int n;
116     char nome[50];
117     int matricula;
118
119     printf("\n-> Digite o tamanho do vetor: ");
120     scanf("%d", &n);
121     v = (Aluno *)malloc(n * sizeof(Aluno));
122
123     preencheAleatorio(v, n);
124     QuickSort(v, 0, n - 1);
125
126     printf("\n-> Vetor Ordenado:");
127     imprimeVetor(v, n);
128
129     printf("\n-> Digite um nome para busca: ");
130     scanf("%s", nome);
131
132     int indNome;
133
134     comp = 0;
135     t = clock();
136     indNome = rec_buscaBinariaNome(v, 0, n - 1, nome);
137     t = clock() - t;

```

```

138     printf("\n-> Informacoes Busca Binaria por Nome:\n");
139     printf("- Tempo Execucao: %f seconds.\n", ((float)t) / CLOCKS_PER_SEC);
140     printf("- Comparacoes: %d\n", comp);
141
142     if(indNome != -1)
143         printf("\n-> O aluno %s foi encontrado na pos %d.\n", nome, indNome);
144     else
145         printf("\n-> O aluno %s NAO foi encontrado!\n", nome);
146
147     printf("\n-> Digite uma matricula para busca: ");
148     scanf("%d", &matricula);
149
150     int indMatricula;
151
152     comp = 0;
153     t = clock();
154     indMatricula = rec_buscaBinariaMatricula(v, 0, n - 1, matricula);
155     t = clock() - t;
156     printf("\n-> Informacoes Busca Binaria por Matricula:\n");
157     printf("- Tempo Execucao: %f seconds.\n", ((float)t) / CLOCKS_PER_SEC);
158     printf("- Comparacoes: %d\n", comp);
159
160     if(indMatricula != -1)
161         printf("\n-> O aluno com matricula %d foi encontrado na pos %d.\n", matricula, indMatricula);
162     else
163         printf("\n-> O aluno com matricula %d NAO foi encontrado!\n", matricula);
164
165     free(v);
166     return 0;
167 }
168

```

## Console

```
messiasfcn@MessiasFCM:/mnt/c/Users/Messi/OneDrive/Área de Trabalho/roteiro12/atv13$ gcc main.c -o tp3
messiasfcn@MessiasFCM:/mnt/c/Users/Messi/OneDrive/Área de Trabalho/roteiro12/atv13$ ./tp3
-> Digite o tamanho do vetor: 12

-> Vetor Ordenado:
-> Nome: aluno1 - Matricula: 20230000 - Notas: [2.600000, 2.200000, 4.700000]
-> Nome: aluno2 - Matricula: 20230001 - Notas: [9.600000, 0.300000, 3.000000]
-> Nome: aluno3 - Matricula: 20230002 - Notas: [6.500000, 0.500000, 8.700000]
-> Nome: aluno4 - Matricula: 20230003 - Notas: [7.200000, 8.100000, 5.500000]
-> Nome: aluno5 - Matricula: 20230004 - Notas: [6.500000, 2.100000, 3.000000]
-> Nome: aluno6 - Matricula: 20230005 - Notas: [5.000000, 2.800000, 0.500000]
-> Nome: aluno7 - Matricula: 20230006 - Notas: [5.200000, 0.400000, 3.700000]
-> Nome: aluno8 - Matricula: 20230007 - Notas: [7.100000, 4.300000, 7.600000]
-> Nome: aluno9 - Matricula: 20230008 - Notas: [1.200000, 3.100000, 6.300000]
-> Nome: aluno10 - Matricula: 20230009 - Notas: [3.400000, 5.000000, 1.100000]
-> Nome: aluno11 - Matricula: 20230010 - Notas: [1.800000, 4.200000, 10.000000]
-> Nome: aluno12 - Matricula: 20230011 - Notas: [6.500000, 3.700000, 0.300000]

-> Digite um nome para busca: aluno6

-> Informacoes Busca Binaria por Nome:
- Tempo Execucao: 0.000002 seconds.
- Comparacoes: 1

-> O aluno aluno6 foi encontrado na pos 5.

-> Digite uma matricula para busca: 20230000

-> Informacoes Busca Binaria por Matricula:
- Tempo Execucao: 0.000001 seconds.
- Comparacoes: 3

-> O aluno com matricula 20230000 foi encontrado na pos 0.
messiasfcn@MessiasFCM:/mnt/c/Users/Messi/OneDrive/Área de Trabalho/roteiro12/atv13$
```



## Problema 2 - Tabela Hash

### 2.1

main.c

```
atv21 > C main.c > main0
1  #include <stdio.h>
2  #include "Hash.h"
3
4  void binary(int n){
5      if(n<2)
6          printf("%d", n%2);
7      else{
8          binary(n/2);
9          printf("%d", n%2);
10     }
11 }
12
13 int main() {
14     Hash *H;
15     int resultado;
16
17     H = criaHash(31);
18     printf("\n-> Hashing por divisao(semTratar):\n");
19     insereHash_semTratarDivisao(H, 10);
20     insereHash_semTratarDivisao(H, 16);
21     insereHash_semTratarDivisao(H, 28);
22     imprimeHash(H);
23
24     if(buscaHash_semTratarDivisao(H, 16, &resultado))
25         printf("- Elemento encontrado: %d\n", resultado);
26     else
27         printf("- Elemento não encontrado.\n");
28
29     destroiHash(H);
30
31     printf("\n-> Hashing por multiplicacao(semTratar):\n");
32     H = criaHash(31);
33     insereHash_semTratarMultiplicacao(H, 11);
34     insereHash_semTratarMultiplicacao(H, 3);
35     insereHash_semTratarMultiplicacao(H, 24);
36     imprimeHash(H);
37
38     if(buscaHash_semTratarMultiplicacao(H, 34, &resultado))
39         printf("- Elemento encontrado: %d\n", resultado);
40     else
41         printf("- Elemento não encontrado.\n");
42
43     destroiHash(H);
44
45     printf("\n-> Hashing por dobra(semTratar):\n");
46     H = criaHash(31);
47     insereHash_semTratarDobra(H, 17);
48
49     insereHash_semTratarDobra(H, 12);
50     insereHash_semTratarDobra(H, 21);
51     imprimeHash(H);
52
53     if(buscaHash_semTratarDobra(H, 12, &resultado))
54         printf("- Elemento encontrado: %d\n", resultado);
55     else
56         printf("- Elemento não encontrado.\n");
57
58     destroiHash(H);
59
60     return 0;
}
```

## Hash.h

```
atv21 > C Hash.h > ⓘ imprimeHash(Hash *)
1  /*----- File: Hash.c -----+
2  |Tabela Hash (Dinâmica)         |
3  |                               |
4  |                               |
5  | Implementado por Guilherme C. Pena em 24/11/2023 |
6  +-----+ */
7
8  #ifndef HASH_H
9  #define HASH_H
10
11 #include <stdio.h>
12 #include <stdlib.h>
13 #include <string.h>
14
15 typedef struct{
16     int **tabela;
17     int tam, qtd;
18 }Hash;
19
20
21 Hash* criaHash(int t){
22     Hash* h;
23     h = (Hash*) malloc (sizeof(Hash));
24     if(h != NULL){
25         h->tam = t; h->qtd = 0;
26         h->tabela = (int**) malloc (t*sizeof(int*));
27         if(h->tabela == NULL) return NULL;
28         int i;
29         for(i = 0; i<t; i++)
30             h->tabela[i] = NULL;
31     }
32     return h;
33 }
34
35
36 void destroiHash(Hash *h){
37     if(h != NULL){
38         int i;
39         for(i = 0; i<h->tam; i++)
40             if(h->tabela[i] != NULL)
41                 free(h->tabela[i]);
42         free(h->tabela);
43         free(h);
44     }
45 }
46
47 int chaveDivisao(int chave, int tam){
```

```

48     return (chave & 0x7FFFFFFF) % tam;
49 }
50
51 int chaveMultiplicacao(int chave, int tam){
52     float A = 0.6180339887; //constante: 0 < A < 1
53     float val = chave * A;
54     val = val - (int) val;
55     return (int) (tam * val);
56 }
57
58 int chaveDobra(int chave, int tam){
59     int pos, n_bits = 30;
60
61     int p = 1;
62     int r = p << n_bits;
63     while((chave & r) != r){ n_bits--; r = p << n_bits; }
64
65     n_bits++;
66     pos = chave;
67     while(pos > tam){
68         int metade_bits = n_bits/2;
69         int parte1 = pos >> metade_bits;
70         parte1 = parte1 << metade_bits;
71         int parte2 = pos ^ parte1;
72         parte1 = pos >> metade_bits;
73         pos = parte1 ^ parte2;
74         n_bits = n_bits/2;
75     }
76     return pos;
77 }
78
79 int valorString(char *str){
80     int i, valor = 1;
81     int tam = strlen(str);
82     for(i=0; i<tam; i++){
83         valor = 31*valor + (i+1)*((int) str[i]);
84     }
85     return valor;
86 }
87
88 int insereHash_semTratarDivisao(Hash* h, int elem){
89     if(h == NULL) return 0;
90     int pos = chaveDivisao(elem, h->tam);
91
92     if(h->tabela[pos] == NULL){
93         int* novo = (int*) malloc (sizeof(int));

```

```

93     if(novo == NULL) return 0;
94     *novo = elem;
95     h->tabela[pos] = novo;
96     h->qtd++;
97 }else *(h->tabela[pos]) = elem;
98 return 1;
99 }
100
101 int insereHash_semTratarMultiplicacao(Hash* h, int elem){
102     if(h == NULL) return 0;
103     int pos = chaveMultiplicacao(elem, h->tam);
104
105     if(h->tabela[pos] == NULL){
106         int* novo = (int*) malloc (sizeof(int));
107         if(novo == NULL) return 0;
108         *novo = elem;
109         h->tabela[pos] = novo;
110         h->qtd++;
111     }else *(h->tabela[pos]) = elem;
112     return 1;
113 }
114
115 int insereHash_semTratarDobra(Hash* h, int elem) {
116     if(h == NULL) return 0;
117     int pos = chaveDobra(elem, h->tam);
118
119     if(h->tabela[pos] == NULL){
120         int* novo = (int*) malloc (sizeof(int));
121         if(novo == NULL) return 0;
122         *novo = elem;
123         h->tabela[pos] = novo;
124         h->qtd++;
125     }
126     return 1;
127 }
128
129 int buscaHash_semTratarDivisao(Hash* h, int elem, int *p){
130     if(h == NULL) return 0;
131     int pos = chaveDivisao(elem, h->tam);
132     if(h->tabela[pos] == NULL) return 0;
133     if(*(h->tabela[pos]) == elem){
134         *p = *(h->tabela[pos]);
135         return 1;
136     }
137     return 0;

```

```

138 }
139
140 int buscaHash_semTratarMultiplicacao(Hash* h, int elem, int *p){
141     if(h == NULL) return 0;
142     int pos = chaveMultiplicacao(elem, h->tam);
143     if(h->tabela[pos] == NULL) return 0;
144     if(*(h->tabela[pos]) == elem){
145         *p = *(h->tabela[pos]);
146         return 1;
147     }
148     return 0;
149 }
150
151 int buscaHash_semTratarDobra(Hash* h, int elem, int *p){
152     if(h == NULL) return 0;
153     int pos = chaveDobra(elem, h->tam);
154     if(h->tabela[pos] == NULL) return 0;
155     if(*(h->tabela[pos]) == elem){
156         *p = *(h->tabela[pos]);
157         return 1;
158     }
159     return 0;
160 }
161
162 int sondagemLinear(int pos, int i, int tam){
163     return ( (pos + i) & 0x7FFFFFFF) % tam;
164 }
165
166 int sondagemQuadratica(int pos, int i, int tam){
167     pos = pos + 2*i + 5*i*i;
168     return ( pos & 0x7FFFFFFF) % tam;
169 }
170
171 int sondagemDuploHash(int H1, int chave, int i, int tam){
172     int H2 = chaveDivisao(chave, tam-1) + 1;
173     return ( (H1 + i*H2) & 0x7FFFFFFF) % tam;
174 }
175
176 int insereHash_EnderAberto(Hash* h, int elem){
177     if(h == NULL) return 0;
178     int i, pos, newPos;
179     pos = chaveDivisao(elem, h->tam);
180     for(i=0; i<h->tam; i++){
181         newPos = sondagemLinear(pos, i, h->tam);
182         //newPos = sondagemQuadratica(pos, i, h->tam);
183         //newPos = sondagemDuploHash(pos, elem, i, h->tam);
184         if(h->tabela[newPos] == NULL){

```

```

185     int* novo = (int*) malloc (sizeof(int));
186     if(novo == NULL) return 0;
187     *novo = elem;
188     h->tabela[newPos] = novo;
189     h->qtd++;
190     return 1;
191 }
192 }
193 return 0;
194 }
195
196 int buscaHash_EnderAberto(Hash* h, int elem, int *p){
197     if(h == NULL) return 0;
198     int i, pos, newPos;
199     pos = chaveDivisao(elem, h->tam);
200     for(i=0; i<h->tam; i++){
201         newPos = sondagemLinear(pos, i, h->tam);
202         //newPos = sondagemQuadratica(pos, i, h->tam);
203         //newPos = sondagemDuploHash(pos, elem, i, h->tam);
204         if(h->tabela[newPos] == NULL) return 0;
205         if(*(h->tabela[newPos]) == elem){
206             *p = *(h->tabela[newPos]);
207             return 1;
208         }
209     }
210     return 0;
211 }
212
213 void imprimeHash(Hash *h){
214     if(h == NULL) return;
215     int i;
216     for(i=0; i<h->tam; i++){
217         printf("- %d: ", i);
218         if(h->tabela[i] == NULL) printf("NULL\n");
219         else printf("%d\n", *(h->tabela[i]));
220     }
221 }
222
223 #endif

```

## Console

```
messiasfcm@MessiasFCM:/mnt/c/Users/Messi/OneDrive/Área de Trabalho/roteiro12/atv21$ gcc main.c Hash.h -o tp4
messiasfcm@MessiasFCM:/mnt/c/Users/Messi/OneDrive/Área de Trabalho/roteiro12/atv21$ ./tp4
-> Hashing por divisão(semTratar):
- 0: NULL
- 1: NULL
- 2: NULL
- 3: NULL
- 4: NULL
- 5: NULL
- 6: NULL
- 7: NULL
- 8: NULL
- 9: NULL
- 10: 10
- 11: NULL
- 12: NULL
- 13: NULL
- 14: NULL
- 15: NULL
- 16: 16
- 17: NULL
- 18: NULL
- 19: NULL
- 20: NULL
- 21: NULL
- 22: NULL
- 23: NULL
- 24: NULL
- 25: NULL
- 26: NULL
- 27: NULL
- 28: 28
- 29: NULL
- 30: NULL
- Elemento encontrado: 16

-> Hashing por multiplicação(semTratar):
- 0: NULL
- 1: NULL
- 2: NULL
- 3: NULL
- 4: NULL
- 5: NULL
- 6: NULL
- 7: NULL
- 8: NULL
- 9: NULL
- 10: NULL
- 11: NULL
- 12: NULL
- 13: NULL
- 14: NULL
- 15: NULL
```

```
- 16: NULL
- 17: NULL
- 18: NULL
- 19: NULL
- 20: NULL
- 21: NULL
- 22: NULL
- 23: NULL
- 24: 11
- 25: 24
- 26: 3
- 27: NULL
- 28: NULL
- 29: NULL
- 30: NULL
- Elemento não encontrado.
```

```
-> Hashing por dobra(semTratar):
```

```
- 0: NULL
- 1: NULL
- 2: NULL
- 3: NULL
- 4: NULL
- 5: NULL
- 6: NULL
- 7: NULL
- 8: NULL
- 9: NULL
- 10: NULL
- 11: NULL
- 12: 12
- 13: NULL
- 14: NULL
- 15: NULL
- 16: NULL
- 17: 17
- 18: NULL
- 19: NULL
- 20: NULL
- 21: 21
- 22: NULL
- 23: NULL
- 24: NULL
- 25: NULL
- 26: NULL
- 27: NULL
- 28: NULL
- 29: NULL
- 30: NULL
- Elemento encontrado: 12
```

```
messiasfcm@messiasfcm:/mnt/c/Users/Messi/OneDrive/Área de Trabalho/roteiro12/atv21$
```



## 2.2

### main.c

```
atv22 > C main.c > main0
1  #include <stdio.h>
2  #include "Hash.h"
3
4  void binary(int n){
5      if(n<2)
6          printf("%d", n%2);
7      else{
8          binary(n/2);
9          printf("%d", n%2);
10     }
11 }
12
13 int main() {
14     Hash *H;
15     int resultado;
16
17     H = criaHash(31);
18     printf("-> Hashing por divisao(EnderAberto):\n");
19     insereHash_EnderAbertoDivisao(H, 10);
20     insereHash_EnderAbertoDivisao(H, 16);
21     insereHash_EnderAbertoDivisao(H, 28);
22     imprimeHash(H);
23
24     if(buscaHash_EnderAbertoDivisao(H, 16, &resultado))
25         printf("- Elemento encontrado: %d\n", resultado);
26     else
27         printf("- Elemento não encontrado.\n");
28
29     destroiHash(H);
30
31     printf("\n-> Hashing por multiplicacao(EnderAberto):\n");
32     H = criaHash(31);
33     insereHash_EnderAbertoMultiplicacao(H, 11);
34     insereHash_EnderAbertoMultiplicacao(H, 3);
35     insereHash_EnderAbertoMultiplicacao(H, 24);
36     imprimeHash(H);
37
38     if(buscaHash_EnderAbertoMultiplicacao(H, 34, &resultado))
39         printf("- Elemento encontrado: %d\n", resultado);
40     else
41         printf("- Elemento não encontrado.\n");
42
43     destroiHash(H);
44
45     printf("\n-> Hashing por dobra(EnderAberto):\n");
46     H = criaHash(31);
47     insereHash_EnderAbertoDobra(H, 17);
48
49     insereHash_EnderAbertoDobra(H, 12);
50     insereHash_EnderAbertoDobra(H, 21);
51     imprimeHash(H);
52
53     if(buscaHash_EnderAbertoDobra(H, 12, &resultado))
54         printf("- Elemento encontrado: %d\n", resultado);
55     else
56         printf("- Elemento não encontrado.\n");
57
58     destroiHash(H);
59
60     return 0;
61 }
```

## Hash.h

```
atv22 > C Hash.h > ...
1  /*----- File: Hash.c -----+
2  |Tabela Hash (Dinâmica)      |
3  |                             |
4  |                             |
5  | Implementado por Guilherme C. Pena em 24/11/2023 |
6  +-----+ */
7
8  #ifndef HASH_H
9  #define HASH_H
10
11 #include <stdio.h>
12 #include <stdlib.h>
13 #include <string.h>
14
15 typedef struct{
16     int **tabela;
17     int tam, qtd;
18 }Hash;
19
20
21 Hash* criaHash(int t){
22     Hash* h;
23     h = (Hash*) malloc (sizeof(Hash));
24     if(h != NULL){
25         h->tam = t; h->qtd = 0;
26         h->tabela = (int**) malloc (t*sizeof(int*));
27         if(h->tabela == NULL) return NULL;
28         int i;
29         for(i = 0; i<t; i++){
30             h->tabela[i] = NULL;
31         }
32         return h;
33     }
34
35
36 void destroiHash(Hash *h){
37     if(h != NULL){
38         int i;
39         for(i = 0; i<h->tam; i++){
40             if(h->tabela[i] != NULL)
41                 free(h->tabela[i]);
42             free(h->tabela);
43             free(h);
44         }
45     }
46
47     int chaveDivisao(int chave, int tam){
```

```

48     return (chave & 0x7FFFFFFF) % tam;
49 }
50
51 int chaveMultiplicacao(int chave, int tam){
52     float A = 0.6180339887; //constante: 0 < A < 1
53     float val = chave * A;
54     val = val - (int) val;
55     return (int) (tam * val);
56 }
57
58 int chaveDobra(int chave, int tam){
59     int pos, n_bits = 30;
60
61     int p = 1;
62     int r = p << n_bits;
63     while((chave & r) != r){ n_bits--; r = p << n_bits; }
64
65     n_bits++;
66     pos = chave;
67     while(pos > tam){
68         int metade_bits = n_bits/2;
69         int parte1 = pos >> metade_bits;
70         parte1 = parte1 << metade_bits;
71         int parte2 = pos ^ parte1;
72         parte1 = pos >> metade_bits;
73         pos = parte1 ^ parte2;
74         n_bits = n_bits/2;
75     }
76     return pos;
77 }
78
79 int valorString(char *str){
80     int i, valor = 1;
81     int tam = strlen(str);
82     for(i=0; i<tam; i++){
83         valor = 31*valor + (i+1)*((int) str[i]);
84     }
85     return valor;
86 }
87
88 int insereHash_semTratar(Hash* h, int elem){
89     if(h == NULL) return 0;
90     int pos = chaveDivisao(elem, h->tam);
91
92     if(h->tabela[pos] == NULL){
93         int* novo = (int*) malloc (sizeof(int));

```

```

93     if(novo == NULL) return 0;
94     *novo = elem;
95     h->tabela[pos] = novo;
96     h->qtd++;
97 }else *(h->tabela[pos]) = elem;
98 return 1;
99 }
100
101 int buscaHash_semTratar(Hash* h, int elem, int *p){
102     if(h == NULL) return 0;
103     int pos = chaveDivisao(elem, h->tam);
104     if(h->tabela[pos] == NULL) return 0;
105     if(*(h->tabela[pos]) == elem){
106         *p = *(h->tabela[pos]);
107         return 1;
108     }
109     return 0;
110 }
111
112 int sondagemLinear(int pos, int i, int tam){
113     return ( (pos + i) & 0x7FFFFFFF) % tam;
114 }
115
116 int sondagemQuadratica(int pos, int i, int tam){
117     pos = pos + 2*i + 5*i*i;
118     return ( pos & 0x7FFFFFFF) % tam;
119 }
120
121 int sondagemDuploHash(int H1, int chave, int i, int tam){
122     int H2 = chaveDivisao(chave, tam-1) + 1;
123     return ( (H1 + i*H2) & 0x7FFFFFFF) % tam;
124 }
125
126 int insereHash_EnderAbertoDivisao(Hash* h, int elem){
127     if(h == NULL) return 0;
128     int i, pos, newPos;
129     pos = chaveDivisao(elem, h->tam);
130     for(i=0; i<h->tam; i++){
131         newPos = sondagemLinear(pos, i, h->tam);
132         if(h->tabela[newPos] == NULL){
133             int* novo = (int*) malloc (sizeof(int));
134             if(novo == NULL) return 0;
135             *novo = elem;
136             h->tabela[newPos] = novo;
137             h->qtd++;
138             return 1;

```

```

139     }
140 }
141 return 0;
142 }
143
144 int insereHash_EnderAbertoMultiplicacao(Hash* h, int elem){
145     if(h == NULL) return 0;
146     int i, pos, newPos;
147     pos = chaveMultiplicacao(elem, h->tam);
148     for(i=0; i<h->tam; i++){
149         newPos = sondagemQuadratica(pos, i, h->tam);
150         if(h->tabela[newPos] == NULL){
151             int* novo = (int*) malloc (sizeof(int));
152             if(novo == NULL) return 0;
153             *novo = elem;
154             h->tabela[newPos] = novo;
155             h->qtd++;
156             return 1;
157         }
158     }
159     return 0;
160 }
161
162 int insereHash_EnderAbertoDobra(Hash* h, int elem){
163     if(h == NULL) return 0;
164     int i, pos, newPos;
165     pos = chaveDobra(elem, h->tam);
166     for(i=0; i<h->tam; i++){
167         newPos = sondagemDuploHash(pos, elem, i, h->tam);
168         if(h->tabela[newPos] == NULL){
169             int* novo = (int*) malloc (sizeof(int));
170             if(novo == NULL) return 0;
171             *novo = elem;
172             h->tabela[newPos] = novo;
173             h->qtd++;
174             return 1;
175         }
176     }
177     return 0;
178 }
179
180 int buscaHash_EnderAbertoDivisao(Hash* h, int elem, int *p){
181     if(h == NULL) return 0;
182     int i, pos, newPos;
183     pos = chaveDivisao(elem, h->tam);
184     for(i=0; i<h->tam; i++){
185         newPos = sondagemLinear(pos, i, h->tam);

```

```

185     newPos = sondagemLinear(pos, i, h->tam);
186     if(h->tabela[newPos] == NULL) return 0;
187     if(*(h->tabela[newPos]) == elem){
188         *p = *(h->tabela[newPos]);
189         return 1;
190     }
191 }
192 return 0;
193 }
194
195 int buscaHash_EnderAbertoMultiplicacao(Hash* h, int elem, int *p){
196     if(h == NULL) return 0;
197     int i, pos, newPos;
198     pos = chaveMultiplicacao(elem, h->tam);
199     for(i=0; i<h->tam; i++){
200         newPos = sondagemQuadratica(pos, i, h->tam);
201         if(h->tabela[newPos] == NULL) return 0;
202         if(*(h->tabela[newPos]) == elem){
203             *p = *(h->tabela[newPos]);
204             return 1;
205         }
206     }
207     return 0;
208 }
209
210 int buscaHash_EnderAbertoDobra(Hash* h, int elem, int *p){
211     if(h == NULL) return 0;
212     int i, pos, newPos;
213     pos = chaveDobra(elem, h->tam);
214     for(i=0; i<h->tam; i++){
215         newPos = sondagemDuploHash(pos, elem, i, h->tam);
216         if(h->tabela[newPos] == NULL) return 0;
217         if(*(h->tabela[newPos]) == elem){
218             *p = *(h->tabela[newPos]);
219             return 1;
220         }
221     }
222     return 0;
223 }
224
225 void imprimeHash(Hash *h){
226     if(h == NULL) return;
227
228     int i;
229     for(i=0; i<h->tam; i++){
230         printf("- %d: ", i);
231         if(h->tabela[i] == NULL) printf("NULL\n");
232         else printf("%d\n", *(h->tabela[i]));
233     }
234 }
235 #endif

```

## Console

```
messiasfcm@MessiasFCM:/mnt/c/Users/Messi/OneDrive/Área de Trabalho/roteiro12/atv22$ gcc main.c Hash.h -o tp5
messiasfcm@MessiasFCM:/mnt/c/Users/Messi/OneDrive/Área de Trabalho/roteiro12/atv22$ ./tp5
-> Hashing por divisão(EnderAberto):
- 0: NULL
- 1: NULL
- 2: NULL
- 3: NULL
- 4: NULL
- 5: NULL
- 6: NULL
- 7: NULL
- 8: NULL
- 9: NULL
- 10: 10
- 11: NULL
- 12: NULL
- 13: NULL
- 14: NULL
- 15: NULL
- 16: 16
- 17: NULL
- 18: NULL
- 19: NULL
- 20: NULL
- 21: NULL
- 22: NULL
- 23: NULL
- 24: NULL
- 25: NULL
- 26: NULL
- 27: NULL
- 28: 28
- 29: NULL
- 30: NULL
- Elemento encontrado: 16

-> Hashing por multiplicação(EnderAberto):
- 0: NULL
- 1: NULL
- 2: NULL
- 3: NULL
- 4: NULL
- 5: NULL
- 6: NULL
- 7: NULL
- 8: NULL
- 9: NULL
- 10: NULL
- 11: NULL
- 12: NULL
- 13: NULL
- 14: NULL
- 15: NULL
```

```

- 16: NULL
- 17: NULL
- 18: NULL
- 19: NULL
- 20: NULL
- 21: NULL
- 22: NULL
- 23: NULL
- 24: 11
- 25: 24
- 26: 3
- 27: NULL
- 28: NULL
- 29: NULL
- 30: NULL
- Elemento não encontrado.

-> Hashing por dobra(EnderAberto):
- 0: NULL
- 1: NULL
- 2: NULL
- 3: NULL
- 4: NULL
- 5: NULL
- 6: NULL
- 7: NULL
- 8: NULL
- 9: NULL
- 10: NULL
- 11: NULL
- 12: 12
- 13: NULL
- 14: NULL
- 15: NULL
- 16: NULL
- 17: 17
- 18: NULL
- 19: NULL
- 20: NULL
- 21: 21
- 22: NULL
- 23: NULL
- 24: NULL
- 25: NULL
- 26: NULL
- 27: NULL
- 28: NULL
- 29: NULL
- 30: NULL
- Elemento encontrado: 12
messiasfcm@MessiasFCM:/mnt/c/Users/Messi/OneDrive/Área de Trabalho/roteiro12/atv22$

```

## 2.3

### main.c

```

atv23 > C main.c > ...
1  #include <stdio.h>
2  #include "HashLSE.h"
3
4  int main(){
5
6      Hash *H;
7      H = criaHash(15);
8
9      int i, resultado;
10     for(i=0; i<100; i++){
11         insereHashLSE(H, i);
12
13         insereHashLSE(H, 32);
14         insereHashLSE(H, 32);
15         insereHashLSE(H, 32);
16
17         imprimeHash(H);
18
19         if(buscaHashLSE(H, 32, &resultado))
20             printf("\n- Elemento encontrado: %d\n", resultado);
21         else
22             printf("\n- Elemento não encontrado.\n");
23
24         destroiHash(H);
25         return 0;
26     }

```



## HashLSE.h

```
atv23 > C HashLSEh > insereHashLSE(Hash *,int)
1  /*----- File: HashLSE.c -----+
2  |Tabela Hash (LSE)                |
3  |                                |
4  |                                |
5  | Implementado por Guilherme C. Pena em 24/11/2023 |
6  +-----+ */
7
8  #ifndef HASH_H
9  #define HASH_H
10
11 #include <stdio.h>
12 #include <stdlib.h>
13 #include <string.h>
14 #include "LSE.h"
15
16 typedef struct{
17     Lista **tabela;
18     int tam, qtd;
19 }Hash;
20
21
22 Hash* criaHash(int t){
23     Hash* h;
24     h = (Hash*) malloc (sizeof(Hash));
25     if(h != NULL){
26         h->tam = t; h->qtd = 0;
27         h->tabela = (Lista**) malloc (t*sizeof(Lista*));
28         if(h->tabela == NULL) return NULL;
29         int i;
30         for(i = 0; i<t; i++){
31             h->tabela[i] = NULL;
32         }
33         return h;
34     }
35 }
36
37 void destroiHash(Hash *h){
38     if(h != NULL){
39         int i;
40         for(i = 0; i<h->tam; i++){
41             if(h->tabela[i] != NULL)
42                 destroiLista(h->tabela[i]);
43             free(h->tabela[i]);
44         }
45         free(h);
46     }
47 }
```

```

48 int chaveDivisao(int chave, int tam){
49     return (chave & 0x7FFFFFFF) % tam;
50 }
51
52 int chaveMultiplicacao(int chave, int tam){
53     float A = 0.6180339887; //constante: 0 < A < 1
54     float val = chave * A;
55     val = val - (int) val;
56     return (int) (tam * val);
57 }
58
59 int chaveDobra(int chave, int tam){
60     int pos, n_bits = 30;
61
62     int p = 1;
63     int r = p << n_bits;
64     while((chave & r) != r){ n_bits--; r = p << n_bits; }
65
66     n_bits++;
67     pos = chave;
68     while(pos > tam){
69         int metade_bits = n_bits/2;
70         int parte1 = pos >> metade_bits;
71         parte1 = parte1 << metade_bits;
72         int parte2 = pos ^ parte1;
73         parte1 = pos >> metade_bits;
74         pos = parte1 ^ parte2;
75         n_bits = n_bits/2;
76     }
77     return pos;
78 }
79
80 int valorString(char *str){
81     int i, valor = 1;
82     int tam = strlen(str);
83     for(i=0; i<tam; i++){
84         valor = 31*valor + (i+1)*((int) str[i]);
85     }
86     return valor;
87 }
88
89 int insereHashLSE(Hash* h, int elem){
90     if(h == NULL) return 0;
91     int pos = chaveDivisao(elem, h->tam);
92     if(h->tabela[pos] == NULL)
93         h->tabela[pos] = criaLista();
94
95     insereIni(h->tabela[pos], elem);
96     h->qtd++;
97     return 1;
98 }
99
100 int buscaHashLSE(Hash* h, int elem, int *p){
101     if(h == NULL) return 0;
102     int pos = chaveDivisao(elem, h->tam);
103     if(h->tabela[pos] == NULL) return 0;
104     return listaBuscaElem(h->tabela[pos], elem, p);
105 }
106
107 void imprimeHash(Hash *h){
108     if(h == NULL) return;
109     int i;
110     for(i=0; i<h->tam; i++){
111         printf("%d: ", i);
112         if(h->tabela[i] == NULL) printf("NULL\n");
113         else imprimeLista(h->tabela[i]);
114     }
115 }
116
117 #endif

```

## LSE.h

```
atv23 > C LSE.h > NO > info
1  /*----- File: LSE.h -----+
2  |Lista Simplesmente Encadeada |
3  |                               |
4  |                               |
5  | Implementado por Guilherme C. Pena em 14/09/2023 |
6  +-----+ */
7
8  #ifndef LISTASE_H
9  #define LISTASE_H
10
11 #include <stdio.h>
12 #include <stdlib.h>
13
14 typedef struct NO{
15     int info;
16     struct NO* prox;
17 }NO;
18
19 typedef struct NO* Lista;
20
21 Lista* crialista(){
22     Lista *li;
23     li = (Lista*) malloc (sizeof(Lista));
24     if(li != NULL){
25         *li = NULL;
26     }
27     return li;
28 }
29
30 int listaVazia(Lista *li){
31     if(li == NULL) return 1;
32     if(*li == NULL) return 1;//True - Vazia!
33     return 0;//False - tem elemento!
34 }
35
36 NO* alocarNO(){
37     return (NO*) malloc (sizeof(NO));
38 }
39
40 void liberarNO(NO* q){
41     free(q);
42 }
43
44 int listaBuscaElem(Lista* li, int elem, int *p){
45     if(li == NULL) return 0;
46     NO* aux = *li;
47     while(aux != NULL){
```

```

48     if(aux->info == elem){
49         *p = aux->info;
50         return 1;
51     }
52     aux = aux->prox;
53 }
54 return 0;
55 }
56
57 int insereIni(Lista* li, int elem){
58     if(li == NULL) return 0;
59     NO* novo = alocarNO();
60     if(novo == NULL) return 0;
61     novo->info = elem;
62     novo->prox = *li;
63     *li = novo;
64     return 1;
65 }
66
67 int insereFim(Lista* li, int elem){
68     if(li == NULL) return 0;
69     NO* novo = alocarNO();
70     if(novo == NULL) return 0;
71     novo->info = elem;
72     novo->prox = NULL;
73     if(listaVazia(li)){
74         *li = novo;
75     }else{
76         NO* aux = *li;
77         while(aux->prox != NULL)
78             aux = aux->prox;
79         aux->prox = novo;
80     }
81     return 1;
82 }
83
84 int removeIni(Lista* li){
85     if(li == NULL) return 0;
86     if(listaVazia(li)) return 0;
87     NO* aux = *li;
88     *li = aux->prox;
89     liberarNO(aux);
90     return 1;
91 }
92

```

```

93 int removeFim(Lista* li){
94     if(li == NULL) return 0;
95     if(listaVazia(li)) return 0;
96     NO* ant, *aux = *li;
97     while(aux->prox != NULL){
98         ant = aux;
99         aux = aux->prox;
100     }
101     if(aux == *li)
102         *li = aux->prox;
103     else
104         ant->prox = aux->prox;
105     liberarNO(aux);
106     return 1;
107 }
108
109 void imprimeLista(Lista* li){
110     if(li == NULL) return;
111     if(listaVazia(li)){
112         printf("Lista Vazia!\n");
113         return;
114     }
115     //printf("Elementos:\n");
116     NO* aux = *li;
117     while(aux != NULL){
118         printf("%d ", aux->info);
119         aux = aux->prox;
120     }
121     printf("\n");
122 }
123
124 void destroiLista(Lista* li){
125     if(li != NULL){
126         NO* aux;
127         while((*li) != NULL){
128             aux = *li;
129             *li = (*li)->prox;
130             liberarNO(aux);
131         }
132         free(li);
133     }
134 }
135
136 #endif

```

## Console

```
messiasfcm@MessiasFCM:/mnt/c/Users/Messi/OneDrive/Área de Trabalho/roteiro12/atv23$ gcc main.c HashLSE.h LSE.h -o tp6
messiasfcm@MessiasFCM:/mnt/c/Users/Messi/OneDrive/Área de Trabalho/roteiro12/atv23$ ./tp6
0: 90 75 60 45 30 15 0
1: 91 76 61 46 31 16 1
2: 32 32 32 92 77 62 47 32 17 2
3: 93 78 63 48 33 18 3
4: 94 79 64 49 34 19 4
5: 95 80 65 50 35 20 5
6: 96 81 66 51 36 21 6
7: 97 82 67 52 37 22 7
8: 98 83 68 53 38 23 8
9: 99 84 69 54 39 24 9
10: 85 70 55 40 25 10
11: 86 71 56 41 26 11
12: 87 72 57 42 27 12
13: 88 73 58 43 28 13
14: 89 74 59 44 29 14

- Elemento encontrado: 32
messiasfcm@MessiasFCM:/mnt/c/Users/Messi/OneDrive/Área de Trabalho/roteiro12/atv23$
```