Lista 12 de Laboratório de Programação II
Nome : Messias Feres Curi Melo
Matrícula : 2022003764

Problema 1 - Algoritmos de Pesquisa

1.1

main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// Dados Gerais
int comp;

void troca(int* a, int *b){
    int aux = *a;
    *a = *b;
    *b = aux;
}

int* copiaVetor(int* v, int n){
    int i;
    int *v2;
    v2 = (int*) malloc (n*sizeof(int));
    for(i=0; i<n; i++) v2[i] = v[i];
    return v2;
}

void imprimeVetor(int* v, int n){
    int i, prim = 1;
    printf("[");
    for(i=0; i<n; i++)
        if(prim){ printf("%d", v[i]); prim = 0; }
        else printf(", %d", v[i]);
    printf("]\n");
}

void preencheAleatorio(int* v, int n, int ini, int fim){
    int i;
    for(i=0; i<n; i++)
        v[i] = ini + rand() % (fim-ini + 1);
}

// Pesquisa Binaria
int particao(int *v, int ini, int fim){
    int i = ini, j = fim;
    int pivo = v[(ini+fim)/2];
    while (1) {
        while(v[i] < pivo){ i++; } //procura algum >= pivo do lado esquerdo
        while(v[j] > pivo){ j--; } //procura algum <= pivo do lado direito

        if(i<j){
            troca(&v[i], &v[j]); //troca os elementos encontrados
            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 < 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;
```

```c
    for(i=0; i<n; i++){
        comp++;
        if(v[i] == elem)
            return i; //Elemento encontrado
    }
    return -1; //Elemento encontrado
}

int main(){
    srand(time(NULL));
    comp = 0;
    clock_t t;

    int *v;
    int n, x;
    printf("-> Digite o tamanho do vetor: ");
    scanf("%d", &n);
    v = (int*) malloc (n*sizeof(int));


    preencheAleatorio(v, n, 1, n);
    QuickSort(v, 0, n-1);

    printf("-> Digite um elemento para busca: ");
    scanf("%d", &x);

    int ind;

    t = clock();
    ind = buscaSequencial(v, n, x);
    t = clock() - t;
    printf("\n-> Informacoes Busca Sequencial:\n");
    printf("- Tempo Execucao:  %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
    printf("- Comparacoes: %d\n", comp);

    if(ind != -1)
        printf("\n-> O elemento %d foi encontrado na pos %d.\n", x, ind);
    else
        printf("\n-> O elemento %d NAO foi encontrado!\n", x);

    comp = 0;
    t = clock();
    ind = rec_buscaBinaria(v, 0, n-1, x);
    t = clock() - t;
    printf("\n-> Informacoes Busca Binaria Recursiva:\n");
    printf("- Tempo Execucao:  %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
    printf("- Comparacoes: %d\n", comp);

    if(ind != -1)
        printf("\n-> O elemento %d foi encontrado na pos %d.\n", x, ind);
    else
        printf("\n-> O elemento %d NAO foi encontrado!\n", x);

    comp = 0;
    t = clock();
    ind = it_buscaBinaria(v, 0, n-1, x);
    t = clock() - t;
    printf("\n-> Informacoes Busca Binaria Iterativa:\n");
    printf("- Tempo Execucao:  %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
    printf("- Comparacoes: %d\n", comp);

    if(ind != -1)
        printf("\n-> O elemento %d foi encontrado na pos %d.\n", x, ind);
    else
        printf("\n-> O elemento %d NAO foi encontrado!\n", x);

    free(v);
    return 0;
}
```

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

```c
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);
}
```

```c
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

```
messiasfcm@MessiasFCM:/mnt/c/Users/Messi/OneDrive/Área de Trabalho/roteiro12/atv12$ gcc main.c -o tp2
messiasfcm@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!
messiasfcm@MessiasFCM:/mnt/c/Users/Messi/OneDrive/Área de Trabalho/roteiro12/atv12$
```

1.3

main.c

atv13 > C main.c > imprimeVetor(Aluno *, int)

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

// Dados do Aluno
typedef struct {
    char nome[50];
    int matricula;
    float notas[3];
} Aluno;

// Dados Gerais
int comp;

void troca(Aluno *a, Aluno *b){
    Aluno aux = *a;
    *a = *b;
    *b = aux;
}

Aluno* copiaVetor(Aluno *v, int n){
    int i;
    Aluno *v2;
    v2 = (Aluno*)malloc(n * sizeof(Aluno));
    for(i = 0; i < n; i++)
        v2[i] = v[i];
    return v2;
}

void imprimeVetor(Aluno *v, int n){
    int i, prim = 1;
    printf("\n");
    for(i = 0; i < n; i++){
        if(prim) {
            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]);
            prim = 0;
        }else
            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]);
    }
    printf("\n\n");
}

void preencheAleatorio(Aluno *v, int n){
    int i, j;
    for(i = 0; i < n; i++){
        sprintf(v[i].nome, "aluno%d", i + 1);
        v[i].matricula = 20230000 + i;
        for (j = 0; j < 3; j++)
            v[i].notas[j] = 0 + (rand() % 101) / 10.0;
    }
}

// Pesquisa Binária
int particao(Aluno *v, int ini, int fim){
    int i = ini, j = fim;
    int pivo = v[(ini + fim) / 2].matricula;
    while(1){
        while(v[i].matricula < pivo){
            i++;
        }
        while(v[j].matricula > pivo){
            j--;
        }

        if(i < j){
            troca(&v[i], &v[j]);
            i++;
            j--;
        }else
            return j;
    }
}

void QuickSort(Aluno *v, int ini, int fim){
    if(ini < fim){
        int q = particao(v, ini, fim);
        QuickSort(v, ini, q);
        QuickSort(v, q + 1, fim);
    }
}

int rec_buscaBinariaNome(Aluno *v, int ini, int fim, char *nome){
    if(ini > fim)
        return -1;
    int meio = (ini + fim) / 2;
    comp++;
    if(strcmp(v[meio].nome, nome) == 0)
        return meio;
    else if(strcmp(nome, v[meio].nome) < 0)
        return rec_buscaBinariaNome(v, ini, meio - 1, nome);
    else
```

```
        return rec_buscaBinariaNome(v, meio + 1, fim, nome);
}

int rec_buscaBinariaMatricula(Aluno *v, int ini, int fim, int matricula){
    if(ini > fim)
        return -1;
    int meio = (ini + fim) / 2;
    comp++;
    if(v[meio].matricula == matricula)
        return meio;
    else if(matricula < v[meio].matricula)
        return rec_buscaBinariaMatricula(v, ini, meio - 1, matricula);
    else
        return rec_buscaBinariaMatricula(v, meio + 1, fim, matricula);
}

int main(){
    srand(time(NULL));
    comp = 0;
    clock_t t;

    Aluno *v;
    int n;
    char nome[50];
    int matricula;

    printf("-> Digite o tamanho do vetor: ");
    scanf("%d", &n);
    v = (Aluno *)malloc(n * sizeof(Aluno));

    preencheAleatorio(v, n);
    QuickSort(v, 0, n - 1);

    printf("\n-> Vetor Ordenado:");
    imprimeVetor(v, n);

    printf("\n-> Digite um nome para busca: ");
    scanf("%s", nome);

    int indNome;

    comp = 0;
    t = clock();
    indNome = rec_buscaBinariaNome(v, 0, n - 1, nome);
    t = clock() - t;
```

```
    printf("\n-> Informacoes Busca Binaria por Nome:\n");
    printf("- Tempo Execucao:  %f seconds.\n", ((float)t) / CLOCKS_PER_SEC);
    printf("- Comparacoes: %d\n", comp);

    if(indNome != -1)
        printf("\n-> O aluno %s foi encontrado na pos %d.\n", nome, indNome);
    else
        printf("\n-> O aluno %s NAO foi encontrado!\n", nome);

    printf("\n-> Digite uma matricula para busca: ");
    scanf("%d", &matricula);

    int indMatricula;

    comp = 0;
    t = clock();
    indMatricula = rec_buscaBinariaMatricula(v, 0, n - 1, matricula);
    t = clock() - t;
    printf("\n-> Informacoes Busca Binaria por Matricula:\n");
    printf("- Tempo Execucao:  %f seconds.\n", ((float)t) / CLOCKS_PER_SEC);
    printf("- Comparacoes: %d\n", comp);

    if(indMatricula != -1)
        printf("\n-> O aluno com matricula %d foi encontrado na pos %d.\n", matricula, indMatricula);
    else
        printf("\n-> O aluno com matricula %d NAO foi encontrado!\n", matricula);

    free(v);
    return 0;
}
```

Console

```
messiasfcm@MessiasFCM:/mnt/c/Users/Messi/OneDrive/Área de Trabalho/roteiro12/atv13$ gcc main.c -o tp3
messiasfcm@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.
messiasfcm@MessiasFCM:/mnt/c/Users/Messi/OneDrive/Área de Trabalho/roteiro12/atv13$
```

Problema 2 - Tabela Hash
2.1

main.c

atv21 > C main.c > main()

```c
#include <stdio.h>
#include "Hash.h"

void binary(int n){
    if(n<2)
        printf("%d", n%2);
    else{
        binary(n/2);
      printf("%d", n%2);
    }
}

int main() {
    Hash *H;
    int resultado;

    H = criaHash(31);
    printf("-> Hashing por divisão(semTratar):\n");
    insereHash_semTratarDivisao(H, 10);
    insereHash_semTratarDivisao(H, 16);
    insereHash_semTratarDivisao(H, 28);
    imprimeHash(H);

    if(buscaHash_semTratarDivisao(H, 16, &resultado))
        printf("- Elemento encontrado: %d\n", resultado);
    else
        printf("- Elemento não encontrado.\n");

    destroiHash(H);

    printf("\n-> Hashing por multiplicação(semTratar):\n");
    H = criaHash(31);
    insereHash_semTratarMultiplicacao(H, 11);
    insereHash_semTratarMultiplicacao(H, 3);
    insereHash_semTratarMultiplicacao(H, 24);
    imprimeHash(H);

    if(buscaHash_semTratarMultiplicacao(H, 34, &resultado))
        printf("- Elemento encontrado: %d\n", resultado);
    else
        printf("- Elemento não encontrado.\n");

    destroiHash(H);

    printf("\n-> Hashing por dobra(semTratar):\n");
    H = criaHash(31);
    insereHash_semTratarDobra(H, 17);
    insereHash_semTratarDobra(H, 12);
    insereHash_semTratarDobra(H, 21);
    imprimeHash(H);

    if(buscaHash_semTratarDobra(H, 12, &resultado))
        printf("- Elemento encontrado: %d\n", resultado);
    else
        printf("- Elemento não encontrado.\n");

    destroiHash(H);

    return 0;
}
```

Hash.h

atv21 > C Hash.h > imprimeHash(Hash *)

```c
/*---------------- File: Hash.c  ----------------------+
|Tabela Hash (Dinâmica)                               |
|                                                     |
|                                                     |
| Implementado por Guilherme C. Pena em 24/11/2023    |
+-----------------------------------------------------+ */

#ifndef HASH_H
#define HASH_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct{
    int **tabela;
    int tam, qtd;
}Hash;


Hash* criaHash(int t){
    Hash* h;
    h = (Hash*) malloc (sizeof(Hash));
    if(h != NULL){
        h->tam = t; h->qtd = 0;
        h->tabela = (int**) malloc (t*sizeof(int*));
        if(h->tabela == NULL) return NULL;
        int i;
        for(i = 0; i<t; i++)
            h->tabela[i] = NULL;
    }
    return h;
}


void destroiHash(Hash *h){
    if(h != NULL){
        int i;
        for(i = 0; i<h->tam; i++)
            if(h->tabela[i] != NULL)
                free(h->tabela[i]);
        free(h->tabela);
        free(h);
    }
}

int chaveDivisao(int chave, int tam){
```

```c
    return (chave & 0x7FFFFFFF) % tam;
}

int chaveMultiplicacao(int chave, int tam){
    float A = 0.6180339887; //constante: 0 < A < 1
    float val = chave * A;
    val = val - (int) val;
    return (int) (tam * val);
}

int chaveDobra(int chave, int tam){
    int pos, n_bits = 30;

    int p = 1;
    int r = p << n_bits;
    while((chave & r) != r){ n_bits--; r = p << n_bits; }

    n_bits++;
    pos = chave;
    while(pos > tam){
        int metade_bits = n_bits/2;
        int parte1 = pos >> metade_bits;
        parte1 = parte1 << metade_bits;
        int parte2 = pos ^ parte1;
        parte1 = pos >> metade_bits;
        pos = parte1 ^ parte2;
        n_bits = n_bits/2;
    }
    return pos;
}

int valorString(char *str){
    int i, valor = 1;
    int tam = strlen(str);
    for(i=0; i<tam; i++)
        valor = 31*valor + (i+1)*((int) str[i]);
    return valor;
}

int insereHash_semTratarDivisao(Hash* h, int elem){
    if(h == NULL) return 0;
    int pos = chaveDivisao(elem, h->tam);

    if(h->tabela[pos] == NULL){
        int* novo = (int*) malloc (sizeof(int));
```

```c
        if(novo == NULL) return 0;
        *novo = elem;
        h->tabela[pos] = novo;
        h->qtd++;
    }else *(h->tabela[pos]) = elem;
    return 1;
}

int insereHash_semTratarMultiplicacao(Hash* h, int elem){
    if(h == NULL) return 0;
    int pos = chaveMultiplicacao(elem, h->tam);

    if(h->tabela[pos] == NULL){
        int* novo = (int*) malloc (sizeof(int));
        if(novo == NULL) return 0;
        *novo = elem;
        h->tabela[pos] = novo;
        h->qtd++;
    }else *(h->tabela[pos]) = elem;
    return 1;
}

int insereHash_semTratarDobra(Hash* h, int elem) {
    if(h == NULL) return 0;
    int pos = chaveDobra(elem, h->tam);

    if(h->tabela[pos] == NULL){
        int* novo = (int*) malloc (sizeof(int));
        if(novo == NULL) return 0;
        *novo = elem;
        h->tabela[pos] = novo;
        h->qtd++;
    }
    return 1;
}

int buscaHash_semTratarDivisao(Hash* h, int elem, int *p){
    if(h == NULL) return 0;
    int pos = chaveDivisao(elem, h->tam);
    if(h->tabela[pos] == NULL) return 0;
    if(*(h->tabela[pos]) == elem){
        *p = *(h->tabela[pos]);
        return 1;
    }
    return 0;
```

```c
}

int buscaHash_semTratarMultiplicacao(Hash* h, int elem, int *p){
    if(h == NULL) return 0;
    int pos = chaveMultiplicacao(elem, h->tam);
    if(h->tabela[pos] == NULL) return 0;
    if(*(h->tabela[pos]) == elem){
        *p = *(h->tabela[pos]);
        return 1;
    }
    return 0;
}

int buscaHash_semTratarDobra(Hash* h, int elem, int *p){
    if(h == NULL) return 0;
    int pos = chaveDobra(elem, h->tam);
    if(h->tabela[pos] == NULL) return 0;
    if(*(h->tabela[pos]) == elem){
        *p = *(h->tabela[pos]);
        return 1;
    }
    return 0;
}

int sondagemLinear(int pos, int i, int tam){
    return ( (pos + i) & 0x7FFFFFFF) % tam;
}

int sondagemQuadratica(int pos, int i, int tam){
    pos = pos + 2*i + 5*i*i;
    return ( pos & 0x7FFFFFFF) % tam;
}

int sondagemDuploHash(int H1, int chave, int i, int tam){
    int H2 = chaveDivisao(chave, tam-1) + 1;
    return ( (H1 + i*H2) & 0x7FFFFFFF) % tam;
}

int insereHash_EnderAberto(Hash* h, int elem){
  if(h == NULL) return 0;
  int i, pos, newPos;
  pos = chaveDivisao(elem, h->tam);
  for(i=0; i<h->tam; i++){
    newPos = sondagemLinear(pos, i, h->tam);
    //newPos = sondagemQuadratica(pos, i, h->tam);
    //newPos = sondagemDuploHash(pos, elem, i, h->tam);
    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 > main()

```c
#include <stdio.h>
#include "Hash.h"

void binary(int n){
    if(n<2)
        printf("%d", n%2);
    else{
        binary(n/2);
      printf("%d", n%2);
    }
}

int main() {
    Hash *H;
    int resultado;

    H = criaHash(31);
    printf("-> Hashing por divisão(EnderAberto):\n");
    insereHash_EnderAbertoDivisao(H, 10);
    insereHash_EnderAbertoDivisao(H, 16);
    insereHash_EnderAbertoDivisao(H, 28);
    imprimeHash(H);

    if(buscaHash_EnderAbertoDivisao(H, 16, &resultado))
        printf("- Elemento encontrado: %d\n", resultado);
    else
        printf("- Elemento não encontrado.\n");

    destroiHash(H);

    printf("\n-> Hashing por multiplicação(EnderAberto):\n");
    H = criaHash(31);
    insereHash_EnderAbertoMultiplicacao(H, 11);
    insereHash_EnderAbertoMultiplicacao(H, 3);
    insereHash_EnderAbertoMultiplicacao(H, 24);
    imprimeHash(H);

    if(buscaHash_EnderAbertoMultiplicacao(H, 34, &resultado))
        printf("- Elemento encontrado: %d\n", resultado);
    else
        printf("- Elemento não encontrado.\n");

    destroiHash(H);

    printf("\n-> Hashing por dobra(EnderAberto):\n");
    H = criaHash(31);
    insereHash_EnderAbertoDobra(H, 17);
    insereHash_EnderAbertoDobra(H, 12);
    insereHash_EnderAbertoDobra(H, 21);
    imprimeHash(H);

    if(buscaHash_EnderAbertoDobra(H, 12, &resultado))
        printf("- Elemento encontrado: %d\n", resultado);
    else
        printf("- Elemento não encontrado.\n");

    destroiHash(H);

    return 0;
}
```

Hash.h

atv22 > C Hash.h > ...

```c
/*---------------- File: Hash.c  ----------------------+
|Tabela Hash (Dinâmica)                               |
|                                                     |
|                                                     |
| Implementado por Guilherme C. Pena em 24/11/2023    |
+-----------------------------------------------------+ */

#ifndef HASH_H
#define HASH_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct{
    int **tabela;
    int tam, qtd;
}Hash;


Hash* criaHash(int t){
    Hash* h;
    h = (Hash*) malloc (sizeof(Hash));
    if(h != NULL){
        h->tam = t; h->qtd = 0;
        h->tabela = (int**) malloc (t*sizeof(int*));
        if(h->tabela == NULL) return NULL;
        int i;
        for(i = 0; i<t; i++)
            h->tabela[i] = NULL;
    }
    return h;
}


void destroiHash(Hash *h){
    if(h != NULL){
        int i;
        for(i = 0; i<h->tam; i++)
            if(h->tabela[i] != NULL)
                free(h->tabela[i]);
        free(h->tabela);
        free(h);
    }
}

int chaveDivisao(int chave, int tam){
```

```c
    return (chave & 0x7FFFFFFF) % tam;
}

int chaveMultiplicacao(int chave, int tam){
    float A = 0.6180339887; //constante: 0 < A < 1
    float val = chave * A;
    val = val - (int) val;
    return (int) (tam * val);
}

int chaveDobra(int chave, int tam){
    int pos, n_bits = 30;

    int p = 1;
    int r = p << n_bits;
    while((chave & r) != r){ n_bits--; r = p << n_bits; }

    n_bits++;
    pos = chave;
    while(pos > tam){
        int metade_bits = n_bits/2;
        int parte1 = pos >> metade_bits;
        parte1 = parte1 << metade_bits;
        int parte2 = pos ^ parte1;
        parte1 = pos >> metade_bits;
        pos = parte1 ^ parte2;
        n_bits = n_bits/2;
    }
    return pos;
}

int valorString(char *str){
    int i, valor = 1;
    int tam = strlen(str);
    for(i=0; i<tam; i++)
        valor = 31*valor + (i+1)*((int) str[i]);
    return valor;
}

int insereHash_semTratar(Hash* h, int elem){
    if(h == NULL) return 0;
    int pos = chaveDivisao(elem, h->tam);

    if(h->tabela[pos] == NULL){
        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;
```

```
    }
  }
  return 0;
}

int insereHash_EnderAbertoMultiplicacao(Hash* h, int elem){
  if(h == NULL) return 0;
  int i, pos, newPos;
  pos = chaveMultiplicacao(elem, h->tam);
  for(i=0; i<h->tam; i++){
    newPos = sondagemQuadratica(pos, i, h->tam);
    if(h->tabela[newPos] == NULL){
        int* novo = (int*) malloc (sizeof(int));
        if(novo == NULL) return 0;
        *novo = elem;
        h->tabela[newPos] = novo;
        h->qtd++;
        return 1;
    }
  }
  return 0;
}

int insereHash_EnderAbertoDobra(Hash* h, int elem){
  if(h == NULL) return 0;
  int i, pos, newPos;
  pos = chaveDobra(elem, h->tam);
  for(i=0; i<h->tam; i++){
    newPos = sondagemDuploHash(pos, elem, i, h->tam);
    if(h->tabela[newPos] == NULL){
        int* novo = (int*) malloc (sizeof(int));
        if(novo == NULL) return 0;
        *novo = elem;
        h->tabela[newPos] = novo;
        h->qtd++;
        return 1;
    }
  }
  return 0;
}

int buscaHash_EnderAbertoDivisao(Hash* h, int elem, int *p){
    if(h == NULL) return 0;
    int i, pos, newPos;
    pos = chaveDivisao(elem, h->tam);
    for(i=0; i<h->tam; i++){
        newPos = sondagemLinear(pos, i, h->tam);
```

```
        newPos = sondagemLinear(pos, i, h->tam);
        if(h->tabela[newPos] == NULL) return 0;
        if(*(h->tabela[newPos]) == elem){
            *p = *(h->tabela[newPos]);
            return 1;
        }
    }
    return 0;
}

int buscaHash_EnderAbertoMultiplicacao(Hash* h, int elem, int *p){
    if(h == NULL) return 0;
    int i, pos, newPos;
    pos = chaveMultiplicacao(elem, h->tam);
    for(i=0; i<h->tam; i++){
        newPos = sondagemQuadratica(pos, i, h->tam);
        if(h->tabela[newPos] == NULL) return 0;
        if(*(h->tabela[newPos]) == elem){
            *p = *(h->tabela[newPos]);
            return 1;
        }
    }
    return 0;
}

int buscaHash_EnderAbertoDobra(Hash* h, int elem, int *p){
    if(h == NULL) return 0;
    int i, pos, newPos;
    pos = chaveDobra(elem, h->tam);
    for(i=0; i<h->tam; i++){
        newPos = sondagemDuploHash(pos, elem, i, h->tam);
        if(h->tabela[newPos] == NULL) return 0;
        if(*(h->tabela[newPos]) == elem){
            *p = *(h->tabela[newPos]);
            return 1;
        }
    }
    return 0;
}

void imprimeHash(Hash *h){
    if(h == NULL) return;
```

```
    int i;
    for(i=0; i<h->tam; i++){
        printf("- %d: ", i);
        if(h->tabela[i] == NULL) printf("NULL\n");
        else printf("%d\n", *(h->tabela[i]));
    }
}

#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 > ...

```c
#include <stdio.h>
#include "HashLSE.h"

int main(){

    Hash *H;
    H = criaHash(15);

    int i, resultado;
    for(i=0; i<100; i++)
        insereHashLSE(H, i);

    insereHashLSE(H, 32);
    insereHashLSE(H, 32);
    insereHashLSE(H, 32);

    imprimeHash(H);

    if(buscaHashLSE(H, 32, &resultado))
        printf("\n- Elemento encontrado: %d\n", resultado);
    else
        printf("\n- Elemento não encontrado.\n");

    destroiHash(H);
    return 0;
}
```

HashLSE.h

atv23 > C HashLSE.h > insereHashLSE(Hash *, int)

```c
/*---------------- File: HashLSE.c  --------------------+
|Tabela Hash (LSE)                                      |
|                                                       |
|                                                       |
| Implementado por Guilherme C. Pena em 24/11/2023      |
+-------------------------------------------------------+ */

#ifndef HASH_H
#define HASH_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "LSE.h"

typedef struct{
    Lista **tabela;
    int tam, qtd;
}Hash;


Hash* criaHash(int t){
    Hash* h;
    h = (Hash*) malloc (sizeof(Hash));
    if(h != NULL){
        h->tam = t; h->qtd = 0;
        h->tabela = (Lista**) malloc (t*sizeof(Lista*));
        if(h->tabela == NULL) return NULL;
        int i;
        for(i = 0; i<t; i++)
            h->tabela[i] = NULL;
    }
    return h;
}


void destroiHash(Hash *h){
    if(h != NULL){
        int i;
        for(i = 0; i<h->tam; i++)
            if(h->tabela[i] != NULL)
                destroiLista(h->tabela[i]);
        free(h->tabela);
        free(h);
    }
}

```

```c
int chaveDivisao(int chave, int tam){
    return (chave & 0x7FFFFFFF) % tam;
}

int chaveMultiplicacao(int chave, int tam){
    float A = 0.6180339887; //constante: 0 < A < 1
    float val = chave * A;
    val = val - (int) val;
    return (int) (tam * val);
}

int chaveDobra(int chave, int tam){
    int pos, n_bits = 30;

    int p = 1;
    int r = p << n_bits;
    while((chave & r) != r){ n_bits--; r = p << n_bits; }

    n_bits++;
    pos = chave;
    while(pos > tam){
        int metade_bits = n_bits/2;
        int parte1 = pos >> metade_bits;
        parte1 = parte1 << metade_bits;
        int parte2 = pos ^ parte1;
        parte1 = pos >> metade_bits;
        pos = parte1 ^ parte2;
        n_bits = n_bits/2;
    }
    return pos;
}

int valorString(char *str){
    int i, valor = 1;
    int tam = strlen(str);
    for(i=0; i<tam; i++)
        valor = 31*valor + (i+1)*((int) str[i]);
    return valor;
}

int insereHashLSE(Hash* h, int elem){
    if(h == NULL) return 0;
    int pos = chaveDivisao(elem, h->tam);
    if(h->tabela[pos] == NULL)
        h->tabela[pos] = criaLista();
    insereIni(h->tabela[pos], elem);
    h->qtd++;
    return 1;
}

int buscaHashLSE(Hash* h, int elem, int *p){
    if(h == NULL) return 0;
    int pos = chaveDivisao(elem, h->tam);
    if(h->tabela[pos] == NULL) return 0;
    return listaBuscaElem(h->tabela[pos], elem, p);
}

void imprimeHash(Hash *h){
    if(h == NULL) return;
    int i;
    for(i=0; i<h->tam; i++){
        printf("%d: ", i);
        if(h->tabela[i] == NULL) printf("NULL\n");
        else imprimeLista(h->tabela[i]);
    }
}

#endif
```

LSE.h

atv23 > C LSE.h > NO > info

```c
/*---------------- File: LSE.h  ---------------------+
|Lista Simplesmente Encadeada                        |
|                                                    |
|                                                    |
| Implementado por Guilherme C. Pena em 14/09/2023   |
+----------------------------------------------------+ */

#ifndef LISTASE_H
#define LISTASE_H

#include <stdio.h>
#include <stdlib.h>

typedef struct NO{
    int info;
    struct NO* prox;
}NO;

typedef struct NO* Lista;

Lista* criaLista(){
    Lista *li;
    li = (Lista*) malloc (sizeof(Lista));
    if(li != NULL){
        *li = NULL;
    }
    return li;
}

int listaVazia(Lista *li){
    if(li == NULL) return 1;
    if(*li == NULL) return 1;//True - Vazia!
    return 0;//False - tem elemento!
}

NO* alocarNO(){
    return (NO*) malloc (sizeof(NO));
}

void liberarNO(NO* q){
    free(q);
}

int listaBuscaElem(Lista* li, int elem, int *p){
    if(li == NULL) return 0;
    NO* aux = *li;
    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$
```