Exercício : 1 _____ / _____ / _____

Busca em tabelas

Este exercício visa comparar diversas técnicas de acesso a uma tabela. Este problema é o escolhido por ser um dos problemas fundamentais em informática. Para cada uma das técnicas abaixo, você deve verificar quantas vezes o algoritmo mostrado será executado.

Busca Linear Para este algoritmo o conjunto de chaves não é mantido ordenado. O algoritmo básico é:

```
 1: função BUSCALIN(inteiro CHAVE, inteiro TABELA[num])
 2: inteiro i ← 1; qtd ← 0
 3: enquanto i ≤ tamanho(TABELA) faça
 4:     qtd ← qtd + 1
 5:     se CHAVE = TABELA[i] então
 6:         ...ACHOU...
 7:         i ← tamanho(TABELA)
 8:     fim se
 9:     qtd ← qtd + 1
10:     i++
11: fim enquanto
12: devolva qtd
```

```
37 BUSCALIN   62 45 61 78  1 94 30 28  4 31 33 92 37  2 23 e  26
98 BUSCALIN   70 98 81 46 29 12 65 40 72 14 20  4 82  2 30 e   4
 3 BUSCALIN   29 66 71 42 48 44 69 35 30 37 94 68 36 19 21 e  30
```

Busca linear ordenada Neste algoritmo, o conjunto de chaves deverá estar ordenado. Eis o algoritmo:

```
 1: função BUSCALINORD (inteiro CHAVE, inteiro TABELA[num])
 2: inteiro i ← 1; qtd ← 1
 3: enquanto (i ≤ tamanho(TABELA)) ∧ (CHAVE > TABELA[i]) faça
 4:     qtd ← qtd + 2
 5:     i++
 6: fim enquanto
 7: se CHAVE = TABELA[i] então
 8:     ...ACHOU...
 9: fim se
10: devolva qtd
```

```
96 BUSCALINORD 5 10 26 34 39 48 56 60 68 70 73 89 92 93 96 e  29
58 BUSCALINORD 1  2  8 10 12 41 42 47 49 53 58 59 60 62 85 e  21
96 BUSCALINORD 2 12 21 26 27 37 43 46 52 54 65 79 83 84 89 e  31
```

Busca linear com sentinela A tabela não precisa estar ordenada, e a busca gasta apenas a metade dos testes da busca linear. A sentinela é a chave que se busca que é forçosamente incluída ao final da tabela. Assim, a saída do laço sempre se dará pela igualdade de chaves. Se a chave procurada for encontrada no meio da tabela, ...ACHOU... Se for encontrada no final da tabela, (onde foi explicitamente colocada) é porque ela não se encontrava lá antes do início do algoritmo.

```
 1: função BUSCALINSEN (inteiro CHAVE, inteiro TABELA[num])
 2: inteiro i ← 1; qtd ← 1
 3: TABELA[ULTIMO+1] ← CHAVE {note que ULTIMO não é alterado}
 4: enquanto CHAVE ≠ TABELA[i]) faça
 5:     qtd++
 6:     i++
 7: fim enquanto
 8: se i ≠ (ULTIMO + 1) então
 9:     ...ACHOU...
10: fim se
11: devolva qtd
```

```
38 BUSCALINSEN 24 35 91 18  8 72 10 88 27 54 28 34 43  3 38 e 15
54 BUSCALINSEN 63 17 35 68 27 86 54 53  5 78 41 58 44 51 25 e  7
61 BUSCALINSEN 75 67 51 60 85 38 49 74 73 78 66 35 24  4  2 e 16
```

Busca através de tabela HASH O gasto em espaço é recompensado com ótimo desempenho. O objetivo de qualquer pesquisa em tabelas hash é garantir um custo de $O(1)$. Para que isto ocorra a probabilidade de colisões deve ser próxima a zero. Uma desvantagem importante da tabela HASH é que a mesma não pode ser "visitada"em ordem, já que os dados não são armazenados em ordem. Para que a busca possa ocorrer, a tabela precisa ser criada previamente usando uma função de criação como a mostrada no quadro, por exemplo: Com esta função de inclusão, a função de busca seria:

```
 1: função BUSCAHASH(inteiro CHAVE, inteiro TABELA[23])
 2: inteiro i ← 1 + CHAVE mod 23
 3: qtd ← 1
 4: enquanto (CHAVE ≠ TABELA[i]) ∧ (TABELA[i] ≠ 0) faça
 5:     qtd ← qtd+2
 6:     i++
 7:     se i > 23 então
 8:         i ← 1
 9:     fim se
10: fim enquanto
11: se TABELA[i] ≠ 0 então
12:     ...ACHOU...
13: fim se
14: devolva qtd
```

Note que os números que vão ser fornecidos à função de busca ainda não são uma tabela HASH. Esta precisa ser criada antes da primeira consulta.

```
 4 BUSCAHASH  18 78 73 90 44 93 81  4 57 65 34 82 37 29 20 e   3
97 BUSCAHASH  55 64 30  8 15 97 53 21 11 82 50 87 45 12 23 e   1
 8 BUSCAHASH  40 18 32 94 75  9 65 22 54 25 16 33 80 86 27 e  11
```

Busca binária Conjunto ordenado, ótimo desempenho

```
 1: função BUSCABIN( inteiro TABELA[15], inteiro CHAVE)
 2: inteiro INIC,METADE,FIM
 3: INIC ← 1 {campo delimita o limite inferior da pesquisa}
 4: qtd ← 1
 5: FIM ← 15 {delimita o limite superior da pesquisa}
 6: repita
 7:     METADE ← ⌊ ((INIC + FIM)/2)
 8:     qtd++
 9:     se CHAVE ≤ TABELA[METADE] então
10:         FIM ← METADE - 1
11:     senão
12:         INIC ← METADE + 1
13:     fim se
14:     qtd ← qtd+2
15: até TABELA[METADE] = CHAVE ∨ INIC > FIM
16: se TABELA[METADE] = CHAVE então
17:     devolva METADE
18: senão
19:     devolva -1
20: fim se
21: devolva qtd
```

```
63 BUSCABIN    1  2  7 22 29 35 40 41 43 63 77 86 89 90 95 e  10
39 BUSCABIN   18 20 21 32 35 39 40 50 51 53 65 79 91 97 99 e  10
21 BUSCABIN   27 28 31 42 50 51 53 64 65 73 74 83 88 93 98 e  13
```

Busca em ABP (árvore binária de pesquisa) Ótimo desempenho, fácil programação usando recursividade Para esta busca, antes a árvore deve ser criada, por exemplo, com a seguinte função, mostrada no quadro. Eis agora a busca

```
 1: função BUSCAABP (inteiro CHAVE, inteiro ARVORE[num][3])
 2: inteiro ONDE, qtd
 3: qtd ← 0
 4: ONDE ← RAIZ
 5: enquanto ONDE ≠ -1 faça
 6:     qtd++
 7:     se CHAVE < TABELA[ONDE][3] então
 8:         ONDE ← TABELA[ONDE][1]
 9:         qtd++
10:     senão
11:         se CHAVE > TABELA[ONDE][3] então
12:             ONDE ← TABELA[ONDE][2]
13:             qtd++
14:         senão
15:             ...ACHOU...
16:         fim se
17:     fim se
18: fim enquanto
19: devolva qtd
```

```
28 BUSCAABP   35 42 64 85 28 24 68 23  6  7 59 71 94 46 37 e  3
34 BUSCAABP   98 53 67 93 77 75 66 69 41 70 51 76 65 59 34 e  7
55 BUSCAABP   32 96 46 21 95 72 28 39 24 34 84 79 53 68 73 e  14
```

Alguns dados experimentais

Para o teste a seguir, fiz 6 passagens de 1000 chaves. Para cada método, em cada passagem foram 3 tentativas: as primeiras 2 com sucesso e a última para uma chave inexistente. Eis os dados:

passagem	Lin	Lin Ord	Lin Sen	Hash	Bin	ABP
1	1265.3	1063.3	817.3	5.7	24.0	18.7
2	1777.3	749.3	621.7	5.0	30.0	18.7
3	1832.0	1314.0	674.0	5.7	29.0	20.0
4	1244.7	613.3	634.7	1.0	26.0	28.0
5	1200.7	1186.7	607.3	18.3	28.0	19.3
6	948.0	832.7	770.3	3.7	28.0	27.3
média	1378.0	959.9	687.6	6.6	27.5	22.0

☞ Para você fazer

Calcule:

```
59 BUSCALIN      26 54 71 31 33 79 95 97 20 29 89 69 84 25 61  e _____
20 BUSCALINORD    3 12 18 26 47 49 52 62 73 74 75 77 78 94 99  e _____
23 BUSCALINSEN   47 24 10 76 42 23 79 53 62 80 78 48 66 16 18  e _____
14 BUSCAHASH     27 99 56 33 21 93 53 48 24  6 72 18 42 54 51  e _____
 9 BUSCABIN       4  9 11 23 24 43 52 59 61 65 66 73 74 84 97  e _____
74 BUSCAABP      58 85 61 25 55 39 70  4 11 52 21 63 76 30 20  e _____
```