4주차 알고리즘 세미나

PoolC 2021 1학기

지난시간 도전문제 – 정수 삼각형

문제 요약 :

- $1 \le N \le 500$,
- 삼각형을 이루고 있는 각 수는 모두 정수이며, 0이상 9999 이하이다.
- 맨 위층부터 시작해서 아래에 있는 수 하나씩 선택하여 내려올 때, 선택한 수의 합이 최대가 되는 경로
- 수를 선택할 때는 대각선의 왼쪽 또는 오른쪽

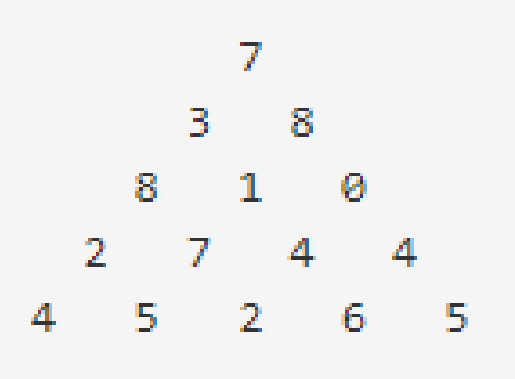
고려해야 할 사항 :

- 완전탐색 시간복잡도?
- 계산에 중복되는 부분이 있을까?
- 어떤 칸에서부터 시작해서 얻을 수 있는 최대값은 항상 고정되어 있다.

지난시간 도전문제 – 정수 삼각형

- 완전탐색 시간복잡도

  - 각 숫자마다 오른쪽, 왼쪽 두개의 선택지에 대해 백트래킹

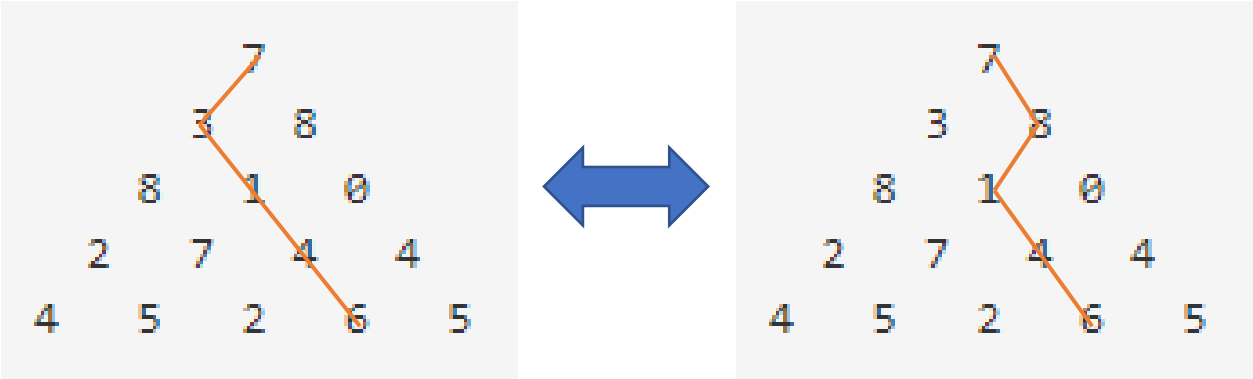


| 각 숫자마다 2개의 선택지 존재
| 각 숫자마다 2개의 선택지 존재
| 각 숫자마다 2개의 선택지 존재
| 각 숫자마다 2개의 선택지 존재

O(2^N)

지난시간 도전문제 – 정수 삼각형

- 계산에서 중복되는 부분이 있을까?



1 - 4 - 6 으로 이어지는 부분이 중복된다.

지난시간 도전문제 – 정수 삼각형

- 어떤 칸에서 시작해서 얻을 수 있는 최대값은 항상 고정되어있다.

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

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

- **따라서 위의 경로와는 상관없이 현재 칸을 기준으로 생각해도 된다.**

- DP 적용을 위한 함수 설정
  - F(int y, int x) : arr[y][x]에서 **시작하는** 경로의 최대값 반환
  - O(N^2)

- 우리가 구해야 하는 정답 : F(0,0) = arr[0][0] +max(F(1,0), F(1,1))

지난시간 도전문제 – 정수 삼각형

- Top-down

```
int F(int y, int x)
    종료 조건(y == N)
        return 0
    if(cache[y][x] != -1)
        return cache[y][x]
    return cache[y][x] = arr[y][x] +
    max( F(y+1,x), F(y+1,x+1) )
```

- Bottom-up

```
for(int i=1; i<=N; i++)
    for(int j=1; j<=i; j++)
        cache[i][j] = arr[i][j] +
        max(cache[i-1][j-1], cache[i-1][j])
        MAX = max(MAX, cache[i][j])
```

지난시간 도전문제 – 정수 삼각형

- Top-down

  - cache[y][x] : arr[y][x] **부터**

    밑으로 내려갈때 얻을 수 있는 최대 값

  - 재귀

  - 계산을 좀 미룬다는 느낌

- Bottom-up

  - cache[y][x] : arr[y][x] **까지**

    내려왔을 때 얻을 수 있는 최대 값

  - 반복문

  - 작은것부터 차근차근 하는 느낌

지난시간 도전문제 – 1로 만들기2

문제 요약 :

- 1 ≤ N ≤ 10^6
- 1로 만들기 문제의 업그레이드 버전.
- 최소 연산으로 1로 만드는 방법의 과정을 출력한다.
- 단, 여러 가지인 경우에는 아무거나 출력한다

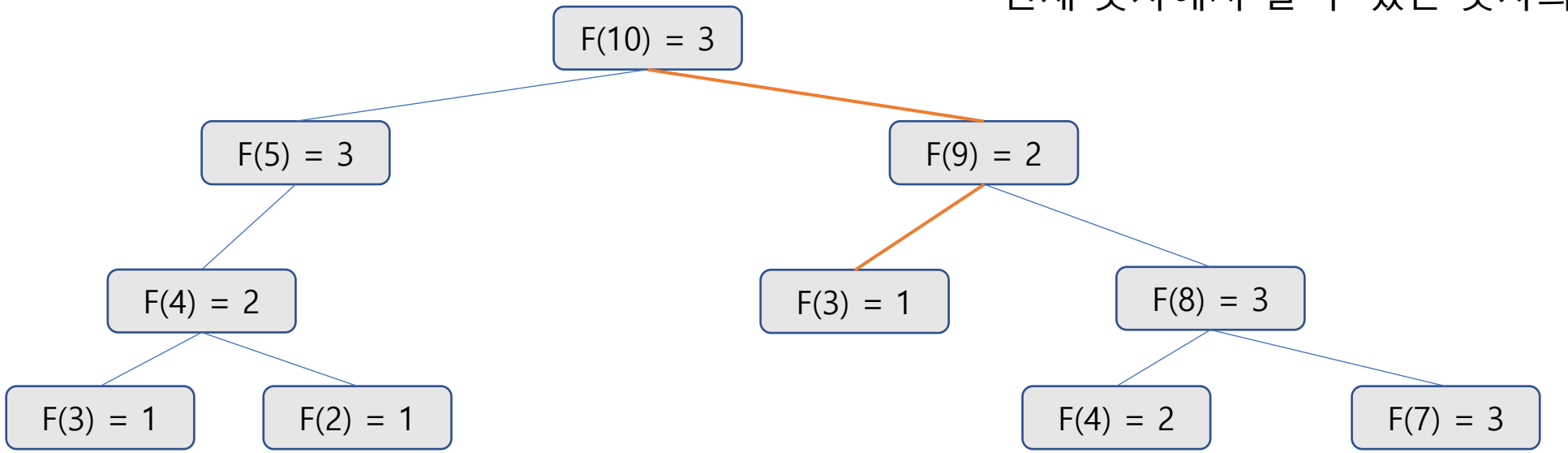
고려해야 할 사항 :

- 답은 이미 구했는데, 그 과정을 어떻게 구할 수 있을까?

지난시간 도전문제 – 1로 만들기2

- 방법 과정 구하기
  - 일반적으로 DP문제에서 방법, 경로를 물어볼 때, 역추적이라는 기법을 사용한다.

현재 숫자에서 갈 수 있는 숫자의 DP값만 비교



지난시간 도전문제 – 1로 만들기2

- 방법 과정 구하기
  - 일반적으로 DP문제에서 방법, 경로를 물어볼 때, 역추적이라는 기법을 사용한다.

```
vector<int> list
void trace(int x)
    if(x%3==0)
        MIN = (MIN, cache[x/3])
    if(x%2==0)
        MIN = (MIN, cache[x/2])
        MIN = (MIN, cache[x-1])
    list.push_back(MIN)
    trace(MIN)
```

list에는 경로상의 숫자가 담긴다

DP값끼리 비교하여 최적의 경로가 어디인지 추적

지난시간 도전문제 – 1로 만들기2

- 방법 과정 구하기
  - Bottom-up
    - 앞 슬라이드처럼 할 수도 있지만

      바로 경로를 저장할 수 있다.

    - before에 대해 재귀연산 하면

      전체 경로가 나오게 됨

```
cache[1] = 0
before[1] = -1
for(int i=2; i<=N; i++)
    cache[i] = cache[i-1]+1
    before[i] = i-1
    if(i%3 == 0)
        cache[i] = min(cache[i], cache[i/3] + 1)
        before[i] = i/3
    if(i%2 == 0)
        cache[i] = min(cache[i], cache[i/2] + 1)
        before[i] = i/2
```

지난시간 도전문제 – 가장 긴 증가하는 부분 수열

문제 요약 :

- $1 \le N \le 1,000, 1 \le A_i \le 1,000$
- 부분 수열 : 수열이 주어졌을때, 수열안에 있는 임의의 수들로 만든 새로운 수열
- 여러 증가 수열 중에 가장 긴 증가하는 부분 수열의 길이 출력
- DP로 해결하는 유명한 LIS(Longest Increasing Subsequence) 문제

고려해야 할 사항 :

- 완전탐색의 시간복잡도?
- 계산의 중복이 있는가?
- 앞에 문제와 마찬가지로 수열의 시작점이 정해지면 최대 길이는 항상 고정된 값을 갖는다.

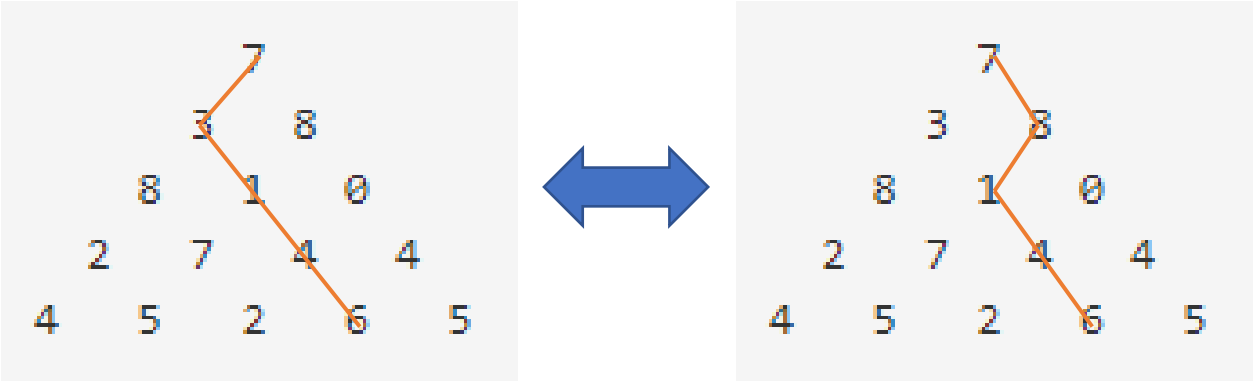
지난시간 도전문제 – 가장 긴 증가하는 부분 수열

- 완전탐색의 시간복잡도

  - 현재 숫자를 고를지, 말지 2가지 선택이 최대 1000번 있으므로

  - O(2^N)

- 계산에 중복이 있는가?




```
6
10 20 10 30 20 50
```

지난시간 도전문제 – 가장 긴 증가하는 부분 수열

- 수열의 시작점이 정해지면 최대 길이는 항상 고정된 값을 갖는다.

  - 앞에서 선택한 숫자들의 영향은 현재 숫자를 부분 수열안에 넣을 수 있는지에 국한된다.

F(n) : n번째 숫자에서부터 시작하는 증가 부분 수열중 최대 길이

max(F(0), F(1), … , F(n)) = 문제가 원하는 답

지난시간 도전문제 – 가장 긴 증가하는 부분 수열

- Top-down

F(x) = 1 + max(F(x+1), F(x+2), ... , F(N-1))

```
int F(int x)
    if(cache[x] != -1)
        return cache[x]
    int ret = 1
    for(int i=x+1; i<N; i++)
        if(arr[i] > arr[x])
            ret = max( ret, 1 + F(i) )
    return cache[x] = ret
```

cache[i] : i번째 원소를 처음에 포함하는
가장 긴 증가하는 부분 수열의 길이

ret의 초기값을 1로 설정

- 어떤 숫자에서 시작하는 최소 길이는 1
- 종료 조건 설정 필요 x

지난시간 도전문제 – 가장 긴 증가하는 부분 수열

- Bottom-up

```
for(int i=0; i<N; i++)
    cache[i] = 1
    for(int j=0; j<i; j++)
        if(arr[i] > arr[j] && cache[j] + 1 > cache[i])
            cache[i] = cache[j] + 1
```

cache[i] : i번째 원소를 끝에 포함하는
가장 긴 증가하는 부분 수열의 길이

지난시간 도전문제 – 가장 긴 증가하는 부분 수열

- 마지막 답 구하기

```
int ret = -1
    for(int i=0; i<N; i++)
        int ret = max(ret, cache[i])
cout<<ret
```

최종적으로 O(N^2)의 시간복잡도

- 이분탐색을 이용한 O(NlogN) ?

지난시간 도전문제 – 입국심사

문제 요약 :

- $1 \le N \le 100,000$, $1 \le M \le 1,000,000,000$, $1 \le T_k \le 10^9$
- 비어있는 심사대가 있으면 거기로 가서 심사를 받을 수 있다
- 심사를 마치는데 걸리는 시간의 최솟값을 출력

고려해야 할 사항 :

- 빠른 입출력
- 오버플로우
- 심사를 마치는데 걸리는 시간의 최소 최대값?

지난시간 도전문제 – 입국심사

- 빠른 입출력

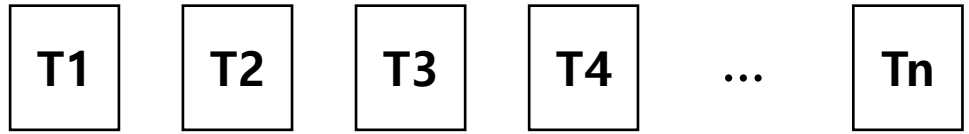
1 ≤ N ≤ 100,000 개의 입력이 주어질 수 있으므로, 빠른 입출력을 사용해 빠르게 입력을 받아야 시간안에 들어옴

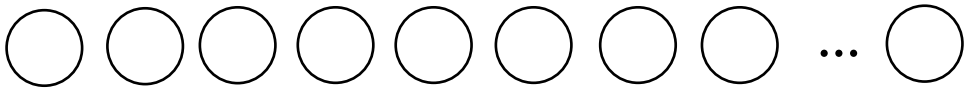
```
ios::sync_with_stdio(false);
cin.tie(NULL);
```

https://www.acmicpc.net/board/view/22716

지난시간 도전문제 – 입국심사

- 심사를 마치는데 걸리는 시간의 최소 최대값?

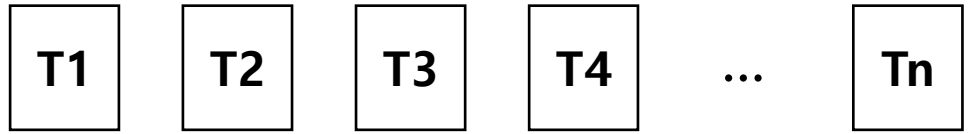




M명의 사람들

지난시간 도전문제 – 입국심사

- 심사를 마치는데 걸리는 시간의 최소 최대값?
  - 최소값



최소값은 1

1명의 사람

지난시간 도전문제 – 입국심사

- 심사를 마치는데 걸리는 시간의 최소 최대값?
  - 최대값

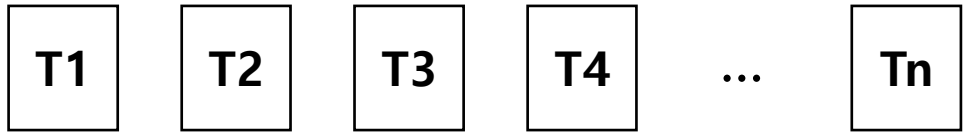
T1

최대값은 1,000,000,000 * 1,000,000,000

○ ○ ○ ○ ○ ○ ○ ○ … ○ 1,000,000,000명의 사람들

지난시간 도전문제 – 입국심사

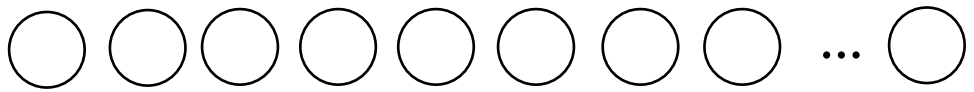
- 오버플로우
  - 10^18은 int 범위를 벗어나지만 long long 범위 안에서 처리 가능
- 시간의 최소값을 찾기위한 이분탐색 진행



임의의 시간 T 설정, - 1 ≤ T ≤ 10^18

각각의 입국심사대마다 T초동안 몇 명의 사람을 통과 시킬 수 있는지 계산 후 더하기

더한 값이 M보다 크거나 같으면 T는 최소값의 후보



M명의 사람들

- O(logT * N)

지난시간 도전문제 – 입국심사

- 실제 구현

bs(l, r) : 시간의 범위가 [l, r]일 때, check를 통과하는 최소 시간을 반환한다.

```
ll bs (ll l, ll r)
    ll mid = (l+r)/2
    if(l > r)
        return MAX
    if(check(mid))
        return min(mid, bs(l, mid-1))
    else
        return bs(mid+1, r)
```

check(x) : 걸리는 시간이 x라고 가정했을 때, 심사에 통과할 수 있는 사람들이 M명 이상인지 반환

```
bool check (ll x)
    ll sum = 0
    for(int i=0; i<N; i++)
        sum += x/T[i]
        if(sum >= M)
            return true

    return sum >= M
```

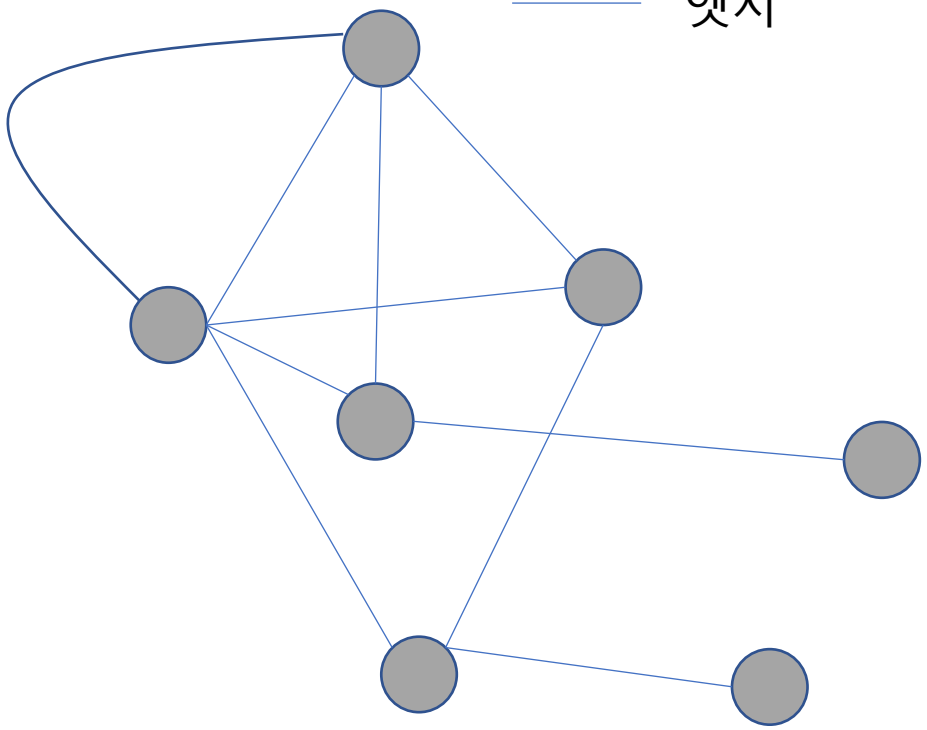
그래프

DFS(깊이우선탐색)
BFS(너비우선탐색)

→ 그래프의 탐색 방법

노드
엣지

→ 정보가 담기는 공간



그래프란?

- 노드와 엣지로 상태공간을 나타낸다.
- 방향, 무방향 둘 다 가능하다.
- 노드 끼리 직접, 간접적으로 연결 될 수 있다.
- 정점과 정점사이의 엣지가 여러개일수도 있다.
- 문제상황의 개괄을 보는데 도움이 된다.
- 거의 모든 상태공간을 그래프로 나타내는게 가능하다.

그래프

- 그래프의 구현 방식

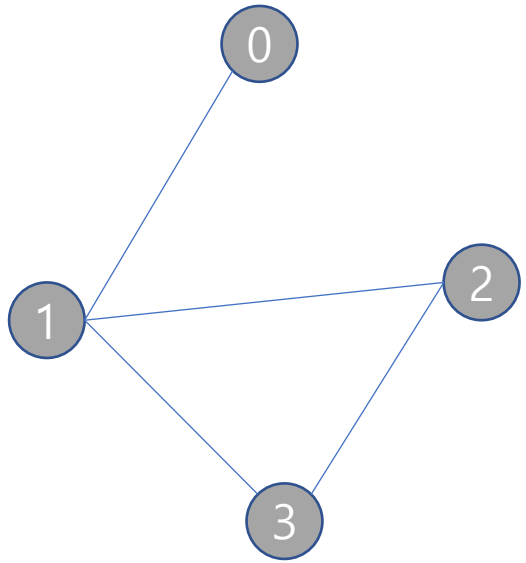
1. 인접 행렬

   - 배열을 이용해 비교적 구현이 간단하다
   - 어떤 노드와 노드가 직접적으로 연결 되어있는지 O(1)만에 판단 가능
   - 메모리가 많이 든다

2. 인접 리스트

   - vector를 이용해 비교적 구현이 어렵다
   - 직접 연결 되어있는지 바로 판단할 수 없다.
   - 메모리가 적게 든다
   - 많이 쓰이는 방법

- 인접 행렬



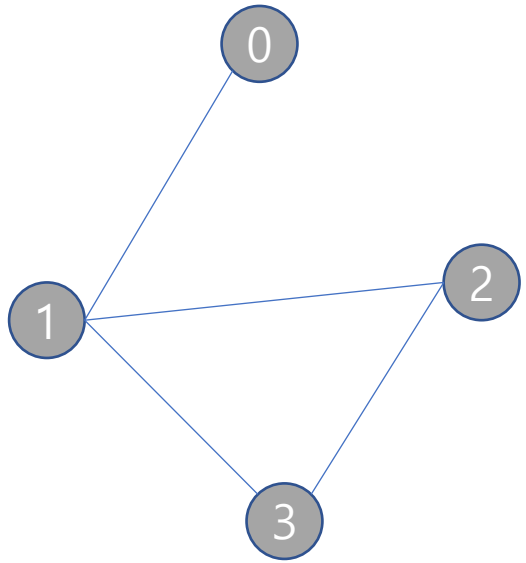
	0	1	2	3
0		O	X	X
1	O		O	O
2	X	O		O
3	X	O	O	

1번 노드가 3번 노드와 직접 연결 되어있는지?

- arr[1][3]을 보면 된다.

N개의 노드가 있을 때, N^2의 메모리가 필요

- 인접 리스트



vector<int> linked[4]

linked[0] : {1}

linked[1] : {0,2,3}

linked[2] : {1,3}

linked[3] : {1,2}

1이 3번 노드와 직접 연결 되어있는지?

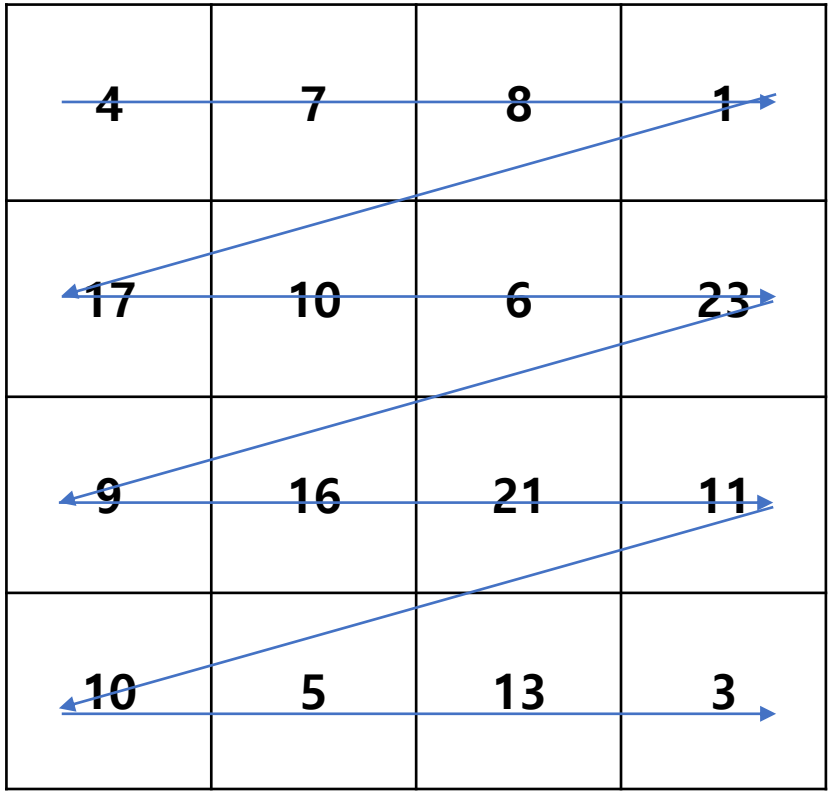
- linked[1]의 원소를 다 둘러본다

N개의 노드가 있을 때, 꼭 N^2의 메모리가 필요하지는 않다

그래프

- 배열안에 5가 있나 탐색

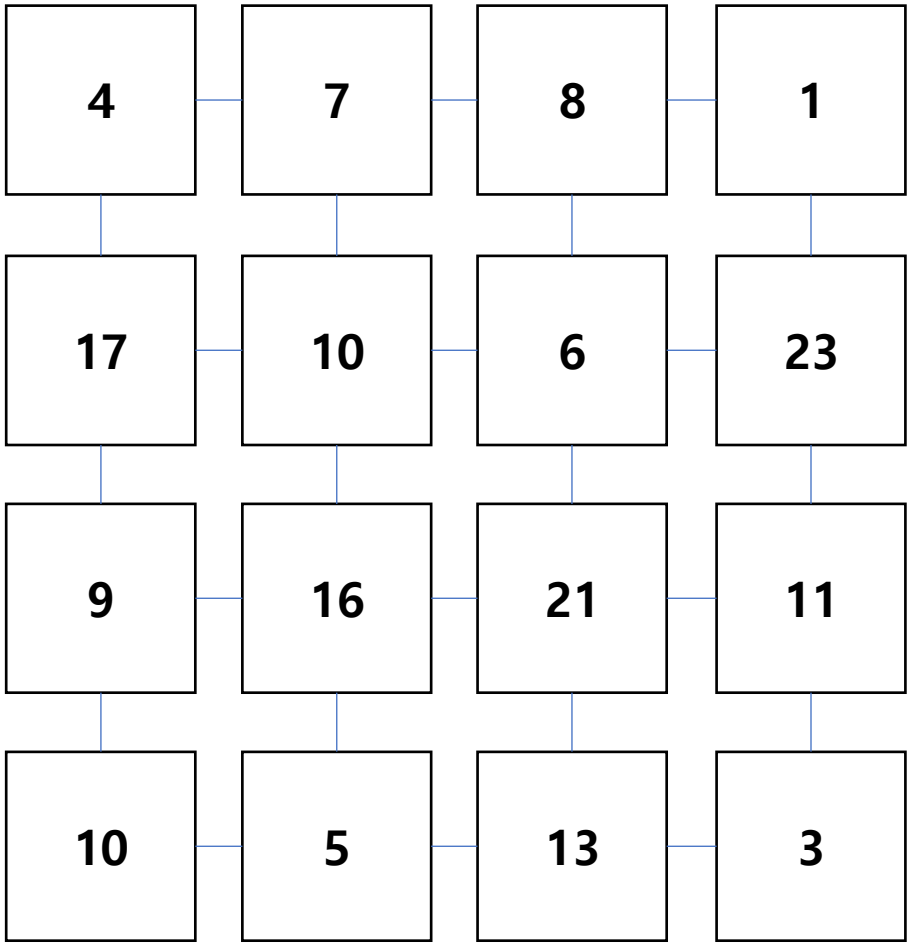
4	7	8	1
17	10	6	23
9	16	21	11
10	5	13	3



그래프

- 배열안에 5가 있나 탐색

4	7	8	1
17	10	6	23
9	16	21	11
10	5	13	3



그래프

- 대표적인 그래프 탐색 방식

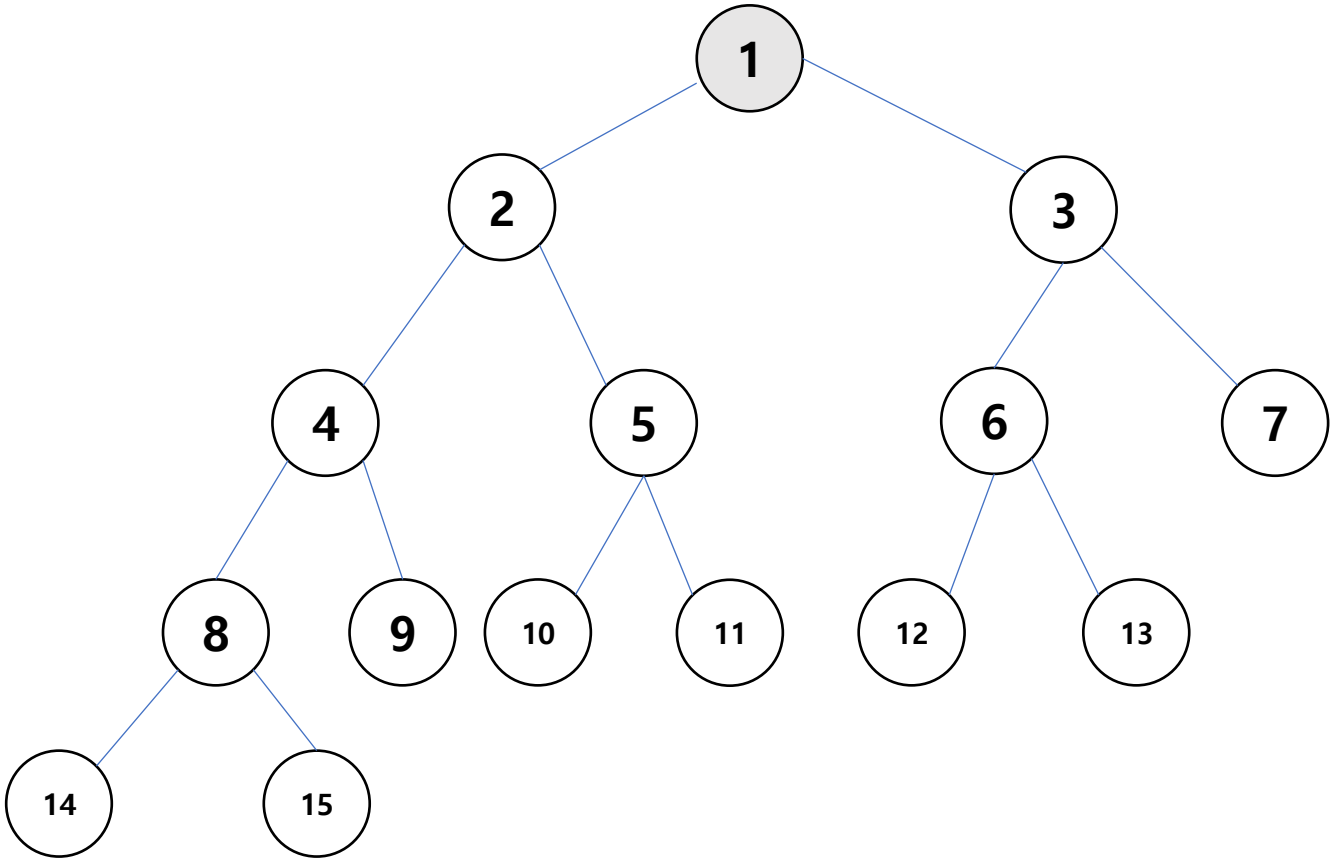
    1. DFS(깊이 우선 탐색)

        - 재귀를 이용한다.
        - 백트래킹과 매우 유사
        - 깊이 우선

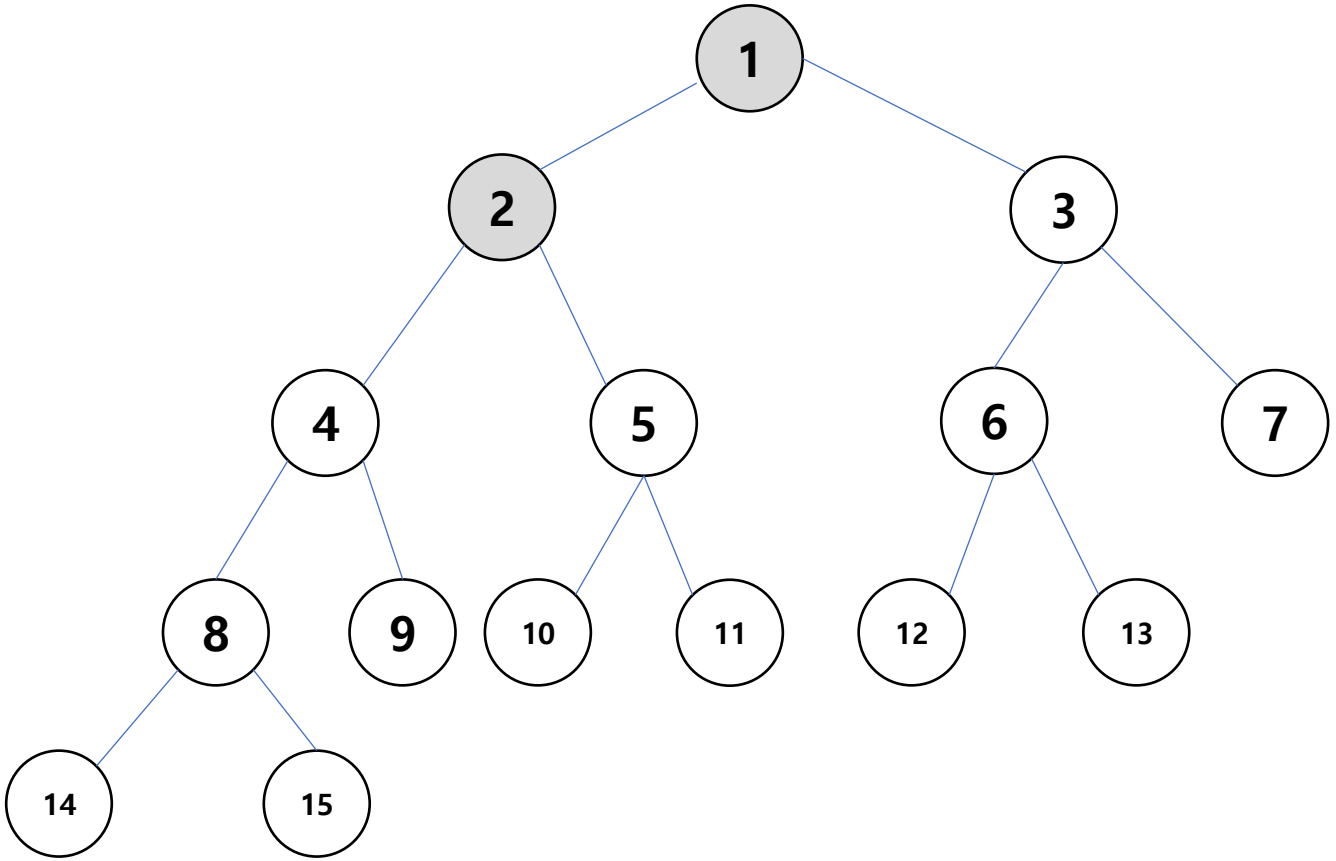
    2. BFS(너비 우선 탐색)

        - Queue를 이용한다.
        - 간선들의 가중치가 1인 그래프에서 최단거리임이 보장된다.
        - 너비 우선

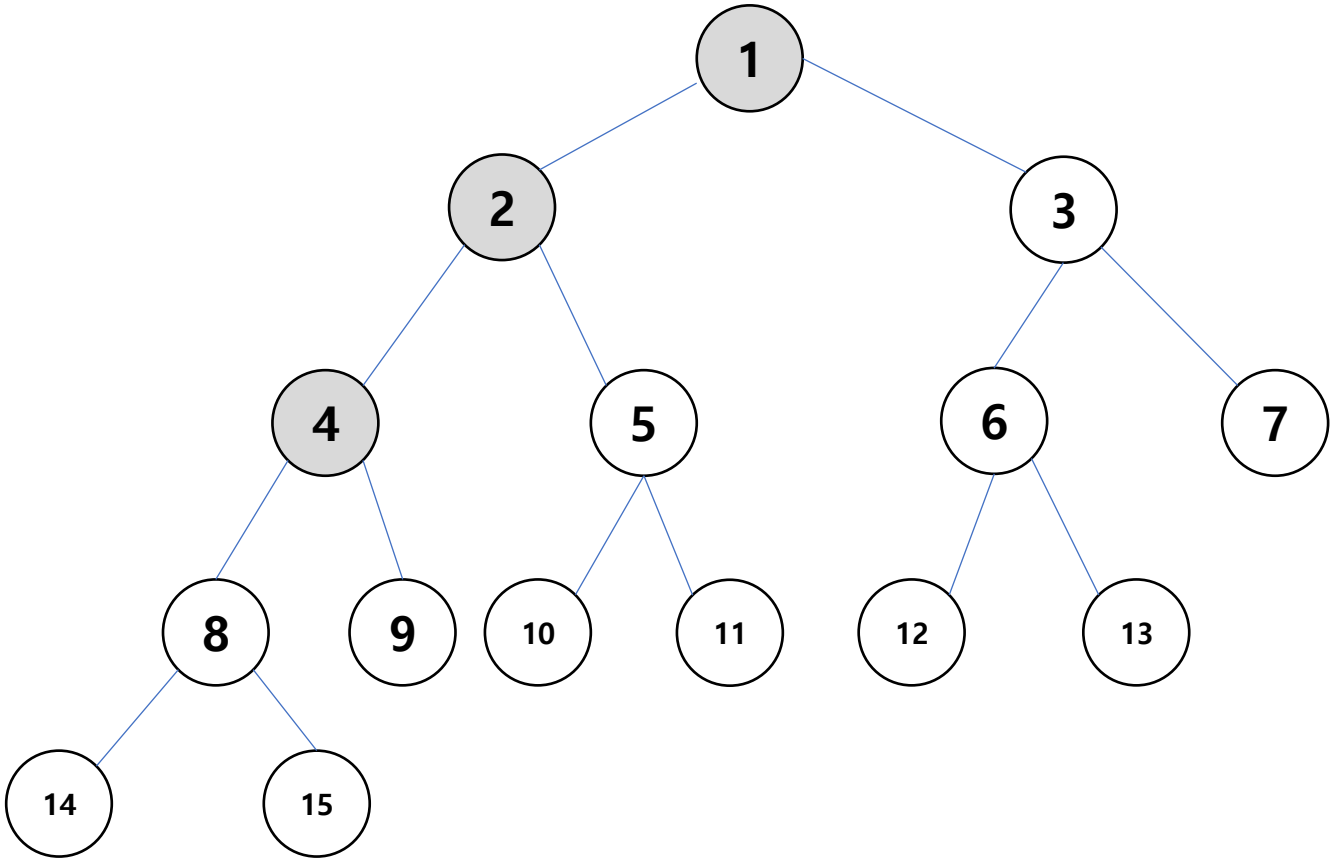
- DFS(깊이 우선 탐색)



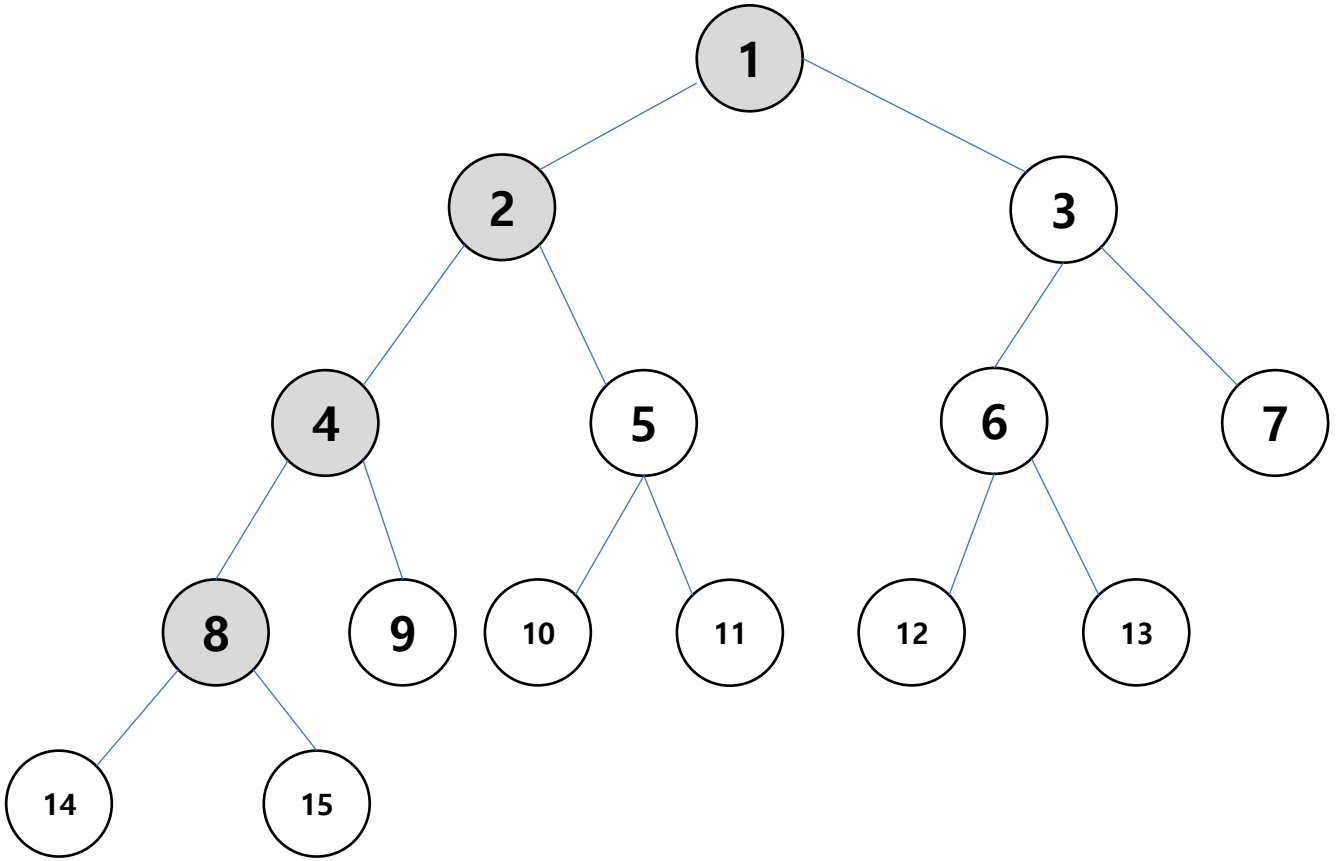
- DFS(깊이 우선 탐색)



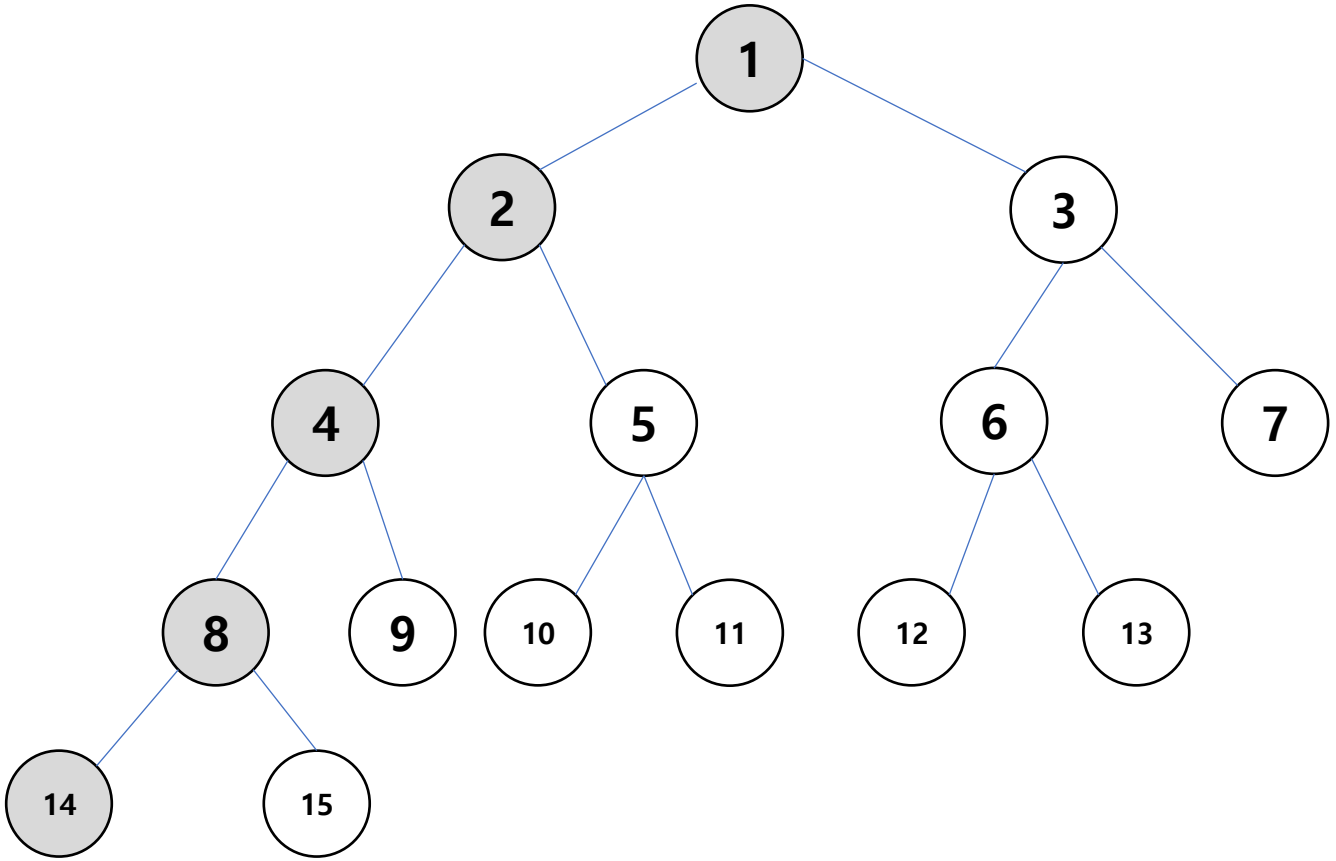
- DFS(깊이 우선 탐색)



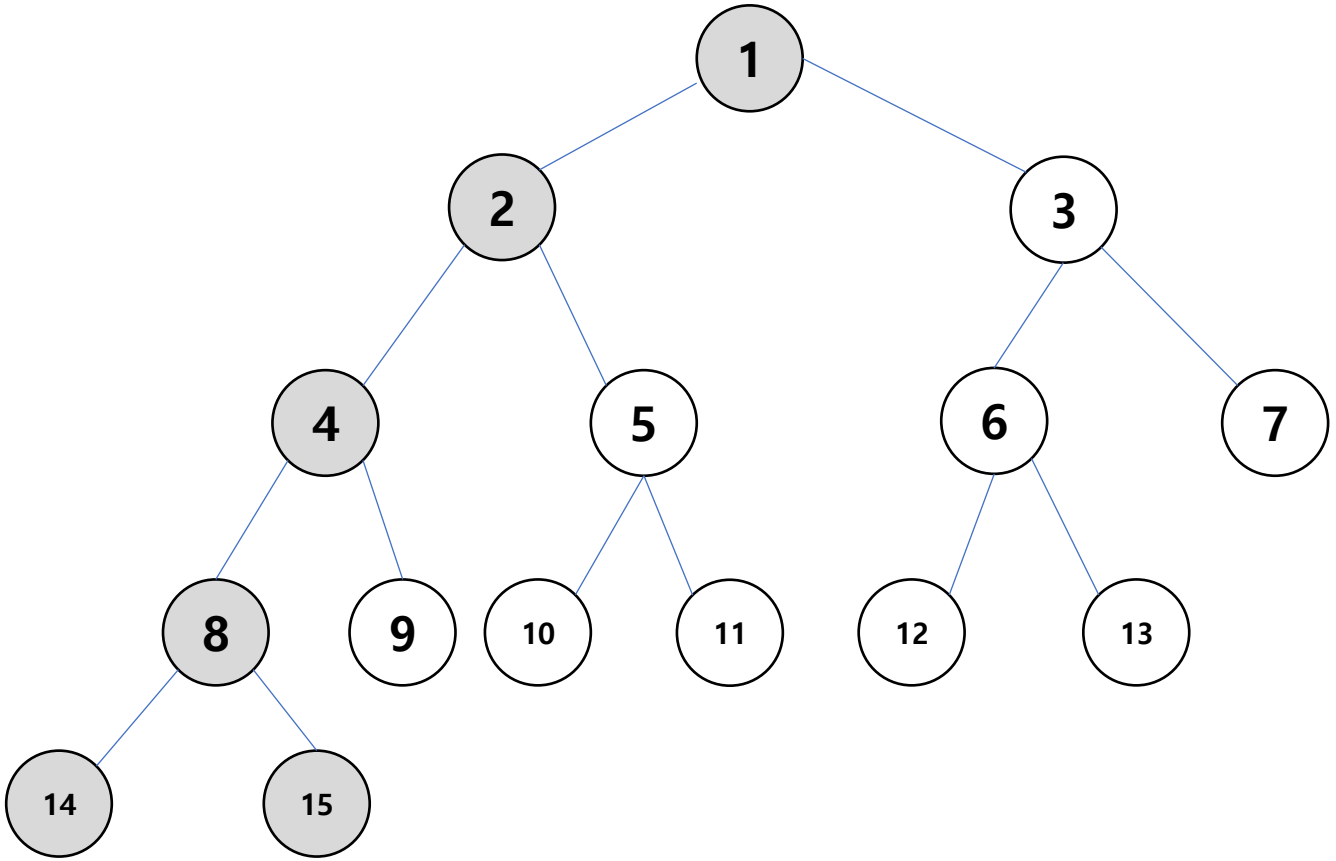
- DFS(깊이 우선 탐색)



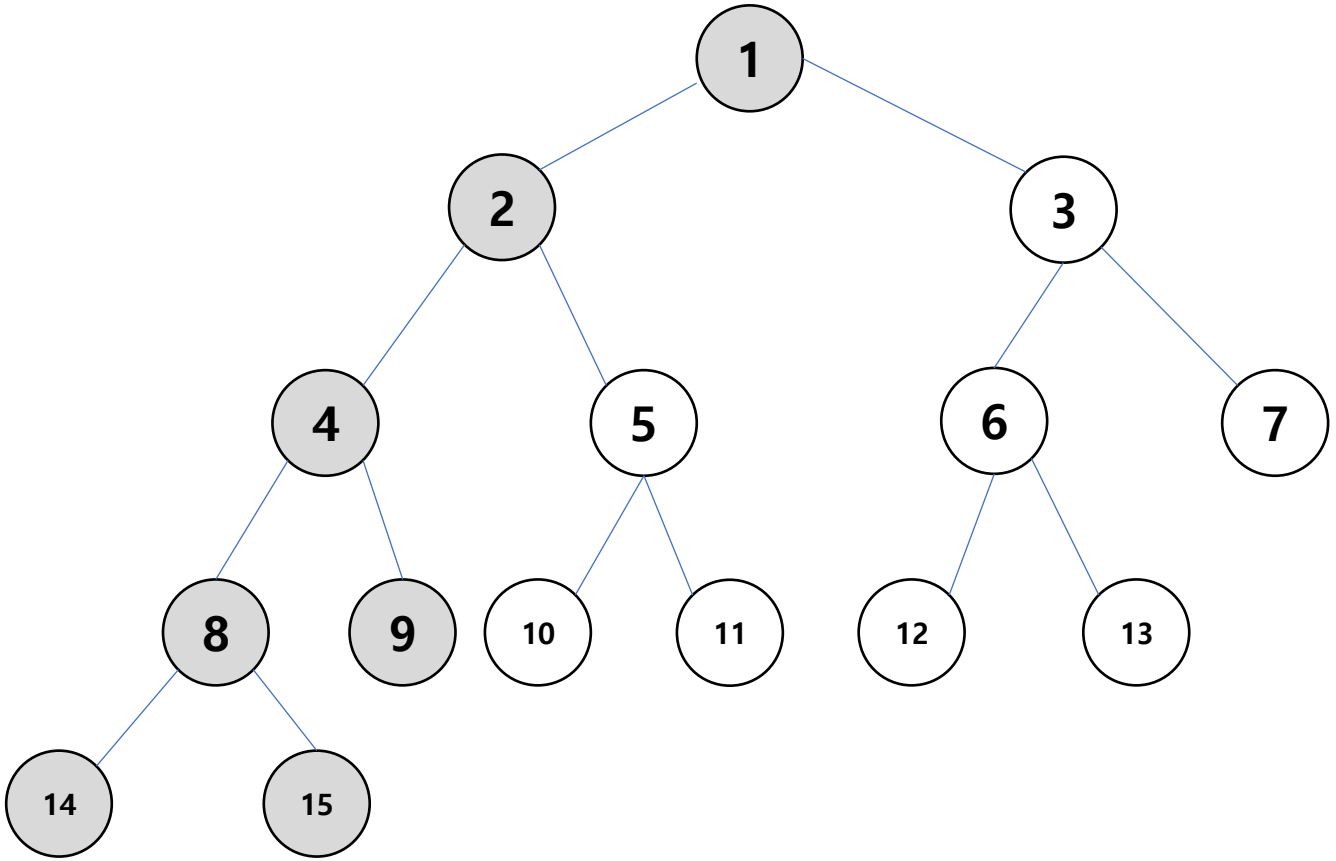
- DFS(깊이 우선 탐색)



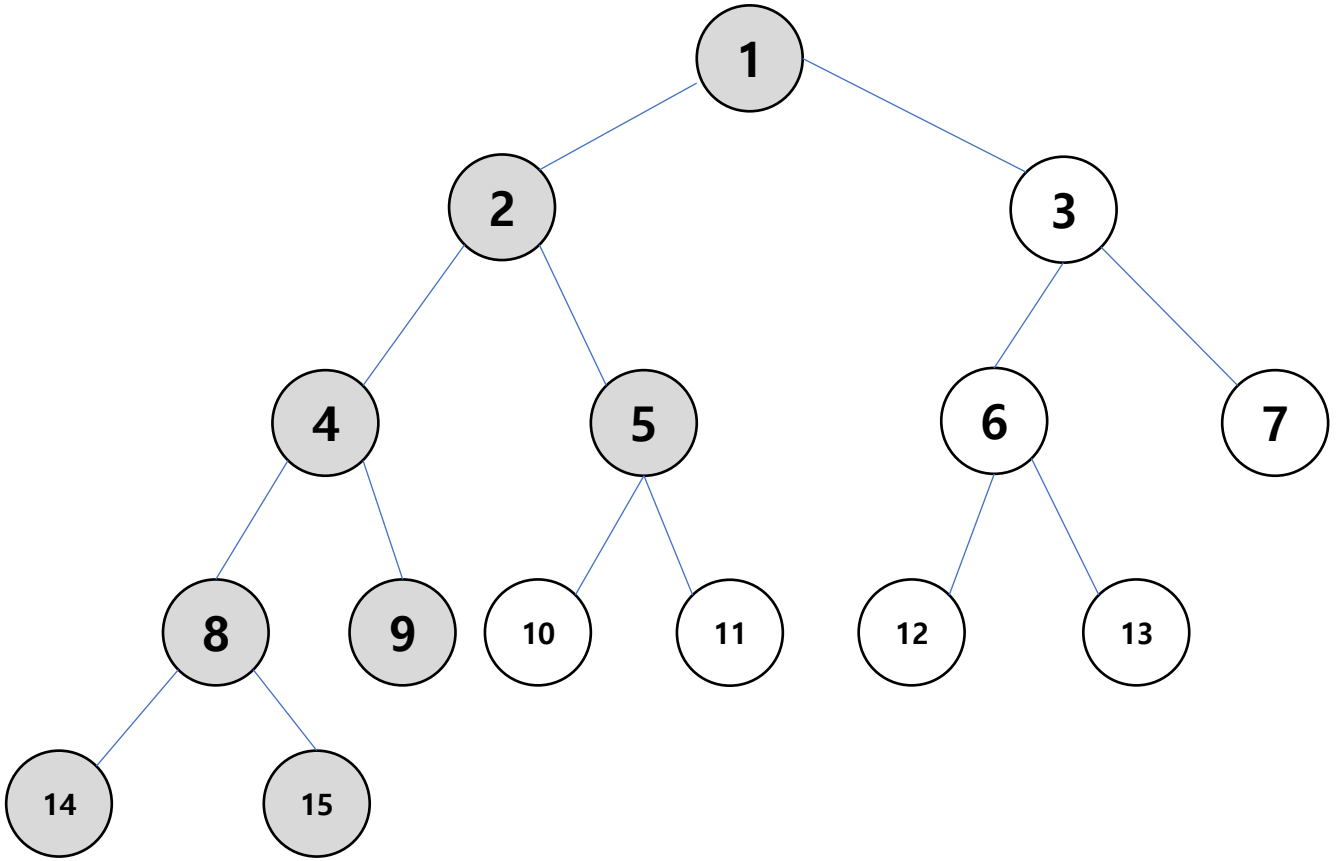
- DFS(깊이 우선 탐색)



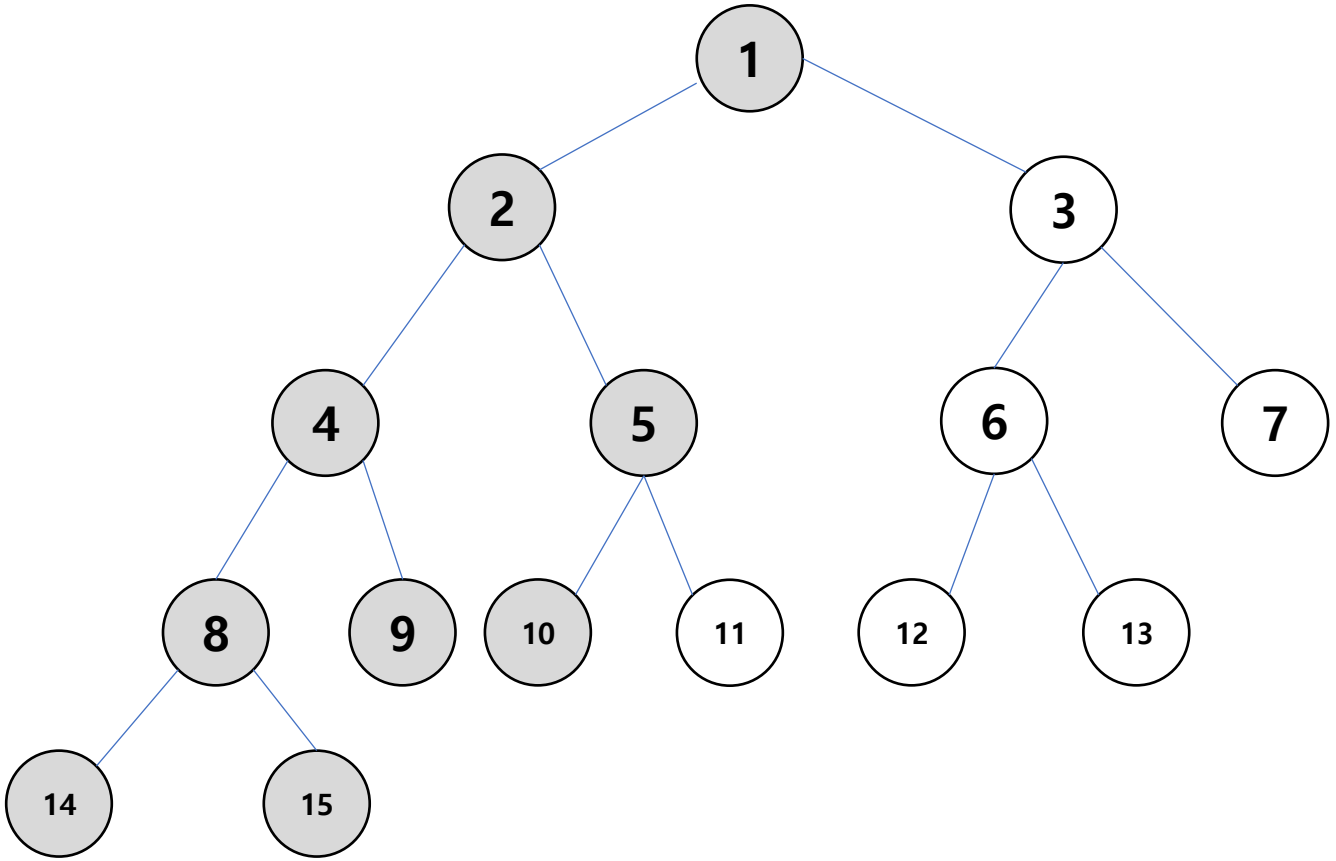
- DFS(깊이 우선 탐색)



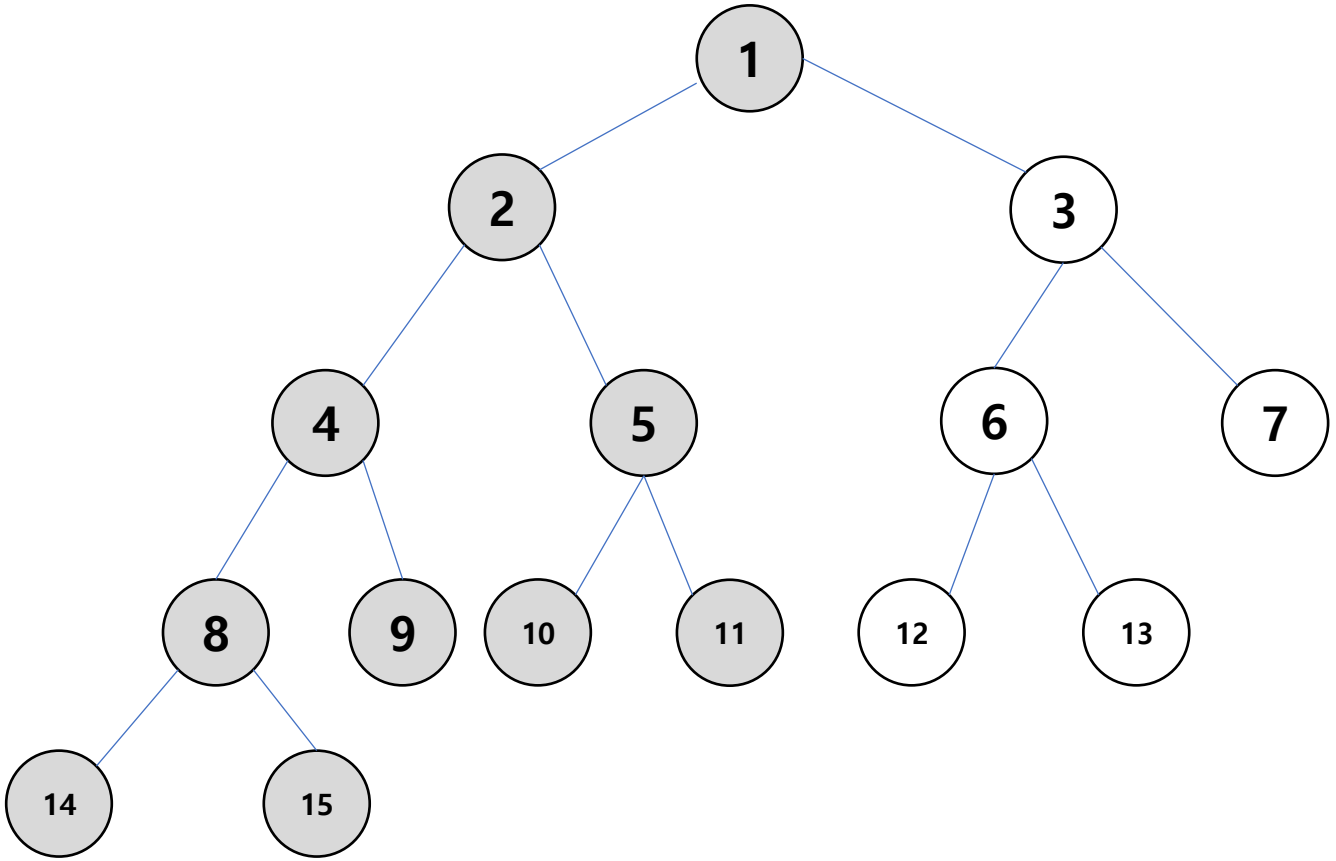
- DFS(깊이 우선 탐색)



- DFS(깊이 우선 탐색)



- DFS(깊이 우선 탐색)



- DFS(깊이 우선 탐색)
  - 현재 노드랑 연결된 노드중에 !visited인 것을 따라간다.
  - 백트래킹과 달리 상태를 되돌리지 않는다.

```
void DFS(int x)
    for(int i=0; i<linked[x].size; i++)
        if(!visited[i])
            visited[i] = true
            DFS(i)
```

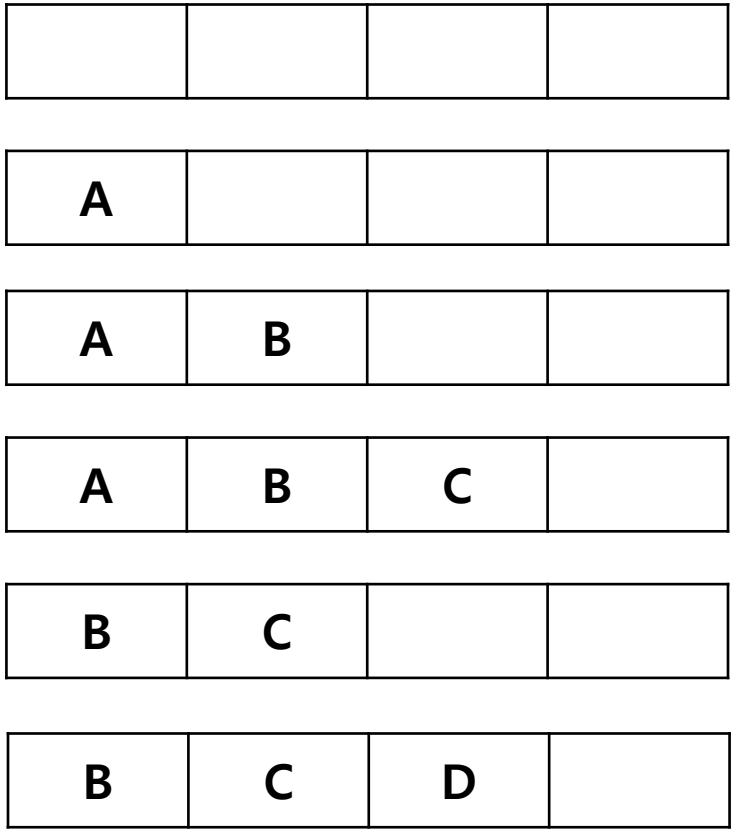
- BFS(너비 우선 탐색)
  - Queue? FIFO(First-In, First-Out)
    매표소에 순서대로 줄 서있는 사람들
  - Enqueue : 큐에 원소를 넣는다.
  - Dequeue : 큐에서 원소를 뺀다.

시작점에서 가까운 지점부터 탐색을 한다

DFS와 다르게 visited가 아닌 discovered로 상태를 표현

모든 정점은 아래 세 개의 상태를 순서대로 거치게 된다

- 아직 발견되지 않은 상태
- 발견되었지만 아직 방문되지는 않은 상태(큐에 저장)
- 방문된 상태



- BFS(너비 우선 탐색)

```
void BFS(int start)
    q.Enqueue(start)
    discovered[start] = true
    while(!q.empty())
        int here = q.Dequeue()
        for(int i=0; i<linked[here].size(); i++)
            if( !discovered[linked[here][i]] )
                discovered[linked[here][i]] = true
                q.Enqueue(linked[here][i])
```

시작점은 미리 Queue에 넣어주고 discovered 처리

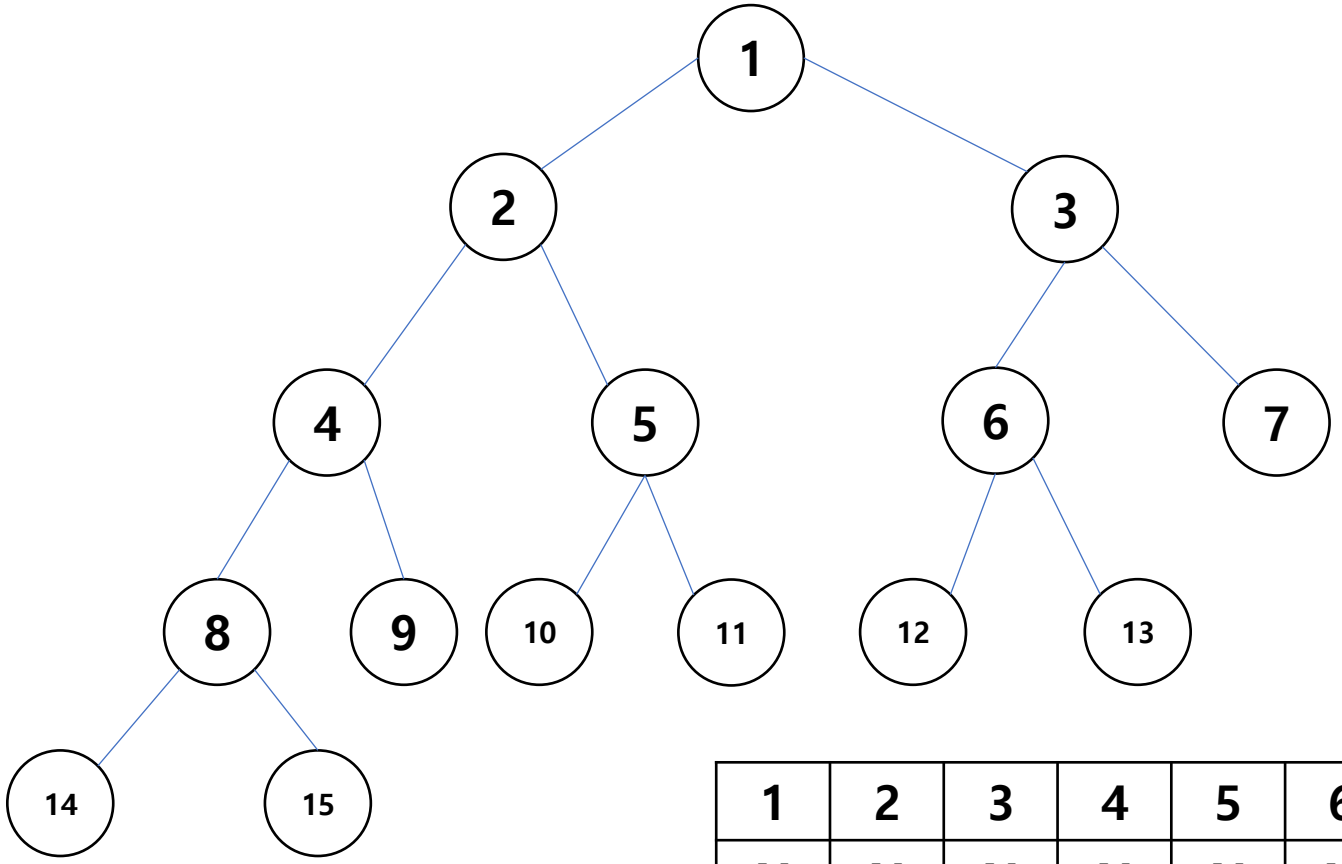
Queue가 빌 때 까지

here 노드와 연결된 노드 중에서

!discovered인게 있으면

해당 지점을 Queue에 넣어주고 discovered 처리

- BFS(너비 우선 탐색)



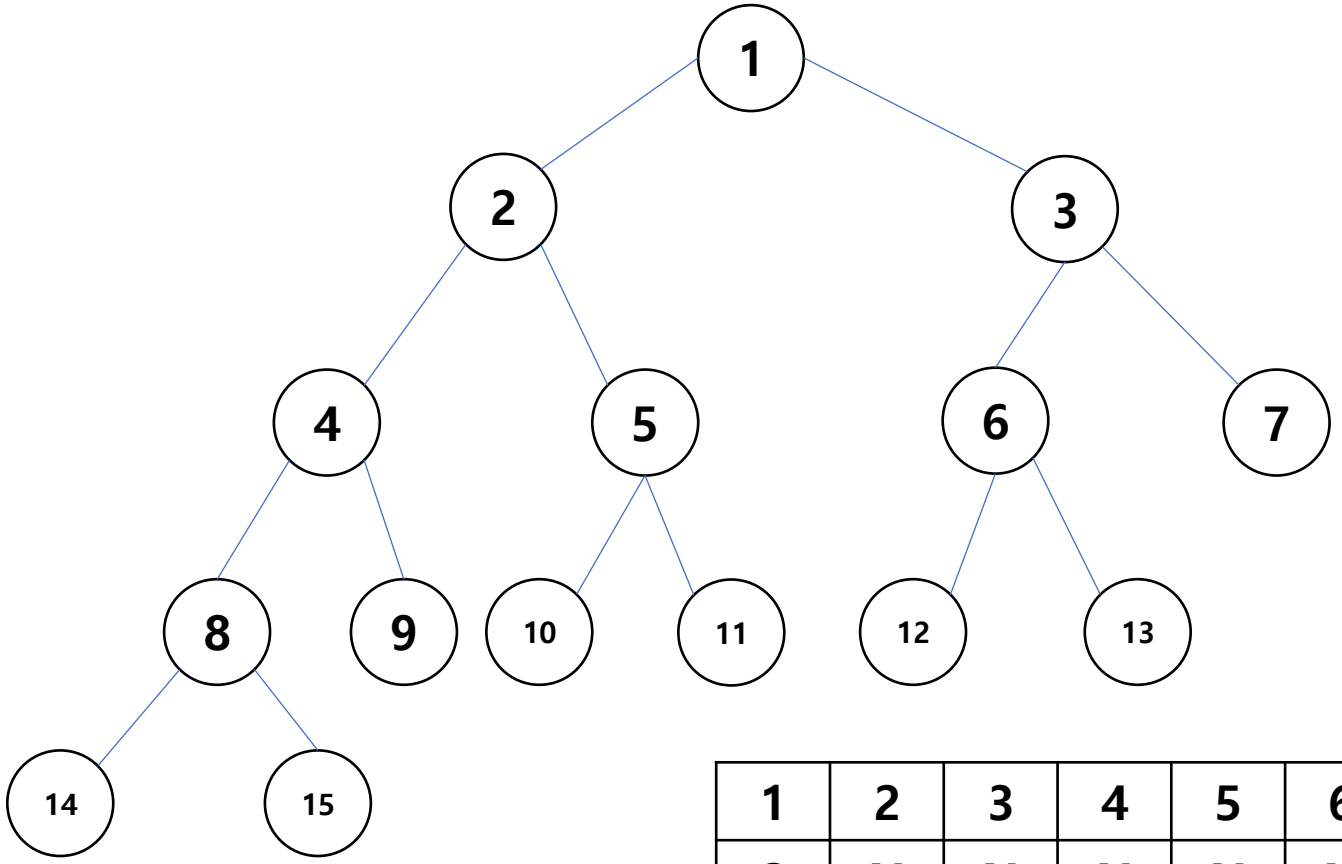
1부터 BFS 시작

Queue

discovered

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

- BFS(너비 우선 탐색)



here : x

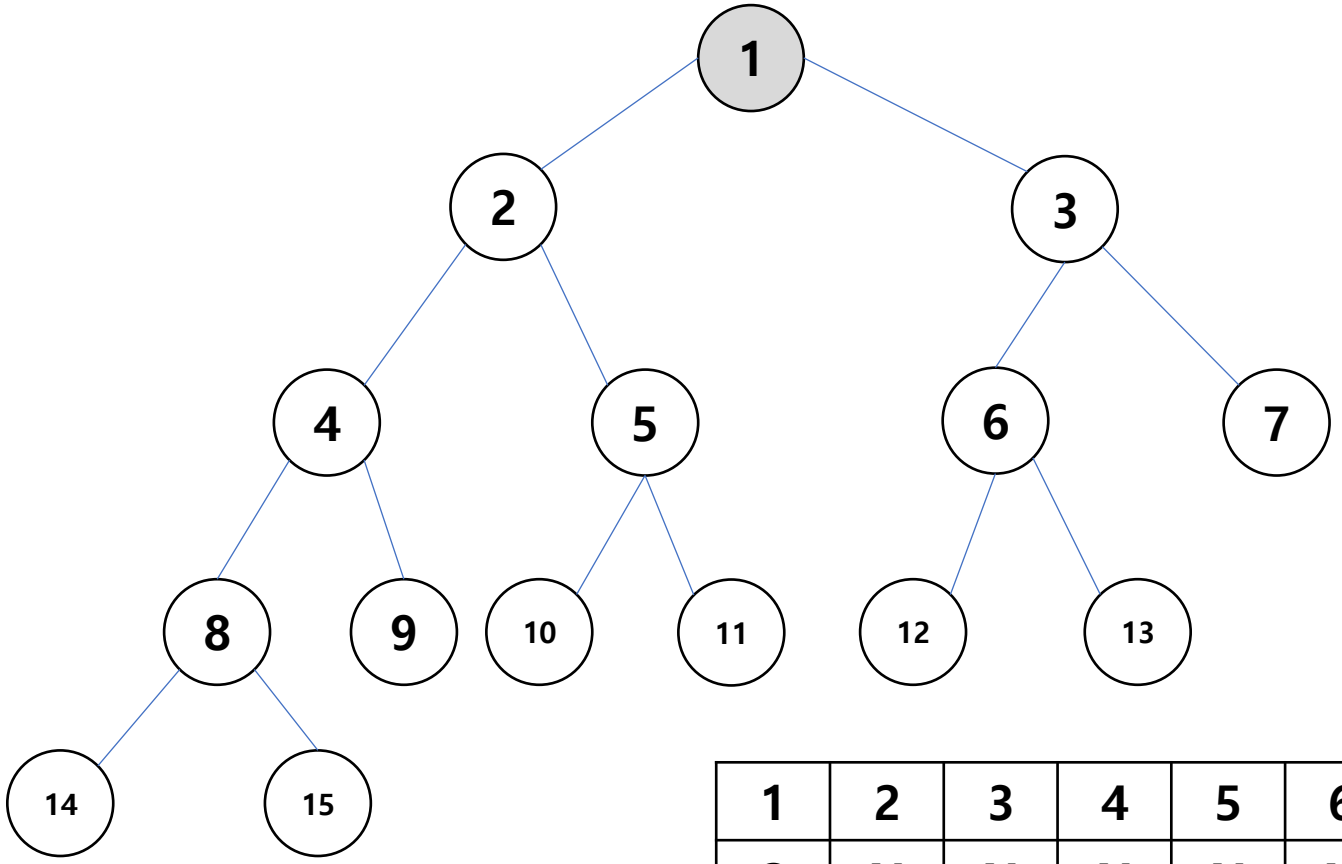
Queue

1							

discovered

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
O	X	X	X	X	X	X	X	X	X	X	X	X	X	X

- BFS(너비 우선 탐색)



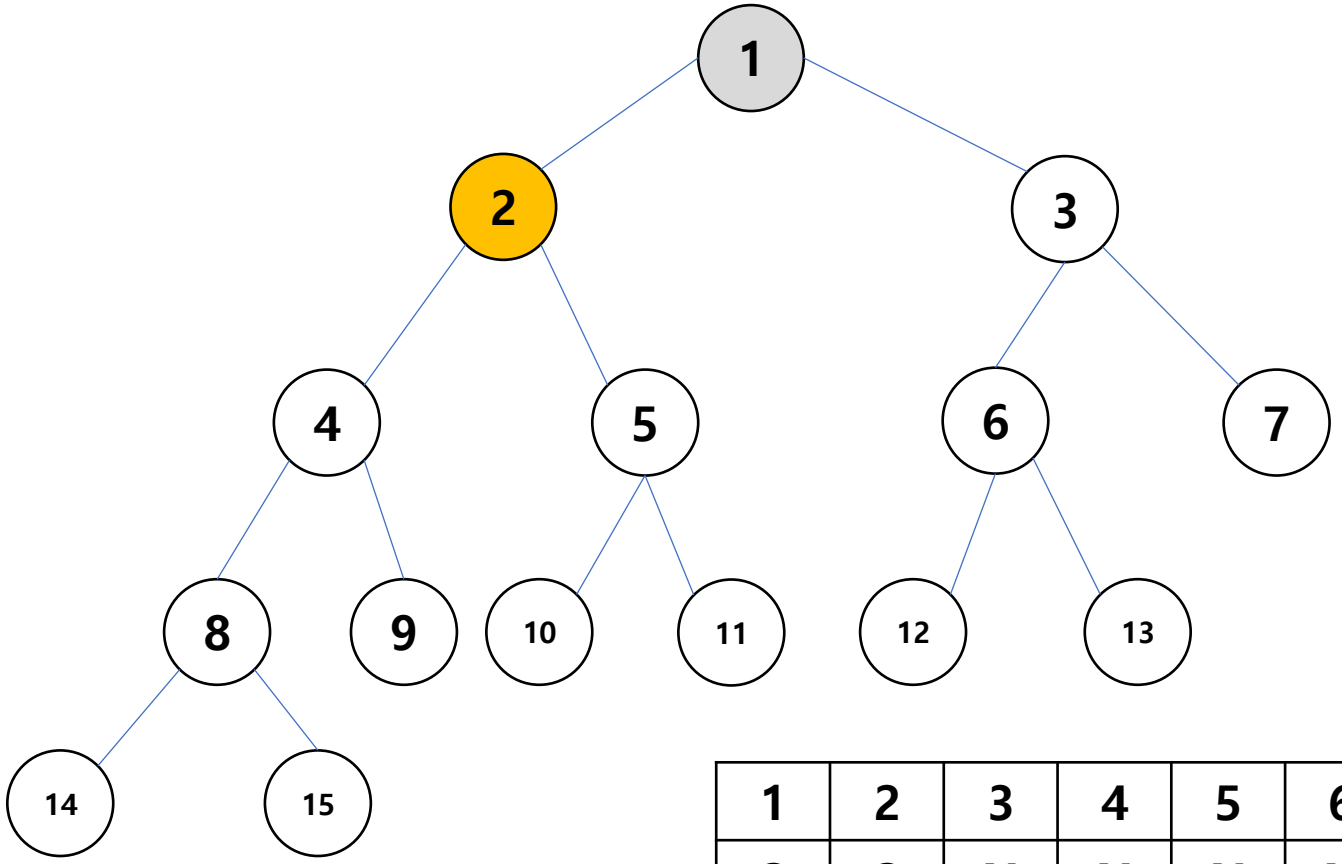
here : 1

Queue

discovered

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
O	X	X	X	X	X	X	X	X	X	X	X	X	X	X

- BFS(너비 우선 탐색)



here : 1

linked[here] : 2

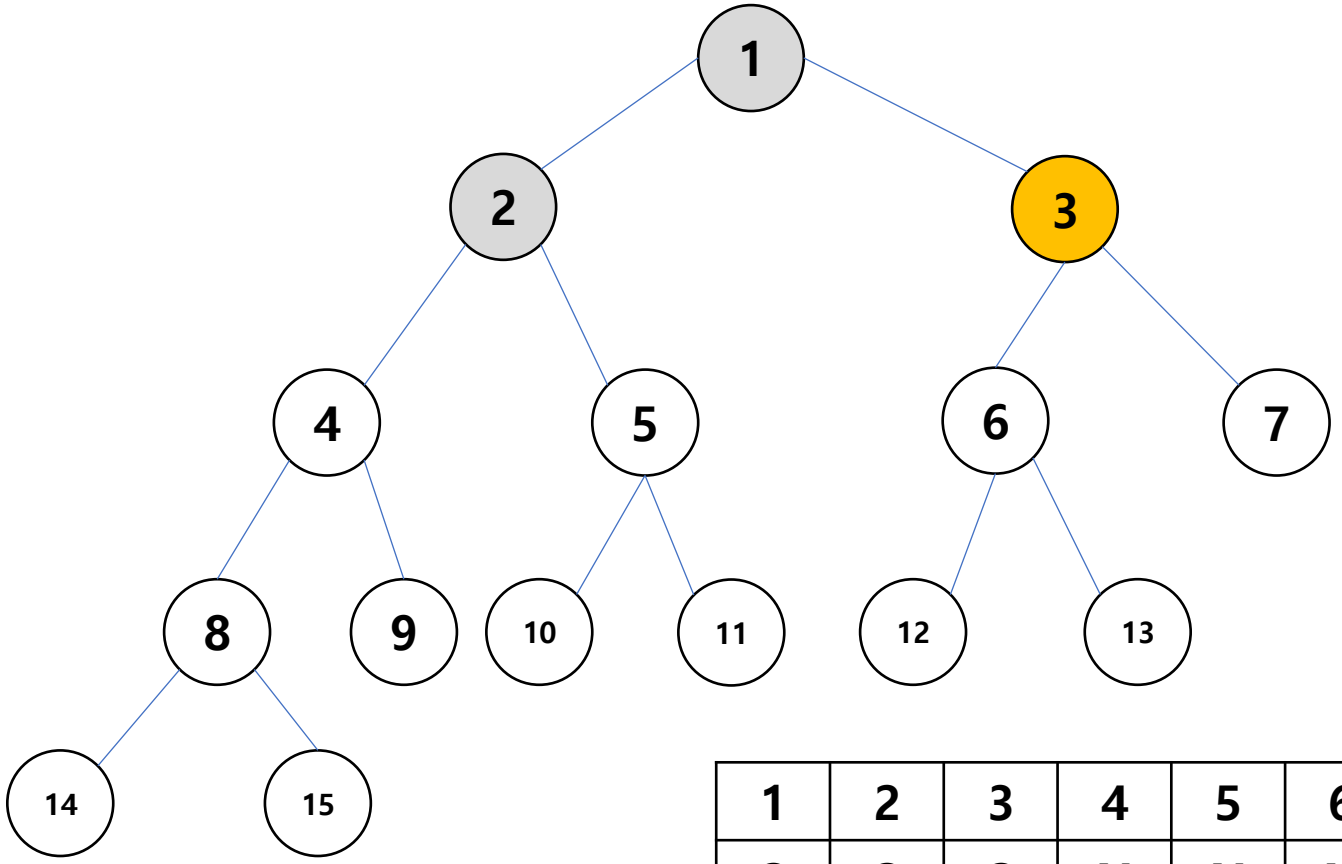
Queue

2							

discovered

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
O	O	X	X	X	X	X	X	X	X	X	X	X	X	X

- BFS(너비 우선 탐색)



here : 1

linked[here] : 2

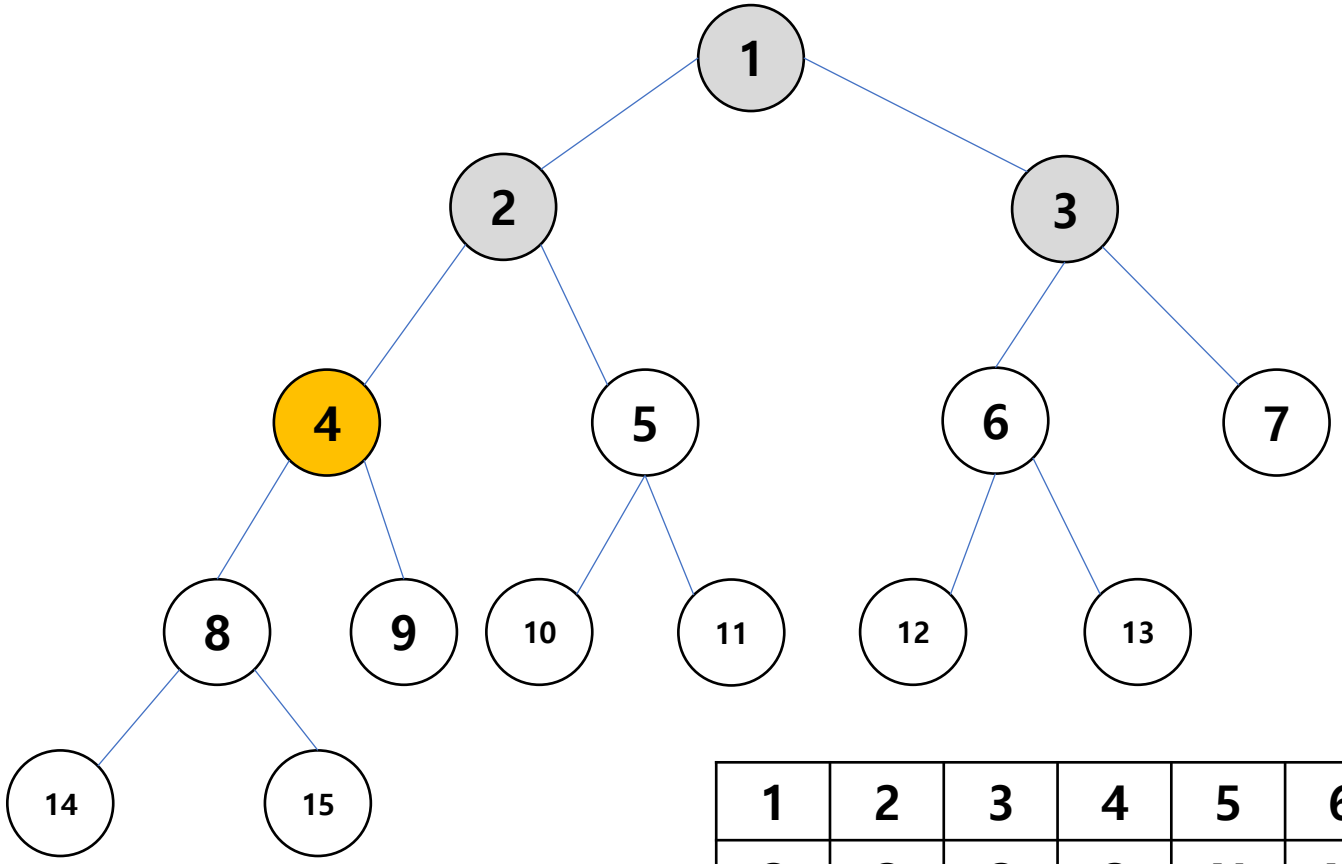
Queue

2	3						

discovered

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
O	O	O	X	X	X	X	X	X	X	X	X	X	X	X

- BFS(너비 우선 탐색)



here : 2

linked[here] : 4

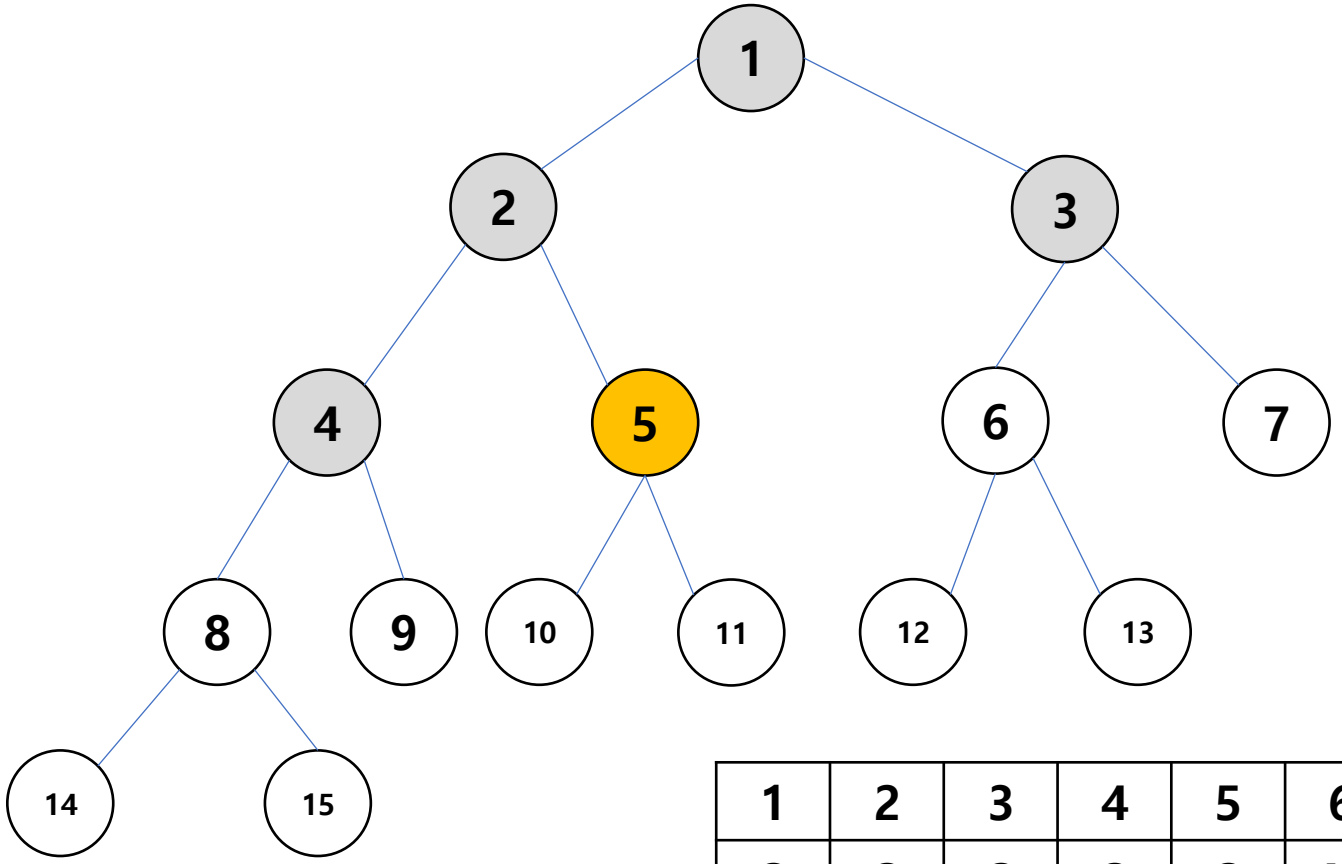
Queue

3	4						

discovered

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
O	O	O	O	X	X	X	X	X	X	X	X	X	X	X

- BFS(너비 우선 탐색)



here : 2

linked[here] : 5

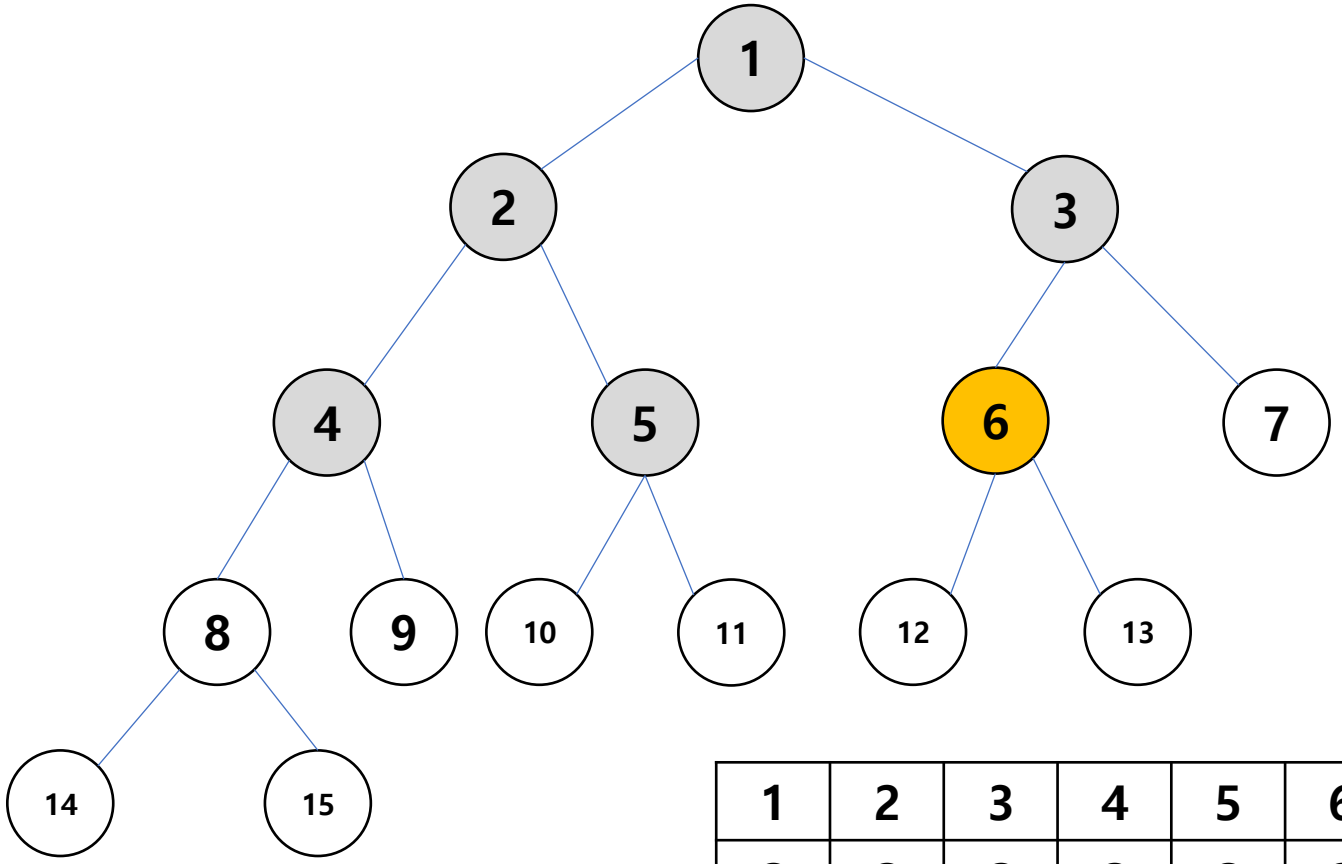
Queue

3	4	5					

discovered

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
O	O	O	O	O	X	X	X	X	X	X	X	X	X	X

- BFS(너비 우선 탐색)



here : 3

linked[here] : 6

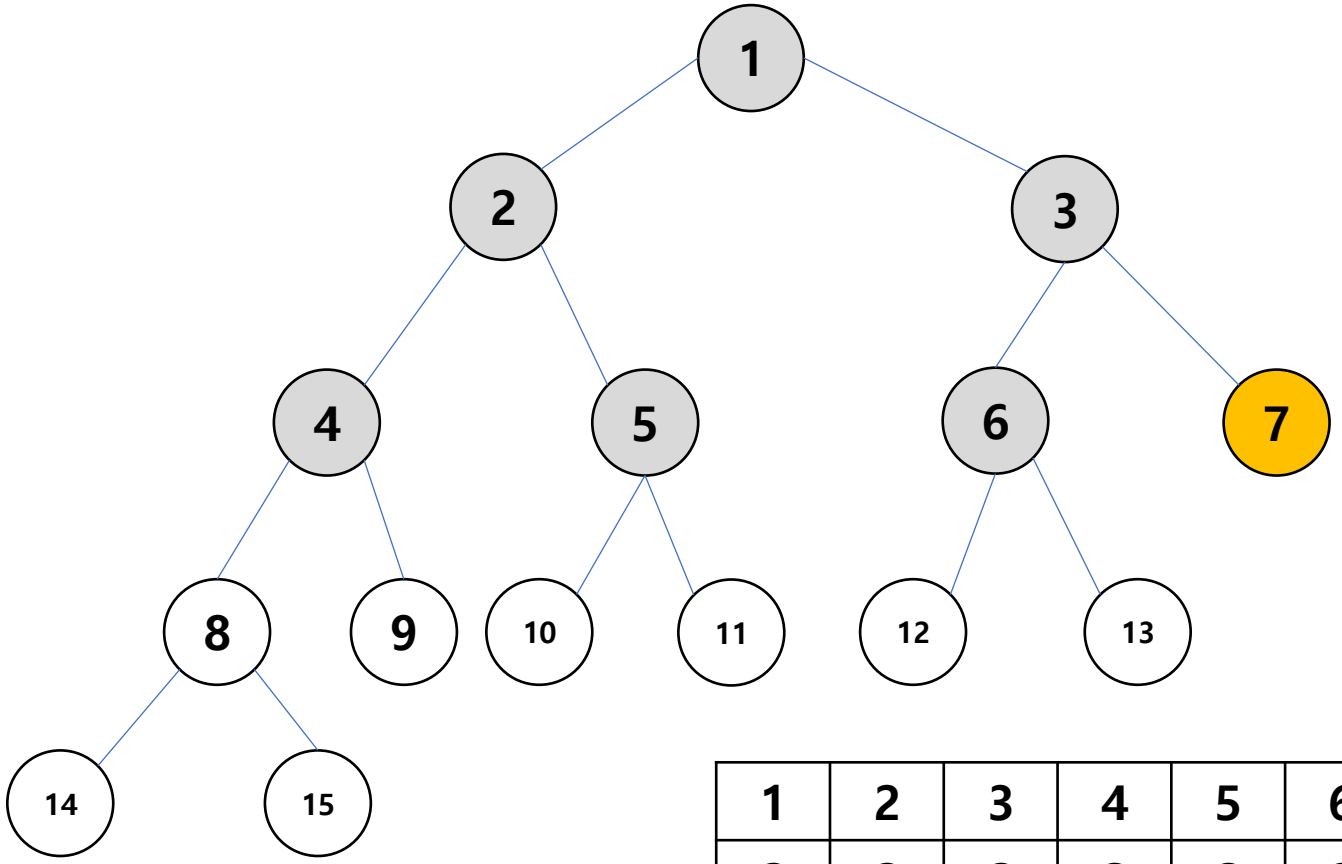
Queue

4	5	6					

discovered

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
O	O	O	O	O	O	X	X	X	X	X	X	X	X	X

- BFS(너비 우선 탐색)



here : 3

linked[here] : 7

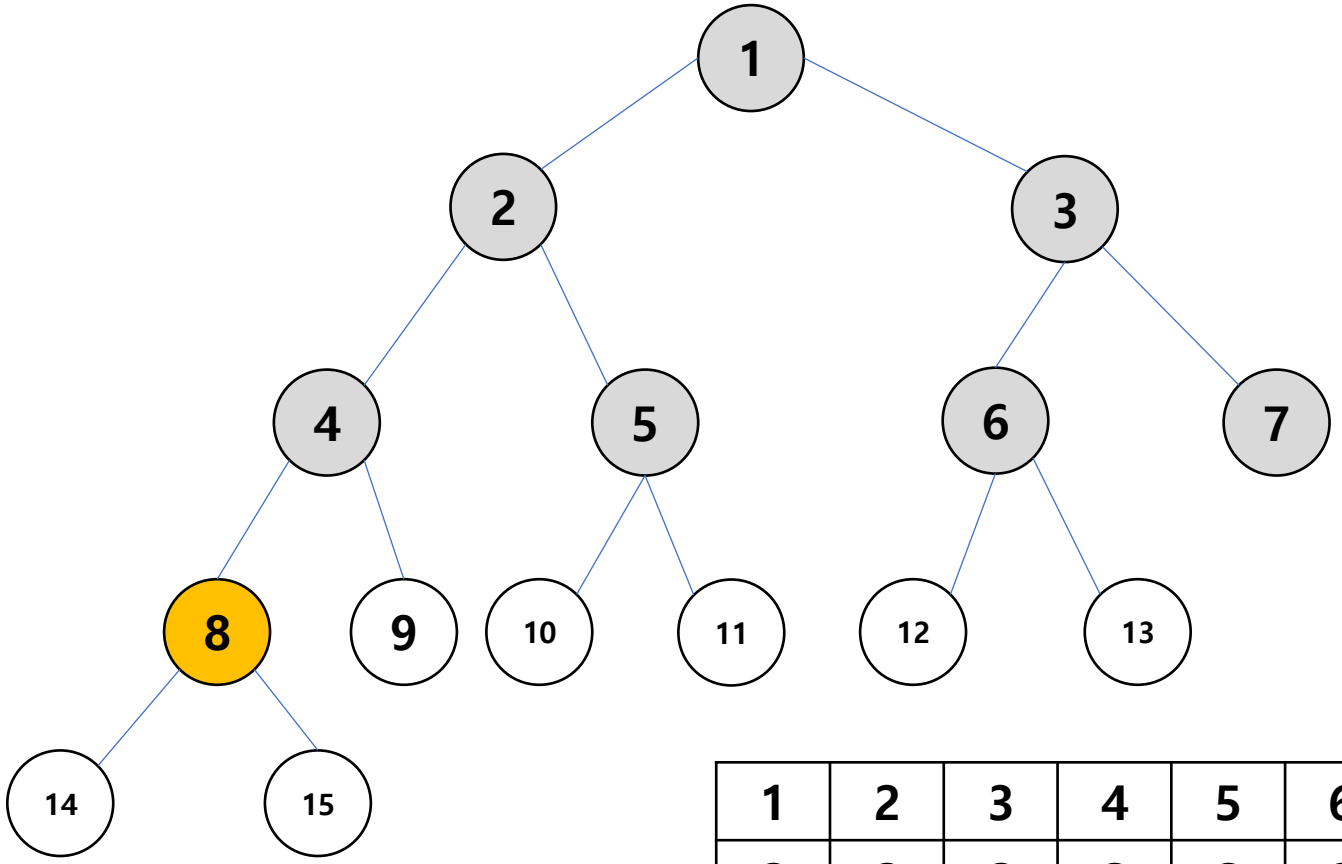
Queue

4	5	6	7				

discovered

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
O	O	O	O	O	O	O	X	X	X	X	X	X	X	X

- BFS(너비 우선 탐색)



here : 4

linked[here] : 8

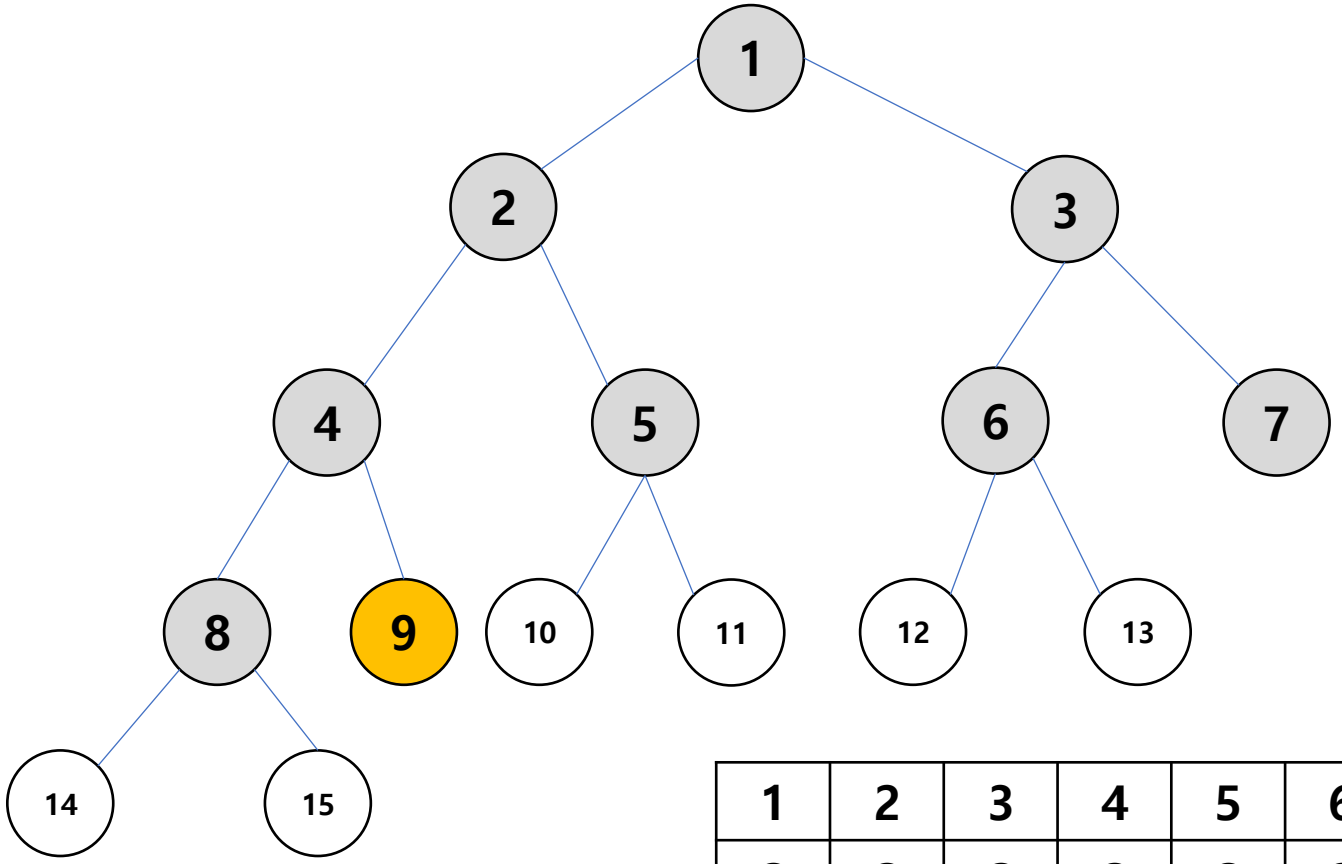
Queue

5	6	7	8				

discovered

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
O	O	O	O	O	O	O	O	X	X	X	X	X	X	X

- BFS(너비 우선 탐색)



here : 4

linked[here] : 9

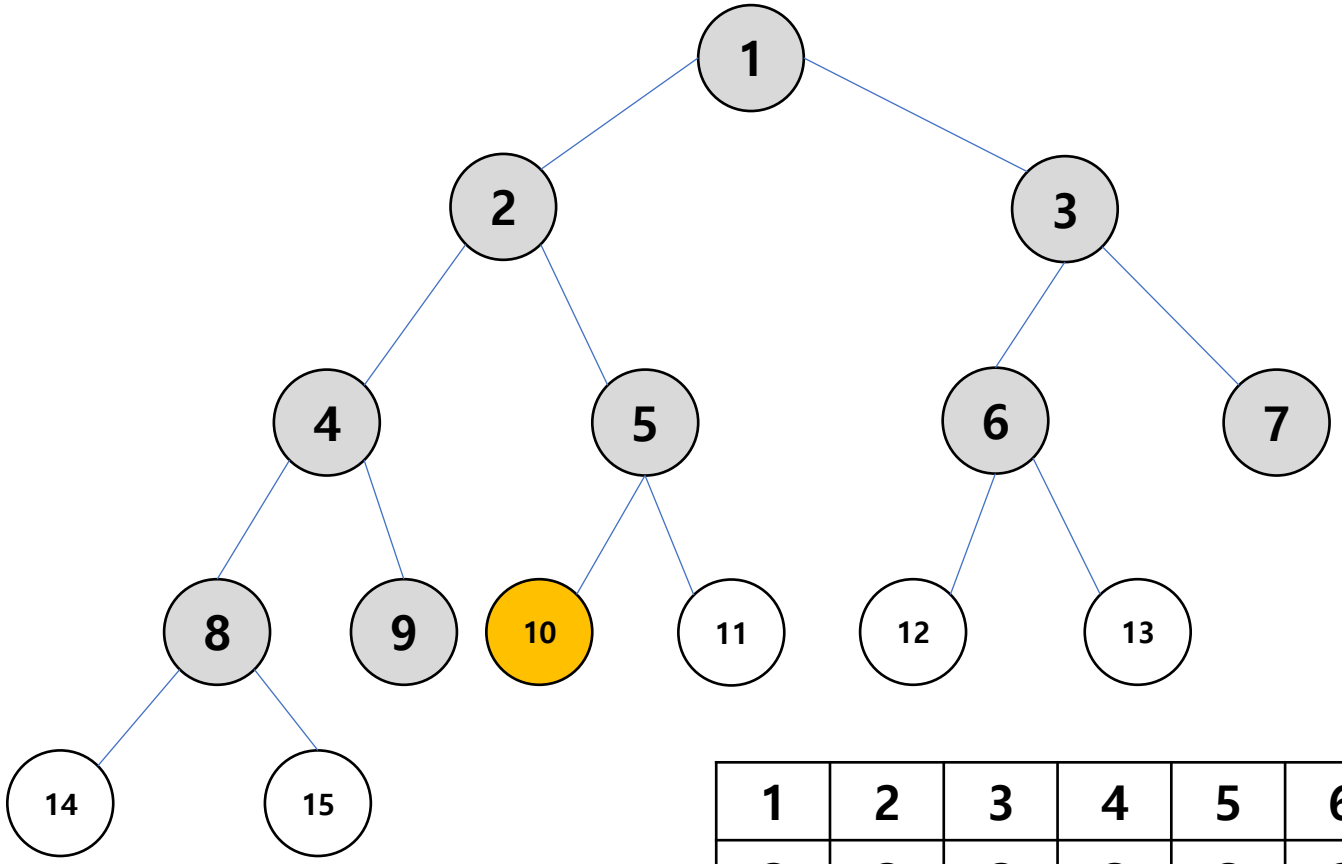
Queue

5	6	7	8	9			

discovered

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
O	O	O	O	O	O	O	O	O	X	X	X	X	X	X

- BFS(너비 우선 탐색)



here : 5

linked[here] : 10

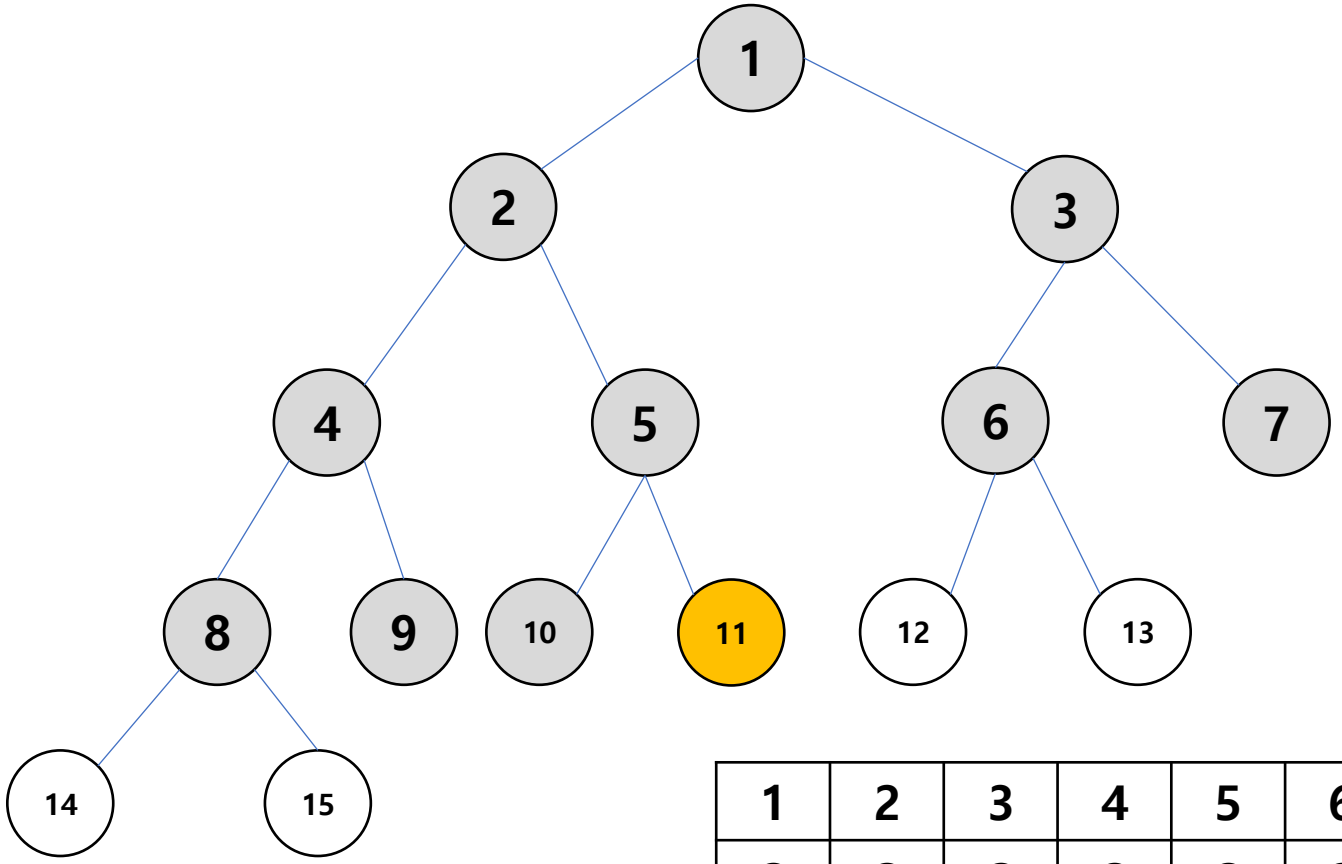
Queue

6	7	8	9	10			

discovered

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
O	O	O	O	O	O	O	O	O	O	X	X	X	X	X

- BFS(너비 우선 탐색)



here : 5

linked[here] : 11

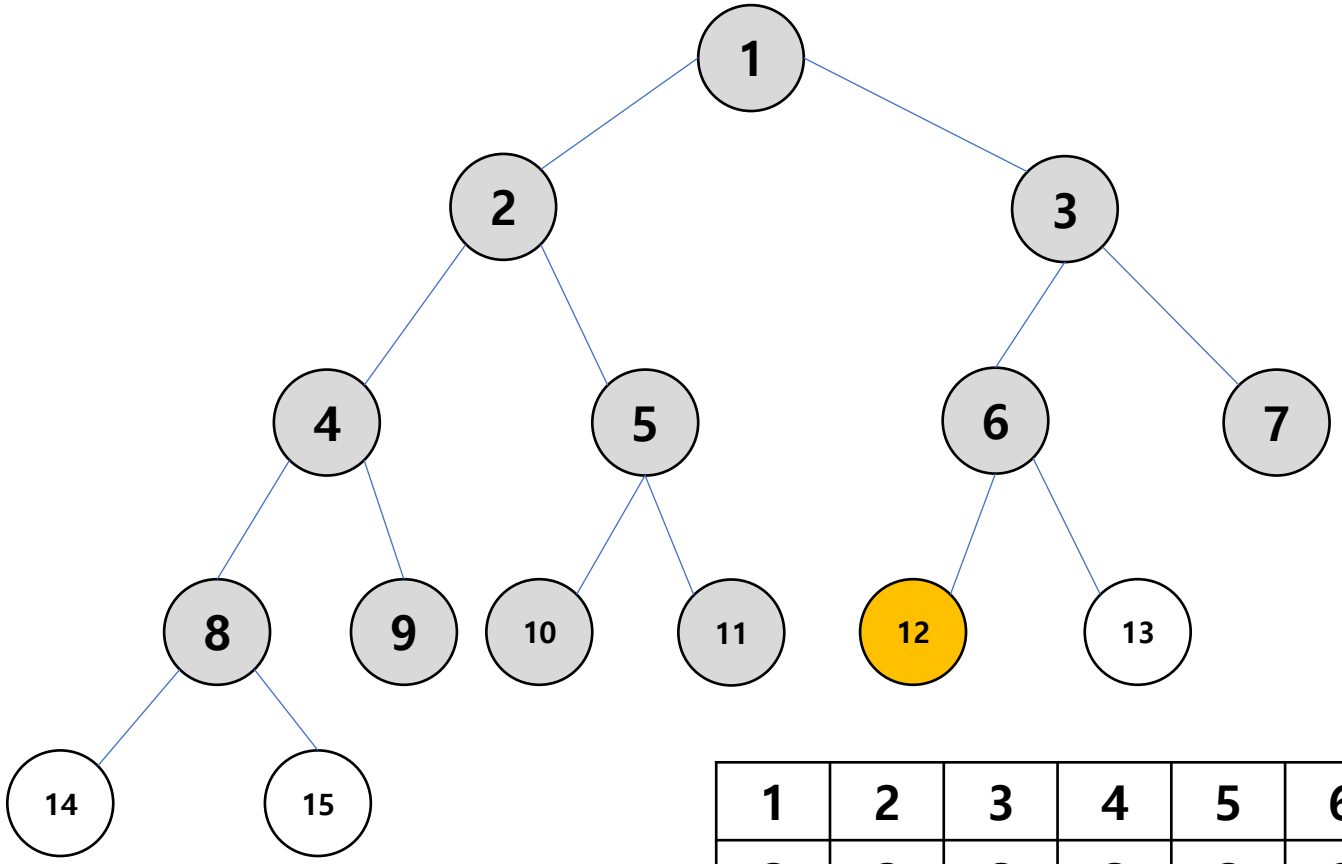
Queue

6	7	8	9	10	11		

discovered

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
O	O	O	O	O	O	O	O	O	O	O	X	X	X	X

- BFS(너비 우선 탐색)



here : 6

linked[here] : 12

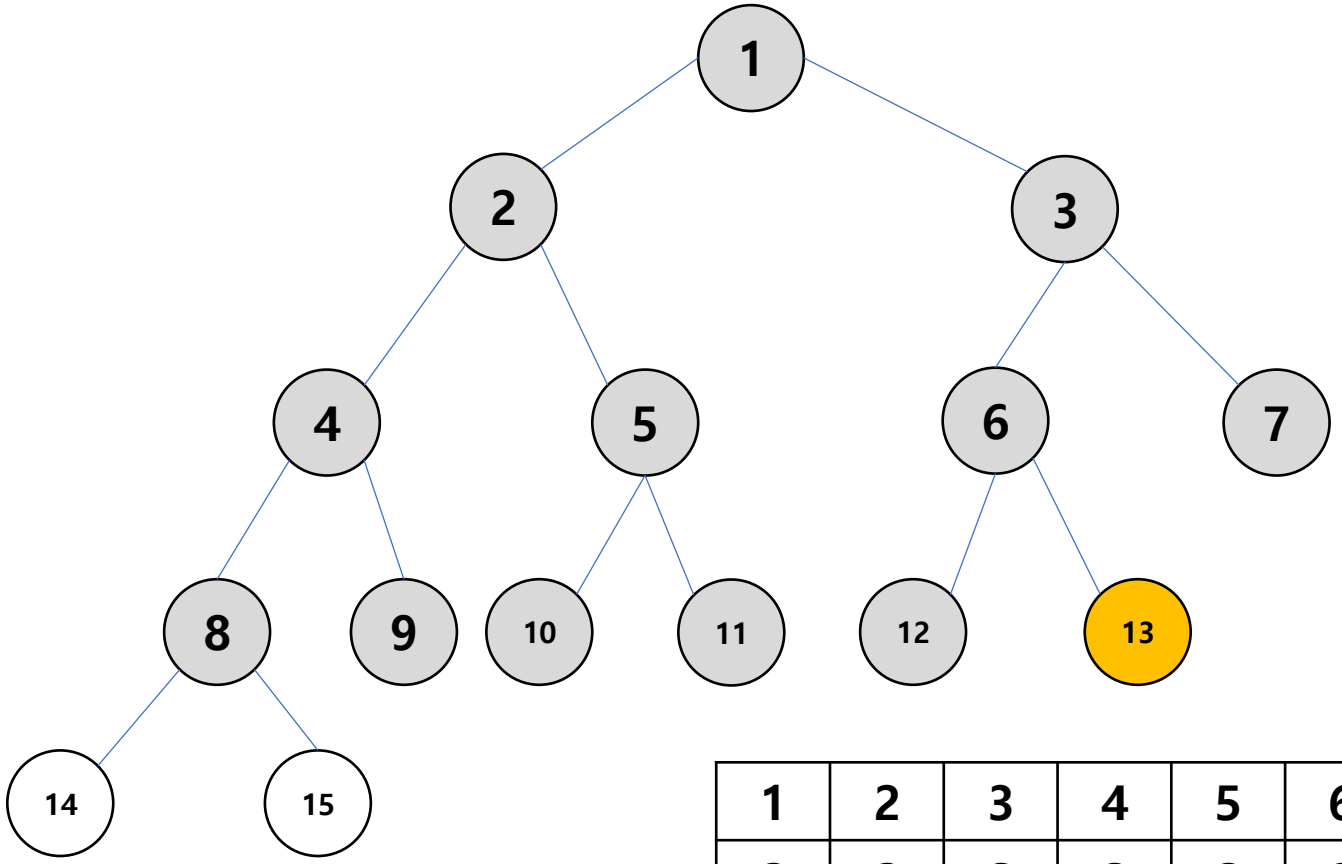
Queue

7	8	9	10	11	12		

discovered

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
O	O	O	O	O	O	O	O	O	O	O	O	X	X	X

- BFS(너비 우선 탐색)



here : 6

linked[here] : 13

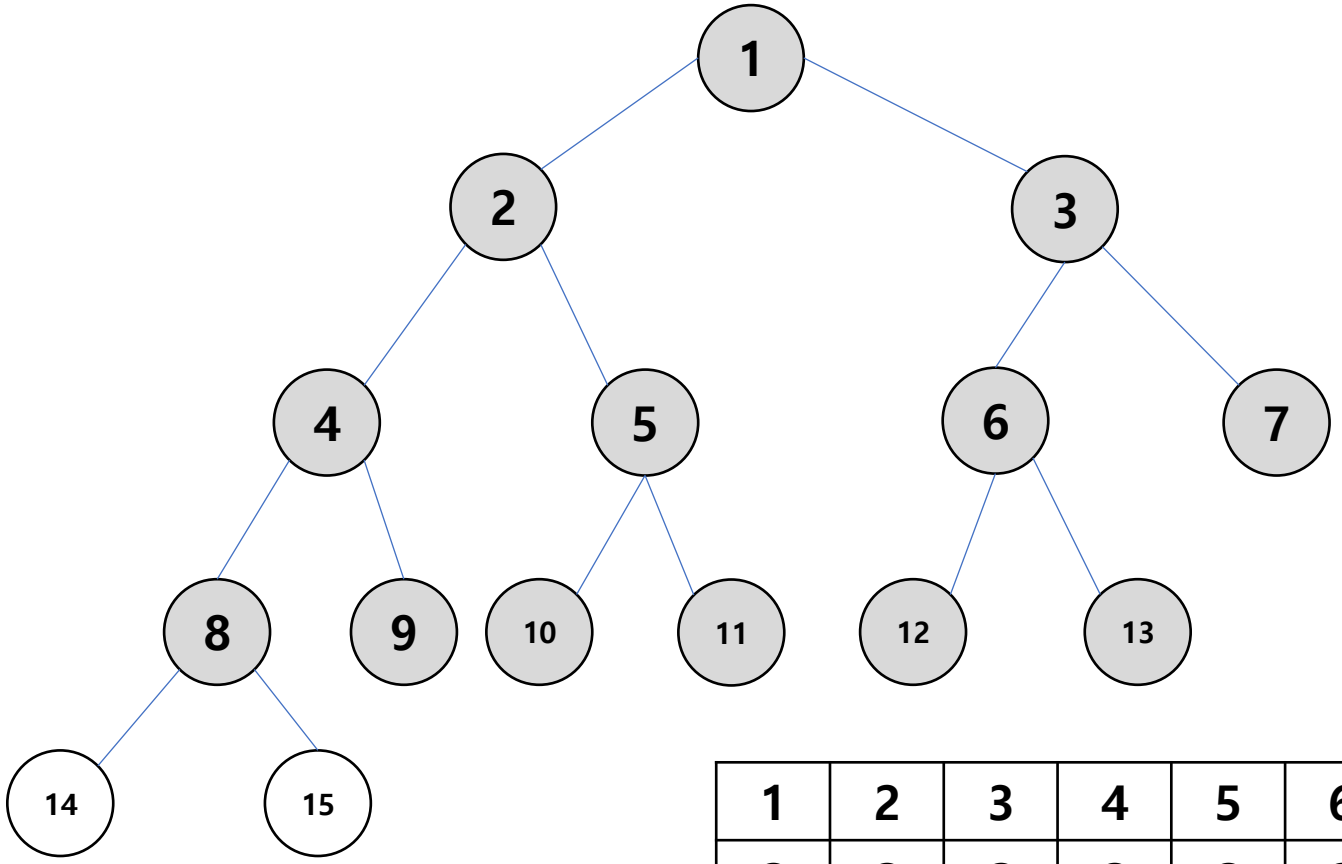
Queue

7	8	9	10	11	12	13	

discovered

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
O	O	O	O	O	O	O	O	O	O	O	O	O	X	X

- BFS(너비 우선 탐색)



here : 7

linked[here] : **X**

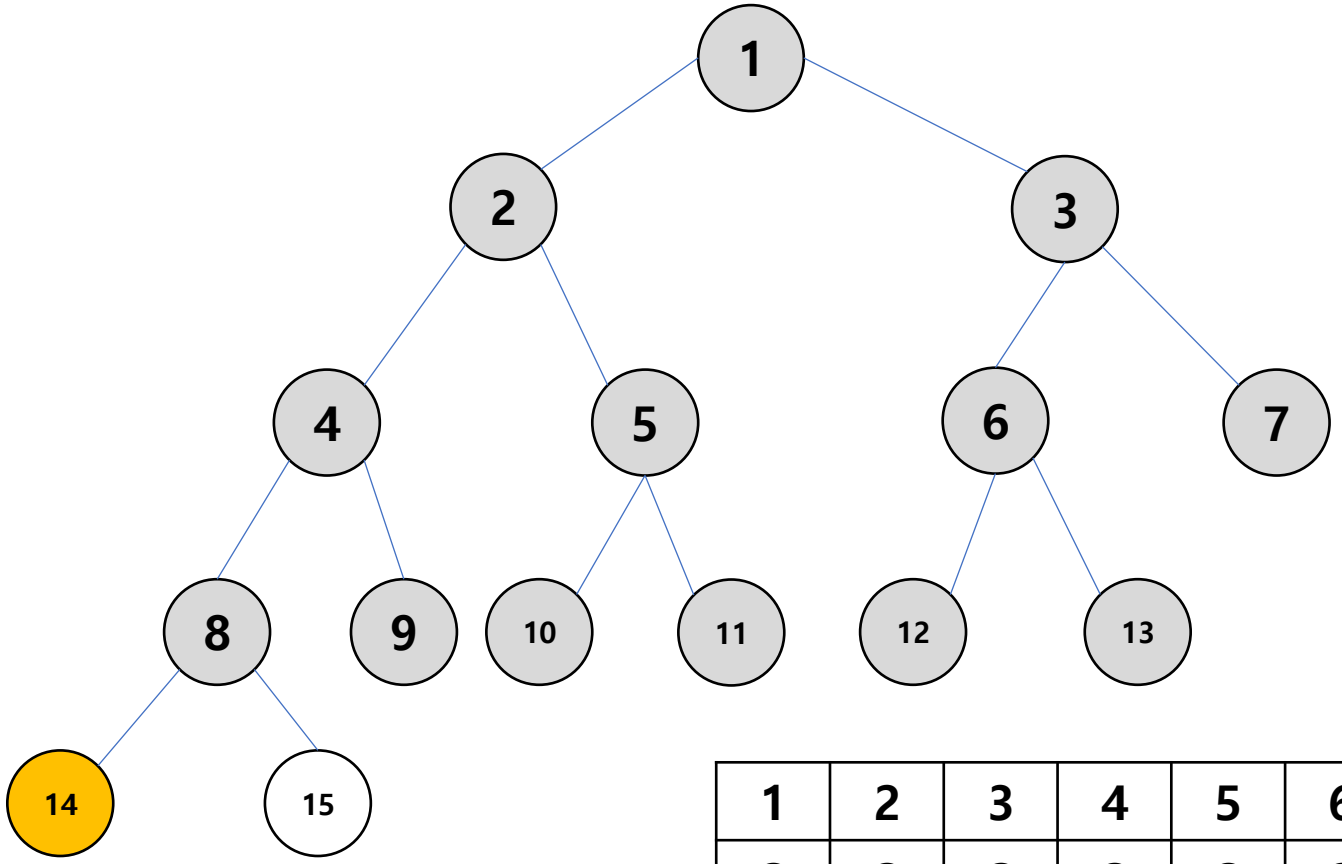
Queue

8	9	10	11	12	13		

discovered

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
O	O	O	O	O	O	O	O	O	O	O	O	O	X	X

- BFS(너비 우선 탐색)



here : 8

linked[here] : 14

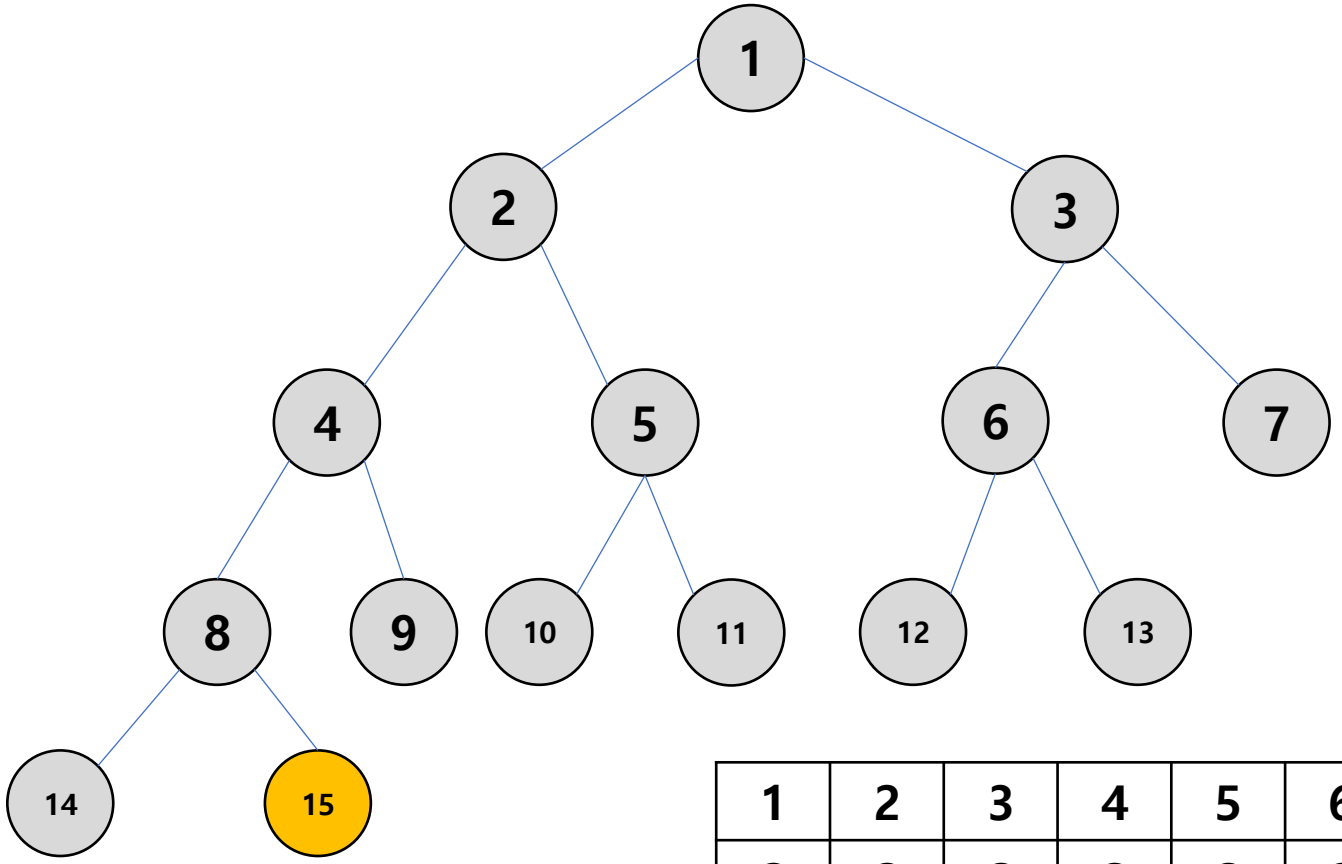
Queue

9	10	11	12	13	14		

discovered

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
O	O	O	O	O	O	O	O	O	O	O	O	O	O	X

- BFS(너비 우선 탐색)



here : 8

linked[here] : 15

Queue

9	10	11	12	13	14	15	

discovered

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
O	O	O	O	O	O	O	O	O	O	O	O	O	O	O

- BFS(너비 우선 탐색)

  가장 중요한 특징

  - 가중치가 1일 때 시작점에서 어느점까지 가는 최단 거리를 구할 수 있다.
    한 점과 연결된 모든 점들에 대해 순서대로 방문 처리를 하므로 자명하다

  - 방문 처리의 순서가 중요하다. 발견과 동시에 방문 처리

  - 방문 여부를 T/F가 아닌 int로 처리해 최단 거리를 구할 수 있다.

  - DFS로는 최단 거리를 쉽게 구할 수 없다

- BFS(너비 우선 탐색)로 최단거리 구하기

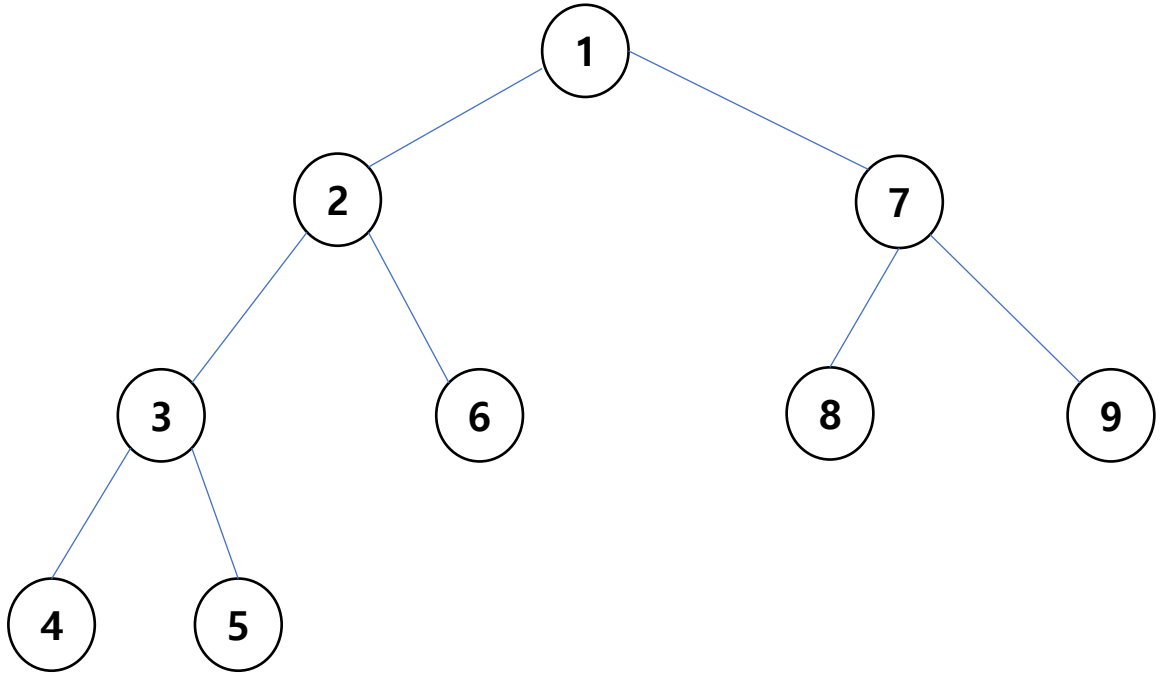
```
void BFS(int start)
    q.Enqueue(start)
    discovered[start] = true
    while(!q.empty())
        int here = q.Dequeue()
        for(int i=0; i<linked[here].size(); i++)
            if( !discovered[linked[here][i]] )
                discovered[linked[here][i]] = true
                q.Enqueue(linked[here][i])
```

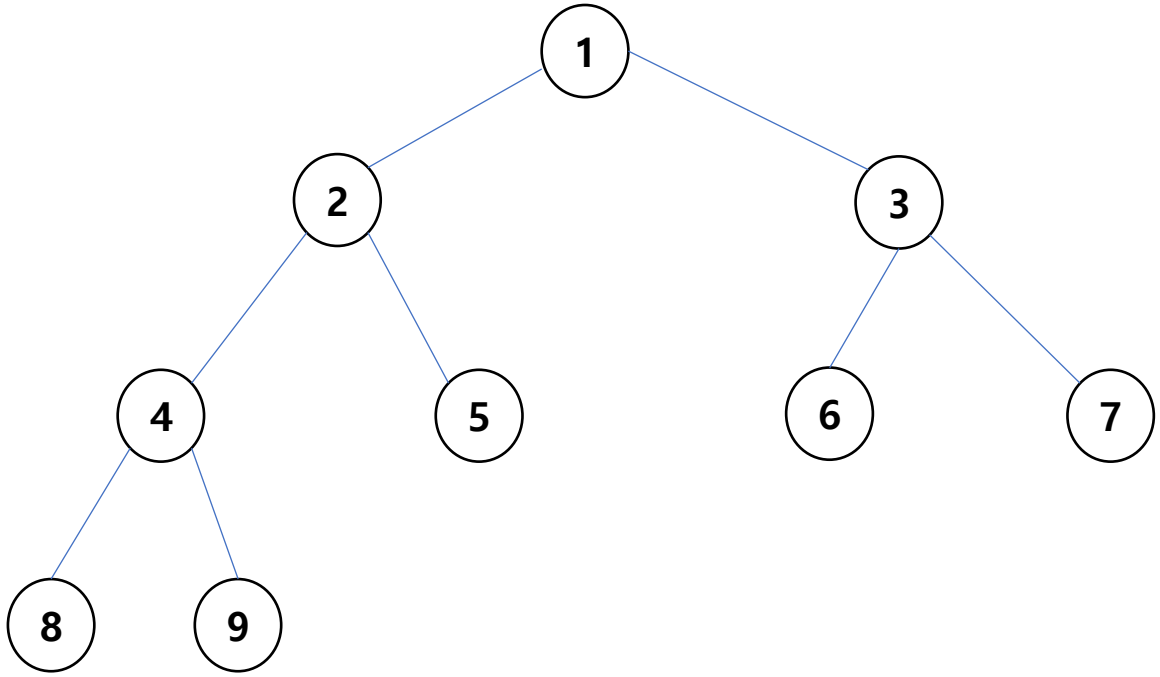
전제 조건 : 간선의 가중치가 전부 1이다

dist[x] : start 부터 x까지의 최단거리, 초기값은 -1

```
void BFS(int start)
    q.Enqueue(start)
    dist[start] = 0
    while(!q.empty())
        int here = q.Dequeue()
        for(int i=0; i<linked[here].size(); i++)
            if( dist[linked[here][i]] == -1 )
                dist[linked[here][i]] = dist[here] + 1
                q.Enqueue(linked[here][i])
```

- DFS <-> BFS





DFS를 이용하는 문제 소개

1. ABCDE : https://www.acmicpc.net/problem/13023

2. 단지번호붙이기 : https://www.acmicpc.net/problem/2667

DFS를 이용하는 문제 – ABCDE

문제 요약 :

https://www.acmicpc.net/problem/13023

- 5 ≤ N ≤ 2,000,
- M개의 줄에는 정수 a와 b가 주어지며, a와 b가 친구라는 뜻이다. (0 ≤ a, b ≤ N-1, a ≠ b)
- 사람들은 0번부터 N-1번까지 번호가 매겨져 있다.
- A-B-C-D-E 관계가 존재하면 1을 출력, 아니면 0을 출력

고려해야 할 사항 :

- 친구 정보를 저장할 자료구조

DFS를 이용하는 문제 – ABCDE

- 친구 정보 저장

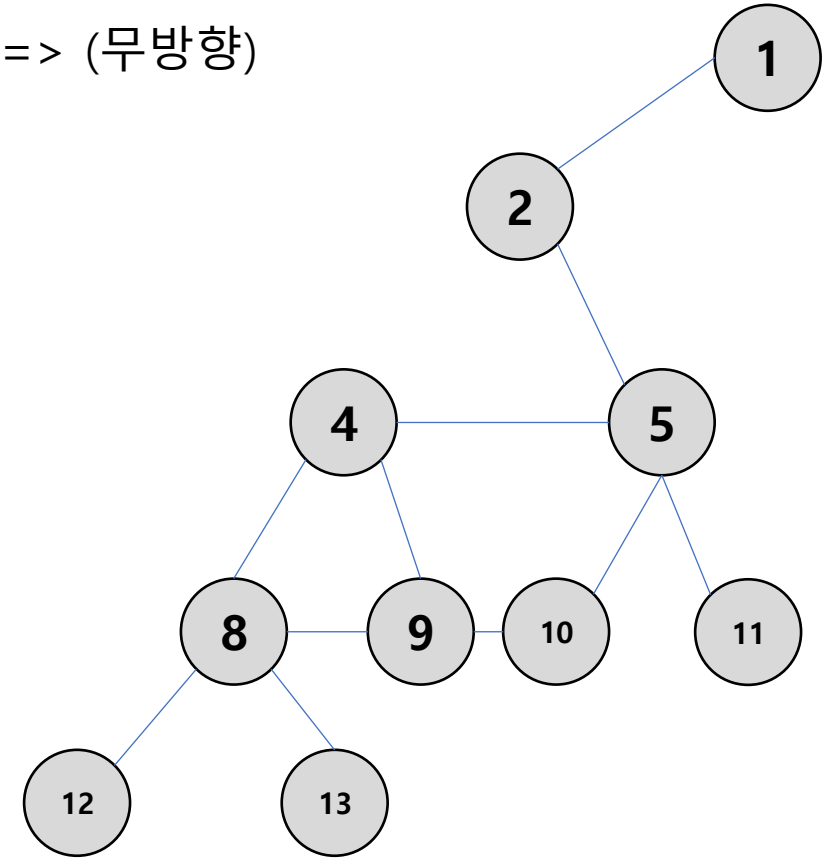
  1. 한 사람은 여러 명의 친구를 가질 수 있다. => (여러 간선)

  2. a와 b가 친구이면 b와 a도 친구이다. => (무방향)

  인접 리스트로 구현된 그래프

  vector<int> friends[N]

  연속으로 5개 이상 이어진 노드가 있으면 된다!



DFS를 이용하는 문제 – ABCDE

- 실제 구현

```
void DFS(int v, int cnt)
    if(cnt == 5)
        flag = true
        return
    visited[v] = true
    for(int i=0; i<linked[v].size(); i++)
        if( !visited[linked[v][i]] )
            DFS(linked[v][i], cnt+1)
    visited[v] = false
```

```
DFS(0, 0)
if(flag)
    cout<<1
else
    cout<<0
```

DFS를 이용하는 문제 – 단지번호붙이기

문제 요약 : https://www.acmicpc.net/problem/13023

- 5 ≤ N ≤ 25, 지도는 정사각형이다.
- 어떤 집의 상하좌우에 다른 집이 있으면 해당 집끼리는 연결된 것이라고 본다. 단, 대각선은 제외
- 연결되어있는 집들을 하나의 단지로 정의한다.
- 총 단지수와 각 단지내 집의 수를 오름차순으로 정렬하여 출력

고려해야 할 사항 :

- 단지를 이루는 집 결정
- 시작점은 어디에?

DFS를 이용하는 문제 – 단지번호붙이기

- 단지를 이루는 집 결정

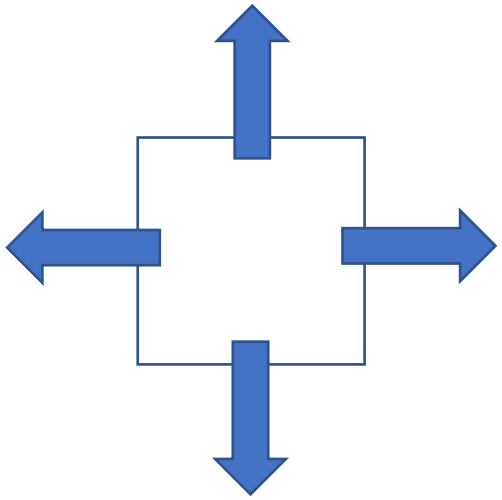
0	1	1	0	1	0	0
0	1	1	0	1	0	1
1	1	1	0	1	0	1
0	0	0	0	1	1	1
0	1	0	0	0	0	0
0	1	1	1	1	1	0
0	1	1	1	0	0	0

<그림 1>

0	1	1	0	2	0	0
0	1	1	0	2	0	2
1	1	1	0	2	0	2
0	0	0	0	2	2	2
0	3	0	0	0	0	0
0	3	3	3	3	3	0
0	3	3	3	0	0	0

<그림 2>

```
int dx[4] = {1,0,-1,0}
int dy[4] = {0,1,0,-1}
```



상하좌우에 1이있으면 linked, 0이면 !linked라고 생각

DFS를 이용하는 문제 – 단지번호붙이기

- 시작점은 어디에?

0	1	1	0	1	0	0
0	1	1	0	1	0	1
1	1	1	0	1	0	1
0	0	0	0	1	1	1
0	1	0	0	0	0	0
0	1	1	1	1	1	0
0	1	1	1	0	0	0

<그림 1>

0	1	1	0	2	0	0
0	1	1	0	2	0	2
1	1	1	0	2	0	2
0	0	0	0	2	2	2
0	3	0	0	0	0	0
0	3	3	3	3	3	0
0	3	3	3	0	0	0

<그림 2>

기준이 되는 시작점을 딱히 정할 수 없다.

- 따라서 모든 점이 시작점이 되도록 DFS 진행

한마디로 DFS를 N^2번 호출 하겠다!

DFS를 이용하는 문제 – 단지번호붙이기

- 실제 구현

0	1	1	0	1	0	0
0	1	1	0	1	0	1
1	1	1	0	1	0	1
0	0	0	0	1	1	1
0	1	0	0	0	0	0
0	1	1	1	1	1	0
0	1	1	1	0	0	0

<그림 1>

0	1	1	0	2	0	0
0	1	1	0	2	0	2
1	1	1	0	2	0	2
0	0	0	0	2	2	2
0	3	0	0	0	0	0
0	3	3	3	3	3	0
0	3	3	3	0	0	0

<그림 2>

```
for(int i=0; i<N; i++)
    for(int j=0; j<N; j++)
        if(!visited[i][j] && board[i][j] == 1)
            ret = DFS(i,j)
```

단지내에 있는 집의 개수가 담긴다

```
int DFS(int y, int x)
    int ret = 1
    visited[y][x] = true
    for(int i=0; i<4; i++)
        if(inRange && !visited &&
        board[y+dy[i]][x+dx[i]] == 1)
            ret += DFS(y+dy[i], x+dx[i])
    return ret
```

항상 inRange 먼저

BFS를 이용하는 문제 소개

1. 단지번호붙이기 : https://www.acmicpc.net/problem/2667

2. 미로 탐색 : https://www.acmicpc.net/problem/2178

3. 토마토 : https://www.acmicpc.net/problem/7576

BFS를 이용하는 문제 – 단지번호붙이기

문제 요약 : https://www.acmicpc.net/problem/13023

- 5 ≤ N ≤ 25, 지도는 정사각형이다.
- 어떤 집의 상하좌우에 다른 집이 있으면 해당 집끼리는 연결된 것이라고 본다. 단, 대각선은 제외
- 연결되어있는 집들을 하나의 단지로 정의한다.
- 총 단지수와 각 단지내 집의 수를 오름차순으로 정렬하여 출력

고려해야 할 사항 :

- 단지를 이루는 집 결정
- 시작점은 어디에?

BFS를 이용하는 문제 – 단지번호붙이기

- 단지를 이루는 집 결정

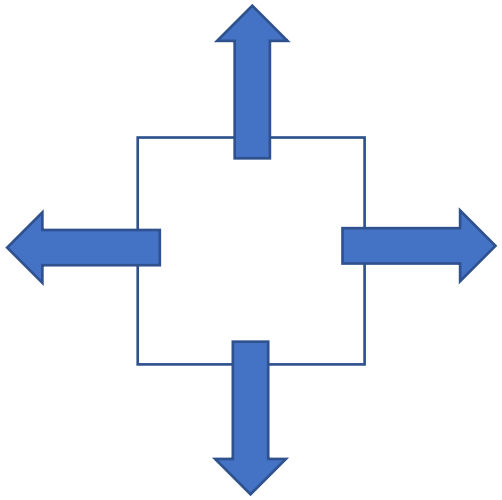
0	1	1	0	1	0	0
0	1	1	0	1	0	1
1	1	1	0	1	0	1
0	0	0	0	1	1	1
0	1	0	0	0	0	0
0	1	1	1	1	1	0
0	1	1	1	0	0	0

<그림 1>

0	1	1	0	2	0	0
0	1	1	0	2	0	2
1	1	1	0	2	0	2
0	0	0	0	2	2	2
0	3	0	0	0	0	0
0	3	3	3	3	3	0
0	3	3	3	0	0	0

<그림 2>

```
int dx[4] = {1,0,-1,0}
int dy[4] = {0,1,0,-1}
```



상하좌우에 1이면 linked, 0이면 !linked라고 생각

BFS를 이용하는 문제 – 단지번호붙이기

- 시작점은 어디에?

0	1	1	0	1	0	0
0	1	1	0	1	0	1
1	1	1	0	1	0	1
0	0	0	0	1	1	1
0	1	0	0	0	0	0
0	1	1	1	1	1	0
0	1	1	1	0	0	0

<그림 1>

0	1	1	0	2	0	0
0	1	1	0	2	0	2
1	1	1	0	2	0	2
0	0	0	0	2	2	2
0	3	0	0	0	0	0
0	3	3	3	3	3	0
0	3	3	3	0	0	0

<그림 2>

기준이 되는 시작점을 딱히 정할 수 없다.

- 따라서 모든 점이 시작점이 되도록 BFS 진행

한마디로 BFS를 N^2번 호출 하겠다!

BFS를 이용하는 문제 – 단지번호붙이기

- 실제 구현

0	1	1	0	1	0	0
0	1	1	0	1	0	1
1	1	1	0	1	0	1
0	0	0	0	1	1	1
0	1	0	0	0	0	0
0	1	1	1	1	1	0
0	1	1	1	0	0	0

<그림 1>

0	1	1	0	2	0	0
0	1	1	0	2	0	2
1	1	1	0	2	0	2
0	0	0	0	2	2	2
0	3	0	0	0	0	0
0	3	3	3	3	3	0
0	3	3	3	0	0	0

<그림 2>

```
for(int i=0; i<N; i++)
    for(int j=0; j<N; j++)
        if(!visited[i][j] && board[i][j] == 1)
            ret = BFS(i,j)
```

```
int BFS(int y, int x)
    visited[y][x] = true
    q.Enqueue(y,x)
    int ret = 1
    while(!q.empty())
        for(int i=0; i<4; i++)
            if(inRange && !visited &&
            board[y+dy[i]][x+dx[i]] == 1)
                ret++
                visited[y+dy[i]][x+dx[i]] = true
                q.Enqueue(y+dy[i], x+dx[i])
    return ret
```

BFS를 이용하는 문제 – 미로 탐색

문제 요약 : https://www.acmicpc.net/problem/2178

- 2 ≤ N, M ≤ 100
- 1은 이동할 수 있는 칸, 0은 이동할 수 없는 칸
- 인접한 칸으로만 이동할 수 있다
- (1,1) 에서 (N,M)까지 이동하는데 지나야하는 칸의 개수의 최소값

고려해야 할 사항 :

- BFS로 구하는 최단거리

BFS를 이용하는 문제 – 미로 탐색

- BFS로 구하는 최단거리

1	0	1	1	1	1
1	0	1	0	1	0
1	0	1	0	1	1
1	1	1	0	1	1

초기 dist의 값은 -1로 설정

```
void BFS()
    dist[0][0] = 0
    q.Enqueue(0,0)

    while(!q.empty())
        pair<int, int> info = q.Dequeue()
        for(int i=0; i<4; i++)
            if(inRange && dist[y+dy[i]][x+dx[i]] == -1
              && board[y+dy[i]][x+dx[i]] == 1)
                dist[y+dy[i]][x+dx[i]] = dist[y][x] + 1
                q.Enqueue(y+dy[i], x+dx[i])

    return ret
```

BFS를 이용하는 문제 – 토마토

문제 요약 : https://www.acmicpc.net/problem/7576

- 2 ≤ M, N ≤ 1,000
- 1은 익은 토마토, 0은 익지 않은 토마토, -1은 토마토가 들어있지 않은 칸
- 토마토가 하나 이상 있는 경우만 입력으로 주어진다
- 토마토가 모두 익을 때까지 걸리는 최소 날짜를 출력
- 모든 토마토가 익어있는 상태면 0, 토마토가 모두 익지 못하는 상황이면 -1을 출력

고려해야 할 사항 :

- 시작점이 한 곳이 아닐 수도 있다
- -1인 칸으로 이동 할 수 없다

BFS를 이용하는 문제 – 토마토

- 시작점이 한 곳이 아닐 수도 있다

1	-1	0	0	0	0
0	-1	0	0	0	0
0	0	0	0	-1	0
0	0	0	0	-1	1

Queue에 1의 위치를 다 넣어준다

그리고 BFS 시작

-1인 칸으로는 이동 할 수 없다.

BFS를 이용하는 문제 – 토마토

- dist 계산

0	X	-1	-1	-1	-1
-1	X	-1	-1	-1	-1
-1	-1	-1	-1	X	-1
-1	-1	-1	-1	X	0

BFS를 이용하는 문제 – 토마토

- dist 계산

0	X	-1	-1	-1	-1
1	X	-1	-1	-1	-1
-1	-1	-1	-1	X	1
-1	-1	-1	-1	X	0

BFS를 이용하는 문제 – 토마토

- dist 계산

0	X	-1	-1	-1	-1
1	X	-1	-1	-1	2
2	-1	-1	-1	X	1
-1	-1	-1	-1	X	0

BFS를 이용하는 문제 – 토마토

- dist 계산

0	X	-1	-1	-1	3
1	X	-1	-1	3	2
2	3	-1	-1	X	1
3	-1	-1	-1	X	0

BFS를 이용하는 문제 – 토마토

- dist 계산

0	X	-1	-1	4	3
1	X	-1	4	3	2
2	3	4	-1	X	1
3	4	-1	-1	X	0

BFS를 이용하는 문제 – 토마토

- dist 계산

0	X	-1	5	4	3
1	X	5	4	3	2
2	3	4	5	X	1
3	4	5	-1	X	0

BFS를 이용하는 문제 – 토마토

- dist 계산

0	X	6	5	4	3
1	X	5	4	3	2
2	3	4	5	X	1
3	4	5	6	X	0

최소 6일이 걸린다

BFS를 이용하는 문제 – 토마토

- 예외처리

1	-1	0	0	-1	0
0	-1	0	0	-1	0
0	-1	0	0	-1	0
0	-1	0	0	-1	1

- BFS 진행 후 전체 배열에 0이 있는지 check

도전 문제!

1. 연구소 : https://www.acmicpc.net/problem/14502

2. 상범 빌딩 : https://www.acmicpc.net/problem/6593

3. 치즈 : https://www.acmicpc.net/problem/2636

4. 중량 제한 : https://www.acmicpc.net/problem/1939

자유로운 질문 및 토의!