**Practical – 02**

**Aim :**

Write a program to implement Huffman Encoding using a greedy strategy.

**Program:**

import java.util.PriorityQueue;

import java.util.HashMap;

import java.util.Map;

class HuffmanNode implements Comparable<HuffmanNode> {

char data;

int freq;

HuffmanNode left, right;

public HuffmanNode(char data, int freq) {

this.data = data;

this.freq = freq;

left = right = null;

}

@Override

public int compareTo(HuffmanNode other) {

return this.freq - other.freq;

}

}

public class HuffmanCoding {

static void printHuffmanCodes(HuffmanNode root, String code) {

if (root == null) {

return;

}

if (root.data != '$') {

System.out.println(root.data + ": " + code);

}

printHuffmanCodes(root.left, code + "0");

printHuffmanCodes(root.right, code + "1");

}

public static void buildHuffmanTree(char[] data, int[] freq, int size) {

PriorityQueue<HuffmanNode> minHeap = new PriorityQueue<>();

for (int i = 0; i < size; i++) {

minHeap.add(new HuffmanNode(data[i], freq[i]));

}

while (minHeap.size() > 1) {

HuffmanNode left = minHeap.poll();

HuffmanNode right = minHeap.poll();

HuffmanNode internalNode = new HuffmanNode('$', left.freq + right.freq);

internalNode.left = left;

internalNode.right = right;

minHeap.add(internalNode);

}

HuffmanNode root = minHeap.poll();

printHuffmanCodes(root, "");

}

public static void main(String[] args) {

char[] data = {'A', 'B', 'C', 'D'};

int[] freq = {23, 12, 34, 10};

int size = data.length;

buildHuffmanTree(data, freq, size);

}

}

/\*

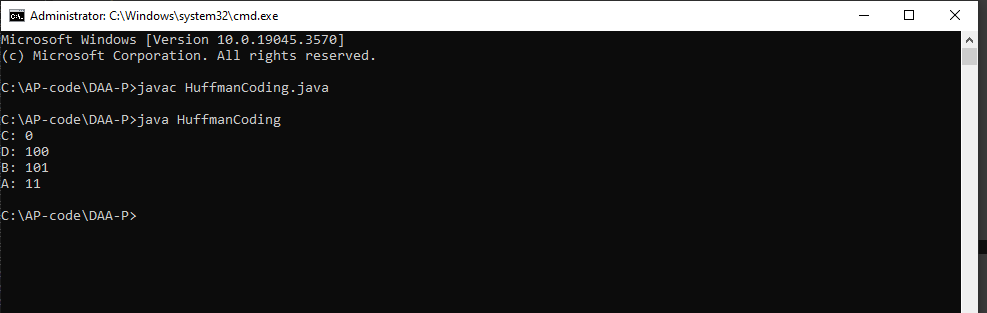
Huffman Coding :

Time complexity: O(nlogn) where n is the number of unique characters.

If there are n nodes, extractMin() is called 2\*(n – 1) times. extractMin() takes O(logn) time as it calls minHeapify(). So, overall complexity is O(nlogn).

\*/

**Output:**

****