Samsung SW Certificate

Lição 01: *Arrays*

SDET

SIDIA

Agenda

- Notação Assintótica
- *Arrays*
  - *Arrays* Unidimensionais
  - *Arrays* Multidimensionais
- Busca
  - Busca Exaustiva
  - Permutações
  - Abordagem Gulosa
- Ordenação
  - *Bubble Sort*
  - *Couting Sort*
  - Comparação
- Dicas
- Problemas

Notação Assintótica

- Denota a ordem de crescimento de funções, estimando o tempo de execução no seu pior caso.
- Basicamente, interessa-se nos termos de maior ordem sem constantes, já que são esses termos que ditarão o comportamento da função.

Exemplo

$f(n) = 3n^2 + \cancel{7n} + \cancel{100} = 3n^2 = O(n^2)$

$f(n) = n^3 + \cancel{10000000000n} = O(n^3)$

Notação Assintótica

n	$\log(n)$	n	$n\log(n)$	n^2	n^3	2^n
10	$0,003\mu$s	$0,01\mu$s	$0,033\mu$s	$0,1\mu$s	1μs	1μs
20	$0,004\mu$s	$0,02\mu$s	$0,086\mu$s	$0,4\mu$s	8μs	1 ms
30	$0,005\mu$s	$0,03\mu$s	$0,147\mu$s	$0,9\mu$s	27μs	1 s
40	$0,005\mu$s	$0,04\mu$s	$0,213\mu$s	$1,6\mu$s	64μs	18,3 min
50	$0,006\mu$s	$0,05\mu$s	$0,282\mu$s	$2,5\mu$s	125μs	13 dias
10^2	$0,007\mu$s	$0,10\mu$s	$0,664\mu$s	10μs	1 ms	4×10^{13} anos
10^3	$0,010\mu$s	$1,00\mu$s	$9,966\mu$s	1 ms	16,7 min	
10^4	$0,013\mu$s	10μs	130μs	100 ms	11,6*dias*	
10^5	$0,017\mu$s	0,10 ms	1,67 ms	10 s	31,7 dias	
10^6	$0,020\mu$s	1 ms	19,93 ms	16,7 min	31.709 anos	
10^7	$0,023\mu$s	0,01 s	0,23 s	1,16 dia	$3,17 \times 10^7$ anos	
10^8	$0,027\mu$s	0,10 s	2,66 s	115,7 dias		
10^9	$0,030\mu$s	1 s	29,90 s	31,7 anos		

Tabela 1: Tempos aproximados de execução para diferentes tamanhos de entrada

Arrays

- O que é um *array*?
  - Uma estrutura de dados que usa variáveis de determinado tipo de dados listando-as com um único nome
- Quando usar um *array*?
  - Quando uma quantidade relativa de variáveis é necessária no programa, é muito ineficiente acessar dados usando nomes diferentes de variáveis um a um.
  - Usando um *array*, mais de 2 variáveis podem ser declaradas por meio de uma única declaração.
  - Isso não significa simplesmente a declaração de múltiplas variáveis, mas possibilita trabalhos difíceis, que geralmente não são facilmente manipulados com múltiplas variáveis, utilizando *arrays*.

Arrays unidimensionais

```
int array[] = new int[N];
array[0] = 7;
array[index] = 17;
```

Complexidade

- Inserção não-ordenada: $O(1)$
- Inserção ordenada: $O(n)$
- Busca linear: $O(n)$
- Busca binária: $O(\log(N))$

Arrays multidimensionais

```
int array[][] = new int[N][N];
array[0][1] = 7;
array[row][col] = 17;
```

Complexidade

- Inserção não-ordenada: $O(1)$
- Busca: $O(n^2)$ (pode ser melhorada em determinadas situações)

Busca Exaustiva

- O método da busca exaustiva é uma técnica que lista e verifica todos os possíveis casos que podem ser considerados como uma solução do problema.
- Também é chamada de técnica de 'força bruta'.
- Após testar todos os possíveis casos, encontre a solução final.
- Geralmente, ela é útil quando o número de casos é relativamente pequeno.

Busca Exaustiva

- Como o método gera e testa todos os possíveis casos, apesar da performance ser baixa, a probabilidade de encontrar uma solução é alta.
- Quando se resolve um problema em um exame de qualificação, é aconselhável abordá-lo com a busca exaustiva primeiro para encontrar a resposta e então usar outro algoritmo para melhorar a performance e verificar a resposta.

Permutações

- Muitas das vezes, todos os casos possíveis são gerados por meio da permutação dos possíveis valores.

Permutações

```
for (int i = 1; i <= 3; i++) {
    for (int j = 1; j <= 3; j++) {
        if (j != i) {
            for (int k = 1; k <= 3; k++) {
                if (k != i && k != j) {
                    System.out.print(i);
                    System.out.print(j);
                    System.out.print(k);
                    System.out.println();
                }
            }
        }
    }
}
```

Complexidade

$O(n^n)$, onde n representa o maior número da lista.

Abordagem Gulosa

- Greedy Algorithm is a shortsighted method used to find an optimal solution.
- Every time you have to decide one from several cases, reach a final solution by selecting one that is considered optimal at the moment.
- Although the decision you make at every time of selection is locally optimal, it is not guaranteed that it is optimal even though the final solution is reached by continuously collecting the selections.
- Generally, if you realize an idea which immediately comes into your head without verification, it becomes a Greedy Approach.

Abordagem Gulosa

1. Escolher uma solução: Após encontrar uma solução ótima para uma parte da questão na situação atual, adicioná-la ao conjunto de soluções parciais.
2. Examinar a viabilidade: Verificar se o novo conjunto de soluções parciais é viável e verificar se ele viola a condição de restrição da questão imediatamente.
3. Verificar a solução: Confirmar se o novo conjunto de soluções parciais se torna uma solução da questão. Se a solução da questão inteira não está completa, reiniciar o processo do passo 1.

Exemplo de Solução com a Abordagem Gulosa

- Redução de troco
  - "Como é possível reduzir o número de notas e moedas dadas a clientes como troco?"

Exemplo de Solução com a Abordagem Gulosa

1. Escolher uma solução: Selecionar a melhor solução aqui sem muitas considerações. Se o troco é formado por moedas com grandes unidades, o número de moedas diminui. Logo, seleciona-se a moeda com a maior unidade no momento e ela é adicionada ao troco.

Exemplo de Solução com a Abordagem Gulosa

2. Examinar a viabilidade: Verificar se o troco excede o valor a ser pago pelo cliente. Caso exceda, retirar a última moeda inserida no troco e retornar ao procedimento 1, e adicionar a moeda que é um nível menor que o da moeda atual.

Exemplo de Solução com a Abordagem Gulosa

3 Verificar a solução: A solução para o problema do troco é que o troco deve ser de fato equivalente ao total a ser pago ao consumidor. Não deve ser pago nem mais nem menos que o que deve ser pago, e, logo, se o total da solução é menor que o total após checar o troco, retornar ao passo 1 e selecionar uma moeda a ser inserida no troco.

Ordenação

- Às vezes é necessário ordenar *arrays* para compor uma solução.
- Existem uma variedade de algoritmos de ordenação, com diferentes peculiaridades e aplicações.

Bubble Sort

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

Complexidade

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

Counting Sort

```
int[] countingSort(int[] a, int k) {
  int c[] = new int[k];
  for (int i = 0; i < a.length; i++)
    c[a[i]]++;
  for (int i = 1; i < k; i++)
    c[i] += c[i - 1];
  int b[] = new int[a.length];
  for (int i = a.length - 1; i >= 0; i--)
    b[--c[a[i]]] = a[i];
  return b;
}
```

Complexidade

$O(n + k)$, onde n representa o tamanho do *array* e k o maior elemento do mesmo.

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

Tabela 2: Comparação entre algoritmos de ordenação

Dicas

- A exceção *ArrayOutOfBounds* é uma das mais comuns quando se trabalha com *arrays*.
  - Ela diz basicamente que ocorreu um acesso a uma posição inválida do *array*.

Dicas

```
int array[] = new int[3];
array[0] = 1;
array[1] = 2;
array[2] = 3;
array[3] = 4; // ArrayOutOfBounds
```

Dicas

```
int array[] = new int[3];
int index;
Scanner scanner = new Scanner();
index = scanner.nextInt();
if (index >= 0 && index < array.length)
  array[index] = 7;
else
  System.out.println("ArrayOutOfBounds");
scanner.close();
```

Nota

Sempre realizar a verificação para assegurar que a posição em questão é de fato válida. Uma posição i é valida se e somente se $0 \leq i \leq n$, onde n representa o tamanho do *array*.

Dicas

```
int array[][] = new int[3][3];
int x, y;
Scanner scanner = new Scanner();
x = scanner.nextInt();
y = scanner.nextInt();
if (x >= 0 && x < array.length &&
    y >= 0 && y < array.length)
  array[x][y] = 7;
else
  System.out.println("ArrayOutOfBounds");
scanner.close();
```

Problemas

- Nos PDFs.