**챕터 7. Greedy Aproach (탐욕 알고리즘) - (3) 최단 경로와 다익스트라 알고리즘**

## 최단 경로 문제: Revisited

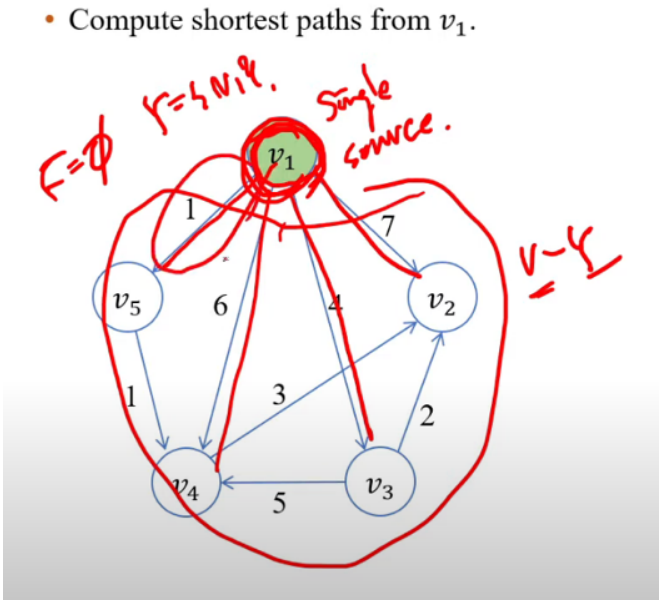
- 모든 정점의 쌍에 대한 최단 경로 구하기 -> All Pairs Shortest
    - 플로이드 알고리즘: 동적 계획법
- 단일 정점에서 모든 다른 정점으로의 최단 경로 구하기
    - 다익스트라 알고리즘: 탐욕법

## 다익스트라 알고리즘:

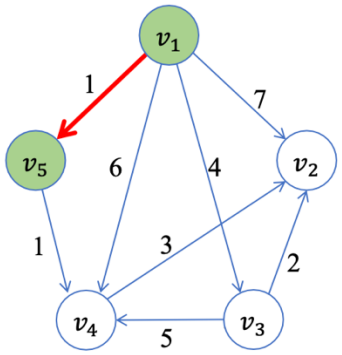
- 최소비용 신장트리 문제의 프림 알고리즘과 유사

$Y = \{v_1\};$
$F = \emptyset;$
**while (답을 구하지 못했음):**
    **$Y$에 속한 정점만 중간에 거쳐 가는 정점으로 하여**
        **$v_1$ 에서 최단경로를 가진 정점 $v$를 $V - Y$에서 선택한다.**
    **새로운 정점 $v$를 $Y$에 추가한다.**
    **(최단경로 상에서) $v$로 가는 간선을 $F$에 추가한다.**
    **if** $(Y == V)$
        **답을 구했음.**

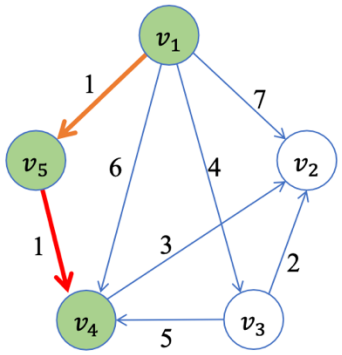
1. 출발 정점 v1, F = Φ
2. 다만 조건이 조금 다른데, 답을 다 구할때까지 반복하는데, 답은 Y가 V집합과 같을때 까지이다.
3. v1을 선택하는 기준이 다르다. (조금 더 복잡)



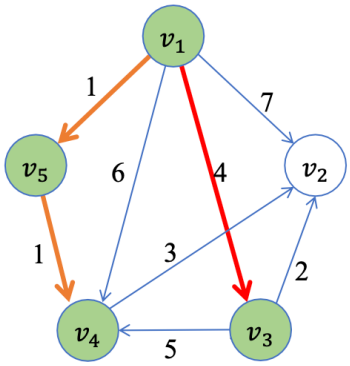
1. Vertex $v_5$ is selected because it is nearest to $v_1$.



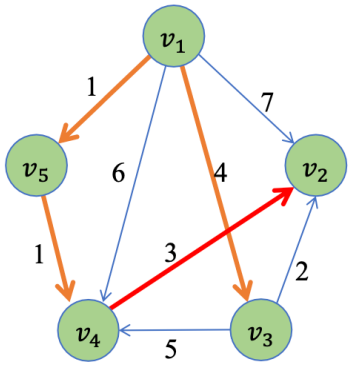

2. Vertex $v_4$ is selected because it has the shortest path from $v_1$ using only vertices in $\{v_5\}$ as intermediates.




3. Vertex $v_3$ is selected because it has the shortest path from $v_1$ using only vertices in $\{v_4, v_5\}$ as intermediates.




4. The shortest path from $v_1$ to $v_2$ is $[v_1, v_5, v_4, v_2]$.




- W[i][j]: 그래프 G의 인접행렬

- touch[i]: Y에 속한 정점들만 중간에 거치도록 하여 v1에서 vi로 가는 현재 최단경로 상의 마지막 간선을 (v, v1)라고 할 때, Y에 속한 정점 v의 인덱스

- length[i]: Y에 속한 정점들만 중간에 거치도록 하여 v1에서 vi로 가는 현재 최단 경로의 길이

| W | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 7 | 4 | 6 | 1 |
| 2 | ∞ | 0 | ∞ | ∞ | ∞ |
| 3 | ∞ | 2 | 0 | 5 | ∞ |
| 4 | ∞ | 3 | ∞ | 0 | ∞ |
| 5 | ∞ | ∞ | ∞ | 1 | 0 |

| | i | 2 | 3 | 4 | 5 | e |
|---|---|---|---|---|---|---|
| init: | touch[i] | 1 | 1 | 1 | 1 | |
| | length[i] | 7 | 4 | 6 | 1 | |
| step 1: | touch[i] | 1 | 1 | 5 | 1 | (1, 5, 1) |
| | length[i] | 7 | 4 | 2 | -1 | |
| step 2: | touch[i] | 4 | 1 | 5 | 1 | (5, 4, 1) |
| | length[i] | 5 | 4 | -1 | -1 | |
| step 3: | touch[i] | 4 | 1 | 5 | 1 | (1, 3, 4) |
| | length[i] | 5 | -1 | -1 | -1 | |
| step 4: | touch[i] | 4 | 1 | 5 | 1 | (4, 2, 3) |
| | length[i] | -1 | -1 | -1 | -1 | |

- touch == nearest, length == distance처럼 프림 알고리즘의 배열 구조와 같게 보면 쉽다.

## ALGORITHM 4.3: Dijkstra's Algorithm

```
void dijkstra(int n, matrix_t& W, set_of_edges& F)
{
    int vnear, min;
    vector<int> touch(n + 1), length(n + 1);

    F.clear();
    for (int i = 2; i <= n; i++) {
        touch[i] = 1;
        length[i] = W[1][i];
    }
```

## ALGORITHM 4.3: Dijkstra's Algorithm (continued)

```
    repeat (n - 1 times) {
        min = INF;
        for (int i = 2; i <= n; i++)
            if (0 <= length[i] && length[i] < min) {
                min = length[i];
                vnear = i;
            }
        e = edge from vertex indexed by touch[vnear];
        add e to F;
        for (int i = 2; i <= n; i++)
            if (length[i] > length[vnear] + W[vnear][i]) {
                length[i] = length[vnear] + W[vnear][i];
                touch[i] = vnear;
            }
        length[vnear] = -1;
    }
}
```

**시간복잡도 분석**

다익스트라 알고리즘: $O(n^2)$ -> 탐욕 접근법

플로이드 알고리즘: $O(n^3)$ -> 동적계획법