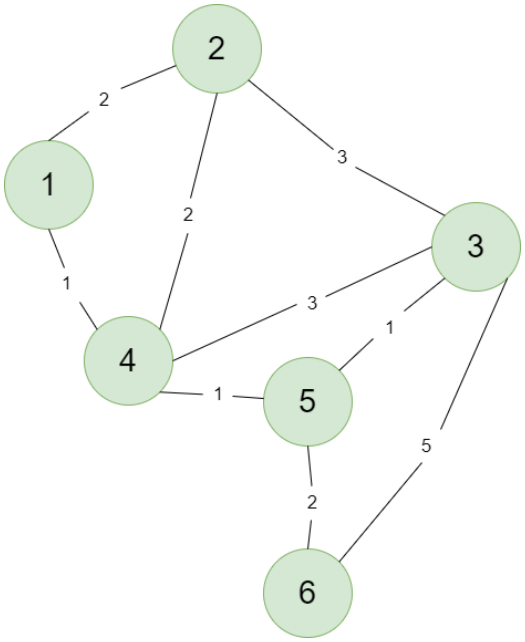
과제

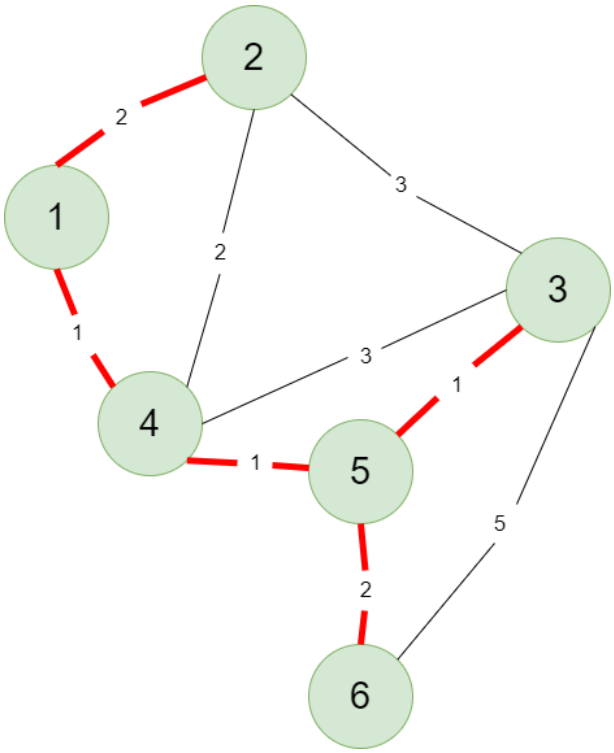
장유선

2023.08.30

1. 문제 정의



다익스트라 알고리즘을 구현하고 위의 예제의 그래프에 대해서 적용



2. 개념 설명

2.1 다익스트라 알고리즘 소개

다익스트라(Dijkstra) 알고리즘은 그래프에서 최단 경로를 찾는 데 사용되는 알고리즘 중 하나이다. 주어진 그래프에서 특정 시작 노드로부터 다른 모든 노드까지의 최단 경로를 찾는 알고리즘.

2.2 알고리즘 원리

1. 시작 노드로부터의 거리를 무한대로 초기화하고, 시작 노드의 거리를 0으로 설정한다.
2. 아직 방문하지 않은 노드 중에서 가장 거리가 짧은 노드를 선택한다.
3. 선택한 노드를 경유하여 다른 노드로 가는 거리를 계산하고, 이 거리가 기존에 알고 있는 거리보다 짧으면 최단 거리를 갱신한다.
4. 모든 노드를 방문할 때까지 2-3단계를 반복한다.
5. 최종적으로 시작 노드로부터 각 노드까지의 최단 거리를 계산한다.

3. Python Code Hard Copy

```python
def dijkstra(graph, start):

    distances = {node: float('inf') for node in graph}
    distances[start] = 0

    unvisited_nodes = list(graph.keys())

    while unvisited_nodes:
        current_node = None

        for node in unvisited_nodes:
            if current_node is None:
                current_node = node
            elif distances[node] < distances[current_node]:
                current_node = node

        if distances[current_node] == float('inf'):
            break

        unvisited_nodes.remove(current_node)

        for neighbor, weight in graph[current_node].items():
            candidate_distance = distances[current_node] + weight

            if candidate_distance < distances[neighbor]:
                distances[neighbor] = candidate_distance
```

```
    return distances

graph = {
    '1': {'2': 2, '4': 1},
    '2': {'3': 3, '4': 2},
    '3': {'2': 3, '5': 1, '6': 5},
    '4': {'2': 2, '5': 1},
    '5': {'3': 1, '4': 1, '6': 2},
    '6': {'3': 5, '5': 2}
}

start_node = '1'
shortest_distances = dijkstra(graph, start_node)
print(shortest_distances)
```

4. Code 설명

4-1. 변수 초기화

```
    distances = {node: float('inf') for node in graph}
    distances[start] = 0  # 시작 노드의 거리는 0
    unvisited_nodes = list(graph.keys())
```

- 각 노드로부터의 최단 거리를 저장할 딕셔너리 초기화
- 초기 거리를 무한대(float('inf'))로 설정
- 시작 노드의 거리는 0
- unvisited_nodes : 아직 방문하지 않은 노드들을 저장할 리스트 초기화

4-2. 시작 노드로부터 모든 노드까지의 최단 거리를 계산

```
    while unvisited_nodes:
        current_node = None

        for node in unvisited_nodes:
            if current_node is None:
                current_node = node
            elif distances[node] < distances[current_node]:
                current_node = node

        if distances[current_node] == float('inf'):
            break
```

```
        unvisited_nodes.remove(current_node)
```

- current_node를 None으로 초기화
- 방문하지 않은 노드(unvisited_nodes)를 방문
- current_node가 아직 설정되지 않았다면, 현재 노드를 current_node로 설정
- current_node가 설정되어 있고, 현재 노드까지의 거리(distances[node])가 current_node까지의 거리(distances[current_node])보다 더 짧다면, current_node를 현재 노드로 업데이트
- current_node가 선택되면 해당 노드를 방문한 것으로 표시 -> unvisited_nodes에서 제거
- current_node로부터의 최단 거리가 무한대인 경우 -> 최단 경로를 찾을 필요가 없으므로 알고리즘을 종료

4-3. 최단 거리 갱신

```
for neighbor, weight in graph[current_node].items():
            candidate_distance = distances[current_node] + weight

            if candidate_distance < distances[neighbor]:
                distances[neighbor] = candidate_distance

    return distances
```

- graph[current_node].items() : 현재 노드(current_node)의 인접한 노드(neighbor)와 해당 엣지의 weight를 순회
- candidate_distance: 현재 노드까지의 거리(distances[current_node])에 인접 노드로 가는 엣지의 weight를 더한 값
- candidate_distance가 더 짧다 = 현재까지 알고 있는 최단 거리보다 더 좋은 경로
- distances[neighbor]를 candidate_distance로 업데이트

5. 결과

```
{'1': 0, '2': 2, '3': 3, '4': 1, '5': 2, '6': 4}
```

6. 결과 화면