Prim

Prim 알고리즘

Prim 알고리즘은 최소 신장 트리를 구하는 알고리즘 중 하나

크루스칼 알고리즘과 같은 용도이지만, `응용 상황에서 두 알고리즘의 효율성이 달라질 수 있음`
Prim 알고리즘은 노드 중심으로 동작합니다. 시작 노드를 선택한 후, 그 노드와 인접한 노드 중 가장 작은 가중치를 가진 간선으로 연결합니다. 이후 연결된 노드를 기준으로 다시 가장 작은 가중치를 가진 간선을 찾아 연결하는 과정을 반복합니다.

알고리즘 동작 방식

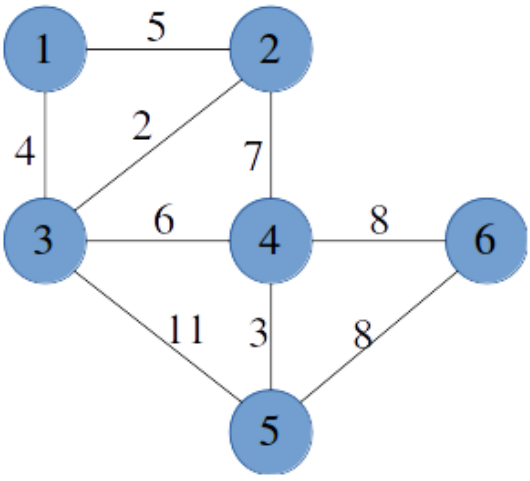
1. 시작 노드를 임의로 선택, 비어있는 T에 포함(이제 T는 노드가 한 개인 트리.)
2. T에 있는 노드 T와 없는 노드 사이의 간선 중 가중치가 최소인 간선 탐색
3. 찾은 간선이 연결하는 두 노드 중, T에 없던 노드를 T에 포함 시킨다

   (1에서 찾은 간선도 같이 T에 포함됨)
4. 모든 노드가 T에 포함될 때 까지 2, 3 반복

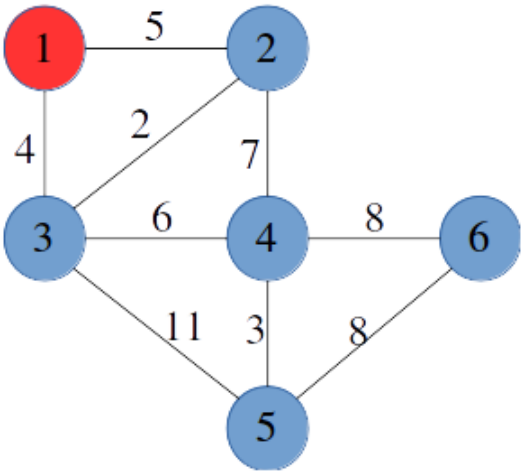
시간 복잡도

$T = O(ElogV)$ (where E : 간선의 개수, V : 노드의 개수)

예시

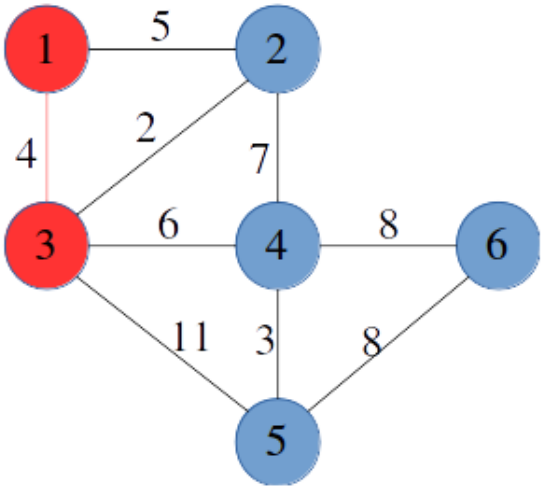


T에 포함된 노드는 빨간색, T에 아직 포함되지 않은 노드는 파란색으로 표시



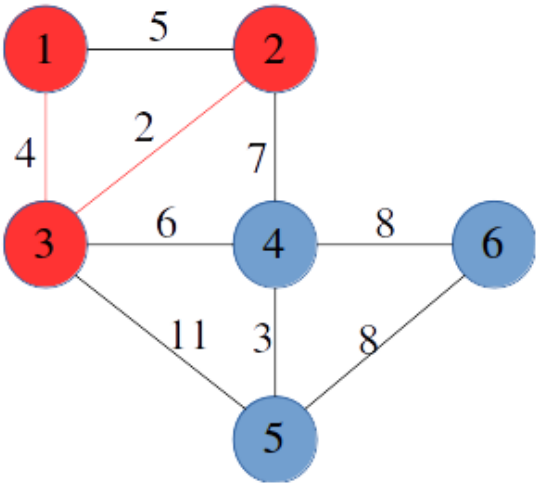
1. 임의의 정점을 선택하여 비어있는 T에 포함시킨다. (이제 T는 노드가 한 개인 트리.)
    - 초기 상태는 T에 아무것도 포함되어있지 않음. 임의의 정점으로는 아무거나 선택해도 상관없음

```
T = [1]
```



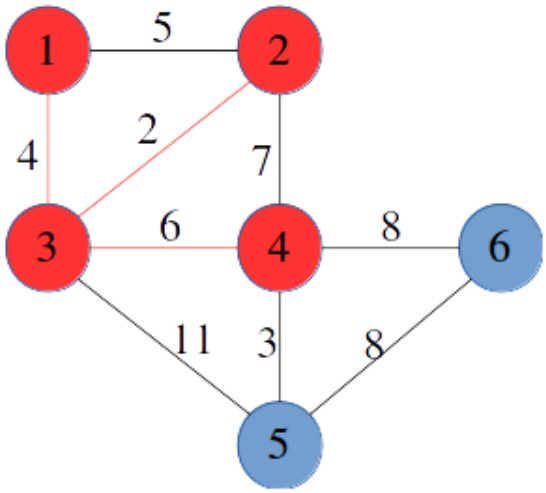
2. T에 있는 노드와 T에 없는 노드 사이의 간선 중 가중치가 최소인 간선을 찾음
    - 노드 1과 노드 3을 연결한 가중치 4의 간선이 최소 비용
3. 찾은 간선이 연결하는 두 노드 중 T에 없던 노드를 T에 포함 시킨다
    - 노드 3을 T에 포함 시킨다

```
T = [1, 3]
```



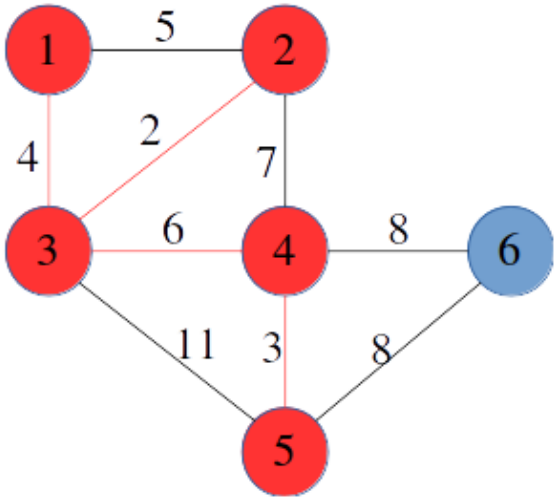
2. T에 있는 노드와 T에 없는 노드 사이의 간선 중 가중치가 최소인 간선을 찾음
    - 노드 3과 노드 2를 연결한 가중치 2의 간선이 최소 비용
3. 찾은 간선이 연결하는 두 노드 중 T에 없던 노드를 T에 포함 시킨다
    - 노드 2를 T에 포함 시킨다

```
T = [1, 3, 2]
```



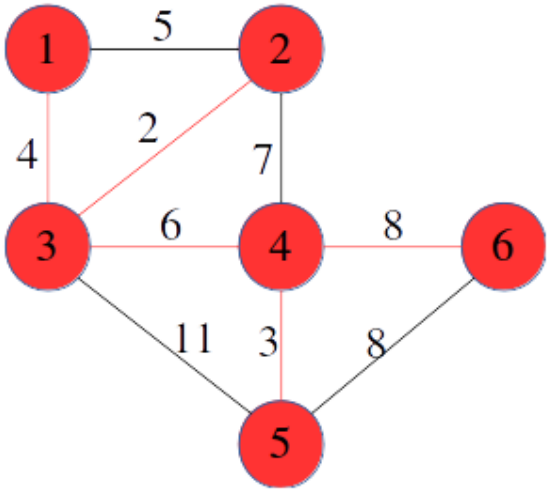
2. T에 있는 노드와 T에 없는 노드 사이의 간선 중 가중치가 최소인 간선을 찾음
    - 노드 3과 노드 4를 연결한 가중치 6의 간선이 최소 비용
3. 찾은 간선이 연결하는 두 노드 중 T에 없던 노드를 T에 포함 시킨다
    - 노드 4를 T에 포함 시킨다

`T = [1, 3, 2, 4]`



2. T에 있는 노드와 T에 없는 노드 사이의 간선 중 가중치가 최소인 간선을 찾음
    - 노드 4과 노드 5를 연결한 가중치 3의 간선이 최소 비용
3. 찾은 간선이 연결하는 두 노드 중 T에 없던 노드를 T에 포함 시킨다
    - 노드 5를 T에 포함 시킨다

`T = [1, 3, 2, 4, 5]`



2. T에 있는 노드와 T에 없는 노드 사이의 간선 중 가중치가 최소인 간선을 찾음
    - 노드 4과 노드 6를 연결한 가중치 3의 간선이 최소 비용
3. 찾은 간선이 연결하는 두 노드 중 T에 없던 노드를 T에 포함 시킨다
    - 노드 6을 T에 포함 시킨다
4. 모든 노드가 T에 포함됨

`T = [1, 3, 2, 4, 5, 6]`

- 가장 낮은 정점을 찾는 과정이 시간복잡도를 결정

구현 예시

heapq 자료구조 사용

```
import heapq


def prim(graph, start):
    # 그래프의 모든 정점을 방문했는지 여부를 나타내는 변수
    visited = set()

    # 시작 정점을 방문한 것으로 처리하고, 연결된 간선을 저장할 우선순위 큐를 초기화합니다.
    visited.add(start)
    edges = graph[start]
    heapq.heapify(edges)
    # 최소 신장 트리를 저장할 변수와, 가중치의 합을 저장할 변수를 초기화합니다.
    mst = []
    weight = 0

    # 모든 정점을 방문할 때까지 반복합니다.
    while edges:
        # 현재 가장 가중치가 낮은 간선을 꺼냅니다.
        vertex1, vertex2, weight = heapq.heappop(edges)

        # 해당 간선이 이미 연결된 정점과 연결하는 경우 무시합니다.
        if vertex1 not in visited:
            # 새로운 정점을 방문한 것으로 처리하고, 최소 신장 트리에 간선을 추가합니다.
            visited.add(vertex1)
            mst.append((vertex1, vertex2, weight))

            # 새로운 정점과 연결된 간선을 우선순위 큐에 추가합니다.
            for edge in graph[vertex1]:
                if edge[0] not in visited:
                    heapq.heappush(edges, edge)

    # 최소 신장 트리와 가중치의 합을 반환합니다.
    return mst, sum(weight for _, _, weight in mst)


# 다음과 같은 그래프를 이용하여 prim() 함수를 호출하면 최소 신장 트리와 가중치의 합을 반환합니다.
graph = {
    'A': [('B', 'A', 2), ('C', 'A', 3)],
    'B': [('A', 'B', 2), ('C', 'B', 4), ('D', 'B', 1)],
    'C': [('A', 'C', 3), ('B', 'C', 4), ('D', 'C', 5)],
    'D': [('B', 'D', 1), ('C', 'D', 5)]
}

mst, weight = prim(graph, 'A')

print(mst)  # [('B', 'A', 2), ('C', 'A', 3), ('D', 'B', 1)]
print(weight)  # 6
```