알고리즘-10주차-ShortestPath
이름 : 유형곤
학번 : 201902722
사용 언어 : Java

<점수표>

9	201802094	10	3485	1 1 try	201 6 tries	511 11 tries	2412 4 tries
10	201902722	10	3631	12 1 try	1008 3 tries	1231 2 tries	1300 2 tries
11	201802114	10	4970	3 1 try	1495 10 tries	1511 2 tries	1741 2 tries

문제 1 : BFS
문제 : 방향 그래프에서 BFS로 탐색할 때, 정점의 순서를 출력하는 문제
해결 방법 : 시작 정점을 큐에 넣고, 연결된 정점을 모두 큐에 넣고 큐가 빌 때 까지 순회.

시간복잡도 : O(V + E) (인접리스트로 구현한 경우)
1. 모든 정점을 방문하고, 그때마다 정점에 연결된 간선들을 확인하므로 O(V+E)

```
public class Week10_BFS {
    public static void bfs(String v) {
        Queue<String> q = new LinkedList<String>();
        q.add(v);
        visited.put(v, true);
        System.out.print(v + " ");
        while(!q.isEmpty()) {
            v = q.poll();
            Iterator<String> iter = graph.get(v).iterator();
            while(iter.hasNext()) {
                String v_ = iter.next();
                if(!visited.getOrDefault(v_, false)) {
                    visited.put(v_, true);
                    q.add(v_);
                    System.out.print(v_ + " ");
                }
            }
        }
        System.out.println();
    }
```

<BFS 소스코드>

```
Console  Problems  Javadoc
<terminated> Week10_BFS (1) [Java Application] C:₩Program Files₩Java₩jre1.8.0_211₩bin₩javaw.exe (2019. 11. 15. 오후 3:29:57)
4 4
AA BB CC DD
AA CC
BB DD
AA BB
BB AA
AA
AA BB CC DD
```

<실행결과>

문제 2 : Dijkstra

문제 : 다익스트라 알고리즘으로 단일 정점에서 모든 정점으로의 최단거리를 구하는 문제

해결 방법 : 모든 정점에 대한 거리를 Infinity로, 시작 정점에 대한 거리를 0으로 초기화한 뒤, 시작 정점에서 가까운 순서대로 최단거리 집합을 확장해가면서 단일 정점에서 모든 정점으로의 최단거리를 구한다.

시간복잡도 : O((V+E) log V)

우선순위 큐를 사용하는 경우 O(E + V log V)

```
static Map<String, LinkedList<Pair>> graph = new HashMap<String, LinkedList<Pair>:
public static void bellmanFord(String start, String end) {
    Map<String, Integer> dist = new HashMap<String, Integer>();
    Map<String, String> prev = new HashMap<String, String>();

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

    boolean updated = false;
    for(int i = 0; i <= dist.keySet().size(); i++) {
        updated = false;
        iterV = graph.keySet().iterator();
        while(iterV.hasNext()) {
            String v = iterV.next();
            Iterator<Pair> iterE = graph.get(v).iterator();
            while(iterE.hasNext()) {
                Pair p = iterE.next();
                if(dist.get(v) != INF && dist.get(p.v) > dist.get(v) + p.cost) {
                    dist.put(p.v, dist.get(v) + p.cost);
                    prev.put(p.v, v);
                    updated = true;
                }
            }
        }
    }
```

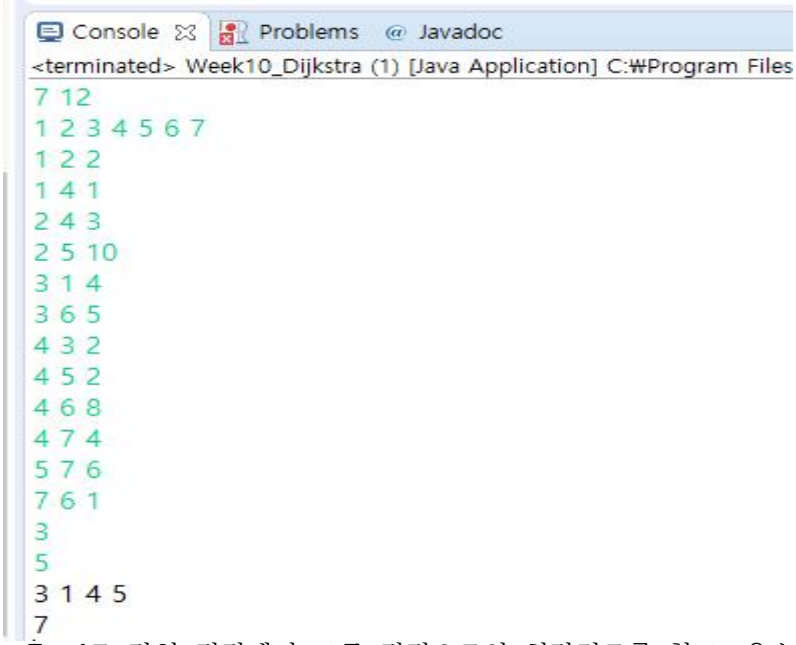
<Dijkstra 소스>

```
        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();
        System.out.println(dist.get(end));
    }
```

<Dijkstra 소스 - 경로추적>

<Dijkstra 실행 결과>

문제 3 : Bellman Ford



문제 : Bellman Ford로 단일 정점에서 모든 정점으로의 최단경로를 찾고, 음수 사이클이 있는가를 판별하는 문제
해결 방법 :

시간복잡도 : O(VE) = O(V^3)
Relaxation : O(E)
모든 간선에 대해서 기존의 경로보다 더 빠른 경로를 찾는 경우 완화(relaxation)를 합니다.
모든 간선에 대해서 수행하므로 시간복잡도가 O(E)입니다.

Relaxation의 반복 횟수 : V
최단경로에는 양수 사이클이 포함될 수 없으므로, V-1번 relaxation을 수행하면 최단경로를 반드시 찾을 수 있습니다. (시작 정점을 제외하여 V-1번.) 그런데 음수 사이클이 존재하는 경우 relaxation을 V번 이상 진행하므로, V번 진행했을 때 relaxation을 한다면 음수 사이클이 있는지도 확인할 수 있습니다. 따라서 V번 수행합니다.
결론적으로, O(E) 알고리즘을 V번 수행하므로 O(VE)가 됩니다. 그런데, dense graph의 경우 (최악인 경우) E를 V(V-1)으로 근사할 수 있으므로 O(VE) = O(V^2(V-1)) = O(V^3)이 됩니다. 물론 dense graph가 아닌 경우에는 V^3보다 더 효율적이므로 대회에 사용할 때는 더 깊게 생각해야 합니다.

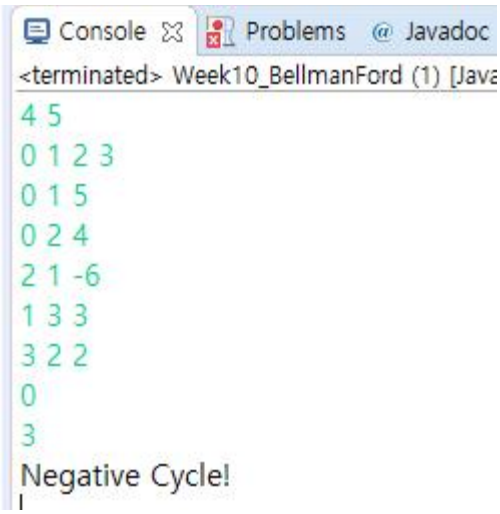
<Bellman Ford 소스코드>

<실행결과>

```java
static Map<String, LinkedList<Pair>> graph = new HashMap<String, LinkedList<Pair>>
public static void bellmanFord(String start, String end) {
    Map<String, Integer> dist = new HashMap<String, Integer>();
    Map<String, String> prev = new HashMap<String, String>();

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

    boolean updated = false;
    for(int i = 0; i <= dist.keySet().size(); i++) {
        updated = false;
        iterV = graph.keySet().iterator();
        while(iterV.hasNext()) {
            String v = iterV.next();
            Iterator<Pair> iterE = graph.get(v).iterator();
            while(iterE.hasNext()) {
                Pair p = iterE.next();
                if(dist.get(v) != INF && dist.get(p.v) > dist.get(v) + p.cost) {
                    dist.put(p.v, dist.get(v) + p.cost);
                    prev.put(p.v, v);
                    updated = true;
                }
            }
        }
    }
```



문제 4 : Astar

문제 : astar 알고리즘으로 Dijkstra보다 더 빠르게 최단 경로를 찾는 문제

해결 방법 :

astar 알고리즘은 Dijkstra와 유사하나, 정점에 포함시키는 순서에 휴리스틱 함수를 고려하여

정점을 최단거리 집합에 추가합니다. 예를 들어서, 원래 Dijkstra 알고리즘은 시작 정점에서부터 가까운 원소를 최단경로 집합에 포함시키므로 시작 정점과 끝 정점의 거리가 먼 경우 비효율적이게 되는데, 휴리스틱 함수를 특정 정점이 끝 정점보다 멀 경우 나중에 정점을 선택하도록 함숫값을 크게 반환하면, 끝 정점에 가까울수록 먼저 탐색합니다. 따라서 다익스트라보다 더 빠르게 경로를 찾을 수 있습니다.

시간복잡도 : Unknown

휴리스틱 함수에 따라서 시간복잡도가 매우 달라질 수 있지만, 일반적으로 실제로 사용할 때는 Dijkstra보다는 빠르게 수행됩니다.

```
public class Vector implements Comparable<Vector>{
    public int v;
    public int cost;
    public int h; //value of heuristic function

    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);
    }
}
static int dist(int a, int b) {
    int x1 = a % w;
    int y1 = a / w;
    int x2 = b % w;
    int y2 = b / w;
    return (int) Math.sqrt(Math.pow(x1-x2, 2) + Math.pow(y1-y2, 2));
}
```

<astar 소스코드>

```
public static void 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)));
            }
        }
    }
    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] = 'P';
        }
        previous = prev.get(previous);
    }

    StringBuilder sb = new StringBuilder();
    for(int i = 0; i < h; i++) {
        for(int j = 0; j < w; j++) {
            sb.append(map[i][j]);
        }
        sb.append("\n");
    }
    System.out.println(sb);
}
```


Console
Problems
Javadoc
<terminated> Week10_AStar (1) [Java Applicati
5 5
WWWWW
WSBEW
WRWRW
WRRRW
WWWWW
WWWWW
WSPEW
WRWRW
WRRRW
WWWWW

<실행결과>