



Estruturas de Dados

Pesquisa

Material produzido por Thiago Caproni e Paulo Muniz de Ávila
¹Douglas Castilho

¹douglas.braz@ifsuldeminas.edu.br

Última Atualização: 14 de fevereiro de 2017

- 1 Introdução
- 2 Pesquisa
- 3 A pesquisa Sequencial
- 4 A pesquisa Binária

1 Introdução

2 Pesquisa

3 A pesquisa Sequencial

4 A pesquisa Binária

*"No mundo da computação, talvez as tarefas mais fundamentais e extensivamente analisadas sejam **ordenação** e **pesquisa**. Essas rotinas são utilizadas em praticamente todos os programas de banco de dados, bem como compiladores, interpretadores e sistemas operacionais. "*

1 Introdução

2 Pesquisa

3 A pesquisa Sequencial

4 A pesquisa Binária

- Banco de dados existem para que, de tempos em tempos, um usuário possa localizar o dado de um registro, simplesmente digitando a sua **chave**.
- Há apenas um método para encontrar informações em um arquivo (*matriz*) desordenado e um outro para um arquivo (*matriz*) ordenado.
- Vamos analisar esses métodos nos próximos slides !

- Encontrar informações em uma matriz desordenada requer uma **pesquisa sequencial**, começando no primeiro elemento e parando quando o elemento procurado ou o final da matriz é encontrado.
- Esse método deve ser usado em dados **desordenados**, mas também pode ser aplicado a dados **ordenados**, com uma perda de desempenho considerada.
- Se os dados foram ordenados, é possível utilizar o método de **pesquisa binária**, o que ajuda a localizar o dado mais rapidamente.

1 Introdução

2 Pesquisa

3 A pesquisa Sequencial

4 A pesquisa Binária

- A pesquisa sequencial é fácil de ser codificada. A função a seguir pesquisa em um elemento em um vetor de inteiros

```
int seq_search(int *item, int key, int count){  
    int t;  
    for(t=0;t<count;t++){  
        if(key==item[t])  
            return t;  
    }  
  
    return -1;  
}
```

- Essa função devolve a posição do elemento encontrado no vetor ($0 \dots n - 1$) ou -1 se o elemento procurado não foi localizado.

Análise da Complexidade

- Como possui apenas uma estrutura de repetição, o **pior caso** da busca acontece quando o número procurado é o último elemento do vetor ou quando o número procurado não se encontra no vetor, realizando então n comparações. Logo, o tempo de execução é $T(n) = O(n)$.
- O **melhor caso** ocorre quando o número procurado é o elemento que se encontra na primeira posição, realizando apenas uma comparação e cujo tempo de execução é constante, ou seja, $T(n) = O(1)$.

```
int seq_search(int *item, int key, int count){
    int t;
    for(t=0; t<count; t++){
        if(key==item[t])
            return t;
    }

    return -1;
}
```

1 Introdução

2 Pesquisa

3 A pesquisa Sequencial

4 A pesquisa Binária

- O algoritmo de busca **binária** é executado somente em vetores **ordenados**.
- Nesse algoritmo o vetor com os dados é dividido ao meio e o número do meio é comparado ao número procurado.
- Se estes forem iguais, a busca termina.
- Se o número procurado for menor que o elemento do meio do vetor, a busca será realizada no vetor à esquerda ao do meio.
- Se o número procurado for maior que o elemento do meio do vetor, a busca será realizada no vetor à direita ao do meio.
- Esse procedimento de **divisão** e **comparação** acontece até que o vetor de dados fique com apenas um elemento ou até o número procurado ser encontrado.

- Analise a implementação abaixo do algoritmo de **busca binária**.

```
int binary_search(int *item, int key, int count)
    int low, high, mid;
    low=0; high = count-1;

    while(low <= high){
        mid=(low+high)/2;
        if(key<item[mid]){
            high=mid-1;
        }else if(key>item[mid]){
            low=mid+1;
        }else{
            return mid;
        }
    }

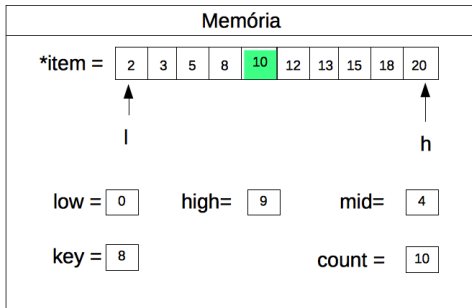
    return -1;
```

- Simulação do algoritmo de **busca binária**.

```
int binary_search(int *item, int key, int count){
    int low,high,mid;
    low=0;high = count-1;

    while(low <= high){
        mid=(low+high)/2;
        if(key<item[mid]){
            high=mid-1;
        }else if(key>item[mid]){
            low=mid+1;
        }else{
            return mid;
        }
    }

    return -1;
}
```

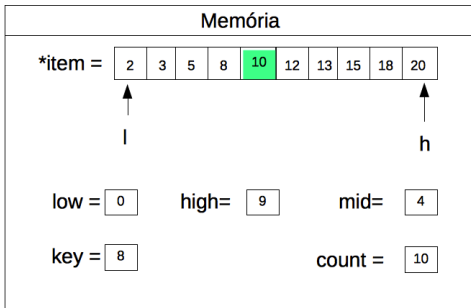


- Simulação do algoritmo de **busca binária**.

```
int binary_search(int *item, int key, int count){
    int low,high,mid;
    low=0;high = count-1;

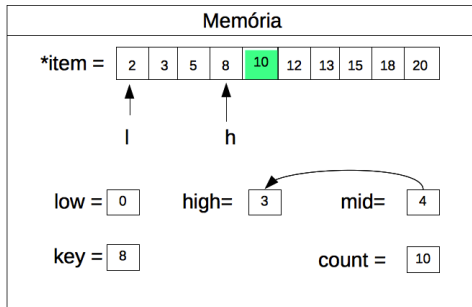
    while(low <= high){
        mid=(low+high)/2;      8 < 10 → V
        if(key<item[mid]){
            high=mid-1;
        }else if(key>item[mid]){
            low=mid+1;
        }else{
            return mid;
        }
    }

    return -1;
}
```



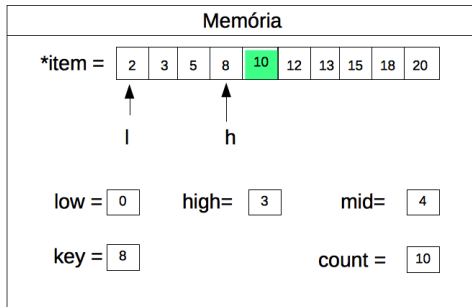
- Simulação do algoritmo de **busca binária**.

```
int binary_search(int *item, int key, int count){  
    int low,high,mid;  
    low=0;high = count-1;  
  
    while(low <= high){  
        mid=(low+high)/2;  
        if(key<item[mid]){  
            high=mid-1;  
        }else if(key>item[mid]){  
            low=mid+1;  
        }else{  
            return mid;  
        }  
    }  
    return -1;  
}
```



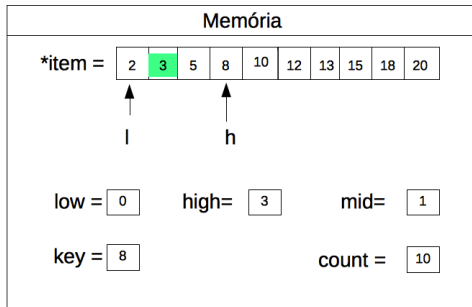
- Simulação do algoritmo de **busca binária**.

```
int binary_search(int *item, int key, int count){  
    int low,high,mid;  
    low=0;high = count-1;     $0 < 3 \rightarrow V$   
    while(low <= high){  
        mid=(low+high)/2;  
        if(key<item[mid]){  
            high=mid-1;  
        }else if(key>item[mid]){  
            low=mid+1;  
        }else{  
            return mid;  
        }  
    }  
    return -1;  
}
```



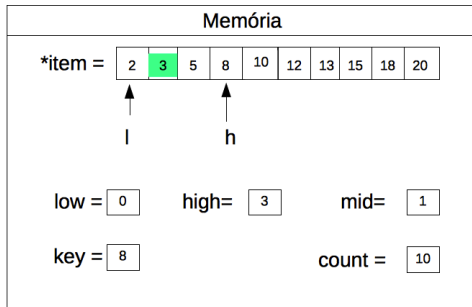
- Simulação do algoritmo de **busca binária**.

```
int binary_search(int *item, int key, int count){  
    int low,high,mid;  
    low=0;high = count-1;  
  
    while(low <= high){  
        mid=(low+high)/2;  
        if(key<item[mid]){  
            high=mid-1;  
        }else if(key>item[mid]){  
            low=mid+1;  
        }else{  
            return mid;  
        }  
    }  
    return -1;  
}
```



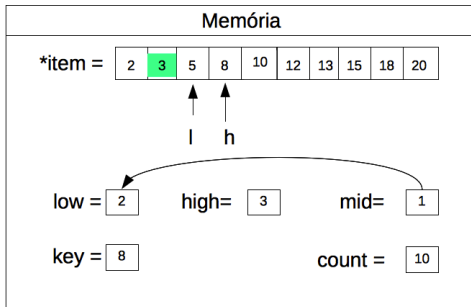
- Simulação do algoritmo de **busca binária**.

```
int binary_search(int *item, int key, int count){  
    int low,high,mid;  
    low=0;high = count-1;  
  
    while(low <= high){  
        mid=(low+high)/2;    8 < 3 → F  
        if(key<item[mid]){  
            high=mid-1;  
        }else if(key>item[mid]){  
            low=mid+1;  
        }else{  
            return mid;  
        }  
    }  
    return -1;  
}
```



- Simulação do algoritmo de **busca binária**.

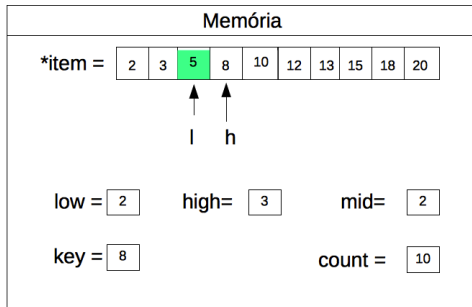
```
int binary_search(int *item, int key, int count){  
    int low,high,mid;  
    low=0;high = count-1;  
  
    while(low <= high){  
        mid=(low+high)/2;  
        if(key<item[mid]){  
            high=mid-1;  
        }else if(key>item[mid]){ 8 > 3 → V  
            low=mid+1;  
        }else{  
            return mid;  
        }  
    }  
  
    return -1;  
}
```



- Simulação do algoritmo de **busca binária**.

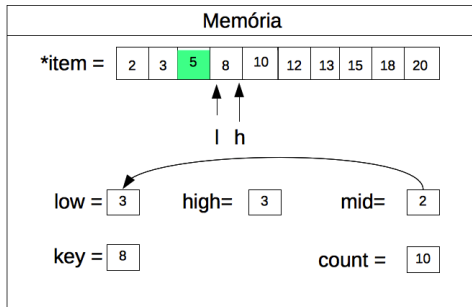
```
int binary_search(int *item, int key, int count){  
    int low,high,mid;  
    low=0;high = count-1;  
  
    while(low <= high){  
        mid=(low+high)/2;  
        if(key<item[mid]){  
            high=mid-1;  
        }else if(key>item[mid]){  
            low=mid+1;  
        }else{  
            return mid;  
        }  
    }  
    return -1;  
}
```

$2 \leq 3 \rightarrow V$



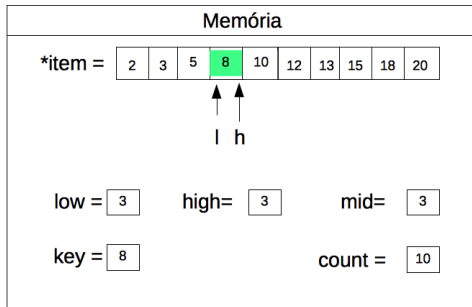
- Simulação do algoritmo de **busca binária**.

```
int binary_search(int *item, int key, int count){  
    int low,high,mid;  
    low=0;high = count-1;  
  
    while(low <= high){  
        mid=(low+high)/2;  
        if(key<item[mid]){ 8 < 5 → F  
            high=mid-1;  
        }else if(key>item[mid]){ 8 > 5 → V  
            low=mid+1;  
        }else{  
            return mid;  
        }  
    }  
    return -1;  
}
```



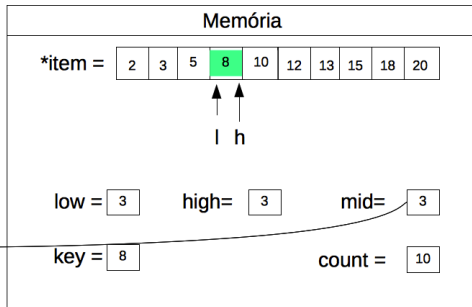
- Simulação do algoritmo de **busca binária**.

```
int binary_search(int *item, int key, int count){  
    int low,high,mid;  
    low=0;high = count-1;  
  
    while(low <= high){  $3 \leq 3 \rightarrow V$   
        mid=(low+high)/2;  
        if(key<item[mid]){  
            high=mid-1;  
        }else if(key>item[mid]){  
            low=mid+1;  
        }else{  
            return mid;  
        }  
    }  
    return -1;  
}
```



- Simulação do algoritmo de **busca binária**.

```
int binary_search(int *item, int key, int count){  
    int low,high,mid;  
    low=0;high = count-1;  
  
    while(low <= high){  
        mid=(low+high)/2;  
        if(key<item[mid]){ 8 < 8 → F  
            high=mid-1;  
        }else if(key>item[mid]){ 8 > 8 → F  
            low=mid+1;  
        }else{  
            return mid;  
        }  
    }  
  
    return -1;  
}
```



Encontrou o elemento na posição 3 do vetor !

Análise do Algoritmo de Pesquisa Binária

- No início da primeira iteração, o vetor tem tamanho n . No início da segunda iteração, vale aproximadamente $\frac{n}{2}$. No início da terceira, $\frac{n}{4}$. No início da $(k + 1)$ -ésima, $\frac{n}{2^k}$.
- Quando k passar de $\log_2 n$, o valor da expressão $\frac{n}{2^k}$ fica menor que **1** e nesse momento o algoritmo finaliza a execução.
- Para entender o valor k encontrado anteriormente, considere que em algum momento o vetor atingirá o tamanho **1**, que ocorre quando $\frac{n}{2^k}$, ou seja:

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

$$n = 2^k$$

$$\log_2 n = \log_2 2^k$$


$$\log_2 n = k$$


- O consumo de tempo da busca binária é, portanto, proporcional a $\log_2 n$.

Obrigado pela atenção!!!
douglas.braz@ifsuldeminas.edu.br





Acesse o portal


 ASCENCIO, A.; CAMPOS, E. de. *Fundamentos da programação de computadores: algoritmos, Pascal, C/C++ e Java*. Pearson Prentice Hall, 2008. ISBN 9788576051480. Disponível em: <<https://books.google.com.br/books?id=p-mTPgAACAAJ>>.


 C: A Reference Manual. Pearson Education, 2007. ISBN 9788131714409. Disponível em: <<https://books.google.com.br/books?id=Wt2NEypdGNIC>>.


 DAMAS, L. *LINGUAGEM C*. LTC. ISBN 9788521615194. Disponível em: <<https://books.google.com.br/books?id=22-vPgAACAAJ>>.

 FEOFILOFF, P. *Algoritmos Em Linguagem C*. CAMPUS - RJ, 2009. ISBN 9788535232493. Disponível em: <<http://books.google.com.br/books?id=LfUQai78VQgC>>.

 KERNIGHAN, B.; RITCHIE, D. *C: a linguagem de programação padrão ANSI*. Campus, 1989. ISBN 9788570015860. Disponível em: <<https://books.google.com.br/books?id=aVWrQwAACAAJ>>.

 LOPES, A.; GARCIA, G. *Introdução à programação: 500 algoritmos resolvidos*. Campus, 2002. ISBN 9788535210194. Disponível em: <<https://books.google.com.br/books?id=Rd-LPgAACAAJ>>.

 MIZRAHI, V. *Treinamento em linguagem C*. Pearson Prentice Hall, 2008. ISBN 9788576051916. Disponível em: <<https://books.google.com.br/books?id=7xt7PgAACAAJ>>.

 SCHILDT, H.; MAYER, R. *C completo e total*. Makron, 1997. ISBN 9788534605953. Disponível em: <<https://books.google.com.br/books?id=Pbl0AAAACAAJ>>.