알고리즘 보고서: 다익스트라와 A* 알고리즘 및 기타 알고리즘

20기 인턴 우정윤

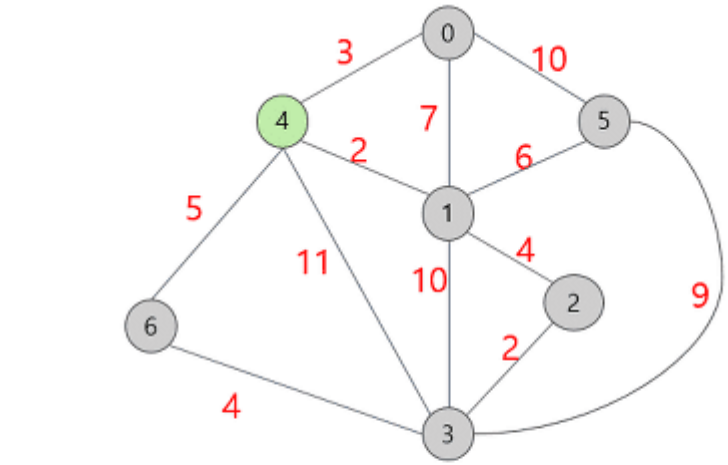
1. 개요

탐색 및 최소거리 탐색은 네비게이션, 로봇경로, 게임 ai등 다양한 분야에서 다루어지고 있는 핵심적인 문제이다. 주어진 그래프에서 최단 경로를 찾는 것은 전형적인 문제이며, 대표적인 해법으로 다익스트라 알고리즘과 A* 알고리즘이 있다.

2. 알고리즘 설명

2-1. 다익스트라 알고리즘

다익스트라 알고리즘은 네덜란드의 컴퓨터 과학자 에츠허르 다익스트라(Edsger W. Dijkstra)가 제안한 최단 경로 탐색 알고리즘으로, 가중치가 있는 그래프에서 출발 노드로부터 다른 모든 노드까지의 최단 거리를 구하는 방법이다. 이 알고리즘은 음의 가중치를 허용하지 않으며, 각 경로의 비용이 0 이상이라는 조건하에서 정확한 결과를 보장한다.



S = { 0, 4 }

distance[ ]=

0	1	2	3	4	5	6
0	5	∞	14	3	10	8

작동 원리를 살펴보면, 우선 출발 노드의 거리를 0으로, 나머지 노드들의 거리를 무한대(∞)로 설정한다. 그 다음, 아직 방문하지 않은 노드 중에서 가장 짧은 거리를 가진 노드를 선택하고, 그 노드를 기준으로 이웃한 노드들의 거리를 갱신한다. 즉, 선택한 노드를 거쳐서 다른 노드로 가는 경로가 기존보다 더 짧다면 그 값을 갱신하는 방식이다. 이 과정을 모든 노드가 방문될 때까지 반복하면, 출발지에서 모든 노드까지의 최단 경로가 완성된다.

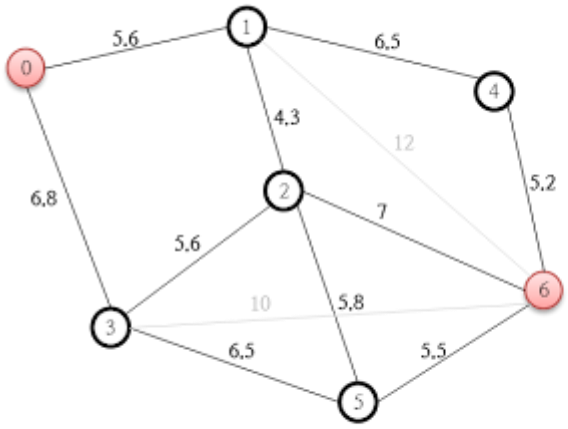
예를 들어, 도시와 도시를 잇는 도로망을 그래프로 표현했을 때, 다익스트라 알고리즘을 사용하면 특정 도시에서 출발하여 모든 다른 도시에 이르는 최소 비용 경로를 한 번의 실행으로 얻을 수 있다. 따라서 내비게이션 시스템, 네트워크 라우팅, 물류 경로 최적화 등 다양한 분야에서 활용된다.

다익스트라 알고리즘의 효율성은 자료구조 선택에 따라 달라진다. 단순한 배열을 사용하면 시간 복잡도는 $O(V^2)$이지만, 우선순위 큐(힙)를 사용하면 $O(E + V \log V)$로 줄일 수 있다. 이 때문에 실제 응용에서는 보통 힙 기반의 우선순위 큐를 결합하여 효율적으로 구현한다.

정리하면, 다익스트라 알고리즘은 출발점으로부터의 거리를 단계적으로 확정해 나가면서 최단 경로를 찾는 탐욕적(greedy) 접근 방식을 사용하는 알고리즘이며, 단일 출발점-모든 도착점 최단 거리 문제를 해결하는 대표적인 방법이다.

2-2. A* 알고리즘

A* 알고리즘은 최단 경로 탐색 알고리즘 중 하나로, 다익스트라 알고리즘을 확장한 형태라고 볼 수 있다. 다익스트라가 출발 노드에서 현재까지의 누적 비용만 고려한다면, A*는 여기에 **목표까지의 예상 비용(휴리스틱)**을 더하여 탐색을 진행한다. 이 때문에 A*는 목표 지점에 도달하기까지 불필요한 경로를 줄이고, 보다 효율적으로 탐색할 수 있다는 장점이 있다.



O =

Node ID	1	3
F Score	17.6	16.8
G Score	5.6	6.8
H Score	12	10
Parent Node	0	0

C =

Node ID	0
F Score	0
G Score	0
H Score	0
Parent Node	-

알고리즘은 크게 두 가지 비용을 합산해 경로를 평가한다. 하나는 **g(n)**이라 불리는 시작점에서 현재 노드까지의 실제 비용이고, 다른 하나는 **h(n)**이라 불리는 현재 노드에서 목표 노드까지의 추정 비용이다. 이 둘을 합한 값 **f(n) = g(n) + h(n)**을 기준으로 탐색을 진행한다. 여기서 h(n)은 문제에 맞는 휴리스틱 함수를 사용하는데, 격자 지도에서는 보통 맨해튼 거리(직선 거리를 가로·세로 단위로 합산한 값)나 유클리드 거리(두 점 사이의 직선 거리)가 자주 쓰인다.

작동 과정은 다음과 같다. 우선 시작 노드를 오픈 리스트(open set)에 넣고, f 값이 가장 작은 노드를 선택해 탐색을 진행한다. 선택된 노드의 인접 노드에 대해 g, h, f 값을 계산하고, 더 짧은 경로가 발견되면 갱신한다. 목표 노드가 선택되면 탐색을 종료하고, 이전 노드 정보를 따라가면서 최단 경로를 재구성한다.

A* 알고리즘은 휴리스틱 함수를 어떻게 정의하느냐에 따라 성능이 크게 달라진다. 만약 h(n)이 실제 비용을 과소평가한다면, 탐색은 최적의 해를 보장하면서도 다익스트라보다 훨씬 빠르게 동작한다. 그러나 h(n)이 과대평가되면 탐색이 잘못된 방향으로 치우쳐 최적 경로를 보장하지 못할 수 있다. 따라서 문제 상황에 맞는 적절한 휴리스틱을 설계하는 것이 A* 알고리즘의 핵심이다.

실제 응용으로는 게임 AI에서 캐릭터 이동 경로 탐색, 로봇 자율주행의 경로 계획, 네비게이션 시스템의 목적지 경로 탐색 등이 있다. 특히 목표 지점이 명확하고, 탐색 공간이 넓은 경우에 다익스트라보다 훨씬 효율적이다.

정리하자면, A* 알고리즘은 다익스트라의 안정성과 휴리스틱 기반 탐색의 효율성을 결합한 방식으로, "최단 경로를 빠르고 똑똑하게 찾는 알고리즘"이라고 할 수 있다.

3. 시간 및 공간 복잡도 분석

다익스트라 알고리즘의 시간 복잡도는 사용되는 자료구조에 따라 달라진다. 단순한 배열을 사용하면 각 단계에서 최소 거리를 가지는 노드를 찾는 데 $O(V)$ 시간이 걸리므로 전체적으로 $O(V^2)$ 시간이 소요된다. 그러나 우선순위 큐(특히 이진 힙)를 사용하면, 최

소 거리를 가지는 노드를 O(log V)에 선택할 수 있고, 간선의 완화(relaxation) 과정도 효율적으로 처리되어 최종적으로 O(E + V log V)의 시간 복잡도를 가진다. 여기서 V는 노드의 개수, E는 간선의 개수이다. 따라서 희소 그래프(sparse graph)에서는 배열 기반보다 우선순위 큐 기반 구현이 훨씬 효율적이다. 공간 복잡도는 각 노드와 간선 정보를 저장하는 데 O(V + E), 거리 배열과 방문 여부 배열이 O(V)이므로 전체적으로 O(V + E)에 해당한다.

A* 알고리즘의 경우도 다익스트라와 비슷한 방식으로 우선순위 큐를 사용하기 때문에 최악의 경우 시간 복잡도는 O(E + V log V)로 동일하다. 그러나 실제 실행 시에는 휴리스틱 함수 덕분에 탐색 범위가 크게 줄어들어 평균적으로 훨씬 더 적은 노드를 탐색한다. 즉, 최악의 경우 복잡도는 다익스트라와 같지만, 실제 성능은 휴리스틱의 품질에 따라 더 좋아질 수 있다. 공간 복잡도 역시 다익스트라와 동일하게 O(V + E)이지만, 오픈 리스트와 클로즈드 리스트 관리가 추가되므로 실제 구현에서는 메모리 사용량이 조금 더 많아질 수 있다.

정리하면, 이론적으로 두 알고리즘의 복잡도는 동일하지만, 실제로는 A*가 목표 탐색 문제에서 훨씬 효율적이며, 다익스트라는 모든 노드까지의 거리를 계산해야 하는 경우에 강점을 가진다.

4. 장점과 한계

다익스트라 알고리즘은 안정성과 보편성이 가장 큰 장점이다. 음의 가중치만 없다면 어떤 그래프에서도 항상 최단 경로를 보장하며, 특정 목표 노드뿐 아니라 모든 노드까지의 최단 경로를 한 번에 구할 수 있다. 따라서 전체 네트워크의 거리 정보를 얻어야 하는 문제(예: 지도 서비스에서 특정 지점까지의 거리 행렬 계산, 네트워크 라우팅 최적화)에서는 매우 유용하다. 그러나 목표 지점이 하나로 정해져 있을 때도 불필요하게 모든 노드를 탐색하기 때문에, 계산량이 많아지는 대규모 그래프에서는 비효율적이라는 한계가 있다.

A* 알고리즘은 다익스트라의 단점을 보완한 방식으로, 탐색 효율성이 가장 큰 장점이다. 휴리스틱 함수를 통해 목표 방향으로 탐색을 유도하기 때문에, 실제로는 탐색해야 하는 노드 수가 크게 줄어든다. 특히 맵 기반의 경로 탐색(게임 캐릭터 이동, 로봇 자율주행 등)에서는 목표가 명확하므로 다익스트라보다 훨씬 빠르게 동작한다. 하지만 A*의 성능은 휴리스틱 함수에 크게 의존한다. 휴리스틱이 실제 비용을 과소평가하면 최적성을 보장하지만, 과대평가하면 탐색이 잘못된 방향으로 치우쳐 최단 경로를 찾지 못할 수 있다. 즉, A*는 휴리스틱 설계가 적절해야만 장점을 제대로 살릴 수 있다는 점에서 제약이 따른다.

결국 두 알고리즘은 상호 보완적인 성격을 가진다. 모든 노드에

대한 최단 거리 정보가 필요하다면 다익스트라가 더 적합하고, 특정 목표 노드까지 빠르게 도달하는 것이 목적이라면 A*가 더 효율적이다.

5. 기타 알고리즘

5-1. 벨만 포드 알고리즘

벨만 포드 알고리즘은 음수 가중치가 포함된 그래프에서도 최단 경로를 찾을 수 있는 알고리즘이다. 다익스트라 알고리즘은 간선 가중치가 모두 0이상일때만 사용할 수 있지만, 벨만-포드는 음수 간선이 있어도 동작한다는 점에서 차별성이 있다. 다만 음수 사이클(계속 돌수록 거리가 무한히 줄어드는 사이클)이 존재하면 최단 경로가 정의되지 않으므로, 이를 탐지하는 기능도 함께 제공한다.

벨만-포드 알고리즘은 완화라는 과정을 반복하는데, 특정 간선을 통해 더 짧은 경로를 찾을 수 있으면 갱신하는 것을 완화라고 한다. 시작 정점의 거리를 0으로 두고, 나머지 모든 정점의 거리를 무한대로 설정한다. 그리고 그래프에 정점이 v개가 있다면, 최단 경로는 최대 v-1개의 간선을 거친다. 따라서 모든 간선에 대해 v-1번 반복하면서 완화를 수행한다.

5-2. 플로이드-워셜 알고리즘

플로이드-워셜은 동적 계획법을 기반으로 동작하는데, 어떤 정

점 i에서 정점 j로 가는 최단 경로를 찾을 때, 중간에 거쳐갈 수 있는 정점들을 하나씩 고려하면서 경로를 갱신한다. 플로이드-워셜 알고리즘은 인접 행렬 형태로 거리 배열을 만들고 간선이 있으면 가중치로 초기화, 없으면 무한대로 설정하고 자기 자신은 0으로 둔다. 모든 정점 k를 1개씩 경유 노드로 고려하고 각 k에 대해 배열을 갱신한다. 반복을 종료하면 i에서 j까지의 최단 경로 길이가 저장되게 된다.

6. 결론 및 응용

다익스트라와 A* 알고리즘은 모두 최단 경로 탐색 문제를 해결하는 핵심 알고리즘으로, 그래프 이론과 인공지능 분야에서 매우 중요한 위치를 차지한다. 다익스트라 알고리즘은 모든 노드까지의 최단 거리를 구하는 데 강점을 가지며, 안정적이고 보편적으로 적용 가능하다. 반면, A* 알고리즘은 휴리스틱을 결합하여 목표 지점 탐색을 훨씬 효율적으로 수행할 수 있으며, 특히 탐색 공간이 큰 문제에서 시간 절약 효과가 두드러진다.

이 두 알고리즘은 현실 세계의 다양한 분야에서 응용된다. 예를 들어, 교통 네비게이션 시스템에서는 다익스트라를 활용해 전체 거리 정보를 계산하거나, A*를 활용해 특정 목적지까지 최적 경로를 빠르게 찾아낼 수 있다. 게임 AI에서는 캐릭터가 장애물을 피하면서 목적지로 이동하는 경로를 A*로 구하는 것이 일반적이다. 또한 로봇 자율주행 및 물류 로봇 분야에서도 경로 계획 알고리

즘으로 다익스트라와 A*가 널리 사용된다. 네트워크 라우팅 최적화, 지도 서비스, 물류 경로 설계 등도 대표적인 응용 사례다.

결론적으로, 다익스트라와 A*는 상호 보완적인 관계에 있으며, 문제의 성격과 목표에 따라 선택적으로 사용된다. 전체 그래프에 대한 전역적인 거리 정보가 필요할 때는 다익스트라가, 특정 목표 지점에 빠르게 도달하는 것이 중요할 때는 A*가 더 적합하다. 앞으로는 GPU 병렬화, 강화학습 기반 휴리스틱 개선 등과 결합하여 두 알고리즘의 효율성과 활용 범위가 더욱 확장될 것으로 기대된다.