Universidade Federal de São João del Rei
Departamento de Ciência da Computação
Curso de Ciência da Computação

Roteiro 12

Adélson de Oliveira Carmo Júnior
212050019

1 Algoritmos de Pesquisa

1.1 Reimplementação

Código

```c
#ifndef _PESQUISA_
#define _PESQUISA_

int getComp();
void setComp(int valor);
int* copiaVetor(int* v, int n);
void imprimeVetor(int* v, int n);
void preencheAleatorio(int* v, int n,
    int ini, int fim);
void troca(int* a, int *b);
int buscaSequencial(int *v, int n, int
    elem);
int particao(int *v, int ini, int fim);
void QuickSort(int *v, int ini, int
    fim);
int rec_buscaBinaria(int *v, int ini,
    int fim, int elem);
int it_buscaBinaria(int *v, int ini,
    int fim, int elem);

#endif
```

codigos/questao11/questao11.h

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "questao11.h"

//Medidas de Complexidade
int comp; //Num. de comparacoes

int getComp(){
    return comp;
}

void setComp(int valor){
    comp = valor;
}

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);
}

void troca(int* a, int *b){
```

```c
  int aux = *a;
  *a = *b;
  *b = aux;
}

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

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++;
            j--;
        }else
            return j; //retorna o local onde foi feita a particao
    }
}

void QuickSort(int *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_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])
            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)
                fim = meio-1;
            else
                ini = meio+1;
    }
    return -1;
}
```

codigos/questao11/questao11.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "questao11.h"

int main(){

    //Atribuicoes iniciais
    srand(time(NULL));
    setComp(0);
    clock_t t;

    int *v1;
    int n, x, ind;

    /* Recebe valores e preenche */
    printf("Digite o tamanho do vetor:\n");
    scanf("%d", &n);
    v1 = (int*) malloc (n*sizeof(int));

    preencheAleatorio(v1, n, 1, 100);
    QuickSort(v1,0,n-1);
    imprimeVetor(v1, n);

    printf("Digite um elemento para busca:\n");
    scanf("%d", &x);
```

```c

    /* Pesquisa Binaria Recursiva */
    t = clock();
    ind = rec_buscaBinaria(v1,0,n-1,x);
    t = clock() - t;
    printf("\nInformacoes da Pesquisa Binaria Recursiva:\n");
    printf("Tempo Execucao:  %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
    printf("Comparacoes: %d\n", getComp());

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

    /* Pesquisa Binaria iterativa */
    t = clock();
    ind = it_buscaBinaria(v1,0,n-1,x);
    t = clock() - t;
    printf("\nInformacoes da Pesquisa Binaria iterativa:\n");
    printf("Tempo Execucao:  %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
    printf("Comparacoes: %d\n", getComp());

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

    /* Pesquisa Sequencial */
    t = clock();
    ind = buscaSequencial(v1,n,x);
    t = clock() - t;
    printf("\nInformacoes da Pesquisa Sequencial:\n");
    printf("Tempo Execucao:  %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
    printf("Comparacoes: %d\n", getComp());

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

    free(v1);

    return 0;
}
```

codigos/questao11/main.c

```
all: questao11.o
  gcc questao11.o main.c -o main

questao11.o: questao11.h questao11.c
  gcc -c questao11.c

clean:
  rm -f questao11.o main
```

codigos/questao11/Makefile

Saída

```
Digite o tamanho do vetor:
10
[6, 8, 19, 26, 30, 32, 39, 49, 58, 94]
Digite um elemento para busca:
49

Informacoes da Pesquisa Binaria Recursiva:
Tempo Execucao:  0.000004 seconds.
Comparacoes: 2
O elemento 49 foi encontrado na pos 7.
```

Figura 1: Questão 1.1 - Saída 1

```
Informacoes da Pesquisa Binaria iterativa:
Tempo Execucao:  0.000003 seconds.
Comparacoes: 4
O elemento 49 foi encontrado na pos 7.

Informacoes da Pesquisa Sequencial:
Tempo Execucao:  0.000003 seconds.
Comparacoes: 12
O elemento 49 foi encontrado na pos 7.
```

Figura 2: Questão 1.1 - Saída 2

1.2 Modificação

Código

```c
#ifndef _PESQUISA_DECRES_
#define _PESQUISA_DECRES_

int getComp();
void setComp(int valor);
int* copiaVetor(int* v, int n);
void imprimeVetor(int* v, int n);
void preencheAleatorio(int* v, int n, int ini, int fim);
void troca(int* a, int *b);
int buscaSequencial(int *v, int n, int elem);
int particao(int *v, int ini, int fim);
void QuickSort(int *v, int ini, int fim);
int rec_buscaBinaria(int *v, int ini, int fim, int elem);
int it_buscaBinaria(int *v, int ini, int fim, int elem);

#endif
```

codigos/questao12/questao12.h

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include "questao12.h"

//Medidas de Complexidade
int comp; //Num. de comparacoes

int getComp(){
    return comp;
}

void setComp(int valor){
    comp = valor;
}

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);
}

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

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

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++;
            j--;
        }else
```

```
68              return j; //retorna o local
                    onde foi feita a particao
69      }
70  }
71
72  void QuickSort(int *v, int ini, int
        fim){
73      if(ini < fim ){
74          int q = particao(v, ini, fim);
75          QuickSort(v, ini, q);
76          QuickSort(v, q+1, fim);
77      }
78  }
79
80  int rec_buscaBinaria(int *v, int ini,
        int fim, int elem){
81      if(ini > fim) return -1;
82      int meio = (ini + fim)/2;
83      comp++;
84      if(v[meio] == elem)
85          return meio;
86      else
87          if(elem > v[meio])
88              return rec_buscaBinaria(v,
                    ini, meio-1, elem);
89          else
90              return rec_buscaBinaria(v,
                    meio+1, fim, elem);
91  }
92
93  int it_buscaBinaria(int *v, int ini,
        int fim, int elem){
94      int meio;
95      while(ini <= fim){
96          meio = (ini + fim)/2;
97          comp++;
98          if(elem == v[meio]) return meio;
99          else
100             if(elem > v[meio])
101                 fim = meio-1;
102             else
103                 ini = meio+1;
104     }
105     return -1;
106 }
```

codigos/questao12/questao12.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4  #include "questao12.h"
5
6  int main(){
7
8      //Atribuicoes iniciais
9      srand(time(NULL));
10     setComp(0);
11     clock_t t;
12
13     int *v1;
14     int n, x, ind;
15
16     /* Recebe valores e preenche */
17     printf("Digite o tamanho do
           vetor:\n");
18     scanf("%d", &n);
19     v1 = (int*) malloc (n*sizeof(int));
20
21     preencheAleatorio(v1, n, 1, 100);
22     QuickSort(v1,0,n-1);
23     imprimeVetor(v1, n);
24
25     printf("Digite um elemento para
           busca:\n");
26     scanf("%d", &x);
27
28     /* Pesquisa Binaria Recursiva */
29     t = clock();
30     ind = rec_buscaBinaria(v1,0,n-1,x);
31     t = clock() - t;
32     printf("\nInformacoes da Pesquisa
           Binaria Recursiva:\n");
33     printf("Tempo Execucao:  %f
           seconds.\n",
           ((float)t)/CLOCKS_PER_SEC);
34     printf("Comparacoes: %d\n",
           getComp());
35
36     if(ind != -1)
37         printf("O elemento %d foi
               encontrado na pos %d.\n", x,
               ind);
38     else
39         printf("O elemento %d NAO foi
               encontrado!\n", x);
40
41     /* Pesquisa Binaria iterativa */
42     t = clock();
43     ind = it_buscaBinaria(v1,0,n-1,x);
44     t = clock() - t;
45     printf("\nInformacoes da Pesquisa
           Binaria iterativa:\n");
46     printf("Tempo Execucao:  %f
           seconds.\n",
           ((float)t)/CLOCKS_PER_SEC);
47     printf("Comparacoes: %d\n",
           getComp());
48
49     if(ind != -1)
50         printf("O elemento %d foi
               encontrado na pos %d.\n", x,
```

```
            ind);
51      else
52          printf("O elemento %d NAO foi
                encontrado!\n", x);
53
54      /* Pesquisa Sequencial */
55      t = clock();
56      ind = buscaSequencial(v1,n,x);
57      t = clock() - t;
58      printf("\nInformacoes da Pesquisa
            Sequencial:\n");
59      printf("Tempo Execucao:  %f
            seconds.\n",
            ((float)t)/CLOCKS_PER_SEC);
60      printf("Comparacoes: %d\n",
            getComp());
61
62      if(ind != -1)
63          printf("O elemento %d foi
                encontrado na pos %d.\n", x,
                ind);
64      else
65          printf("O elemento %d NAO foi
                encontrado!\n", x);
66
67      free(v1);
68
69      return 0;
70  }
```

codigos/questao12/main.c

```
1  all: questao12.o
2    gcc questao12.o main.c -o main
3
4  questao12.o: questao12.h questao12.c
5    gcc -c questao12.c
6
7  clean:
8    rm -f questao12.o main
```

codigos/questao12/Makefile

Saída

```
Digite o tamanho do vetor:
10
[14, 25, 26, 34, 39, 48, 67, 68, 74, 89]
Digite um elemento para busca:
74

Informacoes da Pesquisa Binaria Recursiva:
Tempo Execucao:  0.000012 seconds.
Comparacoes: 3
O elemento 74 NAO foi encontrado!
```

Figura 3: Questão 1.1 - Saída 1

```
Informacoes da Pesquisa Binaria iterativa:
Tempo Execucao:  0.000010 seconds.
Comparacoes: 6
O elemento 74 NAO foi encontrado!

Informacoes da Pesquisa Sequencial:
Tempo Execucao:  0.000010 seconds.
Comparacoes: 15
O elemento 74 foi encontrado na pos 8.
```

Figura 4: Questão 1.1 - Saída 2

1.3 TAD Aluno

Código

```
1  #ifndef _PESQUISA_ALUNO_
2  #define _PESQUISA_ALUNO_
3
4  typedef struct aluno{
5      char nome[50];
6      int matricula;
7      float notas[3];
8  } Aluno;
9
10 int getComp();
11 void setComp(int valor);
12 Aluno* criaAluno(int n);
13 Aluno* copiaAluno(Aluno* a);
14 void imprimeAluno(Aluno* a);
15 void trocaAluno(Aluno* a, Aluno* b);
16 void QuickSortAluno(Aluno* v, int ini,
       int fim);
17 int particaoAluno(Aluno* v, int ini,
       int fim);// ...
18
19 int buscaSequencialPorNome(Aluno* v,
       int n, const char* nome);
```

```c
int rec_buscaBinariaPorNome(Aluno* v, int ini, int fim, const char* nome);
int it_buscaBinariaPorNome(Aluno* v, int ini, int fim, const char* nome);

int buscaSequencialPorMatricula(Aluno* v, int n, int matricula);
int rec_buscaBinariaPorMatricula(Aluno* v, int ini, int fim, int matricula);
int it_buscaBinariaPorMatricula(Aluno* v, int ini, int fim, int matricula);

#endif
```

codigos/questao13/questao13.h

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include "questao13.h"

//Medidas de Complexidade
int comp; //Num. de comparacoes

int getComp(){
    return comp;
}

void setComp(int valor){
    comp = valor;
}

Aluno* criaAluno(int n){
    Aluno *aluno = (Aluno*) malloc (n * sizeof(Aluno));
    for(int i=0; i<n; i++){
        aluno[i].matricula = 0;
        strcpy(aluno[i].nome, "");
        aluno[i].notas[0] = 0;
        aluno[i].notas[1] = 0;
        aluno[i].notas[2] = 0;
    }
    return aluno;
}

Aluno* copiaAluno(Aluno* a) {
    Aluno* novo = (Aluno*)malloc(sizeof(Aluno));
    strcpy(novo->nome, a->nome);
    novo->matricula = a->matricula;
    memcpy(novo->notas, a->notas, sizeof(float) * 3);
    return novo;
}

void imprimeAluno(Aluno* a) {
    printf("Nome: %s, Matricula: %d, Notas: %.2f, %.2f, %.2f\n",
           a->nome, a->matricula, a->notas[0], a->notas[1], a->notas[2]);
}

void trocaAluno(Aluno* a, Aluno* b) {
    Aluno temp;
    memcpy(&temp, a, sizeof(Aluno));
    memcpy(a, b, sizeof(Aluno));
    memcpy(b, &temp, sizeof(Aluno));
}

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

int particaoAluno(Aluno* v, int ini, int fim) {
    Aluno pivo = v[(ini + fim) / 2];
    int i = ini, j = fim;
    while (1) {
        while (strcmp(v[i].nome, pivo.nome) < 0) {
            i++;
        }
        while (strcmp(v[j].nome, pivo.nome) > 0) {
            j--;
        }
        if (i < j) {
            trocaAluno(&v[i], &v[j]);
            i++;
            j--;
        } else {
            return j;
        }
    }
}

int buscaSequencialPorNome(Aluno* v, int n, const char* nome) {
    for (int i = 0; i < n; i++) {
        if (strcmp(v[i].nome, nome) == 0) {
            return i;
        }
    }
```

```
    return -1;
}

int rec_buscaBinariaPorNome(Aluno* v, int ini, int fim, const char* nome) {
    if (ini > fim) return -1;

    int meio = (ini + fim) / 2;
    int comparacao = strcmp(v[meio].nome, nome);

    if (comparacao == 0) {
        return meio;
    } else if (comparacao > 0) {
        return rec_buscaBinariaPorNome(v, ini, meio - 1, nome);
    } else {
        return rec_buscaBinariaPorNome(v, meio + 1, fim, nome);
    }
}

int it_buscaBinariaPorNome(Aluno* v, int ini, int fim, const char* nome) {
    while (ini <= fim) {
        int meio = (ini + fim) / 2;
        int comparacao = strcmp(v[meio].nome, nome);

        if (comparacao == 0) {
            return meio;
        } else if (comparacao > 0) {
            fim = meio - 1;
        } else {
            ini = meio + 1;
        }
    }
    return -1;
}

int buscaSequencialPorMatricula(Aluno* v, int n, int matricula) {
    int i;
    for(i = 0; i < n; i++) {
        comp++;
        if(v[i].matricula == matricula) {
            return i;
        }
    }
    return -1;
}

int rec_buscaBinariaPorMatricula(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(v[meio].matricula > matricula) {
        return rec_buscaBinariaPorMatricula(v, ini, meio - 1, matricula);
    } else {
        return rec_buscaBinariaPorMatricula(v, meio + 1, fim, matricula);
    }
}

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

codigos/questao13/questao13.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include "questao13.h"

int main(){

    //Atribuicoes iniciais
    srand(time(NULL));
    setComp(0);
    clock_t t;

    Aluno *alunos;
    char nome[50];
    int n, matricula, indice;
```

```
17
18      n = 3;
19
20      alunos = criaAluno(n);
21
22      strcpy(alunos[0].nome, "Fulano");
23      alunos[0].matricula = 1;
24      alunos[0].notas[0] = 10;
25      alunos[0].notas[1] = 9;
26      alunos[0].notas[2] = 8;
27
28      strcpy(alunos[1].nome, "Beltrano");
29      alunos[1].matricula = 2;
30      alunos[1].notas[0] = 7;
31      alunos[1].notas[1] = 7;
32      alunos[1].notas[2] = 6;
33
34      strcpy(alunos[2].nome, "Ciclano");
35      alunos[2].matricula = 3;
36      alunos[2].notas[0] = 10;
37      alunos[2].notas[1] = 10;
38      alunos[2].notas[2] = 5;
39
40
41      QuickSortAluno(alunos, 0, n - 1);
42      printf("Vetor Ordenado por
            Nome:\n");
43      for (int i = 0; i < n; i++) {
44          imprimeAluno(&alunos[i]);
45      }
46
47      printf("Digite a matricula de um
            aluno para busca:\n");
48      scanf("%d", &matricula);
49
50      /* Pesquisa Binaria Iterativa por
            Matricula */
51      t = clock();
52      indice =
            it_buscaBinariaPorMatricula(alunos,
            0, n - 1, matricula);
53      t = clock() - t;
54      printf("\nInformacoes da Pesquisa
            Binaria Iterativa por
            Matricula:\n");
55      printf("Tempo Execucao:  %f
            seconds.\n",
            ((float)t)/CLOCKS_PER_SEC);
56      printf("Comparacoes: %d\n",
            getComp());
57
58      if (indice != -1) {
59          printf("O aluno com a matricula
                %d foi encontrado na posicao
                %d.\n", matricula, indice);
60          imprimeAluno(&alunos[indice]);
61      } else {
62          printf("O aluno com a matricula
                %d NAO foi encontrado!\n",
                matricula);
63      }
64
65      /* Pesquisa Binaria Recursiva por
            Matricula */
66      t = clock();
67      indice =
            rec_buscaBinariaPorMatricula(alunos,
            0, n - 1, matricula);
68      t = clock() - t;
69      printf("\nInformacoes da Pesquisa
            Binaria Recursiva por
            Matricula:\n");
70      printf("Tempo Execucao:  %f
            seconds.\n",
            ((float)t)/CLOCKS_PER_SEC);
71      printf("Comparacoes: %d\n",
            getComp());
72
73      if (indice != -1) {
74          printf("O aluno com a matricula
                %d foi encontrado na posicao
                %d.\n", matricula, indice);
75          imprimeAluno(&alunos[indice]);
76      } else {
77          printf("O aluno com a matricula
                %d NAO foi encontrado!\n",
                matricula);
78      }
79
80      /* Pesquisa Sequencial por
            Matricula */
81      t = clock();
82      indice =
            buscaSequencialPorMatricula(alunos,
            n, matricula);
83      t = clock() - t;
84      printf("\nInformacoes da Pesquisa
            Sequencial por Matricula:\n");
85      printf("Tempo Execucao:  %f
            seconds.\n",
            ((float)t)/CLOCKS_PER_SEC);
86      printf("Comparacoes: %d\n",
            getComp());
87
88      if (indice != -1) {
89          printf("O aluno com a matricula
                %d foi encontrado na posicao
                %d.\n", matricula, indice);
90          imprimeAluno(&alunos[indice]);
91      } else {
92          printf("O aluno com a matricula
                %d NAO foi encontrado!\n",
                matricula);
93      }
94
95      getchar();
96      printf("\n\nDigite o nome de um
            aluno para busca:\n");
```

```c
    fgets(nome, sizeof(nome), stdin);

    int len = strlen(nome);
    if (len > 0 && nome[len - 1] == '\n') {
        nome[len - 1] = '\0';
    }

    /* Pesquisa Binaria Iterativa por Nome */
    t = clock();
    indice = it_buscaBinariaPorNome(alunos, 0, n - 1, nome);
    t = clock() - t;
    printf("\nInformacoes da Pesquisa Binaria Iterativa por Nome:\n");
    printf("Tempo Execucao:  %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
    printf("Comparacoes: %d\n", getComp());

    if (indice != -1) {
        printf("O aluno com a nome %s foi encontrado na posicao %d.\n", nome, indice);
        imprimeAluno(&alunos[indice]);
    } else {
        printf("O aluno com a nome %s NAO foi encontrado!\n", nome);
    }

    /* Pesquisa Binaria Recursiva por Nome */
    t = clock();
    indice = rec_buscaBinariaPorNome(alunos, 0, n - 1, nome);
    t = clock() - t;
    printf("\nInformacoes da Pesquisa Binaria Recursiva por Nome:\n");
    printf("Tempo Execucao:  %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
    printf("Comparacoes: %d\n", getComp());

    if (indice != -1) {
        printf("O aluno com a nome %s foi encontrado na posicao %d.\n", nome, indice);
        imprimeAluno(&alunos[indice]);
    } else {
        printf("O aluno com a nome %s NAO foi encontrado!\n", nome);
    }

    /* Pesquisa Sequencial por Nome */
    t = clock();
    indice = buscaSequencialPorNome(alunos, n, nome);
    t = clock() - t;
    printf("\nInformacoes da Pesquisa Sequencial por Nome:\n");
    printf("Tempo Execucao:  %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
    printf("Comparacoes: %d\n", getComp());

    if (indice != -1) {
        printf("O aluno com a nome %s foi encontrado na posicao %d.\n", nome, indice);
        imprimeAluno(&alunos[indice]);
    } else {
        printf("O aluno com a nome %s NAO foi encontrado!\n", nome);
    }

    free(alunos);

    return 0;
}
```

codigos/questao13/main.c

```
all: questao13.o
  gcc questao13.o main.c -o main

questao13.o: questao13.h questao13.c
  gcc -c questao13.c

clean:
  rm -f questao13.o main
```

codigos/questao13/Makefile

Saída

```
Vetor Ordenado por Nome:
Nome: Beltrano, Matricula: 2, Notas: 7.00, 7.00, 6.00
Nome: Ciclano, Matricula: 3, Notas: 10.00, 10.00, 5.00
Nome: Fulano, Matricula: 1, Notas: 10.00, 9.00, 8.00
Digite a matricula de um aluno para busca:
2

Informacoes da Pesquisa Binaria Iterativa por Matricula:
Tempo Execucao:  0.000013 seconds.
Comparacoes: 2
O aluno com a matricula 2 foi encontrado na posicao 0.
Nome: Beltrano, Matricula: 2, Notas: 7.00, 7.00, 6.00
```

Figura 5: Questão 1.1 - Saída 1

```
Informacoes da Pesquisa Binaria Recursiva por Matricula:
Tempo Execucao:  0.000010 seconds.
Comparacoes: 4
O aluno com a matricula 2 foi encontrado na posicao 0.
Nome: Beltrano, Matricula: 2, Notas: 7.00, 7.00, 6.00

Informacoes da Pesquisa Sequencial por Matricula:
Tempo Execucao:  0.000010 seconds.
Comparacoes: 5
O aluno com a matricula 2 foi encontrado na posicao 0.
Nome: Beltrano, Matricula: 2, Notas: 7.00, 7.00, 6.00
```

Figura 6: Questão 1.1 - Saída 2

```
Digite o nome de um aluno para busca:
Fulano

Informacoes da Pesquisa Binaria Iterativa por Nome:
Tempo Execucao:  0.000012 seconds.
Comparacoes: 5
O aluno com a nome Fulano foi encontrado na posicao 2.
Nome: Fulano, Matricula: 1, Notas: 10.00, 9.00, 8.00
```

Figura 7: Questão 1.1 - Saída 3

```
Informacoes da Pesquisa Binaria Recursiva por Nome:
Tempo Execucao:  0.000010 seconds.
Comparacoes: 5
O aluno com a nome Fulano foi encontrado na posicao 2.
Nome: Fulano, Matricula: 1, Notas: 10.00, 9.00, 8.00

Informacoes da Pesquisa Sequencial por Nome:
Tempo Execucao:  0.000010 seconds.
Comparacoes: 5
O aluno com a nome Fulano foi encontrado na posicao 2.
Nome: Fulano, Matricula: 1, Notas: 10.00, 9.00, 8.00
```

Figura 8: Questão 1.1 - Saída 4

2 Tabela Hash

2.1 Reimplementação

Código

```
#ifndef HASH_H
#define HASH_H

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


Hash* criaHash(int t);
void destroiHash(Hash *h);
void preencheAleatorio(int* v, int n, int ini, int fim);
int chaveDivisao(int chave, int tam);
int chaveMultiplicacao(int chave, int tam);
int chaveDobra(int chave, int tam);
int valorString(char *str);
int insereHash_semTratar(Hash* h, int elem);
int buscaHash_semTratar(Hash* h, int elem, int *p);
int sondagemLinear(int pos, int i, int tam);
int sondagemQuadratica(int pos, int i, int tam);
int sondagemDuploHash(int H1, int chave, int i, int tam);
int insereHash_EnderAberto(Hash* h, int elem);
int buscaHash_EnderAberto(Hash* h, int elem, int *p);
void imprimeHash(Hash *h);

#endif
```

codigos/questao21/questao21.h

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "questao21.h"


Hash* criaHash(int t){
```

```
 9      Hash* h;
10      h = (Hash*) malloc (sizeof(Hash));
11      if(h != NULL){
12          h->tam = t; h->qtd = 0;
13          h->tabela = (int**) malloc
                (t*sizeof(int*));
14          if(h->tabela == NULL) return
                NULL;
15          int i;
16          for(i = 0; i<t; i++)
17              h->tabela[i] = NULL;
18      }
19      return h;
20  }
21
22
23  void destroiHash(Hash *h){
24      if(h != NULL){
25          int i;
26          for(i = 0; i<h->tam; i++)
27              if(h->tabela[i] != NULL)
28                  free(h->tabela[i]);
29          free(h->tabela);
30          free(h);
31      }
32  }
33
34  void preencheAleatorio(int* v, int n,
        int ini, int fim){
35      int i;
36      for(i=0; i<n; i++)
37          v[i] = ini + rand() % (fim-ini
                + 1);
38  }
39
40  int chaveDivisao(int chave, int tam){
41      return (chave & 0x7FFFFFFF) % tam;
42  }
43
44  int chaveMultiplicacao(int chave, int
        tam){
45      float A = 0.6180339887;
            //constante: 0 < A < 1
46      float val = chave * A;
47      val = val - (int) val;
48      return (int) (tam * val);
49  }
50
51  int chaveDobra(int chave, int tam){
52      int pos, n_bits = 30;
53
54      int p = 1;
55      int r = p << n_bits;
56      while((chave & r) != r){ n_bits--;
            r = p << n_bits; }
57
58      n_bits++;
59      pos = chave;
60      while(pos > tam){
61          int metade_bits = n_bits/2;
62          int parte1 = pos >> metade_bits;
63          parte1 = parte1 << metade_bits;
64          int parte2 = pos ^ parte1;
65          parte1 = pos >> metade_bits;
66          pos = parte1 ^ parte2;
67          n_bits = n_bits/2;
68      }
69      return pos;
70  }
71
72  int valorString(char *str){
73      int i, valor = 1;
74      int tam = strlen(str);
75      for(i=0; i<tam; i++)
76          valor = 31*valor + (i+1)*((int)
                str[i]);
77      return valor;
78  }
79
80  int insereHash_semTratar(Hash* h, int
        elem){
81      if(h == NULL) return 0;
82      int pos = chaveDivisao(elem,
            h->tam);
83
84      if(h->tabela[pos] == NULL){
85          int* novo = (int*) malloc
                (sizeof(int));
86          if(novo == NULL) return 0;
87          *novo = elem;
88          h->tabela[pos] = novo;
89          h->qtd++;
90      }else *(h->tabela[pos]) = elem;
91      return 1;
92  }
93
94  int buscaHash_semTratar(Hash* h, int
        elem, int *p){
95      if(h == NULL) return 0;
96      int pos = chaveDivisao(elem,
            h->tam);
97      if(h->tabela[pos] == NULL) return 0;
98      if(*(h->tabela[pos]) == elem){
99          *p = *(h->tabela[pos]);
100         return 1;
101     }
102     return 0;
103 }
104
105 int sondagemLinear(int pos, int i, int
        tam){
106     return ( (pos + i) & 0x7FFFFFFF) %
            tam;
107 }
108
109 int sondagemQuadratica(int pos, int i,
        int tam){
110     pos = pos + 2*i + 5*i*i;
```

```
111     return ( pos & 0x7FFFFFFF) % tam;
112 }
113
114 int sondagemDuploHash(int H1, int
        chave, int i, int tam){
115     int H2 = chaveDivisao(chave, tam-1)
            + 1;
116     return ( (H1 + i*H2) & 0x7FFFFFFF)
            % tam;
117 }
118
119 int insereHash_EnderAberto(Hash* h, int
        elem){
120   if(h == NULL) return 0;
121   int i, pos, newPos;
122   pos = chaveDivisao(elem, h->tam);
123   for(i=0; i<h->tam; i++){
124     newPos = sondagemLinear(pos, i,
            h->tam);
125     //newPos = sondagemQuadratica(pos,
            i, h->tam);
126     //newPos = sondagemDuploHash(pos,
            elem, i, h->tam);
127     if(h->tabela[newPos] == NULL){
128         int* novo = (int*) malloc
                (sizeof(int));
129         if(novo == NULL) return 0;
130         *novo = elem;
131         h->tabela[newPos] = novo;
132         h->qtd++;
133         return 1;
134     }
135   }
136   return 0;
137 }
138
139 int buscaHash_EnderAberto(Hash* h, int
        elem, int *p){
140     if(h == NULL) return 0;
141     int i, pos, newPos;
142     pos = chaveDivisao(elem, h->tam);
143     for(i=0; i<h->tam; i++){
144         newPos = sondagemLinear(pos, i,
                h->tam);
145         //newPos =
                sondagemQuadratica(pos, i,
                h->tam);
146         //newPos =
                sondagemDuploHash(pos, elem,
                i, h->tam);
147         if(h->tabela[newPos] == NULL)
                return 0;
148         if(*(h->tabela[newPos]) ==
                elem){
149             *p = *(h->tabela[newPos]);
150             return 1;
151         }
152     }
153     return 0;
154 }
155
156 void imprimeHash(Hash *h){
157     if(h == NULL) return;
158     int i;
159     for(i=0; i<h->tam; i++){
160         printf("%d: ", i);
161         if(h->tabela[i] == NULL)
                printf("NULL\n");
162         else printf("%d\n",
                *(h->tabela[i]));
163     }
164 }
```

codigos/questao21/questao21.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <time.h>
5  #include "questao21.h"
6
7
8  int main() {
9      int buscaElem, x;
10     int tam = 10;
11     int numElementos = 15;
12
13     /* Criacao da tabela hash */
14     Hash* hashDivisao = criaHash(tam);
15     Hash* hashMultiplicacao =
           criaHash(tam);
16     Hash* hashDobra = criaHash(tam);
17
18     /*  Criacao de vetores com valores
           aleatorios */
19     int elementos[numElementos];
20     preencheAleatorio(elementos,
           numElementos, 1, 100);
21
22     /* Insercao e teste de busca com
           funcao de hashing: Divisao */
23     printf("Espalhamento com funcao de
           hashing: Divisao\n");
24     for (int i = 0; i < numElementos;
           i++) {
25         insereHash_semTratar(hashDivisao,
               elementos[i]);
26     }
27     imprimeHash(hashDivisao);
```

```
28
29      printf("Digite um elemento para
            busca:\n");
30      scanf("%d", &x);
31
32      if
            (buscaHash_semTratar(hashDivisao,
            x, &buscaElem)) {
33
34      } else {
35          printf("Elemento %d nao
                encontrado na tabela\n", x);
36      }
37
38      /* Insercao e teste de busca com
            funcao de hashing: Multiplicacao
            */
39      printf("\nEspalhamento com funcao
            de hashing: Multiplicacao\n");
40      for (int i = 0; i <
            sizeof(elementos) /
            sizeof(elementos[0]); i++) {
41          insereHash_semTratar(hashMultiplica
                elementos[i]);
42      }
43      imprimeHash(hashMultiplicacao);
44
45
46      printf("Digite um elemento para
            busca:\n");
47      scanf("%d", &x);
48
49      if
            (buscaHash_semTratar(hashMultiplica
            x, &buscaElem)) {
50          printf("Elemento %d encontrado
                na tabela\n", x);
51      } else {
52          printf("Elemento %d nao
                encontrado na tabela\n", x);
53      }
54
55      /* Insercao e teste de busca com
            funcao de hashing: Dobra */
56      printf("\nEspalhamento com funcao
            de hashing: Dobra\n");
57      for (int i = 0; i <
            sizeof(elementos) /
            sizeof(elementos[0]); i++) {
58          insereHash_semTratar(hashDobra,
                elementos[i]);
59      }
60      imprimeHash(hashDobra);
61
62
63      printf("Digite um elemento para
            busca:\n");
64      scanf("%d", &x);
65
66      if (buscaHash_semTratar(hashDobra,
            x, &buscaElem)) {
67          printf("Elemento %d encontrado
                na tabela\n", x);
68      } else {
69          printf("Elemento %d nao
                encontrado na tabela\n", x);
70      }
71
72      /* Libera a memoria alocada */
73      destroiHash(hashDivisao);
74      destroiHash(hashMultiplicacao);
75      destroiHash(hashDobra);
76
77      return 0;
78  }
```

codigos/questao21/main.c

```
1  all: questao21.o
2    gcc questao21.o main.c -o main
3
4  questao21.o: questao21.h questao21.c
5    gcc -c questao21.c
6
7  clean:
8    rm -f questao21.o main
```

codigos/questao21/Makefile

Saída

2.2 Modificação

Código

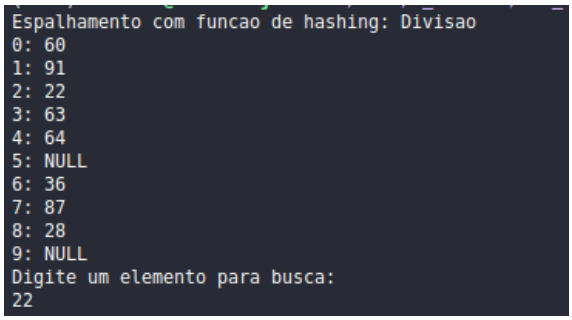

Figura 9: Questão 1.1 - Divisão

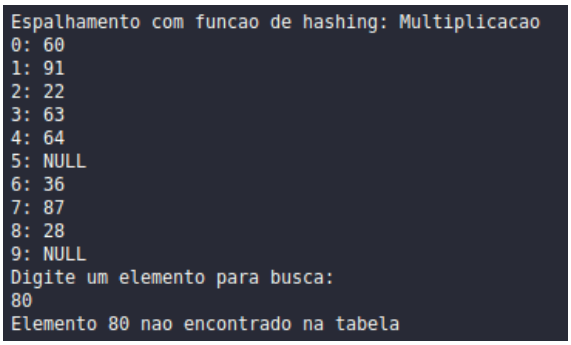

Figura 10: Questão 1.1 - Multiplicação

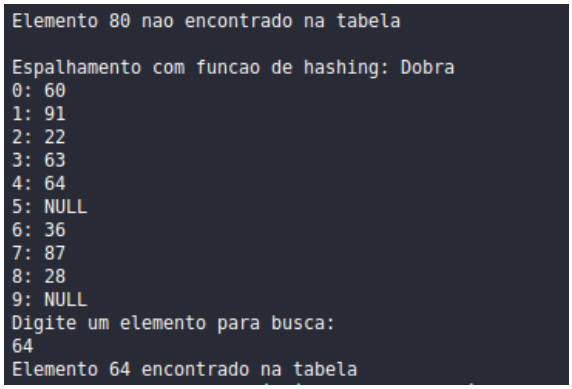

Figura 11: Questão 1.1 - Dobra

```c
#ifndef HASH_H
#define HASH_H

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


Hash* criaHash(int t);
void destroiHash(Hash *h);
void preencheAleatorio(int* v, int n, int ini, int fim);
int chaveDivisao(int chave, int tam);
int chaveMultiplicacao(int chave, int tam);
int chaveDobra(int chave, int tam);
int valorString(char *str);
int insereHash_semTratar(Hash* h, int elem);
int buscaHash_semTratar(Hash* h, int elem, int *p);
int sondagemLinear(int pos, int i, int tam);
int sondagemQuadratica(int pos, int i, int tam);
int sondagemDuploHash(int H1, int chave, int i, int tam);
int escolheSondagem(int pos, int i, int tam, int opcao);
int insereHash_EnderAberto(Hash *h, int elem, int opcaoSondagem);
int buscaHash_EnderAberto(Hash *h, int elem, int *p, int opcaoSondagem);
void imprimeHash(Hash *h);

#endif
```

codigos/questao22/questao22.h

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "questao22.h"

```

```
7
8  Hash* criaHash(int t){
9      Hash* h;
10     h = (Hash*) malloc (sizeof(Hash));
11     if(h != NULL){
12         h->tam = t; h->qtd = 0;
13         h->tabela = (int**) malloc
               (t*sizeof(int*));
14         if(h->tabela == NULL) return
               NULL;
15         int i;
16         for(i = 0; i<t; i++)
17             h->tabela[i] = NULL;
18     }
19     return h;
20 }
21
22
23 void destroiHash(Hash *h){
24     if(h != NULL){
25         int i;
26         for(i = 0; i<h->tam; i++)
27             if(h->tabela[i] != NULL)
28                 free(h->tabela[i]);
29         free(h->tabela);
30         free(h);
31     }
32 }
33
34 void preencheAleatorio(int* v, int n,
       int ini, int fim){
35     int i;
36     for(i=0; i<n; i++)
37         v[i] = ini + rand() % (fim-ini
               + 1);
38 }
39
40 int chaveDivisao(int chave, int tam){
41     return (chave & 0x7FFFFFFF) % tam;
42 }
43
44 int chaveMultiplicacao(int chave, int
       tam){
45     float A = 0.6180339887;
           //constante: 0 < A < 1
46     float val = chave * A;
47     val = val - (int) val;
48     return (int) (tam * val);
49 }
50
51 int chaveDobra(int chave, int tam){
52     int pos, n_bits = 30;
53
54     int p = 1;
55     int r = p << n_bits;
56     while((chave & r) != r){ n_bits--;
           r = p << n_bits; }
57
58     n_bits++;
59     pos = chave;
60     while(pos > tam){
61         int metade_bits = n_bits/2;
62         int parte1 = pos >> metade_bits;
63         parte1 = parte1 << metade_bits;
64         int parte2 = pos ^ parte1;
65         parte1 = pos >> metade_bits;
66         pos = parte1 ^ parte2;
67         n_bits = n_bits/2;
68     }
69     return pos;
70 }
71
72 int valorString(char *str){
73     int i, valor = 1;
74     int tam = strlen(str);
75     for(i=0; i<tam; i++)
76         valor = 31*valor + (i+1)*((int)
               str[i]);
77     return valor;
78 }
79
80 int insereHash_semTratar(Hash* h, int
       elem){
81     if(h == NULL) return 0;
82     int pos = chaveDivisao(elem,
           h->tam);
83
84     if(h->tabela[pos] == NULL){
85         int* novo = (int*) malloc
               (sizeof(int));
86         if(novo == NULL) return 0;
87         *novo = elem;
88         h->tabela[pos] = novo;
89         h->qtd++;
90     }else *(h->tabela[pos]) = elem;
91     return 1;
92 }
93
94 int buscaHash_semTratar(Hash* h, int
       elem, int *p){
95     if(h == NULL) return 0;
96     int pos = chaveDivisao(elem,
           h->tam);
97     if(h->tabela[pos] == NULL) return 0;
98     if(*(h->tabela[pos]) == elem){
99         *p = *(h->tabela[pos]);
100        return 1;
101    }
102    return 0;
103 }
104
105 int sondagemLinear(int pos, int i, int
       tam){
106    return ( (pos + i) & 0x7FFFFFFF) %
           tam;
107 }
108
109 int sondagemQuadratica(int pos, int i,
```

```
    int tam){
110     pos = pos + 2*i + 5*i*i;
111     return ( pos & 0x7FFFFFFF) % tam;
112 }
113
114 int sondagemDuploHash(int H1, int
    chave, int i, int tam){
115     int H2 = chaveDivisao(chave, tam-1)
        + 1;
116     return ( (H1 + i*H2) & 0x7FFFFFFF)
        % tam;
117 }
118
119 int escolheSondagem(int pos, int i, int
    tam, int opcao) {
120     switch (opcao) {
121         case 1:
122             return sondagemLinear(pos,
                i, tam);
123         case 2:
124             return
                sondagemQuadratica(pos,
                i, tam);
125         case 3:
126             return
                sondagemDuploHash(pos,
                pos, i, tam);
127         default:
128             return -1;
129     }
130 }
131
132 int insereHash_EnderAberto(Hash *h, int
    elem, int opcaoSondagem) {
133     if (h == NULL) return 0;
134     int i, pos, newPos;
135     pos = chaveDivisao(elem, h->tam);
136     for (i = 0; i < h->tam; i++) {
137         newPos = escolheSondagem(pos,
            i, h->tam, opcaoSondagem);
138         if (h->tabela[newPos] == NULL) {
139             int *novo = (int
                *)malloc(sizeof(int));
140             if (novo == NULL) return 0;
141             *novo = elem;
142             h->tabela[newPos] = novo;
143             h->qtd++;
144             return 1;
145         }
146     }
147     return 0;
148 }
149
150 int buscaHash_EnderAberto(Hash *h, int
    elem, int *p, int opcaoSondagem) {
151     if (h == NULL) return 0;
152     int i, pos, newPos;
153     pos = chaveDivisao(elem, h->tam);
154     for (i = 0; i < h->tam; i++) {
155         newPos = escolheSondagem(pos,
            i, h->tam, opcaoSondagem);
156         if (h->tabela[newPos] == NULL)
            return 0;
157         if (*(h->tabela[newPos]) ==
            elem) {
158             *p = *(h->tabela[newPos]);
159             return 1;
160         }
161     }
162     return 0;
163 }
164
165 void imprimeHash(Hash *h){
166     if(h == NULL) return;
167     int i;
168     for(i=0; i<h->tam; i++){
169         printf("%d: ", i);
170         if(h->tabela[i] == NULL)
            printf("NULL\n");
171         else printf("%d\n",
            *(h->tabela[i]));
172     }
173 }
```

codigos/questao22/questao22.c

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <string.h>
 4 #include <time.h>
 5 #include "questao22.h"
 6
 7 int main() {
 8     int buscaElem, x;
 9     int tam = 10;
10     int numElementos = 15;
11
12     /* Criacao da tabela hash */
13     Hash *hashDivisao = criaHash(tam);
14     Hash *hashMultiplicacao =
           criaHash(tam);
15     Hash *hashDobra = criaHash(tam);
16
17     /*  Criacao de vetores com valores
           aleatorios */
18     int elementos[numElementos];
19     preencheAleatorio(elementos,
           numElementos, 1, 100);
20
21     /* Insercao e teste de busca com
```

```c
        funcao de hashing: Divisao */
    printf("Espalhamento com funcao de hashing: Divisao\n");
    for (int i = 0; i < numElementos; i++) {
        insereHash_EnderAberto(hashDivisao, elementos[i], 1);
    }
    imprimeHash(hashDivisao);

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

    if (buscaHash_EnderAberto(hashDivisao, x, &buscaElem, 1)) {
        printf("Elemento %d encontrado na tabela\n", x);
    } else {
        printf("Elemento %d nao encontrado na tabela\n", x);
    }

    /* Insercao e teste de busca com funcao de hashing: Multiplicacao */
    printf("\nEspalhamento com funcao de hashing: Multiplicacao\n");
    for (int i = 0; i < numElementos; i++) {
        insereHash_EnderAberto(hashMultiplicacao, elementos[i], 2);
    }
    imprimeHash(hashMultiplicacao);

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

    if (buscaHash_EnderAberto(hashMultiplicacao, x, &buscaElem, 2)) {
        printf("Elemento %d encontrado na tabela\n", x);
    } else {
        printf("Elemento %d nao encontrado na tabela\n", x);
    }

    /* Insercao e teste de busca com funcao de hashing: Dobra */
    printf("\nEspalhamento com funcao de hashing: Dobra\n");
    for (int i = 0; i < numElementos; i++) {
        insereHash_EnderAberto(hashDobra, elementos[i], 3); // 3: Sondagem Duplo Hash
    }
    imprimeHash(hashDobra);

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

    if (buscaHash_EnderAberto(hashDobra, x, &buscaElem, 3)) { // 3: Sondagem Duplo Hash
        printf("Elemento %d encontrado na tabela\n", x);
    } else {
        printf("Elemento %d nao encontrado na tabela\n", x);
    }

    /* Libera a memoria alocada */
    destroiHash(hashDivisao);
    destroiHash(hashMultiplicacao);
    destroiHash(hashDobra);

    return 0;
}
```

codigos/questao22/main.c

```
all: questao22.o
  gcc questao22.o main.c -o main

questao22.o: questao22.h questao22.c
  gcc -c questao22.c

clean:
  rm -f questao22.o main
```

codigos/questao22/Makefile

Saída

```
Espalhamento com funcao de hashing: Divisao
0: 87
1: 50
2: 22
3: 93
4: 84
5: 94
6: 16
7: 87
8: 78
9: 36
Digite um elemento para busca:
20
Elemento 20 nao encontrado na tabela
```

Figura 12: Questão 1.1 - Divisão

```
Espalhamento com funcao de hashing: Multiplicacao
0: 93
1: 94
2: 50
3: 36
4: 84
5: 87
6: 16
7: 87
8: 78
9: 22
Digite um elemento para busca:
16
Elemento 16 encontrado na tabela
```

Figura 13: Questão 1.1 - Multiplicação

```
Espalhamento com funcao de hashing: Dobra
0: 50
1: 93
2: 22
3: 36
4: 84
5: 87
6: 16
7: 87
8: 78
9: 94
Digite um elemento para busca:
50
Elemento 50 encontrado na tabela
```

Figura 14: Questão 1.1 - Dobra

2.3 Simplesmente encadeada

Código

```c
#ifndef LSE_H
#define LSE_H

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

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

typedef struct NO* Lista;

Lista* criaLista();
int listaVazia(Lista *li);
NO* alocarNO();
void liberarNO(NO* q);
int listaBuscaElem(Lista* li, int elem,
    int *p);
int insereIni(Lista* li, int elem);
int insereFim(Lista* li, int elem);
int removeIni(Lista* li);
int removeFim(Lista* li);
void imprimeLista(Lista* li);
void destroiLista(Lista* li);

#endif
```

codigos/questao23/lse.h

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "lse.h"

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

```c
    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){
        if(aux->info == elem){
            *p = aux->info;
            return 1;
        }
        aux = aux->prox;
    }
    return 0;
}

int insereIni(Lista* li, int elem){
    if(li == NULL) return 0;
    NO* novo = alocarNO();
    if(novo == NULL) return 0;
    novo->info = elem;
    novo->prox = *li;
    *li = novo;
    return 1;
}

int insereFim(Lista* li, int elem){
    if(li == NULL) return 0;
    NO* novo = alocarNO();
    if(novo == NULL) return 0;
    novo->info = elem;
    novo->prox = NULL;
    if(listaVazia(li)){
        *li = novo;
    }else{
        NO* aux = *li;
        while(aux->prox != NULL)
            aux = aux->prox;
        aux->prox = novo;
    }
    return 1;
}

int removeIni(Lista* li){
    if(li == NULL) return 0;
    if(listaVazia(li)) return 0;
    NO* aux = *li;
    *li = aux->prox;
    liberarNO(aux);
    return 1;
}

int removeFim(Lista* li){
    if(li == NULL) return 0;
    if(listaVazia(li)) return 0;
    NO* ant, *aux = *li;
    while(aux->prox != NULL){
        ant = aux;
        aux = aux->prox;
    }
    if(aux == *li)
        *li = aux->prox;
    else
        ant->prox = aux->prox;
    liberarNO(aux);
    return 1;
}

void imprimeLista(Lista* li){
    if(li == NULL) return;
    if(listaVazia(li)){
        printf("Lista Vazia!\n");
        return;
    }
    //printf("Elementos:\n");
    NO* aux = *li;
    while(aux != NULL){
        printf("%d ", aux->info);
        aux = aux->prox;
    }
    printf("\n");
}

void destroiLista(Lista* li){
    if(li != NULL){
        NO* aux;
        while((*li) != NULL){
            aux = *li;
            *li = (*li)->prox;
            liberarNO(aux);
        }
        free(li);
    }
}
```

codigos/questao23/lse.c

```c
#ifndef HASH_H
#define HASH_H

#include "lse.h"

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


void preencheAleatorio(int* v, int n, int ini, int fim);
Hash* criaHash(int t);
void destroiHash(Hash *h);
int chaveDivisao(int chave, int tam);
int chaveMultiplicacao(int chave, int tam);
int chaveDobra(int chave, int tam);
int valorString(char *str);
int insereHashLSE(Hash* h, int elem);
int buscaHashLSE(Hash* h, int elem, int *p);
void imprimeHash(Hash *h);
#endif
```

codigos/questao23/questao23.h

```c

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "questao23.h"

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);
}

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);
    }
}

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]);
    }
}
```

codigos/questao23/questao23.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "lse.h"
#include "questao23.h"

int main() {
    int buscaElem, x;
    int tam = 10;
    int numElementos = 15;

    /* Criacao da tabela hash */
    Hash *hashDivisao = criaHash(tam);
    Hash *hashMultiplicacao =
        criaHash(tam);
    Hash *hashDobra = criaHash(tam);

    /* Criacao de vetores com valores
        aleatorios */
    int elementos[numElementos];
    preencheAleatorio(elementos,
        numElementos, 1, 100);

    /* Insercao e teste de busca com
        funcao de hashing: Divisao */
    printf("Espalhamento com funcao de
        hashing: Divisao\n");
    for (int i = 0; i < numElementos;
        i++) {
        insereHashLSE(hashDivisao,
            elementos[i]);
    }
    imprimeHash(hashDivisao);

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

    if (buscaHashLSE(hashDivisao, x,
        &buscaElem)) {
        printf("Elemento %d encontrado
            na tabela\n", x);
    } else {
        printf("Elemento %d nao
            encontrado na tabela\n", x);
    }

    /* Insercao e teste de busca com
        funcao de hashing: Multiplicacao
        */
    printf("\nEspalhamento com funcao
        de hashing: Multiplicacao\n");
    for (int i = 0; i < numElementos;
        i++) {
        insereHashLSE(hashMultiplicacao,
            elementos[i]);
    }
    imprimeHash(hashMultiplicacao);

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

    if (buscaHashLSE(hashMultiplicacao,
        x, &buscaElem)) {
        printf("Elemento %d encontrado
            na tabela\n", x);
    } else {
        printf("Elemento %d nao
```

```
            encontrado na tabela\n", x);
    }

    /* Insercao e teste de busca com
        funcao de hashing: Dobra */
    printf("\nEspalhamento com funcao
        de hashing: Dobra\n");
    for (int i = 0; i < numElementos;
        i++) {
        insereHashLSE(hashDobra,
            elementos[i]);
    }
    imprimeHash(hashDobra);

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

    if (buscaHashLSE(hashDobra, x,
        &buscaElem)) {
        printf("Elemento %d encontrado
            na tabela\n", x);
    } else {
        printf("Elemento %d nao
            encontrado na tabela\n", x);
    }

    /* Libera a memoria alocada */
    destroiHash(hashDivisao);
    destroiHash(hashMultiplicacao);
    destroiHash(hashDobra);

    return 0;
}
```

codigos/questao23/main.c

```
all: questao23.o lse.o
  gcc questao23.o lse.o main.c -o main

questao23.o: questao23.h questao23.c
  gcc -c questao23.c

lse.o: lse.h lse.c
  gcc -c lse.c

clean:
  rm -f questao23.o lse.o main
```

codigos/questao23/Makefile

Saída

```
Espalhamento com funcao de hashing: Divisao
0: 60 50
1: 91
2: 22
3: 63 93
4: 64 94 84
5: NULL
6: 36 16
7: 87 87
8: 28 78
9: NULL
Digite um elemento para busca:
87
Elemento 87 encontrado na tabela
```

Figura 15: Questão 1.1 - Divisão

```
Espalhamento com funcao de hashing: Multiplicacao
0: 60 50
1: 91
2: 22
3: 63 93
4: 64 94 84
5: NULL
6: 36 16
7: 87 87
8: 28 78
9: NULL
Digite um elemento para busca:
28
Elemento 28 encontrado na tabela
```

Figura 16: Questão 1.1 - Multiplicação

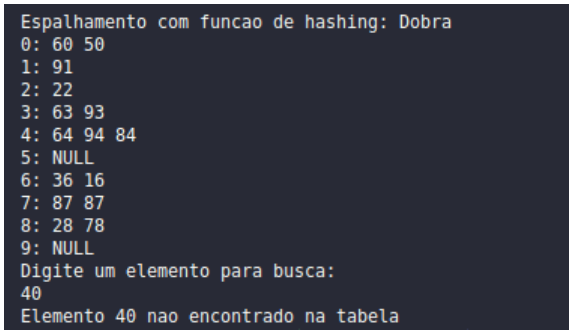

Figura 17: Questão 1.1 - Dobra