

1.1/busca.h

```
1  #ifndef BUSCA_H
2  #define BUSCA_H
3
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <time.h>
7
8  //Medidas de Complexidade
9  int comp; //Num. de comparacoes
10
11 int* copiaVetor(int* v, int n){
12     int i;
13     int *v2;
14     v2 = (int*) malloc (n*sizeof(int));
15     for(i=0; i<n; i++) v2[i] = v[i];
16     return v2;
17 }
18 void imprimeVetor(int* v, int n){
19     int i, prim = 1;
20     printf("[");
21     for(i=0; i<n; i++)
22         if(prim){ printf("%d", v[i]); prim = 0; }
23         else printf(", %d", v[i]);
24     printf("]\n");
25 }
26
27 void preencheAleatorio(int* v, int n, int ini, int fim){
28     int i;
29     for(i=0; i<n; i++)
30         v[i] = ini + rand() % (fim-ini + 1);
31 }
32
33 void troca(int* a, int *b){
34     int aux = *a;
35     *a = *b;
36     *b = aux;
37 }
38
39 int buscaSequencial(int *v, int n, int elem){
40     int i;
41     for(i=0; i<n; i++){
42         comp++;
43         if(v[i] == elem)
44             return i; //Elemento encontrado
45     }
46     return -1; //Elemento encontrado
47 }
48
49 int particao(int *v, int ini, int fim){
50     int i = ini, j = fim;
51     int pivo = v[(ini+fim)/2];
52     while (1) {
53         while(v[i] < pivo){ i++; } //procura algum >= pivo do lado esquerdo
54         while(v[j] > pivo){ j--; } //procura algum <= pivo do lado direito
55
56         if(i<j){
57             troca(&v[i], &v[j]); //troca os elementos encontrados
58         }
59     }
60 }
```

```
58         i++;
59         j--;
60     }else
61         return j; //retorna o local onde foi feita a particao
62     }
63 }
64
65 void QuickSort(int *v, int ini, int fim){
66     if(ini < fim ){
67         int q = particao(v, ini, fim);
68         QuickSort(v, ini, q);
69         QuickSort(v, q+1, fim);
70     }
71 }
72
73 int rec_buscaBinaria(int *v, int ini, int fim, int elem){
74     if(ini > fim) return -1;
75     int meio = (ini + fim)/2;
76     comp++;
77     if(v[meio] == elem)
78         return meio;
79     else
80         if(elem < v[meio])
81             return rec_buscaBinaria(v, ini, meio-1, elem);
82         else
83             return rec_buscaBinaria(v, meio+1, fim, elem);
84 }
85
86 int it_buscaBinaria(int *v, int ini, int fim, int elem){
87     int meio;
88     while(ini <= fim){
89         meio = (ini + fim)/2;
90         comp++;
91         if(elem == v[meio]) return meio;
92         else
93             if(elem < v[meio])
94                 fim = meio-1;
95             else
96                 ini = meio+1;
97     }
98     return -1;
99 }
100
101 #endif
```

1.1/main.c

```
1  #include "busca.h"
2
3
4  int main(){
5
6      //Atribuicoes iniciais
7      srand(time(NULL));
8      comp = 0;
9      clock_t t;
10
11      /*
12      //Template de Calculo do Tempo de Execucao
13      t = clock();
14      //Chamada do Algoritmo aqui...
15      t = clock() - t;
16      printf ("It took me %d clicks (%f seconds).\n",t,((float)t)/CLOCKS_PER_SEC);
17      */
18
19      int *v;
20      int n, x;
21      printf("Digite o tamanho do vetor:\n");
22      scanf("%d", &n);
23      v = (int*) malloc (n*sizeof(int));
24
25
26      preencheAleatorio(v, n, 1, n);
27      //imprimeVetor(v, n);
28      QuickSort(v, 0, n-1);
29
30      printf("Digite um elemento para busca:\n");
31      scanf("%d", &x);
32
33      int ind;
34
35      t = clock();
36      ind = buscaSequencial(v, n, x);
37      t = clock() - t;
38      printf("-----Informacoes Busca Sequencial:\n");
39      printf("Tempo Execucao:  %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
40      printf("Comparacoes: %d\n", comp);
41
42      //imprimeVetor(v, n);
43
44      if(ind != -1)
45          printf("O elemento %d foi encontrado na pos %d.\n", x, ind);
46      else
47          printf("O elemento %d NAO foi encontrado!\n", x);
48
49
50
51      //imprimeVetor(v, n);
52
53      comp = 0;
54      t = clock();
55      ind = rec_buscaBinaria(v, 0, n-1, x);
56      t = clock() - t;
57      printf("-----Informacoes Busca Binaria Recursiva:\n");
```

```
58 | printf("Tempo Execucao: %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
59 | printf("Comparacoes: %d\n", comp);
60 |
61 | if(ind != -1)
62 |     printf("O elemento %d foi encontrado na pos %d.\n", x, ind);
63 | else
64 |     printf("O elemento %d NAO foi encontrado!\n", x);
65 |
66 |
67 | comp = 0;
68 | t = clock();
69 | ind = it_buscaBinaria(v, 0, n-1, x);
70 | t = clock() - t;
71 | printf("-----Informacoes Busca Binaria Iterativa:\n");
72 | printf("Tempo Execucao: %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
73 | printf("Comparacoes: %d\n", comp);
74 |
75 | if(ind != -1)
76 |     printf("O elemento %d foi encontrado na pos %d.\n", x, ind);
77 | else
78 |     printf("O elemento %d NAO foi encontrado!\n", x);
79 |
80 |
81 | free(v);
82 | return 0;
83 | }
```

```
[lucascosta@fedora 1.1]$ gcc main.c -o main
[lucascosta@fedora 1.1]$ ./main
Digite o tamanho do vetor:
7
Digite um elemento para busca:
25
-----Informacoes Busca Sequencial:
Tempo Execucao:  0.000004 seconds.
Comparacoes: 7
0 elemento 25 NAO foi encontrado!
-----Informacoes Busca Binaria Recursiva:
Tempo Execucao:  0.000003 seconds.
Comparacoes: 3
0 elemento 25 NAO foi encontrado!
-----Informacoes Busca Binaria Iterativa:
Tempo Execucao:  0.000002 seconds.
Comparacoes: 3
0 elemento 25 NAO foi encontrado!
[lucascosta@fedora 1.1]$
```

1.2/busca.h

```
1  #ifndef BUSCA_H
2  #define BUSCA_H
3
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <time.h>
7
8  //Medidas de Complexidade
9  int comp; //Num. de comparacoes
10
11 int* copiaVetor(int* v, int n){
12     int i;
13     int *v2;
14     v2 = (int*) malloc (n*sizeof(int));
15     for(i=0; i<n; i++) v2[i] = v[i];
16     return v2;
17 }
18 void imprimeVetor(int* v, int n){
19     int i, prim = 1;
20     printf("[");
21     for(i=0; i<n; i++)
22         if(prim){ printf("%d", v[i]); prim = 0; }
23         else printf(", %d", v[i]);
24     printf("]\n");
25 }
26
27 void preencheAleatorio(int* v, int n, int ini, int fim){
28     int i;
29     for(i=0; i<n; i++)
30         v[i] = ini + rand() % (fim-ini + 1);
31 }
32
33 void troca(int* a, int *b){
34     int aux = *a;
35     *a = *b;
36     *b = aux;
37 }
38
39 int buscaSequencial(int *v, int n, int elem){
40     int i;
41     for(i=0; i<n; i++){
42         comp++;
43         if(v[i] == elem)
44             return i; //Elemento encontrado
45     }
46     return -1; //Elemento encontrado
47 }
48
49 int particao(int *v, int ini, int fim){
50     int i = ini, j = fim;
51     int pivo = v[(ini+fim)/2];
52     while (1) {
53         while(v[i] > pivo){ i++; } //procura algum >= pivo do lado esquerdo
54         while(v[j] < pivo){ j--; } //procura algum <= pivo do lado direito
55
56         if(i<j){
57             troca(&v[i], &v[j]); //troca os elementos encontrados
```

```
58         i++;
59         j--;
60     }else
61         return j; //retorna o local onde foi feita a particao
62     }
63 }
64
65 void QuickSort(int *v, int ini, int fim){
66     if(ini < fim ){
67         int q = particao(v, ini, fim);
68         QuickSort(v, ini, q);
69         QuickSort(v, q+1, fim);
70     }
71 }
72
73 int rec_buscaBinaria(int *v, int ini, int fim, int elem){
74     if(ini > fim) return -1;
75     int meio = (ini + fim)/2;
76     comp++;
77     if(v[meio] == elem)
78         return meio;
79     else
80         if(elem > v[meio])
81             return rec_buscaBinaria(v, ini, meio-1, elem);
82         else
83             return rec_buscaBinaria(v, meio+1, fim, elem);
84 }
85
86 int it_buscaBinaria(int *v, int ini, int fim, int elem){
87     int meio;
88     while(ini <= fim){
89         meio = (ini + fim)/2;
90         comp++;
91         if(elem == v[meio]) return meio;
92         else
93             if(elem > v[meio])
94                 fim = meio-1;
95             else
96                 ini = meio+1;
97     }
98     return -1;
99 }
100
101 #endif
```

1.2/main.c

```
1  #include "busca.h"
2
3
4  int main(){
5
6      //Atribuicoes iniciais
7      srand(time(NULL));
8      comp = 0;
9      clock_t t;
10
11      /*
12      //Template de Calculo do Tempo de Execucao
13      t = clock();
14      //Chamada do Algoritmo aqui...
15      t = clock() - t;
16      printf ("It took me %d clicks (%f seconds).\n",t,((float)t)/CLOCKS_PER_SEC);
17      */
18
19      int *v;
20      int n, x;
21      printf("Digite o tamanho do vetor:\n");
22      scanf("%d", &n);
23      v = (int*) malloc (n*sizeof(int));
24
25
26      preencheAleatorio(v, n, 1, n);
27      //imprimeVetor(v, n);
28      QuickSort(v, 0, n-1);
29
30      printf("Digite um elemento para busca:\n");
31      scanf("%d", &x);
32
33      int ind;
34
35      t = clock();
36      ind = buscaSequencial(v, n, x);
37      t = clock() - t;
38      printf("-----Informacoes Busca Sequencial:\n");
39      printf("Tempo Execucao:  %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
40      printf("Comparacoes: %d\n", comp);
41
42      //imprimeVetor(v, n);
43
44      if(ind != -1)
45          printf("O elemento %d foi encontrado na pos %d.\n", x, ind);
46      else
47          printf("O elemento %d NAO foi encontrado!\n", x);
48
49
50
51      //imprimeVetor(v, n);
52
53      comp = 0;
54      t = clock();
55      ind = rec_buscaBinaria(v, 0, n-1, x);
56      t = clock() - t;
57      printf("-----Informacoes Busca Binaria Recursiva:\n");
```



```
58 | printf("Tempo Execucao: %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
59 | printf("Comparacoes: %d\n", comp);
60 |
61 | if(ind != -1)
62 |     printf("O elemento %d foi encontrado na pos %d.\n", x, ind);
63 | else
64 |     printf("O elemento %d NAO foi encontrado!\n", x);
65 |
66 |
67 | comp = 0;
68 | t = clock();
69 | ind = it_buscaBinaria(v, 0, n-1, x);
70 | t = clock() - t;
71 | printf("-----Informacoes Busca Binaria Iterativa:\n");
72 | printf("Tempo Execucao: %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
73 | printf("Comparacoes: %d\n", comp);
74 |
75 | if(ind != -1)
76 |     printf("O elemento %d foi encontrado na pos %d.\n", x, ind);
77 | else
78 |     printf("O elemento %d NAO foi encontrado!\n", x);
79 |
80 |
81 | free(v);
82 | return 0;
83 | }
```

```
[lucascosta@fedora 1.2]$ gcc main.c -o main
[lucascosta@fedora 1.2]$ ./main
Digite o tamanho do vetor:
75
Digite um elemento para busca:
16
-----Informacoes Busca Sequencial:
Tempo Execucao: 0.000004 seconds.
Comparacoes: 75
0 elemento 16 NAO foi encontrado!
-----Informacoes Busca Binaria Recursiva:
Tempo Execucao: 0.000002 seconds.
Comparacoes: 6
0 elemento 16 NAO foi encontrado!
-----Informacoes Busca Binaria Iterativa:
Tempo Execucao: 0.000003 seconds.
Comparacoes: 6
0 elemento 16 NAO foi encontrado!
[lucascosta@fedora 1.2]$ █
```

1.3/busca.h

```
1  #ifndef BUSCA_H
2  #define BUSCA_H
3
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <time.h>
7  #include <string.h>
8  #define MAX 200
9
10 typedef struct{
11     char nome[MAX];
12     int matricula;
13     double nota1, nota2, nota3;
14 } Aluno;
15
16 //Medidas de Complexidade
17 int comp; //Num. de comparacoes
18
19 int* copiaVetor(int* v, int n){
20     int i;
21     int *v2;
22     v2 = (int*) malloc (n*sizeof(int));
23     for(i=0; i<n; i++) v2[i] = v[i];
24     return v2;
25 }
26
27 Aluno* copiaVetor_aluno(Aluno* v, int n){
28     int i;
29     Aluno *v2;
30     v2 = (Aluno*) malloc (n*sizeof(Aluno));
31     for(i=0; i<n; i++) v2[i] = v[i];
32     return v2;
33 }
34
35 void imprimeVetor(int* v, int n){
36     int i, prim = 1;
37     printf("[");
38     for(i=0; i<n; i++)
39         if(prim){ printf("%d", v[i]); prim = 0; }
40         else printf(", %d", v[i]);
41     printf("]\n");
42 }
43
44 void preencheAleatorio(int* v, int n, int ini, int fim){
45     int i;
46     for(i=0; i<n; i++)
47         v[i] = ini + rand() % (fim-ini + 1);
48 }
49
50 void troca_nome(Aluno* a, Aluno *b){
51     Aluno aux = *a;
52     *a = *b;
53     *b = aux;
54 }
55
56
57
```

```
58 int buscaSequencial_nome(Aluno *v, int n, char *elem){
59     int i;
60     for(i=0; i<n; i++){
61         comp++;
62         if(strcmp(v[i].nome, elem) == 0)
63             return i; //Elemento encontrado
64     }
65     return -1; //Elemento nao encontrado
66 }
67
68 int particao_nome(Aluno *v, int ini, int fim){
69     int i = ini, j = fim;
70     Aluno pivo = v[(ini+fim)/2];
71     while (1) {
72         while(strcmp(v[i].nome, pivo.nome) < 0){ i++; } //procura algum >= pivo do
lado esquerdo
73         while(strcmp(v[j].nome, pivo.nome) > 0){ j--; } //procura algum <= pivo do
lado direito
74
75         if(i<j){
76             troca_nome(&v[i], &v[j]); //troca os elementos encontrados
77             i++;
78             j--;
79         }else
80             return j; //retorna o local onde foi feita a particao
81     }
82 }
83
84 void QuickSort_nome(Aluno *v, int ini, int fim){
85     if(ini < fim){
86         int q = particao_nome(v, ini, fim);
87         QuickSort_nome(v, ini, q);
88         QuickSort_nome(v, q+1, fim);
89     }
90 }
91
92 int rec_buscaBinaria_nome(Aluno *v, int ini, int fim, char *elem){
93     if(ini > fim) return -1;
94     int meio = (ini + fim)/2;
95     comp++;
96     if(strcmp(v[meio].nome, elem) == 0)
97         return meio;
98     else
99         if(strcmp(v[meio].nome, elem) > 0)
100             return rec_buscaBinaria_nome(v, ini, meio-1, elem);
101         else
102             return rec_buscaBinaria_nome(v, meio+1, fim, elem);
103 }
104
105 int it_buscaBinaria_nome(Aluno *v, int ini, int fim, char *elem){
106     int meio;
107     while(ini <= fim){
108         meio = (ini + fim)/2;
109         comp++;
110         if(strcmp(v[meio].nome, elem) == 0) return meio;
111         else
112             if(strcmp(v[meio].nome, elem) > 0)
113                 fim = meio-1;
114             else
115                 ini = meio+1;
```

```
116     }
117     return -1;
118 }
119
120 void troca_matricula(Aluno* a, Aluno *b){
121     Aluno aux = *a;
122     *a = *b;
123     *b = aux;
124 }
125
126 int buscaSequencial_matricula(Aluno *v, int n, int elem){
127     int i;
128     for(i=0; i<n; i++){
129         comp++;
130         if(v[i].matricula == elem)
131             return i; //Elemento encontrado
132     }
133     return -1; //Elemento encontrado
134 }
135
136 int particao_matricula(Aluno *v, int ini, int fim){
137     int i = ini, j = fim;
138     Aluno pivo = v[(ini+fim)/2];
139     while (1) {
140         while(v[i].matricula < pivo.matricula){ i++; } //procura algum >= pivo do
lado esquerdo
141         while(v[j].matricula > pivo.matricula){ j--; } //procura algum <= pivo do
lado direito
142
143         if(i<j){
144             troca_matricula(&v[i], &v[j]); //troca os elementos encontrados
145             i++;
146             j--;
147         }else
148             return j; //retorna o local onde foi feita a particao
149     }
150 }
151
152 void QuickSort_matricula(Aluno *v, int ini, int fim){
153     if(ini < fim ){
154         int q = particao_matricula(v, ini, fim);
155         QuickSort_matricula(v, ini, q);
156         QuickSort_matricula(v, q+1, fim);
157     }
158 }
159
160 int rec_buscaBinaria_matricula(Aluno *v, int ini, int fim, int elem){
161     if(ini > fim) return -1;
162     int meio = (ini + fim)/2;
163     comp++;
164     if(v[meio].matricula == elem)
165         return meio;
166     else
167         if(elem < v[meio].matricula)
168             return rec_buscaBinaria_matricula(v, ini, meio-1, elem);
169         else
170             return rec_buscaBinaria_matricula(v, meio+1, fim, elem);
171 }
172
173 int it_buscaBinaria_matricula(Aluno *v, int ini, int fim, int elem){
```

```
174     int meio;
175     while(ini <= fim){
176         meio = (ini + fim)/2;
177         comp++;
178         if(elem == v[meio].matricula) return meio;
179         else
180             if(elem < v[meio].matricula)
181                 fim = meio-1;
182             else
183                 ini = meio+1;
184     }
185     return -1;
186 }
187
188 // void troca(int* a, int *b){
189 //     int aux = *a;
190 //     *a = *b;
191 //     *b = aux;
192 // }
193
194 // int buscaSequencial(int *v, int n, int elem){
195 //     int i;
196 //     for(i=0; i<n; i++){
197 //         comp++;
198 //         if(v[i] == elem)
199 //             return i; //Elemento encontrado
200 //     }
201 //     return -1; //Elemento encontrado
202 // }
203
204 // int particao(int *v, int ini, int fim){
205 //     int i = ini, j = fim;
206 //     int pivo = v[(ini+fim)/2];
207 //     while (1) {
208 //         while(v[i] < pivo){ i++; } //procura algum >= pivo do lado esquerdo
209 //         while(v[j] > pivo){ j--; } //procura algum <= pivo do lado direito
210 //
211 //         if(i<j){
212 //             troca(&v[i], &v[j]); //troca os elementos encontrados
213 //             i++;
214 //             j--;
215 //         }else
216 //             return j; //retorna o local onde foi feita a particao
217 //     }
218 // }
219
220 // void QuickSort(int *v, int ini, int fim){
221 //     if(ini < fim ){
222 //         int q = particao(v, ini, fim);
223 //         QuickSort(v, ini, q);
224 //         QuickSort(v, q+1, fim);
225 //     }
226 // }
227
228 // int rec_buscaBinaria(int *v, int ini, int fim, int elem){
229 //     if(ini > fim) return -1;
230 //     int meio = (ini + fim)/2;
231 //     comp++;
232 //     if(v[meio] == elem)
233 //         return meio;
```

```
234 //     else
235 //         if(elem < v[meio])
236 //             return rec_buscaBinaria(v, ini, meio-1, elem);
237 //         else
238 //             return rec_buscaBinaria(v, meio+1, fim, elem);
239 // }
240
241 // int it_buscaBinaria(int *v, int ini, int fim, int elem){
242 //     int meio;
243 //     while(ini <= fim){
244 //         meio = (ini + fim)/2;
245 //         comp++;
246 //         if(elem == v[meio]) return meio;
247 //         else
248 //             if(elem < v[meio])
249 //                 fim = meio-1;
250 //             else
251 //                 ini = meio+1;
252 //     }
253 //     return -1;
254 // }
255
256 #endif
```

1.3/main.c

```
1  #include "busca.h"
2
3
4  int main(){
5
6      //Atribuicoes iniciais
7      srand(time(NULL));
8      comp = 0;
9      clock_t t;
10
11     /*
12     //Template de Calculo do Tempo de Execucao
13     t = clock();
14     //Chamada do Algoritmo aqui...
15     t = clock() - t;
16     printf ("It took me %d clicks (%f seconds).\n",t,((float)t)/CLOCKS_PER_SEC);
17     */
18
19     Aluno *v;
20     int n, matricula;
21     char nome[MAX];
22     char buffer[MAX];
23     printf("Digite o tamanho do vetor:\n");
24     scanf("%d", &n);
25     v = (Aluno*) malloc (n*sizeof(Aluno));
26
27     for(int i = 0; i < n; i++){
28         printf("Digite a matricula do Aluno:\n");
29         scanf("%d", &v[i].matricula);
30         printf("Digite o nome do Aluno:\n");
31         fgets(buffer, sizeof(buffer), stdin);
32         scanf("%200[^\n]", v[i].nome);
33         v[i].nota1 = 10;
34         v[i].nota2 = 10;
35         v[i].nota3 = 10;
36     }
37     //preencheAleatorio(v, n, 1, n);
38     //imprimeVetor(v, n);
39
40     //por nome
41     QuickSort_nome(v, 0, n-1);
42
43     printf("Digite um nome para busca:\n");
44     fgets(buffer, sizeof(buffer), stdin);
45     scanf("%200[^\n]", nome);
46
47     int ind;
48
49     t = clock();
50     ind = buscaSequencial_nome(v, n, nome);
51     t = clock() - t;
52     printf("-----Informacoes Busca Sequencial:\n");
53     printf("Tempo Execucao:  %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
54     printf("Comparacoes: %d\n", comp);
55
56     //imprimeVetor(v, n);
57
```



```
58     if(ind != -1)
59         printf("O elemento %s foi encontrado na pos %d.\n", nome, ind);
60     else
61         printf("O elemento %s NAO foi encontrado!\n", nome);
62
63
64
65     //imprimeVetor(v, n);
66
67     comp = 0;
68     t = clock();
69     ind = rec_buscaBinaria_nome(v, 0, n-1, nome);
70     t = clock() - t;
71     printf("-----Informacoes Busca Binaria Recursiva:\n");
72     printf("Tempo Execucao:  %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
73     printf("Comparacoes: %d\n", comp);
74
75     if(ind != -1)
76         printf("O elemento %s foi encontrado na pos %d.\n", nome, ind);
77     else
78         printf("O elemento %s NAO foi encontrado!\n", nome);
79
80
81     comp = 0;
82     t = clock();
83     ind = it_buscaBinaria_nome(v, 0, n-1, nome);
84     t = clock() - t;
85     printf("-----Informacoes Busca Binaria Iterativa:\n");
86     printf("Tempo Execucao:  %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
87     printf("Comparacoes: %d\n", comp);
88
89     if(ind != -1)
90         printf("O elemento %s foi encontrado na pos %d.\n", nome, ind);
91     else
92         printf("O elemento %s NAO foi encontrado!\n", nome);
93
94     //por matricula
95     QuickSort_matricula(v, 0, n-1);
96
97     printf("Digite uma matricula para busca:\n");
98     scanf("%d", &matricula);
99
100
101     t = clock();
102     ind = buscaSequencial_matricula(v, n, matricula);
103     t = clock() - t;
104     printf("-----Informacoes Busca Sequencial:\n");
105     printf("Tempo Execucao:  %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
106     printf("Comparacoes: %d\n", comp);
107
108     //imprimeVetor(v, n);
109
110     if(ind != -1)
111         printf("O elemento %d foi encontrado na pos %d.\n", matricula, ind);
112     else
113         printf("O elemento %d NAO foi encontrado!\n", matricula);
114
115
116
117     //imprimeVetor(v, n);
```

```
118
119     comp = 0;
120     t = clock();
121     ind = rec_buscaBinaria_matricula(v, 0, n-1, matricula);
122     t = clock() - t;
123     printf("-----Informacoes Busca Binaria Recursiva:\n");
124     printf("Tempo Execucao:  %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
125     printf("Comparacoes: %d\n", comp);
126
127     if(ind != -1)
128         printf("0 elemento %d foi encontrado na pos %d.\n", matricula, ind);
129     else
130         printf("0 elemento %d NAO foi encontrado!\n", matricula);
131
132
133     comp = 0;
134     t = clock();
135     ind = it_buscaBinaria_matricula(v, 0, n-1, matricula);
136     t = clock() - t;
137     printf("-----Informacoes Busca Binaria Iterativa:\n");
138     printf("Tempo Execucao:  %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
139     printf("Comparacoes: %d\n", comp);
140
141     if(ind != -1)
142         printf("0 elemento %d foi encontrado na pos %d.\n", matricula, ind);
143     else
144         printf("0 elemento %d NAO foi encontrado!\n", matricula);
145
146     free(v);
147     return 0;
148 }
```

```
[lucascosta@fedora 1.3]$ ./main
Digite o tamanho do vetor:
3
Digite a matricula do Aluno:
1245
Digite o nome do Aluno:
Lucas
Digite a matricula do Aluno:
1456
Digite o nome do Aluno:
Larissa
Digite a matricula do Aluno:
1485
Digite o nome do Aluno:
Luana
Digite um nome para busca:
Lucas
-----Informacoes Busca Sequencial:
Tempo Execucao: 0.000005 seconds.
Comparacoes: 3
0 elemento Lucas foi encontrado na pos 2.
-----Informacoes Busca Binaria Recursiva:
Tempo Execucao: 0.000003 seconds.
Comparacoes: 2
0 elemento Lucas foi encontrado na pos 2.
-----Informacoes Busca Binaria Iterativa:
Tempo Execucao: 0.000002 seconds.
Comparacoes: 2
0 elemento Lucas foi encontrado na pos 2.
Digite uma matricula para busca:
1485
-----Informacoes Busca Sequencial:
Tempo Execucao: 0.000005 seconds.
Comparacoes: 5
0 elemento 1485 foi encontrado na pos 2.
-----Informacoes Busca Binaria Recursiva:
Tempo Execucao: 0.000003 seconds.
Comparacoes: 2
0 elemento 1485 foi encontrado na pos 2.
-----Informacoes Busca Binaria Iterativa:
Tempo Execucao: 0.000002 seconds.
Comparacoes: 2
0 elemento 1485 foi encontrado na pos 2.
[lucascosta@fedora 1.3]$ █
```

2.1/hash.h

```
1  #ifndef HASH_H
2  #define HASH_H
3
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <string.h>
7
8  typedef struct{
9      int **tabela;
10     int tam, qtd;
11 }Hash;
12
13
14 Hash* criaHash(int t){
15     Hash* h;
16     h = (Hash*) malloc (sizeof(Hash));
17     if(h != NULL){
18         h->tam = t; h->qtd = 0;
19         h->tabela = (int**) malloc (t*sizeof(int*));
20         if(h->tabela == NULL) return NULL;
21         int i;
22         for(i = 0; i<t; i++)
23             h->tabela[i] = NULL;
24     }
25     return h;
26 }
27
28
29 void destroiHash(Hash *h){
30     if(h != NULL){
31         int i;
32         for(i = 0; i<h->tam; i++)
33             if(h->tabela[i] != NULL)
34                 free(h->tabela[i]);
35         free(h->tabela);
36         free(h);
37     }
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     }
78     return valor;
79 }
80 int insereHash_semTratar_div(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 insereHash_semTratar_mul(Hash* h, int elem){
95     if(h == NULL) return 0;
96     int pos = chaveMultiplicacao(elem, h->tam);
97
98     if(h->tabela[pos] == NULL){
99         int* novo = (int*) malloc (sizeof(int));
100         if(novo == NULL) return 0;
101         *novo = elem;
102         h->tabela[pos] = novo;
103         h->qtd++;
104     }else *(h->tabela[pos]) = elem;
105     return 1;
106 }
107
108 int insereHash_semTratar_dobra(Hash* h, int elem){
109     if(h == NULL) return 0;
110     int pos = chaveDobra(elem, h->tam);
111
112     if(h->tabela[pos] == NULL){
113         int* novo = (int*) malloc (sizeof(int));
114         if(novo == NULL) return 0;
115         *novo = elem;
116         h->tabela[pos] = novo;
117         h->qtd++;
```

```
118     }else *(h->tabela[pos]) = elem;
119     return 1;
120 }
121
122 int buscaHash_semTratar_div(Hash* h, int elem, int *p){
123     if(h == NULL) return 0;
124     int pos = chaveDivisao(elem, h->tam);
125     if(h->tabela[pos] == NULL) return 0;
126     if(*(h->tabela[pos]) == elem){
127         *p = *(h->tabela[pos]);
128         return 1;
129     }
130     return 0;
131 }
132
133 int buscaHash_semTratar_mul(Hash* h, int elem, int *p){
134     if(h == NULL) return 0;
135     int pos = chaveMultiplicacao(elem, h->tam);
136     if(h->tabela[pos] == NULL) return 0;
137     if(*(h->tabela[pos]) == elem){
138         *p = *(h->tabela[pos]);
139         return 1;
140     }
141     return 0;
142 }
143
144 int buscaHash_semTratar_dobra(Hash* h, int elem, int *p){
145     if(h == NULL) return 0;
146     int pos = chaveDobra(elem, h->tam);
147     if(h->tabela[pos] == NULL) return 0;
148     if(*(h->tabela[pos]) == elem){
149         *p = *(h->tabela[pos]);
150         return 1;
151     }
152     return 0;
153 }
154
155 int sondagemLinear(int pos, int i, int tam){
156     return ( (pos + i) & 0x7FFFFFFF) % tam;
157 }
158
159 int sondagemQuadratica(int pos, int i, int tam){
160     pos = pos + 2*i + 5*i*i;
161     return ( pos & 0x7FFFFFFF) % tam;
162 }
163
164 int sondagemDuploHash(int H1, int chave, int i, int tam){
165     int H2 = chaveDivisao(chave, tam-1) + 1;
166     return ( (H1 + i*H2) & 0x7FFFFFFF) % tam;
167 }
168
169 int insereHash_EnderAberto(Hash* h, int elem){
170     if(h == NULL) return 0;
171     int i, pos, newPos;
172     pos = chaveDivisao(elem, h->tam);
173     for(i=0; i<h->tam; i++){
174         newPos = sondagemLinear(pos, i, h->tam);
175         //newPos = sondagemQuadratica(pos, i, h->tam);
176         //newPos = sondagemDuploHash(pos, elem, i, h->tam);
177         if(h->tabela[newPos] == NULL){
```

```
178     int* novo = (int*) malloc (sizeof(int));
179     if(novo == NULL) return 0;
180     *novo = elem;
181     h->tabela[newPos] = novo;
182     h->qtd++;
183     return 1;
184 }
185 }
186 return 0;
187 }
188
189 int buscaHash_EnderAberto(Hash* h, int elem, int *p){
190     if(h == NULL) return 0;
191     int i, pos, newPos;
192     pos = chaveDivisao(elem, h->tam);
193     for(i=0; i<h->tam; i++){
194         newPos = sondagemLinear(pos, i, h->tam);
195         //newPos = sondagemQuadratica(pos, i, h->tam);
196         //newPos = sondagemDuploHash(pos, elem, i, h->tam);
197         if(h->tabela[newPos] == NULL) return 0;
198         if(*(h->tabela[newPos]) == elem){
199             *p = *(h->tabela[newPos]);
200             return 1;
201         }
202     }
203     return 0;
204 }
205
206 void imprimeHash(Hash *h){
207     if(h == NULL) return;
208     int i;
209     for(i=0; i<h->tam; i++){
210         printf("%d: ", i);
211         if(h->tabela[i] == NULL) printf("NULL\n");
212         else printf("%d\n", *(h->tabela[i]));
213     }
214 }
215
216 #endif
```

2.1/main.c

```
1  #include <stdio.h>
2  #include "hash.h"
3
4  // void binary(int n){
5  //   if(n<2)
6  //       printf("%d", n%2);
7  //   else{
8  //       binary(n/2);
9  //       printf("%d", n%2);
10 //   }
11 // }
12
13 int main(){
14
15     // int x = 1;
16     // int d;
17     // printf("%d: ", x); binary(x); printf("\n");
18
19     // d = x << 4;
20
21     // printf("%d: ", d); binary(d); printf("\n");
22
23     // d = d >> 2;
24
25     // printf("%d: ", d); binary(d); printf("\n");
26
27     Hash *H;;
28     H = criaHash(5);
29
30     int busca;
31
32
33     insereHash_semTratar_div(H, 31);
34     insereHash_semTratar_div(H, 32);
35     insereHash_semTratar_div(H, 64);
36
37     printf("Divisão:.\n");
38
39     if(!buscaHash_semTratar_div(H, 64, &busca))
40         printf("Elemento NAO encontrado.\n");
41     else
42         printf("Elemento %d encontrado.\n", busca);
43     if(!buscaHash_semTratar_div(H, 31, &busca))
44         printf("Elemento NAO encontrado.\n");
45     else
46         printf("Elemento %d encontrado.\n", busca);
47     if(!buscaHash_semTratar_div(H, 33, &busca))
48         printf("Elemento NAO encontrado.\n");
49     else
50         printf("Elemento %d encontrado.\n", busca);
51
52     imprimeHash(H);
53     destroiHash(H);
54     H = criaHash(5);
55
56     insereHash_semTratar_mul(H, 31);
57     insereHash_semTratar_mul(H, 32);
```



```
58     insereHash_semTratar_mul(H, 64);
59
60     printf("Multiplicação:.\n");
61
62     if(!buscaHash_semTratar_mul(H, 64, &busca))
63         printf("Elemento NAO encontrado.\n");
64     else
65         printf("Elemento %d encontrado.\n", busca);
66     if(!buscaHash_semTratar_mul(H, 31, &busca))
67         printf("Elemento NAO encontrado.\n");
68     else
69         printf("Elemento %d encontrado.\n", busca);
70     if(!buscaHash_semTratar_mul(H, 33, &busca))
71         printf("Elemento NAO encontrado.\n");
72     else
73         printf("Elemento %d encontrado.\n", busca);
74
75     imprimeHash(H);
76     destroiHash(H);
77     H = criaHash(10);
78
79     insereHash_semTratar_dobra(H, 31);
80     insereHash_semTratar_dobra(H, 32);
81     insereHash_semTratar_dobra(H, 64);
82
83     printf("Dobra:.\n");
84
85     if(!buscaHash_semTratar_dobra(H, 64, &busca))
86         printf("Elemento NAO encontrado.\n");
87     else
88         printf("Elemento %d encontrado.\n", busca);
89     if(!buscaHash_semTratar_dobra(H, 31, &busca))
90         printf("Elemento NAO encontrado.\n");
91     else
92         printf("Elemento %d encontrado.\n", busca);
93     if(!buscaHash_semTratar_dobra(H, 33, &busca))
94         printf("Elemento NAO encontrado.\n");
95     else
96         printf("Elemento %d encontrado.\n", busca);
97
98     imprimeHash(H);
99     destroiHash(H);
100
101     // insereHash_EnderAberto(H, 31);
102     // insereHash_EnderAberto(H, 32);
103     // insereHash_EnderAberto(H, 62);
104
105     // if(!buscaHash_EnderAberto(H, 31, &busca))
106     //     printf("Elemento NAO encontrado.\n");
107     // else
108     //     printf("Elemento %d encontrado.\n", busca);
109
110     // if(!buscaHash_EnderAberto(H, 62, &busca))
111     //     printf("Elemento NAO encontrado.\n");
112     // else
113     //     printf("Elemento %d encontrado.\n", busca);
114
115     return 0;
116 }
```

```
[lucascosta@fedora 2.1]$ gcc main.c -o main
[lucascosta@fedora 2.1]$ ./main
Divisão:.
Elemento 64 encontrado.
Elemento 31 encontrado.
Elemento NAO encontrado.
0: NULL
1: 31
2: 32
3: NULL
4: 64
Multiplicação:.
Elemento 64 encontrado.
Elemento 31 encontrado.
Elemento NAO encontrado.
0: 31
1: NULL
2: 64
3: 32
4: NULL
Dobra:.
Elemento 64 encontrado.
Elemento NAO encontrado.
Elemento NAO encontrado.
0: NULL
1: NULL
2: NULL
3: NULL
4: 32
5: NULL
6: NULL
7: NULL
8: 64
9: NULL
[lucascosta@fedora 2.1]$
```

2.2/hash.h

```
1  #ifndef HASH_H
2  #define HASH_H
3
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <string.h>
7
8  typedef struct{
9      int **tabela;
10     int tam, qtd;
11 }Hash;
12
13
14 Hash* criaHash(int t){
15     Hash* h;
16     h = (Hash*) malloc (sizeof(Hash));
17     if(h != NULL){
18         h->tam = t; h->qtd = 0;
19         h->tabela = (int**) malloc (t*sizeof(int*));
20         if(h->tabela == NULL) return NULL;
21         int i;
22         for(i = 0; i<t; i++)
23             h->tabela[i] = NULL;
24     }
25     return h;
26 }
27
28
29 void destroiHash(Hash *h){
30     if(h != NULL){
31         int i;
32         for(i = 0; i<h->tam; i++)
33             if(h->tabela[i] != NULL)
34                 free(h->tabela[i]);
35         free(h->tabela);
36         free(h);
37     }
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     }
78     return valor;
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     }
101     return 1;
102 }
103
104 int sondagemLinear(int pos, int i, int tam){
105     return ( (pos + i) & 0x7FFFFFFF) % tam;
106 }
107
108 int sondagemQuadratica(int pos, int i, int tam){
109     pos = pos + 2*i + 5*i*i;
110     return ( pos & 0x7FFFFFFF) % tam;
111 }
112
113 int sondagemDuploHash(int H1, int chave, int i, int tam){
114     int H2 = chaveDivisao(chave, tam-1) + 1;
115     return ( (H1 + i*H2) & 0x7FFFFFFF) % tam;
116 }
117 }
```

```
118
119 int insereHash_EnderAberto_lin(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 insereHash_EnderAberto_quad(Hash* h, int elem){
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){
148             int* novo = (int*) malloc (sizeof(int));
149             if(novo == NULL) return 0;
150             *novo = elem;
151             h->tabela[newPos] = novo;
152             h->qtd++;
153             return 1;
154         }
155     }
156     return 0;
157 }
158
159 int insereHash_EnderAberto_duplo(Hash* h, int elem){
160     if(h == NULL) return 0;
161     int i, pos, newPos;
162     pos = chaveDivisao(elem, h->tam);
163     for(i=0; i<h->tam; i++){
164         //newPos = sondagemLinear(pos, i, h->tam);
165         //newPos = sondagemQuadratica(pos, i, h->tam);
166         newPos = sondagemDuploHash(pos, elem, i, h->tam);
167         if(h->tabela[newPos] == NULL){
168             int* novo = (int*) malloc (sizeof(int));
169             if(novo == NULL) return 0;
170             *novo = elem;
171             h->tabela[newPos] = novo;
172             h->qtd++;
173             return 1;
174         }
175     }
176     return 0;
177 }
```

```
178
179 int buscaHash_EnderAberto_lin(Hash* h, int elem, int *p){
180     if(h == NULL) return 0;
181     int i, pos, newPos;
182     pos = chaveDivisao(elem, h->tam);
183     for(i=0; i<h->tam; i++){
184         newPos = sondagemLinear(pos, i, h->tam);
185         //newPos = sondagemQuadratica(pos, i, h->tam);
186         //newPos = sondagemDuploHash(pos, elem, i, h->tam);
187         if(h->tabela[newPos] == NULL) return 0;
188         if(*(h->tabela[newPos]) == elem){
189             *p = *(h->tabela[newPos]);
190             return 1;
191         }
192     }
193     return 0;
194 }
195
196 int buscaHash_EnderAberto_quad(Hash* h, int elem, int *p){
197     if(h == NULL) return 0;
198     int i, pos, newPos;
199     pos = chaveDivisao(elem, h->tam);
200     for(i=0; i<h->tam; i++){
201         // newPos = sondagemLinear(pos, i, h->tam);
202         newPos = sondagemQuadratica(pos, i, h->tam);
203         //newPos = sondagemDuploHash(pos, elem, i, h->tam);
204         if(h->tabela[newPos] == NULL) return 0;
205         if(*(h->tabela[newPos]) == elem){
206             *p = *(h->tabela[newPos]);
207             return 1;
208         }
209     }
210     return 0;
211 }
212
213 int buscaHash_EnderAberto_duplo(Hash* h, int elem, int *p){
214     if(h == NULL) return 0;
215     int i, pos, newPos;
216     pos = chaveDivisao(elem, h->tam);
217     for(i=0; i<h->tam; i++){
218         //newPos = sondagemLinear(pos, i, h->tam);
219         //newPos = sondagemQuadratica(pos, i, h->tam);
220         newPos = sondagemDuploHash(pos, elem, i, h->tam);
221         if(h->tabela[newPos] == NULL) return 0;
222         if(*(h->tabela[newPos]) == elem){
223             *p = *(h->tabela[newPos]);
224             return 1;
225         }
226     }
227     return 0;
228 }
229
230
231
232 void imprimeHash(Hash *h){
233     if(h == NULL) return;
234     int i;
235     for(i=0; i<h->tam; i++){
236         printf("%d: ", i);
237         if(h->tabela[i] == NULL) printf("NULL\n");
```

```
238 |         else printf("%d\n", *(h->tabela[i]));  
239 |     }  
240 | }  
241 |  
242 | #endif
```

2.2/main.c

```
1  #include <stdio.h>
2  #include "hash.h"
3
4  // void binary(int n){
5  //   if(n<2)
6  //       printf("%d", n%2);
7  //   else{
8  //       binary(n/2);
9  //       printf("%d", n%2);
10 //   }
11 // }
12
13 int main(){
14
15     // int x = 1;
16     // int d;
17     // printf("%d: ", x); binary(x); printf("\n");
18
19     // d = x << 4;
20
21     // printf("%d: ", d); binary(d); printf("\n");
22
23     // d = d >> 2;
24
25     // printf("%d: ", d); binary(d); printf("\n");
26
27     Hash *H;;
28     H = criaHash(5);
29
30     int busca;
31
32     insereHash_EnderAberto_lin(H, 31);
33     insereHash_EnderAberto_lin(H, 32);
34     insereHash_EnderAberto_lin(H, 64);
35
36     printf("Linear:.\n");
37
38     if(!buscaHash_EnderAberto_lin(H, 64, &busca))
39         printf("Elemento NAO encontrado.\n");
40     else
41         printf("Elemento %d encontrado.\n", busca);
42     if(!buscaHash_EnderAberto_lin(H, 31, &busca))
43         printf("Elemento NAO encontrado.\n");
44     else
45         printf("Elemento %d encontrado.\n", busca);
46     if(!buscaHash_EnderAberto_lin(H, 33, &busca))
47         printf("Elemento NAO encontrado.\n");
48     else
49         printf("Elemento %d encontrado.\n", busca);
50
51     imprimeHash(H);
52     destroiHash(H);
53     H = criaHash(5);
54
55     insereHash_EnderAberto_quad(H, 31);
56     insereHash_EnderAberto_quad(H, 32);
57     insereHash_EnderAberto_quad(H, 64);
```



```
58
59     printf("Quadratica:.\n");
60
61     if(!buscaHash_EnderAberto_quad(H, 64, &busca))
62         printf("Elemento NAO encontrado.\n");
63     else
64         printf("Elemento %d encontrado.\n", busca);
65     if(!buscaHash_EnderAberto_quad(H, 31, &busca))
66         printf("Elemento NAO encontrado.\n");
67     else
68         printf("Elemento %d encontrado.\n", busca);
69     if(!buscaHash_EnderAberto_quad(H, 33, &busca))
70         printf("Elemento NAO encontrado.\n");
71     else
72         printf("Elemento %d encontrado.\n", busca);
73
74     imprimeHash(H);
75     destroiHash(H);
76     H = criaHash(5);
77
78     insereHash_EnderAberto_duplo(H, 31);
79     insereHash_EnderAberto_duplo(H, 32);
80     insereHash_EnderAberto_duplo(H, 64);
81
82     printf("Duplo:.\n");
83
84     if(!buscaHash_EnderAberto_duplo(H, 64, &busca))
85         printf("Elemento NAO encontrado.\n");
86     else
87         printf("Elemento %d encontrado.\n", busca);
88     if(!buscaHash_EnderAberto_duplo(H, 31, &busca))
89         printf("Elemento NAO encontrado.\n");
90     else
91         printf("Elemento %d encontrado.\n", busca);
92     if(!buscaHash_EnderAberto_duplo(H, 33, &busca))
93         printf("Elemento NAO encontrado.\n");
94     else
95         printf("Elemento %d encontrado.\n", busca);
96
97     imprimeHash(H);
98     destroiHash(H);
99
100     return 0;
101
102 }
```

```
[lucascosta@fedora 2.2]$ gcc main.c -o main
[lucascosta@fedora 2.2]$ ./main
Linear:.
Elemento 64 encontrado.
Elemento 31 encontrado.
Elemento NAO encontrado.
0: NULL
1: 31
2: 32
3: NULL
4: 64
Quadratica:.
Elemento 64 encontrado.
Elemento 31 encontrado.
Elemento NAO encontrado.
0: NULL
1: 31
2: 32
3: NULL
4: 64
Duplo:.
Elemento 64 encontrado.
Elemento 31 encontrado.
Elemento NAO encontrado.
0: NULL
1: 31
2: 32
3: NULL
4: 64
[lucascosta@fedora 2.2]$
```

2.3/hash.h

```
1  #ifndef HASH_H
2  #define HASH_H
3
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <string.h>
7  #include "LSE.h"
8
9  typedef struct{
10     Lista **tabela;
11     int tam, qtd;
12 }Hash;
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 (t*sizeof(Lista*));
21         if(h->tabela == NULL) return NULL;
22         int i;
23         for(i = 0; i<t; i++)
24             h->tabela[i] = NULL;
25     }
26     return h;
27 }
28
29
30 void destroiHash(Hash *h){
31     if(h != NULL){
32         int i;
33         for(i = 0; i<h->tam; i++)
34             if(h->tabela[i] != NULL)
35                 destroiLista(h->tabela[i]);
36         free(h->tabela);
37         free(h);
38     }
39 }
40
41 int chaveDivisao(int chave, int tam){
42     return (chave & 0x7FFFFFFF) % tam;
43 }
44
45 int chaveMultiplicacao(int chave, int tam){
46     float A = 0.6180339887; //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--; r = p << n_bits; }
```

```
58
59     n_bits++;
60     pos = chave;
61     while(pos > tam){
62         int metade_bits = n_bits/2;
63         int parte1 = pos >> metade_bits;
64         parte1 = parte1 << metade_bits;
65         int parte2 = pos ^ parte1;
66         parte1 = pos >> metade_bits;
67         pos = parte1 ^ parte2;
68         n_bits = n_bits/2;
69     }
70     return pos;
71 }
72
73 int valorString(char *str){
74     int i, valor = 1;
75     int tam = strlen(str);
76     for(i=0; i<tam; i++){
77         valor = 31*valor + (i+1)*((int) str[i]);
78     }
79     return valor;
80 }
81
82 int insereHashLSE(Hash* h, int elem){
83     if(h == NULL) return 0;
84     int pos = chaveDivisao(elem, h->tam);
85     if(h->tabela[pos] == NULL)
86         h->tabela[pos] = criaLista();
87     insereIni(h->tabela[pos], elem);
88     h->qtd++;
89     return 1;
90 }
91
92 int buscaHashLSE(Hash* h, int elem, int *p){
93     if(h == NULL) return 0;
94     int pos = chaveDivisao(elem, h->tam);
95     if(h->tabela[pos] == NULL) return 0;
96     return listaBuscaElem(h->tabela[pos], elem, p);
97 }
98
99 void imprimeHash(Hash *h){
100     if(h == NULL) return;
101     int i;
102     for(i=0; i<h->tam; i++){
103         printf("%d: ", i);
104         if(h->tabela[i] == NULL) printf("NULL\n");
105         else imprimeLista(h->tabela[i]);
106     }
107 }
108 #endif
```

2.3/LSE.h

```
1  #ifndef LISTASE_H
2  #define LISTASE_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     Lista *li;
16     li = (Lista*) malloc (sizeof(Lista));
17     if(li != NULL){
18         *li = NULL;
19     }
20     return li;
21 }
22
23 int listaVazia(Lista *li){
24     if(li == NULL) return 1;
25     if(*li == NULL) return 1;//True - Vazia!
26     return 0;//False - tem elemento!
27 }
28
29 NO* alocarNO(){
30     return (NO*) malloc (sizeof(NO));
31 }
32
33 void liberarNO(NO* q){
34     free(q);
35 }
36
37 int listaBuscaElem(Lista* li, int elem, int *p){
38     if(li == NULL) return 0;
39     NO* aux = *li;
40     while(aux != NULL){
41         if(aux->info == elem){
42             *p = aux->info;
43             return 1;
44         }
45         aux = aux->prox;
46     }
47     return 0;
48 }
49
50 int insereIni(Lista* li, int elem){
51     if(li == NULL) return 0;
52     NO* novo = alocarNO();
53     if(novo == NULL) return 0;
54     novo->info = elem;
55     novo->prox = *li;
56     *li = novo;
57     return 1;
```

```
58 }
59
60 int insereFim(Lista* li, int elem){
61     if(li == NULL) return 0;
62     NO* novo = alocarNO();
63     if(novo == NULL) return 0;
64     novo->info = elem;
65     novo->prox = NULL;
66     if(listaVazia(li)){
67         *li = novo;
68     }else{
69         NO* aux = *li;
70         while(aux->prox != NULL)
71             aux = aux->prox;
72         aux->prox = novo;
73     }
74     return 1;
75 }
76
77 int removeIni(Lista* li){
78     if(li == NULL) return 0;
79     if(listaVazia(li)) return 0;
80     NO* aux = *li;
81     *li = aux->prox;
82     liberarNO(aux);
83     return 1;
84 }
85
86 int removeFim(Lista* li){
87     if(li == NULL) return 0;
88     if(listaVazia(li)) return 0;
89     NO* ant, *aux = *li;
90     while(aux->prox != NULL){
91         ant = aux;
92         aux = aux->prox;
93     }
94     if(aux == *li)
95         *li = aux->prox;
96     else
97         ant->prox = aux->prox;
98     liberarNO(aux);
99     return 1;
100 }
101
102 void imprimeLista(Lista* li){
103     if(li == NULL) return;
104     if(listaVazia(li)){
105         printf("Lista Vazia!\n");
106         return;
107     }
108     //printf("Elementos:\n");
109     NO* aux = *li;
110     while(aux != NULL){
111         printf("%d ", aux->info);
112         aux = aux->prox;
113     }
114     printf("\n");
115 }
116
117 void destroiLista(Lista* li){
```

```
118 |         if(li != NULL){  
119 |             NO* aux;  
120 |             while((*li) != NULL){  
121 |                 aux = *li;  
122 |                 *li = (*li)->prox;  
123 |                 liberarNO(aux);  
124 |             }  
125 |             free(li);  
126 |         }  
127 |     }  
128 |  
129 | #endif
```

2.3/main.c

```
1  #include <stdio.h>
2  #include "hash.h"
3
4
5
6  int main(){
7
8      Hash *H;
9      H = criaHash(15);
10
11     int busca;
12
13     int i;
14     for(i=0; i<100; i++)
15         insereHashLSE(H, i);
16
17     imprimeHash(H);
18
19     if(buscaHashLSE(H, 12, &busca)){
20         printf("Elemento %d encontrado\n", busca);
21     }
22     else{
23         printf("Elemento NÃO encontrado\n");
24     }
25
26     if(buscaHashLSE(H, 102, &busca)){
27         printf("Elemento %d encontrado\n", busca);
28     }
29     else{
30         printf("Elemento NÃO encontrado\n");
31     }
32
33     destroiHash(H);
34
35     return 0;
36 }
```



```
[lucascosta@fedora 2.3]$ gcc main.c -o main
[lucascosta@fedora 2.3]$ ./main
0: 90 75 60 45 30 15 0
1: 91 76 61 46 31 16 1
2: 92 77 62 47 32 17 2
3: 93 78 63 48 33 18 3
4: 94 79 64 49 34 19 4
5: 95 80 65 50 35 20 5
6: 96 81 66 51 36 21 6
7: 97 82 67 52 37 22 7
8: 98 83 68 53 38 23 8
9: 99 84 69 54 39 24 9
10: 85 70 55 40 25 10
11: 86 71 56 41 26 11
12: 87 72 57 42 27 12
13: 88 73 58 43 28 13
14: 89 74 59 44 29 14
Elemento 12 encontrado
Elemento NÃO encontrado
[lucascosta@fedora 2.3]$
```