Buscas em estruturas lineares

Definições e busca binária

Prof. Marcelo de Souza

45EST – Algoritmos e Estruturas de Dados
Universidade do Estado de Santa Catarina

UDESC

Material de apoio

Leitura principal:

- Capítulo 5 de Goodrich et al. (2014)[1] – Recursão.

Leitura complementar:

- Capítulo 5 de Ziviani (2010)[2] – Pesquisa em memória binária.
- Capítulo 13 de Pereira (2008)[3] – Ordenação e busca.

[1] Michael T Goodrich et al. (2014). *Data structures and algorithms in Java*. 6ª ed. John Wiley & Sons.
[2] Nivio Ziviani (2010). *Projeto de Algoritmos com Implementa'ções em Java e C++*. Cengage Learning.
[3] Silvio do Lago Pereira (2008). *Estruturas de Dados Fundamentais: Conceitos e Aplicações*.

Buscas em estruturas lineares

Conceitos básicos

Buscar um elemento consiste em verificar se ele está armazenado em uma estrutura de dados.

Buscas em estruturas lineares

Conceitos básicos

Buscar um elemento consiste em verificar se ele está armazenado em uma estrutura de dados.

O retorno pode ser:

1. o próprio elemento;
2. a posição onde ele se encontra (-1, caso não seja encontrado); ou
3. um valor lógico indicando o sucesso ou a falha da busca.

Buscas em estruturas lineares

Conceitos básicos

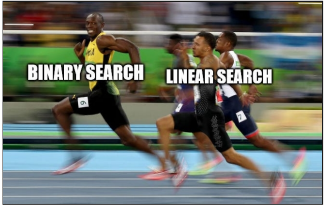
Buscar um elemento consiste em verificar se ele está armazenado em uma estrutura de dados.

O retorno pode ser:

1. o próprio elemento;
2. a posição onde ele se encontra (-1, caso não seja encontrado); ou
3. um valor lógico indicando o sucesso ou a falha da busca.

Estratégias:

- Busca sequencial (ou linear) – $\mathcal{O}(n)$.
- Busca binária – $\mathcal{O}(\log n)$.



Busca sequencial

É a forma mais simples de busca: percorre a estrutura até encontrar o elemento.

- Logo, sua complexidade assintótica é $\mathcal{O}(n)$ no pior caso.

Busca sequencial

É a forma mais simples de busca: percorre a estrutura até encontrar o elemento.

- Logo, sua complexidade assintótica é $\mathcal{O}(n)$ no pior caso.

Uma busca sequencial simples em um *array* de inteiros:

```
1 public int search(int[] array, int value) {
2   for(int i = 0; i < array.length; i++)
3     if(array[i] == value)
4       return i;
5   return -1;
6 }
```

Busca sequencial

É a forma mais simples de busca: percorre a estrutura até encontrar o elemento.

- Logo, sua complexidade assintótica é $\mathcal{O}(n)$ no pior caso.

Uma busca sequencial simples em um *array* de inteiros:

```
1 public int search(int[] array, int value) {
2   for(int i = 0; i < array.length; i++)
3     if(array[i] == value)
4       return i;
5   return -1;
6 }
```

Uma busca sequencial simples em uma lista de strings:

```
1 public int search(List<String> array, String value) {
2   for(int i = 0; i < array.size(); i++)
3     if(array.get(i).equals(value))
4       return i;
5   return -1;
6 }
```

Busca sequencial

Para implementar uma busca sequencial genérica, devemos usar um comparador.

- Se o tipo genérico implementa a interface `Comparable`, podemos usar um comparador genérico (`DefaultComparator`).
- Caso contrário, devemos implementar um comparador específico.

Busca sequencial

Para implementar uma busca sequencial genérica, devemos usar um comparador.

- Se o tipo genérico implementa a interface `Comparable`, podemos usar um comparador genérico (`DefaultComparator`).
- Caso contrário, devemos implementar um comparador específico.

Classe que implementa a busca para um tipo genérico `E`:

```
1 public class GenericSequentialSearch<E> {
2   Comparator<E> comp;
3
4   public GenericSequentialSearch() { this(new DefaultComparator<E>()); }
5
6   public GenericSequentialSearch(Comparator<E> c) { comp = c; }
7
8   // [...] Métodos de busca
9 }
```

Busca sequencial

Busca sequencial genérica para um *array*:

```
1 public int indexOf(E[] array, E value) {
2   for(int i = 0; i < array.length; i++)
3     if(comp.compare(array[i], value) == 0)
4       return i;
5   return -1;
6 }
```

Busca sequencial

Busca sequencial genérica para um *array*:

```
1 public int indexOf(E[] array, E value) {
2   for(int i = 0; i < array.length; i++)
3     if(comp.compare(array[i], value) == 0)
4       return i;
5   return -1;
6 }
```

Busca sequencial genérica para uma lista:

```
1 public int indexOf(List<E> array, E value) {
2   for(int i = 0; i < array.size(); i++)
3     if(comp.compare(array.get(i), value) == 0)
4       return i;
5   return -1;
6 }
```

Busca binária

A **busca binária** é uma técnica mais eficiente de busca em um **array ordenado**.

- Sua complexidade assintótica é $\mathcal{O}(\log n)$ no pior caso.

Busca binária

A **busca binária** é uma técnica mais eficiente de busca em um **array ordenado**.

- Sua complexidade assintótica é $\mathcal{O}(\log n)$ no pior caso.

Pré-condições:

- Os elementos devem estar **ordenados para o algoritmo funcionar**.
- A estrutura deve permitir **acesso aleatório** (*arrays*) **para garantir a eficiência**.

Busca binária

A **busca binária** é uma técnica mais eficiente de busca em um **array ordenado**.

- Sua complexidade assintótica é $\mathcal{O}(\log n)$ no pior caso.

Pré-condições:

- Os elementos devem estar **ordenados para o algoritmo funcionar**.
- A estrutura deve permitir **acesso aleatório** (*arrays*) **para garantir a eficiência**.

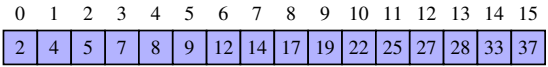
Funcionamento:

1. Avalia o elemento central da lista.
2. Caso seja o elemento buscado, sucesso.
3. Caso contrário, avalia em qual sub-lista o elemento pode estar.
4. Repete a busca com a sub-lista correspondente.

Busca binária

Exemplo de funcionamento

Busca binária do elemento 22 no array:

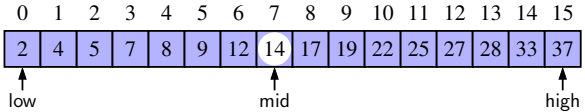


Busca binária

Exemplo de funcionamento

Busca binária do elemento 22 no array:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	4	5	7	8	9	12	14	17	19	22	25	27	28	33	37

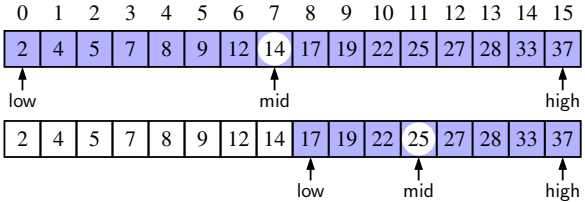


Busca binária

Exemplo de funcionamento

Busca binária do elemento 22 no array:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	4	5	7	8	9	12	14	17	19	22	25	27	28	33	37

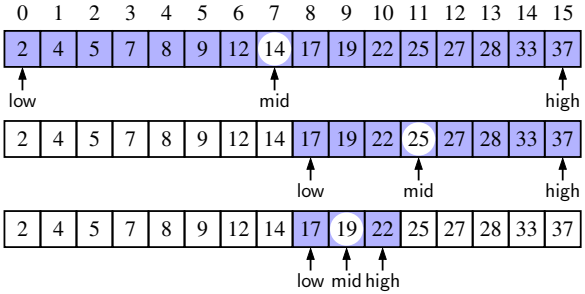


Busca binária

Exemplo de funcionamento

Busca binária do elemento 22 no array:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	4	5	7	8	9	12	14	17	19	22	25	27	28	33	37

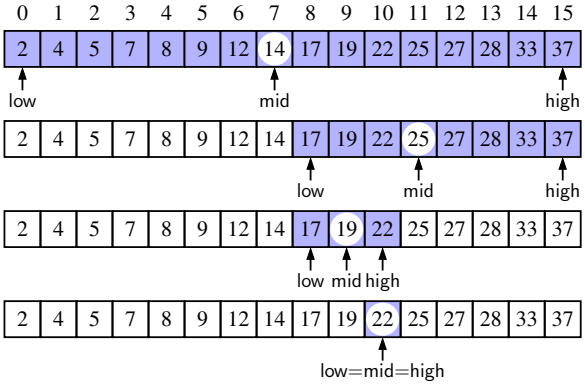


Busca binária

Exemplo de funcionamento

Busca binária do elemento 22 no array:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	4	5	7	8	9	12	14	17	19	22	25	27	28	33	37

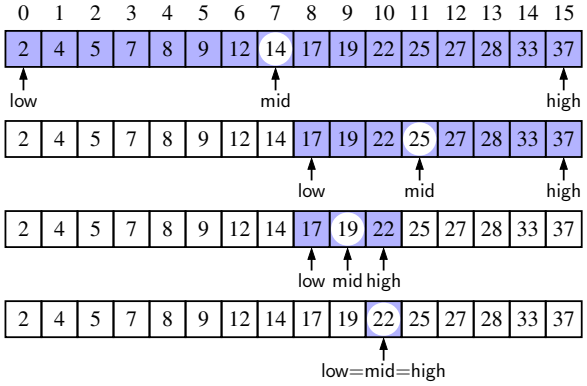


Busca binária

Exemplo de funcionamento

Busca binária do elemento 22 no array:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	4	5	7	8	9	12	14	17	19	22	25	27	28	33	37



Conclusões:

- Encontra o elemento em 4 avaliações: (14, 25, 19, 22).
- Note que $4 = \log_2 16$ (pior caso).
- Ao buscar o elemento 23, o algoritmo identifica sua inexistência quando `high < low`.

Busca binária

Uma busca binária simples em um *array* de inteiros:

```java
public int search(int[] array, int value) {
  int start = 0;
  int end = array.length - 1;
  int mid;

  do {
    mid = (start + end) / 2;
    if(array[mid] < value)
      start = mid + 1;
    else
      end = mid - 1;
  } while(array[mid] != value && start <= end);

  if(array[mid] == value)
    return mid;
  else
    return -1;
}
```

Busca binária

Uma busca binária simples em uma lista de strings:

```
1  public int search(List<String> list, String value) {
2    int start = 0, end = list.size() - 1, mid;
3
4    do {
5      mid = (start + end) / 2;
6      if(list.get(mid).compareTo(value) < 0)
7        start = mid + 1;
8      else
9        end = mid - 1;
10   } while(list.get(mid).compareTo(value) != 0 && start <= end);
11
12   if(list.get(mid).compareTo(value) == 0)
13     return mid;
14   else return -1;
15 }
```

Busca binária

Uma busca binária simples em uma lista de strings:

```java
public int search(List<String> list, String value) {
  int start = 0, end = list.size() - 1, mid;

  do {
    mid = (start + end) / 2;
    if(list.get(mid).compareTo(value) < 0)
      start = mid + 1;
    else
      end = mid - 1;
  } while(list.get(mid).compareTo(value) != 0 && start <= end);

  if(list.get(mid).compareTo(value) == 0)
    return mid;
  else return -1;
}
```

Note que essa implementação aceita que a lista seja encadeada, o que implica na operação `array.get(i)` executar em tempo linear, perdendo a complexidade $\mathcal{O}(\log n)$ no pior caso.

Busca binária

Uma busca binária genérica para um *array* (usando um comparador):

```
1  public int indexOf(E[] array, E value) {
2    int start = 0;
3    int end = array.length - 1;
4    int mid;
5
6    do {
7      mid = (start + end) / 2;
8      if(comp.compare(array[mid], value) < 0)
9        start = mid + 1;
10     else
11       end = mid - 1;
12   } while(comp.compare(array[mid], value) != 0 && start <= end);
13
14   if(comp.compare(array[mid], value) == 0)
15     return mid;
16   else
17     return -1;
18 }
```

Busca binária

Uma busca binária genérica para uma lista (usando um comparador):

```
1  public int indexOf(List<E> array, E value) {
2    int start = 0, end = array.size() - 1, mid;
3
4    do {
5      mid = (start + end) / 2;
6      if(comp.compare(array.get(mid), value) < 0)
7        start = mid + 1;
8      else
9        end = mid - 1;
10   } while(comp.compare(array.get(mid), value) != 0 && start <= end);
11
12   if(comp.compare(array.get(mid), value) == 0)
13     return mid;
14   else return -1;
15 }
```

Busca binária

Uma busca binária genérica para uma lista (usando um comparador):

```
1  public int indexOf(List<E> array, E value) {
2    int start = 0, end = array.size() - 1, mid;
3
4    do {
5      mid = (start + end) / 2;
6      if(comp.compare(array.get(mid), value) < 0)
7        start = mid + 1;
8      else
9        end = mid - 1;
10   } while(comp.compare(array.get(mid), value) != 0 && start <= end);
11
12   if(comp.compare(array.get(mid), value) == 0)
13     return mid;
14   else return -1;
15 }
```

Note que essa implementação aceita que a lista seja encadeada, o que implica na operação `array.get(i)` executar em tempo linear, perdendo a complexidade $\mathcal{O}(\log n)$ no pior caso.

45EST – Algoritmos e Estruturas de Dados
Prof. Marcelo de Souza