개요

그래프 이론에 최단 경로 문제란 가장 짧은 경로에서 두 꼭짓점을 찾는 문제로서, 가중 그래프에서는 구성하는 변들의 가중치 합이 최소가 되도록 하는 경로를 찾는 문제이다. 예를 들면, 도로 지도 상의 한 지점에서 다른 지점으로 갈 때 가장 빠른 길을 찾는 것과 비슷한 문제이다. 이 때, 각 도로 구간에서 걸리는 시간을 변의 가중치라고 할 수 있다.

보통은 주어진 가중 그래프에서 $\sum_{p\in P} f(p)$가 v 에서 v'로 가는 모든 경로들 중 최소가 되도록 하는 경로를 찾는 문제이다. 이런 문제는 단일-쌍 최단 경로 문제라고 부르며, 아래의 일반화된 문제들과는 차이가 있다.

- 단일-출발 최단 경로 문제: 단일 꼭짓점 v 에서 출발하여 그래프 내의모든 다른 꼭짓점들에 도착하는 가장 짧은 경로를 찾는 문제이다.
- 단일-도착 최단 경로 문제: 모든 꼭짓점들로부터 출발하여 그래프 내의 한 단일 꼭짓점 v 로 도착하는 가장 짧은 경로를 찾는 문제이다. 이 문제에서 그래프 내의 꼭짓점들을 거꾸로 뒤집으면 출발 최단 경로 문제로 바뀔 수 있다.
- 전체-쌍 최단 경로 문제: 그래프 내의 모든 꼭짓점 쌍들 사이의 최단 경로를 찾는 문제이다.

위의 일반화된 문제들은, 전체-쌍 중 단일-쌍만으로 찾아가는 단순 접근 방식보다, 확실히 더 효율적인 알고리즘을 가진다. 이번 실험에서는 단일-도착 최단 경로 문제를 Floyd 알고리즘을 사용하여 풀어보고자 한다.

1. path() 함수의 단계적 해석

주어진 그래프의 가중치를 저장한 행렬을 W, 최단거리의 가중치 합을 저장한 행렬을 D, 시작점-끝점을 제외한 최단거리 경로 중 최고 차수를 저장한 행렬을 P라고 하자 (경로가 없을 때 = Infinity = 999). 행렬 W, D, P와 path의 의사 코드는 다음과 같다.

```
Matrix: W
  [  0]   [ 10]   [  5]   [999]   [999]
  [999]   [  0]   [  2]   [  1]   [999]
  [999]   [  3]   [  0]   [  9]   [  2]
  [999]   [999]   [  0]   [999]   [  4]
  [  7]   [999]   [999]   [  5]   [  0]

Matrix: D
  [  0]   [  8]   [  5]   [  9]   [  7]
  [ 11]   [  0]   [  2]   [  1]   [  4]
  [  9]   [  3]   [  0]   [  4]   [  2]
  [ 11]   [ 19]   [ 16]   [  0]   [  4]
  [  7]   [ 15]   [ 12]   [  5]   [  0]

Matrix: P
  [  0]   [  2]   [  0]   [  2]   [  2]
  [  4]   [  0]   [  0]   [  0]   [  2]
  [  4]   [  0]   [  0]   [  1]   [  0]
  [  4]   [  4]   [  4]   [  0]   [  0]
  [  0]   [  2]   [  0]   [  0]   [  0]
```

[그림 1] 행렬 W, D, P

```
void path(int q, int r) {
   if(P[q][r] != 0) {
      path(q, P[q][r]);
      printf("[v%d]\n", P[q][r]);
      path(P[q][r], r);
   }
}
```

[코드 1] path( )

의사 코드를 살펴보면 q가 시작점, r이 끝점이라고 할 때, P[q][r]은 시작점과 끝점을 지나는 최단 경로의 최고차수 값이 저장되어 있다. q=0, r=3이라고 하였을 때, 행렬 P에서 해당 인덱스에 저장된 값을 통해 결과 값이 0이면 q에서 r로 가는 것이 최단 경로이고, 다른 값이라면, 결과값 점을 통해서 해당 경로로 가야한다는 것을 의미한다. 즉 재귀를 이용하여 path(q, P[q][r])을 호출하고 가장 짧은 경로를 선택하여 따라간다. 모든 재귀가 종료되면 출력된 값이 최종적으로 도출되는 최단 경로이다.

```
path(0, 3) = 2
path(0, 2) = 0
path(0, 2) return
[v2]
path(2, 3) = 1
path(2, 1) = 0
path(2, 1) return
[v1]
path(1, 3) = 0
path(1, 3) return
path(2, 3) return
path(0, 3) return
```

[그림 2] path(0, 3)의 단계적 결과

즉 위의 결과를 통해 P에 저장된 값이 0면 리턴이 되고 내부 재귀에서 r이 최단 경로로 가는 점인 것을 알 수 있다. 그리고 출력한 뒤, 다음 최단 거리의 점을 찾고 이것이 반복되다가 최종 목표 지점에 도달할시 모든 재귀를 반환하고 종료하는 것을 알 수 있다.

2. Floyd 알고리즘 구현 및 v0부터 v3까지의 최단 경로

```
void floyd(void) {
    for(int k = 0; k < NUM; k++) // 거쳐가는 노드
        for(int i = 0; i < NUM; i++) // 출발 노드
            for(int j = 0; j < NUM; j++) // 도착 노드
                if (D[i][k] + D[k][j] < D[i][j]) {
                    P[i][j] = k;
                    D[i][j] = D[i][k] + D[k][j];
                }
}
```

[코드 2] floyd( )

Floyd 알고리즘은 최단거리의 가중치를 저장한 행렬 W, 최단거리의 가중치 합을 저장한 행렬 D, 최단거리의 최고 차항을 저장한 행렬 P을 구한다. 3중 for문을 사용하여 k는 거쳐가는 노드, i는 출발점, j는 도착점을 결정한다. 만약 경유하는 것이 바로 가는 것보다 빠를 경우 거쳐가는 점 k를 P[i][j]에 저장하고, D[i][j]에 경유하는 최단 경로의 가중치 합을 저장한다.

floyd( )를 통해 주어진 v0부터 v3까지의 최단 경로를 구하면 [v0, v2, v1, v3]이다.

3. {v0, v1, v2, v3, v4} -> {a2, a3, a4, a0, a1}

주어진 명세서는 a3, a4, a5, a1, a2로 명시되었으나, 이 실험에서는 시각적 편의를 위하여 시작 인덱스를 0

으로시작하였다. {v0, v1, v2, v3, v4}를 {a2, a3, a4, a0, a1}으로 치환하면 행렬 W, D, P는 다음과 같다.

```
Matrix: W
  [  0]   [  4]   [999]   [999]   [999]
  [  5]   [  0]   [  7]   [999]   [999]
  [999]   [999]   [  0]   [ 10]   [  5]
  [  1]   [999]   [999]   [  0]   [  2]
  [  9]   [  2]   [999]   [  3]   [  0]

Matrix: D
  [  0]   [  4]   [ 11]   [ 19]   [ 16]
  [  5]   [  0]   [  7]   [ 15]   [ 12]
  [  9]   [  7]   [  0]   [  8]   [  5]
  [  1]   [  4]   [ 11]   [  0]   [  2]
  [  4]   [  2]   [  9]   [  3]   [  0]

Matrix: P
  [  0]   [  0]   [  1]   [  4]   [  2]
  [  0]   [  0]   [  0]   [  4]   [  2]
  [  4]   [  4]   [  0]   [  4]   [  0]
  [  0]   [  4]   [  4]   [  0]   [  0]
  [  3]   [  0]   [  1]   [  0]   [  0]
```

[그림 3] 행렬 W, D, P

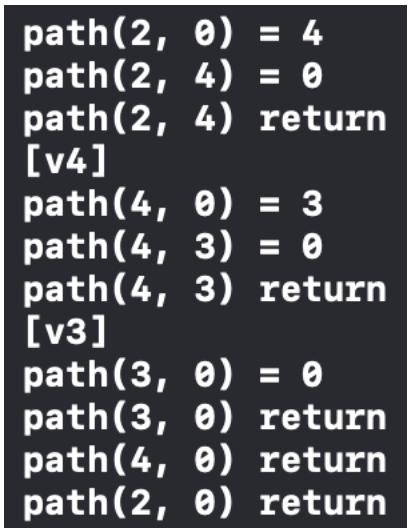
이렇게 변환된 행렬을 가지고 v0에서 v3까지의 최단 경로를 구하면 [v0, v1, v2, v4, v3]이다.

```
path(0, 3) = 4
path(0, 4) = 2
path(0, 2) = 1
path(0, 1) = 0
path(0, 1) return
[v1]
path(1, 2) = 0
path(1, 2) return
path(0, 2) return
[v2]
path(2, 4) = 0
path(2, 4) return
path(0, 4) return
[v4]
path(4, 3) = 0
path(4, 3) return
path(0, 3) return
```

[그림 4] path(0, 3)의 단계적 결과

4. a2에서 a0까지의 최단거리

최단 경로는 [a2, a4, a3, a0]이며 이를 시각적으로 그래프를 가정했을 때, 치환하기 전인 v0에서 v3까지의 최단 거리와 동일하다는 것을 알 수 있다.



[그림 5] path(2, 0)의 단계적 결과