004

다수 인원 대상 지하철 역 중간지점 찾기

불사조팀

2018112025 박익범 (팀장)
2018112008 김균호
2020112377 김민수
2020213314 한지윤

CONTENTS

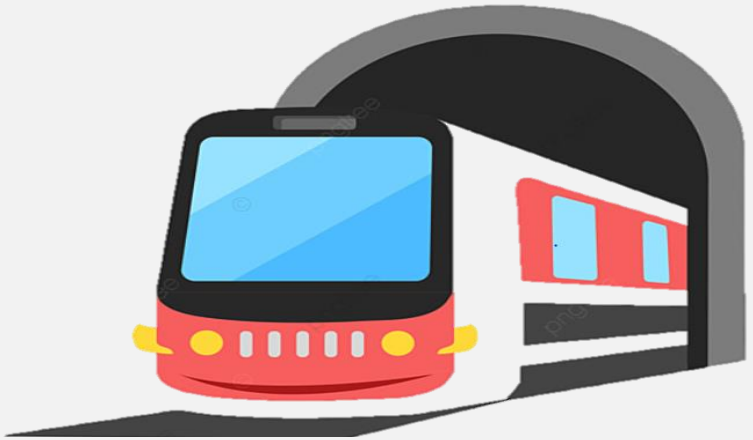
1 서론

[주제]

N명의 사람들이 모임을 가지고자 할 때,

개인이 선호하는 지하철역을 입력 받아

가장 적합한 중간 지점의 지하철역을 구해준다!

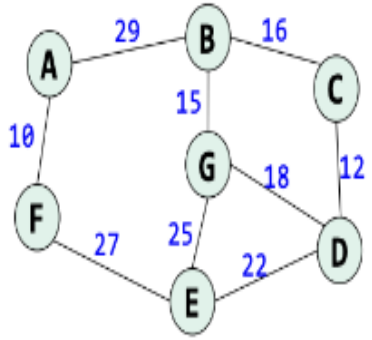


1 서론

[지하철역 데이터 입력하기]

필요한 데이터 : 지하철 역의 이름, 환승 지점, 역 사이의 거리, 지하철 역의 위도와 경도

인접 행렬을 이용한 가중치 그래프의 표현 예



(a) 가중치 그래프

	A	B	C	D	E	F	G
A	0	29	∞	∞	∞	10	∞
B	29	0	16	∞	∞	∞	15
C	∞	16	0	12	∞	∞	∞
D	∞	∞	12	0	22	∞	18
E	∞	∞	∞	22	0	27	25
F	10	∞	∞	∞	27	0	∞
G	∞	15	∞	18	25	∞	0

(b) 인접 행렬을 이용한 표현

동대입구	37.55905041	127.0052968
잠실새내	37.51160835	127.0863007
당산	37.5347735	126.9026073
약수	37.55449098	127.0108871
충정로	37.55974797	126.9644842
녹번	37.60080316	126.9358144

<역의 위도와 경도를 데이터를 입력한 TXT 파일>

1 서론

	A	B	C	D	E	F
1		소요산	동두천	보산	두천중	지행
2	소요산	0	2.5	inf	inf	inf
3	동두천	2.5	0	1.6	inf	inf
4	보산	inf	1.6	0	1.4	inf
5	동두천중앙	inf	inf	1.4	0	1
6	지행	inf	inf	inf	1	0

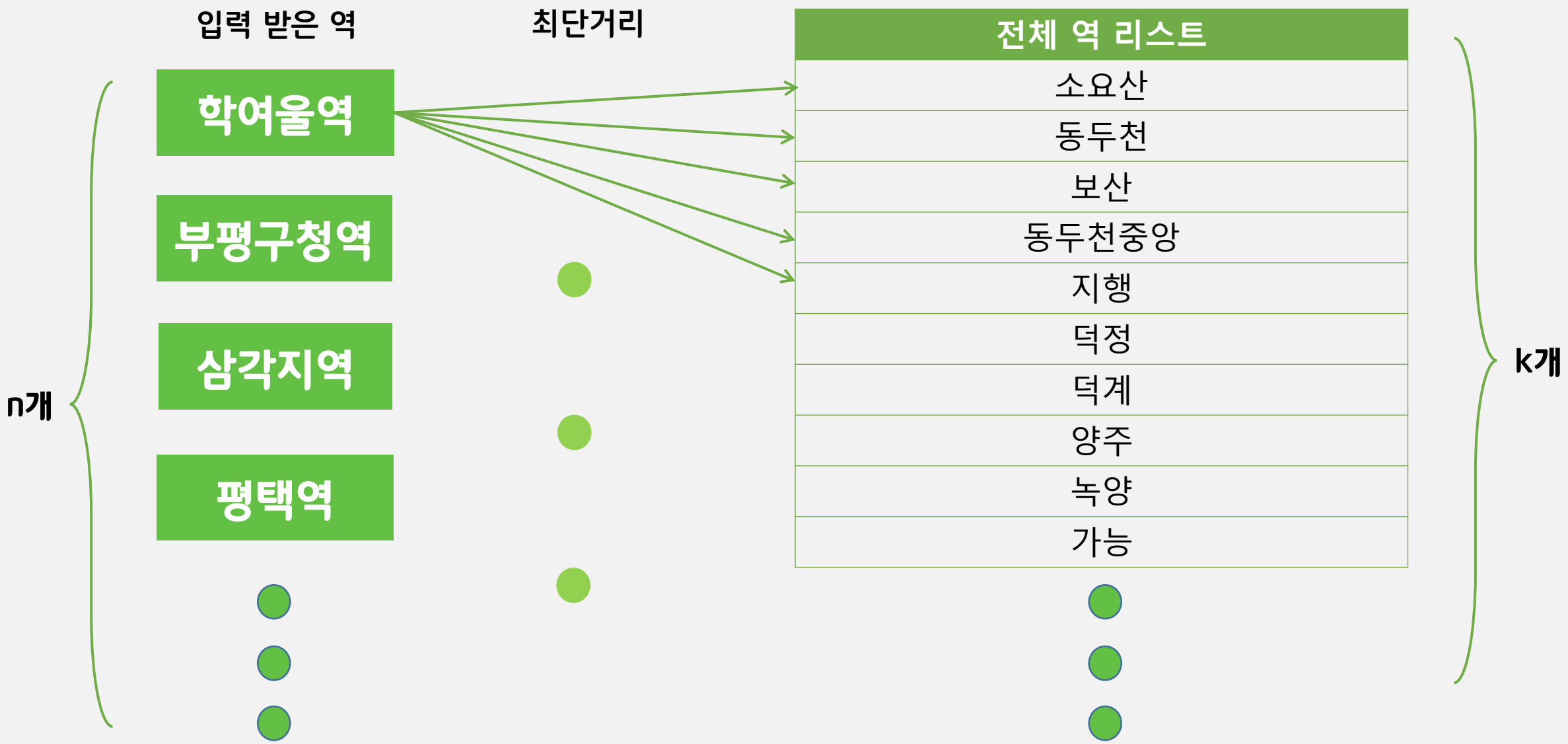
<이해를 돕기 위한 EXCEL 파일>

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

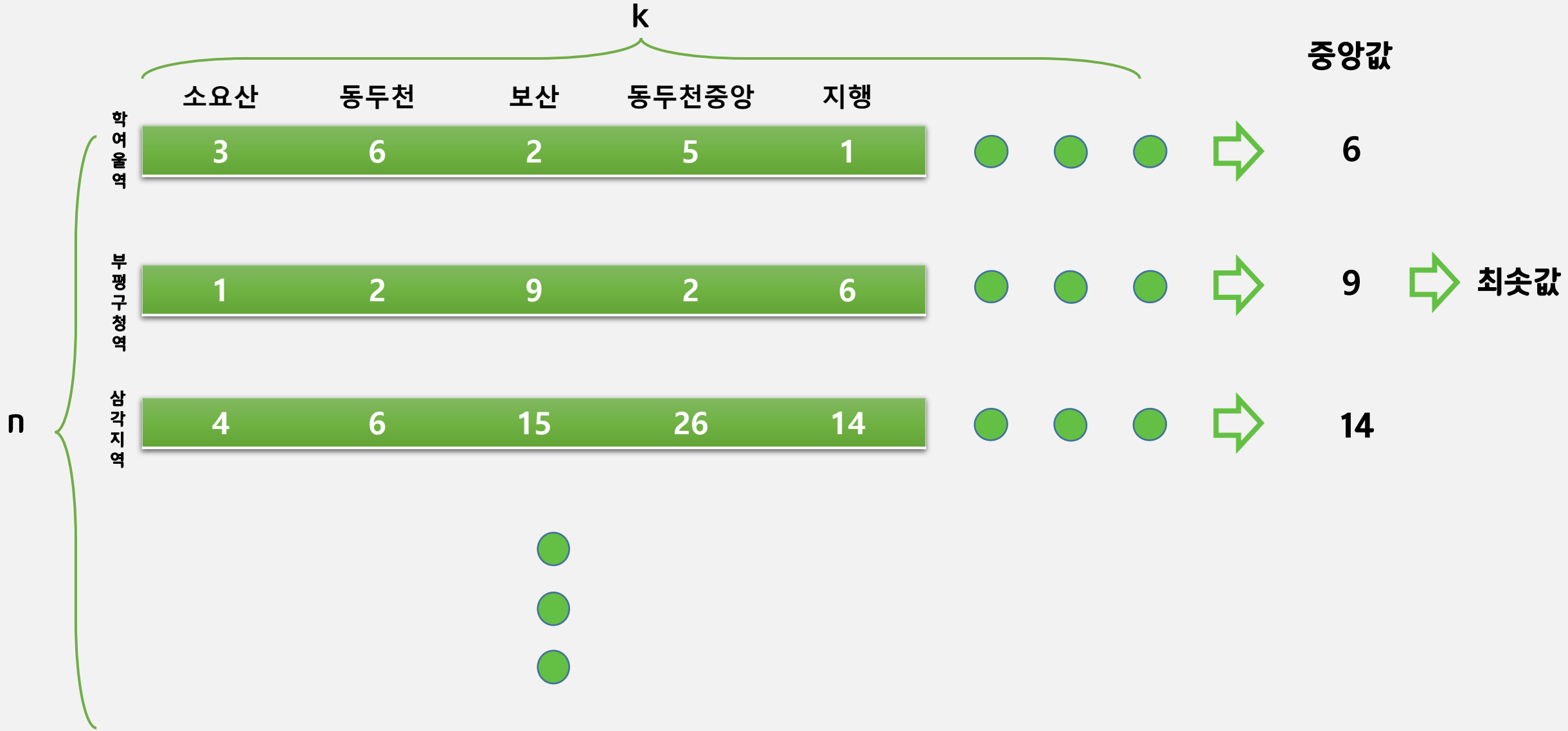
소요산	동두천	보산	동두천중앙		지행	덕정
강매	화전	수색	가좌	신촌(경의중앙)		홍대입구
전	완정	독정	검암	검바위	아시아드경기장	
0	2.5	inf	inf	inf	inf	inf
f	inf	inf	inf	inf	inf	inf
f	inf	inf	inf	inf	inf	inf
2.5	0	1.6	inf	inf	inf	inf
f	inf	inf	inf	inf	inf	inf

<역의 이름과 역 사이의 거리 데이터를 입력한 TXT 파일>

2 해결방안 - 1


입력 받은 역
최단거리
학여울역
부평구청역
삼각지역
평택역
n개
전체 역 리스트
소요산
동두천
보산
동두천중앙
지행
덕정
덕계
양주
녹양
가능
k개

2 해결방안 – 1


k
중앙값
소요산
동두천
보산
동두천중앙
지행
학여울역
3
6
2
5
1
6
부평구청역
1
2
9
2
6
9
최솟값
삼각지역
4
6
15
26
14
14
n

2 해결방안 – 1 <중앙값>

- 평균이 아닌 중앙값을 고른 이유

평균은 극단적인 값에 영향을 많이 받음

-> 극단적인 값에 영향을 덜 받는 중앙값 선택

- 알고리즘

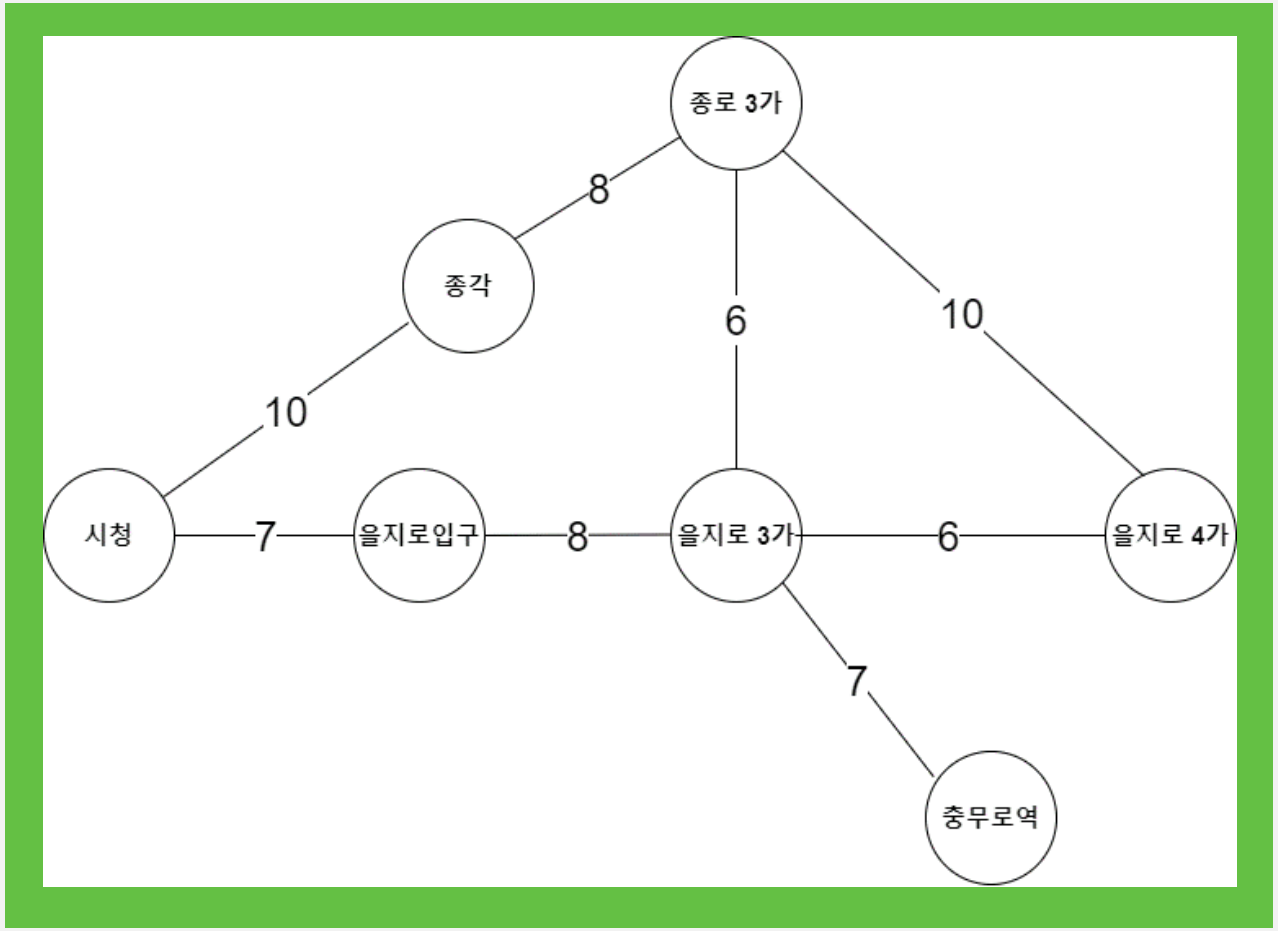
우선순위 큐를 이용

해당 값을 모두 push

If 홀수 -> pop: (n-1)/2 한 후 top

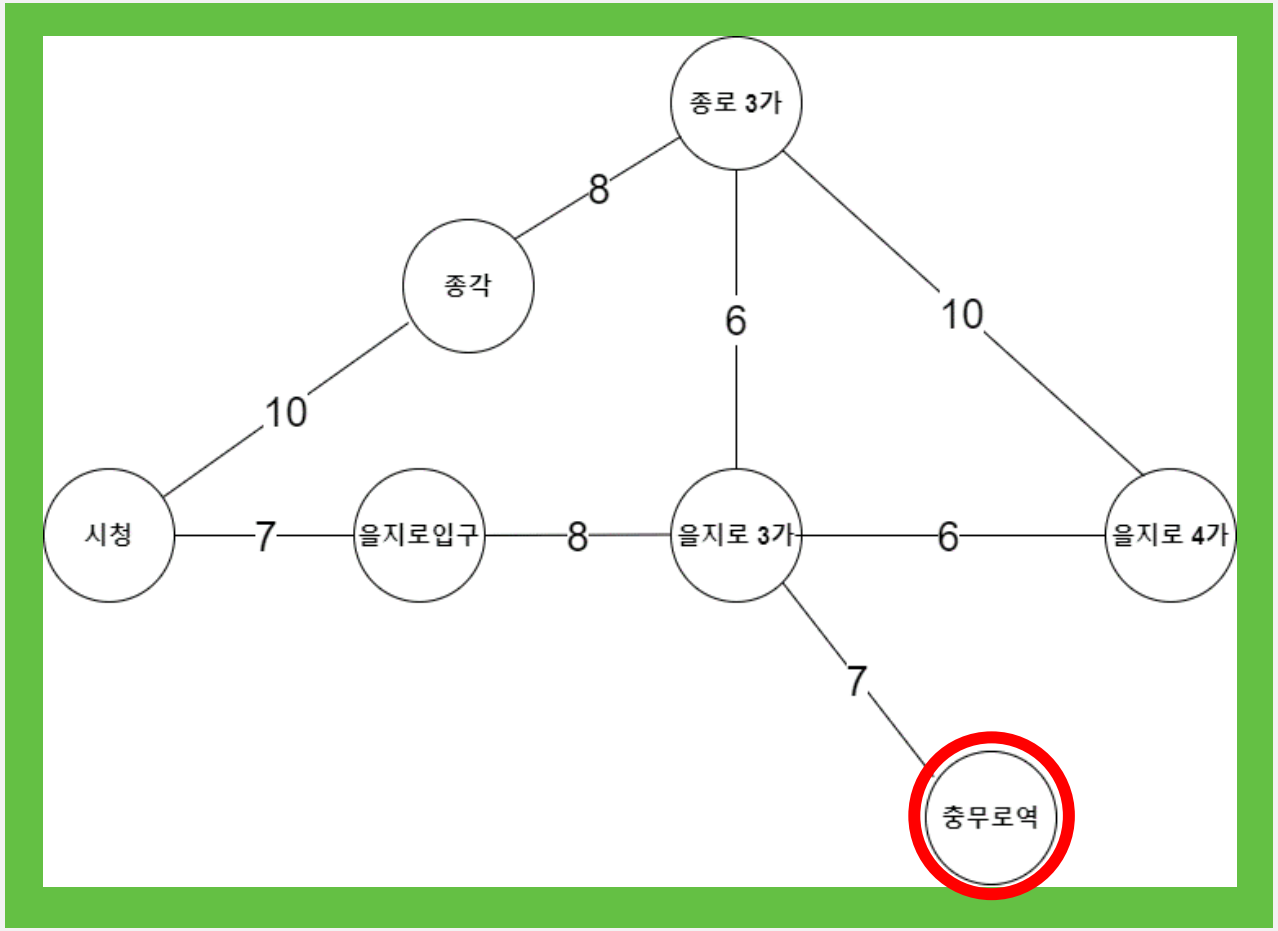
If 짝수 -> pop: n/2 -1 한 후 top에 있는 값 a+ 한 번 더 pop 하고 top에 있는 값 b, (a+b)/2

2 해결방안 – 1 <최단거리>



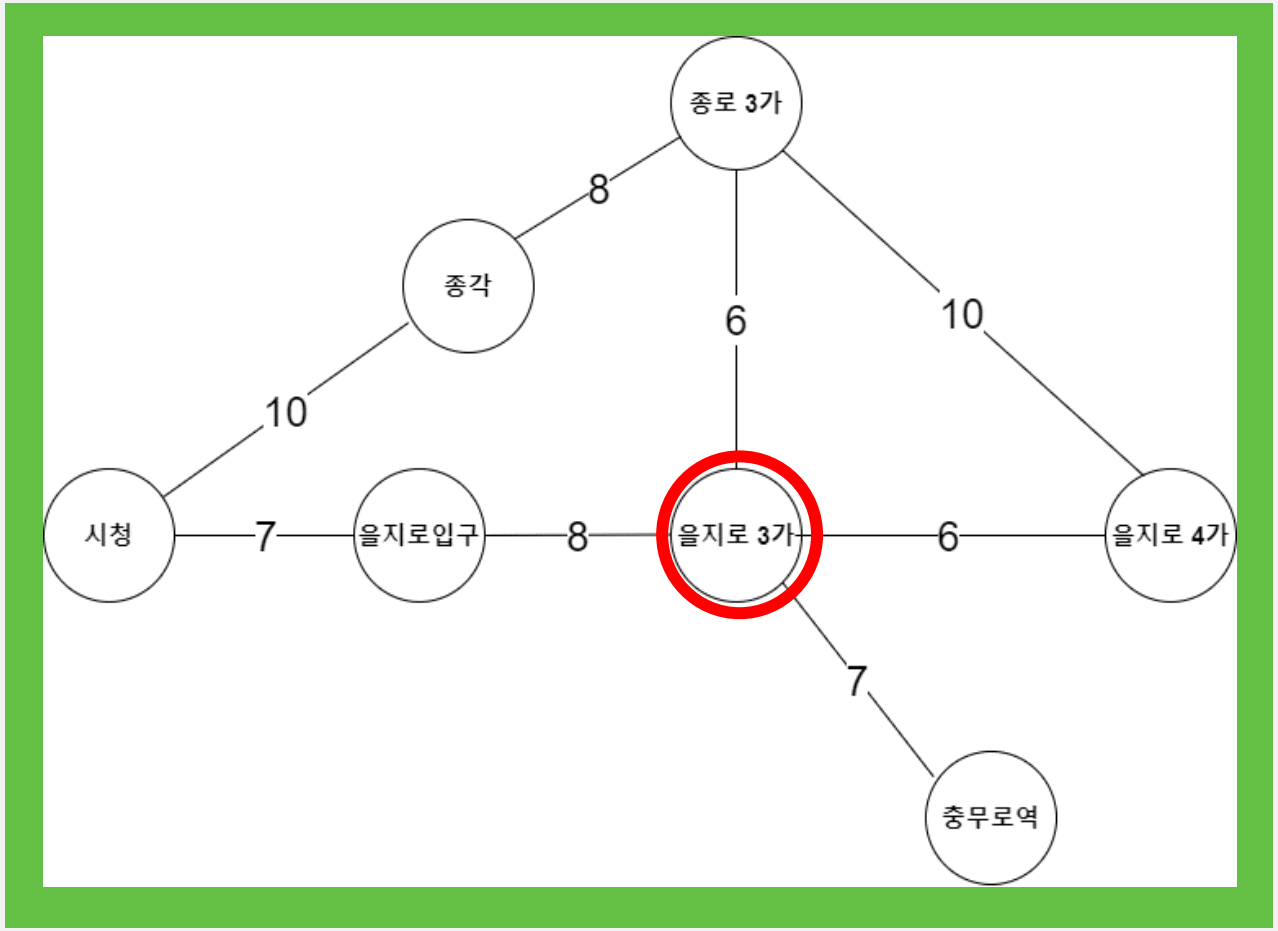
노드 이름	충무로	을지로3가	을지로입구	시청	종각	종로3가	을지로4가	우선순위 큐	POP
거리	0	INF	INF	INF	INF	INF	INF	(거리: -0, 노드: 충무로)	

2 해결방안 – 1 <최단거리>



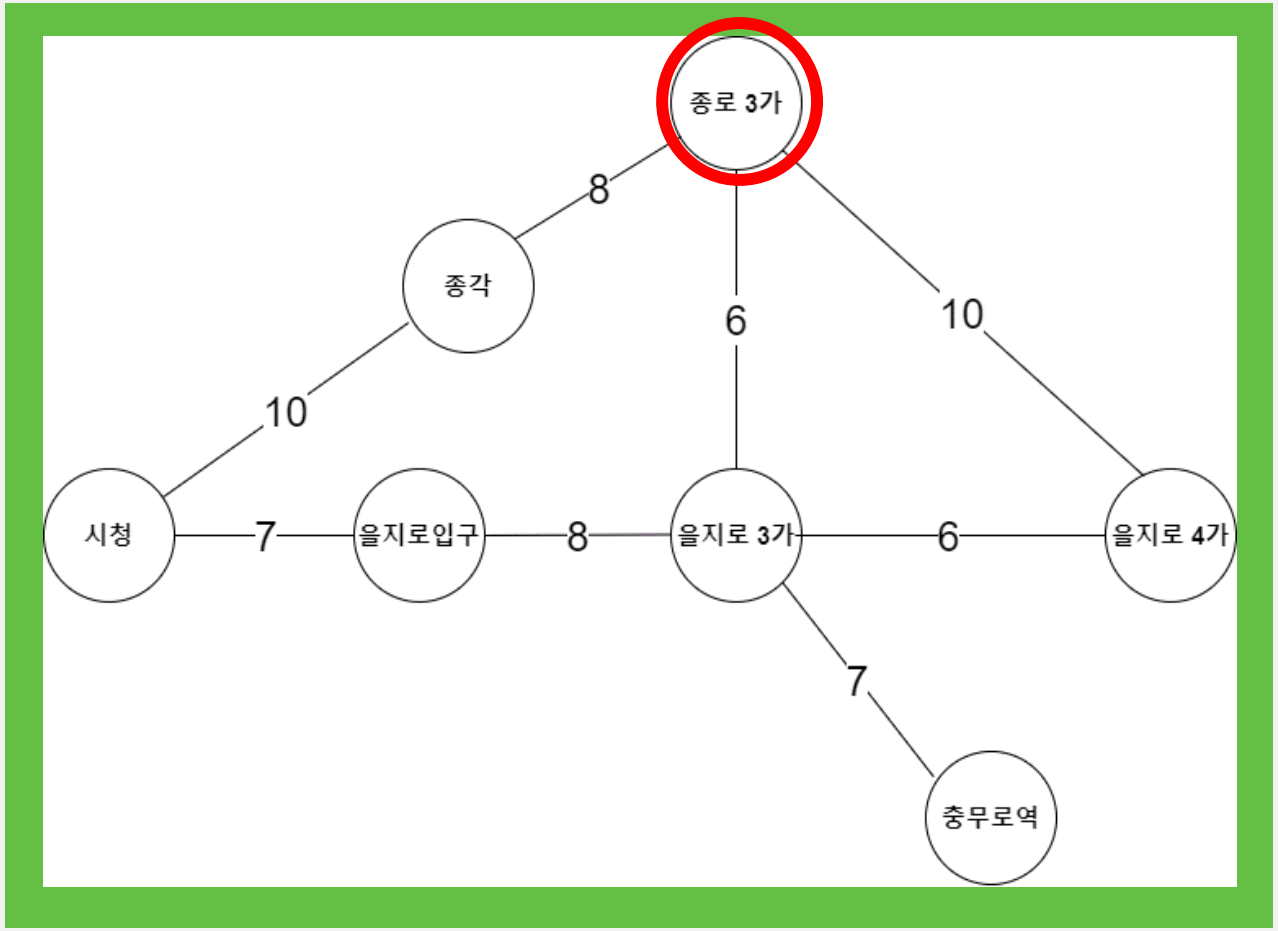
노드 이름	충무로	을지로3가	을지로입구	시청	종각	종로3가	을지로4가	우선순위 큐	POP
거리	0	7	INF	INF	INF	INF	INF	(거리: -7, 노드: 을지로3가)	

2 해결방안 – 1 <최단거리>



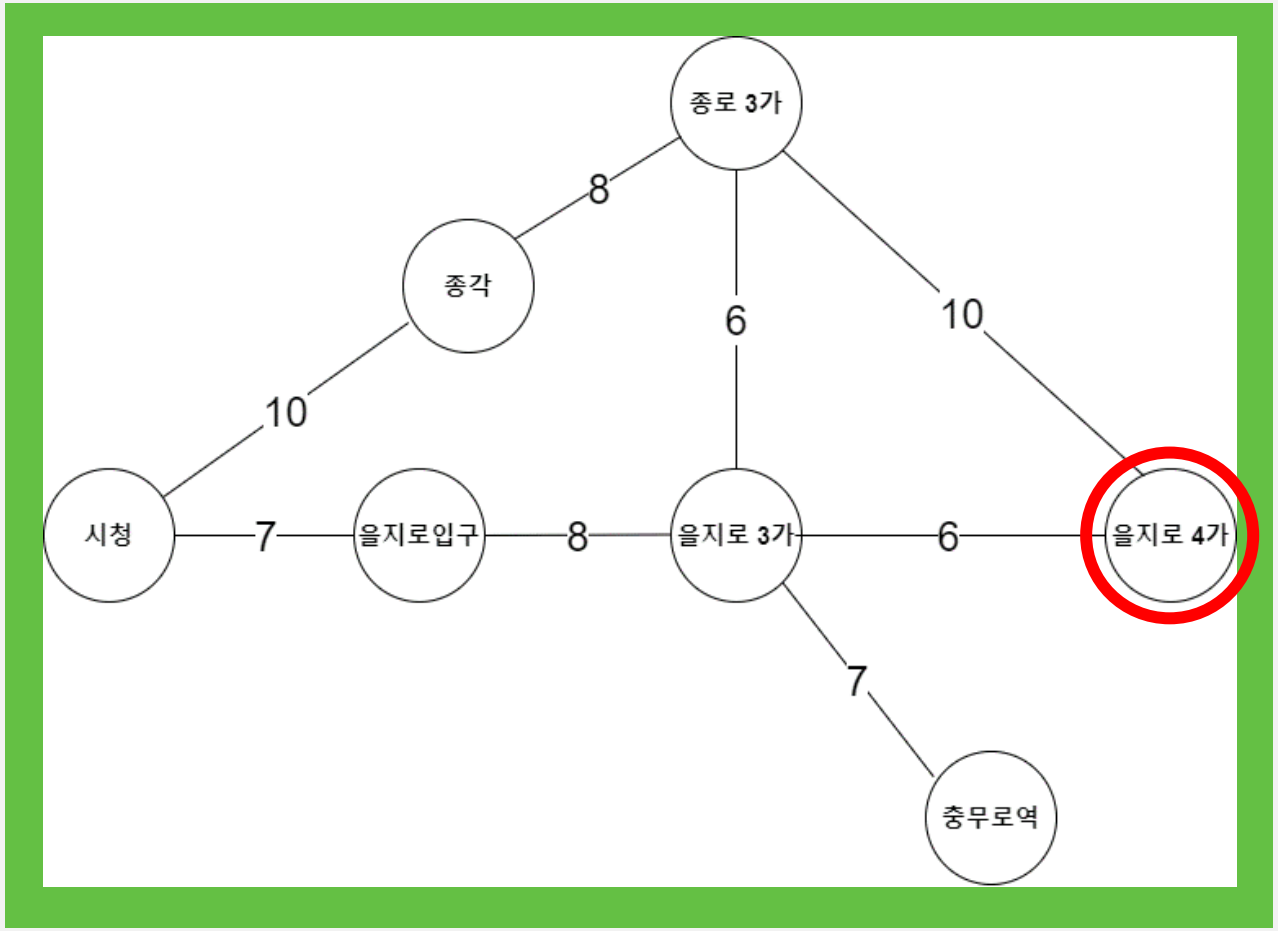
노드 이름	충무로	을지로3가	을지로입구	시청	종각	종로3가	을지로4가	우선순위 큐	POP
거리	0	7	15	INF	INF	13	13	(거리: -13, 노드: 종로3가)	(거리: -13, 을지로4가), (거리:: -15, 을지로입구)

2 해결방안 – 1 <최단거리>



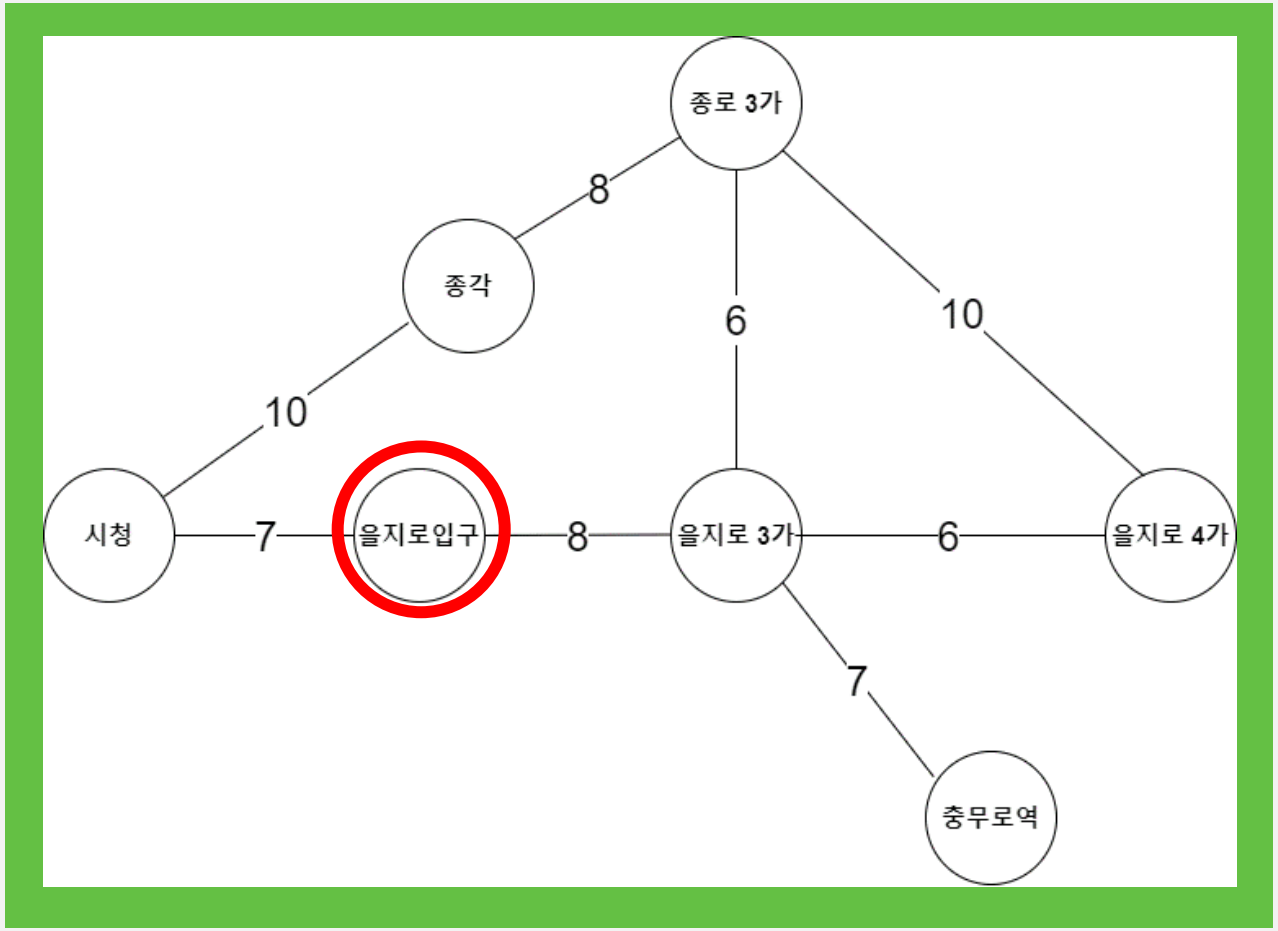
노드 이름	충무로	을지로3가	을지로입구	시청	종각	종로3가	을지로4가	우선순위 큐	POP
거리	0	7	15	INF	21	13	13	(거리: -13, 을지로4가), (거리:-15, 을지로입구) (거리:-21, 종각)	

2 해결방안 – 1 <최단거리>



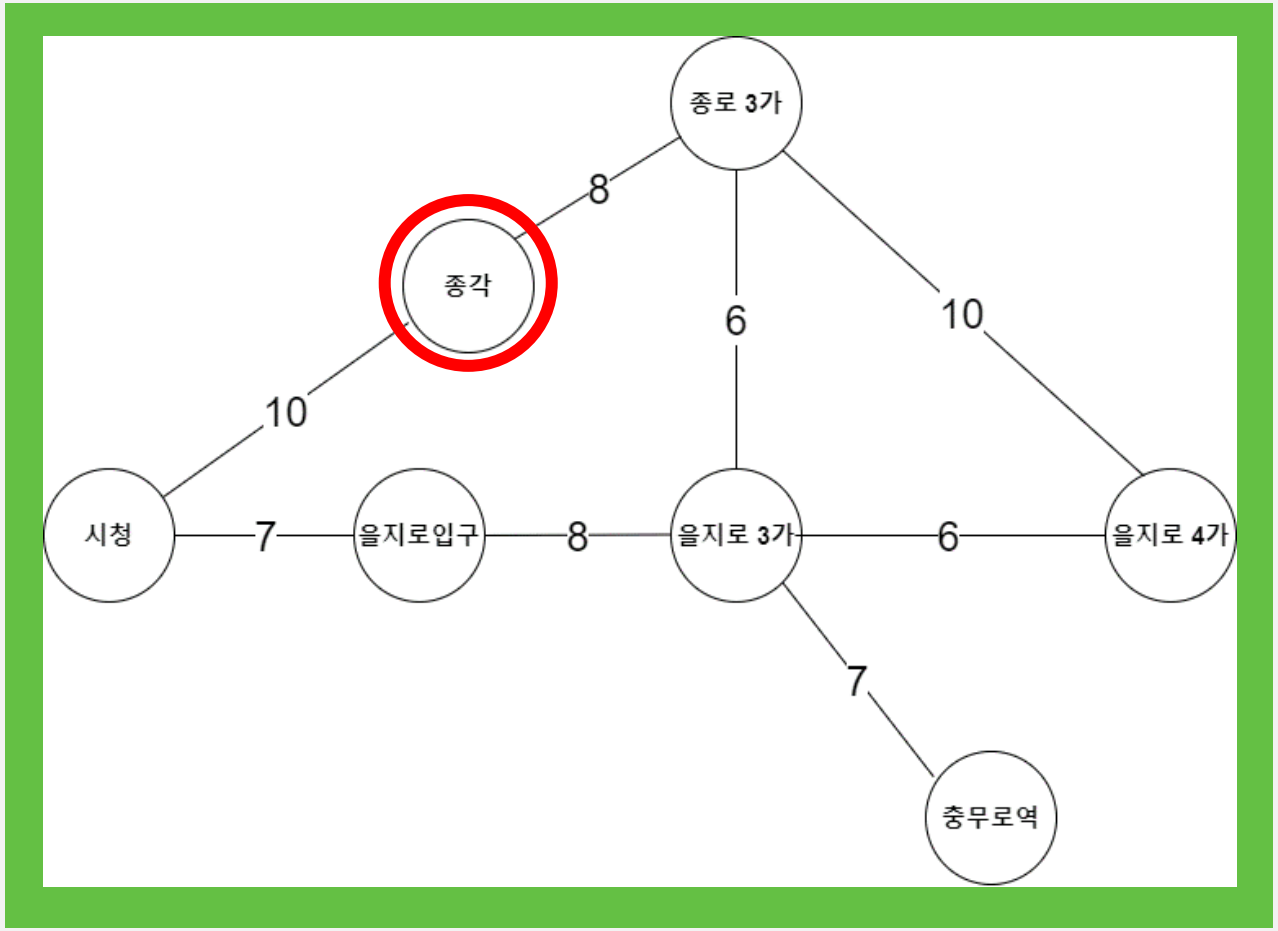
노드 이름	충무로	을지로3가	을지로입구	시청	종각	종로3가	을지로4가	우선순위 큐	POP
거리	0	7	15	INF	21	13	13	(거리: -15, 을지로입구)	(거리:-21, 종각)

2 해결방안 – 1 <최단거리>



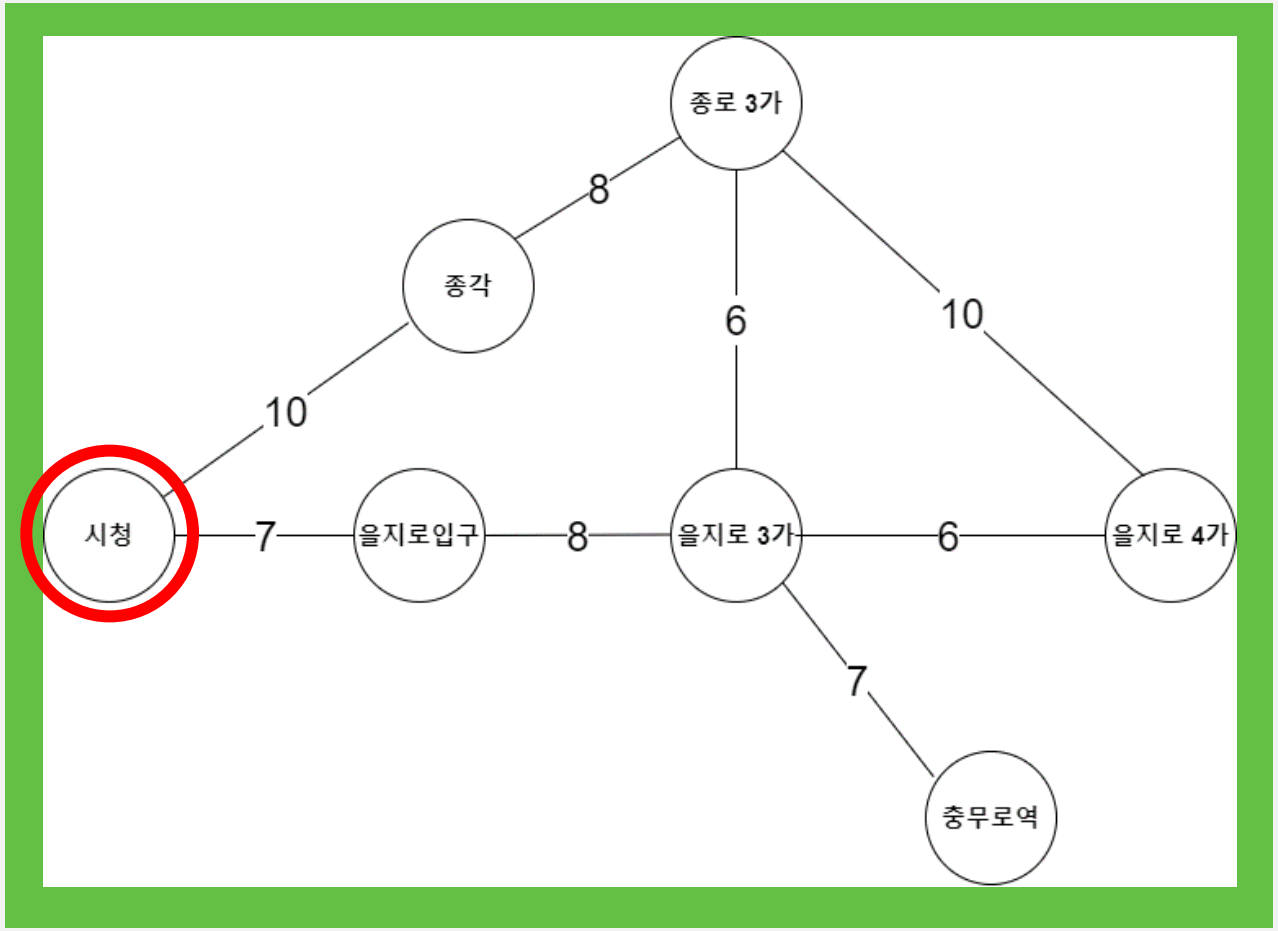
노드 이름	충무로	을지로3가	을지로입구	시청	종각	종로3가	을지로4가	우선순위 큐 POP
거리	0	7	15	22	21	13	13	(거리:-21, 종각) (거리: -22, 시청)

2 해결방안 – 1 <최단거리>



노드 이름	충무로	을지로3가	을지로입구	시청	종각	종로3가	을지로4가	우선순위 큐 POP
거리	0	7	15	22	21	13	13	(거리: -22, 시청)

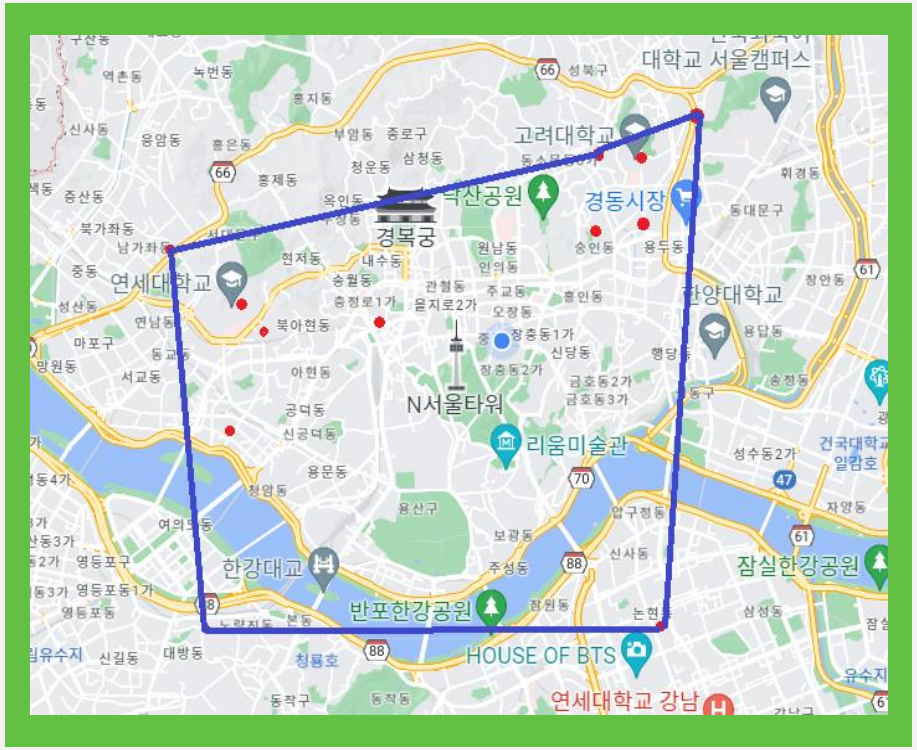
2 해결방안 – 1 <최단거리>



노드 이름	충무로	을지로3가	을지로입구	시청	종각	종로3가	을지로4가	우선순위 큐
거리	0	7	15	22	21	13	13	EMPTY

3 해결방안 – 2

Approach: 가장 자리 역만 이어 다각형을 만든다음, 그 다각형의 중심 좌표를 구하면 어떨까?



3 해결 방안 2-절차

기준점 찾기

탐색을 시작할 첫 기준 좌표를 정한다.

좌표 정렬

탐색을 위하여 반시계 방향으로 정렬한다.

탐색

스택을 이용하여, 볼록 껍질 좌표를 구한다.

결과 출력

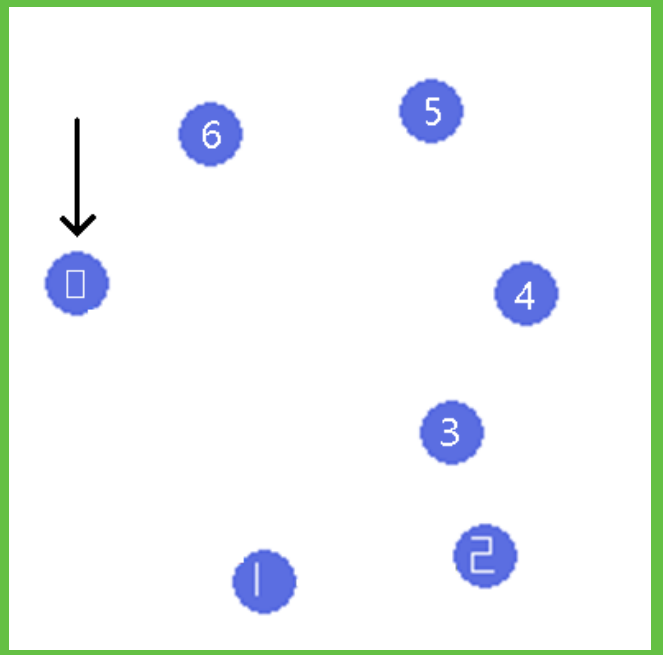
구한 볼록 껍질 좌표들의 평균 좌표를 구해, 가장 가까운 역을 출력한다.

3 해결 방안 – 2 : 첫 기준점 찾기

1. X 좌표가 가장 작은 역을 기준점으로 정렬

```
bool x_comp(Point A,Point B){
  if A.x==B.x
      order by y location ascending
  else
      order by x location ascending
}
```

After Sort: 0 6 1 5 3 2 4

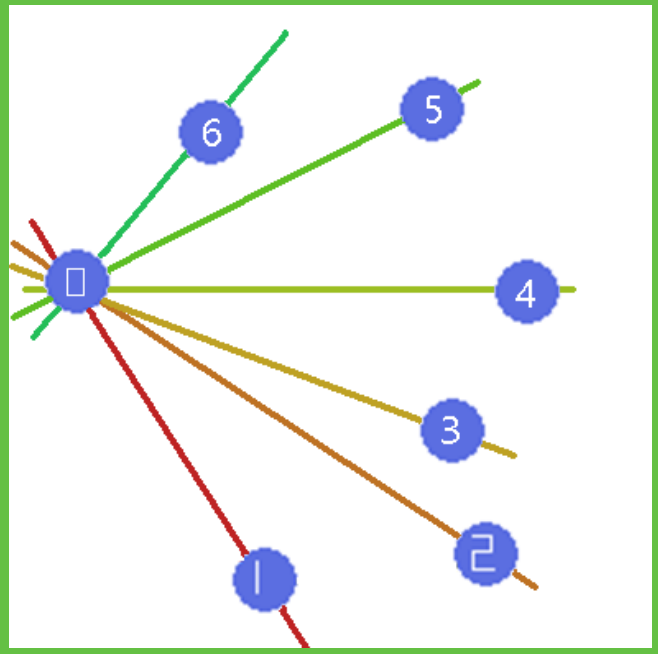


3 해결 방안 – 2 : 반시계 정렬

2. 기울기가 작은 순서대로 오름차순 정렬

```
bool gradient_comp(Point A,Point B){
  return get_gradient(first_location,A)<
  get_gradient(first_location,B);
}
```

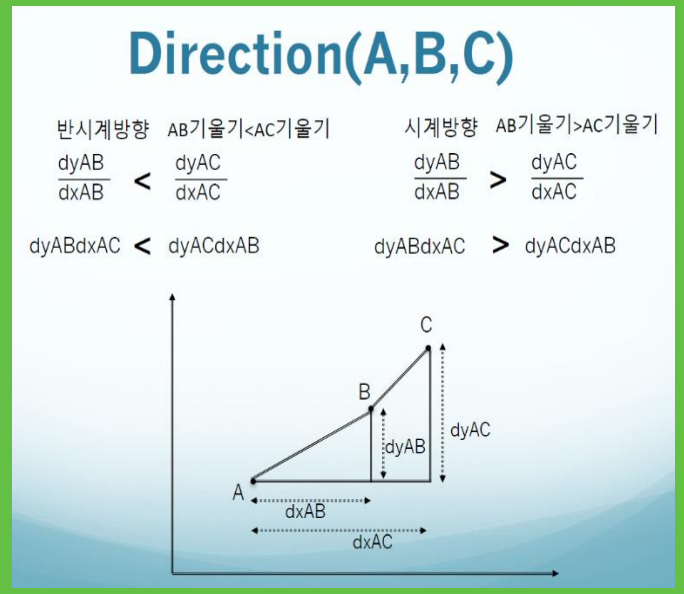
After Sort: 0 1 2 3 4 5 6



3 해결 방안 – 2 : 선분의 방향

3. 두 선분을 비교해, 반시계 방향인 경우만 push

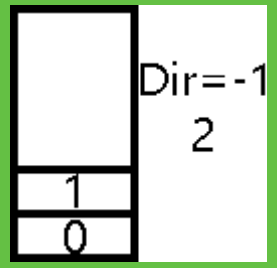
```
int Direction(Point A,Point B,Point C){
  if line AB,BC are clockwise:
    return 1
  else if line AB,BC are counterclockwise:
    return -1
  if line AB,BC's gradient are same:
    return 0
}
```

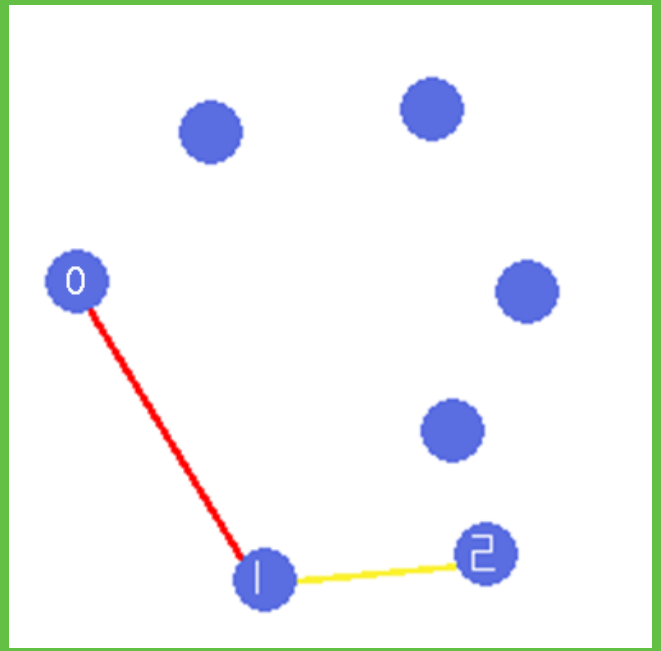


3 해결 방안 – 2 : 스택 탐색(1)

4. 초기 두 점을 스택에 push, Direction()을 이용해 선분의 방향 확인

```
//arr[stk[]]: 해당 index의 좌표
//시계 방향이거나 일직선시, 스택에서 제거
for 2 to location size
    while top>=2 and gradient(arr[stk[t-2]],arr[stk[t-1],arr[t]]>=0
        top--;
stk[top++]=t;
```

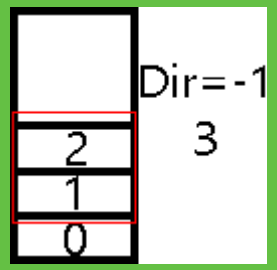


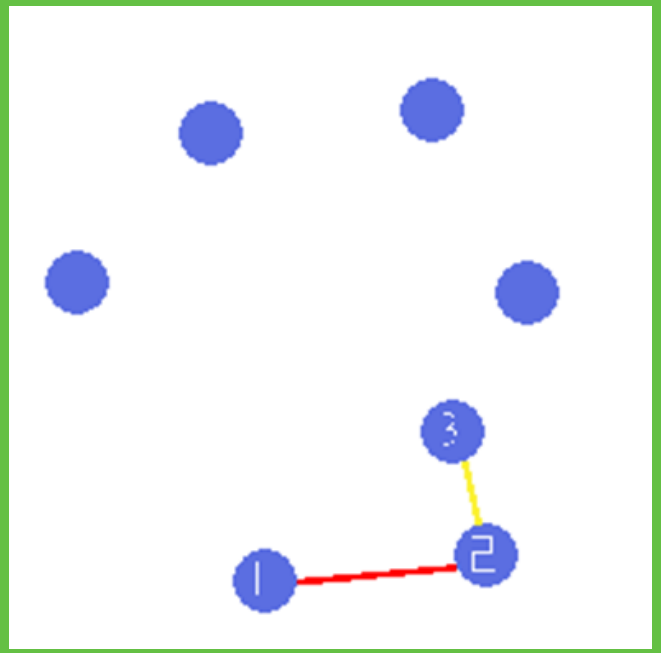


3 해결 방안 – 2 : 스택 탐색(2)

5. 스택 top의 두 점을 스택에 push, Direction()을 이용해
선분의 방향 확인

```
//arr[stk[]]: 해당 index의 좌표
//시계 방향이거나 일직선시, 스택에서 제거
for 2 to location size
    while top>=2 and gradient(arr[stk[t-2]],arr[stk[t-1],arr[t]]>=0
        top--;
stk[top++]=t;
```

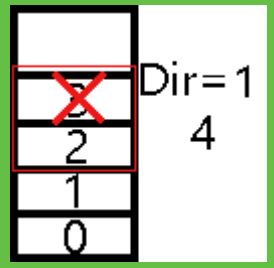


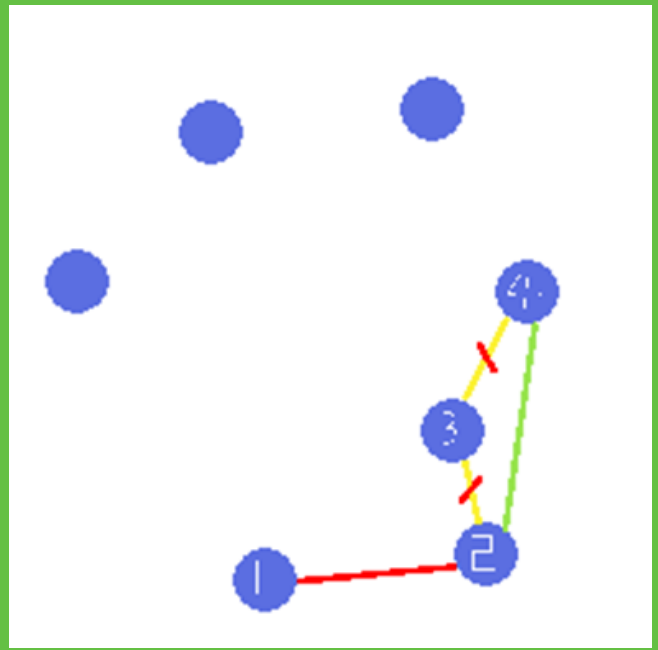


3 해결 방안 – 2 : 스택 탐색(3)

6. 두 선분이 시계 방향이 된 경우, 스택 Top 삭제

```
//arr[stk[]]: 해당 index의 좌표
//시계 방향이거나 일직선시, 스택에서 제거
for 2 to location size
    while top>=2 and gradient(arr[stk[t-2]],arr[stk[t-1],arr[t]]>=0
        top--;
stk[top++]=t;
```

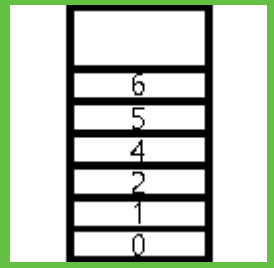


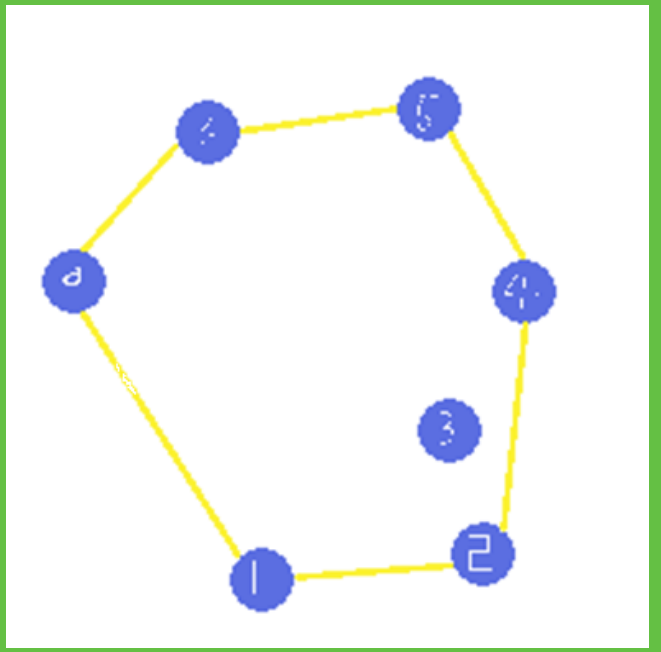


3 해결 방안 – 2 : 최종 좌표

7. 모든 좌표를 돌면, 스택에 있는 좌표가 볼록 껍질의 좌표에 해당됨.

```
//arr[stk[]]: 해당 index의 좌표
//시계 방향이거나 일직선시, 스택에서 제거
for 2 to location size
    while top>=2 and gradient(arr[stk[t-2]],arr[stk[t-1],arr[t]]>=0
        top--;
stk[top++]=t;
```



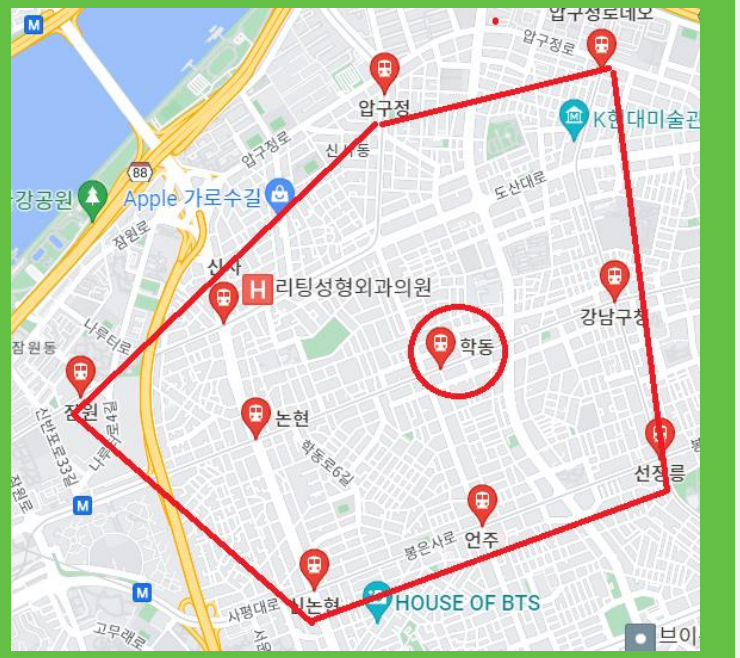


3 해결 방안 – 2 : 가장 가까운 역

유클리드 거리를 구해, 가장 가까운 지하철 역 반환

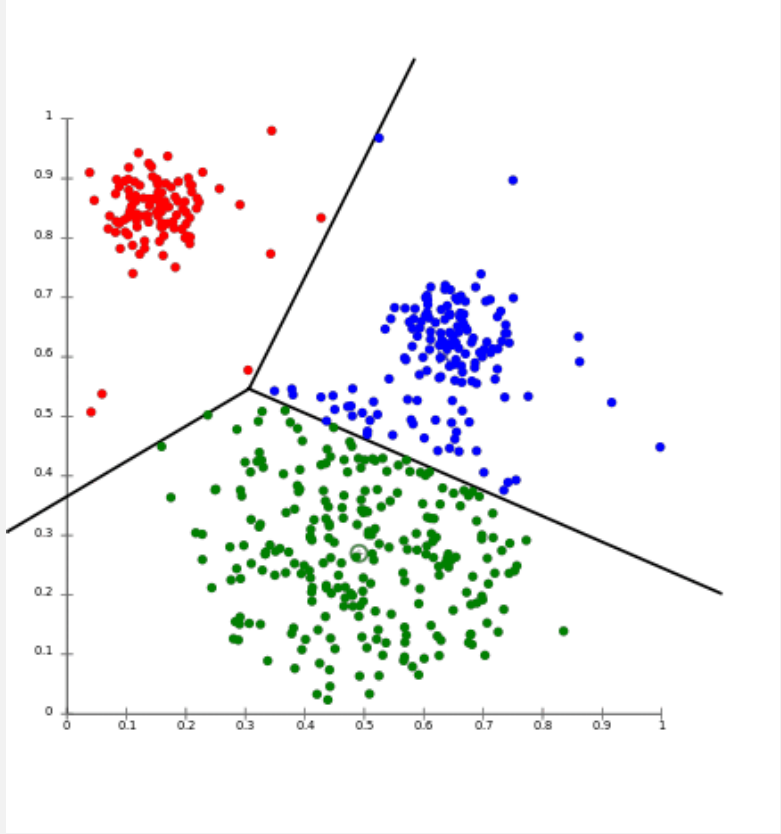
```
중간 좌표 결과: 37.5159 127.027
가장 가까운 역과 좌표: 학동역
37.5144 127.032
```

```
10
압구정역
압구정로데오역
신사역
잠원역
강남구청역
선정릉역
학동역
논현역
언주역
신논현역
```



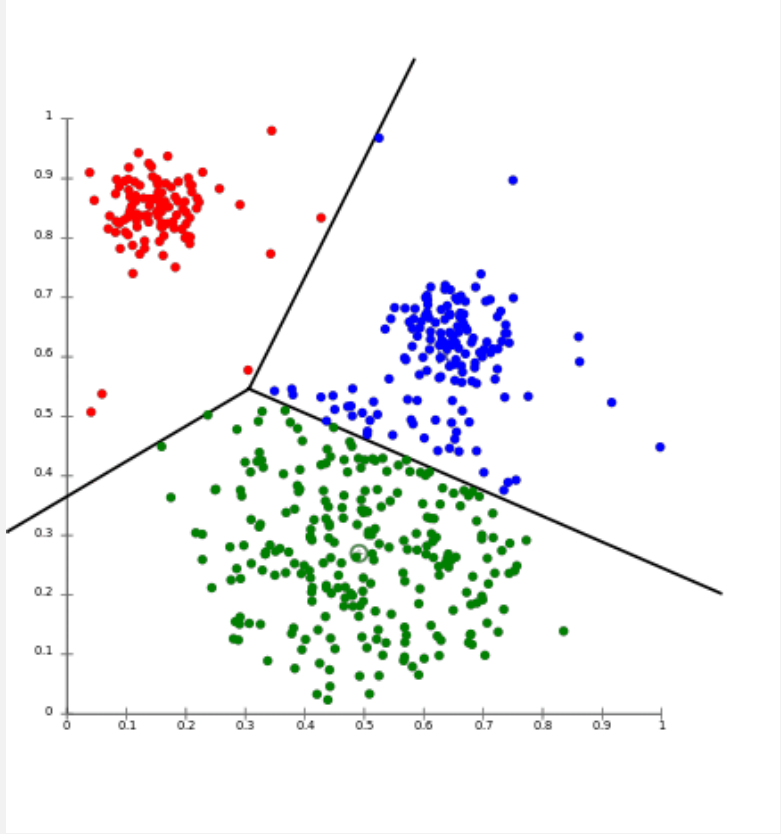
2 해결방안 – 3 <분포의 평균>

Idea : N명의 사람들이 각자의
역에서 출발하니까,
분포를 나누고, 분포의 중심들을 연결해서
평균의 위치를 구하면 어떨까?

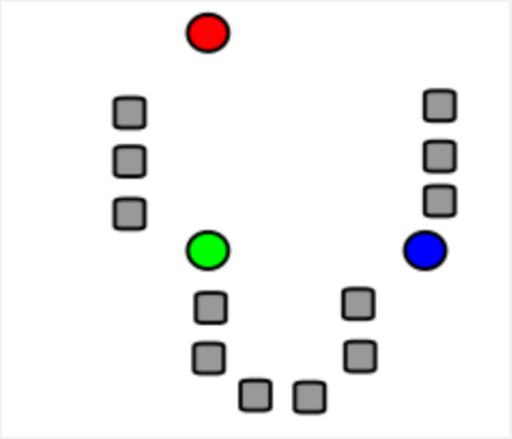


2 해결방안 – 3 <분포의 평균>

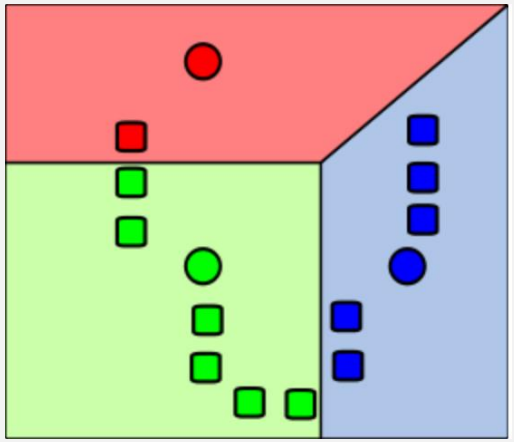
분포를 나누지 말고,
모든 분포의 평균값을 구해
주변의 역을 출력하자!



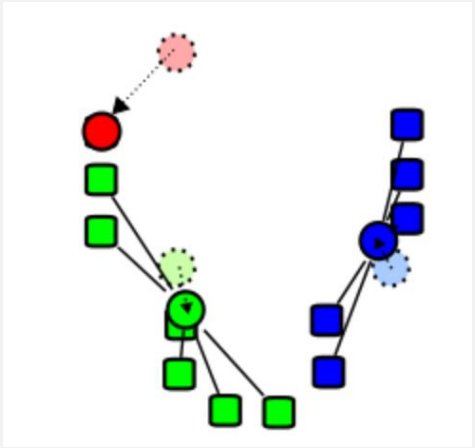
2 해결방안 – 3 <분포의 평균>



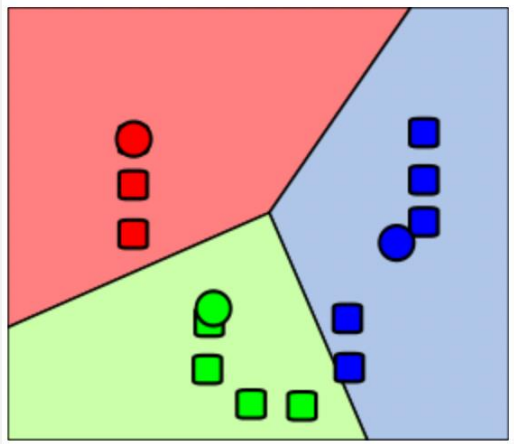
(1)
초기값
(클러스터의 중심) 설정



(2)
설정한 평균값을 기준으로
groupping 해준다.

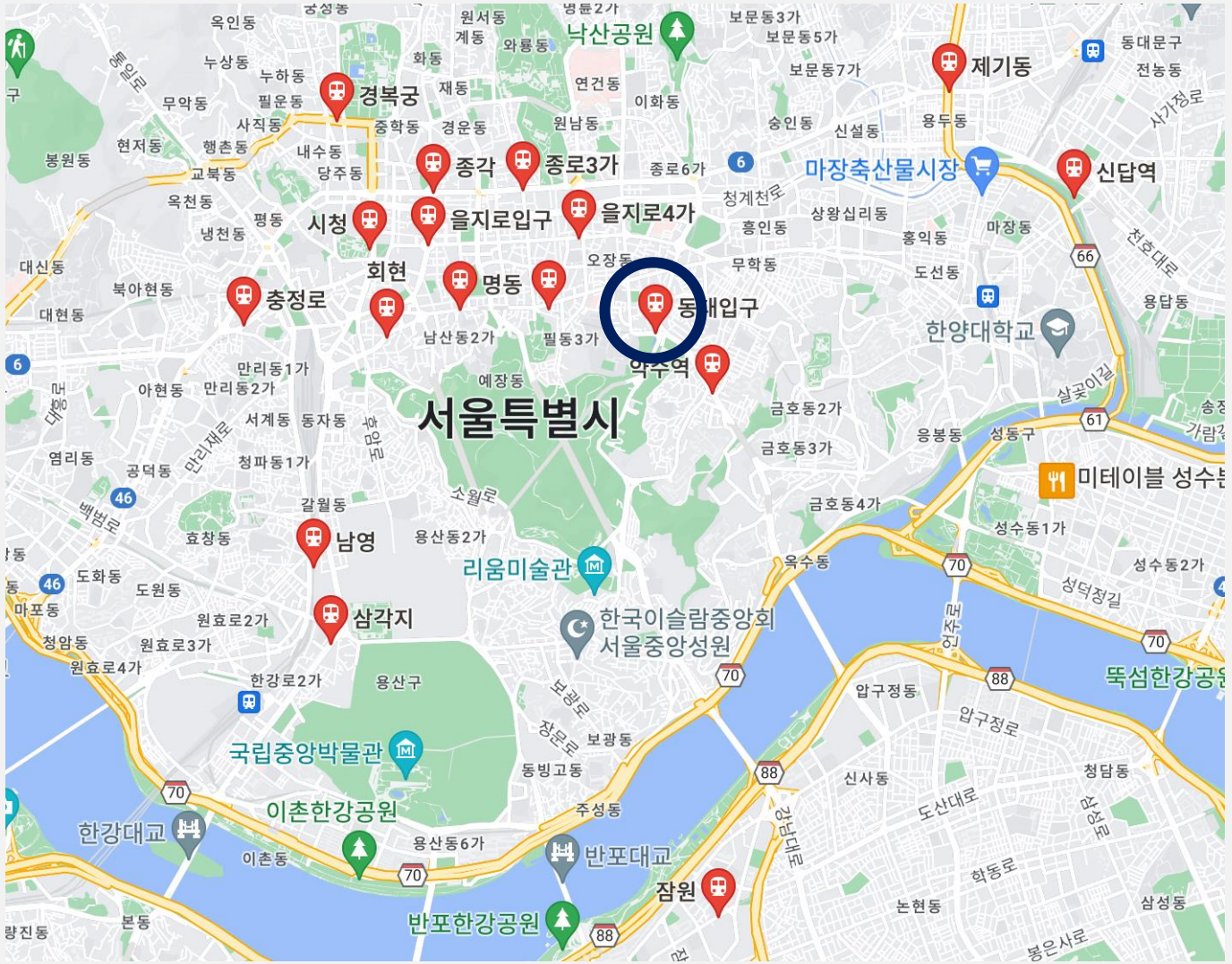


(3)
그룹에 있는 오브젝트들의
평균을 구해서
평균값을 재설정한다.

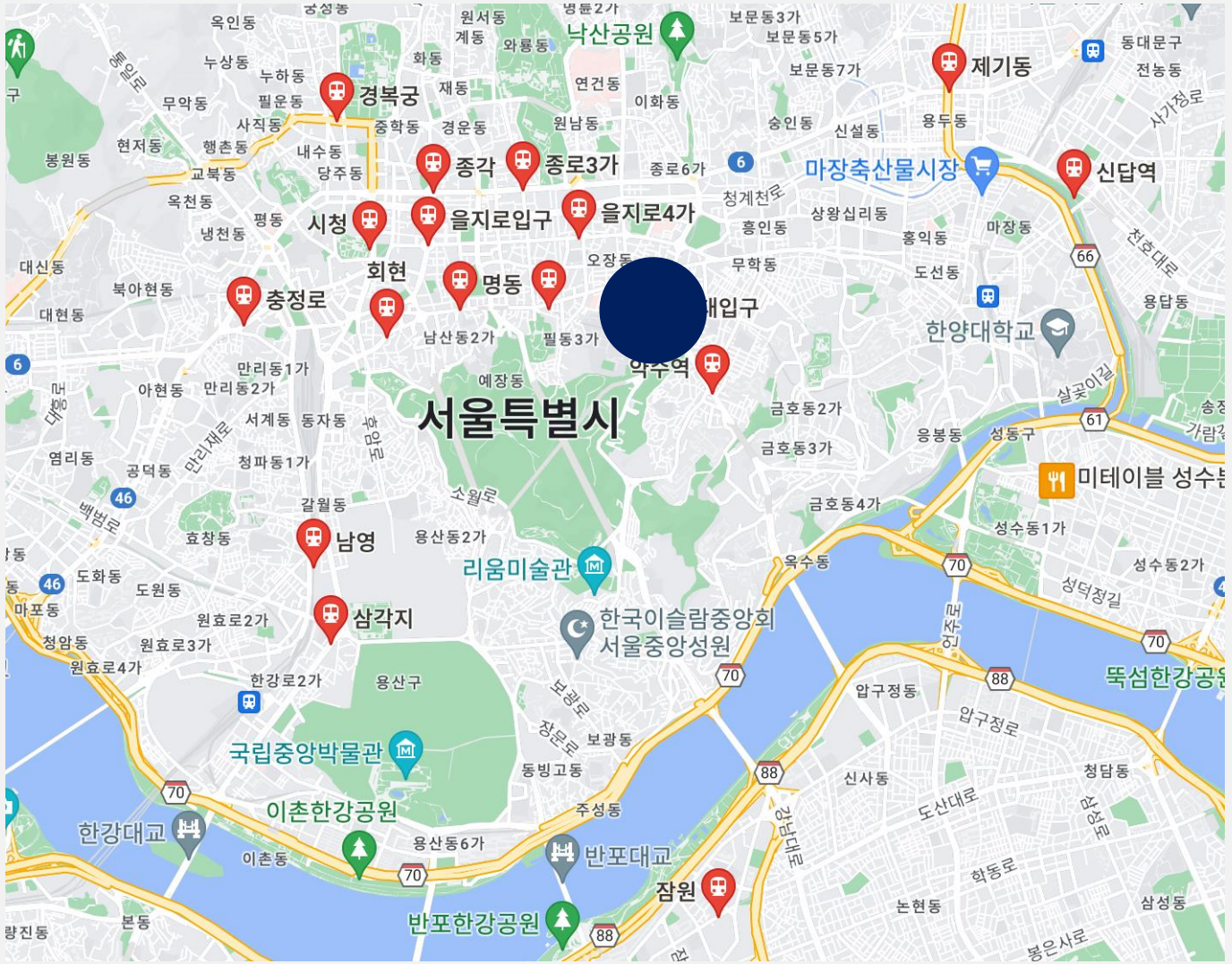


(4)
이과정을 계속 반복해서,
그룹의 평균에 수렴하게
만든다

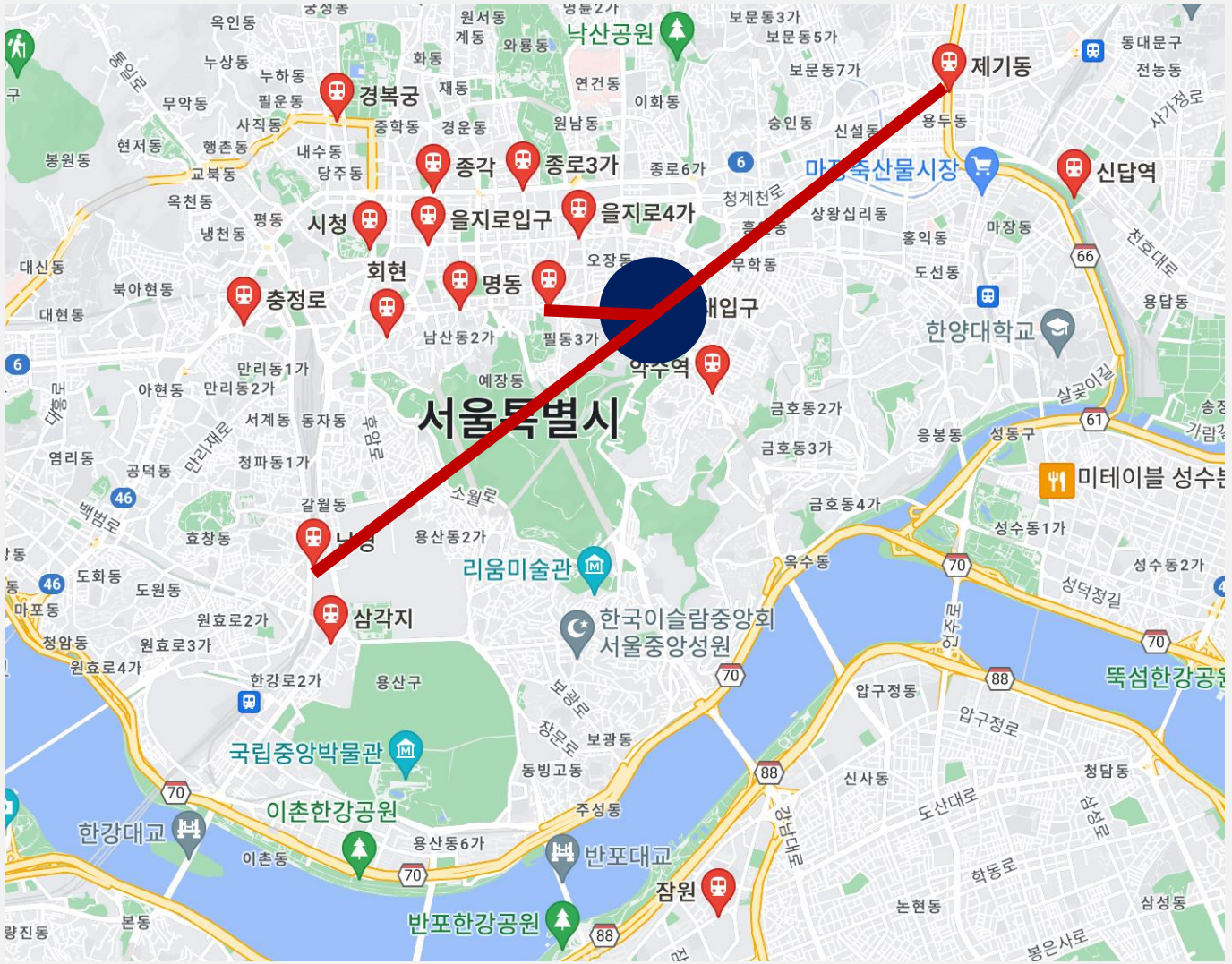
2 해결방안 – 3 <분포의 평균>



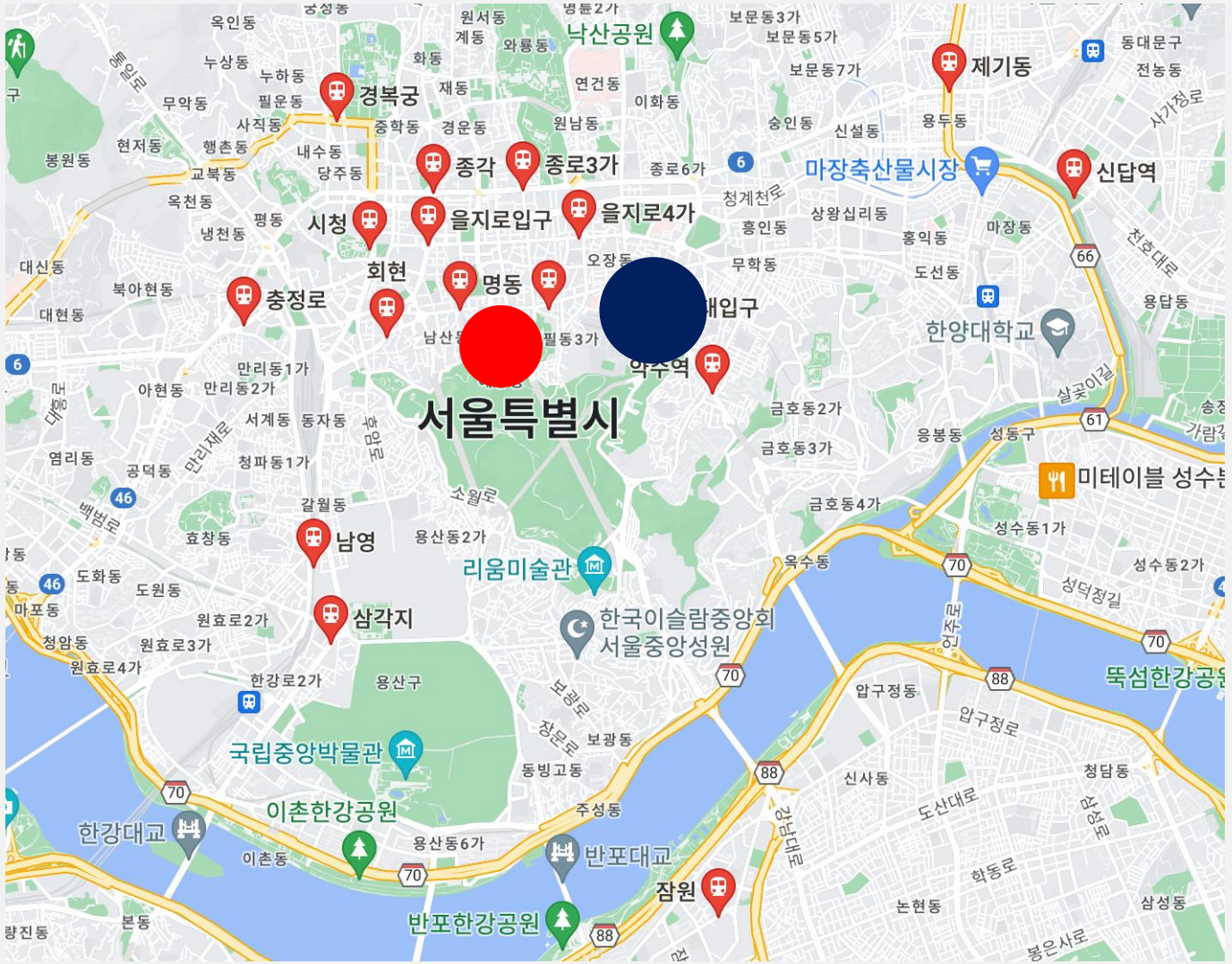
2 해결방안 – 3 <분포의 평균>



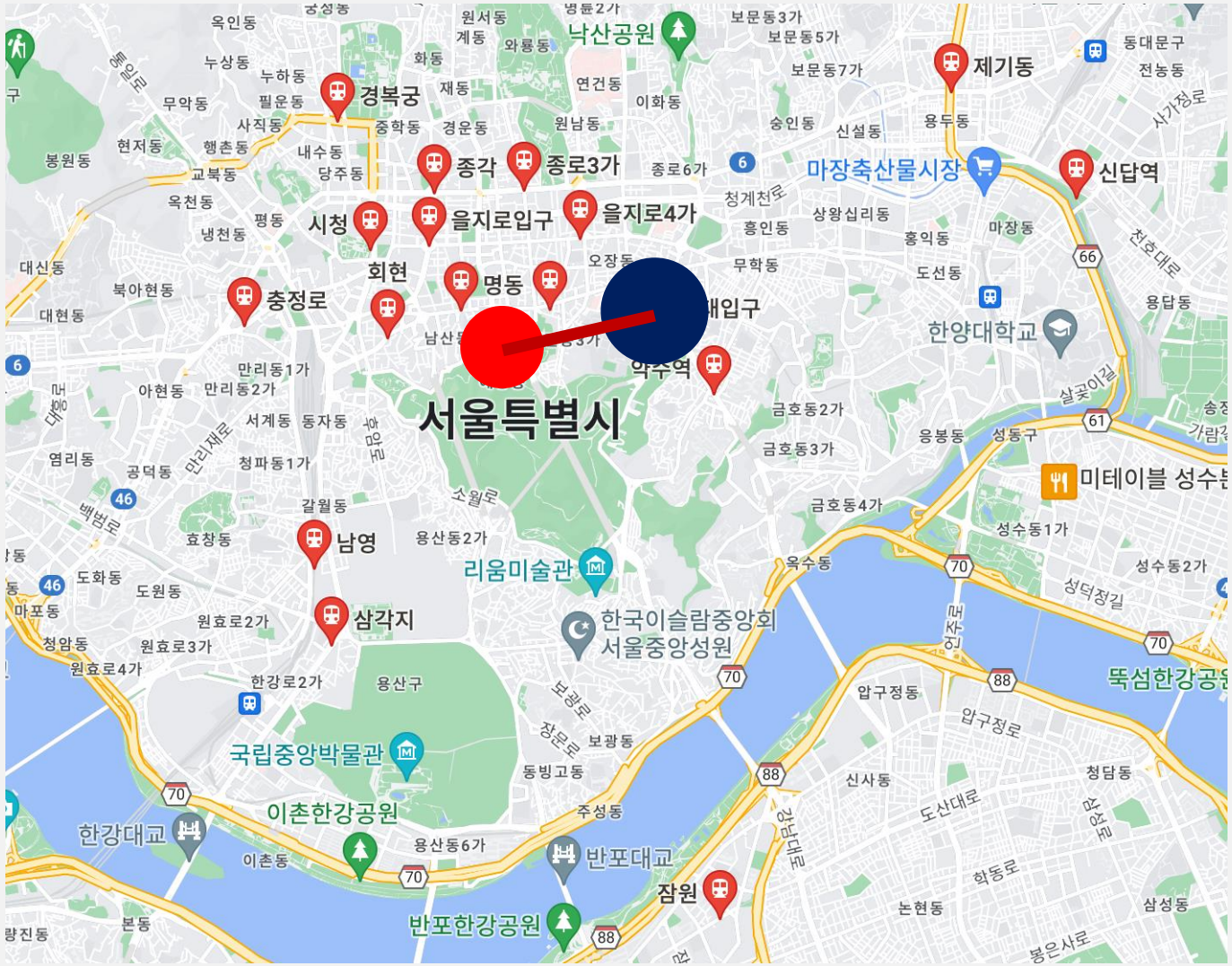
2 해결방안 – 3 <분포의 평균>



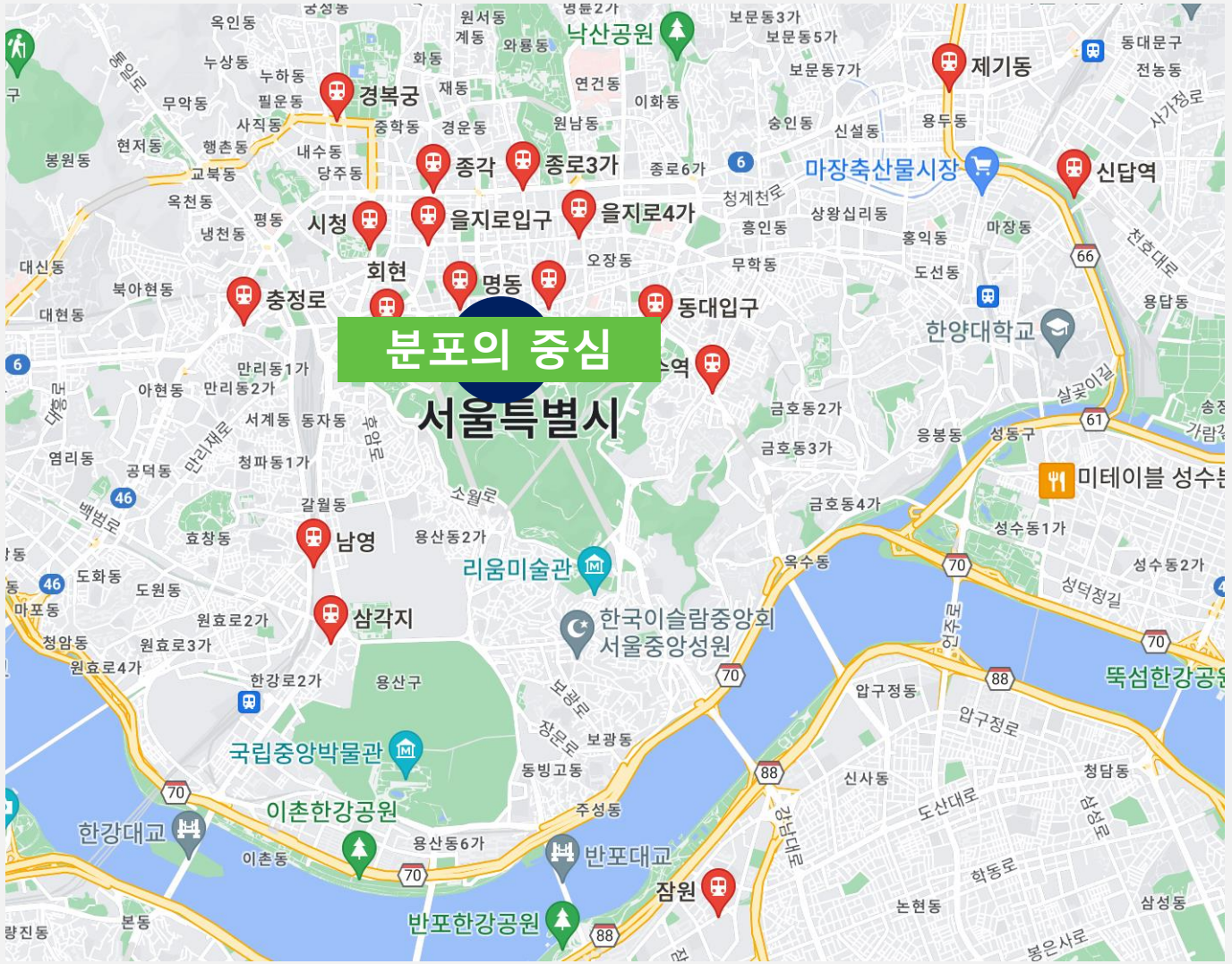
2 해결방안 – 3 <분포의 평균>



2 해결방안 – 3 <분포의 평균>



2 해결방안 – 3 <분포의 평균>



3 결과 및 성능 비교

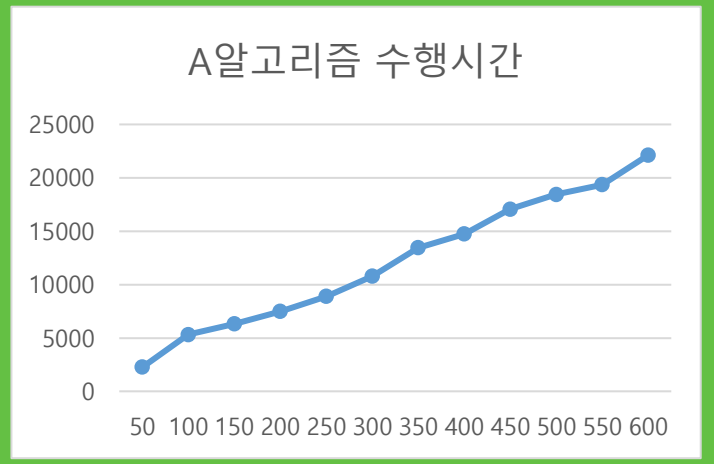
A: 최단경로 찾기와 중앙값을 이용하여 중간지점 찾기

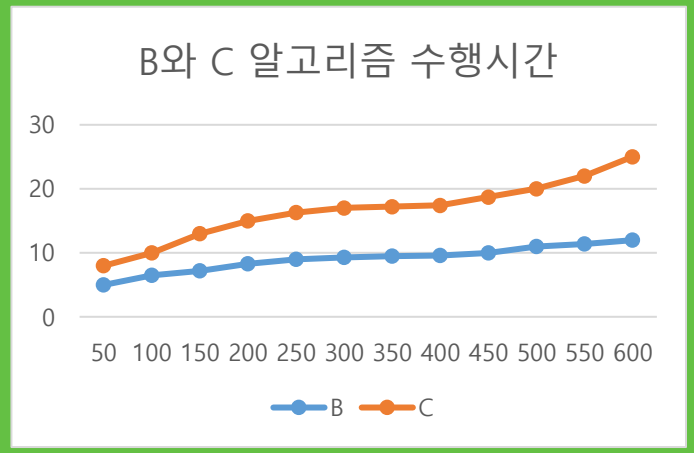
B: 볼록 껍질의 중심을 이용하여 중간지점 찾기

C: 분포의 평균을 이용하여 중간지점 찾기

단위:N명/ms

	50	100	150	200	250	300	350	400	450	500	550	600
A	2280	5324	6334	7503	8903	10800	13462	14743	17068	18453	19363	22143
B	5	6.5	7.2	8.3	9	9.3	9.5	9.6	10	11	11.4	12
C	8	10	13	15	16.3	17	17.2	17.4	18.7	20	22	25





3 결과 및 성능 분석

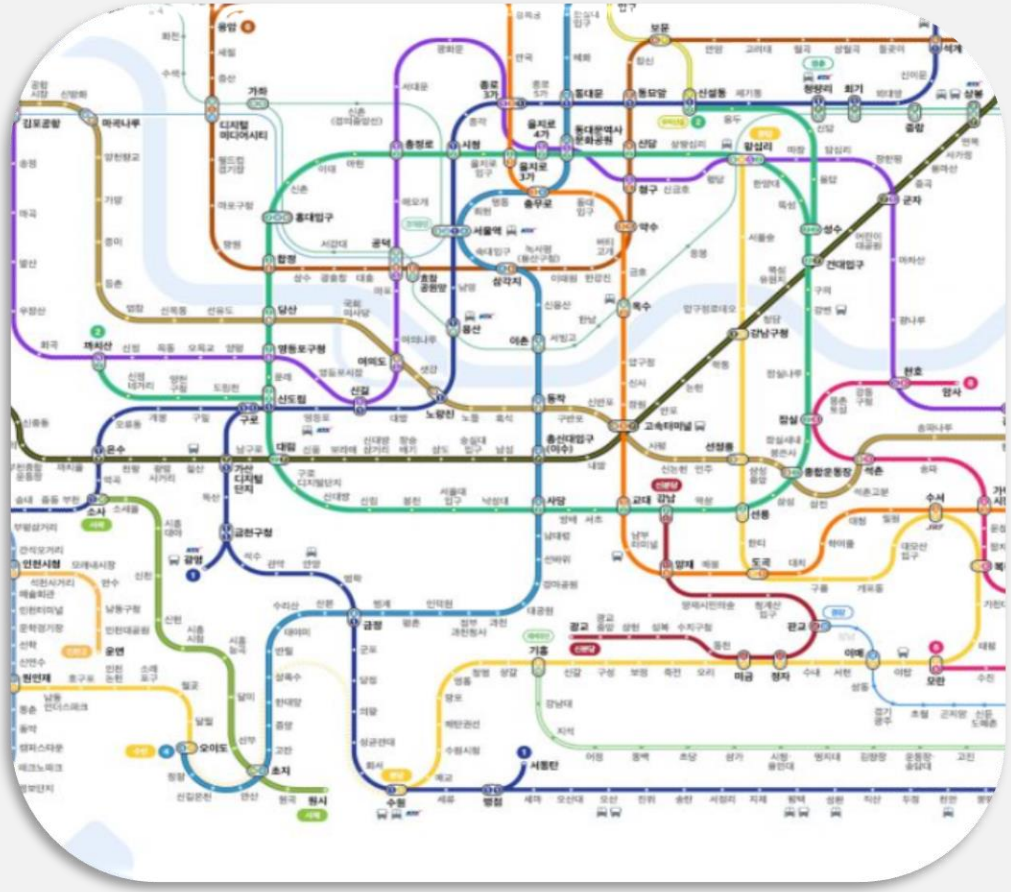
A: 평균이 아닌 중앙값 이용, 실제 거리 고려-> 오차 ↓

B,C: 이상 값도 평균 포함-> 오차 ↑ but, A 대비 속도 ↑

	A	B	C
장점	실제 지하철 간 거리를 탐색해 중앙역을 계산->가장 현실적이고 정확한 결과	- 가장 빠름 - 입력의 크기가 증가해도 경과 시간이 큰 폭으로 증가하지 않음.	-분포의 평균값을 구하는 방식 -> 안정적인 결과 -한 역이 아닌 반경 1km 내의 주변 역을 반환 ->정확도 높아짐
단점	한 개의 입력마다 모든 지하철 역의 최단거리를 구해야 함 -> 입력의 수가 증가함에 따라 소요 시간 역시 크게 증가, 세 개의 알고리즘 중에서 가장 오래 걸림	- 입력 받은 좌표가 한 분포로 극단적으로 치중된 경우 다수의 사람들에게 불리한 결과 - 반환한 결과 값이 환승을 고려하지 않아 정확성이 다소 떨어질 수 있음	- input의 수가 늘어 날 수록 시간 복잡도 기하급수적 증가 - 비정상적인 값이 들어있을 경우 최적의 위치가 아닐 수 도 있음

4 기대효과

- 직접 거리를 비교해 보며 중간 장소를 찾을 필요 없음
- 약속 장소를 정하는데 걸리는 시간 단축
- 쉽게 약속 장소를 정할 수 있도록 도움
- 소수의 인원이 아닌 다수의 인원도 측정 가능해 더욱 효과적



004 감 사 합 니 다