202301558 김결 컴퓨터 알고리즘 과제

9주차 과제 / 동적 계획 알고리즘 – 모든 쌍 최단 경로 문제

모든 쌍 최단 경로 문제는 말 그대로 모든 쌍에 대해서 최단 경로를 구하는 문제이다. n개의 점에 대하여 n x n의 표를 채우는 거라고 생각하면 된다. 물론 대각선을 기준으로 값이 똑같기 때문에 위쪽에만 채우면 된다.(강의 자료 7쪽 그림처럼)

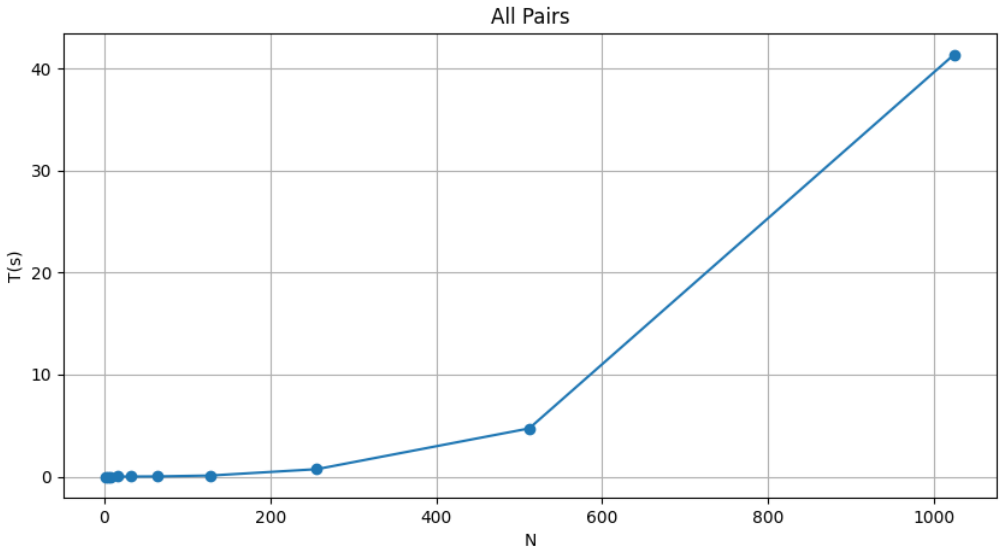
1. 다익스트라 알고리즘을 n번 사용하기

앞서 배운 다익스트라 알고리즘은 한 점에 대하여 최단경로들을 찾아주는 알고리즘이다. 표로 생각하면 한 열을 채우는 것이다. 그러면 이걸 n번 돌려주면 표가 다 채워질 것이다. 당연하게도 O(n^2)짜리 알고리즘을 n번 반복하니까 시간복잡도는 O(n^3)이다.

강의자료상에서 바로 다른 알고리즘이 나오길래 구현은 건너뛰려고 했는데, O(n^3)짜리 알고리즘의 성능 분석 그래프를 그리면 얼마나 노트북이 좋아할지 궁금해서 해 보았다.

```
# 그래프를 받아오는 def
# 배열에 점들을 집어넣기
# 거리를 저장해줄 배열
# 모든 점들에 대해서 for, 다익스트라 호출
# 리턴
```

의사코드인데 너무 간단해서 바로 성능 그래프를 첨부한다.(다익스트라는 깃허브에서 주웠다)

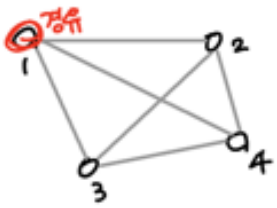


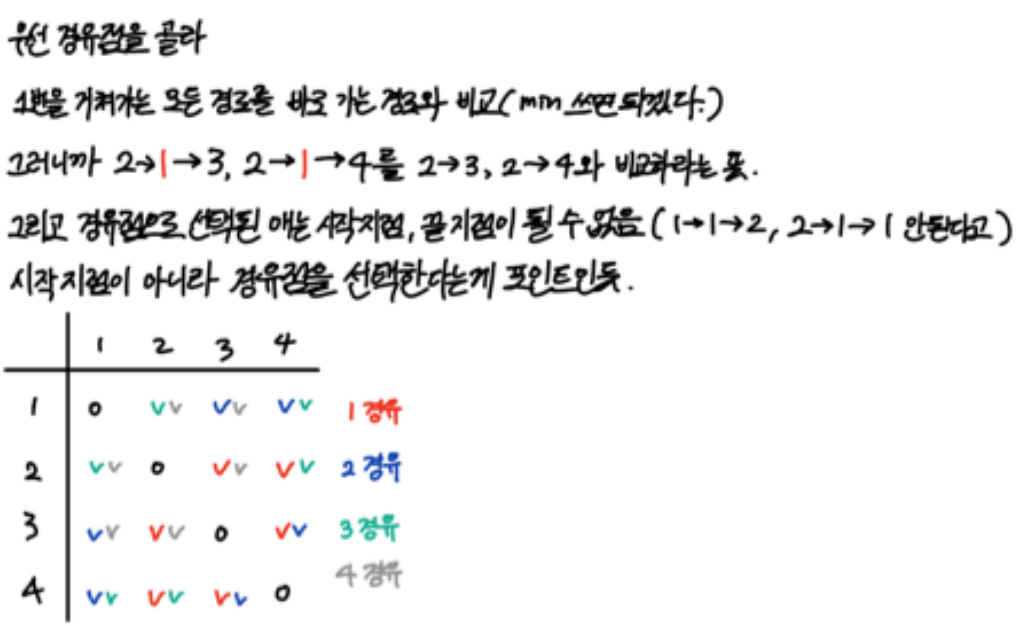
2의 13승까지 돌려보려고 했는데 역시 강종을 때려버려서 실패했다. 1024까지가 한계인 것 같다.

202301558 김결 컴퓨터 알고리즘 과제

2. 플로이드-워샬 알고리즘

강의 자료로는 이해가 잘 안 돼서 깃허브에서 도움을 받았다. 이해한 대로 필기를 해 보았다.





그리고 경로를 만들 수 없는 경우 무한대로 설정한다고 한다

아래는 내가 적은 의사 코드이다

```
# 가중치 정보, 노드 정보를 담은 배열을 받는 def
# n개의 경유점에 대하여(아우터 루프)
# n개의 시작점에 대하여(이너 루프)
# n개의 도착점에서(이너이너 루프...?ㅋㅋ)
# 바로 가는 길 vs 경유점 거쳐 가는 길
```

그리고 강의 자료를 확인했는데 아주 비슷해서 바로 구현했다. 강의 자료에 나온 대로 간단해서 편하게 구현할 수 있었다.

```
inf = float('inf')
```

202301558 김결 컴퓨터 알고리즘 과제

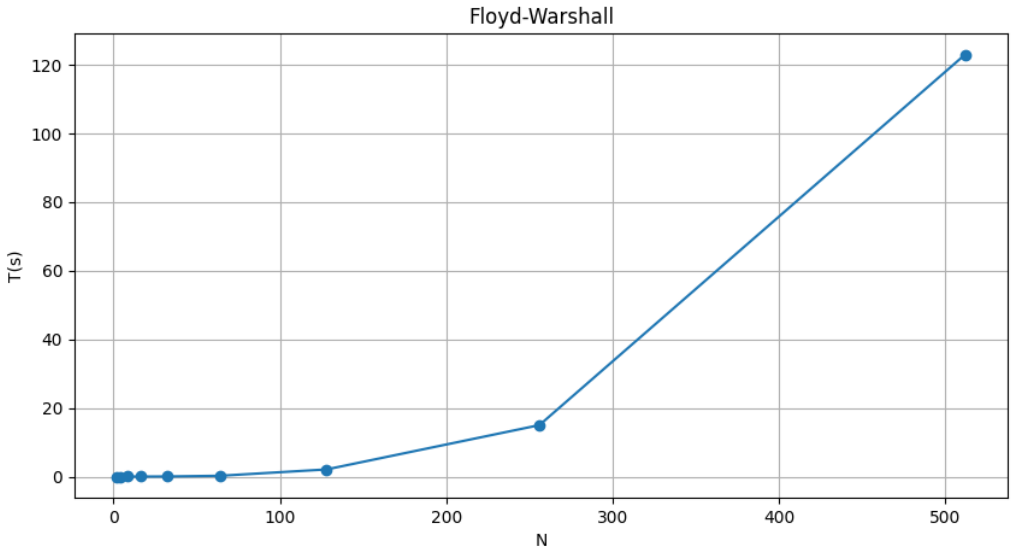
```
n = 5
D = [[inf] * n for _ in range(n)]

for i in range(n):
    D[i][i] = 0

D[0][1] = 4
D[0][2] = 2
D[0][3] = 5

D[1][2] = 1
D[1][4] = 5

D[2][0] = 1
D[2][1] = 3
D[2][3] = 1
D[2][4] = 2

D[3][0] = -2
D[3][4] = 2

D[4][1] = -3
D[4][2] = 3
D[4][3] = 1

def allpair(D):
    for k in range(0, n):
        for a in range(0, n):
            for b in range(0, n):
                D[a][b] = min(D[a][b], D[a][k] + D[k][b])
    return D

print(D)
allpair(D)
print(D)
```

(강의자료와 똑같이 하려고 했는데 강의 자료엔 0번 노드가 없다는걸 지금 깨달았다.)

출력 결과도 강의자료와 똑같이 예쁘게 잘 나왔다.

```
[[0, 4, 2, 5, inf], [inf, 0, 1, inf, 5], [1, 3, 0, 1, 2], [-2, inf, inf, 0, 2], [inf, -3, 3, 1, 0]]
[[0, 1, 2, 3, 4], [0, 0, 1, 2, 3], [-1, -1, 0, 1, 2], [-2, -1, 0, 0, 2], [-3, -3, -2, -1, 0]]
```

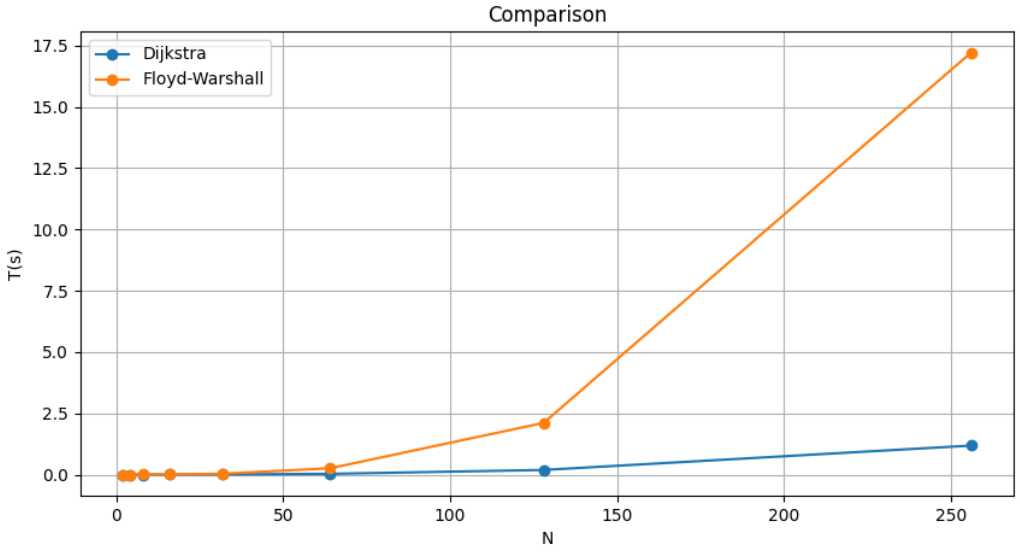
이제 성능을 분석해볼 건데, 코드를 보면 누가 봐도 O(n^3)짜리이다. For이 세 개 있으니까 당연한 거다. 뭐 알아볼 것도 없다.



얘도 1024개까지 돌려보려고 했는데 하루 종일 무한 작동중이라서 512개까지로 줄였다. n^3 짜리는 정말 엄청나다는 교훈을 얻었다... 세상에...

3. 다익스트라 vs 플로이드 워셜

이건 그냥 혼자 해본 공부인데, 둘 다 O(n^3)짜리라면 그래프가 완전히 겹치거나 비슷하게 나오려나? 싶어서 같이 그려 보았다. 그런데 아래처럼 꽤 속도 차이가 나길래 원인을 좀 찾아보았다.



202301558 김결 컴퓨터 알고리즘 과제

(우선 내 생각에는 N수가 작은 게 가장 큰 원인 같기는 하다.)

https://www.researchgate.net/post/Floyd-Warshall_algorithm_or_Dijkstras_Algorithm

여기에 달린 답변들을 읽어보고 서치해본 결과, 플로이드 워셜 알고리즘 같은 경우에 희소그래프에서 보여주는 성능이 좋지 않다고 한다. 스펄스할 경우 힙을 사용하는 다익스트라와 달리 플로이드 같은 경우엔 for을 세번 돌면서 무의미한 계산을 해야 하기 때문인 것 같다. 플로이드 워셜 그래프의 성능을 구할때 랜덤 그래프 생성 함수를 주워서 사용했는데, 이때 스펄스한 그래프들이 만들어졌을 수도 있겠다. 실제로 몇 번 더 돌려 봤더니 간격이 줄기도 하고 더 심해지기도 하고 그랬다. 또, 메모리 공간을 다익스트라보다 많이 잡아먹는다는 단점도 있었다.(https://www.geeksforgeeks.org/time-and-space-complexity-of-floyd-warshall-algorithm/)

반대로 플로이드 워셜은 훨씬 넓은 범위에서 사용할 수 있고 적용하기도 편하다고 하는데, 가장 큰 강점은 음수를 가질 때도 적용이 가능하다는 거다. https://www.baeldung.com/cs/dijkstra-negative-weights 이 글은 다익스트라 알고리즘은 음의 가중치가 있을 때 무한 싸이클의 생성과 이상한 결과를 도출할 수 있음을 안내해 주고 있다. 반면 플로이드 워셜 같은 경우 힙을 사용하는게 아니라 모든 경우에 대해 단순히 비교연산을 하기 때문에, 방문한 정점을 다시 방문해서 최솟값을 찾아내는 다익스트라와는 다르다.(https://stackoverflow.com/questions/22891420/why-do-all-pair-shortest-path-algorithms-work-with-negative-weights 이 글을 참고했다)

마지막으로 공부하다가 찾은 페이퍼(https://arxiv.org/pdf/2109.01872)에서, 플로이드 워셜 알고리즘을 발전시킬 수 있다는 정보를 찾았다. 무한대로 설정되어 있는(경로를 만들 수 없는) 경우에 대해 비교하는 연산을 없애기, 그리고 더 발전시켜서 이너루프들이 도는 수를 조절하는 방법으로 성능 향상을 볼 수 있다고 한다. 무슨 소리인지 완벽하게 알아들을 수는 없었지만... 이런게 있다는 걸 알아간다.