알고리즘

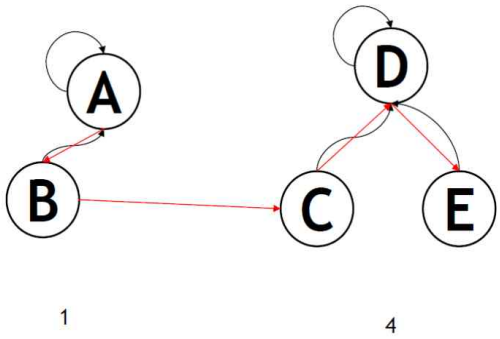
- 07. Union-Find -

제 출 일	2018.11.23
학 과	컴퓨터공학과
학 번	201402391
이 름	이 병 만

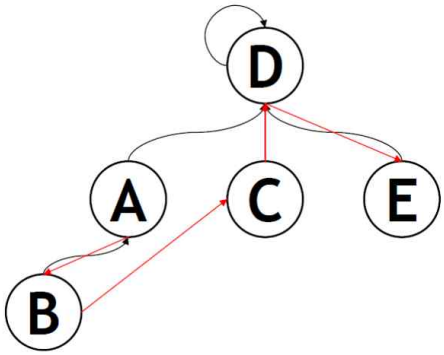
※ 과제 목표 및 해결 방법

- 1차 출력 (Output1.txt) – 각 노드가 가진 번호를 기준으로 자신과 부모의 ID 출력
- 2차 출력 (Output2.txt) – 모든 Tree를 번호 순서대로 Union 후 모든 노드의 자신과 부모의 ID 출력

Output1의 예시



Output2의 예시



※ 주요 부분 코드 설명 (알고리즘 부분 코드 캡쳐)

1. find-set

```
public static Node find_set(Node input) {
    if (input.parent.id.equals(input.id))
        return input;
    else
        return find_set(input.parent);
}
```

[그림1. find-set]

➔ 해당 input 노드의 부모 노드를 찾는 함수이다. input 값이 들어오면 부모의 값과 비교해서 같다면 input을 반환해주고 다르다면 부모 노드를 따라가면서 다음의 과정을 반복해준다.

```
public void union(Node S1, Node S2) {
    Node x = find_set(S1);
    Node y = find_set(S2);
    x.parent = y;
}
```

[그림2. union]

➔ 인자로 받은 두 노드를 합쳐주는 함수이다. Union 시킬 노드의 부모 노드를 반환받는다. 노드를 반환 받고서 x의 부모를 y의 값에 넣어준다.

```java
public String toString(String path) {
    try {
        String savePath = main.class.getResource("").getPath(); // Get a Absolute path
        File file = new File(savePath + path);
        FileWriter fw = new FileWriter(file);
        BufferedWriter bufWriter = new BufferedWriter(fw);

        if(head == null){
            bufWriter.write("[]");
            bufWriter.close();
            return "[]";
        }
        // 탐색을 시작합니다.
        Node temp = head;
        String str = "";
        // 다음 노드가 없을 때까지 반복문을 실행합니다.
        // 마지막 노드는 다음 노드가 없기 때문에 아래의 구문은 마지막 노드는 제외됩니다.
        String a[];
        while(temp != null){
            str = temp.id + "\t" + temp.parent.id+"\n";
            a = str.split("\n");
            bufWriter.write(a[0]);
            bufWriter.newLine();
            temp = temp.next;
        }
        bufWriter.close();
    } catch(Exception e) {
        System.out.println(e);
    }
    return path;
}
```

[그림3. toString]

➔ 다음 함수는 출력을 파일 형식으로 저장하기 위해서 사용한다. 우선 파일 경로를 찾고서 저장할 파일의 이름을 받는다. head가 null이라면 공백이라는 의미를 위해 '[ ]'을 사용한다. burwirter를 사용해서 str에 저장된 값을 받아서 한 줄씩 작성해준다. 파일에 다 작성하면 파일을 닫아주고 파일 이름을 반환해준다.

```java
public class main {
    public static void main(String[] args) throws IOException {
        LinkedList LinkedList = File_read.fileRead("Data.txt"); // File Read
        Node temp = LinkedList.head;

        while(temp != null) {
            LinkedList.union(temp, LinkedList.get(temp.num));
            temp = temp.next;
        }

        System.out.println(LinkedList.toString("Output1.txt") + " saved");

        temp = LinkedList.head;
        while(temp.next != null) {
            LinkedList.union(temp, temp.next);
            temp = temp.next;
        }

        System.out.println(LinkedList.toString("Output2.txt") + " saved");
    }
}
```

[그림4. main]

➔ 파일을 읽어서 LinkedList를 생성한다. temp 변수를 생성해서 LinkedList를 다룬다.
temp가 null이 아닐 때까지 반복하면서 해당 노드와 해당 노드가 가지고 있는 ID 번호 번째에 있는 노드를 찾아서 union을 한다. 반복문을 종료하면 Output1.txt를 생성한다.
다시 temp를 설정하고 해당 노드와 다음 노드를 union을 해준다. 반복문을 종료하면 Output2.txt를 생성한다.

실행 결과 및 분석

처음의 과제 자료를 봤을 때는 무슨 말인지 이해를 못했다. Union-find에 대해서 이해를 잘못하고 있었다. 처음에 생각했던 부분은 weigthted Union을 위해서 하는 방식과 동일하다고 생각했었는데 그것이 아니었다. Union-Find는 기준으로 정해준 노드로부터 붙여주기 때문에 한쪽으로 치우쳐지는 트리가 생길 수가 있다. 그렇기 때문에 다음과 같은 문제를 해결해 줄 수 있는 방법이 weigthted Union인 것이다. 이번 과제는 Union-Find 였기 때문에 앞에 있는 노드가 뒤에 있는 노드의 부모가 되게 한다.