

#3. 최소 신장 트리

나정휘

<https://justicehui.github.io/>

최소 신장 트리

용어 정의

- 부분 그래프(Subgraph) : 그래프의 정점과 간선의 일부를 선택해서 만든 그래프
- 신장 부분 그래프(Spanning ~) : 그래프의 모든 정점을 포함하는 부분 그래프
- 신장 포레스트(Spanning Forest) : 사이클이 없는 신장 부분 그래프
- 신장 트리(Spanning Tree) : 모든 정점이 연결된 신장 포레스트
- 최소 비용 신장 트리(Minimum ~) : 간선의 가중치의 합이 최소인 신장 트리

최소 신장 트리

최소 신장 트리 (Minimum Spanning Tree)

- 그래프가 주어지면 최소 신장 트리를 구하는 알고리즘
- = 최소 비용으로 모든 정점을 연결하는 비용을 구하는 알고리즘
- 여러 가지 알고리즘이 있음
 - Prim's Algorithm (V^2 , $E \log V$, $E + V \log V$ 등)
 - Kruskal's Algorithm ($E \log E$)
 - Boruvka's Algorithm ($E \log V$, 이건 설명 안 함)
 - Sollin's Algorithm이라고 부르기도 함

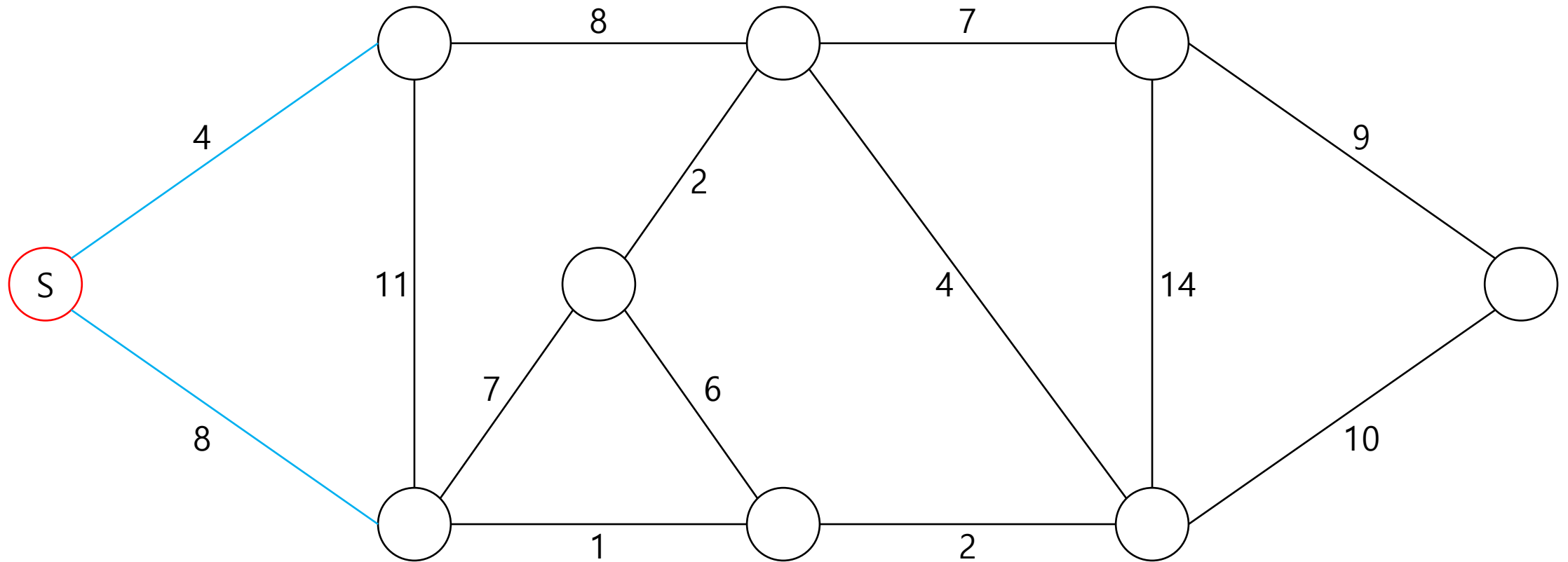
Prim's Algorithm

Prim's Algorithm

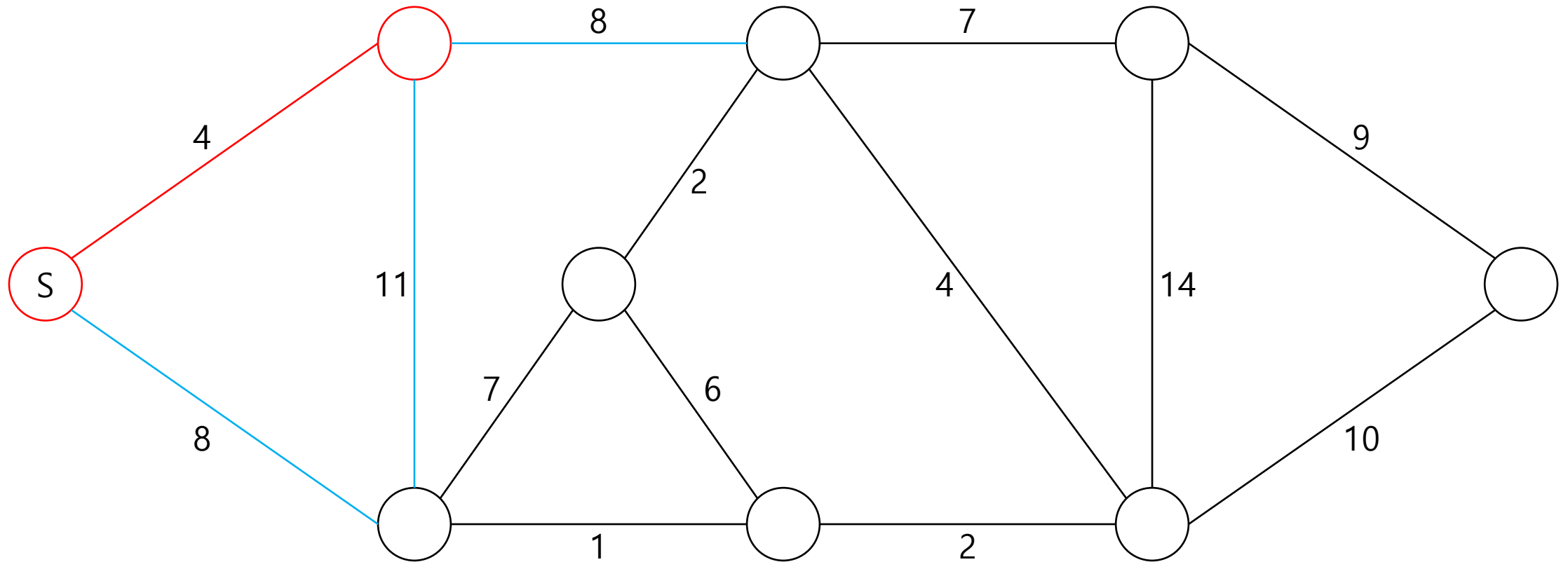
Prim's Algorithm

- 시간 복잡도: $O(V^2)$ / $O(E \log E)$ / $O(E + V \log V)$
- Spanning Tree에 정점을 하나씩 포함시키면서 확장하는 방식으로 진행
- 그리디 기반 알고리즘
 1. 시작점(S)을 MST에 넣음
 2. 현재 MST에 있는 정점에서 뻗어 나가는 간선 중 가중치가 가장 작은 간선(e) 선택
 3. 만약 MST에 e를 추가할 수 있다면(사이클이 생기지 않는다면) e를 MST에 추가
 - 사이클 판별은 $e = (u, v)$ 에서 u와 v가 이미 MST에 포함되었는지 확인하면 됨

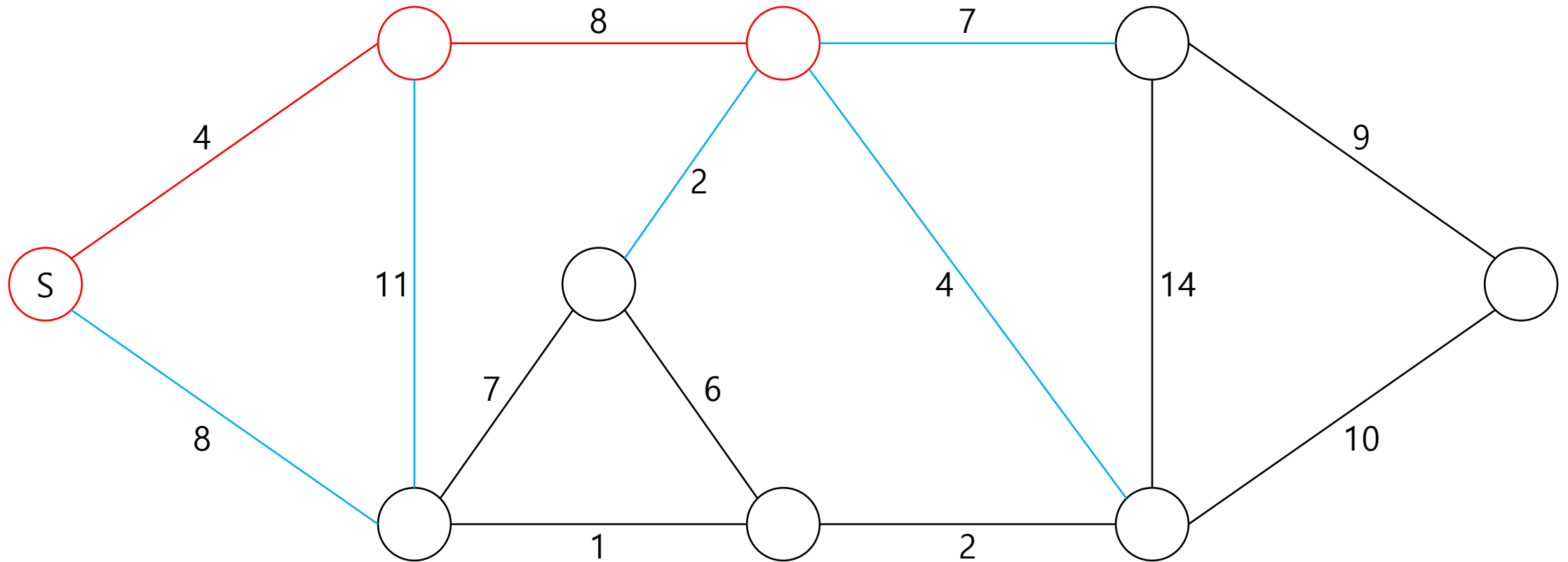
Prim's Algorithm



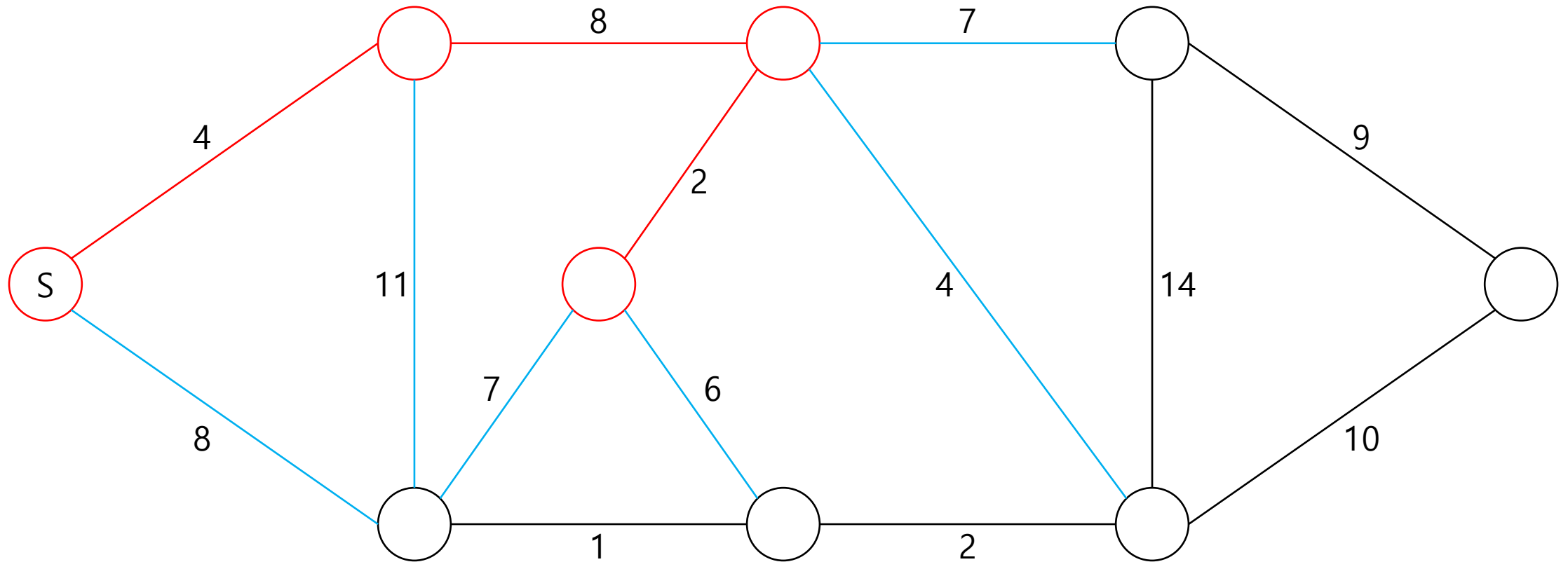
Prim's Algorithm



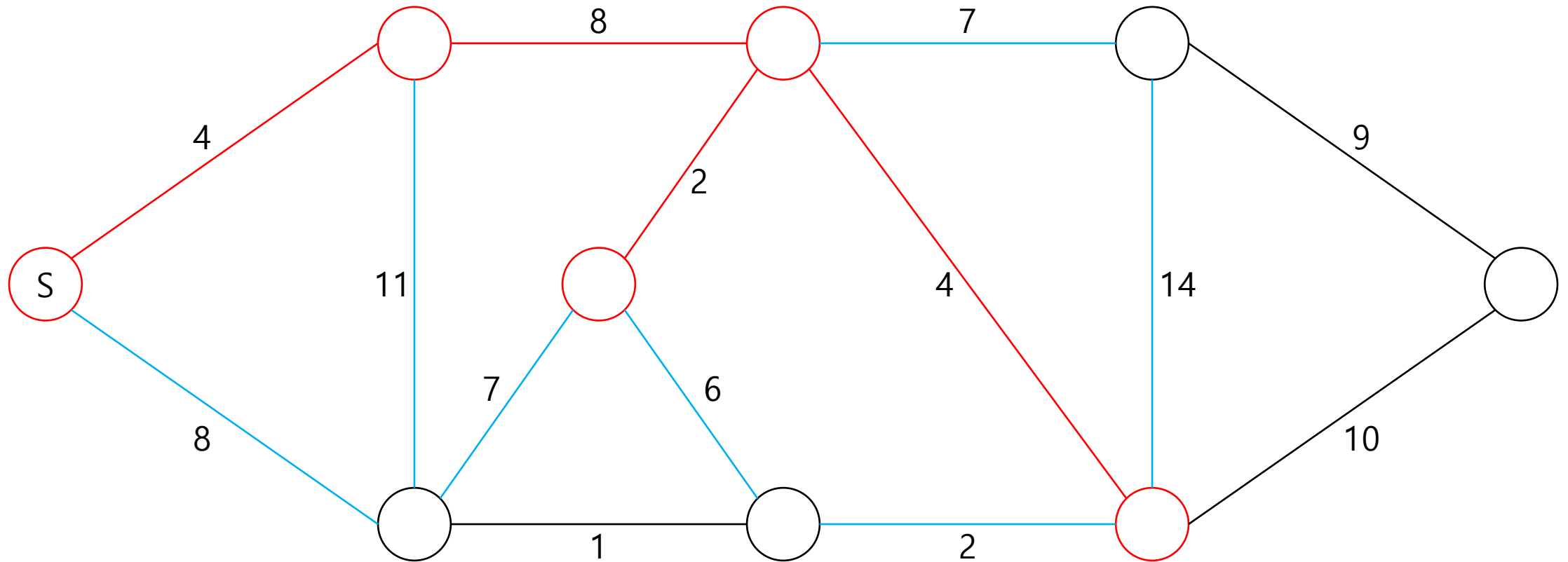
Prim's Algorithm



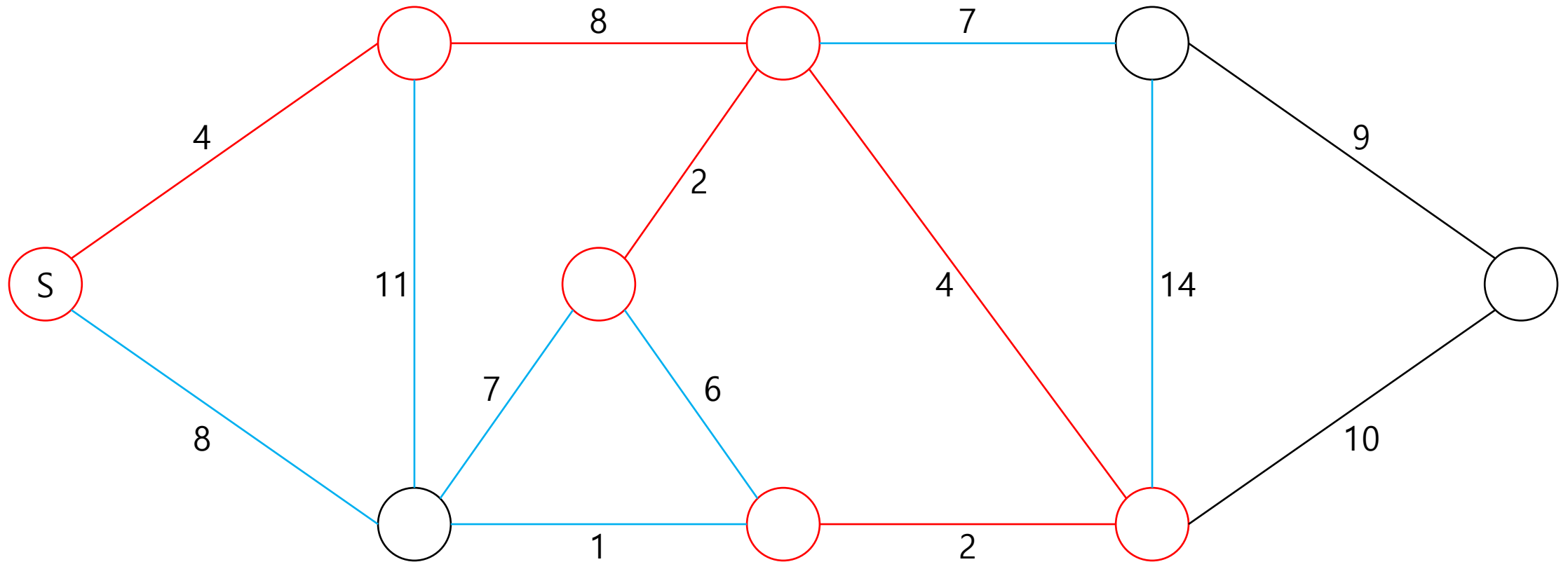
Prim's Algorithm



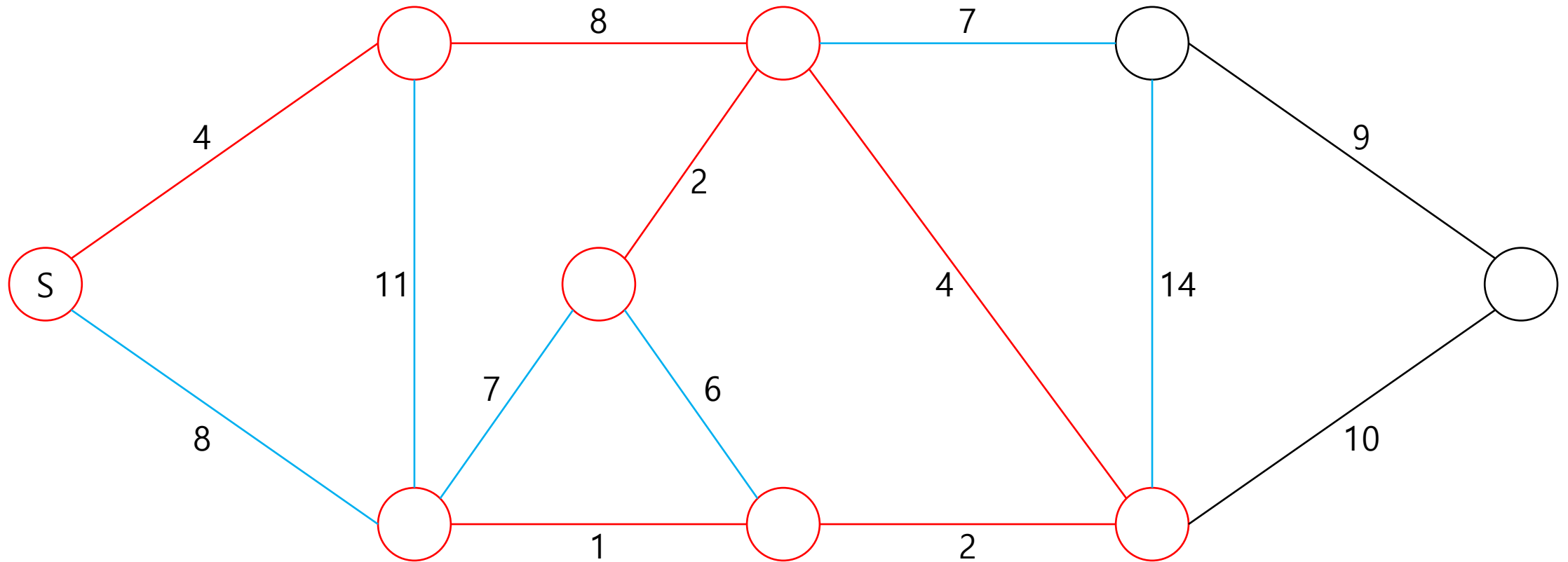
Prim's Algorithm



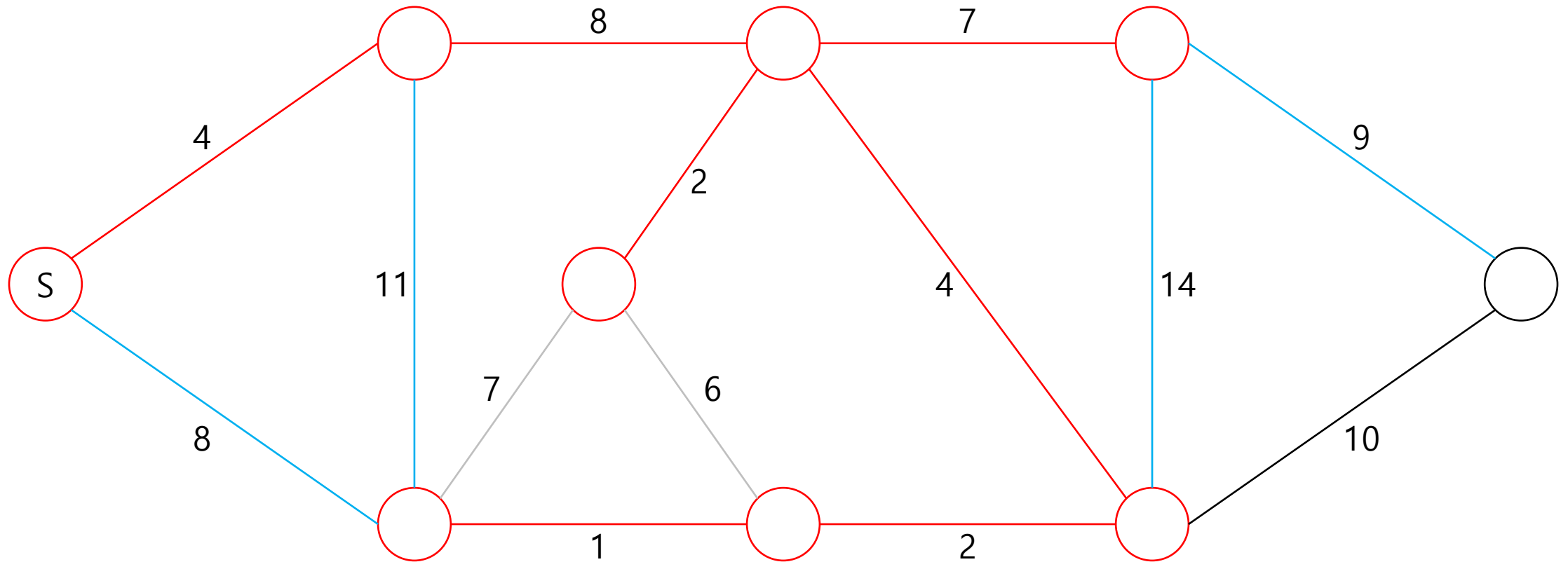
Prim's Algorithm



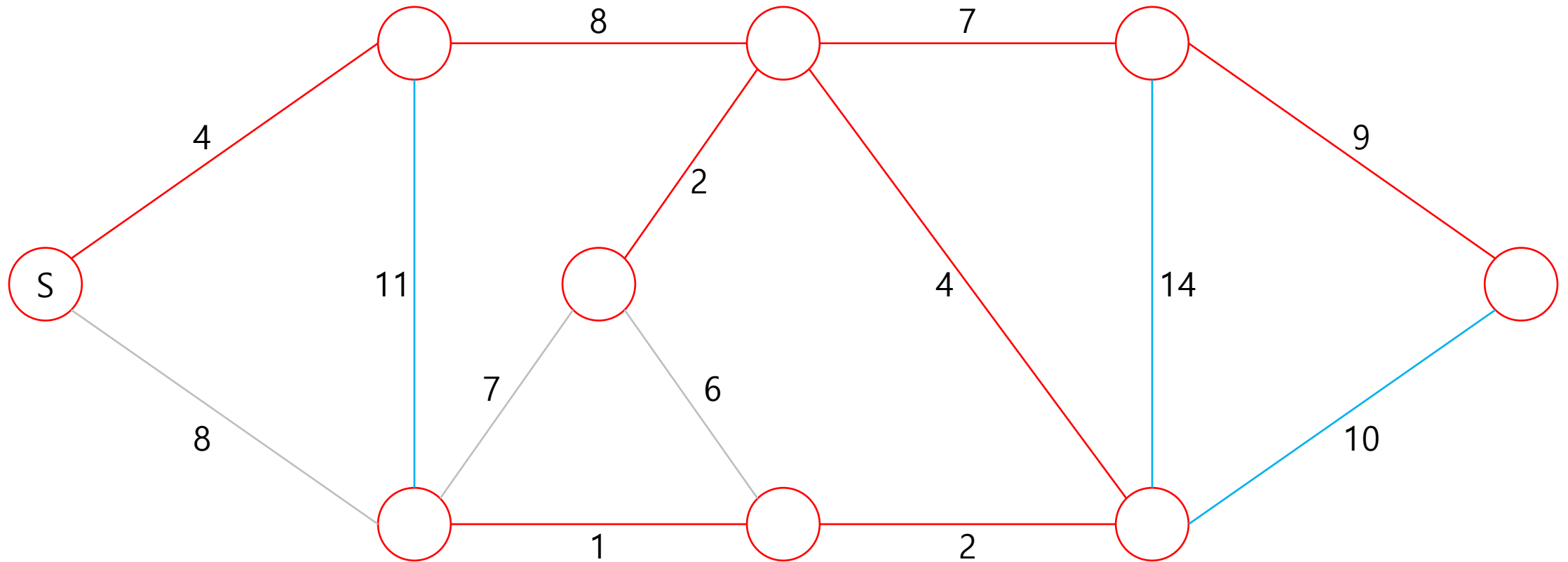
Prim's Algorithm



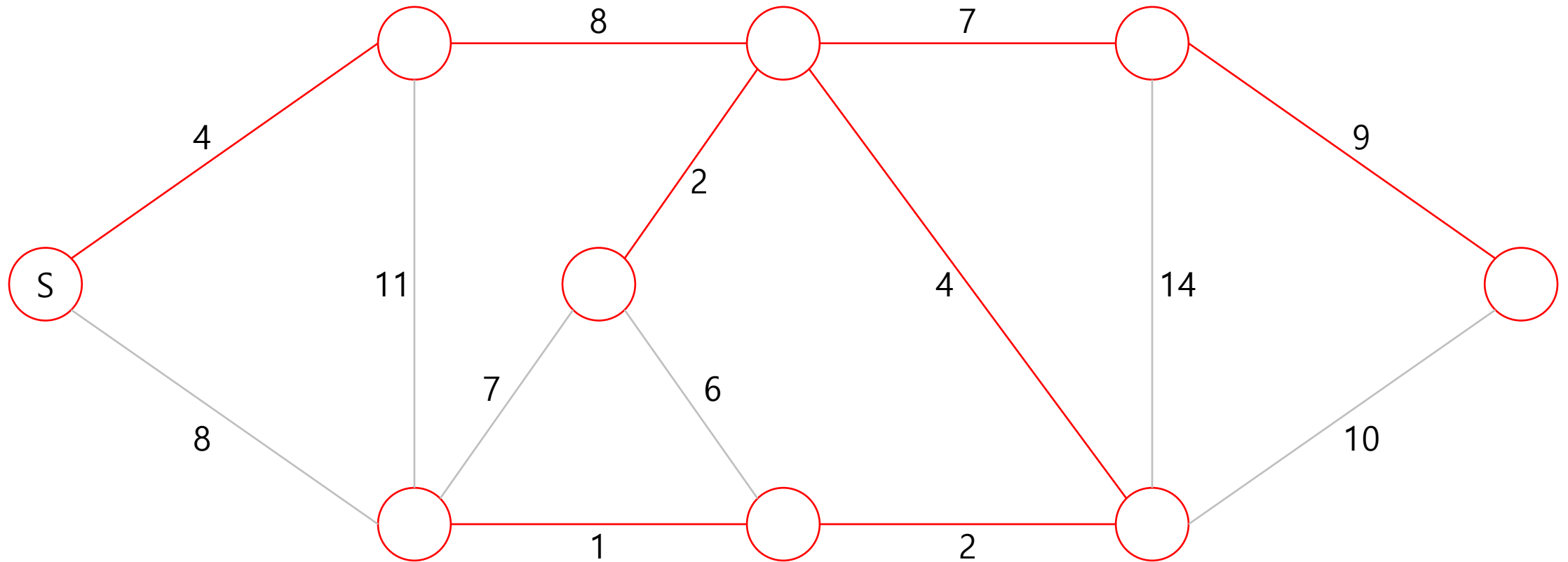
Prim's Algorithm



Prim's Algorithm



Prim's Algorithm



Prim's Algorithm



```
using PII = pair<int, int>;
int N, M, C[10101], D[10101];
vector<PII> G[10101];
```

```
int Prim(){
    int ret = 0;
    for(int i=1; i<=N; i++) D[i] = 1e9;
    D[1] = 0;
    for(int iter=1; iter<=N; iter++){
        int v = -1;
        for(int i=1; i<=N; i++){
            if(C[i]) continue;
            if(v == -1 || D[v] > D[i]) v = i;
        }
        C[v] = 1; ret += D[v];
        for(auto [i,w] : G[v]) D[i] = min(D[i], w);
    }
    return ret;
}
```

// D[i]: i번 정점을 MST에 추가하기 위해 필요한 비용

// 이미 MST에 포함된 정점 스킵

// MST에 간선 추가

// v에서 나가는 간선 정보 반영

Prim's Algorithm

정당성 증명

- 수학적 귀납법을 사용해 증명
 - 현재까지 만든 포레스트 F 를 포함하는 최소 스패닝 트리 T 가 존재할 때
 - F 와 $V-F$ 를 연결하는 최소 간선 e 를 추가한 $F+e$ 를 포함하는 최소 스패닝 트리 T' 이 존재함을 증명
 - 만약 e 가 T 에 포함되면 $T' = T$ 이다.
 - 그렇지 않은 경우, $T+e$ 는 e 를 포함하는 단순 사이클 C 를 갖는다.
 - 이때 C 는 $F+e$ 에 속하지 않으면서 F 와 $V-F$ 를 연결하는 간선 f 를 갖는다.
 - e 는 F 와 $V-F$ 를 연결하는 최소 간선이므로 f 의 가중치는 e 보다 크거나 같아야 한다.
 - 단순 사이클에서 간선 하나를 뺀 $T-f+e$ 는 트리가 되고, 이것의 가중치는 T 이하이므로
 - $T' = T-f+e$ 는 $F+e$ 를 포함하는 최소 스패닝 트리이다.

Prim's Algorithm

시간 복잡도

- V번의 iteration
 - MST에 포함되지 않은 정점 중 비용이 최소인 정점 찾기 : $O(V)$
 - v에서 갈 수 있는 정점들의 거리 갱신 : $O(\deg(v))$
- $O(\sum(V + \deg(i))) = O(V^2 + E) = O(V^2)$
 - Handshaking Lemma : $\sum(\deg(i)) = 2E$
- v에서 뺀어 나가는 간선은 모두 봐야 하므로 $O(\deg(v))$ 가 하한임
- 비용이 최소인 정점을 빠르게 찾을 수 있을까?
 - Min Heap!

Prim's Algorithm



```
int Prim(){
    int ret = 0;
    priority_queue<PII, vector<PII>, greater<>> pq; // {거리, 정점} pair를 저장하는 min-heap
    C[1] = 1; // 시작점 S = 1은 MST에 포함
    for(auto [i,w] : G[1]) pq.emplace(w, i); // S에서 나가는 간선들 Heap에 추가
    while(!pq.empty()){
        auto [c,v] = pq.top(); pq.pop(); // 비용이 가장 작은 정점 v 구함
        if(C[v]) continue; // heap에 같은 정점이 여러 번 들어갈 수 있으니 주의
        C[v] = 1; ret += c; // MST에 추가
        for(auto [i,w] : G[v]) pq.emplace(w, i); // v에서 나가는 간선 정보 반영
    }
    return ret;
}
```

Prim's Algorithm

시간 복잡도

- 각 간선을 한 번씩 보기 때문에 거리 갱신은 최대 $O(E)$ 번 발생
- Heap에 원소 $O(E)$ 번 삽입
- Heap의 크기는 최대 $O(E)$ 이므로 시간 복잡도는 $O(E \log E)$

참고

- 각 정점마다 Heap에 원소가 최대 한 개 존재하도록 구현하면 $O(E \log V)$
- Heap의 decrease key 연산을 $O(1)$ 에 구현하면 $O(E + V \log V)$ 도 가능 (Fibonacci Heap, Thin Heap)
 - GCC 확장의 `__gnu_pbds::priority_queue`는 `pq.modify(iterator, value)`를 이용해 decrease key 가능
 - C++ 표준의 `std::priority_queue`는 decrease key 연산을 지원하지 않음

Prim's Algorithm – $O(E + V \log V)$



```
using thin_heap = __gnu_pbds::priority_queue<pair<ll,ll>, greater<>, __gnu_pbds::thin_heap_tag>;
using heap_iter = __gnu_pbds::detail::thin_heap<pair<ll,ll>, greater<>, allocator<char>>::point_iterator;

int N, M, C[505050];
vector<pair<int,int>> G[505050];
constexpr int INF = 1e9+7;

long long Prim(){
    long long res = 0;
    thin_heap pq;

    // pq에서 iter[i] = {i까지의 거리, i}를 저장하고 있는 노드의 이터레이터
    vector<heap_iter> iter(N);
    // 처음에는 모든 정점을 아주 큰 값으로 초기화
    for(int i=0; i<N; i++) iter[i] = pq.push({INF, i});
    pq.modify(iter[0], {0, 0}); // 0번 정점까지의 거리는 0으로 초기화

    while(!pq.empty()){
        auto [c,v] = pq.top(); pq.pop();
        C[v] = 1; res += c;
        for(auto [i,w] : G[v]){
            // 만약 이 간선을 이용하는 게 더 작다면
            // amortized O(1) 시간에 decrease key 연산 수행
            if(!C[i] && w < iter[i]->first){
                pq.modify(iter[i], {w, i});
            }
        }
    }
    return res;
}
```

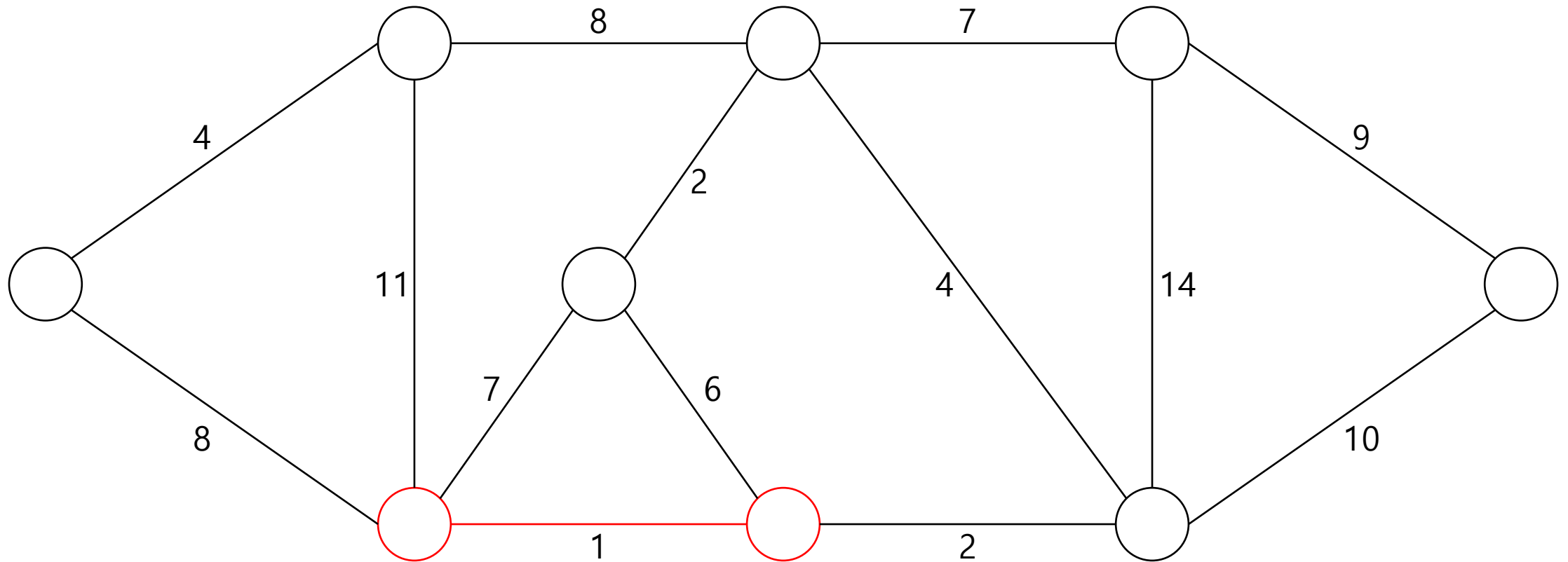
Kruskal's Algorithm

Kruskal's Algorithm

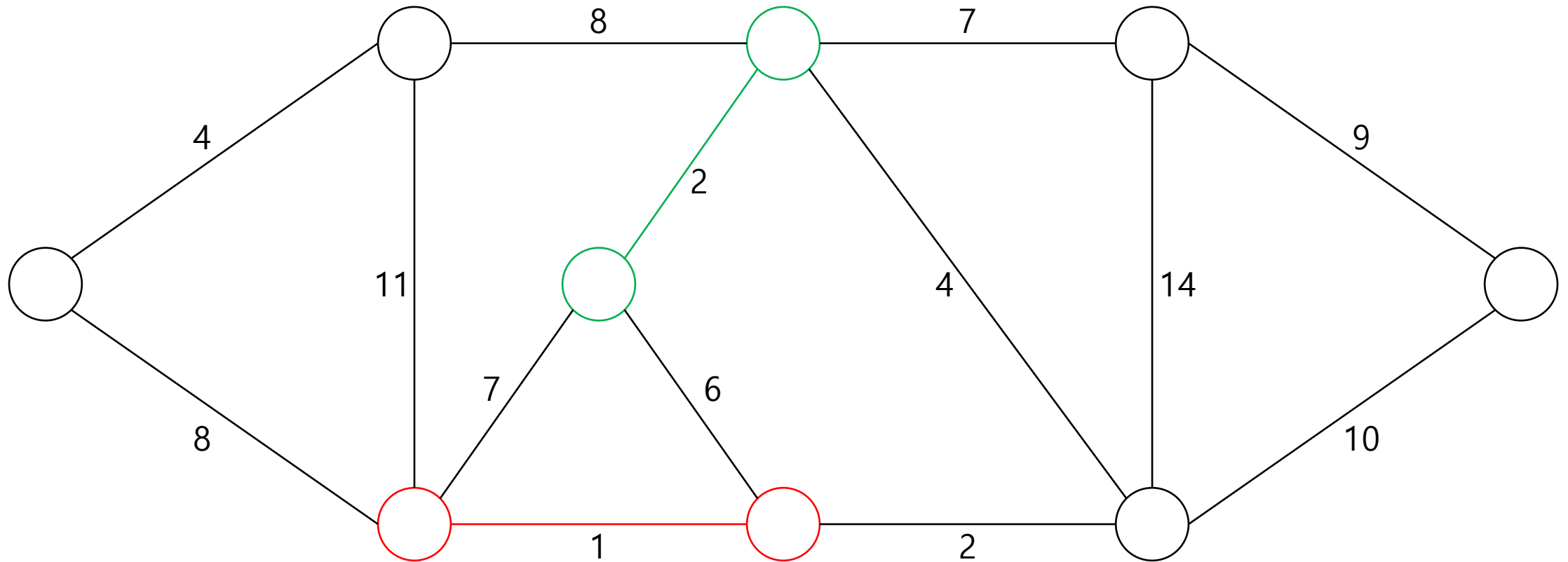
Kruskal's Algorithm

- 시간 복잡도 : $O(E \log E)$
- Spanning Tree에 간선을 하나씩 포함시키는 방식
- 그리디 기반 알고리즘
 1. 모든 간선을 가중치 오름차순으로 정렬
 2. 가중치가 작은 간선부터 보면서, 사이클이 생기지 않는다면 해당 간선을 MST에 추가
 - Prim's Algorithm과 다르게 알고리즘 진행 도중 트리가 여러 개 있을 수 있음
 - $e = (u, v)$ 에서 u 와 v 가 같은 트리에 있는지 확인해야 함 $\rightarrow$ Union-Find

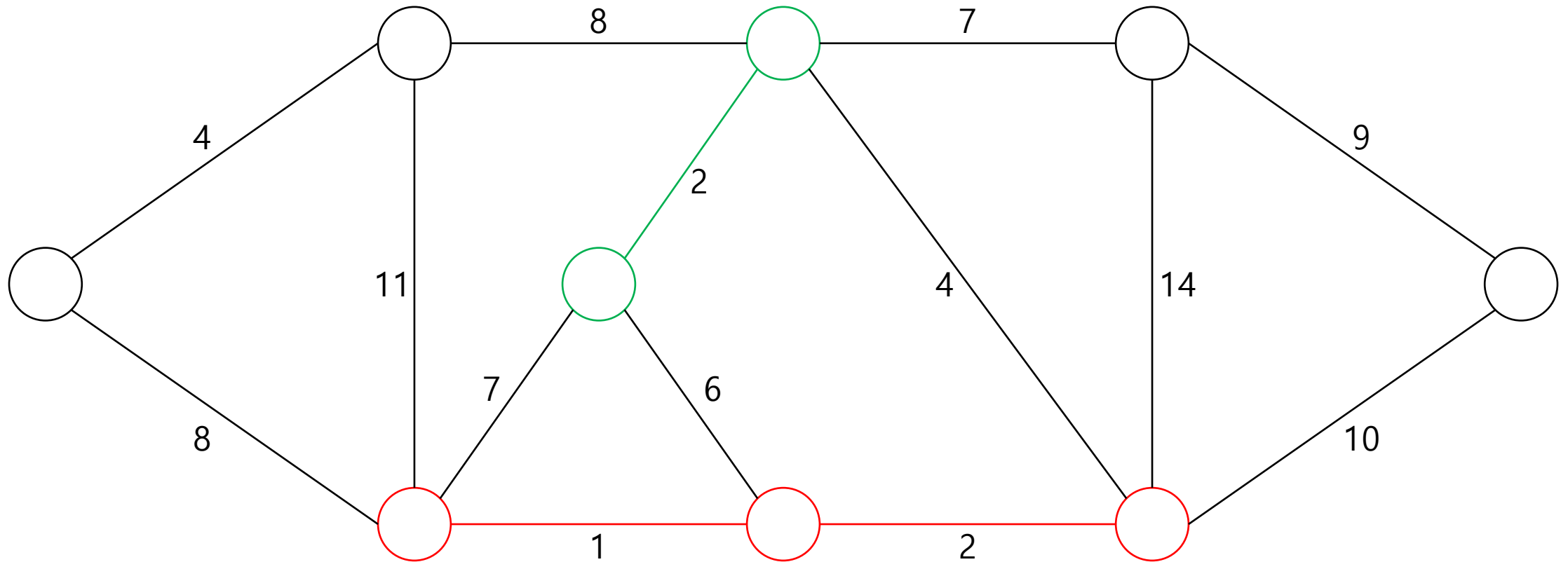
Kruskal's Algorithm



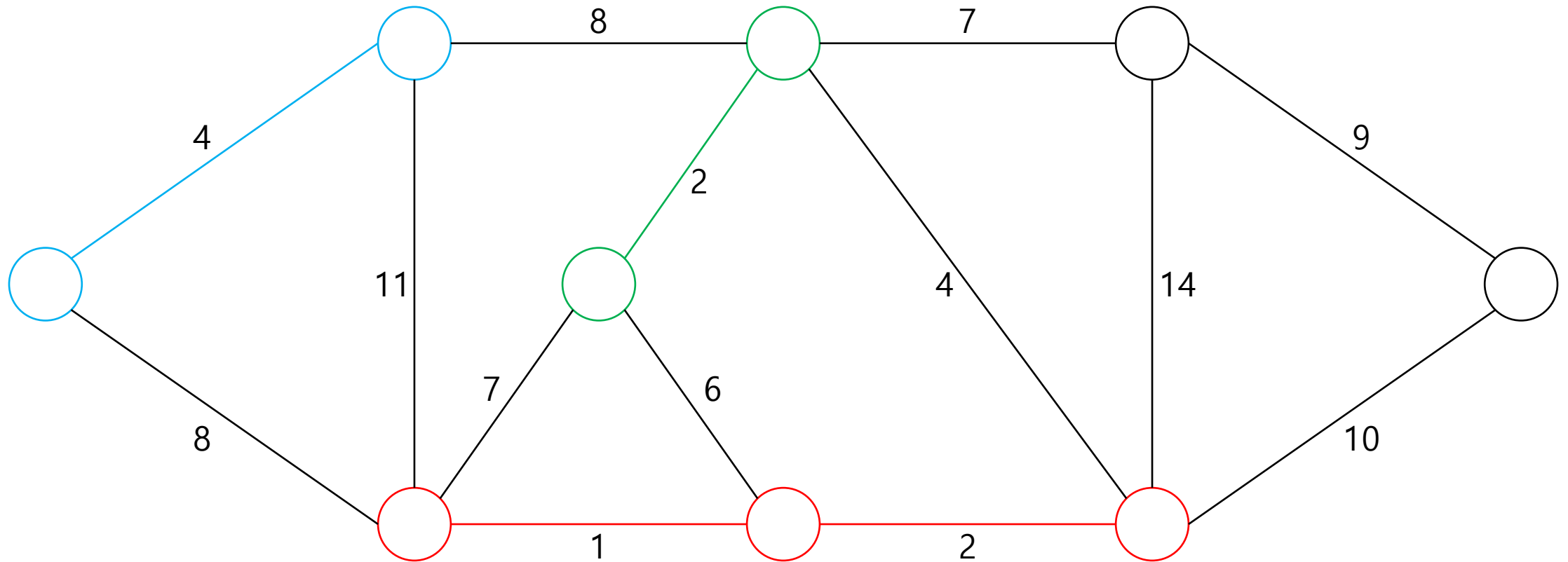
Kruskal's Algorithm



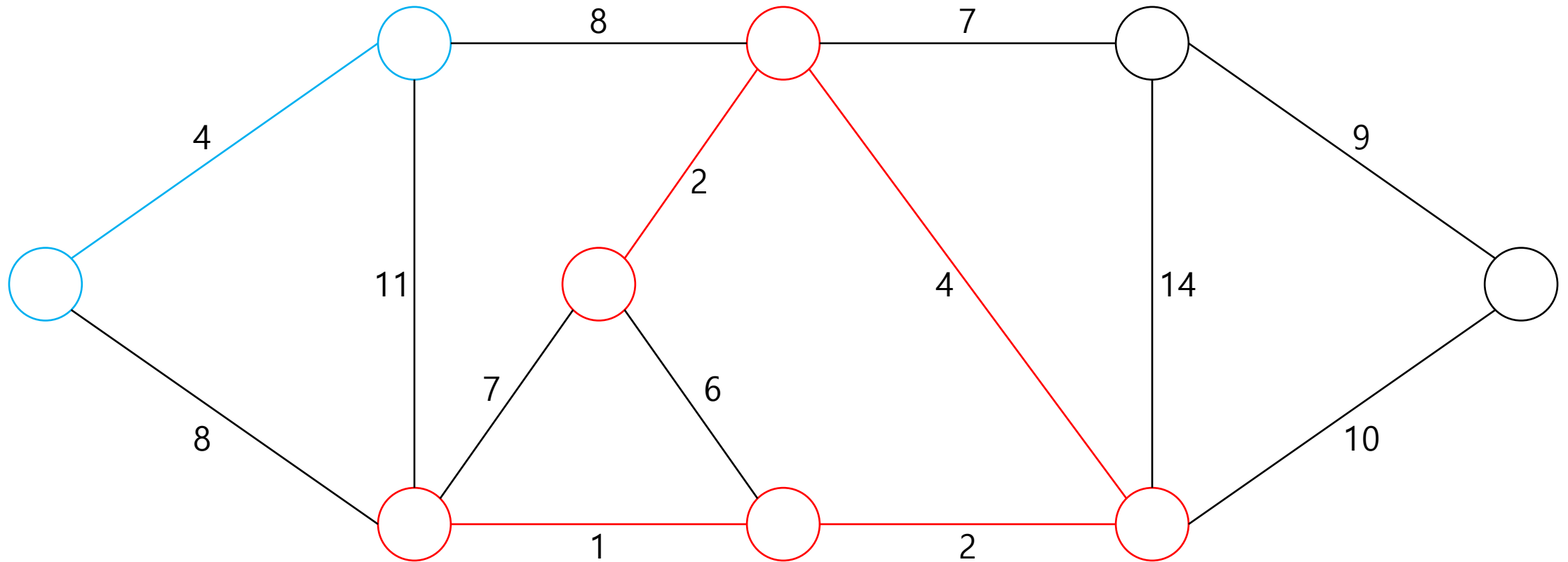
Kruskal's Algorithm



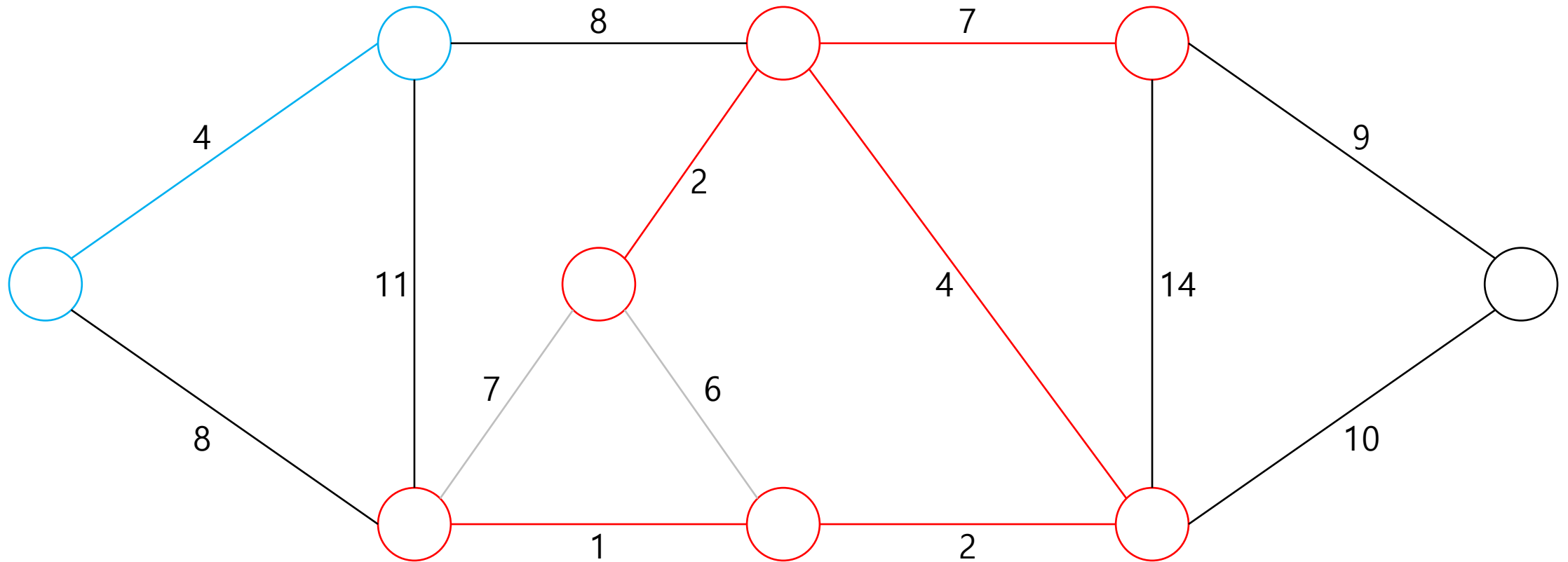
Kruskal's Algorithm



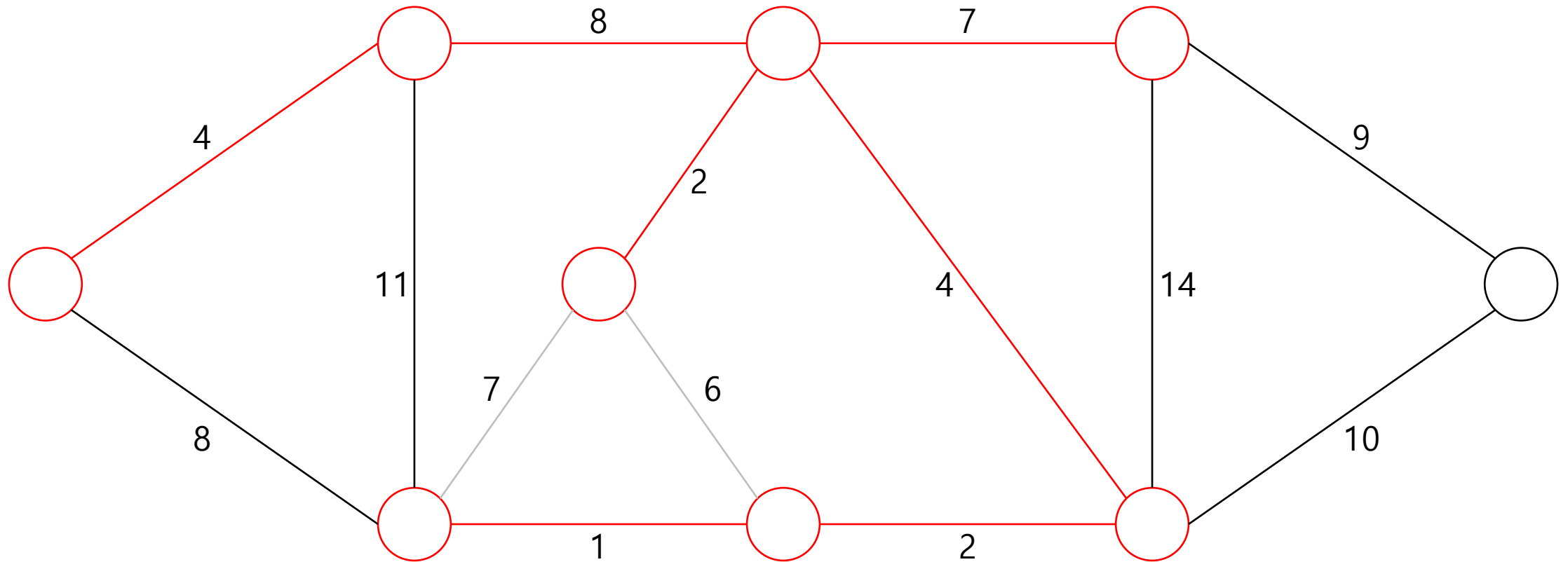
Kruskal's Algorithm



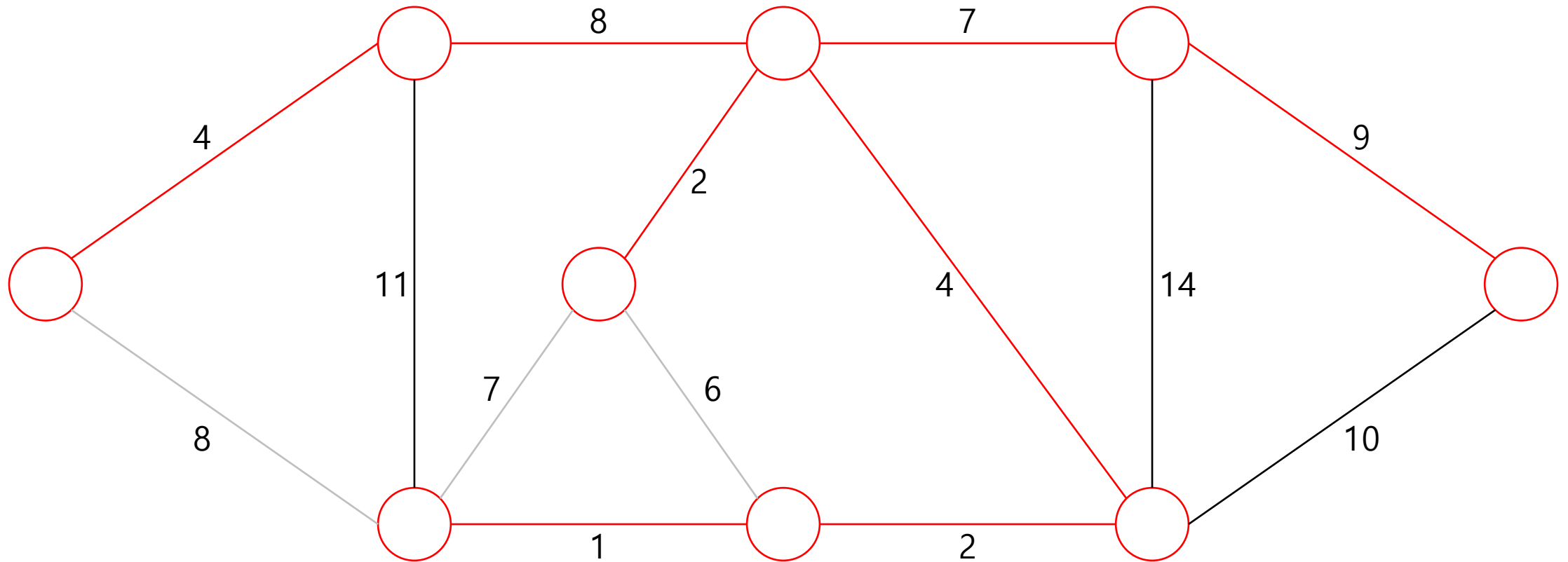
Kruskal's Algorithm



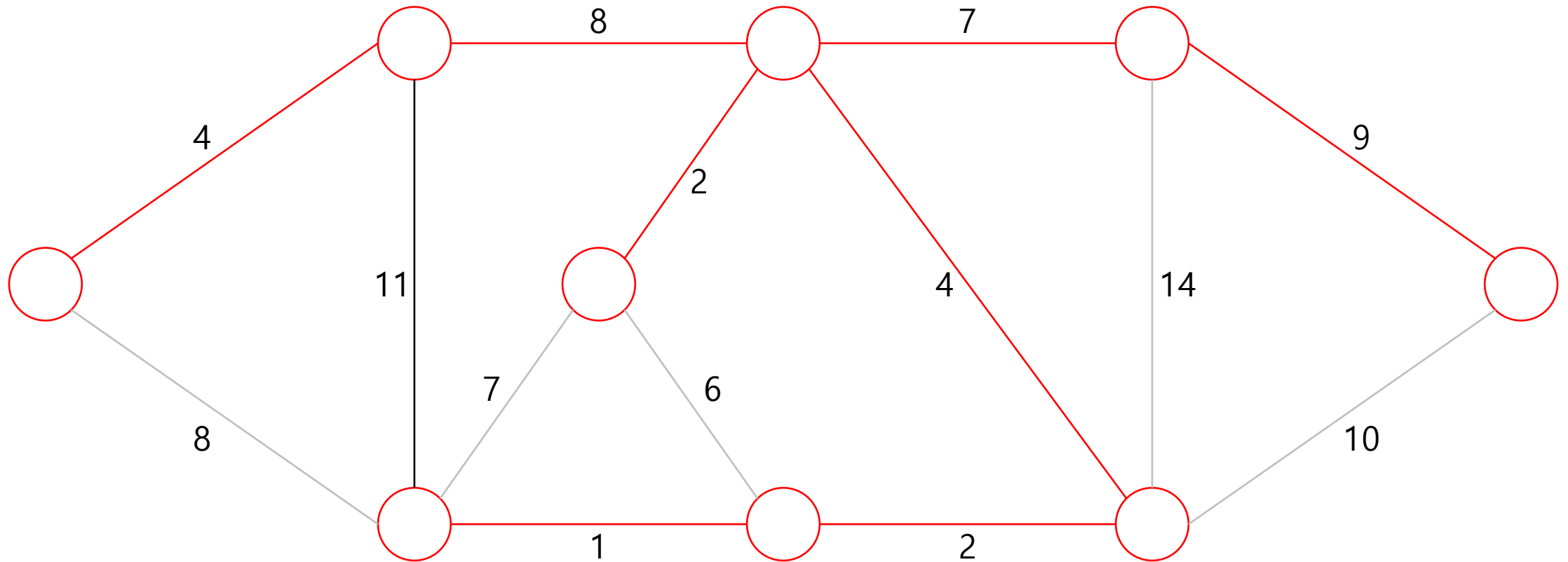
Kruskal's Algorithm



Kruskal's Algorithm



Kruskal's Algorithm



Kruskal's Algorithm



```
int N, M, P[10101];
vector<tuple<int,int,int>> E; // {비용, 정점 1, 정점 2}로 저장

int Find(int v){ return v == P[v] ? v : P[v] = Find(P[v]); }
bool Union(int u, int v){ return Find(u) != Find(v) && (P[P[u]]=P[v], true); }

int Kruskal(){
    int ret = 0;
    for(int i=1; i<=N; i++) P[i] = i;           // Union-Find 초기화
    sort(E.begin(), E.end());                  // 간선 오름차순으로 정렬
    for(auto [w,u,v] : E) if(Union(u, v)) ret += w; // 사이클이 생기지 않으면 MST에 추가
    return ret;
}
```

Kruskal's Algorithm

정당성 증명

- 수학적 귀납법을 사용해 증명
 - 현재까지 만든 포레스트 F 를 포함하는 최소 스패닝 트리 T 가 존재할 때
 - 사이클을 만들지 않는 최소 간선 e 를 추가한 $F+e$ 를 포함하는 최소 스패닝 트리 T' 이 존재함을 증명
 - 만약 e 가 T 에 포함되면 $T' = T$ 이다.
 - 그렇지 않은 경우, $T+e$ 는 e 를 포함하는 단순 사이클 C 를 갖는다.
 - 이때 C 는 $F+e$ 에 속하지 않으면서 F 와 $V-F$ 를 연결하는 간선 f 를 갖는다.
 - f 는 아직 알고리즘 과정에서 고려되지 않았으므로 e 보다 가중치가 크거나 같아야 한다.
 - 단순 사이클에서 간선 하나를 끊어낸 $T-f+e$ 는 트리가 되고, 이것의 가중치는 T 이하이므로
 - $T' = T-f+e$ 는 $F+e$ 를 포함하는 최소 스패닝 트리이다.

Kruskal's Algorithm

시간 복잡도

- 간선 정렬 : $O(E \log E)$
- Union-Find 연산 : $O(\log V)$ 연산을 $O(E)$ 번 수행
- 시간 복잡도 : $O(E \log E)$

최소 신장 트리

Prim vs Kruskal

- 구현: Kruskal이 쉬움
- 속도: 일반적으로 Kruskal이 빠름
- Prim: 어떤 상황에서 더 유리할까?
- 간선의 가중치가 정점 번호에 대한 수식으로 표현되고, 메모리 제한이 작은 경우
 - $O(V^2)$ Prim's Algorithm은 공간 복잡도 $O(V)$
 - $O(E \log E)$ Prim's Algorithm과 Kruskal's Algorithm은 공간 복잡도 $O(V+E)$
 - Prim은 힙에 간선 E 개 들어갈 수 있고, Kruskal은 모든 간선을 다 구해서 정렬해야 함
 - BOJ 20390 완전그래프의 최소 스패닝 트리

최소 신장 트리 - 예시

BOJ 28119 Traveling SCCC President

- N개의 정점과 M개의 간선으로 구성된 무향 가중치 그래프가 주어짐
- 각 정점마다 한 번씩 주어진 순서대로 방문해서 회의를 진행해야 함
- 간선을 따라 이동할 때마다 간선의 가중치 만큼의 비용이 들지만
- 한 번 방문한 정점으로는 순간이동을 할 수 있음
- S번 건물에서 출발해서 N개의 회의를 모두 마친 다음 다시 S번 건물로 돌아오는 최소 비용을 구하는 문제
- 같은 정점을 여러 번 방문해도 됨

최소 신장 트리 - 예시

BOJ 28119 Traveling SCCC President

- 모든 정점을 최소한 한 번씩 간선을 통해 방문해야 하므로 정답은 최소 신장 트리보다 크거나 같음
- 최소 비용으로 모든 정점을 한 번씩 방문하는 방법
 - MST를 만든 다음, DFS 하듯이 이동하면 MST의 간선만 사용해 모든 정점을 방문할 수 있음
 - 자식 정점으로 내려가는 것은 간선을 통해서, 이전 정점으로 돌아가는 것은 순간이동으로...
- 모든 정점을 한 번씩 다 방문한 다음에는 매번 순간이동으로 이동할 수 있음
- 따라서 정답은 최소 신장 트리의 가중치와 같음