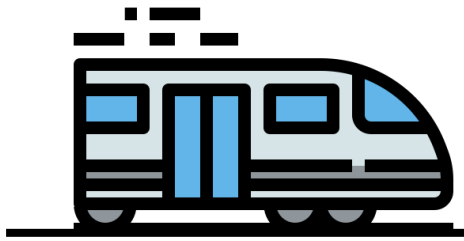
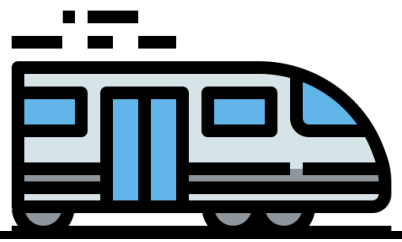
컴퓨터공학과 2020112073 김예지, 2020112046 이재원, 2019111994 이주윤, 2020111655 심여은

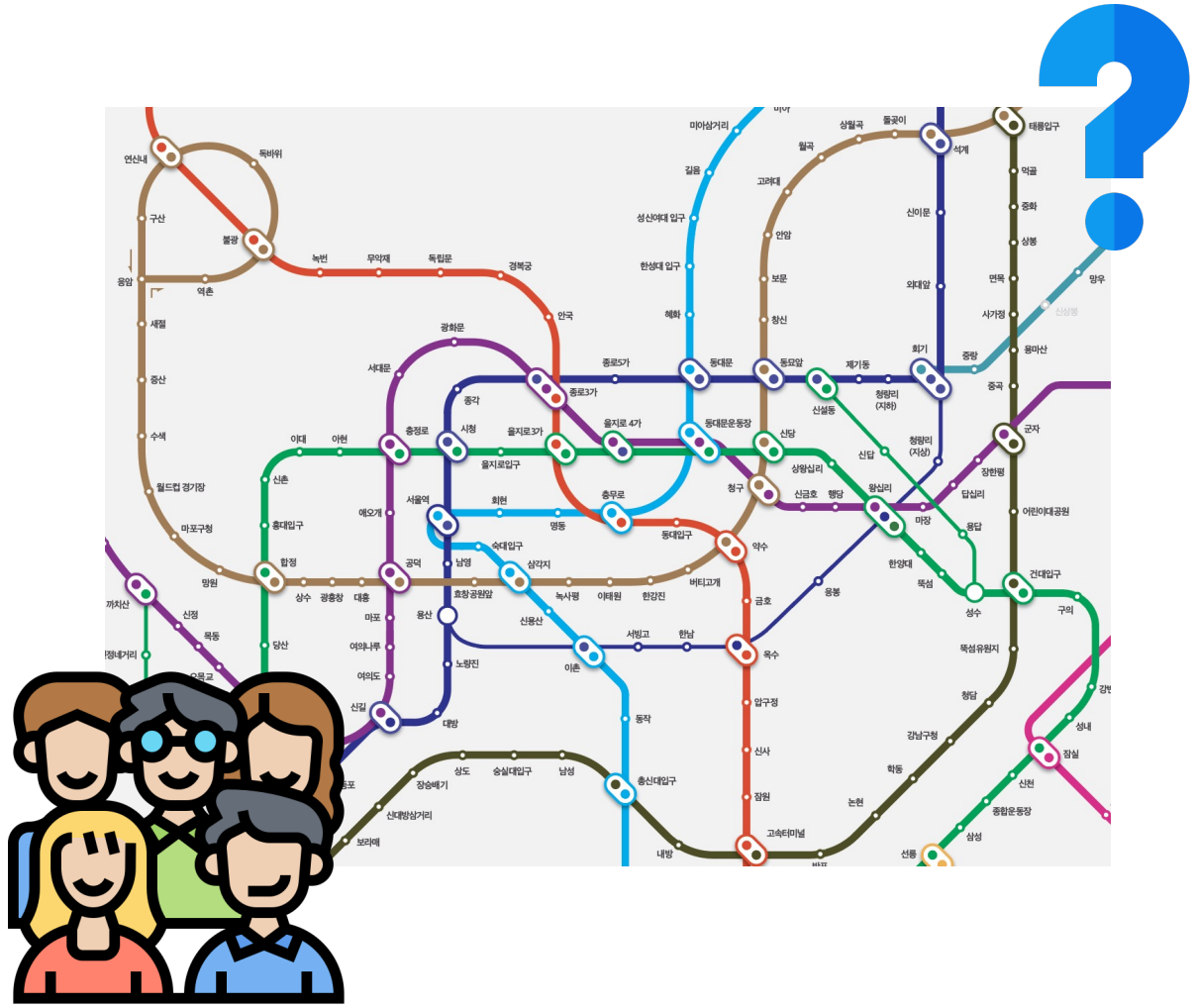
〈지하철 약속 역 추천 알고리즘〉

컴퓨터 알고리즘 및 실습
알고리즘 프로젝트 3조



< 목차 >





친구들이 서로 다른 역에 위치해 있을 때,
어떤 역이 가장 정확한 중간 지점 역일까?

"지하철 약속 역 추천 알고리즘"

```
인원 수: 5
1번 친구의 출발 호선: 2
1번 친구의 출발역: 신낭
2번 친구의 출발 호선: 2
2번 친구의 출발역: 성수
3번 친구의 출발 호선: 3
3번 친구의 출발역: 황제
4번 친구의 출발 호선: 4
4번 친구의 출발역: 충무로
5번 친구의 출발 호선: 5
5번 친구의 출발역: 올림픽공원
```

-입력 : 총 인원 수, 각각의 출발 호선, 출발 역 이름

-출력 : 입력된 역들의 중간지점

-사용자가 없는 역이나 호선을 입력할 경우
다시 입력하도록 알림

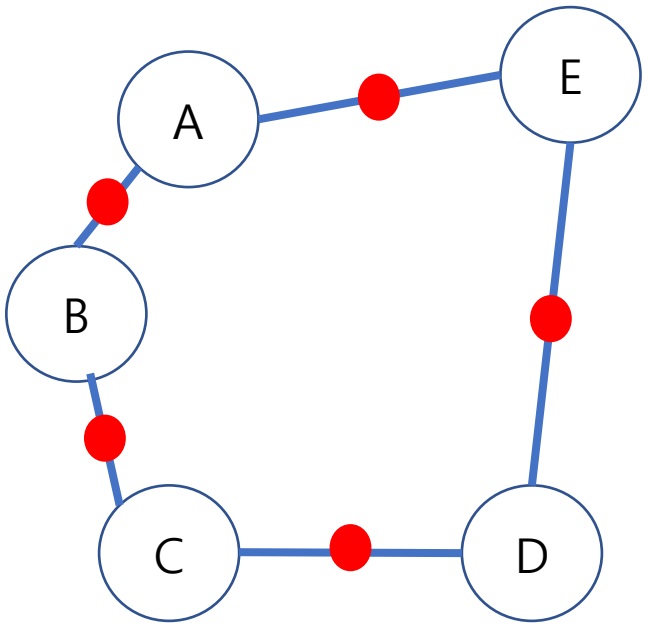
2. 수집한 데이터

01호선	신설동	156	37.575297	127.025087
01호선	제기동	157	37.578103	127.034893
01호선	청량리	158	37.580178	127.046835
01호선	동묘앞	159	37.572627	127.016429
01호선	소요산	1916	37.9481	127.061034
01호선	지제	1723	37.0188	127.070444
01호선	서울	150	37.554648	126.972559
01호선	시청	151	37.564718	126.977108
01호선	종각	152	37.570161	126.982923

```
vector<line> line_1{
    {"서울역", 37.554648, 126.972559, {"시청"}, {120}, 1, {4}, {133}, 1},
    {"시청", 37.564718, 126.977108, {"종각", "서울역"}, {120, 120}, 2, {1,
        2}, {84, 84}, 2},
    {"종각", 37.570161, 126.982923, {"종로3가", "시청"}, {120, 90}, 2, {},
        {}, 0},
    {"종로3가", 37.571607, 126.991806, {"종로5가", "종각"}, {90, 90}, 2, {3,
        5}, {98, 260}, 2},
    {"종로5가", 37.570926, 127.001849, {"동대문", "종로3가"}, {90, 90}, 2, {},
        {}, 0},
    {"동대문", 37.57142, 127.009745, {"동묘앞", "종로5가"}, {90, 60}, 2, {4},
        {162}, 1},
    {"동묘앞", 37.572627, 127.016429, {"신설동", "동대문"}, {60, 90}, 2, {6},
        {80}, 1},
    {"신설동", 37.575297, 127.025087, {"제기동", "동묘앞"}, {90, 90}, 2, {2},
        {108}, 1},
    {"제기동", 37.578103, 127.034893, {"신설동"}, {90}, 1, {}, {}, 0},
```

- 서울교통공사에서 제공하는 역 정보 기반

- 역에 대한 정보(역명, 역 좌표, 근접역 소요시간, 환승역, 환승소요시간)를 구조체 벡터로 나타냄

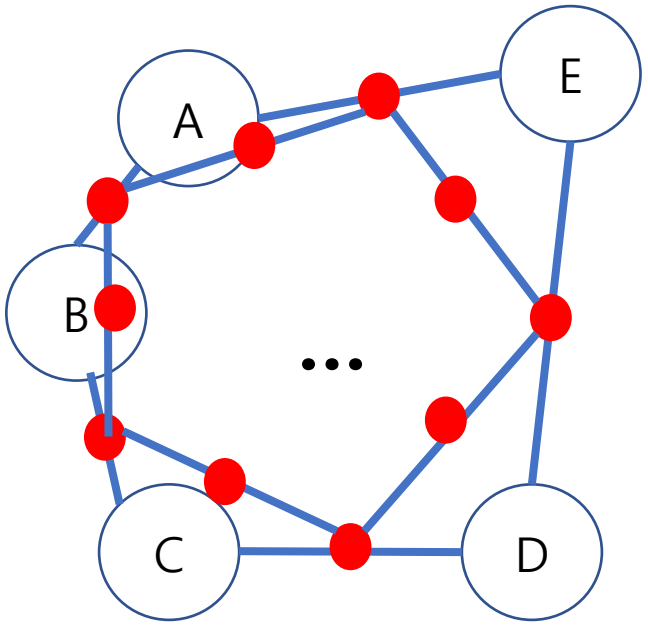


- 입력받은 역을 resultSet 벡터에 저장하고,
  그 중 두 개를 뽑아 두 지점의 정보 받기

- 두 역이 같은 호선인지 체크하고, 다른 호선이더라도
  해당 역이 같은 호선의 환승역일 경우 호선을 갱신

- 같은 호선일 경우 : 두 역 사이의 총 소요시간의 절반과
  같거나 큰 역 -> 중간역 찾기

- 다른 호선일 경우 : 두 역 사이에 환승이 가능한지 체크.
  소요시간에 환승시간을 포함하여 중간역 찾기

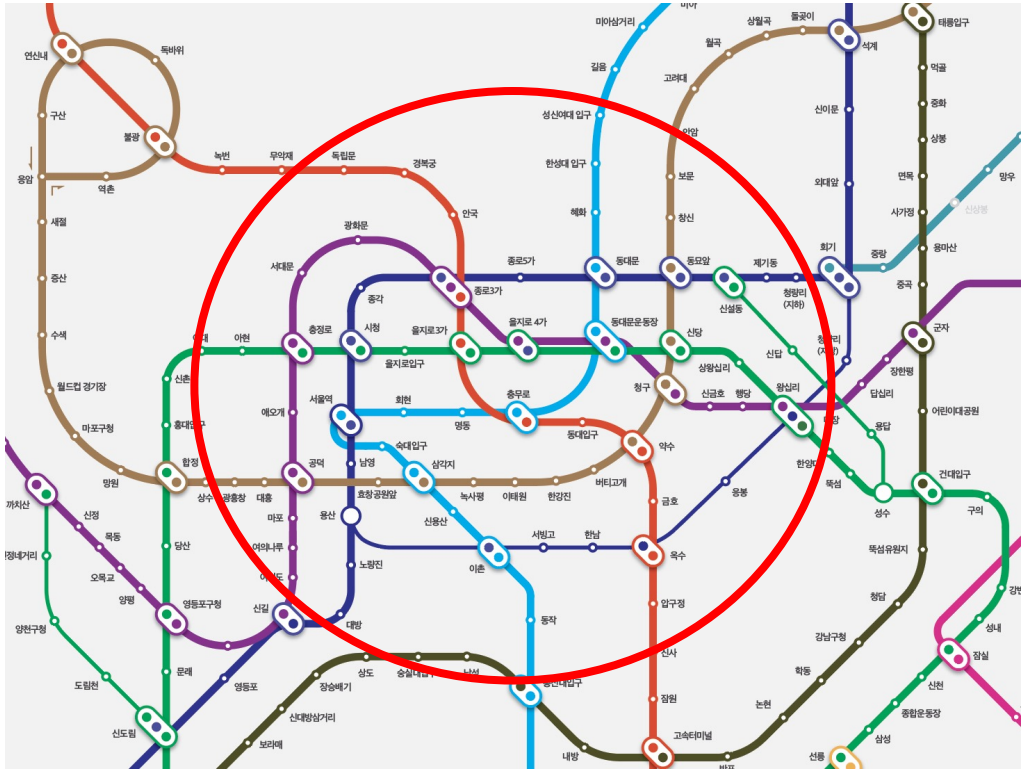


- 두 역의 중간지점을 resultSet에 저장하고, brute force 로 검사하여 중복역을 제거
- resultSet의 아이템 두개를 뽑아 중간지점 찾기를 반복
- 중간역이 하나만 남았거나, 중간지점찾기 반복이 100회 이상 넘어갈 경우 루프를 멈춤
- 최종결과 (역, 호선) 출력

```
----------[3] 커스텀 알고리즘----------
[3 , 안국] [5 , 아차산] [5 , 장한평] [2 , 사당] [3 , 고속터미널]
[5 , 신금호] [5 , 장한평] [2 , 신촌] [2 , 방배] [3 , 옥수]
[5 , 왕십리] [2 , 동대문역사문화공원] [2 , 대림] [3 , 고속터미널] [3 , 을지로3가]
[2 , 상왕십리] [2 , 신촌] [2 , 낙성대] [3 , 옥수] [5 , 을지로4가]
[2 , 을지로입구] [2 , 문래] [3 , 교대] [3 , 동대입구] [2 , 신당]
[2 , 신촌] [2 , 봉천] [3 , 압구정] [2 , 을지로3가] [2 , 동대문역사문화공원]
[2 , 문래] [2 , 방배] [3 , 금호] [2 , 동대문역사문화공원] [2 , 을지로입구]
[2 , 신림] [3 , 고속터미널] [3 , 충무로] [2 , 을지로4가] [2 , 신촌]
[2 , 사당] [3 , 압구정] [2 , 을지로3가] [2 , 시청] [2 , 영등포구청]
[3 , 교대] [3 , 금호] [2 , 을지로입구] [2 , 신촌] [2 , 신대방]
[3 , 신사] [3 , 충무로] [2 , 충정로] [2 , 당산] [2 , 낙성대]
[3 , 옥수] [2 , 을지로3가] [2 , 신촌] [2 , 구로디지털단지] [2 , 서초]
[3 , 약수] [2 , 시청] [2 , 당산] [2 , 서울대입구] [3 , 고속터미널]
[2 , 을지로3가] [2 , 신촌] [2 , 대림] [2 , 사당] [3 , 압구정]
[2 , 시청] [2 , 합정] [2 , 신림] [3 , 교대] [3 , 금호]
[2 , 이대] [2 , 문래] [2 , 낙성대] [3 , 신사] [3 , 충무로]
[2 , 합정] [2 , 구로디지털단지] [2 , 서초] [3 , 옥수] [2 , 을지로입구]
[2 , 영등포구청] [2 , 서울대입구] [3 , 고속터미널] [2 , 신림] [2 , 아현]
[2 , 구로디지털단지] [2 , 사당] [2 , 방배] [2 , 당산] [2 , 홍대입구]
[2 , 신림] [2 , 사당] [2 , 신대방] [2 , 합정] [2 , 당산]
[2 , 서울대입구] [2 , 봉천] [2 , 문래] [2 , 합정] [2 , 신도림]
[2 , 봉천] [2 , 구로디지털단지] [2 , 합정] [2 , 당산] [2 , 신대방]
[2 , 신대방] [2 , 영등포구청] [2 , 합정] [2 , 문래] [2 , 신대방]
[2 , 신도림] [2 , 합정] [2 , 합정] [2 , 신도림]
[2 , 당산] [2 , 당산]
최종 결과 : 당산역 2호선
```

-출력 결과 : 사용자가 입력한 데이터와 resultSet의 중간역들 출력

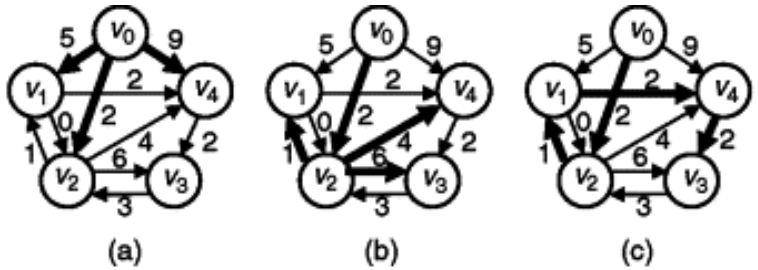
-시간복잡도 : O(N^2 logN)
(N: 입력 데이터 수)



- 출발역의 위도와 경도로 중심 구하고, 중심을 기반으로 원 생성
- 원 내부에 존재하는 모든 역들을 찾고, 중심지와의 거리를 기준으로 정렬
- 원 내부의 역들 중에서 출발역들의 호선과 가장 많이 겹치는 역 탐색
- 추천 역 결정

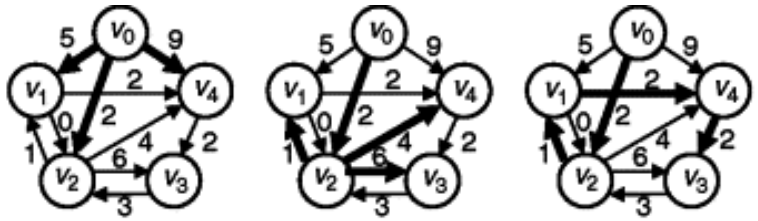
```
인원 수: 5
1번 친구의 출발 호선: 2
1번 친구의 출발역: 신촌
2번 친구의 출발 호선: 2
2번 친구의 출발역: 성수
3번 친구의 출발 호선: 3
3번 친구의 출발역: 홍제
4번 친구의 출발 호선: 4
4번 친구의 출발역: 충무로
5번 친구의 출발 호선: 5
5번 친구의 출발역: 올림픽공원

[2,신촌][2,성수][3,홍제][4,충무로][5,올림픽공원]

0.0062721 -> 금호
0.00202292 -> 약수
0.00891416 -> 동?입구
0.00717713 -> 청구
0.00809691 -> 신금호
0.00742344 -> 버티고개
0.00202292 -> 약수
0.00717713 -> 청구

[6,약수][3,약수][3,금호][6,청구][5,청구][6,버티고개][5,신금호][3,동대입구]

추천 역: [6,약수]
```

출력 결과 : 사용자가 입력한 데이터와 중심지 근점 역, 최종 추천 역
시간복잡도 : O(N) (N: 입력 데이터 수)



vertex	Predecessors					Shortest-Path Estimates				
	v_0	v_1	v_2	v_3	v_4	v_0	v_1	v_2	v_3	v_4
non	NIL	NIL	NIL	NIL	NIL	0	∞	∞	∞	∞
v_0	NIL	v_0	v_0	NIL	v_0	0	5	2	∞	9
v_2	NIL	v_2	v_0	v_2	v_2	0	3	2	8	6
v_1	NIL	v_2	v_0	v_2	v_1	0	3	2	8	5
v_4	NIL	v_2	v_0	v_4	v_1	0	3	2	7	5
v_3	NIL	v_2	v_0	v_4	v_1	0	3	2	7	5

- 역들의 가중치를 계산한 weight vector를 만들기 위해 입력데이터의 모든 호선들에 대해서 가중치를 계산
- 입력데이터의 역들을 출발역으로 설정
- 출발역과 같은 호선일 경우 : 모든 역의 가중치를 출발역과 떨어져 있는 거리로 초기화.
- 다른 호선이더라도 해당 호선으로 환승 가능하면 출발역과 같은 호선으로 처리



vertex	Predecessors					Shortest-Path Estimates				
	v_0	v_1	v_2	v_3	v_4	v_0	v_1	v_2	v_3	v_4
non	NIL	NIL	NIL	NIL	NIL	0	∞	∞	∞	∞
v_0	NIL	v_0	v_0	NIL	v_0	0	5	2	∞	9
v_2	NIL	v_2	v_0	v_2	v_2	0	3	2	8	6
v_1	NIL	v_2	v_0	v_2	v_1	0	3	2	8	5
v_4	NIL	v_2	v_0	v_4	v_1	0	3	2	7	5
v_3	NIL	v_2	v_0	v_4	v_1	0	3	2	7	5

- 출발역과 다른 호선일 경우 : 출발역에서 환승 가능한 모든 역을 찾고, 각 호선의 가중치를 계산하여, 기존보다 작은 값일 경우 갱신
- 환승 가능한 역이 없을 경우 : 가중치 최대값으로 설정
- 모든 출발역에 대해 반복
- 모든 출발역에 대해 도출된 가중치를 합함
- 가중치 합이 가장 작은 역 -> 중간역

```
인원 수 : 5
1번 친구의 출발 호선 : 3
1번 친구의 출발역 : 홍제
2번 친구의 출발 호선 : 4
2번 친구의 출발역 : 충무로
3번 친구의 출발 호선 : 5
3번 친구의 출발역 : 올림픽공원
4번 친구의 출발 호선 : 2
4번 친구의 출발역 : 신촌
5번 친구의 출발 호선 : 2
5번 친구의 출발역 : 성수
[3 , 홍제] [4 , 충무로] [5 , 올림픽공원] [2 , 신촌] [2 , 성수]
----------------[2] 다익스트라 알고리즘----------------
------resultWeight------
20035 20034 20033 20030 20031 20032 20035 20038 20041
91 86 81 78 75 72 71 70 71 72 73 74 75 76 79 82 85 88 91 94 97 99 100 101 101 100 99 98 99 100 101 104 107 110 113 116 118 119 120 121 122 123 124
127 130 133 136 139 144 149 154
93 90 87 84 81 78 75 72 69 66 63 60 59 60 61 62 63 64 65 66 67 68 67 68 71 74 77 80 83 88 93 98 103
91 86 85 84 82 79 76 73 70 67 64 61 62 67 72 77 82 87 92 97 102 107 112 117 122 127
151 146 141 136 131 126 121 116 111 106 101 96 91 90 89 88 86 83 80 77 73 72 71 70 71 72 75 78 81 82 85 88 91 94 97 100 103 106 109 112 115 118 121
 124 127 132 137 142 147 152
120 115 110 105 100 95 90 89 88 87 86 85 84 85 86 87 88 87 88 89 94 99 104 109 114 119 124 129 134 139 144 149 154
134 129 124 121 118 115 112 109 106 103 100 97 94 89 84 81 78 77 76 75 74 73 72 71 72 73 76 79 82 85 88 91 94 97 102 107 112 117 122 127 132
10115 10111 10107 10103 10099 10095 10091 10087 10083 10079 10075 10073 10071 10069 10071 10073 10075 10079

모두에게 최단 거리인 역 (다익스트라 알고리즘):
역 이름 : 교대
호선 : 3
```

출력 결과 : 사용자가 입력한 데이터와 가중치 배열, 추천역 출력
시간복잡도 : O(n^2) (N: 입력 데이터 수)

Middle 알고리즘

직관적으로
이해하기 쉬움
시간복잡도가 높음

Circle 알고리즘

위도/경도를 고려
-> 거리적으로 최적
특정 상황에서 탐색 범위 ⬆️
-> 중심지와 멀어질 가능성

Dijkstra 알고리즘

항상 최적해를 도출
시간복잡도가 높음