#1. 그리디 기법

나정휘 (jhnah917)
https://justiceHui.github.io/

귀류법

귀류법

- 명제의 결론이 부정이라고 가정했을 때 모순이 발생함을 보여 원래 명제가 참임을 증명

- 모든 n에 대해 P(n)이 참임을 증명
  - P(n)이 거짓이 되는 n이 있다고 가정
  - P(n)이 거짓이 되는 n이 있으면 모순이 일어나는 것을 보임
  - P(n)이 거짓이 되는 n이 없으므로 모든 n에 대해 P(n)은 참

귀류법

n > 1인 정수의 소인수분해가 유일함을 증명

- 소인수분해의 존재성은 강한 수학적 귀납법을 이용해 증명할 수 있음
- 유일하게 존재함을 증명해 보자

- (Euclid Lemma) $a \mid bc$ 이고 $\gcd(a, b) = 1$이면 $a \mid c$임

- n의 서로 다른 소인수분해 $n = p_1p_2...p_s = q_1q_2...q_t$가 존재한다고 가정하자.
  - 일반성을 잃지 않고 $s \geq t$인 경우만 생각
  - $q_1 \mid n = p_1p_2p_3...p_s$ 인데 p_1과 q_1은 모두 소수이므로 $q_1 \mid p_1$이거나 $q_1 \mid p_2p_3...p_s$
  - 동일한 논리로 $q_1 \mid p_1$ or $q_1 \mid p_2$ or $q_1 \mid p_3$ or ... or $q_1 \mid p_s$
  - 따라서 $q_1 = p_i$인 i가 존재하고, $q_1 = p_1$이 되도록 p를 재배열하자.

  - 비슷하게 $q_2 = p_2$, $q_3 = p_3$, ... 이 되도록 p를 재배열할 수 있음
  - 만약 $s > t$ 이면 $p_{t+1}p_{t+2}...p_s \neq 1$이므로 $s = t$가 되어야 함
  - 모든 $1 \leq i \leq t$에 대해 $p_i = q_i$이므로 소인수분해는 유일하게 존재함

귀류법

√3이 유리수가 아님을 증명

- √3이 유리수라고 가정하자.
- √3은 n,m > 0이고 gcd(n, m) = 1인 두 정수 n, m을 이용해 $\sqrt{3} = m/n$으로 나타낼 수 있음
- 양변을 제곱하고 이항하면 $3n^2 = m^2 > 1$

- m > 1이므로 m을 소수들의 곱 $m = p_1p_2...p_k$으로 나타낼 수 있음
- $3 \mid m^2$ 이므로 $3 \mid p_1^2p_2^2...p_k^2$, 유클리드 보조 정리에 의해 $3 = p_1$이 되도록 p를 재배열할 수 있음

- $3n^2 = 3^2p_2^2p_3^2...p_k^2$ 이므로 $n^2 = 3p_2^2p_3^2...p_k^2$
- $3 \mid n^2$ 이므로 유클리드 보조 정리에 의해 3 | n
- n과 m 모두 3의 배수이므로 gcd(n, m) ≥ 3

- gcd(n, m) = 1이라고 가정했는데 gcd(n, m) ≥ 3 이므로 모순 발생
- 따라서 √3은 유리수가 아님

귀류법

소수는 무한히 많음을 증명

- 소수가 유한하다고 가정하고, 그 소수를 $p_1 \le p_2 \le \cdots \le p_k$라고 하자.
- $n = p_1p_2\cdots p_k + 1 > 1$ 을 생각해 보자.
- 산술의 기본 정리에 의해 $n = q_1q_2...q_j$ 로 나타낼 수 있음 (모든 $1 \le i \le j$ 에 대해 $q_i \in \{p_1, p_2, \dots, p_k\}$)

- n과 $p_1p_2...p_k$ 모두 q_1의 약수이므로 $n - p_1p_2...p_k$도 q_1의 약수임
- 따라서 $(n - p_1p_2...p_k) / q_1 = 1 / q_1$이 정수

- $q_1 \neq 1$이므로 $1 / q_1$이 정수라는 것에 모순 발생

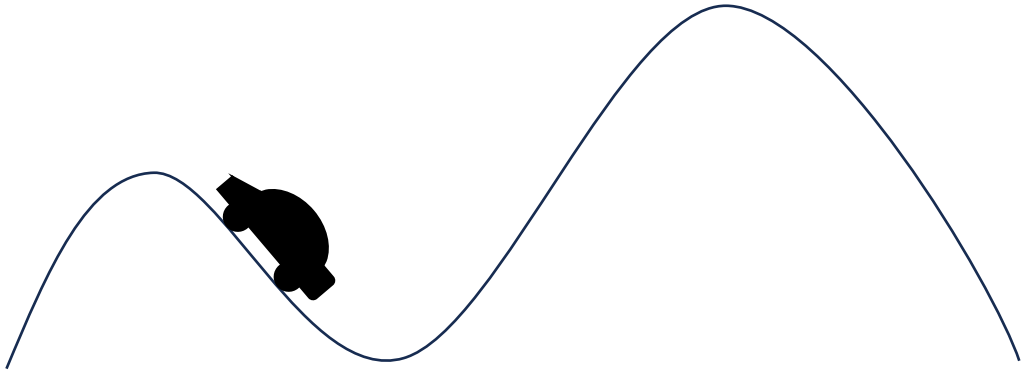
그리디

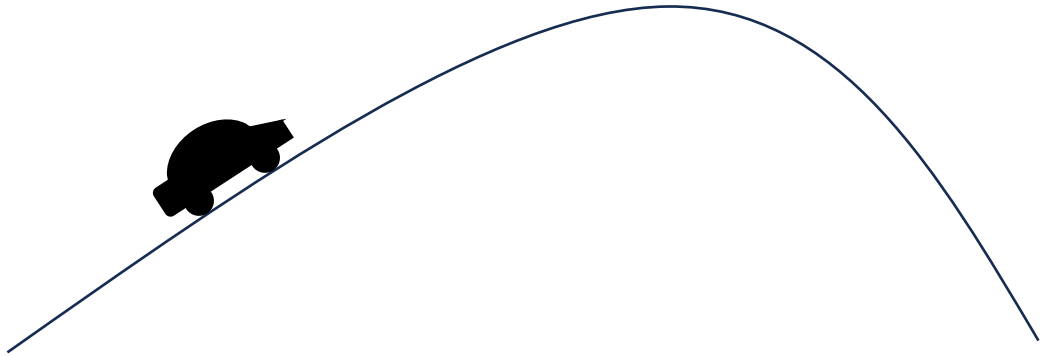
그리디

- Greedy: [형용사] 탐욕스러운
- 현재 상태에서 가장 좋은 선택을 하는 방법
  - 현재 상태에서 가장 좋은 선택이 전체 범위에서도 가장 좋은 선택이라는 것은 보장 못함
  - 만약 이 전략이 전체 범위에서도 최적이라면, 그리디 기법으로 문제를 해결할 수 있음

- 동적 계획법 vs 그리디 기법
  - 동적 계획법: D[i]를 계산할 때 $1 \le j < i$인 모든 D[j]를 확인
  - 그리디 기법: D[i]를 계산할 때 적당한 D[j] 하나만 확인

- Local Optimum vs Global Optimum





그리디 예시 - 1

BOJ 11047 동전 0

- N가지 종류의 동전을 사용해서 K원을 만들어야 함
- 사용하는 동전의 개수를 최소화
- 동전은 서로 약수/배수 관계, 1원 짜리 동전 항상 존재

- 가격이 큰 동전부터 최대한 많이 사용하는 전략
  - 증명?

그리디 예시 - 1

BOJ 11047 동전 0

- 그리디 기법으로 구한 답보다 더 좋은 해가 없다는 것을 증명
  - 가격 내림차순으로 동전을 정렬하자 : C[i] = i번째로 비싼 동전의 금액
  - 그리디에서 각 동전을 사용한 개수를 저장한 리스트 A
  - 실제 최적해에서 각 동전을 사용한 개수를 저장한 리스트 B

  - (귀류법) A보다 더 적은 동전을 사용한 최적해 B가 존재한다고 가정
    - A와 B가 처음으로 다른 지점 i가 존재
    - 비싼 동전부터 최대한 많이 가져가는 전략에 의해 A[i] > B[i]
    - C[i] * (A[i] - B[i])원 만큼의 차이를 채우기 위해 C[i]보다 싼 동전을 더 사용할 텐데
    - C[i]의 약수면서 싼 동전으로 C[i] * (A[i] - B[i])원을 만드는데 A[i]-B[i]보다 많은 동전이 필요하므로
    - B는 A보다 동전을 더 적게 사용할 수 없다.

그리디 예시 - 2

BOJ 25312 200% Mixed Juice!

- N개의 물건이 있고, i번째 물건의 무게는 W_i, 가격은 P_i원
- 무게의 합이 Xkg 이하가 되도록 물건을 적당히 선택할 때
- 가능한 가격의 합의 최댓값을 구하는 문제
- 단, 물건을 원하는 대로 분할할 수 있음
  - (W_i, P_i)를 무게가 x, W_i-x인 물건으로 분할하면
  - (x, P_i/W_i*x), (W_i-x, P_i/W_i*(W_i-x))로 분할됨

- 무게 대비 가격(효율, P_i/W_i)이 높은 물건부터 가져가는 전략

그리디 예시 – 2

BOJ 25312 200% Mixed Juice!

- 그리디 기법으로 구한 답보다 더 좋은 해가 없다는 것을 증명
  - 가격이 0 이하인 물건은 신경 쓰지 않아도 됨
  - 무게의 합이 X 이하인 경우, 모두 가져가는 것이 최적
    - 그리디 알고리즘은 이 경우를 잘 처리함
  - 그렇지 않은 경우, 무게의 합이 X가 되도록 가져가는 것이 최적
    - 그리디 알고리즘은 이 경우 무게의 합이 X가 됨
    - 그리디 알고리즘이 가격을 최대화하는지 증명하자.

그리디 예시 – 2

BOJ 25312 200% Mixed Juice!

- 증명 (cont.)
  - 물건을 효율에 대한 내림차순으로 정렬하자. 편의상 효율이 모두 다르다고 가정한다.
  - 그리디에서 각 물건을 가져간 무게를 저장한 리스트 A, 실제 최적해의 리스트 B

  - (귀류법) A보다 가격이 더 비싼 **최적해** B가 존재한다고 가정
    - A와 B가 처음으로 달라지는 지점 i가 존재
    - 효율이 높은 것부터 최대한 많이 가져가는 그리디 알고리즘에 의해 A[i] > B[i]
    - B는 최적해이기 때문에 A[j] < B[j]인 j(> i)가 존재

    - 새로운 해 B'를 구성한다. B'[i] = B[i] + eps, B'[j] = B[j] – eps이고, 다른 모든 B'[k] = B[k]이다.
    - B와 B'의 무게는 동일하지만 가격은 B'가 더 비싸다.
    - B가 최적해라는 가정에 모순이 생겼으므로 A보다 더 비싼 최적해는 존재하지 않는다.

그리디 예시 – 3

BOJ 11399 ATM

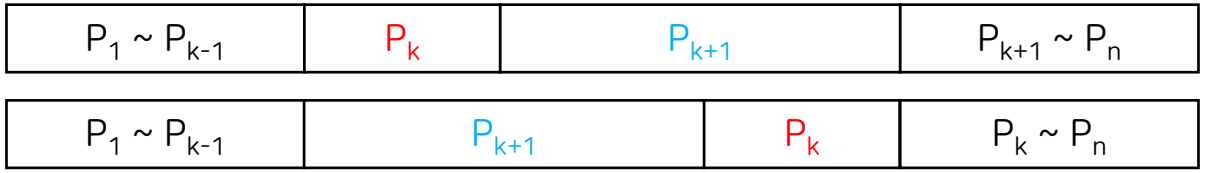
- i번째 사람은 ATM기를 P_i 시간 동안 점유함
- 모든 사람의 대기 시간의 합을 최소로 하는 처리 순서를 찾는 문제

- 소요 시간이 짧은 사람부터 처리하면 됨

그리디 예시 – 3

BOJ 11399 ATM

- P_i가 작은 사람부터 처리하는 것이 최적임을 증명
  - 두 사람 P_k, P_{k+1} 중 먼저 처리해야 하는 것을 결정하자.
  - 인접한 원소의 순서만 잘 결정하면, 버블 정렬 느낌으로 전체 원소를 정렬할 수 있음



  - P_k와 P_{k+1}의 대기 시간만 보면 됨
  - 위 : $\text{sum}(P_1 \sim P_{k-1}) + \text{sum}(P_1 \sim P_{k-1}) + P_k$
  - 아래 : $\text{sum}(P_1 \sim P_{k-1}) + \text{sum}(P_1 \sim P_{k-1}) + P_{k+1}$
  - 동류항 없애면 각각 P_k, P_{k+1}만 남음
  - 그러므로 $P_k < P_{k+1}$이면 P_k가 먼저 오도록 정렬하는 것이 이득

그리디 예시 - 4

BOJ 1931 회의실 배정

- 한 개의 회의실과 N개의 회의가 있음
- 각 회의는 시작 시간과 종료 시간이 있어서, 해당 기간 동안 회의실을 점유함
- 몇 개의 회의를 선택해서 서로 겹치지 않도록 회의를 진행할 때
- 진행 가능한 회의의 최대 개수를 구하는 문제

- 끝나는 시간이 빠른 회의부터 선택하는 그리디 전략으로 답을 구할 수 있음

그리디 예시 - 4

BOJ 1951 회의실 배정

- 종료 시간이 가장 빠른 회의를 포함하는 최적해가 있다는 것을 증명
  - 종료 시간이 가장 빠른 회의 m을 포함하지 않는 최적해 O가 존재한다고 하자.
  - 당연히 O에는 겹치는 회의가 존재하지 않는다.
  - O에서 종료 시간이 가장 빠른 회의 x를 제거하고 m을 추가한 O'를 생각해보자.
  - O와 O'의 크기는 동일하고, O'에는 겹치는 회의가 존재하지 않는다.
  - 그러므로 모든 최적해는 m을 포함하도록 바꿀 수 있다.

  - 종료 시간이 가장 빠른 회의를 선택한 뒤
  - 그 회의와 겹치는 회의를 모두 제거하고
  - 다시 종료 시간이 가장 빠른 회의를 선택하는 방식

정리

지금까지 본 증명 방법

- A보다 좋은 최적해 B가 없음을 보임
  - ex. 동전 0
- A보다 좋은 최적해 B가 있다고 가정한 뒤, B가 최적해가 아님을 보임
  - ex. 200% Mixed Juice! (Fractional Knapsack Problem)
- 원소의 우선순위를 정한 뒤, 우선순위가 높은 것부터 선택하는 것이 최적임을 보임
  - ex. ATM (SJF Scheduling)
- 어떤 원소를 포함하는 최적해가 항상 존재함을 보임
  - ex. 회의실 배정

Exchange Argument

Exchange Argument

- BOJ 11399 ATM의 증명 방법을 일반화한 것이라고 생각할 수 있음

- 특정 비교 기준으로 봤을 때 inversion이 없으면 최적임을 증명할 수 있다면
- 인접한 두 원소의 순서를 결정하는 것으로 전체 원소의 순서를 결정할 수 있음
  - ex. 버블 정렬, 삽입 정렬, ...

- 버블 정렬과 다른 일반적인 비교 기반 정렬의 결과물을 동일하므로
  - ex. quick sort, heap sort, merge sort, ...
- 비교 함수를 잘 작성한 뒤 std::sort를 사용하면 됨

그리디 예시 - 5

BOJ 14908 구두 수선공

- i번째 작업을 수행하는데 T_i일 걸림
- i번째 작업의 완료가 하루 지연될 때마다 S_i원 벌금 내야 함
- 벌금을 최소로 하는 작업 순서를 정하는 문제

그리디 예시 - 5

BOJ 14908 구두 수선공

- $\text{sum}(T_{1..k-1}) * S_k + (\text{sum}(T_{1..k-1}) + T_k) * S_{k+1}$
- $\text{sum}(T_{1..k-1}) * S_{k+1} + (\text{sum}(T_{1..k-1}) + T_{k+1}) * S_k$

- k번째 원소가 k+1번째보다 먼저 오기 위해서는 $T_k * S_{k+1} \le T_{k+1} * S_k$가 되어야 함
- $T_k / S_k \le T_{k+1} / S_{k+1}$이 되도록 정렬해야 함
- T_i / S_i 오름차순 정렬

그리디 예시 – 6

BOJ 2180 소방서의 고민

- 일차함수 $f_i(x) = a_i x + b_i$가 여러 개 주어짐 ($a_i, b_i > 0$)
- $x = 0$에서 시작해서 $x \leftarrow x + f_i(x)$를 한 번씩 적용할 때
- 최종 결과의 최솟값을 구하는 문제

그리디 예시 – 6

BOJ 2180 소방서의 고민

- $(a_{k+1}+1)\ ((a_k+1)x + b_k) + b_{k+1}$
- $(a_k+1)\ ((a_{k+1}+1)x + b_{k+1}) + b_k$

x	$a_kx + b_k$	$a_{k+1}x + b_{k+1}$	f

x	$a_{k+1}x + b_{k+1}$	$a_kx + b_k$	f

- $a_{k+1}x + a_ka_{k+1}x + a_{k+1}b_k + x + a_kx + b_k + b_{k+1}$
- $a_kx + a_ka_{k+1}x + a_kb_{k+1} + x + a_{k+1}x + b_{k+1} + b_k$

- $a_{k+1}b_k \le a_kb_{k+1}$이 되도록 정렬
- $b_k / a_k \le b_{k+1} / a_{k+1}$이 되도록 정렬
- b_i / a_i 오름차순 정렬

#2. 그리디 + 우선순위 큐

나정휘 (jhnah917)
https://justiceHui.github.io/

이진 힙

우선순위 큐 (Priority Queue)

- Queue: 먼저 들어온 데이터가 먼저 나가는 자료구조
- Priority Queue: 우선순위가 높은 데이터가 먼저 나가는 자료구조
  - 원소 삽입
  - 가장 큰 원소 검색
  - 가장 큰 원소 제거

- 목표: N개의 데이터가 있을 때 세 연산을 모두 O(log N) 정도에 처리하는 것

이진 힙

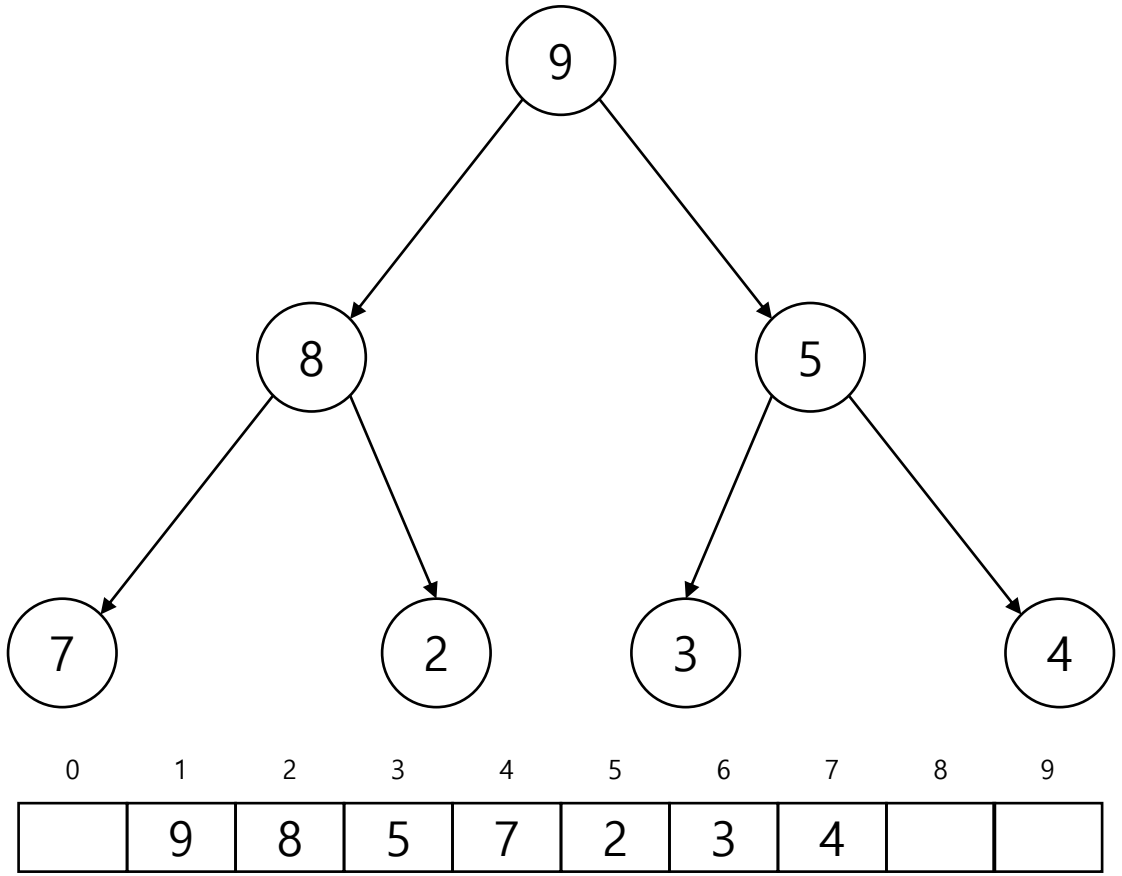
힙 구조

- 각 정점의 값이 자식 정점보다 큰 트리 구조

- 주로 사용하는 건 이진 힙 (Binary Heap)
  - 완전 이진 트리(Complete Binary Tree) 형태
  - 루트 : 1번 인덱스
  - x의 부모 : x / 2
  - x의 왼쪽/오른쪽 자식 : 2x, 2x+1

- 이진 힙 외에도 다양한 힙 자료구조가 있음
  - binomial heap, leftist heap, pairing heap, fibonacci heap, thin heap, ...

이진 힙



이진 힙

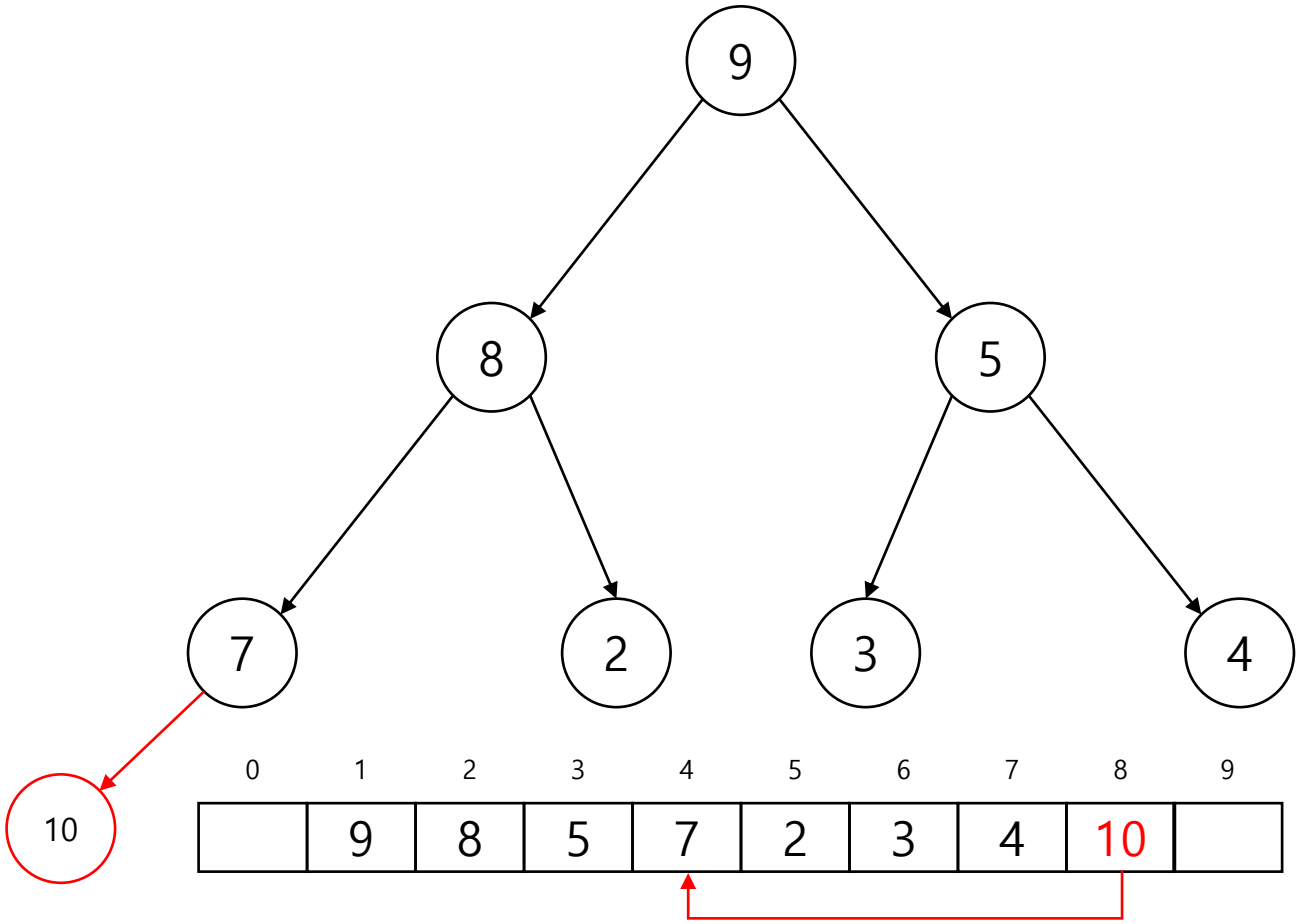
원소 삽입

- 배열의 맨 뒤에 원소 추가
- 만약 부모가 더 작으면 부모와 교환

이진 힙

원소 삽입

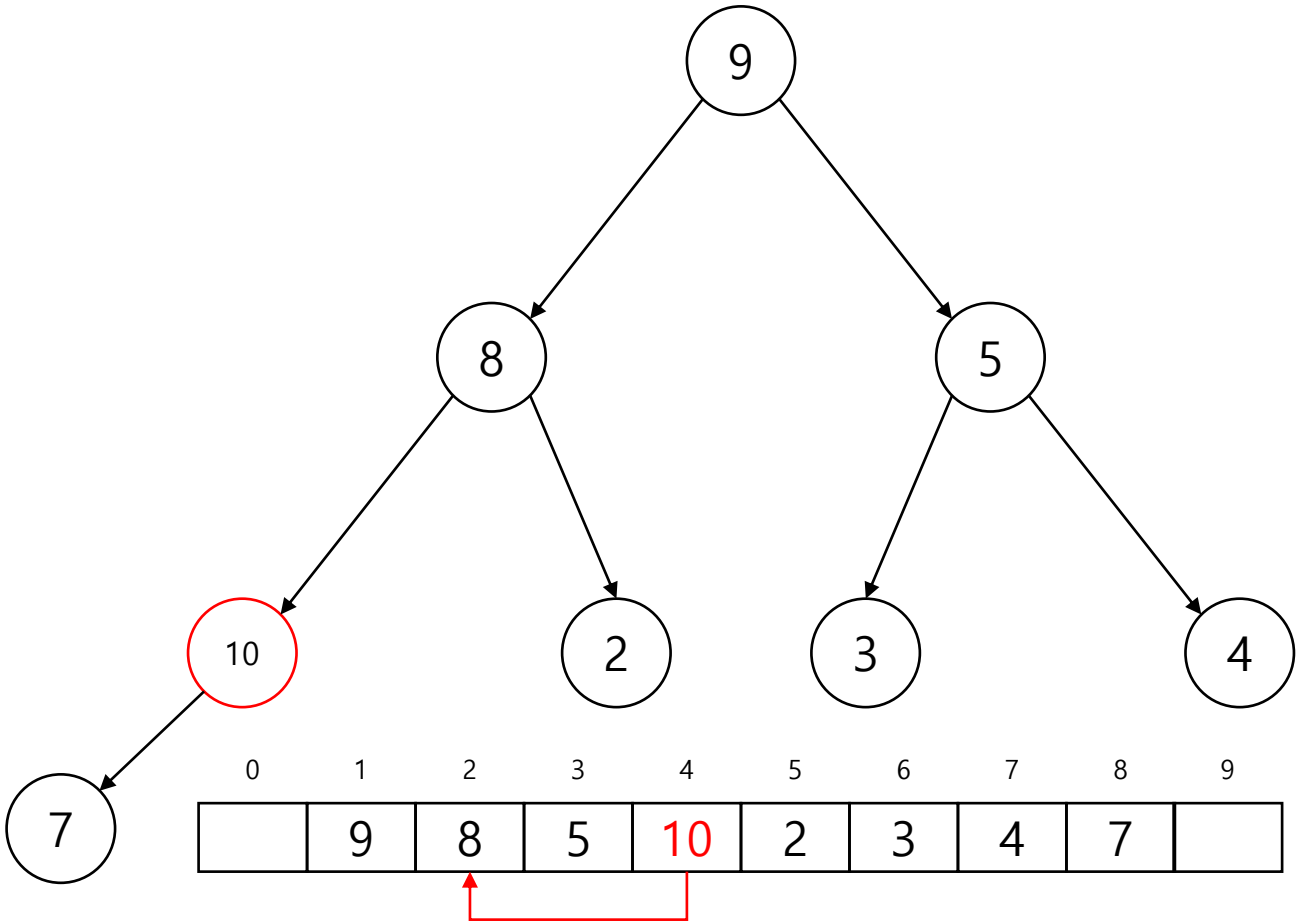
- INSERT 10



이진 힙

원소 삽입

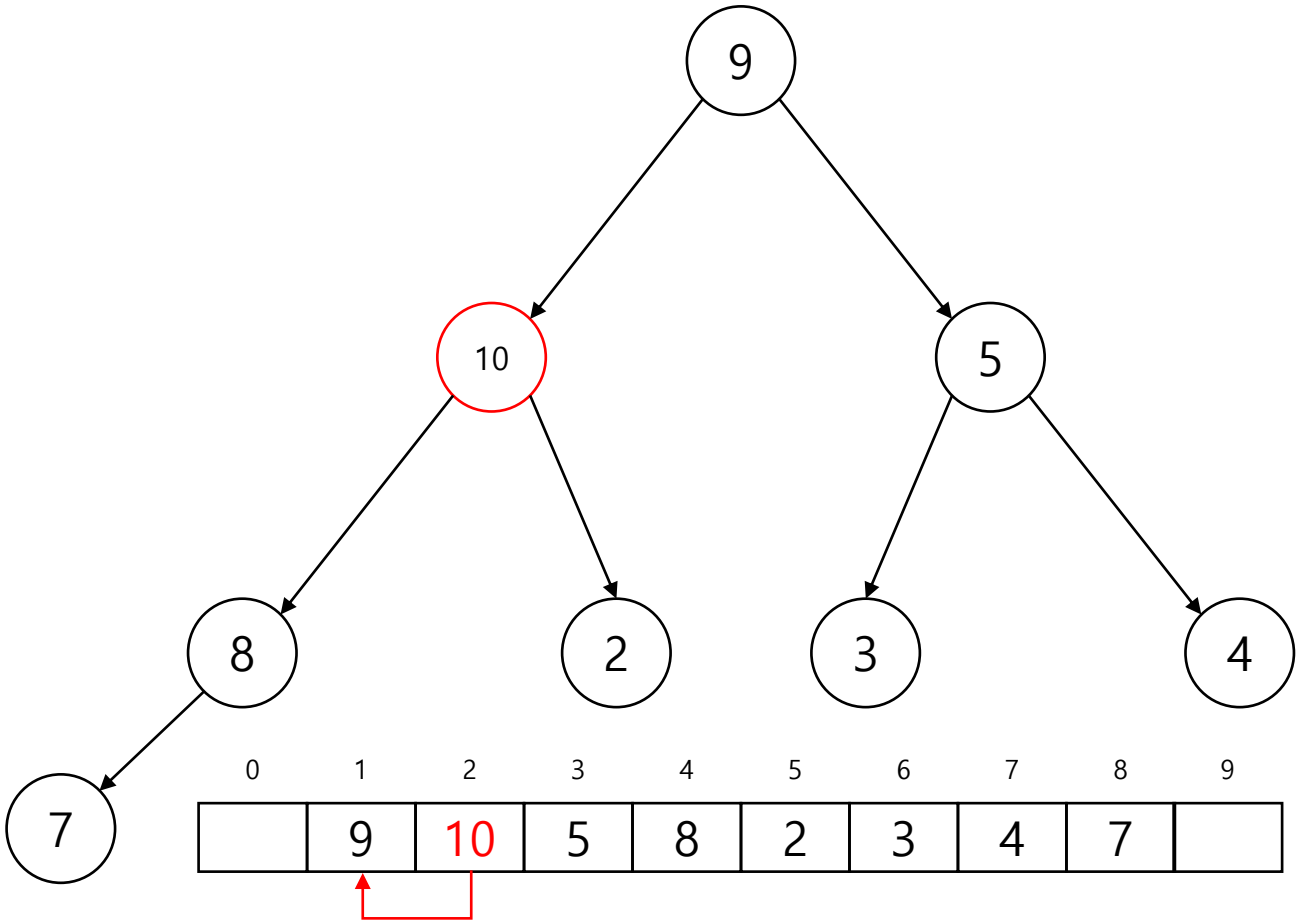
- INSERT 10



이진 힙

원소 삽입

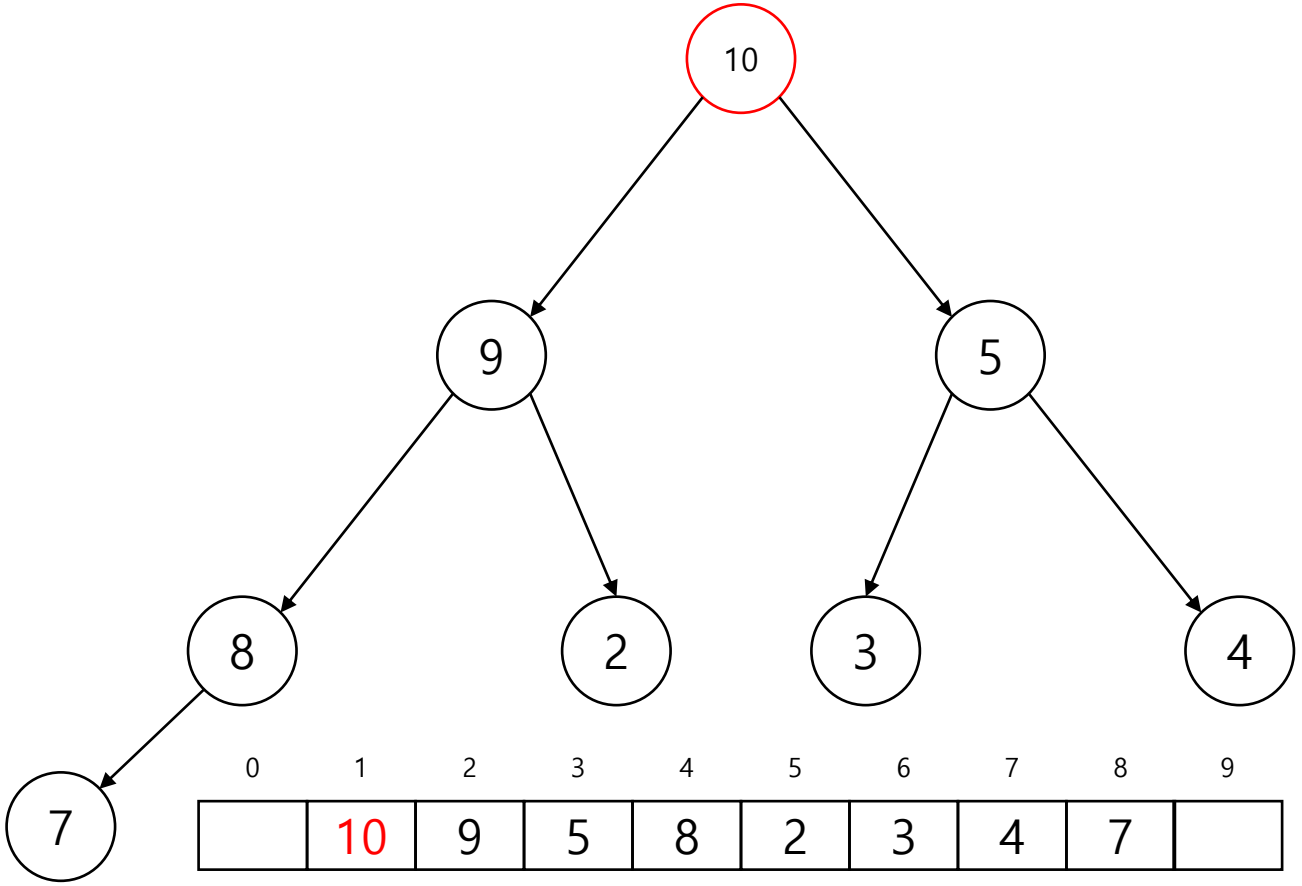
- INSERT 10



이진 힙

원소 삽입

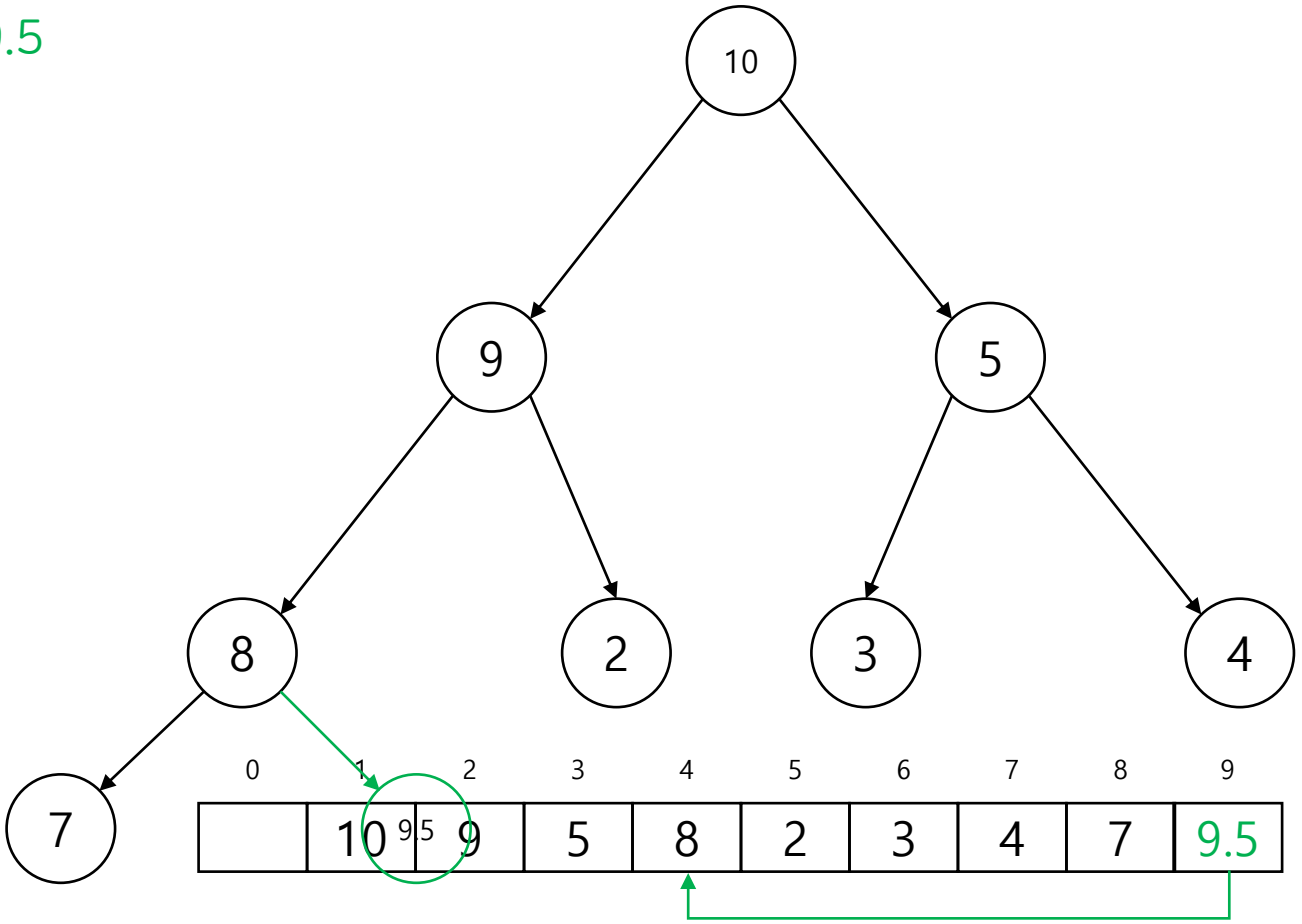
- INSERT 10



이진 힙

원소 삽입

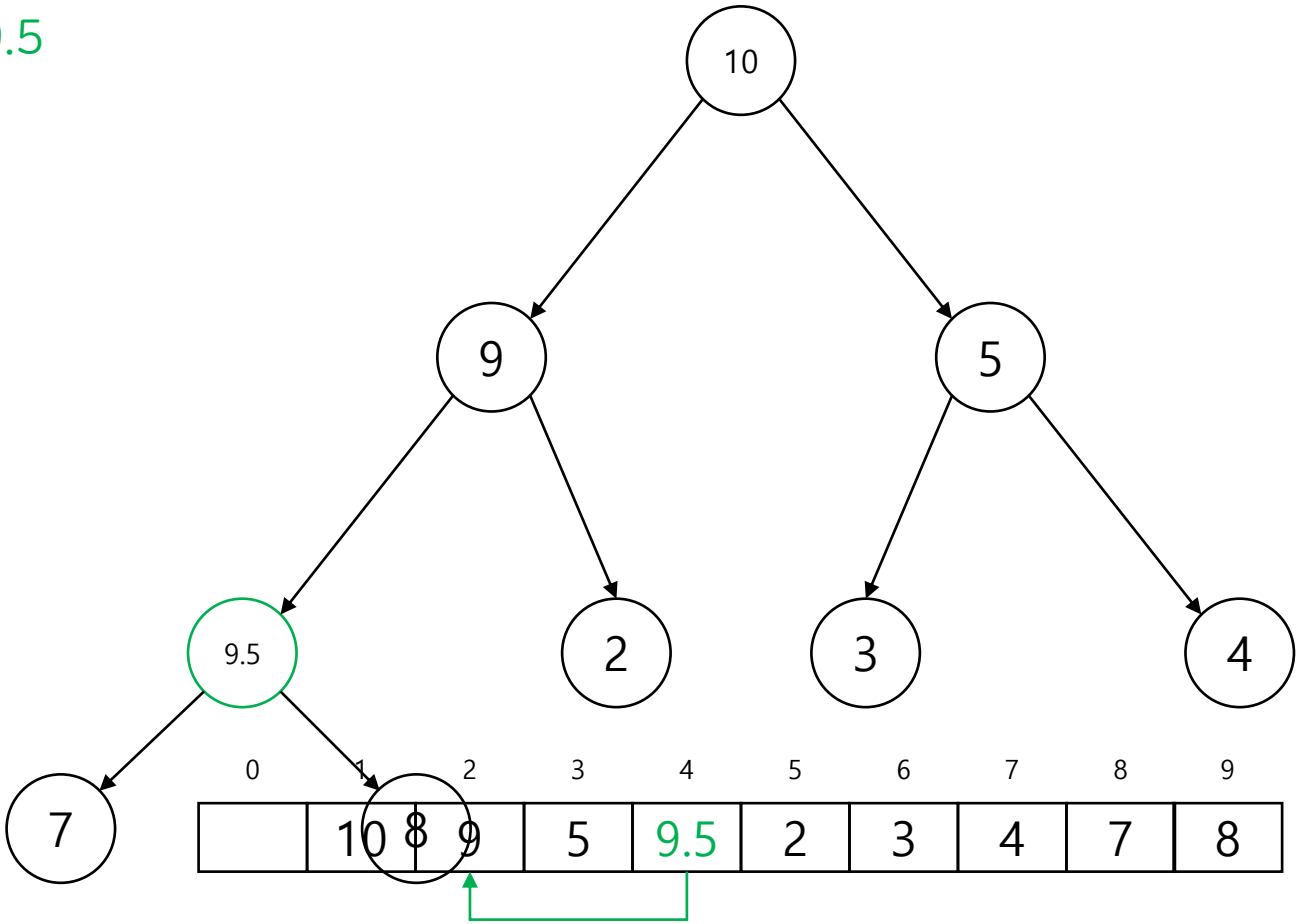
- INSERT 10
- INSERT 9.5



이진 힙

원소 삽입

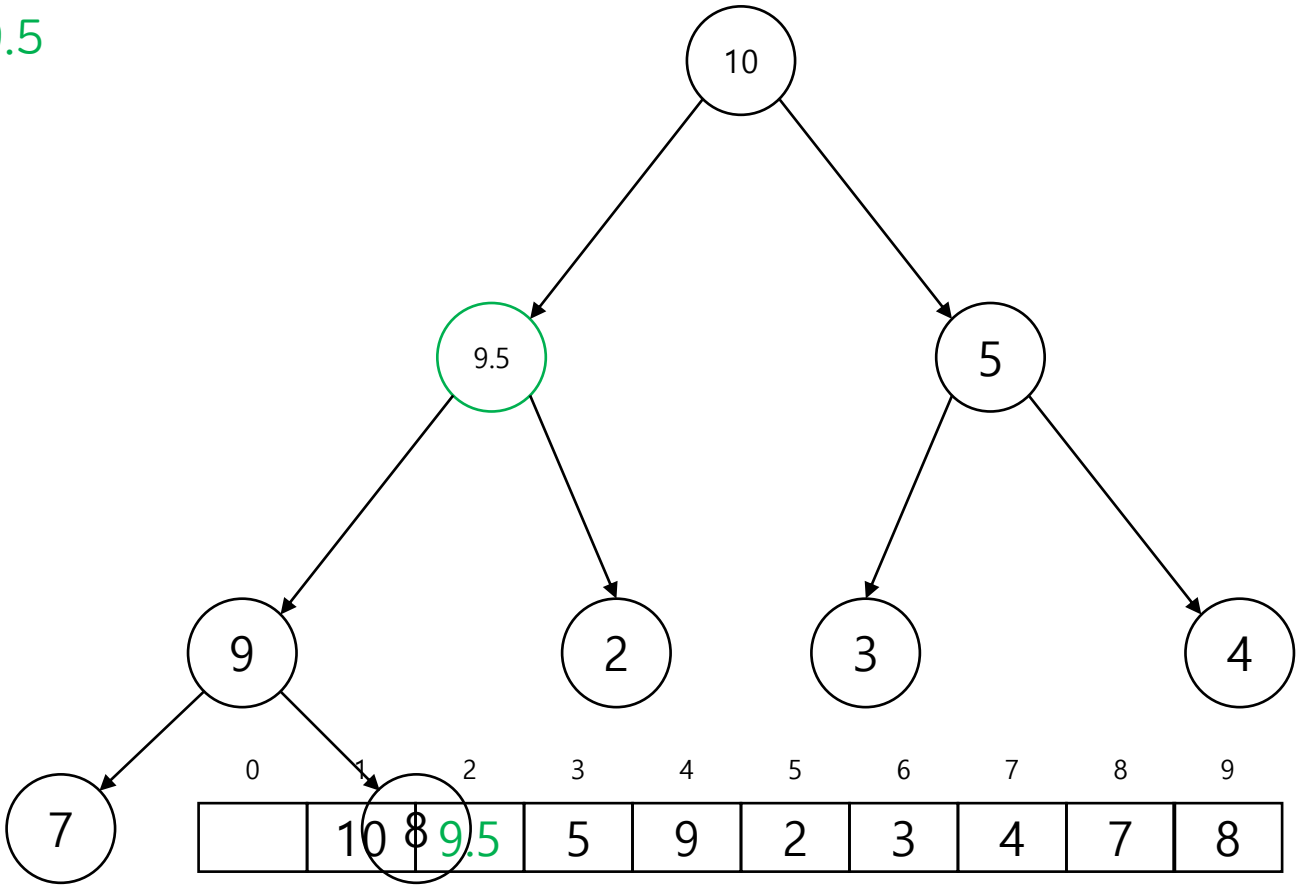
- INSERT 10
- INSERT 9.5



이진 힙

원소 삽입

- INSERT 10
- INSERT 9.5



이진 힙

원소 삽입

- 시간 복잡도 : O(h) = O(log N)

가장 큰 원소 탐색

- return heap[1];
- 시간 복잡도 : O(1)

이진 힙

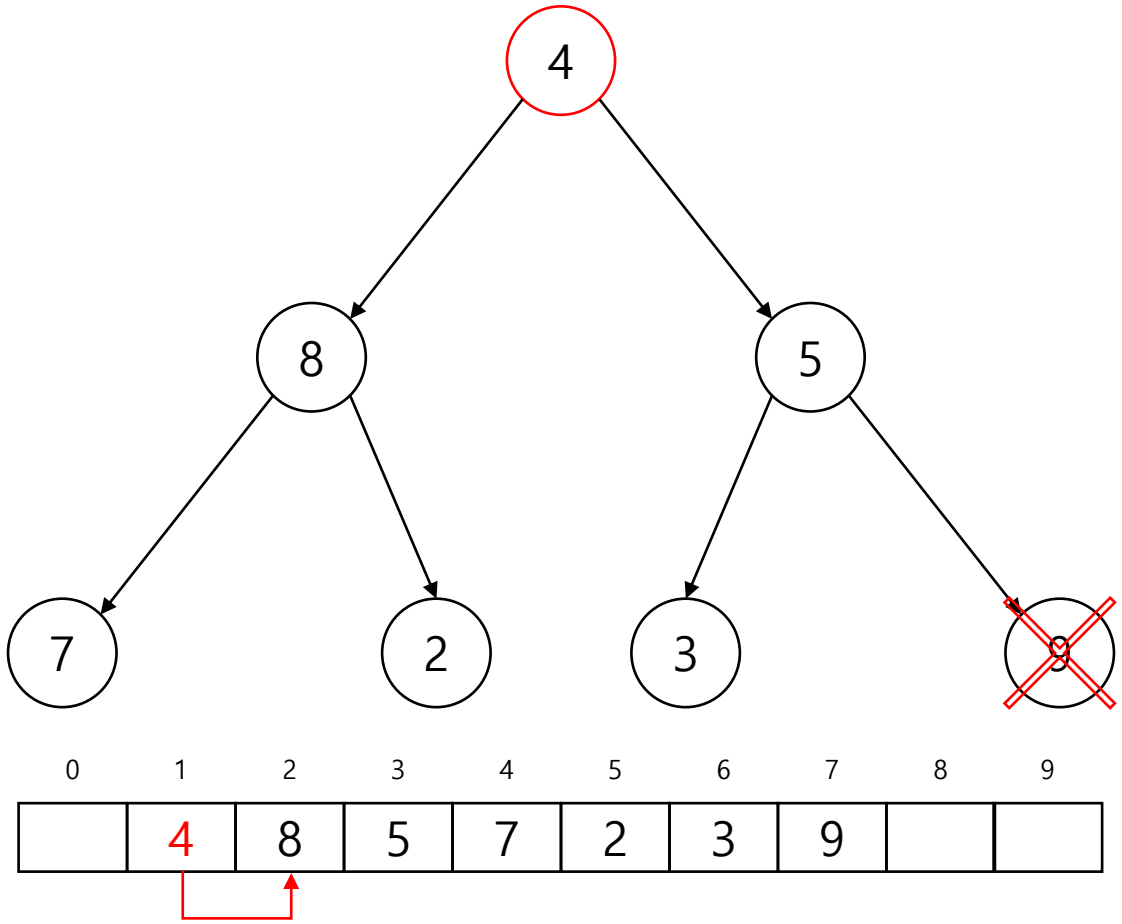
가장 큰 원소 제거

- 루트(가장 큰 원소)와 마지막 정점의 값을 바꿈
- 마지막 정점 제거
- 현재 루트에 있는 값이 자식보다 작으면 밑으로 내림
  - 두 자식 모두 현재 정점보다 크면 더 큰 방향으로 이동

이진 힙

가장 큰 원소 제거

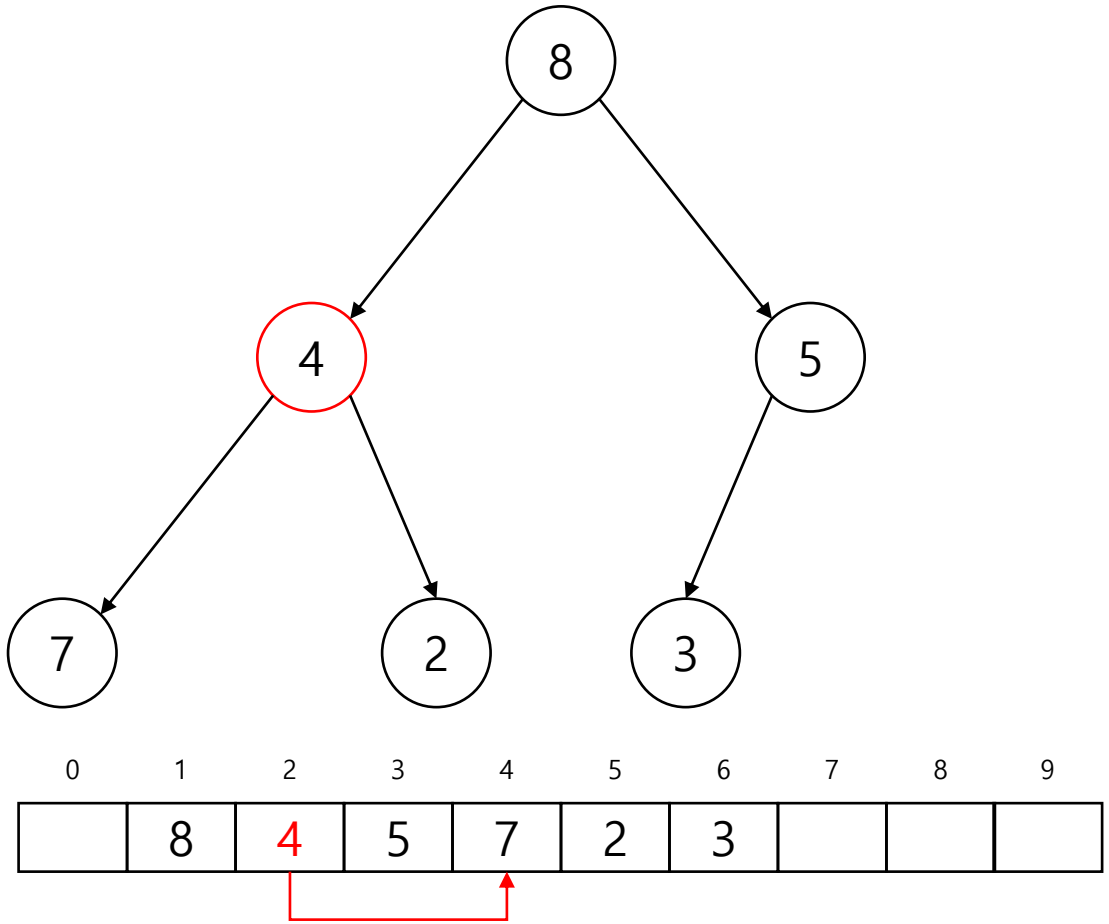
- POP 9



이진 힙

가장 큰 원소 제거

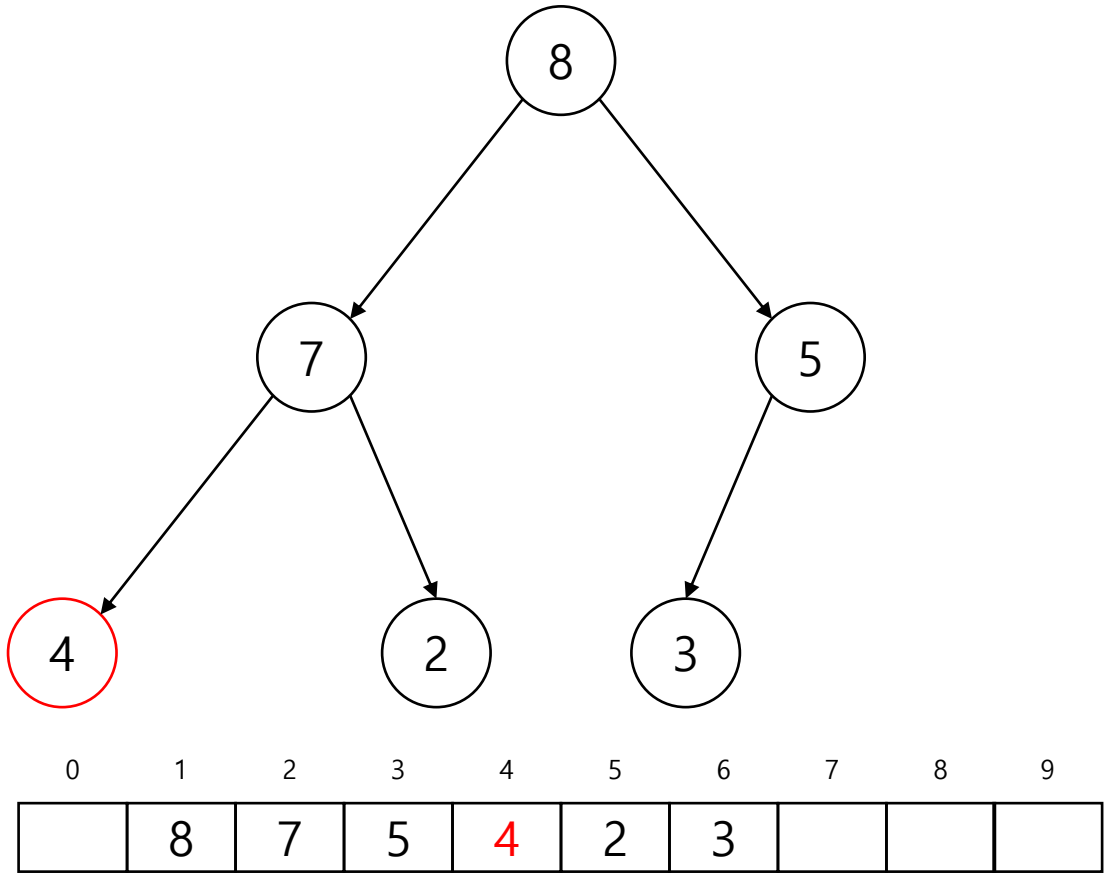
- POP 9



이진 힙

가장 큰 원소 제거

- POP 9



이진 힙

가장 큰 원소 제거

- 시간 복잡도 : O(h) = O(log N)

이진 힙

구현

```cpp
int Heap[101010], sz = 0;

void push(int v){
    Heap[++sz] = v;
    for(int i=sz; i>1; i>>=1){
        if(Heap[i] > Heap[i/2]) swap(Heap[i], Heap[i/2]);
        else break;
    }
}

int pop(){
    swap(Heap[1], Heap[sz--]);
    for(int i=1; i*2<=sz; ){
        int ch = i * 2;
        if(ch+1 <= sz && Heap[ch+1] > Heap[ch]) ch += 1;
        if(Heap[ch] > Heap[i]) swap(Heap[ch], Heap[i]), i = ch;
        else break;
    }
    return Heap[sz+1];
}
```

우선순위 큐

std::priority_queue

- #include <queue>
- 이진 힙으로 구현되어 있음
- 기본값은 max heap – 가장 큰 원소가 맨 위에 있음
  - min heap: priority_queue<int, vector<int>, greater<>> pq;
  - max heap: priority_queue<int, vector<int>, less<>> pq;
- pq.push(x)
- pq.pop(), pq.top()
- pq.size(), pq.empty()

그리디 예시 - 7

허프만 코드

- N가지의 문자로 구성된 텍스트 파일을 압축해야 함
- i번째 문자의 등장 횟수는 F[i]이고, 각 문자를 적당한 이진수로 표현해서 파일의 용량을 최소화하려고 함
- 즉, 심볼들의 출현 빈도가 정해져 있을 때, 심볼 단위로 인코딩해서 전체 길이를 최소화하는 문제

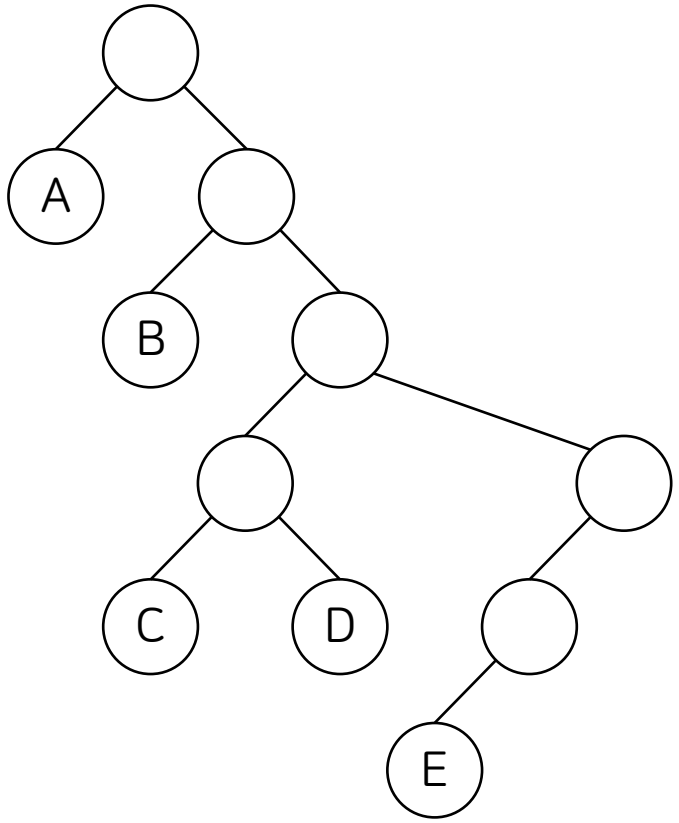
- prefix code: 어떠한 코드도 다른 코드의 prefix가 되지 않는 코드
  - 0, 10, 1100, 1101, 11100 은 prefix code
  - 0, 10, 110, 1100 은 prefix code가 아님

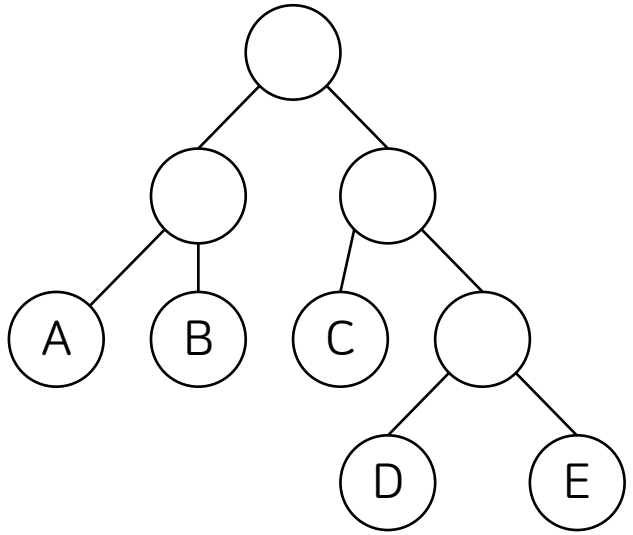
- prefix code의 장점: 앞에서부터 그리디하게 매칭하는 방식으로 디코딩할 수 있음
  - 앞에서부터 한 글자씩 읽으면서, 처음으로 완성되는 코드를 가져가는 방식으로 디코딩 가능
  - ex. 등장할 수 있는 코드가 0, 10, 1100, 1101, 11100
  - 인코딩된 문자열이 01100101101111100010 이라면?
  - 0 / 1100 / 10 / 1101 / 11100 / 0 / 10 으로 나눠서 디코딩하면 됨

그리디 예시 – 7

허프만 코드

- prefix code tree
  - 0, 10, 1100, 1101, 11100 ACBDEAB 0 1100 10 1101 11100 0 10
  - 00, 01, 10, 110, 111 ACBDEAB 00 10 01 110 111 00 01
  - 가중치 합이 최소가 되도록 트리를 구성하는 문제

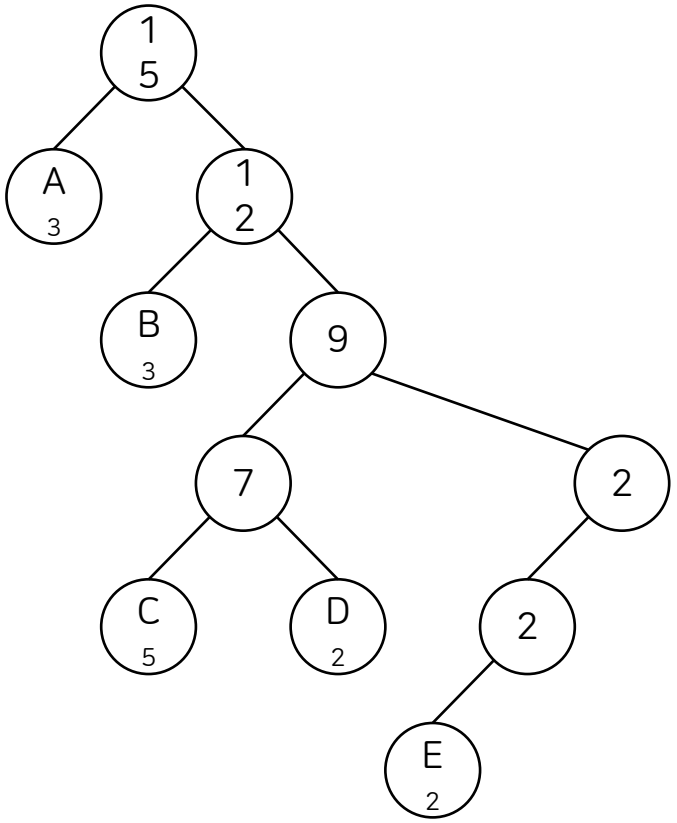


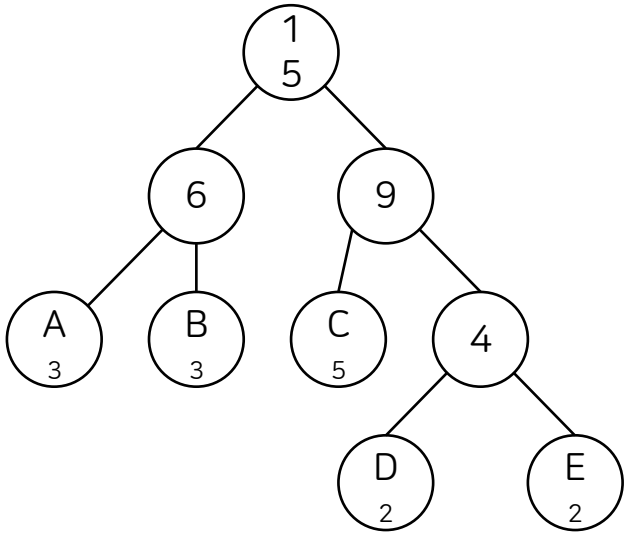


그리디 예시 - 7

허프만 코드

- 가중치 합이 최소가 되도록 트리를 구성하는 문제
  - 가중치: 각 정점마다, 그 정점을 루트로 하는 서브 트리 아래에 있는 심볼의 등장 횟수의 합
  - F[A] = 3, F[B] = 3, F[C] = 5, F[D] = 2, F[E] = 2 이면?

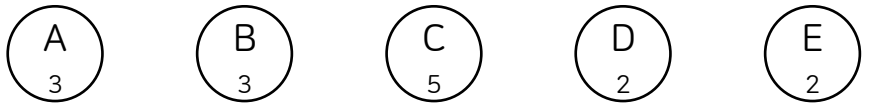




그리디 예시 - 7

허프만 코드

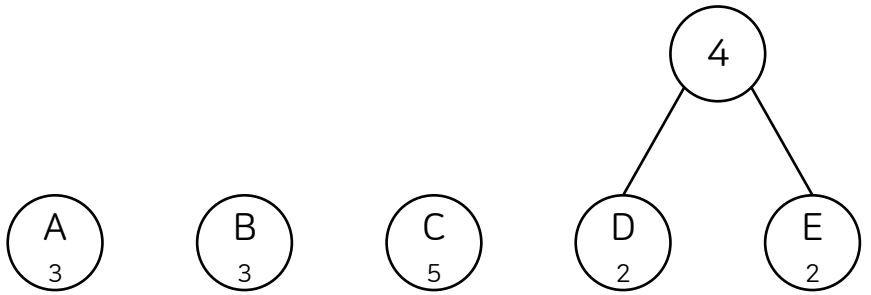
- 그리디 알고리즘
  - 처음에는 각 심볼마다 자신만 있는 트리로 초기화
  - 아직 트리가 2개 이상 있으면, 가중치가 가장 작은 두 루트 정점을 합쳐서 새로운 트리를 만드는 것을 반복



그리디 예시 - 7

허프만 코드

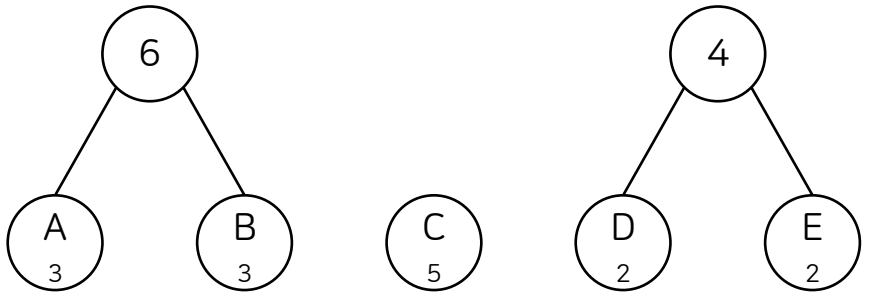
- 그리디 알고리즘
  - 처음에는 각 심볼마다 자신만 있는 트리로 초기화
  - 아직 트리가 2개 이상 있으면, 가중치가 가장 작은 두 루트 정점을 합쳐서 새로운 트리를 만드는 것을 반복



그리디 예시 - 7

허프만 코드

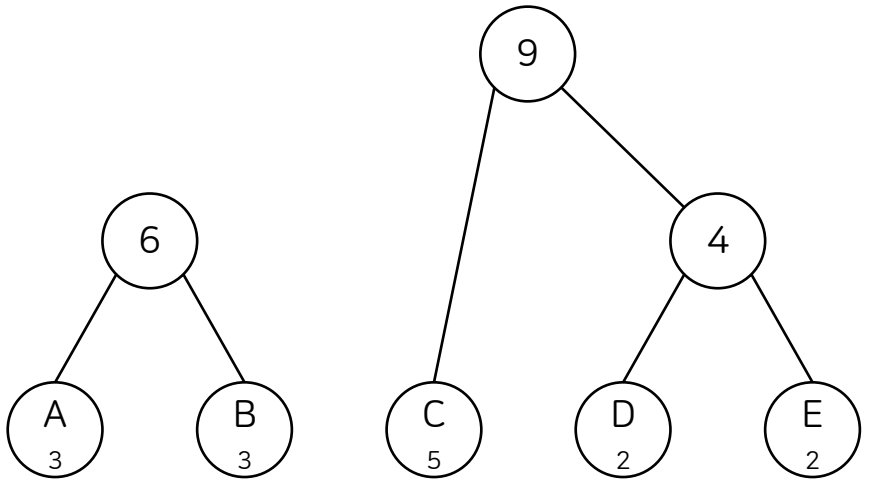
- 그리디 알고리즘
  - 처음에는 각 심볼마다 자신만 있는 트리로 초기화
  - 아직 트리가 2개 이상 있으면, 가중치가 가장 작은 두 루트 정점을 합쳐서 새로운 트리를 만드는 것을 반복



그리디 예시 - 7

허프만 코드

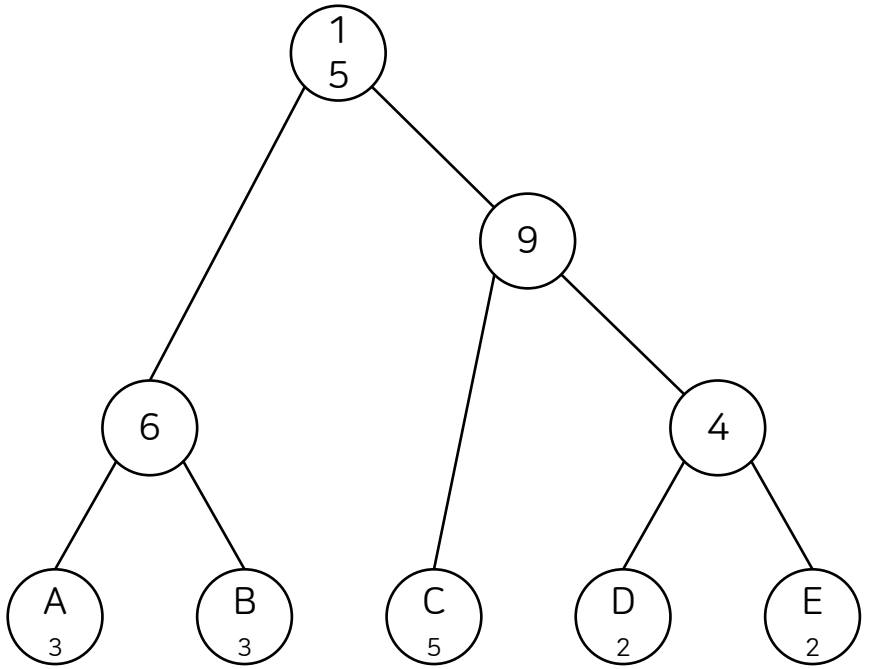
- 그리디 알고리즘
  - 처음에는 각 심볼마다 자신만 있는 트리로 초기화
  - 아직 트리가 2개 이상 있으면, 가중치가 가장 작은 두 루트 정점을 합쳐서 새로운 트리를 만드는 것을 반복



그리디 예시 - 7

허프만 코드

- 그리디 알고리즘
  - 처음에는 각 심볼마다 자신만 있는 트리로 초기화
  - 아직 트리가 2개 이상 있으면, 가중치가 가장 작은 두 루트 정점을 합쳐서 새로운 트리를 만드는 것을 반복



그리디 예시 - 7

허프만 코드

- 그리디 알고리즘
  - 처음에는 각 심볼마다 자신만 있는 트리로 초기화
  - 아직 트리가 2개 이상 있으면, 가중치가 가장 작은 두 루트 정점을 합쳐서 새로운 트리를 만드는 것을 반복

```
void Solve(){
    cin >> N;
    for(int i=1; i<=N; i++) cin >> A[i];

    int res = 0;
    priority_queue<ll, vector<ll>, greater<>> pq;
    for(int i=1; i<=N; i++) pq.push(A[i]);
    while(pq.size() > 1){
        int u = pq.top(); pq.pop();
        int v = pq.top(); pq.pop();
        res += u + v;
        pq.push(u + v);
    }
    cout << res << "\n";
}
```

그리디 예시 – 8

예시 8

- 양의 정수로 구성된 배열 A[1..N]이 주어짐
- 원소를 K개 선택할 때, 선택한 원소들의 합을 최대화하는 문제
- 정렬을 이용해 간단하게 해결할 수 있지만, 그리디 기법으로 해결해 보자

- ex. A = [3, 2, 4, 6, 1, 5], K = 3
  - 앞에 있는 원소부터 차례대로 처리
  - i = 1: 3을 넣으면 합이 증가하므로 정답에 추가 — 현재 선택한 수: {3}
  - i = 2: 2를 넣으면 합이 증가하므로 정답에 추가 — 현재 선택한 수: {3, 2}
  - i = 3: 4를 넣으면 합이 증가하므로 정답에 추가 — 현재 선택한 수: {3, 2, 4}
  - i = 4: 6을 넣고 싶지만 이미 3개의 원소를 선택한 상황
    - 앞에서 한 행동을 하나 무르면 6을 넣을 수 있음
    - 가장 작은 수를 넣은 행동을 취소하자 — 현재 선택한 수: {3, 6, 4}
  - i = 5: 1은 선택된 가장 작은 수보다 작으므로 넘어감
  - i = 6: 선택된 가장 작은 수인 3을 없애고 5 추가 — 현재 선택한 수: {5, 6, 4}

- 작은 값부터 제거하는 우선 순위 큐를 이용해 O(N log N)에 해결 가능

그리디 예시 - 9

예시 9

- 수행하는데 1초가 걸리는 작업이 N개 있음
- i번째 작업은 D[i]초 안에 끝내야 하며, 마감 기한 안에 작업을 완료하면 V[i]원을 받을 수 있음
- 얻을 수 있는 금액을 최대화하는 문제

- 틀린 전략
  - 마감 기한 오름차순으로 정렬한 다음, 지금까지 수행한 작업이 D[i]개 미만이라면 i번째 작업을 수행
  - D = {1, 2, 2}, V = {1, 3, 2} 이면 이 전략은 1/2번 작업을 선택하지만, 2/3번 작업을 선택하는 것이 최적
  - 이 전략은 작업의 개수를 최대화하지만 가중치를 최대화하지 못함

그리디 예시 - 9

예시 9

- 수행하는데 1초가 걸리는 작업이 N개 있음
- i번째 작업은 D[i]초 안에 끝내야 하며, 마감 기한 안에 작업을 완료하면 V[i]원을 받을 수 있음
- 얻을 수 있는 금액을 최대화하는 문제

- 틀린 전략
  - 마감 기한 오름차순으로 정렬한 다음, 지금까지 수행한 작업이 D[i]개 미만이라면 i번째 작업을 수행
  - D = {1, 2, 2}, V = {1, 3, 2} 이면 이 전략은 1/2번 작업을 선택하지만, 2/3번 작업을 선택하는 것이 최적
  - 이 전략은 작업의 개수를 최대화하지만 가중치를 최대화하지 못함

- 관찰
  - D[a] ≤ D[b] 이면 일단 a번 작업을 수행하기로 결정한 다음
  - 혹시라도 a 대신 b 를 수행하는 것이 이득이었다면, a를 수행하려고 했던 시간에 b를 수행해도 됨
  - 예시 8처럼 가중치가 작은 작업을 버리는 전략?

그리디 예시 - 9

예시 9

- 수행하는데 1초가 걸리는 작업이 N개 있음
- i번째 작업은 D[i]초 안에 끝내야 하며, 마감 기한 안에 작업을 완료하면 V[i]원을 받을 수 있음
- 얻을 수 있는 금액을 최대화하는 문제

- 관찰
  - D[a] ≤ D[b] 이면 일단 a번 작업을 수행하기로 결정한 다음
  - 혹시라도 a 대신 b 를 수행하는 것이 이득이었다면, a를 수행하려고 했던 시간에 b를 수행해도 됨
  - 예시 8처럼 가중치가 작은 작업을 버리는 전략?

- 올바른 전략
  - 마감 기한 오름차순으로 정렬
  - 지금까지 수행한 작업이 D[i]개 미만이면 i번째 작업을 수행
  - 지금까지 수행한 작업이 D[i]개면 가장 가중치가 작은 작업을 취소하고 i번째 작업을 수행

그리디 예시 - 10

예시 10

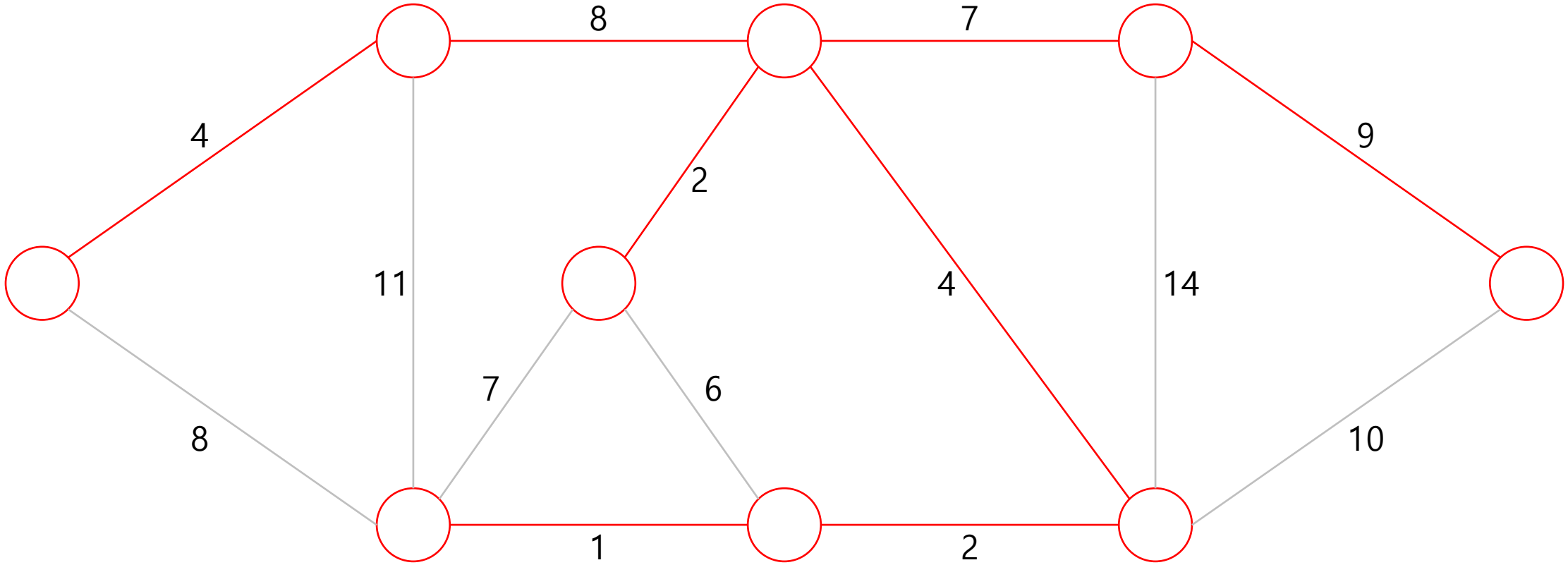
- 수행하는데 1초가 걸리는 작업이 N개 있음
- i번째 작업을 수행하는데 T[i]초가 걸리며, D[i]초 안에 끝내야 함
- 수행하는 작업의 개수를 최대화하는 문제

- 풀이는 직접 고민해 보자.

그리디 예시 - 11

최소 신장 트리

- 연결 무향 가중치 그래프 G = (V, E)가 주어지면, 모든 정점이 연결되도록 간선을 몇 개 선택하는 문제
- 이때 선택한 간선은 트리를 이루어야 하며, 가중치의 합을 최소화해야 함



그리디 예시 – 11

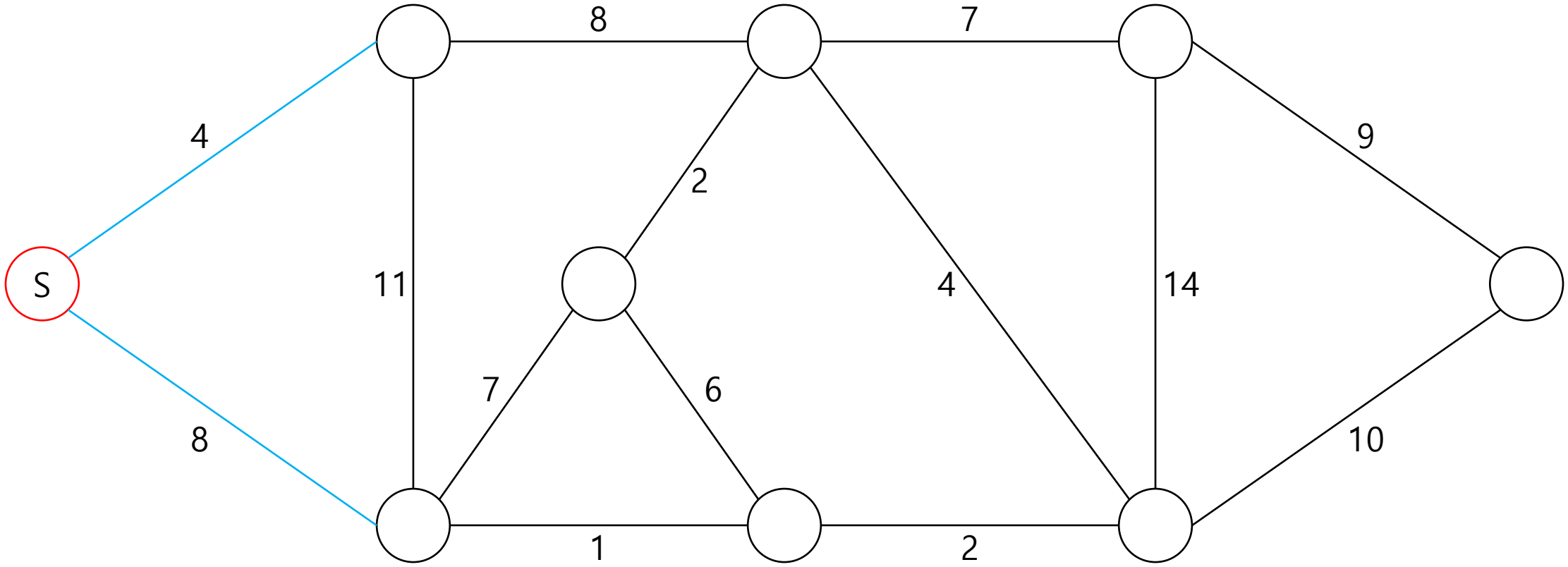
Prim's Algorithm

- 시간 복잡도: $O(V^2)$ / $O(E \log E)$ / $O(E + V \log V)$

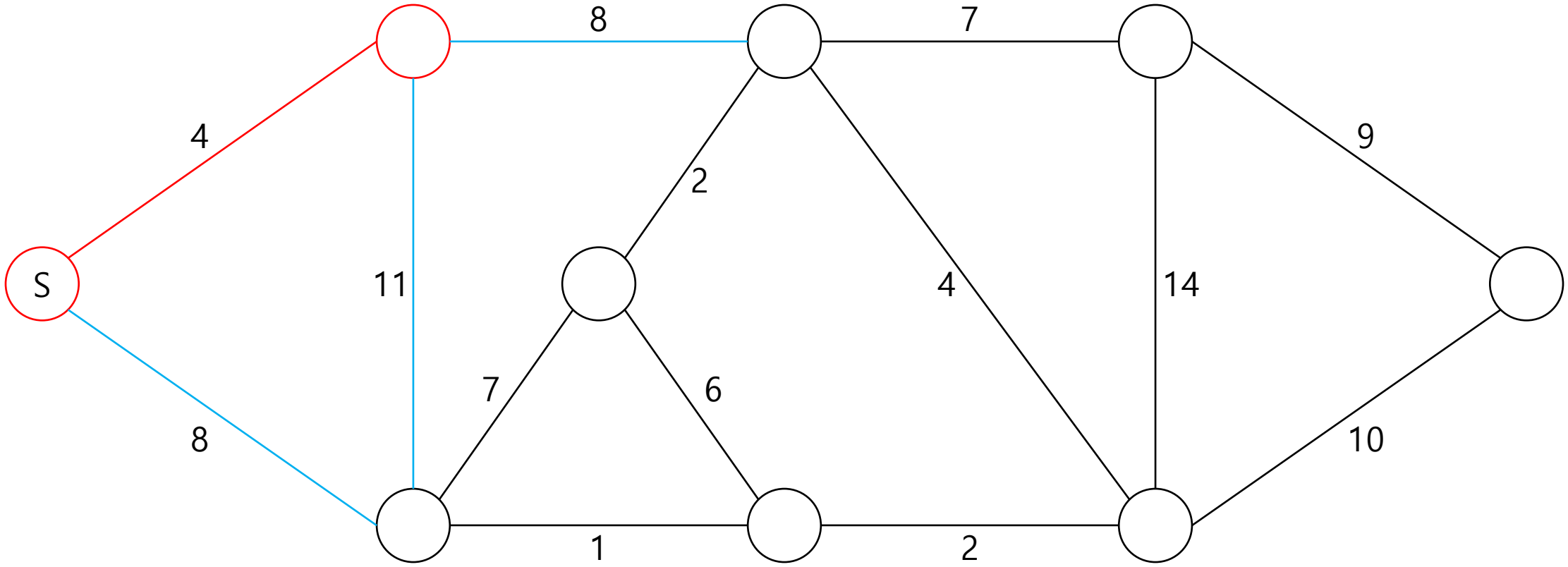
- Spanning Tree에 정점을 하나씩 포함시키면서 확장하는 방식으로 진행

- 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

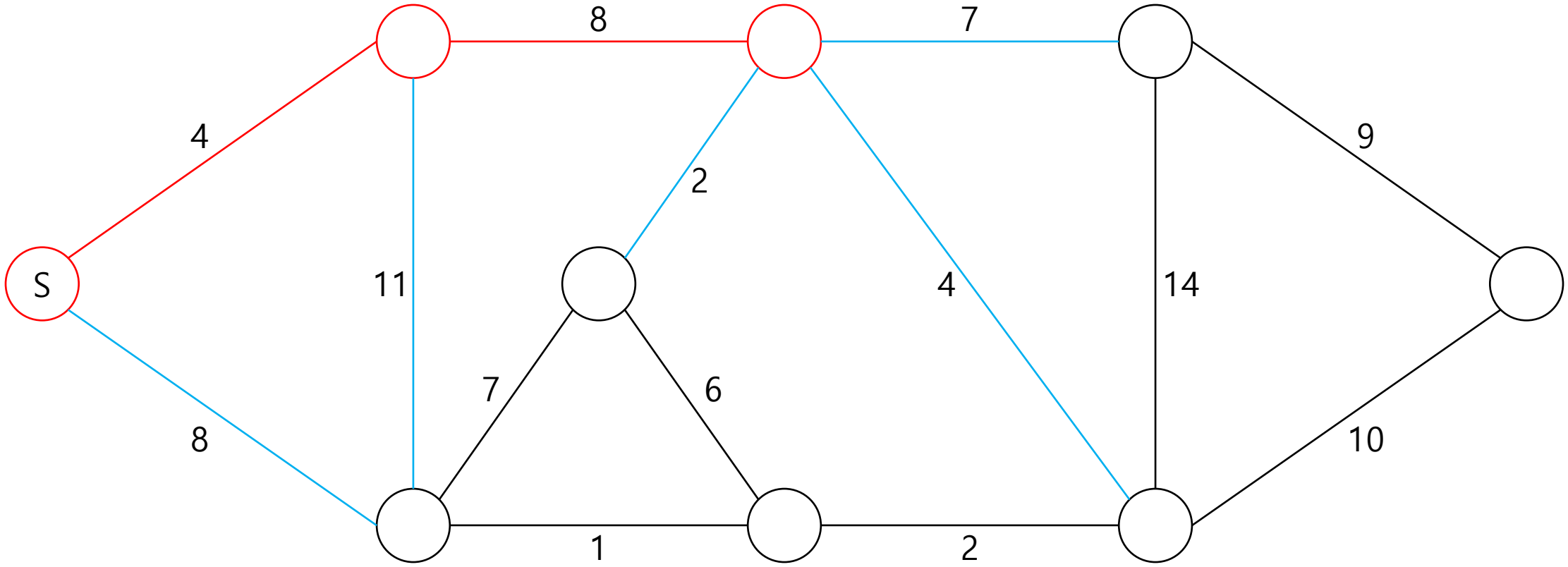
그리디 예시 - 11



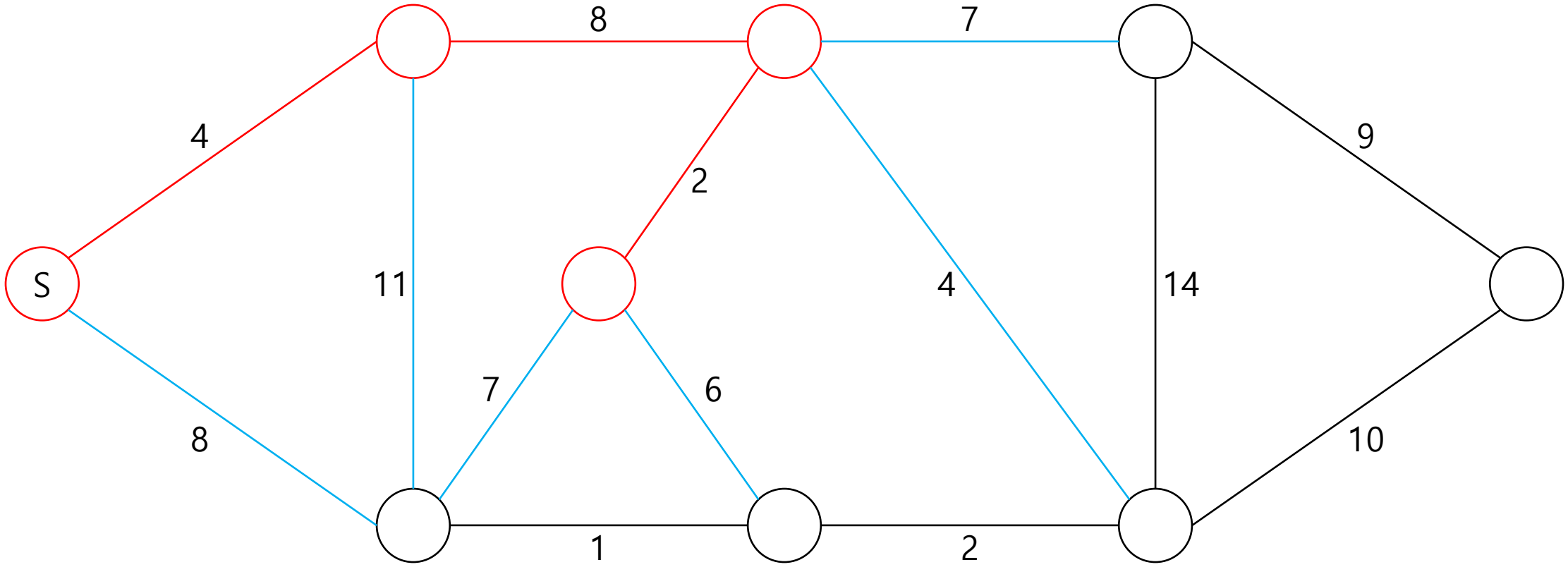
그리디 예시 - 11



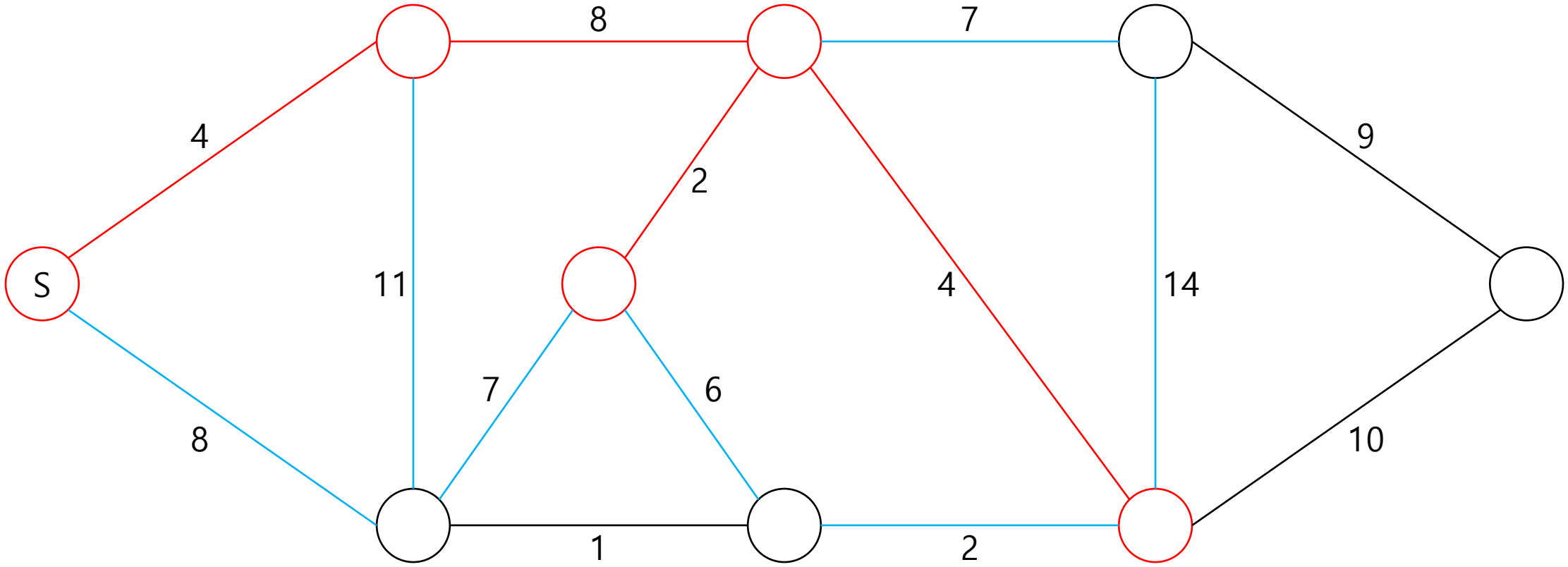
그리디 예시 - 11



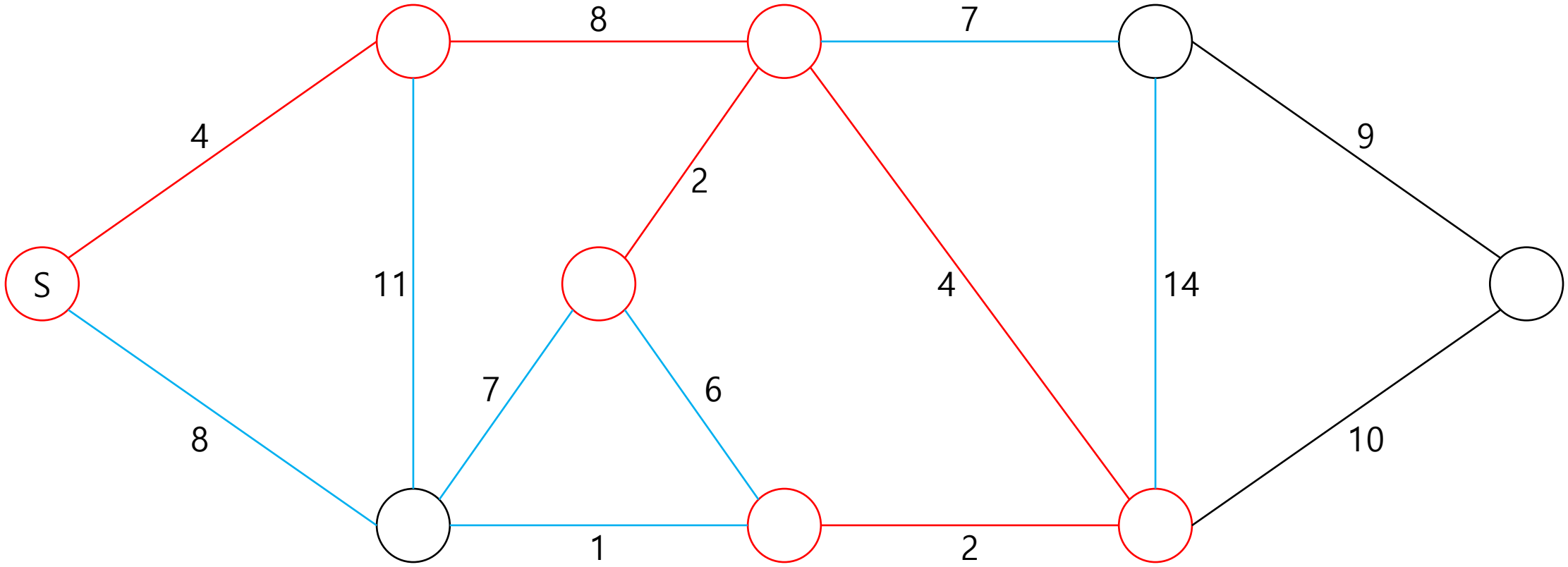
그리디 예시 - 11



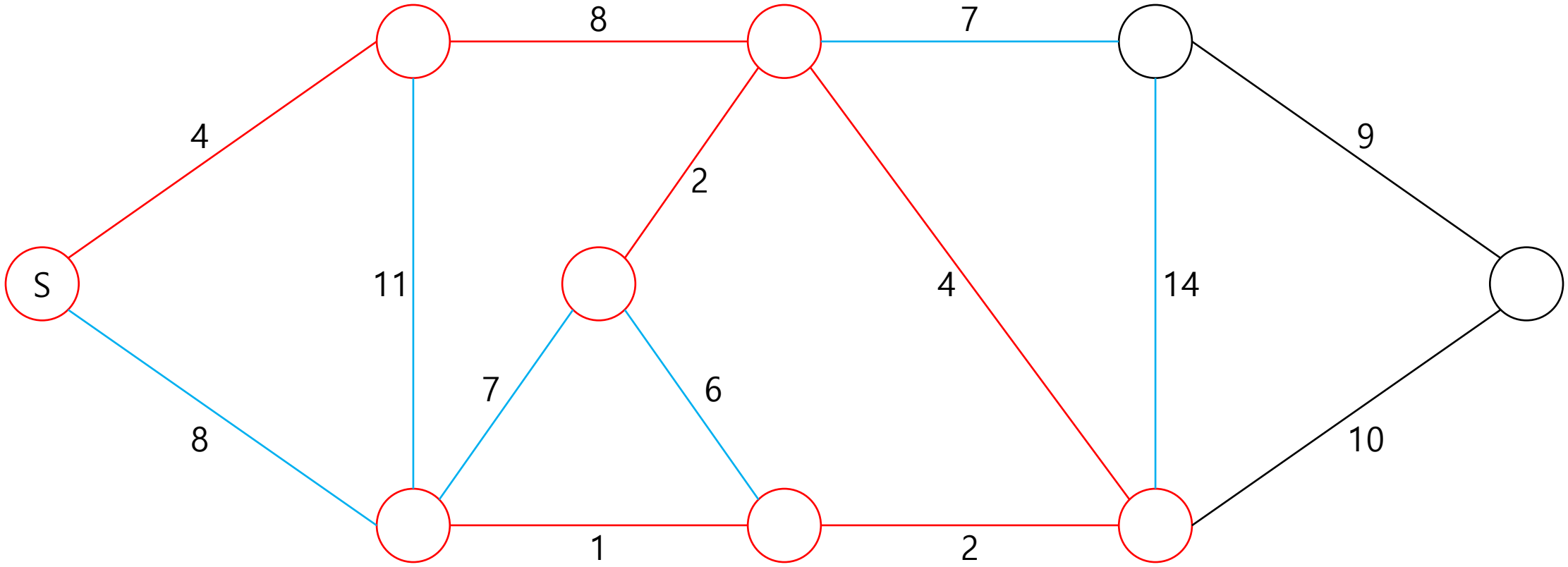
그리디 예시 - 11



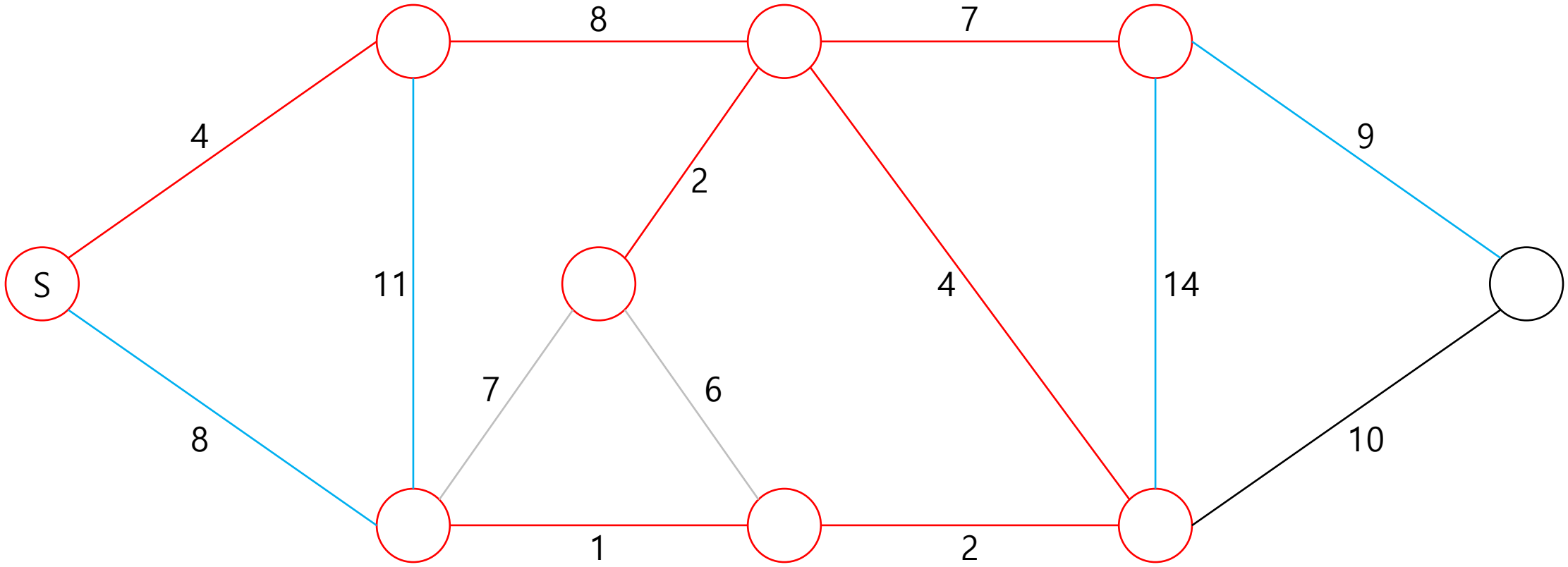
그리디 예시 - 11



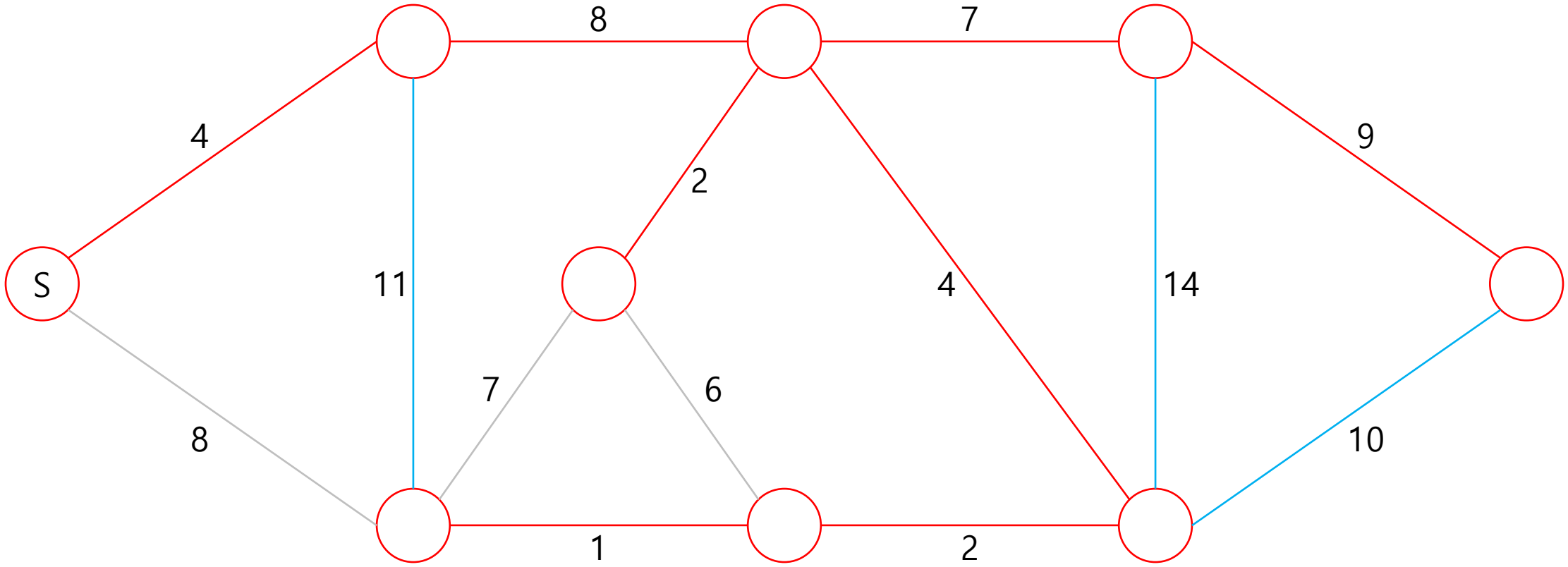
그리디 예시 - 11



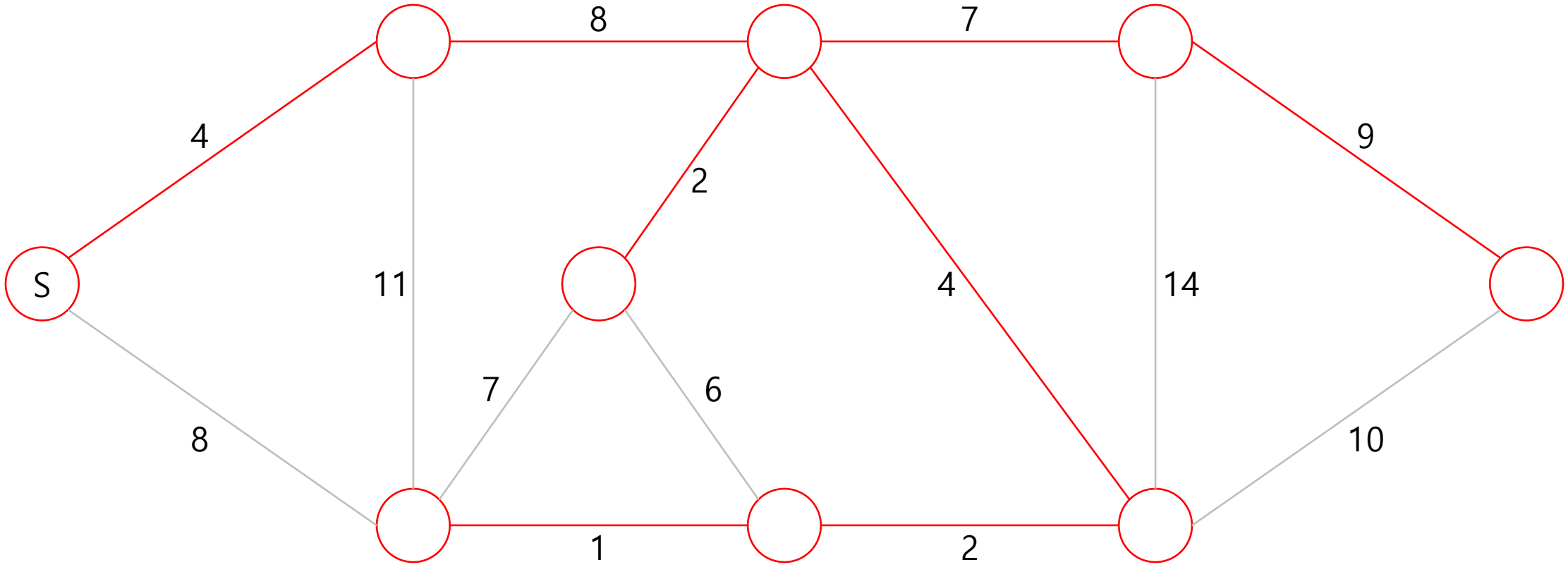
그리디 예시 - 11



그리디 예시 - 11



그리디 예시 - 11



그리디 예시 - 11

```cpp
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[i]: i번 정점을 MST에 추가하기 위해 필요한 비용
    D[1] = 0;
    for(int iter=1; iter<=N; iter++){
        int v = -1;
        for(int i=1; i<=N; i++){
            if(C[i]) continue;                      // 이미 MST에 포함된 정점 스킵
            if(v == -1 || D[v] > D[i]) v = i;
        }
        C[v] = 1; ret += D[v];                      // MST에 간선 추가
        for(auto [i,w] : G[v]) D[i] = min(D[i], w); // v에서 나가는 간선 정보 반영
    }
    return ret;
}
```

그리디 예시 – 11

정당성 증명

- 수학적 귀납법을 사용해 증명
  - 현재까지 만든 포레스트 F를 포함하는 최소 스패닝 트리 T가 존재할 때
  - F와 V-F를 연결하는 최소 간선 e를 추가한 F+e를 포함하는 최소 스패닝 트리 T'이 존재함을 증명

  - 만약 e가 T에 포함되면 T' = T이다.
  - 그렇지 않은 경우, T+e는 e를 포함하는 단순 사이클 C를 갖는다.
  - 이때 C는 F+e에 속하지 않으면서 F와 V-F를 연결하는 간선 f를 갖는다.
  - e는 F와 V-F를 연결하는 최소 간선이므로 f의 가중치는 e보다 크거나 같아야 한다.
  - 단순 사이클에서 간선 하나를 끊어낸 T-f+e는 트리가 되고, 이것의 가중치는 T 이하이므로
  - T' = T-f+e는 F+e를 포함하는 최소 스패닝 트리이다.

그리디 예시 – 11

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 뻗어 나가는 간선은 모두 봐야 하므로 O(deg(v))보다 빠르게 할 수 없음
- 비용이 최소인 정점을 빠르게 찾을 수 있을까?
  - Min Heap!

그리디 예시 - 11

```cpp
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;
}
```

그리디 예시 – 11

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)

#3. 그리디 + 매트로이드

나정휘 (jhnah917)
https://justiceHui.github.io/

매트로이드

매트로이드 (Matroid)

- 아래 네 가지 조건을 만족하는 순서쌍 $M = (S, I)$를 "매트로이드", I의 원소들을 "독립 집합"이라고 부름
  - S는 유한 집합
  - $I \subseteq 2^S$, 즉, I는 S의 부분 집합들로 구성된 집합
  - (상속성) $B \in I$ 이고 $A \subseteq B$ 이면 $A \in I$, 따라서 항상 $\emptyset \in I$를 만족해야 함
  - (확장성) $A \in I, B \in I, |A| < |B|$ 이면 $A \cup \{x\} \in I$를 만족하는 $x \in B \setminus A$가 존재함

- 확장(extension) / 기저(basis)
  - $A \in I$이고 $x \notin A$ 일 때 $A \cup \{x\} \in I$ 이면 x를 A의 "확장"이라고 부름
  - 만약 $A \in I$의 확장이 존재하지 않으면 A를 "기저"라고 부름
  - 즉, $A \in I$이고 $A \subsetneq B \in I$인 B가 없으면 A는 기저

매트로이드

매트로이드 (Matroid)

- 예시 1 – Graphic Matroid
  - $M_G = (S, I)$ 에서 S를 무향 그래프의 간선 집합, I를 사이클이 존재하지 않는 간선 집합으로 정의하자
    - 즉, I는 forest를 이루는 간선 집합
  - M_G는 매트로이드
    - 포레스트에서 간선을 몇 개 제거해도 포레스트이기 때문에 상속성을 만족함
    - 즉, 사이클이 없는 그래프에서 간선을 제거해서 사이클을 만들 수 없음
    - $|E_A| < |E_B|$ 를 만족하는 두 포레스트 $G_A = (V, E_A), G_B = (V, E_B)$를 생각해 보자
    - G_A는 $|V| - |E_A|$개의 트리, G_B는 $|V| - |E_B|$개의 트리로 구성되어 있으므로 G_B가 더 적은 개수의 트리를 갖고 있음
    - 따라서 G_B의 어떤 트리 T는 G_A에서 서로 다른 트리에 속한 두 정점 u, v를 포함하고 있음
    - 따라서 T의 $u - v$ 경로에 속한 간선 중 최소한 하나는 $|E_A|$에 속하지 않음
    - 그러므로 그러한 간선이 E_A의 확장이 되므로 확장성도 만족
  - G가 연결 그래프면 M_G의 기저는 스패닝 트리

매트로이드

매트로이드 (Matroid)

- 예시 2 – Vector Matroid
  - 벡터 공간 V에서 $S = \{v_1, v_2, \cdots, v_n\} \subseteq V$, I를 선형 독립인 S의 부분 집합들로 정의하면 $M = (S, I)$는 매트로이드
  - 상속성은 자명하고, 확장성도 쉽게 증명할 수 있음
    - A, B가 각각 선형 독립이고, $|A| < |B|$ 이면 A의 벡터들이 B의 모든 vector space를 span할 수 없음

- 예시 3 – Uniform Matroid
  - 어떤 집합 S가 있을 때, 크기가 k 이하인 모든 부분 집합들을 I로 정의하면 $M = (S, I)$는 매트로이드

- 예시 4 – Partition Matroid
  - S의 분할 $S_1, S_2, \cdots, S_m$과 양수 $k_1, k_2, \cdots, k_m$이 있을 때, $I = \{X \subseteq S; \forall i \ |X \cap S_i| \leq k_i\}$로 정의하면 매트로이드

- 예시 5 – Cographic Matroid
  - 무향 그래프 $G = (V, E)$에서 $S = E$, $I = \{X \subseteq E, E - X$가 연결 그래프$\}$로 정의하면 매트로이드

매트로이드

매트로이드 (Matroid)

- Theorem 1. 매트로이드에 있는 모든 기저들의 크기는 동일함
  - A가 매트로이드 $M = (S, I)$의 기저이면서 $|A| < |B|$인 기저 B가 존재한다고 가정하자
  - 확장성에 의해 $A \cup \{x\} \in I, x \in B \setminus A$를 만족하는 x가 존재할 텐데, 이는 A가 기저라는 것에 모순

- 따라서 원소를 추가할 수 있을 때마다 계속 추가하면 항상 최대 크기의 집합을 얻을 수 있음
  - Local Optimum = Global Optimum 이므로 그리디하게 원소를 선택할 수 있음

매트로이드

매트로이드 (Matroid)

- 가중치 매트로이드
  - 매트로이드 $M = (S, I)$에서 S의 각 원소 x가 가중치 $w(x)$를 갖는 상황
  - 집합 X의 가중치는 $w(X) = \sum_{x \in X} w(x)$, 즉 원소들의 가중치 합으로 정의

  - 예시 11(최소 신장 트리): graphic matroid에서 가중치가 최소인 기저를 찾는 문제
  - 예시 9(마감 시간과 보상이 있는 단위 시간 작업 스케줄링): 가중치가 최대인 독립 집합을 찾는 문제

- 최대 가중치 독립 집합을 찾는 그리디 전략
  - 가중치가 큰 원소부터 차례대로 보면서, 독립 집합에 추가할 수 있으면 추가하는 전략

- Theorem 2. 이 그리디 알고리즘은 음수 가중치가 없을 때 항상 최대 가중치 독립 집합을 찾음
  - 최소 가중치 독립 집합은 가중치가 작은 원소부터 보면 됨

매트로이드

매트로이드 (Matroid)

- Theorem 2. 이 그리디 알고리즘은 음수 가중치가 없을 때 항상 최대 가중치 독립 집합을 찾음
  - 그리디 알고리즘으로 구한 독립 집합 A보다 더 좋은 독립 집합 B가 존재한다고 가정하자
  - Theorem 1에 의해 A는 기저이고, 모든 가중치가 0 이상이므로 B도 기저라고 생각해도 됨

  - A, B의 원소들을 각각 가중치 내림차순으로 $A = \{a_1, a_2, \cdots, a_n\}, B = \{b_1, b_2, \cdots, b_n\}$이라고 하자
  - $w(A) < w(B)$이므로 처음으로 $w(a_k) < w(b_k)$가 되는 지점 $1 \leq k \leq n$이 존재
  - $A_{k-1} = \{a_1, a_2, \cdots, a_{k-1}\}, \mathrm{B_k} = \{\mathrm{b_1}, \mathrm{b_2}, \cdots, b_{k-1}, b_k\}$ 라고 하자

  - $A_{k-1} \in I, B_k \in I, |A_{k-1}| < |B_k|$ 이므로 $A_{k-1} \cup \{b_i\} \in I$를 만족하는 $b_i \in B_k \setminus A_{k-1}$이 존재함 $(1 \leq i \leq k)$
  - $w(a_k) < w(b_k) \leq w(b_i)$이므로 A보다 더 좋은 해 B가 있었다면 그리디 알고리즘은 a_k 대신 b_i를 선택함
  - 따라서 A는 그리디 알고리즘의 결과물이 될 수 없음

그리디 예시 – 11

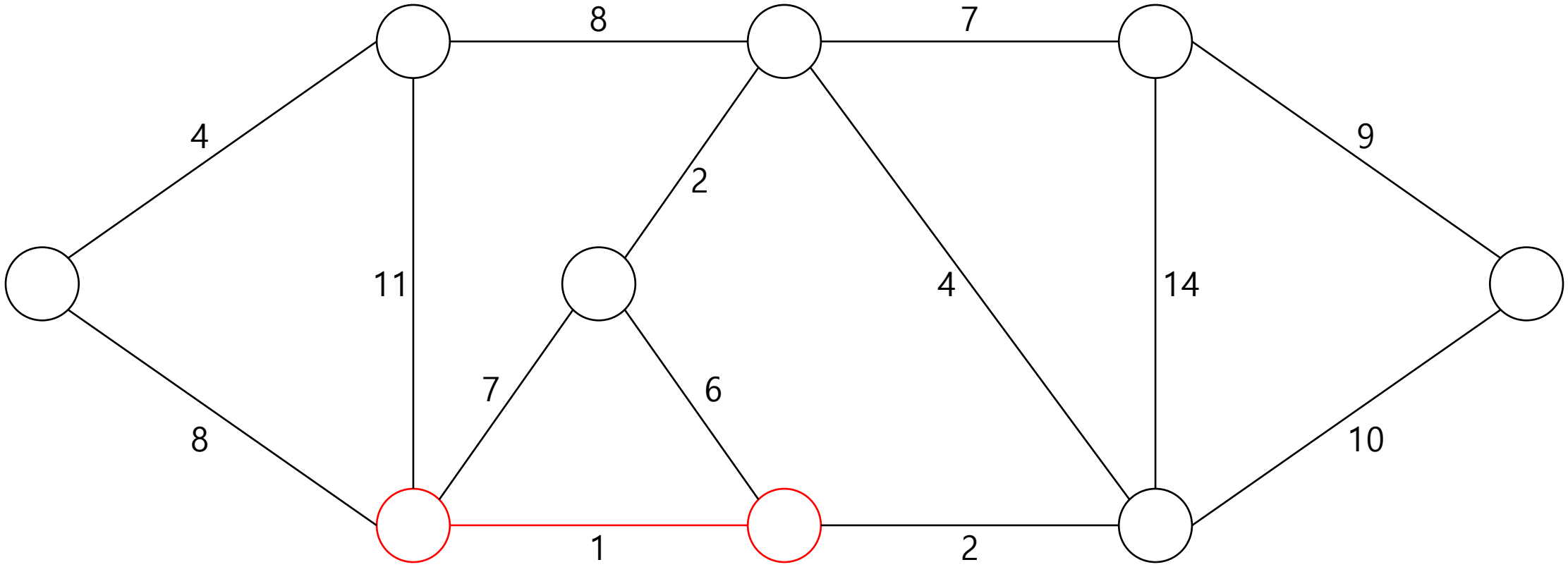
최소 신장 트리

- 연결 무향 가중치 그래프 G = (V, E)가 주어지면, 모든 정점이 연결되도록 간선을 몇 개 선택하는 문제
- 이때 선택한 간선은 트리를 이루어야 하며, 가중치의 합을 최소화해야 함

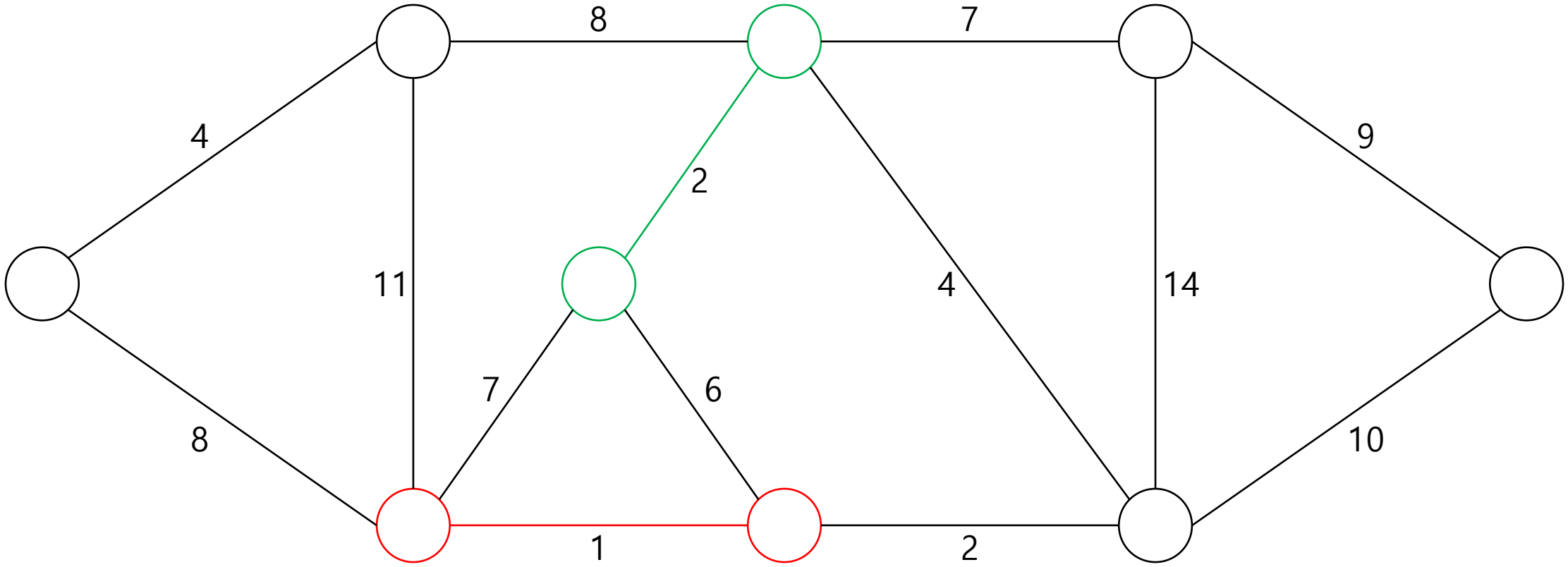
- Graphic Matroid에서 최소 가중치 독립 집합을 찾는 문제
  - 가중치가 작은 간선부터 보면서 간선을 추가했을 때 사이클이 생기지 않으면 그 간선을 추가
  - Kruskal's Algorithm

  - 사이클을 판별은 어떻게?
  - 서로소 집합(Disjoint Set) / 유니온 파인드(Union-Find)

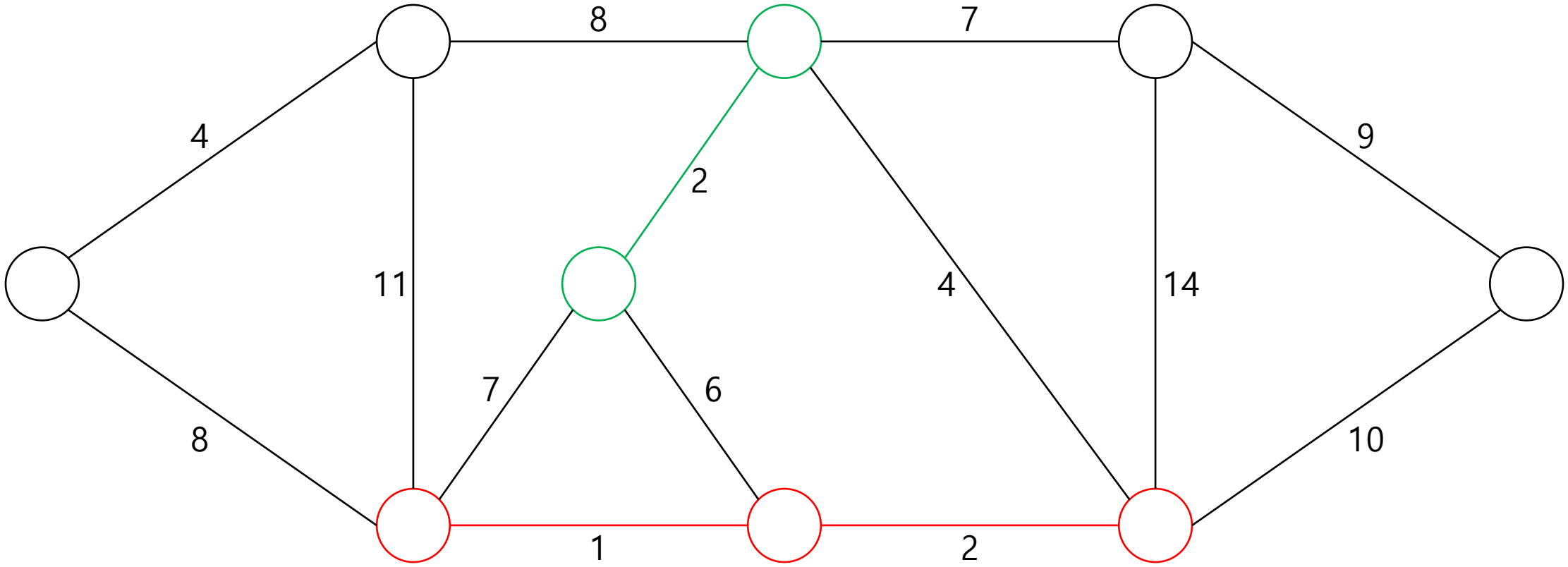
그리디 예시 - 11



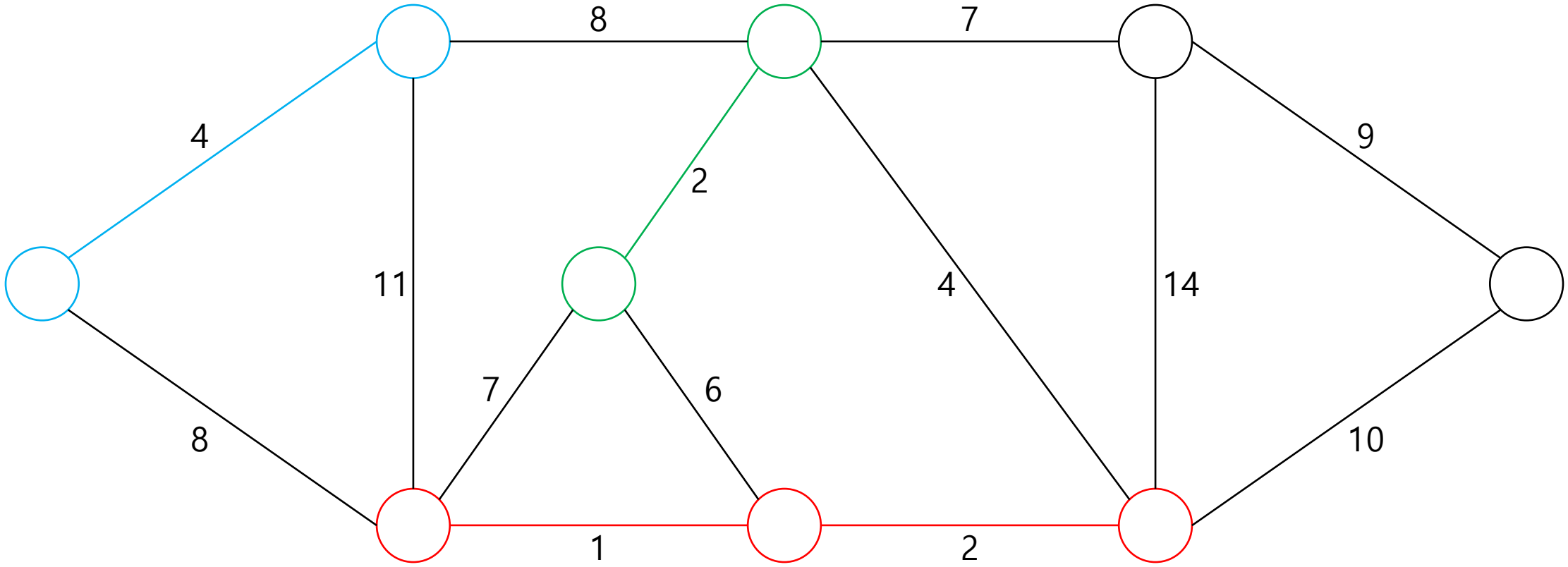
그리디 예시 - 11



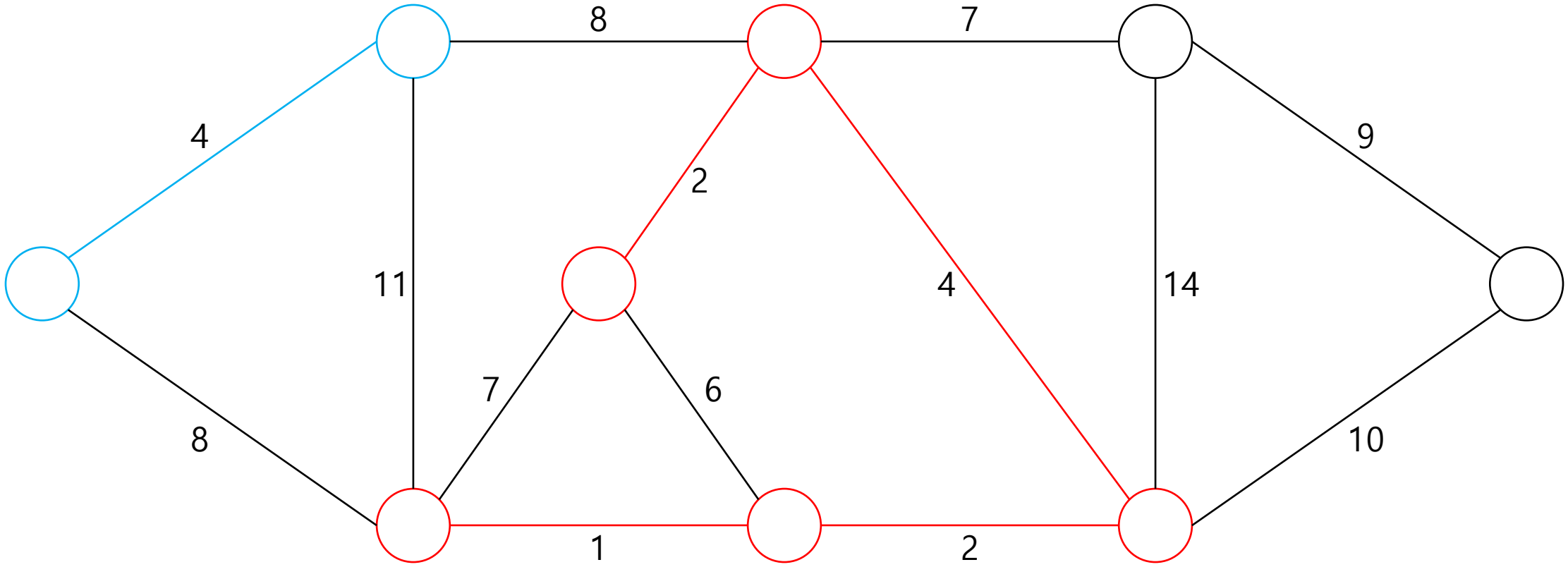
그리디 예시 - 11



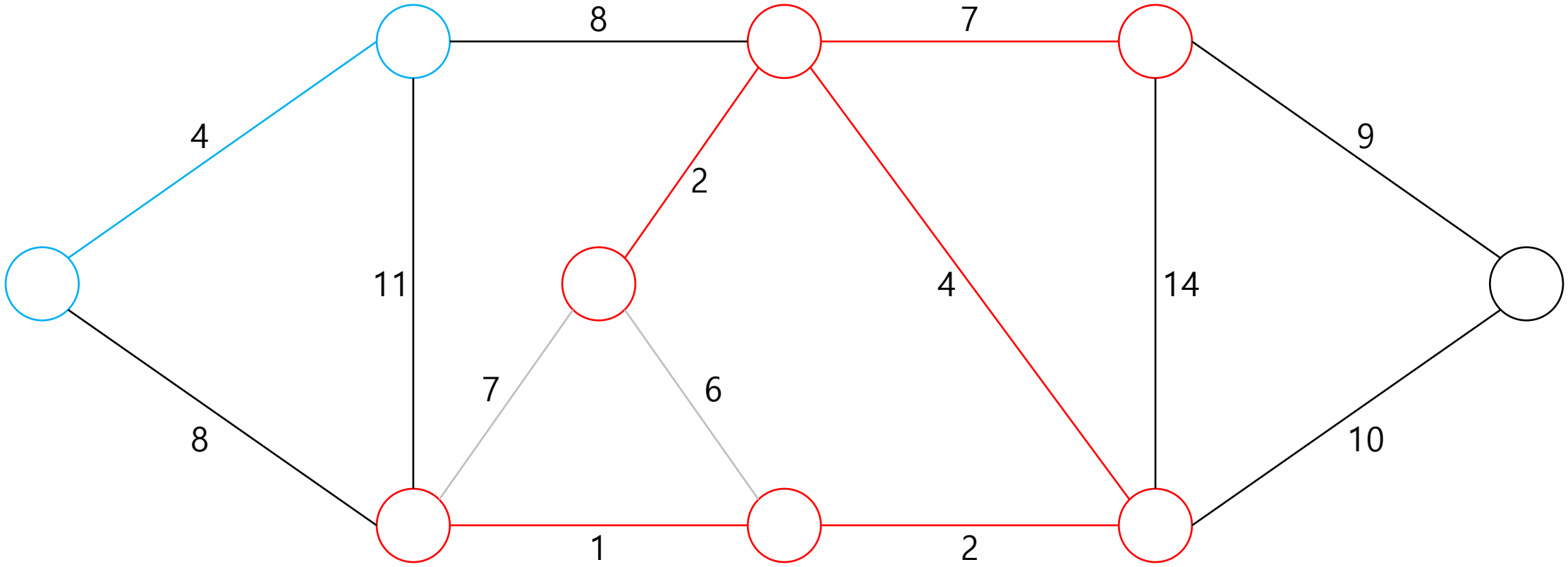
그리디 예시 - 11



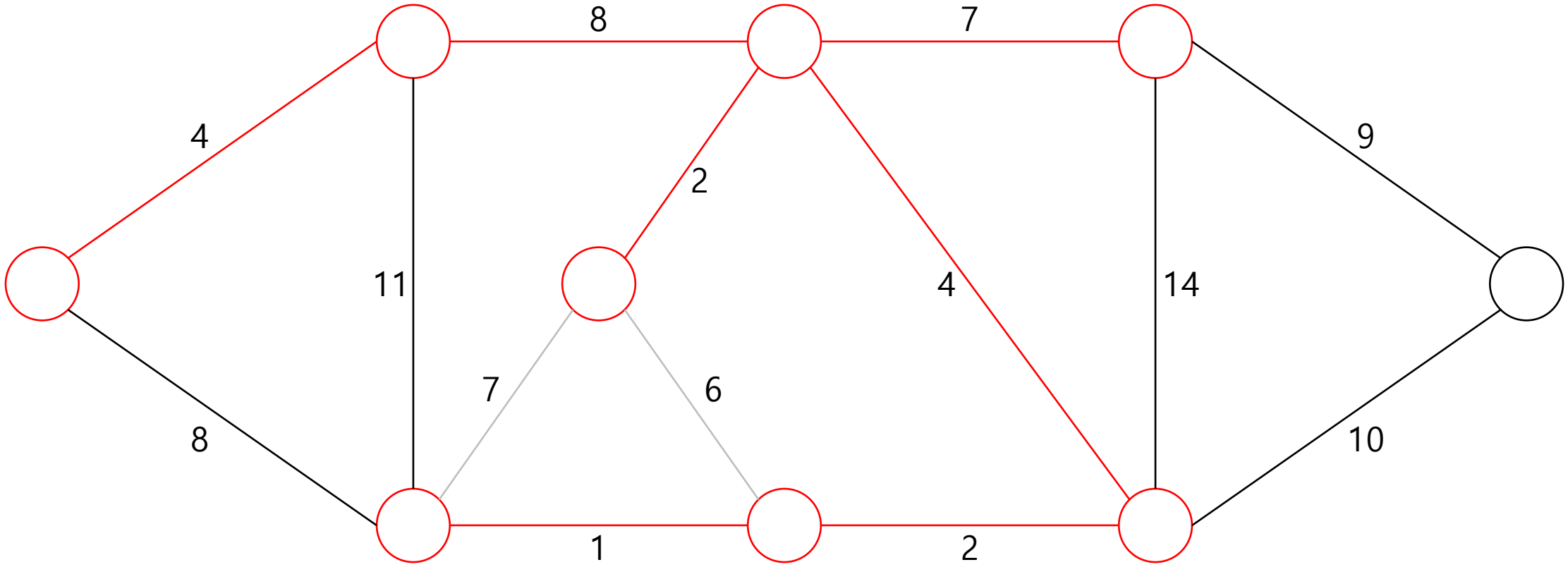
그리디 예시 - 11



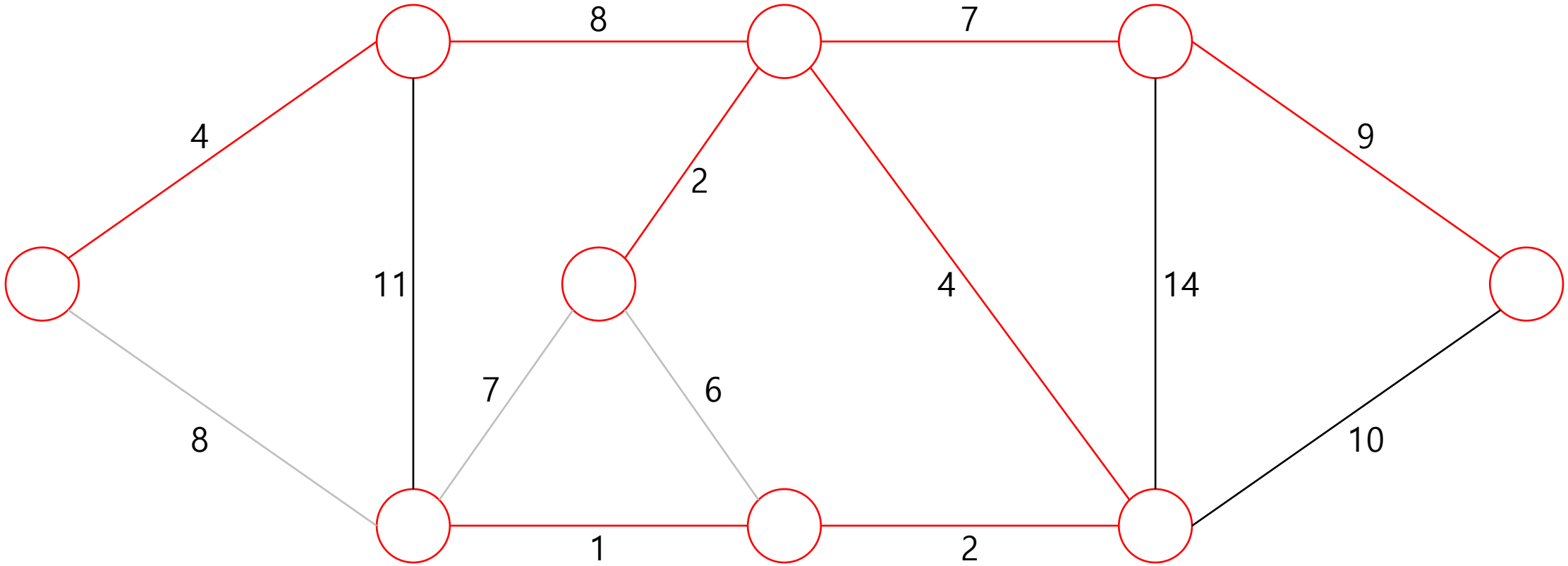
그리디 예시 - 11



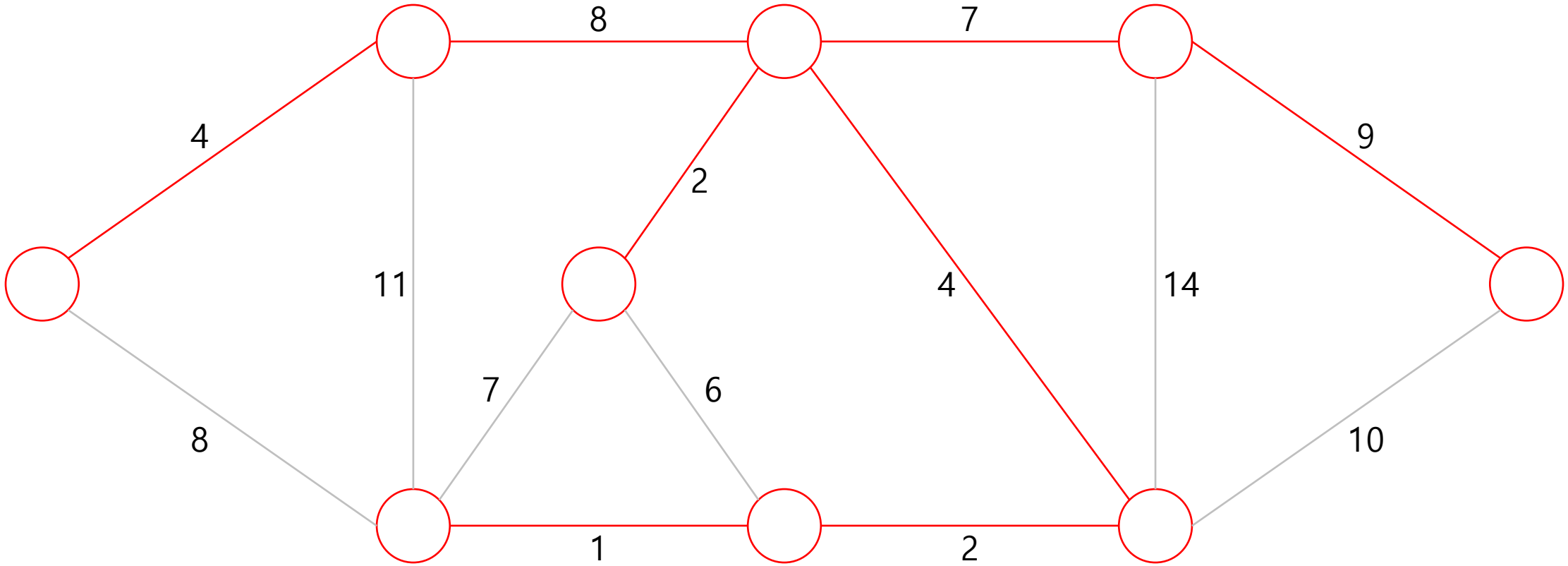
그리디 예시 - 11



그리디 예시 - 11



그리디 예시 - 11



서로소 집합

서로소 집합족

- 교집합이 공집합인 집합으로 구성된 집합족
  - 두 집합 A, B가 A = B이거나 A ∩ B = Ø
  - 서로소인 두 정수 A, B의 소인수분해를 생각해 보자.

- 각 원소가 속한 집합을 유일하게 특정할 수 있음

서로소 집합

Union Find (또는 Disjoint Set)

- 서로소 집합을 관리하는 자료구조
  - Init : 모든 원소가 자기 자신만을 원소로 하는 집합에 속하도록 초기화
  - Union(u, v) : u가 속한 집합과 v가 속한 집합을 병합
  - Find(v) : v가 속한 집합을 반환

- 위 3가지 연산을 빠르게 구현하는 것이 목표

서로소 집합

Union Find

- 각 집합을 Rooted Tree로 표현한다고 생각해 보자.
  - Init은 포레스트에서 모든 간선을 제거하는 것과 동일
  - Find(v)는 v가 속한 트리의 루트 정점을 반환하면 됨
  - Union(u, v)는 u가 속한 트리의 루트를 v가 속한 트리의 루트의 자식으로 넣어주면 됨

  - 시간 복잡도는?

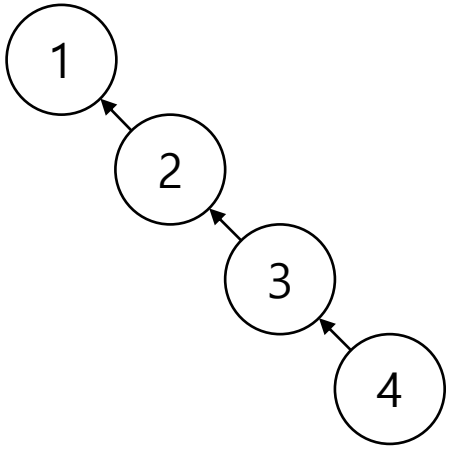
서로소 집합

```cpp
int P[101010];

void Init(int n){
    for(int i=1; i<=n; i++) P[i] = i;
}

int Find(int v){
    if(v == P[v]) return v;
    return Find(P[v]);
}

void Union(int u, int v){
    int u_root = Find(u), v_root = Find(v);
    if(u_root != v_root) P[u_root] = v_root;
}
```

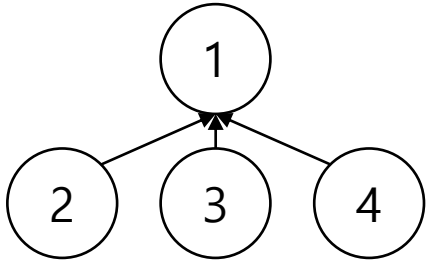
서로소 집합

Union Find

- Init : O(N)
- Find : 트리의 높이를 h라고 하면 O(h)
- Union : Find와 동일



- 최악의 경우 O(N)
- 최선의 경우 O(1)



서로소 집합

최적화

- 트리의 높이를 줄여야 함
  - Union by Rank
  - Union by Size
  - Path Compression

- 3개 중 하나만 사용해도 M번 연산했을 때 O(M log N)이 보장됨
  - Union by Rank, Union by Size는 O(log N)
  - Path Compression은 amortized O(log N)

- Union by Rank와 Path Compression을 함께 사용하면 amortized O(log* N)
  - log* N : N을 1 이하로 만들기 위해서 필요한 log의 횟수
  - log* N = 1 + log* (log N)

서로소 집합

Union by Rank

- 높이가 낮은 트리를 높은 트리 아래로 넣는 방법
  - Rank[i] : i를 루트로 하는 트리 높이의 상한
  - 만약 두 트리의 높이가 동일하면 Union 후 Rank 1 증가

- rank가 k인 정점은 최대 $N/2^{k-1}$개 있음을 증명 가능
- 따라서 트리의 높이는 최대 $\log_2 N$

```cpp
int P[101010], R[101010];

void Init(int n){
    for(int i=1; i<=n; i++) P[i] = i, R[i] = 1;
}

int Find(int v){
    if(v == P[v]) return v;
    return Find(P[v]);
}

void Union(int u, int v){
    u = Find(u); v = Find(v);
    if(u == v) return;
    if(R[u] > R[v]) swap(u, v);
    P[u] = v;
    if(R[u] == R[v]) R[v] += 1;
}
```

서로소 집합

Union by Size

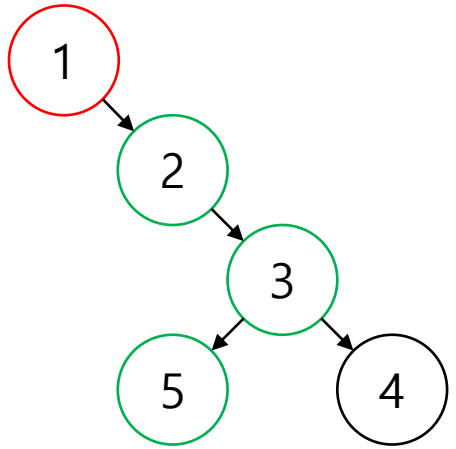
- 정점이 적은 트리를 많은 트리 아래로 넣는 방법
  - Size[i] = i를 루트로 하는 트리의 정점 개수
  - u를 v 밑으로 넣으면 Size[v]는 Size[u] 만큼 증가

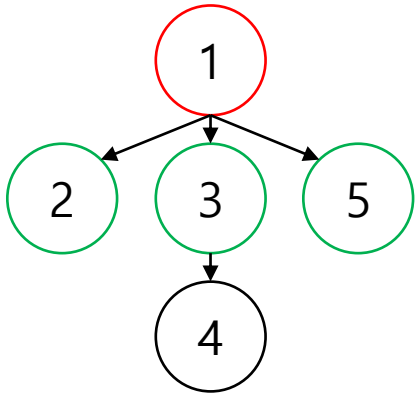
- 어떤 트리가 다른 트리 밑으로 붙을 때마다
- 트리의 크기는 2배 이상 증가함

- 따라서 각 정점의 높이는 최대 $\log_2 N$번 증가함

```cpp
int P[101010], S[101010];

void Init(int n){
    for(int i=1; i<=n; i++) P[i] = i, R[i] = 1;
}

int Find(int v){
    if(v == P[v]) return v;
    return Find(P[v]);
}

void Union(int u, int v){
    u = Find(u); v = Find(v);
    if(u == v) return;
    if(S[u] > S[v]) swap(u, v);
    P[u] = v; S[v] += S[u];
}
```

Union Find





Path Compression

- 경로 압축
  - Find를 통해 루트를 찾았다면
  - 루트까지의 경로 상에 있는 정점을 모두 루트의 바로 밑(자식)으로 붙임

- 한 번의 연산에서는 O(N) 만큼의 시간이 걸릴 수 있음
  - Union(N, N-1), Union(N-1, N-2), ... , Union(2, 1) 하면 높이 N

- M(≥ N번)의 연산을 하면 O(M log N)이 됨을 증명할 수 있음
- 따라서 시간 복잡도는 amortized O(log N)
  - 증명은 생략

- Union by Rank + Path Compression은 amortized O(log* N)
  - 증명은 (링크) 참고

```
int P[101010];

void Init(int n){
    for(int i=1; i<=n; i++) P[i] = i;
}

int Find(int v){
    if(v == P[v]) return v;
    return P[v] = Find(P[v]);
}

void Union(int u, int v){
    u = Find(u); v = Find(v);
    if(u != v) P[u] = v;
}
```