Samsung SW Certificate

Lição 02: *Arrays*

SDET

SIDIA

Agenda

- Geração de Subconjuntos
- Busca Não-Ordenada
  - Busca Sequencial
- Busca Ordenada
  - Busca Sequencial
  - Busca Binária
- Algoritmo de Seleção
- Ordenação
  - *Selection Sort*
  - Comparação
- Dicas
- Problemas

Geração de Subconjuntos

- Existem problemas nos quais é necessário a análise de todos os subconjuntos do conjunto original, *i.e.*, todos os conjuntos que podem ser formados a partir do original.

Geração de Subconjuntos

```
void generateSubsets(int array[]) {
  int i, j;
  int n = array.length;
  for (i = 0; i < (1 << (n)); i++) {
    for (j = 0; j < n; j++) {
      if ((i & (1 << j)) != 0)
          System.out.print(array[j] + " ");
    }
    System.out.println();
  }
}
```

Complexidade

$O(n2^n)$, onde n representa o tamanho do *array*.

Busca Não-Ordenada

- Quando os dados não estão ordenados, é necessário visitar todos os elementos do *array*.

Busca Sequencial

```
int search(int[] array, int key) {
  int i;
  int n = array.length;
  for (i = 0; i < n; i++) {
    if (array[i] == key)
      return i;
  }
  return -1;
}
```

Complexidade

$O(n)$, onde n representa o tamanho do *array*.

Nota

Não é mais vantajoso ordenar o *array* e depois realizar uma busca binária nele, pois o processo de ordenação leva o custo para pelo menos $O(n \log(N))$.

Busca Ordenada

- Quando os dados estão ordenados, existem formas de melhorar a busca.

Busca Sequencial

```
int search(int[] array, int key) {
  int i;
  int n = array.length;
  for (i = 0; i < n; i++) {
    if (array[i] > key)
      break;
    if (array[i] == key)
      return i;
  }
  return -1;
}
```

Complexidade

$O(n)$, onde n representa o tamanho do *array*.

Busca Binária

- Um método para determinar a posição da próxima busca por meio de comparação com o valor da chave do item no centro dos dados e proceder com a busca.
  - Repetindo a busca binária circularmente até que uma chave alvo seja encontrada, a busca pode ser realizada mais rapidamente ao modo que o limite de busca é reduzido pela metade.
- Os dados precisam estar ordenados para se conduzir a busca binária.

Busca Binária

1. Selecionar o elemento no centro dos dados.
2. Comparar o valor do elemento central e o valor a ser buscado.
3. Se o valor a ser buscado for menor que o central, realizar novas buscas na metade da esquerda dos dados. Se for maior, realizar novas buscas na parte direita dos dados.
4. Repetir os processos $1 - 3$ até que o valor desejado seja encontrado.

Busca Binária

```
boolean binarySearch(int[] array, int key) {
  int start = 0;
  int end = array.length - 1;
  while (start <= end) {
    int middle = start + ((end - start) / 2);
    if (array[middle] == key)
      return true;
    else if (array[middle] > key)
      end = middle - 1;
    else
      start = middle + 1;
  }
  return false;
}
```

Complexidade

$O(\log(n))$, onde n representa o tamanho do *array*.

Algoritmo de Seleção

- Algoritmo de seleção refere-se a um método para buscar por um elemento o qual é o K-ésimo maior ou menor dentre os dados.
  - Também representa um algoritmo para encontrar valores de mínimo, máximo ou intermediário.
- Processo de seleção
  - Seleção é feita por meio do processo abaixo:
    1. Ordenação dos dados utilizando um algoritmo de ordenação.
    2. Trazer elementos para a ordem desejada.

Algoritmo de Seleção

```
void selection(int[] array, int k) {
  int n = array.length;
  for (int i = 0; i < k; i++) {
    int minIdx = i;
    int minVal = array[i];
    for (int j = i + 1; j < n; j++) {
      if (array[j] < minValue) {
        minIdx = j;
        minVal = array[j];
      }
    }
    int temp = array[index];
    array[index] = array[i];
    array[i] = temp;
  }
  return array[k - 1];
}
```

Selection Sort

```
void selectionSort(int[] array) {
  int n = array.length;
  for (int i = 0; i < n - 1; i++) {
    int index = i;
    for (int j = i + 1; j < n; j++) {
      if (array[j] < array[index])
        index = j;
    }
    int smallerNumber = array[index];
    array[index] = array[i];
    array[i] = smallerNumber;
  }
}
```

Complexidade

$O(n^2)$, onde n representa o tamanho do *array*.

Comparação

Algoritmo	Tempo Médio	Pior Caso	Técnica	Considerações
Bubble Sort	$O(n^2)$	$O(n^2)$	Comparação e troca	Simples de codificar
Counting Sort	$O(n+k)$	$O(n+k)$	Método de contagem	Apenas quando n for muito pequeno
Selection Sort	$O(n^2)$	$O(n^2)$	Comparação e troca	Baixa frequência de trocas

Tabela 1: Comparação entre algoritmos de ordenação

Problemas

- Nos PDFs.