Floyd-Warshall

이산수학
천세진

1

Floyd-Warshall
플로이드-워셜 알고리즘

- Directed, weighted graph G=(V,E)가 주어졌을 때,
  - Negative-weight edges 가 존재할 수 있음
  - 예로, -4, -1

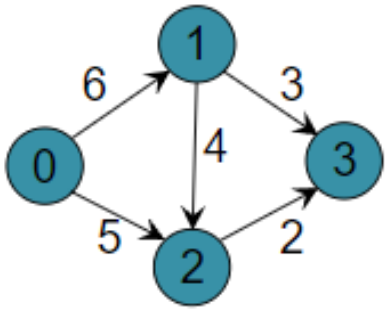
- 그래프 내 vertices의 모든 상간에 최소 경로비용을 계산

- 이용 사례: 도로망(Road map)
  - 각 위치는 vertex이고, 위치간 연결하는 도로는 edge로 표현

2

Weighted, Directed 그래프를 인접 행렬에 저장하기

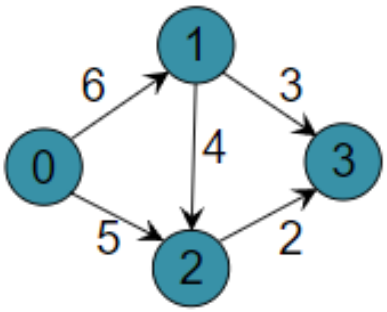
- D(i,j) 는 edge (i,j)의 weight
  - Weight는 비용, 시간, 거리가 될 수 있다.
- 인접 행렬(adjacency matrix)에서 무한대 값(INF)는 어떠한 edge가 없다는 것임



3

모든 vertex간에 최소비용 경로를 어떻게 계산할 것인가?

	0	**1**	**2**	**3**
0	0	6	5	INF
1	INF	0	4	3
2	INF	INF	0	2
3	INF	INF	INF	0



4

우선, Warshall 알고리즘부터

- 관계행렬일때 사용되며, 연결관계가 있는지를 체크하기 위해 사용됨
  - 즉 행렬의 값이 0과 1로만 표현될때임
- 지속적인 부울곱(Boolean product)보다 계산이 용이함

$$W_0 = M_R = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \Rightarrow W_3 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

5

Warshall 알고리즘

- STAGE0: N x N 행렬이라면 W_N까지 구한다
- STAGE1: W_{k-1}의 1을 W_k로 옮겨 쓴다
- STAGE2: W_{k-1} 에서, 값이 1인 k열의 위치(p1, p2, …)와 k행의 위치(q1,q2, …)을 나열한다
- STAGE3: W_k의 (pi, qi)의 위치에 모두 1로 변경한다

6

예제와 함께 수행

$$W_0 = M_R = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

- STAGE1: 4 X 4 이기 때문에, W_4까지 계산함
- STAGE2: W_1

$$W_1 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

1. 우선 W0에서 1값을 W1에 복사

2. W0 : k = 1 에 대한 다음 행/열의 위치 계산

1열위치: (2)
1행위치: (2)

W1 (2, 2) 에 대해서 1로 업데이트

7

예제와 함께 수행

$$W_0 = M_R = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

- STAGE1: 4 X 4 이기 때문에, W_4까지 계산함
- STAGE2: W_1

$$W_1 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

1. 우선 W0에서 1값을 W1에 복사

2. W0 : k = 1 에 대한 다음 행/열의 위치 계산

1열위치: (2)
1행위치: (2)

W1 (2, 2) 에 대해서 1로 업데이트

$$W_1 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

8

예제와 함께 수행

• STAGE2: $\boldsymbol{W_2}$

$$W_2 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

1. 우선 W1에서 1값을 W2에 복사

$$W_1 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

2. W1 : k = 2 에 대한 다음 행/열의 위치 계산

2열위치: (1,2)
2행위치: (1,2,3)

W2 (1,1), (1,2), (1,3), (2,1), (2,2), (2,3)에 대해 1로 업데이트

$$W_2 = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

9

예제와 함께 수행

• STAGE2: $\boldsymbol{W_3}$

$$W_3 = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

1. 우선 W2에서 1값을 W3에 복사

$$W_2 = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

2. W2 : k = 3 에 대한 다음 행/열의 위치 계산

3열위치: (1,2)
3행위치: (4)

W3 (1,4), (2,4), 에 대해 1로 업데이트

$$W_3 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

10

예제와 함께 수행

- STAGE2: $\boldsymbol{W_4}$

$$W_3 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

2. W3 : k = 4 에 대한 다음 행/열의 위치 계산

4열위치: (1,2,3)
4행위치: x

더 이상 업데이트 하지 않음

$$W_4 = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

1. 우선 W3에서 1값을 W4에 복사

11

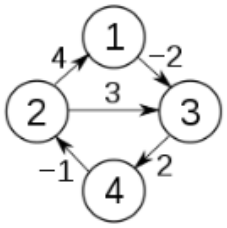
Algorithm

```
def warshall(a):
    assert (len(row) == len(a) for row in a)
    n = len(a)
    for k in range(n):
        for i in range(n):
            for j in range(n):
                a[i][j] = a[i][j] or (a[i][k] and a[k][j])
    return a
```

12

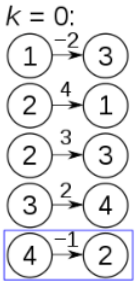
Floyd-Warshall Algorithms

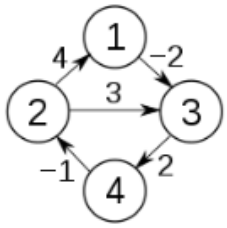
```
let dist be a |V| × |V| array of minimum distances initialized to ∞ (infinity)
for each edge (u, v) do
    dist[u][v] ← w(u, v)  // The weight of the edge (u, v)
for each vertex v do
    dist[v][v] ← 0
for k from 1 to |V|
    for i from 1 to |V|
        for j from 1 to |V|
            if dist[i][j] > dist[i][k] + dist[k][j]
                dist[i][j] ← dist[i][k] + dist[k][j]
            end if
```



13

Initialize



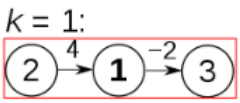


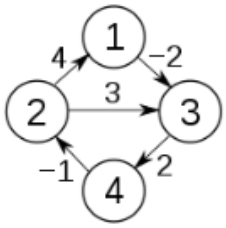
$k = 0$		j			
		1	**2**	**3**	**4**
i	**1**	0	∞	−2	∞
	2	4	0	3	∞
	3	∞	∞	0	2
	4	∞	−1	∞	0

```
let dist be a |V| × |V| array of minimum distances initialized to ∞ (infinity)
for each edge (u, v) do
    dist[u][v] ← w(u, v)  // The weight of the edge (u, v)
for each vertex v do
    dist[v][v] ← 0
for k from 1 to |V|
    for i from 1 to |V|
        for j from 1 to |V|
            if dist[i][j] > dist[i][k] + dist[k][j]
                dist[i][j] ← dist[i][k] + dist[k][j]
            end if
```

14

K=1



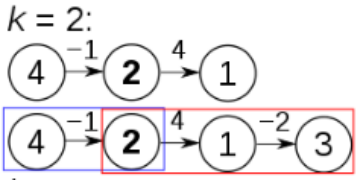


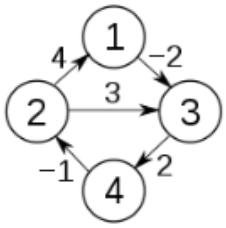
$k=1$		j			
		1	**2**	**3**	**4**
i	**1**	0	∞	−2	∞
	2	4	0	**2**	∞
	3	∞	∞	0	2
	4	∞	−1	∞	0

```
let dist be a |V| × |V| array of minimum distances initialized to ∞ (infinity)
for each edge (u, v) do
    dist[u][v] ← w(u, v)  // The weight of the edge (u, v)
for each vertex v do
    dist[v][v] ← 0
for k from 1 to |V|
    for i from 1 to |V|
        for j from 1 to |V|
            if dist[i][j] > dist[i][k] + dist[k][j]
                dist[i][j] ← dist[i][k] + dist[k][j]
            end if
```

15

K=2



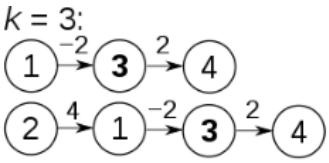


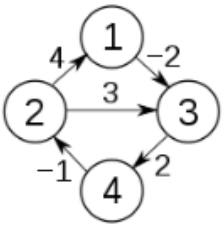
$k=2$		j			
		1	**2**	**3**	**4**
i	**1**	0	∞	−2	∞
	2	4	0	2	∞
	3	∞	∞	0	2
	4	**3**	−1	**1**	0

```
let dist be a |V| × |V| array of minimum distances initialized to ∞ (infinity)
for each edge (u, v) do
    dist[u][v] ← w(u, v)  // The weight of the edge (u, v)
for each vertex v do
    dist[v][v] ← 0
for k from 1 to |V|
    for i from 1 to |V|
        for j from 1 to |V|
            if dist[i][j] > dist[i][k] + dist[k][j]
                dist[i][j] ← dist[i][k] + dist[k][j]
            end if
```

16

K=3



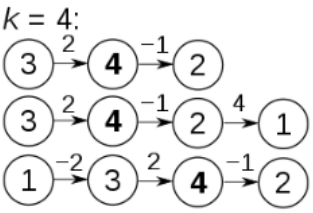


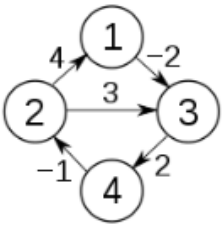
$k=3$		j			
		1	**2**	**3**	**4**
i	**1**	0	∞	−2	**0**
	2	4	0	2	**4**
	3	∞	∞	0	2
	4	3	−1	1	0

```
let dist be a |V| × |V| array of minimum distances initialized to ∞ (infinity)
for each edge (u, v) do
    dist[u][v] ← w(u, v)  // The weight of the edge (u, v)
for each vertex v do
    dist[v][v] ← 0
for k from 1 to |V|
    for i from 1 to |V|
        for j from 1 to |V|
            if dist[i][j] > dist[i][k] + dist[k][j]
                dist[i][j] ← dist[i][k] + dist[k][j]
            end if
```

17

K=4





$k=4$		j			
		1	**2**	**3**	**4**
i	**1**	0	**−1**	−2	0
	2	4	0	2	4
	3	**5**	**1**	0	2
	4	3	−1	1	0

```
let dist be a |V| × |V| array of minimum distances initialized to ∞ (infinity)
for each edge (u, v) do
    dist[u][v] ← w(u, v)  // The weight of the edge (u, v)
for each vertex v do
    dist[v][v] ← 0
for k from 1 to |V|
    for i from 1 to |V|
        for j from 1 to |V|
            if dist[i][j] > dist[i][k] + dist[k][j]
                dist[i][j] ← dist[i][k] + dist[k][j]
            end if
```

18

Performance

- 시간복잡도 | V^3 |
- 병렬처리가 가능하기 때문에 유용하게 사용됨
  - OpenMP 혹은 GPU 연산 가능
- All Pairs Shortest Paths (APSP)라고 대신 이야기함

19

References

- https://www.geeksforgeeks.org/floyd-warshall-algorithm-dp-16/
- https://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm

20