



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

```
1 #ifndef _PESQUISA_
2 #define _PESQUISA_
3
4 int getComp();
5 void setComp(int valor);
6 int* copiaVetor(int* v, int n);
7 void imprimeVetor(int* v, int n);
8 void preencheAleatorio(int* v, int n,
    int ini, int fim);
9 void troca(int* a, int *b);
10 int buscaSequencial(int *v, int n, int
    elem);

11 int particao(int *v, int ini, int fim);
12 void QuickSort(int *v, int ini, int
    fim);
13 int rec_buscaBinaria(int *v, int ini,
    int fim, int elem);
14 int it_buscaBinaria(int *v, int ini,
    int fim, int elem);
15
16 #endif

codigos/questao11/questao11.h

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4 #include "questao11.h"
5
6 //Medidas de Complexidade
7 int comp; //Num. de comparacoes
8
9 int getComp(){
10     return comp;
11 }
12
13 void setComp(int valor){
14     comp = valor;
15 }
16
17 int* copiaVetor(int* v, int n){
18     int i;
19     int *v2;
20     v2 = (int*) malloc (n*sizeof(int));
21     for(i=0; i<n; i++) v2[i] = v[i];

22     return v2;
23 }
24 void imprimeVetor(int* v, int n){
25     int i, prim = 1;
26     printf("[");
27     for(i=0; i<n; i++)
28         if(prim){ printf("%d", v[i]);
29                     prim = 0; }
30     else printf(", %d", v[i]);
31     printf("]\n");
32 }
33 void preencheAleatorio(int* v, int n,
    int ini, int fim){
34     int i;
35     for(i=0; i<n; i++)
36         v[i] = ini + rand() % (fim-ini
37             + 1);
38
39 void troca(int* a, int *b){
```

```

40  int aux = *a;
41  *a = *b;
42  *b = aux;
43 }
44
45 int buscaSequencial(int *v, int n, int
    elem){
46     int i;
47     for(i=0; i<n; i++){
48         comp++;
49         if(v[i] == elem)
50             return i; //Elemento
                encontrado
51     }
52     return -1; //Elemento encontrado
53 }
54
55 int particao(int *v, int ini, int fim){
56     int i = ini, j = fim;
57     int pivo = v[(ini+fim)/2];
58     while (1) {
59         while(v[i] < pivo){ i++; }
                //procura algum >= pivo do
                lado esquerdo
60         while(v[j] > pivo){ j--; }
                //procura algum <= pivo do
                lado direito
61
62         if(i<j){
63             troca(&v[i], &v[j]);
                //troca os elementos
                encontrados
64             i++;
65             j--;
66         }else
67             return j; //retorna o local
                onde foi feita a partica
68     }
69 }
70
71 void QuickSort(int *v, int ini, int

```

```

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

```

codigos/questao11/questao11.c

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4  #include "questao11.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                                     50         printf("0 elemento %d foi
28         /* Pesquisa Binaria Recursiva */           encontrado na pos %d.\n", x,
29         t = clock();                               ind);
30         ind = rec_buscaBinaria(v1,0,n-1,x); 51     else
31         t = clock() - t;                       52         printf("0 elemento %d NAO foi
32         printf("\nInformacoes da Pesquisa           encontrado!\n", x);
33             Binaria Recursiva:\n");           53
34         printf("Tempo Execucao:  %f           54         /* Pesquisa Sequencial */
35             seconds.\n",                       t = clock();
36             ((float)t)/CLOCKS_PER_SEC);       55         ind = buscaSequencial(v1,n,x);
37         printf("Comparacoes:  %d\n",         56         t = clock() - t;
38             getComp());                       57         printf("\nInformacoes da Pesquisa
39                                     58             Sequencial:\n");
40                                     59         printf("Tempo Execucao:  %f
41                                     seconds.\n",
42                                     ((float)t)/CLOCKS_PER_SEC);
43         if(ind != -1)                       60         printf("Comparacoes:  %d\n",
44             printf("0 elemento %d foi           getComp());
45             encontrado na pos %d.\n", x,
46             ind);
47     else
48         printf("0 elemento %d NAO foi
49             encontrado!\n", x);
50
51     /* Pesquisa Binaria iterativa */
52     t = clock();
53     ind = it_buscaBinaria(v1,0,n-1,x);
54     t = clock() - t;
55     printf("\nInformacoes da Pesquisa
56         Binaria iterativa:\n");
57     printf("Tempo Execucao:  %f
58         seconds.\n",
59         ((float)t)/CLOCKS_PER_SEC);
60     printf("Comparacoes:  %d\n",
61         getComp());
62
63     if(ind != -1)
64         printf("0 elemento %d foi
65             encontrado na pos %d.\n", x,
66             ind);
67     else
68         printf("0 elemento %d NAO foi
69             encontrado!\n", x);
70
71     free(v1);
72     return 0;
73 }

```

codigos/questao11/main.c

```

1 all: questao11.o
2 gcc questao11.o main.c -o main
3
4 questao11.o: questao11.h questao11.c
5 gcc -c questao11.c

```

```

6
7 clean:
8 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
0 elemento 49 foi encontrado na pos 7.

```

```

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

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

```

Figura 1: Questão 1.1 - Saída 1

Figura 2: Questão 1.1 - Saída 2

1.2 Modificação

Código

```
1 #ifndef _PESQUISA_DECRES_
2 #define _PESQUISA_DECRES_
3
4 int getComp();
5 void setComp(int valor);
6 int* copiaVetor(int* v, int n);
7 void imprimeVetor(int* v, int n);
8 void preencheAleatorio(int* v, int n,
    int ini, int fim);
9 void troca(int* a, int *b);
10 int buscaSequencial(int *v, int n, int
    elem);

11 int particao(int *v, int ini, int fim);
12 void QuickSort(int *v, int ini, int
    fim);
13 int rec_buscaBinaria(int *v, int ini,
    int fim, int elem);
14 int it_buscaBinaria(int *v, int ini,
    int fim, int elem);
15
16 #endif

codigos/questao12/questao12.h

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4 #include <string.h>
5 #include "questao12.h"
6
7 //Medidas de Complexidade
8 int comp; //Num. de comparacoes
9
10 int getComp(){
11     return comp;
12 }
13
14 void setComp(int valor){
15     comp = valor;
16 }
17
18 int* copiaVetor(int* v, int n){
19     int i;
20     int *v2;
21     v2 = (int*) malloc (n*sizeof(int));
22     for(i=0; i<n; i++) v2[i] = v[i];
23     return v2;
24 }
25 void imprimeVetor(int* v, int n){
26     int i, prim = 1;
27     printf("[");
28     for(i=0; i<n; i++)
29         if(prim){ printf("%d", v[i]);
30             prim = 0; }
31         else printf(", %d", v[i]);
32     printf("]\n");
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 void troca(int* a, int *b){
41     int aux = *a;
42     *a = *b;
43     *b = aux;
44 }
45
46 int buscaSequencial(int *v, int n, int
    elem){
47     int i;
48     for(i=0; i<n; i++){
49         comp++;
50         if(v[i] == elem)
51             return i; //Elemento
    encontrado
52     }
53     return -1; //Elemento encontrado
54 }
55
56 int particao(int *v, int ini, int fim){
57     int i = ini, j = fim;
58     int pivo = v[(ini+fim)/2];
59     while (1) {
60         while(v[i] < pivo){ i++; }
61         //procura algum >= pivo do
        lado esquerdo
62         while(v[j] > pivo){ j--; }
63         //procura algum <= pivo do
        lado direito
64         if(i<j){
65             troca(&v[i], &v[j]);
66             //troca os elementos
        encontrados
67             i++;
68             j--;
69         }else
70             break;
71     }
72     return i;
73 }
```

```

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,
91                                 meio+1, fim, elem);
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
32     t = clock() - t;
33     printf("\nInformacoes da Pesquisa Binaria Recursiva:\n");
34     printf("Tempo Execucao: %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
35     printf("Comparacoes: %d\n", getComp());
36
37     if(ind != -1)
38         printf("0 elemento %d foi encontrado na pos %d.\n", x, ind);
39     else
40         printf("0 elemento %d NAO foi encontrado!\n", x);
41
42     /* Pesquisa Binaria iterativa */
43     t = clock();
44     ind = it_buscaBinaria(v1,0,n-1,x);
45     t = clock() - t;
46     printf("\nInformacoes da Pesquisa Binaria iterativa:\n");
47     printf("Tempo Execucao: %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
48     printf("Comparacoes: %d\n", getComp());
49
50     if(ind != -1)
51         printf("0 elemento %d foi encontrado na pos %d.\n", x,

```

```

        ind);
51     else
52         printf("0 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());

```

```

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

```

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
0 elemento 74 NAO foi encontrado!

```

Figura 3: Questão 1.1 - Saída 1

```

61
62     if(ind != -1)
63         printf("0 elemento %d foi
            encontrado na pos %d.\n", x,
            ind);
64     else
65         printf("0 elemento %d NAO foi
            encontrado!\n", x);
66
67     free(v1);
68
69     return 0;
70 }

```

codigos/questao12/main.c

```

6
7 clean:
8   rm -f questao12.o main
        codigos/questao12/Makefile

```

```

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

Informacoes da Pesquisa Sequencial:
Tempo Execucao:  0.000010 seconds.
Comparacoes: 15
0 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);

```

```

20 int rec_buscaBinariaPorNome(Aluno* v,          v, int ini, int fim, int matricula);
    int ini, int fim, const char* nome); 25 int it_buscaBinariaPorMatricula(Aluno*
21 int it_buscaBinariaPorNome(Aluno* v,          v, int ini, int fim, int matricula);
    int ini, int fim, const char* nome); 26
22                                          27 #endif
23 int buscaSequencialPorMatricula(Aluno*
    v, int n, int matricula);
24 int rec_buscaBinariaPorMatricula(Aluno*

```

codigos/questao13/questao13.h

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4 #include <string.h>
5 #include "questao13.h"
6
7 //Medidas de Complexidade
8 int comp; //Num. de comparacoes
9
10 int getComp(){
11     return comp;
12 }
13
14 void setComp(int valor){
15     comp = valor;
16 }
17
18 Aluno* criaAluno(int n){
19     Aluno *aluno = (Aluno*) malloc (n *
        sizeof(Aluno));
20     for(int i=0; i<n; i++){
21         aluno[i].matricula = 0;
22         strcpy(aluno[i].nome, "");
23         aluno[i].notas[0] = 0;
24         aluno[i].notas[1] = 0;
25         aluno[i].notas[2] = 0;
26     }
27     return aluno;
28 }
29
30 Aluno* copiaAluno(Aluno* a) {
31     Aluno* novo =
        (Aluno*)malloc(sizeof(Aluno));
32     strcpy(novo->nome, a->nome);
33     novo->matricula = a->matricula;
34     memcpy(novo->notas, a->notas,
        sizeof(float) * 3);
35     return novo;
36 }
37
38 void imprimeAluno(Aluno* a) {
39     printf("Nome: %s, Matricula: %d,
        Notas: %.2f, %.2f, %.2f\n",
40         a->nome, a->matricula,
        a->notas[0], a->notas[1],
        a->notas[2]);
41 }
42
43 void trocaAluno(Aluno* a, Aluno* b) {
44     Aluno temp;
45     memcpy(&temp, a, sizeof(Aluno));
46     memcpy(a, b, sizeof(Aluno));
47     memcpy(b, &temp, sizeof(Aluno));
48 }
49
50 void QuickSortAluno(Aluno* v, int ini,
    int fim) {
51     if (ini < fim) {
52         int q = particaoAluno(v, ini,
            fim);
53         QuickSortAluno(v, ini, q);
54         QuickSortAluno(v, q + 1, fim);
55     }
56 }
57
58 int particaoAluno(Aluno* v, int ini,
    int fim) {
59     Aluno pivo = v[(ini + fim) / 2];
60     int i = ini, j = fim;
61     while (1) {
62         while (strcmp(v[i].nome,
            pivo.nome) < 0) {
63             i++;
64         }
65         while (strcmp(v[j].nome,
            pivo.nome) > 0) {
66             j--;
67         }
68         if (i < j) {
69             trocaAluno(&v[i], &v[j]);
70             i++;
71             j--;
72         } else {
73             return j;
74         }
75     }
76 }
77
78 int buscaSequencialPorNome(Aluno* v,
    int n, const char* nome) {
79     for (int i = 0; i < n; i++) {
80         if (strcmp(v[i].nome, nome) ==
            0) {
81             return i;
82         }
83     }

```

```

84     return -1;
85 }
86
87 int rec_buscaBinariaPorNome(Aluno* v,
88     int ini, int fim, const char* nome)
89 {
90     if (ini > fim) return -1;
91
92     int meio = (ini + fim) / 2;
93     int comparacao =
94         strcmp(v[meio].nome, nome);
95
96     if (comparacao == 0) {
97         return meio;
98     } else if (comparacao > 0) {
99         return
100             rec_buscaBinariaPorNome(v,
101                 ini, meio - 1, nome);
102     } else {
103         return
104             rec_buscaBinariaPorNome(v,
105                 meio + 1, fim, nome);
106     }
107 }
108
109 int it_buscaBinariaPorNome(Aluno* v,
110     int ini, int fim, const char* nome)
111 {
112     while (ini <= fim) {
113         int meio = (ini + fim) / 2;
114         int comparacao =
115             strcmp(v[meio].nome, nome);
116
117         if (comparacao == 0) {
118             return meio;
119         } else if (comparacao > 0) {
120             fim = meio - 1;
121         } else {
122             ini = meio + 1;
123         }
124     }
125     return -1;
126 }
127
128 int buscaSequencialPorMatricula(Aluno*
129     v, int n, int matricula) {
130     int i;
131     for(i = 0; i < n; i++) {
132         comp++;
133
134         if(v[i].matricula == matricula)
135         {
136             return i;
137         }
138     }
139     return -1;
140 }
141
142 int it_buscaBinariaPorMatricula(Aluno*
143     v, int ini, int fim, int matricula) {
144     int meio;
145     while(ini <= fim) {
146         meio = (ini + fim) / 2;
147         comp++;
148         if(v[meio].matricula ==
149             matricula) {
150             return meio;
151         } else if(v[meio].matricula >
152             matricula) {
153             fim = meio - 1;
154         } else {
155             ini = meio + 1;
156         }
157     }
158     return -1;
159 }

```

codigos/questao13/questao13.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4 #include <string.h>
5 #include "questao13.h"
6
7 int main(){
8

```

```

9 //Atribuições iniciais
10 srand(time(NULL));
11 setComp(0);
12 clock_t t;
13
14 Aluno *alunos;
15 char nome[50];
16 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
43           Nome:\n");
44     for (int i = 0; i < n; i++) {
45         imprimeAluno(&alunos[i]);
46     }
47
48     printf("Digite a matricula de um
49           aluno para busca:\n");
50     scanf("%d", &matricula);
51
52     /* Pesquisa Binaria Iterativa por
53           Matricula */
54     t = clock();
55     indice =
56         it_buscaBinariaPorMatricula(alunos,
57                                     0, n - 1, matricula);
58     t = clock() - t;
59     printf("\nInformacoes da Pesquisa
60           Binaria Iterativa por
61           Matricula:\n");
62     printf("Tempo Execucao: %f
63           seconds.\n",
64           ((float)t)/CLOCKS_PER_SEC);
65     printf("Comparacoes: %d\n",
66           getComp());
67
68     if (indice != -1) {
69         printf("0 aluno com a matricula
70               %d foi encontrado na posicao
71               %d.\n", matricula, indice);
72         imprimeAluno(&alunos[indice]);
73     } else {
74         printf("0 aluno com a matricula
75               %d NAO foi encontrado!\n",
76               matricula);
77     }
78
79     /* Pesquisa Sequencial por
80           Matricula */
81     t = clock();
82     indice =
83         buscaSequencialPorMatricula(alunos,
84                                     n, matricula);
85     t = clock() - t;
86     printf("\nInformacoes da Pesquisa
87           Sequencial por Matricula:\n");
88     printf("Tempo Execucao: %f
89           seconds.\n",
90           ((float)t)/CLOCKS_PER_SEC);
91     printf("Comparacoes: %d\n",
92           getComp());
93
94     if (indice != -1) {
95         printf("0 aluno com a matricula
96               %d foi encontrado na posicao
97               %d.\n", matricula, indice);
98         imprimeAluno(&alunos[indice]);
99     } else {
100         printf("0 aluno com a matricula
101               %d NAO foi encontrado!\n",
102               matricula);
103     }
104
105     getchar();
106     printf("\n\nDigite o nome de um
107           aluno para busca:\n");

```

```

97     fgets(nome, sizeof(nome), stdin);
98
99     int len = strlen(nome);
100    if (len > 0 && nome[len - 1] ==
        '\n') {
101        nome[len - 1] = '\0';
102    }
103
104    /* Pesquisa Binaria Iterativa por
        Nome */
105    t = clock();
106    indice =
        it_buscaBinariaPorNome(alunos,
            0, n - 1, nome);
107    t = clock() - t;
108    printf("\nInformacoes da Pesquisa
        Binaria Iterativa por Nome:\n");
109    printf("Tempo Execucao: %f
        seconds.\n",
        ((float)t)/CLOCKS_PER_SEC);
110    printf("Comparacoes: %d\n",
        getComp());
111
112    if (indice != -1) {
113        printf("0 aluno com a nome %s
            foi encontrado na posicao
            %d.\n", nome, indice);
114        imprimeAluno(&alunos[indice]);
115    } else {
116        printf("0 aluno com a nome %s
            NAO foi encontrado!\n",
            nome);
117    }
118
119    /* Pesquisa Binaria Recursiva por
        Nome */
120    t = clock();
121    indice =
        rec_buscaBinariaPorNome(alunos,
            0, n - 1, nome);
122    t = clock() - t;
123    printf("\nInformacoes da Pesquisa
        Binaria Recursiva por Nome:\n");
124    printf("Tempo Execucao: %f
        seconds.\n",
        ((float)t)/CLOCKS_PER_SEC);

125    printf("Comparacoes: %d\n",
        getComp());
126
127    if (indice != -1) {
128        printf("0 aluno com a nome %s
            foi encontrado na posicao
            %d.\n", nome, indice);
129        imprimeAluno(&alunos[indice]);
130    } else {
131        printf("0 aluno com a nome %s
            NAO foi encontrado!\n",
            nome);
132    }
133
134    /* Pesquisa Sequencial por Nome */
135    t = clock();
136    indice =
        buscaSequencialPorNome(alunos,
            n, nome);
137    t = clock() - t;
138    printf("\nInformacoes da Pesquisa
        Sequencial por Nome:\n");
139    printf("Tempo Execucao: %f
        seconds.\n",
        ((float)t)/CLOCKS_PER_SEC);
140    printf("Comparacoes: %d\n",
        getComp());
141
142    if (indice != -1) {
143        printf("0 aluno com a nome %s
            foi encontrado na posicao
            %d.\n", nome, indice);
144        imprimeAluno(&alunos[indice]);
145    } else {
146        printf("0 aluno com a nome %s
            NAO foi encontrado!\n",
            nome);
147    }
148
149    free(alunos);
150
151    return 0;

```

codigos/questao13/main.c

```

1 all: questao13.o
2 gcc questao13.o main.c -o main
3
4 questao13.o: questao13.h questao13.c
5 gcc -c questao13.c

```

```

6
7 clean:
8 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
0 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
0 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
0 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
0 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
0 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
0 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

```

1 #ifndef HASH_H
2 #define HASH_H
3
4 typedef struct{
5     int **tabela;
6     int tam, qtd;
7 }Hash;
8
9
10 Hash* criaHash(int t);
11 void destroiHash(Hash *h);
12 void preencheAleatorio(int* v, int n,
13     int ini, int fim);
14 int chaveDivisao(int chave, int tam);
15 int chaveMultiplicacao(int chave, int
16     tam);
17 int chaveDobra(int chave, int tam);
18 int valorString(char *str);
19 int insereHash_semTratar(Hash* h, int

```

```

    elem);
20 int buscaHash_semTratar(Hash* h, int
21     elem, int *p);
22 int sondagemLinear(int pos, int i, int
23     tam);
24 int sondagemQuadratica(int pos, int i,
25     int tam);
26 int sondagemDuploHash(int H1, int
27     chave, int i, int tam);
28 int insereHash_EnderAberto(Hash* h, int
29     elem);
30 int buscaHash_EnderAberto(Hash* h, int
31     elem, int *p);
32 void imprimeHash(Hash *h);
33
34 #endif

```

codigos/questao21/questao21.h

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <time.h>

```

```

5 #include "questao21.h"
6
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
14            (t*sizeof(int*));
15        if(h->tabela == NULL) return
16            NULL;
17        int i;
18        for(i = 0; i<t; i++)
19            h->tabela[i] = NULL;
20    }
21    return h;
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,
35     int ini, int fim){
36     int i;
37     for(i=0; i<n; i++)
38         v[i] = ini + rand() % (fim-ini
39             + 1);
40 }
41
42 int chaveDivisao(int chave, int tam){
43     return (chave & 0x7FFFFFFF) % tam;
44 }
45
46 int chaveMultiplicacao(int chave, int
47     tam){
48     float A = 0.6180339887;
49     //constante: 0 < A < 1
50     float val = chave * A;
51     val = val - (int) val;
52     return (int) (tam * val);
53 }
54
55 int chaveDobra(int chave, int tam){
56     int pos, n_bits = 30;
57
58     int p = 1;
59     int r = p << n_bits;
60     while((chave & r) != r){ n_bits--;
61         r = p << n_bits; }
62
63     n_bits++;
64     pos = chave;
65     while(pos > tam){
66         int metade_bits = n_bits/2;
67         int parte1 = pos >> metade_bits;
68         parte1 = parte1 << metade_bits;
69         int parte2 = pos ^ parte1;
70         parte1 = pos >> metade_bits;
71         pos = parte1 ^ parte2;
72         n_bits = n_bits/2;
73     }
74     return pos;
75 }
76
77 int valorString(char *str){
78     int i, valor = 1;
79     int tam = strlen(str);
80     for(i=0; i<tam; i++)
81         valor = 31*valor + (i+1)*((int)
82             str[i]);
83     return valor;
84 }
85
86 int insereHash_semTratar(Hash* h, int
87     elem){
88     if(h == NULL) return 0;
89     int pos = chaveDivisao(elem,
90         h->tam);
91
92     if(h->tabela[pos] == NULL){
93         int* novo = (int*) malloc
94             (sizeof(int));
95         if(novo == NULL) return 0;
96         *novo = elem;
97         h->tabela[pos] = novo;
98         h->qtd++;
99     }else *(h->tabela[pos]) = elem;
100     return 1;
101 }
102
103 int buscaHash_semTratar(Hash* h, int
104     elem, int *p){
105     if(h == NULL) return 0;
106     int pos = chaveDivisao(elem,
107         h->tam);
108     if(h->tabela[pos] == NULL) return 0;
109     if(*(h->tabela[pos]) == elem){
110         *p = *(h->tabela[pos]);
111         return 1;
112     }
113     return 0;
114 }
115
116 int sondagemLinear(int pos, int i, int
117     tam){
118     return ( (pos + i) & 0x7FFFFFFF) %
119         tam;
120 }
121
122 int sondagemQuadratica(int pos, int i,
123     int tam){
124     pos = pos + 2*i + 5*i*i;

```

```

111     return ( pos & 0x7FFFFFFF) % tam;
112 }
113
114 int sondagemDuploHash(int H1, int
115     chave, int i, int tam){
116     int H2 = chaveDivisao(chave, tam-1)
117         + 1;
118     return ( (H1 + i*H2) & 0x7FFFFFFF)
119         % tam;
120 }
121
122 int insereHash_EnderAberto(Hash* h, int
123     elem){
124     if(h == NULL) return 0;
125     int i, pos, newPos;
126     pos = chaveDivisao(elem, h->tam);
127     for(i=0; i<h->tam; i++){
128         newPos = sondagemLinear(pos, i,
129             h->tam);
130         //newPos = sondagemQuadratica(pos,
131             i, h->tam);
132         //newPos = sondagemDuploHash(pos,
133             elem, i, h->tam);
134         if(h->tabela[newPos] == NULL){
135             int* novo = (int*) malloc
136                 (sizeof(int));
137             if(novo == NULL) return 0;
138             *novo = elem;
139             h->tabela[newPos] = novo;
140             h->qtd++;
141             return 1;
142         }
143     }
144     return 0;
145 }
146
147 int buscaHash_EnderAberto(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 =
            sondagemQuadratica(pos, i,
            h->tam);
        //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]));
    }
}

```

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 =
16        criaHash(tam);
17    Hash* hashDobra = criaHash(tam);
18
19    /* Criacao de vetores com valores
20        aleatorios */
21    int elementos[numElementos];
22    preencheAleatorio(elementos,
23        numElementos, 1, 100);
24
25    /* Insercao e teste de busca com
26        funcao de hashing: Divisao */
27    printf("Espalhamento com funcao de
28        hashing: Divisao\n");
29    for (int i = 0; i < numElementos;
30        i++) {
31        insereHash_semTratar(hashDivisao,
32            elementos[i]);
33    }
34    imprimeHash(hashDivisao);

```

```

28                                     encontrado na tabela\n", x);
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(hashMultiplicacao, elementos[i]);
42     }
43     imprimeHash(hashMultiplicacao);
44
45     printf("Digite um elemento para busca:\n");
46     scanf("%d", &x);
47
48     if (buscaHash_semTratar(hashMultiplicacao, x, &buscaElem)) {
49         printf("Elemento %d encontrado na tabela\n", x);
50     } else {
51         printf("Elemento %d nao encontrado na tabela\n", x);
52     }
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 printf("Digite um elemento para busca:\n");
63 scanf("%d", &x);
64
65 if (buscaHash_semTratar(hashDobra, x, &buscaElem)) {
66     printf("Elemento %d encontrado na tabela\n", x);
67 } else {
68     printf("Elemento %d nao encontrado na tabela\n", x);
69 }
70
71 /* Libera a memoria alocada */
72 destroiHash(hashDivisao);
73 destroiHash(hashMultiplicacao);
74 destroiHash(hashDobra);
75
76 return 0;
77
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

```

Espalhamento com funcao de hashing: Divisao
0: 60
1: 91
2: 22
3: 63
4: 64
5: NULL
6: 36
7: 87
8: 28
9: NULL
Digite um elemento para busca:
22

```

Figura 9: Questão 1.1 - Divisão

```

Espalhamento com funcao de hashing: Multiplicacao
0: 60
1: 91
2: 22
3: 63
4: 64
5: NULL
6: 36
7: 87
8: 28
9: NULL
Digite um elemento para busca:
80
Elemento 80 nao encontrado na tabela

```

Figura 10: Questão 1.1 - Multiplicação

```

Elemento 80 nao encontrado na tabela

Espalhamento com funcao de hashing: Dobra
0: 60
1: 91
2: 22
3: 63
4: 64
5: NULL
6: 36
7: 87
8: 28
9: NULL
Digite um elemento para busca:
64
Elemento 64 encontrado na tabela

```

Figura 11: Questão 1.1 - Dobra

```

1 #ifndef HASH_H
2 #define HASH_H
3
4 typedef struct{
5     int **tabela;
6     int tam, qtd;
7 }Hash;
8
9
10 Hash* criaHash(int t);
11 void destroiHash(Hash *h);
12 void preencheAleatorio(int* v, int n,
13     int ini, int fim);
14 int chaveDivisao(int chave, int tam);
15 int chaveMultiplicacao(int chave, int
16     tam);
17 int chaveDobra(int chave, int tam);
18 int valorString(char *str);
19 int insereHash_semTratar(Hash* h, int
20     elem);

```

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>

```

```

18 int buscaHash_semTratar(Hash* h, int
19     elem, int *p);
20 int sondagemLinear(int pos, int i, int
21     tam);
22 int sondagemQuadratica(int pos, int i,
23     int tam);
24 int sondagemDuploHash(int H1, int
25     chave, int i, int tam);
26 int escolheSondagem(int pos, int i, int
27     tam, int opcao);
28 int insereHash_EnderAberto(Hash *h, int
29     elem, int opcaoSondagem);
30 int buscaHash_EnderAberto(Hash *h, int
31     elem, int *p, int opcaoSondagem);
32 void imprimeHash(Hash *h);
33
34 #endif

```

codigos/questao22/questao22.h

```

4 #include <time.h>
5 #include "questao22.h"
6

```

```

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
14            (t*sizeof(int*));
15        if(h->tabela == NULL) return
16            NULL;
17        int i;
18        for(i = 0; i<t; i++)
19            h->tabela[i] = NULL;
20    }
21    return h;
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,
35     int ini, int fim){
36     int i;
37     for(i=0; i<n; i++)
38         v[i] = ini + rand() % (fim-ini
39             + 1);
40 }
41
42 int chaveDivisao(int chave, int tam){
43     return (chave & 0x7FFFFFFF) % tam;
44 }
45
46 int chaveMultiplicacao(int chave, int
47     tam){
48     float A = 0.6180339887;
49     //constante: 0 < A < 1
50     float val = chave * A;
51     val = val - (int) val;
52     return (int) (tam * val);
53 }
54
55 int chaveDobra(int chave, int tam){
56     int pos, n_bits = 30;
57
58     int p = 1;
59     int r = p << n_bits;
60     while((chave & r) != r){ n_bits--;
61         r = p << n_bits; }
62     n_bits++;
63
64     pos = chave;
65     while(pos > tam){
66         int metade_bits = n_bits/2;
67         int parte1 = pos >> metade_bits;
68         parte1 = parte1 << metade_bits;
69         int parte2 = pos ^ parte1;
70         parte1 = pos >> metade_bits;
71         pos = parte1 ^ parte2;
72         n_bits = n_bits/2;
73     }
74     return pos;
75 }
76
77 int valorString(char *str){
78     int i, valor = 1;
79     int tam = strlen(str);
80     for(i=0; i<tam; i++)
81         valor = 31*valor + (i+1)*((int)
82             str[i]);
83     return valor;
84 }
85
86 int insereHash_semTratar(Hash* h, int
87     elem){
88     if(h == NULL) return 0;
89     int pos = chaveDivisao(elem,
90         h->tam);
91
92     if(h->tabela[pos] == NULL){
93         int* novo = (int*) malloc
94             (sizeof(int));
95         if(novo == NULL) return 0;
96         *novo = elem;
97         h->tabela[pos] = novo;
98         h->qtd++;
99     }else *(h->tabela[pos]) = elem;
100    return 1;
101 }
102
103 int buscaHash_semTratar(Hash* h, int
104     elem, int *p){
105     if(h == NULL) return 0;
106     int pos = chaveDivisao(elem,
107         h->tam);
108     if(h->tabela[pos] == NULL) return 0;
109     if(*(h->tabela[pos]) == elem){
110         *p = *(h->tabela[pos]);
111         return 1;
112     }
113     return 0;
114 }
115
116 int sondagemLinear(int pos, int i, int
117     tam){
118     return ( (pos + i) & 0x7FFFFFFF) %
119         tam;
120 }
121
122 int sondagemQuadratica(int pos, int i,

```



```

110     int tam){
111         pos = pos + 2*i + 5*i*i;
112     }
113     return ( pos & 0x7FFFFFFF) % tam;
114 }
115 int sondagemDuploHash(int H1, int
116     chave, int i, int tam){
117     int H2 = chaveDivisao(chave, tam-1)
118         + 1;
119     return ( (H1 + i*H2) & 0x7FFFFFFF)
120         % tam;
121 }
122 int escolheSondagem(int pos, int i, int
123     tam, int opcao) {
124     switch (opcao) {
125         case 1:
126             return sondagemLinear(pos,
127                 i, tam);
128         case 2:
129             return
130                 sondagemQuadratica(pos,
131                     i, tam);
132         case 3:
133             return
134                 sondagemDuploHash(pos,
135                     pos, i, tam);
136         default:
137             return -1;
138     }
139 }
140 int insereHash_EnderAberto(Hash *h, int
141     elem, int opcaoSondagem) {
142     if (h == NULL) return 0;
143     int i, pos, newPos;
144     pos = chaveDivisao(elem, h->tam);
145     for (i = 0; i < h->tam; i++) {
146         newPos = escolheSondagem(pos,
147             i, h->tam, opcaoSondagem);
148         if (h->tabela[newPos] == NULL)
149             return 0;
150         if (*(h->tabela[newPos]) ==
151             elem) {
152             *p = *(h->tabela[newPos]);
153             return 1;
154         }
155     }
156     return 0;
157 }
158 void imprimeHash(Hash *h){
159     if(h == NULL) return;
160     int i;
161     for(i=0; i<h->tam; i++){
162         printf("%d: ", i);
163         if(h->tabela[i] == NULL)
164             printf("NULL\n");
165         else printf("%d\n",
166             *(h->tabela[i]));
167     }
168 }
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

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
14    Hash *hashDivisao = criaHash(tam);
15    Hash *hashMultiplicacao =
16        criaHash(tam);
17    Hash *hashDobra = criaHash(tam);
18
19    /* Criacao de vetores com valores
20       aleatorios */
21    int elementos[numElementos];
22    preencheAleatorio(elementos,
23        numElementos, 1, 100);
24
25    /* Insercao e teste de busca com
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```



```

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

```

1 #ifndef LSE_H
2 #define LSE_H
3
4 #include <stdio.h>
5 #include <stdlib.h>
6
7 typedef struct NO{
8     int info;
9     struct NO* prox;
10 }NO;
11
12 typedef struct NO* Lista;
13
14 Lista* criaLista();
15 int listaVazia(Lista *li);

```

```

16 NO* alocarNO();
17 void liberarNO(NO* q);
18 int listaBuscaElem(Lista* li, int elem,
19                     int *p);
19 int insereIni(Lista* li, int elem);
20 int insereFim(Lista* li, int elem);
21 int removeIni(Lista* li);
22 int removeFim(Lista* li);
23 void imprimeLista(Lista* li);
24 void destroiLista(Lista* li);
25
26 #endif

```

codigos/questao23/lse.h

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include "lse.h"
5
6 Lista* criaLista(){

```

```

7     Lista *li;
8     li = (Lista*) malloc
9         (sizeof(Lista));
10     if(li != NULL){
11         *li = NULL;
12     }

```

```

12     return li;
13 }
14
15 int listaVazia(Lista *li){
16     if(li == NULL) return 1;
17     if(*li == NULL) return 1; //True -
        Vazia!
18     return 0; //False - tem elemento!
19 }
20
21 NO* alocarNO(){
22     return (NO*) malloc (sizeof(NO));
23 }
24
25 void liberarNO(NO* q){
26     free(q);
27 }
28
29 int listaBuscaElem(Lista* li, int elem,
    int *p){
30     if(li == NULL) return 0;
31     NO* aux = *li;
32     while(aux != NULL){
33         if(aux->info == elem){
34             *p = aux->info;
35             return 1;
36         }
37         aux = aux->prox;
38     }
39     return 0;
40 }
41
42 int insereIni(Lista* li, int elem){
43     if(li == NULL) return 0;
44     NO* novo = alocarNO();
45     if(novo == NULL) return 0;
46     novo->info = elem;
47     novo->prox = *li;
48     *li = novo;
49     return 1;
50 }
51
52 int insereFim(Lista* li, int elem){
53     if(li == NULL) return 0;
54     NO* novo = alocarNO();
55     if(novo == NULL) return 0;
56     novo->info = elem;
57     novo->prox = NULL;
58     if(listaVazia(li)){
59         *li = novo;
60     }else{
61         NO* aux = *li;
62         while(aux->prox != NULL)
63             aux = aux->prox;
64         aux->prox = novo;
65     }
66     return 1;
67 }
68
69 int removeIni(Lista* li){
70     if(li == NULL) return 0;
71     if(listaVazia(li)) return 0;
72     NO* aux = *li;
73     *li = aux->prox;
74     liberarNO(aux);
75     return 1;
76 }
77
78 int removeFim(Lista* li){
79     if(li == NULL) return 0;
80     if(listaVazia(li)) return 0;
81     NO* ant, *aux = *li;
82     while(aux->prox != NULL){
83         ant = aux;
84         aux = aux->prox;
85     }
86     if(aux == *li)
87         *li = aux->prox;
88     else
89         ant->prox = aux->prox;
90     liberarNO(aux);
91     return 1;
92 }
93
94 void imprimeLista(Lista* li){
95     if(li == NULL) return;
96     if(listaVazia(li)){
97         printf("Lista Vazia!\n");
98         return;
99     }
100     //printf("Elementos:\n");
101     NO* aux = *li;
102     while(aux != NULL){
103         printf("%d ", aux->info);
104         aux = aux->prox;
105     }
106     printf("\n");
107 }
108
109 void destroiLista(Lista* li){
110     if(li != NULL){
111         NO* aux;
112         while((*li) != NULL){
113             aux = *li;
114             *li = (*li)->prox;
115             liberarNO(aux);
116         }
117         free(li);
118     }
119 }

```

codigos/questao23/lse.c

```

1 #ifndef HASH_H
2 #define HASH_H
3
4 #include "lse.h"
5
6 typedef struct{
7     Lista **tabela;
8     int tam, qtd;
9 }Hash;
10
11
12 void preencheAleatorio(int* v, int n,
13     int ini, int fim);
14 Hash* criaHash(int t);

```

```

1
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <string.h>
5 #include "questao23.h"
6
7 void preencheAleatorio(int* v, int n,
8     int ini, int fim){
9     int i;
10    for(i=0; i<n; i++)
11        v[i] = ini + rand() % (fim-ini
12            + 1);
13 }
14
15 Hash* criaHash(int t){
16     Hash* h;
17     h = (Hash*) malloc (sizeof(Hash));
18     if(h != NULL){
19         h->tam = t; h->qtd = 0;
20         h->tabela = (Lista**) malloc
21             (t*sizeof(Lista*));
22         if(h->tabela == NULL) return
23             NULL;
24         int i;
25         for(i = 0; i<t; i++)
26             h->tabela[i] = NULL;
27     }
28     return h;
29 }
30
31 void destroiHash(Hash *h){
32     if(h != NULL){
33         int i;
34         for(i = 0; i<h->tam; i++)
35             if(h->tabela[i] != NULL)
36                 destroiLista(h->tabela[i]);
37         free(h->tabela);
38         free(h);
39     }
40 }

```

```

14 void destroiHash(Hash *h);
15 int chaveDivisao(int chave, int tam);
16 int chaveMultiplicacao(int chave, int
17     tam);
18 int chaveDobra(int chave, int tam);
19 int valorString(char *str);
20 int insereHashLSE(Hash* h, int elem);
21 int buscaHashLSE(Hash* h, int elem, int
22     *p);
23 void imprimeHash(Hash *h);
24 #endif

```

codigos/questao23/questao23.h

```

38
39 int chaveDivisao(int chave, int tam){
40     return (chave & 0x7FFFFFFF) % tam;
41 }
42
43 int chaveMultiplicacao(int chave, int
44     tam){
45     float A = 0.6180339887;
46     //constante: 0 < A < 1
47     float val = chave * A;
48     val = val - (int) val;
49     return (int) (tam * val);
50 }
51
52 int chaveDobra(int chave, int tam){
53     int pos, n_bits = 30;
54
55     int p = 1;
56     int r = p << n_bits;
57     while((chave & r) != r){ n_bits--;
58         r = p << n_bits; }
59
60     n_bits++;
61     pos = chave;
62     while(pos > tam){
63         int metade_bits = n_bits/2;
64         int parte1 = pos >> metade_bits;
65         parte1 = parte1 << metade_bits;
66         int parte2 = pos ^ parte1;
67         parte1 = pos >> metade_bits;
68         pos = parte1 ^ parte2;
69         n_bits = n_bits/2;
70     }
71     return pos;
72 }
73
74 int valorString(char *str){
75     int i, valor = 1;
76     int tam = strlen(str);
77     for(i=0; i<tam; i++)
78         valor = 31*valor + (i+1)*((int)

```

```

        str[i]);
76     return valor;
77 }
78
79 int insereHashLSE(Hash* h, int elem){
80     if(h == NULL) return 0;
81     int pos = chaveDivisao(elem,
82         h->tam);
83     if(h->tabela[pos] == NULL)
84         h->tabela[pos] = criaLista();
85     insereIni(h->tabela[pos], elem);
86     h->qtd++;
87     return 1;
88 }
89 int buscaHashLSE(Hash* h, int elem, int
90     *p){
91     if(h == NULL) return 0;
92     int pos = chaveDivisao(elem,

```

```

        h->tam);
92     if(h->tabela[pos] == NULL) return 0;
93     return
94         listaBuscaElem(h->tabela[pos],
95             elem, p);
96 }
97 void imprimeHash(Hash *h){
98     if(h == NULL) return;
99     int i;
100     for(i=0; i<h->tam; i++){
101         printf("%d: ", i);
102         if(h->tabela[i] == NULL)
103             printf("NULL\n");
104         else imprimeLista(h->tabela[i]);
105     }
106 }

```

codigos/questao23/questao23.c

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <time.h>
5  #include "lse.h"
6  #include "questao23.h"
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 =
16         criaHash(tam);
17     Hash *hashDobra = criaHash(tam);
18
19     /* Criacao de vetores com valores
20     aleatorios */
21     int elementos[numElementos];
22     preencheAleatorio(elementos,
23         numElementos, 1, 100);
24
25     /* Insercao e teste de busca com
26     funcao de hashing: Divisao */
27     printf("Espalhamento com funcao de
28         hashing: Divisao\n");
29     for (int i = 0; i < numElementos;
30         i++) {
31         insereHashLSE(hashDivisao,
32             elementos[i]);
33     }
34     imprimeHash(hashDivisao);
35

```

```

29     printf("Digite um elemento para
30         busca:\n");
31     scanf("%d", &x);
32
33     if (buscaHashLSE(hashDivisao, x,
34         &buscaElem)) {
35         printf("Elemento %d encontrado
36             na tabela\n", x);
37     } else {
38         printf("Elemento %d nao
39             encontrado na tabela\n", x);
40     }
41
42     /* Insercao e teste de busca com
43     funcao de hashing: Multiplicacao
44     */
45     printf("\nEspalhamento com funcao
46     de hashing: Multiplicacao\n");
47     for (int i = 0; i < numElementos;
48         i++) {
49         insereHashLSE(hashMultiplicacao,
50             elementos[i]);
51     }
52     imprimeHash(hashMultiplicacao);
53
54     printf("Digite um elemento para
55         busca:\n");
56     scanf("%d", &x);
57
58     if (buscaHashLSE(hashMultiplicacao,
59         x, &buscaElem)) {
60         printf("Elemento %d encontrado
61             na tabela\n", x);
62     } else {
63         printf("Elemento %d nao
64             encontrado na tabela\n", x);
65     }
66 }

```

```

52         encontrado na tabela\n", x); 64
53     }
54     /* Insercao e teste de busca com
55     funcao de hashing: Dobra */
56     printf("\nEspalhamento com funcao
57     de hashing: Dobra\n");
58     for (int i = 0; i < numElementos;
59     i++) {
60         insereHashLSE(hashDobra,
61         elementos[i]);
62     }
63     imprimeHash(hashDobra);
64     printf("Digite um elemento para
65     busca:\n");
66     scanf("%d", &x);
67
68     if (buscaHashLSE(hashDobra, x,
69     &buscaElem)) {
70         printf("Elemento %d encontrado
71         na tabela\n", x);
72     } else {
73         printf("Elemento %d nao
74         encontrado na tabela\n", x);
75     }
76
77     /* Libera a memoria alocada */
78     destroiHash(hashDivisao);
79     destroiHash(hashMultiplicacao);
80     destroiHash(hashDobra);
81
82     return 0;
83 }

```

codigos/questao23/main.c

```

1 all: questao23.o lse.o
2 gcc questao23.o lse.o main.c -o main
3
4 questao23.o: questao23.h questao23.c
5 gcc -c questao23.c
6
7 lse.o: lse.h lse.c
8 gcc -c lse.c
9
10 clean:
11 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

```
Espalhamento com funcao de hashing: Dobra
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:
40
Elemento 40 nao encontrado na tabela
```

Figura 17: Questão 1.1 - Dobra