-플로이드 워셜(O(N3)) : 모든 쌍 간의 최단경로를 구함. 다익스트라를 모든 노드에 적용시켜도 되지만,

모든 쌍을 구할 때는 플로이드 워셜법이 더 빠르다. 밑처럼 구현.

for k in range(4): # 거쳐가는 노드 k

for s in range(4): # 출발 노드 s tart

for e in range(4): # 도착 노드 e nd

DP\_Array[s][e]= min(DP\_Array[s][e],DP\_Array[s][k]+DP\_Array[k][e])

-다익스트라 : 한 시작정점에서 다른 모든 정점으로의 최단 경로를 구함.

음수 가중치가 없는 그래프에서만 사용가능.

BFS이며, 힙 또는 우선순위 큐 응용 / 그리디

1. 현재 검사 노드를 A라고 함.
2. A에서 방문완료되지 않은 모든 정점을 차례로 검사

d[A] + p[A][X] < d[X] 이면 d[X] 업데이트

1. 검사가 끝나면 A는 방문완료 표시. 다시는 방문하지 않음
2. 방문하지않은 모든 정점 중 값이 가장 낮은 정점을 우선순위 큐로 뽑고 A에 등록
3. 모든 정점을 방문할 때까지 반복
4. 업데이트 시 업데이트 하는 노드에 자기를 기록하여, 모든 노드가 자기 이전 노드를 기록하고 있으면 경로도 구할 수 있다.

-밸만포드 : 한 정점에서 다른 모든 정점으로의 최단 경로를 구함

음수 가중치가 있는 그래프에서도 사용가능

음수 사이클을 확인 가능

1. 시작 정점을 제외한 모든 정점을 INF 로 초기화
2. INF 가 아닌 모든 정점을 검사 -> 인접한 정점으로 가서 현재 + 인접이 인접에 지금 저장돼있는 것보다 작으면 업데이트.
3. 2 과정을 (정점 개수 – 1) 번 반복
4. 2. 를 다시 시행. 만약 다시 업데이트 되면, 음수 사이클이 있는 것.

의사코드

for \_ in range(len(graph) - 1):

for node in graph:

for neighbour in graph[node]:

if distance[neighbour] > distance[node] + graph[node][neighbour]:

distance[neighbour] = distance[node] + graph[node][neighbour]

for node in graph:

for neighbour in graph[node]:

if distance[neighbour] > distance[node] + graph[node][neighbour]:

return -1

다익스트라와 마찬가지로, 자기 이전 노드를 기록하여 경로를 구할 수 있음.