

# 실습보고서

2020. 11. 17 (화)

[알고리즘 00분반]_11주차
201501513_김성현

조건사항

운영체제 : MAC OS

IDE : IntelliJ

라이브러리 : io(입출력), util(코드설계), nio(파일읽기)

1. Dijkstra algorithm(Minimum Priority Queue)을 사용하여 다음 graph의 최단 경로 cost를 계산하는 프로그램을 구현하라.

### <실행 결과>

```
Run: Dijkstra_revise
"/Applications/IntelliJ IDEA.app/Contents/jbr/Contents/Home/bin/java" -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=52564:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath
/Users/sunghyun/IdeaProjects/Algorithms/out/production/Algorithms HomeWork_11thWeek.Dijkstra_revise
S[1]
-----
Q[0] : d[2] = 2147483647 => 10
Q[1] : d[3] = 2147483647 => 3
S[5]
-----
Q[0] : d[2] = 10 => 7
Q[1] : d[4] = 2147483647 => 11
Q[2] : d[5] = 2147483647 => 5
S[5]
-----
Q[0] : d[4] = 11
S[2]
-----
Q[0] : d[3] = 3
Q[1] : d[4] = 11 => 9
S[4]
-----
Q[0] : d[5] = 5
Process finished with exit code 0
```

### <코드 설명>

```

PriorityQueue<Vertex> que = new PriorityQueue<>();
que.offer(new Vertex( name: 1, direct: 0, distance: 0));
d[1] = 0;

while(set.size() != size){
    Vertex vertex = que.poll();

    set.add(vertex.name);
    System.out.println("S["+vertex.name+"]");
    System.out.println("-----");

    int start_point = vertex.name-1;
    int count = 0;
    for(int i = 0; i < list.get(start_point).size(); i++){
        Vertex temp = list.get(start_point).get(i);
        System.out.print("Q["+count+++"] : d["+temp.direct+"] = "+d[temp.direct]);

        if(!set.contains(temp.direct)){
            int new_distance = d[start_point+1] + temp.distance;

            if(new_distance < d[temp.direct]){
                d[temp.direct] = new_distance;
                System.out.print(" => "+d[temp.direct]);
            }
            que.offer(new Vertex(temp.direct, direct: 0, d[temp.direct]));
        }
        System.out.println();
    }
    System.out.println();
}

//print();
}

```

다음과 같이 Vertex 클래스를 따로 생성하여 compareTo 메소드를 사용하여 Priority Queue에 대해서 값이 자동으로 min heap으로 되도록 작성하였습니다.

또한 dynamic programming을 구현하기 위해 d라는 int형 배열을 생성하여 해당 값이 다른 길을 찾을 때 짧은 길이 존재한다면 업데이트하도록 조건문을 구현하였습니다.

그리고 HashSet을 이용하여 해당 vertex가 가지고 있는 name, 즉 출발점을 기억하고 이를 결과 집합으로 사용하여 해당 크기가 입력의 문자열 개수와 같다면 반복문을 탈출하도록 구현하였습니다. (해당 Queue의 값이 존재하지 않는 경우를 조건으로 뒀을 때, 지난 경과를 이따금씩 반복하는 부분이 있어 집합의 크기로 수정하였습니다.)

2. Prim's algorithm을 통해 MST를 만드는 프로그램을 구현하라.

### <실행 결과>

```
Graph 그리기
0 - 1 : 4 <=> 1 - 0 : 4
0 - 7 : 8 <=> 7 - 0 : 8
1 - 2 : 8 <=> 2 - 1 : 8
1 - 7 : 11 <=> 7 - 1 : 11
2 - 3 : 7 <=> 3 - 2 : 7
2 - 5 : 4 <=> 5 - 2 : 4
2 - 8 : 2 <=> 8 - 2 : 2
3 - 4 : 9 <=> 4 - 3 : 9
3 - 5 : 14 <=> 5 - 3 : 14
4 - 5 : 10 <=> 5 - 4 : 10
5 - 6 : 2 <=> 6 - 5 : 2
6 - 7 : 1 <=> 7 - 6 : 1
6 - 8 : 6 <=> 8 - 6 : 6
7 - 8 : 7 <=> 8 - 7 : 7
initial queue : [0 - 1 : 4]
graph list : [[0 - 1 : 4, 0 - 7 : 8], [1 - 0 : 4, 1 - 2 : 8, 1 - 7 : 11], [2 - 1 : 8, 2 - 3 : 7, 2 - 5 : 4, 2 - 8 : 2], [3 - 2 : 7, 3 - 4 : 9, 3 - 5 : 14], [4 - 3 : 9, 4 - 5 : 10], [5 - 2 : 4, 5 - 3 : 14, 5 - 4 : 10, 5 - 6 : 2], [6 - 5 : 2, 6 - 7 : 1, 6 - 8 : 6], [7 - 0 : 8, 7 - 1 : 11, 7 - 6 : 1, 7 - 8 : 7], [8 - 2 : 2, 8 - 6 : 6, 8 - 7 : 7]]

Prim's Algorithm
S[1]
-----
1 - 0 : 0
1 - 2 : 2147483647 => 0
1 - 7 : 2147483647 => 11

S[2]
-----
2 - 1 : 4
2 - 3 : 2147483647 => 7
2 - 5 : 2147483647 => 4
2 - 8 : 2147483647 => 2

S[8]
-----
8 - 2 : 8
8 - 6 : 2147483647 => 6
8 - 7 : 11 => 7

S[5]
-----
5 - 2 : 8
5 - 3 : 7
5 - 4 : 2147483647 => 10
5 - 6 : 6 => 2

S[6]
-----
6 - 5 : 4
6 - 7 : 7 => 1
6 - 8 : 2

S[7]
-----
7 - 0 : 0
7 - 1 : 4
7 - 6 : 2
7 - 8 : 2

S[6]
-----
6 - 5 : 4
6 - 7 : 1
6 - 8 : 2

S[7]
-----
7 - 0 : 0
7 - 1 : 4
7 - 6 : 2
7 - 8 : 2

S[3]
-----
3 - 2 : 8
3 - 4 : 10 => 9
3 - 5 : 4

S[4]
-----
4 - 3 : 7
4 - 5 : 4

S[4]
-----
4 - 3 : 7
4 - 5 : 4

S[7]
-----
7 - 0 : 0
7 - 1 : 4
7 - 6 : 2
7 - 8 : 2

result
a - b : 4
b - c : 8
c - d : 7
d - e : 9
e - f : 4
f - g : 2
g - h : 1
h - i : 2
총합 : 37

Process finished with exit code 0
```

### <코드 설명>

```

public void makeGraph(){
    String[] make_str = stringToArr(make);
    length = make_str.length;

    graph = new ArrayList<>();
    for(int i = 0; i < length; i++){ //vertex개수만큼 배열형태 생성
        graph.add(new ArrayList<>());
    }

    System.out.println("Graph 그리기");
    for(int i = 0; i < relation.length; i++){
        String[] str = stringToArr(relation[i]);
        int start = askii(str[0]);
        int end = askii(str[1]);
        int weight = Integer.parseInt(str[2]);

        Vertex u = new Vertex(start, end, weight);
        Vertex n = new Vertex(end, start, weight);

        graph.get(start).add(u);
        graph.get(end).add(n);

        System.out.println(u.toString()+" <=> "+n.toString());
    }
}

```

그래프를 생성하기 위해 makeGraph라는 메소드를 생성하여 data11\_mst.txt로부터 받아온 데이터들을 이용해 undirected graph를 생성하였습니다. ArrayList를 활용하여 이차원배열에서 필요한 값만 해당되는 그래프를 생성하는 것을 목표로 작성하였습니다.

```

public void solution(){
    makeGraph();//그래프 생성

    int[] key = new int[length];
    for(int i = 1; i < length; i++){
        key[i] = Integer.MAX_VALUE;
    }
    int[] parent = new int[length];
    set = new HashSet<>();

    PriorityQueue<Vertex> que = new PriorityQueue<>();
    que.offer(graph.get(0).get(0));
    key[graph.get(0).get(0).end] = graph.get(0).get(0).weight;

    System.out.println("initial queue : "+ que);
    System.out.println("graph list : "+graph);
    System.out.println("\nPrim's Algorithm");

    while(!que.isEmpty()){
        Vertex u = que.poll();
        set.add(u.start);

        System.out.println("S["+u.end+"]");
        System.out.println("-----");
        for(Vertex v : graph.get(u.end)){
            System.out.print(v.start+" - "+v.end+" : "+key[v.end]);
            if(v.weight < key[v.end] && !set.contains(v.end)){
                key[v.end] = v.weight;
                parent[v.end] = v.start;
                System.out.print(" => "+key[v.end]);
                que.offer(v);
            }
        }
        System.out.println();
    }
    System.out.println();
}

System.out.println("\nresult");
for(int i = 1; i < parent.length; i++){
    System.out.println((char)(parent[i]+97)+" - "+(char)(i+97)+" : "+key[i]);
    total += key[i];
}
System.out.println("총합 : "+total);
}

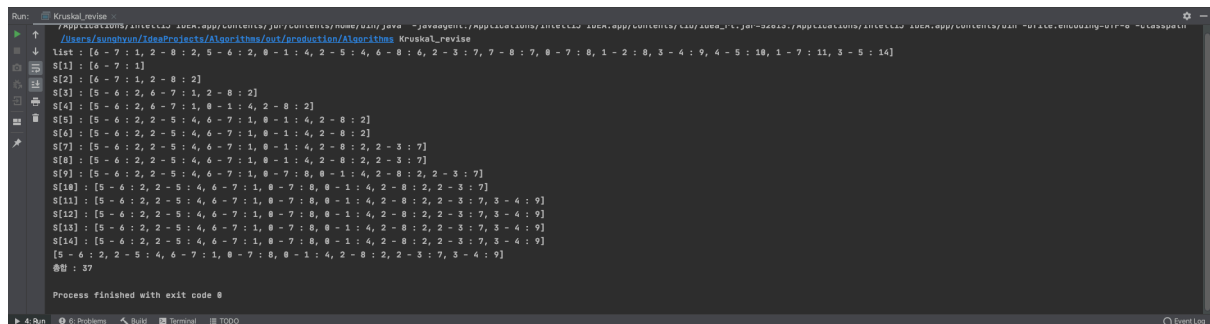
```

그리고 위에서 만들어 놓은 그래프를 통해 Priority Queue를 활용(compareTo를 활용하여 min heap을 형성)하여 가장 초기에 있는 initial vertex인 a-b를 표현하고자 사진과 같이 작성하게 되었습니다.

83, 84번 줄에 대해서 기존에 임의의 vertex를 넣기 위해 new Vertex(0, 0, 0)을 넣어 시작을 했지만 동일한 총합을 가진 a-b 대신에 a-h가 들어가게 되어 조교님이 ppt에서 보여주신대로 구현하기 위해 다음과 같이 작성하였습니다.(기존에 작성해놓은 코드는 주석으로 작성해놓았습니다.)

3. Kruskal을 통해 MST를 만드는 프로그램을 구현하라.

### <실행 결과>



```
Run: Kruskal_reviser
/Users/sunghyun/ideaProjects/Algorithms/out/production/Algorithms/Kruskal_reviser
list : [6-7 : 1, 2-8 : 2, 5-6 : 2, 8-1 : 4, 2-5 : 4, 6-8 : 6, 2-3 : 7, 7-8 : 7, 8-7 : 8, 1-2 : 8, 3-4 : 9, 4-5 : 10, 1-7 : 11, 3-5 : 14]
S[1] : [6-7 : 1]
S[2] : [6-7 : 1, 2-8 : 2]
S[3] : [5-6 : 2, 6-7 : 1, 2-8 : 2]
S[4] : [5-6 : 2, 6-7 : 1, 8-1 : 4, 2-8 : 2]
S[5] : [5-6 : 2, 2-5 : 4, 6-7 : 1, 8-1 : 4, 2-8 : 2]
S[6] : [5-6 : 2, 2-5 : 4, 6-7 : 1, 8-1 : 4, 2-8 : 2]
S[7] : [5-6 : 2, 2-5 : 4, 6-7 : 1, 8-1 : 4, 2-8 : 2, 2-3 : 7]
S[8] : [5-6 : 2, 2-5 : 4, 6-7 : 1, 8-1 : 4, 2-8 : 2, 2-3 : 7]
S[9] : [5-6 : 2, 2-5 : 4, 6-7 : 1, 8-7 : 8, 8-1 : 4, 2-8 : 2, 2-3 : 7]
S[10] : [5-6 : 2, 2-5 : 4, 6-7 : 1, 8-7 : 8, 8-1 : 4, 2-8 : 2, 2-3 : 7]
S[11] : [5-6 : 2, 2-5 : 4, 6-7 : 1, 8-7 : 8, 8-1 : 4, 2-8 : 2, 2-3 : 7, 3-4 : 9]
S[12] : [5-6 : 2, 2-5 : 4, 6-7 : 1, 8-7 : 8, 8-1 : 4, 2-8 : 2, 2-3 : 7, 3-4 : 9]
S[13] : [5-6 : 2, 2-5 : 4, 6-7 : 1, 8-7 : 8, 8-1 : 4, 2-8 : 2, 2-3 : 7, 3-4 : 9]
S[14] : [5-6 : 2, 2-5 : 4, 6-7 : 1, 8-7 : 8, 8-1 : 4, 2-8 : 2, 2-3 : 7, 3-4 : 9]
[5-6 : 2, 2-5 : 4, 6-7 : 1, 8-7 : 8, 8-1 : 4, 2-8 : 2, 2-3 : 7, 3-4 : 9]
000 : 27
Process finished with exit code 0
```

### <코드 설명>

```
public void solution(String vertexs, String[] relation){
    String[] index = vertexs.split( regex: " ");
    int length = index.length;

    path = new LinkedList<>();
    set = new LinkedList<>();
    set_end = new LinkedList<>();
    included = new int[length];

    LinkedList<Vertex> list = new LinkedList<>();
    for(int i = 0; i < relation.length; i++){
        String[] str = relation[i].split( regex: " ");
        list.add(new Vertex(askii(str[0]), askii(str[1]), Integer.parseInt(str[2])));
    }
    Collections.sort(list);

    for(int i = 0; i < length; i++){
        included[i] = i;
    }
    System.out.println("list : "+list);

    for(int i = 0; i < list.size(); i++){
        path.add(new LinkedList<>());
        path.get(i).add(list.get(i));

        set.add(new HashSet<>());
        set_end.add(new HashSet<>());
        set.get(i).add(list.get(i).start);
        set_end.get(i).add(list.get(i).end);
        //set.get(i).add(list.get(i).end);
    }

    HashSet<Vertex> result = new HashSet<>();
}
```

먼저 입력받은 txt 파일로부터 vertex의 개수와 edge의 관계를 parameter로 받아와 path라는 undirected graph를 생성하고 set을 통해 각 vertex가 포함되는 관계를 표현하려고 하였습니다.

그리고 included라는 Int형 배열을 생성하여 각 배열이 어디에 소속되어 있는지를 표현하고자 하였습니다. 배열의 기본값으로 0부터 입력받은 vertex개수만큼 각기 다른 linkedlist에 소속되어 있음을 구현하였습니다.

```

for(int i = 0; i < list.size(); i++){
    Vertex vertex = list.get(i);

    if(find(vertex.start) != find(vertex.end)){
        result.add(vertex);
        union(find(vertex.start), find(vertex.end));
    }

    System.out.println("S["+(i+1)+"] : "+result);
}

//System.out.println(path);

System.out.println(result);

System.out.println("\nKruskal MST");
int total = 0;
Iterator it = result.iterator();
while(it.hasNext()){
    Vertex v = (Vertex)it.next();
    System.out.println((char)(v.start+97)+" - "+(char)(v.end+97)+" : "+v.weight);
    total += v.weight;
}
System.out.println("총합 : "+total);
}

```

그리고 compareTo에 의한 Collections.sort를 진행하여 정렬을 한 후, 반복문을 통해 list에 속해있는 vertex들을 대상으로 해당 vertex의 노드가 어디에 소속되어 있는지 확인한 다음 해당 값이 같은 소속이 아니라면 union을 하는 형태로 구성하였습니다. 그리고 반복문 내에서 어떤 값들이 추가되는 지를 탐색순서를 표현하고자 하였습니다.



```

public void union(int u, int v){
    if(path.get(u).size() < path.get(v).size()){
        for(int i = 0; i < path.get(u).size(); i++){
            path.get(v).add(path.get(u).get(i));
        }

        path.remove(u);

        set.get(v).addAll(set.remove(u));
        set_end.get(v).addAll(set_end.remove(u));

        included[v] = u;
    }
    else{
        for(int i = 0; i < path.get(v).size(); i++){
            path.get(u).add(path.get(v).get(i));
        }

        path.remove(v);

        set.get(u).addAll(set.remove(v));
        set_end.get(u).addAll(set_end.remove(v));

        included[u] = v;
    }
}

public int find(int u){
    if(included[u] == u){
        return u;
    }
    return included[u] = find(included[u]);
}

```

Union은 linkedlist로 생성된 path를 크기가 작은 집합이 큰 집합에 속하도록 하며 소속된 곳을 교체하도록 하였습니다. 그리고 find는 초기값과 비교를 하며 초기값과 동일하지 않을 경우 해당 node가 소속되어 있는 곳으로 지정하고 반환되도록 하였습니다.