

UFSJ - Ciências da Computação

Laboratório de Programação 2

Roteiro 12

Nome: Geraldo Arthur Detomi

1.1) Reimplemente os algoritmos de busca sequencial e busca binária e teste-os comparando os valores finais de número de comparações e tempo de execução.

roteiro_12/tempo.h

```
1  #ifndef TEMPO_H
2  #define TEMPO_H
3
4  #include <stdio.h>
5  #include <sys/resource.h>
6  #include <sys/time.h>
7
8  // Estrutura para armazenar os tempos de execução
9  typedef struct Temporizador {
10     struct timeval start_tv, end_tv;
11     struct rusage start_usage, end_usage;
12 } Temporizador;
13
14 // Inicia a contagem do temporizador
15 void iniciarTemporizador(Temporizador *t);
16
17 // Finaliza a contagem do temporizador
18 void finalizarTemporizador(Temporizador *t);
19
20 // Calcula o tempo real decorrido em segundos
21 double calcularTempoReal(Temporizador *t);
22
23 // Calcula o tempo de CPU em modo usuário em segundos
24 double calcularTempoUsuario(Temporizador *t);
25
26 // Calcula o tempo de CPU em modo sistema/kernel em segundos
27 double calcularTempoSistema(Temporizador *t);
28
29 // Imprime todos os tempos medidos de forma formatada
30 void imprimirTempos(Temporizador *t);
31
32 #endif
```

roteiro_12/tempo.c

```
1  #include "tempo.h"
2
3  // Inicia a medição do tempo, armazenando os valores atuais
4  void iniciarTemporizador(Temporizador *t) {
5     gettimeofday(&t->start_tv, NULL);
6     getrusage(RUSAGE_SELF, &t->start_usage);
7 }
8
9  // Finaliza a medição do tempo, armazenando os valores finais
10 void finalizarTemporizador(Temporizador *t) {
11     gettimeofday(&t->end_tv, NULL);
12     getrusage(RUSAGE_SELF, &t->end_usage);
13 }
14
15 double calcularTempo(struct timeval inicio, struct timeval fim) {
16     time_t seg = fim.tv_sec - inicio.tv_sec;
17     suseconds_t microseg = fim.tv_usec - inicio.tv_usec;
18
19     // Ajusta caso microsegundos do fim sejam menores que os do início
20     if (microseg < 0) {
21         seg -= 1;
22         microseg += 1000000;
23     }
24
25     return (double)seg + (double)microseg / 1e6;
26 }
27
28 // Calcula diferença entre tempos reais (start e end) em segundos
29 double calcularTempoReal(Temporizador *t) {
30     return calcularTempo(t->start_tv, t->end_tv);
31 }
```

```

31 }
32
33 // Calcula diferença entre tempos de sistema (start e end) em segundos
34 double calcularTempoSistema(Temporizador *t) {
35     return calcularTempo(t->start_usage.ru_stime, t->end_usage.ru_stime);
36 }
37
38 // Imprime todos os tempos de execução de forma legível
39 void imprimirTempos(Temporizador *t) {
40     printf("Tempo de execução:\n");
41     printf("Tempo Real: %.8f segundos\n", calcularTempoReal(t));
42     printf("Tempo Sistema: %.8f segundos\n", calcularTempoSistema(t));
43 }

```

roteiro_12/1-1.c

```

1  #include "tempo.h"
2
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  void liberar_vetor(int **v) {
7      free(*v);
8      *v = NULL;
9  }
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++)
16         v2[i] = v[i];
17     return v2;
18 }
19
20 void imprimeVetor(int *v, int n) {
21     int i, prim = 1;
22     printf("[");
23     for (i = 0; i < n; i++)
24         if (prim) {
25             printf("%d", v[i]);
26             prim = 0;
27         } else
28             printf(", %d", v[i]);
29     printf("]\n");
30 }
31
32 int buscaSequencial(int *v, int n, int elem, int *comp) {
33     int i;
34     for (i = 0; i < n; i++) {
35         (*comp)++;
36         if (v[i] == elem)
37             return i; // Elemento encontrado
38     }
39     return -1; // Elemento encontrado
40 }
41
42 int it_buscaBinaria(int *v, int ini, int fim, int elem, int *comp) {
43     int meio;
44     while (ini <= fim) {
45         meio = (ini + fim) / 2;
46         (*comp)++;
47         if (elem == v[meio])
48             return meio;
49         else if (elem < v[meio])
50             fim = meio - 1;
51         else
52             ini = meio + 1;
53     }
54     return -1;
55 }
56
57 int compara_crescente(const void *a, const void *b) {
58     int x = *(int *)a;
59     int y = *(int *)b;
60
61     if (x < y)
62         return -1;
63     if (x > y)
64         return 1;
65
66     return 0;

```

```

67 | }
68 |
69 | enum algoritmos_para_teste {
70 |     PESQUISA_SEQUENCIAL = 0,
71 |     PESQUISA_BINARIA,
72 | };
73 |
74 | #define QTD_ALGORITMOS_TESTE 2
75 |
76 | int main() {
77 |     int n;
78 |
79 |     printf("Digite o tamanho do vetor:");
80 |     scanf("%d", &n);
81 |
82 |     int *array_original = (int *)malloc(n * sizeof(int));
83 |
84 |     for (int i = 0; i < n; i++) {
85 |         scanf("%d", &array_original[i]);
86 |     }
87 |
88 |     qsort(array_original, n, sizeof(int), compara_crescente);
89 |
90 |     int value;
91 |     printf("Digite um valor para buscar no vetor:\n");
92 |     scanf("%d", &value);
93 |
94 |     printf("Vetor final da busca:\n");
95 |     imprimeVetor(array_original, n);
96 |
97 |     for (int i = 0; i < QTD_ALGORITMOS_TESTE; i++) {
98 |         int comp = 0, index;
99 |
100 |         Temporizador tempo_teste;
101 |         iniciarTemporizador(&tempo_teste);
102 |
103 |         printf("Elemento buscado = %d\n", value);
104 |
105 |         switch (i) {
106 |             case PESQUISA_SEQUENCIAL:
107 |
108 |                 printf("----- Busca Sequencial ----- \n");
109 |
110 |                 index = buscaSequencial(array_original, n, value, &comp);
111 |
112 |                 if (index != -1) {
113 |                     printf("Elemento encontrado na indice %d\n", index);
114 |                 } else {
115 |                     printf("Elemento não encontrado\n");
116 |                 }
117 |
118 |                 break;
119 |             case PESQUISA_BINARIA:
120 |                 printf("----- Busca binária ----- \n");
121 |
122 |                 index = it_buscaBinaria(array_original, 0, n - 1, value, &comp);
123 |
124 |                 if (index != -1) {
125 |                     printf("Elemento encontrado na posição %d\n", index);
126 |                 } else {
127 |                     printf("Elemento não encontrado\n");
128 |                 }
129 |
130 |                 break;
131 |         }
132 |
133 |         finalizarTemporizador(&tempo_teste);
134 |         printf("Quantidade de comparações realizadas: %d\n", comp);
135 |         imprimirTempos(&tempo_teste);
136 |         printf("----- \n");
137 |     }
138 |
139 |     liberar_vetor(&array_original);
140 |     return 0;
141 |
142 |     return 0;
143 | }

```

Saída do terminal:

```

C:\arthurdetomi at arthurdetomi-System-Product-Name in ~/Documents/UFSJ-Graduacao/UFSJ-2025_1/Lab_Prog_2/roteiro_12
.: ./1-1.out < input/1000-misturado.txt
Digite o tamanho do vetor: Digite um valor para buscar no vetor:
Vetor final da busca:
[1, 2, 3, 4, 4, 8, 11, 12, 13, 13, 15, 16, 19, 20, 20, 24, 26, 27, 28, 33, 34, 36, 36, 37, 37, 39, 43, 44,
92, 93, 99, 102, 103, 104, 106, 109, 112, 115, 116, 116, 119, 122, 130, 130, 135, 137, 140, 147, 147, 148,
9, 192, 192, 195, 196, 198, 200, 200, 201, 204, 209, 210, 211, 211, 214, 217, 219, 219, 220, 222, 223, 224,
5, 258, 260, 261, 265, 266, 268, 269, 270, 273, 275, 276, 276, 280, 280, 282, 286, 295, 295, 298, 299, 305,
8, 351, 353, 354, 357, 358, 359, 363, 363, 364, 365, 366, 370, 370, 373, 374, 379, 382, 382, 386, 387, 387,
6, 417, 418, 418, 424, 425, 431, 432, 433, 435, 438, 443, 448, 449, 450, 451, 453, 454, 456, 456, 460, 463,
3, 484, 487, 487, 493, 494, 497, 498, 499, 499, 503, 509, 510, 513, 517, 520, 523, 525, 526, 526, 530, 534,
4, 584, 585, 589, 590, 590, 591, 592, 593, 594, 594, 595, 596, 598, 598, 598, 600, 604, 606, 608, 609, 610,
6, 667, 669, 670, 671, 672, 672, 673, 674, 679, 681, 681, 689, 690, 691, 693, 694, 695, 696, 700, 700, 700,
0, 734, 734, 738, 740, 741, 746, 747, 750, 752, 753, 755, 755, 768, 770, 772, 773, 775, 776, 777, 777, 778,
6, 827, 828, 830, 830, 836, 837, 838, 842, 846, 847, 851, 852, 853, 853, 855, 855, 857, 861, 862, 864, 865,
5, 897, 902, 902, 903, 905, 906, 907, 907, 909, 909, 910, 912, 915, 915, 916, 918, 919, 919, 919, 931, 932,
8, 962, 962, 963, 963, 964, 964, 967, 968, 968, 975, 976, 980, 984, 985, 990, 990, 991, 992, 992, 992, 994,
1032, 1034, 1035, 1039, 1040, 1041, 1051, 1052, 1055, 1058, 1058, 1059, 1059, 1060, 1061, 1062, 1063, 1066
8, 1102, 1103, 1106, 1107, 1110, 1110, 1112, 1113, 1114, 1116, 1120, 1121, 1122, 1128, 1131, 1131, 1131, 11
165, 1170, 1171, 1172, 1175, 1177, 1179, 1179, 1179, 1180, 1181, 1181, 1182, 1182, 1182, 1183, 1185, 1188,
1215, 1215, 1216, 1219, 1220, 1222, 1222, 1222, 1225, 1226, 1226, 1226, 1226, 1230, 1231, 1234, 1236, 1236
3, 1274, 1275, 1276, 1281, 1281, 1288, 1291, 1292, 1294, 1296, 1300, 1301, 1303, 1307, 1309, 1311, 1317, 13
340, 1341, 1341, 1345, 1345, 1347, 1353, 1358, 1365, 1366, 1368, 1369, 1369, 1372, 1372, 1374, 1375, 1382,
1416, 1417, 1422, 1423, 1427, 1428, 1428, 1430, 1430, 1430, 1432, 1434, 1435, 1436, 1437, 1443, 1445, 1447
5, 1480, 1481, 1481, 1482, 1483, 1485, 1486, 1491, 1494, 1496, 1497, 1504, 1504, 1507, 1507, 1507, 1508, 15
524, 1524, 1527, 1528, 1534, 1536, 1538, 1539, 1539, 1541, 1546, 1547, 1549, 1558, 1558, 1560, 1561, 1561,
1585, 1586, 1588, 1589, 1589, 1590, 1599, 1601, 1601, 1607, 1609, 1610, 1611, 1615, 1619, 1622, 1623, 1625
3, 1664, 1669, 1672, 1672, 1673, 1673, 1673, 1675, 1676, 1679, 1682, 1684, 1689, 1689, 1689, 1691, 1692, 16
709, 1715, 1720, 1723, 1726, 1727, 1729, 1732, 1737, 1740, 1740, 1744, 1745, 1748, 1749, 1754, 1756, 1760,
1787, 1791, 1795, 1797, 1798, 1803, 1803, 1804, 1805, 1806, 1806, 1807, 1808, 1814, 1815, 1816, 1819, 1829
0, 1853, 1857, 1861, 1862, 1871, 1873, 1874, 1874, 1875, 1876, 1877, 1879, 1880, 1881, 1883, 1886, 1887, 18
917, 1919, 1921, 1922, 1923, 1928, 1932, 1933, 1934, 1937, 1942, 1943, 1944, 1944, 1945, 1945, 1947, 1947,
1968, 1974, 1974, 1975, 1976, 1976, 1976, 1977, 1979, 1980, 1981, 1983, 1984, 1985, 1985, 1986, 1989, 1993
Elemento buscado = 1997
----- Busca Sequencial -----
Elemento encontrado na indice 997
Quantidade de comparações realizadas: 998
Tempo de execução:
Tempo Real: 0.00001200 segundos
Tempo Sistema: 0.00001200 segundos
-----
Elemento buscado = 1997
----- Busca binária -----
Elemento encontrado na posição 998
Quantidade de comparações realizadas: 9
Tempo de execução:
Tempo Real: 0.00000800 segundos
Tempo Sistema: 0.00000800 segundos
-----

```

1-2) Em relação ao algoritmo de busca binária, quais alterações devem ser realizadas para que a busca seja feita em um vetor ordenado de forma decrescente? Aplique as mudanças nas versões recursiva e iterativa do método e teste o seu código.

roteiro_12/1-2.c

```

1  #include "tempo.h"
2
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  void liberar_vetor(int **v) {
7      free(*v);
8      *v = NULL;
9  }
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++)
16         v2[i] = v[i];
17     return v2;
18 }
19
20 void imprimeVetor(int *v, int n) {
21     int i, prim = 1;
22     printf("[");
23     for (i = 0; i < n; i++)
24         if (prim) {
25             printf("%d", v[i]);
26             prim = 0;
27         } else

```

```
28     printf(", %d", v[i]);
29     printf("]\n");
30 }
31
32 int buscaSequencial(int *v, int n, int elem, int *comp) {
33     int i;
34     for (i = 0; i < n; i++) {
35         (*comp)++;
36         if (v[i] == elem)
37             return i; // Elemento encontrado
38     }
39     return -1; // Elemento encontrado
40 }
41
42 int it_buscaBinaria(int *v, int ini, int fim, int elem, int *comp) {
43     int meio;
44     while (ini <= fim) {
45         meio = (ini + fim) / 2;
46         (*comp)++;
47         if (elem == v[meio])
48             return meio;
49         else if (elem < v[meio])
50             ini = meio + 1;
51         else
52             fim = meio - 1;
53     }
54     return -1;
55 }
56
57 int rec_buscaBinaria(int *v, int ini, int fim, int elem, int *comp) {
58     if (ini > fim)
59         return -1;
60     int meio = (ini + fim) / 2;
61     (*comp)++;
62     if (v[meio] == elem)
63         return meio;
64     else if (elem < v[meio])
65         return rec_buscaBinaria(v, meio + 1, fim, elem, comp);
66     else
67         return rec_buscaBinaria(v, ini, meio - 1, elem, comp);
68 }
69
70 int compara_decrescente(const void *a, const void *b) {
71     int x = *(int *)a;
72     int y = *(int *)b;
73
74     if (x < y)
75         return 1;
76     if (x > y)
77         return -1;
78
79     return 0;
80 }
81
82 enum algoritmos_para_teste {
83     PESQUISA_SEQUENCIAL = 0,
84     PESQUISA_BINARIA,
85 };
86
87 #define QTD_ALGORITMOS_TESTE 2
88
89 int main() {
90     int n;
91
92     printf("Digite o tamanho do vetor:");
93     scanf("%d", &n);
94
95     int *array_original = (int *)malloc(n * sizeof(int));
96
97     for (int i = 0; i < n; i++) {
98         scanf("%d", &array_original[i]);
99     }
100
101     qsort(array_original, n, sizeof(int), compara_decrescente);
102
103     int value;
104     printf("Digite um valor para buscar no vetor:\n");
105     scanf("%d", &value);
106
107     printf("Vetor final da busca:\n");
108     imprimeVetor(array_original, n);
```

```

109
110 for (int i = 0; i < QTD_ALGORITMOS_TESTE; i++) {
111     int comp = 0, index;
112
113     Temporizador tempo_teste;
114     iniciarTemporizador(&tempo_teste);
115
116     printf("Elemento buscado = %d\n", value);
117
118     switch (i) {
119     case PESQUISA_SEQUENCIAL:
120
121         printf("----- Busca Sequencial -----\\n");
122
123         index = buscaSequencial(array_original, n, value, &comp);
124
125         if (index != -1) {
126             printf("Elemento encontrado no indice %d\\n", index);
127         } else {
128             printf("Elemento não encontrado\\n");
129         }
130
131         break;
132     case PESQUISA_BINARIA:
133         printf("----- Busca binária -----\\n");
134
135         index = it_buscaBinaria(array_original, 0, n - 1, value, &comp);
136
137         if (index != -1) {
138             printf("Elemento encontrado no indice %d\\n", index);
139         } else {
140             printf("Elemento não encontrado\\n");
141         }
142
143         break;
144     }
145
146     finalizarTemporizador(&tempo_teste);
147     printf("Quantidade de comparações realizadas: %d\\n", comp);
148     imprimirTempos(&tempo_teste);
149     printf("-----\\n");
150 }
151
152 liberar_vetor(&array_original);
153 return 0;
154
155 return 0;
156 }

```

Saída do terminal:

```

└─ arthurdetomi at arthurdetomi-System-Product-Name in ~/Documents/UFSJ-Graduacao/UFSJ-2025_1/Lab_Prog_2/roteiro_12
└─ ./1-2.out < input/100-misturado.txt
Digite o tamanho do vetor: Digite um valor para buscar no vetor:
Vetor final da busca:
[200, 200, 197, 195, 193, 193, 192, 189, 185, 184, 180, 178, 178, 175, 173, 172, 167, 167, 167, 165, 164, 163, 162, 161, 160, 159, 158, 157, 156, 155, 154, 153, 152, 151, 150, 149, 148, 147, 146, 145, 144, 143, 142, 141, 140, 139, 138, 137, 136, 135, 134, 133, 132, 131, 130, 129, 128, 127, 126, 125, 124, 123, 122, 121, 117, 116, 115, 115, 113, 113, 106, 105, 100, 97, 96, 95, 94, 89, 88, 85, 84, 84, 83, 82, 81, 80, 79, 78, 77, 76, 75, 74, 73, 72, 71, 70, 69, 68, 67, 66, 65, 64, 63, 62, 61, 60, 59, 58, 57, 56, 55, 54, 53, 52, 51, 50, 49, 48, 47, 46, 45, 44, 43, 42, 41, 40, 39, 38, 37, 36, 35, 34, 33, 32, 31, 30, 29, 29, 28, 27, 25, 25, 21, 18, 17, 14, 12, 12, 10, 9, 4, 4, 3, 2]
Elemento buscado = 82
----- Busca Sequencial -----
Elemento encontrado no indice 61
Quantidade de comparações realizadas: 62
Tempo de execução:
Tempo Real: 0.00000800 segundos
Tempo Sistema: 0.00000800 segundos
-----
Elemento buscado = 82
----- Busca binária -----
Elemento encontrado no indice 61
Quantidade de comparações realizadas: 3
Tempo de execução:
Tempo Real: 0.00000600 segundos
Tempo Sistema: 0.00000600 segundos
-----

```

1-3) Utilizando o código feito na aula prática para a ordenação de um vetor de struct, adapte esse código para que a struct seja de um aluno com os campos....

roteiro_12/1-3.c

```

1 | #include <stdio.h>
2 | #include <stdlib.h>

```

```
3 | #include <string.h>
4 |
5 | typedef struct {
6 |     char nome[50];
7 |     int matricula;
8 | } Aluno;
9 |
10 | void liberar_vetor(Aluno **v) {
11 |     free(*v);
12 |     *v = NULL;
13 | }
14 |
15 | int *copiaVetor(int *v, int n) {
16 |     int i;
17 |     int *v2;
18 |     v2 = (int *)malloc(n * sizeof(int));
19 |     for (i = 0; i < n; i++)
20 |         v2[i] = v[i];
21 |     return v2;
22 | }
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) {
29 |             printf("%d", v[i]);
30 |             prim = 0;
31 |         } else
32 |             printf(", %d", v[i]);
33 |     printf("]\n");
34 | }
35 |
36 | int comp;
37 |
38 | void troca(Aluno *a, Aluno *b) {
39 |     Aluno aux = *a;
40 |     *a = *b;
41 |     *b = aux;
42 | }
43 |
44 | int particao(Aluno *v, int ini, int fim, int (*compara)(Aluno *, Aluno *)) {
45 |     int i = ini, j = fim;
46 |     Aluno pivo = v[(ini + fim) / 2];
47 |     while (1) {
48 |         while (compara(&(v[i]), &pivo) < 0) {
49 |             i++;
50 |         }
51 |         while (compara(&(v[j]), &pivo) > 0) {
52 |             j--;
53 |         }
54 |
55 |         if (i < j) {
56 |             troca(&v[i], &v[j]);
57 |             i++;
58 |             j--;
59 |         } else
60 |             return j;
61 |     }
62 | }
63 |
64 | void QuickSort(Aluno *v, int ini, int fim, int (*compara)(Aluno *, Aluno *)) {
65 |     if (ini < fim) {
66 |         int q = particao(v, ini, fim, compara);
67 |         QuickSort(v, ini, q, compara);
68 |         QuickSort(v, q + 1, fim, compara);
69 |     }
70 | }
71 |
72 | int rec_buscaBinaria(Aluno *v, int ini, int fim, Aluno elem,
73 |                     int (*compara)(Aluno *, Aluno *)) {
74 |     if (ini > fim)
75 |         return -1;
76 |     int meio = (ini + fim) / 2;
77 |     comp++;
78 |     if (compara(&(v[meio]), &elem) == 0)
79 |         return meio;
80 |     else if (compara(&(v[meio]), &elem) > 0)
81 |         return rec_buscaBinaria(v, ini, meio - 1, elem, compara);
82 |     else
83 |         return rec_buscaBinaria(v, meio + 1, fim, elem, compara);
```

```
84 }
85
86 int comparaNome(Aluno *a, Aluno *b) { return strcmp(a->nome, b->nome); }
87
88 int comparaMatricula(Aluno *a, Aluno *b) { return a->matricula - b->matricula; }
89
90 void print_aluno(const Aluno *aluno) {
91     printf("-----\n");
92     printf("Aluno Info:\n");
93     printf("Nome: %s\n", aluno->nome);
94     printf("Matricula: %d\n", aluno->matricula);
95     printf("-----\n");
96 }
97
98 int main() {
99     int n;
100
101     printf("Digite a quantidade de alunos:\n");
102     scanf("%d", &n);
103
104     Aluno *alunos = malloc(n * sizeof(Aluno));
105
106     printf("Cadastro de alunos:\n");
107     for (int i = 0; i < n; i++) {
108         printf("Nome:\n");
109         scanf("%s", alunos[i].nome);
110         printf("Matricula:\n");
111         scanf("%d", &alunos[i].matricula);
112     }
113
114     Aluno al_busca;
115     printf("Digite um nome de um aluno para busca:\n");
116     scanf("%s", al_busca.nome);
117
118     printf("Nome de aluno buscado : %d\n", al_busca.matricula);
119
120     QuickSort(alunos, 0, n - 1, comparaNome);
121
122     printf("Lista de alunos cadastrados:\n");
123     for (int i = 0; i < n; i++) {
124         printf("%s, %d", alunos[i].nome, alunos[i].matricula);
125     }
126     printf("\n");
127
128     int index = rec_buscaBinaria(alunos, 0, n - 1, al_busca, comparaNome);
129
130     if (index != -1) {
131         printf("----- Sucesso ! ----- \n");
132         printf("Aluno encontrado\n");
133         print_aluno(&alunos[index]);
134         printf("-----\n");
135     } else {
136         printf("----- Not Found ----- \n");
137         printf("Aluno não encontrado\n");
138         printf("-----\n");
139     }
140
141     printf("Digite uma matricula para busca:\n");
142     scanf("%d", &al_busca.matricula);
143
144     QuickSort(alunos, 0, n - 1, comparaMatricula);
145
146     printf("Matricula buscada : %d\n", al_busca.matricula);
147
148     index = rec_buscaBinaria(alunos, 0, n - 1, al_busca, comparaMatricula);
149
150     if (index != -1) {
151         printf("----- Sucesso ! ----- \n");
152         printf("Aluno encontrado\n");
153         print_aluno(&alunos[index]);
154         printf("-----\n");
155     } else {
156         printf("----- Not Found ----- \n");
157         printf("Aluno não encontrado\n");
158         printf("-----\n");
159     }
160
161     liberar_vetor(&alunos);
162
163     return 0;
164 }
```


165 | }

Saída do terminal:

```
[arthurdetomi at arthurdetomi-System-Product-Name] in ~/Documents/UFSJ-Graduacao/UFSJ-2025_1/Lab_Prog_2/roteiro_12 on mainxxx 25-06-28  
./1-3.out < in  
Digite a quantidade de alunos:  
Cadastro de alunos:  
Nome:  
Matricula:  
Nome:  
Matricula:  
Nome:  
Matricula:  
Nome:  
Matricula:  
Nome:  
Matricula:  
Nome:  
Matricula:  
Nome:  
Matricula:  
Nome:  
Matricula:  
Nome:  
Matricula:  
Nome:  
Matricula:  
Digite um nome de um aluno para busca:  
Nome de aluno buscado : Lara  
Lista de alunos cadastrados:  
{Ana, 1002}, {Beatriz, 1004}, {Bruna, 1010}, {Carlos, 1003}, {Daniel, 1005}, {Felipe, 1007}, {Joao, 1001}, {Lara, 1008}, {Mariana, 1009}  
----- Sucesso ! -----  
Aluno encontrado  
-----  
Aluno Info:  
Nome: Lara  
Matricula: 1008  
-----  
-----  
Digite uma matricula para busca:  
Matricula buscada : 1003  
----- Sucesso ! -----  
Aluno encontrado  
-----  
Aluno Info:  
Nome: Carlos  
Matricula: 1003  
-----  
-----
```

2-1) Reimplemente o TAD: Tabela Hash

roteiro_12/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  enum opcoes_func_sondagem { LINEAR, QUADRATICA, DUPLO };
9
10 #define QTD_SONDAGENS 3
11
12 typedef struct {
13     int **tabela;
14     int tam, qtd;
15 } Hash;
16
17 Hash *criaHash(int t) {
18     Hash *h;
19     h = (Hash *)malloc(sizeof(Hash));
20     if (h != NULL) {
21         h->tam = t;
22         h->qtd = 0;
23         h->tabela = (int **)malloc(t * sizeof(int *));
24         if (h->tabela == NULL)
25             return NULL;
26         int i;
27         for (i = 0; i < t; i++)
28             h->tabela[i] = NULL;
29     }
30     return h;
31 }
32

```

```
33 void destroiHash(Hash *h) {
34     if (h != NULL) {
35         int i;
36         for (i = 0; i < h->tam; i++)
37             if (h->tabela[i] != NULL)
38                 free(h->tabela[i]);
39         free(h->tabela);
40         free(h);
41     }
42 }
43
44 int chaveDivisao(int chave, int tam) { return (chave & 0x7FFFFFFF) % tam; }
45
46 int chaveMultiplicacao(int chave, int tam) {
47     float A = 0.6180339887; // constante: 0 < A < 1
48     float val = chave * A;
49     val = val - (int)val;
50     return (int)(tam * val);
51 }
52
53 int chaveDobra(int chave, int tam) {
54     int pos, n_bits = 30;
55
56     int p = 1;
57     int r = p << n_bits;
58     while ((chave & r) != r) {
59         n_bits--;
60         r = p << n_bits;
61     }
62
63     n_bits++;
64     pos = chave;
65     while (pos > tam) {
66         int metade_bits = n_bits / 2;
67         int partel = pos >> metade_bits;
68         partel = partel << metade_bits;
69         int parte2 = pos ^ partel;
70         partel = pos >> metade_bits;
71         pos = partel ^ 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)str[i]);
82     return valor;
83 }
84
85 int insereHash_semTratar(Hash *h, int elem) {
86     if (h == NULL)
87         return 0;
88     int pos = chaveDivisao(elem, h->tam);
89
90     if (h->tabela[pos] == NULL) {
91         int *novo = (int *)malloc(sizeof(int));
92         if (novo == NULL)
93             return 0;
94         *novo = elem;
95         h->tabela[pos] = novo;
96         h->qtd++;
97     } else
98         *(h->tabela[pos]) = elem;
99     return 1;
100 }
101
102 int buscaHash_semTratar(Hash *h, int elem, int *p) {
103     if (h == NULL)
104         return 0;
105     int pos = chaveDivisao(elem, h->tam);
106     if (h->tabela[pos] == NULL)
107         return 0;
108     if (*(h->tabela[pos]) == elem) {
109         *p = *(h->tabela[pos]);
110         return 1;
111     }
112     return 0;
113 }
```

```

114
115 int sondagemLinear(int pos, int i, int tam) {
116     return ((pos + i) & 0x7FFFFFFF) % tam;
117 }
118
119 int sondagemQuadratica(int pos, int i, int tam) {
120     pos = pos + 2 * i + 5 * i * i;
121     return (pos & 0x7FFFFFFF) % tam;
122 }
123
124 int sondagemDuploHash(int H1, int chave, int i, int tam) {
125     int H2 = chaveDivisao(chave, tam - 1) + 1;
126     return ((H1 + i * H2) & 0x7FFFFFFF) % tam;
127 }
128
129 int insereHash_EnderAberto(Hash *h, int elem, int (*func_chave)(int, int),
130                             enum opcoes_func_sondagem op) {
131     if (h == NULL)
132         return 0;
133     int i, pos, newPos;
134     pos = func_chave(elem, h->tam);
135     for (i = 0; i < h->tam; i++) {
136         switch (op) {
137             case LINEAR:
138                 newPos = sondagemLinear(pos, i, h->tam);
139                 break;
140             case QUADRATICA:
141                 newPos = sondagemQuadratica(pos, i, h->tam);
142                 break;
143             case DUPL0:
144                 newPos = sondagemDuploHash(pos, elem, i, h->tam);
145                 break;
146         }
147
148         if (h->tabela[newPos] == NULL) {
149             int *novo = (int *)malloc(sizeof(int));
150             if (novo == NULL)
151                 return 0;
152             *novo = elem;
153             h->tabela[newPos] = novo;
154             h->qtd++;
155             return 1;
156         }
157     }
158     return 0;
159 }
160
161 int buscaHash_EnderAberto(Hash *h, int elem, int *p) {
162     if (h == NULL)
163         return 0;
164     int i, pos, newPos;
165     pos = chaveDivisao(elem, h->tam);
166     for (i = 0; i < h->tam; i++) {
167         newPos = sondagemLinear(pos, i, h->tam);
168         // newPos = sondagemQuadratica(pos, i, h->tam);
169         // newPos = sondagemDuploHash(pos, elem, i, h->tam);
170         if (h->tabela[newPos] == NULL)
171             return 0;
172         if (*(h->tabela[newPos]) == elem) {
173             *p = *(h->tabela[newPos]);
174             return 1;
175         }
176     }
177     return 0;
178 }
179
180 void imprimeHash(Hash *h) {
181     if (h == NULL)
182         return;
183     int i;
184     for (i = 0; i < h->tam; i++) {
185         printf("%d: ", i);
186         if (h->tabela[i] == NULL)
187             printf("NULL\n");
188         else
189             printf("%d\n", *(h->tabela[i]));
190     }
191 }
192
193 #endif

```

roteiro_12/2-1.c

```
1 #include "Hash.h"
2
3 #include <math.h>
4
5 void liberar_vetor(int **v) {
6     free(*v);
7     *v = NULL;
8 }
9
10 int *copiaVetor(int *v, int n) {
11     int i;
12     int *v2;
13     v2 = (int *)malloc(n * sizeof(int));
14     for (i = 0; i < n; i++)
15         v2[i] = v[i];
16     return v2;
17 }
18
19 void imprimeVetor(int *v, int n) {
20     int i, prim = 1;
21     printf("[");
22     for (i = 0; i < n; i++)
23         if (prim) {
24             printf("%d", v[i]);
25             prim = 0;
26         } else
27             printf(", %d", v[i]);
28     printf("]\n");
29 }
30
31 enum opcoes_func_hash { DIVISAO = 0, MULT, DOBRA };
32
33 #define QTD_OPCOES 3
34
35 int main() {
36     int n;
37
38     printf("Digite o tamanho do vetor:");
39     scanf("%d", &n);
40
41     int *array = (int *)malloc(n * sizeof(int));
42
43     for (int i = 0; i < n; i++) {
44         scanf("%d", &array[i]);
45     }
46
47     printf("-----\n");
48     imprimeVetor(array, n);
49     printf("-----\n");
50
51     int (*funcoes_chave[])(int, int) = {&chaveDivisao, &chaveMultiplicacao,
52                                         &chaveDobra};
53     char *funcoes_nome[] = {"Divisão", "Multiplicação", "Dobra"};
54
55     for (int fn_chave_idx = 0; fn_chave_idx < QTD_OPCOES; fn_chave_idx++) {
56         const int INC = 3;
57
58         Hash *hash_table = criaHash(n + INC);
59
60         for (int j = 0; j < n; j++) {
61             insereHash_EnderAberto(hash_table, array[j], funcoes_chave[fn_chave_idx],
62                                   LINEAR);
63         }
64
65         printf("-----\n");
66         printf("Função hash aplicada : %s\n", funcoes_nome[fn_chave_idx]);
67         imprimeHash(hash_table);
68         printf("-----\n");
69
70         destroiHash(hash_table);
71     }
72
73     liberar_vetor(&array);
74     return 0;
75 }
```

Saída do terminal:

```

[arthurdetomi at arthurdetomi-System-Product-Name in ~/Document
.: ./2-1.out < input/10-misturado.txt
Digite o tamanho do vetor:-----
[47, 132, 88, 191, 5, 73, 64, 103, 39, 26]
-----
Função hash aplicada : Divisão
0: 103
1: 39
2: 132
3: 26
4: NULL
5: 5
6: NULL
7: NULL
8: 47
9: 191
10: 88
11: 73
12: 64
-----
Função hash aplicada : Multiplicação
0: 47
1: 191
2: 5
3: 73
4: 39
5: 88
6: 26
7: 132
8: 64
9: 103
10: NULL
11: NULL
12: NULL
-----
Função hash aplicada : Dobra
0: 103
1: NULL
2: 47
3: 39
4: 191
5: 5
6: 26
7: NULL
8: 73
9: 64
10: NULL
11: 88
12: 132
-----

```

2-2) Dessa vez, altere o TAD anterior...

roteiro_12/2-2.c

```

1  #include "Hash.h"
2
3  #include <math.h>
4
5  void liberar_vetor(int **v) {
6      free(*v);
7      *v = NULL;
8  }
9
10 int *copiaVetor(int *v, int n) {
11     int i;
12     int *v2;
13     v2 = (int *)malloc(n * sizeof(int));
14     for (i = 0; i < n; i++)
15         v2[i] = v[i];
16     return v2;
17 }
18
19 void imprimeVetor(int *v, int n) {
20     int i, prim = 1;
21     printf("[");
22     for (i = 0; i < n; i++)
23         if (prim) {
24             printf("%d", v[i]);
25             prim = 0;
26         } else
27             printf(", %d", v[i]);
28     printf("]\n");
29 }

```

```
30
31 enum opcoes_func_hash { DIVISAO = 0, MULT, DOBRA };
32
33 int main() {
34     int n;
35
36     printf("Digite o tamanho do vetor:");
37     scanf("%d", &n);
38
39     int *array = (int *)malloc(n * sizeof(int));
40
41     for (int i = 0; i < n; i++) {
42         scanf("%d", &array[i]);
43     }
44
45     printf("-----\n");
46     imprimeVetor(array, n);
47     printf("-----\n");
48
49     int (*funcoes_chave[])(int, int) = {&chaveDivisao, &chaveMultiplicacao,
50                                         &chaveDobra};
51     char *funcoes_chave_nome[] = {"Divisão", "Multiplicação", "Dobra"};
52     char *funcoes_sondagem_nomes[] = {"Sondagem Linear", "Sondagem Quadratica",
53                                       "Sondagem Duplo Hash"};
54
55     for (int fn_sd_idx = 0; fn_sd_idx < QTD_SONDAGENS; fn_sd_idx++) {
56         const int INC = 3;
57
58         Hash *hash_table = criaHash(n + INC);
59
60         for (int j = 0; j < n; j++) {
61             insereHash_EnderAberto(hash_table, array[j], funcoes_chave[DIVISAO],
62                                   fn_sd_idx);
63         }
64
65         printf("-----\n");
66         printf("Função hash aplicada : %s\n", funcoes_chave_nome[DIVISAO]);
67         printf("Função de sondagem aplicada: %s\n",
68               funcoes_sondagem_nomes[fn_sd_idx]);
69         imprimeHash(hash_table);
70         printf("-----\n");
71
72         destroiHash(hash_table);
73     }
74
75     liberar_vetor(&array);
76     return 0;
77 }
```

Saída do terminal:

```

└─.: ./2-2.out < input/10-misturado.txt
Digite o tamanho do vetor:-----
[47, 132, 88, 191, 5, 73, 64, 103, 39, 26]
-----
Função hash aplicada : Divisão
Função de sondagem aplicada: Sondagem Linear
0: 103
1: 39
2: 132
3: 26
4: NULL
5: 5
6: NULL
7: NULL
8: 47
9: 191
10: 88
11: 73
12: 64
-----
Função hash aplicada : Divisão
Função de sondagem aplicada: Sondagem Quadratica
0: 39
1: NULL
2: 132
3: NULL
4: NULL
5: 5
6: 73
7: 26
8: 47
9: 191
10: 88
11: 103
12: 64
-----
Função hash aplicada : Divisão
Função de sondagem aplicada: Sondagem Duplo Hash
0: 39
1: NULL
2: 132
3: 26
4: 64
5: 5
6: NULL
7: 103
8: 47
9: 191
10: 88
11: NULL
12: 73
-----

```

2-3) TAD: Tabela Hash com encadeamento...

roteiro_12/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)
25 |         return 1;
26 |     if (*li == NULL)
27 |         return 1; // True - Vazia!

```

```
28     return 0;    // False - tem elemento!
29 }
30
31 NO *alocarNO() { return (NO *)malloc(sizeof(NO)); }
32
33 void liberarNO(NO *q) { free(q); }
34
35 int listaBuscaElem(Lista *li, int elem, int *p) {
36     if (li == NULL)
37         return 0;
38     NO *aux = *li;
39     while (aux != NULL) {
40         if (aux->info == elem) {
41             *p = aux->info;
42             return 1;
43         }
44         aux = aux->prox;
45     }
46     return 0;
47 }
48
49 int insereIni(Lista *li, int elem) {
50     if (li == NULL)
51         return 0;
52     NO *novo = alocarNO();
53     if (novo == NULL)
54         return 0;
55     novo->info = elem;
56     novo->prox = *li;
57     *li = novo;
58     return 1;
59 }
60
61 int insereFim(Lista *li, int elem) {
62     if (li == NULL)
63         return 0;
64     NO *novo = alocarNO();
65     if (novo == NULL)
66         return 0;
67     novo->info = elem;
68     novo->prox = NULL;
69     if (listaVazia(li)) {
70         *li = novo;
71     } else {
72         NO *aux = *li;
73         while (aux->prox != NULL)
74             aux = aux->prox;
75         aux->prox = novo;
76     }
77     return 1;
78 }
79
80 int removeIni(Lista *li) {
81     if (li == NULL)
82         return 0;
83     if (listaVazia(li))
84         return 0;
85     NO *aux = *li;
86     *li = aux->prox;
87     liberarNO(aux);
88     return 1;
89 }
90
91 int removeFim(Lista *li) {
92     if (li == NULL)
93         return 0;
94     if (listaVazia(li))
95         return 0;
96     NO *ant, *aux = *li;
97     while (aux->prox != NULL) {
98         ant = aux;
99         aux = aux->prox;
100     }
101     if (aux == *li)
102         *li = aux->prox;
103     else
104         ant->prox = aux->prox;
105     liberarNO(aux);
106     return 1;
107 }
108 }
```



```

109 void imprimeLista(Lista *li) {
110     if (li == NULL)
111         return;
112     if (listaVazia(li)) {
113         printf("Lista Vazia!\n");
114         return;
115     }
116     // printf("Elementos:\n");
117     NO *aux = *li;
118     while (aux != NULL) {
119         printf("%d ", aux->info);
120         aux = aux->prox;
121     }
122     printf("\n");
123 }
124
125 void destroiLista(Lista *li) {
126     if (li != NULL) {
127         NO *aux;
128         while ((*li) != NULL) {
129             aux = *li;
130             *li = (*li)->prox;
131             liberarNO(aux);
132         }
133         free(li);
134     }
135 }
136
137 #endif
138

```

roteiro_12/HashLSE.h

```

1  #ifndef HASH_H
2  #define HASH_H
3
4  #include "LSE.h"
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <string.h>
8
9  typedef struct {
10     Lista **tabela;
11     int tam, qtd;
12 } Hash;
13
14 Hash *criaHash(int t) {
15     Hash *h;
16     h = (Hash *)malloc(sizeof(Hash));
17     if (h != NULL) {
18         h->tam = t;
19         h->qtd = 0;
20         h->tabela = (Lista **)malloc(t * sizeof(Lista *));
21         if (h->tabela == NULL)
22             return NULL;
23         int i;
24         for (i = 0; i < t; i++)
25             h->tabela[i] = NULL;
26     }
27     return h;
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) { return (chave & 0x7FFFFFFF) % tam; }
42
43 int chaveMultiplicacao(int chave, int tam) {
44     float A = 0.6180339887; // constante: 0 < A < 1
45     float val = chave * A;
46     val = val - (int)val;
47     return (int)(tam * val);
48 }
49

```

```

50 int chaveDobra(int chave, int tam) {
51     int pos, n_bits = 30;
52
53     int p = 1;
54     int r = p << n_bits;
55     while ((chave & r) != r) {
56         n_bits--;
57         r = p << n_bits;
58     }
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]);
79     return valor;
80 }
81
82 int insereHashLSE(Hash *h, int elem) {
83     if (h == NULL)
84         return 0;
85     int pos = chaveDivisao(elem, h->tam);
86     if (h->tabela[pos] == NULL)
87         h->tabela[pos] = criaLista();
88     insereIni(h->tabela[pos], elem);
89     h->qtd++;
90     return 1;
91 }
92
93 int buscaHashLSE(Hash *h, int elem, int *p) {
94     if (h == NULL)
95         return 0;
96     int pos = chaveDivisao(elem, h->tam);
97     if (h->tabela[pos] == NULL)
98         return 0;
99     return listaBuscaElem(h->tabela[pos], elem, p);
100 }
101
102 void imprimeHash(Hash *h) {
103     if (h == NULL)
104         return;
105     int i;
106     for (i = 0; i < h->tam; i++) {
107         printf("%d: ", i);
108         if (h->tabela[i] == NULL)
109             printf("NULL\n");
110         else
111             imprimeLista(h->tabela[i]);
112     }
113 }
114
115 #endif

```

roteiro_12/2-3.c

```

1 #include "HashLSE.h"
2
3 void liberar_vetor(int **v) {
4     free(*v);
5     *v = NULL;
6 }
7
8 int *copiaVetor(int *v, int n) {
9     int i;
10    int *v2;
11    v2 = (int *)malloc(n * sizeof(int));
12    for (i = 0; i < n; i++)
13        v2[i] = v[i];

```

```

14     return v2;
15 }
16
17 void imprimeVetor(int *v, int n) {
18     int i, prim = 1;
19     printf("[");
20     for (i = 0; i < n; i++)
21         if (prim) {
22             printf("%d", v[i]);
23             prim = 0;
24         } else
25             printf(", %d", v[i]);
26     printf("]\n");
27 }
28
29 int main() {
30     int n;
31
32     printf("Digite o tamanho do vetor:");
33     scanf("%d", &n);
34
35     int *array = (int *)malloc(n * sizeof(int));
36
37     for (int i = 0; i < n; i++) {
38         scanf("%d", &array[i]);
39     }
40
41     printf("-----\n");
42     imprimeVetor(array, n);
43     printf("-----\n");
44
45     Hash *hash_table = criaHash(n);
46     for (int i = 0; i < n; i++) {
47         insereHashLSE(hash_table, array[i]);
48     }
49
50     printf("----- Mostrando Tabela ----- \n");
51     imprimeHash(hash_table);
52     printf("-----\n");
53
54     liberar_vetor(&array);
55     return 0;
56 }

```

Saída do terminal:

```

└─ arthurdetomi at arthurdetomi-System-Product-Name in ~/Documents
└─ ./2-3.out < input/10-misturado.txt
Digite o tamanho do vetor:-----
[47, 132, 88, 191, 5, 73, 64, 103, 39, 26]
-----
----- Mostrando Tabela -----
0: NULL
1: 191
2: 132
3: 103 73
4: 64
5: 5
6: 26
7: 47
8: 88
9: 39
-----

```