**1256 – HASH TABLE**

1. import java.io.BufferedReader; // import BufferedReader class

2. import java.io.IOException; // import IOException class

3. import java.io.InputStreamReader; // import InputStreamReader class

4. import java.io.PrintWriter; // import PrintWriter class

5. import java.util.Arrays; // import Arrays class

6.

7. public class HashTable { // HashTable class declaration

8.

9.     static BufferedReader inputReader = new BufferedReader(new InputStreamReader(System.in)); // instantiate

10.                                                                                                // BufferedReader

11.                                                                                                // object

12.     static PrintWriter outputWriter = new PrintWriter(System.out); // instantiate PrintWriter object

13.

14.     public static void main(String[] args) throws IOException { // main method declaration

15.         int testCases = readInt(); // read number of test cases

16.

17.         while (testCases-- > 0) { // loop through test cases

18.             int[] parameters = readIntArray(); // read parameters

19.             int size = parameters[0]; // read size

20.             String[] hashTable = new String[size]; // instantiate hash table

21.             Arrays.fill(hashTable, ""); // fill hash table with empty strings

22.

23.             String[] elements = readLine().split("\\s"); // read elements

24.             for (String element : elements) { // loop through elements

25.                 int index = toInt(element) % size; // calculate index

26.                 hashTable[index] += " -> " + element; // add element to hash table

27.             } // end of loop through elements

28.

29.             for (int i = 0; i < size; i++) { // loop through hash table

30.                 outputWriter.println(i + hashTable[i] + " -> \\"); // print hash table

31.             } // end of loop through hash table

32.

33.             if (testCases != 0) { // if it is not the last test case

34.                 outputWriter.println(); // print blank line

35.             } // end of if

36.         } // end of loop through test cases

37.

38.         outputWriter.close();

39.     } // end of main method

40.

41.     private static String readLine() throws IOException {

42.         return inputReader.readLine();

43.     }

44.

45.     private static int readInt() throws IOException {

46.         return Integer.parseInt(inputReader.readLine());

47.     }

48.

49.     private static int[] readIntArray() throws IOException {

50.         String[] line = inputReader.readLine().split("\\s");

51.         int length = line.length;

52.         int[] array = new int[length];

53.         for (int i = 0; i < length; i++) {

54.             array[i] = Integer.parseInt(line[i]);

55.         }

56.         return array;

57.     }

58.

59.     private static int toInt(String s) {

60.         return Integer.parseInt(s);

61.     }

62. } // end of HashTable class

63.

**EXPLICAÇÃO DO ALGORITMO**

### Declaração de Variáveis e Entrada

1. Importação de Classes:

- As classes necessárias são importadas, como BufferedReader, IOException, InputStreamReader, PrintWriter e Arrays.

2. Declaração da Classe:

- Uma classe chamada HashTable é declarada.

3. Declaração de Variáveis Estáticas:

- Um objeto BufferedReader chamado inputReader é criado para ler a entrada do usuário.

- Um objeto PrintWriter chamado outputWriter é criado para imprimir a saída.

### Método main

4. Método main:

- O ponto de entrada do programa. É onde a execução do programa começa.

5. Leitura do Número de Casos de Teste:

- int testCases = readInt();

- Lê o número de casos de teste.

6. Loop Principal (Por Cada Caso de Teste):

- while (testCases-- > 0) {

- Inicia um loop que será executado para cada caso de teste.

7. Leitura dos Parâmetros:

- int[] parameters = readIntArray();

- Lê os parâmetros do caso de teste, onde parameters[0] é o tamanho da tabela e parameters[1] é a quantidade de chaves.

8. Inicialização da Tabela Hash:

- String[] hashTable = new String[size];

- Inicializa a tabela hash como um array de strings.

9. Preenchimento da Tabela Hash:

- Arrays.fill(hashTable, "");

- Preenche a tabela hash com strings vazias.

10. Leitura e Inserção das Chaves na Tabela Hash:

- String[] elements = readLine().split("\\s");

- Lê as chaves como uma linha e as divide em um array de strings.

- Para cada chave, calcula o índice usando a função de dispersão toInt(element) % size e insere a chave na tabela hash.

11. Saída da Tabela Hash:

- outputWriter.println(i + hashTable[i] + " -> \\");

- Imprime cada linha da tabela hash.

12. Verificação de Casos de Teste:

- if (testCases != 0) { outputWriter.println(); }

- Se não for o último caso de teste, i–mprime uma linha em branco.

13. Fechamento do outputWriter:

- outputWriter.close();

- Fecha o objeto PrintWriter.

### Métodos Auxiliares

14. Métodos Auxiliares:

- readLine(), readInt(), readIntArray(), e toInt(String s) são métodos auxiliares para facilitar a leitura de entrada.

### Funcionamento Geral

O programa lê o número de casos de teste e, para cada caso, lê os parâmetros, inicializa a tabela hash, insere as chaves usando a função de dispersão, imprime a tabela hash e repete o processo para cada caso de teste.

### Exemplo de Execução

Para o exemplo de entrada com 2 casos de teste:

1. Caso de Teste 1:

- Tamanho da tabela: 13

- Chaves: 44, 45, 49, 70, 27, 73, 92, 97, 95

- A tabela hash é impressa após o cálculo dos índices para cada chave.

2. Caso de Teste 2:

- Tamanho da tabela: 7

- Chaves: 35, 12, 2, 17, 19, 51, 88, 86

- A tabela hash é impressa após o cálculo dos índices para cada chave.

O programa continua lendo e processando casos de teste até que todos sejam tratados. A saída é formatada de acordo com as especificações fornecidas.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

EXPLICAÇÃO DE FUNCIONAMENTO DE ALGORITMOS

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Vamos analisar o exemplo fornecido com dois casos de teste:

1. \*\*Entrada:\*\*

2. ```

3. 2

4. 13 9

5. 44 45 49 70 27 73 92 97 95

6. 7 8

7. 35 12 2 17 19 51 88 86

8. ```

9.

1. \*\*Saída Esperada:\*\*

2. ```

3. 0 -> \

4. 1 -> 27 -> 92 -> \

5. 2 -> \

6. 3 -> \

7. 4 -> 95 -> \

8. 5 -> 44 -> 70 -> \

9. 6 -> 45 -> 97 -> \

10. 7 -> \

11. 8 -> 73 -> \

12. 9 -> \

13. 10 -> 49 -> \

14. 11 -> \

15. 12 -> \

16.

17. 0 -> 35 -> \

18. 1 -> \

19. 2 -> 2 -> 51 -> 86 -> \

20. 3 -> 17 -> \

21. 4 -> 88 -> \

22. 5 -> 12 -> 19 -> \

23. 6 -> \

24. ```

25.

Explicação:

Caso de Teste 1:

- Número de testes (N): 2

- Primeiro teste (M = 13, C = 9):

- Tamanho da tabela (M): 13

- Quantidade de chaves (C): 9

- Chaves: 44, 45, 49, 70, 27, 73, 92, 97, 95

- A função de dispersão é h(x) = x mod 13.

- As chaves são inseridas na tabela hash usando o índice calculado pela função de dispersão.

- O tratamento de colisões é feito por encadeamento externo, onde as colisões são resolvidas mantendo uma lista encadeada de elementos no mesmo índice.

- A saída mostra a tabela hash resultante.

Caso de Teste 2:

- Segundo teste (M = 7, C = 8):

- Tamanho da tabela (M): 7

- Quantidade de chaves (C): 8

- Chaves: 35, 12, 2, 17, 19, 51, 88, 86

- A função de dispersão é h(x) = x mod 7.

- As chaves são inseridas na tabela hash usando o índice calculado pela função de dispersão.

- O tratamento de colisões é feito por encadeamento externo.

- A saída mostra a tabela hash resultante.

Os índices na saída representam os endereços-base na tabela hash, e cada índice é seguido pelos elementos associados a esse índice. O caractere "->" é utilizado para separar os elementos na mesma posição da tabela. A barra invertida ("\") indica o final da lista encadeada para cada índice.

Esta implementação da tabela hash resolve colisões usando encadeamento externo e é capaz de lidar com diferentes tamanhos de tabela e quantidades de chaves em cada caso de teste. O código é modular, com funções para leitura de entrada, conversão de tipos e implementação da lógica da tabela hash.