실습보고서

2020. 11. 22 (일)

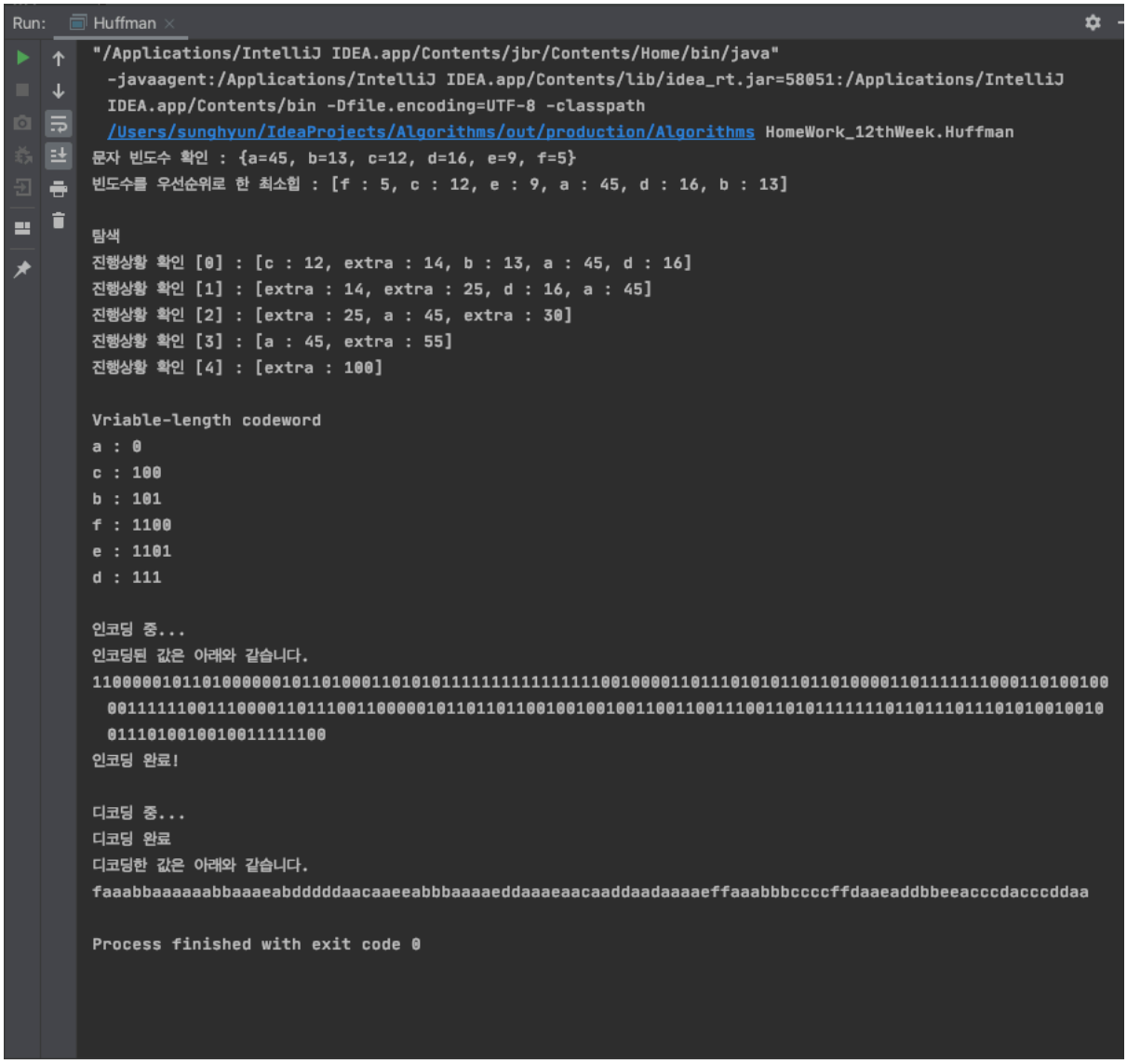
[알고리즘 00분반]_12주차
201501513_김성현

조건사항
운영체제 : MAC OS
IDE : IntelliJ
라이브러리 : io(입출력), util(코드설계), nio(파일읽기)

Error 사항) 파일을 불러오고 파일을 생성할 때 상대경로로 "./"를 주어서 같은 패키지 내가 아닌 같은 디렉토리에서 파일을 읽어오고 파일을 생성하게 됩니다.

1. data12_huffman.txt 파일을 읽고, Greedy algorithm을 사용하여 Huffman code를 얻기 위한 Tree를 구축하라. 그리고 얻어진 Tree로부터 Huffman code 값들을 얻고 input data를 encode/table data로 인코딩하여 출력하라. 반대로 encoded data와 table data로부터 input data를 복원하는 디코딩 과정을 추가하라.

<실행 결과>



<코드 설명>

```
System.out.println("\n탐색");
for(int i = 0; i < size-1; i++){
    node left = que.poll();
    node right = que.poll();

    int total = left.frequency + right.frequency;

    node node = new node( name: "extra", total);
    node.left = left;
    node.right = right;

    que.offer(node);
    System.out.println("진행상황 확인 ["+i+"] : "+que);
}
```

Txt파일로 입력받은 문자열을 HashMap을 통하여 해당 문자의 빈도수가 얼마인지 체크하도록 하였습니다.

알고리즘면에서는 먼저 그리드 알고리즘을 사용하기 위해 위에서 진행상황 확인에 보이듯이 Priority Queue를 적용하여 빈도수에 따라 정렬되게 만들었으며 해당 노드들의 좌우 자식 노드를 생성하도록 하여 결과적으로 마지막 extra : 100이 남았을 때 해당 노드에 대해 트리가 형성되도록 하였습니다.

```
table = new HashMap<>();
System.out.println("\nVriable-length codeword");
print_codeword(que.poll(), code: "");
System.out.println();
```

그리고 마지막으로 남은 노드에 대해서 자식노드를 탐색해가며 variable-length codeword를 생성하도록 하였습니다. 그리고 그 코드들을 HashMap으로 table이라는 변수에 저장하도록 하였습니다.

```java
public void makeFile(String str) throws IOException {
    String path = "./";
    String fileName_encode = path+"201501513_encoded.txt";
    String fileName_table = path+"201501513_table.txt";

    String[] forEncode = str.split( regex: "");
    BufferedWriter bw = new BufferedWriter(new FileWriter(fileName_encod
    String encoding = "";
    for(int i = 0; i < forEncode.length; i++){
        encoding += table.get(forEncode[i]);
    }
    bw.write(encoding);
    System.out.println("인코딩된 값은 아래와 같습니다.\n"+encoding);
    bw.flush();

    bw = new BufferedWriter(new FileWriter(fileName_table,  append: true)
    Iterator it = table.keySet().iterator();
    while(it.hasNext()){
        String s = (String)it.next();
        bw.write( str: s+" "+table.get(s)+"\n");
    }

    bw.flush();
    bw.close();
}
```

그리고 makeFile이라는 메소드를 통해 현재 자바파일이 위치한 directory에 대해서 encoded된 파일과 table 파일을 BufferedWriter를 통해 생성되도록 하였습니다.

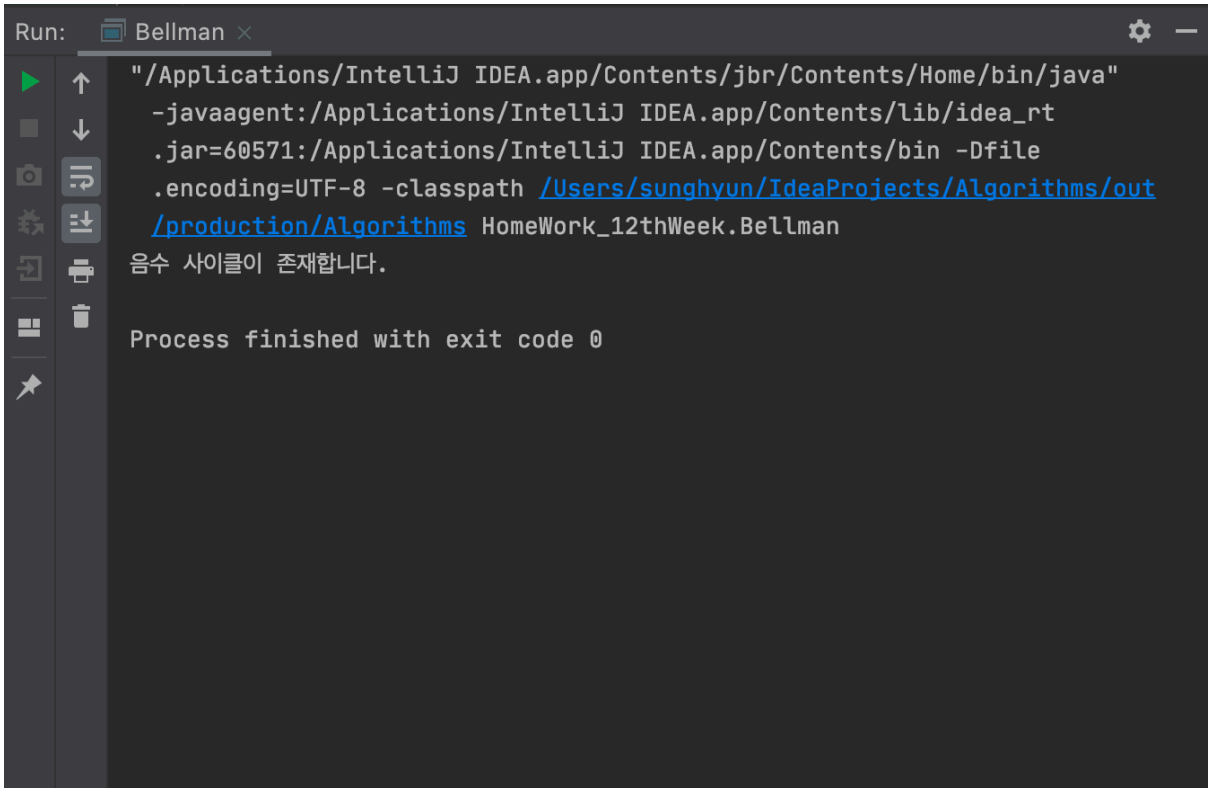
그리고 결과화면과 같이 인코딩 상황을 알려주고 인코딩된 값을 출력하도록 하였습니다.
또한 디코딩 역시 마찬가지로 상황을 알려주고 decoding이라는 메소드를 통해 파일을 읽고 해당 파일에 대한 값들에 따라 decode가 실행되도록 하였습니다.

```java
public void decoding(String encoded, HashMap<String, String> table, Str
    //debugging
    //System.out.println(encoded);
    //System.out.println(table);
    StringBuilder sb = new StringBuilder("");
    String str = "";
    for(int i = 0; i < encoded.length(); i++){
        str += encoded.charAt(i);
        //System.out.println(str);
        if(table.containsKey(str)){
            sb.append(table.get(str));
            str = "";
        }
    }

    System.out.println("디코딩 완료\n디코딩한 값은 아래와 같습니다.\n"+sb.toString(
    String fileName = path+"20150513_decode";
    BufferedWriter bw = new BufferedWriter(new FileWriter(fileName, app
    bw.write(sb.toString());

    bw.flush();
    bw.close();
}
```

디코드 방식은 선형시간을 유지하기 위해 HashMap을 사용하여 str이라는 변수에 하나씩 추가하며 해당 키값이 존재하면 value를 StringBuilder에 저장시키는 방식으로 진행하였으며 이또한 학번 + decode라는 txt파일로 저장되도록 만들었고, 결과화면에 출력되도록 하였습니다.

2. data12_bellman.txt를 읽어 Bellman-Ford Algorithm을 이용하여 s에서 t까지의 최단거리를 구하여라.

<실행 결과>



<코드 설명>

```
String path = "./";
List<String> list = Files.readAllLines(Paths.get( first: path+"data12_bel
int data = 3;
String[] input_data = new String[data];
String[] input_edge = new String[list.size()-data];
for(int i = 0; i < 3; i++){
    input_data[i] = list.get(i);
}
for(int i = data; i < list.size(); i++){
    input_edge[i-data] = list.get(i);
}

//print(input_data);
//print(input_edge);

bellman_solution bs = new bellman_solution();
bs.solution(input_data, input_edge);
```

Huffman에서와 마찬가지로 자바파일이 존재하는 디렉토리에서 파일을 불러오도록 하였으며 txt파일에서 위에 3개는 설정을 아래는 간선을 의미하는 것이기 때문에 구분을 지어 각기 다른 배열에

저장하도록 하였습니다.

```java
public void solution(String[] str1, String[] str2) {
    //노드 개수
    int length = Integer.parseInt(str1[0]);

    d = new int[length];
    edge = new int[length][length];

    //시작과 끝점
    String[] split_str1_1 = str1[1].split( regex: " ");
    int start = Integer.parseInt(split_str1_1[0]);
    int end = Integer.parseInt(split_str1_1[1]);
    for(int i = 0; i < d.length; i++){
        d[i] = Integer.MAX_VALUE;
    }
    d[start] = 0;

    //간선 개수
    int number = Integer.parseInt(str1[2]);

    ArrayList<vertex> list = new ArrayList<>();

    for(int i = 0; i < number; i++){
        String[] split = str2[i].split( regex: " ");
        int u = Integer.parseInt(split[0]);
        int v = Integer.parseInt(split[1]);
        int weight = Integer.parseInt(split[2]);

        edge[u][v] = weight;
        list.add(new vertex(u, v, weight));
    }
```

그리고 d를 dp할 배열로 지정하여 입력받은 노드의 개수랑 동일하게 지정해두고, edge는 해당 간선들이 올바르게 적용되었는지 확인하는 용도로 이차원배열을 선언해 저장하도록 하였습니다.

그리고 vertex들을 Arraylist를 사용해 추후 간선들에 대해 최단 거리를 찾기위해 반복하기 위해 저장되도록 하였습니다.

```
for(int i = 1; i < length; i++){
    for(int j = 0; j < number; j++){
        vertex vertex = list.get(j);
        int u = vertex.start;
        int v = vertex.end;
        int w = vertex.weight;
        test[u][v] = w;

        if(d[u] != Integer.MAX_VALUE && d[v] > d[u]+w){
            d[v] = d[u]+w;
            test[u][v] = d[u]+w;
        }
    }
}
```

그리고 시작 노드와 연결된 노드, 그리고 간선의 가중치를 각 간선들에 대해 탐색을 해가며 해당 값이 최소가 될때까지 노드의 개수-1 만큼 반복을 해주도록 하였습니다.

```
boolean isCycle = false;

for(int i = 0; i < number; i++){
    vertex vertex = list.get(i);
    int u = vertex.start;
    int v = vertex.end;
    int w = vertex.weight;
    if(d[u] != Integer.MAX_VALUE && d[v] > d[u]+w){
        isCycle = true;
    }
}
```

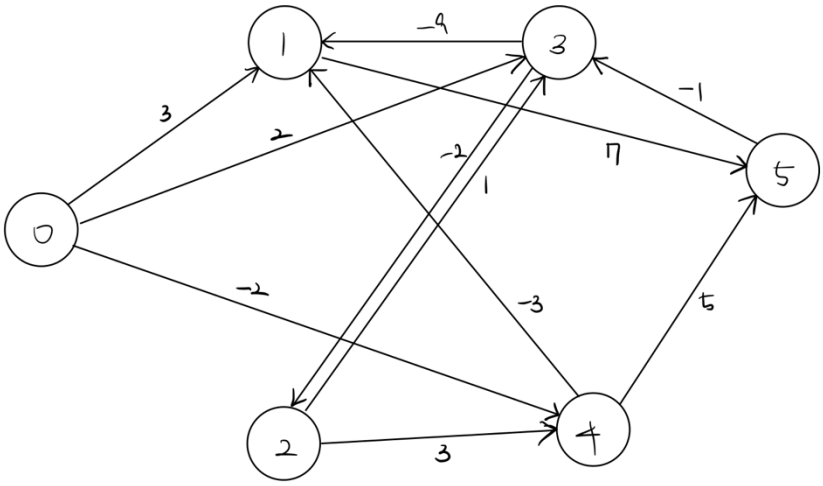
마지막으로는 음수 사이클이 존재하는지 확인하기 위해 해당 list에 대해서 반복을 진행하였을 때 최단거리의 값에 변화가 있으면 사이클이 존재하므로 이를 확인하기 위해 isCycle이라는 Boolean 변수를 두어 확인하도록 하였습니다.

```java
if(isCycle){
    System.out.println("음수 사이클이 존재합니다.");
}
else{
    System.out.println("최단 거리는 다음과 같습니다.");
    for(int i = 0; i < d.length; i++){
        System.out.print(d[i]+" ");
    }
    System.out.println();
}
```

그리고 출력을 할 때, 음수 사이클이 존재하면 음수사이클이 존재함을 말하고, 아닌 경우에는 최단 거리를 출력하도록 하였습니다.

<Path>

```
  d    1  2  3  4  5
------------------
  0 |  0  0  0  0  0
  1 | -7  0  2 -2  3
  2 | -8 -1 -1 -2  3
  3 |-10 -3 -2 -2 -1
  4 |-11 -4 -4 -2 -3
  5 |-13 -6 -5 -2 -4
음수 사이클이 존재합니다.
```



상단의 표는 파일 내에 test 배열을 출력한 것이며, 세로축이 t를 가르킵니다.