

# 자료구조설계

제 출 일	2017.10.1.
과 제 번 호	04
분 반	01
학 과	컴퓨터공학과
학 번	201602038
이 름	이 미 진

## 1. 구현 내용 설명

```
8
A
B
C
D
E
F
G
H
A B 6
A E 1
A F 5
B C 1
B F 3
C D 2
C F 1
C G 2
C H 4
D H 1
E F 2
F G 2
G H 6
```

```
try {
    BufferedReader br = new BufferedReader(new FileReader(new File("C://input.txt")));
    int size = Integer.parseInt(br.readLine()); // readLine으로 파일 내용을 한 줄 읽어 size에 저장한다. 저장된 내용은 정점의 수.
    vertices = new String[size]; // vertices를 size만큼 생성

    for( int i=0; i<size; i++ ) { // input.txt 파일로부터 배열 생성
        vertices[i] = br.readLine(); // 한 줄씩 읽어들이 vertices 배열에 순서대로 저장한다.
    }

    catch (FileNotFoundException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

input.txt 파일에는 정점의 수, 정점의 나열, 간선과 가중치의 나열이 입력되어 있고, 이를 통해 그래프를 나타낼 수 있다.

try catch를 통해 예외처리가 가능하도록 하였고,

BufferedReader를 통해 로컬 디스크(C:)에 있는 input.txt 파일을 readLine()을 이용해 한 줄씩 파일 내용을 읽어온다.

size 변수에는 파일의 첫째줄에 있는 정점의 수가 저장되고, 그 이후로 for문을 통해 size만큼 파일 내용을 한 줄씩 vertices 배열에 저장한다.

## 1) dijkstra 알고리즘

```
public void dijkstra(int i) { // dijkstra 알고리즘 수행
    for(int j = 0; j < size; j++){
        // 초기화
        prevPath[j] = null;
        weight[j] = a[i][j];
        if( !(weight[j] == MAX_LENGTH) ) {
            // 시작 정점과 인접하게 연결되어있을 경우
            prevPath[j] = "0"; // 시작 정점으로 초기화
        }
        visit[i] = true; // 방문했으므로 true
        weight[i] = 0;

        for( int rep=1 ; rep < size ; rep++ ){
            int n;
            int s = shortDistance(weight, visit);

            for( n=0; n < size; n++ ){
                // 이전 거리보다 짧음, 방문X, 직접 연결, 시작 정점과 이어진 경로가 있음의 경우를
                // 모두 만족할 경우
                if( (a[s][n] + weight[s] < weight[n]) && !(visit[n] || a[s][n]
                == MAX_LENGTH || weight[s] == MAX_LENGTH) ) {
                    prevPath[n] = String.valueOf(s);
                    weight[n] = a[s][n] + weight[s];
                }
            }
            visit[s] = true; // 방문했으므로 true
        }
    }
}
```

dijkstra 메소드가 호출되면, size(args.length)만큼 prevPath

배열은 null로, weight 배열은 a배열로 초기화 시킨다.

그 과정에서, weight 배열의 j번째 인덱스가 MAX\_LENGTH가 아니면  
prevPath의 j번째 인덱스는 string 0으로 초기화 된다.

반복이 끝나면 방문했다고 체크를 하고, dijkstra() 메소드가 호출될 때 인자로  
받은 int i를 weight의 인덱스로 주고 0으로 초기화한다.( weight[0] = 0 )

shortDistance 메소드를 호출해 최단 거리 노드의 index를 s 변수에  
저장한다.

반복하면서 정점을 탐색하는데, 탐색하는 정점이 이전에 계산된 거리보다  
짧거나, 방문되지 않았거나, s와 n이 직접 연결되어 있거나, s까지 시작  
정점과 이어진 경로가 있다는 경우를 모두 만족하면, 이전 경로를 새로운  
value 값으로 바꾸고, weight 값을 변경한다.

## 2) 최단 거리 정점을 찾는 shortDistance 메소드

```
public int shortDistance(int[] weight, boolean[] visit) {  
    // 최단거리 정점을 찾는다.  
    int s = MAX_LENGTH;  
    int index = 0;  
    for( int i=0; i<size; i++ ) {  
        if( !(visit[i] || s < weight[i]) ) { // 방문하지 않았거나  
            MAX_LENGTH보다 weight[i]가 작을때  
                index = i;  
                s = weight[i];  
            }  
        }  
    }  
    return index; // index 반환  
}
```

인자로 weight 배열과 방문여부 visit 배열을 받는다.

size(args.length) 만큼 반복을 하면서 visit[i]가 false일 경우와

s(MAX\_LENGTH) 보다 weight[i]가 작을 경우,

index는 0로, s 는 weight[i] 로 변경된다. 반복이 종료되면 index 값을 반환한다.

## 3) 한 정점에서 다음 정점으로의 최단 경로 및 가중치를 출력하는 printPaths 메소드

```
public void printPaths(){// 한 정점에서 다음 정점으로의 최단 경로 및 가중치를 출력  
    Stack<Integer> stack = new Stack<Integer>(); // stack 객체 생성  
  
    for( int i=1; i < size; i++ ){  
        System.out.println(vertices[0]); // 시작 정점 출력  
  
        int pp = i; // 이전 경로를 i로 초기화  
        stack.add(pp);  
  
        for(;;) {  
            if( prevPath[pp] != "0" ) { // 이전 경로가 시작 정점이  
                stack.add(Integer.parseInt(prevPath[pp]));  
                pp = Integer.parseInt(prevPath[pp]);  
            }  
            else {  
                break;  
            }  
        }  
        for( ; !stack.isEmpty(); ) { // 스택이 비어있지 않을 경우  
            int sp = stack.pop(); // 스택에서 pop한 원소를 sp에  
            저장  
            System.out.print("-> " + vertices[sp]); // 경로 출력  
            System.out.println("(" + weight[sp] + ")"); // 가중치 출력  
        }  
    }  
}
```

stack을 이용하기 위해 stack 객체를 생성하였다.

size(args.length)만큼 시작 정점을 출력함과 동시에 경로와 가중치가

출력되도록 구현하였다.

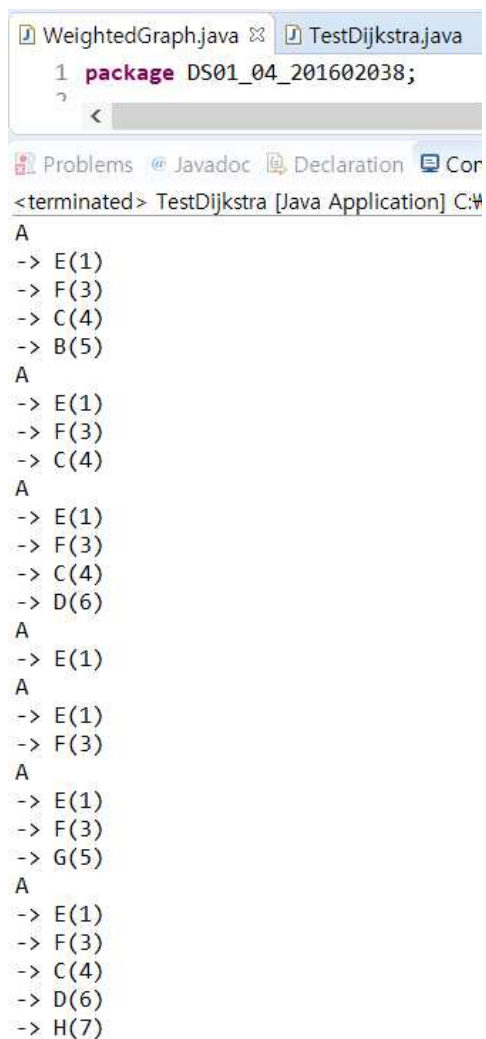
이전 경로 변수 pp를 1로 초기화 시킨 후 stack의 add를 이용해 pp를 스택에 삽입한다.

for문을 이용해 true일 경우 계속 반복하도록 하고, 이전 경로가 시작 정점이 아닐 경우

Integer.parseInt(prevPath[pp])를 stack에 삽입하고, pp를 다시 이전 경로로 하여 탐색을 계속 하도록 한다. 이전 경로가 시작 정점이 되면 반복을 종료한다.

반복이 종료된 후엔 stack이 empty가 아닐때까지 stack에서 pop을 통해 원소를 꺼내 sp 변수에 저장한다. 그리고 vertices배열과 weight배열에서 sp값으로 각각 경로와 가중치를 출력한다.

## 2. 실행 결과 화면



The screenshot shows an IDE with two tabs: 'WeightedGraph.java' and 'TestDijkstra.java'. The 'TestDijkstra.java' tab is active, displaying the following code:

```
1 package DS01_04_201602038;  
2
```

Below the code editor, there are tabs for 'Problems', 'Javadoc', 'Declaration', and 'Cor'. The 'Problems' tab is selected, showing the output of the 'TestDijkstra [Java Application] C:\' application. The output consists of multiple lines, each starting with 'A' followed by a list of nodes and their weights, separated by arrows. The output is as follows:

```
A  
-> E(1)  
-> F(3)  
-> C(4)  
-> B(5)  
A  
-> E(1)  
-> F(3)  
-> C(4)  
A  
-> E(1)  
-> F(3)  
-> C(4)  
-> D(6)  
A  
-> E(1)  
A  
-> E(1)  
-> F(3)  
A  
-> E(1)  
-> F(3)  
-> G(5)  
A  
-> E(1)  
-> F(3)  
-> C(4)  
-> D(6)  
-> H(7)
```