파이썬으로 배우는 알고리즘 기초

Chap 6. 분기 한정법

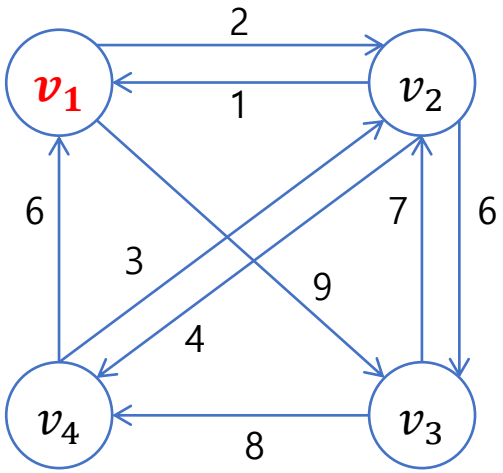
3.6 외판원 문제와 동적 계획법

3.6 외판원 문제와 동적 계획법

- **외판원 문제**: 동적 계획법으로 풀기
  - W: 주어진 그래프 $G = (V, E)$의 인접 행렬
  - V: 모든 도시의 집합
  - A: V의 부분 집합
  - $D[v_i][A]$: A에 속한 도시를 각각 한 번씩만 거쳐서 v_i에서 v_1 으로 가는 최단 경로의 길이

3.6 외판원 문제와 동적 계획법

- 주어진 그래프 G=(V, E)
  - 인접행렬 W의 구성



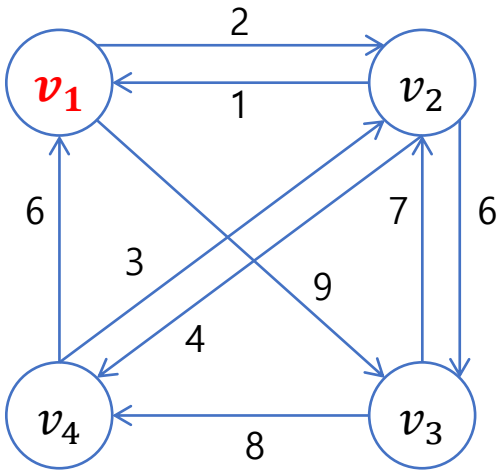
W	1	2	3	4
1	0	2	9	∞
2	1	0	6	4
3	∞	7	0	8
4	6	3	∞	0

3.6 외판원 문제와 동적 계획법

재귀적 관계식 찾기

- $V = \{v_1, v_2, v_3, v_4\}$
- If $A = \{v_3\}$,
  - $D[v_2][A] = lengh(v_2\ v_3\ v_1) = \infty$
- If $A = \{v_3, v_4\}$,
  - $D[v_2][A] = \min(lengh(v_2\ v_3\ v_4\ v_1), lengh(v_2\ v_4\ v_3\ v_1))$

$$= \min(20, \infty) = 20$$



3.6 외판원 문제와 동적 계획법

- **재귀 관계식**
  - 일반적으로 $i \neq 1$ 이고 $v_i \notin A$이면,
    - $D[v_i][A] = \underset{j:v_j \in A}{\text{minimum}}\left(W[i][j] + D\left[v_j\right][A - \{v_j\}]\right), A \neq \phi$
    - $D[v_i][\phi] = W[i][1], A = \phi$

3.6 외판원 문제와 동적 계획법

$A = \phi$:
- $D[v_2][\phi] = W[2][1] = 1$
- $D[v_3][\phi] = W[3][1] = \infty$
- $D[v_4][\phi] = W[4][1] = 6$

$A = \{v_2\}$:
- $D[v_3][A] = \underset{j:v_j \in A}{\text{minimum}}\left(W[3][j] + D\left[v_j\right][A - \{v_j\}]\right)$
$= W[3][2] + D[v_2][\phi] = 7 + 1 = 8$
- $D[v_4][A] = 4$

$A = \{v_3\}$:
- $D[v_2][A] = \infty$
- $D[v_4][A] = \infty$

$A = \{v_4\}$:
- $D[v_2][A] = 10$
- $D[v_3][A] = 14$

3.6 외판원 문제와 동적 계획법

$A = \{v_2, v_3\}$:

- $D[v_4][A] = \underset{j:v_j \in A}{\text{minimum}}\left(W[4][j] + D[v_j][A - \{v_j\}]\right)$
  $= \text{minimum}(W[4][2] + D[v_2][\{v_3\}], W[4][3] + D[v_3][\{v_2\}])$
  $= \text{minimum}(3 + \infty, \infty + 8) = \infty$

$A = \{v_2, v_4\}$:

- $D[v_3][A] = 12$

$A = \{v_3, v_4\}$:

- $D[v_2][A] = 20$

$A = \{v_2, v_3, v_4\}$:

- $D[v_1][A] = \underset{j:v_j \in A}{\text{minimum}}\left(W[1][j] + D[v_j][A - \{v_j\}]\right)$
  $= \text{minimum}(W[1][2] + D[v_2][\{v_3, v_4\}],$
  $W[1][3] + D[v_3][\{v_2, v_4\}],$
  $W[1][4] + D[v_4][\{v_2, v_3\}])$
  $= \text{minimum}(2 + 20, 9 + 12, \infty + \infty) = 21$

3.6 외판원 문제와 동적 계획법


```
void travel(int n, int W[][], int P[][], int &minlength) {
    int i, j, k, A, D[1..n][subset of V-{v1}];
    for (i = 2; i <= n; i++)
        D[i][∅] = W[i][1];
    for (k = 1; k <= n - 2; k++)
        for (/* all subsets A ⊆ V-{v1} containing k vertices */)
            for (/* all i s.t. i != 1 and vi∉A */) {
                // Find the minimum from i to j
                D[i][A] = minimum_{j:vj∈A}(W[i][j] + D[j][A − {vj}]);
                P[i][A] = j; // the value that gave the minimum
            }
    D[1][V − {v1}] = minimum_{2≤j≤n}(W[1][j] + D[j][V − {v1, vj}]);
    P[1][V − {v1}] = j; // the value that gave the minimum
}
```

3.6 외판원 문제와 동적 계획법

비트를 이용한 부분 집합의 표현과 연산

- $V - \{v_1\} = \{v_2, v_3, v_4\}$

v_4 v_3 v_2
↓ ↓ ↓
1 0 1
↓ ↓
$A = \{\ v_4,$ $v_2\ \} = 5$

$A = \phi$: 000 = 0

$A = \{v_2\}$: 001 = 1

$A = \{v_3\}$: 010 = 2

$A = \{v_4\}$: 100 = 4

$A = \{v_2, v_3\}$: 011 = 3

$A = \{v_2, v_4\}$: 101 = 5

$A = \{v_3, v_4\}$: 110 = 6

$A = \{v_2, v_3, v_4\}$: 111 = 7

$(=2^{n-1} - 1)$

3.6 외판원 문제와 동적 계획법

- $v_i \in A$: `isIn(i, A)`

```
def isIn (i, A):
    if ((A & (1 << (i - 2))) != 0):
        return True
    else:
        return False
```

3.6 외판원 문제와 동적 계획법

- $A - \{v_j\}$: `diff(A, j)`

```
def diff (A, j):
    t = 1 << (j - 2)
    return (A & (~t))
```

3.6 외판원 문제와 동적 계획법

- A의 원소가 k개인가?: `count(A, n)`

```
def count (A, n):
    count = 0
    for i in range(n):
        if ((A & (1 << i)) != 0):
            count += 1
    return count
```

Algorithm 3.11: Dynamic Programming for the TSP

```
def travel (W):
    n = len(W) - 1
    size = 2 ** (n - 1)
    D = [[0] * size for _ in range(n + 1)]
    P = [[0] * size for _ in range(n + 1)]
    for i in range(2, n + 1):
        D[i][0] = W[i][1]
    for k in range(1, n - 1):
        for A in range(1, size):
            if (count(A, n) == k):
                for i in range(2, n + 1):
                    if (not isIn(i, A)):
                        D[i][A], P[i][A] = minimum(W, D, i, A)
    A = size - 1
    D[1][A], P[1][A] = minimum(W, D, 1, A)
    return D, P
```

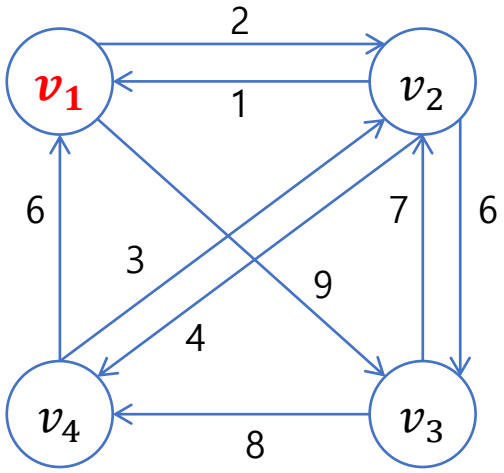
Algorithm 3.11: Dynamic Programming for the TSP

```
def minimum (W, D, i, A):
    minValue = INF
    minJ = 1
    n = len(W) - 1
    for j in range(2, n + 1):
        if (isIn(j, A)):
            m = W[i][j] + D[j][diff(A, j)]
            if (minValue > m):
                minValue = m
                minJ = j
    return minValue, minJ
```

3.6 외판원 문제와 동적 계획법


```
INF = 999
W = [
    [-1, -1, -1, -1, -1],
    [-1, 0, 2, 9, INF],
    [-1, 1, 0, 6, 4],
    [-1, INF, 7, 0, 8],
    [-1, 6, 3, INF, 0]
]

D, P = travel(W)
print('D =')
for i in range(1, len(D)):
    print(D[i])
print('P =')
for i in range(1, len(P)):
    print(P[i])
```



D	0	1	2	3	4	5	6	7
1								21
2	1		∞		10		20	
3	∞	8			14	12		
4	6	4	∞	∞				

- **최적의 일주여행 경로 찾기**
  - $P[i][A]$: A에 속한 모든 도시를 정확히 한 번만 거쳐서 v_i에서 v_1으로 가는 최단 경로에서 v_i 다음에 오는 첫 번째 도시의 인덱스

P	0	1	2	3	4	5	6	7
1								3
2			1		4		3	
3		2			4	4		
4		2	1	1				

$P[1][7], A = \{v_2, v_3, v_4\}$:

$P[3][5], A = \{v_2, v_4\}$:

$P[4][1], A = \{v_2\}$:

$P[2][1], A = \{\phi\}$:

Optimal tour: v_1, v_3, v_4, v_2, v_1

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