파이썬으로 배우는 알고리즘 기초

Chap 3. 동적계획

3.2 최단 경로와 플로이드 알고리즘

3.2 최단 경로 (플로이드 알고리즘)

- **최단 경로** 문제
  - 주어진 그래프에서 모든 정점의 쌍에 대한 최단 경로를 구하시오.

  - 엄밀한 문제 정의:
    - $G = (V, E)$: G는 그래프, V는 정점(*vertex*)의 집합, E는 간선(*edge*)의 집합
    - 그래프 G는 방향성(*directed*), 가중치(*weighted*) 그래프임
    - 최단 경로는 단순 경로(*simple path*): 같은 정점을 두 번 거치지 않음(*acyclic*)

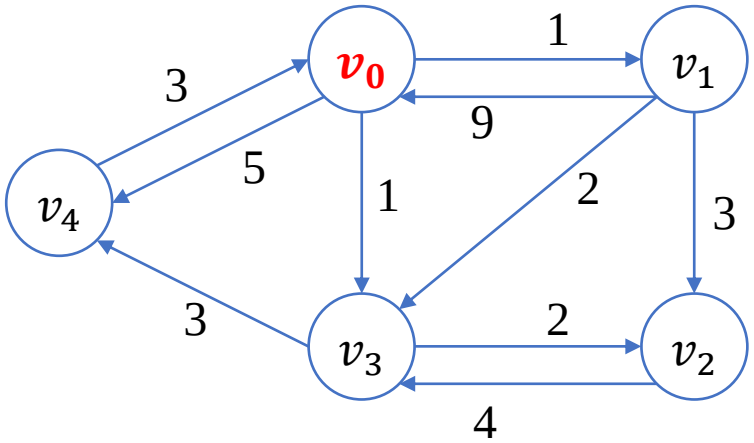
3.2 최단 경로 (플로이드 알고리즘)

- 최단 경로 문제의 이해
  - 단순무식한 방법으로 해결하기
    - 각 정점에서 다른 정점으로 가는 모든 경로를 구한 뒤
    - 그 경로들 중에서 가장 짧은 경로를 찾는 방법
    - 효율성 분석 (최악의 경우 = 모든 정점간의 간선이 존재할 때)
      - $(n-2)(n-3)\cdots 1 = (n-2)!$ (지수 시간 복잡도)

  - 최단 경로 문제는 **최적화 문제**(*optimization problem*)
    - 최적화 문제는 하나 이상의 해답 후보가 있을 수 있고,
    - 해답 후보 중에서 가장 최적의 값(*optimal value*)을 가진 해답을 찾는 문제

3.2 최단 경로 (플로이드 알고리즘)

- **최단 경로 문제의 입력 사례**
  - **그래프의 표현: 인접 행렬**(*adjacency matrix*)
  - v_0**에서** v_2**로 가는 가능한 단순 경로의 수는?** 3
  - v_0**에서** v_2**로 가는 최단 경로와 경로 길이는?** $[v_0, v_3, v_2], length = 3$



W	0	1	2	3	4
0	0	1	∞	1	5
1	9	0	3	2	∞
2	∞	∞	0	4	∞
3	∞	∞	2	0	3
4	3	∞	∞	∞	0

3.2 최단 경로 (플로이드 알고리즘)

- 최단 경로: 동적계획(Dynamic Programming)
  - 1단계: 재귀 관계식을 찾는다.
    - $\boldsymbol{D}$: 각 정점의 쌍이 가지는 최단 경로의 길이를 나타내는 행렬
    - $\boldsymbol{D[i][j]}$: v_i에서 v_j로 가는 최단 경로의 길이
    - 목표: 인접 행렬 W에서 최단 경로의 행렬 D와의 재귀 관계식 구하기
  - 2단계: 상향식 방법으로 해답을 구한다.
    - 초기화: $D^0 = W$,
    - 최종 목표: $D^n = D$.
    - 상향식 계산: $D^0, D^1, \cdots, D^n$

3.2 최단 경로 (플로이드 알고리즘)

- 최단 경로 행렬의 이해
  - D^k: k개의 중간 정점을 지나는 최단 경로 길이의 행렬
  - $D^k[i][j]$: v_i에서 v_j로 k개의 중간 정점을 지나는 최단 경로 길이
  - D^0: 다른 어떤 정점도 지나지 않는 최단 경로의 길이 $(= W)$
  - D^n: 다른 모든 정점을 지날 수 있는 최단 경로의 길이 $(= D)$

3.2 최단 경로 (플로이드 알고리즘)

$$D^0 \rightarrow D^1 \rightarrow D^2 \rightarrow \cdots \rightarrow D^n$$

W	0	1	2	3	4
0	0	1	∞	1	5
1	9	0	3	2	∞
2	∞	∞	0	4	∞
3	∞	∞	2	0	3
4	3	∞	∞	∞	0

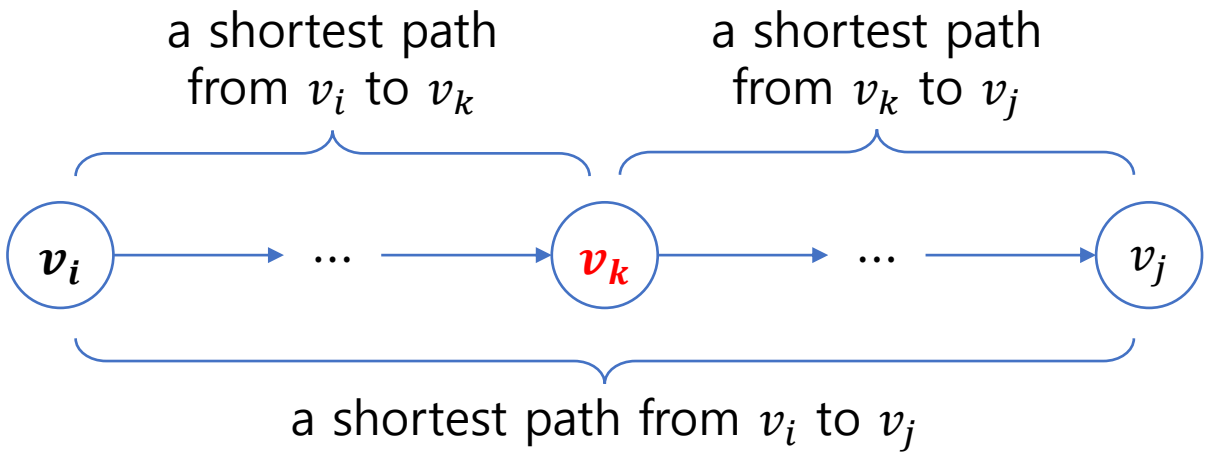
D	0	1	2	3	4
0	0	1	**3**	1	4
1	8	0	3	2	5
2	10	11	0	4	7
3	6	7	2	0	3
4	3	4	6	4	0

3.2 최단 경로 (플로이드 알고리즘)

- **재귀 관계식 구하기**
  - $D^0 = W$, D^k는 D^{k-1}로부터 구함 $(1 \leq k \leq n)$
  - $D^{k-1}[i][j]$: v_i에서 v_j로 $k-1$개의 중간 정점을 지남
  - $D^k[i][j]$: 다음과 같은 두 가지의 경우를 고려
    - 경우 1: 하나의 정점을 더 지나게 해 줘도 새로운 최단 경로가 없는 경우
      - $D^k[i][j] = D^{k-1}[i][j]$
    - 경우 2: 하나의 정점(v_k)을 더 지나면 새로운 최단 경로가 있는 경우
      - $D^k[i][j] = D^{k-1}[i][k] + D^{k-1}[k][j]$

3.2 최단 경로 (플로이드 알고리즘)

- **최단 경로의 재귀 관계식**
  - $D^0 = W$
  - $D^k[i][j] = \min\left(D^{k-1}[i][j],\ D^{k-1}[i][k] + D^{k-1}[k][j]\right)$



3.2 최단 경로 (플로이드 알고리즘)

$D^0 = W$

D^0	0	1	2	3	4
0	0	1	∞	1	5
1	9	0	3	2	∞
2	∞	∞	0	4	∞
3	∞	∞	2	0	3
4	3	∞	∞	∞	0

$D^1[i][j] = \min(D^0[i][j], D^0[i][k] + D^0[k][j])$

D^1	0	1	2	3	4
0					
1				2	
2					
3					
4		4		4	

3.2 최단 경로 (플로이드 알고리즘)

Algorithm 3.3: Floyd's Algorithm for Shortest Paths

```
def floyd (W):
    D = W
    n = len(W)
    for k in range(n):
        for i in range(n):
            for j in range(n):
                D[i][j] = min(D[i][j], D[i][k] + D[k][j])
    return D
```

3.2 최단 경로 (플로이드 알고리즘)


```
INF = 999
W = [
    [0, 1, INF, 1, 5],
    [9, 0, 3, 2, INF],
    [INF, INF, 0, 4, INF],
    [INF, INF, 2, 0, 3],
    [3, INF, INF, INF, 0]
]

D = floyd(W)
for i in range(len(D)):
    print(D[i])
```

```
[0, 1, 3, 1, 4]
[8, 0, 3, 2, 5]
[10, 11, 0, 4, 7]
[6, 7, 2, 0, 3]
[3, 4, 6, 4, 0]
```

3.2 최단 경로 (플로이드 알고리즘)

- 좋아. 좋은데... 최단 경로는?
  - Algorithm 3.3에서는 최단 경로의 **길이**만 구함
  - 최단 **경로**를 구하기 위해서는 그 과정을 기록해야 함

  - $P[i][j]$: v_i에서 v_j로 가는 최단 경로가 **거쳐야 하는** 새로운 정점
    - v_i에서 v_j로 가는 최단 경로의 중간에 놓여있는 정점이 최소한 하나가 있는 경우에는 그 놓여있는 정점 중에서 가장 큰 인덱스
    - 최단 경로의 중간에 놓여있는 **정점이 없는** 경우에는 -1

Algorithm 3.4: Floyd's Algorithm for Shortest Paths 2

```
def floyd2 (W):
    n = len(W)
    D = W
    P = [[-1] * n for _ in range(n)]
    for k in range(n):
        for i in range(n):
            for j in range(n):
                if (D[i][j] > D[i][k] + D[k][j]):
                    D[i][j] = D[i][k] + D[k][j]
                    P[i][j] = k
    return D, P
```

3.2 최단 경로 (플로이드 알고리즘)

- 그러면... 최단 경로는 어떻게 찾지?
  - $P[i][j] = -1$이면, 간선 (v_i, v_j)가 최단 경로임
  - $P[i][j] = k$ 인 경우에는 *inorder* 탐색을 함
    - (v_i, v_k)의 최단 경로 출력 후
    - v_k를 출력하고
    - (v_k, v_j)의 최단 경로 출력

P	0	1	2	3	4
0	-1	-1	3	-1	3
1	4	-1	-1	-1	3
2	4	4	-1	-1	3
3	4	4	-1	-1	-1
4	-1	0	3	0	-1

```
(4, 2)

(4, 2): 3
(4, 3): 0
(4, 0): -1
print '0'
(0, 3): -1
print '3'
(3, 2): -1
```

Algorithm 3.5: Print Shortest Path

```
def path (P, u, v):
    if (P[u][v] != -1):
        path(P, u, P[u][v])
        print('v%d'%(P[u][v]), end='-> ')
        path(P, P[u][v], v)
```

```
D, P = floyd2(W)
for i in range(len(D)):
    print(D[i])
for i in range(len(P)):
    print(P[i])
```

3.2 최단 경로 (플로이드 알고리즘)


```
u = 4
v = 2
print('shortest path from v%d to v%d:'%(u, v), end=' ')
print('v%d'%(u), end='-> ')
path(P, u, v)
print('v%d'%(v), end=' ')
```

```
    [0, 1, 3, 1, 4]                 [-1, -1, 3, -1, 3]
    [8, 0, 3, 2, 5]                 [4, -1, -1, -1, 3]
    [10, 11, 0, 4, 7]               [4, 4, -1, -1, 3]
    [6, 7, 2, 0, 3]                 [4, 4, -1, -1, -1]
    [3, 4, 6, 4, 0]                 [-1, 0, 3, 0, -1]

    shortest path from v4 to v2: v4-> v0-> v3-> v2
```

주니온TV@Youtube

자세히 보면 유익한 코딩 채널

https://bit.ly/2JXXGqz

- 여러분의 **구독**과 **좋아요**는 강의제작에 큰 힘이 됩니다.
- 강의자료 및 소스코드: **구글 드라이브**에서 다운로드
  (다운로드 주소는 영상 하단 설명란 참고)

  https://bit.ly/3fN0q8t