최단경로 알고리즘의 이해

1. 최단 경로 문제란?

- 최단 경로 문제란 두 노드를 잇는 가장 짧은 경로를 찾는 문제임
- 가중치 그래프(Weighted Graph)에서 간선(Edge)의 가중치 합이 최소가 되도록 하는 경로를 찾는 것이 목적

최단 경로 문제 종류

1. 단일 출발 및 단일 도착(single-source and single-dsetination shortest path problem)최단 경로 문제
    - 그래프 내의 특정 노드 u에서 출발, 또다른 특정 노드 v에 도착하는 가장 짧은 경로를 찾는 문제
2. 단일 출발(single-source shortest path problem)최단 경로 문제
    - 그래프 내의 특정 노드 u와 그래프 내 다른 모든노드 각각의 가장 짧은 경로를 찾는 문제
3. 전체 쌍(all-pair)최단 경로 : 그래프 내의 모든 노드 쌍(u, v)에 대한 최단 경로를 찾는 문제

---

2. 최단 경로 알고리즘 - 다익스트라 알고리즘

- 다익스트라 알고리즘은 위의 최단 경로 문제 종류 중, 2번에 해당
    - 하나의 정점에서 다른 모든 정점 간의 각각 **가장 짧은 거리**를 구하는 문제

다익스트라 알고리즘 로직

- 첫 정점을 기준으로 연결되어 있는 정점들을 추가해 가며, 최단 거리를 갱신하는 기법
- 다익스트라 알고리즘은 너비우선탐색(BFS)와 유사
    - 첫 정점부터 각 노드간의 거리를 저장하는 배열을 만든 후, 첫 정점의 인접 노드 간의 거리부터 먼저 계산하면서, 첫 정점부터 해당 노드간의 가장 짧은 거리를 해당 배열에 업데이트

  다익스트라 알고리즘의 다양한 변형 로직이 있지만, 가장 개선된 우선순위 큐를 사용하는 방식에 집중!
- 우선순위 큐를 활용한 다익스트라 알고리즘
    - 우선순위 큐는 MinHeap 방식을 활용해서, 현재 가장 짧은 거리를 가진 노드 정보를 먼저 꺼내개됨

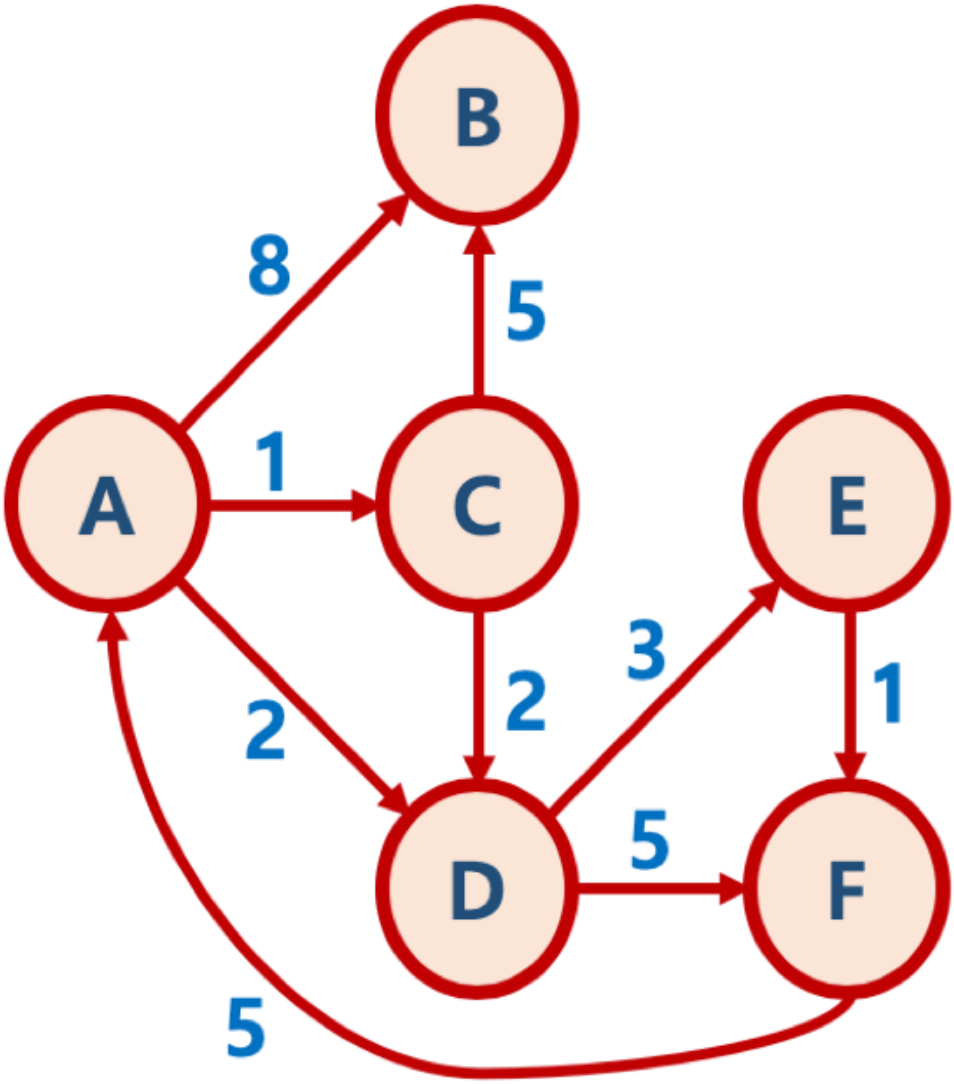
    1. 첫 정점을 기준으로 배열을 선언하여 첫 정점에서 각 정점까지의 거리를 저장
        - 초기에는 첫 정점의 거리는 0, 나머지는 무한대로 저장함 (inf라고 표현함)

- 우선순위 큐에 (첫 정점, 거리 0)만 먼저 넣음

2. 우선순위 큐에서 노드를 꺼냄
    - 처음에는 첫 정점만 저장되어 있으므로, 첫 정점이 꺼내짐
    - 첫 정점에 인접한 노드들 각각에 대해, 첫 정점에서 각 노드로 가는 거리와 현재 배열에 저장되어 있는 첫 정점에서 각 정점까지의 거리를 비교한다.
    - 배열에 저장되어 있는 거리보다, 첫 정점에서 해당 노드로 가는 거리가 더 짧을 경우, 배열에 해당 노드의 거리를 업데이트 한다.
    - 배열에 해당 노드의 거리가 업데이트된 경우, 우선순위 큐에 넣는다.
        - 결과적으로 너비 우선 탐색 방식과 유사하게, 첫 정점에 인접한 노드들을 순차적으로 방문하게 됨
        - 만약 배열에 기록된 현재까지 발견된 가장짧은 거리보다, 더 긴 거리(루트)를 가진 노드의 경우에는 해당 노드와 인접한 노드간의 거리 계산을 하지 않음
3. 2번의 과정을 우선순위 큐에 꺼낼 노드가 없을 때까지 반복한다.

---

3. 예제로 이해하는 다익스트라 알고리즘(우선순위 큐 활용)



1단계 : 초기화

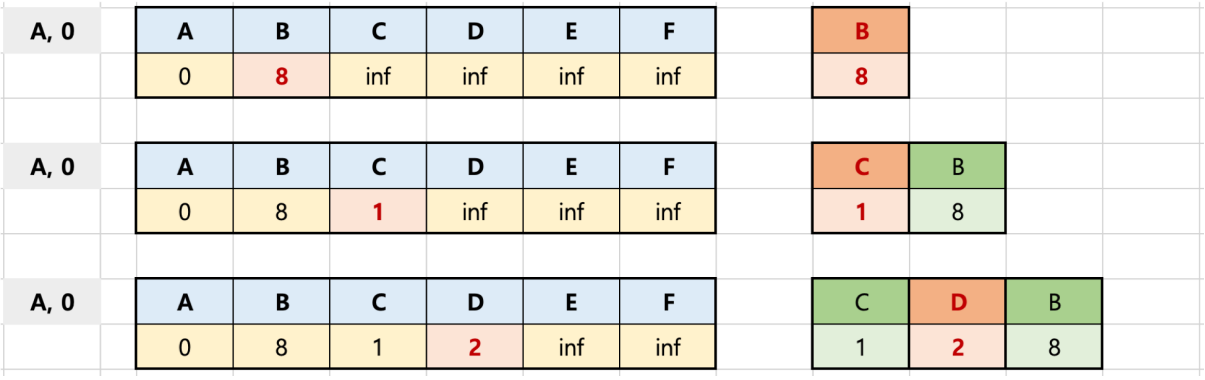
- 첫 정점을 기준으로 배열을 선언하여 첫 정점에서 각 정점까지의 거리를 저장
  - 초기에는 첫 정점의 거리는 0, 나머지는 무한대로 저장함 (inf 라고 표현함)
  - 우선순위 큐에 (첫 정점, 거리 0)만 먼저 넣음

큐에서 추출	거리 저장 배열						우선순위 큐
	A	B	C	D	E	F	A
	0	inf	inf	inf	inf	inf	0

2단계 : 우선순위 큐에서 추출한 (A, 0) [노드, 첫 노드와의 거리] 를 기반으로 인접한 노드와의 거리 계산

- 우선순위 큐에서 노드를 꺼냄
  - 처음에는 첫 정점만 저장되어 있으므로, 첫 정점이 꺼내짐
  - 첫 정점에 인접한 노드들 각각에 대해, 첫 정점에서 각 노드로 가는 거리와 현재 배열에 저장되어 있는 첫 정점에서 각 정점까지의 거리를 비교한다.
  - 배열에 저장되어 있는 거리보다, 첫 정점에서 해당 노드로 가는 거리가 더 짧을 경우, 배열에 해당 노드의 거리를 업데이트한다.
  - 배열에 해당 노드의 거리가 업데이트된 경우, 우선순위 큐에 넣는다.
    - 결과적으로 너비 우선 탐색 방식과 유사하게, 첫 정점에 인접한 노드들을 순차적으로 방문하게 됨
    - 만약 배열에 기록된 현재까지 발견된 가장 짧은 거리보다, 더 긴 거리(루트)를 가진 (노드, 거리)의 경우에는 해당 노드와 인접한 노드간의 거리 계산을 하지 않음

  이전 표에서 보듯이, 첫 정점 이외에 모두 inf 였었으므로, 첫 정점에 인접한 노드들은 모두 우선순위 큐에 들어가고, 첫 정점과 인접한 노드간의 거리가 배열에 업데이트됨



A, 0	A	B	C	D	E	F
	0	8	inf	inf	inf	inf

B
8

A, 0	A	B	C	D	E	F
	0	8	1	inf	inf	inf

C	B
1	8

A, 0	A	B	C	D	E	F
	0	8	1	2	inf	inf

C	D	B
1	2	8

3단계 : 우선순위 큐에서 (C, 1) [노드, 첫 노드와의 거리] 를 기반으로 인접한 노드와의 거리 계산

- 우선순위 큐가 MinHeap(최소 힙) 방식이므로, 위 표에서 넣어진 (C, 1), (D, 2), (B, 8) 중 (C, 1) 이 먼저 추출됨(pop)
- 위 표에서 보듯이 1단계까지의 A - B 최단 거리는 8 인 상황임
  - A - C 까지의 거리는 1, C 에 인접한 B, D에서 C - B는 5, 즉 A - C - B 는 1 + 5 = 6 이므로, A - B 최단 거리 8보다 더 작은 거리를 발견, 이를 배열에 업데이트
    - 배열에 업데이트했으므로 B, 6 (즉 A에서 B까지의 현재까지 발견한 최단 거리) 값이 우선순위 큐에 넣어짐
  - C - D 의 거리는 2, 즉 A - C - D 는 1 + 2 = 3 이므로, A - D의 현재 최단 거리인 2 보다 긴 거리, 그래서 D 의 거리는 업데이트되지 않음

C, 1	A	B	C	D	E	F
	0	6	1	2	inf	inf

D	B	B
2	6	8

4단계 : 우선순위 큐에서 (D, 2) [노드, 첫 노드와의 거리] 를 기반으로 인접한 노드와의 거리 계산

- 지금까지 접근하지 못했던 E와 F 거리가 계산됨
  - A - D까지의 거리인 2에 D - E 가 3 이므로 이를 더해서 E, 5
  - A - D까지의 거리인 2에 D - F 가 5 이므로 이를 더해서 F, 7

D, 2	A	B	C	D	E	F
	0	6	1	2	5	inf

E	B	B
5	6	8

D, 2	A	B	C	D	E	F
	0	6	1	2	5	7

E	B	F	B
5	6	7	8

5단계 : 우선순위 큐에서 (E, 5) [노드, 첫 노드와의 거리] 를 기반으로 인접한 노드와의 거리 계산

- A - E 거리가 5인 상태에서, E에 인접한 F를 가는 거리는 1, 즉 A - E - F 는 5 + 1 = 6, 현재 배열에 A - F 최단거리가 7로 기록되어 있으므로, F, 6 으로 업데이트
- 우선순위 큐에 F, 6 추가

E, 5	A	B	C	D	E	F
	0	6	1	2	5	6

B	F	F	B
6	6	7	8

6단계: 우선순위 큐에서 (B, 6), (F, 6) 를 순차적으로 추출해 각 노드 기반으로 인접한 노드와의 거리 계산

- 예제의 방향 그래프에서 B 노드는 다른 노드로 가는 루트가 없음
- F 노드는 A 노드로 가는 루트가 있으나, 현재 A - A 가 0 인 반면에 A - F - A 는 6 + 5 = 11, 즉 더 긴 거리이므로 업데이트되지 않음

B, 6	A	B	C	D	E	F
	0	6	1	2	5	6

F	F	B
6	7	8

F, 6	A	B	C	D	E	F
	0	6	1	2	5	6

F	B
7	8

7단계: 우선순위 큐에서 (F, 7), (B, 8) 를 순차적으로 추출해 각 노드 기반으로 인접한 노드와의 거리 계산

- A - F 로 가는 하나의 루트의 거리가 7 인 상황이나, 배열에서 이미 A - F 로 가는 현재의 최단 거리가 6인 루트의 값이 있는 상황이므로, 더 긴거리인 F, 7 루트 기반 인접 노드까지의 거리는 계산할 필요가 없음, 그래서 계산없이 스킵함
  - 계산하더라도 A - F 거리가 6인 루트보다 무조건 더 긴거리가 나올 수 밖에 없음
- B, 8 도 현재 A - B 거리가 6이므로, 인접 노드 거리 계산이 필요 없음.

> 우선순위 큐를 사용하면 불필요한 계산 과정을 줄일 수 있음

F, 7	A	B	C	D	E	F
	0	6	1	2	5	6

B
8

B, 8	A	B	C	D	E	F
	0	6	1	2	5	6

우선순위 큐 사용 장점

- 지금까지 발견된 가장 짧은 거리의 노드에 대해서 먼저 계산
- 더 긴 거리로 계산된 루트에 대해서는 계산을 스킵할 수 있음

4. 다익스트라 알고리즘 파이썬 구현 (우선순위 큐 활용까지 포함)

- heapq 라이브러리 활용을 통해 우선순위 큐 사용하기
  - 데이터가 리스트 형태일 경우, 0번 인덱스를 우선순위로 인지, 우선순위가 낮은 순서대로 pop 할 수 있음

```
import heapq

queue = []

heapq.heappush(queue,[2,'A'])
heapq.heappush(queue,[5,'B'])
```

```
heapq.heappush(queue,[1,'C'])
heapq.heappush(queue,[7,'D'])
print(queue)

for index in range(len(queue)):
  print(heapq.heappop(queue))

>>
[[1, 'C'], [5, 'B'], [2, 'A'], [7, 'D']]
[1, 'C']
[2, 'A']
[5, 'B']
[7, 'D']
```

다익스트라 알고리즘

- 탐색할 그래프의 시작 정점과 다른 정점들간의 최단 거리 구하기

```
mygraph = {
    'A': {'B': 8, 'C': 1, 'D': 2},
    'B': {},
    'C': {'B': 5, 'D': 2},
    'D': {'E': 3, 'F': 5},
    'E': {'F': 1},
    'F': {'A': 5}
}
```

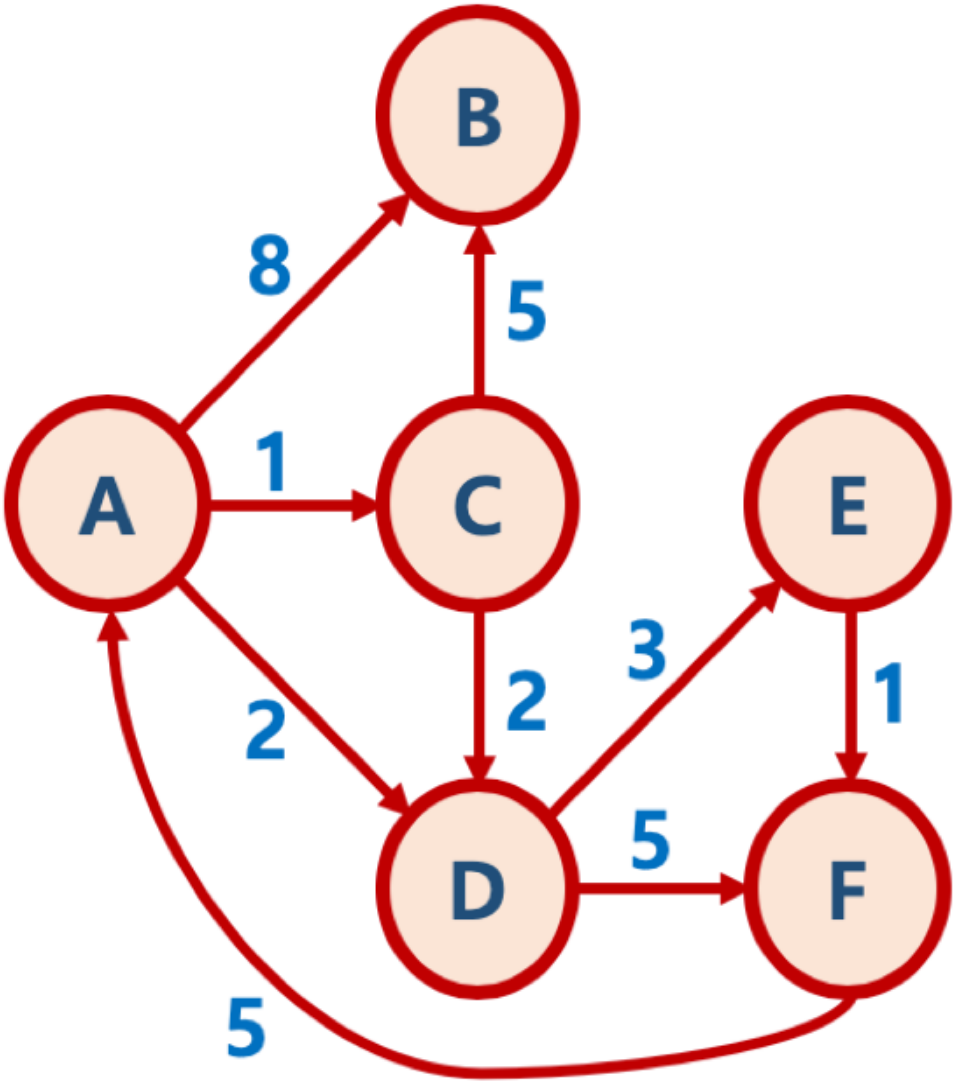
```
import heapq

def dijkstra(graph, start):
  distances = {node : float('inf') for node in graph}
  distances[start] = 0
  queue=[]
  heapq.heqppush(queue,[distances[start],start]

  while queue:
    current_distance, current_node = heapq.heappop(queue)

    if distances[current_node] < current_distance:#continue문을 실행하면 밑에를 실행하지 않고 while문으로 돌아감
      continue
    for adjacent, weight in graph[current_node].items():
      distance = current_distance + weight

      if distance < distances[adjacent]:
        distances[adjacent] = distance
        heapq.heappush(queue,[distance,adjacent])
  return distances
```




```
dijkstra(mygraph, 'A')
>>{'A': 0, 'B': 6, 'C': 1, 'D': 2, 'E': 5, 'F': 6}
```

5. 시간 복잡도

- 위 다익스트라 알고리즘은 크게 다음 두 가지 과정을 거침
  - 과정 1 : 각 노드마다 인접한 간선들을 모두 검사하는 과정
  - 가정 2 : 우선순위 큐에 노드/거리 정보를 넣고 삭제(pop)하는 과정
- 각 과정별 시간 복잡도
  - 과정 1 : 각 노드는 최대 한 번씩 방문하므로 (첫 노드와 해당 노드간의 갈 수 있는 루트가 있는 경우만 해당), 그래프의 모든 간선은 최대 한 번씩 검사

        - 즉, 각 노드마다 인접한 간선들을 모두 검사하는 과정은 O(E) 시간이 걸림, E는 간선(edge)의 약자
    - 과정 2 : 우선순위 큐에 가장 많은 노드, 거리 정보가 들어가는 경우, 우선순위 큐에 노드/거리 정보를 넣고, 삭제하는 과정이 최악의 시간이 걸림
        - 우선순위 큐에 가장 많은 노드, 거리 정보가 들어가는 시나리오는 그래프의 모든 간선이 검사될 때마다, 배열의 최단 거리가 갱신되고, 우선순위 큐에 노드/거리가 추가되는 것임
        - 이때 추가는 각 각선마다 최대 한 번 일어날 수 있으므로, 최대 O(E)의 시간이 걸리고, O(E)개의 노드/거리 정보에 대한 우선순위 큐를 유지하는 작업은 O(logE)가 걸림
            - 따라서 해당 과정의 시간 복잡도는 O(ElogE)

총 시간 복잡도

과정1 + 과정2 = O(E) + $O(ElogE)$ = $O(E+ElogE)$=$O(ElogE)$