



Algoritmos de Busca



Prof. Luiz Gustavo Almeida Martins

Introdução

Frequentemente precisamos realizar a **busca por um elemento** em um determinado conjunto de dados

Introdução

Frequentemente precisamos realizar a **busca por um elemento** em um determinado conjunto de dados

Recuperação de informações a partir de uma **grande massa de dados previamente armazenada**

Introdução

Frequentemente precisamos realizar a **busca por um elemento** em um determinado conjunto de dados

Recuperação de informações a partir de uma **grande massa de dados previamente armazenada**

A informação é organizada em **registros**

Introdução

Frequentemente precisamos realizar a **busca por um elemento** em um determinado conjunto de dados

Recuperação de informações a partir de uma **grande massa de dados previamente armazenada**

A informação é organizada em **registros**

Cada registro possui uma **chave** (*key*) usada na pesquisa

Introdução

Frequentemente precisamos realizar a **busca por um elemento** em um determinado conjunto de dados

Recuperação de informações a partir de uma **grande massa de dados previamente armazenada**

A informação é organizada em **registros**

Cada registro possui uma **chave** (*key*) usada na pesquisa

Um conjunto de registros é chamado de **tabela** ou **arquivo**

Tipo Abstrato de Dados (TAD)

É importante tratar os algoritmos de pesquisa como um TAD:

Conjunto de operações associado a uma estrutura de dados

Proporciona uma independência de implementação para as operações

Tipo Abstrato de Dados (TAD)

É importante tratar os algoritmos de pesquisa como um TAD:

Conjunto de operações associado a uma estrutura de dados

Proporciona uma independência de implementação para as operações

Operações mais comuns:

Inicializar a estrutura de dados

Pesquisar um ou mais registros com uma determinada chave

Inserir um novo registro

Retirar um registro específico

Ordenar um arquivo de acordo com a chave

Mesclar dois arquivos para formar um arquivo maior

Etc.

Pesquisa em um dicionário

Dicionário é o nome comumente utilizado para descrever uma **estrutura de dados para pesquisa**

Pesquisa em um dicionário

Dicionário é o nome comumente utilizado para descrever uma **estrutura de dados para pesquisa**

Analogia com um dicionário da língua portuguesa:

chave = palavras

Demais campos = entradas associadas com as palavras (pronúncia, definição, sinônimos, etc.)

Pesquisa em um dicionário

Dicionário é o nome comumente utilizado para descrever uma **estrutura de dados para pesquisa**

Analogia com um dicionário da língua portuguesa:

chave = palavras

Demais campos = entradas associadas com as palavras (pronúncia, definição, sinônimos, etc.)

Geralmente possui as seguintes operações:

Inicializa

Insere

Remove

Pesquisa

Método de pesquisa

A escolha do método mais adequado para uma determinada aplicação depende:

Método de pesquisa

A escolha do método mais adequado para uma determinada aplicação depende:

Quantidade de dados envolvidos

Método de pesquisa

A escolha do método mais adequado para uma determinada aplicação depende:

Quantidade de dados envolvidos

Frequência de inserções e remoções no arquivo

Método de pesquisa

A escolha do método mais adequado para uma determinada aplicação depende:

Quantidade de dados envolvidos

Frequência de inserções e remoções no arquivo

Se o **conteúdo do arquivo** é **estável** (não muda muito com o tempo), é importante **minimizar o tempo de pesquisa através de sua estruturação**

Método de pesquisa

A escolha do método mais adequado para uma determinada aplicação depende:

Quantidade de dados envolvidos

Frequência de inserções e remoções no arquivo

Se o **conteúdo do arquivo é estável** (não muda muito com o tempo), é importante **minimizar o tempo de pesquisa através de sua estruturação**

O tempo necessário para essa estruturação é compensado pelo ganho na eficiência das futuras pesquisas

Método de pesquisa

Diferentes estratégias podem ser empregadas:

Método de pesquisa

Diferentes estratégias podem ser empregadas:

Busca linear ou sequencial

Método de pesquisa

Diferentes estratégias podem ser empregadas:

Busca linear ou sequencial

Busca binária

Método de pesquisa

Diferentes estratégias podem ser empregadas:

Busca linear ou sequencial

Busca binária

Árvore binária de busca

Método de pesquisa

Diferentes estratégias podem ser empregadas:

Busca linear ou sequencial

Busca binária

Árvore binária de busca

Árvores balanceadas

Método de pesquisa

Diferentes estratégias podem ser empregadas:

Busca linear ou sequencial

Busca binária

Árvore binária de busca

Árvores balanceadas

Tabelas de dispersão (*hash table*)

Método de pesquisa

Diferentes estratégias podem ser empregadas:

Busca linear ou sequencial

Busca binária

} Dados organizados em
estruturas lineares
(ex: vetores)

Árvore binária de busca

Árvores balanceadas

Tabelas de dispersão (*hash table*)

Método de pesquisa

Diferentes estratégias podem ser empregadas:

Busca linear ou sequencial
Busca binária

**Dados organizados em
estruturas lineares
(ex: vetores)**

Árvore binária de busca
Árvores balanceadas

**Dados organizados em
árvores**

Tabelas de dispersão (*hash table*)

Método de pesquisa

Diferentes estratégias podem ser empregadas:

Busca linear ou sequencial
Busca binária

**Dados organizados em
estruturas lineares
(ex: vetores)**

Árvore binária de busca
Árvores balanceadas

**Dados organizados em
árvores**

Tabelas de dispersão (*hash table*)

**Dados organizados em
tabelas**

Método de pesquisa

Diferentes estratégias podem ser empregadas:

Busca linear ou sequencial
Busca binária

**Dados organizados em
estruturas lineares
(ex: vetores)**

Árvore binária de busca
Árvores balanceadas

**Dados organizados em
árvores**

Tabelas de dispersão (*hash table*)

**Dados organizados em
tabelas**

Escolha afeta na forma de implementação do TAD

Formas de implementação

Implementações Lineares

Representação por vetor

Um **registro** é formado por:

Um **campo chave**

Um ou mais campos com informações adicionais

Representação por vetor

Um **registro** é formado por:

Um **campo chave**

Um ou mais **campos** com informações adicionais

Exemplo:

```
struct registro {  
    int chave;  
    char nome[100];  
    // Poderíamos ter outros campos em nosso registro  
};
```

Representação por vetor

Um **registro** é formado por:

Um **campo chave**

Um ou mais **campos** com informações adicionais

Exemplo:

```
struct registro {  
    int chave;  
    char nome[100];  
    // Poderíamos ter outros campos em nosso registro  
};
```

```
typedef struct registro Registro;
```

Representação por vetor

Um **arquivo** contém um **conjunto de registros**

Representação por vetor

Um **arquivo** contém um **conjunto de registros**

Exemplo:

```
#define MAX 10
```

```
struct arquivo {  
    Registro itens[MAX];  
    int tamanho;  
};
```


Representação por vetor

Um **arquivo** contém um **conjunto de registros**

Exemplo:

```
#define MAX 10
```

```
struct arquivo {  
    Registro itens[MAX];  
    int tamanho;  
};
```

```
typedef struct arquivo Arq;
```

Operações básicas

Inicializar a estrutura de dados:

Vetor de itens é alocado estaticamente ao se criar um arquivo

Operações básicas

Inicializar a estrutura de dados:

Vetor de itens é alocado estaticamente ao se criar um arquivo

Campo tamanho precisa ser inicializado com ZERO

Operações básicas

Inicializar a estrutura de dados:

Vetor de itens é alocado estaticamente ao se criar um arquivo
Campo tamanho precisa ser inicializado com ZERO

Implementação em C:

```
Arq * inicia_arq() {  
    Arq *arq = (Arq *) malloc(sizeof(Arq));  
  
    if (arq != NULL)  
        arq->tamanho = 0;  
  
    return arq;  
}
```

Operações básicas

Inserir um novo registro no arquivo:

O local mais adequado depende do critério de ordenação

Operações básicas

Inserir um novo registro no arquivo:

O local mais adequado depende do critério de ordenação

Sucesso depende da disponibilidade de espaço

Operações básicas

Inserir um novo registro no arquivo:

O local mais adequado depende do critério de ordenação

Sucesso depende da disponibilidade de espaço

Implementação em C:

```
int insere_registro(Arq *arq, Registro Elem) {  
    if( arq->tamanho == MAX ) // ou usar a função arq_cheio() se existir  
        return 0;  
  
    arq->items[arq->tamanho] = Elem; // Coloca o registro na última posição  
    arq->tamanho++;                  // Incrementa o número de registros  
    return 1;  
}
```

Operações básicas

Remover um registro do arquivo:

Envolve a pesquisa do registro

Operações básicas

Remover um registro do arquivo:

Envolve a pesquisa do registro

Sucesso depende da existência do registro desejado

Operações básicas

Remover um registro do arquivo:

Envolve a pesquisa do registro

Sucesso depende da existência do registro desejado

Implementação em C:

```
int remove_registro(Arq *arq, int chave) {  
    if( arq->tamanho == 0 ) return 0; // ou usar a função arq_vazio() se existir  
    int x, posicao = pesquisa(arq, chave); // Localiza o registro  
    if (posicao == -1) return 0; // Chave não encontrada  
    for (x = posicao; x < arq->tamanho; x++)  
        arq->itens[x] = arq->itens[x+1];  
    arq->tamanho--;  
    return 1;  
}
```

Operações básicas

Pesquisar um registro no arquivo:

Envolve **localizar** um registros que possua a **chave especificada**

Operações básicas

Pesquisar um registro no arquivo:

Envolve **localizar** um registros que possua a **chave especificada**

Sucesso também depende da existência do registro

Operações básicas

Pesquisar um registro no arquivo:

Envolve **localizar** um registros que possua a **chave especificada**

Sucesso também depende da existência do registro

Retorna a **posição do registro** ou um **valor inválido** quando o registro não é encontrado

Método de pesquisa

Busca linear ou sequencial

Busca sequencial

Consiste em pesquisar sequencialmente, a partir do 1º registro, até encontrar a chave procurada ou terminar o arquivo

É a **forma mais simples** de pesquisa

Busca sequencial

Consiste em pesquisar sequencialmente, a partir do 1º registro, até encontrar a chave procurada ou terminar o arquivo

É a **forma mais simples** de pesquisa

Pesquisa sequencial em um arquivo não ordenado:

```
int busca_seq(Arq *arq, int chave) {  
    if (arq->tamanho == 0)  
        return -1;  
    int x = 0;  
    while (x < arq->tamanho && arq->itens[x].chave != chave)  
        x++;  
    if (x == arq->tamanho)  
        return -1; // Registro não foi encontrado  
    return x; // Retorna posição do registro no arquivo  
}
```


Busca sequencial

Algoritmo ineficiente para um número elevado de dados

Busca sequencial

Algoritmo ineficiente para um número elevado de dados

Análise do algoritmo:

No pior caso (quando a chave não está no arquivo), serão necessárias **verificar TODOS os registros: $O(N)$**

Busca sequencial

Algoritmo ineficiente para um número elevado de dados

Análise do algoritmo:

No pior caso (quando a chave não está no arquivo), serão necessárias **verificar TODOS os registros: $O(N)$**

No caso médio, considerando que todas as entradas possuem a **mesma probabilidade de ocorrência**, temos **$N/2$ comparações (comportamento esperado)**

Cálculo do nº de operações do caso médio

Considerações:

Busca **sempre encontra o registro** procurado

p_i é a probabilidade do i -ésimo registro ser o procurado

A recuperação do i -ésimo registro envolve i comparações

Cálculo do nº de operações do caso médio

Considerações:

Busca **sempre encontra o registro** procurado

p_i é a probabilidade do i -ésimo registro ser o procurado

A recuperação do i -ésimo registro envolve i comparações

$$f(n) = 1 \times p_1 + 2 \times p_2 + 3 \times p_3 + \dots + n \times p_n$$

Cálculo de $f(n)$ depende do conhecimento sobre a distribuição das probabilidades p_i

Cálculo do nº de operações do caso médio

Considerações:

Busca **sempre encontra o registro** procurado

p_i é a probabilidade do i -ésimo registro ser o procurado

A recuperação do i -ésimo registro envolve i comparações

$$f(n) = 1 \times p_1 + 2 \times p_2 + 3 \times p_3 + \cdots + n \times p_n$$

Cálculo de $f(n)$ depende do conhecimento sobre a distribuição das probabilidades p_i

Se todas as entradas são igualmente prováveis ($p_i = 1/n$):

$$f(n) = (1 + 2 + 3 + \cdots + n) \times 1/n$$

Cálculo do nº de operações do caso médio

Considerações:

Busca **sempre encontra o registro** procurado

p_i é a probabilidade do i -ésimo registro ser o procurado

A recuperação do i -ésimo registro envolve i comparações

$$f(n) = 1 \times p_1 + 2 \times p_2 + 3 \times p_3 + \dots + n \times p_n$$

Cálculo de $f(n)$ depende do conhecimento sobre a distribuição das probabilidades p_i

Se todas as entradas são igualmente prováveis ($p_i = 1/n$):

$$f(n) = (1 + 2 + 3 + \dots + n) \times 1/n = (n+n^2)/2 \times 1/n$$

Cálculo do nº de operações do caso médio

Considerações:

Busca **sempre encontra o registro** procurado

p_i é a probabilidade do i -ésimo registro ser o procurado

A recuperação do i -ésimo registro envolve i comparações

$$f(n) = 1 \times p_1 + 2 \times p_2 + 3 \times p_3 + \dots + n \times p_n$$

Cálculo de $f(n)$ depende do conhecimento sobre a distribuição das probabilidades p_i

Se todas as entradas são igualmente prováveis ($p_i = 1/n$):

$$f(n) = (1 + 2 + 3 + \dots + n) \times 1/n = (n+n2)/2 \times 1/n = (n+1)/2$$

Análise empírica do caso médio

Suponha um vetor com 7 posições:

1	3	4	5	6	7	2
---	---	---	---	---	---	---

Quantas comparações tenho que fazer para encontrar o número 6?

Análise empírica do caso médio

Suponha um vetor com 7 posições:

1	3	4	5	6	7	2
---	---	---	---	---	---	---

Quantas comparações tenho que fazer para encontrar o número 6?

5 comparações

Análise empírica do caso médio

Considere que o número 6 poderia estar em outro local do vetor, com mesma probabilidade:

1	6	4	5	3	7	2
---	---	---	---	---	---	---

Quantas comparações tenho que fazer para encontrar o número 6?

Análise empírica do caso médio

Considere que o número 6 poderia estar em outro local do vetor, com mesma probabilidade:

1	6	4	5	3	7	2
---	---	---	---	---	---	---

Quantas comparações tenho que fazer para encontrar o número 6?

2 comparações

Análise empírica do caso médio

Veja algumas opções de vetor:

Em cada uma precisamos de
**diferentes números de
comparações** para achar o
número 6

Na média, quantas
comparações temos que fazer
para encontrar o número 6?

2	6	5	1	7	3	4
5	2	6	3	4	7	1
5	1	2	3	7	4	6
1	4	2	3	5	6	7
6	7	3	5	4	2	1
7	2	6	3	4	5	1
5	4	1	7	3	6	2
6	2	5	3	4	1	7
1	4	2	5	7	6	3
6	4	2	5	7	3	1
6	7	1	4	3	2	5
5	2	3	6	1	4	7
4	5	3	2	6	7	1
5	4	6	7	3	1	2
7	1	6	2	3	5	4

Análise empírica do caso médio

Veja algumas opções de vetor:

Em cada uma precisamos de
**diferentes números de
comparações** para achar o
número 6

Na média, quantas
comparações temos que fazer
para encontrar o número 6?

2	6	5	1	7	3	4
5	2	6	3	4	7	1
5	1	2	3	7	4	6
1	4	2	3	5	6	7
6	7	3	5	4	2	1
7	2	6	3	4	5	1
5	4	1	7	3	6	2
6	2	5	3	4	1	7
1	4	2	5	7	6	3
6	4	2	5	7	3	1
6	7	1	4	3	2	5
5	2	3	6	1	4	7
4	5	3	2	6	7	1
5	4	6	7	3	1	2
7	1	6	2	3	5	4
=====						
4	1	4	1	1	3	1

Quantas vezes o 6 aparece em cada posição?

Análise empírica do caso médio

Veja algumas opções de vetor:

Em cada uma precisamos de
**diferentes números de
comparações** para achar o
número 6

Na média, quantas
comparações temos que fazer
para encontrar o número 6?

2	6	5	1	7	3	4
5	2	6	3	4	7	1
5	1	2	3	7	4	6
1	4	2	3	5	6	7
6	7	3	5	4	2	1
7	2	6	3	4	5	1
5	4	1	7	3	6	2
6	2	5	3	4	1	7
1	4	2	5	7	6	3
6	4	2	5	7	3	1
6	7	1	4	3	2	5
5	2	3	6	1	4	7
4	5	3	2	6	7	1
5	4	6	7	3	1	2
7	1	6	2	3	5	4

Quantas vezes o 6 aparece em cada posição?

Quantas comparações ocorre por posição?

4	1	4	1	1	3	1
1	2	3	4	5	6	7



Análise empírica do caso médio

Número de comparações:

4x na primeira coluna

☞ $4 \times 1 = 4$

1x na segunda

☞ $1 \times 2 = 2$

4x na terceira

☞ $4 \times 3 = 12$

1x na quarta

☞ $1 \times 4 = 4$

1x na quinta

☞ $1 \times 5 = 5$

3x na sexta

☞ $3 \times 6 = 18$

1x na sétima

☞ $1 \times 7 = 7$

2	6	5	1	7	3	4
5	2	6	3	4	7	1
5	1	2	3	7	4	6
1	4	2	3	5	6	7
6	7	3	5	4	2	1
7	2	6	3	4	5	1
5	4	1	7	3	6	2
6	2	5	3	4	1	7
1	4	2	5	7	6	3
6	4	2	5	7	3	1
6	7	1	4	3	2	5
5	2	3	6	1	4	7
4	5	3	2	6	7	1
5	4	6	7	3	1	2
7	1	6	2	3	5	4
<hr/>						
4	1	4	1	1	3	1
1	2	3	4	5	6	7

Análise empírica do caso médio

Número de comparações:

4x na primeira coluna

☞ $4 \times 1 = 4$

1x na segunda

☞ $1 \times 2 = 2$

4x na terceira

☞ $4 \times 3 = 12$

1x na quarta

☞ $1 \times 4 = 4$

1x na quinta

☞ $1 \times 5 = 5$

3x na sexta

☞ $3 \times 6 = 18$

1x na sétima

☞ $1 \times 7 = 7$

Total: $4+2+12+4+5+18+7 = 52$

2	6	5	1	7	3	4
5	2	6	3	4	7	1
5	1	2	3	7	4	6
1	4	2	3	5	6	7
6	7	3	5	4	2	1
7	2	6	3	4	5	1
5	4	1	7	3	6	2
6	2	5	3	4	1	7
1	4	2	5	7	6	3
6	4	2	5	7	3	1
6	7	1	4	3	2	5
5	2	3	6	1	4	7
4	5	3	2	6	7	1
5	4	6	7	3	1	2
7	1	6	2	3	5	4
<hr/>						
4	1	4	1	1	3	1
1	2	3	4	5	6	7

Análise empírica do caso médio

Número de comparações:

52

Número de vetores:

15

2	6	5	1	7	3	4
5	2	6	3	4	7	1
5	1	2	3	7	4	6
1	4	2	3	5	6	7
6	7	3	5	4	2	1
7	2	6	3	4	5	1
5	4	1	7	3	6	2
6	2	5	3	4	1	7
1	4	2	5	7	6	3
6	4	2	5	7	3	1
6	7	1	4	3	2	5
5	2	3	6	1	4	7
4	5	3	2	6	7	1
5	4	6	7	3	1	2
7	1	6	2	3	5	4
=====						
4	1	4	1	1	3	1
1	2	3	4	5	6	7

Análise empírica do caso médio

Número de comparações:

52

Número de vetores:

15

Média de comparações: $52/15 = 3,46$

2	6	5	1	7	3	4
5	2	6	3	4	7	1
5	1	2	3	7	4	6
1	4	2	3	5	6	7
6	7	3	5	4	2	1
7	2	6	3	4	5	1
5	4	1	7	3	6	2
6	2	5	3	4	1	7
1	4	2	5	7	6	3
6	4	2	5	7	3	1
6	7	1	4	3	2	5
5	2	3	6	1	4	7
4	5	3	2	6	7	1
5	4	6	7	3	1	2
7	1	6	2	3	5	4
<hr/>						
4	1	4	1	1	3	1
1	2	3	4	5	6	7

Análise empírica do caso médio

Número de comparações:

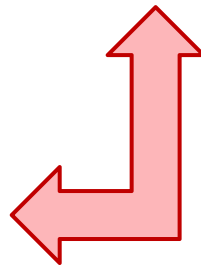
52

Número de vetores:

15

Média de comparações: $52/15 = 3,46$

$$f(n) = \frac{n+1}{2} = \frac{7+1}{2} = 4$$



2	6	5	1	7	3	4
5	2	6	3	4	7	1
5	1	2	3	7	4	6
1	4	2	3	5	6	7
6	7	3	5	4	2	1
7	2	6	3	4	5	1
5	4	1	7	3	6	2
6	2	5	3	4	1	7
1	4	2	5	7	6	3
6	4	2	5	7	3	1
6	7	1	4	3	2	5
5	2	3	6	1	4	7
4	5	3	2	6	7	1
5	4	6	7	3	1	2
7	1	6	2	3	5	4
<hr/>						
4	1	4	1	1	3	1
1	2	3	4	5	6	7

Análise empírica do caso médio

Novo experimento:

Geração aleatória de **10.000** vetores

Análise empírica do caso médio

Novo experimento:

Geração aleatória de **10.000** vetores

Qtde de ocorrências do 6 por coluna:

Coluna 1: 1429

Coluna 2: 1452

Coluna 3: 1422

Coluna 4: 1439

Coluna 5: 1466

Coluna 6: 1405

Coluna 7: 1387

Análise empírica do caso médio

Novo experimento:

Geração aleatória de **10.000** vetores

Número de comparações:

Coluna 1: 1429 x 1 = 1429

Coluna 2: 1452 x 2 = 2904

Coluna 3: 1422 x 3 = 4266

Coluna 4: 1439 x 4 = 5756

Coluna 5: 1466 x 5 = 7330

Coluna 6: 1405 x 6 = 8430

Coluna 7: 1387 x 7 = 9709

=====

39.824 comparações

Análise empírica do caso médio

Novo experimento:

39.824 comparações

10.000 vetores

Análise empírica do caso médio

Novo experimento:

39.824 comparações

10.000 vetores

Média de comparações: $39.824/10.000 = 3,9824 \approx 4$

Análise empírica do caso médio

Novo experimento:

39.824 comparações

10.000 vetores

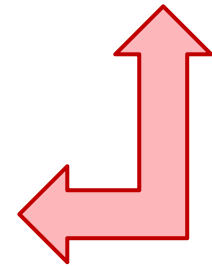
Conclusão:

No caso médio (esperado), o número médio de comparações é $(n+1)/2$

Considerando uma **distribuição uniforme** dos números no vetor

Média de comparações: $39.824/10.000 = 3,9824 \approx 4$

$$f(n) = \frac{n+1}{2} = \frac{7+1}{2} = 4$$



Busca sequencial com sentinela

Consiste em incluir um registro extra com a chave desejada no final do arquivo

Mudança na estrutura:

Quantidade de itens precisa prever um registro a mais

```
struct arquivo {  
    Registro itens[MAX+1];  
    int tamanho;  
};
```

Mudança na operação:

Elimina o teste lógico de final de arquivo

Busca sequencial com sentinela

Implementação em C:

```
int busca_seq(Arq *arq, int chave) {  
    if (arq->tamanho == 0)  
        return -1;  
  
    arq->itens[arq->tamanho].chave = chave; // Inclui sentinela  
  
    int x = 0;  
    while (arq->itens[x].chave != chave)  
        x++;  
    if (x == arq->tamanho)  
        return -1; // Registro não foi encontrado  
    return x; // Retorna posição do registro no arquivo  
}
```

Busca sequencial com sentinela

Não existe grande mudança na complexidade do algoritmo

Pesquisa com sucesso:

Melhor caso: $C(n) = 1$

Pior caso: $C(n) = n$

Caso médio: $C(n) = (n+1) / 2$

Pesquisa sem sucesso:

$C(n) = n + 1$

Redução na quantidade dos testes lógicas

Busca sequencial ordenada

A ordenação do vetor de pesquisa torna o algoritmo um pouco mais eficiente

Não precisa percorrer todo o vetor para determinar a inexistência da chave

Busca sequencial ordenada

A ordenação do vetor de pesquisa torna o algoritmo um pouco mais eficiente

Não precisa percorrer todo o vetor para determinar a inexistência da chave

Evita percorrimento para valores "fora do intervalo"

Busca sequencial ordenada

A ordenação do vetor de pesquisa torna o algoritmo um pouco mais eficiente

Não precisa percorrer todo o vetor para determinar a inexistência da chave

Evita percorrimento para valores "fora do intervalo"

Desvantagem:

Custo para a ordenação do vetor

Busca sequencial ordenada

Implementação em C:

```
int busca_seqOrd(Arq *arq, int chave) {  
    if ( arq->tamanho == 0 ||  
        arq->itens[x] > chave ||  
        arq->itens[arq->tamanho-1] < chave )  
        return -1;  
  
    int x = 0;  
    while (arq->itens[x].chave < chave)  
        x++;  
    if (arq->itens[x].chave > chave)  
        return -1; // Registro não foi encontrado  
    return x; // Retorna posição do registro no arquivo  
}
```

Busca sequencial ordenada

Mantém os custos quando a **chave é encontrada**:

Melhor caso: $C(n) = 1$

Pior caso: $C(n) = n$

Caso médio: $C(n) = (n+1) / 2$

Busca sequencial ordenada

Mantém os custos quando a **chave é encontrada**:

Melhor caso: $C(n) = 1$

Pior caso: $C(n) = n$

Caso médio: $C(n) = (n+1) / 2$

Ganho no caso médio (esperado) quando a pesquisa falha (**chave não existe**):

Melhor caso: $C(n) = 1$

Pior caso: $C(n) = n$

Caso médio: $C(n) = (n+1) / 2$

Método de pesquisa

Busca binária

Busca binária

Pesquisa **mais eficiente** em vetores

Precisa que os **registros estejam ordenados**

Busca binária

Pesquisa **mais eficiente** em vetores

Precisa que os **registros estejam ordenados**

Funcionamento:

Verifique o registro que está no **meio da tabela** (central)

Busca binária

Pesquisa **mais eficiente** em vetores

Precisa que os **registros estejam ordenados**

Funcionamento:

Verifique o registro que está no **meio da tabela** (central)

Se for a chave desejada, pára a busca e retorna a posição

Busca binária

Pesquisa **mais eficiente em vetores**

Precisa que os **registros estejam ordenados**

Funcionamento:

Verifique o registro que está no **meio da tabela** (central)

Se for a chave desejada, pára a busca e retorna a posição

Se a chave for menor, deve-se procurar na 1ª metade da tabela (esquerda do registro atual)

Busca binária

Pesquisa **mais eficiente em vetores**

Precisa que os **registros estejam ordenados**

Funcionamento:

Verifique o registro que está no **meio da tabela** (central)

Se for a chave desejada, pára a busca e retorna a posição

Se a chave for menor, deve-se procurar na 1ª metade da tabela (esquerda do registro atual)

Se a chave for maior, deve-se procurar na 2ª metade da tabela (direita do registro atual)

Busca binária

Pesquisa **mais eficiente** em vetores

Precisa que os **registros estejam ordenados**

Funcionamento:

Verifique o registro que está no **meio da tabela** (central)

Se for a chave desejada, pára a busca e retorna a posição

Se a chave for menor, deve-se procurar na 1ª metade da tabela (esquerda do registro atual)

Se a chave for maior, deve-se procurar na 2ª metade da tabela (direita do registro atual)

Repita o processo sobre o **novo espaço de busca** até:

Chave ser encontrada (**sucesso**)

Busca binária

Pesquisa **mais eficiente** em vetores

Precisa que os **registros estejam ordenados**

Funcionamento:

Verifique o registro que está no **meio da tabela** (central)

Se for a chave desejada, pára a busca e retorna a posição

Se a chave for menor, deve-se procurar na 1ª metade da tabela (esquerda do registro atual)

Se a chave for maior, deve-se procurar na 2ª metade da tabela (direita do registro atual)

Repita o processo sobre o **novo espaço de busca** até:

Chave ser encontrada (**sucesso**)

Ficar apenas um registro cuja chave é diferente (**falha**)

Busca binária: exemplo

Índice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Chaves	a	b	c	d	e	f	g	h	h	i	j	j	k	l	m	n	o	o	p	q	r	s	s	t	u	v	v	x	x	z	z	z

Pesquisar 'r'

Busca binária: exemplo

Pesquisar 'r'

Índice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Chaves	a	b	c	d	e	f	g	h	h	i	j	j	k	l	m	n	o	o	p	q	r	s	s	t	u	v	v	x	x	z	z	z

Índice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Chaves	a	b	c	d	e	f	g	h	h	i	j	j	k	l	m	n	o	o	p	q	r	s	s	t	u	v	v	x	x	z	z	z
	i										m																	f				

Início: 0
Fim: 31
Meio: 15
Nº de Comparações: 1

i: início; f: final; m: meio

Busca binária: exemplo

Pesquisar 'r'

Índice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Chaves	a	b	c	d	e	f	g	h	h	i	j	j	k	l	m	n	o	o	p	q	r	s	s	t	u	v	v	x	x	z	z	z

Índice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Chaves	a	b	c	d	e	f	g	h	h	i	j	j	k	l	m	n	o	o	p	q	r	s	s	t	u	v	v	x	x	z	z	z
	i															m															f	

Início: 0
Fim: 31
Meio: 15
Nº de Comparações: 1

Índice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Chaves	a	b	c	d	e	f	g	h	h	i	j	j	k	l	m	n	o	o	p	q	r	s	s	t	u	v	v	x	x	z	z	z
																	i	m														f

Início: 16
Fim: 31
Meio: 23
Nº de Comparações: 2

i: início; f: final; m: meio

Busca binária: exemplo

Pesquisar 'r'

Índice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Chaves	a	b	c	d	e	f	g	h	h	i	j	j	k	l	m	n	o	o	p	q	r	s	s	t	u	v	v	x	x	z	z	z

Índice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Chaves	a	b	c	d	e	f	g	h	h	i	j	j	k	l	m	n	o	o	p	q	r	s	s	t	u	v	v	x	x	z	z	z

Início: 0
Fim: 31
Meio: 15
Nº de Comparações: 1

Índice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Chaves	a	b	c	d	e	f	g	h	h	i	j	j	k	l	m	n	o	o	p	q	r	s	s	t	u	v	v	x	x	z	z	z

Início: 16
Fim: 31
Meio: 23
Nº de Comparações: 2

Índice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Chaves	a	b	c	d	e	f	g	h	h	i	j	j	k	l	m	n	o	o	p	q	r	s	s	t	u	v	v	x	x	z	z	z

Início: 16
Fim: 22
Meio: 19
Nº de Comparações: 3

i: início; f: final; m: meio

Busca binária: exemplo

Pesquisar 'r'

Índice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Chaves	a	b	c	d	e	f	g	h	h	i	j	j	k	l	m	n	o	o	p	q	r	s	s	t	u	v	v	x	x	z	z	z

Índice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Chaves	a	b	c	d	e	f	g	h	h	i	j	j	k	l	m	n	o	o	p	q	r	s	s	t	u	v	v	x	x	z	z	z

Início: 0

Fim: 31

Meio: 15

Nº de Comparações: 1

Índice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Chaves	a	b	c	d	e	f	g	h	h	i	j	j	k	l	m	n	o	o	p	q	r	s	s	t	u	v	v	x	x	z	z	z

Início: 16

Fim: 31

Meio: 23

Nº de Comparações: 2

Índice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Chaves	a	b	c	d	e	f	g	h	h	i	j	j	k	l	m	n	o	o	p	q	r	s	s	t	u	v	v	x	x	z	z	z

Início: 16

Fim: 22

Meio: 19

Nº de Comparações: 3

Índice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Chaves	a	b	c	d	e	f	g	h	h	i	j	j	k	l	m	n	o	o	p	q	r	s	s	t	u	v	v	x	x	z	z	z

Início: 20

Fim: 22

Meio: 21

Nº de Comparações: 4

i: início; f: final; m: meio



Busca binária: exemplo

Pesquisar 'r'

Índice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Chaves	a	b	c	d	e	f	g	h	h	i	j	j	k	l	m	n	o	o	p	q	r	s	s	t	u	v	v	x	x	z	z	z

Índice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Chaves	a	b	c	d	e	f	g	h	h	i	j	j	k	l	m	n	o	o	p	q	r	s	s	t	u	v	v	x	x	z	z	z

Início: 0

Fim: 31

Meio: 15

Nº de Comparações: 1

Índice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Chaves	a	b	c	d	e	f	g	h	h	i	j	j	k	l	m	n	o	o	p	q	r	s	s	t	u	v	v	x	x	z	z	z

Início: 16

Fim: 31

Meio: 23

Nº de Comparações: 2

Índice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Chaves	a	b	c	d	e	f	g	h	h	i	j	j	k	l	m	n	o	o	p	q	r	s	s	t	u	v	v	x	x	z	z	z

Início: 16

Fim: 22

Meio: 19

Nº de Comparações: 3

Índice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Chaves	a	b	c	d	e	f	g	h	h	i	j	j	k	l	m	n	o	o	p	q	r	s	s	t	u	v	v	x	x	z	z	z

Início: 20

Fim: 22

Meio: 21

Nº de Comparações: 4

Índice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Chaves	a	b	c	d	e	f	g	h	h	i	j	j	k	l	m	n	o	o	p	q	r	s	s	t	u	v	v	x	x	z	z	z

Início: 20

Fim: 20

Meio: 20

Nº de Comparações: 5

i: início; f: final; m: meio

Busca binária: implementação

```
int busca_bin(Arq *arq, int chave) {
```

```
}
```

Busca binária: implementação

```
int busca_bin(Arq *arq, int chave) {  
    if (arq->tamanho == 0)    return -1; // Arquivo vazio
```

```
}
```

Busca binária: implementação

```
int busca_bin(Arq *arq, int chave) {  
    if (arq->tamanho == 0)    return -1; // Arquivo vazio
```

```
    int meio, ini = 0, fim = arq->tamanho-1;  
    while ( ini <= fim ) {
```

```
    }
```

```
}
```

Busca binária: implementação

```
int busca_bin(Arq *arq, int chave) {  
    if (arq->tamanho == 0)    return -1; // Arquivo vazio
```

```
    int meio, ini = 0, fim = arq->tamanho-1;  
    while ( ini <= fim ) {  
        meio = (ini + fim) / 2; // Divisão inteira
```

```
    }
```

```
}
```

Busca binária: implementação

```
int busca_bin(Arq *arq, int chave) {  
    if (arq->tamanho == 0)    return -1;  // Arquivo vazio  
  
    int meio, ini = 0, fim = arq->tamanho-1;  
    while ( ini <= fim ) {  
        meio = (ini + fim) / 2;  // Divisão inteira  
        if ( arq->itens[meio].chave == chave )  
            return meio;  // Retorna posição do registro no arquivo  
  
    }  
  
}
```

Busca binária: implementação

```
int busca_bin(Arq *arq, int chave) {  
    if (arq->tamanho == 0)    return -1; // Arquivo vazio  
  
    int meio, ini = 0, fim = arq->tamanho-1;  
    while ( ini <= fim ) {  
        meio = (ini + fim) / 2; // Divisão inteira  
        if ( arq->itens[meio].chave == chave )  
            return meio; // Retorna posição do registro no arquivo  
        else if ( arq->itens[meio].chave > chave )  
            fim = meio-1;  
  
    }  
  
}
```

Busca binária: implementação

```
int busca_bin(Arq *arq, int chave) {  
    if (arq->tamanho == 0)    return -1; // Arquivo vazio  
  
    int meio, ini = 0, fim = arq->tamanho-1;  
    while ( ini <= fim ) {  
        meio = (ini + fim) / 2; // Divisão inteira  
        if ( arq->itens[meio].chave == chave )  
            return meio; // Retorna posição do registro no arquivo  
        else if ( arq->itens[meio].chave > chave )  
            fim = meio-1;  
        else  
            ini = meio+1;  
    }  
  
}
```


Busca binária: implementação

```
int busca_bin(Arq *arq, int chave) {  
    if (arq->tamanho == 0)    return -1; // Arquivo vazio  
  
    int meio, ini = 0, fim = arq->tamanho-1;  
    while ( ini <= fim ) {  
        meio = (ini + fim) / 2; // Divisão inteira  
        if ( arq->itens[meio].chave == chave )  
            return meio; // Retorna posição do registro no arquivo  
        else if ( arq->itens[meio].chave > chave )  
            fim = meio-1;  
        else  
            ini = meio+1;  
    }  
    return -1; // Registro não foi encontrado  
}
```

Busca binária: análise

Eficiência está relacionada com a **quantidade de iterações**

Número de comparações:

São realizadas no máximo 2 comparações: $=$ e $>$

Custo constante

Número de divisões:

Arquivo é **dividido ao meio** a cada iteração

No pior caso, o processo ocorre até não ser mais possível dividir o arquivo, ou seja, **restar um único registro**

Custo relacionado com o **tamanho do vetor** (**fator crítico**)

Busca binária: análise

Quantas iterações precisamos para tratar um vetor?

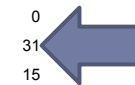
Índice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Chaves	a	b	c	d	e	f	g	h	h	i	j	j	k	l	m	n	o	o	p	q	r	s	s	t	u	v	v	x	x	z	z	z

Índice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Chaves	a	b	c	d	e	f	g	h	h	i	j	j	k	l	m	n	o	o	p	q	r	s	s	t	u	v	v	x	x	z	z	z

i m f

Num. de Comparações:

Início: 0
Fim: 31
Meio: 15



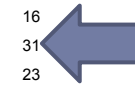
Tamanho vetor	$N = 32$
Iteração	1

Índice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Chaves	a	b	c	d	e	f	g	h	h	i	j	j	k	l	m	n	o	o	p	q	r	s	s	t	u	v	v	x	x	z	z	z

i m f

Num. de Comparações:

Início: 16
Fim: 31
Meio: 23



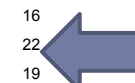
Tamanho vetor	$\frac{N}{2} = 16$
Iteração	2

Índice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Chaves	a	b	c	d	e	f	g	h	h	i	j	j	k	l	m	n	o	o	p	q	r	s	s	t	u	v	v	x	x	z	z	z

i m f

Num. de Comparações:

Início: 16
Fim: 22
Meio: 19



Tamanho vetor	$\frac{N}{4} = 7$
Iteração	3

Índice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Chaves	a	b	c	d	e	f	g	h	h	i	j	j	k	l	m	n	o	o	p	q	r	s	s	t	u	v	v	x	x	z	z	z

i m f

Num. de Comparações:

Início: 20
Fim: 22
Meio: 21



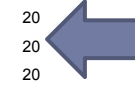
Tamanho vetor	$\frac{N}{8} = 3$
Iteração	4

Índice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Chaves	a	b	c	d	e	f	g	h	h	i	j	j	k	l	m	n	o	o	p	q	r	s	s	t	u	v	v	x	x	z	z	z

i m f

Num. de Comparações:

Início: 20
Fim: 20
Meio: 20



Tamanho vetor	$\frac{N}{16} = 1$
Iteração	5



Busca binária: análise

Se um vetor tem tamanho n , qual será seu tamanho após i divisões ao meio?

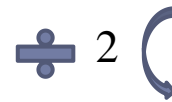
Busca binária: análise

Se um vetor tem tamanho n , qual será seu tamanho após i divisões ao meio?

Tamanho vetor	Nº de divisões
$n = \frac{n}{2^0}$	0

Busca binária: análise

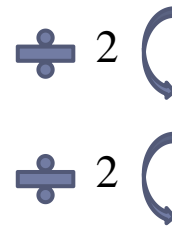
Se um vetor tem tamanho n , qual será seu tamanho após i divisões ao meio?



Tamanho vetor	Nº de divisões
$n = \frac{n}{2^0}$	0
$\frac{n}{2} = \frac{n}{2^1}$	1

Busca binária: análise

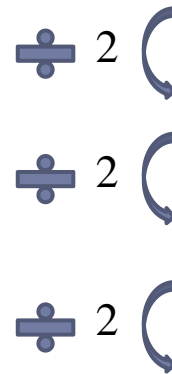
Se um vetor tem tamanho n , qual será seu tamanho após i divisões ao meio?



Tamanho vetor	Nº de divisões
$n = \frac{n}{2^0}$	0
$\frac{n}{2} = \frac{n}{2^1}$	1
$\frac{\frac{n}{2}}{2} = \frac{n}{4} = \frac{n}{2^2}$	2

Busca binária: análise

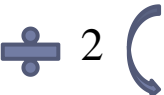
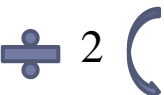



Se um vetor tem tamanho n , qual será seu tamanho após i divisões ao meio?



Tamanho vetor	Nº de divisões
$n = \frac{n}{2^0}$	0
$\frac{n}{2} = \frac{n}{2^1}$	1
$\frac{n}{2} = \frac{n}{4} = \frac{n}{2^2}$	2
$\frac{n}{4} = \frac{n}{8} = \frac{n}{2^3}$	3

Busca binária: análise

Se um vetor tem tamanho n , qual será seu tamanho após i divisões ao meio?

	Tamanho vetor	Nº de divisões
$\div 2$ 	$n = \frac{n}{2^0}$	0
$\div 2$ 	$\frac{n}{2} = \frac{n}{2^1}$	1
$\div 2$ 	$\frac{n}{2} = \frac{n}{4} = \frac{n}{2^2}$	2
$\div 2$ 	$\frac{n}{4} = \frac{n}{8} = \frac{n}{2^3}$	3
$\div 2$ 	$\frac{n}{8} = \frac{n}{16} = \frac{n}{2^4}$	4

Busca binária: análise

Se um vetor tem tamanho n , qual será seu tamanho após i divisões ao meio?

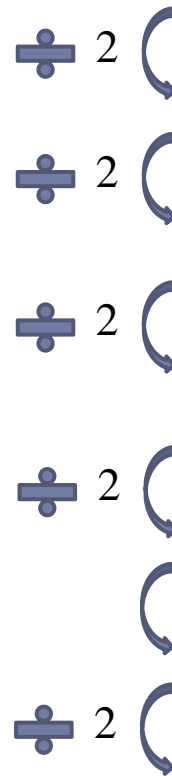
	Tamanho vetor	Nº de divisões
	$n = \frac{n}{2^0}$	0
$\div 2$ 	$\frac{n}{2} = \frac{n}{2^1}$	1
$\div 2$ 	$\frac{n}{2} \div 2 = \frac{n}{4} = \frac{n}{2^2}$	2
$\div 2$ 	$\frac{n}{4} \div 2 = \frac{n}{8} = \frac{n}{2^3}$	3
$\div 2$ 	$\frac{n}{8} \div 2 = \frac{n}{16} = \frac{n}{2^4}$	4

$\div 2$ 	$\frac{n}{2^i}$	i

Busca binária: análise

Se um vetor tem tamanho n , qual será seu tamanho após i divisões ao meio?

$$\frac{n}{2^i}$$



Tamanho vetor	Nº de divisões
$n = \frac{n}{2^0}$	0
$\frac{n}{2} = \frac{n}{2^1}$	1
$\frac{n}{2} \div 2 = \frac{n}{4} = \frac{n}{2^2}$	2
$\frac{n}{4} \div 2 = \frac{n}{8} = \frac{n}{2^3}$	3
$\frac{n}{8} \div 2 = \frac{n}{16} = \frac{n}{2^4}$	4
...	...
$\frac{n}{2^i}$	i

Busca binária: análise

Qual é o valor de i para que o vetor tenha tamanho 1 (não seja possível dividi-lo)?

$$\frac{n}{2^i} = 1$$

Busca binária: análise

Qual é o valor de i para que o vetor tenha tamanho 1 (não seja possível dividi-lo)?

$$\frac{n}{2^i} = 1$$

Desenvolvendo a fórmula: $n = 2^i$

Busca binária: análise

Qual é o valor de i para que o vetor tenha tamanho 1 (não seja possível dividi-lo)?

$$\frac{n}{2^i} = 1$$

Desenvolvendo a fórmula: $n = 2^i$

Aplicando \log_2 nos 2 lados: $\log_2 n = \log_2 2^i$

Busca binária: análise

Qual é o valor de i para que o vetor tenha tamanho 1 (não seja possível dividi-lo)?

$$\frac{n}{2^i} = 1$$

Desenvolvendo a fórmula: $n = 2^i$

Aplicando \log_2 nos 2 lados: $\log_2 n = \log_2 2^i$
 $\log_2 n = i \log_2 2$

Busca binária: análise

Qual é o valor de i para que o vetor tenha tamanho 1 (não seja possível dividi-lo)?

$$\frac{n}{2^i} = 1$$

Desenvolvendo a fórmula: $n = 2^i$

Aplicando \log_2 nos 2 lados: $\log_2 n = \log_2 2^i$
 $\log_2 n = i \cancel{\log_2 2}^1$

Busca binária: análise

Qual é o valor de i para que o vetor tenha tamanho 1 (não seja possível dividi-lo)?

$$\frac{n}{2^i} = 1$$

Desenvolvendo a fórmula: $n = 2^i$

Aplicando \log_2 nos 2 lados: $\log_2 n = \log_2 2^i$
 $\log_2 n = i \cancel{\log_2 2}^1$
 $i = \log_2 n$

Busca binária: análise

Custo total:

Busca binária: análise

Custo total:

Comparações por iteração tem **custo constante**

Busca binária: análise

Custo total:

Comparações por iteração tem **custo constante**

Quantidade máxima de divisões (i) é $\log_2 n$

Busca binária: análise

Custo total:

Comparações por iteração tem **custo constante**

Quantidade máxima de divisões (i) é $\log_2 n$

$$C(n) = \log_2 n$$

Busca binária: análise

Custo total:

Comparações por iteração tem **custo constante**

Quantidade máxima de divisões (i) é $\log_2 n$

$$C(n) = \log_2 n$$

Ressalva: não é recomendada para **aplicações muito dinâmicas**

Custo para manter a tabela ordenada é alto

Envolve o deslocamento de elementos

Exercícios

1. Construa um algoritmo para realizar a **análise empírica** da complexidade do caso médio da busca sequencial, considerando 100, 1.000 e 10.000 vetores de 15 posições (cada uma com um valor distinto de 1 a 15). Demonstre a complexidade usando o número 10 como chave. Ao final, o algoritmo deve apresentar o número de ocorrências do 10 em cada posição do vetor, a quantidade de comparações realizadas para encontrá-lo (detalhada como nos slides) e a média de comparações.
2. Implemente o algoritmo de busca binária de **forma recursiva** e compare sua complexidade com a versão iterativa, considerando 100 execuções sobre vetores de 1.000 inteiros distintos para encontrar o número 169.

Bibliografia

Slides adaptados do material da Profa. Gina M. B. Oliveira e do Prof. Dr. Bruno Travençolo.

BACKES, A. Linguagem C Descomplicada: portal de vídeo-aulas para estudo de programação. Disponível em:

<https://programacaodescomplicada.wordpress.com/indice/estrutura-de-dados/>

CORMEN, T.H. et al. Algoritmos: Teoria e Prática, Campus, 2002

ZIVIANI, N. Projeto de algoritmos: com implementações em Pascal e C (2ª ed.), Thomson, 2004

MORAES, C.R. Estruturas de Dados e Algoritmos: uma abordagem didática (2ª ed.), Futura, 2003