

BCC202 - Estruturas de Dados I

Aula 19: Pesquisa Sequencial e Pesquisa Binária

Pedro Silva

Universidade Federal de Ouro Preto, UFOP
Departamento de Computação, DECOM
Email: silvap@ufop.edu.br

2021



Conteúdo

Introdução

Pesquisa Sequencial

Pesquisa Binária

Considerações Finais

Bibliografia

Conteúdo

Introdução

Pesquisa Sequencial

Pesquisa Binária

Considerações Finais

Bibliografia

Pesquisa:

- ▶ Estudo de como recuperar informação a partir de uma grande massa de informação previamente armazenada.
- ▶ A informação é dividida em registros.
- ▶ Cada registro possui uma chave para ser usada na pesquisa.
- ▶ **Objetivo da pesquisa:** Encontrar uma ou mais ocorrências de registros com chaves iguais à chave de pesquisa.
- ▶ **Pesquisa com sucesso X sem sucesso.**

Abordagens de pesquisa em Memória Primária

- ▶ Pesquisa Sequencial.
- ▶ Pesquisa Binária.
- ▶ Árvores de Pesquisa:
 - ▶ Árvores Binárias de Pesquisa.
 - ▶ Árvores AVL.
- ▶ Transformação de Chave (*Hashing*):
 - ▶ Listas Encadeadas.
 - ▶ Endereçamento Aberto.
 - ▶ *Hashing* Perfeito.

Algoritmos de Pesquisa - TADs

- ▶ É importante considerar os algoritmos de pesquisa como tipos abstratos de dados (TADs), de tal forma que haja uma independência de implementação para as operações.
- ▶ Operações mais comuns:
 1. Inicializar a estrutura de dados.
 2. Pesquisar um ou mais registros com determinada chave.
 3. Inserir um novo registro.
 4. Retirar um registro específico.

Dicionário

- ▶ Dicionário é um TAD com as operações:
 1. Inicializa.
 2. Pesquisa.
 3. Insere.
 4. Retira.

- ▶ Analogia com um dicionário da língua portuguesa:
 - ▶ Chaves × Palavras.

 - ▶ Registros × Entradas associadas com pronúncia, definição, sinônimos, outras informações.

Conteúdo

Introdução

Pesquisa Sequencial

Pesquisa Binária

Considerações Finais

Bibliografia

Pesquisa Sequencial

Método de pesquisa mais simples: a partir do primeiro registro, pesquise sequencialmente até encontrar a chave procurada; então pare.

Armazenamento de um conjunto de registros por meio de um array.

A Busca (find):

- ▶ Pesquisa retorna o índice do registro que contém a chave x.
- ▶ Caso não esteja presente, o valor retornado é -1.
- ▶ A implementação não suporta mais de um registro com uma mesma chave, pois retorna o primeiro encontrado.

TAD Dicionário

```
1 typedef long TChave;
2
3 typedef struct {
4     TChave chave;
5     /* outros componentes */
6 } TRegistro;
7
8 typedef struct {
9     TRegistro *v;
10    int n, max;
11 } TDicionario;
```

```
12 /* inicializa um dicionário */
13 void TDicionario_Inicio(TDicionario *t) {
14     t->n = 0;
15     t->max = 10;
16     t->v = (TRegistro*) malloc(sizeof(TRegistro)* t->max);
17 }
18
19 /* encontra e retorna o índice da chave x no dicionário */
20 int TDicionario_Find(TDicionario *t, TChave c) {
21     int i;
22     for (i = t->n-1; i >= 0; i--)
23         if (t->v[i].chave == c)
24             return i;
25     // retorna -1 caso a chave não seja encontrada
26     return -1;
27 }
```

TAD Dicionário Operações II

```
28 /* insere um registro no dicionário */
29 void TDicionario_Inserere(TDicionario *t, TRegistro *x) {
30     if (t->n == t->max) {
31         t->max *= 2;
32         t->v = (TRegistro*) realloc(t->v, sizeof(TRegistro) * t->max);
33     }
34
35     // n é o tamanho
36     t->v[t->n++] = x;
37 }
```

Análise Pesquisa Sequencial

Análise:

- ▶ Pesquisa com Sucesso:
 - ▶ melhor caso: $C(n) = 1$.
 - ▶ pior caso: $C(n) = n$.
 - ▶ caso médio: $C(n) = (n + 1) / 2$.
- ▶ Pesquisa sem sucesso:
 - ▶ $C(n) = n + 1$.
- ▶ O algoritmo de pesquisa sequencial é a melhor escolha para o problema de pesquisa com $n < 25$.

Conteúdo

Introdução

Pesquisa Sequencial

Pesquisa Binária

Considerações Finais

Bibliografia

Pesquisa Binária

Pesquisa em tabela pode ser mais eficiente se registros forem mantidos em ordem.

Para saber se uma chave está presente na tabela:

1. Compare a chave com o registro que está na posição do meio da tabela.
2. Se a chave é menor então o registro procurado está na primeira metade da tabela
3. Se a chave é maior então o registro procurado está na segunda metade da tabela.
4. Repita até que a chave seja encontrada ou que se constate que a chave não existe na tabela.

Exemplo de Pesquisa Binária pela chave **G**

0	1	2	3	4	5	6	7
A	B	C	D	E	F	G	H

Exemplo de Pesquisa Binária pela chave **G**

0	1	2	3	4	5	6	7
A	B	C	D	E	F	G	H

Exemplo de Pesquisa Binária pela chave **G**

0	1	2	3	4	5	6	7
A	B	C	D	E	F	G	H

Exemplo de Pesquisa Binária pela chave **G**

0	1	2	3	4	5	6	7
A	B	C	D	E	F	G	H

Exemplo de Pesquisa Binária pela chave **G**

0	1	2	3	4	5	6	7
A	B	C	D	E	F	G	H

Exemplo de Pesquisa Binária pela chave **G**

0	1	2	3	4	5	6	7
A	B	C	D	E	F	G	H

Implementação da Pesquisa Binária

```
1 typedef long TChave;
2
3 typedef struct {
4     TChave Chave;
5     /* outros componentes */
6 } TRegistro;
7
8 typedef struct {
9     TRegistro *v;
10    int n, max;
11 } TDicionario;
12
13 /* inicializa um dicionário */
14 void TDicionario_Inicio(TDicionario *t) {
15     t->n = 0;
16     t->max = 10;
17     t->v = (TRegistro*) malloc(sizeof(TRegistro)* t->max);
18 }
```

Implementação da Pesquisa Binária Recursiva

```
19 /* encontra o índice da chave x no dicionário */
20 int TDicionario_Find(TDicionario *t, TChave x) {
21     return TDicionario_Binaria(t, 0, t->n-1, x); // t->n é o tamanho
22 }
23
24 /* encontra o índice da chave x no dicionário entre esq e dir */
25 int TDicionario_Binaria(TDicionario *t, int esq,
26                          int dir, TChave x) {
27     int meio = (esq+dir)/2;
28
29     if (t->v[meio].chave != x && esq == dir)
30         return -1;
31     else if (x > t->v[meio].chave)
32         return TDicionario_Binaria(t, meio+1, dir, x);
33     else if (x < t->v[meio].chave)
34         return TDicionario_Binaria(t, esq, meio-1, x);
35     else
36         return meio;
37 }
```

Implementação da Pesquisa Binária Iterativa

```
38 /* encontra o índice da chave x no dicionário */
39 int TDicionario_Find(TDicionario *t, TChave c) {
40     int i, esq, dir;
41     if (t->n == 0) return -1;
42
43     esq = 0;
44     dir = t->n-1;
45     do {
46         i = (esq + dir) / 2;
47         if (c > t->v[i].chave) esq = i + 1;
48         else dir = i - 1;
49     }
50     while (c != t->v[i].chave && esq <= dir);
51
52     if (c == t->v[i].chave) return i;
53     else return -1;
54 }
```


Análise de Pesquisa Binária

Análise

- ▶ A cada iteração do algoritmo, o tamanho da tabela é dividido ao meio.
- ▶ **Logo:** o número de vezes que o tamanho da tabela é dividido ao meio é cerca de $\log n$.
- ▶ **Ressalva:** o custo para manter a tabela ordenada é alto: a cada inserção na posição p da tabela implica no deslocamento dos registros a partir da posição p para as posições seguintes.
- ▶ Consequentemente, a pesquisa binária não deve ser usada em aplicações muito dinâmicas.

Conteúdo

Introdução

Pesquisa Sequencial

Pesquisa Binária

Considerações Finais

Bibliografia

Conclusão

- ▶ Foram vistas duas abordagens de pesquisa.
- ▶ Estrutura de dados melhores são necessárias para tornar a pesquisa mais eficiente.

Árvores de Pesquisa

Conteúdo

Introdução

Pesquisa Sequencial


Pesquisa Binária

Considerações Finais

Bibliografia

Bibliografia

Os conteúdos deste material, incluindo figuras, textos e códigos, foram extraídos ou adaptados de:

 Cormen, Thomas H. and Leiserson, Charles E. and Rivest, Ronald L. and Stein, Clifford.

Introduction to Algorithms.

The MIT Press, 2011.

Exercício

- ▶ Dada a sequência de números: 0 1 2 3 4 5 6 7 8 9 10.
- ▶ Faça uma pesquisa binária pelo elemento 9, apresentando a sequência de execução a cada passo (Teste de Mesa).