




알고리즘9주차

201902694-박재우

RANK	TEAM	SCORE	1-NEIGHBOR  [1 POINT]	2-BREADTHFIRSTSEARCH  [2 POINTS]	3-DEPTHFIRSTSEARCH  [2 POINTS]
21	201902694	5796	84 1 try	294 5 tries	258 5 tries

Submissions

time	problem	lang	result
10/26/20-19:24	2-BREADTHFIRSTSEARCH	JAVA	CORRECT
10/26/20-19:23	2-BREADTHFIRSTSEARCH	JAVA	COMPILER-ERROR
10/26/20-19:05	2-BREADTHFIRSTSEARCH	JAVA	WRONG-ANSWER
10/26/20-19:03	2-BREADTHFIRSTSEARCH	JAVA	WRONG-ANSWER
10/26/20-18:48	3-DEPTHFIRSTSEARCH	JAVA	CORRECT
10/26/20-18:42	3-DEPTHFIRSTSEARCH	JAVA	WRONG-ANSWER
10/26/20-18:18	3-DEPTHFIRSTSEARCH	JAVA	WRONG-ANSWER
10/26/20-18:12	3-DEPTHFIRSTSEARCH	JAVA	WRONG-ANSWER
10/26/20-17:50	3-DEPTHFIRSTSEARCH	JAVA	WRONG-ANSWER
10/26/20-16:53	2-BREADTHFIRSTSEARCH	JAVA	WRONG-ANSWER
10/26/20-16:36	2-BREADTHFIRSTSEARCH	JAVA	WRONG-ANSWER
10/26/20-15:53	1-NEIGHBOR	JAVA	CORRECT

Clarifications

No clarifications.

Clarification Requests

No clarification request.

request clarification

문제 1

- 문제/목표
인접한 노드 개수 출력
- 해결방법(소스코드 첨부)

```
public class first {  
  
    class Node {  
        boolean marked = false;  
        String data;  
        LinkedList<Node> edge;  
  
        public Node(String data) {  
            this.data = data;  
            edge = new LinkedList<Node>();  
        }  
    }  
    static Node[] nodes;  
    public first(String[] test) {  
        nodes = new Node[test.length];  
        nodes = create(test);  
    }  
    public static Node[] getNode() {  
        return nodes;  
    }  
  
    public Node[] create(String[] test) {  
        Node[] nodes = new Node[test.length];  
        for(int i=0;i<test.length;i++) {  
            nodes[i] = new Node(test[i]);  
        }  
        return nodes;  
    }  
  
    static void addEdge(Node n1, Node n2) {  
        if(!n1.edge.contains(n2)) {  
            n1.edge.add(n2);  
        }  
        if(!n2.edge.contains(n1)) {  
            n2.edge.add(n1);  
        }  
    }  
  
    static int searches(Node[] nodes,String data) {  
        for(int i=0;i<nodes.length;i++) {  
            if(nodes[i].data.equals(data)) {  
                return i;  
            }  
        }  
        return -1;  
    }  
  
    public static void main(String[] args) {  
        Scanner sc = new Scanner(System.in);  
        int n =sc.nextInt();  
        int m = sc.nextInt();  
        sc.nextLine();  
        String[] test = sc.nextLine().split(" ");  
  
        first t= new first(test);  
        Node[] nodes = t.getNode();  
  
        String[][] edge = new String[m][2];  
        for(int i=0;i<m;i++)  
            edge[i] = sc.nextLine().split(" ");  
        for(int i=0;i<m;i++) {  
            int n1 = searches(nodes,edge[i][0]);
```

```

        int n2 = searches(nodes, edge[i][1]);
        addEdge(nodes[n1], nodes[n2]);
    }
    String search = sc.next();
    int index = searches(nodes, search);
    System.out.print(nodes[index].edge.size());

}

}

```

문제 조건에 맞게 데이터를 입력받은 후 입력받은 node의 데이터로 node를 생성한다.

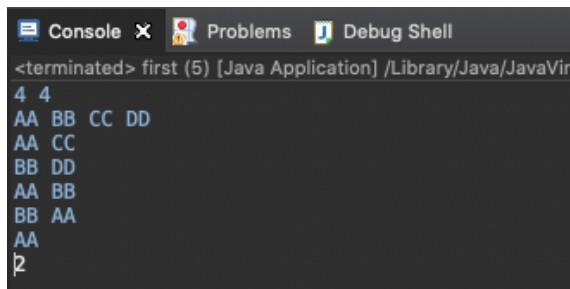
Node에는 연결된 node가 들어있는 edge, 방문여부 marked, 데이터를 생성한다.

Node를 생성한 후, 입력받는 edge정보에 따라서 각 node들의 edge정보를 추가한다.

Edge는 Linedlist로 되어있으며 추가시에는 addEdge함수를 통해 이미 연결된 노드가 추가되어있는지 확인한뒤에 추가한다.

위의 edge추가 과정이 완료되면 마지막으로 받은 인접한 노드의 개수를 찾아야 할 노드를 search함수를 통해서 미리 저장된 node의 배열에서 그 인덱스를 찾은 뒤 그 노드의 edge 사이즈를 확인하고 이를 출력한다.

- 결과



```

<terminated> first (5) [Java Application] /Library/Java/JavaVir
4 4
AA BB CC DD
AA CC
BB DD
AA BB
BB AA
AA
2

```

- 시간복잡도 : 전체코드에서 반복문을 각각 1번씩 사용하므로 $O(n)$ 이다.

- 자신만의 생각/느낀점

단순히 무방향 그래프를 생성할 줄만 알면 값은 금방 구해지기에 구현하는데 없었다고 생각한다.

- 난이도 (2/5)

문제 2

- 문제/목표

BFS를 이용하여 사전순으로 방문하기

- 해결방법(소스코드 첨부)

```
public class second {  
  
    class Node {  
        boolean marked = false;  
        String data;  
        LinkedList<Node> edge;  
  
        public Node(String data) {  
            this.data = data;  
            edge = new LinkedList<Node>();  
        }  
    }  
    static Node[] nodes;  
    public second(String[] test) {  
        nodes = new Node[test.length];  
        nodes = create(test);  
    }  
    public static Node[] getNode() {  
        return nodes;  
    }  
  
    public Node[] create(String[] test) {  
        Node[] nodes = new Node[test.length];  
        for(int i=0;i<test.length;i++) {  
            nodes[i] = new Node(test[i]);  
        }  
        return nodes;  
    }  
  
    static void addEdge(Node n1, Node n2) {  
        if(!n1.edge.contains(n2)) {  
            n1.edge.add(n2);  
        }  
        if(!n2.edge.contains(n1)) {  
            n2.edge.add(n1);  
        }  
    }  
  
    static int searches(Node[] nodes,String data) {  
        for(int i=0;i<nodes.length;i++) {  
            if(nodes[i].data.equals(data)) {  
                return i;  
            }  
        }  
        return -1;  
    }  
  
    public static void bfs(Node node) {  
        Queue<Node> queue = new LinkedList<Node>();  
        System.out.print(node.data+" ");  
        node.marked = true;  
        queue.add(node);  
        while(!queue.isEmpty()) {  
            Node r = queue.remove();  
            sort(r.edge);  
            for (Node n : r.edge) {  
                /*System.out.println(r.data);  
                for(Node a : r.edge) {  
                    System.out.println("n : "+a.data+" m : "+a.marked);  
                }  
                System.out.println();*/  
                if(n.marked == false) {
```

```

        System.out.print(n.data+" ");
        n.marked = true;
        queue.add(n);
    }
}

}

}

public static void sort(LinkedList<Node> edge) {
    String temp_data;
    LinkedList<Node> temp_edge;
    boolean temp_marked;
    for(int i = 0 ; i < edge.size() ; i ++){
        for(int j = 0 ; j < edge.size() -i -1 ; j ++){
            if(edge.get(j).data.compareTo(edge.get(j+1).data) > 0) {
                temp_data = edge.get(j).data;
                temp_edge = edge.get(j).edge;
                temp_marked = edge.get(j).marked;
                edge.get(j).data = edge.get(j+1).data;
                edge.get(j).edge = edge.get(j+1).edge;
                edge.get(j).marked = edge.get(j+1).marked;
                edge.get(j+1).data = temp_data;
                edge.get(j+1).edge = temp_edge;
                edge.get(j+1).marked = temp_marked;
            }
        }
    }
}

public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int n =sc.nextInt();
    int m = sc.nextInt();
    sc.nextLine();
    String[] test = sc.nextLine().split(" ");

    second t= new second(test);
    Node[] nodes = t.getNode();

    String[][] edge = new String[m][2];
    for(int i=0;i<m;i++){
        edge[i] = sc.nextLine().split(" ");
    }
    for(int i=0;i<m;i++){
        int n1 = searches(nodes,edge[i][0]);
        int n2 = searches(nodes,edge[i][1]);
        addEdge(nodes[n1],nodes[n2]);
    }
    String search = sc.next();
    int index = searches(nodes,search);
    bfs(nodes[index]);

}

}

```

Node를 생성하고 graph를 만드는 과정은 문제 1번과 같다.

생성된 graph에서 마지막으로 받은 node데이터를 이용해 node배열에서 찾은 첫번째 방문 노드를 이용하여 bfs과정을 실행한다.

첫번째 노드의 데이터를 출력하고 방문기록을 남긴뒤 첫번째 노드를 큐에 삽입한다.

큐가 빌때까지 while문을 반복하는데 큐를 remove시키면서 나온 노드의 edge를 사전순으로 정렬한다.

첫번째 노드를 큐에 넣고순차적으로 방문해서 방문기록을 남기고 데이터를 출력한다.

- 결과

```
12 11
A B C D E F G H I J K L
A B
A C
A D
B E
B F
D G
D H
E I
E J
G K
G L
A
A B C D E F G H I J K L
```

- 시간복잡도 : node의 edge를 사전순으로 정렬할때 버블정렬을 사용하므로 $O(n^2)$ 이다.

- 자신만의 생각/느낀점

일반적으로 bfs를 구현했을때에는 시간이 생각보다 오래걸리지 않았지만,
사전순으로 방문해야하는 조건을 만족시키는데 많이 오래걸렸다. 결국 다른방법을 사용하
기위해 처음코드와는 코드의 결과가 많이 달라졌다.

- 난이도 4/5

문제 3

- 문제/목표

2번문제를 dfs로 해결하기

- 해결방법(소스코드 첨부)

```
package week9;

import java.util.*;

public class third {

    class Node {
        boolean marked = false;
        String data;
        LinkedList<Node> edge;

        public Node(String data) {
            this.data = data;
            edge = new LinkedList<Node>();
        }
    }

    static Node[] nodes;
    public third(String[] test) {
        nodes = new Node[test.length];
        nodes = create(test);
    }

    public static Node[] getNode() {
```

```

        return nodes;
    }

    public Node[] create(String[] test) {
        Node[] nodes = new Node[test.length];
        for(int i=0;i<test.length;i++) {
            nodes[i] = new Node(test[i]);
        }
        return nodes;
    }

    static void addEdge(Node n1, Node n2) {
        if(!n1.edge.contains(n2)) {
            n1.edge.add(n2);
        }
        if(!n2.edge.contains(n1)) {
            n2.edge.add(n1);
        }
    }

    static int searches(Node[] nodes,String data) {
        for(int i=0;i<nodes.length;i++) {
            if(nodes[i].data.equals(data)) {
                return i;
            }
        }
        return -1;
    }

    static String ar = "";

    public static void dfs(Node node) {
        ar += node.data + " ";
        node.marked = true;
        sort(node.edge);
        for(Node j : node.edge) {
            if(j.marked == false)
                dfs(j);
        }
    }

    public static void sort(LinkedList<Node> edge) {
        String temp_data;
        LinkedList<Node> temp_edge;
        boolean temp_marked;
        for(int i = 0 ; i < edge.size() ; i ++ ) {
            for(int j = 0 ; j < edge.size() -i -1 ; j ++ ) {
                if(edge.get(j).data.compareTo(edge.get(j+1).data) > 0) {
                    temp_data = edge.get(j).data;
                    temp_edge = edge.get(j).edge;
                    temp_marked = edge.get(j).marked;
                    edge.get(j).data = edge.get(j+1).data;
                    edge.get(j).edge = edge.get(j+1).edge;
                    edge.get(j).marked = edge.get(j+1).marked;
                    edge.get(j+1).data = temp_data;
                    edge.get(j+1).edge = temp_edge;
                    edge.get(j+1).marked = temp_marked;
                }
            }
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n =sc.nextInt();
        int m = sc.nextInt();
        sc.nextLine();
        String[] test = sc.nextLine().split(" ");
    }

```

```

-      third t= new third(test);
-      Node[] nodes = t.getNode();
-
-      String[][] edge = new String[m][2];
-      for(int i=0;i<m;i++)
-          edge[i] = sc.nextLine().split(" ");
-      for(int i=0;i<m;i++) {
-          int n1 = searches(nodes,edge[i][0]);
-          int n2 = searches(nodes,edge[i][1]);
-          addEdge(nodes[n1],nodes[n2]);
-      }
-      String search = sc.next();
-      int index = searches(nodes,search);
-      dfs(nodes[index]);
-      System.out.printf("%s",ar.trim());
-
-    }
-
- }

```

노드를 이용하여 graph를 생성하는과정은 1번과 같다.

마지막으로 입력받은 data를 통해 첫번째로 방문할 노드를 찾은 다음 bfs함수에 넣어준다.

Bfs함수에서는 입력받은 node의 데이터를 string 변수에 추가한뒤에 방문기록을 남기고

입력받은 node의 edge의 개수만큼 그 edge안의 노드를 재귀적으로 node를 bfs함수를 통해 전달한다. 이렇게 깊이를 우선하면서 탐색하면 bfs알고리즘이 완성된다.

- 결과

```

Console x Problems Debug
<terminated> second (4) [Java Application]
12 11
A B C D E F G H I J K L
A B
A C
A D
B E
B F
D G
D H
E I
E J
G K
G L
A
A B C D E F G H I J K L

```

- 시간복잡도 : 재귀함수를 사용하므로 $T(n)$ 이다

- 자신만의 생각/느낌점

2번문제를 해결했으면 금방 할 수 있지만 2번문제도 어려웠기에 쉬웠다고 생각이 들지는 않는다.

- 난이도 4/5