

Chapter. 02

알고리즘

최단 거리 Shortest Path

FAST CAMPUS
ONLINE

알고리즘 공채 대비반 I

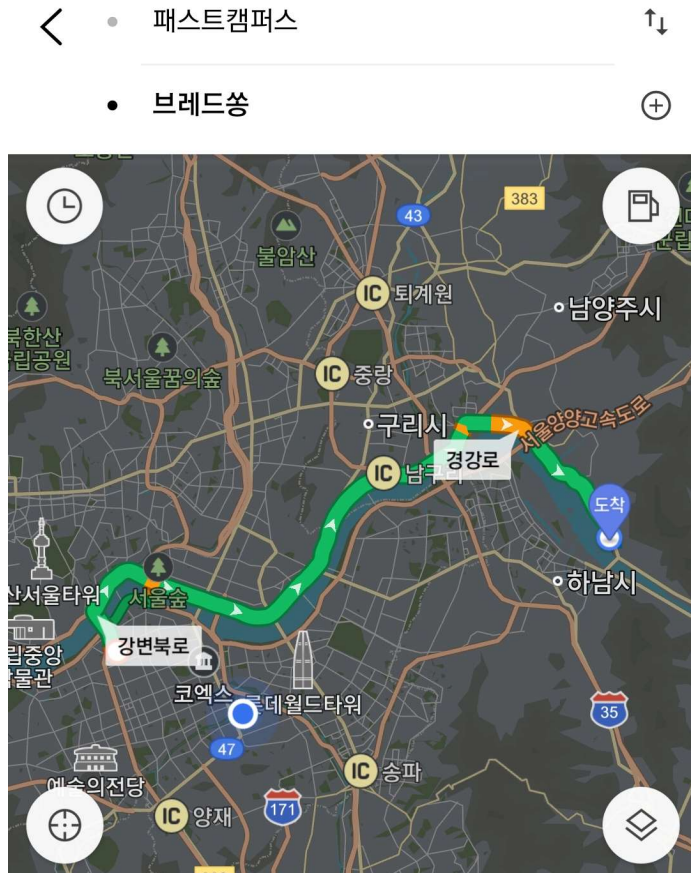
강사. 류호석

Chapter. 02

알고리즘 최단 거리(Shortest Path)



I 최단 거리(Shortest Path)란?



최단 거리 := 그래프의 시작점에서 다른 지점
까지의 최단 거리

일상에서의 최단 거리

I 최단 거리(Shortest Path) 알고리즘

이름	간선의 가중치	시작점	도착점	시간 복잡도
BFS	모두 1	한 정점	모든 정점	$O(V + E)$
Dijkstra	≥ 0	한 정점	모든 정점	$O(E \log V)$
Floyd-Warshall	제약 없음	모든 정점	모든 정점	$O(V^3)$
Bellman-Ford	제약 없음	한 정점	모든 정점	$O(V \times E)$
SPFA	제약 없음	한 정점	모든 정점	$O(V \times E)$
A*	≥ 0	한 정점	한 정점	$O(b^d)$

I 최단 거리(Shortest Path) 알고리즘

이름	간선의 가중치	시작점	도착점	시간 복잡도
BFS	모두 1	한 정점	모든 정점	$O(V + E)$
Dijkstra	≥ 0	한 정점	모든 정점	$O(E \log V)$
Floyd-Warshall	제약 없음	모든 정점	모든 정점	$O(V^3)$
Bellman-Ford	제약 없음	한 정점	모든 정점	$O(V \times E)$
SPFA	제약 없음	한 정점	모든 정점	$O(V \times E)$
A*	≥ 0	한 정점	한 정점	$O(b^d)$

I BFS Remind

탐색(Search) = **시작점**에서 간선을 0개 이상
사용해서 **갈 수 있는 정점들**은 무엇인가?

Depth First Search / **Breadth First Search**

BFS → 다른 정점까지 **최소 이동 횟수**도 계산 가능!

한 번의 이동 == 간선의 가중치가 1

I Dijkstra Algorithm 요약

<Input>

- Graph $G(V, E) :=$ Nonnegative-Weighted Graph
- Start Vertex S / Vertices $\{S_1, \dots\}$

<Output>

- **시작점**에서 모든 점까지의 최단 거리

<Time Complexity>

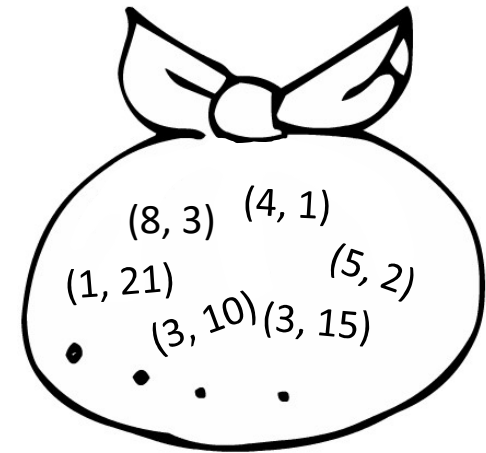
- $O(E \log V)$

I Dijkstra Algorithm

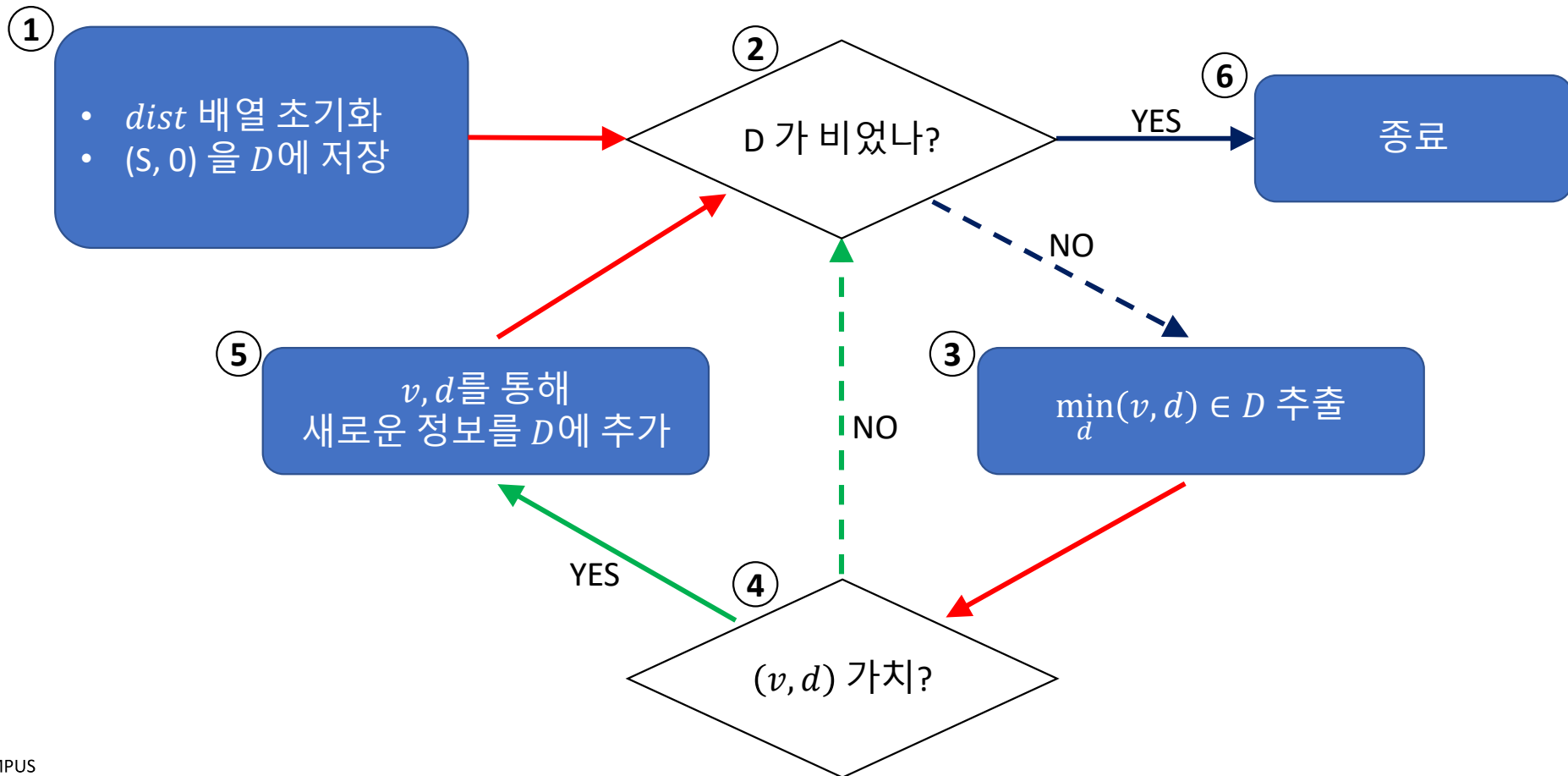
<필요한 정보>

$dist[i] :=$ 시작점에서 i 번 정점까지 가능한 최단 거리

자료구조 $D := \{(v, d) \mid \text{시작점에서 } v \text{ 까지 } d \text{ 만에 갈 수 있음을 확인했다}\}$



I Dijkstra Algorithm

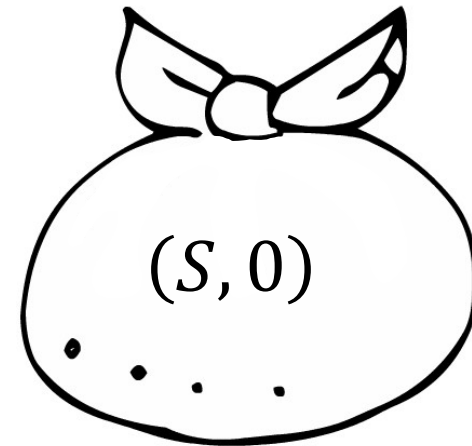


I Dijkstra Algorithm

①

- $dist$ 배열 초기화
- $(S, 0)$ 을 D 에 저장

1. $dist[i] = \begin{cases} 0 & \text{if } i == S \\ \infty & \text{else} \end{cases}$
2. D 에 $(S, 0)$ 을 추가한다.



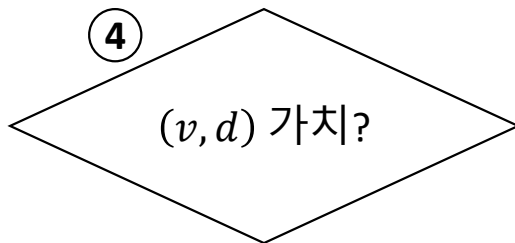
I Dijkstra Algorithm

③

 $\min_d(v, d) \in D$ 추출

D 에서 가장 작은 d 를 갖는 자료 (v, d) 를 뽑는다.

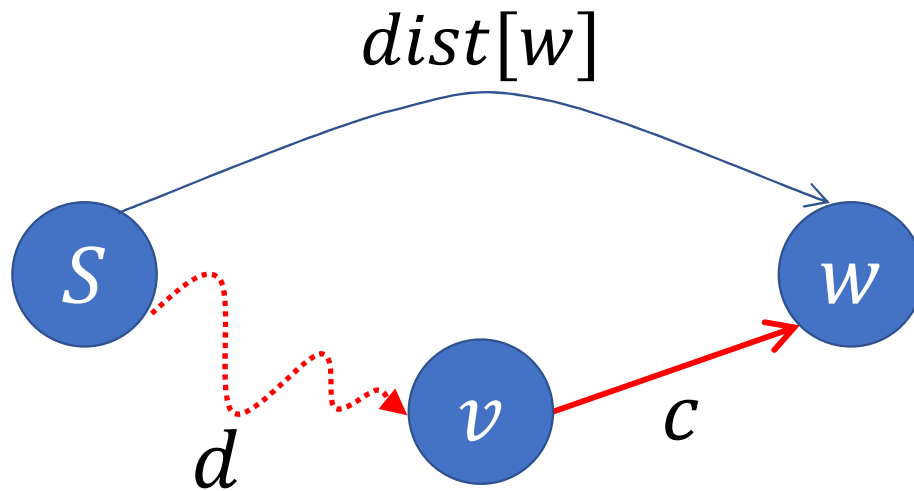
④

 $dist[v] < d$

→ v 까지의 최단 거리보다 d 가 크다면, 이미 가치가 없는 정보 이므로 폐기한다.

I Dijkstra Algorithm

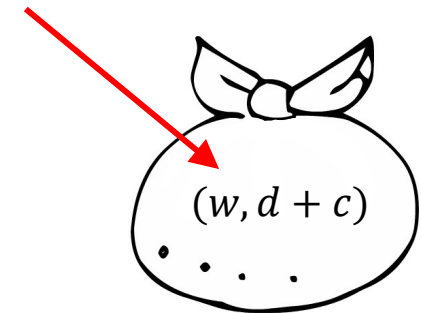
5 v, d 를 통해
새로운 정보를 D 에 추가



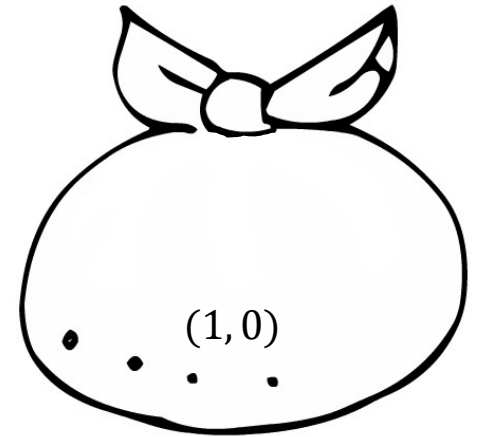
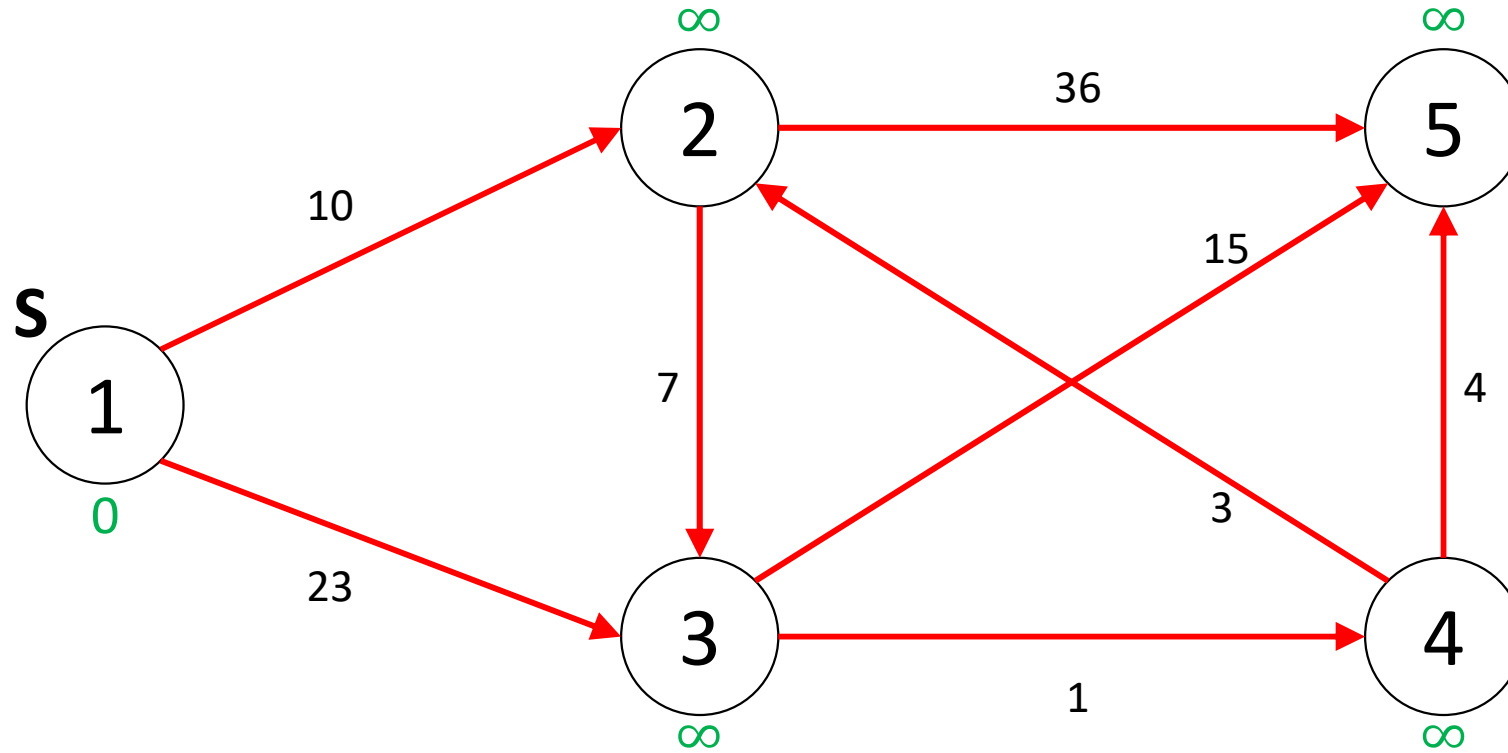
$$d + c < dist[w]$$

$$\Rightarrow dist[w] = d + c$$

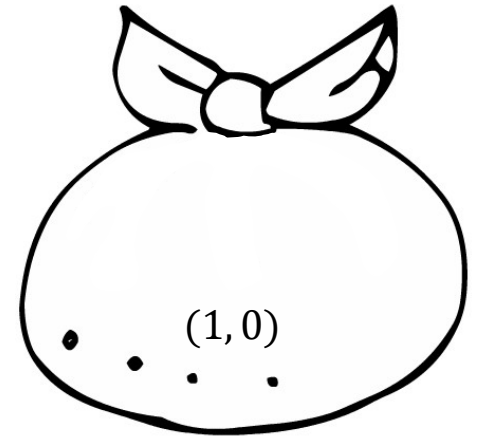
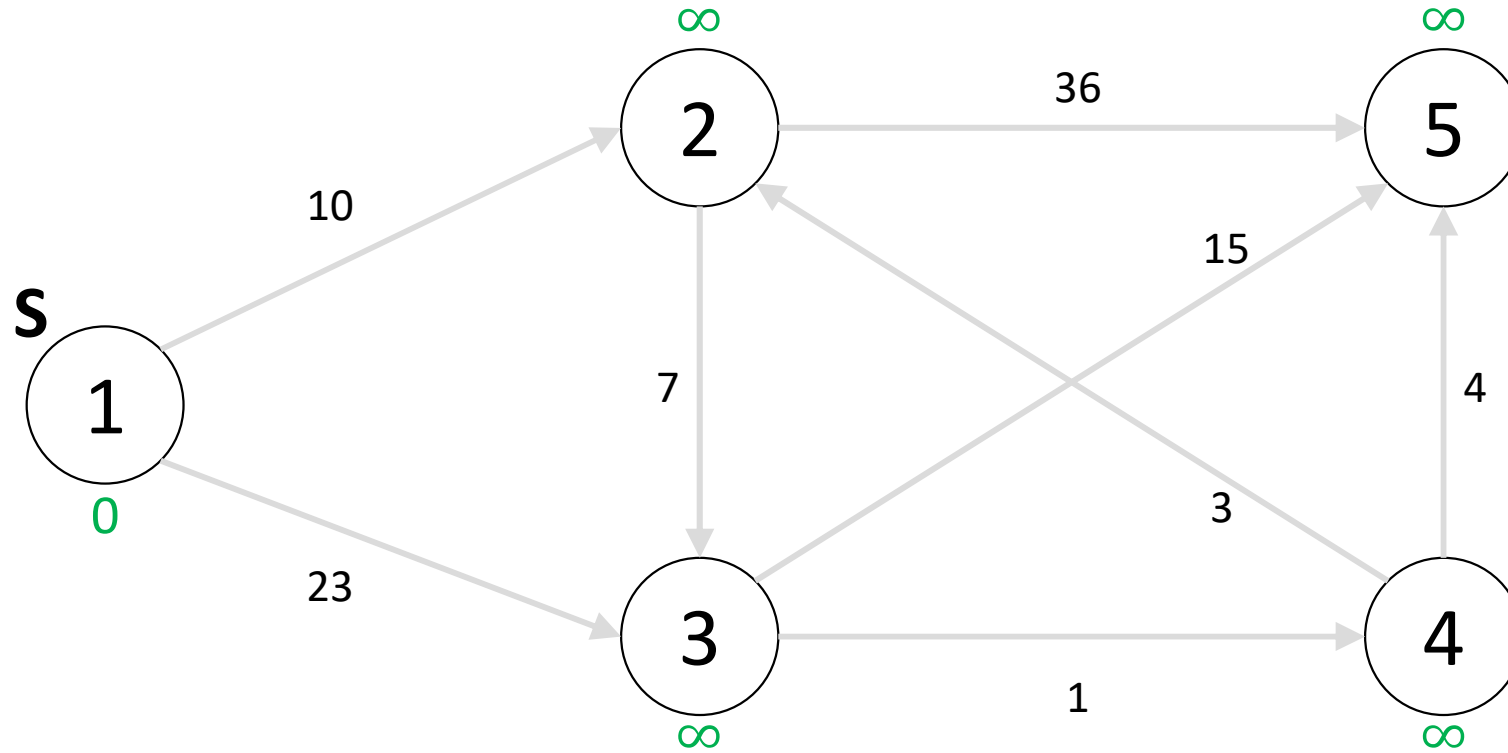
$$\Rightarrow (w, d + c)$$



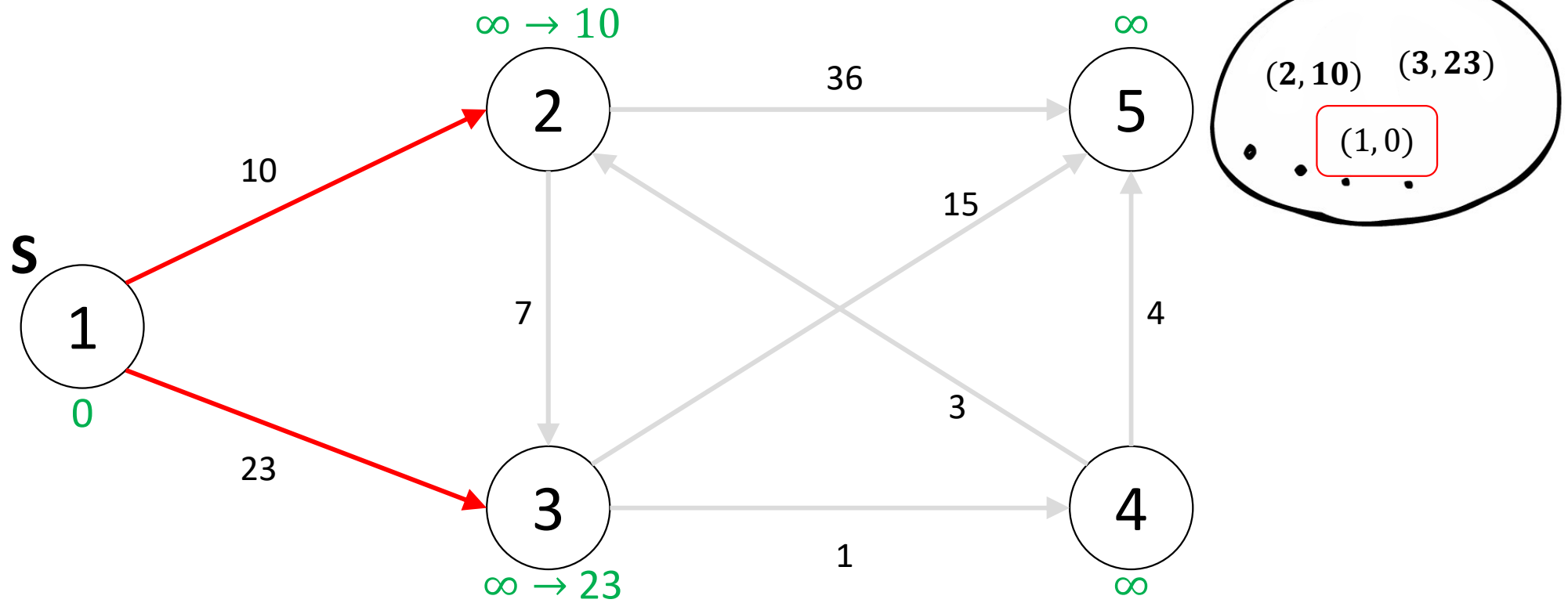
I Dijkstra Algorithm



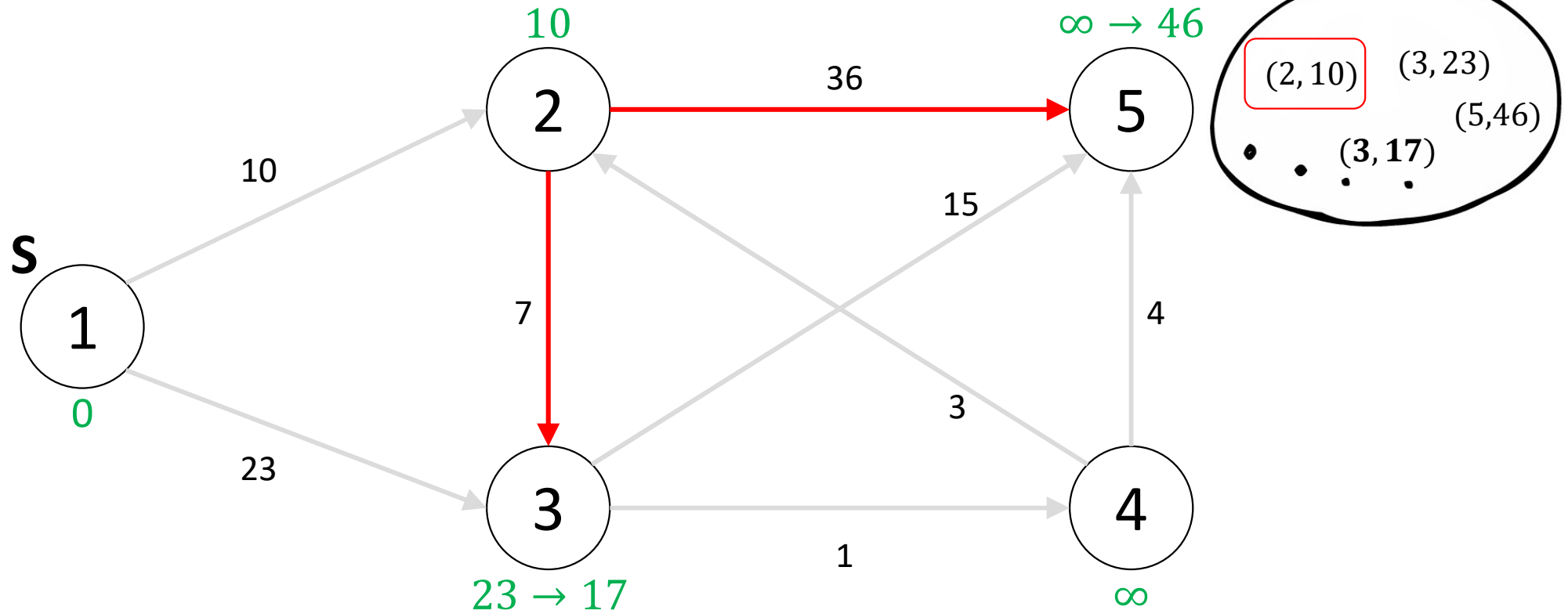
I Dijkstra Algorithm



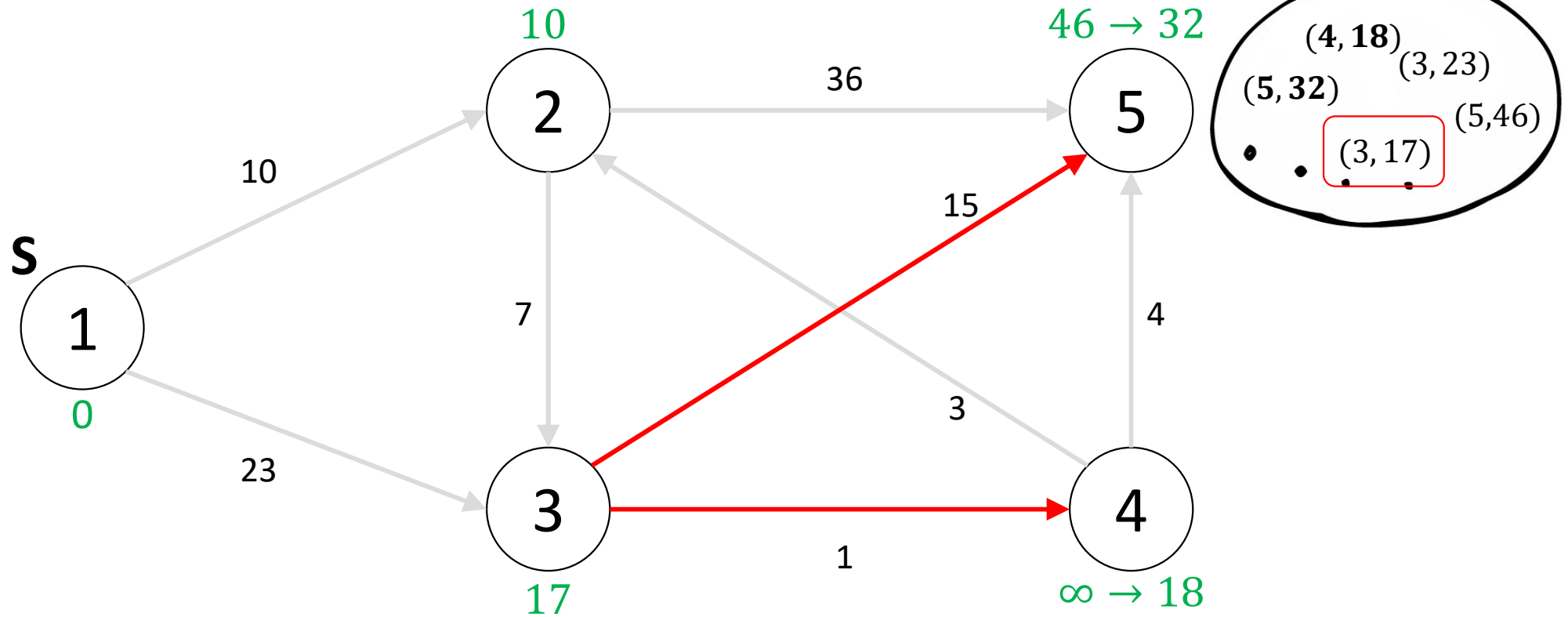
I Dijkstra Algorithm



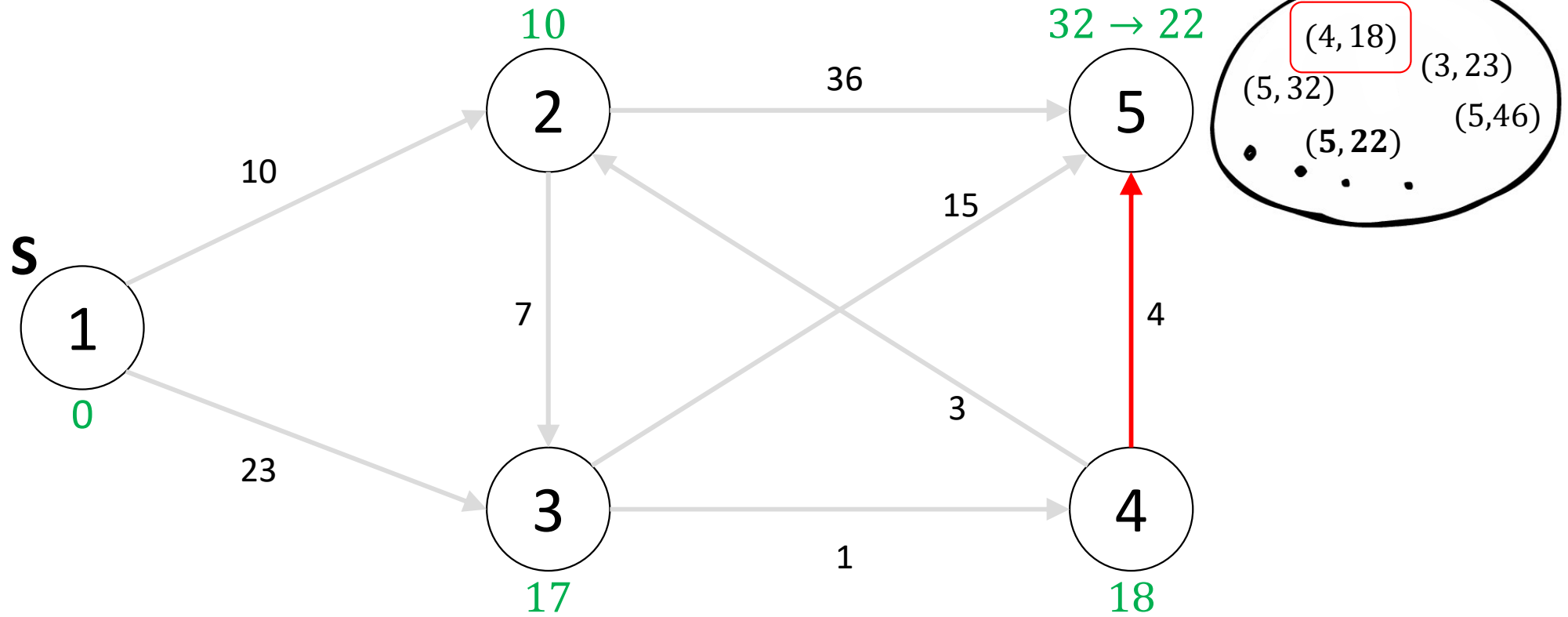
I Dijkstra Algorithm



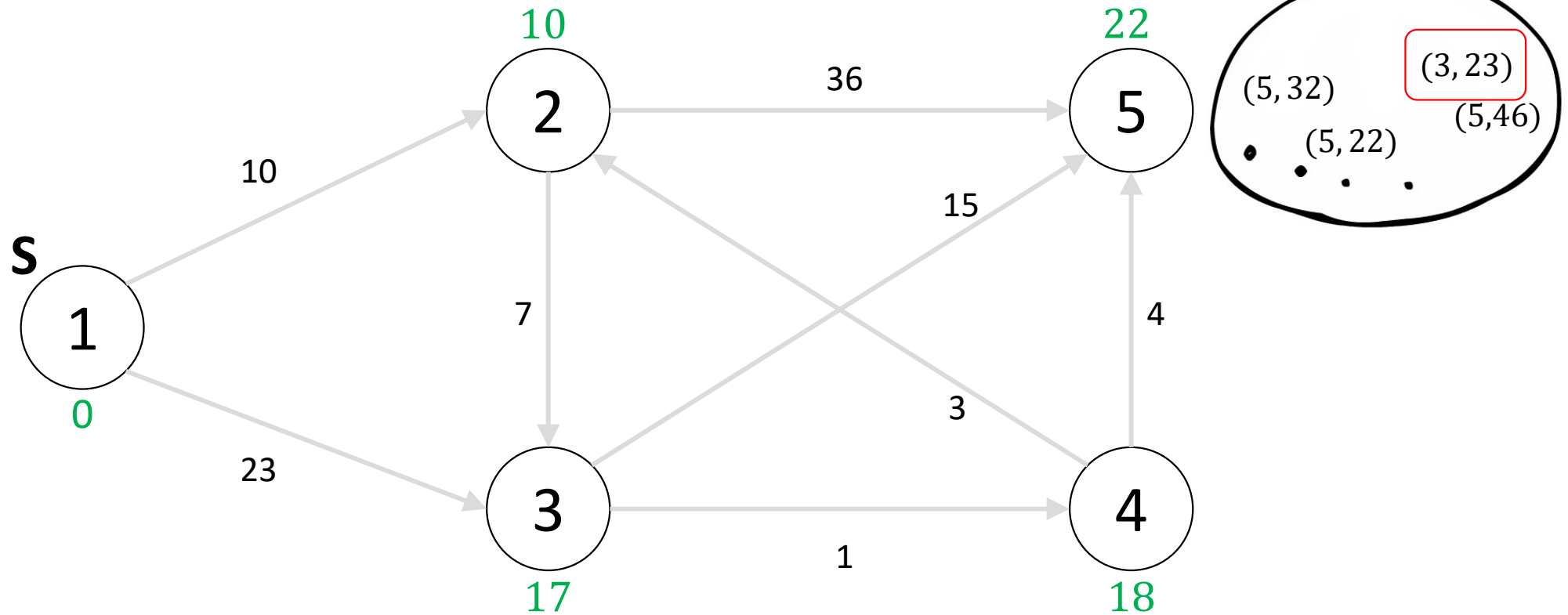
I Dijkstra Algorithm



I Dijkstra Algorithm



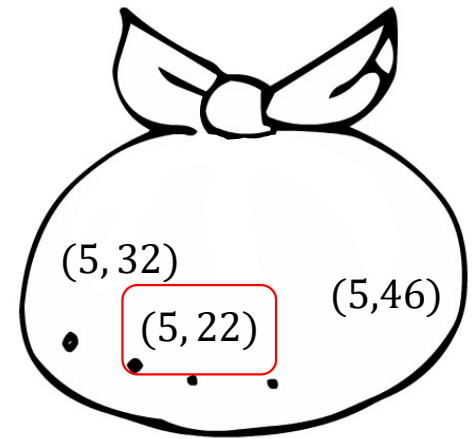
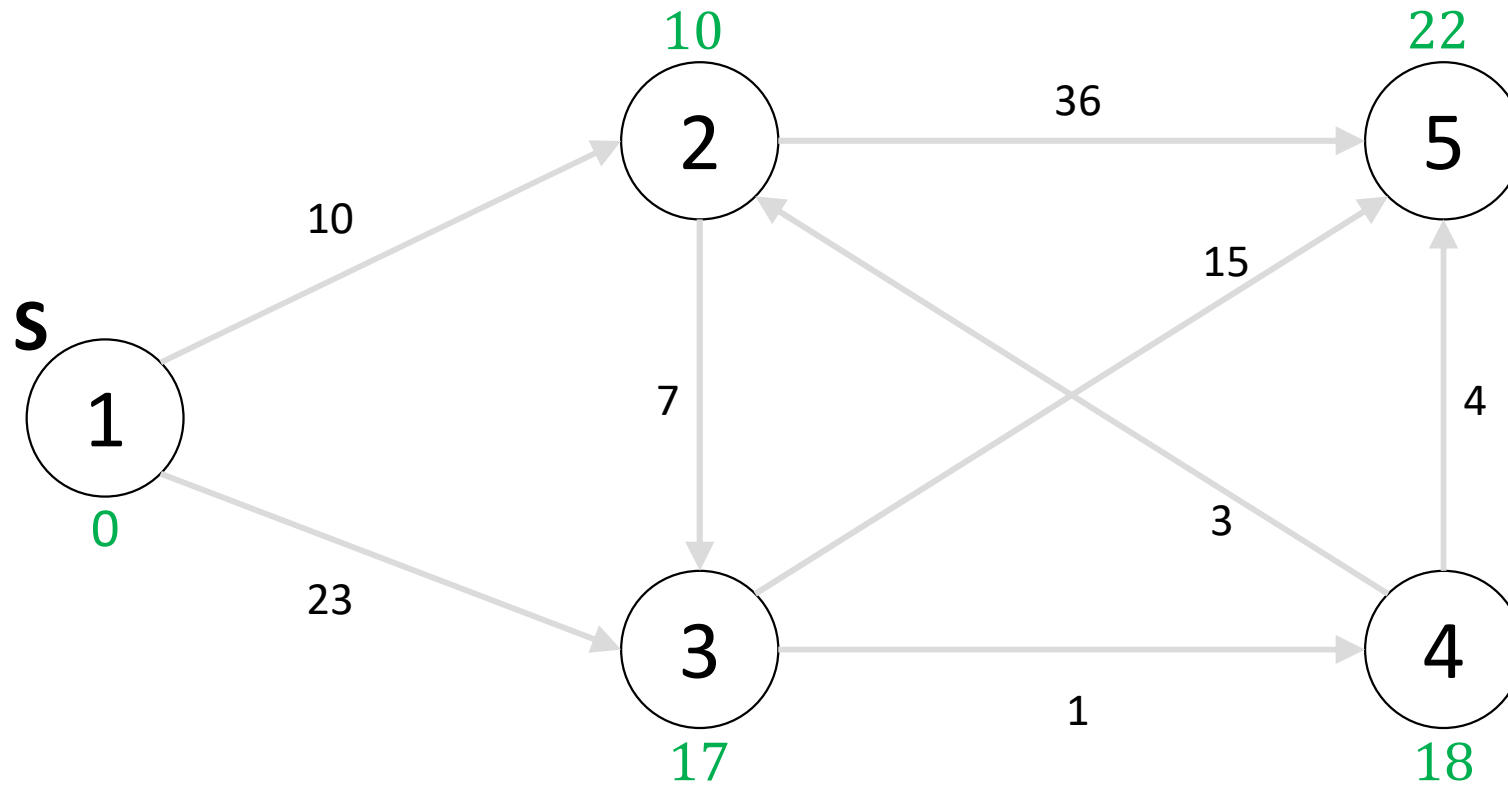
I Dijkstra Algorithm



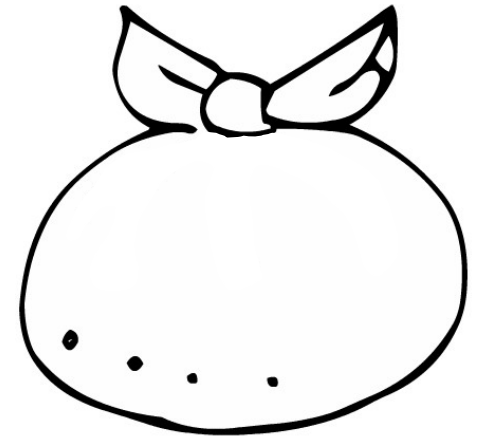
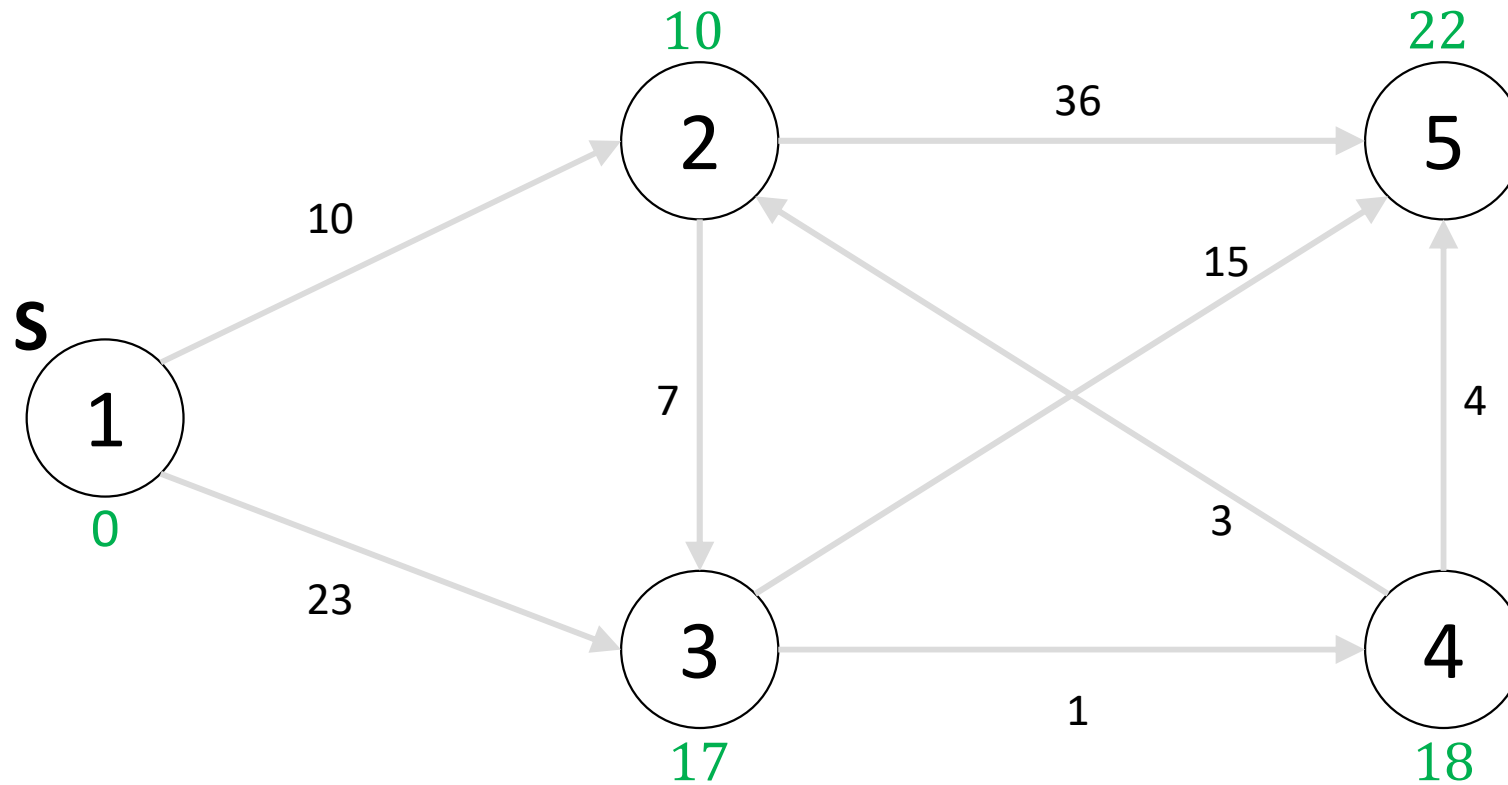
3번 점까지 17만에 가는 길이 있다

➔ 굳이 23이라는 거리를 이용해서 다른 정점들을 갱신할 필요가 없다.

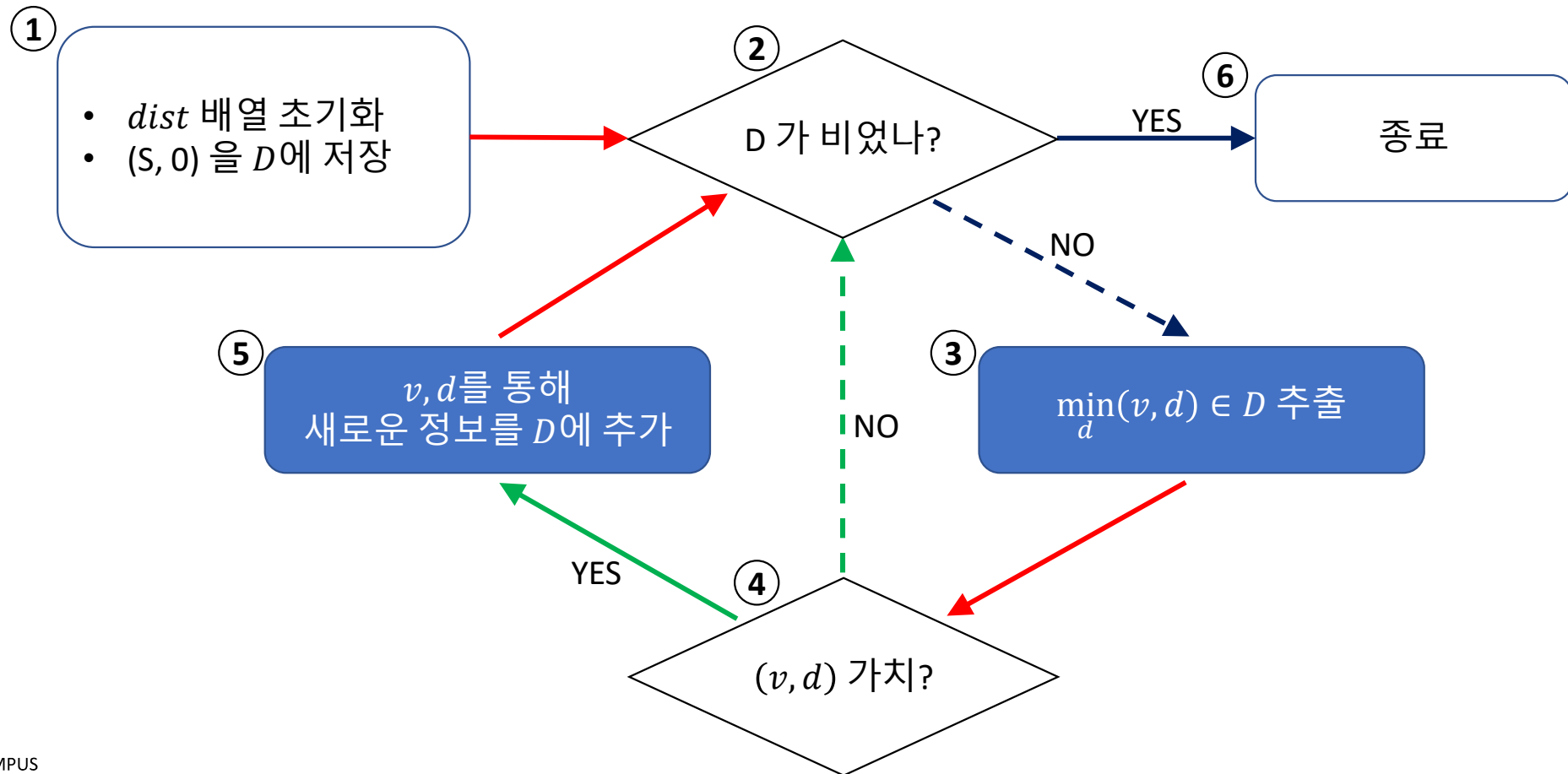
I Dijkstra Algorithm



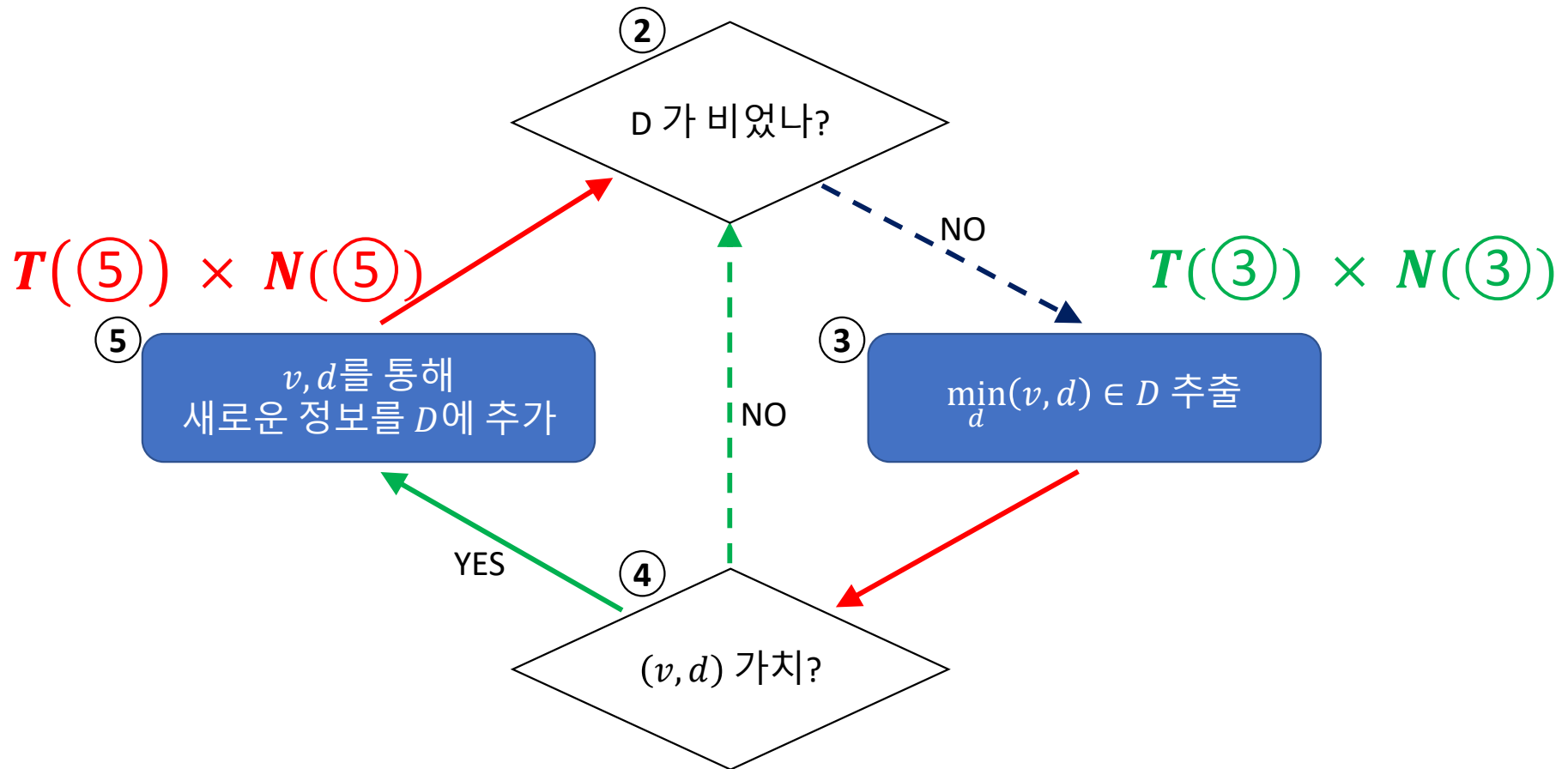
I Dijkstra Algorithm



I Dijkstra Algorithm – 시간 복잡도



I Dijkstra Algorithm – 시간 복잡도



I Dijkstra Algorithm – 시간 복잡도

$$T(\text{전체 시간}) = T(\textcircled{5}) \times N(\textcircled{5}) + T(\textcircled{3}) \times N(\textcircled{3})$$

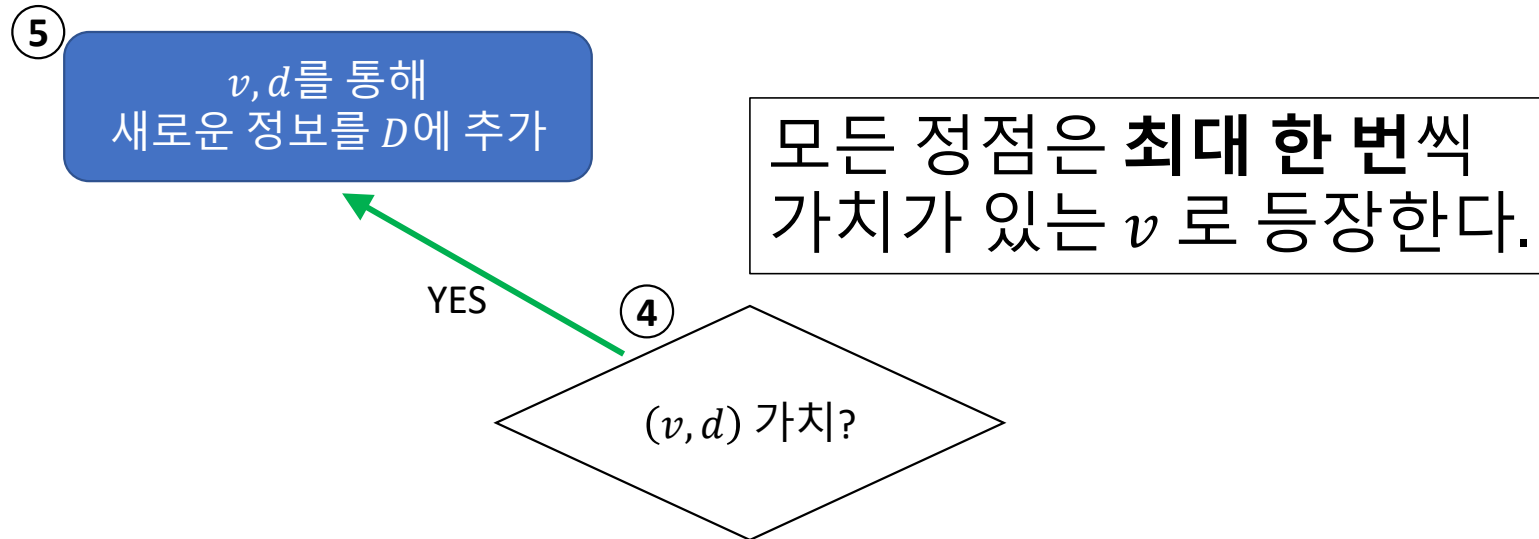
$N(\textcircled{3}) := D$ 에서 원소가 추출(poll)된 횟수

$N(\textcircled{5}) := D$ 에 원소가 추가(add)된 횟수

$$\rightarrow N(\textcircled{3}) \leq N(\textcircled{5})$$

$$\rightarrow T(\text{전체 시간}) \leq [T(\textcircled{5}) + T(\textcircled{3})] \times N(\textcircled{5})$$

I Dijkstra Algorithm – 시간 복잡도



$$N(\textcircled{5}) := \deg(1) + \deg(2) + \dots + \deg(V) = |E| \text{ (간선의 개수)}$$

$$\rightarrow T(\text{전체 시간}) \leq [T(\textcircled{5}) + T(\textcircled{3})] \times E$$

I Dijkstra Algorithm – 시간 복잡도

$$T(\text{전체 시간}) \leq [T(\textcircled{5}) + T(\textcircled{3})] \times E$$

$T(\textcircled{3}) := D$ 에서 최소 d 를 갖는 자료를 추출하는 시간

$T(\textcircled{5}) := D$ 에 임의의 (v, d) 라는 자료를 삽입하는 시간

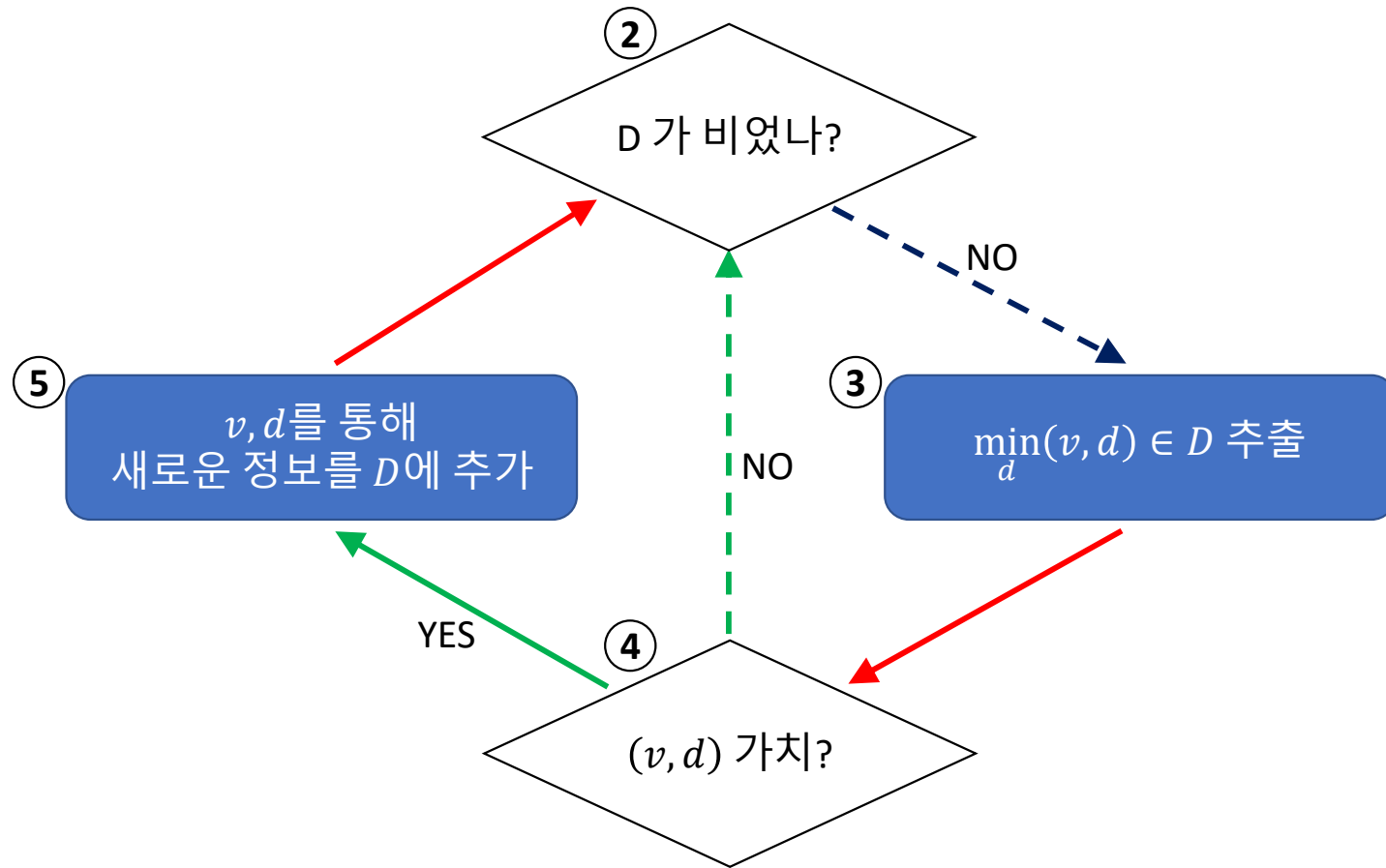
이 둘은 D 를 어떤 “자료구조로 구현하냐”에 따라 달라진다.

	$T(\textcircled{3})$	$T(\textcircled{5})$
1차원 배열	$O(N)$	$O(1)$
Min Heap / Priority Queue	$O(\log E)$	$O(\log E)$

$$T(\text{전체 시간}) \leq [T(\textcircled{5}) + T(\textcircled{3})] \times E = O(E \log E)$$

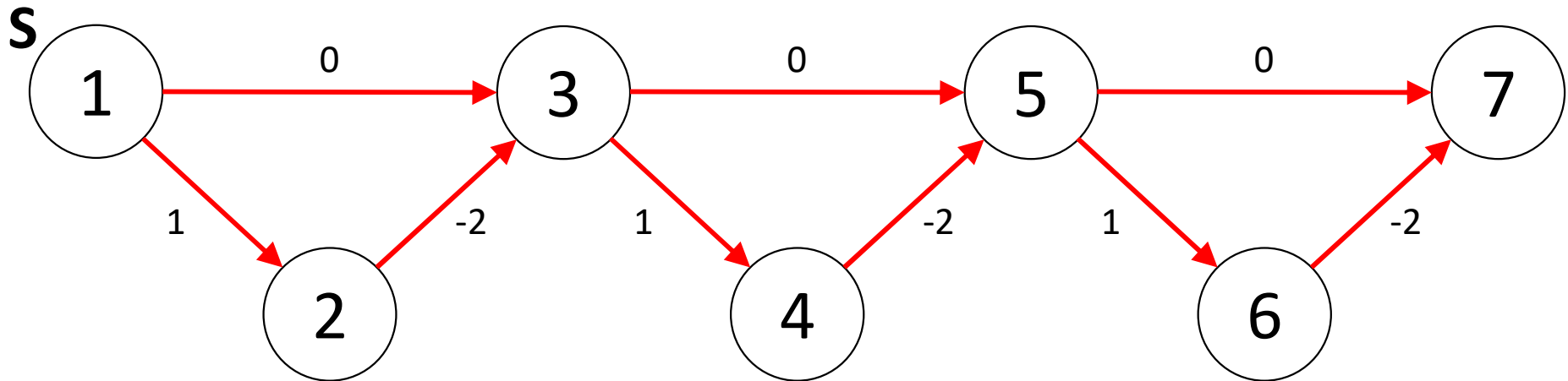
I Dijkstra Algorithm – 시간 복잡도

$$T(\text{전체 시간}) = O(E \log E) = O(E \log V)$$



I Dijkstra Algorithm; 음수 간선이 있으면 왜 안되나?

<시간 복잡도가 보장되지 않는 Example>




한 정점이 v 로 추출되는 횟수가 한 번이 아니다

→ $N(\textcircled{5}) \gg |E|$ 이 된다.

I Dijkstra Algorithm; 코드 설명

①

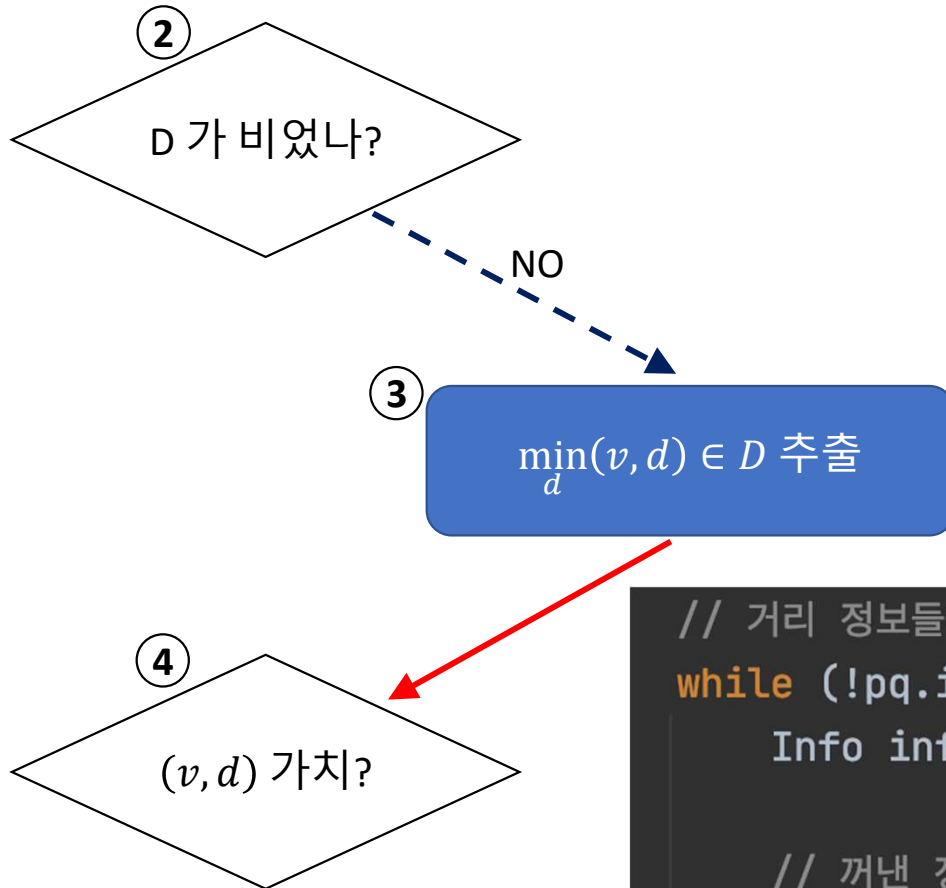
- *dist* 배열 초기화
 - (S, 0) 을 *D*에 저장
- 

```
static void dijkstra(int start) {
    // 모든 정점까지에 대한 거리를 무한대로 초기화 해주기.
    // ※주의사항※
    // 문제의 정답으로 가능한 거리의 최댓값보다 큰 값을 보장해야 한다.
    for (int i = 1; i <= N; i++) dist[i] = Integer.MAX_VALUE;

    // 최소 힙 생성
    PriorityQueue<Info> pq = new PriorityQueue<>(Comparator.comparingInt(o -> o.dist));
    // 다른 방법) PriorityQueue<Info> pq = new PriorityQueue<>((o1, o2) -> o1.dist - o2.dist);

    // 시작점에 대한 정보(Information)을 기록에 추가하고, 거리 배열(dist)에 갱신해준다.
    pq.add(new Info(start, _dist: 0));
    dist[start] = 0;
}
```

I Dijkstra Algorithm; 코드 설명



```
// 거리 정보들이 모두 소진될 때까지 거리 갱신을 반복한다.
while (!pq.isEmpty()) {
    Info info = pq.poll();

    // 꺼낸 정보가 최신 정보랑 다르면, 의미없이 낮은 정보이므로 폐기한다.
    if (dist[info.idx] < info.dist) continue;
```

I Dijkstra Algorithm; 코드 설명

5

v, d 를 통해
새로운 정보를 D 에 추가

```
// 연결된 모든 간선들을 통해서 다른 정점들에 대한 정보를 갱신해준다.  
for (Edge e : edges[info.idx]) {  
    if (dist[info.idx] + e.weight >= dist[e.to]) continue;  
  
    // e.to 까지 갈 수 있는 더 짧은 거리를 찾았다면 이에 대한 정보를 갱신하고 PQ에 기록해준다.  
    dist[e.to] = dist[info.idx] + e.weight;  
    pq.add(new Info(e.to, dist[e.to]));  
}
```

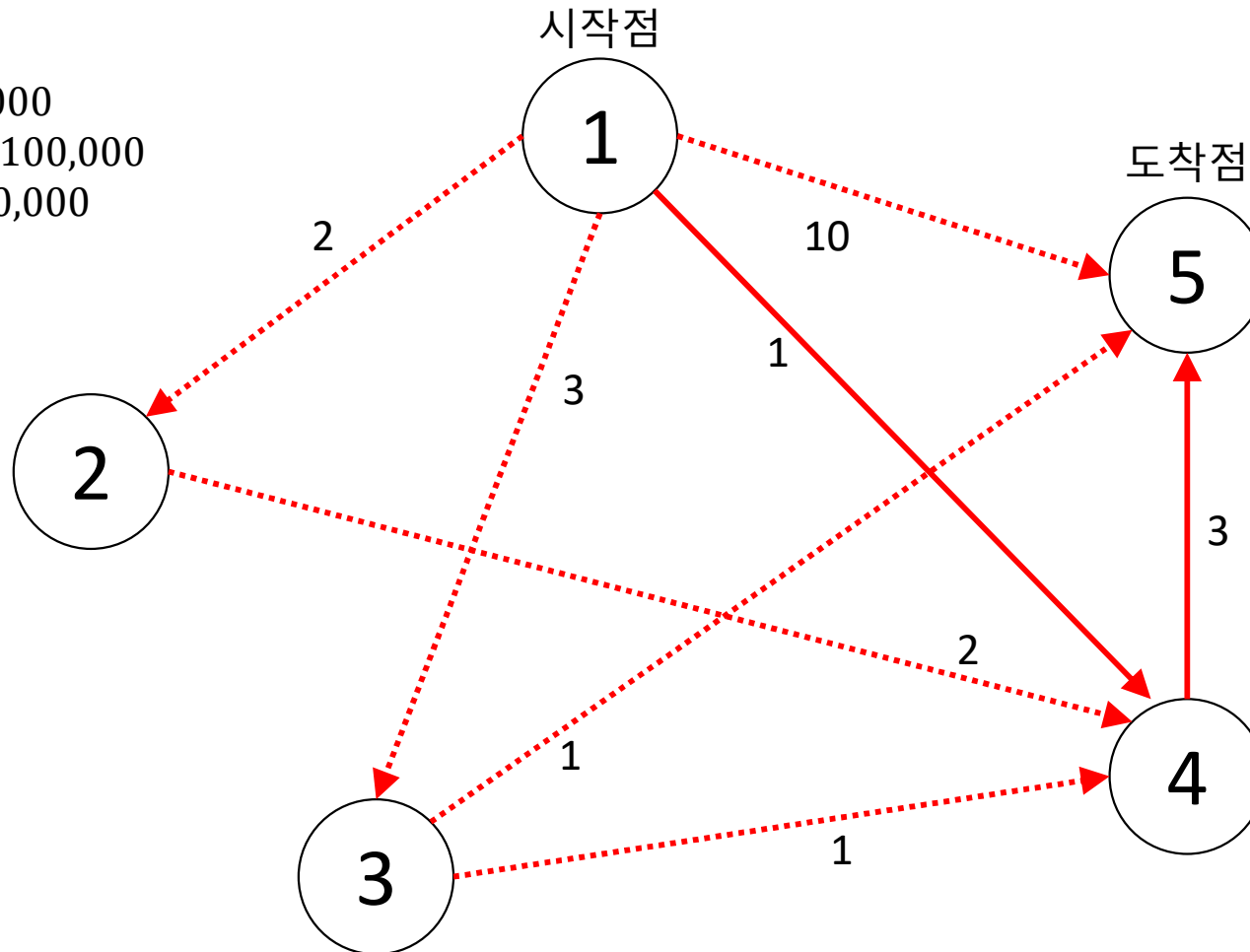
I BOJ 1916 – 최소비용 구하기

난이도: 3

$1 \leq \text{도시 수}, N \leq 1,000$

$1 \leq \text{버스 개수}, M \leq 100,000$

$0 \leq \text{버스 비용} \leq 100,000$



I 정답의 최대치 확인

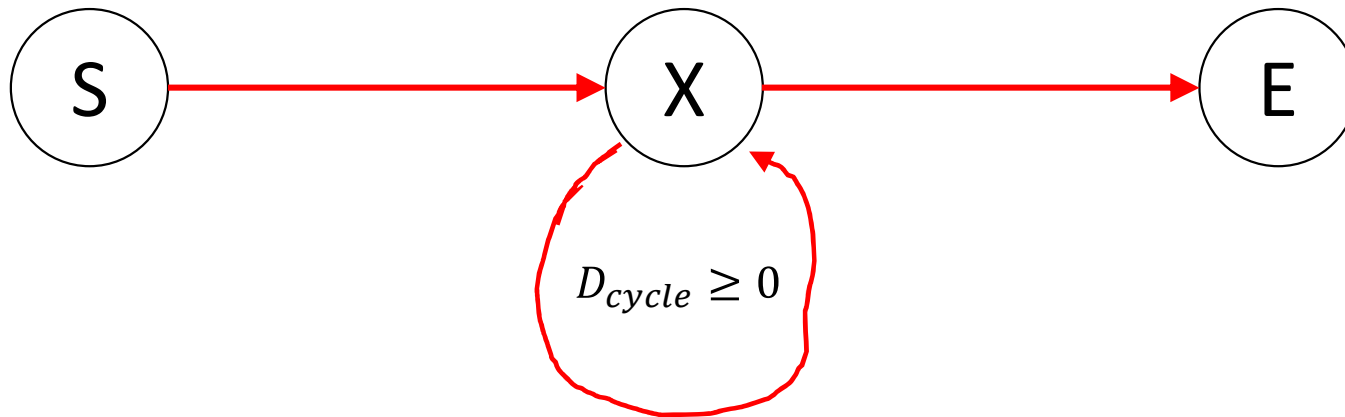
<최단거리의 특성>

같은 정점을 두 번 방문할 이유가 없다.

따라서 정답은 최대

$$\text{버스 비}(10\text{만}) * \text{정점 수}(1\text{천}) = 1\text{억}$$

이하이므로 Integer로 충분하다.



I 시간, 공간 복잡도 계산하기

Dijkstra Algorithm 을 통해 시작점에서 모든 점까지의
최단 거리 계산 ➔ 자동으로 도착점에 대한 결과 도출

$O(E \log V)$, 10만 $\times \log$ 1천 \cong 100만 ➔ 1초 안에 가능

I 구현

```
static void dijkstra(int start) {  
    // 모든 정점까지에 대한 거리를 무한대로 초기화 해주기.  
    /* TODO */  
  
    // 최소 힙 생성  
    /* TODO */  
  
    // 시작점에 대한 정보(Information)을 기록에 추가하고, 거리 배열(dist)에 갱신해준다.  
    /* TODO */  
  
    // 거리 정보들이 모두 소진될 때까지 거리 갱신을 반복한다.  
    /* TODO */  
}
```