




## 알고리즘-11주차-MoreGraph

이름 : 유형곤

학번 : 201902722

사용 언어 : Java

### <점수표>

RANK	TEAM	SCORE	1-DELIVERYDRONE  [3 POINTS]	2-NoMoney  [3 POINTS]	3-BANDWIDTH  [4 POINTS]
1	201902722	10 92	14 1 try	25 1 try	53 1 try

### 문제 1 : DeliveryDrone

문제 : 다익스트라에서 간선뿐만 아니라 각 정점에 가중치를 부여하는 문제

해결 방법 : 다익스트라 알고리즘으로 단일 정점에서 모든 정점으로의 최단경로를 구하되, 각 정점에 대한 가중치를 계산하기 위해서 거리를 갱신할 때,

$alt = dist[now.v] + next.weight + weight[now.v]$  와 같이 현재 정점의 가중치를 더해줘야 한다. 단, 시작 정점에서는 가중치를 생각하지 않는다. 따라서 시작 정점의 가중치를 마지막 결과에서 빼준다.

시간복잡도 :  $O(E \log V)$

(우선순위 큐를 사용한 다익스트라의 시간복잡도)

```
dist.put(start, 0);

PriorityQueue<Pair> q = new PriorityQueue<Pair>();
q.add(new Pair(start, 0));

while(!q.isEmpty()) {
    Pair now = q.poll();
    Iterator<Pair> iterPair = graph.get(now.v).iterator();
    while(iterPair.hasNext()) {
        Pair next = iterPair.next();
        if(dist.get(next.v) > dist.get(now.v) + next.cost + delay.get(now.v)) {
            dist.put(next.v, dist.get(now.v) + next.cost + delay.get(now.v));
            prev.put(next.v, now.v);
            q.add(new Pair(next.v, dist.get(next.v)));
        }
    }
}
```

```

        String previous = end;
        Deque<String> stack = new LinkedList<String>();
        while(previous != null) {
            stack.push(previous);
            previous = prev.get(previous);
        }
        while(!stack.isEmpty()) {
            System.out.print(stack.pop() + " ");
        }
        System.out.println();
        return (dist.get(end));
    }

    String start = br.readLine();
    String end = br.readLine();
    int dist = dijkstra(start, end);
    System.out.println(dist - delay.get(start)); // 시작점에서는 딜레이가 없으므로 그만큼 빼준다.
    br.close();

```

<DeliveryDrone 소스코드>

```

<terminated> Drone [Java Application] C:\Program Files\Java\jre1.8.0_211\bin\javaw.exe (201
F 12
A B 10
A C 15
B D 12
B F 15
C E 10
D F 1
F E 5
A
E
A B D F E
43

```

<DeliveryDrone 실행결과>

문제 2 : NoMoney

문제 : astar 알고리즘으로 최단경로를 찾는 문제

해결방법 : 주어진 맵을 그래프로 바꾸되, 벽 ('W')이 있는 경우에는 연결하지 말고, 숫자인 경우에는 그 숫자 만큼의 비용을 가진 간선을 연결하여 시작 정점에서부터 끝 정점으로까지의 최단거리를 찾는다. (astar 알고리즘을 이용하여)

시간복잡도 : Unknown (단, 일반적으로 다익스트라의 시간복잡도보다 빠르다.)

```
class Vector implements Comparable<Vector>{
    int v;
    int cost;
    int h;

    public Vector(int v, int cost, int h) {
        this.v = v;
        this.cost = cost;
        this.h = h;
    }

    @Override
    public int compareTo(Vector o) {
        return Integer.compare(this.cost + this.h, o.cost + o.h);
    }
}
```

```
//이동경로 추적
int moneySpent = 0;
Integer previous = end;
while(previous != null) {
    int x = previous % w;
    int y = previous / w;
    if(map[y][x] != 'S' && map[y][x] != 'E' && map[y][x] != 'W') {
        moneySpent += map[y][x] - '0';
    }
    previous = prev.get(previous);
}
return moneySpent;
```

<NoMoney 소스코드>

```

public static int astar(int start, int end) {
    Map<Integer, Integer> dist = new HashMap<Integer, Integer>();
    Map<Integer, Integer> prev = new HashMap<Integer, Integer>();

    Iterator<Integer> iterV = graph.keySet().iterator();
    while(iterV.hasNext()) {
        int v = iterV.next();
        dist.put(v, INF);
    }
    dist.put(start, 0);

    PriorityQueue<Vector> q = new PriorityQueue<Vector>();
    q.add(new Vector(start, 0, dist(start, end)));

    while(!q.isEmpty()) {
        Vector now = q.poll();
        if(dist.get(now.v) < now.cost) {
            continue;
        }
        Iterator<Vector> iterVector = graph.get(now.v).iterator();
        while(iterVector.hasNext()) {
            Vector next = iterVector.next();
            if(dist.get(next.v) > dist.get(now.v) + next.cost) {
                dist.put(next.v, dist.get(now.v) + next.cost);
                prev.put(next.v, now.v);
                q.add(new Vector(next.v, dist.get(next.v), dist(next.v, end)));
            }
        }
    }
}

```

Console

<terminated> NoMoney [Java Application] C:\Program Files\

7 7 24

WWWWWWW

WS7201W

W28091W

W12181W

W98920W

W3651EW

WWWWWWW

13

<NoMoney 실행결과>

문제 3 : Bandwidth

문제 : 네트워크에서 여러 가지 경로로 데이터를 전송받을 수 있을 때, 데이터를 모두 다운로드 받을 때까지의 최소 시간을 구하는 문제

해결 방법 : Undirected graph 상에서 Ford-Fulkerson 알고리즘으로 최대유량(여기선 동시 데이터 전송량 = 대역폭)을 구하면 된다.

시간복잡도 :  $O(Ef)$

한 번 유량을 흘려보낼 때 마다 최소 1씩은 흘려보내는데, 최대 E개의 간선에 흘려보내므로 E 곱하기 f(최대유량)

```
//해쉬맵같은 인접행렬같은 해쉬맵 생성
st = new StringTokenizer(br.readLine());
while(st.hasMoreTokens()) {
    String node = st.nextToken();
    residual.put(node, new HashMap<>());
    graph.put(node, new HashMap<>());
}

//간선 저장
for(int i = 0; i < E; i++) {
    st = new StringTokenizer(br.readLine());
    String s = st.nextToken();
    String e = st.nextToken();
    int cost = Integer.parseInt(st.nextToken());
    graph.get(s).put(e, cost);
    graph.get(e).put(s, cost);
    residual.get(s).put(e, cost);
    residual.get(e).put(s, cost);
}
```

<Bandwidth 소스코드>

```

public static int fordFulkerson(String src, String dst) {
    int maxFlow = 0;
    //증가 경로가 존재하는가?
    while(bfs(src, dst)) {
        int pathFlow = INF;
        for(String v = dst; !v.equals(src); v = prev.get(v)) {
            String u = prev.get(v);
            pathFlow = Math.min(pathFlow, residual.get(u).get(v));
        }
        for(String v = dst; !v.equals(src); v = prev.get(v)) {
            String u = prev.get(v);
            residual.get(u).put(v, residual.get(u).get(v) - pathFlow);
            //a->b의 유량 변화는 b->a의 유량 변화와 반대 (이 성질로 최대 유량을 찾을 수 있음.)
            residual.get(v).put(u, residual.get(v).get(u) + pathFlow);
        }
        //가능한 모든 경로에 대한 유량을 더한 것이 최대 유량임.
        maxFlow += pathFlow;
    }
    return maxFlow;
}

//일반적인 BFS로, 여유 용량이 있는 경우에만 정점 순회
public static boolean bfs(String src, String dst) {
    HashMap<String, Boolean> visited = new HashMap<String, Boolean>();
    Queue<String> q = new LinkedList<String>();
    visited.put(src, true);
    q.add(src);

    while(!q.isEmpty()) {
        String u = q.poll();
        Iterator<String> iter = graph.get(u).keySet().iterator();
        while(iter.hasNext()) {
            String v = iter.next();
            if(!visited.getOrDefault(v, false) && residual.get(u).getOrDefault(v, 0) > 0) {
                visited.put(v, true);
                q.add(v);
                prev.put(v, u);
            }
        }
    }
    return visited.getOrDefault(dst, false);
}

```

```

Console
<terminated> Network [Java Application] C:\Program F
A B 3
B C 3
C D 5
D E 4
B E 6
A
E
100

```

<Bandwidth 실행결과>