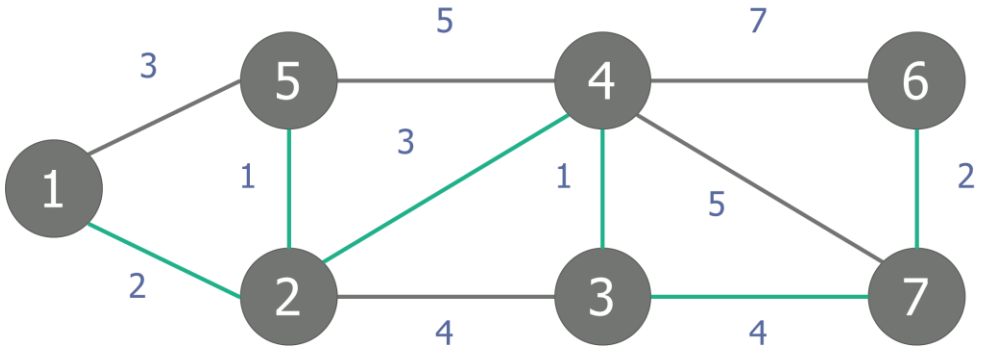
최소 스패닝 트리

Minimum Spanning Tree

- 스패닝 트리 : 그래프에서 일부 간선을 선택해서 만든 트리
- 최소 스패닝 트리 : 스패닝 트리 중에 선택한 간선의 가중치의 합이 최소인 트리
- 최소 비용 : 2(1 ~ 2) + 1(5 ~ 2) + 3(2 ~ 4) + 1(4 ~ 3) + 4(3 ~ 7) + 2(6 ~ 7)



- 프림(Prim) : MST의 정점을 점점 더 확장해 나아가는 방법
- 크루스칼(Kruskal) : MST의 간선을 점점 더 확장해 나아가는 방법

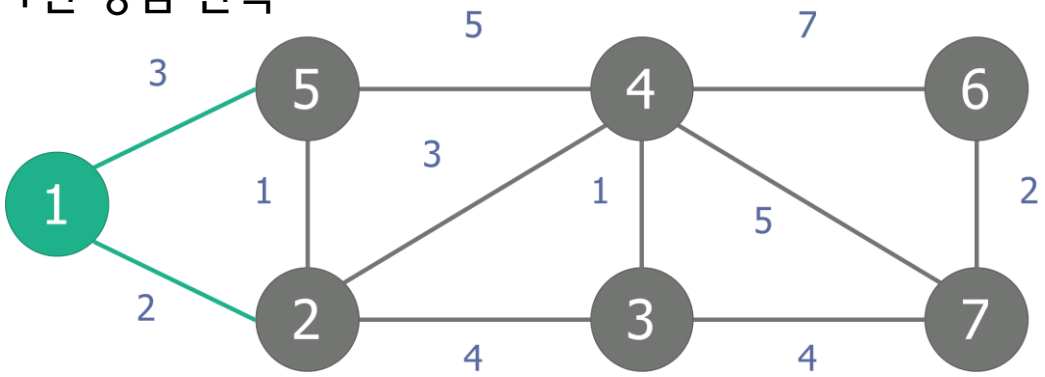
최소 스패닝 트리 - 프림

Minimum Spanning Tree

① 그래프에서 아무 정점이나 선택한다.
② 선택한 정점과 선택하지 않은 정점을 연결하는 간선 중에 최소값을 고른다.
이 간선을 (u, v)라고 한다. (u : 선택, v : 선택하지 않음)
③ 선택한 간선을 MST에 추가하고, v를 선택한다.
④ 모든 정점을 선택하지 않았다면, 2번 단계로 돌아간다.

- 각각의 정점을 선택하고 모든 간선을 살펴봐야 한다.
- 우선 순위 큐를 이용하면 최솟값을 O(logE)만에 찾을 수 있다.
- 시간 복잡도 : O(ElogE) : 모든 간선이 힘에 1번씩 들어갔다가 나오기 때문

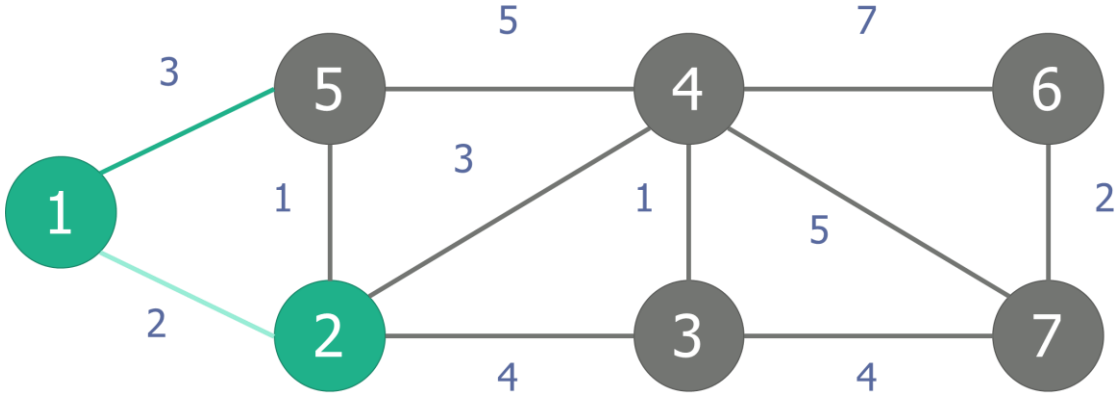
1번 정점 선택



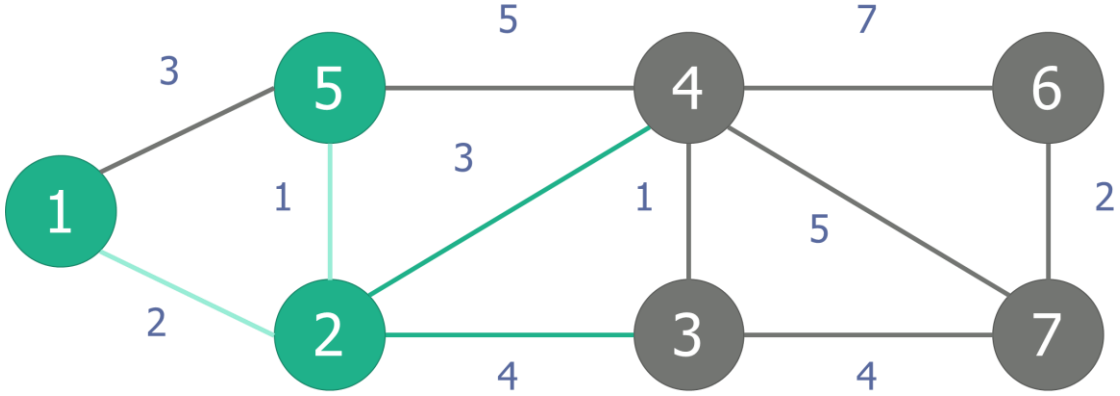
최소 스패닝 트리 - 프림

[Minimum Spanning Tree]

2번 정점 선택 (1 – 2 간선의 가중치 : 2, 1 – 5 간선의 가중치 : 3)



5번 정점 선택 (1 – 5 간선의 가중치 : 3, 2- 5 간선의 가중치 : 1, 2 – 4 간선의 가중치 : 3
2 - 3 간선의 가중치 : 4)

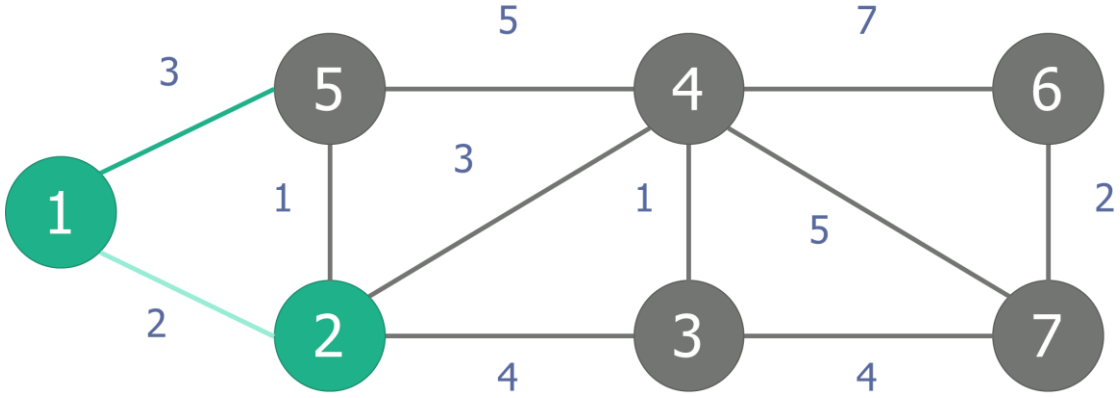


Gaussian

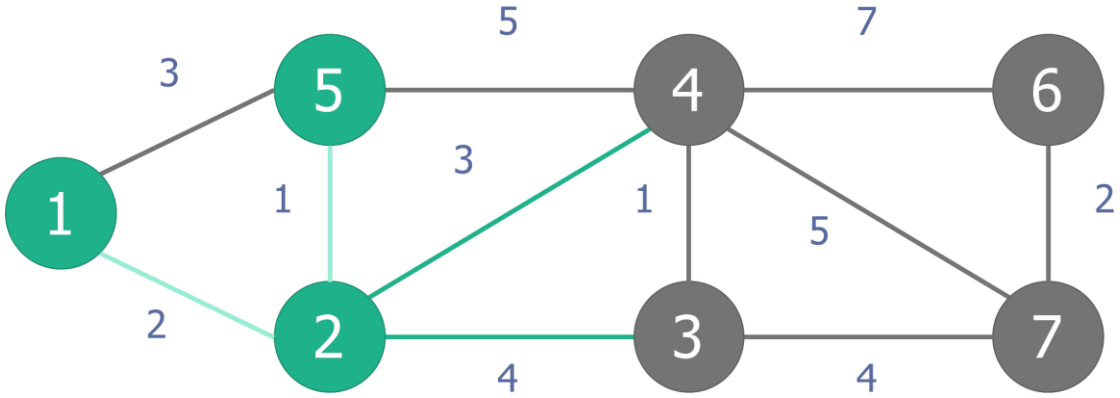
최소 스패닝 트리 - 프림

[Minimum Spanning Tree]

2번 정점 선택 (1 – 2 간선의 가중치 : 2, 1 – 5 간선의 가중치 : 3)



5번 정점 선택 (1 – 5 간선의 가중치 : 3, 2- 5 간선의 가중치 : 1, 2 – 4 간선의 가중치 : 3
2 - 3 간선의 가중치 : 4)

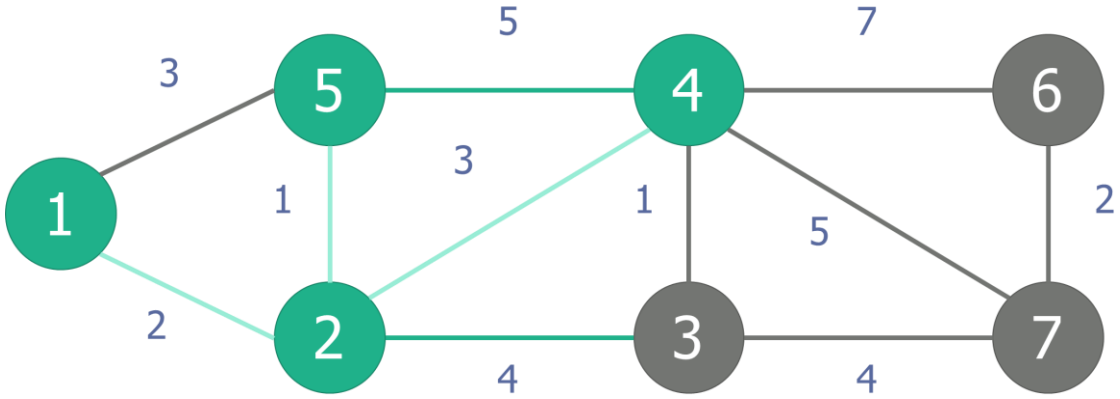


Gaussian

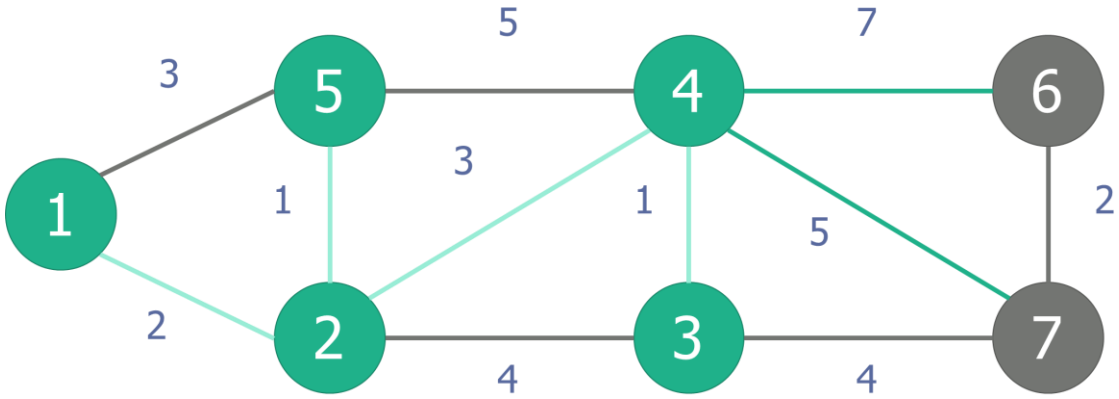
최소 스패닝 트리 - 프림

[Minimum Spanning Tree]

4번 정점 선택 (2 – 4 간선의 가중치가 3으로 가장 작다)



3번 정점 선택 (4 – 3 간선의 가중치가 1로 가장 작다)

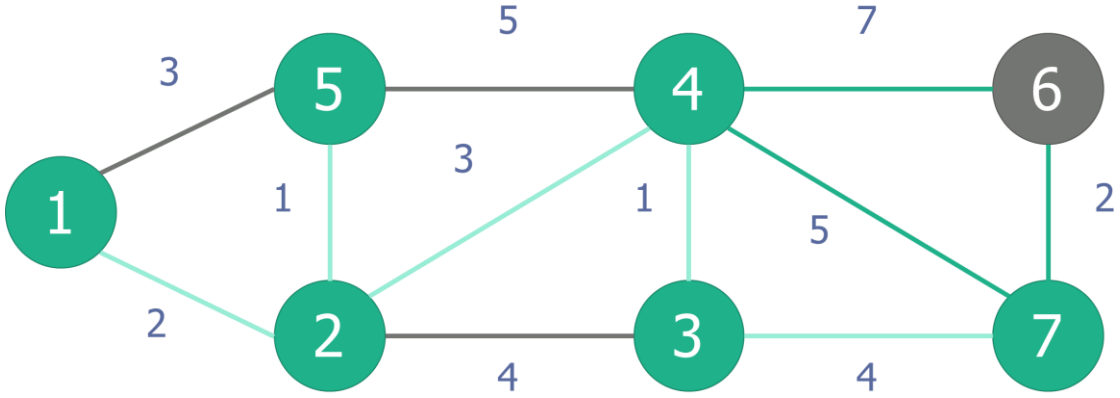


Gaussian

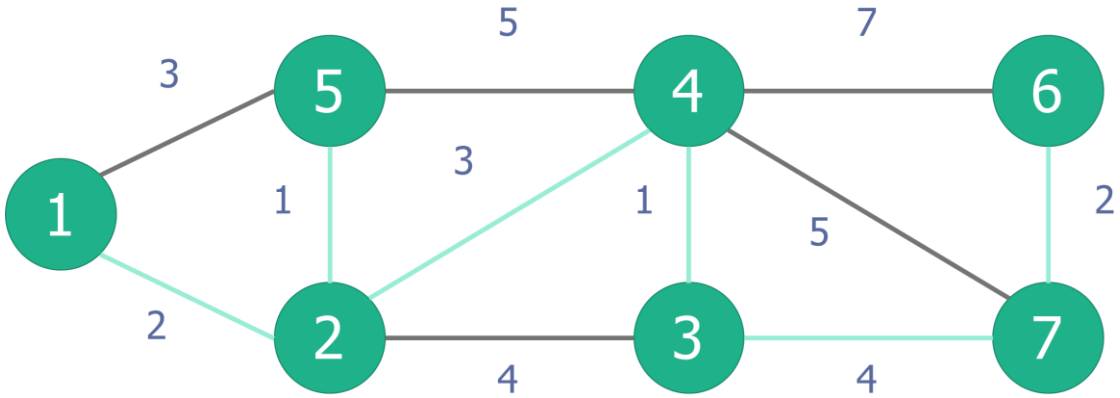
최소 스패닝 트리 - 프림

[Minimum Spanning Tree]

7번 정점 선택 (4 – 7 간선의 가중치가 5로 가장 작다)



4번 정점 선택 (7 -6 간선의 가중치가 2로 가장 작다)



Gaussian

최소 스패닝 트리 - 프림

Minimum Spanning Tree

- 프림 알고리즘을 실제 구현할 때에는 정점의 쌍 (u, v) 2개와 가중치를 모두 저장하지 않는다. 여기서 (u : 방문한 점), (v : 방문하지 않은 점) 중 v와 가중치만 저장 한다.
- minHeap의 우선순위 큐를 생성한 후 (간선 가중치, 다음 정점)을 넣어서 방문하지 않은 다음 정점 중 가중치의 최솟값들을 선택해 나아가면 MST를 얻을 수 있다.

```
priority_queue<pi, vector<pi>, greater<pi>> pq;

visited[1] = true;
for (int i = 0; i < A[1].size(); ++i) {
        int next = A[1][i].first;
        int cost = A[1][i].second;
        pq.push(pi(cost, next));
}
```

```
int ans = 0;
while (!pq.empty()) {
        pi p = pq.top();
        int here = p.second;
        int cost = p.first;
        pq.pop();

        if (visited[here]) continue;
        visited[here] = true;

        ans += cost;
        for (int i = 0; i < A[here].size(); ++i) {
                int next = A[here][i].first;
                int next_cost = A[here][i].second;
                pq.push(pi(next_cost, next));
        }
}
```