



---

# REPORT

---

Huffman coding



교 과 목	알고리즘
담 당 교 수	우진운 교수님
학 과	소프트웨어학과
학 번	32141868
이 름	박유현
제 출 일	2019.05.07



## 1. 소스코드

다음의 소스코드는 Greedy method가 적용된 Huffman coding기법을 Java로 구현한 것이다. 구현 과정에서 java.util.HashMap, java.util.List, java.util.ArrayList, java.util.PriorityQueue 자료구조를 사용하였고 Tree는 직접 구현하여 사용하였다. 그리고 정형화된 코드인 getter, setter 메서드는 축약하여 나타내었다.

### 1.1 Node.java

```
package tree;

public class Node {

    private String key;      //저장된 문자
    private Integer cnt;    //문자의 갯수
    private Node left;
    private Node right;

    public Node(String key) {
        this.key = key;
        left = null;
        right = null;
        cnt = 1;
    }

    public Node getLeft() {...}
    public void setLeft(Node left) {...}
    public Node getRight() {...}
    public void setRight(Node right) {...}
    public String getKey() {...}
    public Integer getCnt() {...}

    public void addCnt() {
        cnt++;
    }
}
```

## 1.2 Tree.java (1/2)

```
package tree;

import java.util.HashMap;

public class Tree implements Comparable<Tree> {
    private Node root;
    private int weight;    //서브트리의 노드들의 cnt 총합
    private HashMap<String, String> codeMap;

    public Tree() {
        root = null;
        weight = 0;
    }

    public Node getRoot() {...}
    public void setRoot(Node root) {...}
    public int getWeight() {...}
    public void setWeight(int weight) {...}
    public HashMap<String, String> getCodeMap() {...}

    /**
     * 자기자신 Tree와 입력받은 Tree를 각각 left, right child로 갖는 새로운 Tree를 생성하여 반환
     * 새롭게 생성된 Tree의 root Node는 key값이 dummy인 dummy Node임
     * @param tree
     * @return
     */
    public Tree merge(Tree tree) {
        Tree res = new Tree();
        res.root = new Node("dummy");
        res.root.setLeft(root);
        res.root.setRight(tree.root);
        res.weight = weight + tree.weight;
        return res;
    }
}
```

## 1.2 Tree.java (2/2)

```
/**
 * Huffman code를 계산하여 Tree.codeMap에 저장
 * 재귀 호출 함수인 calcCodeRecur를 사용
 */
public void calcCodeList() {
    codeMap = new HashMap<>();
    calcCodeRecur(root, "");
}

/**
 * calcCodeList의 재귀호출함수
 * tree의 왼쪽자식으로 갈때는 0, 오른쪽자식으로 갈때는 1을 추가한다.
 * @param node
 * @param resStr
 */
private void calcCodeRecur(Node node, String resStr) {
    if (node.getLeft() != null)
        calcCodeRecur(node.getLeft(), resStr+"0");
    if (node.getRight() != null)
        calcCodeRecur(node.getRight(), resStr+"1");
    if (!node.getKey().equals("dummy"))
        codeMap.put(node.getKey(), resStr);
}

@Override
public int compareTo(Tree o) {
    if (weight > o.weight)
        return 1;
    else if (weight == o.weight)
        return 0;
    else
        return -1;
}
}
```

### 1.3 HuffmanResultElement.java

```
package tree;

import java.util.HashMap;

public class HuffmanResultElement {

    private String input;
    private String output;
    private HashMap<String, String> codeMap;

    public HuffmanResultElement(String input, String output, HashMap<String, String> codeMap) {
        this.input = input;
        this.output = output;
        this.codeMap = codeMap;
    }

    public String getInput() {...}
    public void setInput(String input) {...}
    public String getOutput() {...}
    public void setOutput(String output) {...}
    public HashMap<String, String> getCodeMap() {...}
    public void setCodeMap(HashMap<String, String> codeMap) {...}
}
```

## 1.4 Main.java (1/2)

```
package main;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.PriorityQueue;
import tree.HuffmanResultElement;
import tree.Node;
import tree.Tree;

public class Main {
    public static void main(String[] args) {
        File file = new File("E:/text2.txt");
        String text = readAll(file);
        HuffmanResultElement res = encodeHuffman(text);
        System.out.println(text);
        System.out.println(res.getCodeMap());
        System.out.println(res.getOutput());
        System.out.println("length : " + res.getOutput().length());
    }

    public static String readAll(File file) {
        //file의 내용을 읽어 1개의 String으로 반환, 줄바꿈은 공백으로 처리함
        String result = "";
        FileReader fr = null;
        BufferedReader br = null;
        try {
            fr = new FileReader(file);
            br = new BufferedReader(fr);
            while (true) {
                String temp = br.readLine();
                if (temp == null) {
                    result = result.substring(0, result.length()-1);
                    break;
                }
                else
                    result += temp + " ";
            }
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                br.close();
                fr.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        return result;
    }
}
```

## 1.4 Main.java (2/2)

```
/**
 * 입력받은 문자열을 Huffman coding을 통해 변환하여 반환
 * @param str
 * @return
 */
public static HuffmanResultElement encodeHuffman(String str) {
    //입력 문자열을 확인하여 각각의 문자가 몇 개 있는지 확인하여 Node로 HashMap에 저장
    HashMap<String, Node> map = new HashMap<>();
    for (int i = 0; i < str.length(); i++) {
        String key = Character.toString(str.charAt(i));
        if (map.get(key) == null)
            map.put(key, new Node(key));
        else
            map.get(key).addCnt();
    }

    //HashMap에 저장된 Node들을 Node가 1개인 Tree로 만들어 우선순위큐에 저장
    //우선순위는 Tree 클래스에 구현된 Comparable을 통해 정해짐
    List<Node> list = new ArrayList<>();
    list.addAll(map.values());
    PriorityQueue<Tree> pq = new PriorityQueue<>();
    for (int i = 0; i < list.size(); i++) {
        Tree tempTree = new Tree();
        tempTree.setRoot(list.get(i));
        tempTree.setWeight(list.get(i).getCnt());
        pq.add(tempTree);
    }

    //우선순위 큐에서 가장 weight가 작은 Tree를 두 개씩 꺼내 병합
    Tree resTree = null;
    while (true) {
        Tree temp1 = pq.poll();
        Tree temp2 = pq.poll();
        if (temp2 == null)
            break;
        resTree = temp1.merge(temp2);
        pq.add(resTree);
    }

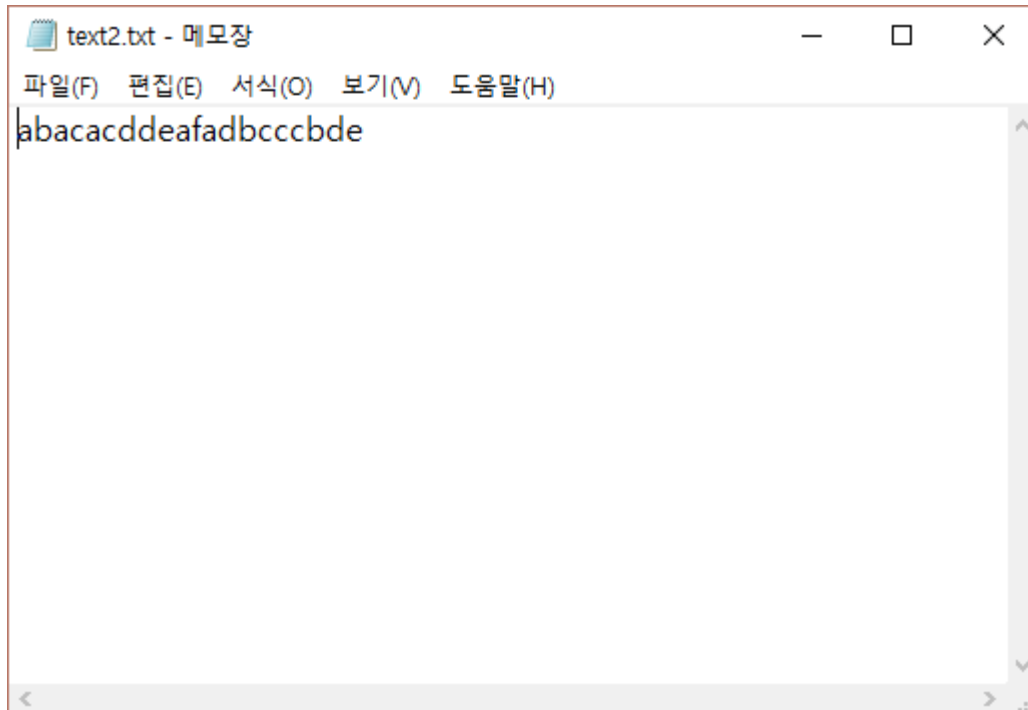
    //완성된 Tree를 통해 Huffman code table 작성 후 HashMap에 저장
    resTree.calcCodeList();
    HashMap<String, String> resMap = resTree.getCodeMap();

    //입력된 문자열을 code table을 통해 변환 (실제로는 binary로 저장되나 출력을 위해 문자열로 저장)
    String outputStr = "";
    for (int i = 0; i < str.length(); i++)
        outputStr += resMap.get(Character.toString(str.charAt(i)));

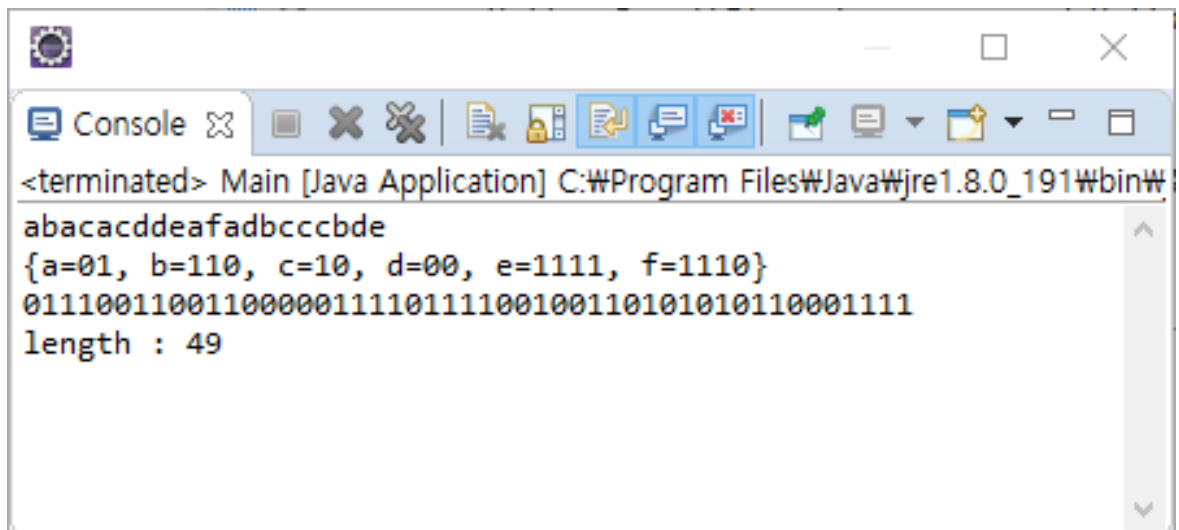
    HuffmanResultElement res = new HuffmanResultElement(str, outputStr, resMap);
    return res;
}
```

## 2. 결과

### 2.1.1 테스트 입력 파일

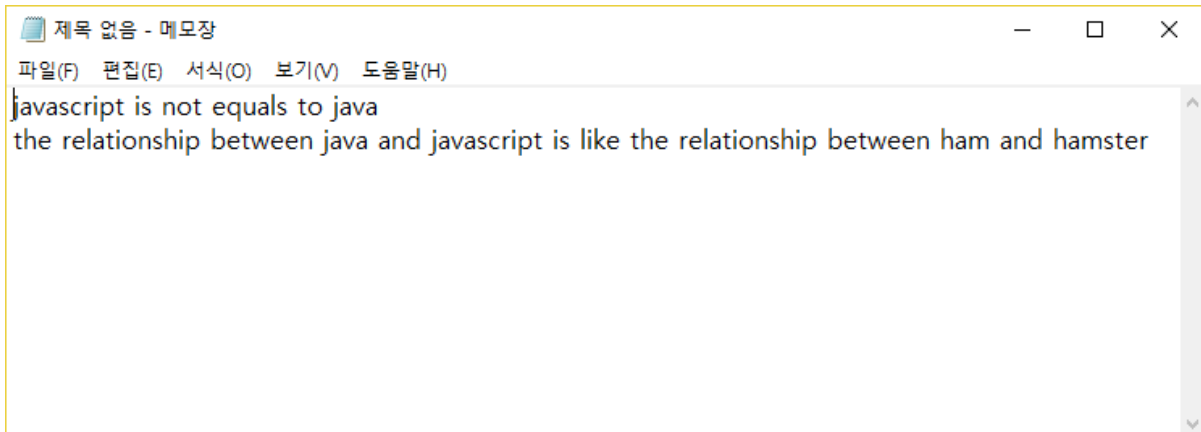


### 2.1.2 테스트 결과



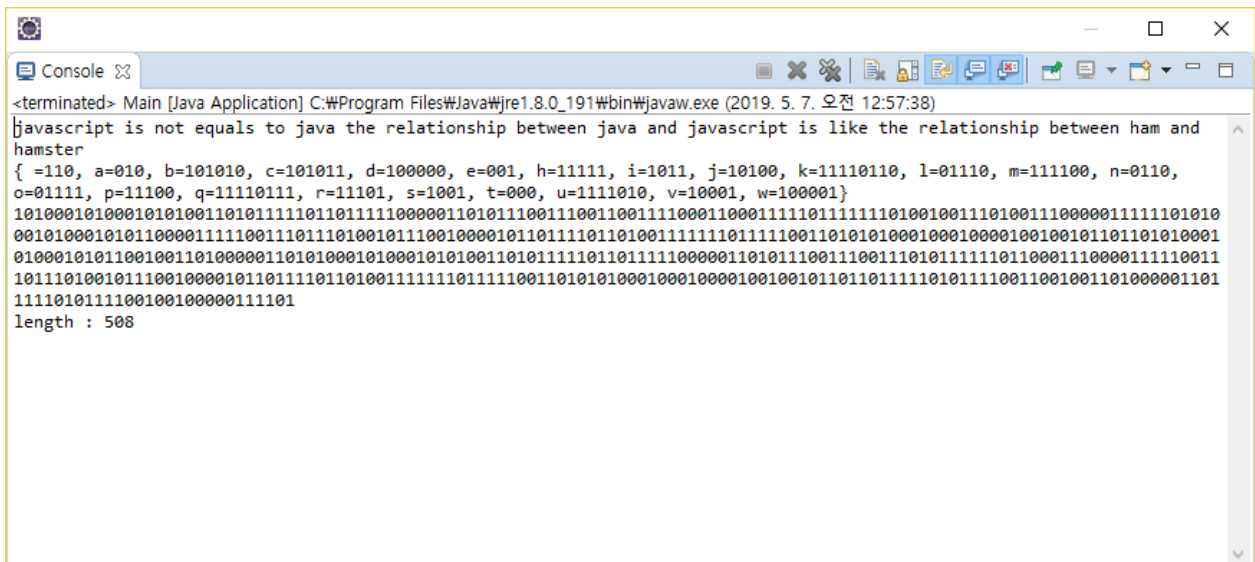


## 2.2.1 테스트 입력 파일



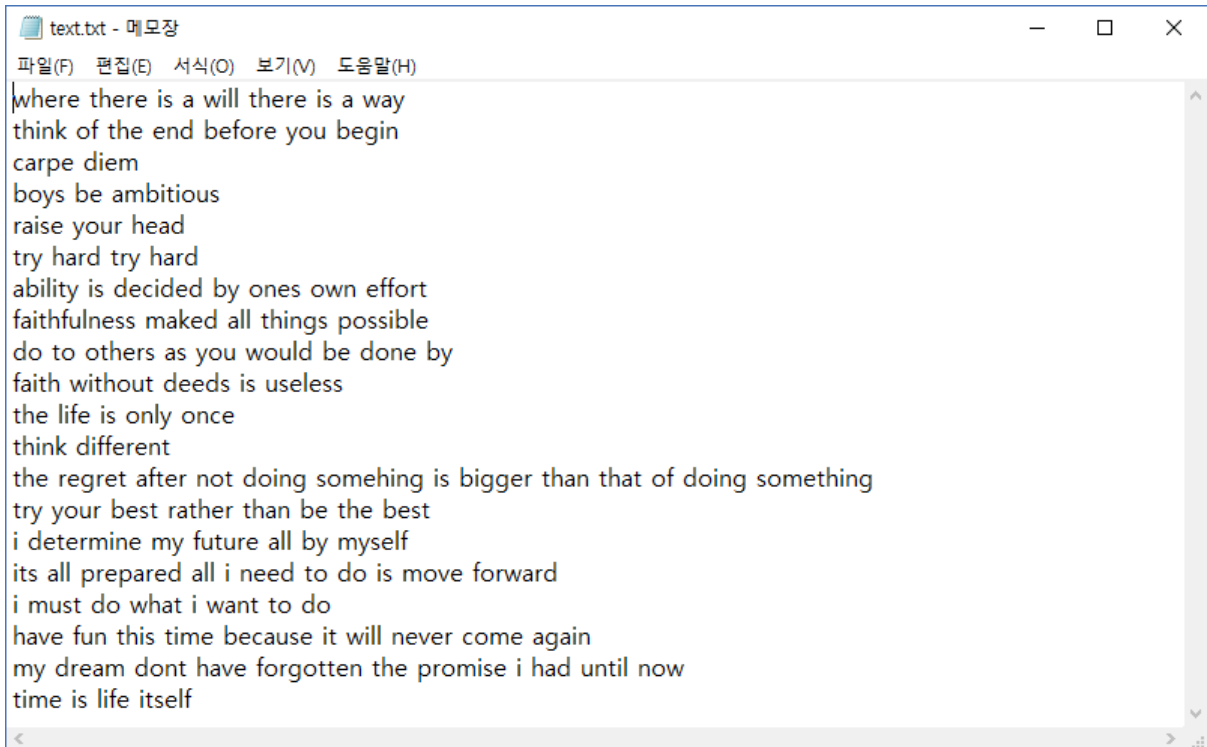
```
제목 없음 - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
javascript is not equals to java
the relationship between java and javascript is like the relationship between ham and hamster
```

## 2.2.2 테스트 결과



```
<terminated> Main [Java Application] C:\Program Files\Java\jre1.8.0_191\bin\javaw.exe (2019. 5. 7. 오전 12:57:38)
javascript is not equals to java the relationship between java and javascript is like the relationship between ham and hamster
{ =110, a=010, b=101010, c=101011, d=100000, e=001, h=11111, i=1011, j=10100, k=11110110, l=01110, m=111100, n=0110,
o=01111, p=11100, q=11110111, r=11101, s=1001, t=000, u=1111010, v=10001, w=100001}
1010001010001010001101011111011011111000001101011100111001100111100011000111110111111010010011101001111101010
00101000101011000011111001110110100101110010000101101110110100111111011110011010100010001000100100101101101010001
01000101011001001101000001101010001010001010011010111101101111000001101011100111001110101111101100011110000111110011
10111010010111001000010110111101101111101111001101010100010001000100100101101101111101011110011001101000001101
1111010111100100000111101
length : 508
```

### 2.3.1 테스트 입력 파일



### 2.3.2 테스트 결과

