

**Data Structure Design Project**

**최종 보고서**

|  |  |
| --- | --- |
| **과 목 :** | **Data Structure Design** |
| **제출 일자 :** | **2018.12.26** |
| **담당 교수 :** | **정재은 교수님** |
| **학 과 :** | **컴퓨터 공학부** |
| **Members:** | **20143592 최영재** |
|  | **20146703 박지호** |
|  | **20153479 이동민**  **20152335 정진혁** |

**-310관 혼잡도 개선 및 다음 강의실 경로 안내**

**1. 프로젝트 아이디어 제안**

‘중앙대학교의 100주년 기념관 내에서 Dynamic Graph를 활용한 어떠한 프로그램이 있으면 모두에게 도움이 될까?’ 라는 물음에서 본 프로젝트의 아이디어는 시작한다. 평소 중앙대학교 서울캠퍼스에서 이용 학생이 가장 많은 본 건물에서는 강의실에 도착하기 위해 1층에서 엘리베이터를 기다리는 수 많은 학생들을 쉽게 볼 수 있다. 우리는 이 문제를 해소하기 위한 방법을 찾아보고, 또 학생들이 강의실을 쉽고 빠르게 찾아갈 수 있는 프로그램을 만들고자 하였다. 요일과 시간대 별로 건물의 이용 학생이 많고, 학생들이 도착지로 선택하는 강의실이 있는 층을 방문하는 엘리베이터의 개수를 늘린다면 이 문제를 해결할 수 있을 것이라고 생각하였다. 이를 위해서는 기존의 10대의 엘리베이터가 방문하는 운행 층을 유동적으로 바꾸는 방법을 고안하였다.

그리고 이렇게 개선된 엘리베이터 운영 시스템을 개인에게 맞춤형 강의실 이동안내 프로그램 안에 적용하기 위해서는 하나의 메인프레임에서 이를 효과적으로 선보일 수 있는 시나리오 구성이 핵심적이다. 무엇보다 이용자들이 직관적으로 본 프로그램이 해결하고자 하는 문제를 받아들이기 위한 구성이 필요할 것으로 예상되었다.

**2. 개발 배경**

**2.1 현 310관 엘리베이터 시스템의 비효율성 지적**

현재 310관에는 총 10개의 엘리베이터가 존재한다. 대부분의 수업은 3층과 6층,7층에서 이루어지며, 1층 출입문과 가장 가까운 A구역에 강의실이 몰려있다. 때문에, 많은 학생들은 1층의 A구역 엘리베이터에서 엘리베이터를 기다리는 경우가 많다. 수업이 밀집되어 있는 1교시와 점심시간 이후의 수업의 경우, 수업시작 15분 전부터 학생들은 모이기 시작한다. 때문에 개선된 엘리베이터에서는 시간/이용 학생수에 따라 특정시간(수업 시작 15분 전 ~ 수업 시작 5분 후)에 엘리베이터 운행방식을 가변적으로 설정하는 것이 필요하다. 그리고 본 프로젝트에서는 이 체계를 Magic Time System이라고 명칭하였다.

**2.2 개선된 엘리베이터 운영 시스템과 개인 맞춤형 경로 안내의 통합**

우선 엘리베이터 운영시스템을 개선하고, 학생 개인에게 맞춤형 강의실 이동 경로를 추천해주는 기능을 한 프로젝트 안에 담기 위에 프로그램을 구성하는 것이 중요하다. 이는 그래프를 형성하여 최단 경로를 찾는 알고리즘을 수행하는 전처리 단계에서 개선된 엘리베이터 시스템을 이용해 층간 노드의 연결하는 방식으로 문제를 접근하는 것이 가장 효과적일 것이라고 생각하였다. 따라서 이용자가 필요한 맞춤형 경로를 직접적으로 보여주고 그 경로를 찾는 과정에서 엘리베이터 시스템을 적용하도록 하였다.

**2.3 직관적인 최단 경로 안내를 위한 결과 시각화**

이용자가 강의실을 이동하기 위해 본 프로그램을 사용할 경우, 이용자는 프로그램의 실행 결과를 토대로 빠르게 경로를 인식할 수 있어야 한다. 이 과정에서는 단순히 문자로 출력된 결과를 보고 이동을 하는 것보다는 직접 이동 과정을 결과로 볼 수 있는 것이 구현되어야 한다. 본 프로그램은 이를 수행하기 위해 Processing이라는 Tool을 이용하여 프로그램에서 얻은 경로를 시각화하여 보여주고자 한다. 이용자는 이동을 하면서 본 프로그램 결과대로 다음 강의실로 쉽게 도착할 수 있게 된다.

**3. 프로젝트 목표**

본 프로젝트의 목표는 아래와 같다.

1. 종래의 엘리베이터 운행 시스템을 개선하여 층간 이동을 필요로 하는 대기 인원을 최소화한다.
2. 개인의 시간표를 입력했을 때, 다음 강의실로의 안내를 보다 정확하고, 직관적으로 보여준다.

이 두 가지 목표를 달성함으로써 중앙대학교의 혼잡 문제를 현실적으로 해결할 수 있다고 판단하였다.

**4. 개발 과정**

**4.1 데이터**

**4.1.1 데이터 가공**

**(1) 310관 강의정보**

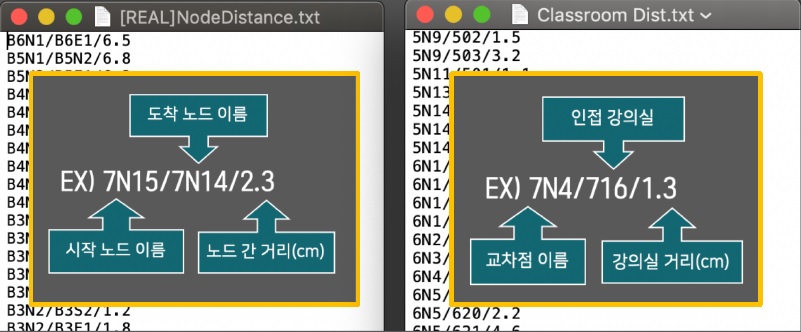
교수님이 제공해주신 310관 강의정보를 활용하였다. 이 강의정보를 바탕으로 310관의 유동인원이 어떻게 되는지 파악하기 위해서 데이터를 가공하였다.

우선 일별로 시트를 나누었다. 월, 화, 수, 목, 금의 5개의 시트로 구성하였다. 각 시트에는 시간마다 어느 층에 인원이 몇 명이 되는지에 대한 정보를 강의의 수강인원을 바탕으로 계산하여 작성하였다. 그리고 강의 시작시간의 경우 정시에 시작하는 강의도 있고 15분, 30분에 시작하는 강의들도 있었다. 그렇기 때문에 이 부분을 표현하기 위해서 1시간을 30분씩 나누어서 표현하였다. 이렇게 하여 각 시간대에 수강인원의 합계를 구할 수 있었다. 이 합계를 가지고 주로 2,3교시와 5,7교시에 그리고 6,7층에 강의가 많이 있다는 정보도 알 수 있었다.

처음에는 강의실 데이터를 종합하여 가공할 때 층 마다 구역을 나눠서 계산하였었다. 7층을 보게 되면 엘리베이터가 있는 구역이 크게 3군데가 있다. 그 중에 두 곳(A,B 구역) 만 기준으로 잡아서 구역 별로 인원을 따로 체크한 것이다. 하지만 이렇게 구역까지 나눌 필요는 없다고 생각하였다. 원래 이렇게 나누었던 이유는 A구역 엘리베이터가 있는 쪽에서 강의를 듣게 되면 A구역의 엘리베이터를 이용하는게 적합하다고 판단하였었다. 하지만 강의실로 이동하는데에 있어서 층간이동이 시간의 대부분을 차지하지, 층에서의 수평이동은 별로 중요하지 않다고 생각하여서 구역 별로 나누었던 부분은 하지 않기로 하였다.

**(2) 층내 노드간 거리 & 층내 강의실과 노드간 거리**

층 내의 교차점 노드간 그리고 강의실과 노드간 거리에 관한 정보이다. 직접 측정하나 도면에서 거리로 측정하나 서로 변환할 수 있는 관계라 310관 도면을 시설팀으로부터 획득하여 직접 노드간 거리를 측정하는 방식으로 진행하였다.



**NodeDistance.txt 와 RoomDistance.txt**

코너점을 중간에 포함한 노드들을 제외하고는 노드간 거리를 직선 거리로 측정하였다. 코너점을 포함하면 그에 맞춰 여러 번 측정하고 더하는 것으로 하였다. 또한 강의실 거리는 인접 강의실을 교차점에 직선으로 잇는 선을 측정하는 것으로 하였다.

**(3) 층간 이동 거리 정보**

각 층에서 형성된 그래프를 서로 연결하기 위해서는 층 간 데이터가 필요하다. 이는 층간 이동을 하기 위한 수단으로 이어지게 된다. 크게 3곳의 엘리베이터와 3곳의 계단 그리고 지하층과 1층,2층,3층을 연결하고 있는 상행 에스컬레이터가 있다. 이 과정에서 엘리베이터는 개선된 엘리베이터 시스템인 Magic Time System의 결과를 바탕으로 10개의 엘리베이터가 각각 층간 edge를 형성하게 되고 계단의 경우 중앙계단은 올라가는 계단과 내려가는 계단의 위치가 변측적이기 때문에 각각 해당층에서 올라가는 계단을 윗 층의 올라오는 계단과 연결 짓는 작업이 필요하게 된다. 그리고 각층의 그래프를 잇는 이 노드들의 weight는 이동시간을 기반으로 한다.

**4.2 엘리베이터**

**4.2.1 Magic time**

**(1) Magic time이란?**

Magic time이란 따로 붙인 이름이다. 특정 층에 많은 인원이 몰리게 될 때 그 층을 Noise Floor로 명명하였다. 이러한 Noise floor를 가지는 시간을 magic time이라고 하였다. 지금 현재의 엘리베이터보다 더 효율적인 운행을 하기 위해서 magic time을 시행하기로 하였다. 그리고 이 magic time을 시행하는 시간은 정각을 기준으로 정각 15분전부터 정각 5분후까지를 기준으로 삼았다. 그 이유는 보통 강의가 정각에 시작되며, 강의 시작 전,후의 시간으로 많은 사람들이 몰리기 때문에 이 시간을 magic time으로 정하고 엘리베이터를 운행하기로 하였다.

**(2) Magic time조건**

특정 층에 많은 인원이 몰리게 되면, 사람이 너무 많아서 사람들이 정상적으로 이동하기 힘들다고 판단하였다. 그래서 어떠한 조건으로 noise floor를 선정하여 magic time을 실행해야 할지에 대해 많은 회의를 하였다. 회의를 한 결과는 다음과 같다.

해당 시간 310관 이용자가 X명 이상이고, 그 해당 시간에 이용자가 Y% 이상인 층을 Noise floor로 하기로 하였다. 하지만 이렇게 되면 해당 시간에 엄청 많은 인원수가 몰리지는 않지만 한 층에만 집중적으로 몰리는 층이 있기 때문에 이 부분도 고려하기 위해서 해당 시간에 이용자가 X명 이상은 아니지만 Z명이상으로 많을 경우를 noise floor로 선정하였다.

**(3) 엘리베이터 운행 정보**

Magic time에 운행하는 엘리베이터는 평소의 운행하는 엘리베이터와는 다르다. 엘리베이터마다 각각 운행하는 층은 다음과 같다.

|  |  |
| --- | --- |
| A 구역 지상 층, C구역 엘리베이터 | 1층과 Noise층만을 운행 |
| A 구역 지하, B구역 지하층 운행 엘리베이터 | 정상 운행 |
| B 구역 지상 층 운행 엘리베이터 | Noise 층을 제외한 홀/짝 운행 |

모든 엘리베이터를 바꿔서 운행하게 되거나 하면 noise층을 가지 않는 사용자에게는 불편할 수도 있다. 그렇기 때문에 그러한 불편함을 해소하기 위해서 정상운행 하는 엘리베이터도 남겨두었다.

**(4) 임의의 X, Y, Z 값에 따른 평균 운반 인원 수**

특정 값인 X, Y, Z에 맞는 층을 Noise floor로 선정하고 magic time을 시행하기로 하였다. 그러기 위해서 가장 많은 사람을 윗 층으로 실어 나를 수 있는 최적의 X, Y, Z 값을 찾을 필요가 있었다. 다음과 같은 과정을 거쳐서 X, Y, Z 값을 계산하였다.

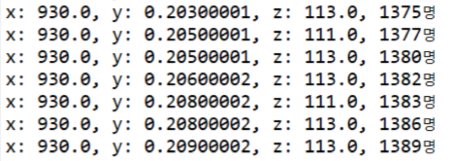
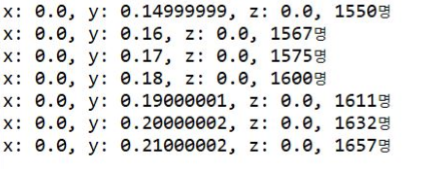
우선, 임의로 X, Y, Z 값을 주고 설계를 진행하였다. X, Y, Z 값이 있게 되면 그 조건에 맞는 층들이 noise floor로 정해지게 된다. 이렇게 noise floor가 정해지게 되면 위의 (3) 엘리베이터 운행 정보에서 정한 운행 정보대로 운행 했을 때 엘리베이터 마다 소요시간이 나오게 된다. 이 값을 날짜(day)별, 시간(time)별, 엘리베이터 번호 별(Elevator\_num)로 나눠서 삼차원 배열에 저장하였다.

이 소요시간을 바탕으로 magic time 시행시간인 20분동안 엘리베이터가 얼마나 운행할 수 있는지 계산하고 그 횟수에 엘리베이터의 정원 (ex. 정원이 25명이면 실질적으로 엘리베이터에 25명이 모두 탈 수 없기 때문에 적정 정원을 5를 빼서 20명으로 하여 계산하였다) 값을 곱해서 20분, 즉 magic time 시행 시간동안 얼마나 많은 사람들을 운반할 수 있는지 계산하였다.

이렇게 계산된 운반 인원 수를 요일, 시간에 관계없이 모두 더해서 평균 값을 계산하였다. 왜냐하면 이 X, Y, Z값은 바뀌는 값이 아닌 계속 적용되는 값으로 이 값에 따라서 magic floor가 바뀌기 때문이다. 그러면 임의로 정한 X, Y, Z 값에 따른 사람을 윗층으로 운반할 수 있는 평균 수가 나오게 된다.

**(5) 최적의 X, Y, Z 값 찾기**

위(4)에서는 임의의 X, Y, Z값을 지정하고 그 값을 가지고 평균 운반 수를 구하였다. 하지만 프로젝트의 취지에 맞게 진행하기 위해서는 적정한 X, Y, Z 값이 필요했다. 그래서 X, Y, Z 값을 반복문을 통해서 계속 바꿔주면서 최적의 값을 찾으려 노력하였다.



**[Fig. 4.2.1]** X:0, Y: 0,21, z: 0일때 평균 운반 인원 수 (왼쪽)

**[Fig. 4.2.2]** X: 930, Y: 0.209, Z: 113일 때 평균 운반 인원 수 (오른쪽)

<Fig. 4.2.1>에서 X의 값이 0인 경우는 매 시간마다 magic time을 시행할 경우를 이야기한다. 그래서 가장 높은 평균 운반 수인 1657명이 나오게 된다. 이 결과는 당연하다. 매 시간마다 급행을 이용한다면 상식적으로 더 많은 인원을 효율적으로 나를 수 있게 된다. <Fig. 4.2.2>에서는 X 값을 달리했을 때의 결과이다. 시간때에 930명 이상의 유동인원이 있고 그 시간때에 0.209 % 이상인 층을 noise floor로 하고 예외적인 상황의 Z 값을 113명으로 두었을 때 평균 운반 수는 1389명이 나오게 된다.

**(6) Magic time을 위한 추가조건**

시뮬레이션을 하였을 때 X가 0인경우, 즉 매 시간마다 magic time을 시행하는 경우가 물론 가장 많은 인원을 나를 수 있게 된다. 하지만 이렇게 되면 매 시간마다, 각 엘리베이터마다 운행하는 층이 계속해서 바뀌게 된다. 그렇게 되면 사용자에게 혼란을 끼칠 수 있다. 그렇기 때문에 X 값을 어떠한 값으로 해야 타당할지에 대해서 논의하였다. 토의 결과는 다음과 같다.

실제 시간을 측정한 것을 바탕으로 나온 Worst case의 값은 940명 이었다. 즉 유동인원 940명 까지는 모두 잘 운반할 수 있다고 가정하였다. 그래서 940명 이상일 때 magic time을 시행하는게 맞다고 생각하였다. 그래서 940명 부근에서 X, Y, Z 값을 증가시키며 사람을 운반 할 수 있는 최대의 값을 찾았다. 그 결과가 <Fig. 4.2.2>이다. Magic time을 매 시간 운행 했을 때보다는 적긴 하지만 그래도 정상 운행할 때의 940명 보다는 현저히 많은 수치이다.

**(7) Magic time의 효율성**

Magic time을 적용해서 엘리베이터를 다이나믹(dynamic)하게 운행했을 때 과연 좋은 것일까? 하는 의문이 들 수도 있다. 이러한 의문을 해결하기 위해서 위와 같이 시뮬레이션을 시행한 것이다. Magic time을 시행했을 때 평균 운반 수를 보게 되면 확실히 정상적으로 즉, 지금처럼 엘리베이터를 운행했을 때의 Worst case의 값인 940명 보다 1.5배 정도 많은 수치이다. 그렇기 때문에 기준(X, Y, Z)을 두고 noise floor를 선정하여 magic time을 시행하는게 효율적일 수 있다는 이야기이다.

**(8) Dynamic**

매일, 매 시간마다 있는 강의수도 다르며 듣는 수강생들의 수도 다르다. 그래서 기준에 맞는 noise floor도 계속해서 바뀌게 되고, 엘리베이터 마다 각각 운행하는 층들이 달라지게 된다. 이 부분을 적용하여 그래프에 추가하기 때문에 Dynamic Graph가 될 수 있었다.

**4.3 Graph**

**4.3.1 Grpah 생성**

Graph를 생성할 때에서 4가지 부분을 고려하였다.

첫 번째로 어떤 것을 Node로 정할 것이냐 하는 부분이다. 처음에는 교차점, 계단, 엘리베이터, 에스컬레이터와 강의실 전체를 Node라고 생각하고 Graph를 만들려고 하였다. 그러나 그렇게 되면 Node가 많아지는 것과 Path가 부정확해질 것이라 판단되어서 Node안에 배열형태로 저장하는 형식으로 바꾸게 되었다.

두 번째로 Node들간 어떤 방식으로 연결을 할 것인가의 부분이다. 토의 끝에 연결과정에서 Node들간의 Node와 weight(거리)들을 Node안에 저장하는 방식(ArrayList)을 택하였다. 각 Node에는 해당 Node와 연결한 된 Node를 저장하는 ArrayList와 weight를 저장할 ArrayList를 따로 만들지만. 같은 index로 해당Node와 이어져있는 Node와 weight를 함께 참조할 수 있게 하였다. 예를 들어Node A과 Node B을 이어줄 때 Node A안에서 ArrayList형태로 Node B와의 weight(거리)를 저장하고 Node B의 weight(거리)를 저장한 ArrayList의 해당 index로 Node B와 ArrayList형태로 연결한다,. Node B도 똑같이 Node A의 거리를 저장하고, Node A와 연결하게 된다.

세 번째는 Node와 강의실간의 weight(거리)는 어떻게 할 것인가의 부분이다. 해당 Node와 강의실이 연결해야 한다면, 그 Node 안에 있는 배열(강의실배열)에 강의실을 저장하고, 그 강의실이 저장되어있는 배열(강의실배열)의 해당 index를 가지고, 거리를 저장하는 배열(강의실거리배열)에 weight를 저장하여, 같은 index로 강의실과 거리를 참조할 수 있게 하였다.

네 번째는 층과 층 사이의 연결 부분이다. 층 간에는 3가지 종류가 있는데, 엘리베이터, 에스컬레이터, 계단이다. 엘리베이터는 현재 운행하고 있는 10개만을 고려했으며, 사용자가 클릭 이벤트를 하였을 경우, 10개의 구역의 엘리베이터 운행정보를 가지고 와 해당 층의 엘리베이터들과 연결하게 된다. 그리고, 에스컬레이터는 경우는 1층씩 올라가는 것으로 연결 하였고, 계단은 1층을 제외하고 나머지 층은 1층 씩 올라가거나 내려가는 식으로 연결하지만, 2층에는 강의가 없는 관계로 가운데 계단 사용을 제외하고 1층에서 바로 3층으로 올라가는 식으로 연결하였다.

**4.3.2 Shortest Path 찾기**

일단 이 프로젝트에서는 같은 층간에서의 이동은 따로 고려하지 않는다. 따라서 층간 이동이 있을 시에만 Path를 찾게 된다. 과정을 설명하기 전에 전제조건을 설명하고 넘어가겠다.

1. 앞서 언급한 것과 같이 같은 층간에서의 이동은 따로 고려하지 않는다.
2. 하행은 무조건 계단을 이용한다.
3. 상행, 하행에서 엘리베이터를 이용하는 경우 무조건 출발지 층에서 도착지 층을 도착하는 경우라고 생각한다.
4. 엘리베이터로 이용이 가능한 경우 계단과 에스컬레이터를 이용한path를 찾지 않게 한다.
5. 계단과 에스컬레이터가 같이 있는 경우(지하) 하행은 계단, 상행은 에스컬레이터를 이용한다.
6. 3층 미만으로 이동 시 상행이든 하행이든 계단이나 에스컬레이터를 이용한다.

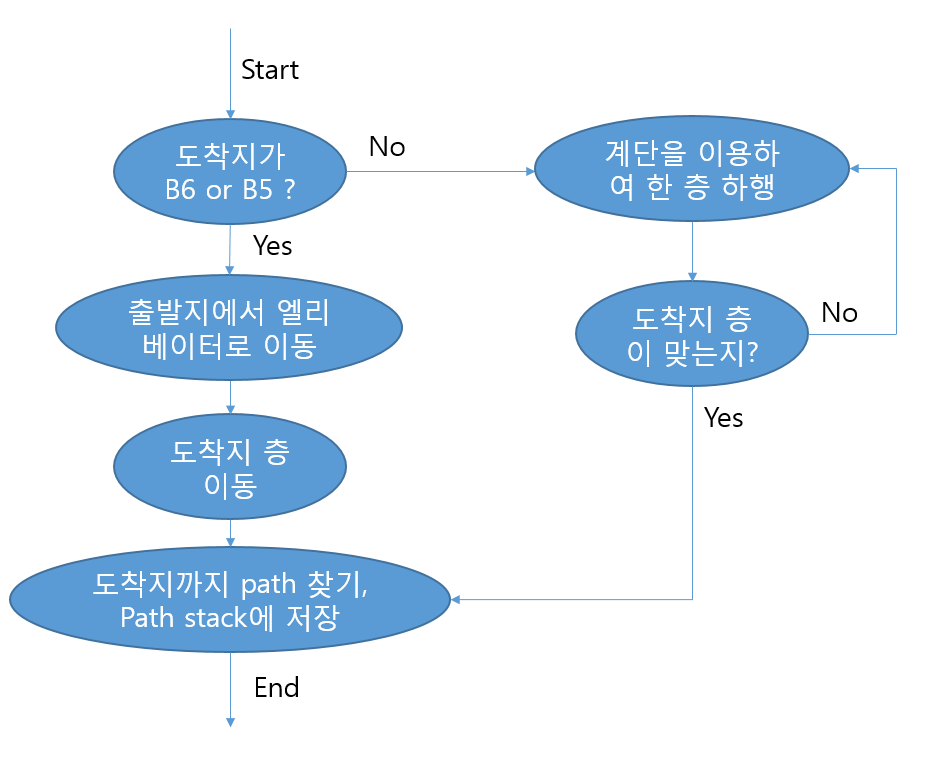
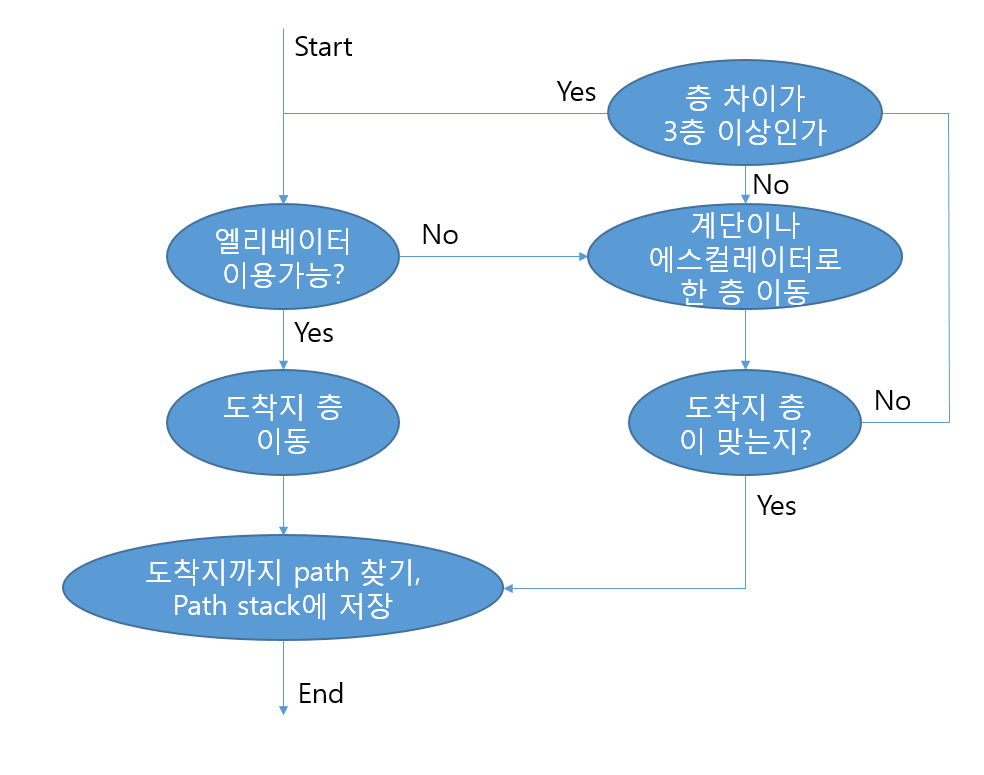
path찾는 과정은 다음과 같다.

첫 번째로 수평이동이다. input으로 출발지와 도착지를 받게 되면, 출발지에서 엘리베이터나 에스컬레이터, 계단을 이용하기 위해서는 출발지를 저장하고 있는 Node에서 엘리베이터, 에스컬레이터, 계단의 Node를 찾아가야 된다. 따라서 처음에 출발지를 저장하고 있는Node를 찾고, 그 Node와 엘리베이터, 에스컬레이터, 계단의 Node간의 Path를 찾게 된다. 이 과정에서 Dijkstra 알고리즘을 사용하여 계산을 하게 된다. 그리고 이렇게 나온 Path들은 stack에 쌓이게 된다.

두 번째는 층간 이동이다. 엘리베이터를 이용하는 경우는 비교적 간단하다. 엘리베이터를 이용할 경우 바로 도착지점 층에 갈 수 있는지 10개 각각 조사를 한다. 만약 가는 것이 있다면 가장 빠르게 갈 수 있는 엘리베이터를 이용하고 도착지점 층으로 이동하면, 해당 엘리베이터(도착지층) Node와 도착지를 Dijkstra알고리즘으로 Path를 찾게 된다. 에스컬레이터는 상행에서만 고려를 한다. 따라서 상행이고, 에스컬레이터를 이용할 경우 한 개의 층을 이동한다. 그리고 만약 도착지의 층과 같은 층이 아니라면, 처음부터 지금까지의 Path를 stack에 저장하고, 첫 번째 과정으로 돌아가게 된다. (출발지는 층간 이동한 Node로 바뀐다.)만약 도착지와 같은 층이라면 엘리베이터와 마찬가지로 Dijkstra 알고리즘으로 Path를 찾는다. 계단은 상행, 하행 둘 다 가능하다. 하지만 에스컬레이터와 같은 알고리즘을 가지기 때문에 따로 설명하지는 않겠다. 그리고 만약 도착지에 도착하는 path들은 goalRoute라는 stack에 저장한다.

세 번째는 첫 번째, 두 번째 과정이 끝날 경우 goalRoute에 있는 stack에 있는 것을 하나씩 꺼내면서, 가장 거리가 작은 Path를 찾는 것이다.

도식화하면 다음과같다.



**[Fig. 4.3.1]** 상행 도식화 **[Fig. 4.3.2]** 하행 도식화

**4.4 Visualization**

**4.4.1 시간표**

**(1) swing vs JavaFX**

우선 결론부터 말하자면 swing을 이용하여 GUI를 표현하였다. 처음에는 JavaFX를 이용하려고 했고 이용하였다. 왜냐하면 다들 주위에서 JavaFX가 GUI를 구현하는데 있어서 swing보다 더 직관적이고 쉽다고 했기 때문이었다. 하지만 무엇보다도 swing을 쓰게 된 계기는 설치하고 설정하는 과정에 있어서 문제가 많아서 잘 되지 않았다. 그리고 swing은 많이 다루어 보았기에 더 친숙해서 swing을 선택하게 되었다.

**(2) 수강하는 강의 입력 받기**

전체 강의 정보가 담겨있는 엑셀 파일을 학사팀을 통해서 구할 수 있었다. 개인 강의시간표에 따른 루트 추천을 해주어야 하기 때문이다. 310관에서만 강의를 듣는 것이 아니기에 학교 전체의 강의정보가 필요하였다. 파싱(parsing)된 데이터를 가지고 강의의 일부를 입력하게 되면 그 강의명을 포함(contain)하는 강의를 찾아서 CourseInfo class에 저장하였다. 그리고 저장된 데이터를 콤보 박스에 표시하여 사용자가 자신이 수강하는 강의를 선택할 수 있게 하였다.

**(3) JTable에 강의 표시**

(2)번에서 입력한 강의 정보를 시간표처럼 한눈에 볼 수 있도록 JTable에 표시하기로 하였다. 강의를 검색하고 콤보 박스에서 선택하게 되면 선택한 강의가 강의 날짜와 시간에 맞게 table에 표시되도록 하였다. 강의를 콤보 박스에서 선택하지 않은 경우에는 강의를 선택해야 한다는 팝업창을 띄우며 예외처리를 해주었다.

**(4) 테이블 클릭 이벤트**

수강하는 강의를 클릭하였을 때의 행(row)값과 열(column)값을 알아낸다. 알아낸 행 값과 열 값을 가지고 그 테이블에 어떠한 강의 정보가 저장되어 있는지 판단한다. 판단한 정보를 가지고 도착지와 출발지를 계산하여 그 값을 가지고 graph를 생성하고 다익스트라 알고리즘을 통해서 최적의 루트를 찾게 된다.

**(5) 출발지와 도착지**

Frame 클래스의 cal\_start\_end\_place 함수에서 계산하게 된다. 강의를 클릭 했을 때 클릭한 강의를 들으러 가는 곳으로 정하고 그 전의 강의 정보를 찾아서 전 수업의 강의실에서 클릭한 강의의 강의실로 안내하도록 하였다. 그리고 클릭한 시간대, 즉 이동하는 시간대의 엘리베이터의 운행정보도 넘겨주어서 graph를 생성하였다.

우선 클릭한 부분의 강의정보를 바탕으로 도착지 강의실이 어디인지 찾고 테이블의 행을 위로 올리면서 출발지를 찾게 된다. 출발지는 전에 강의가 없는 경우는 디폴트(default)값으로 후문에서 오는 것으로 설정하였으며 공강 시간이 2시간 이상이 될 경우에는 전의 강의실이 아닌 제2 공학관인 6층 PC실에서 오는 것으로 설정하였다. 그 외의 경우에는 테이블에 클릭한 곳에 써 있는 강의명과 다른 강의명이 나올 때 까지 탐색한다. 이렇게 하여 출발지와 도착지를 시간표 테이블에서 찾아내었다.

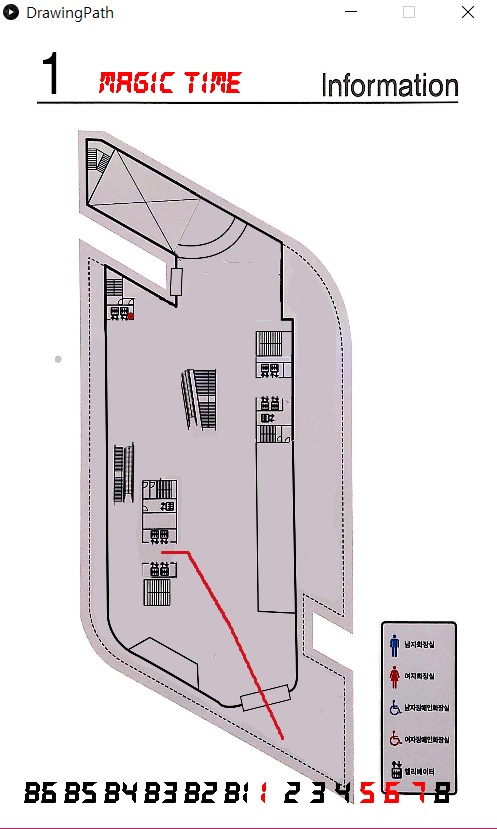
운행하는 엘리베이터 정보는 위 에서처럼 테이블에 쓰여있는 정보와 달라지기 전까지 올라가며 강의의 시작시간을 탐색하여 탐색한 시간을 바탕으로 이동하는 시간의 엘리베이터 정보를 계산하였다.

**4.4.2 프로세싱(Processing)**

결과 화면 구현은 프로세싱(Processing)으로 진행하였다. 프로세싱(Processing)은 자바(java) 기반으로 뉴 미디어 아트, 시각 디자인 공동체를 위해 개발된 오픈 소스 프로그래밍 언어이자 통합 개발 환경(IDE)으로 다른 툴보다 결과 화면을 시각적으로 잘 구현해 줄 것으로 기대하여 채택하였다.

프로세싱(Processing)이 자바 기반 언어이기 때문에 Eclipse 환경에 그대로 적용하는데 크게 무리가 없었고, 결과 화면에 텍스트 출력을 지원하여 프로젝트 주요 목적인 매직 타임(Magic Time)에 관한 정보를 담기에도 적합하였다.

처음 개발에 대한 개요는 앞서 개발된 코드들이 낸 총 3가지 결과(①최단경로 결과 ②magic time 시행 여부 ③이용할 엘리베이터 운행 정보에 관한 결과)를 String으로 받고 그 결과를 구현하는 클래스 내에서 표현하기 적합한 형태의 자료구조로 변환하여 그대로 출력하기로 계획하였다. 또한 우리가 구현한 그래프 내의 모든 노드를 500\*800 이미지에서의 좌표로 nodeLocation.txt 파일에 저장하고 파일 입출력 하였다.







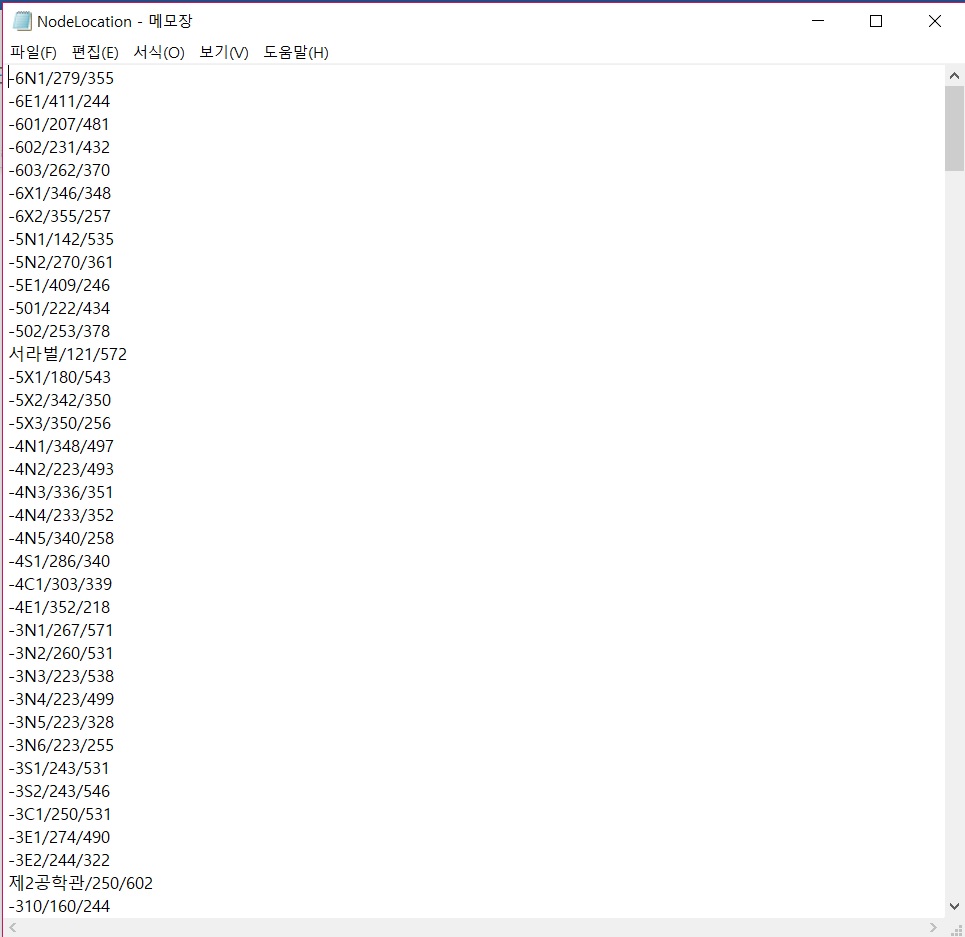




**Processing 결과 화면**

프로세싱(Processing)이 자바 기반 언어이기 때문에 Eclipse 환경에 그대로 적용하는데 크게 무리가 없었고, 결과 화면에 텍스트 출력을 지원하여 프로젝트 주요 목적인 매직 타임(Magic Time)에 관한 정보를 담기에도 적합하였다.

처음 개발에 대한 개요는 앞서 개발된 코드들이 낸 총 3가지 결과(①최단경로 결과 ②magic time 시행 여부 ③이용할 엘리베이터 운행 정보에 관한 결과)를 String으로 받고 그 결과를 구현하는 클래스 내에서 표현하기 적합한 형태의 자료구조로 변환하여 그대로 출력하기로 계획하였다. 또한 우리가 구현한 그래프 내의 모든 노드를 500\*800 이미지에서의 좌표로 nodeLocation.txt 파일에 저장하고 파일 입출력 하였다.

****

**nodeLocation.txt**

각각의 노드를 직접 500\*800 크기의 프로세싱 작업창에서 x,y 좌표를 조사하였다. 약 300여개의 노드가 존재했고 DrawingPath 클래스 내의 노드 위치를 찾는 메소드는 매번 파일 입출력을 통해 노드 위치를 표시한다. 양식은 “노드이름/X좌표/Y좌표” 이다.

**(1) 최단경로 결과**

최단경로 결과는 곧 프로세싱 결과 화면에서 동적인 선으로 나타내어 지는 부분이다. 층별로 이미지를 로드(load)하여 나타내야 하기 때문에 결과 String을 약속된 기호(“/”)로 경로 노드를 구분하여 그것들을 층별로 구분해야 했다. 그래서 pathDivided라는 이차원 String 배열로 경로를 나타내기 편하게 자료구조를 차용하였다.

최초에 경로를 나타내는 데에 움직이는 즉, 동적인 선이 아닌 정적인 선을 노드마다 시간 간격을 두고 나타내는 방식으로 구현하였지만 사용자가 알아보기 힘들 것이라 판단하여 경로 노드 간의 선을 동적으로 나타내기 위해 프로세싱내 자료형인 PVector을 사용하였다. 그리고 1초에 60번 반복하는 프로세싱의 draw라는 함수의 특징을 이용하였다. 경로선을 더 세밀하게 나눌수록 정확한 경로가 표시되지만 그리는데 오랜 시간이 걸린다는 점을 고려해 그 경로 노드간 점들을 70등분하여 찍는 것으로 하였다.

또한 그래프 형성을 위해 자료 수집하는 당시 교차점 노드에 대한 기준 설정 오류로 인해 프로세싱에서 노드가 찍히지 않은 부분은 선을 잇지 않아 다소 비현실적인 선의 움직임으로 인해 데이터를 수정해야할 일이 생겼다. 그러나 너무나 많은 손실을 요구하여 차라리 부가적인 코너점 노드를 예외로 처리하여 예외적인 경로 발생시 프로세싱 코드 내에서 그 사이에 추가하는 방식을 사용하였고 그 과정에서 String의 ArrayList가 핵심적으로 사용되었다.

**(2) magic time 시행 여부**

프로젝트 주요 목적인 magic time이 최소 30분 단위로 바뀌는 엘리베이터 운행 정보 때문에 사용자가 이용하기 어려울 것이라 생각해 그 정보를 담아야 한다고 판단하였다. 따라서 제시각각 변하는 magic time 여부를 Frame 클래스에서 Boolean 자료형으로 매 클릭 이벤트마다 저장하여 그것을 넘겨주면 magic time 시행시에 위의 결과 화면 창의 2번 부분을 빨간색으로 강조하는 것으로 하였다. magic time이 시행되지 않는 경우 검은색으로 표현된다.

**(3) 이용할 엘리베이터 운행 정보에 관한 결과**

magic time 시에 총 10개의 엘리베이터 중 4개는 noise 층을 운행하고 4개는 기존과 같이 정상 운행, 2개는 noise 층을 제외한 홀/짝 운행을 한다. 시간마다 출발지, 목적지마다 바뀌는 경로, 그리고 그 때마다 탑승하는 엘리베이터가 바뀌기 때문에 그 엘리베이터에 대한 운행정보를 담아야 한다고 판단했다. 위의 결과 화면의 3번 부분이 그것이고 결과 화면 예시의 경로 경우엔 1,5,6,7층을 운행하는 엘리베이터를 탑승한다고 정보를 나타내어 주는 것이다.

**5. 프로젝트의 한계점**

**5.1 310관 이외의 건물에서 진행되는 수업의 예외처리 미흡**

현재 310관 근처에 있는 서라벌관과 제2공학관, 기숙사에서 진행되는 수업에 관해서는 인접 게이트에서 부터 최단경로를 탐색하도록 처리가 되었으나,

그 이외의 건물에서 진행되는 수업에 관해서는 예외처리가 미흡하다. 뿐만 아니라, 310관 이외의 건물에서 진행되는 강의실에서 똑같이 310관 이외의 건물로 향할 경우 이를 팝업다이얼로그로 안내하지 못하였다. 이 부분에 관해서는 이용자에게 최단경로를 제안하지 못하는 점을 안내하면 좋을 것 같다.

**5.2 경로 시각화 과정에서 Noisy Floor에 대한 안내 미흡**

현재 Processing을 이용한 경로안내 창의 하단부분에서 이용자가 이용하는 이동수단을 통해 이동할 수 있는 층이 표시되고 있다. 이는 계단이나 에스컬레이터를 이용하는 이용자에게는 혼란을 야기할 수 있다. 따라서 이 부분에는 단순히 해당 시간의 Noisy Floor와 이용해야할 엘리베이터 호수를 표기하고 계단과 에스컬레이터를 이용하는 경우에는 이 부분에 알림이 뜨지 않도록 설정하는 것이 이용자의 혼란을 줄일 수 있을 것으로 생각된다.