<A-STAR 알고리즘>

*기본형 A-STAR 알고리즘

A-STAR 알고리즘은 시작 노드만을 지정해 다른 모든 노드에 대한 최단 경로를 파악하는 다익스트라 알고리즘과 다르게 **시작 노드와 목적지 노드를 분명하게 지정해 이 두 노드 간의 최단 경로를 파악할 수 있는 최적경로 탐색 알고리즘으로 휴리스틱 추정값을 통해 알고리즘을 개선할 수 있다.** 이러한 휴리스틱 추정값을 어떤 방식으로 제공하느냐에 따라 얼마나 빨리 최단 경로를 파악할 수 있느냐가 결정되며, 일반적인 A-SATR 알고리즘의 link 별 가중치는 거리에 의해 결정되나, 새롭게 수정된 A-STAR 알고리즘(교통약자 전용 테마길 추천 알고리즘)은 부산시 특정 관내 도로 link 별로 설정한 안전비용(cost)을 통해 이를 구현할 수 있고, 안전비용은 실도로조사를 통해 수집한 불량 노면 도로항목에 대해 AHP/TOPSIS 분석을 이용하여 산출한다.

*수정된 A-STAR 최적경로탐색기법 제안

새롭게 수정되어 제안하는 최적경로탐색기법은 AHP 와 TOPSIS 를 응용하여 A-STAR 알고리즘의 비용 산정에서 주행도로의 안전성을 정량화하고 출발지에서 목적지까지의 최단비용을 발생시키는 경로를 최적경로로 탐색하는 방법이다. 이때 비용 산정은 AHP/TOPSIS 분석을 통해 도로안전등급을 순위화하고 0~1 로 정규화과정을 거친 후 각도로 link 에 도로안전등급을 할당한다. 이후 A-STAR 알고리즘의 비용을 0~1 로 정규화된 안전등급으로 정의되면서 수정 A-SATR 최적경로탐색 알고리즘이 완성된다.

참고: https://koreascience.kr/article/JAKO202106153137631.page

<다익스트라 알고리즘 (Dijkstra algorithm)>

출발지로부터 인접한 노드를 하나하나 방문하며 각 노드로 가는 최단거리를 갱신한다.

```
if(dist[Spot.end] > dist[cur] + Spot.cost)
   dist[Spot.end] = dist[cur] + Spot.cost;
```

위 코드는 dist[v] = min(현재까지의 최단거리 비용, 새로운 노드를 거치는 거리비용)을 뜻한다.

아래는 코드 적용 일부분이다.

```
import heapq  # 우선순위 큐 구현을 위함

def dijkstra(graph, start):
    distances = {node: int(1e9) for node in graph}  # 처음 초기값은 무한대
    distances[start] = 0  # 시작 노드까지의 거리는 0
    queue = []
    heapq.heappush(queue, [distances[start], start])  # 시작 노드부터 탐색
시작

    while queue:  # queue 에 남아있는 노드가 없을 때까지 탐색
        dist, node = heapq.heappop(queue)  # 탐색할 노드, 거리

        # 기존 최단거리보다 멀다면 무시
```

```
        if distances[node] < dist:
            continue

        # 노드와 연결된 인접노드 탐색
        for next_node, next_dist in graph[node].items():
            distance = dist + next_dist  # 인접노드까지의 거리
            if distance < distances[next_node]:  # 기존 거리 보다 짧으면 갱신
                distances[next_node] = distance
                heapq.heappush(queue, [distance, next_node])  # 다음 인접
거리를 계산 하기 위해 큐에 삽입
    return distances
```

[기타]

이동제약자를 위한 최적 경로 탐색 앱 https://github.com/gaeunpark924/Comprehensive-Design-2

카카오 API 적용 https://apis.map.kakao.com/web/guide/