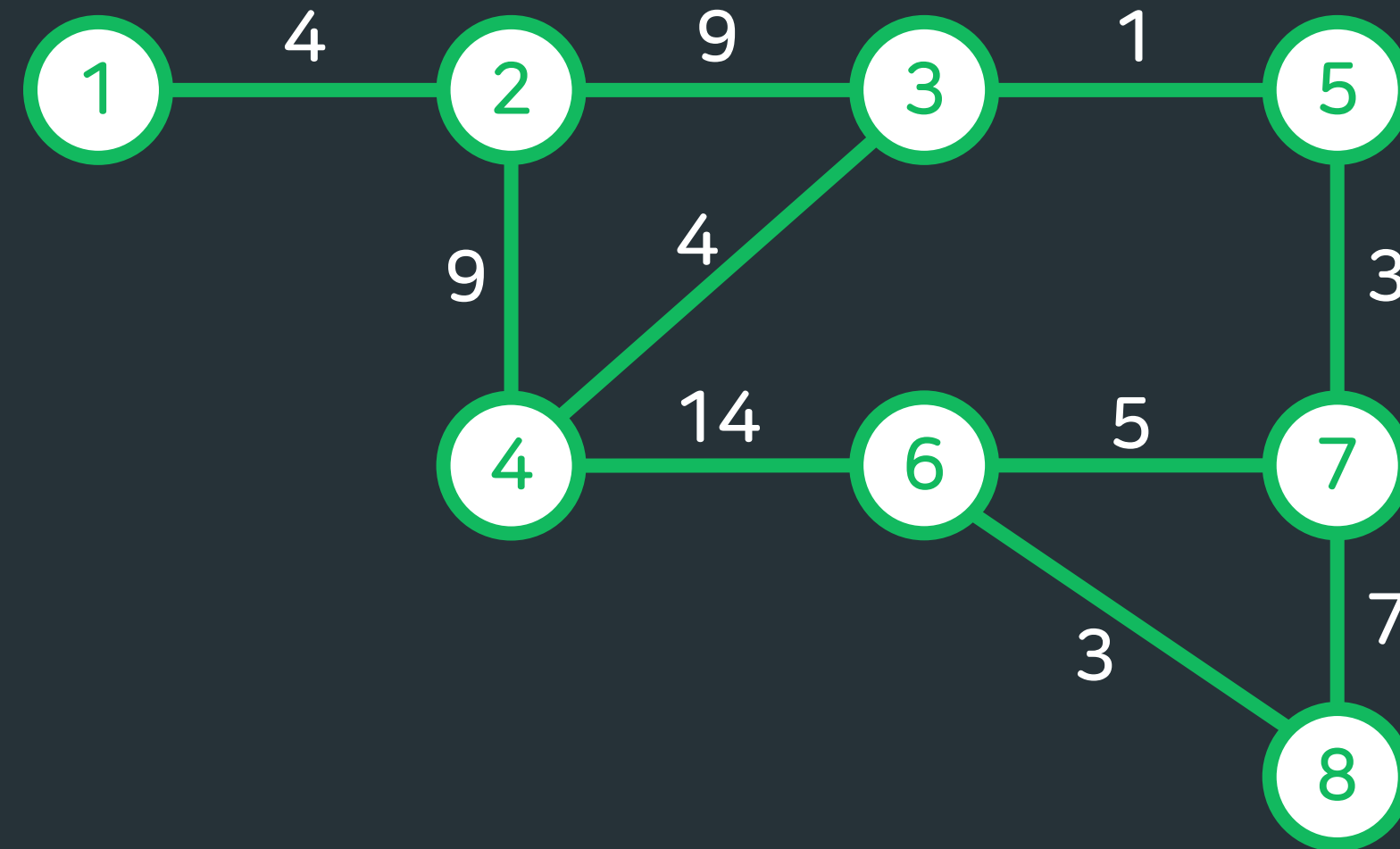


# 알튜비튜

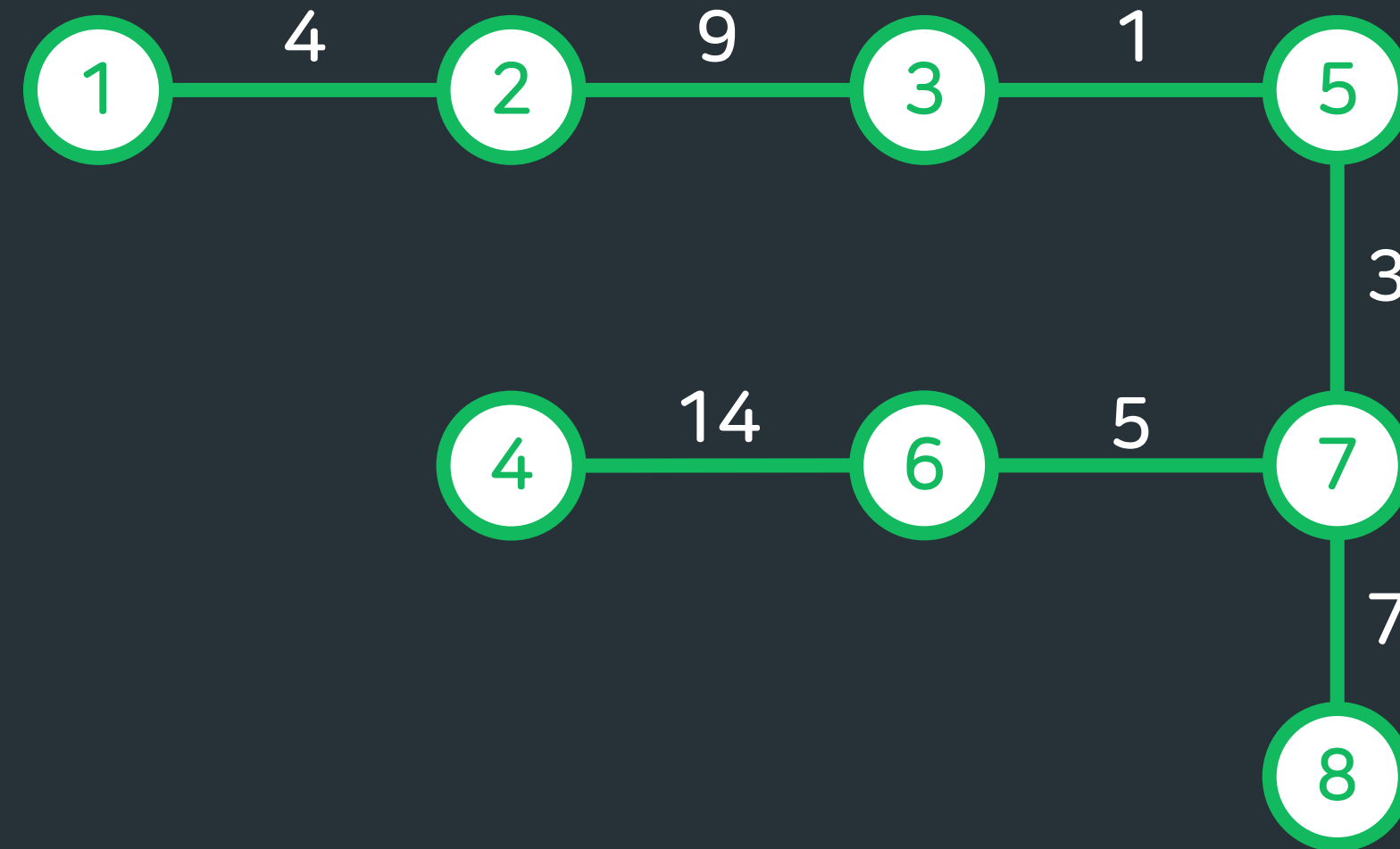
## 최소 신장 트리

하나의 그래프에서 트리를 만들 수 있는 방법은 많습니다.  
그 중 간선의 가중치 합이 가장 작은 트리는 어떻게 구할까요?  
크루스칼, 프림 알고리즘으로 대표할 수 있는 최소 신장 트리 문제입니다.

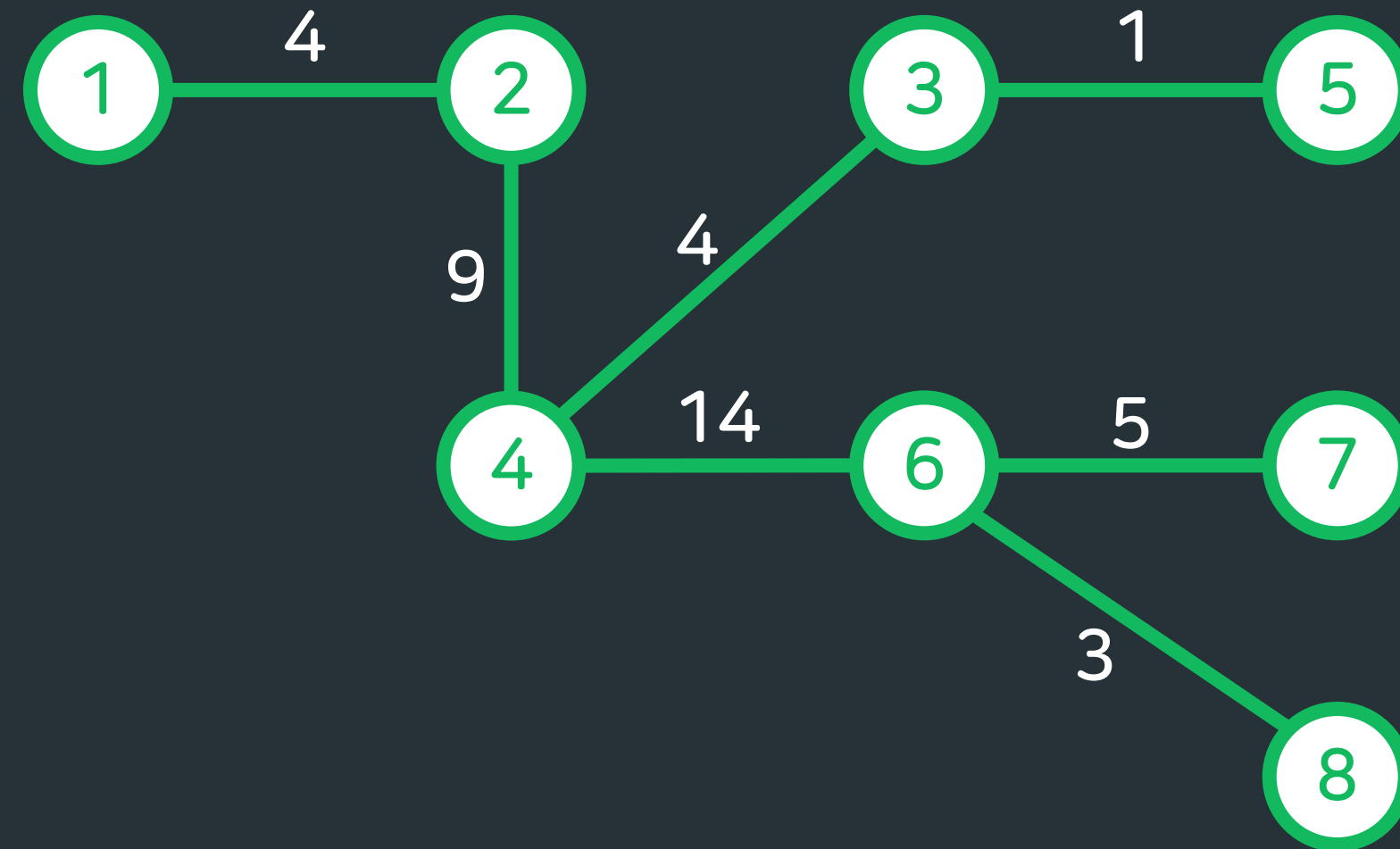
# 그래프에서 트리 만들기?



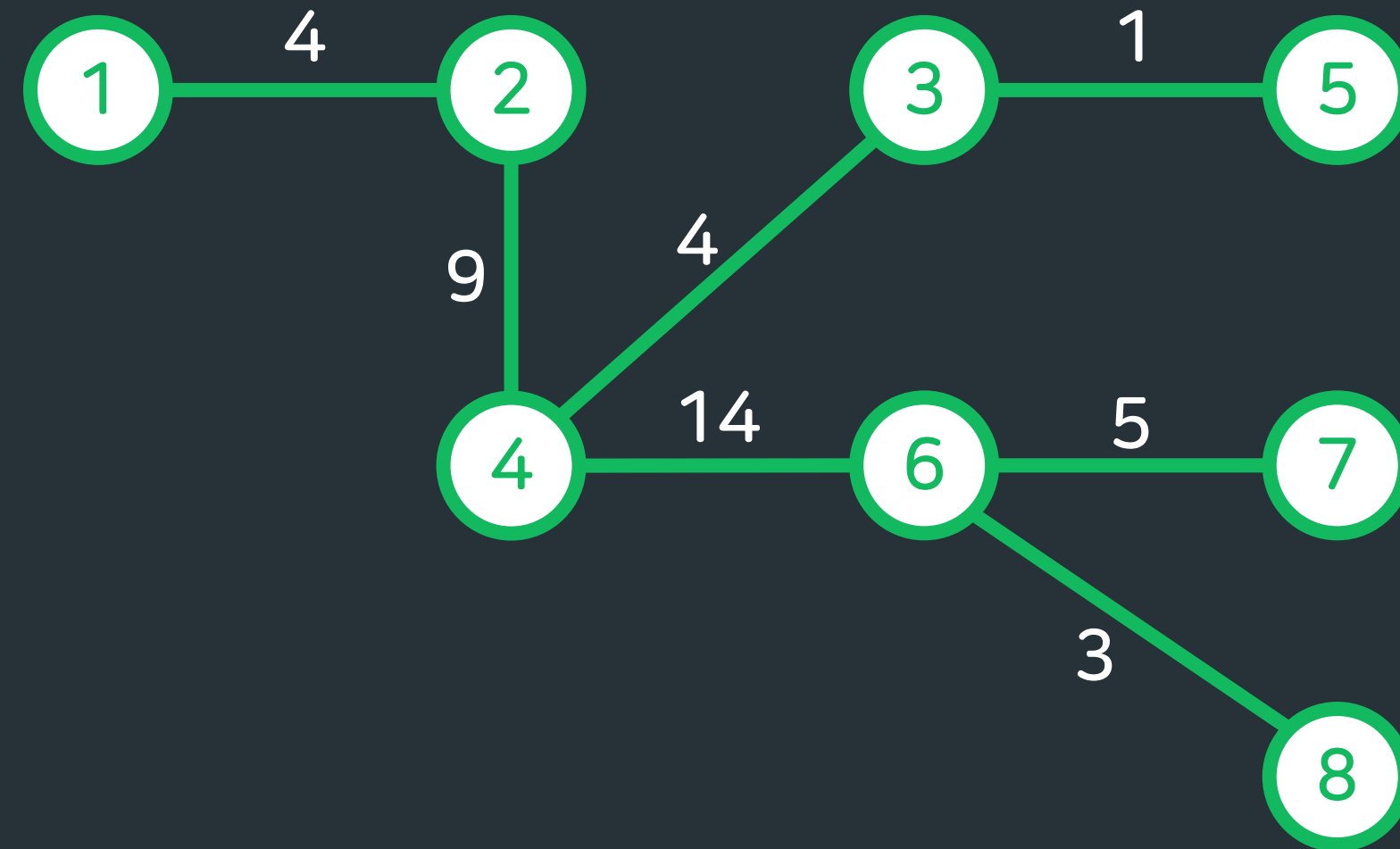
# 그래프에서 트리 만들기?



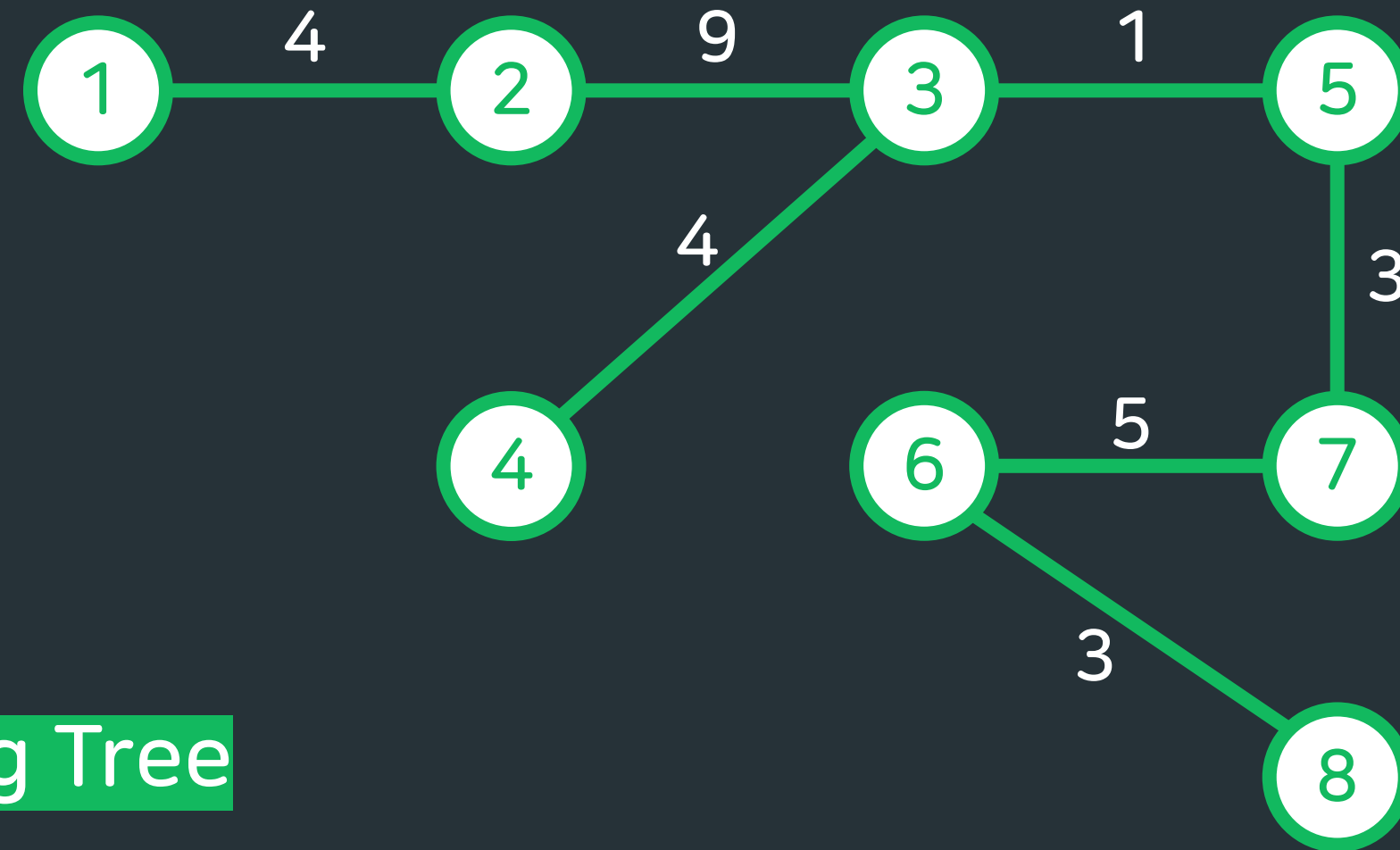
# 그래프에서 트리 만들기?



# 그래프에서 트리 만들기?



가중치의 합이 가장 작은 트리는?

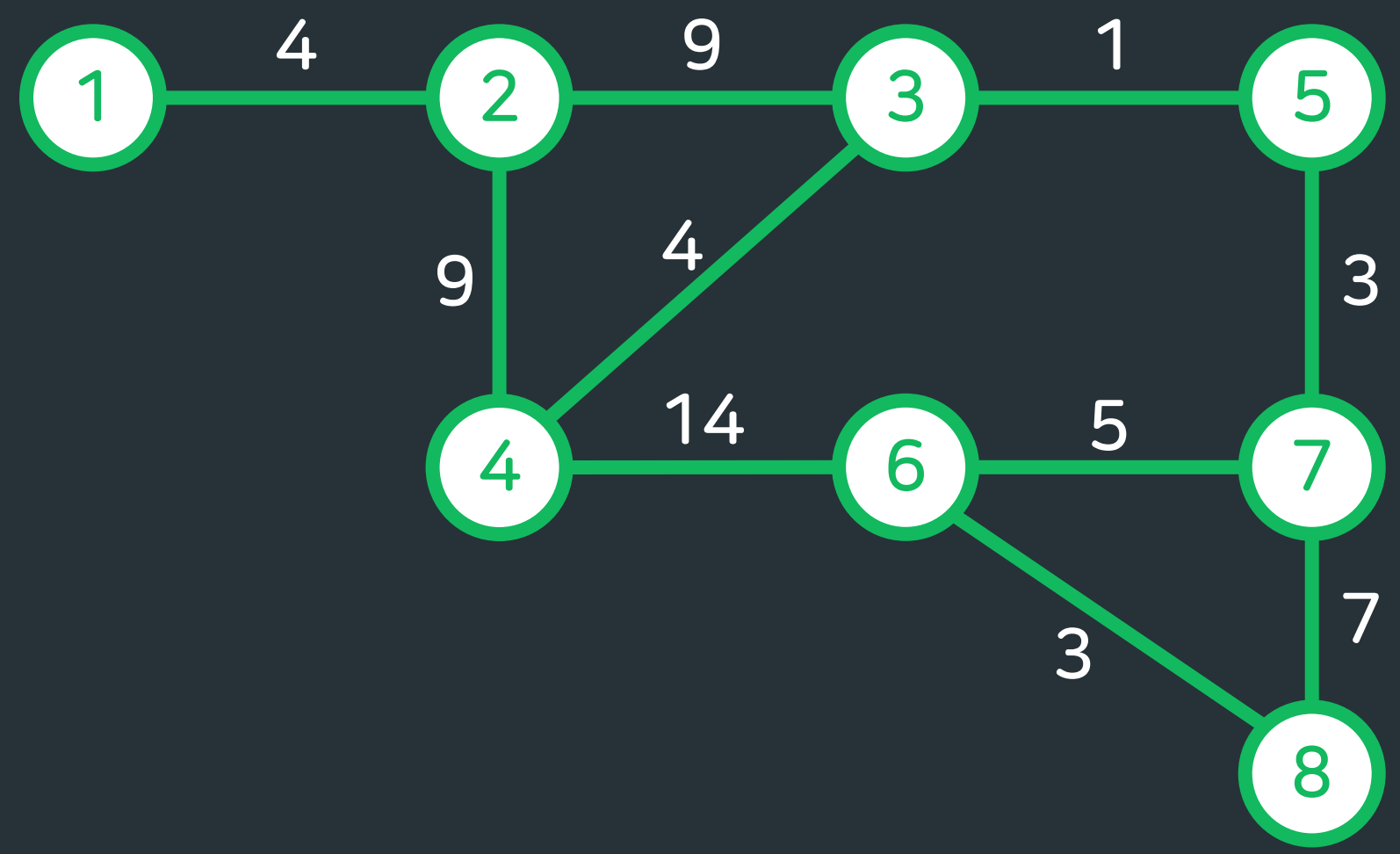


## Minimum Spanning Tree

- 하나의 그래프로 만들 수 있는 트리들을 신장 트리(Spanning Tree)라고 부름
- 신장 트리 중 간선의 **가중치 합**이 **가장 작은 트리**가 최소 신장 트리
- MST를 구하는 알고리즘으로는 **크루스칼**, **프림**이 있음

## Kruskal

- 유니온 파인드 알고리즘을 이용해 MST를 구하는 알고리즘
- 유니온 파인드에서 같은 집합이라면 사이클이 발생한다는 점을 이용
- 가중치가 가장 작은 간선부터 선택하며 사이클이 발생하지 않는다면 트리에 포함
- 유니온 파인드의 시간 복잡도가  $O(1)$ 에 가깝기 때문에 간선을 정렬하는 시간 복잡도만 고려
- 간선이 많지 않을 때 주로 사용
- 간선의 수를  $E$ 라고 할 때, 시간 복잡도는  $O(E \log E)$



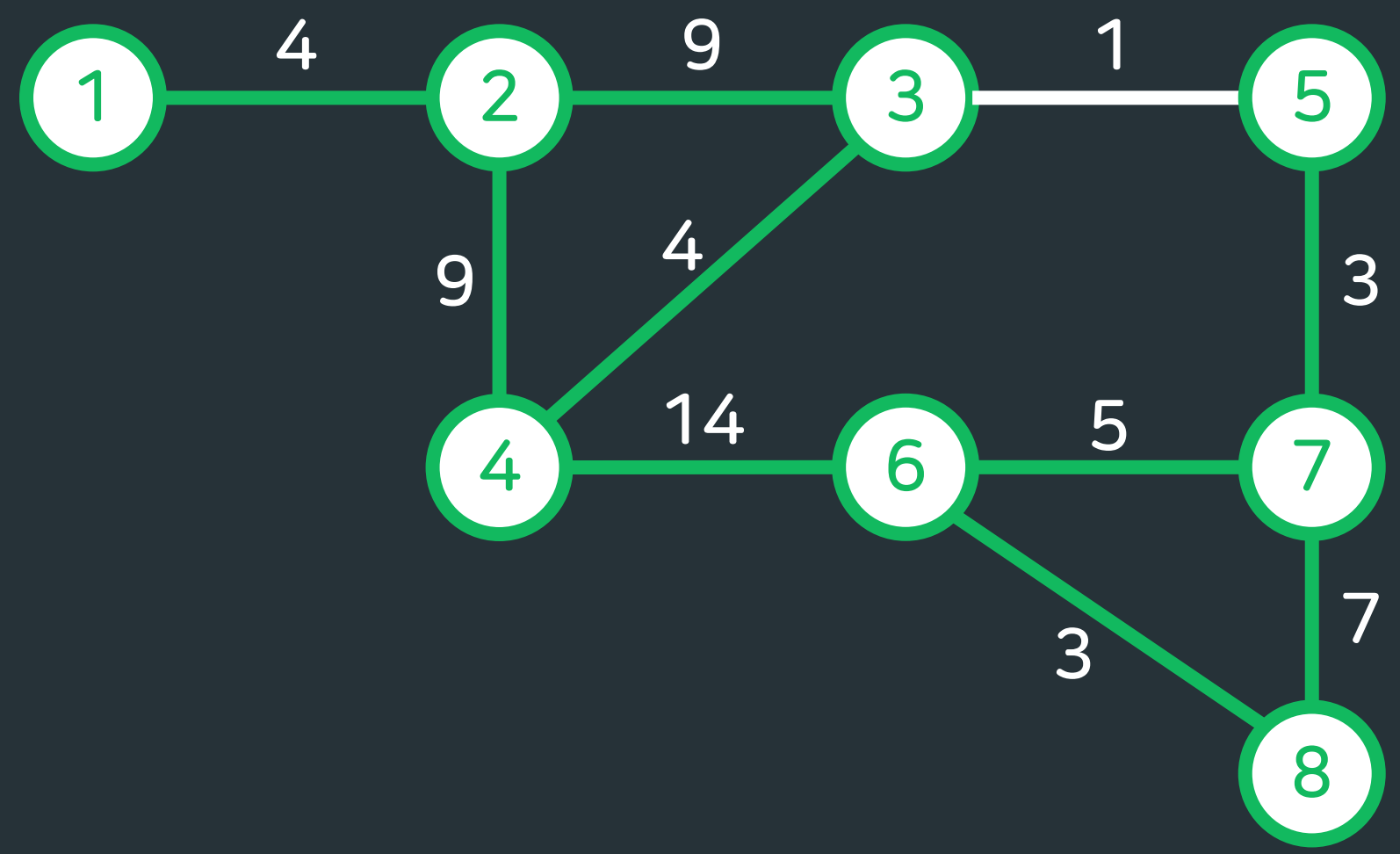
3-5 6-8 5-7 3-4 1-2 6-7 7-8 2-3 2-4 4-6

Weight	1	3	3	4	4	5	7	9	9	14
--------	---	---	---	---	---	---	---	---	---	----

1 2 3 4 5 6 7 8

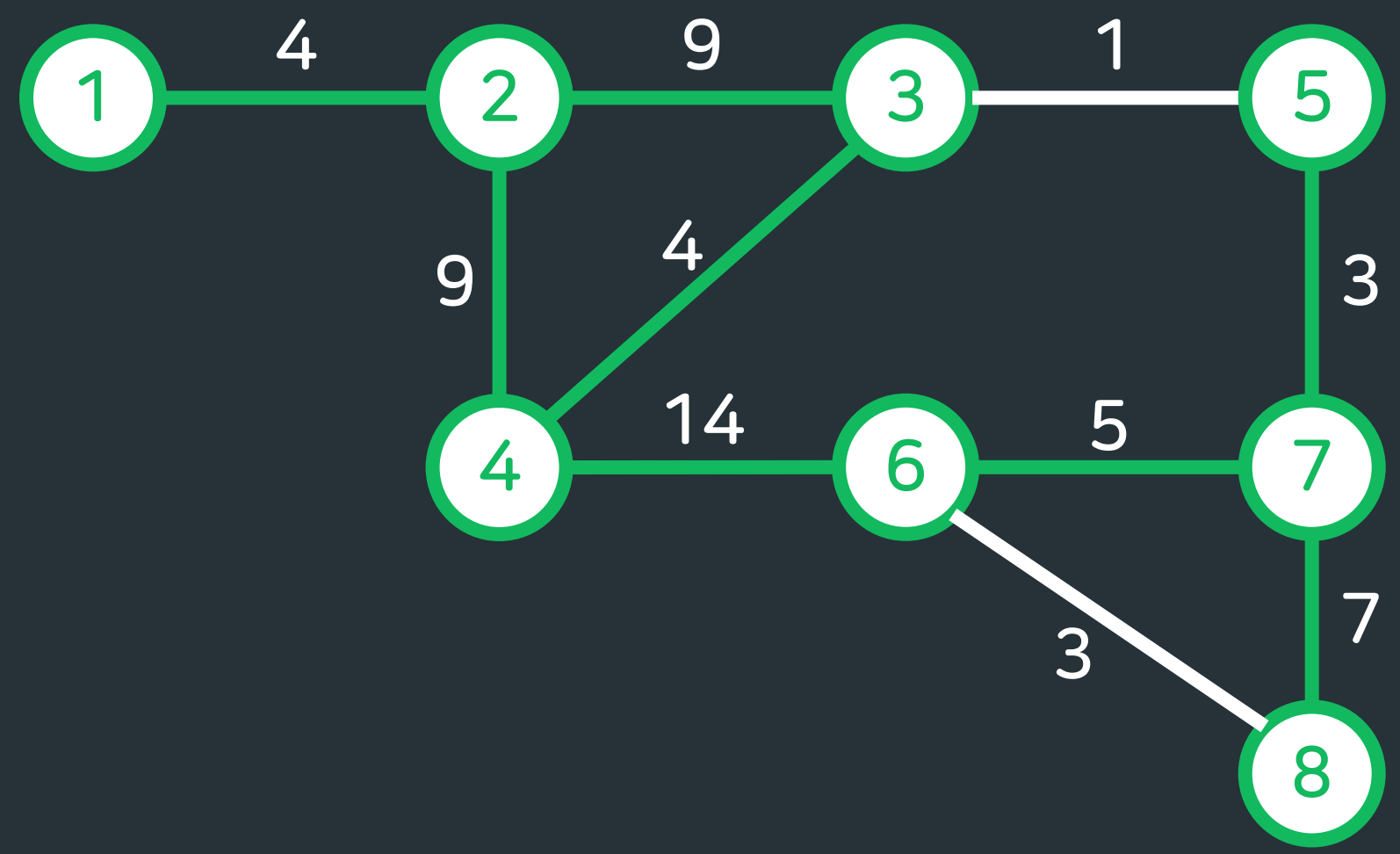
Parent	-1	-1	-1	-1	-1	-1	-1	-1		
--------	----	----	----	----	----	----	----	----	--	--





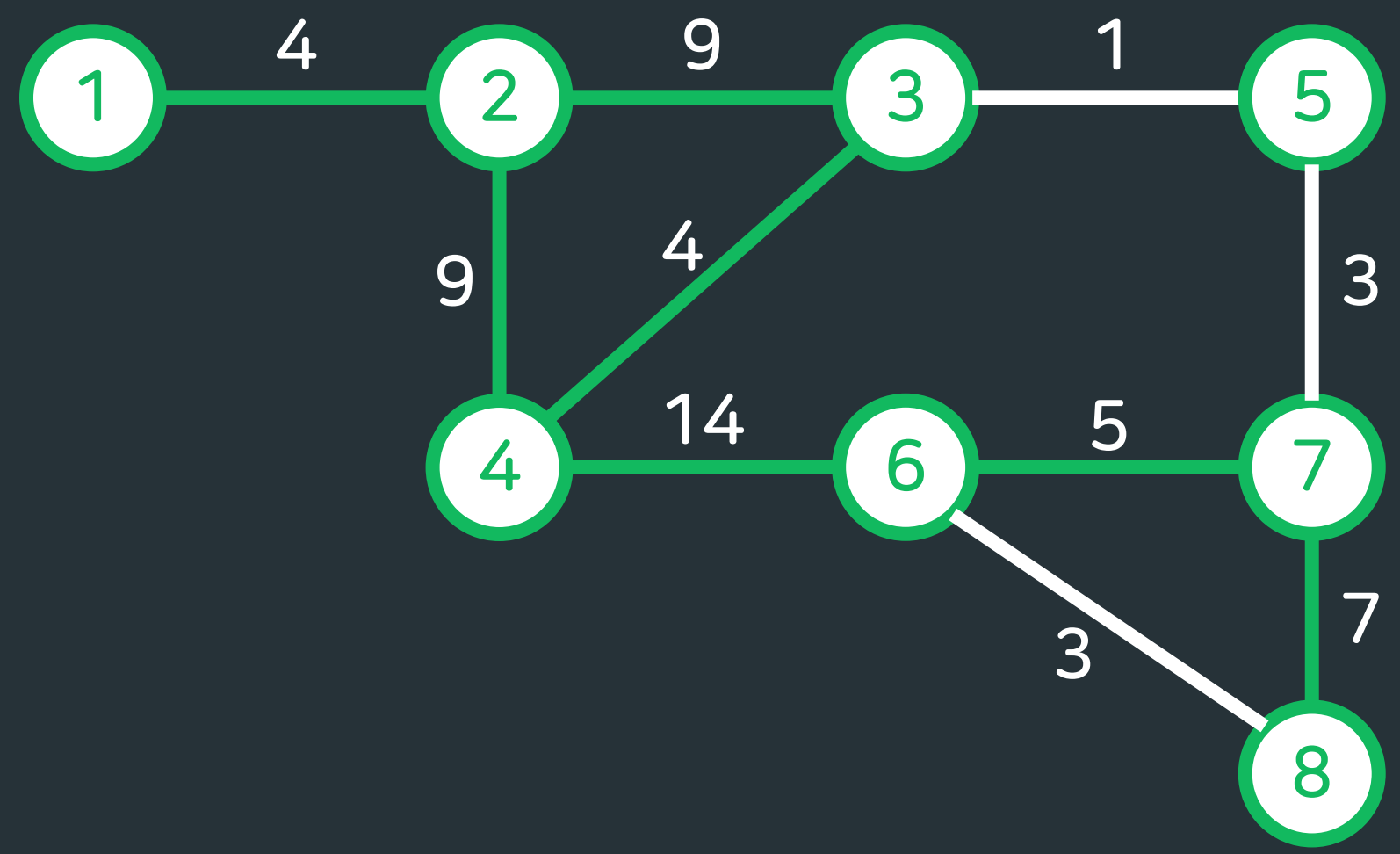
3-5 6-8 5-7 3-4 1-2 6-7 7-8 2-3 2-4 4-6

Weight	1	3	3	4	4	5	7	9	9	14
	1	2	3	4	5	6	7	8		
Parent	-1	-1	-2	-1	3	-1	-1	-1		



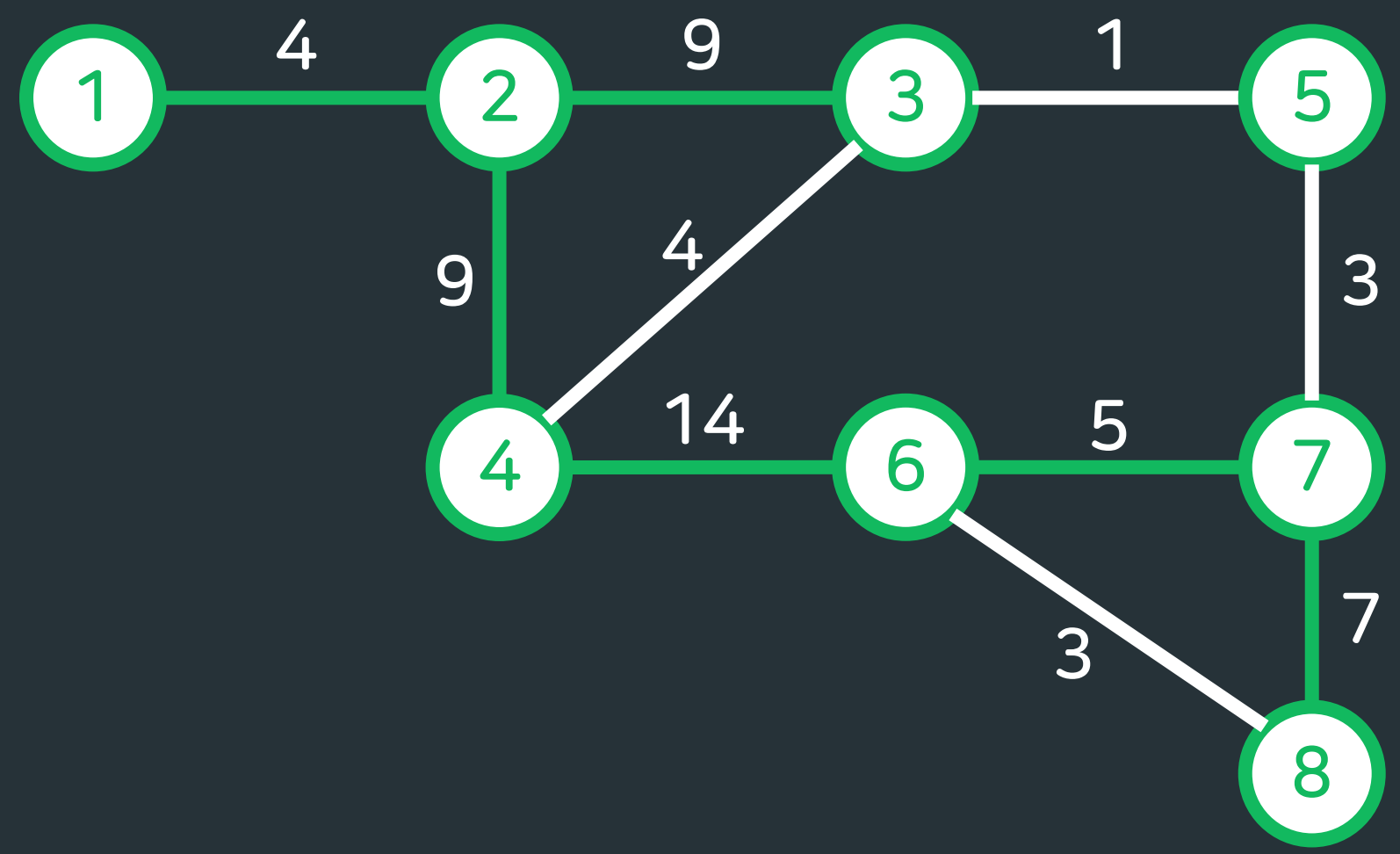
3-5 6-8 5-7 3-4 1-2 6-7 7-8 2-3 2-4 4-6

Weight	1	3	3	4	4	5	7	9	9	14
	1	2	3	4	5	6	7	8		
Parent	-1	-1	-2	-1	3	-2	-1	6		



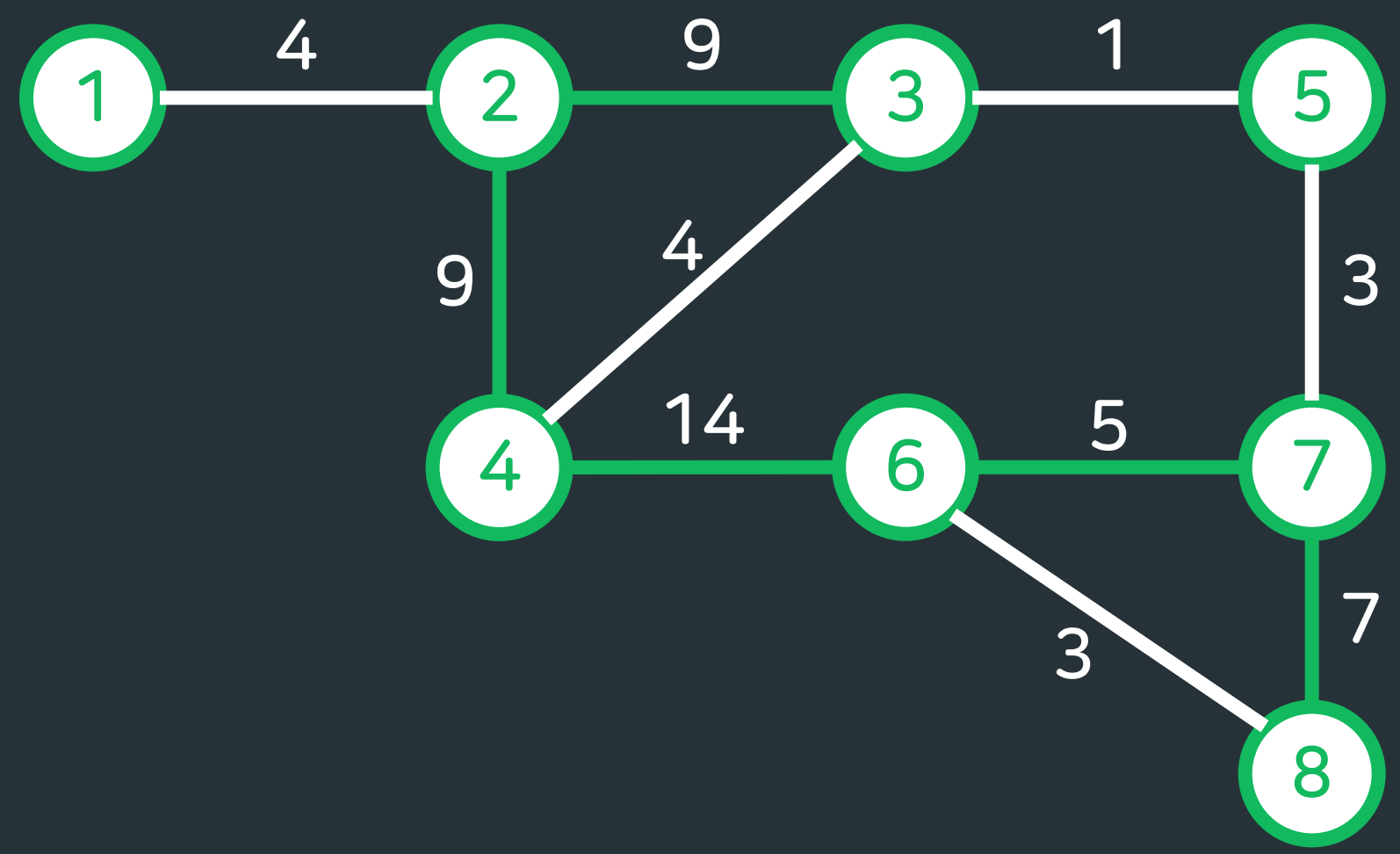
3-5 6-8 5-7 3-4 1-2 6-7 7-8 2-3 2-4 4-6

Weight	1	3	3	4	4	5	7	9	9	14
	1	2	3	4	5	6	7	8		
Parent	-1	-1	-3	-1	3	-2	3	6		



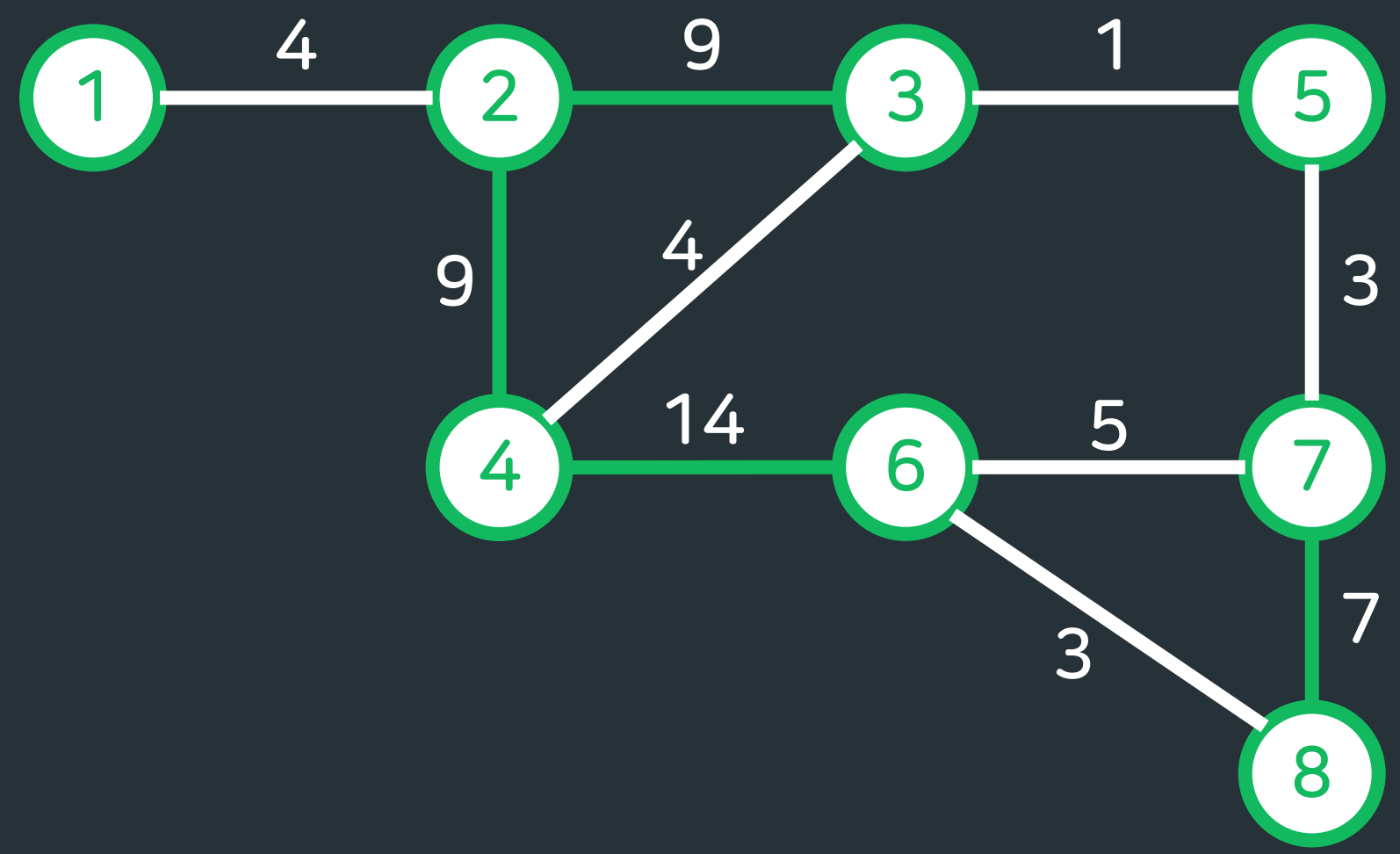
3-5 6-8 5-7 3-4 1-2 6-7 7-8 2-3 2-4 4-6

Weight	1	3	3	4	4	5	7	9	9	14
	1	2	3	4	5	6	7	8		
Parent	-1	-1	-4	3	3	-2	3	6		



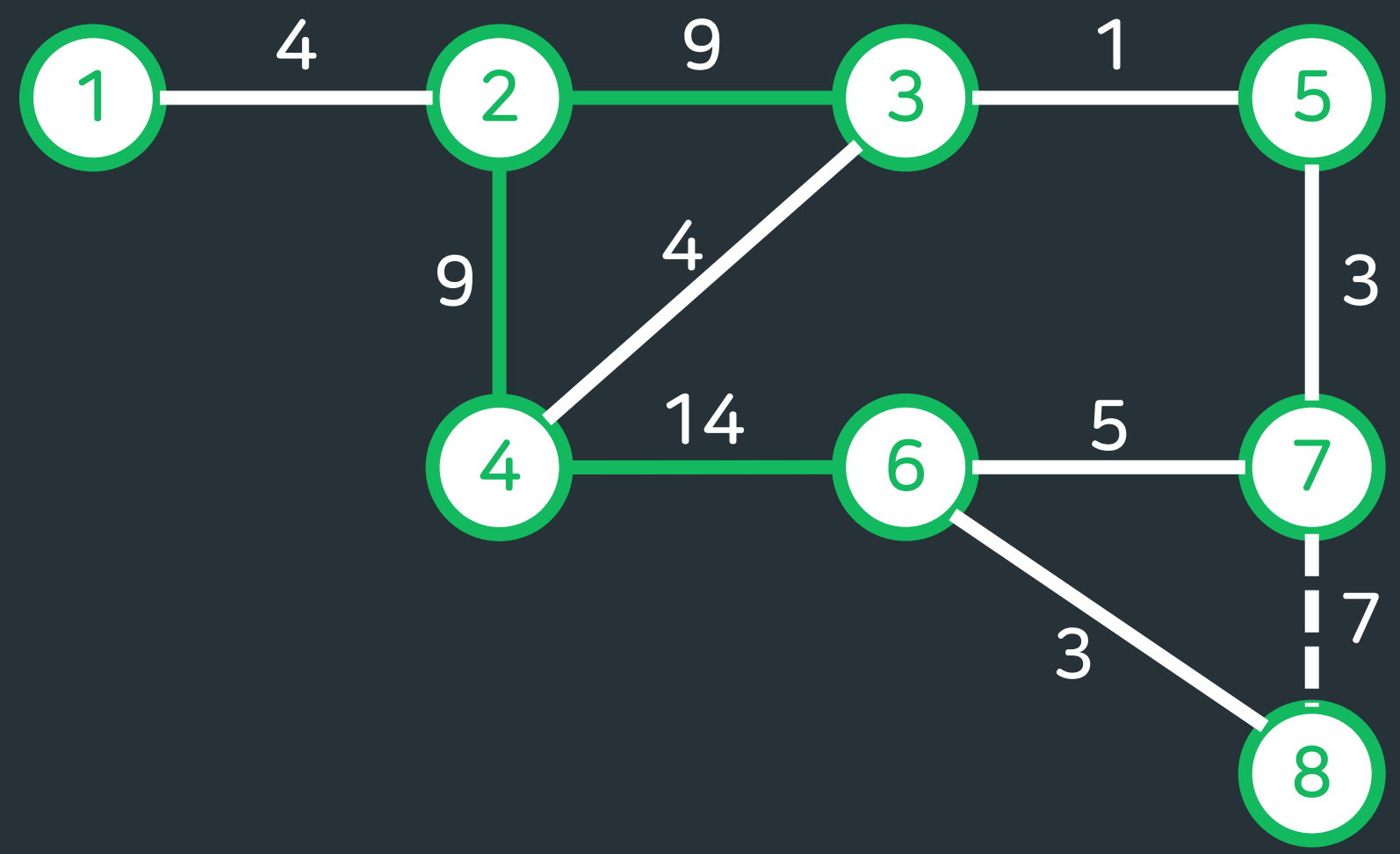
3-5 6-8 5-7 3-4 1-2 6-7 7-8 2-3 2-4 4-6

Weight	1	3	3	4	4	5	7	9	9	14
	1	2	3	4	5	6	7	8		
Parent	-2	1	-4	3	3	-2	3	6		



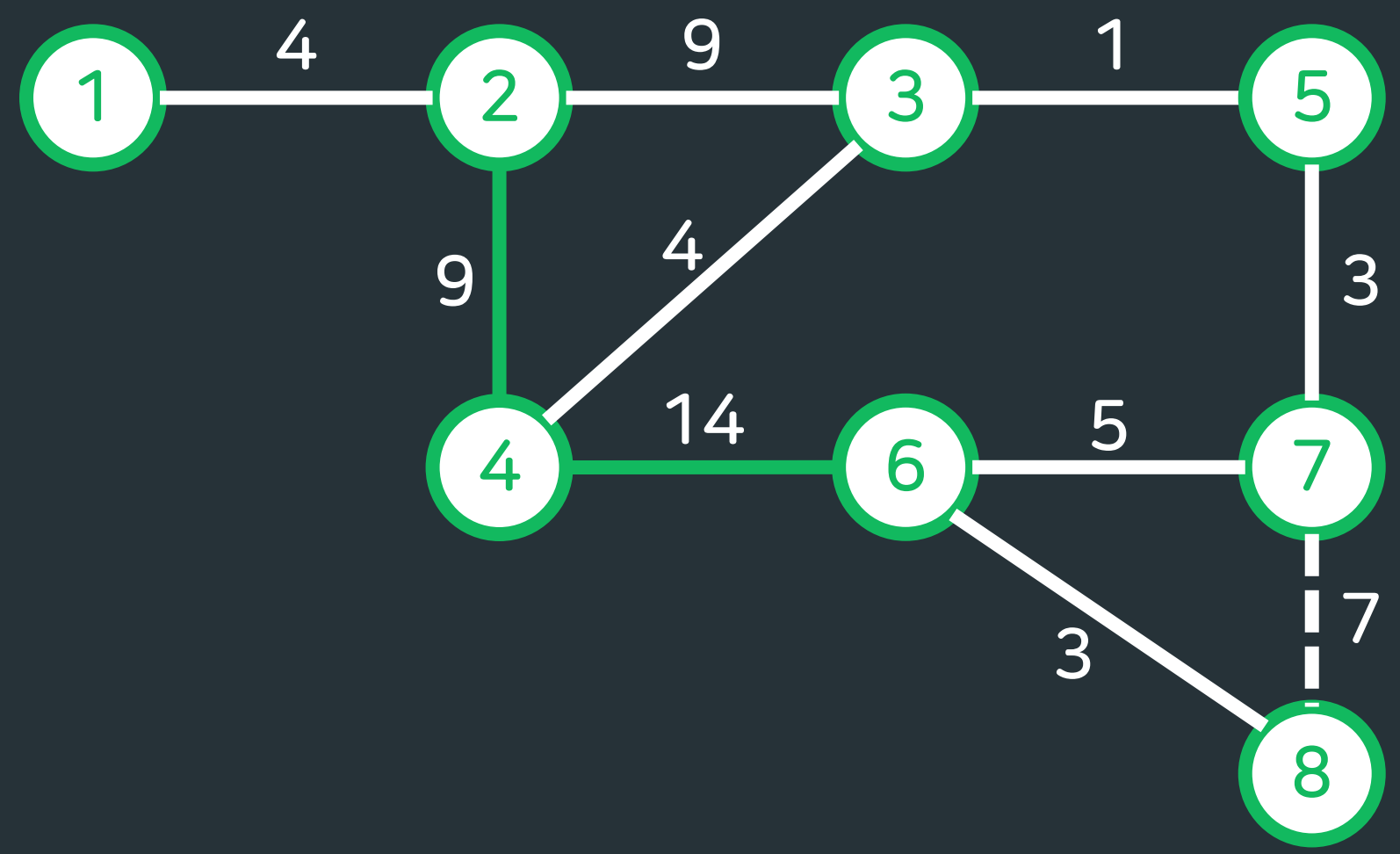
3-5 6-8 5-7 3-4 1-2 6-7 7-8 2-3 2-4 4-6

Weight	1	3	3	4	4	5	7	9	9	14
	1	2	3	4	5	6	7	8		
Parent	-2	1	-6	3	3	3	3	3		



3-5 6-8 5-7 3-4 1-2 6-7 7-8 2-3 2-4 4-6

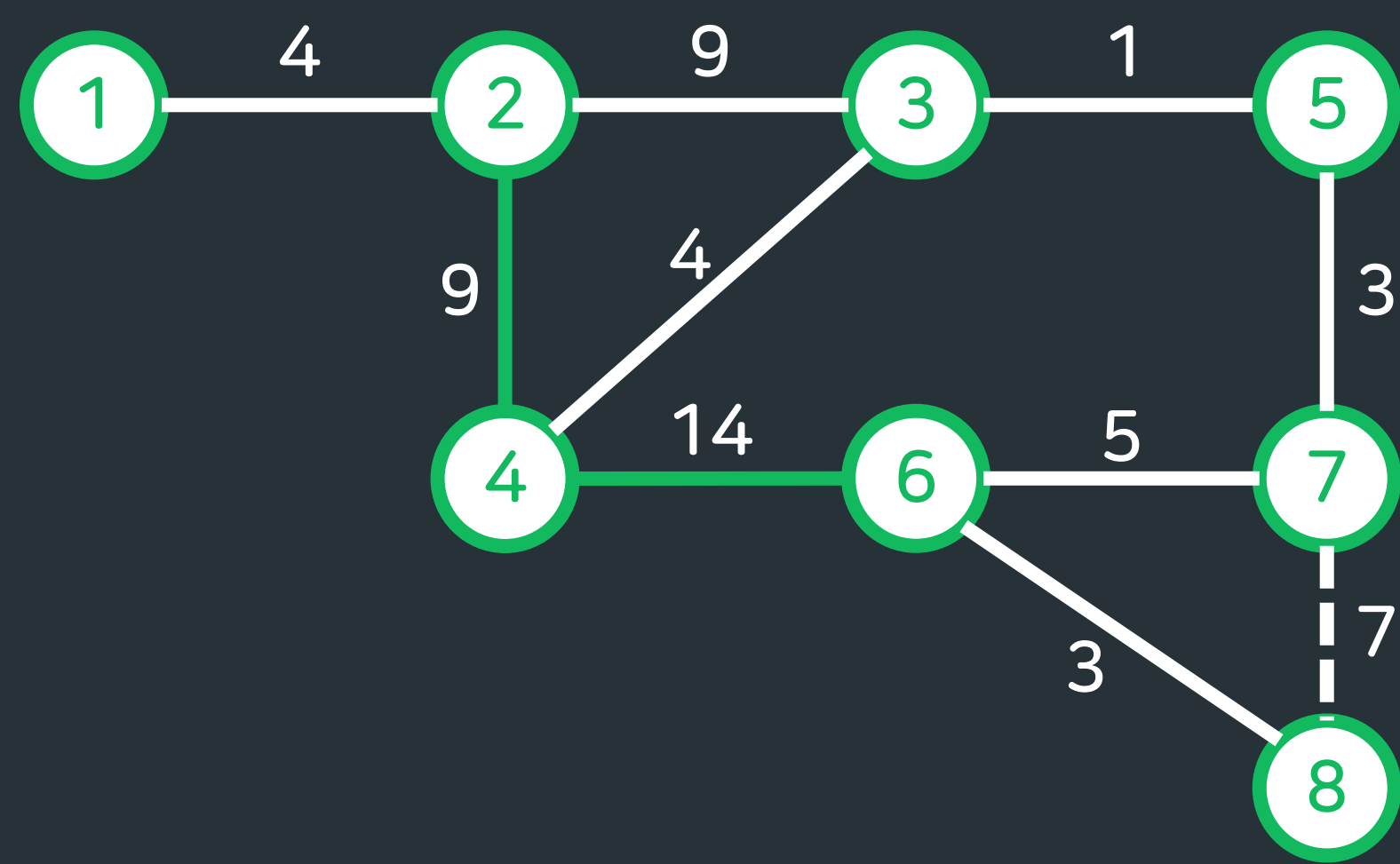
Weight	1	3	3	4	4	5	7	9	9	14
	1	2	3	4	5	6	7	8		
Parent	-2	1	-6	3	3	3	3	3		



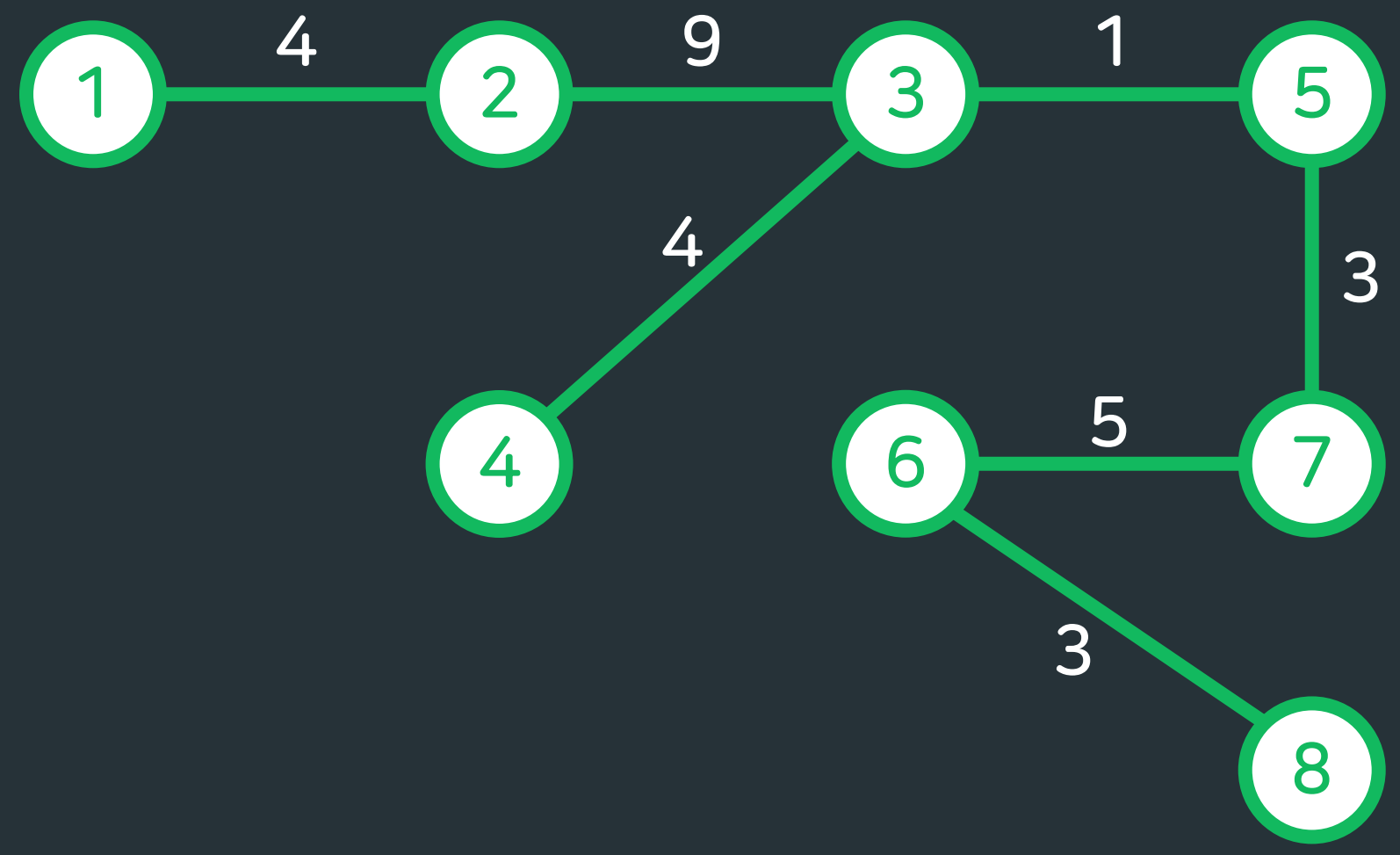
3-5 6-8 5-7 3-4 1-2 6-7 7-8 2-3 2-4 4-6

Weight	1	3	3	4	4	5	7	9	9	14
	1	2	3	4	5	6	7	8		
Parent	3	3	-8	3	3	3	3	3		





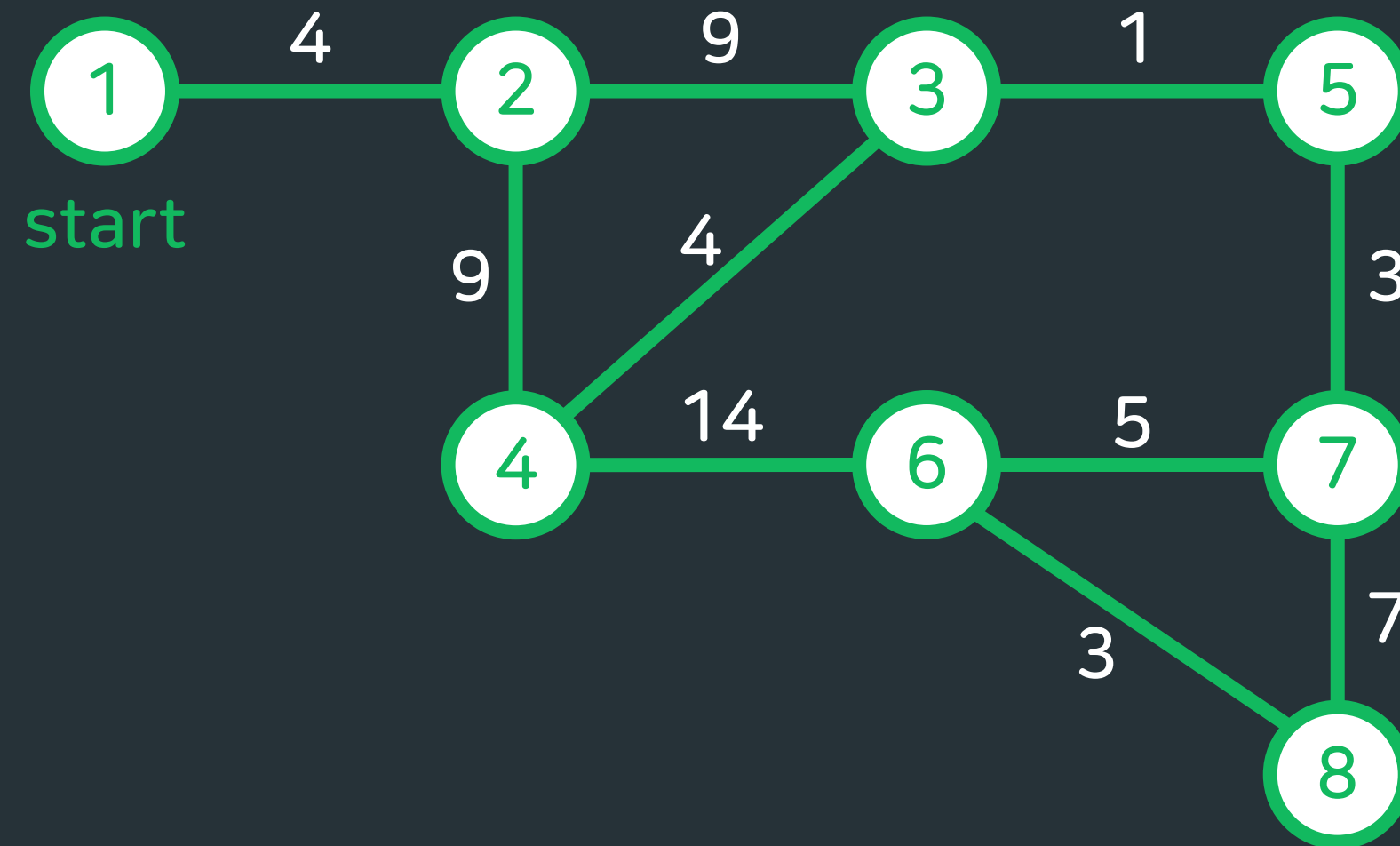
트리를 만들기 위해 필요한 간선의 수  $V-1$ 개를 모두 고르면  
break!



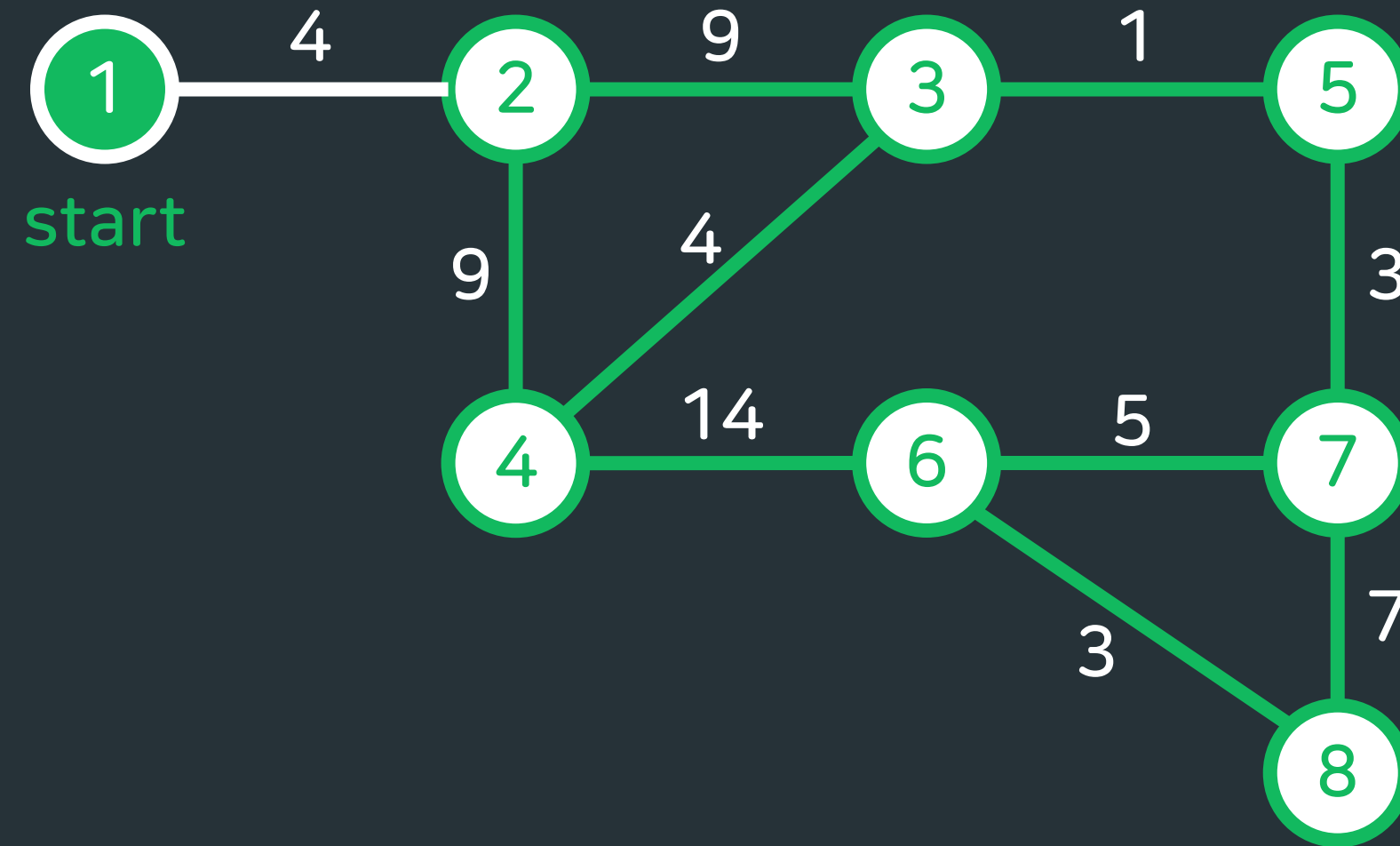
## Prim

- 특정 정점에서 시작하여 접근할 수 있는 정점 중 가중치가 가장 작은 정점을 우선으로 접근
- 시작점으로부터의 누적 거리를 고려한 다익스트라와 달리 간선 자체의 가중치만 고려
- 시작점이 특별하게 주어진 경우에 주로 사용
- 시간 복잡도는 다익스트라와 같은  $O(V \log V + E \log V)$

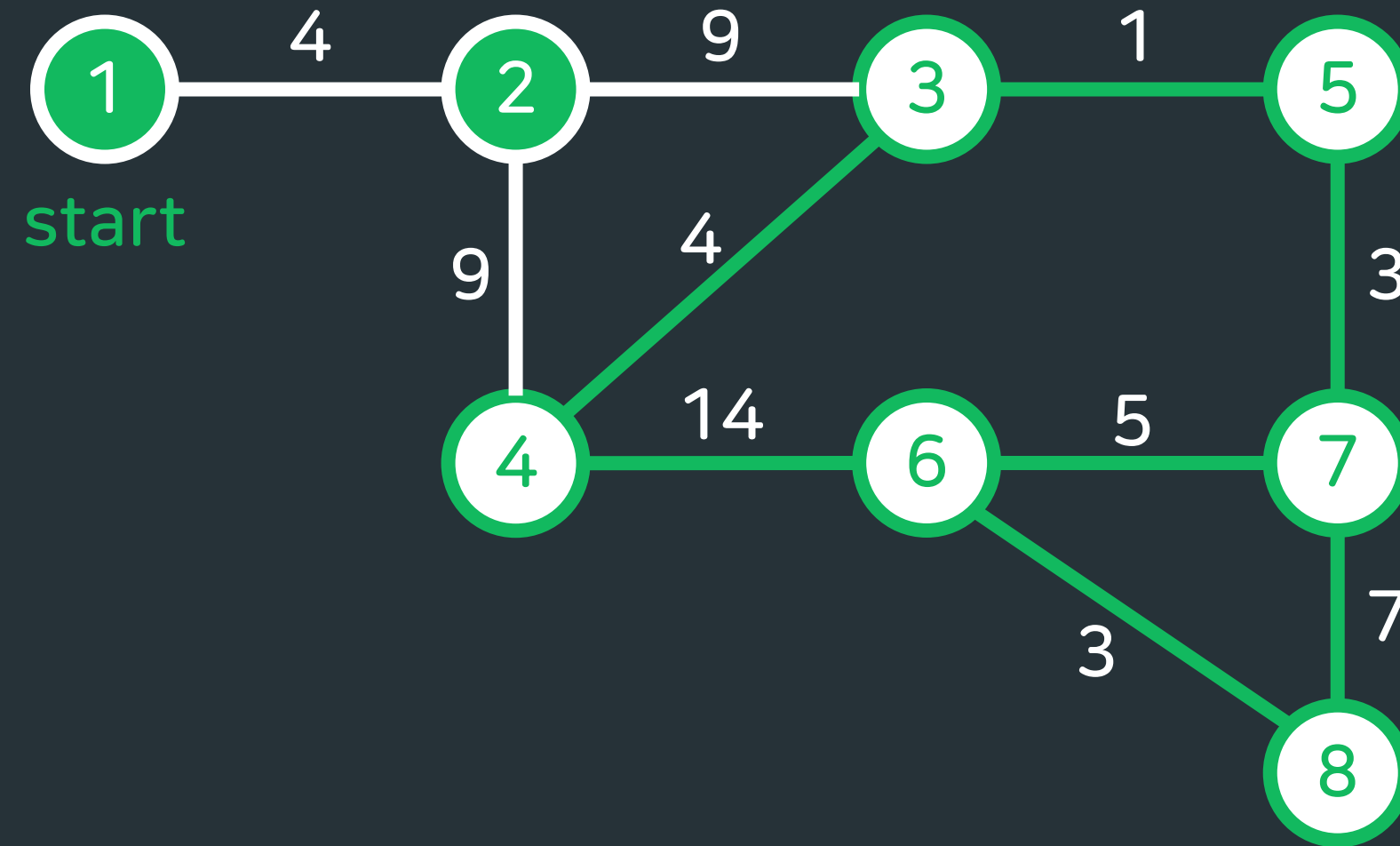
\*시간 복잡도 구하는 과정은 다익스트라와 동일하니 최단 경로 ppt를 참고해주세요



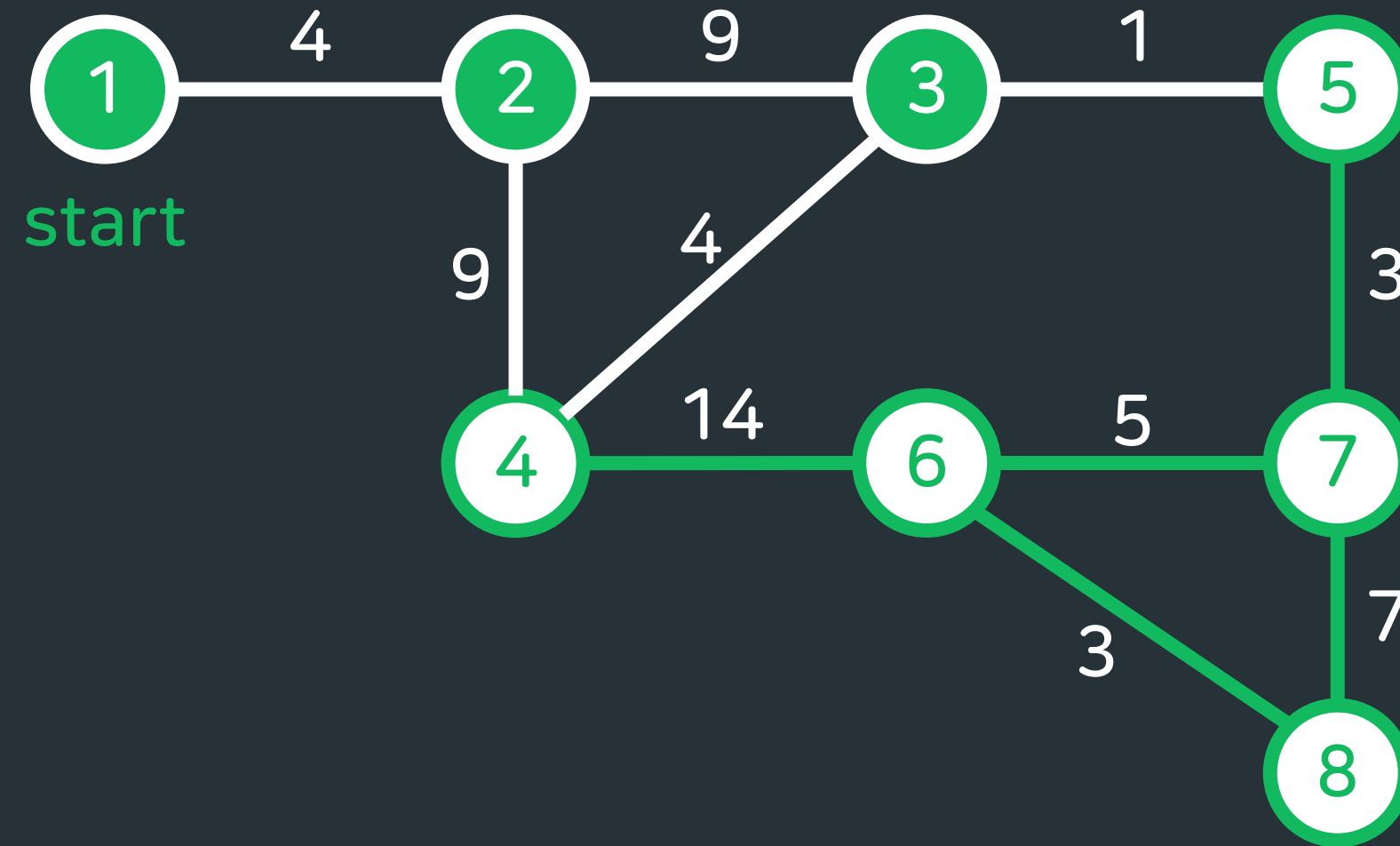
1	2	3	4	5	6	7	8
0	INF	INF	INF	INF	INF	INF	INF



1	2	3	4	5	6	7	8
0	4	INF	INF	INF	INF	INF	INF

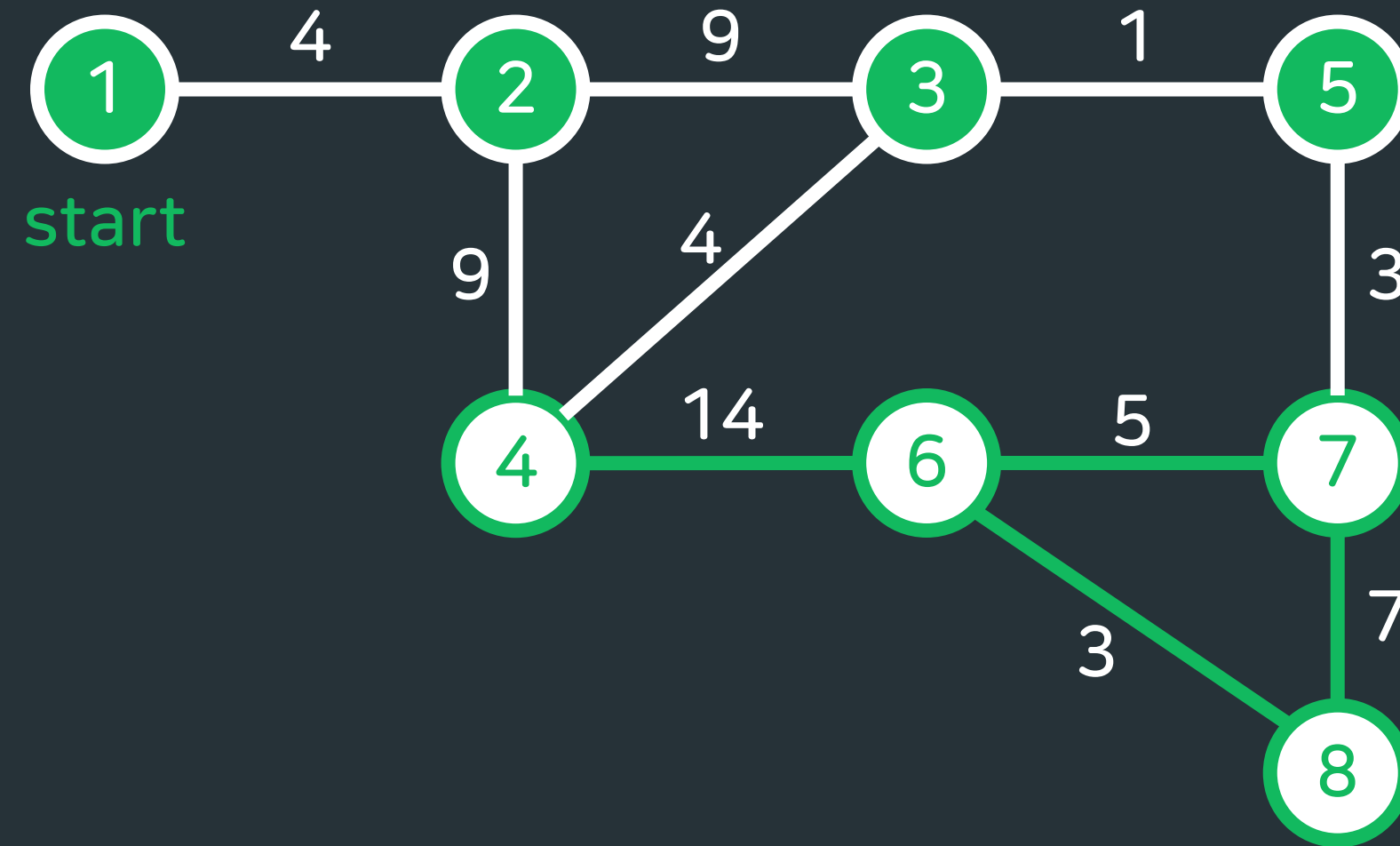


1	2	3	4	5	6	7	8
0	4	9	9	INF	INF	INF	INF



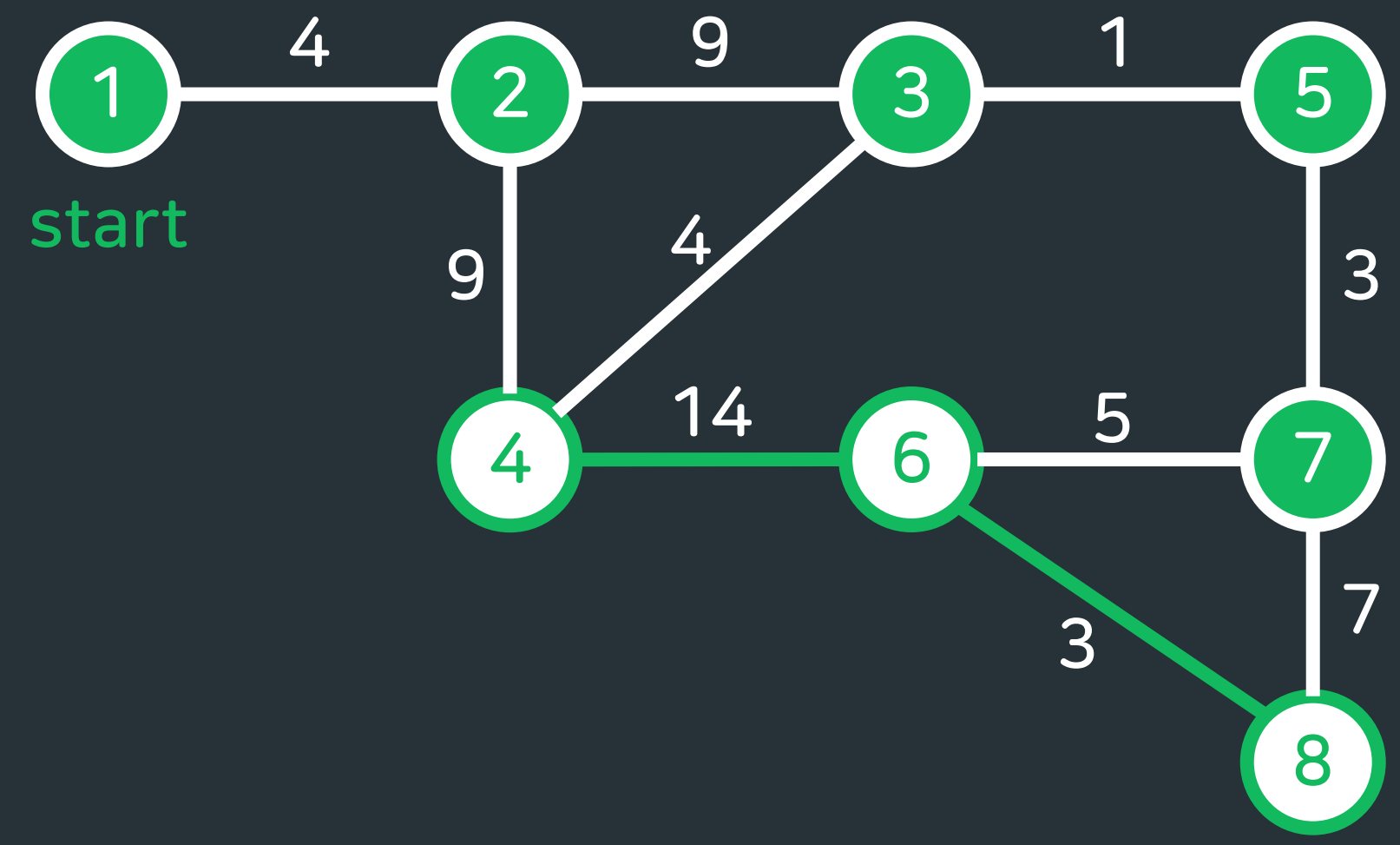
1	2	3	4	5	6	7	8
0	4	9	4	1	INF	INF	INF

$$4 < 9$$

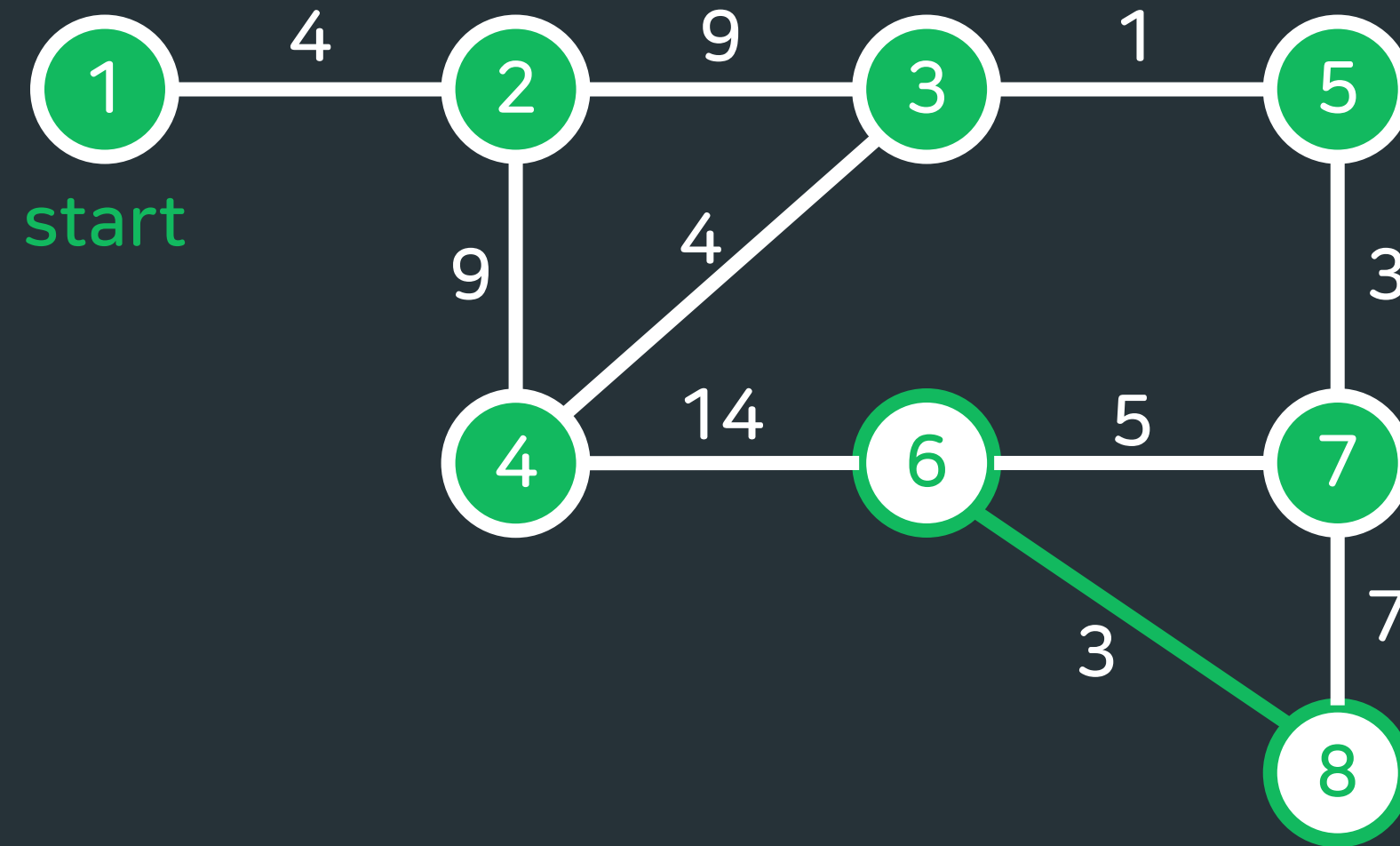


1	2	3	4	5	6	7	8
0	4	9	4	1	INF	3	INF



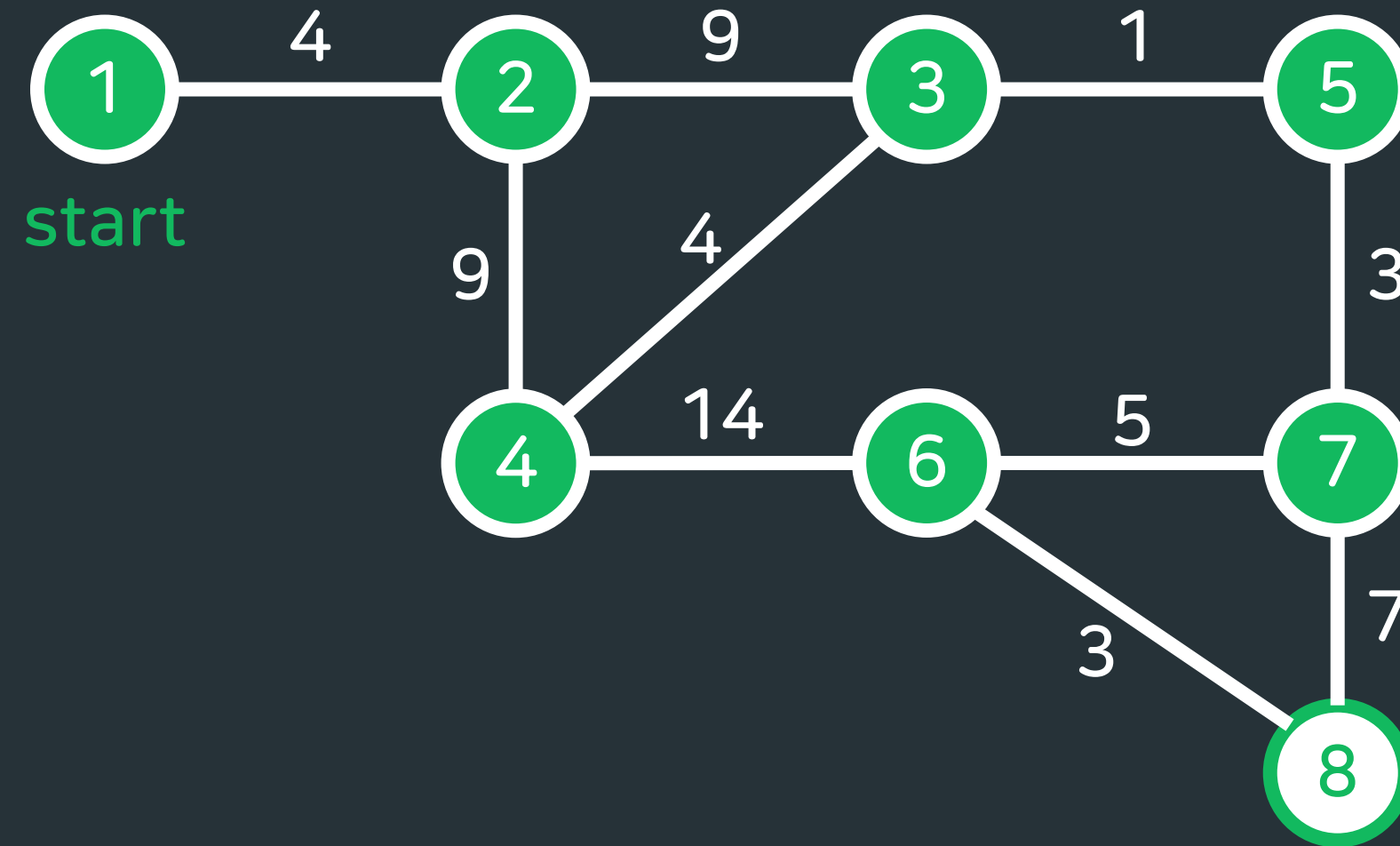


1	2	3	4	5	6	7	8
0	4	9	4	1	5	3	7



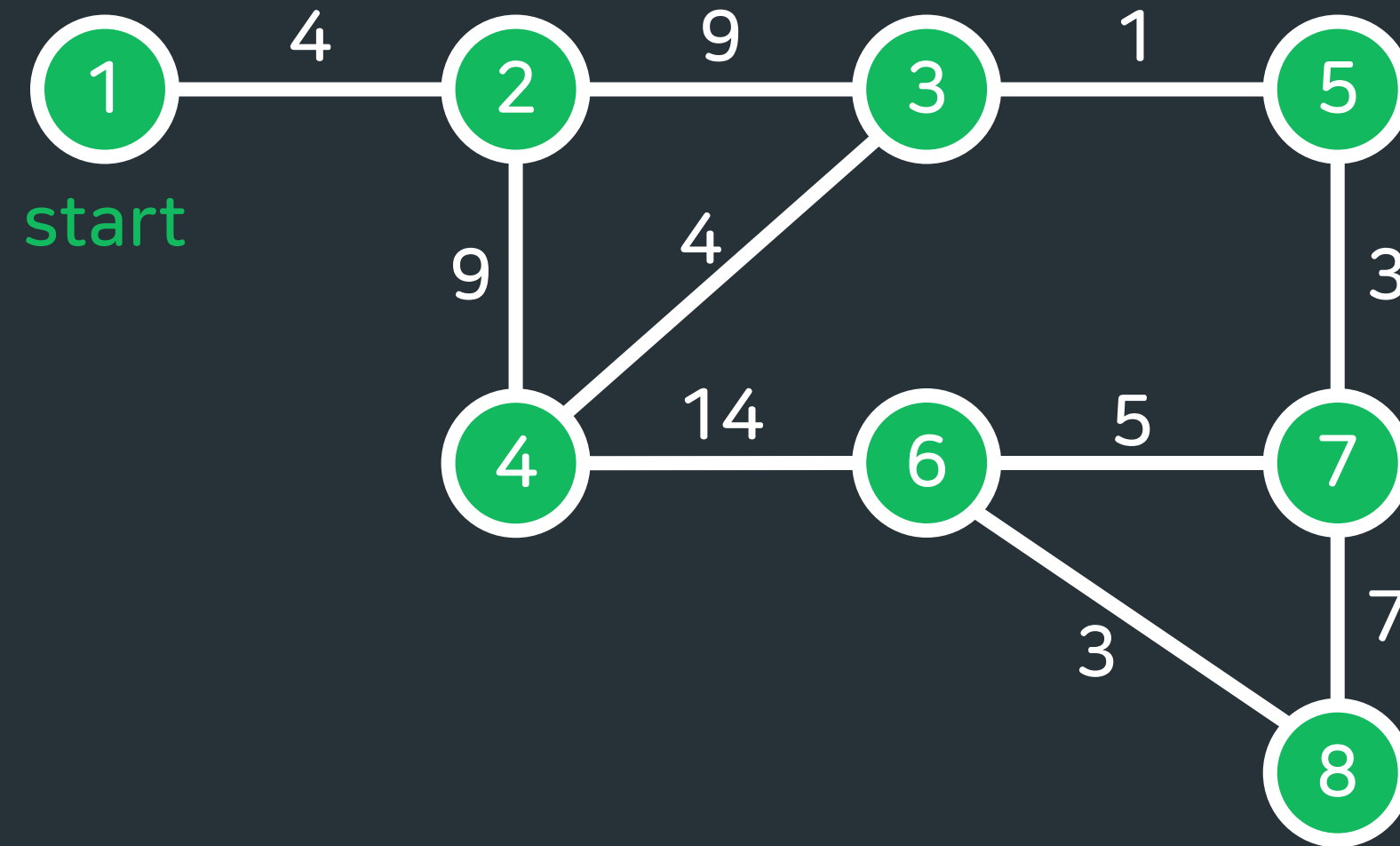
1	2	3	4	5	6	7	8
0	4	9	4	1	5	3	7

$14 > 5$

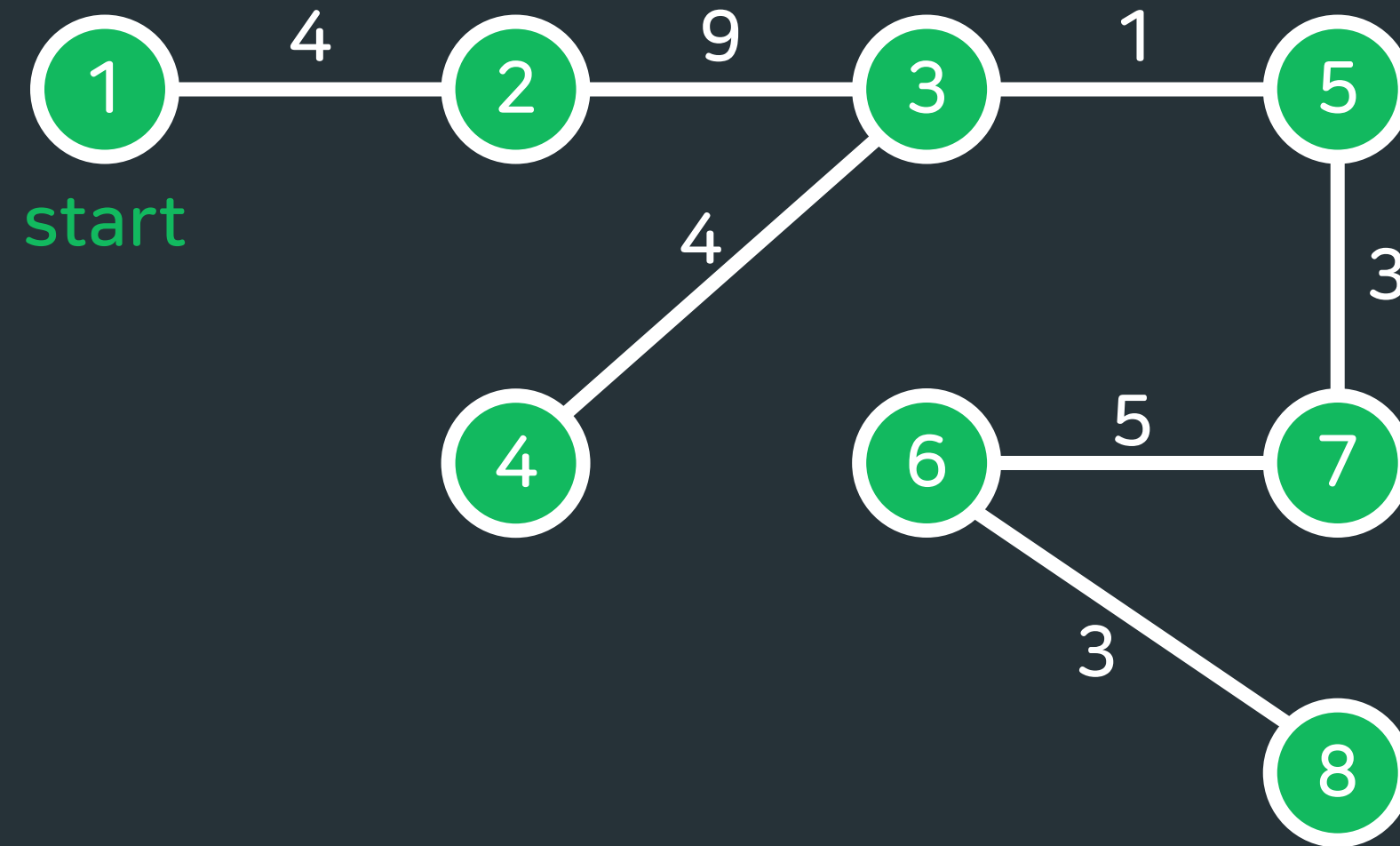


1	2	3	4	5	6	7	8
0	4	9	4	1	5	3	3

3 < 7



1	2	3	4	5	6	7	8
0	4	9	4	1	5	3	3



1	2	3	4	5	6	7	8
0	4	9	4	1	5	3	3

# 구현할 때 주의할 점



```
vector<int> dijkstra(int vertex, int start, vector<vector<ci>> &graph) {  
    vector<int> dist(vertex + 1, INF);  
    priority_queue<ci, vector<ci>, greater<>> pq; //first : 시작점으로부터의 거리, second : 정점  
  
    //시작 위치 초기화  
    dist[start] = 0;  
    pq.push({0, start});  
  
    while (!pq.empty()) {  
        int weight = pq.top().first;  
        int node = pq.top().second;  
        pq.pop();  
  
        if (weight > dist[node]) //이미 확인했던 정점  
            continue;  
        for (int i = 0; i < graph[node].size(); i++) {  
            int next_node = graph[node][i].first; //연결된 정점  
            int next_weight = weight + graph[node][i].second; //시작점으로부터 연결된 정점까지의 거리  
            if (dist[next_node] > next_weight) { //더 짧은 경로로 갈 수 있다면  
                dist[next_node] = next_weight;  
                pq.push({next_weight, next_node});  
            }  
        }  
    }  
    return dist;  
}
```

이 부분만 없으면 될까?

# dist 배열만 있다면?



1	2	3
0	INF	INF

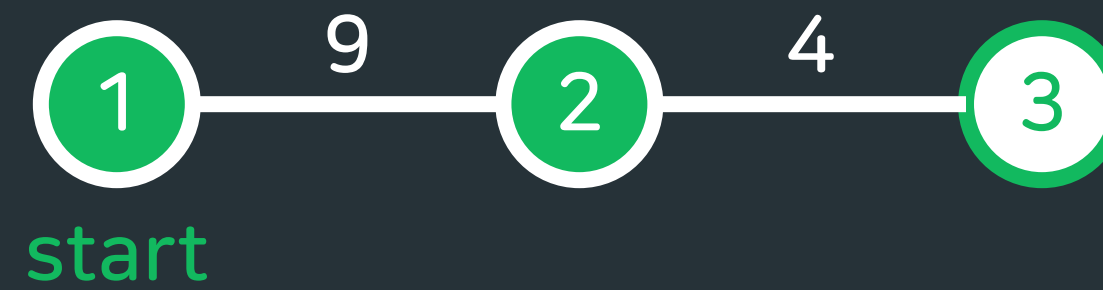
# dist 배열만 있다면?



1	2	3
0	9	INF



# dist 배열만 있다면?



1	2	3
0	9	4

# dist 배열만 있다면?



1	2	3
0	9	4

4 < 9?

# dist 배열만 있다면?



1	2	3
0	9	4

다익스트라는 가중치가 누적돼서 이미 방문한 정점을 또 방문할 일이 없었으나,  
간선의 가중치만 사용하는 프림은 재방문 가능성이 있으므로 **visited** 배열 필요

# 어떤 알고리즘을 사용해야?

- 크루스칼 시간 복잡도 :  $O(E \log E)$
- 프림 시간 복잡도 :  $O(V \log V + E \log V)$
- 크루스칼 알고리즘 연산 횟수에 영향을 주는 요소는 오직 간선의 수
- 프림 알고리즘 연산 횟수에 주로 영향을 주는 요소는 정점의 수
- 간선이 많거나, 특정 시작 정점이 주어지면 프림
- 간선이 적거나, 특정한 시작 정점이 없다면 크루스칼
- 개인 취향의 영역에 가까움...

## /<> 1197번 : 최소 스패닝 트리 - Gold 4

### 문제

- 그래프에 대한 최소 신장 트리는?

### 제한 사항

- 정점의 개수  $V$ 는  $1 \leq V \leq 10,000$
- 간선의 개수  $E$ 는  $1 \leq E \leq 100,000$
- 간선의 가중치  $C$ 는  $-1,000,000 \leq C \leq 1,000,000$

### 예제 입력 1

```
3 3
1 2 1
2 3 2
1 3 3
```

### 예제 출력 1

```
3
```

## /<> 4386번 : 별자리 만들기 - Gold 4

### 문제

- 2차원 평면에 위치한 별들의 (x, y) 좌표가 주어진다.
- 별 2개를 연결하는 비용은 별 사이의 거리와 같다.
- 모든 별들을 가장 적은 비용으로 연결할 방법은?

### 제한 사항

- 별의 개수  $n$ 은  $1 \leq n \leq 100$
- x, y 좌표는  $0.0 \leq x, y \leq 1000.0$

### 예제 입력 1

```
3
1.0 1.0
2.0 2.0
2.0 4.0
```

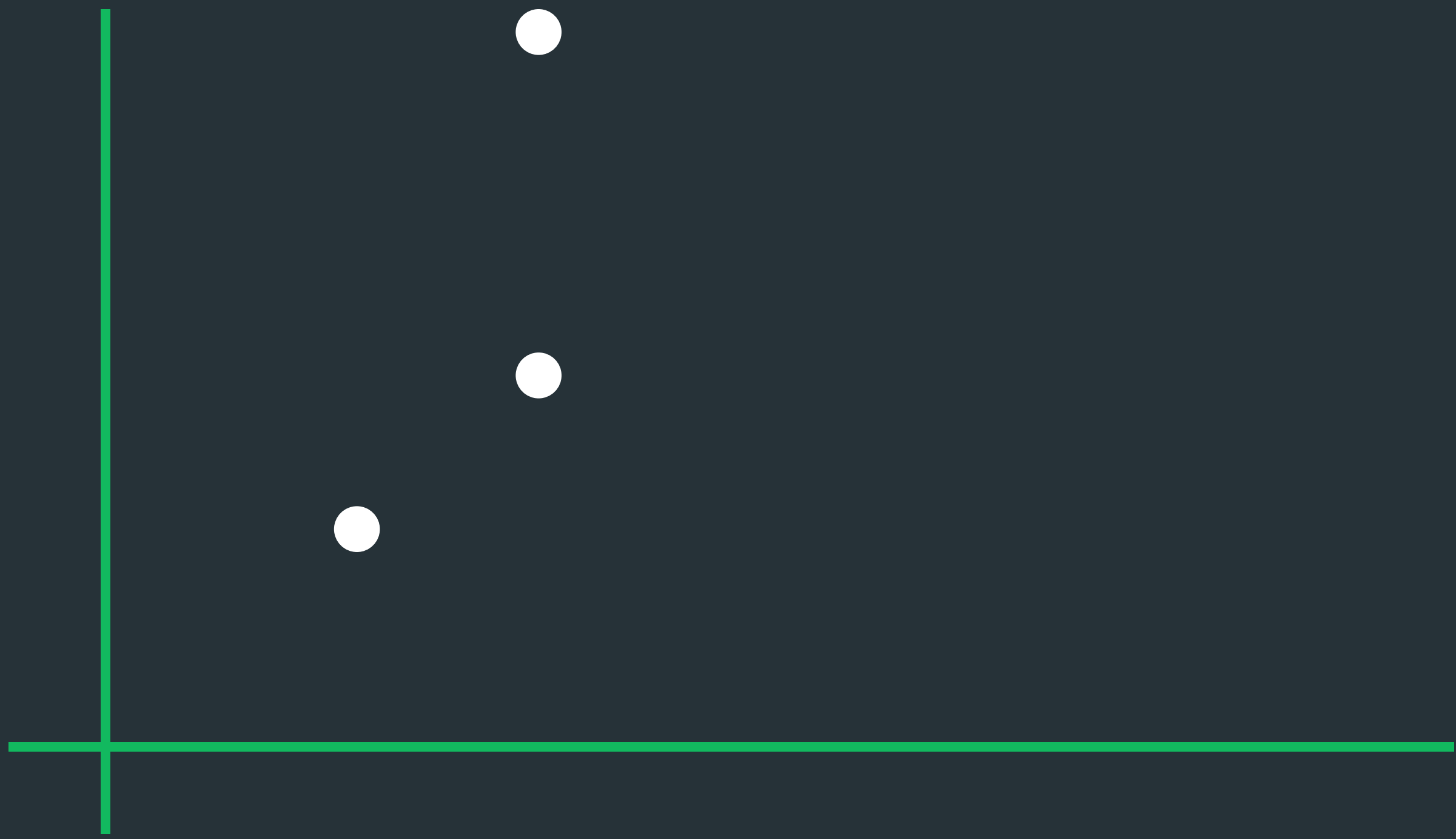
### 예제 출력 1

```
3.41
```

## Hint

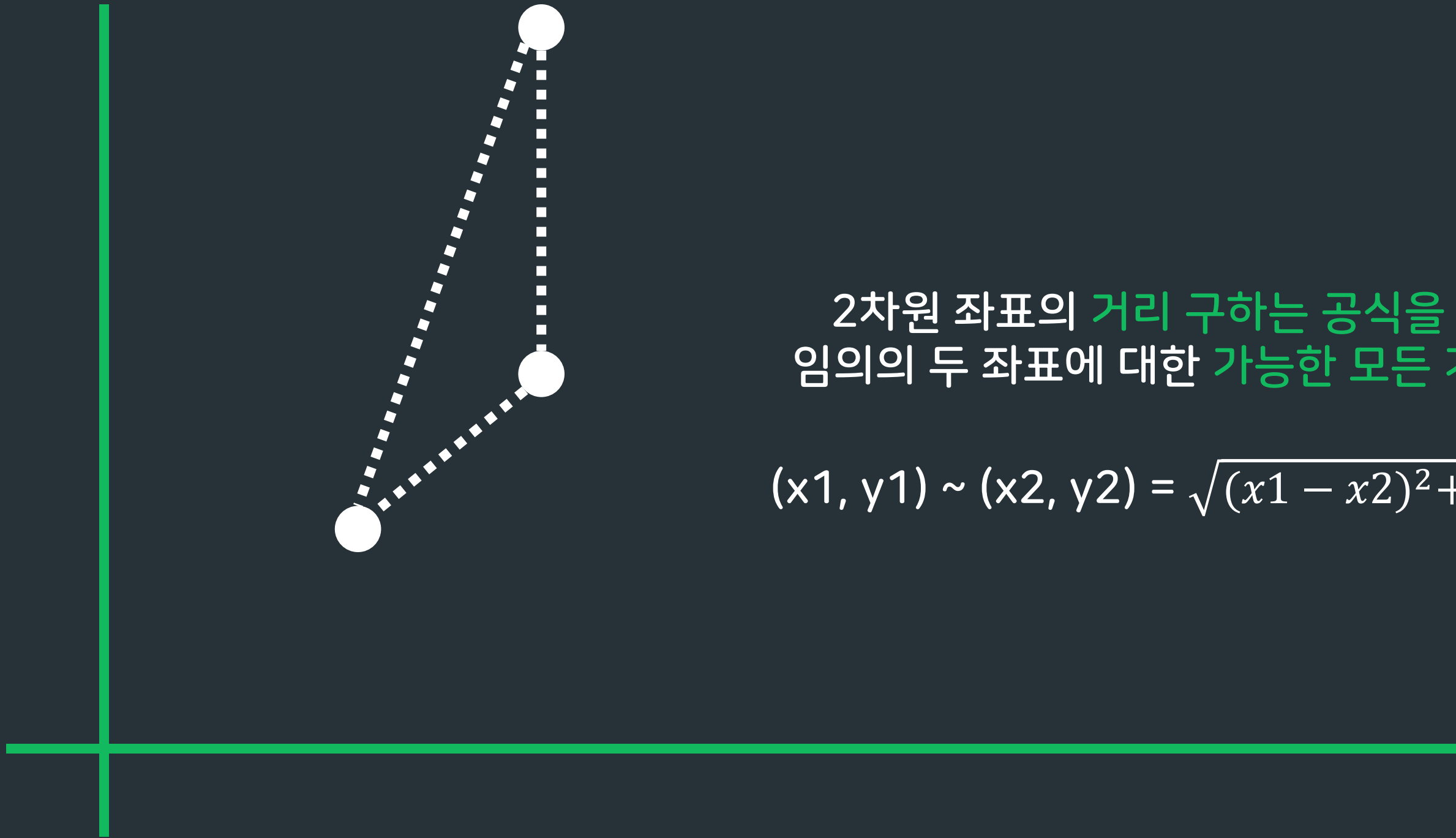
- 입력 범위가 작네요!
- 2차원 평면에서 두 좌표 사이의 **거리를 구하는 공식**은 무엇이었나요?

# 간선이 없다면?





## 간선이 없다면?



2차원 좌표의 **거리 구하는 공식**을 이용하여  
임의의 두 좌표에 대한 **가능한 모든 거리** 구하기

$$(x1, y1) \sim (x2, y2) = \sqrt{(x1 - x2)^2 + (y1 - y2)^2}$$

## 정리

- 그래프에서 만들 수 있는 모든 트리를 신장 트리라고 칭함
- 그 중 간선의 가중치 총 합이 가장 작은 트리가 최소 신장 트리(MST)
- MST를 구하는 알고리즘은 크루스칼, 프림이 있음
- 크루스칼은 유니온 파인드 알고리즘을 활용하고, 프림은 다익스트라와 유사
- 간선이 적다면 크루스칼, 간선이 많거나 시작점이 주어지면 프림


## 이것도 알아보세요

- 지난 3주간 트리 자료구조와 트리를 활용하는 유니온 파인드, 최소 신장 트리 알고리즘에 대해 배웠습니다. 3개의 주제를 함께 복습하시면 지난 내용이 더 잘 이해될거예요!

## 필수

- /<> 16235번 : 나무 재테크 - Gold 4
- /<> 1713번 : 후보 추천하기 - Silver 2

## 3문제 이상 선택

- /<> 1368번 : 물대기 - Gold 2
- /<> 1774번 : 우주신과의 교감 - Gold 3
- /<> 16202번 : MST 게임 - Gold 4
- /<> 17472번 : 다리 만들기 2 - Gold 2
- /<> 21924번 : 도시 건설 - Gold 4
-  2019 카카오 개발자 겨울 인턴십 : 호텔 방 배정 - Level 4