과제 3 – 프로그래밍 과제

1. Github Link

2. 코드 및 자료구조에 대한 상세설명

   1) Dijikstra 알고리즘이란?

   특정 노드에서 시작하여 인접한 노드 중 가장 가중치가 적은 경로를 탐색하여 모든 노드의 최소 비용 경로를 구하는 알고리즘이다.

   2) 알고리즘 구현 방식

   시작할 특정 노드에서 모든 노드 간의 가중치를 저장하는 배열을 생성하여 인접한 노드들 간의 가중치를 구해 가장 짧은 경로를 해당 배열에 추가해 나가는 방식이다. Priority queue를 사용하여 가장 가중치가 작은 경로를 먼저 추출한다. 여기에서 추출한 경로와 현재 노드에서 이동할 수 있는 노드들 간의 경로를 비교하여 추출된 노드의 가중치가 더 작으면 배열을 업데이트한다.

   3) 코드 구현

   코드를 스스로 처음부터 끝까지 구현하기에 어려움을 겪었기 때문에, 한 블로그에서 코드를 참고하였다. (출처: https://brownbears.tistory.com/554)
   파이썬의 모듈 중 하나인 heapq를 사용하였는데, 이 모듈은 최소 힙을 구현해주는 기능을 한다. 인자로 넘겨받은 리스트를 최소 힙으로 재구성해준다. 코드에 대한 설명은 이미지 속 주석으로 표시하였다.

```python
def dijkstra(start):
    # 초기 배열 설정, 최소 가중치(거리)들의 리스트이다.
    distances = {node: sys.maxsize for node in graph}
    # 시작 노드의 거리는 0으로 설정
    distances[start] = 0
    # priority queue
    queue = []
    # 시작 노드부터 탐색 시작 하기 위함. heapq 모듈은 첫번째 데이터를 기준으로 정렬하기 때문에
    # 거리 값을 기준으로 정렬하기 위해 (거리, 노드) 순으로 queue에 저장한다.
    heapq.heappush(queue, (distances[start], start))

    # priority queue에 데이터가 하나도 없을 때까지 반복
    while queue:
        # 가장 낮은 거리를 가진 노드와 그 거리를 추출
        current_distance, node = heapq.heappop(queue)
        # 이미 계산되어 저장한 거리와 추출된 거리와 비교하여 저장된 거리가 더 작다면 비교하지 않고
        # 큐의 다음 데이터로 넘어간다.
        if distances[node] < current_distance:
            continue

        # 대상인 노드에서 인접한 노드와 그 거리를 탐색한다.
        for adjacency_node, distance in graph[node].items():
            # 현재 노드에서 인접한 노드를 지나갈 때까지의 거리를 더하여 가중치로 저장한다.
            weighted_distance = current_distance + distance
            # 배열에 저장된 거리보다 위의 가중치가 더 작을 경우
            if weighted_distance < distances[adjacency_node]:
                # 배열에 저장된 거리를 더 작은 가중치의 값으로 변경
                distances[adjacency_node] = weighted_distance
                # 다음 인접 거리를 계산 하기 위해 해당 거리와 노드를 priority queue에 삽입
                heapq.heappush(queue, (weighted_distance, adjacency_node))

    return distances
```

3. Test-case 실행결과

그래프는 딕셔너리의 형태로 각 노드별로 연결된 노드와, 그 사이의 가중치를 연결하여 표현하였고, test case를 임의로 만들어 실행하였다.

추가적으로 프로그램 실행 결과와 실제 최단거리 계산 결과값이 맞는지 확인해보았다. (다음 페이지)

Test case 1)

```
graph = {
    'A': {'B': 10, 'C': 3},
    'B': {'C': 1, 'D': 2},
    'C': {'B': 4, 'D': 8, 'E': 2},
    'D': {'E': 7},
    'E': {'D': 9},
}

result = dijkstra('A')
print(result)
```

```
{'A': 0, 'C': 3, 'B': 7, 'E': 5, 'D': 9}
```

Test case 2)

```
graph = {
    'A': {'B': 5, 'D':2, 'E':4},
    'B': {'C':3,'D':3},
    'C': {'E':4},
    'D': {'C':3 },
    'E': {'D':1},
}

result = dijkstra('A')
print(result)
```

```
{'A': 0, 'C': 5, 'B': 5, 'E': 4, 'D': 2}
```

Test case 3)

```
graph = {
    'A': {'B': 7,'C':4, 'D':6, 'E':1},
    'B': {},
    'C': {'B':2, 'D': 5},
    'D': {'B':3 },
    'E': {'D':1},
}

result = dijkstra('A')
print(result)
```

```
{'A': 0, 'C': 4, 'B': 5, 'E': 1, 'D': 2}
```