알고리즘 응용
- dijkstra algorithm-

제출일자	2021.05.20
분 반	00
이 름	강인한
학 번	201701969

1	코드 설명

```
for i in range(m):  #노드의 생성과정 4*4의 경우 16개의 노드 생성
    for j in range(n):
        if j+1<m:
            if array[i][j]=='R': # R이라면 오른쪽 노드로의 간선 값 은 0
                graph[i * m + j].append((i * m + j + 1, 0))
            else:   # R이 아니라면 간선 값은 1
                graph[i * m + j].append((i * m + j + 1, 1))
        if j-1>=0:
            if array[i][j]=='L':
                graph[i * m + j].append((i * m + j - 1, 0))
            else:
                graph[i * m + j].append((i * m + j - 1, 1))
        if i-1>=0:
            if array[i][j]=='U':
                graph[i * m + j].append(((i-1) * m + j, 0))
            else:
                graph[i * m + j].append(((i-1) * m + j, 1))
        if i+1<n:
            if array[i][j]=='D':
                graph[i * m + j].append(((i+1) * m + j, 0))
            else:
                graph[i * m + j].append(((i+1) * m + j, 1))
```

먼저 m*n의 노드를 생성해주어야 한다. 이때 이차 배열에서 맞닿아 있는 노드 간 간선의 가중치는 1이고 방향이 가르치고 있다면 그 방향의 간선 가중치는 0이다.

그다음으로는 다익스트라 알고리즘의 구현이다. 노드에서 가장 짧은 거리의 노드를 빠르게 알아내기 위해 최소 힙 구조를 기반으로 한다. 다익스트라 함수에서는 먼저 시작 노드로 가기 위한 거리를 0으로 설정하여 큐에 삽입한다. 그리고 while문을 통해 큐가 비어있지 않을 때까지 반복이 시작된다. 그 후 이미 처리된 노드인지 확인하는 과정이 필요하고 인접한 노드 중에서 현재 노드를 거쳐서 다른 노드로 이동하는 거리가 더 짧은 경우 heappush해준다.

```python
def dijkstra(start):
    q = []
    heapq.heappush(q, (0, start))
    distance[start] = 0
    while q:
        dist, now = heapq.heappop(q)
        if distance[now] < dist:
            continue
        for i in graph[now]:
            cost = dist + i[1]
            if cost < distance[i[0]]:
                distance[i[0]] = cost
                heapq.heappush(q, (cost, i[0]))
```

2	결과 화면

3	시간 복잡도

힙 연산에 대한 시간복잡도는 O(logE)이고 힙 연산을 이용한 다익스트라 알고리즘은 O(ElogV)가 된다.