

알고리즘

- 09. BFS, DFS -

제 출 일	2018.11.30
학 과	컴퓨터공학과
학 번	201402391
이 름	이 병 만

※ 과제 목표 및 해결 방법

- BFS 구현
- DFS 구현

※ 주요 부분 코드 설명 (알고리즘 부분 코드 캡처)

1. BFS

```
public class BFS {
    private static final Graph NIL = null;
    private Graph u;
    private Graph v;

    public BFS() {}

    public BFS(int[][] arr, int s_vertex, Graph[] g) {
        for (int i=0; i<arr.length; i++) {
            if (i != s_vertex-1)
                g[i] = new Graph("WHITE", Integer.MAX_VALUE, NIL, i+1);
        }

        g[s_vertex-1] = new Graph("GRAY", 0, NIL, s_vertex);
        Queue<Graph> que = new LinkedList<Graph>();
        que.add(g[s_vertex-1]);

        while (!que.isEmpty()) {
            u = que.poll();
            for (int i=0; i<arr.length; i++) {
                if (arr[u.getIndex()-1][i] == 1) {
                    v = g[i];
                    if (v.getColor() == "WHITE") {
                        v.setColor("GRAY");
                        v.setDistance(u.getDistance() + 1);
                        v.setPie(u);
                        que.add(v);
                    }
                }
            }
            u.setColor("BLACK");
        }
        BFS_print("Output_BFS.txt", g);
    }
}
```

→ BFS는 너비 우선 탐색으로 시작 정점으로부터 인접한 정점을 방문하면서 그래프를 탐색하는 것입니다.

Graph 변수를 선언한다.(u, v)

BFS 생성자를 위한 함수를 선언한다. 인자로 txt파일에서 읽어온 2차원 배열과 시작 정점, 그래프를 담는 배열을 받는다. 반복문을 실행하면서 시작 정점을 제외한 나머지 정점을 화이트로 색깔을 칠해준다. 시작 정점 큐에 넣으면 회색으로 칠한다. 반복문으로 큐가 비었는지 확인 후 비어있지 않으면 큐에서 원소 하나를 꺼낸다. 배열을 돌면서 u의 정점과 인접한 정점 v를 찾는다. 찾으면 v의 색깔이 화이트면 v의 색깔을 그레이로 바꾸고 거리를 1증가시키고 어느 정점에서 왔는지를 넣어주는 정점 u를 넣어주고 정점 v를 큐에 넣는다. 인접한 정점이 없으면 정점 u를 검은색으로 칠해준다.

```

public void BFS_print(String path, Graph[] g) {
    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);

        for(int i=0; i<g.length; i++) {
            bufWriter.write(g[i].getIndex()+" "+g[i].getDistance());
            System.out.println(g[i].getIndex()+" "+g[i].getDistance());
            bufWriter.newLine();
        }

        bufWriter.close();
    } catch(Exception e) {
        System.out.println(e);
    }
}

```

➔ BFS을 출력을 위한 함수이다. 출력한 것을 txt에 써줄 파일을 열어서 해당 인덱스와 탐색 횟수를 파일에 써준다. 모든 정점에 대한 탐색 횟수를 적으면 파일을 닫는다.

```

public class Graph {
    private static final Graph NIL = null;
    private String color;
    private int distance;
    private Graph pie = NIL;
    private int index;

    public Graph() {}

    public Graph(String color, int distance, Graph pie, int index) {
        this.color = color;
        this.distance = distance;
        this.pie = NIL;
        this.index = index;
    }
}

```

➔ 그래프가 가지는 속성들을 세팅해주는 클래스이다. 색, 거리 pie, index를 가지고 그래프 생성자를 위한 함수를 선언하였다.

```

public class main {
    public static void main(String[] args) throws IOException {
        Scanner sc = new Scanner(System.in);
        int arr[][] = File_read.fileRead("graph.txt");
        System.out.print("시작 정점 선택 : ");
        int start = sc.nextInt();

        Graph[] g = new Graph[arr.length];
        new BFS(arr, start, g);
        sc.close();
    }
}

```

→ 파일을 읽고 2차원 배열을 반환받고 시작 정점을 지정해준다, 지정한 정점으로부터 탐색을 시작한다.

※ 실행 결과 및 분석

```

<terminated> main (14) [Java]
0 1 0 0 1 0 0 0
1 0 0 0 0 1 0 0
0 0 0 1 0 1 1 0
0 0 1 0 0 0 0 1
1 0 0 0 0 0 0 0
0 1 1 0 0 0 1 0
0 0 1 0 0 1 0 1
0 0 0 1 0 0 1 0

시작 정점 선택 : 3
1 3
2 2
3 0
4 1
5 4
6 1
7 1
8 2

```

2. DFS

```
public class DFS {
    private static final Graph NIL = null;
    private Graph u;
    private Graph v;
    int time;;

    public DFS() {}

    public DFS(int[][] arr, Graph[] g) {
        for (int i=0; i<arr.length; i++) {
            g[i] = new Graph("WHITE", 0, NIL, 0, i+1);
        }

        time = 0;
        for (int i=0; i<arr.length; i++) {
            u = g[i];
            if (u.getColor() == "WHITE") {
                DFS_visit(g, u, arr);
            }
        }
        DFS_print("Output_DFS.txt",g);
    }
}
```

→ DFS는 깊이 우선 탐색으로 시작 정점으로부터 계속가다가 갈 수 없는 곳까지 간 뒤 다시 돌아오는 탐색과정이다
Graph 변수를 선언한다.(u, v) 방문한 시간을 측정하는 time 변수를 선언
DFS 생성자를 위한 함수를 선언한다. 인자로 txt파일에서 읽어온 2차원 배열과 시작 정점, 그래프를 담는 배열을 받는다. 반복문을 실행하면서 모든 정점을 화이트로 색깔을 칠해준다. time 변수를 초기화 해준다. 배열을 돌면서 u의 색깔이 화이트면 DFS_visit함수를 호출한다.

```
public void DFS_visit(Graph[] g, Graph u, int[][] arr) {
    time = time + 1;
    u.setDepart_time(time);
    u.setColor("GRAY");
    for (int i=0; i<arr.length; i++) {
        if (arr[u.getIndex()-1][i] == 1) {
            v = g[i];
            if (v.getColor() == "WHITE") {
                v.setPie(u);
                DFS_visit(g, v, arr);
            }
        }
    }
    u.setColor("BLACK");
    time++;
    u.setFinal_time(time);
}
```

→ DFS가 방문하는 과정을 나타내는 함수이다. 정점을 방문했으므로 시간을 +1해준고 u의 색깔을 그레이로 바꿔주고 시간을 넣어준다. 반복문을 돌면서 정점 u와 인접한 정점을 찾는다. 찾은 정점을 v라 한다. v의 색깔이 화이트면 이전 정점인 u를 pie의 값으로 넣어준다. 다시 정점 v로 DFS_visit함수를 호출한다. 반복문을 끝나면 정점 u의 색깔을 방문하였으므로 블랙으로 바꾼다. 그리고 time을 +1증가하고 u의 정점 빠져나온 시간으로 넣어준다.


```

public void DFS_print(String path, Graph[] g) {
    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);

        for(int i=0; i<g.length; i++) {
            bufWriter.write(g[i].getIndex()+" "+g[i].getDepart_time() + " " + g[i].getFinal_time());
            System.out.println(g[i].getIndex()+" "+g[i].getDepart_time() + " " + g[i].getFinal_time());
            bufWriter.newLine();
        }

        bufWriter.close();
    } catch(Exception e) {
        System.out.println(e);
    }
}

```

➔ 출력을 위한 함수이다. 출력한 것을 txt에 써줄 파일을 열어서 해당 인덱스와 도착한 시간과 빠져나온 시간을 파일에 써준다. 모든 정점에 대한 탐색 횟수를 적으면 파일을 닫는다.

```

public class Graph {
    private static final Graph NIL = null;
    private String color;
    private int depart_time;
    private Graph pie = NIL;
    private int final_time;
    private int index;

    public Graph() {}

    public Graph(String color, int depart_time, Graph pie, int final_time, int index) {
        this.color = color;
        this.depart_time = depart_time;
        this.pie = NIL;
        this.final_time = final_time;
        this.index = index;
    }
}

```

➔ 그래프가 가지는 속성들을 세팅해주는 클래스이다. 색, 도착 시간 pie, 빠져나온 시간, index를 가지고 그래프 생성자를 위한 함수를 선언하였다.

```

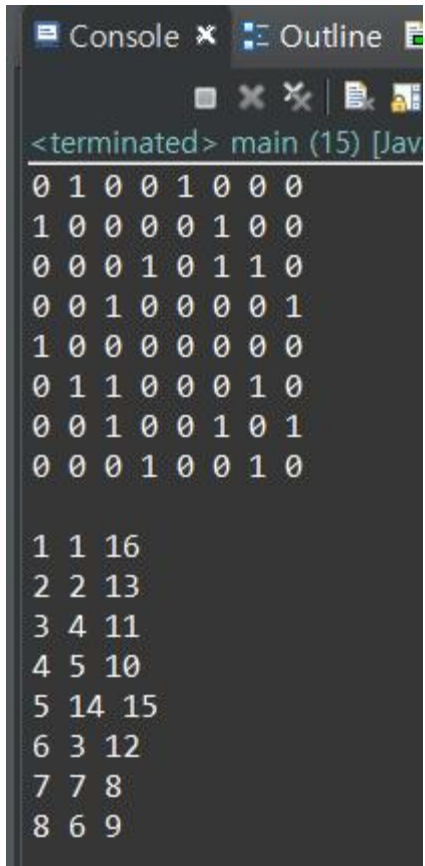
public class main {

    public static void main(String[] args) throws IOException {
        Scanner sc = new Scanner(System.in);
        int arr[][] = File_read.fileRead("graph.txt");
        Graph[] g = new Graph[arr.length];
        new DFS(arr, g);
        sc.close();
    }
}

```

➔ 파일을 읽고 2차원 배열을 반환받고 시작 정점을 지정해준다, DFS 탐색을 시작한다.]

※ 실행 결과 및 분석



```
<terminated> main (15) [Java]
0 1 0 0 1 0 0 0
1 0 0 0 0 1 0 0
0 0 0 1 0 1 1 0
0 0 1 0 0 0 0 1
1 0 0 0 0 0 0 0
0 1 1 0 0 0 1 0
0 0 1 0 0 1 0 1
0 0 0 1 0 0 1 0

1 1 16
2 2 13
3 4 11
4 5 10
5 14 15
6 3 12
7 7 8
8 6 9
```