설계 프로젝트 C: 민혜의 신나는 세계일주^O^/

2015104197 이민혜 / 2015104220 정종윤

① 시스템 개요 및 목표

ⓐ 개요

최적의 여행 루트를 얻기 위한 경로 탐색 프로그램을 만든다. 유저는 출발장소에서 시작하여 주어진 시간 내에 도착지에 도달해야 한다. 출발 장소와 도착 장소 사이에는 지역이 존재하며, 각 지역에는 관광지가 존재한다. 관광지는 방문 시에 시간을 소모하지만 만족도를 얻을 수 있다.

프로그램은 최적의 경로를 탐색하기 위해, 사용자에게서 탐색 가능한 시간을 입력 받아 다음과 같은 기능을 수행할 수 있다. 첫 번째로 주어진 시간 내에 가장 많은 관광지를 가는 경로를 탐색하거나, 두 번째로 주어진 시간 내에 가장 큰 만족도를 얻을 수 있는 경로를 탐색하는 방법이다. 이때, 지역은 중복으로 지나갈 수 있지만 지역에 있는 관광지를 중복 방문 할 수 없다.

ⓑ 목표

시나리오에 명시된 기능들을 모두 포함하는 경로 탐색 프로그램을 구현하며, 각각의 조건을 어떻게 최적의 알고리즘과 자료구조로 표현 가능한지 고민해본다. 객체 간의 관계를 표현하는 그래프를 프로젝트에 적용하고, 만족도 또는 관광지 개수를 기준으로 하는 힙을 이용한 우선순위 큐, 다익스트라 알고리즘 등을 활용해 구현할 수 있는 능력을 배양한다.

② 개발 환경 및 팀 구성

ⓐ 개발 환경

Windows 10 / Intel Core i5

Visual Studio C++ / Eclipse Java

ⓑ 팀 구성

2015104197 이민혜 – 알고리즘 분석 및 구현

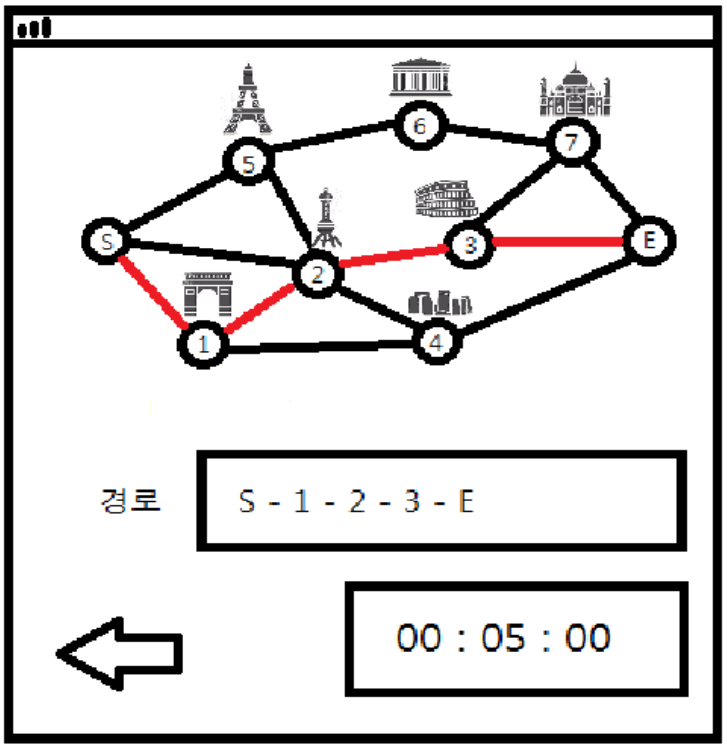
2015104220 정종윤 – GUI 구현 및 알고리즘 구현

③ 인터페이스와 자료 흐름도

ⓐ 사용자 인터페이스

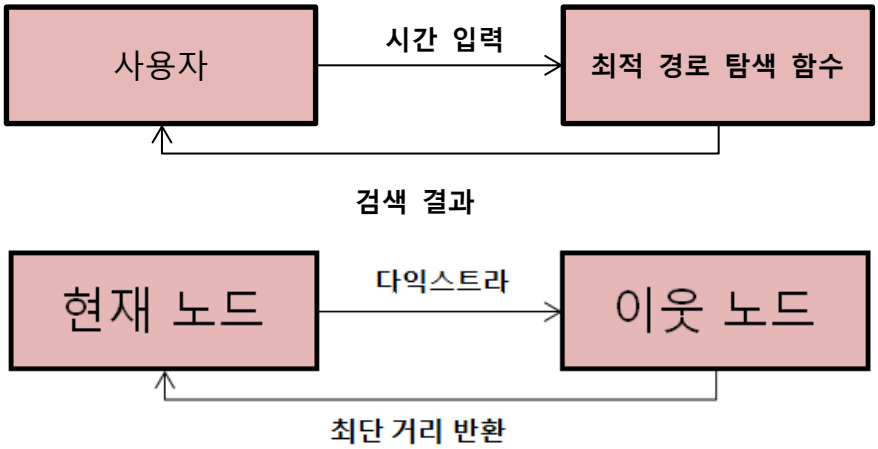
[그림1] 프로그램을 실행 시킨 메인 화면 모습, 프로그램 제목과 선택지(관광지 위주, 만``족도 위주, 프로그램 종료)를 출력하여 사용자의 응답을 대기한다.

[그림2] 조건1,2를 선택했을 때 나오는 화면. 사용자에게서 시간을 입력 받아 다음 화면으로 넘어가거나 다시 메인 화면으로 돌아갈 수 있다.



[그림3] 사용자의 선택과 입력을 바탕으로 하여 최적의 경로를 탐색하고 그 결과를 시각화된 지도와 텍스트를 통해 출력한다. 사용자는 이전화면으로 돌아가 시간을 수정하여 재 탐색이 가능하다.

ⓑ 자료 흐름도



④ 문제 해결 방법

ⓐ 왜 기존의 알고리즘으로 해결 불가능한가?

우선 우리는 문제에서 제시한 조건에 주목할 필요가 있다.

- **주어진 시간 내**에 가장 **많은 관광지**를 가거나 가장 **큰 만족도**를 얻을 수 있어야 하고
- **지역 중복 방문**은 허용되지만, **관광지 중복 방문**은 허용하지 않는다.

문제 해결을 위해 살펴보아야 할 중요한 변수로는 **주어진 시간 내**, **많은 관광지, 가장 큰 만족도, 지역 중복 방문,** 그리고 **관광지 중복 방문**이 있다. 우리는 이러한 변수들로 인해 일반적인 그래프에서 활용 가능한 알고리즘들을 이번 문제에 적용하기 어려운 이유를 살펴보고자 한다.

우선 **다익스트라 알고리즘**을 살펴보자. 다익스트라 알고리즘은 그래프에서 노드 사이의 최단 경로를 찾는 알고리즘이다. 일반적으로 가능한 적은 비용으로 가장 빠르게 해답에 도달하는 문제에 유용하게 쓰인다. 하지만 문제에서는 시간을 사용자에게서 입력 받고, 해당 시간 내에서 최적의 경로를 찾아야 한다. 다익스트라 알고리즘은 단순히 최단 경로의 정보만을 제공하므로, 문제에서 제공한 정보를 활용할 수 없다. 따라서 단순히 다익스트라 알고리즘만으로는 해답을 찾을 수 없다.

이어서 **크루스칼 알고리즘**과 **프림 알고리즘**을 살펴보자. 이 알고리즘들은 일반적으로 그래프의 모든 노드를 방문하는 최소 비용의 신장 트리를 찾기 위해 사용된다. 최소 비용의 신장 트리를 찾을 수 있긴 하지만, 주어진 시간 내에 최대 관광지 방문과 최대 만족

도 획득을 찾기 위한 경로 탐색 알고리즘으로 사용하기에는 부적절하다.

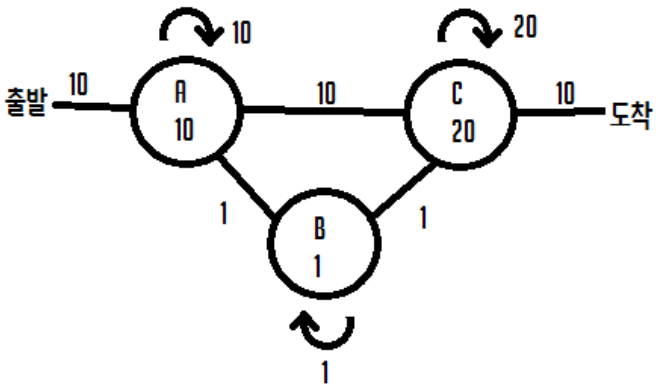
또 다른 경로 탐색 알고리즘으로는 **플로이드 알고리즘**이 있다. 플로이드 알고리즘은 그래프에서 모든 노드 사이의 최단 경로의 거리를 구하는 알고리즘이다. 또한 어떤 경유지를 지나서 최소 비용을 만들었는지 기록해두면, 최소 비용뿐만 아니라 최소 비용 경로까지도 구할 수 있다. 하지만 문제에서 주어진 그래프가 완전 그래프가 아니고 모든 노드 사이의 최단 경로를 구할 필요가 없기 때문에, **O(n^3)**의 복잡도를 가진 플로이드 알고리즘만을 활용하기에는 효율성이 떨어진다.

따라서 우리는 기존의 알고리즘들을 응용하고 발전시켜 새로운 알고리즘을 찾아야 한다.

ⓑ 해결 방안을 찾아보자

우선 주어진 시간 내에 가장 큰 만족도를 얻을 수 있는 경우에 대해 살펴보자. 물론 지역 중복 방문은 허용되지만, 관광지 중복 방문은 허용하지 않는다.

주어진 그래프는 **무방향 그래프(Undirected Graph)**이다. 여기에서 적용 가능한 방안은 **힙(Heap)**을 이용한 **우선순위 큐(Priority Queue)**이다. 주어진 시간 내에 만족도를 최대로 얻어야 하므로, 이웃 노드를 탐색하여 만족도가 높은 것을 먼저 방문하는 방법이다. 여기서 가정해야 할 점은, (1) 우선순위 큐가 1회 이동으로 획득할 수 있는 만족도를 Primary Key로 선택하고 (2) 이미 방문한 노드는 더 높은 만족도를 얻을 방법을 취하기 위하여 다시 우선순위 큐에 넣지 않는다는 점이다. **다익스트라 알고리즘(Dijkstra Algorithm)**과 일부 유사하다. 지역 재방문이 허용 되는데도 우선순위 큐에 넣지 않는 이유에 대해서는 아래에서 후술한다.



위 사진에서 둥근 노드는 Vertex를, 노드 안에 적힌 숫자는 관광지 방문 시 획득할 수 있

는 만족도를, 자기 자신으로 향하는 Edge는 관광지 방문 시 걸리는 시간을, Edge에 적힌 숫자는 이동하는데 걸리는 시간을 의미한다.

가정 (1)을 적용하기 위해서 지역에서 관광지를 방문하는데 걸리는 시간을 자기 자신으로 향하는 Edge로 취급한다는 점이다. 아래는 노드 A에 있을 때의 우선순위 큐(Priority Queue A)와 누적 만족도(Total Satisfaction)의 예시다. 여기서 임의로 분자는 1회 이동으로 얻을 수 있는 만족도를, 분모는 1회 이동 시 사용자가 위치하는 노드를 의미한다.

$$Priority\ Queue\ A = \left[\frac{10}{A}, \frac{0}{B}, \frac{0}{C}\right]$$
$$Total\ Satisfaction = 0$$

현재 A로 방문, 즉 자기 자신의 관광지를 방문하면 10의 만족도를 얻을 수 있기 때문에, 방문한다. 다음은 우선순위 큐에 따라 수행이 진행된 모습이다.

$$Priority\ Queue\ A = \left[\frac{0}{B}, \frac{0}{C}\right]$$
$$Total\ Satisfaction = 10$$

만족도가 누적되었고 우선순위 큐 A에는 2개의 선택지가 남아있다. 다음으로 선택할 수 있는 것으로는 노드 B로 가거나, 노드 C로 가는 길 밖에 없다. 우선 B로 이동해보자.

$$Priority\ Queue\ A = \left[\frac{0}{B}, \frac{0}{C}\right]$$
$$Priority\ Queue\ B = \left[\frac{1}{B}, \frac{0}{C}\right]$$
$$Total\ Satisfaction = 10$$

우선순위 큐 A에서 B로 가는 방법을 선택했기 때문에 B에서 우선순위 큐 B(Priority Queue B)를 새로 생성했다. 가정 (2)에 의해 이미 방문한 노드 A는 다시 큐에 삽입하지 않는다. 우선순위 큐 B의 첫 번째 항목인 B의 관광지 방문을 실행한다.

$$Priority\ Queue\ A = \left[\frac{0}{B}, \frac{0}{C}\right]$$
$$Priority\ Queue\ B = \left[\frac{0}{C}\right]$$
$$Total\ Satisfaction = 11$$

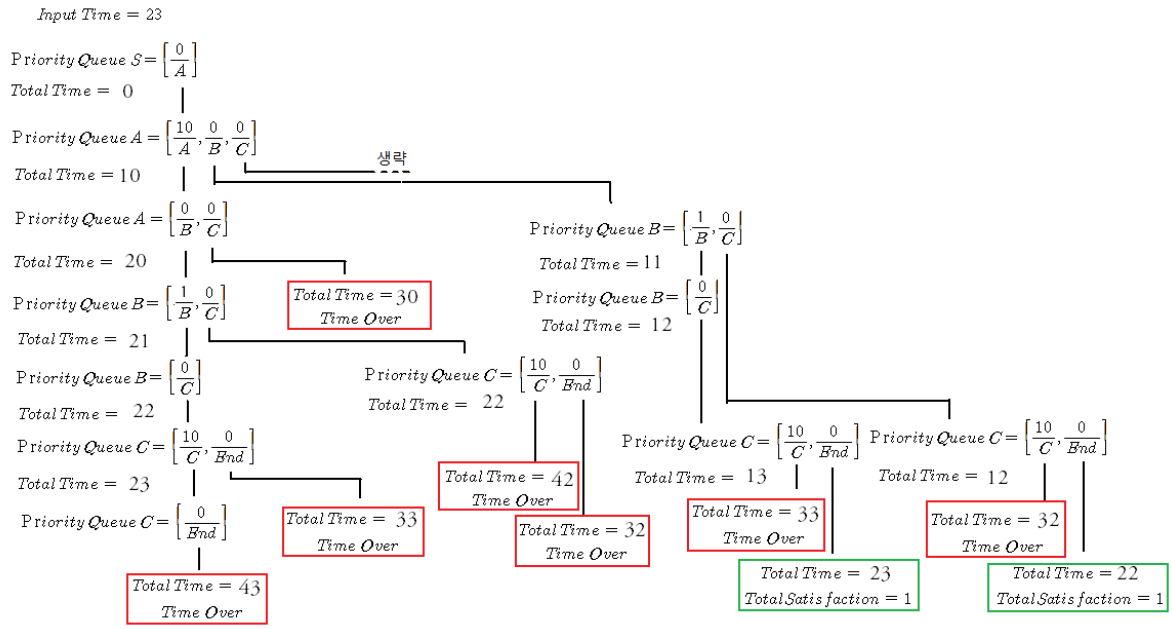
누적 만족도가 11이 되고 남은 경로는 C로 가는 것만 남았고, 같은 방법으로 계속 이어진다. 이제 이러한 방법을 주어진 문제에 최적화하여 적용하기 위해서는 다음과 같은 조건이 필요하다.

- 사용자의 입력 시간을 저장하는 변수를 1회 이동마다 갱신하여, 현재까지 이동

에 걸린 누적 시간이 사용자 입력 시간보다 클 경우에는 현재 열려있는 우선순위 큐 항목을 닫는다.

- 사용자의 입력 시간이 출발지에서 도착지까지 최단 경로(즉, 관광지를 아무것도 들리지 않고 가는 경우)로 이동하는 데 걸리는 시간보다 작다면, 계산 할 필요 없이 예외 처리한다.
- 사용자의 입력 시간 내에 이동 가능한 경로가 여러 개 생길 수 있기 때문에, 함수 실행 전에 변수를 선언하여 누적된 만족도가 가장 높은 것으로 계속 갱신한다.

만약 사용자 입력 시간이 23이고, 위에서 든 예시 그래프를 이러한 방법을 이용해 최적의 경로를 탐색하다 보면, 아래와 같은 과정과 결론이 나온다.

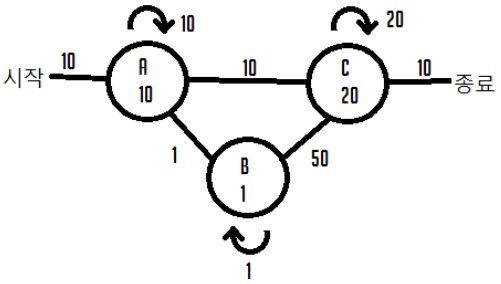


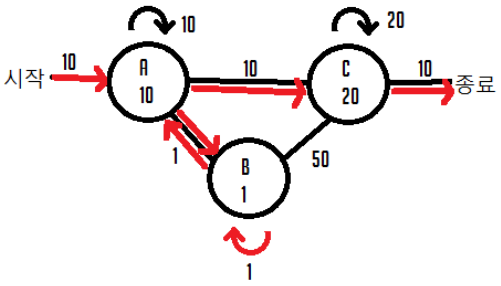
따라서 사용자 입력 시간이 23일 때, Start – A – B – Sights of B – C – End로 가는 경로가 최선이라고 볼 수 있다.

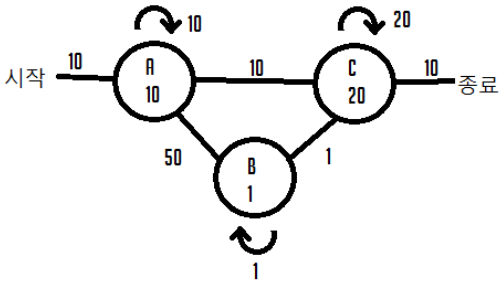
최대한 많은 관광지를 들리는 경우 역시 탐색 시간을 Primary Key로 활용하여 적용한다.

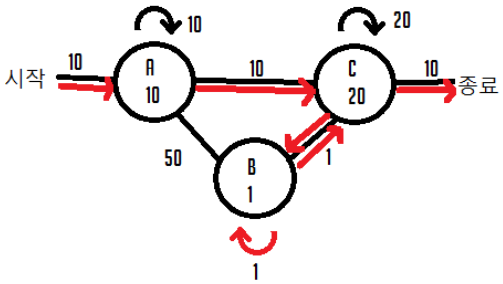
ⓒ 예외적인 경우의 탐색 방법

하지만 여기에서 지역 재방문의 경우는 고려되지 않았다. 아래와 같은 예시를 보자. 사진과 같은 그래프가 있을 때, 사용자 입력 시간이 33인 경우이다. 간단한 그래프이니만큼









직접 계산해보자면, 왼쪽에 그려진 각각의 그래프는 그 오른쪽에 있는 경로로 이동해야 만족도 1을 얻은 채 탐색이 끝날 수 있다. 따라서 위에서 언급한 방법만을 그대로 사용하면 한 번 들린 지역을 다시 방문하지 않기 때문에, 최적의 경로를 탐색할 수 없다.

즉, 오른쪽 아래 사진의 경우처럼 중복 방문 시 더 빠른 경로를 찾았는데(Start - A – C – B – Sight of B – C – End), 그렇다고 해서 우선순위 큐에 중복 방문을 허용하게 된다면, 불필요한 탐색이 많아진다(C – B – C – B – ……). 결국, 프로그램이 기억하고 있는 우선순위 큐에는 위에서 언급한 순서(Start - A – B – Sight of B – C – End)여야 하지만, 실제 이동은 그와 달라서 출력은 실제 이동 순서(Start – A – C – B – Sight of B – C – End)로 이루어져야 한다. 따라서 우리는 인접한 노드의 Edge를 그대로 사용해서는 안되며, 다른 노드를 거쳐가더라도 최적의 효율을 발생시킬 수 있는 경로를 찾아 사용해야 한다. 그러므로 우리는 중복 방문이 있을 경우를 대비해 출력용 큐를 따로 구현하여야 한다.

이때, 인접한 노드들을 방문할 때, 다른 노드를 거쳐 가는 것이 비용 면에서 이득이라면 그 노드로 가는 최단 거리를 구하기 위해 **다익스트라 알고리즘(Dijkstra Algorithm)**을 사용한다. 여기에서 다익스트라 알고리즘이 사용 가능한 이유는 노드에 포함된 만족도와 같은 가중치를 고려하지 않고, 순수히 출발지(현재 유저가 있는 노드)에서 도착지(인접 노드)까지 가장 짧은 경로만을 탐색하기 때문이다.

이때 모든 노드끼리의 최소 거리를 미리 구해놓고 사용하는 **플로이드 알고리즘(Floyd Algorithm)**을 사용할 수도 있다. 하지만 우리는 인접 노드와의 최소 거리만을 사용할 것이기 때문에, 각 노드에 인접 노드의 정보를 담고 거리가 필요할 때 마다 탐색하기로 한

다.

ⓓ 활용하기 위한 자료구조/알고리즘

→ 무방향 그래프(Undirected Graph)

기본적으로 제공되는 데이터는 노드와 노드 사이의 Edge에 방향성이 없고 이동이 자유롭다. 따라서 단순 연결선을 갖는 무방향 그래프를 사용할 것이다.

→ 힙(Heap)과 힙을 이용한 우선순위 큐(Priority Queue)

우선순위 큐를 구성하는 데에는 배열을 기반으로 구현하는 방법, 연결리스트를 기반으로 구현하는 방법, 힙(Heap)을 이용하는 방법이 있다. 그 중 가장 효율이 좋은 힙을 사용할 계획이다.

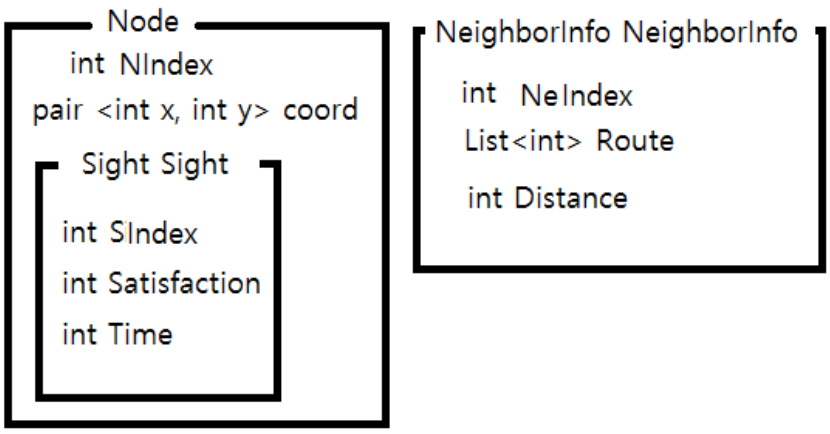
표현방법	삽입	삭제
비정렬 배열	O(1)	O(N)
비정렬 연결리스트	O(1)	O(N)
Heap	O(logN)	O(logN)

[표1] 자료구조 별 삽입, 삭제 시간 복잡도 및 효율 비교

→ 다익스트라 알고리즘(Dijkstra Algorithm)

시간 복잡도는 최악의 시간 복잡도가 나오는 경우는 모든 꼭지점이 연결된 완전 그래프일 때 $O(V^2)$의 시간 복잡도가 나온다. 하지만 주어진 그래프는 완전 그래프가 아니고, 힙을 사용하기 때문에 시간복잡도는 $O((V+E)\log V)$에 가깝게 된다.

→ 노드 아이템(Node Item)



ⓔ 수도코드

```
Function(방문 여부 확인 배열, 현재 위치, 선택해온 노드 경로){
If(도착){
Done
Max값 갱신, return}
else{
        현재 위치와 선택 노드 경로 가중치 계산 후 시간 초과 여부 확인
        If(시간초과)
                return
        else{
                선택 경로에 현재 위치 추가
                현재 위치에서 유망 노드 우선순위 큐 생성
                For(우선순위 큐)
                        Function(방문 여부 배열, 우선 순위 큐 아이템, 선택 경로)
        }
}
```

⑤ 참고 자료

-설계 프로젝트C 수업 자료

-조중필, 한현상, 이주호(2016). 알고리즘 문제 풀이 전략. 한빛 미디어

-RICHARD NEAPOLITAN(2004), 알고리즘 FOUNDATIONS OF ALGORITHMS USING C++ PSEUDOCODE. 사이텍미디어