2025 겨울방학 알고리즘 스터디

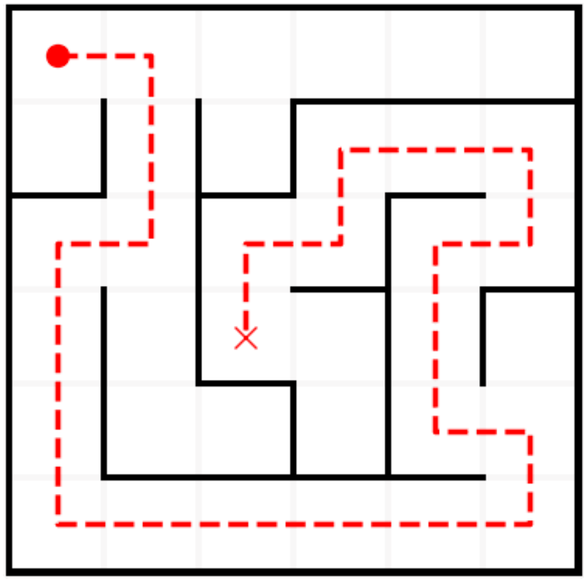
그래프 탐색

컴퓨터 공학과 20230546 서보경

목차

깊이 우선 탐색(dfs)이란?

깊이 우선 탐색(dfs)은 그래프나 트리에서 한 정점에서 시작하여 가능한 한 깊이 있는 정점까지 탐색한 후, 더 이상 갈 곳이 없으면 되돌아오는 방식이다. 이 작업들은 시간 복잡도 O(V+E)을 보장한다. 주로 그래프 탐색의 일종으로, 깊은 정점을 우선적으로 방문해야 하는 경우에 사용된다. 또한 스택(LIFO) 구조를 사용하여 구현되며, 경로 탐색, 백트래킹, 연결 요소 찾기 등 다양한 응용이 가능하다.



dfs를 사용하는 대표적 예제 : 미로 탈출

깊이 우선 탐색(dfs)

*n*개의 정점과 *n* - 1개의 간선으로 구성된 트리 T가 있다. 정점 번호는 0부터 *n* - 1까지이고 0번 정점이 루트이다. 모든 간선의 길이는 1이다. 트리 T의 각 정점에는 사과가 0개 또는 1개 놓여있다. 루트 노드에서 거리가 *k*이하인 노드에 있는 사과를 수확하려고 한다. 수확할 수 있는 사과 개수를 출력하자.

입력

첫 번째 줄에 정점의 수 *n*과 정수 *k*가 공백을 사이에 두고 순서대로 주어진다.

두 번째 줄부터 *n* - 1개 줄에 걸쳐 간선의 정보가 주어진다. 한 줄에 하나의 간선 정보가 주어진다. 하나의 간선 정보는 부모 정점 번호 *p*와 자식 정점 번호 *c*가 공백을 사이에 두고 순서대로 주어진다.

다음 줄에는 0번 정점부터 *n* - 1번 정점까지 정점의 사과 정보를 나타내는 *n*개의 정수가 공백을 사이에 두고 순서대로 주어진다. *i*번째 수는 *i* - 1번 정점에 있는 사과의 수를 나타낸다. 사과의 수는 0 또는 1이다.

출력

첫 번째 줄에 수확할 수 있는 사과 개수를 출력한다.

제한

$2 \le n \le 100,000$

$0 \le p, c \le n - 1, p \ne c$

간선들로 만들어진 그래프는 트리이다.

$0 \le k \le n - 1$

정점에 있는 사과의 수는 0 또는 1이다.

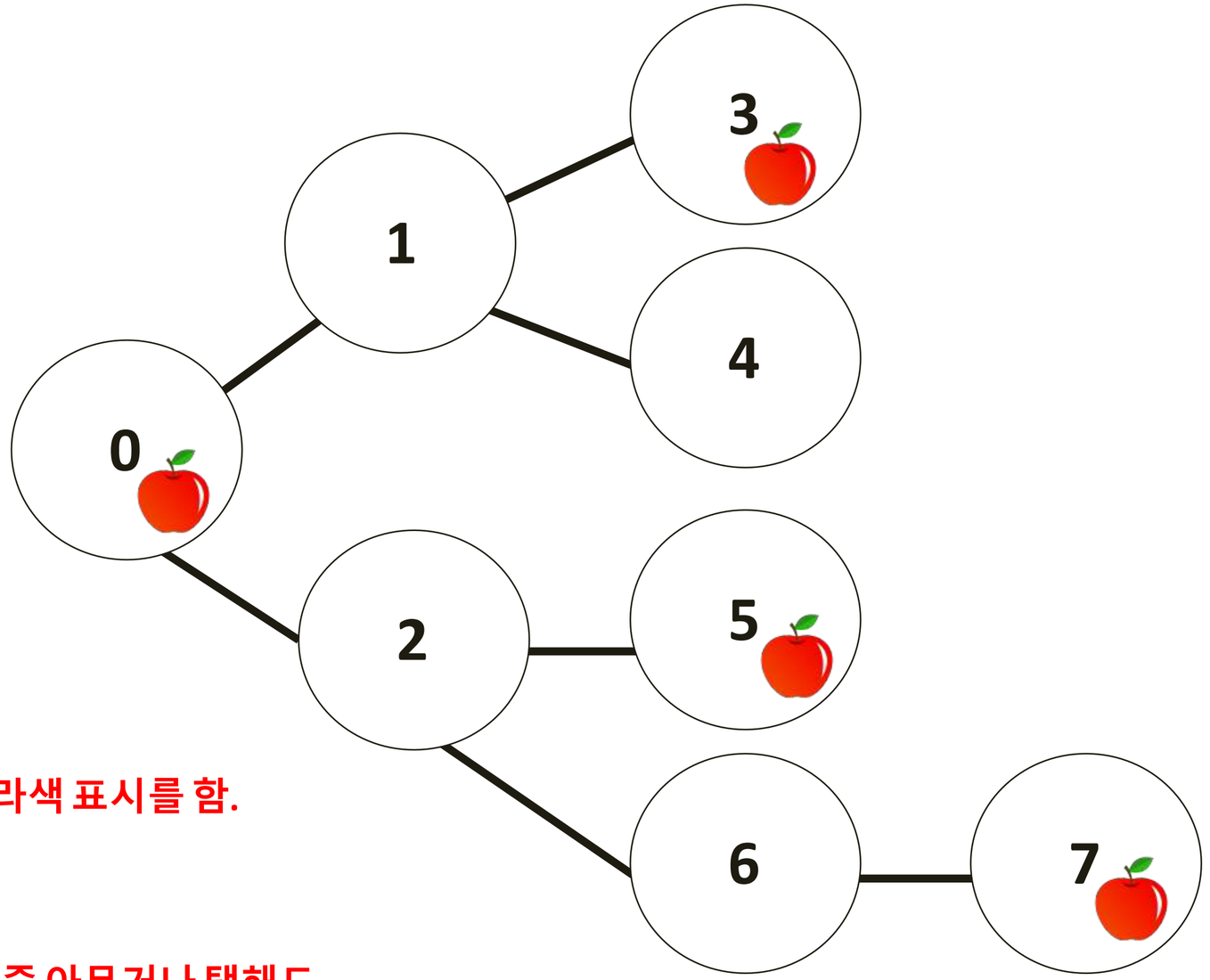
0번 부터 n-1번 까지 정점이 있으며, 이때 0번 정점은 루트 이다.
각 정점에는 사과가 0개 혹은 1개가 놓여져 있다.

1. **간선 정보를 입력 받은 뒤 그래프를 생성 한다.**
2. **사과 정보를 입력받은 후, 각 정점에 사과가 있는지 없는지 체크 한다.**
3. **깊이 우선 탐색을 통해 거리가 k 이하인 정점에서 총 몇 개의 사과를 얻을 수 있는지 계산한다.**

깊이 우선 탐색(dfs)

예제 입력 1 복사

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

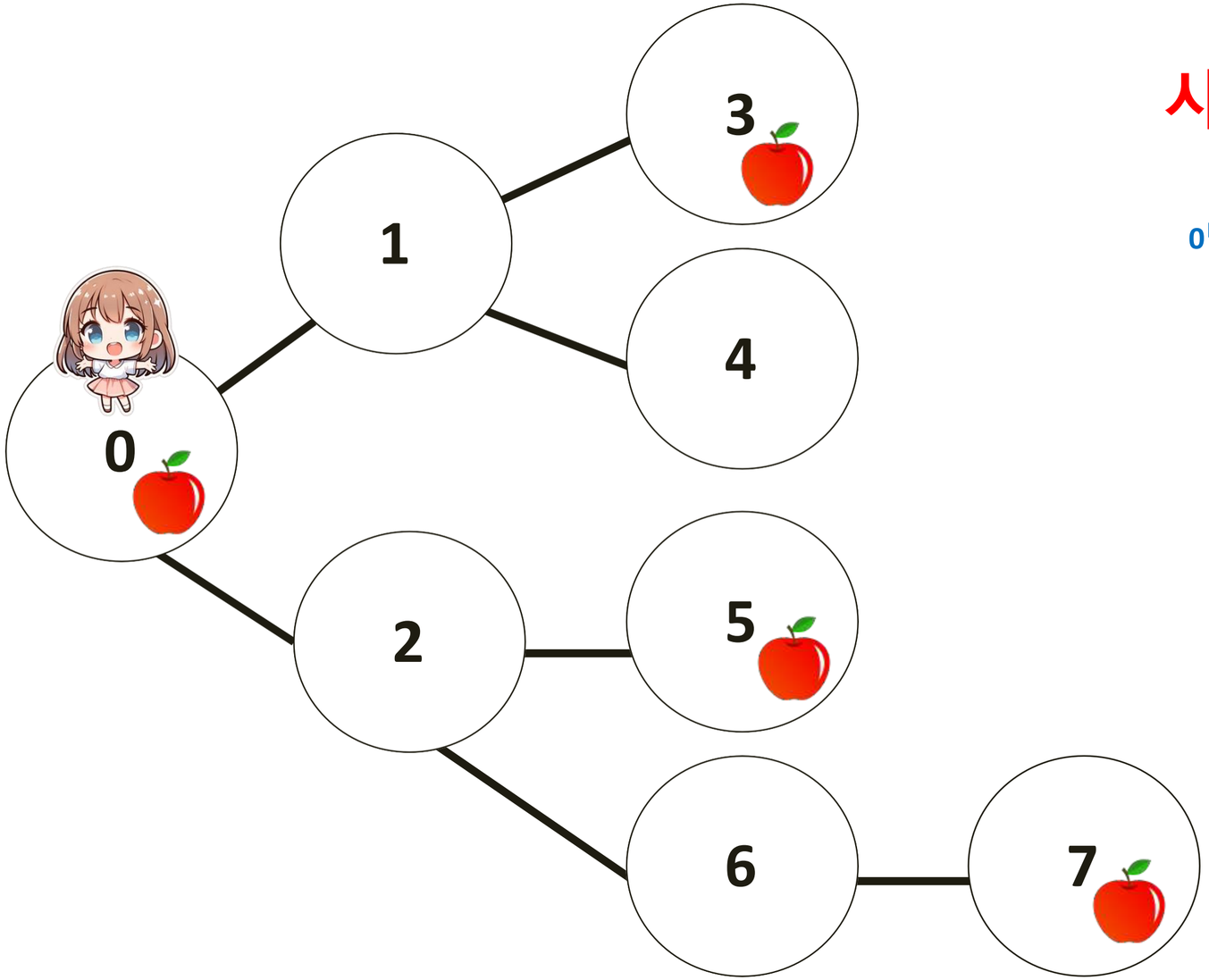


1 - 중복 방문이 일어나면 안되므로 이미 방문한 점은 보라색 표시를 함.

2 - 첫 방문은 루트 정점부터!

3 – 문제에 조건이 명시되어 있지 않으면 갈 수 있는 정점중 아무거나 택해도 상관 X

깊이 우선 탐색(dfs)

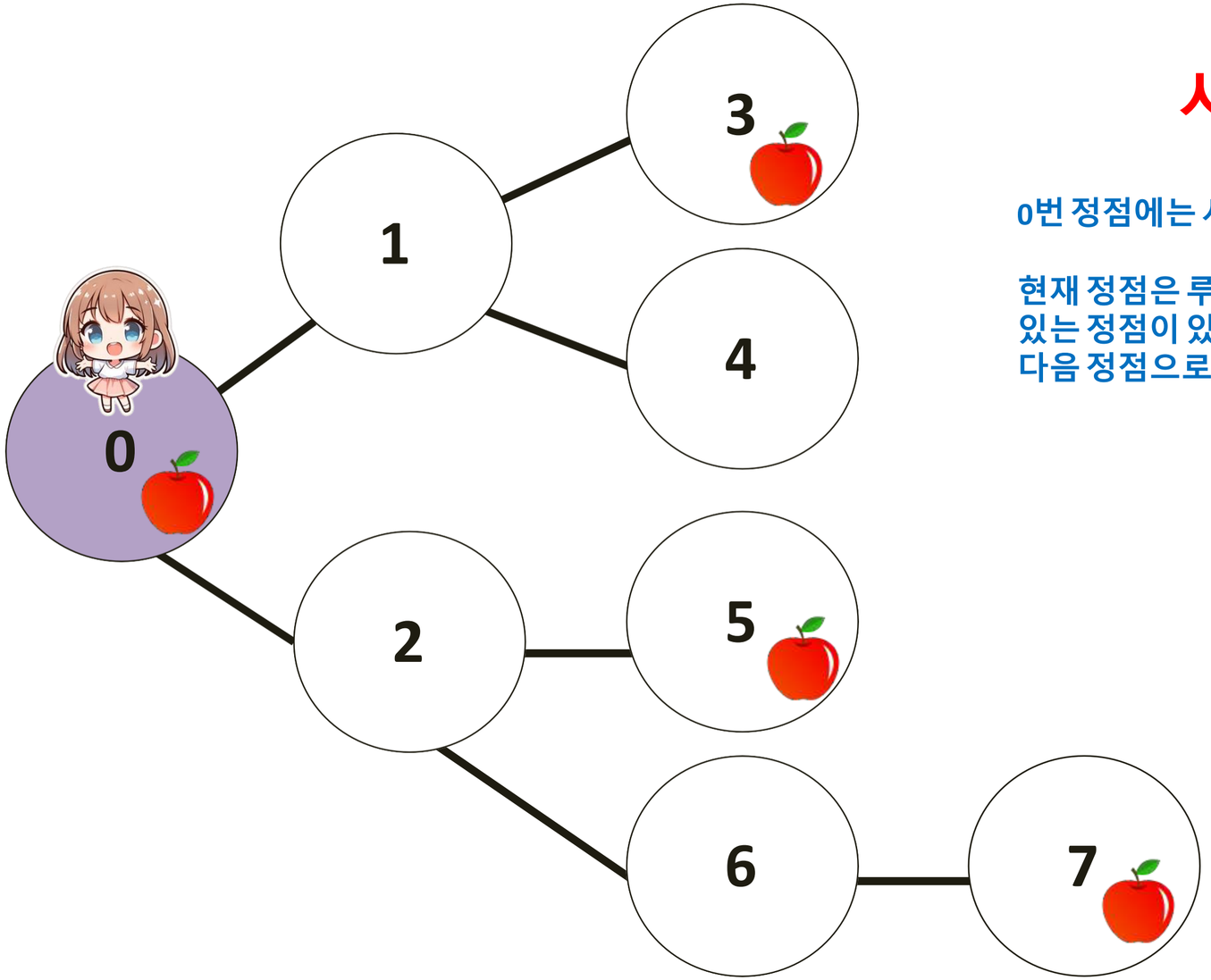


사과 합계 : 0

0번 정점에 도착했다.

STACK

깊이 우선 탐색(dfs)



사과 합계 : 1

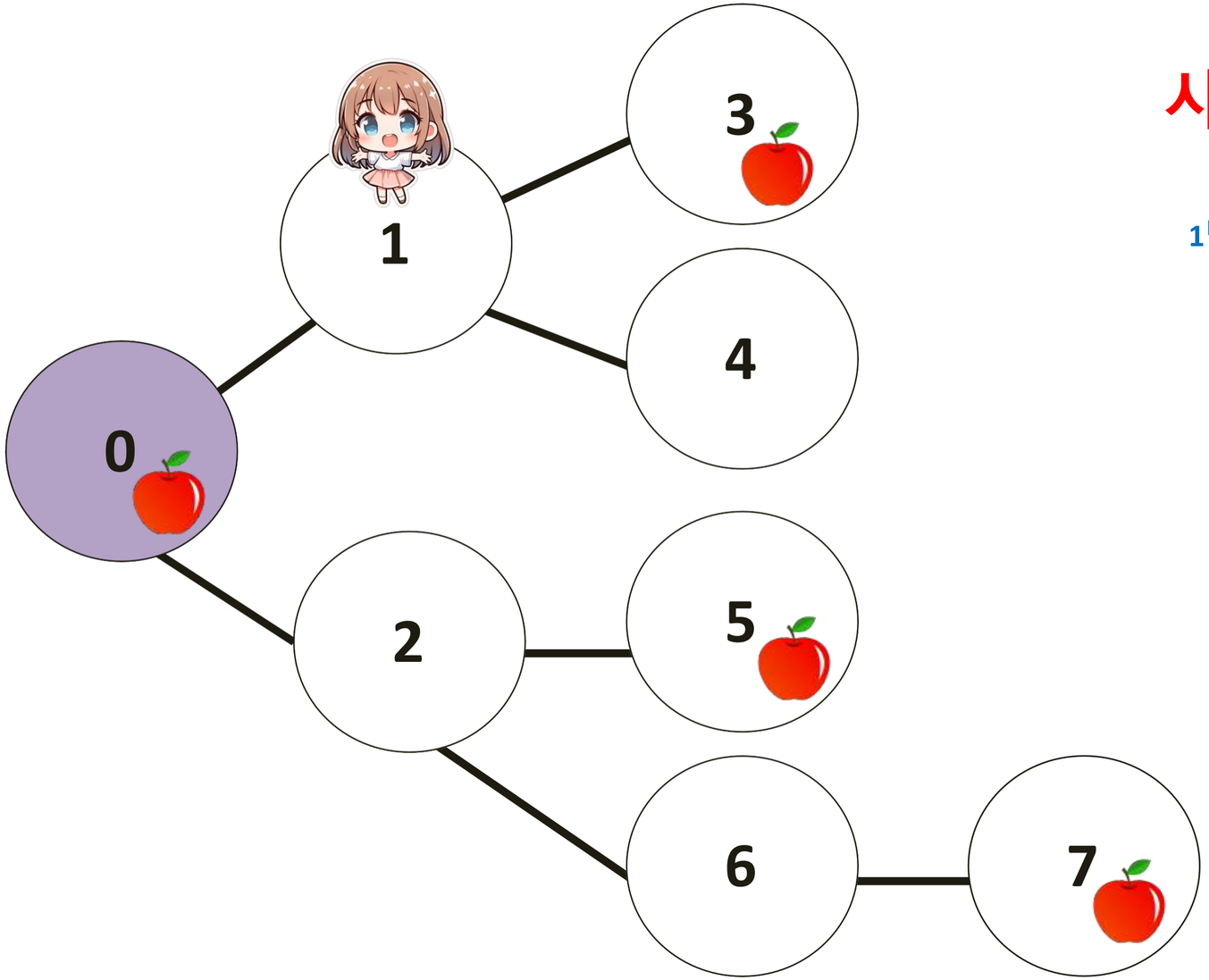
0번 정점에는 사과가 있으니 수확해준다.

현재 정점은 루트로부터 '0' 떨어진 거리이며, 갈 수 있는 정점이 있으므로 지금 정점을 스택에 넣어주고 다음 정점으로 이동할 준비를 해준다.

STACK

0번 정점

깊이 우선 탐색(dfs)



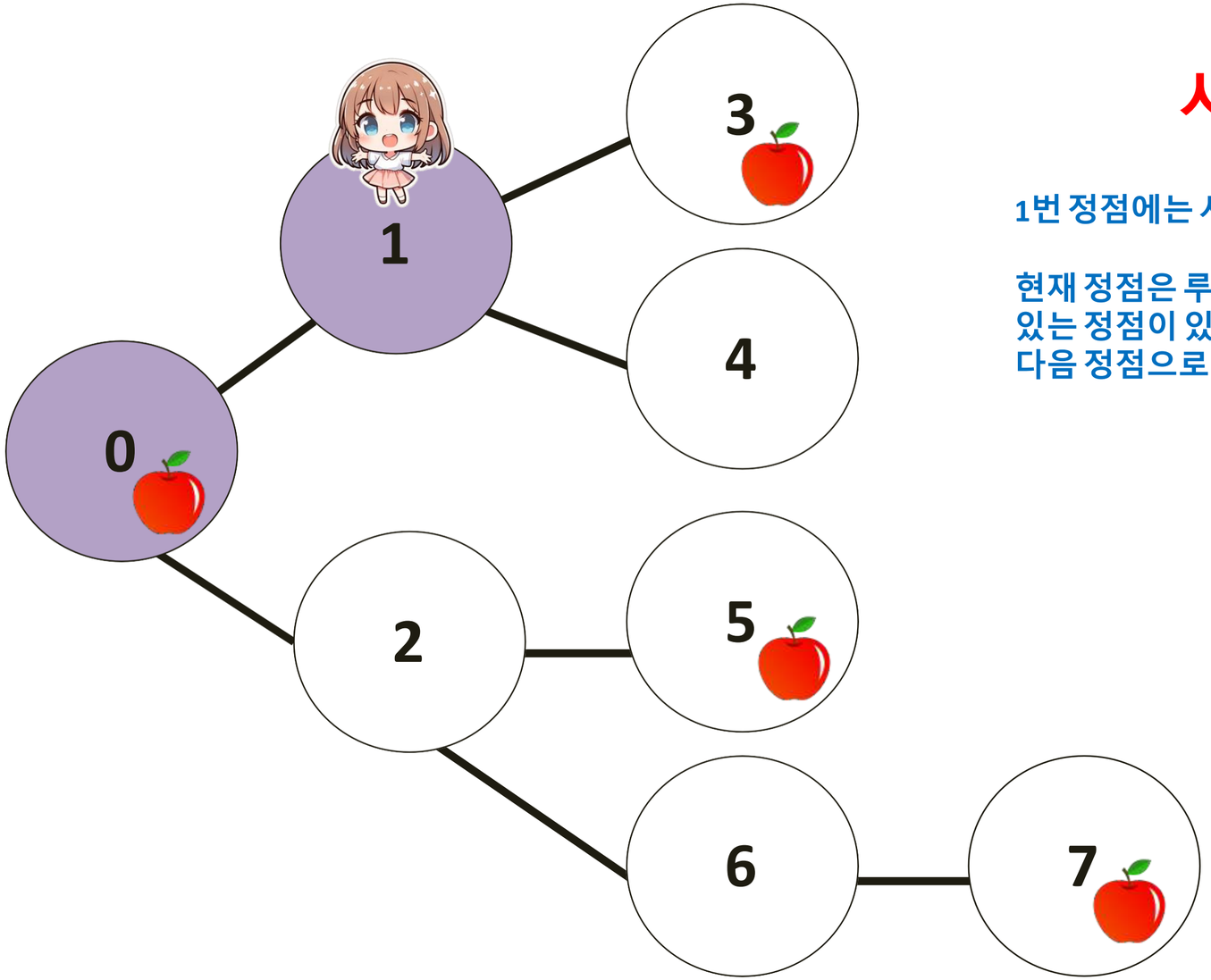
사과 합계 : 1

1번 정점에 도착했다.

STACK

0번 정점

깊이 우선 탐색(dfs)



사과 합계 : 1

1번 정점에는 사과가 없다.

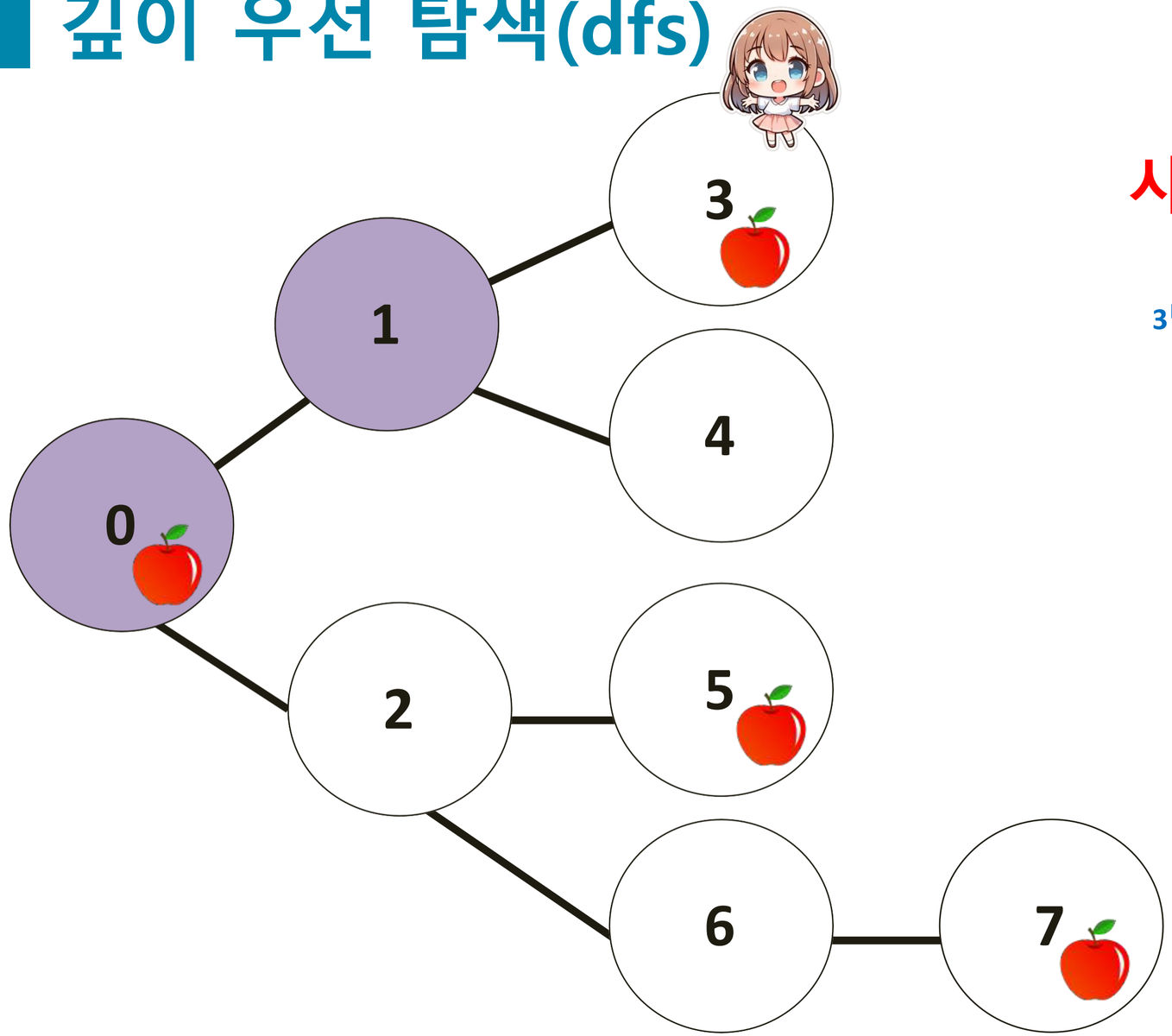
현재 정점은 루트로부터 '1' 떨어진 거리이며, 갈 수 있는 정점이 있으므로 지금 정점을 스택에 넣어주고 다음 정점으로 이동할 준비를 해준다.

STACK

1번 정점

0번 정점

깊이 우선 탐색(dfs)



사과 합계 : 1

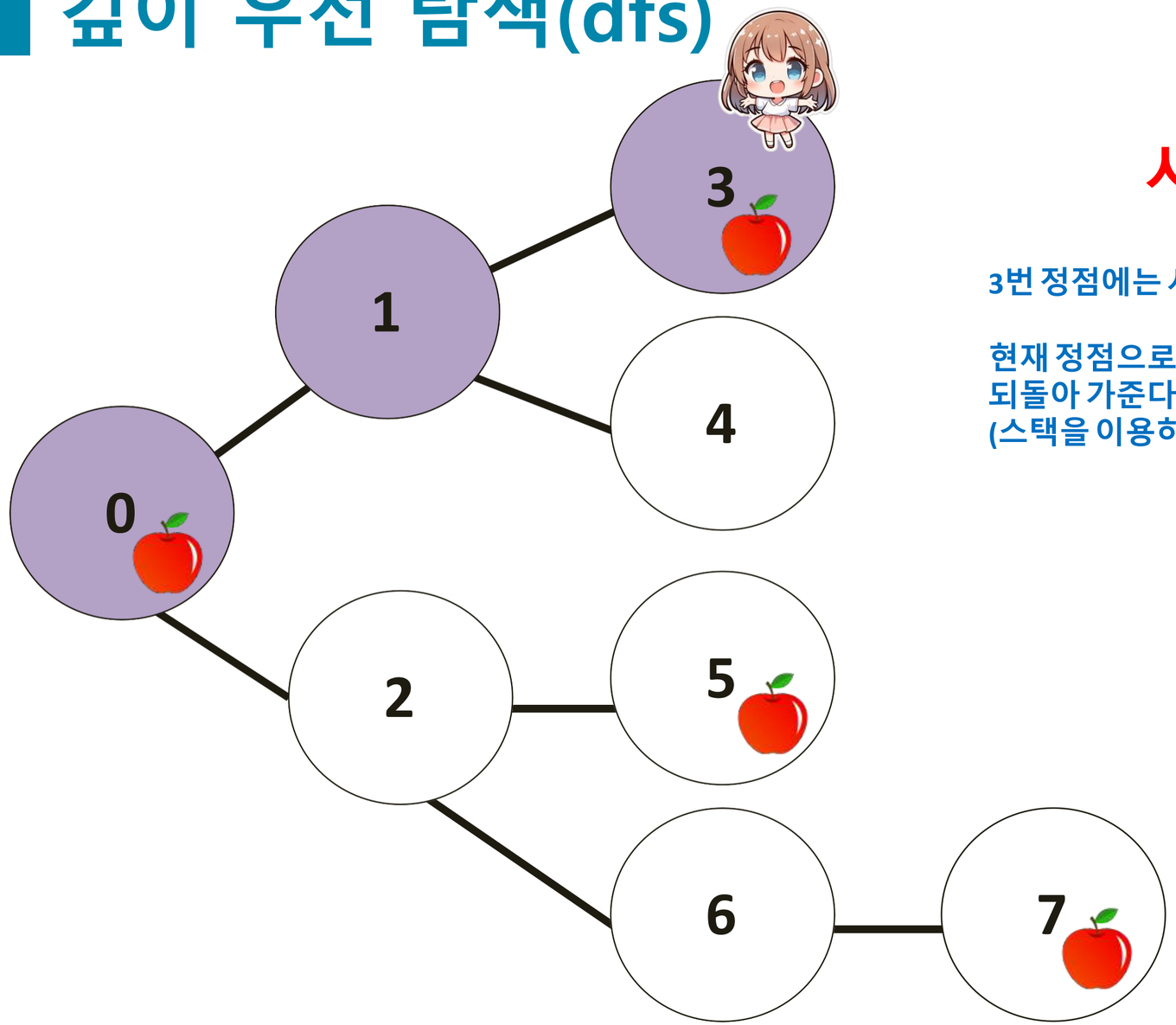
3번 정점에 도착했다.

STACK

1번 정점

0번 정점

깊이 우선 탐색(dfs)



사과 합계 : 2

3번 정점에는 사과가 있으니 수확해준다.

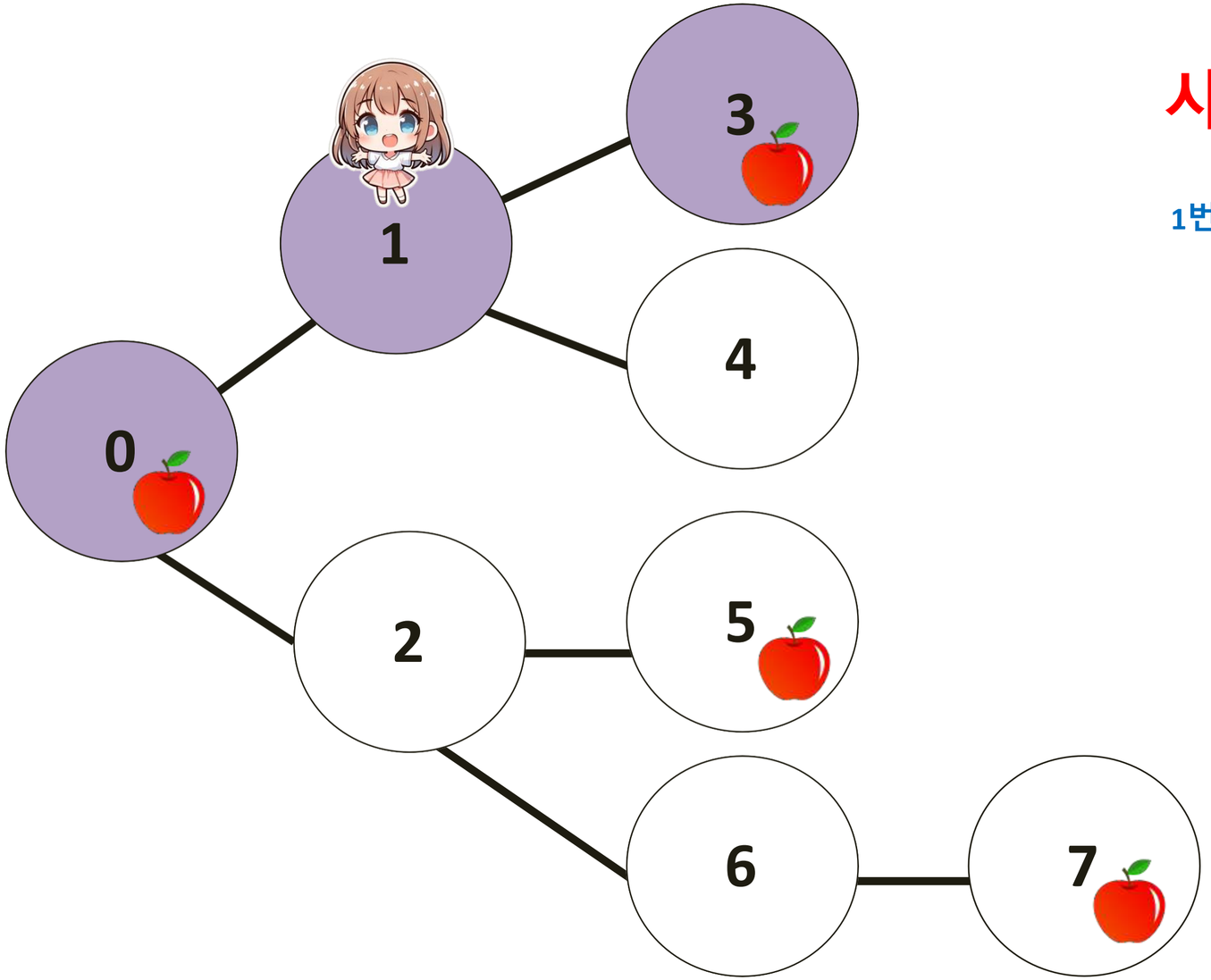
현재 정점으로부터 갈 수 있는 정점이 없으므로
되돌아 가준다.
(스택을 이용하여 지금까지 왔던 길을 되돌아가기)

STACK

1번 정점

0번 정점

깊이 우선 탐색(dfs)



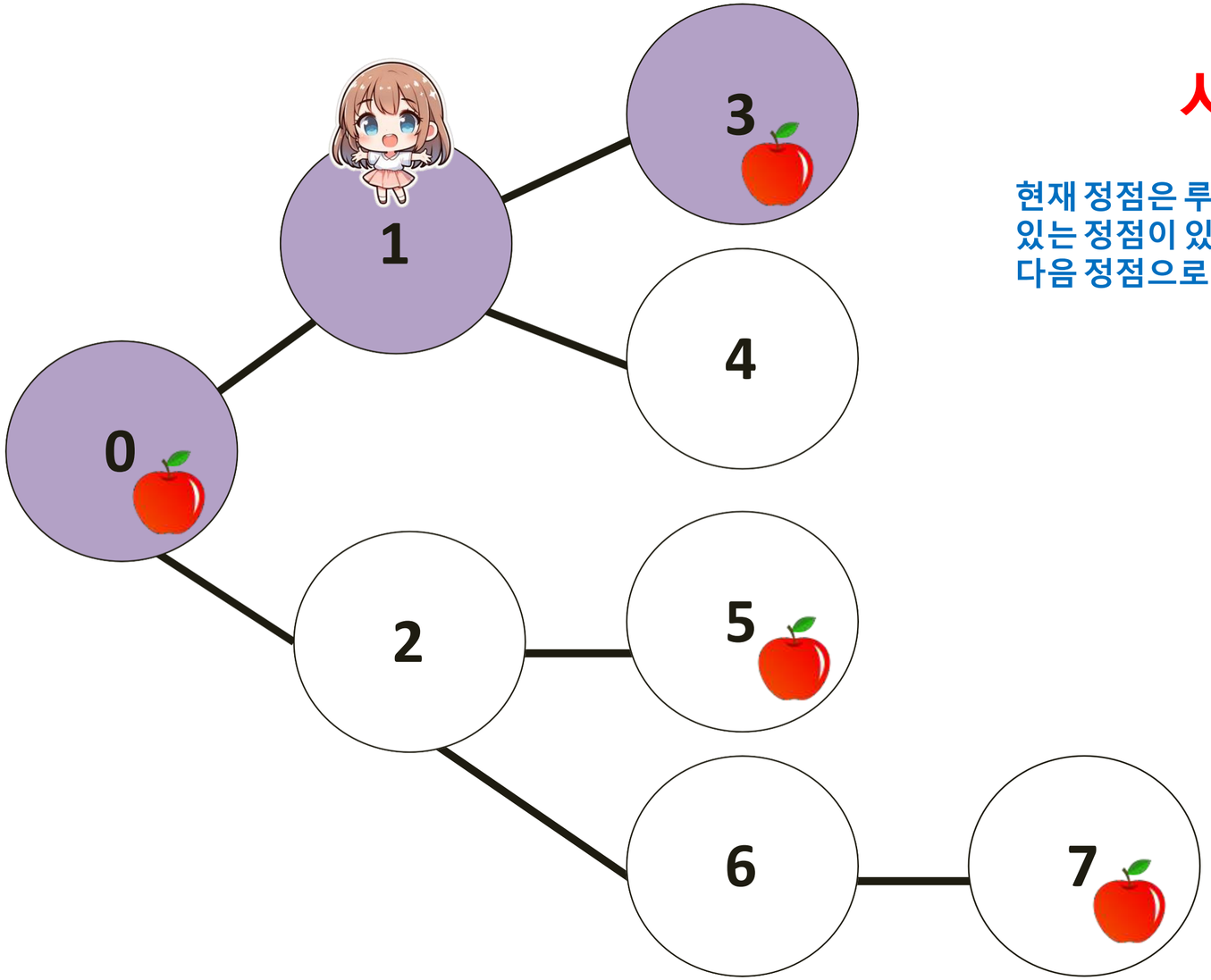
사과 합계 : 2

1번 정점으로 되돌아 왔다.

STACK

0번 정점

깊이 우선 탐색(dfs)



사과 합계 : 2

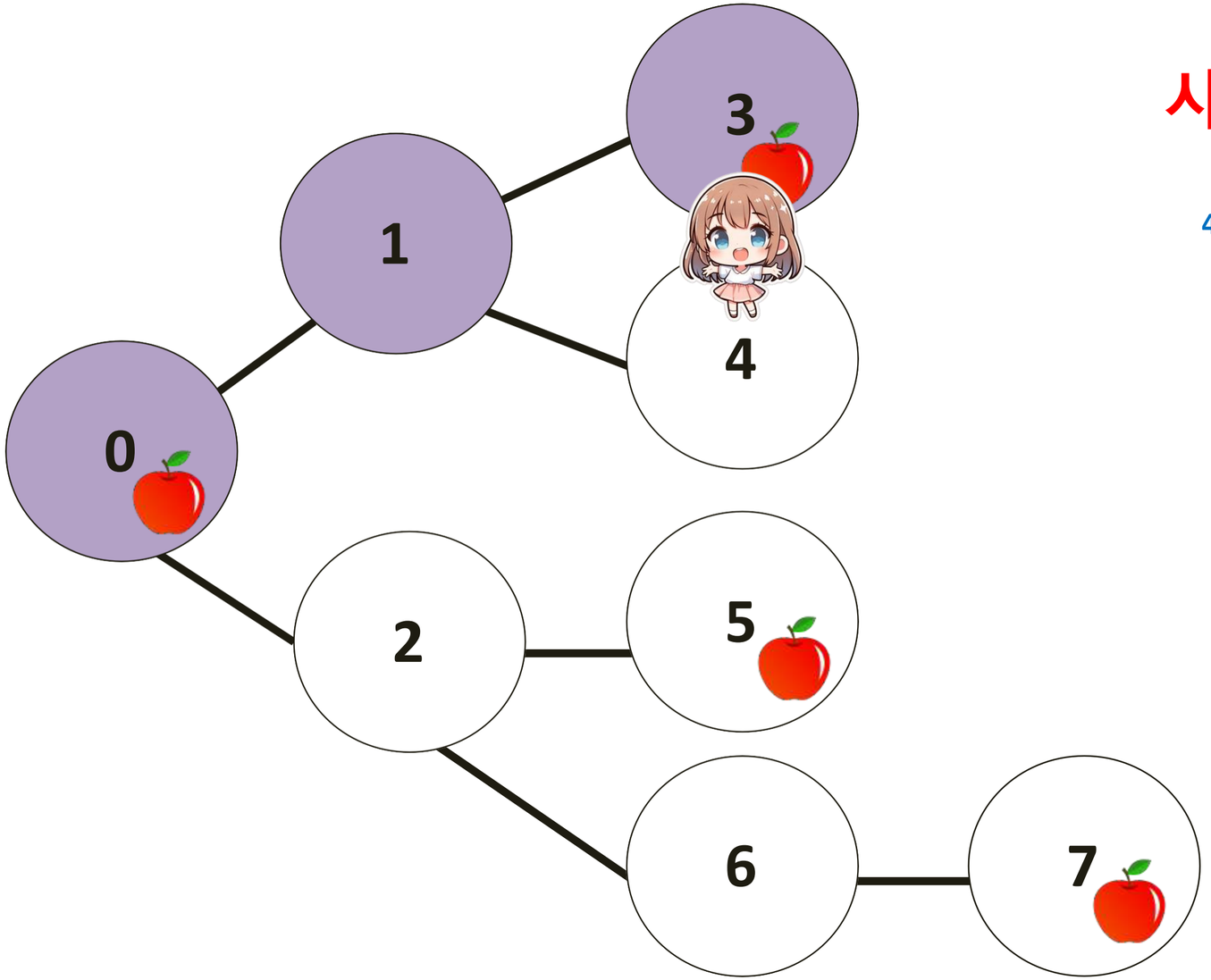
현재 정점은 루트로부터 '1' 떨어진 거리이며, 갈 수 있는 정점이 있으므로 지금 정점을 스택에 넣어주고 다음 정점으로 이동할 준비를 해준다.

STACK

1번 정점

0번 정점

깊이 우선 탐색(dfs)



사과 합계 : 2

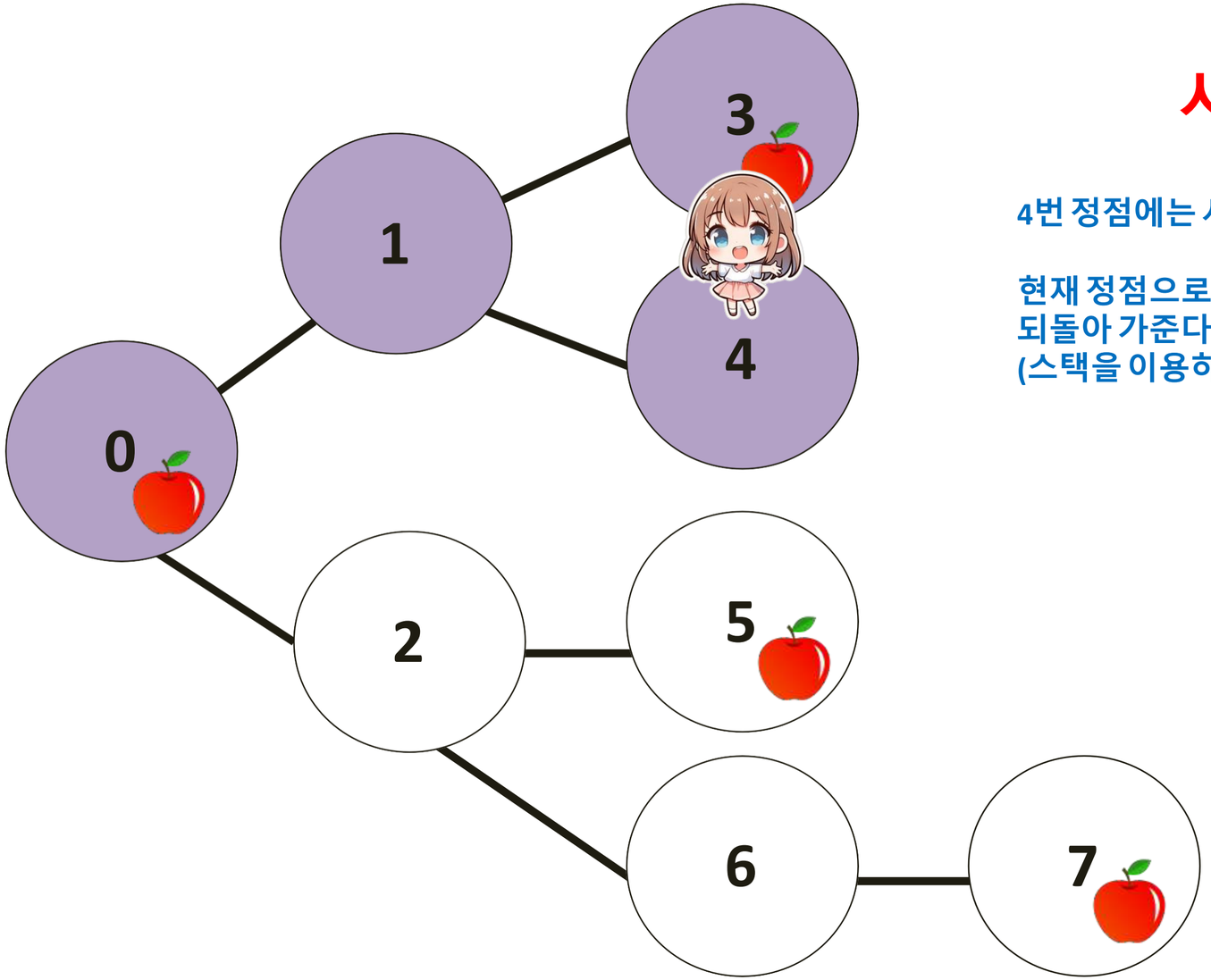
4번 정점에 도착했다.

STACK

1번 정점

0번 정점

깊이 우선 탐색(dfs)



사과 합계 : 2

4번 정점에는 사과가 없다.

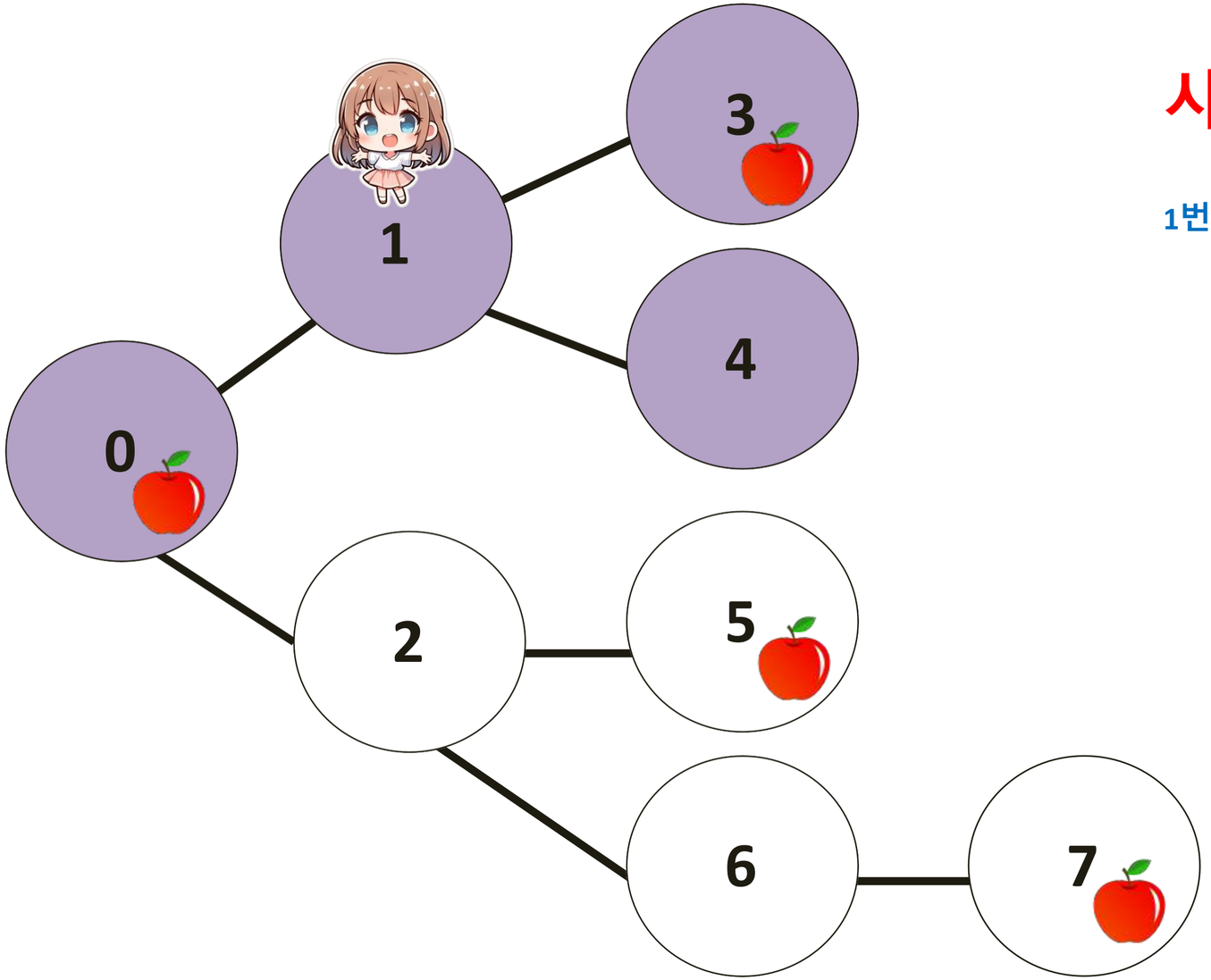
현재 정점으로부터 갈 수 있는 정점이 없으므로
되돌아 가준다.
(스택을 이용하여 지금까지 왔던 길을 되돌아가기)

STACK

1번 정점

0번 정점

깊이 우선 탐색(dfs)



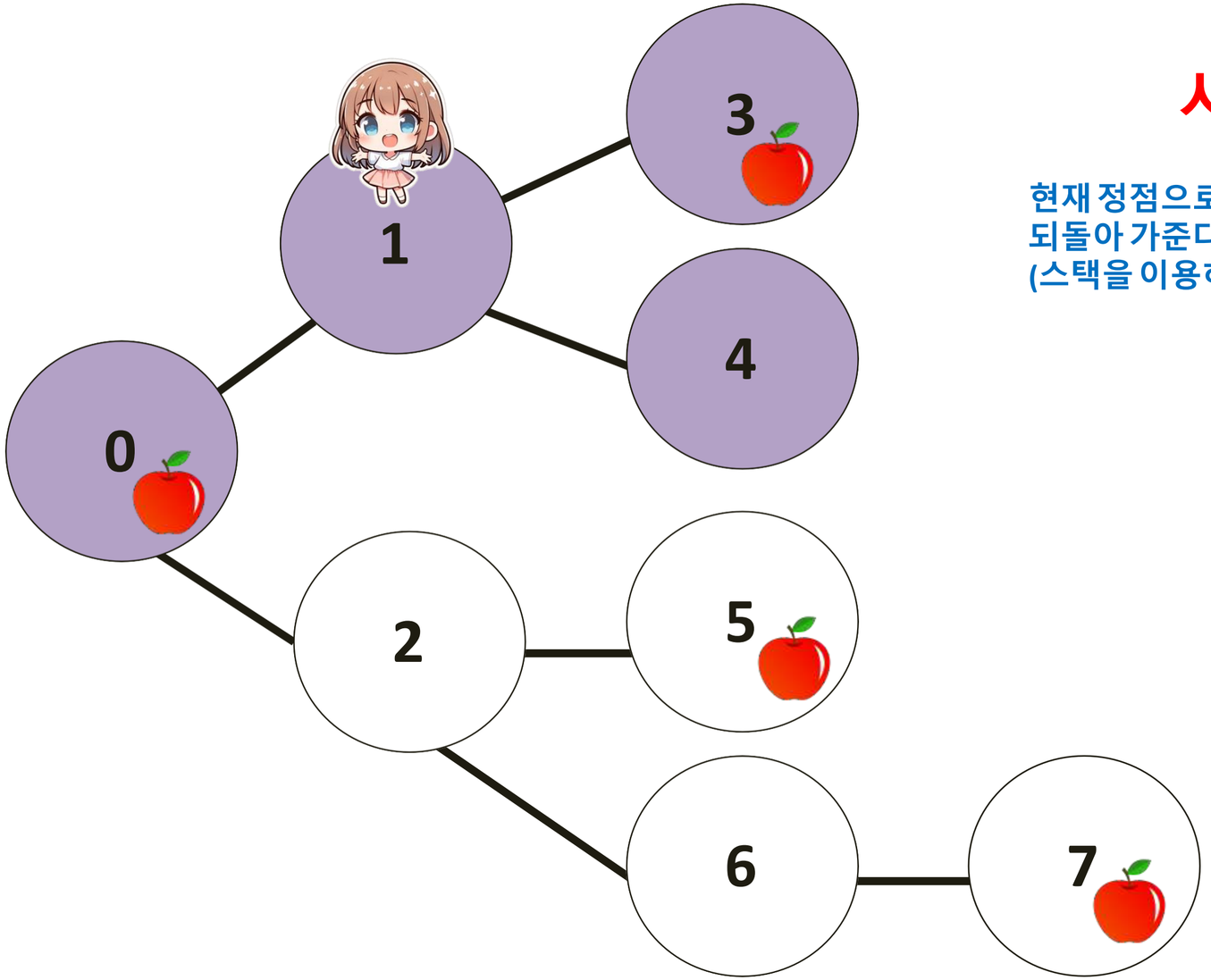
사과 합계 : 2

1번 정점으로 되돌아 왔다.

STACK

0번 정점

깊이 우선 탐색(dfs)



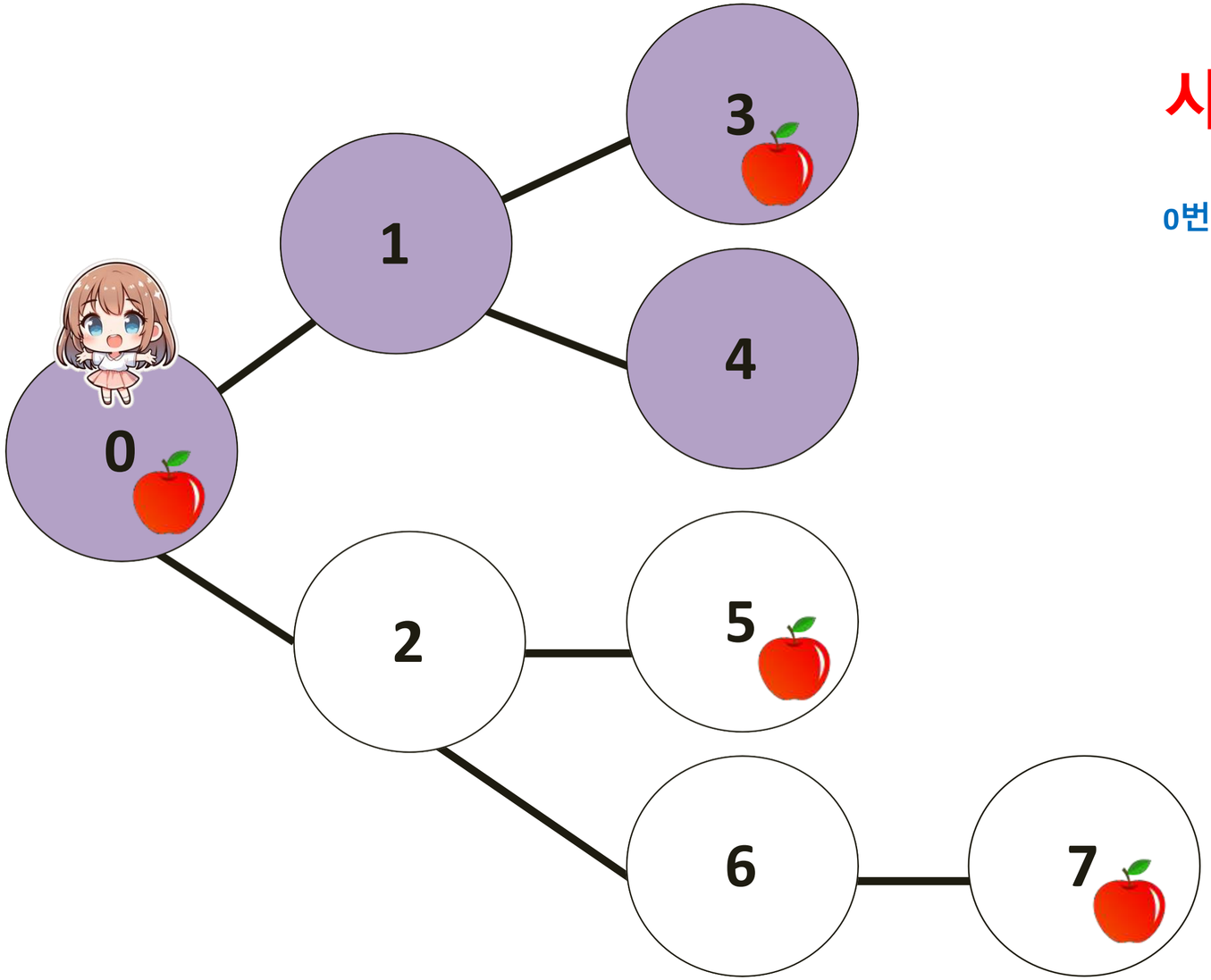
사과 합계 : 2

현재 정점으로부터 갈 수 있는 정점이 없으므로
되돌아 가준다.
(스택을 이용하여 지금까지 왔던 길을 되돌아가기)

STACK

0번 정점

깊이 우선 탐색(dfs)

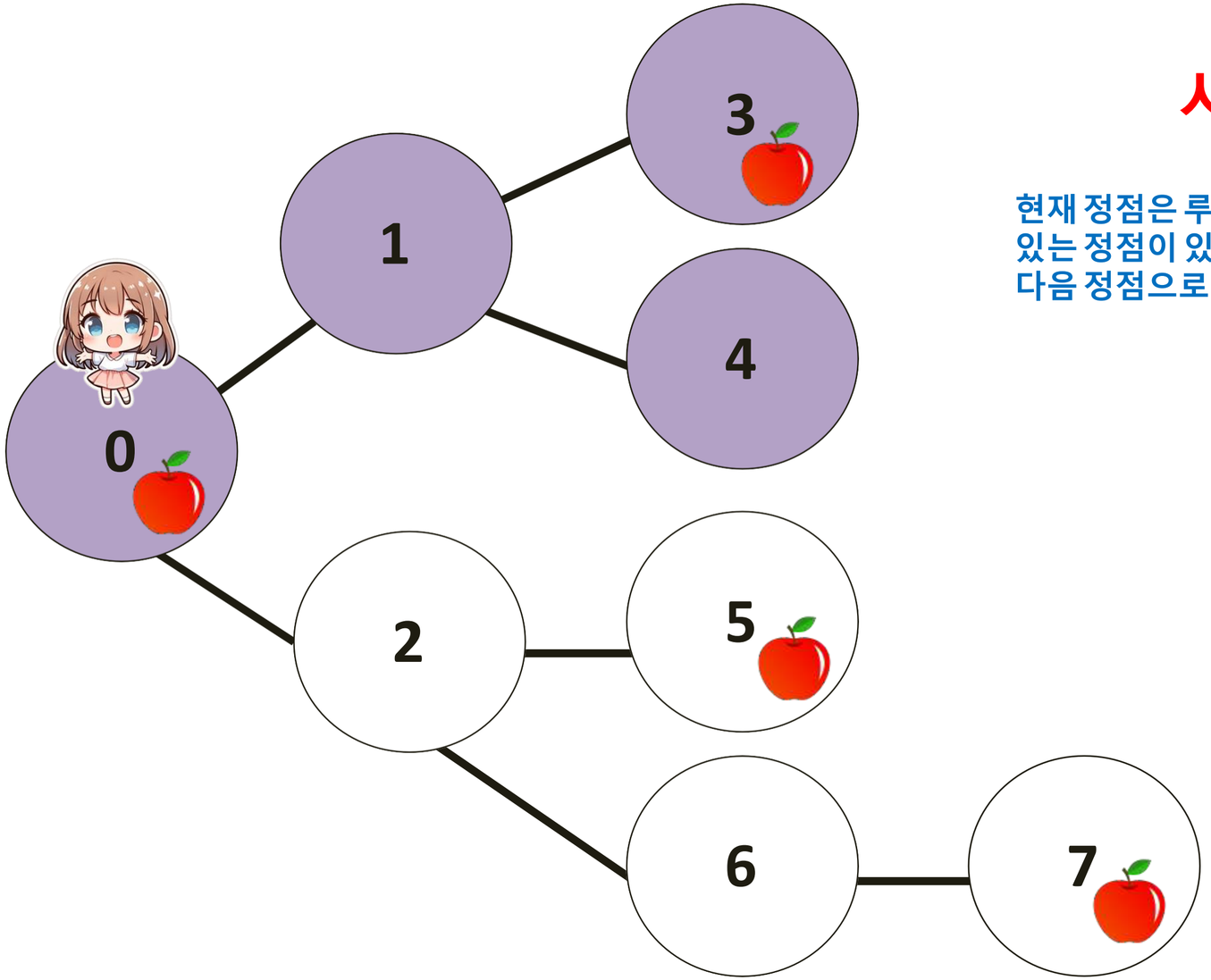


사과 합계 : 2

0번 정점으로 되돌아 왔다.

STACK

깊이 우선 탐색(dfs)



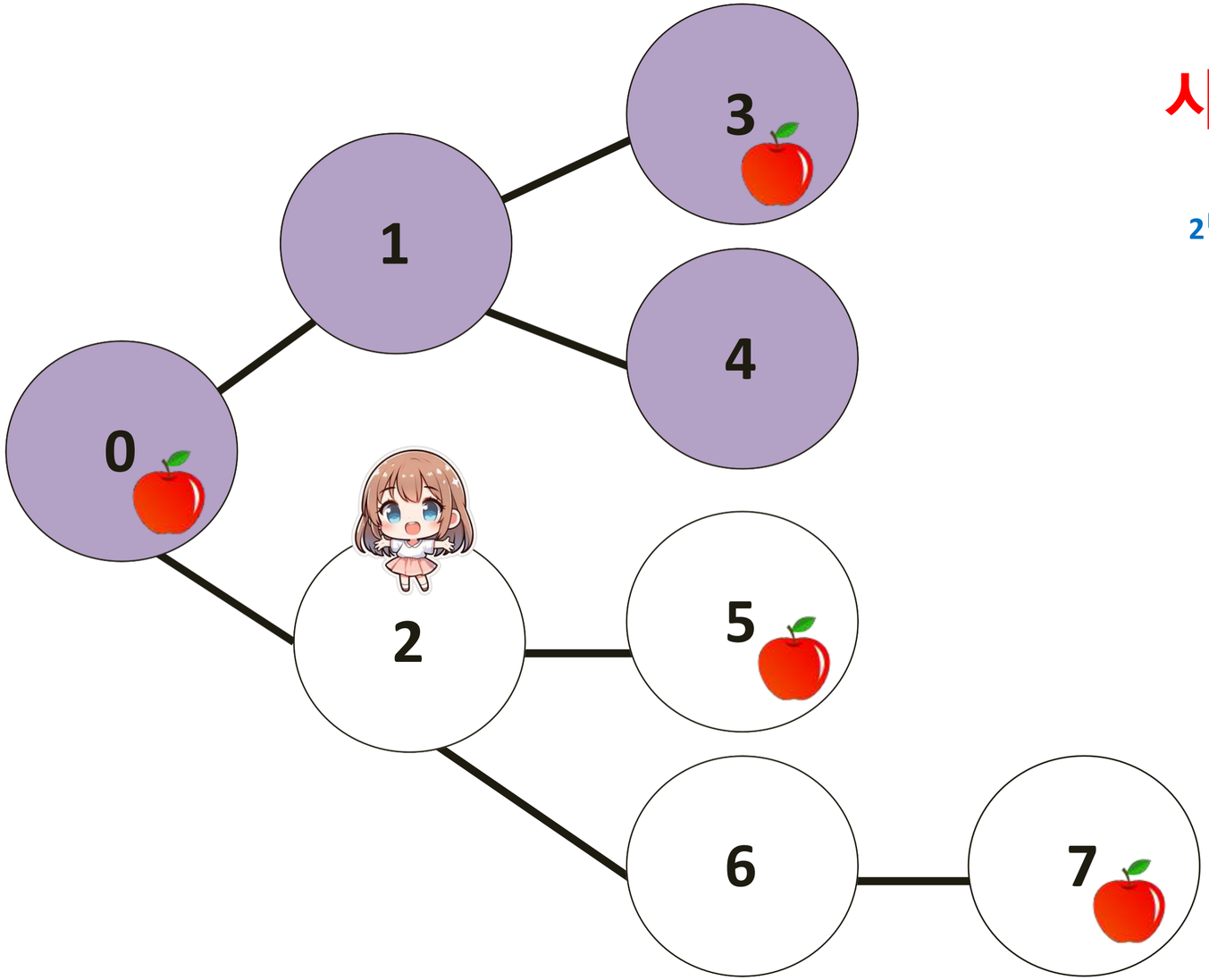
사과 합계 : 2

현재 정점은 루트로부터 '0' 떨어진 거리이며, 갈 수 있는 정점이 있으므로 지금 정점을 스택에 넣어주고 다음 정점으로 이동할 준비를 해준다.

STACK

0번 정점

깊이 우선 탐색(dfs)



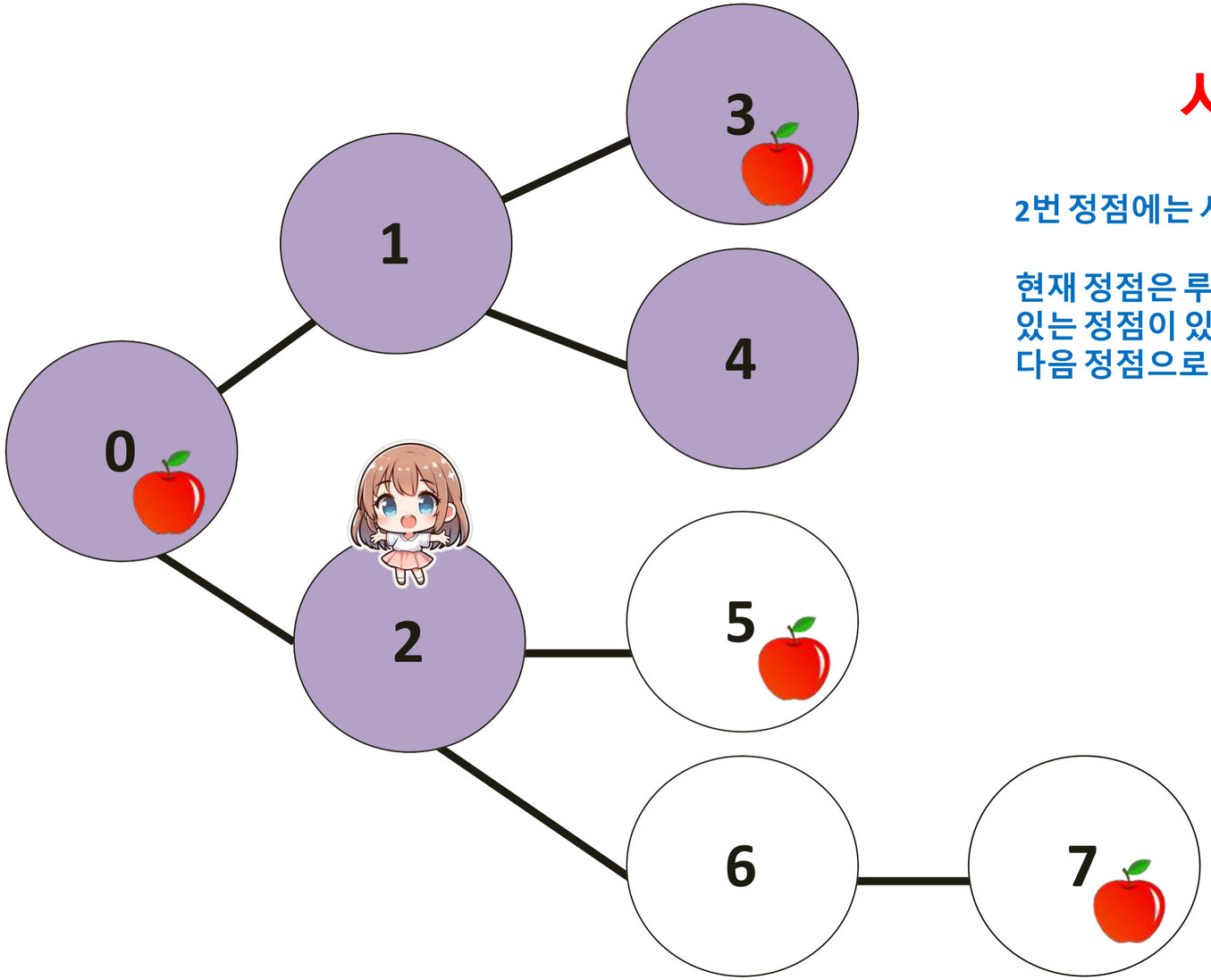
사과 합계 : 2

2번 정점에 도착했다.

STACK

0번 정점

깊이 우선 탐색(dfs)



사과 합계 : 2

2번 정점에는 사과가 없다.

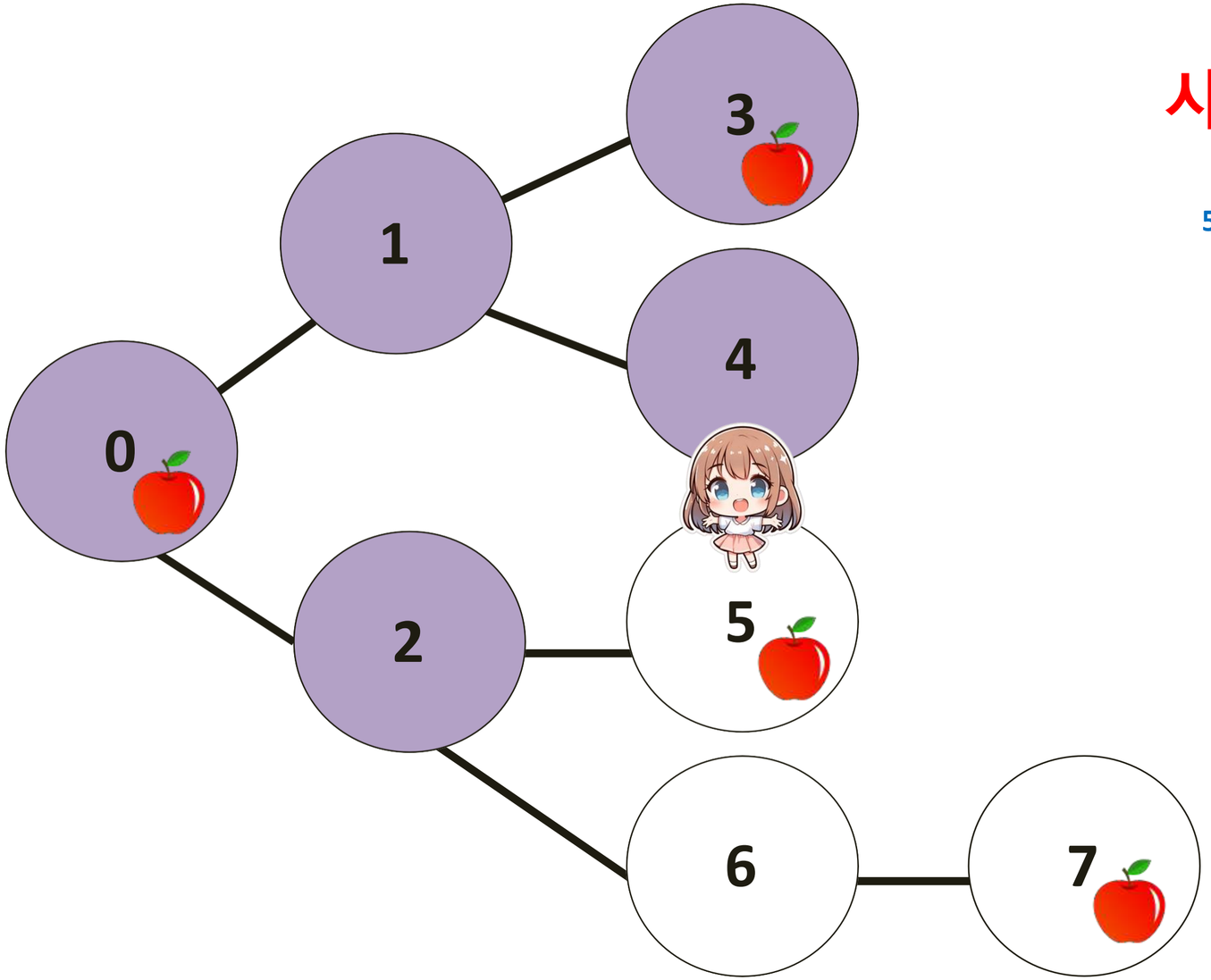
현재 정점은 루트로부터 '1' 떨어진 거리이며, 갈 수 있는 정점이 있으므로 지금 정점을 스택에 넣어주고 다음 정점으로 이동할 준비를 해준다.

STACK

2번 정점

0번 정점

깊이 우선 탐색(dfs)



사과 합계 : 2

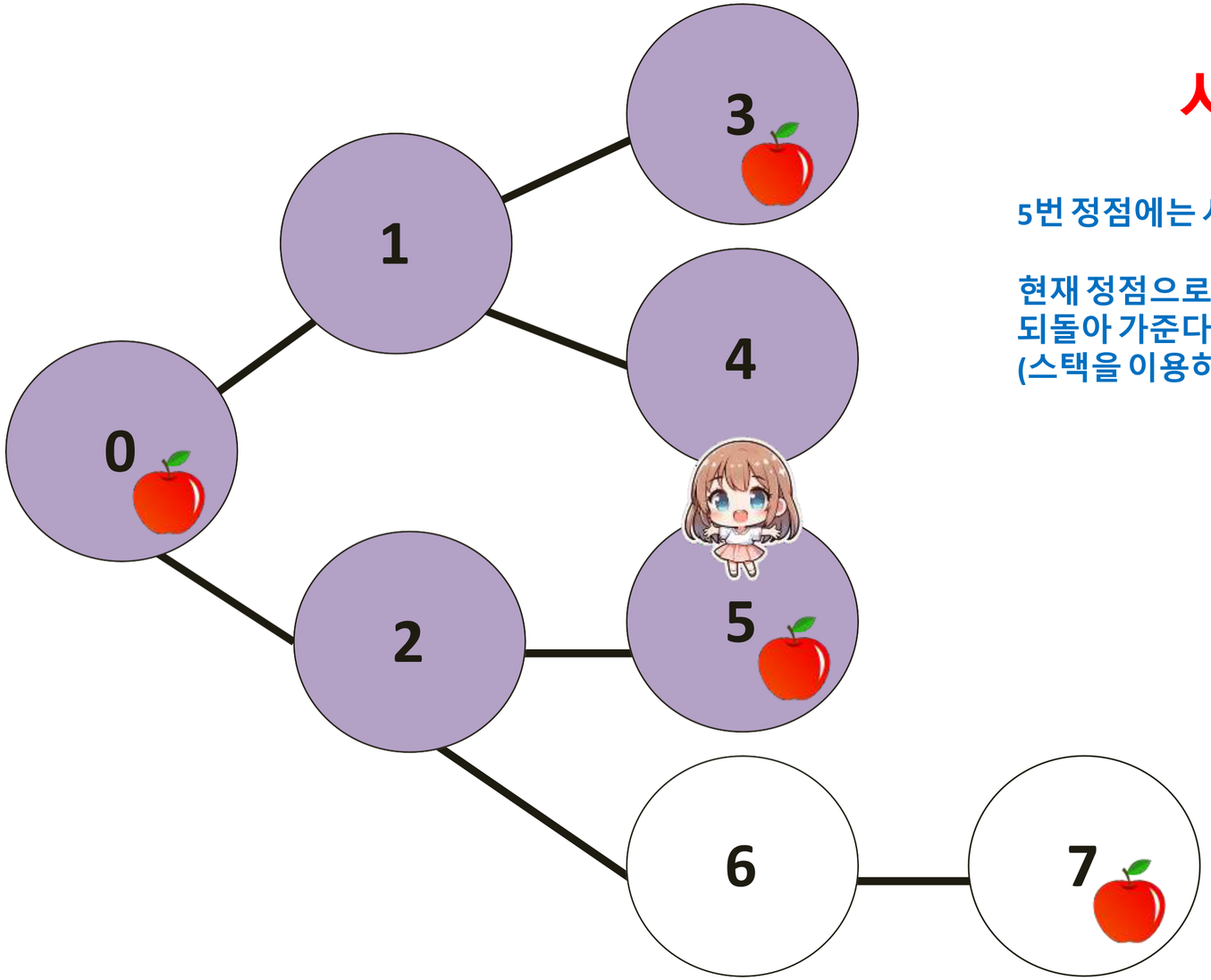
5번 정점에 도착했다.

STACK

2번 정점

0번 정점

깊이 우선 탐색(dfs)



사과 합계 : 3

5번 정점에는 사과가 있으니 수확해준다.

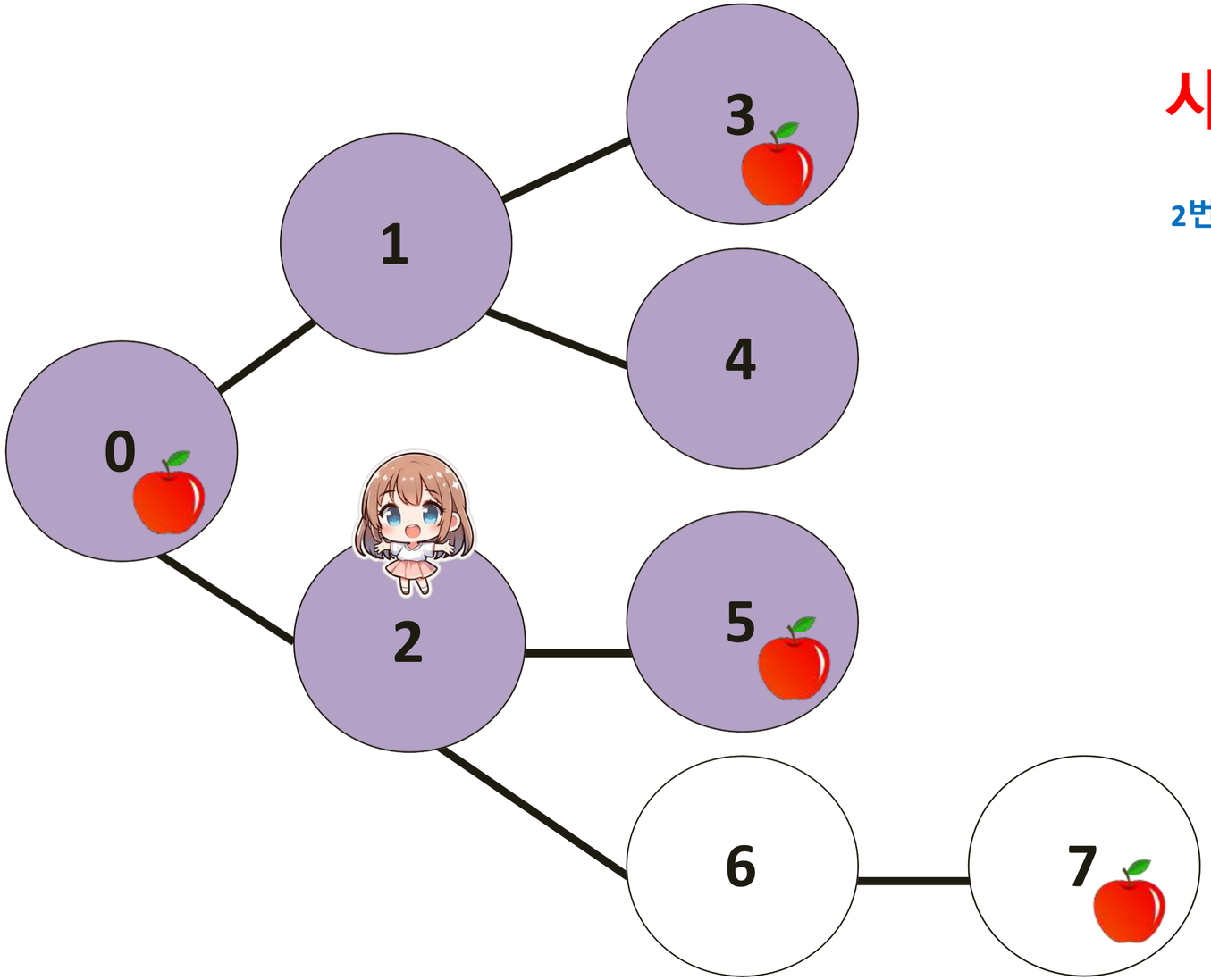
현재 정점으로부터 갈 수 있는 정점이 없으므로
되돌아 가준다.
(스택을 이용하여 지금까지 왔던 길을 되돌아가기)

STACK

2번 정점

0번 정점

깊이 우선 탐색(dfs)



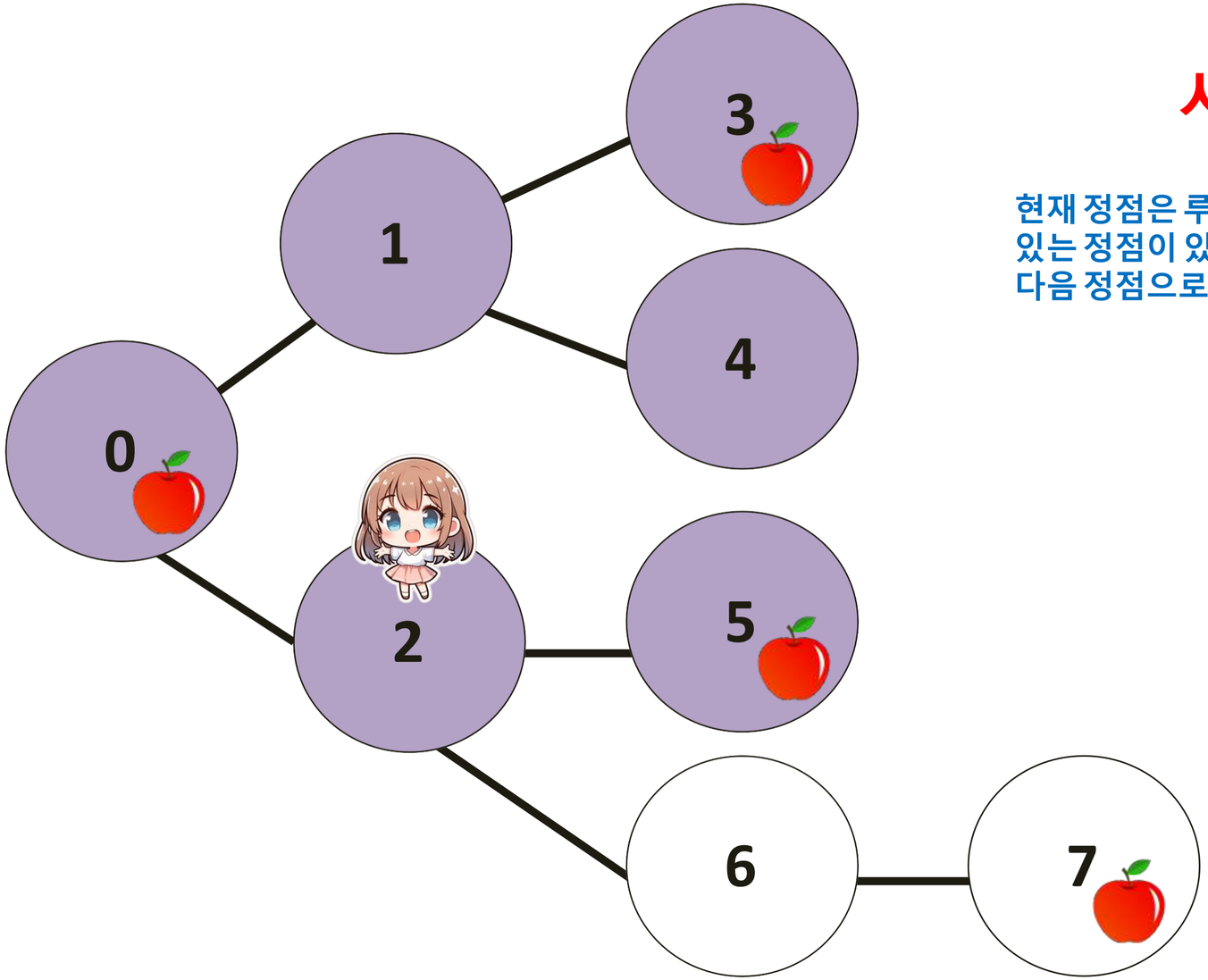
사과 합계 : 3

2번 정점으로 되돌아 왔다.

STACK

0번 정점

깊이 우선 탐색(dfs)



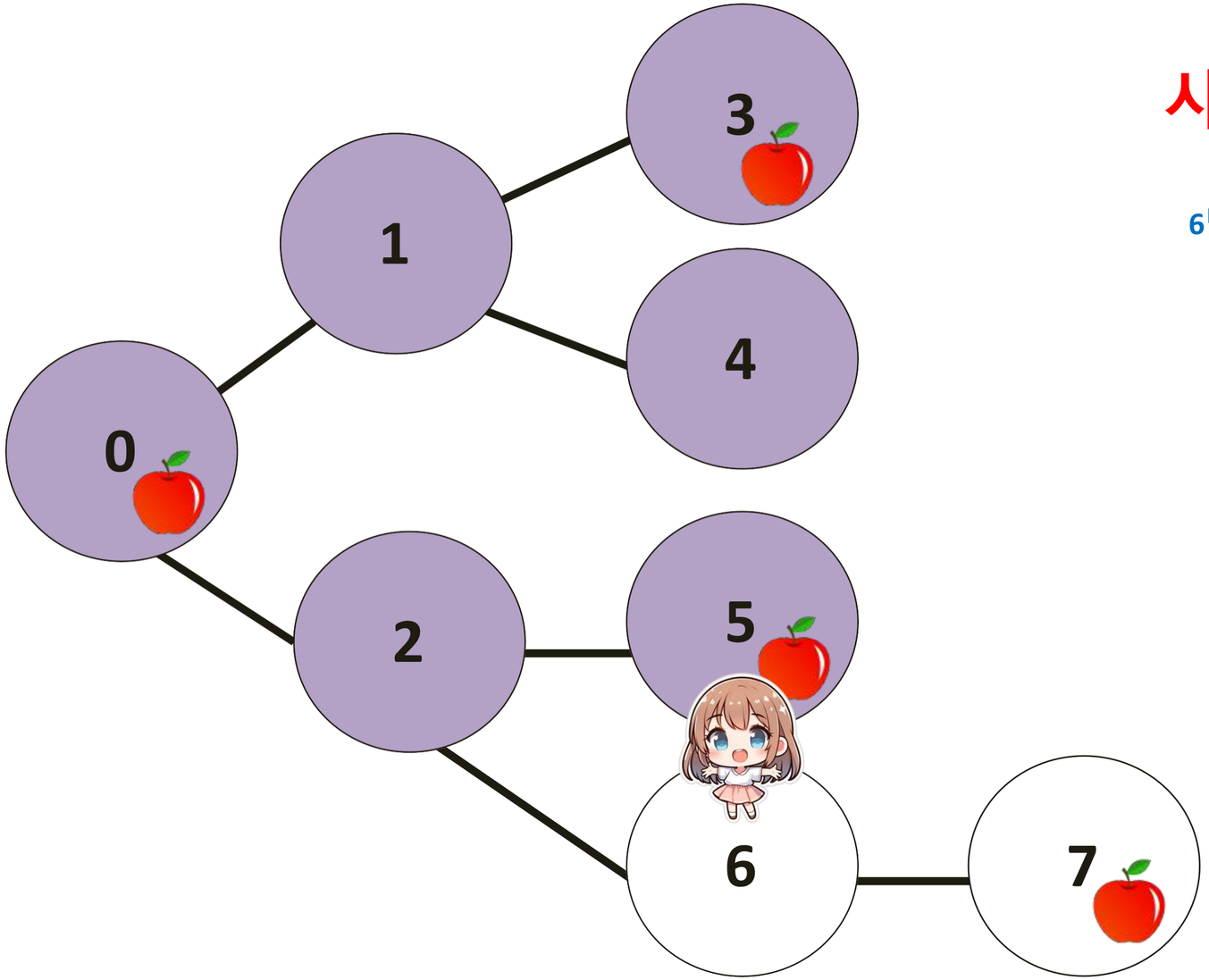
사과 합계 : 3

현재 정점은 루트로부터 '1' 떨어진 거리이며, 갈 수 있는 정점이 있으므로 지금 정점을 스택에 넣어주고 다음 정점으로 이동할 준비를 해준다.

STACK

0번 정점

깊이 우선 탐색(dfs)



사과 합계 : 3

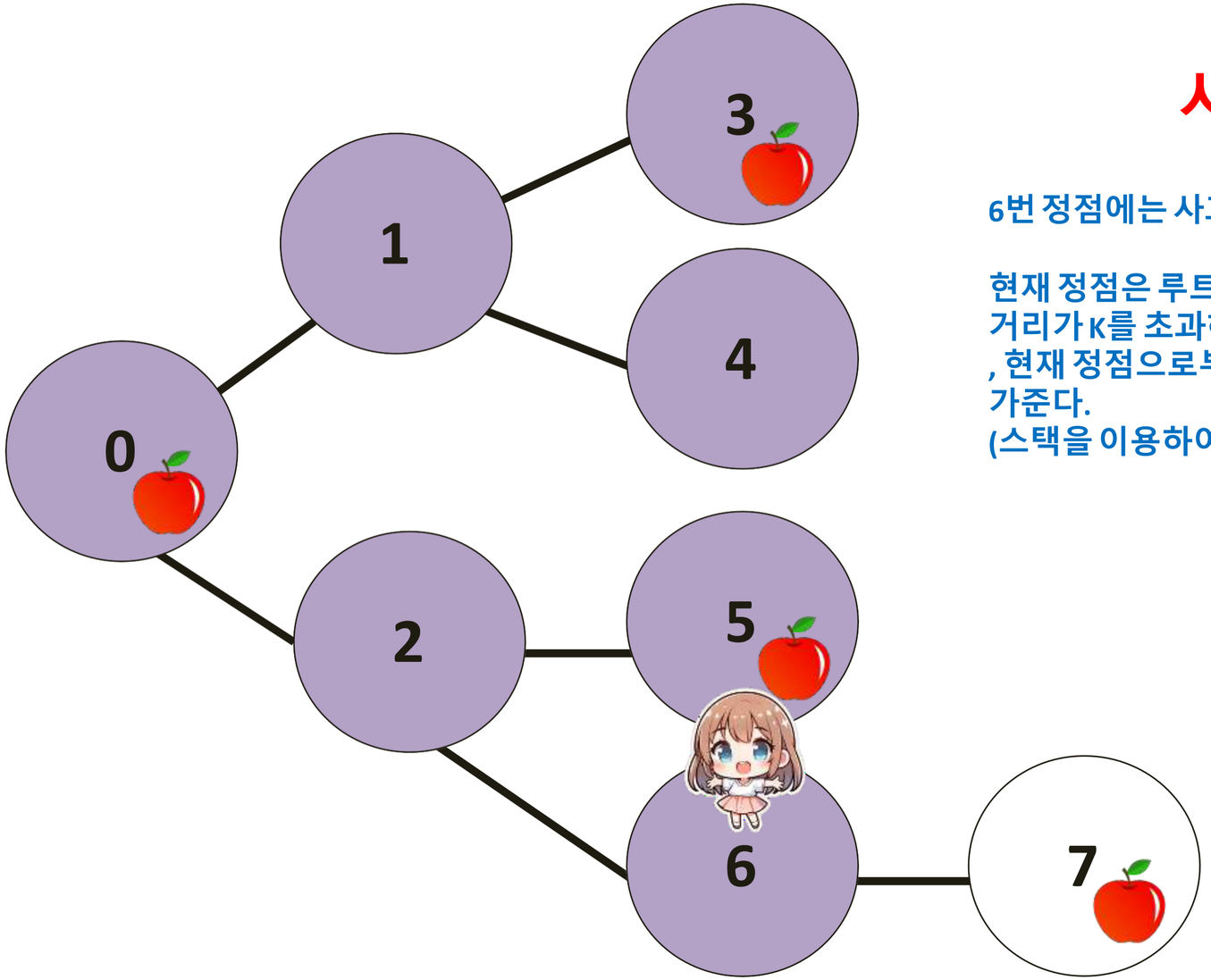
6번 정점에 도착했다.

STACK

2번 정점

0번 정점

깊이 우선 탐색(dfs)



사과 합계 : 3

6번 정점에는 사과가 없다.

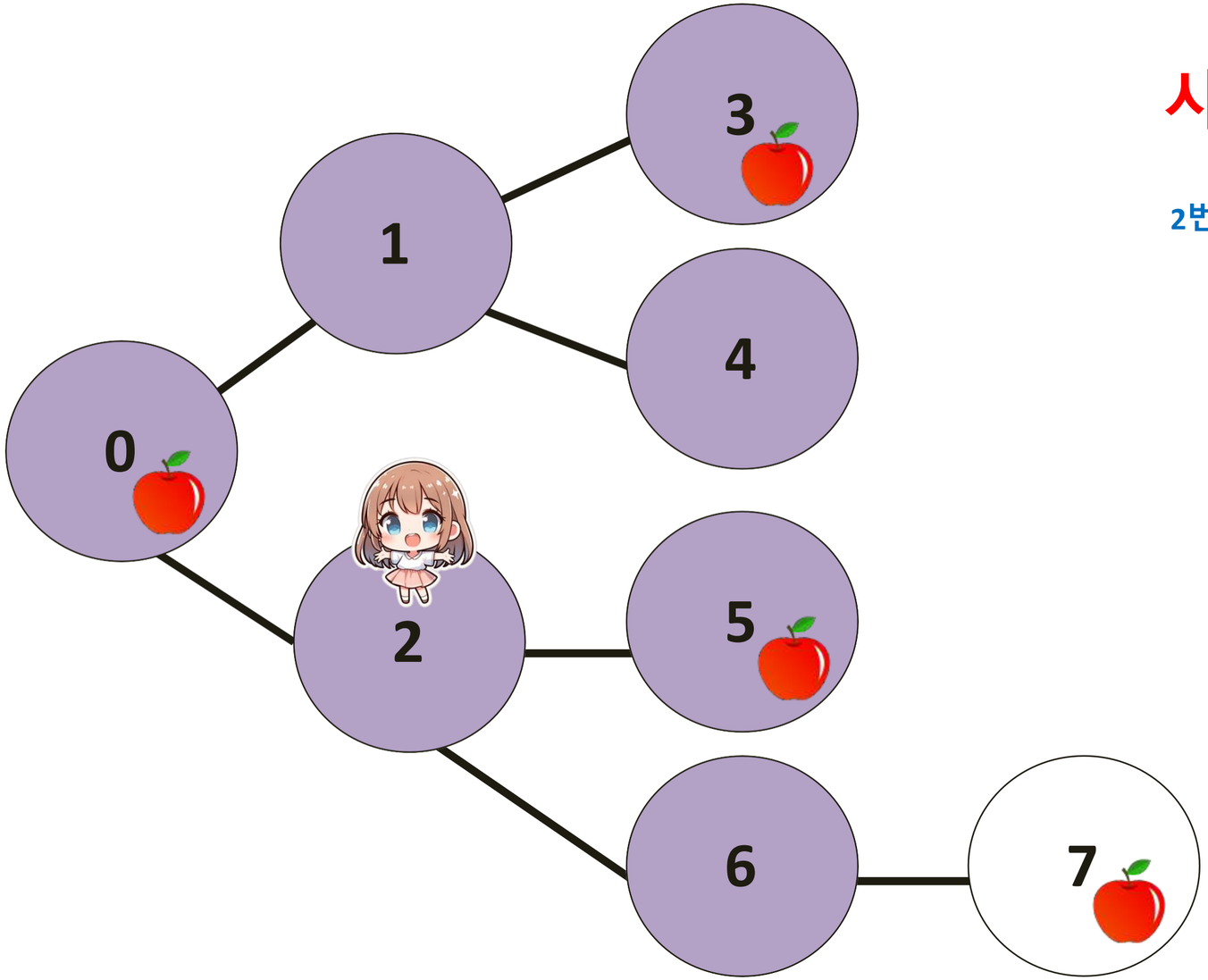
현재 정점은 루트로부터 '2' 떨어진 거리이기 때문에, 거리가 K를 초과하는 새로운 정점에 방문할 수 없다. 즉, 현재 정점으로부터 갈 수 있는 정점이 없기에 되돌아가준다.
(스택을 이용하여 지금까지 왔던 길을 되돌아가기)

STACK

2번 정점

0번 정점

깊이 우선 탐색(dfs)



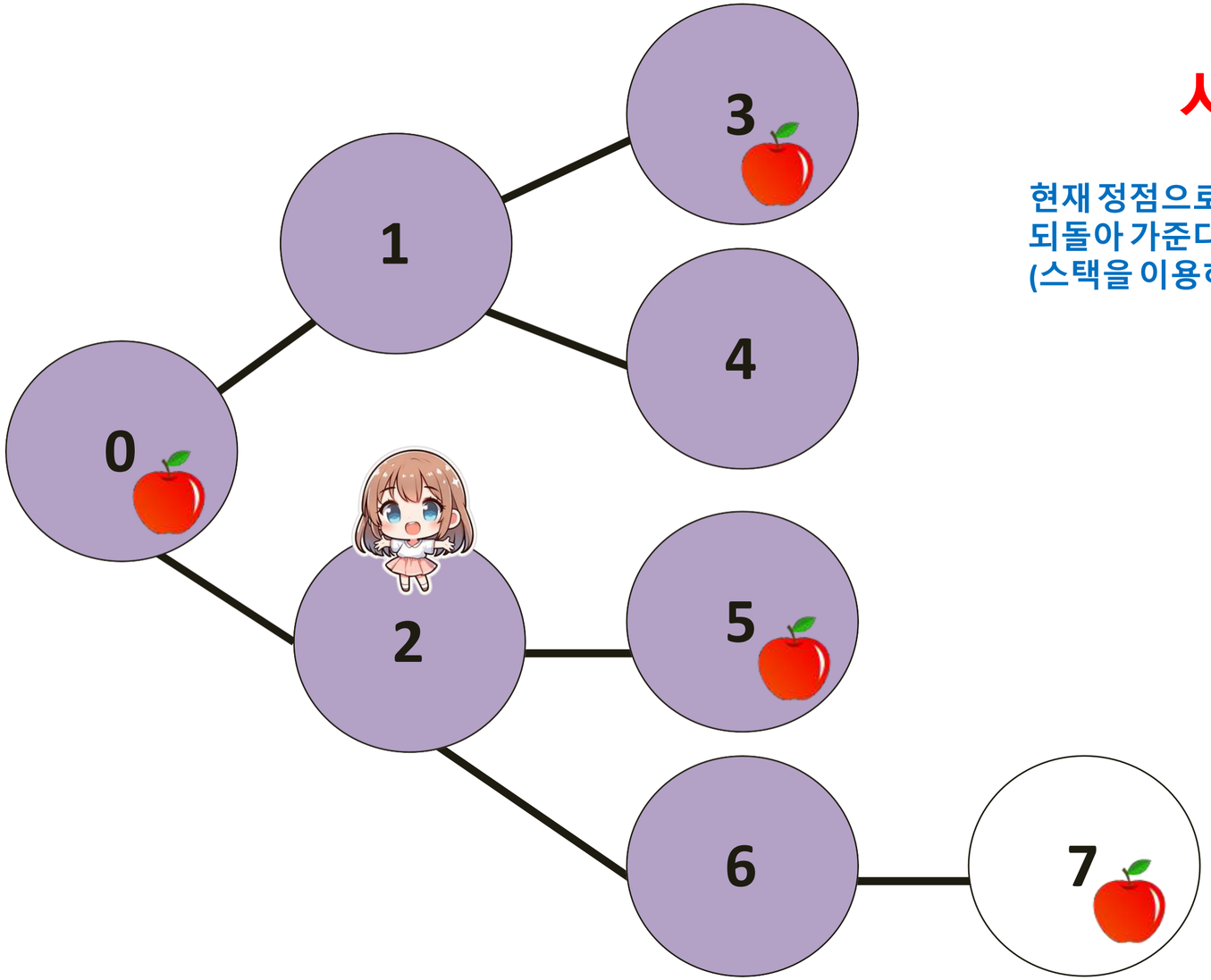
사과 합계 : 3

2번 정점으로 되돌아 왔다.

STACK

0번 정점

깊이 우선 탐색(dfs)



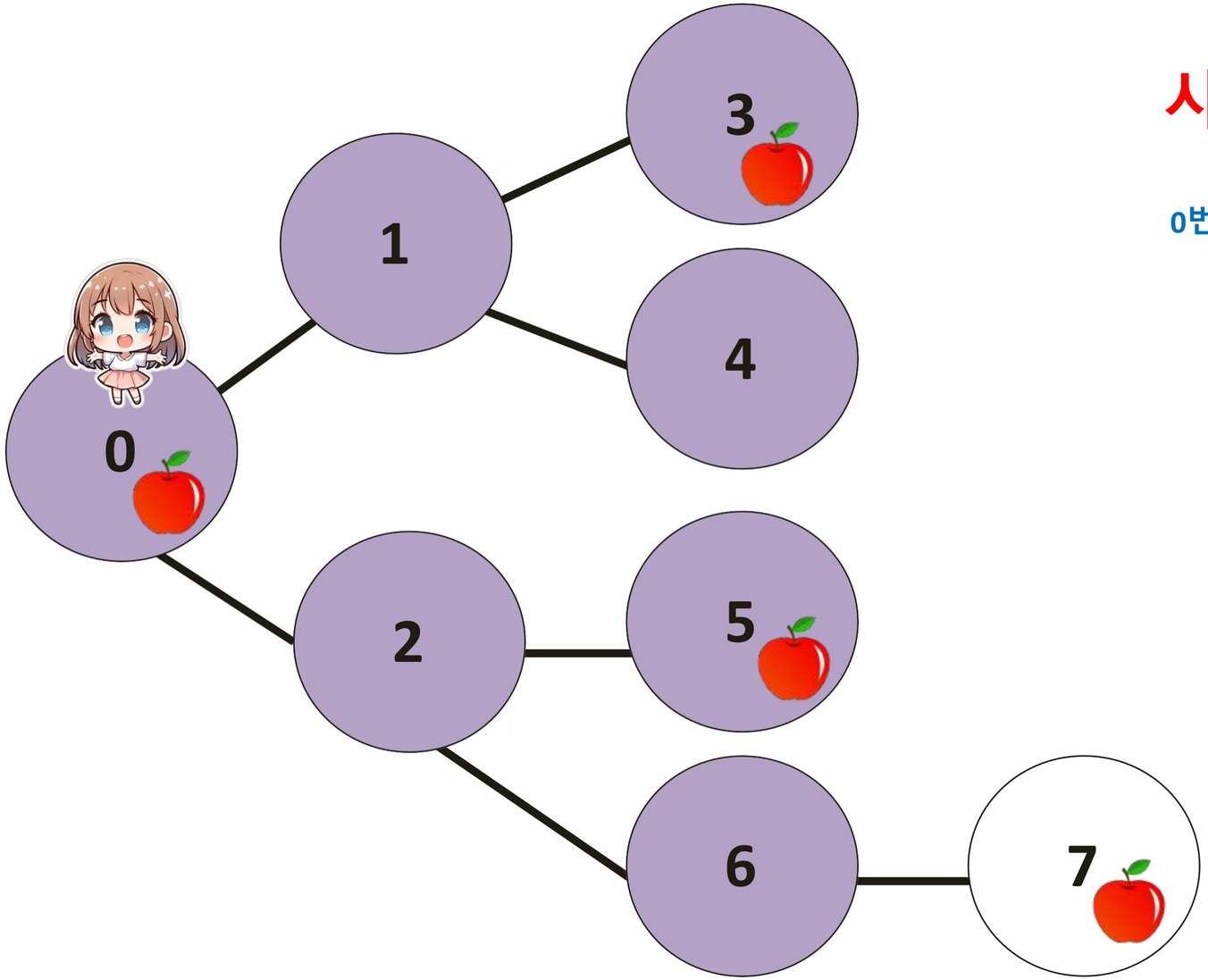
사과 합계 : 3

현재 정점으로부터 갈 수 있는 정점이 없으므로
되돌아 가준다.
(스택을 이용하여 지금까지 왔던 길을 되돌아가기)

STACK

0번 정점

깊이 우선 탐색(dfs)

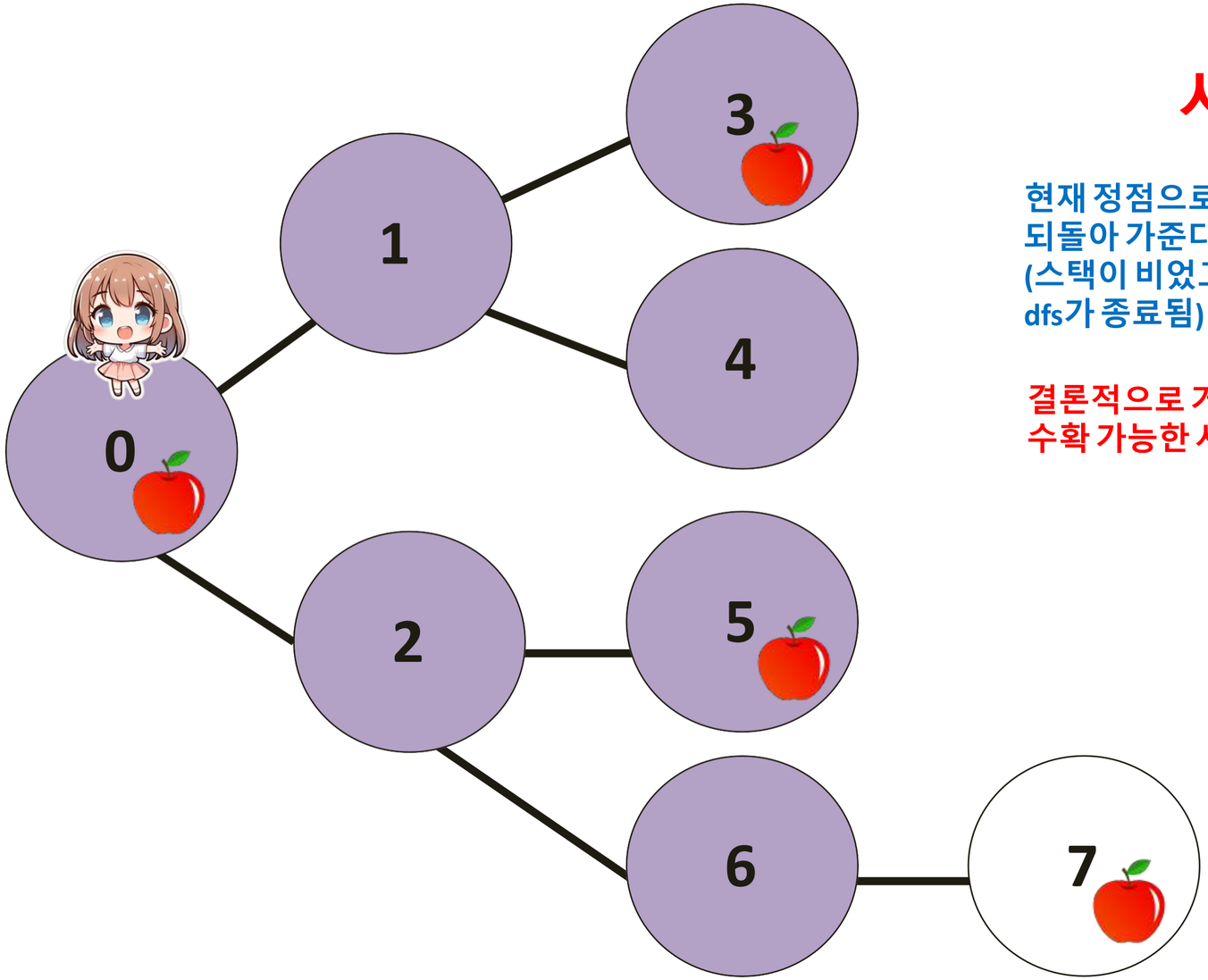


사과 합계 : 3

0번 정점으로 되돌아 왔다.

STACK

깊이 우선 탐색(dfs)



사과 합계 : 3

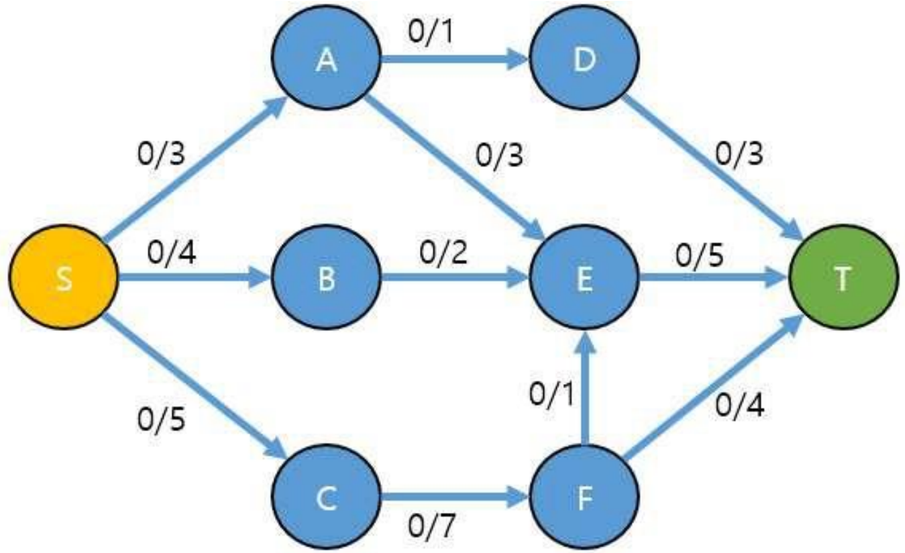
현재 정점으로부터 갈 수 있는 정점이 없으므로 되돌아 가준다.
(스택이 비었고 더 이상 갈 수 있는 정점이 없어, dfs가 종료됨)

결론적으로 거리가 '2' 이하인 트리 노드에서 수확 가능한 사과는 총 3개다.

STACK

너비 우선 탐색(bfs)이란?

BFS는 그래프를 탐색하는 방법 중 하나로, 현재 노드에서 가까운 노드부터 차례대로 탐색하는 방식이다. 큐(Queue)를 활용하여 구현하며, 탐색이 계층적으로 이루어지기 때문에 최단 경로 탐색 문제에서 자주 사용된다.
BFS의 핵심 개념은 모든 인접 노드를 먼저 방문한 후 다음 단계로 넘어가는 것이다. 방문 여부를 저장하여 중복 탐색을 방지하며, 최단 거리 계산, 네트워크 분석, 퍼즐 문제 해결 등에 널리 활용된다.



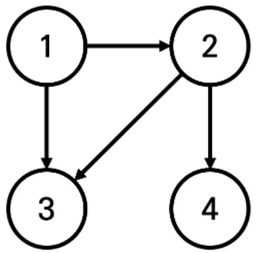
bfs를 사용하는 대표적 예제 : 최단 경로

너비 우선 탐색(bfs)

어떤 나라에는 1번부터 *N*번까지의 도시와 *M*개의 단방향 도로가 존재한다. 모든 도로의 거리는 1이다.

이 때 특정한 도시 *X*로부터 출발하여 도달할 수 있는 모든 도시 중에서, 최단 거리가 정확히 *K*인 모든 도시들의 번호를 출력하는 프로그램을 작성하시오. 또한 출발 도시 *X*에서 출발 도시 *X*로 가는 최단 거리는 항상 0이라고 가정한다.

예를 들어 *N*=4, *K*=2, *X*=1일 때 다음과 같이 그래프가 구성되어 있다고 가정하자.



이 때 1번 도시에서 출발하여 도달할 수 있는 도시 중에서, 최단 거리가 2인 도시는 4번 도시 뿐이다. 2번과 3번 도시의 경우, 최단 거리가 1이기 때문에 출력하지 않는다.

입력

첫째 줄에 도시의 개수 *N*, 도로의 개수 *M*, 거리 정보 *K*, 출발 도시의 번호 *X*가 주어진다. (2 ≤ *N* ≤ 300,000, 1 ≤ *M* ≤ 1,000,000, 1 ≤ *K* ≤ 300,000, 1 ≤ *X* ≤ *N*) 둘째 줄부터 *M*개의 줄에 걸쳐서 두 개의 자연수 *A*, *B*가 공백을 기준으로 구분되어 주어진다. 이는 *A*번 도시에서 *B*번 도시로 이동하는 단방향 도로가 존재한다는 의미다. (1 ≤ *A*, *B* ≤ *N*) 단, *A*와 *B*는 서로 다른 자연수이다.

출력

*X*로부터 출발하여 도달할 수 있는 도시 중에서, 최단 거리가 *K*인 모든 도시의 번호를 한 줄에 하나씩 오름차순으로 출력한다.

이 때 도달할 수 있는 도시 중에서, 최단 거리가 *K*인 도시가 하나도 존재하지 않으면 -1을 출력한다.

1번 부터 N번 까지의 도시가 있고 M개의 단방향 도로가 존재한다.
모든 도로의 거리가 1이라고 할 때 최단 거리가 K인 도시를 찾아야 한다.

1. **간선 정보를 입력 받은 뒤 그래프를 생성 한다.**

2. **너비 우선 탐색을 통해 최단 거리가 K인 도시를 찾아 준다.**

*** 깊이 우선 탐색을 통해 해결할 수 없는 이유***

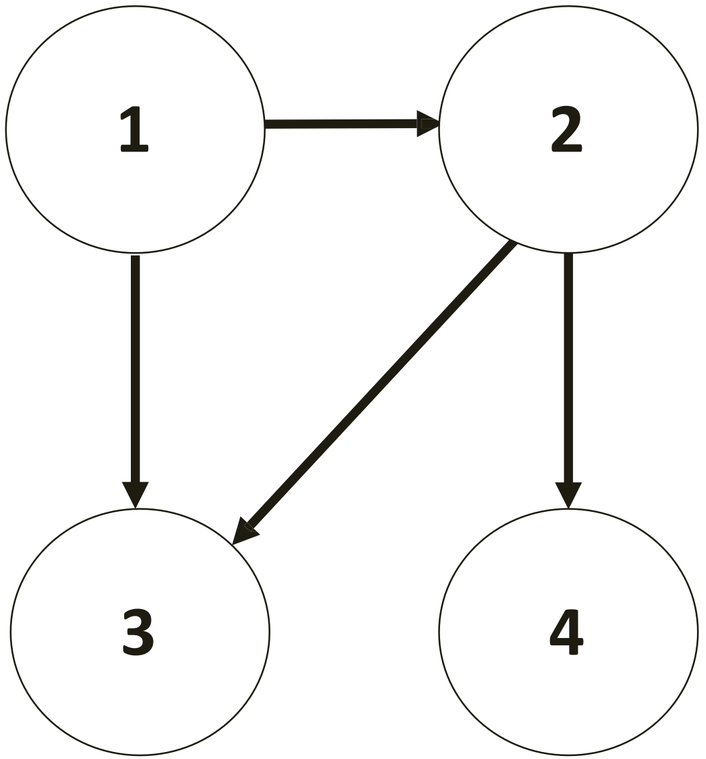
1번 정점으로 부터 3번 정점까지 가는 최단 경로는 1이다.
그러나 1번정점 -> 2번정점 -> 3번정점으로 가게되면 최단 경로가 2로 계산된다.

1번 정점으로 부터 2번 정점으로 가는거도 비슷한 논리이다.

너비 우선 탐색(bfs)

예제 입력 3 복사

```
4 4 1 1
1 2
1 3
2 3
2 4
```

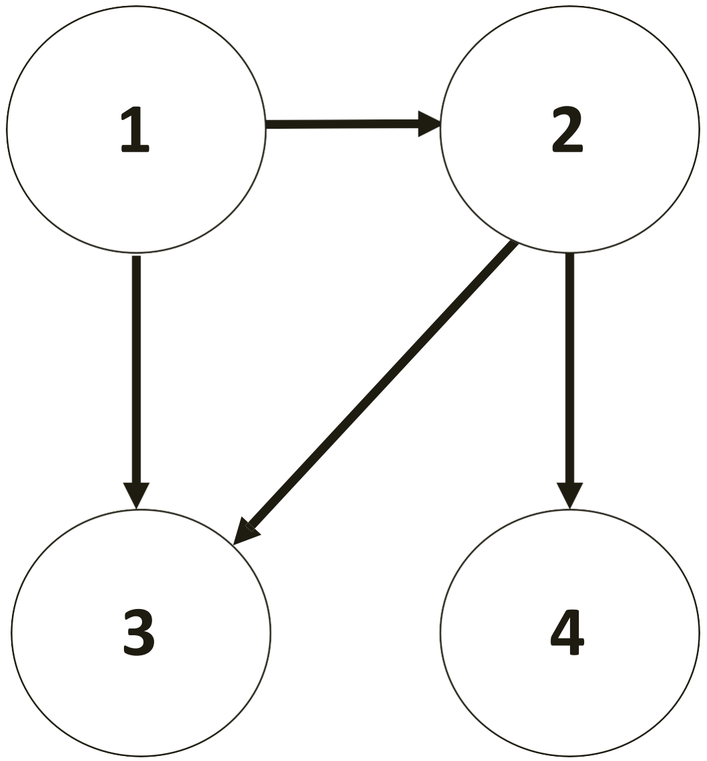


1 - 중복 방문이 일어나면 안되므로 이미 방문한 점은 보라색 표시를 함.

2 - 첫 방문은 루트 정점부터!

3 – Queue가 빌때까지 bfs

너비 우선 탐색(bfs)

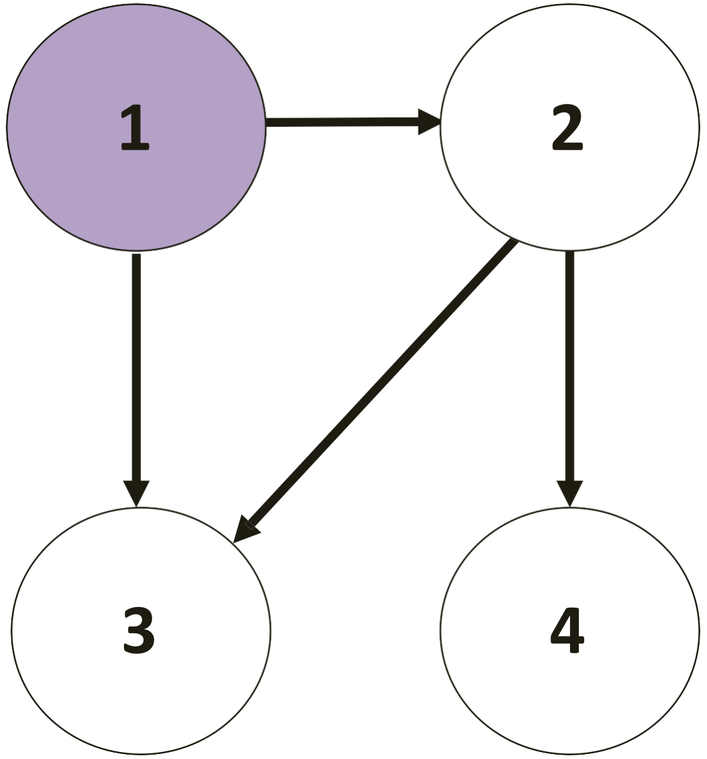


result : [ ]

아직 탐색을 시작하기 전이다.

Queue

너비 우선 탐색(bfs)



result : [ ]

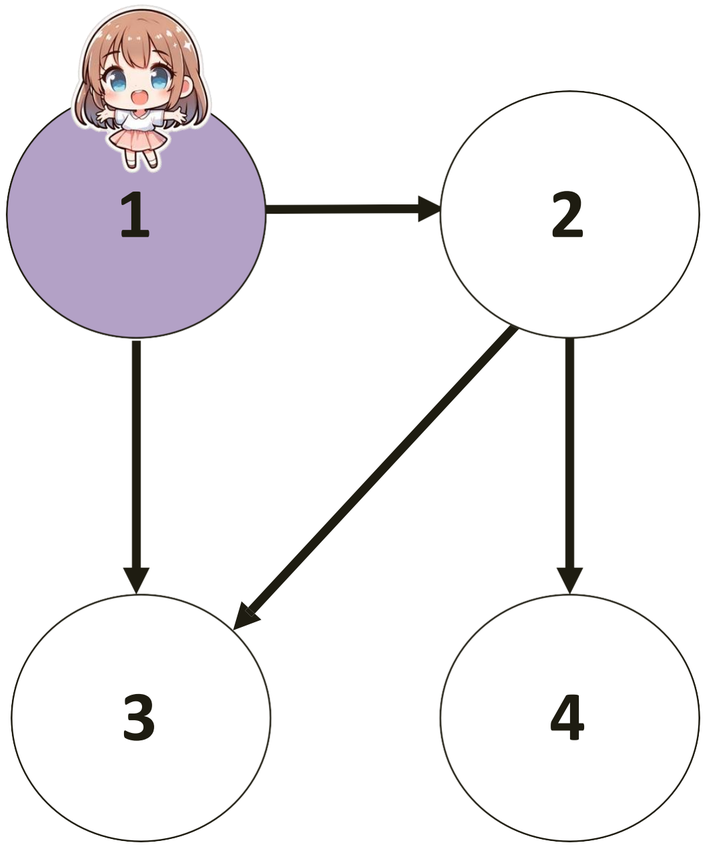
1번 도시에서 출발하라는 명령을 받았으므로

1번 도시를 큐에 넣어준 다음, 방문처리를 해준다.

Queue

1번 도시(거리 0)

너비 우선 탐색(bfs)



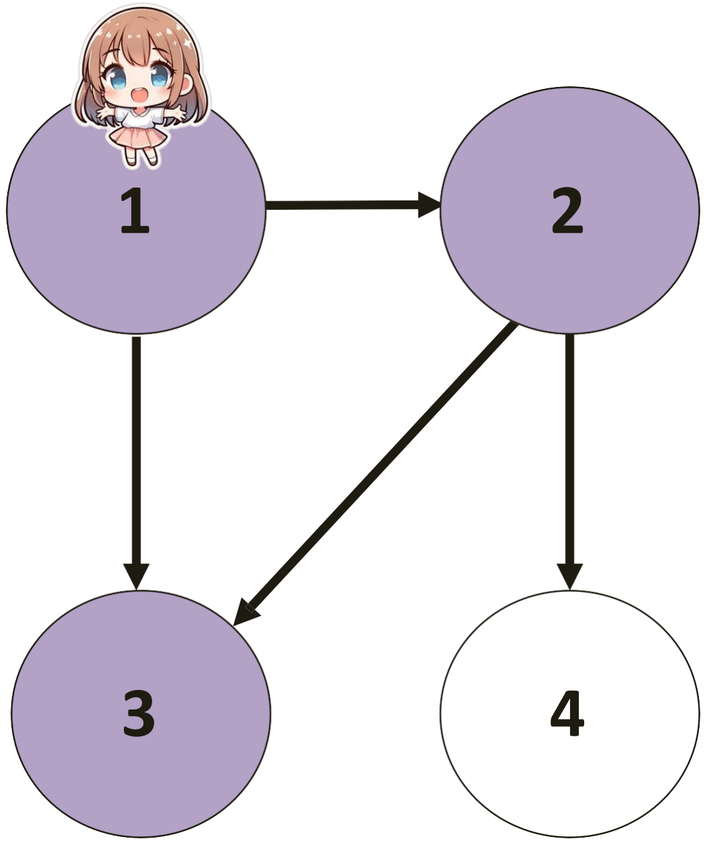
result : [ ]

큐에서 원소를 빼고 그 정점을 방문해준다.

큐에 들어있던 건 1번 도시였다. 거리는 0이다.

Queue

너비 우선 탐색(bfs)



result : [ ]

현재 위치인 1번 도시로 부터 갈 수 있는 도시를 모두 큐에 넣고 방문 처리를 해준다.

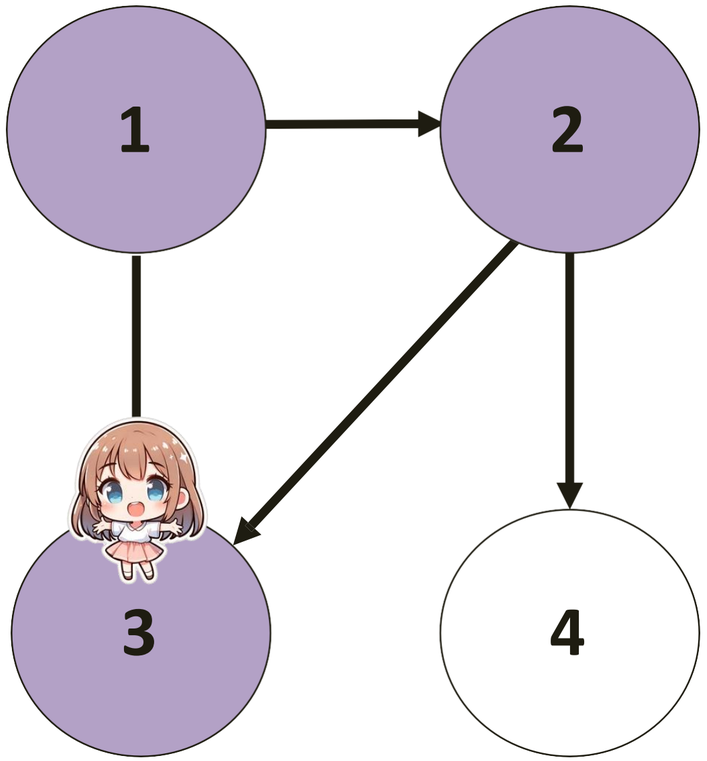
이때, 거리가 1만큼 떨어져 있음에 유의한다.

Queue

2번 도시(거리 1)

3번 도시(거리 1)

너비 우선 탐색(bfs)



result : [3]

큐에서 원소를 빼고 그 정점을 방문해준다.

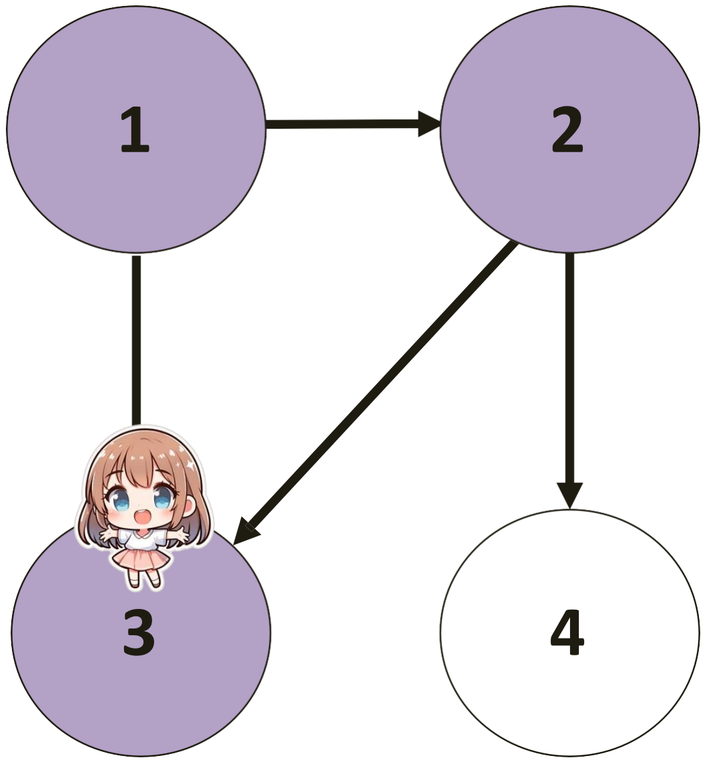
큐에 들어있던 건 3번 도시였다. 거리는 1이다.

우리가 찾던 최단 거리가 1인 도시 이므로 결과에 추가해준다.

Queue

2번 도시(거리 1)

너비 우선 탐색(bfs)



result : [3]

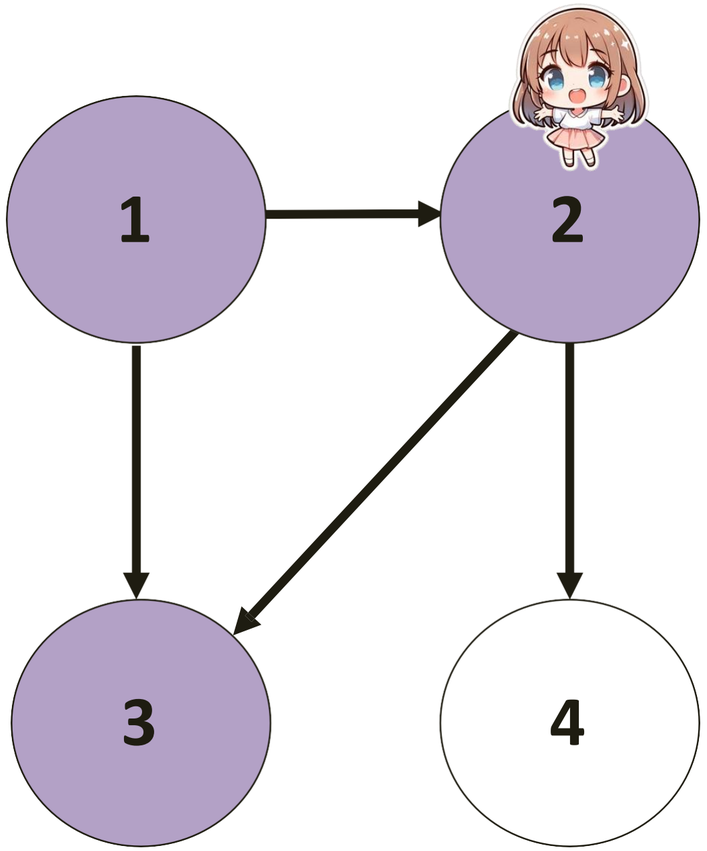
현재 3번 도시에서 갈 수 있는 도시가 없으므로

아무것도 추가하지 않는다.

Queue

2번 도시(거리 1)

너비 우선 탐색(bfs)



result : [3, 2]

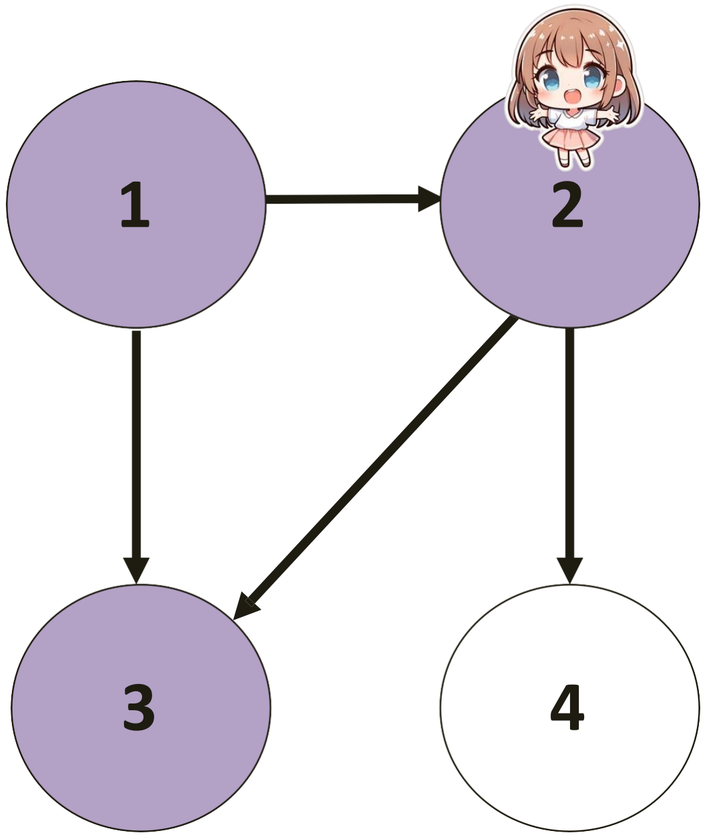
큐에서 원소를 빼고 그 정점을 방문해준다.

큐에 들어있던 건 2번 도시였다. 거리는 1이다.

우리가 찾던 최단 거리가 1인 도시 이므로 결과에 추가해준다.

Queue

너비 우선 탐색(bfs)



result : [3, 2]

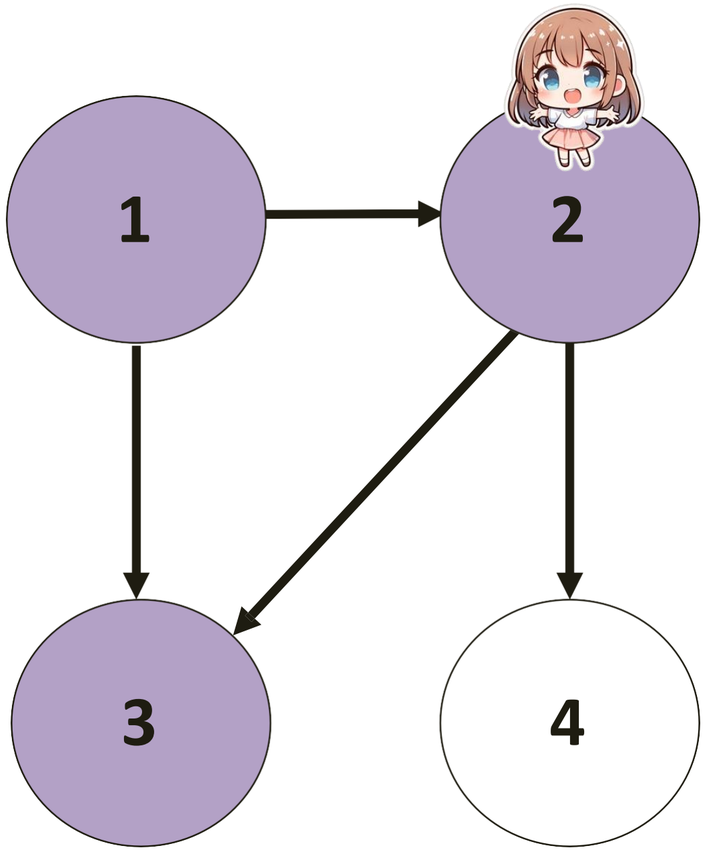
현재 2번 도시에서 갈 수 있는 도시가 있지만

현재 거리로부터 더 멀어지면 답과 상관이

없어지므로 큐에 추가 하지 않는다.

Queue

너비 우선 탐색(bfs)



result : [3, 2]

큐가 비었으므로 bfs가 종료된다.

결론적으로 1번도시로 부터 최단거리가 '1'인 도시는
2번 도시, 3번 도시이다.

Queue