**== REPORT ==**

**< 최단 경로 탐색 팀 프로젝트 >**

3가지 알고리즘을 이용한 지하철 최단 경로 탐색

|  |  |
| --- | --- |
|  |  |
| 과 목 | 자료구조 및 알고리즘 1 |
| 담당 교수 | 박 OO |
| 제출 일자 | 2024.05.19 |
| 이름 학번 | 김OO 21OOOOOO |
| 이름 학번 | 강OO 21OOOOOO |
| 이름 학번 | 권도윤, 21102345 |
| 김OO | Floyd-warshall 구현 |
| 강OO | Dijkstra 구현 |
| 권OO | SPFA 구현 |

**Ⅰ. 서론**

본 프로젝트는 지하철 노선도에서 최단 거리를 구하는 방법을 찾아 구현한다. 최단거리를 찾는 방법은 여러가지 임으로 본 팀은 최적의 경로를 찾는 것은 물론 여러 방법에 따른 소요 시간 또한 중요한 요소라고 판단해 3가지 알고리즘을 구현하고 그 결과를 비교, 분석했다. 구현한 알고리즘에는 Dijkstra, SPFA, Floyd-Warshall 이렇게 세가지가 있다.

**Ⅱ. 알고리즘**

**1. Dijkstra**

**1.1 알고리즘 소개**

Dijkstra 알고리즘은 가장 대중적인 그래프 탐색 알고리즘으로, 가중치가 있는 그래프를 이용하여 특정 시작 노드에서 다른 모든 노드까지의 최단 경로를 찾는다. 현재까지 알고 있던 최단 경로를 계속해서 갱신하면서 찾아가는 방식을 사용한다. 주로 네트워크 라우팅, 길 찾기, 지도 애플리케이션에 사용된다.

**1.2 알고리즘 특징**

1. 가중치가 있는 그래프이다. 모든 간선은 양의 가중치를 가져야 한다.

2. 특정 출발점에서 다른 모든 노드까지 최단 경로로 방문하며 각 노드 까지의 최단 경로를 모두 찾는다.

3. 현재까지의 최단 경로를 확장하여 전체 최단 경로를 정점 집합에 추가한다.

**1.3 동작 방식**

1.출발 노드, 도착 노드를 설정한다.

2.출발 노드 기준으로 각 노드의 최소 비용을 저장한다.

3.현재 위치한 노드의 인접 노드 중 방문하지 않은 노드를 구별하고, 방문하지 않는 노드 중에서 가장 비용이 적은 노드 선택한다. 그 노드를 방문 처리한다.

4.해당 노드를 거쳐서 특정한 노드로 가는 비용을 고려하여 최소 비용을 갱신한다.

5. 3~4번을 반복한다.

**1.4 시간 복잡도**

Dijkstra 알고리즘의 시간 복잡도는 방문하지 않은 노드 중 비용이 가장 작은 노드를 선택하는 방식에서 차이가 난다. 순차 탐색에서 노드의 개수가 n이라고 할 때 각 노드마다 최소 거리값을 갖는 노드를 선택해야 하는 순차 탐색이 수행되므로 (n-1)\*n=O(n^2)의 시간 복잡도를 가진다. 순차 탐색을 사용할 경우 노드 개수에 따라 탐색 시간이 매우 오래 걸릴 수 있다. 이를 개선하기 위해 우선순위 큐를 도입한다.

우선순위 큐를 최소 힙으로 구현하면 루트 노드가 최소 거리를 갖는 노드가 된다. 우선 순위 큐에서 사용할 우선순위의 기준은 시작 노드로부터 가장 가까운 거리가 된다. 따라서 큐의 정렬은 최단 거리인 노드를 기준으로, 그 다음 최단 거리를 가지는 노드를 앞에 배치한다. 만약 기존 최단 거리보다 더 작은 값을 갖는 노드가 있다면 그 노드와 거리를 우선 순위 큐에 넣는다. 이때 edge의 수를 E, 노드의 수를 V라고 했을 때 O(ElogV)의 시간 복잡도를 가진다.

우선순위 큐에서 꺼낸 노드는 연결된 노드만 탐색하므로 최악의 경우라도 총 edge 수인 E만큼만 반복한다. 즉 하나의 간선에 대해서는 O(loge)이고, E는 V^2보다 항상 작기 때문에 E개의 edge를 우선 순위 큐에 넣었다 빼는 최악의 경우에 대해서는 O(ElogV)이다.

**1.5 공간 복잡도**

공간 복잡도는 nx.Graph로 그래프 데이터 구조를 사용하여서 노드가 차지하는 공간 O(V)와 간선이 차지하는 공간 O(E)를 모두 고려해야 한다. 따라서 그래프 자체를 저장하기 위한 공간 복잡도는 O(V+E)이다

**2. SPFA**

**2.1 알고리즘 소개**

SPFA 알고리즘은 최단 경로 문제를 해결하기 위해 Bellman-Ford 알고리즘을 최적화한 방법으로 큐를 사용하여 거리가 갱신된 노드만 다시 처리함으로써, 불필요한 연산을 줄이고 효율성을 높인 알고리즘이다.

\*Bellman-Ford 알고리즘: 그래프 내의 음수 가중치를 가진 간선이 존재하는 경우에도 최단 경로를 찾을 수 있는 알고리즘

**2.2 알고리즘 특징**

1. 가중치가 있는 그래프이다.

2. 이전에 큐에 추가된 노드보다 거리가 짧게 노드의 거리가 갱신될 때마다 해당 노드를 큐에 추가하여 불필요한 연산 감소를 통해 실행 시간을 최적화한다.

**2.3 동작 방식**

1. 초기화 모든 노드의 거리를 무한대로 초기화하고, 시작 노드의 거리를 0으로 설정한다. 시작 노드를 큐에 추가하고, 노드가 큐에 있는지 여부를 추적한다.

2. 큐 기반 탐색

2.1 큐에서 노드를 하나 꺼내고, 현재 노드의 모든 인접 노드를 검사한다.

2.2 현재 노드에서 인접 노드까지의 거리를 계산하고, 계산된 거리가 기존의 거리보다 짧으면 갱신한다.

2.3 갱신된 인접 노드를 큐에 추가하고, 큐에 있음을 표시한다.

3. 큐가 빌 때까지 위 과정을 반복한다.

**2.4 시간 복잡도**

본 프로젝트에 사용된 지하철 노선의 경우 선형구간이 환승역보다 훨씬 많다. 대부분의 노드가 선형 구간에 위치해 있기 때문에, 큐에 추가되는 횟수와 간선의 처리는 선형 구간에서 주로 발생한다. 선형 구간에서는 방향이 두개이고 한방향으로 이동할 때 각 노드가 한 번만 큐에 추가, 각 간선이 한 번만 처리되므로, 시간 복잡도는 O(V + E)이다. 환승역에서는 큐에 노드가 추가될 기회가 더 많아져 시간 복잡도가 O(VE)로 늘어날 수 있지만 그 수는 선형구간에 비해 많지 않다. 따라서 최종 시간 복잡도는 O(V + E)이다.

**2.5 공간 복잡도**

Graph 구조를 사용하여 각 노드와 엣지를 저장함으로 공간 복잡도는 O(V+E) 이다.

**3. Floyd-Warshall**

**3.1 알고리즘 소개**

Floyd warshall 알고리즘은 간선 가중치 방향 그래프의 모든 정점 쌍 사이의 최단 경로를 계산하는 알고리즘이다. 또한 음의 순환이 있는지 감지하는 데에도 사용할 수 있다.

**3.2 알고리즘 특징**

1. 모든 노드 쌍에 대한 경로를 탐색한다.

2. 음수 가중치와 음수 사이클을 포함한 그래프에도 적용할 수 있다.

**2.3 동작 방식**

1. 초기화: 노드의 거리를 무한대로, 시작 노드의 거리를 0으로 초기화한다.

2. 최단 경로 경신: 모든 노드를 경유점으로 하여 최단 경로 계산

이때, 현재까지 계산한 최단 경로 활용

3. 반복: 최단 경로 경신을 반복 시행하면서 모든 노드 간 최단 거리 행렬 완성

**2.4 시간 복잡도**

이 알고리즘에서는 최단 경로 행렬을 만들기 위해 3중 for문을 사용한다. 그러므로 시간 복잡도는 O(V^3)이다.

**2.4 공간 복잡도**

공간 복잡도는, 최단 경로 행렬의 크기가 그래프의 인접 행렬 크기와 같으므로, O(V^2)이다.

**Ⅲ. 알고리즘 시간, 공간 복잡도 비교**

<표1>은 각 알고리즘 별 시간 복잡도와 공간 복잡도를 정리한 것이다. 시간 복잡도 측면에서 보았을 때 SPFA가 가장 빠르고 다음이 Dijkstra이며 Floyd-Warshall이 가장 느린 것을 알 수 있다. 이러한 결과는 직선구간의 수가 환승역보다 더 많은 지하철 노선에서 최단거리를 구하는 코드를 구현했기 때문에 나타난다. 공간 복잡도는 Dijkstra와 SPFA가 O(V+E)로 같으며 Floyd-Warshall이 O(V^2)으로 가장 큰 공간 복잡도를 가지는 것을 알 수 있다. <표2>는 각 알고리즘 별 실제 소요 시간과 메모리 사용량을 나타낸 표이다. 실제 실행시간은 시간 복잡도를 비교한 결과와 같이 SPFA, Dijkstra, Floyd-Warshall 순서로 빠르다. 메모리 사용량 또한 공간 복잡도를 비교한 결과와 동일한 경향을 보여주었다.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Dijkstra** | **SPFA** | **Floyd-Warshall** |
| 시간 복잡도 | O(E log V) | O(V+E) | O(V^3) |
| 공간 복잡도 | O(V+E) | O(V+E) | O(V^2) |

<표1> 알고리즘별 시, 공간 복잡도

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Dijkstra** | **SPFA** | **Floyd-Warshall** |
| 시간(초) | 0.034055 | 0.027629 | 8.564762 |
| 메모리(kb) | 228.18 | 213.63 | 1421.12 |

<표2> 알고리즘별 소요 시간 및 사용 메모리

**IV. 결론**

본 프로젝트에서 세가지의 최단경로를 찾아주는 알고리즘을 구현하고 실제 실행 시간과 메모리 사용량을 비교했다. 메모리의 사용량은 공간 복잡도에 따른 순서를 보여주었다. 소요 시간은 SPFA알고리즘이 가장 빠르게 동작했고 그 다음 Dijkstra, Floyd-Warshall 인 것을 확인할 수 있다. 이러한 결과는 지하철 노선이 가지는 특징 때문에 나타난 것을 알 수 있다.

**Ⅳ. 참고문헌**

Noto, Masato, and Hiroaki Sato. "A method for the shortest path search by extended Dijkstra algorithm." *Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics.'cybernetics evolving to systems, humans, organizations, and their complex interactions'(cat. no. 0*. Vol. 3. IEEE, 2000.

Chen, Hao, and Hee-Jong Suh. "An improved Bellman-Ford algorithm based on SPFA." *The Journal of the Korea institute of electronic communication sciences* 7.4 (2012): 721-726.

Hougardy, Stefan. "The Floyd–Warshall algorithm on graphs with negative cycles." *Information Processing Letters* 110.8-9 (2010): 279-281.