자료구조 및 실습
보고서

[제12주] DS_12_201502273_김현종

2018.06.11.

201502273/김현종

1.내용

```
public class Edge {
    private int src;
    private int dst;
    private int weight;

    /*
     * Edge의 Constructor
     * src, dst, weight를 받아 초기화한다.
     */
    public Edge(int src, int dst, int weight) {
        this.src = src;
        this.dst = dst;
        this.weight = weight;
    }

    /*
     * src의 getter
     */
    public int getSrc() {
        return src;
    }

    /*
     * dst의 getter
     */
    public int getDst() {
        return dst;
    }

    /*
     * weight의 getter
     */
    public int getWeight() {
        return weight;
    }
}
```

Edge.java

```
import javax.swing.*;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.nio.file.Paths;
import java.util.Comparator;
import java.util.LinkedList;
import java.util.PriorityQueue;
import java.util.Queue;

class Graph {
        private int[][] edge;
        private int size;

        /*
         * Graph의 Constructor
         * vertex의 갯수에 해당하는 size를 받아 초기화한다.
         */
        public Graph(int size) {
                this.size = size;
                this.edge = new int[size][size];
        }

        public int size() { //size의 getter
                return size;
        }

        /*
         * src와 dst가 존재하는 vertex인지 확인하고
```

```
	 * src에서 dst로 향하는 edge를 추가한다.
	 */
	public void addEdge(int src, int dst) {
		// TODO
		if(src < this.size && dst < this.size)this.edge[src][dst] = 1;
	}

	/*
	 * src와 dst가 존재하는 vertex인지 확인하고
	 * src에서 dst로 향하는 edge를 제거한다.
	 */
	public void removeEdge(int src, int dst) {
		// TODO
		if(src < this.size && dst < this.size)this.edge[src][dst] = 0;
	}

	/*
	 * graph 데이터를 읽어오는 함수
	 */
	public void readGraph(File file) throws IOException {
		if (file == null) {
			String currentPath = Paths.get(".", "src").toString();
			JFileChooser jFileChooser = new JFileChooser();
			jFileChooser.setCurrentDirectory(new File(currentPath));
			jFileChooser.showOpenDialog(new JFrame());
			file = jFileChooser.getSelectedFile();
		}
		if (file != null) {
			BufferedReader br = new BufferedReader(new FileReader(file));
			this.size = Integer.parseInt(br.readLine());
			this.edge = new int[size][size];
			for (int i = 0; i < size; ++i) {
				String[] temps = br.readLine().split(" ");
				for (int j = 0; j < temps.length; ++j) {
					edge[i][j] = Integer.parseInt(temps[j]);
				}
			}
			System.out.println("File selected!");
		} else {
			System.out.println("No file Selected");
		}
	}

	/*
	 * 현재 만들어진 graph를 출력한다.
	 */
	public void print() {
		System.out.println("This Graph ::\n");
		for (int i = 0; i < size; ++i) {
			for (int j = 0; j < size; ++j) {
				System.out.print(edge[i][j] + " ");
			}
			System.out.println();
		}
		System.out.println();
	}

	/*
	 * 그래프를 Breath First Search로 탐색하는 함수
	 * 시작 vertex를 받아오고 인접 vertex를 q에 넣는다.
	 * 그리고 q에서 vertex를 하나씩 꺼내와 출력하고 인접 vertex를 다 q에 넣는다.
	 * 한번 들렀던 vertex는 다시 들리지 않는다.
	 * q가 빌때까지 반복한다.
	 */
	public void bfs(int vertex) {
		// TODO
		boolean[] isFounds = new boolean[this.size];
		Queue<Integer> q = new LinkedList<Integer>();
		q.add(vertex);
		while (!q.isEmpty()) {
			int curr = q.poll();
			System.out.print(curr + " -> ");
```

```
                        isFounds[curr] = true;

                        for (int i = 0; i < this.size; i++) {
                                if (!isFounds[i] && this.edge[curr][i] != 0) {
                                        q.add(i);
                                        isFounds[i] = true;
                                }
                        }
                }
        }

        /*
         * Depth First Search를 하는 메소드
         * 들렀던 vertex 리스트와 현재 vertex를 받아 현재 vertex를 출력하고
         * 아직 들리지 않은 vertex를 재귀적으로 호출한다.
         */
        public void dfs(boolean[] isFounds, int vertex) {
                // TODO
                isFounds[vertex] = true;
                System.out.print(vertex + " -> ");

                for (int i = 0; i < this.size; i++) {
                        if (!isFounds[i] && this.edge[vertex][i] != 0) {
                                dfs(isFounds, i);
                        }
                }
        }

        /*
         * edge를 우선순위 큐에 넣어 weight를 가지고 Min Heap을 구성한다.
         * 가장 작은 Weight를 갖는 edge부터 가져와 같은 union에 있는 vertex인지 확인하고
서로 다른 union에 있으면
         * 하나의 union으로 union시켜준다. 그리고 해당 edge의 정보(src, dst, weight)를
출력하며 모든 vertex를 잇던가
         * 우선순위 큐가 빌때까지 반복한다.
         */
        public void kruskalAlgorithm() {
                // TODO
                PriorityQueue<Edge> pq = new PriorityQueue<Edge>(this.size, new
Comparator<Edge>() {

                        @Override
                        public int compare(Edge o1, Edge o2) {
                                // TODO Auto-generated method stub
                                if (o1.getWeight() > o2.getWeight()) {
                                        return 1;
                                } else {
                                        return -1;
                                }

                        }
                });

                for (int i = 0; i < this.size; i++) {
                        for (int j = i + 1; j < this.size; j++) {
                                if (this.edge[i][j] != 0) {
                                        pq.add(new Edge(i, j, this.edge[i][j]));
                                }
                        }
                }
                int foundVertex = 1;

                int[] union = new int[this.size];
                for (int i = 0; i < union.length; i++) {
                        union[i] = -1;
                }
                int totalCost = 0;
                while (foundVertex != this.size && !pq.isEmpty()) {
                        Edge curr = pq.poll();
                        if (!isInUnion(union, curr)) {
                                union(union, curr);
                                foundVertex++;
```

```
                                        totalCost += curr.getWeight();
                                        System.out.println(curr.getSrc() + "\t->\t" +
curr.getDst() + ":\t" + curr.getWeight());
                                }
                        }
                        System.out.println("Total cost : " + totalCost);
                }

                public void primAlgorithm(int start) {
                        // TODO
                        boolean[] found = new boolean[this.size];
                        int[] cost = new int[this.size];
                }

                /*
                 * 해당 edge의 src와 dst가 같은 union인지 확인하는 메소드
                 * src의 root와 dst의 root가 같으면 true를 반환한다.
                 */
                private boolean isInUnion(int[] union, Edge edge) {
                        int tempDst = edge.getDst();
                        int tempSrc = edge.getSrc();

                        while (union[tempSrc] >= 0) {
                                tempSrc = union[tempSrc];
                        }

                        while (union[tempDst] >= 0) {
                                tempDst = union[tempDst];
                        }
                        return tempSrc == tempDst;
                }

                /*
                 * edge의 src와 dst를 union하는 함수
                 * src와 dst의 root를 찾아서 둘 중 멤버수가 적은 union을 멤버수가 많은 union에
붙힌다.
                 */
                private void union(int[] union, Edge edge) {
                        int src = edge.getSrc();
                        int dst = edge.getDst();

                        int tempSrc = src;
                        while (union[tempSrc] >= 0) {
                                tempSrc = union[tempSrc];
                        }

                        int tempDst = dst;
                        while (union[tempDst] >= 0) {
                                tempDst = union[tempDst];
                        }

                        if (union[tempDst] < union[tempSrc]) {
                                union[tempSrc] += union[tempDst];
                                union[tempDst] = tempSrc;
                        } else {
                                union[tempDst] += union[tempSrc];
                                union[tempSrc] = tempDst;
                        }
                }
}
```

Graph.java

```
import java.io.IOException;
import java.util.Comparator;
import java.util.PriorityQueue;
import java.util.Queue;
import java.util.Scanner;

public class MainClass_12_201502273 {
    public static void main(String[] args) throws IOException {
        System.out.println("==== Graph ====");
```

```
        System.out.println("==== Start ====");
        int num;
        Scanner scanner = new Scanner(System.in);

        System.out.print("Input Graph's vertex size: ");
        int size = scanner.nextInt();
        if (size > 0) {
            Graph graph = new Graph(size);

            while (true) {
                System.out.println("1: Add Edge :: 2: Remove Edge :: 3: BFS ");
                System.out.println("4: DFS :: 5: print :: 6: READ FILE \n7: Kruskal ::
8:Prim :: 9: Exit ");
                num = scanner.nextInt();
                switch (num) {
                    case 1: {
                        int src, dst;
                        System.out.print("Input src : ");
                        src = scanner.nextInt();
                        System.out.print("Input dst : ");
                        dst = scanner.nextInt();
                        graph.addEdge(src, dst);
                        break;
                    }
                    case 2: {
                        int src, dst;
                        System.out.print("Input src : ");
                        src = scanner.nextInt();
                        System.out.print("Input dst : ");
                        dst = scanner.nextInt();
                        graph.removeEdge(src, dst);
                        break;
                    }
                    case 3: {
                        System.out.print("Input vertex : ");
                        int vertex = scanner.nextInt();
                        graph.bfs(vertex);
                        System.out.println();
                        break;
                    }
                    case 4: {
                        System.out.print("Input vertex : ");
                        int vertex = scanner.nextInt();
                        graph.dfs(new boolean[graph.size()], vertex);
                        System.out.println();
                        break;
                    }
                    case 5: {
                        graph.print();
                        break;
                    }
                    case 6: {
                        System.out.println("Select File : ");
                        graph.readGraph(null);
                        break;
                    }
                    case 7: {
                        graph.kruskalAlgorithm();
                        break;
                    }
                    case 8: {
                        System.out.print("Input vertex : ");
                        int vertex = scanner.nextInt();
                        graph.primAlgorithm(vertex);
                        break;
                    }
                    case 9: {
                        System.out.println("==== END ====");
                        return;
                    }
                    default:
                        break;
```

```
                    }
                }
            }
        }
}
```

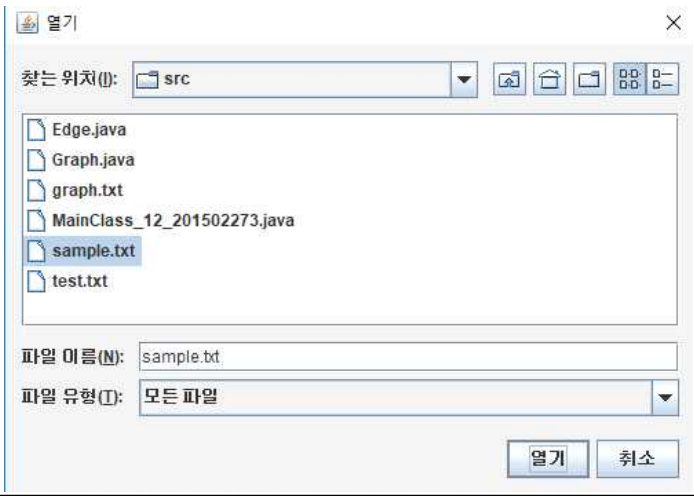
MainClass_12_201502273.java

2.결과

```
==== Graph ====
==== Start ====
Input Graph's vertex size: 8
1: Add Edge :: 2: Remove Edge :: 3: BFS
4: DFS :: 5: print :: 6: READ FILE
7: Kruskal :: 8:Prim :: 9: Exit
1
Input src : 1
Input dst : 5
1: Add Edge :: 2: Remove Edge :: 3: BFS
4: DFS :: 5: print :: 6: READ FILE
7: Kruskal :: 8:Prim :: 9: Exit
1
Input src : 5
Input dst : 3
1: Add Edge :: 2: Remove Edge :: 3: BFS
4: DFS :: 5: print :: 6: READ FILE
7: Kruskal :: 8:Prim :: 9: Exit
5
This Graph ::

0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

1: Add Edge :: 2: Remove Edge :: 3: BFS
4: DFS :: 5: print :: 6: READ FILE
7: Kruskal :: 8:Prim :: 9: Exit
6
```

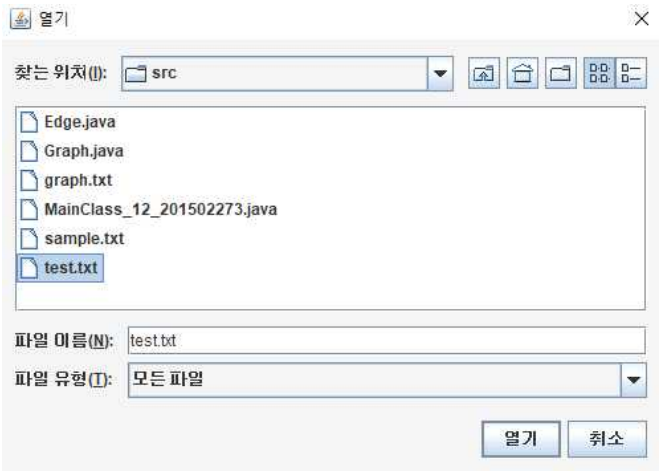


```
Select File :
File selected!
1: Add Edge :: 2: Remove Edge :: 3: BFS
4: DFS :: 5: print :: 6: READ FILE
7: Kruskal :: 8:Prim :: 9: Exit
5
This Graph ::

0 4 8 0 0 0 0
4 0 9 8 10 0 0
8 9 0 2 0 1 0
0 8 2 0 7 9 0
0 10 0 7 0 5 6
0 0 1 9 5 0 2
0 0 0 0 6 2 0

1: Add Edge :: 2: Remove Edge :: 3: BFS
4: DFS :: 5: print :: 6: READ FILE
7: Kruskal :: 8:Prim :: 9: Exit
7
2       ->      5:      1
5       ->      6:      2
2       ->      3:      2
0       ->      1:      4
4       ->      5:      5
0       ->      2:      8
Total cost : 22
```

```
1: Add Edge :: 2: Remove Edge :: 3: BFS
4: DFS :: 5: print :: 6: READ FILE
7: Kruskal :: 8:Prim :: 9: Exit
6
```



```
Select File :
File selected!
1: Add Edge :: 2: Remove Edge :: 3: BFS
4: DFS :: 5: print :: 6: READ FILE
7: Kruskal :: 8:Prim :: 9: Exit
3
Input vertex : 0
0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 ->
1: Add Edge :: 2: Remove Edge :: 3: BFS
4: DFS :: 5: print :: 6: READ FILE
7: Kruskal :: 8:Prim :: 9: Exit
4
Input vertex : 0
0 -> 1 -> 3 -> 2 -> 5 -> 4 -> 7 -> 6 ->
1: Add Edge :: 2: Remove Edge :: 3: BFS
4: DFS :: 5: print :: 6: READ FILE
7: Kruskal :: 8:Prim :: 9: Exit
5
This Graph ::

0 1 1 0 0 0 0 0
1 0 0 1 1 0 0 0
1 0 0 0 0 1 1 0
0 1 1 0 0 0 0 1
0 1 0 0 0 0 0 1
0 0 1 0 1 0 0 1
0 0 1 0 0 0 0 1
0 0 0 1 1 1 1 0

1: Add Edge :: 2: Remove Edge :: 3: BFS
4: DFS :: 5: print :: 6: READ FILE
7: Kruskal :: 8:Prim :: 9: Exit
9
==== END ====
```

임의로 Edge를 추가하면 weight가 1로 설정되는 것을 볼 수 있고 sample.txt 파일을 읽어와 그래프를 받아오고 Kruskal's Algorithm을 실행시키면 해당 그래프에서 최소 신장 트리를 위의 결과와 같이 얻어내는 것을 볼 수 있고 총 cost는 22가 나오는 것을 볼 수 있다. 또한 test.txt파일을 가져와 DFS와 BFS를 수행해본 결과 BFS는 0번 vertex에서 출발하여 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7의 순서로 검색하는 것을 볼 수 있고 DFS의 경우에는 똑같이 0번 vertex에서 출발하여 0 -> 1 -> 3 -> 2 -> 5 -> 4 -> 7 -> 6의 순서로 검색하는 것을 볼 수 있다.

깨달은 점 및 결론

실습에서 사용해본 그래프를 확장하여 미로찾기, 네비게이션, SNS 등 폭넓은 분야에 활용할 수 있을 것으로 보인다. 그러한 모델에서는 연결되지 않은 Vertex가 많으므로 지금의 실습과 같이 행렬을 통째로 사용하는 것 보다는 Sparse Matrix를 표현하는 방식으로 구현하는 것이 효율적일 것으로 생각된다.