25-1 이니로
알고리즘 멘토링

멘토 - 김수성

파일 합치기 3 / 13975

백준 13975 / https://www.acmicpc.net/problem/13975

문제

소설가인 김대전은 소설을 여러 장(chapter)으로 나누어 쓰는데, 각 장은 각각 다른 파일에 저장하곤 한다. 소설의 모든 장을 쓰고 나서는 각 장이 쓰여진 파일을 합쳐서 최종적으로 소설의 완성본이 들어있는 한 개의 파일을 만든다. 이 과정에서 두 개의 파일을 합쳐서 하나의 임시파일을 만들고, 이 임시파일이나 원래의 파일을 계속 두 개씩 합쳐서 파일을 합쳐나가고, 최종적으로는 하나의 파일로 합친다. 두 개의 파일을 합칠 때 필요한 비용(시간 등)이 두 파일 크기의 합이라고 가정할 때, 최종적인 한 개의 파일을 완성하는데 필요한 비용의 총 합을 계산하시오.

예를 들어, C1, C2, C3, C4가 네 개의 장을 수록하고 있는 파일이고, 파일 크기가 각각 40, 30, 30, 50 이라고 하자. 이 파일들을 합치는 과정에서, 먼저 C2와 C3를 합쳐서 임시파일 X1을 만든다. 이때 비용 60이 필요하다. 그 다음으로 C1과 X1을 합쳐 임시파일 X2를 만들면 비용 100이 필요하다. 최종적으로 X2와 C4를 합쳐 최종파일을 만들면 비용 150이 필요하다. 따라서, 최종의 한 파일을 만드는데 필요한 비용의 합은 60+100+150=310 이다. 다른 방법으로 파일을 합치면 비용을 줄일 수 있다. 먼저 C1과 C2를 합쳐 임시파일 Y1을 만들고, C3와 C4를 합쳐 임시파일 Y2를 만들고, 최종적으로 Y1과 Y2를 합쳐 최종파일을 만들 수 있다. 이때 필요한 총 비용은 70+80+150=300 이다.

소설의 각 장들이 수록되어 있는 파일의 크기가 주어졌을 때, 이 파일들을 하나의 파일로 합칠 때 필요한 최소비용을 계산하는 프로그램을 작성하시오.

입력

프로그램은 표준 입력에서 입력 데이터를 받는다. 프로그램의 입력은 T개의 테스트 데이터로 이루어져 있는데, T는 입력의 맨 첫 줄에 주어진다.각 테스트 데이터는 두 개의 행으로 주어지는데, 첫 행에는 소설을 구성하는 장의 수를 나타내는 양의 정수 K (3 ≤ K ≤ 1,000,000)가 주어진다. 두 번째 행에는 1장부터 K장까지 수록한 파일의 크기를 나타내는 양의 정수 K개가 주어진다. 파일의 크기는 10,000을 초과하지 않는다.

출력

프로그램은 표준 출력에 출력한다. 각 테스트 데이터마다 정확히 한 행에 출력하는데, 모든 장을 합치는데 필요한 최소비용을 출력한다.

파일 합치기 3 / 13975

파일 두개를 합치면 하나의 파일이 되는데
이 때 두 파일의 합만큼 비용이 들음

이 합을 최소화 해야 함

예제 입력 1 복사

```
2
4
40 30 30 50
15
1 21 3 4 5 35 5 4 3 5 98 21 14 17 32
```

예제 출력 1 복사

```
300
826
```

파일 합치기 3 / 13975

첫 번째 테스트 케이스
40 30 30 50
60 40 50 -> 60의 비용
90 60 -> 90의 비용
150 -> 150의 비용　60 + 90 + 150 = 300

예제 입력 1 복사

```
2
4
40 30 30 50
15
1 21 3 4 5 35 5 4 3 5 98 21 14 17 32
```

예제 출력 1 복사

```
300
826
```

파일 합치기 3 / 13975

N이 4일 때 각 배열의 값을 A1, A2, A3, A4라고 해보자
처음에 A1, A2를 합치면 A1 + A2의 비용이 들음
그리고 배열에 A1 + A2가 생김

이 값을 또 사용하면 A1 + A2만큼의 비용이 더 들음

예제 입력 1 복사

```
2
4
40 30 30 50
15
1 21 3 4 5 35 5 4 3 5 98 21 14 17 32
```

예제 출력 1 복사

```
300
826
```

파일 합치기 3 / 13975

이 값을 또 사용하면 A1 + A2만큼의 비용이 더 들음
-> 합친 파일을 사용하면 그 파일을 구성하는 파일들의 비용이 들음
-> 일단 작은 파일부터 합치는 방법을 사용해야 비용 최소화 가능

예제 입력 1 복사

```
2
4
40 30 30 50
15
1 21 3 4 5 35 5 4 3 5 98 21 14 17 32
```

예제 출력 1 복사

```
300
826
```

파일 합치기 3 / 13975

일단 작은 파일부터 합치는 방법을 사용해야 비용 최소화 가능
-> 최소 우선순위 큐를 사용하자

예제 입력 1 복사

```
2
4
40 30 30 50
15
1 21 3 4 5 35 5 4 3 5 98 21 14 17 32
```

예제 출력 1 복사

```
300
826
```

파일 합치기 3 / 13975

4
40 30 30 50
우선 우선순위 큐에 모든 값을 넣음

우선순위 큐

비용
0

파일 합치기 3 / 13975

4

40 30 30 50

먼저 합친 값은 나중에도 영향을 줌

-> 작은 값부터 합치는게 이득

우선순위 큐

비용

0

파일 합치기 3 / 13975

4
40 30 30 50
제일 작은 두 값을 합치고 다시 우선순위 큐에 넣음

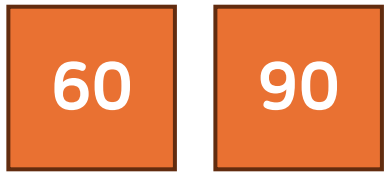
우선순위 큐

비용
60

파일 합치기 3 / 13975

4
40 30 30 50
우선순위 큐의 크기가 1이 될 때까지 반복

우선순위 큐



비용
150

파일 합치기 3 / 13975

4
40 30 30 50
우선순위 큐의 크기가 1이 될 때까지 반복

우선순위 큐

150

비용
300

파일 합치기 3 / 13975

C++

```cpp
#include <iostream>
#include <queue>
using namespace std;
using ll = long long;

const ll MAX = 1010101;
ll n, a[MAX];
```

```cpp
int main(){
    ios::sync_with_stdio(0); // fastio
    cin.tie(0), cout.tie(0); // fastio

    ll t; cin >> t;
    while(t--) solve();

    return 0;
}
```

```cpp
void solve(){
    cin >> n;
    for(int i = 1;i <= n;i++) cin >> a[i];

    // 최소 우선순위 큐
    priority_queue <ll, vector<ll>, greater<ll>> pq;
    // 먼저 모든 값을 우선순위 큐에 넣음
    for(int i = 1;i <= n;i++) pq.push(a[i]);

    ll result = 0;
    // 우선순위 큐의 크기가 1이 될 때 까지 반복
    while(pq.size() > 1){
        // 제일 작은 수
        ll fi = pq.top(); pq.pop();
        // 두 번째로 작은 수
        ll se = pq.top(); pq.pop();

        // 합쳐서 우선순위 큐에 다시 넣음
        pq.push(fi + se);
        // 정답에 추가
        result += fi + se;
    }

    cout << result << "\n";
}
```

파일 합치기 3 / 13975

Python

```python
import sys
import heapq
input = sys.stdin.readline

t = int(input().rstrip())
for _ in range(t):
    n = int(input().rstrip())
    a = list(map(int, input().rstrip().split()))
    result = 0

    # 최소 우선순위 큐
    pq = []
    # 먼저 모든 값을 우선순위 큐에 넣음
    for i in a:
        heapq.heappush(pq, i)

    # 우선순위 큐의 크기가 1이 될 때 까지 반복
    while len(pq) > 1:
        # 제일 작은 수
        fi = heapq.heappop(pq)
        # 두 번째로 작은 수
        se = heapq.heappop(pq)

        # 합쳐서 우선순위 큐에 다시 넣음
        heapq.heappush(pq, fi + se)
        # 정답에 추가
        result += fi + se

    print(result)
```

질문?

가운데를 말해요 / 1655

백준 1655 / https://www.acmicpc.net/problem/1655

문제

백준이는 동생에게 "가운데를 말해요" 게임을 가르쳐주고 있다. 백준이가 정수를 하나씩 외칠때마다 동생은 지금까지 백준이가 말한 수 중에서 중간값을 말해야 한다. 만약, 그 동안 백준이가 외친 수의 개수가 짝수개라면 중간에 있는 두 수 중에서 작은 수를 말해야 한다.

예를 들어 백준이가 동생에게 1, 5, 2, 10, -99, 7, 5를 순서대로 외쳤다고 하면, 동생은 1, 1, 2, 2, 2, 2, 5를 차례대로 말해야 한다. 백준이가 외치는 수가 주어졌을 때, 동생이 말해야 하는 수를 구하는 프로그램을 작성하시오.

입력

첫째 줄에는 백준이가 외치는 정수의 개수 N이 주어진다. N은 1보다 크거나 같고, 100,000보다 작거나 같은 자연수이다. 그 다음 N줄에 걸쳐서 백준이가 외치는 정수가 차례대로 주어진다. 정수는 -10,000보다 크거나 같고, 10,000보다 작거나 같다.

출력

한 줄에 하나씩 N줄에 걸쳐 백준이의 동생이 말해야 하는 수를 순서대로 출력한다.

가운데를 말해요 / 1655

N이 주어지고 N개의 수가 주어짐
1~N번까지의 수가 있을 때 가운데 수를 출력

예제 입력 1 복사

```
7
1
5
2
10
-99
7
5
```

예제 출력 1 복사

```
1
1
2
2
2
2
5
```

가운데를 말해요 / 1655

나이브하게 푸는 법
-> 각 수가 들어올 때마다 정렬을 해서 가운데 인덱스를 출력
-> 정렬하는데 O(N log N), N번 해야 함
-> O(N^2 log N)

N이 1e6까지 들어옴
-> 시간 안에 절대 안 됨

가운데를 말해요 / 1655

빠르게 푸는 법은 없을까?
-> 우선순위 큐가 항상 중간 값을 가리키게 하면
-> 삽입 O(log N), 반환 O(1)

-> 우선순위 큐를 두 개 사용해보자

가운데를 말해요 / 1655

최대 우선순위 큐에 절반,
최소 우선순위 큐에 절반 삽입

그리고 항상 최대 우선순위 큐의 값이
최소 우선순위 큐의 값보다 작게 유지
-> 항상 최대 우선순위 큐의 최댓값이 중간 값이 됨

가운데를 말해요 / 1655

최대 우선순위 큐에 절반,
최소 우선순위 큐에 절반 삽입

7
1 5 2 10 -99 7 5

예제 입력 1 복사

```
7
1
5
2
10
-99
7
5
```

예제 출력 1 복사

```
1
1
2
2
2
2
5
```

가운데를 말해요 / 1655

최대 우선순위 큐에 절반,
최소 우선순위 큐에 절반 삽입

7 출력

1 5 2 10 -99 7 5 1 1 2 2 2 2 5

최대 우선순위 큐

최소 우선순위 큐

가운데를 말해요 / 1655

최대 우선순위 큐에 절반,
최소 우선순위 큐에 절반 삽입

7 출력

1 5 2 10 -99 7 5 1 1 2 2 2 2 5

최대 우선순위 큐

최소 우선순위 큐

가운데를 말해요 / 1655

최대 우선순위 큐에 절반,
최소 우선순위 큐에 절반 삽입

7 출력

1 5 2 10 -99 7 5 1 1 2 2 2 2 5

최대 우선순위 큐

최소 우선순위 큐

가운데를 말해요 / 1655

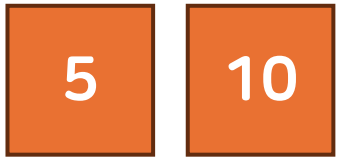
최대 우선순위 큐에 절반,
최소 우선순위 큐에 절반 삽입

7 출력

1 5 2 10 -99 7 5 1 1 2 2 2 2 5

최대 우선순위 큐

최소 우선순위 큐



가운데를 말해요 / 1655

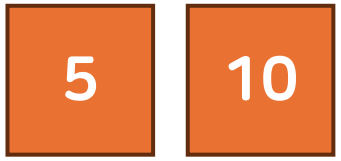
최대 우선순위 큐에 절반,
최소 우선순위 큐에 절반 삽입

7
1 5 2 10 -99 7 5

출력
1 1 2 2 2 2 5

최대 우선순위 큐

최소 우선순위 큐



가운데를 말해요 / 1655

최대 우선순위 큐에 절반,
최소 우선순위 큐에 절반 삽입

7 출력

1 5 2 10 -99 7 5 1 1 2 2 2 2 5

최대 우선순위 큐

최소 우선순위 큐

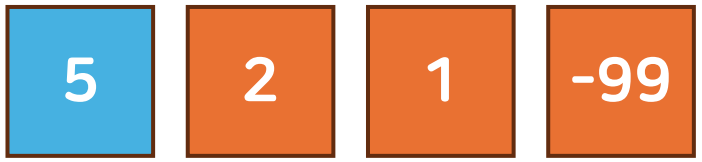
가운데를 말해요 / 1655

최대 우선순위 큐에 절반,
최소 우선순위 큐에 절반 삽입

7 출력

1 5 2 10 -99 7 5 1 1 2 2 2 2 5

최대 우선순위 큐



최소 우선순위 큐

가운데를 말해요 / 1655

예제에서는 나오지 않았지만
최대 우선순위 큐의 값이
최소 우선순위 큐의 값보다 클 수 도 있음

항상 최대 우선순위 큐의 값이
최소 우선순위 큐의 값보다 작아야 함
-> 이 때는 서로의 값을 스왑 해줘야 함

가운데를 말해요 / 1655

C++

```cpp
#include <iostream>
#include <queue>
using namespace std;
using ll = long long;

const ll MAX = 1010101;
ll n, a[MAX];
priority_queue <ll> pq1;
priority_queue <ll, vector<ll>, greater<ll>> pq2;
```

```cpp
int main(){
    ios::sync_with_stdio(0); // fastio
    cin.tie(0), cout.tie(0); // fastio

    cin >> n;
    for(int i = 1;i <= n;i++){
        ll x; cin >> x;
        // 홀수 번째에는 최대 우선순위 큐에 삽입
        if(i % 2) pq1.push(x);
        // 짝수 번째에는 최소 우선순위 큐에 삽입
        else pq2.push(x);

        // 최대 우선순위 큐의 값이 최소 우선순위 큐의 값보다 크면
        if(!pq2.empty() && pq1.top() > pq2.top()){
            ll t1 = pq1.top();
            ll t2 = pq2.top();
            // 서로 스왑해 줌
            pq1.pop(); pq2.pop();
            pq1.push(t2); pq2.push(t1);
        }

        // 최대 우선순위 큐의 최댓값이 중간 값
        cout << pq1.top() << "\n";
    }
}
```

가운데를 말해요 / 1655

Python

```python
import sys
import heapq
input = sys.stdin.readline

pq1 = []
pq2 = []
```

```python
n = int(input().rstrip())
for i in range(n):
    x = int(input().rstrip())

    # 인덱스가 0 부터 시작
    # 홀수 번째에는 최대 우선순위 큐에 삽입
    if i % 2 == 0:
        heapq.heappush(pq1, -x)
    # 짝수 번째에는 최소 우선순위 큐에 삽입
    else:
        heapq.heappush(pq2, x)

    # 최대 우선순위 큐의 값이 최소 우선순위 큐의 값보다 크면
    if len(pq2) != 0 and -pq1[0] > pq2[0]:
        t1 = -pq1[0]
        t2 = pq2[0]
        # 서로 스왑해 줌
        heapq.heappop(pq1)
        heapq.heappop(pq2)
        heapq.heappush(pq1, -t2)
        heapq.heappush(pq2, t1)

    # 최대 우선순위 큐의 값이 중간 값
    print(-pq1[0])
```

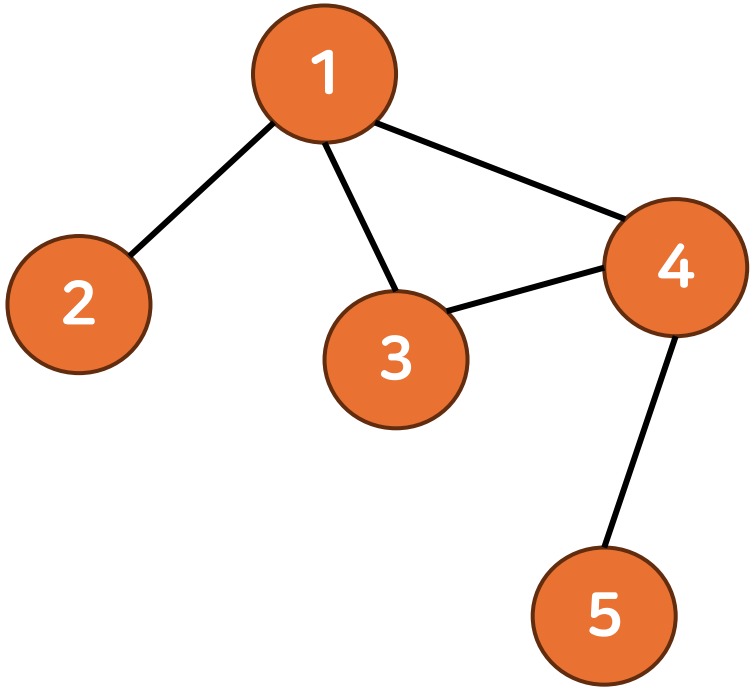
질문?

4주차 - BFS / DFS

그래프

그래프

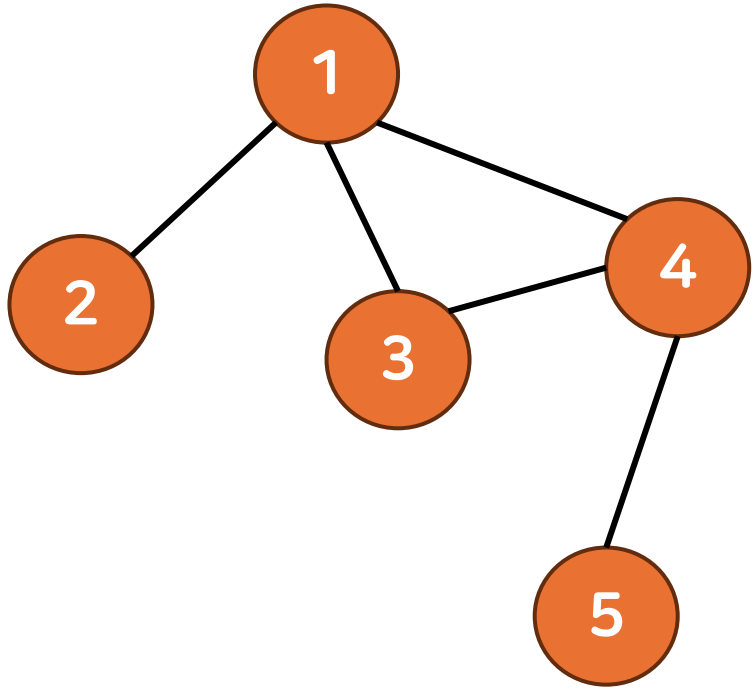
정점들의 집합과
정점들을 연결하는 간선들의 집합으로
이루어진 자료구조



그래프

그래프

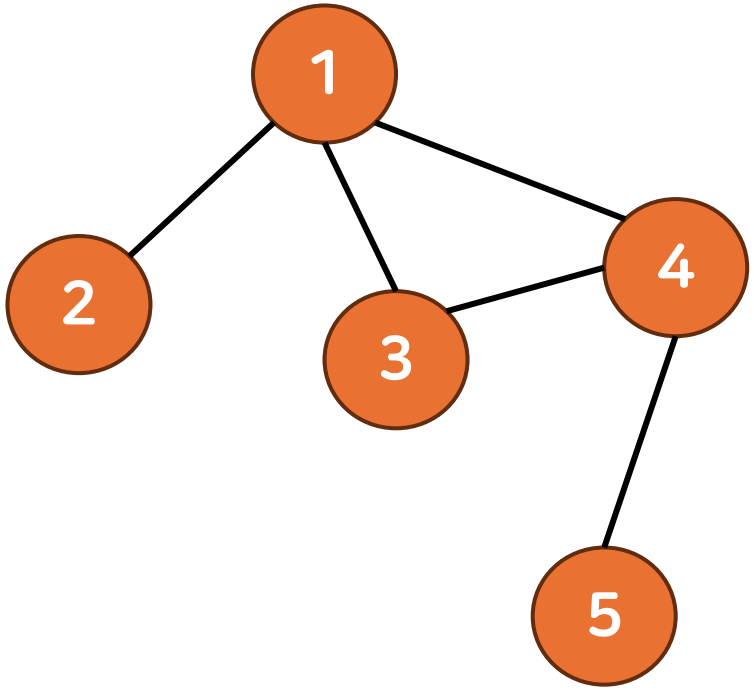
아래의 그래프는
1, 2, 3, 4, 5의 정점과
(1, 2), (1, 3), (1, 4), (3, 4), (4, 5)의 간선을 가짐



그래프

그래프

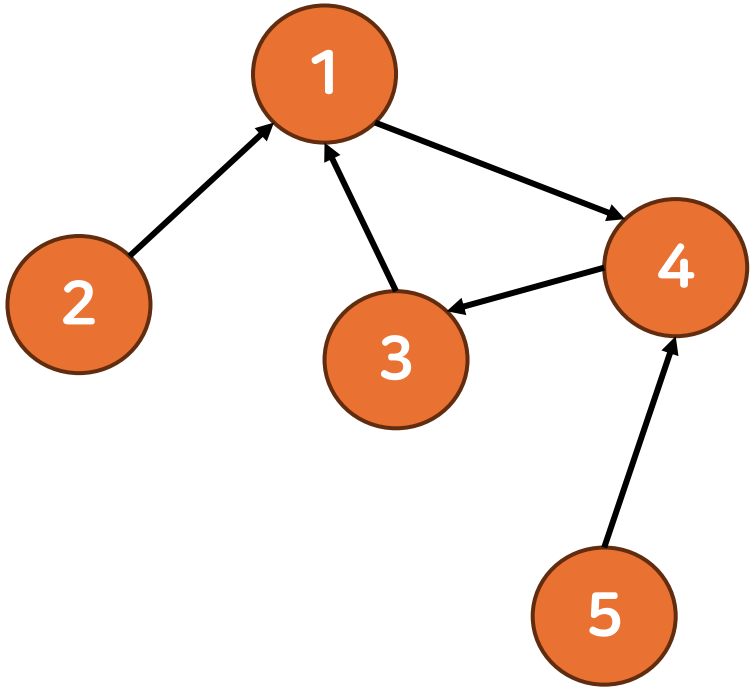
간선에 방향이 없으면 무방향 그래프



그래프

그래프

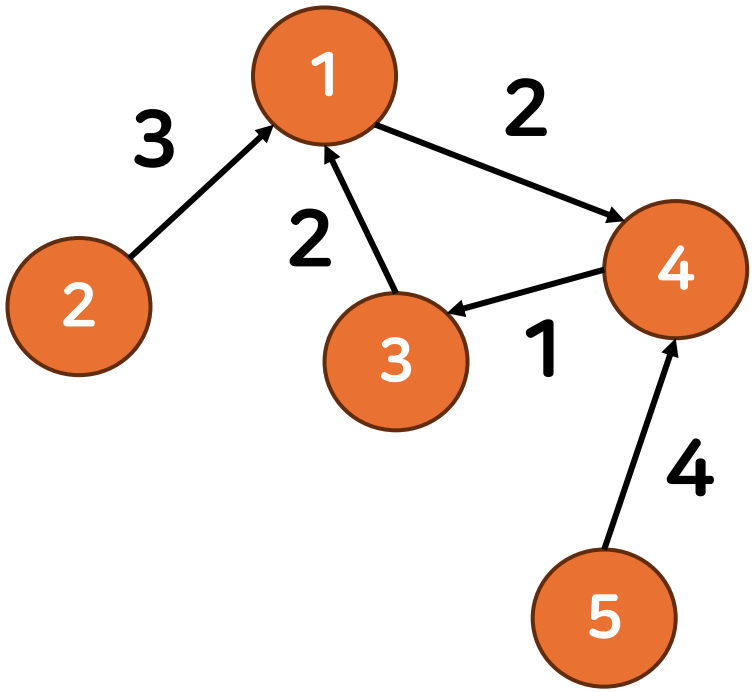
간선에 방향이 있으면 방향 그래프



그래프

그래프

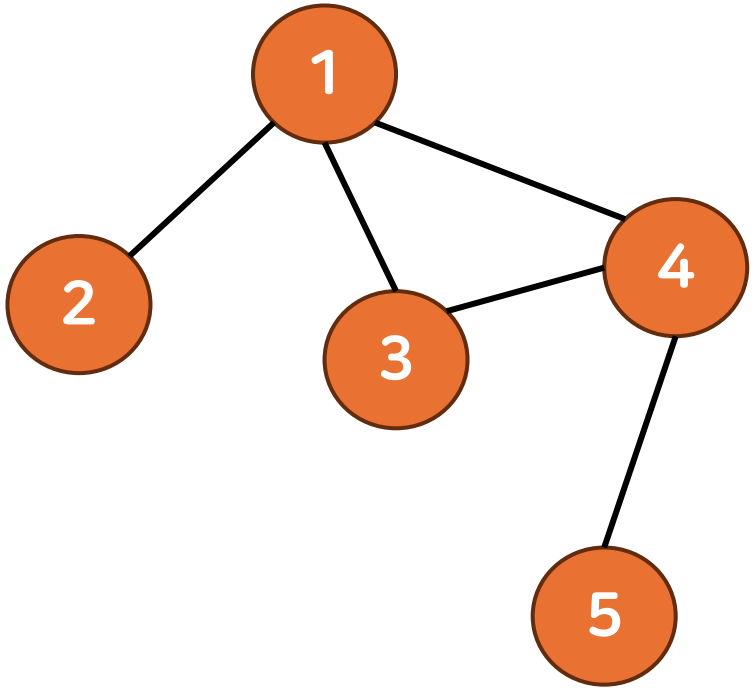
간선에 방향이 있으면 가중치 그래프



그래프

그래프

모든 정점이 간선을 통해 연결되어 있으면 연결 그래프

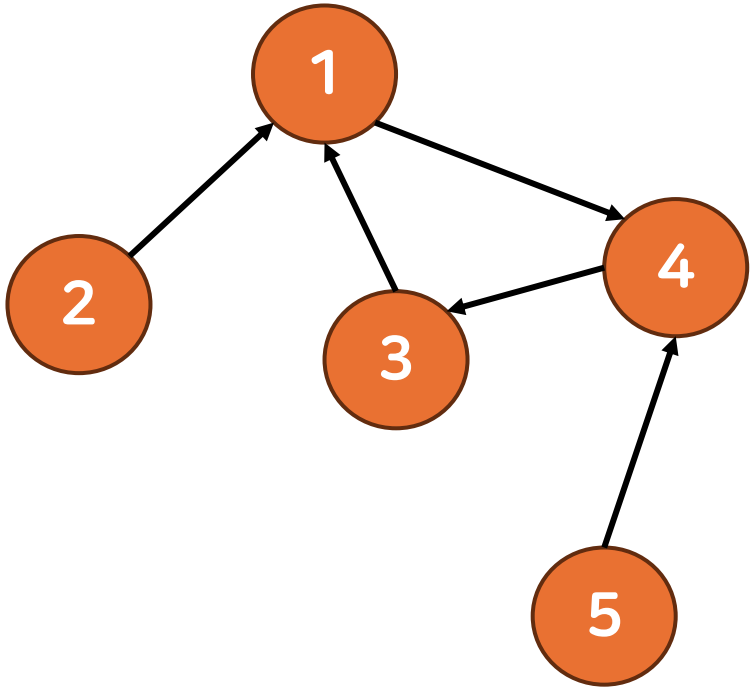


그래프

차수(Degree)

한 정점에 연결 되어 있는 간선의 수

방향 그래프이면 들어오는 간선, 나가는 간선 존재



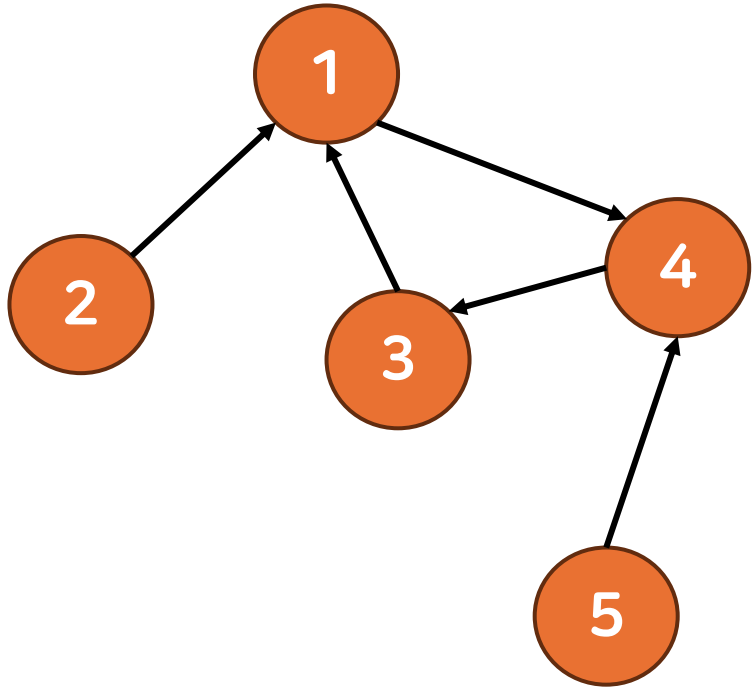
그래프

차수(Degree)

방향 그래프이면 들어오는 간선, 나가는 간선 존재

입력 차수(In-Degree) : 한 정점으로 들어오는 간선의 수

출력 차수(Out-Degree) : 한 정점에서 나가는 간선의 수

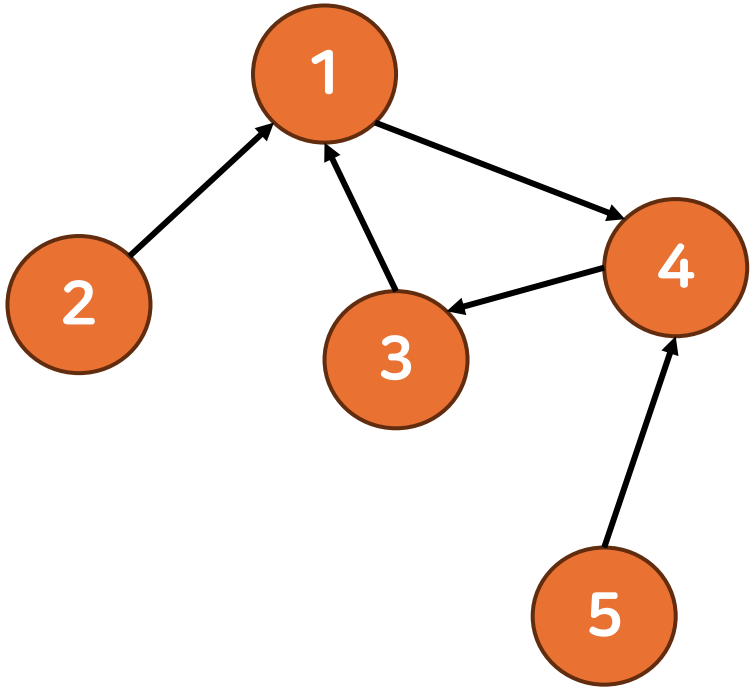


그래프

사이클

시작점과 끝점이 같고 나머지는 중복으로 방문하지 않는 경로

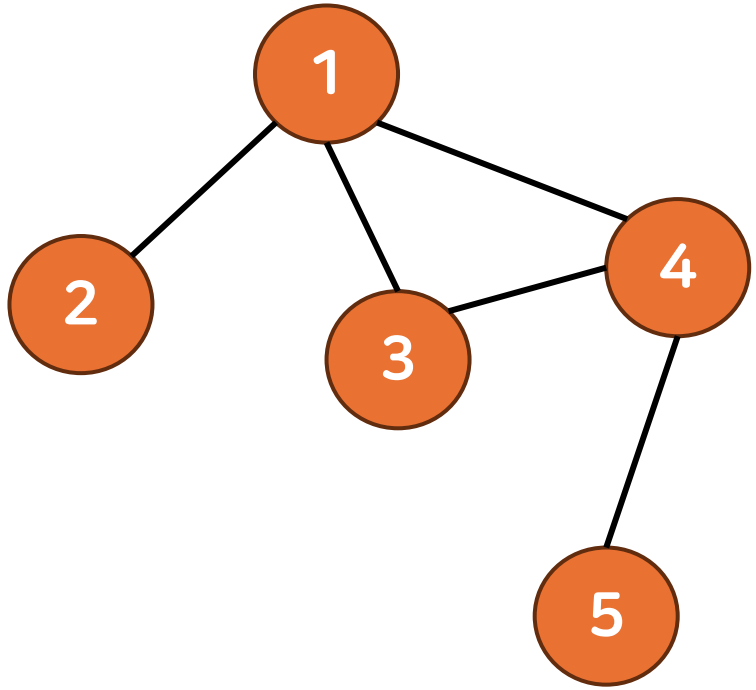
EX) 1-> 3 -> 4



그래프

사이클

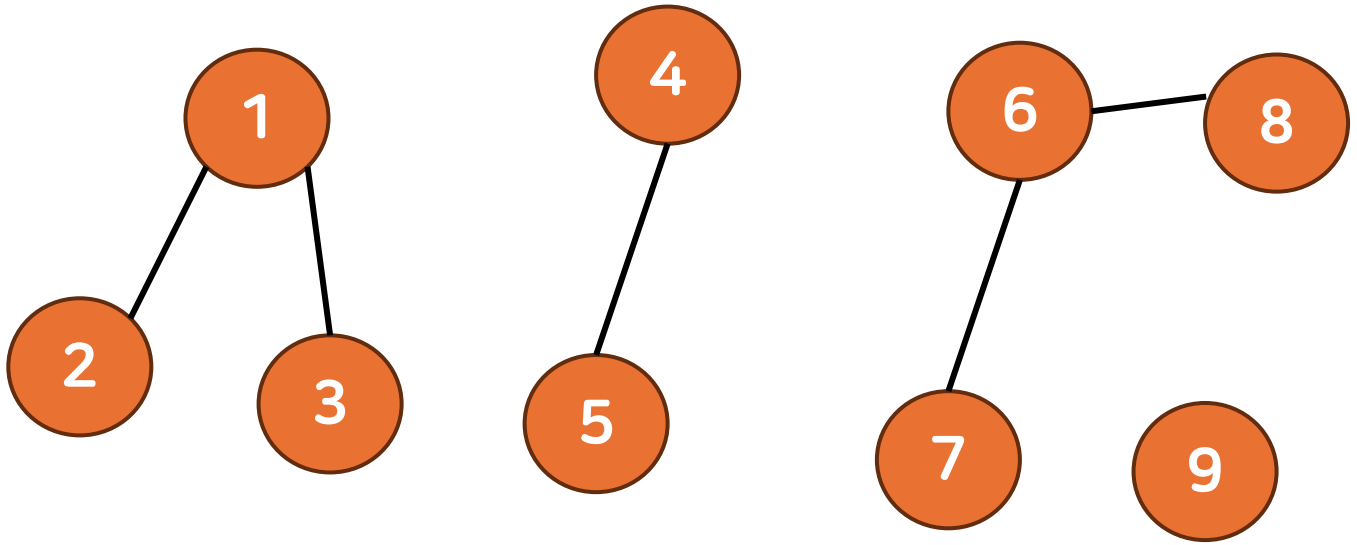
무방향 그래프에서는 1 -> 4 -> 1 같이 한 간선으로도
사이클이 존재 함
보통 길이 K 이상의 사이클을 요구 함



그래프

연결 요소(Connected Component)

간선을 통해서 서로 직/간접적으로 연결된
정점들의 집합
아래 그래프는 (1, 2, 3), (4, 5), (6, 7, 8), (9)로
4개의 연결 요소 존재

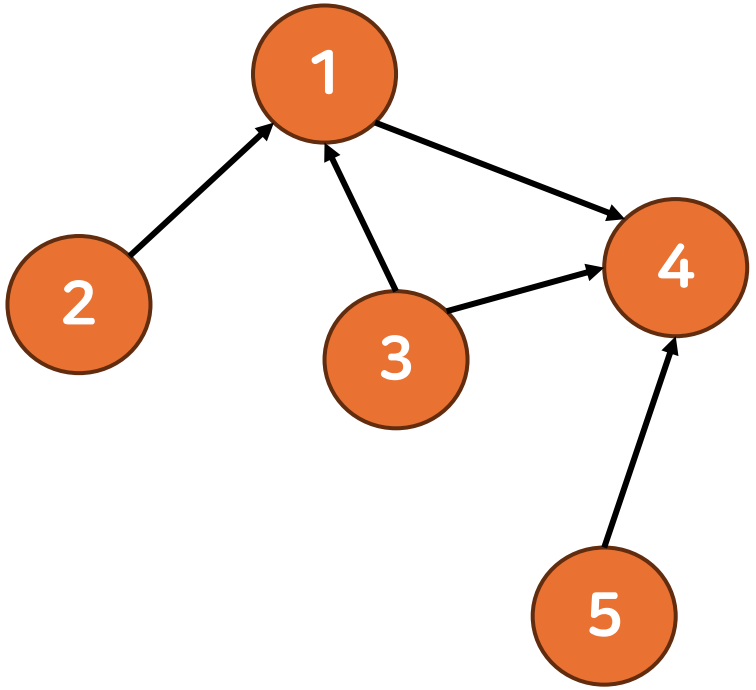


질문?

특수한 그래프

비순환 방향 그래프(DAG)

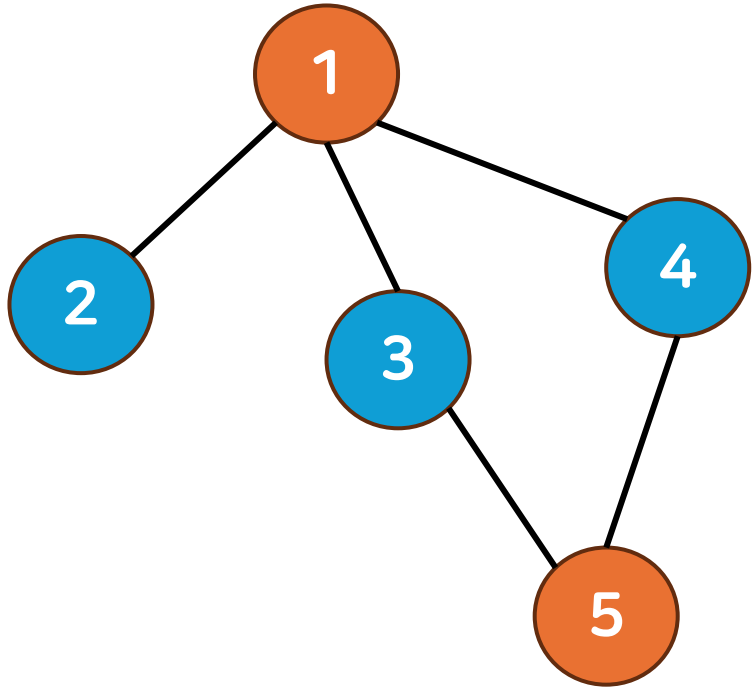
사이클이 없는 방향 그래프



특수한 그래프

이분 그래프(Bipartite Graph)

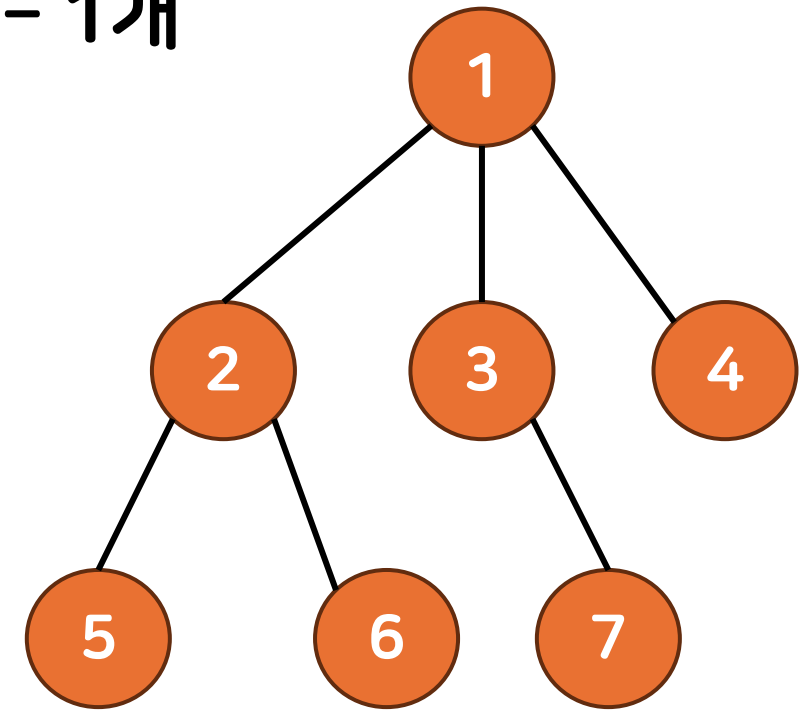
서로 연결되어 있는 정점들을
다른 집합으로 나눌 수 있는 그래프



특수한 그래프

트리

연결 요소가 1개인 사이클이 없는 연결 그래프
정점의 개수가 N개 일 때
간선의 개수가 N - 1개



질문?

그래프의 표현 방식

인접 행렬
인접 리스트
간선 리스트

거의 간선 리스트 사용
가끔 인접 행렬 사용

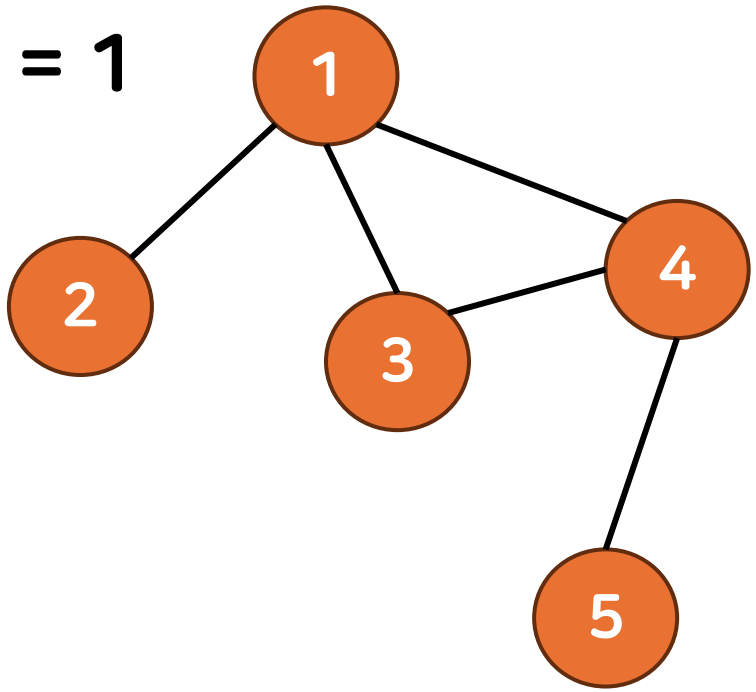
인접 행렬

인접 행렬

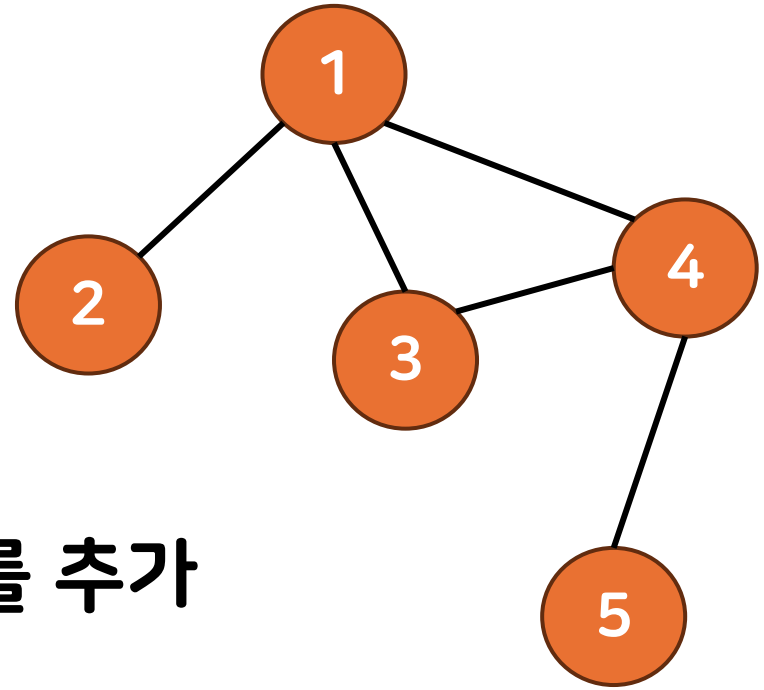
정점의 개수가 N일 때
N * N 2차원 배열로 간선을 정의

S에서 E로 가는 간선이 있으면 A[S][E] = 1
가중치 그래프면 A[S][E] = 가중치

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



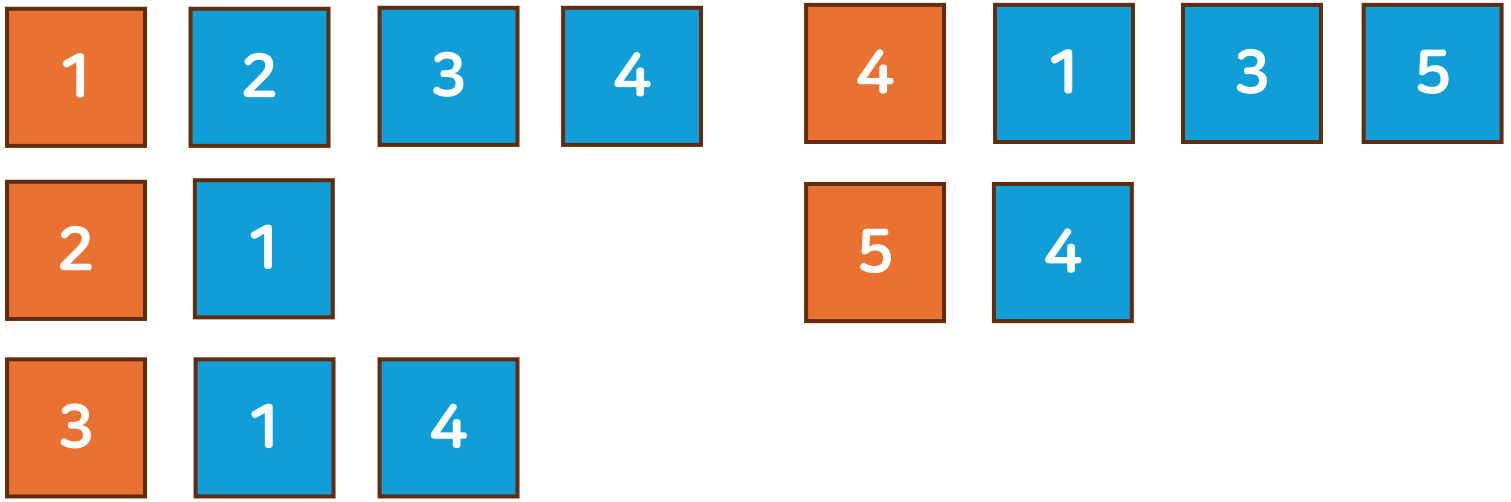
간선 리스트



간선 리스트

정점의 개수가 N일 때
N개의 동적 배열로 간선을 정의

S에서 E로 가는 간선이 있으면 A[S]에 E를 추가
가중치 그래프에서는 (E, 가중치)를 추가



그래프의 표현 방식

인접 행렬

공간 복잡도 O(N^2)

S, E를 연결하는 간선 확인 O(1)

모든 간선 순회 O(N^2)

간선 리스트

공간 복잡도 O(N + V)

S, E를 연결하는 간선 확인 O(deg(S))

모든 간선 순회 O(N + V)

질문?

DFS / 깊이 우선 탐색

DFS

한 정점에서 시작
현재 정점과 인접한 정점 중에서
방문하지 않은 정점을 방문

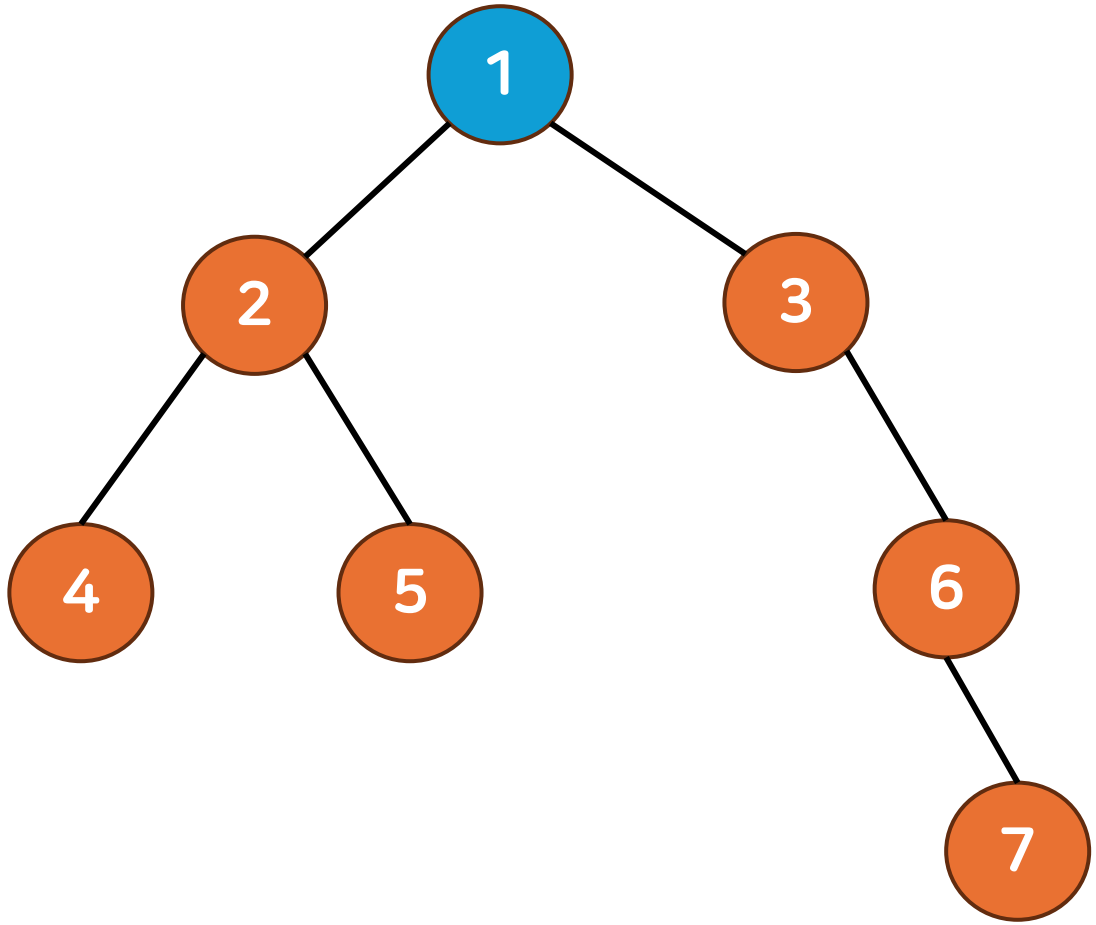
인접한 정점 중에서 방문하지 않은 정점이 없으면
이전으로 돌아감

-> 재귀로 구현

DFS / 깊이 우선 탐색

DFS

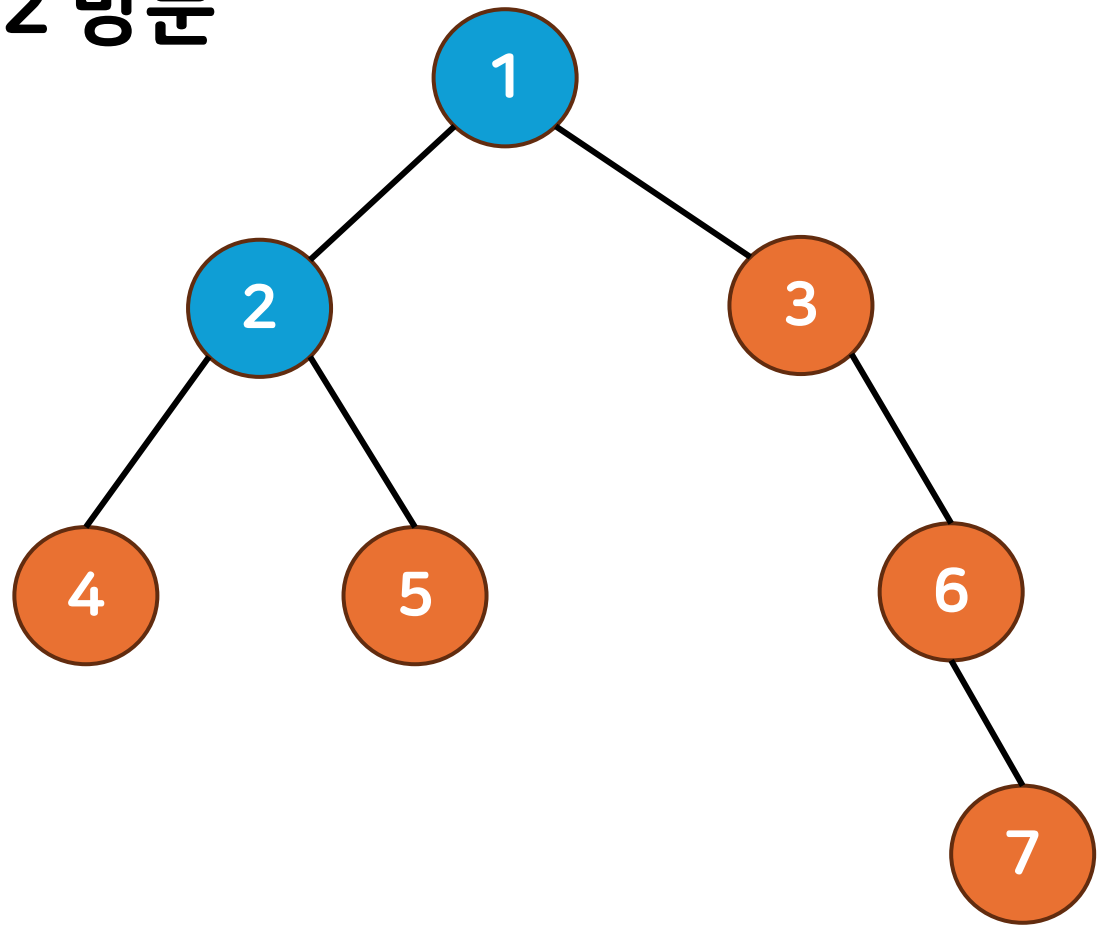
1에서 시작



DFS / 깊이 우선 탐색

DFS

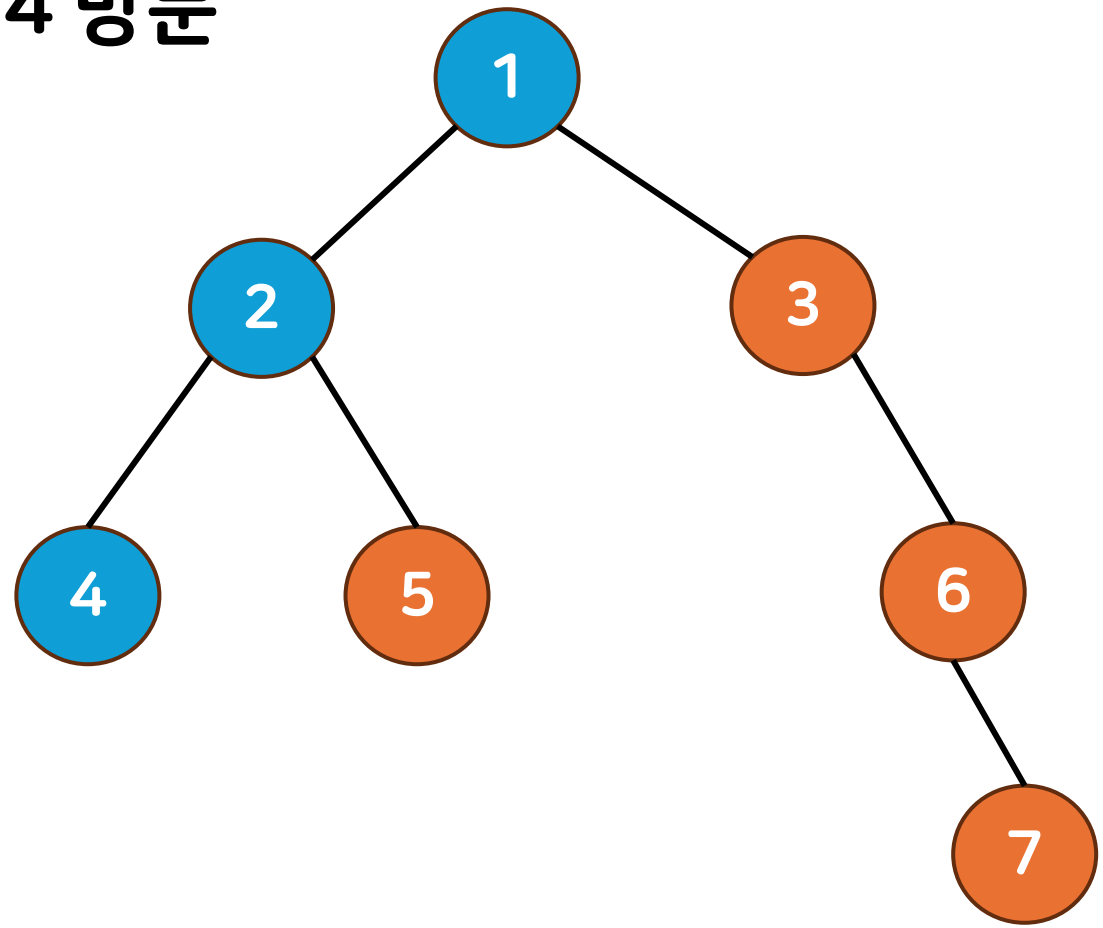
1과 인접한 2 방문



DFS / 깊이 우선 탐색

DFS

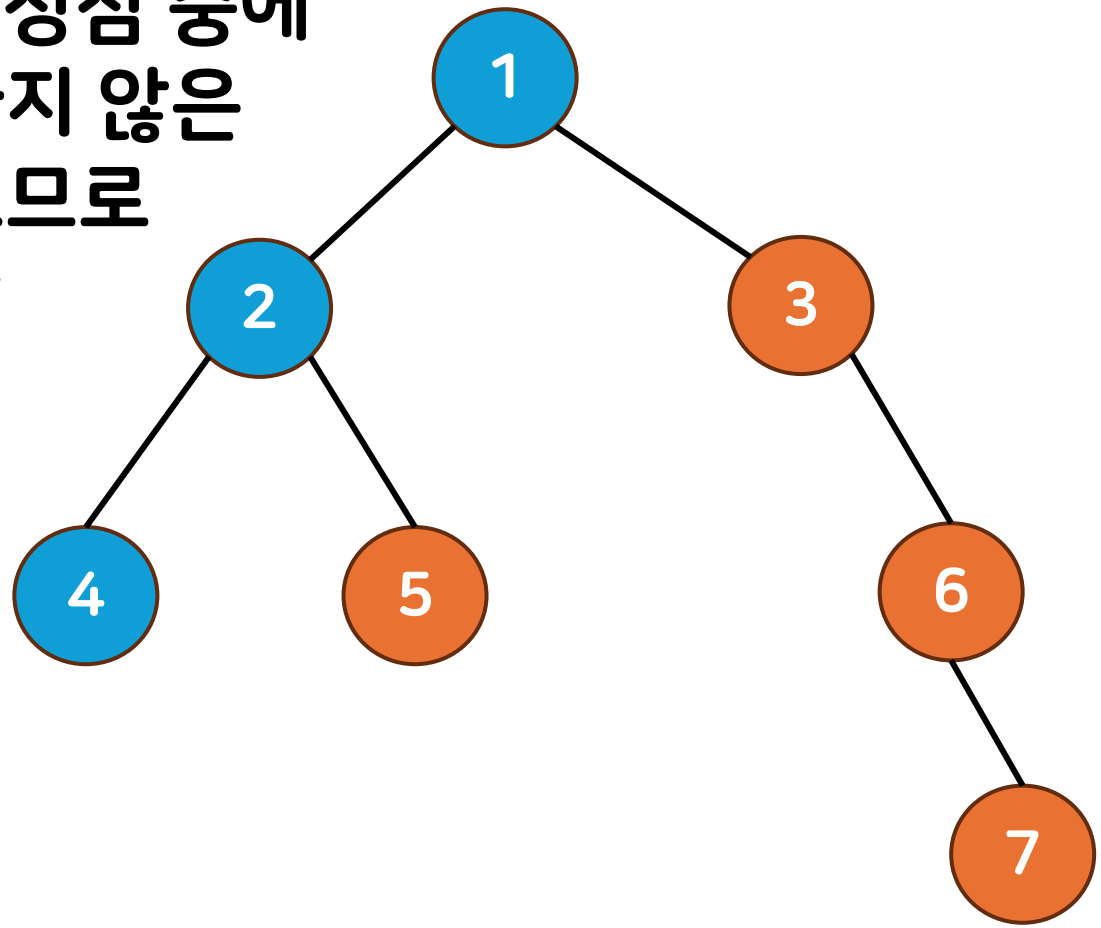
2와 인접한 4 방문



DFS / 깊이 우선 탐색

DFS

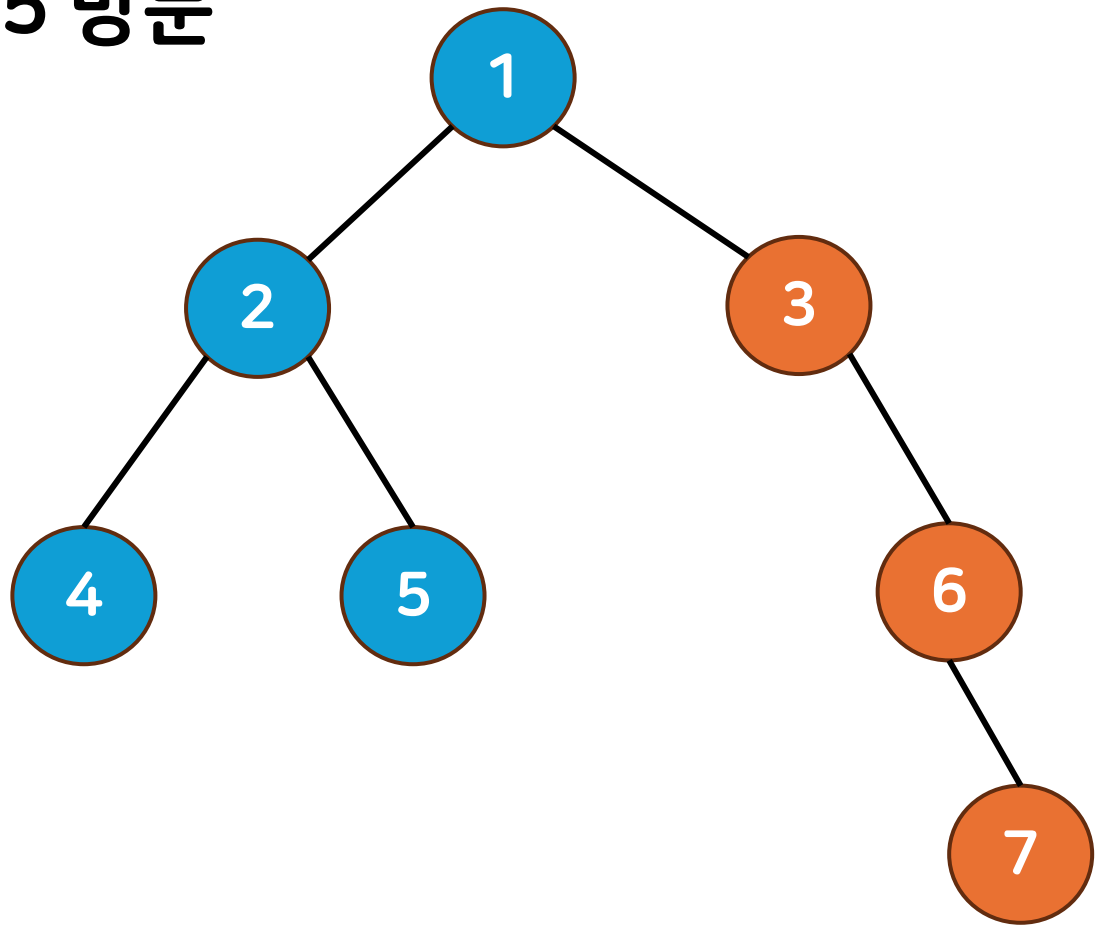
4에 인접한 정점 중에
아직 방문하지 않은
정점이 없으므로
2로 돌아 감



DFS / 깊이 우선 탐색

DFS

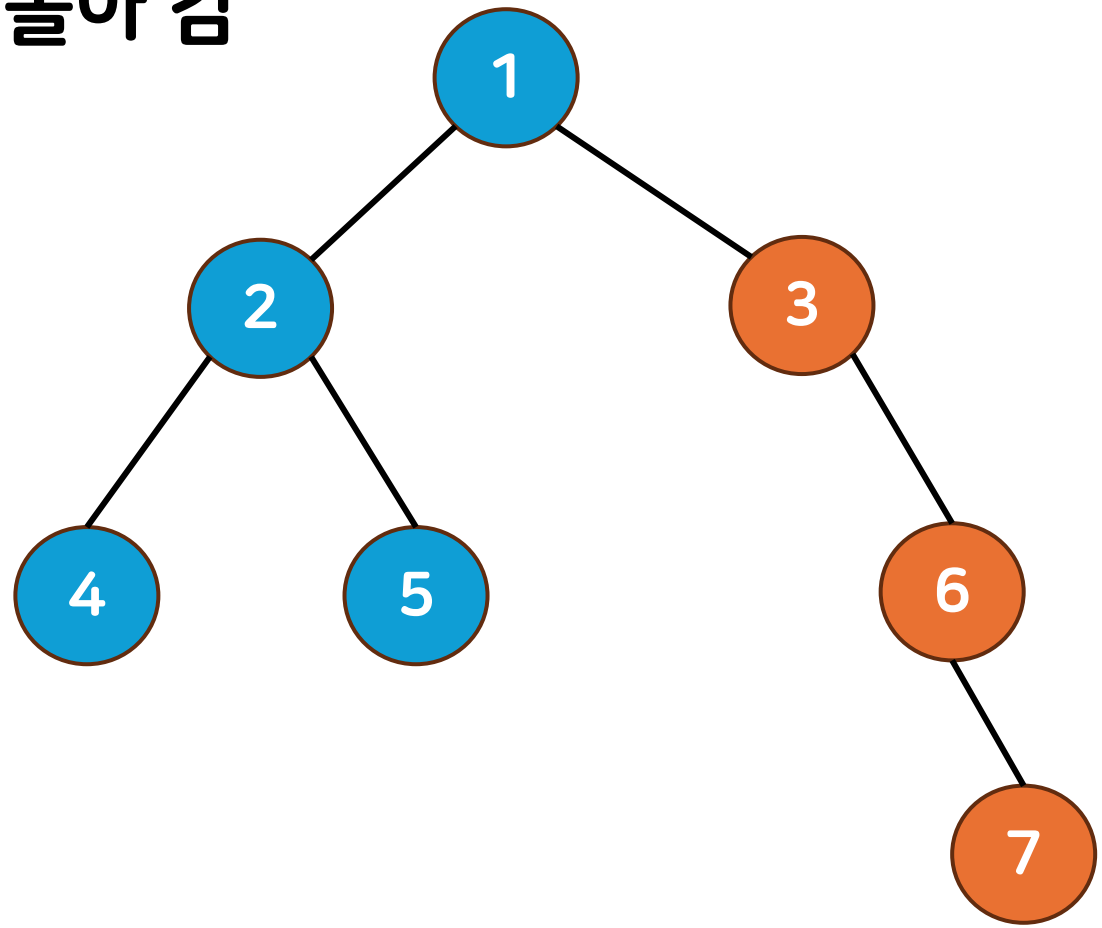
2와 인접한 5 방문



DFS / 깊이 우선 탐색

DFS

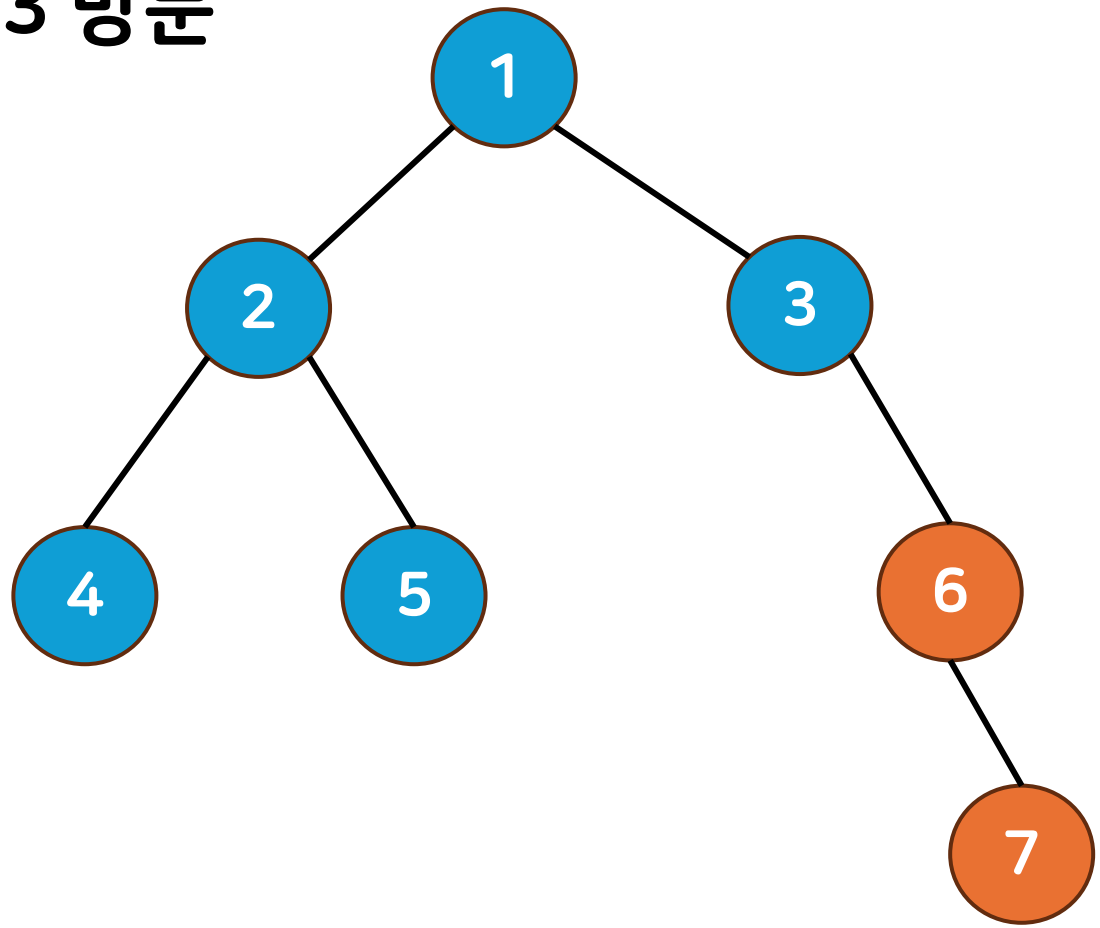
1까지 다시 돌아 감



DFS / 깊이 우선 탐색

DFS

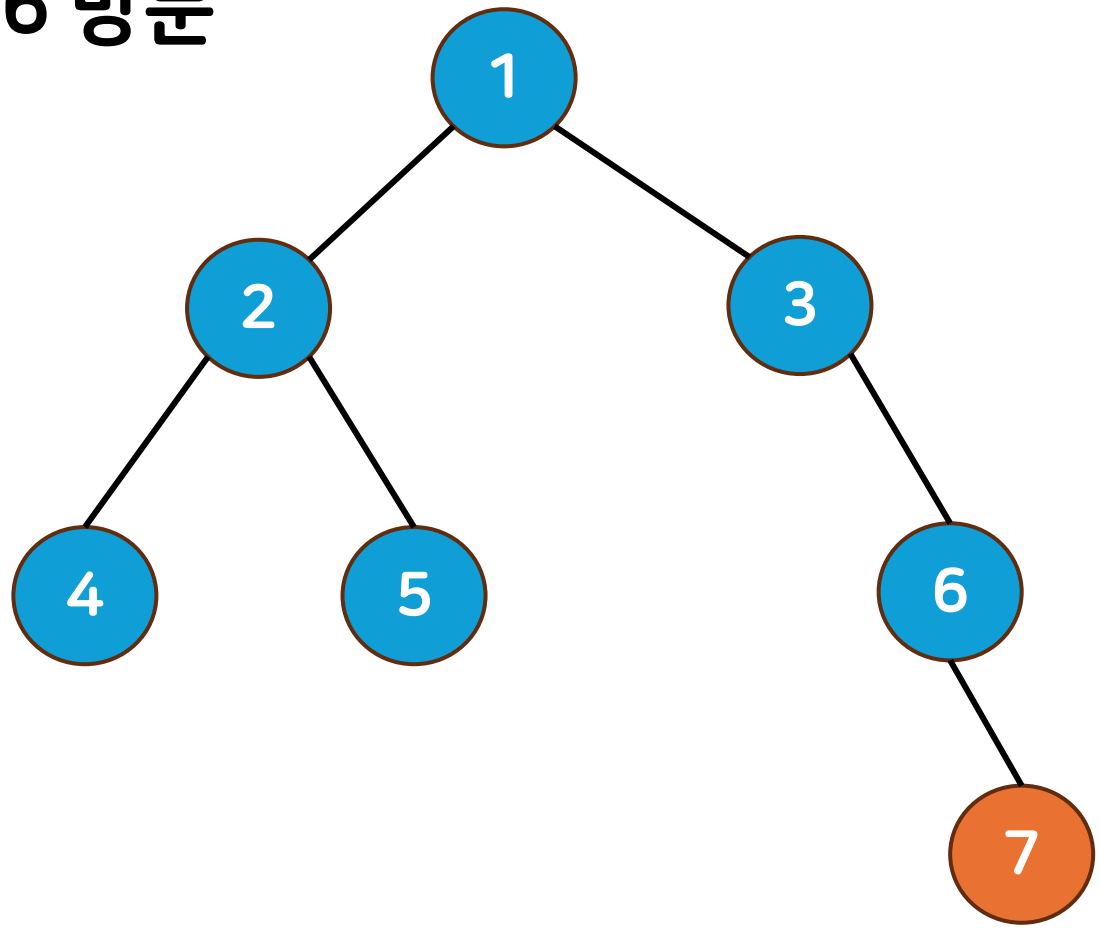
1과 인접한 3 방문



DFS / 깊이 우선 탐색

DFS

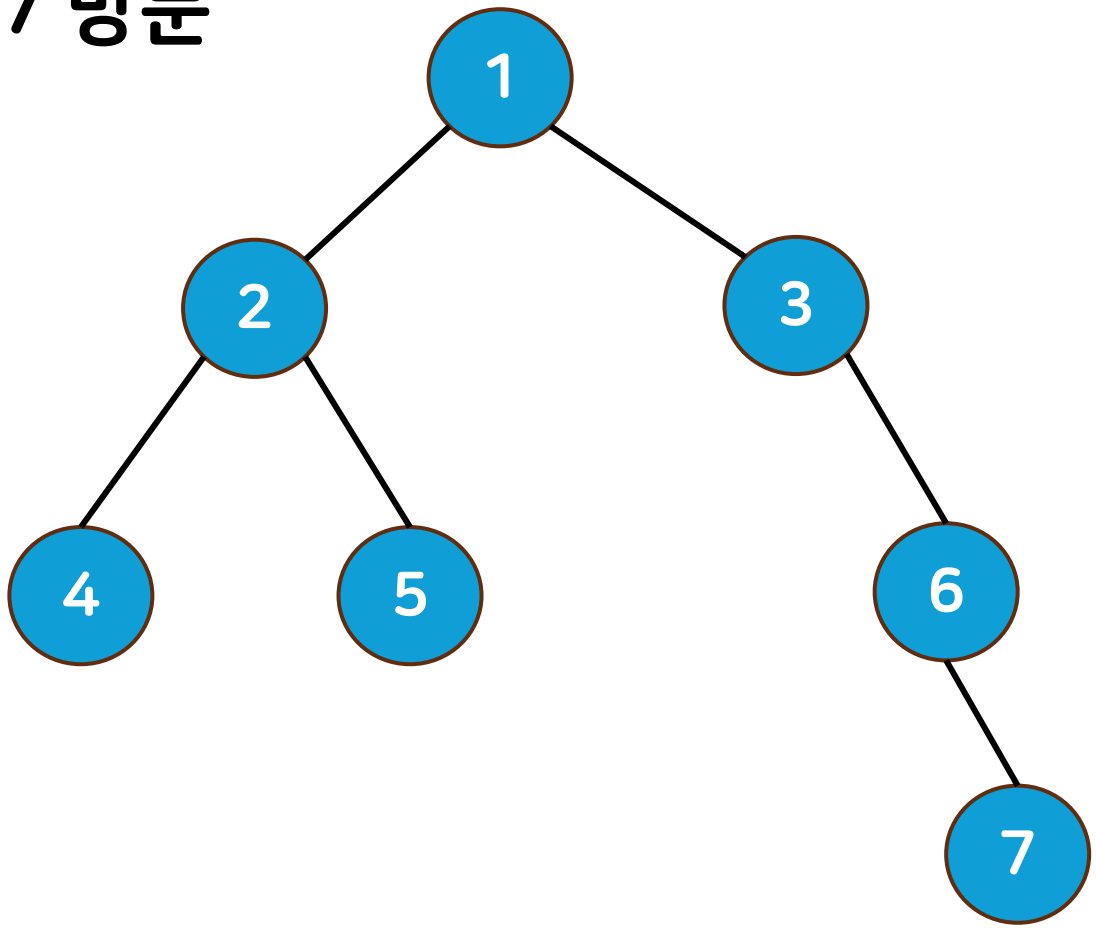
3과 인접한 6 방문



DFS / 깊이 우선 탐색

DFS

6과 인접한 7 방문



DFS / 깊이 우선 탐색

시간 복잡도

각 정점은 한번만 방문하고
방문 할 때마다 인접한 간선을 봐야 함

-> O(V + E)
E은 정점의 개수, V는 간선의 개수

DFS / 깊이 우선 탐색

C++

```
Received Output:
1
2
4
5
3
6
7
```

```cpp
#include <iostream>
#include <vector>
using namespace std;

vector <int> a[11];
bool v[11];
```

```cpp
int main(){
    ios::sync_with_stdio(0); // fastio
    cin.tie(0), cout.tie(0); // fastio

    add(1, 2); add(2, 4); add(2, 5);
    add(1, 3); add(3, 6); add(6, 7);
    dfs(1);

    return 0;
}
```

```cpp
void dfs(int cur){
    // 방문한 정점으로 표시
    v[cur] = 1;
    cout << cur << "\n";

    for(auto& nxt : a[cur]){
        // 다음 정점이 방문한 정점이면 넘어 감
        if(v[nxt]) continue;
        dfs(nxt);
    }
}

void add(int s, int e){
    // 양방향 간선 추가
    a[s].push_back(e);
    a[e].push_back(s);
}
```

DFS / 깊이 우선 탐색

Python

```
import sys
input = sys.stdin.readline
v = [0] * 10
a = [[] for _ in range(10)]
```

```
add(1, 2)
add(2, 4)
add(2, 5)
add(1, 3)
add(3, 6)
add(6, 7)

dfs(1)
```

```
Received Output:
1
2
4
5
3
6
7
```

```
def add(s, e):
    # 양방향 간선 추가
    a[s].append(e)
    a[e].append(s)

def dfs(cur):
    # 방문한 정점으로 표시
    v[cur] = 1
    print(cur)

    for nxt in a[cur]:
        # 다음 정점이 방문한 정점이면 넘어 감
        if v[nxt] == 1:
            continue
        dfs(nxt)
```

질문?

BFS / 넓이 우선 탐색

BFS

한 정점에서 시작
현재 정점과 인접한 정점 중에서
방문하지 않은 정점을 모두 방문

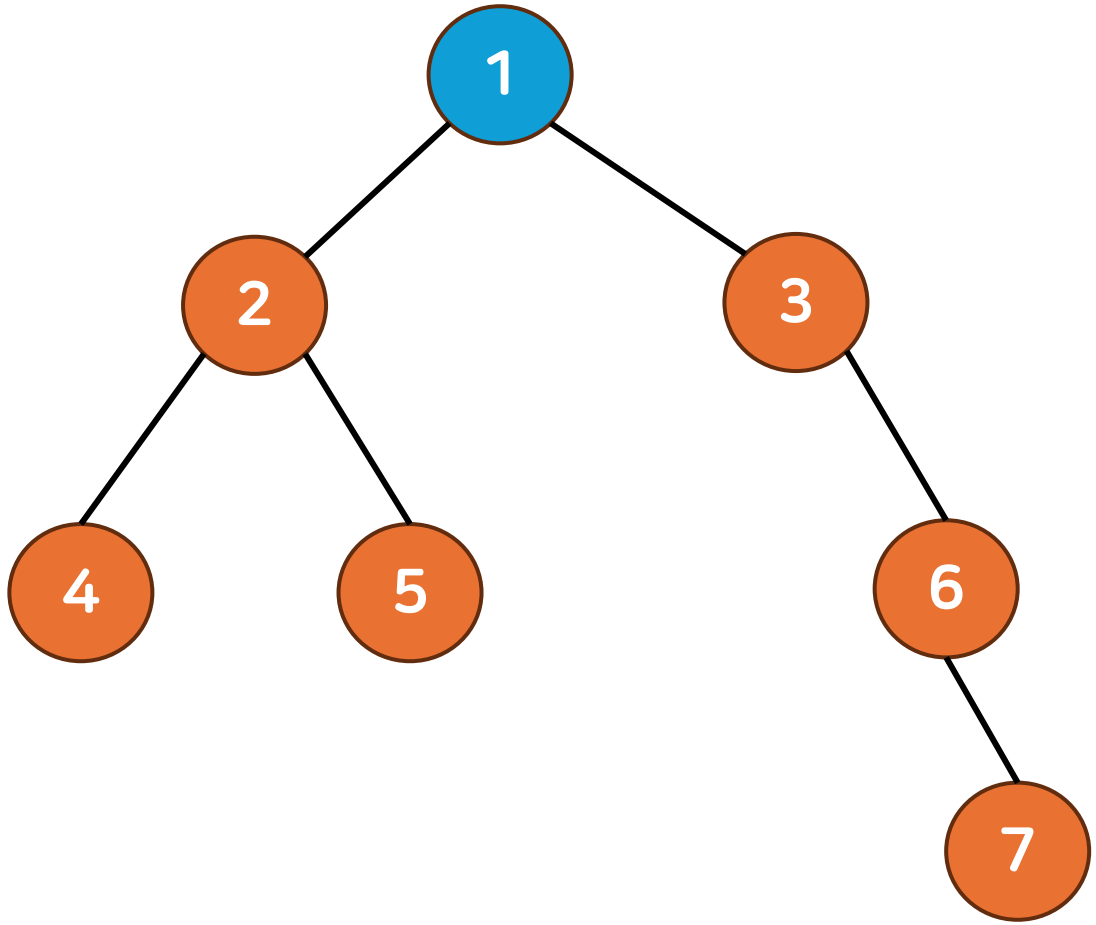
시작 정점과 가까운 거리부터 방문 하게 됨
정점들의 방문 대기열을 만들어야 함

-> 큐로 구현

BFS / 넓이 우선 탐색

BFS

1에서 시작

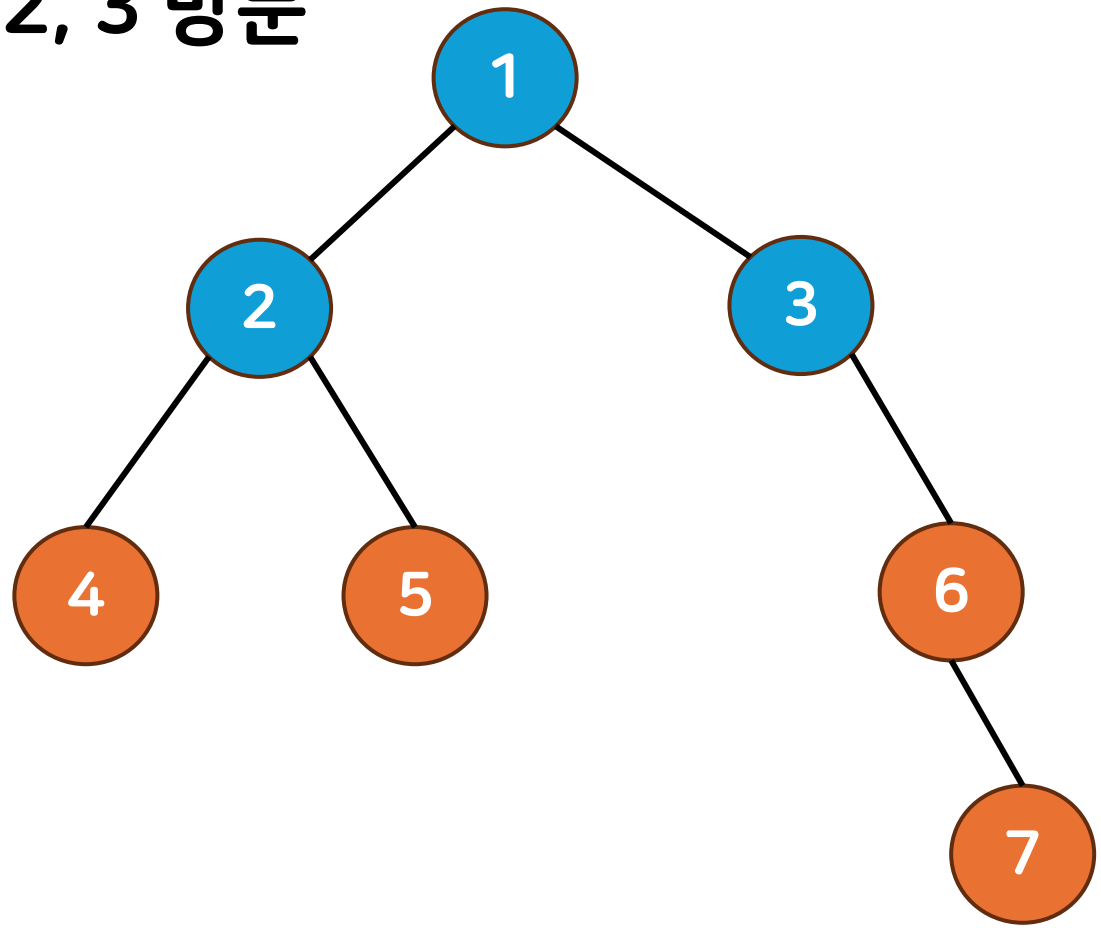


큐

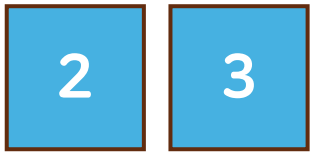
BFS / 넓이 우선 탐색

BFS

1과 인접한 2, 3 방문



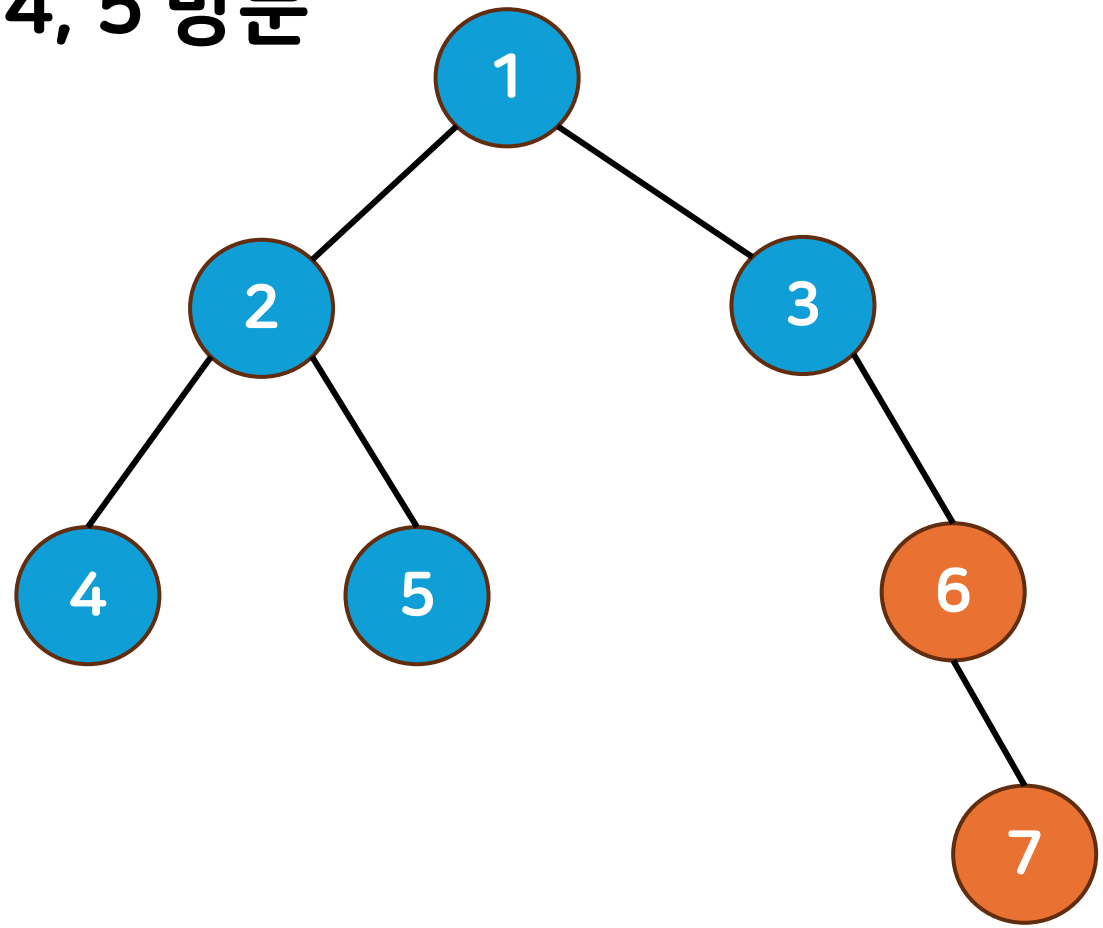
큐



BFS / 넓이 우선 탐색

BFS

2와 인접한 4, 5 방문

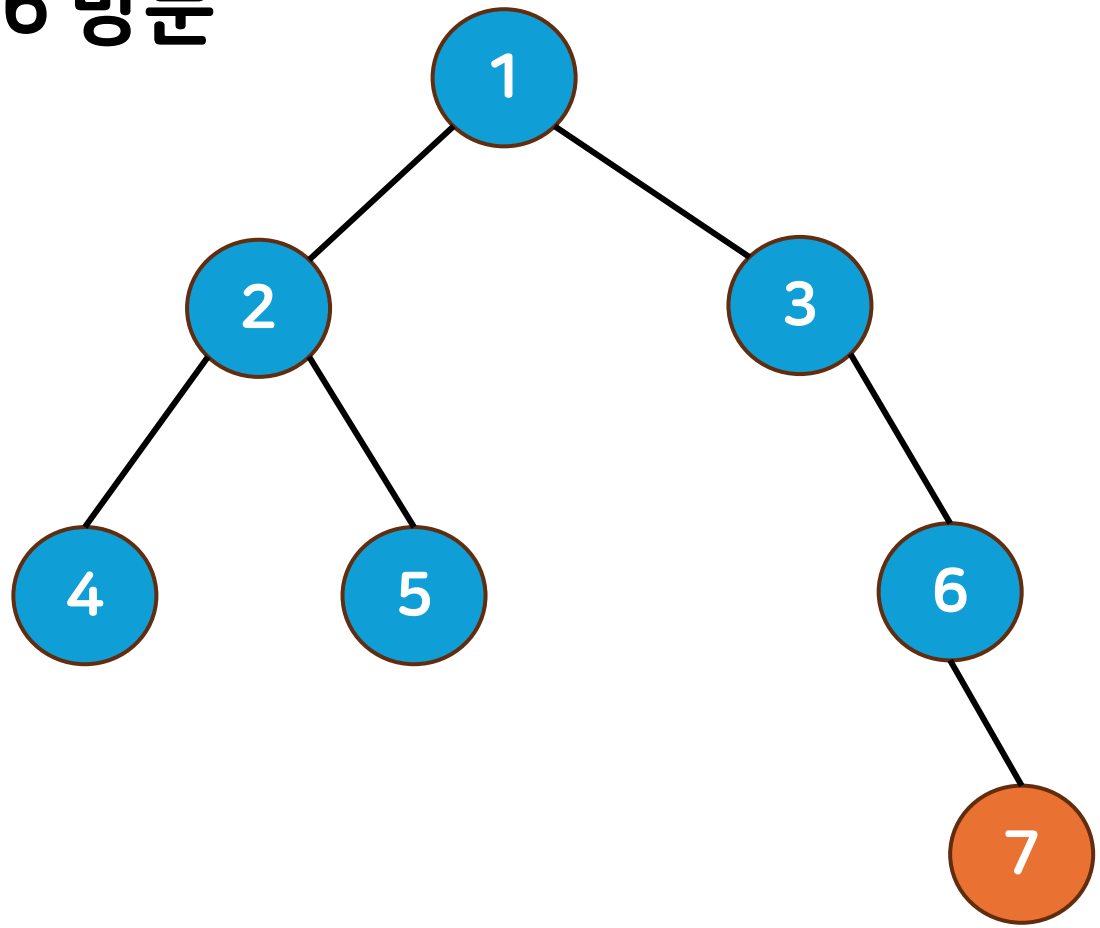


큐

BFS / 넓이 우선 탐색

BFS

3과 인접한 6 방문

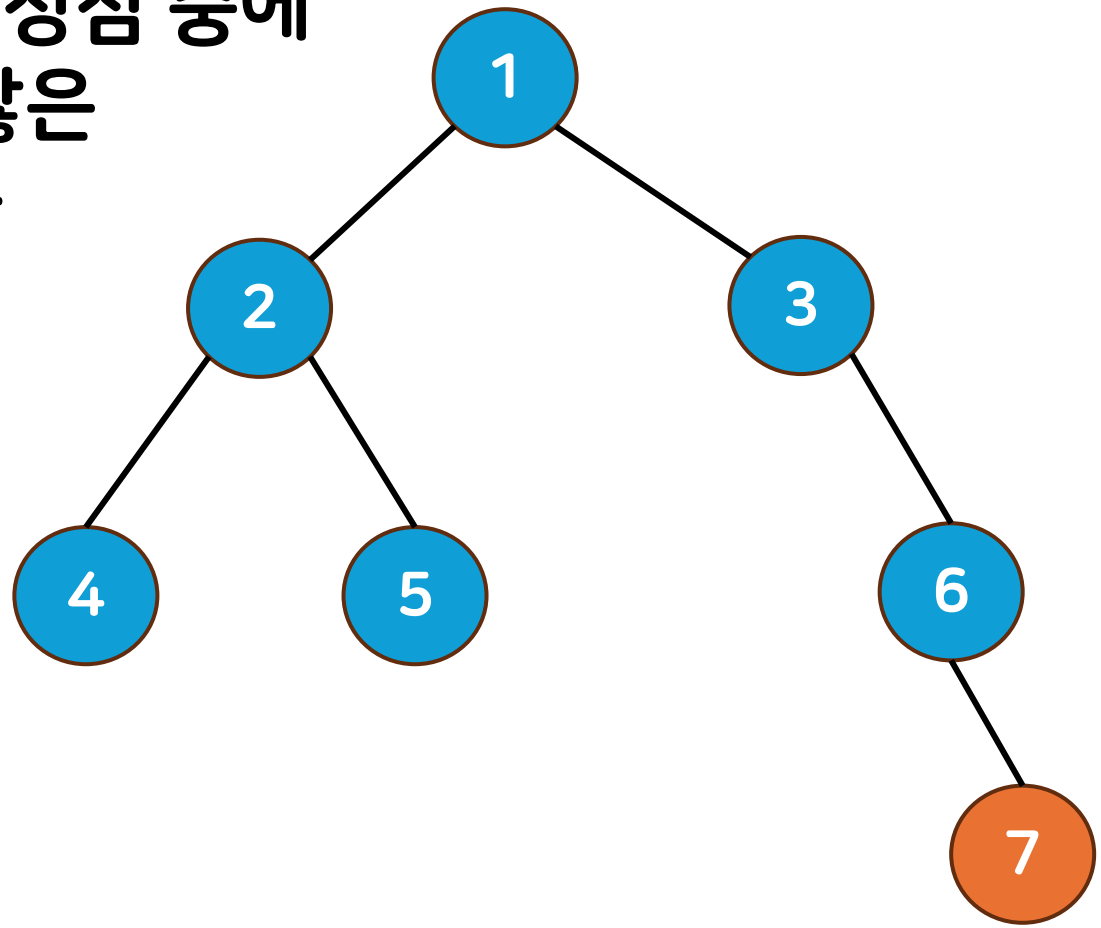


큐

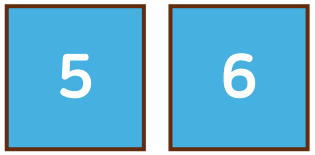
BFS / 넓이 우선 탐색

BFS

4는 인접한 정점 중에
방문 하지 않은
정점이 없음



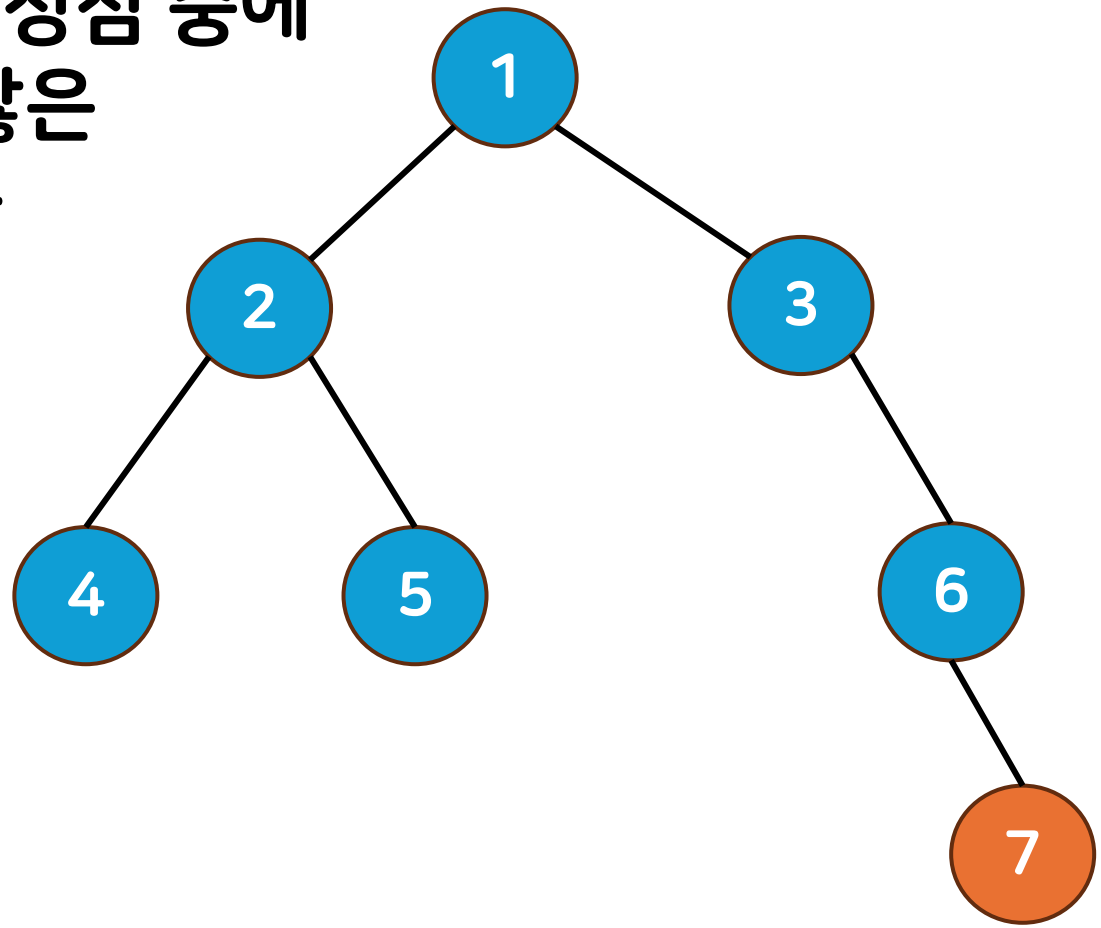
큐



BFS / 넓이 우선 탐색

BFS

5는 인접한 정점 중에
방문 하지 않은
정점이 없음

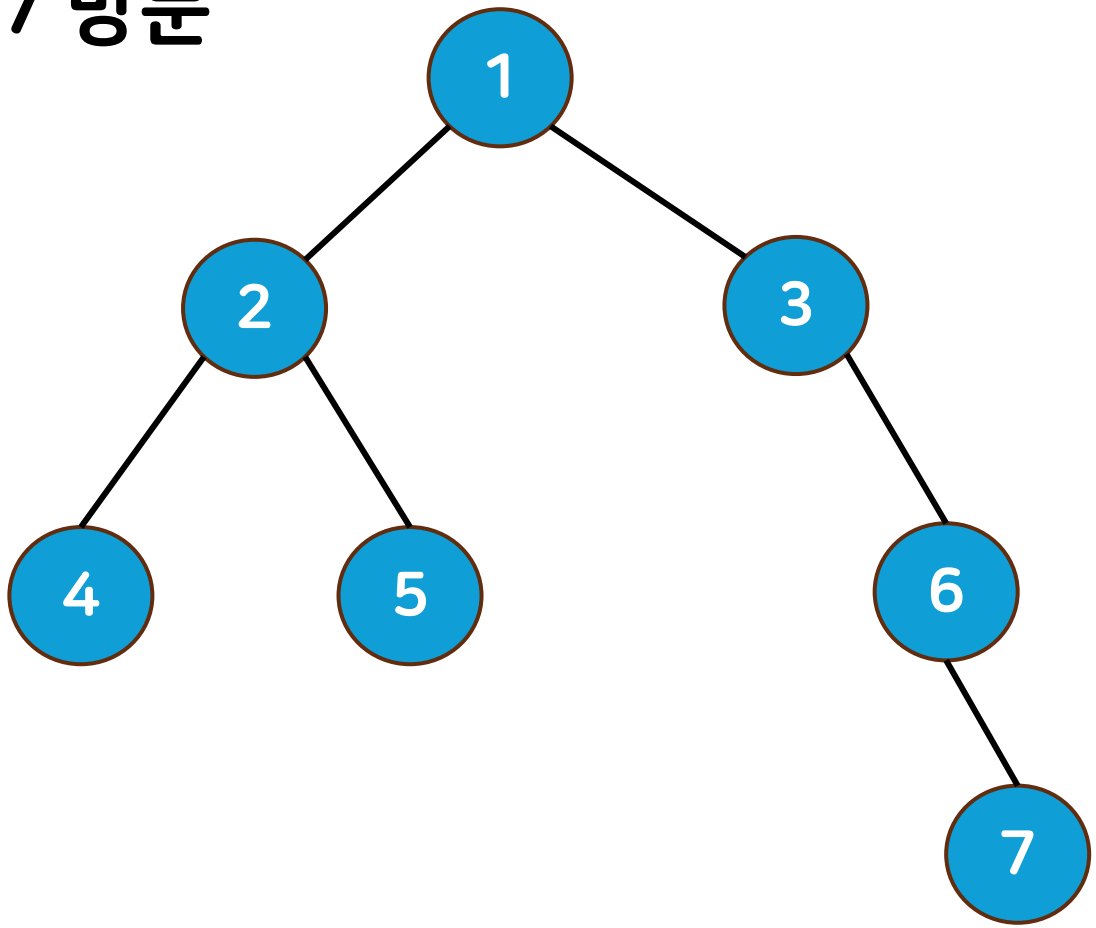


큐

BFS / 넓이 우선 탐색

BFS

6과 인접한 7 방문



큐

BFS / 넓이 우선 탐색

BFS

처음에 방문한 노드를 0
다음에 방문한 노드를 전 노드 값에 +1을 해주면
최단거리를 구할 수 있음

BFS / 넓이 우선 탐색

시간 복잡도

각 정점은 한번만 방문하고
방문 할 때마다 인접한 간선을 봐야 함

-> O(V + E)
E은 정점의 개수, V는 간선의 개수

BFS / 넓이 우선 탐색

C++

```
Received Output:
1
2
3
4
5
6
7
```

```cpp
#include <iostream>
#include <vector>
using namespace std;

vector <int> a[11];
bool v[11];
```

```cpp
int main(){
    ios::sync_with_stdio(0); // fastio
    cin.tie(0), cout.tie(0); // fastio

    add(1, 2); add(2, 4); add(2, 5);
    add(1, 3); add(3, 6); add(6, 7);
    dfs(1);

    return 0;
}
```

```cpp
void bfs(int st){
    queue <int> q;
    // 큐에 시작점 추가 및 방문 처리
    q.push(st); v[st] = 1;
    while(!q.empty()){
        int cur = q.front(); q.pop();
        cout << cur << "\n";
        for(auto& nxt : a[cur]){
            // 방문한 정점이면 넘어 감
            if(v[nxt]) continue;
            // 큐에 다음 정점 삽입 및 방문 처리
            v[nxt] = 1;
            q.push(nxt);
        }
    }
}

void add(int s, int e){
    // 양방향 간선 추가
    a[s].push_back(e);
    a[e].push_back(s);
}
```

BFS / 넓이 우선 탐색

Python

```python
import sys
from collections import deque
input = sys.stdin.readline
v = [0] * 10
a = [[] for _ in range(10)]
```

```
Received Output:
1
2
3
4
5
6
7
```

```python
add(1, 2)
add(2, 4)
add(2, 5)
add(1, 3)
add(3, 6)
add(6, 7)

bfs(1)
```

```python
def add(s, e):
    # 양방향 간선 추가
    a[s].append(e)
    a[e].append(s)

def bfs(st):
    q = deque()
    # 큐에 시작점 추가 및 방문 처리
    q.append(st)
    v[st] = 1

    while len(q) != 0:
        cur = q.popleft()
        print(cur)
        for nxt in a[cur]:
            # 방문한 정점이면 넘어 감
            if v[nxt] == 1:
                continue
            # 큐에 다음 정점 삽입 및 방문 처리
            v[nxt] = 1
            q.append(nxt)
```

질문?

DFS와 BFS / 1260

백준 1260 / https://www.acmicpc.net/problem/1260

문제

그래프를 DFS로 탐색한 결과와 BFS로 탐색한 결과를 출력하는 프로그램을 작성하시오. 단, 방문할 수 있는 정점이 여러 개인 경우에는 정점 번호가 작은 것을 먼저 방문하고, 더 이상 방문할 수 있는 점이 없는 경우 종료한다. 정점 번호는 1번부터 N번까지이다.

입력

첫째 줄에 정점의 개수 N(1 ≤ N ≤ 1,000), 간선의 개수 M(1 ≤ M ≤ 10,000), 탐색을 시작할 정점의 번호 V가 주어진다. 다음 M개의 줄에는 간선이 연결하는 두 정점의 번호가 주어진다. 어떤 두 정점 사이에 여러 개의 간선이 있을 수 있다. 입력으로 주어지는 간선은 양방향이다.

출력

첫째 줄에 DFS를 수행한 결과를, 그 다음 줄에는 BFS를 수행한 결과를 출력한다. V부터 방문된 점을 순서대로 출력하면 된다.

DFS와 BFS / 1260

DFS의 결과와 BFS의 결과 출력
같은 탐색 우선순위의 정점들을 방문하는
순서가 다르면 결과도 달라짐
-> 이 문제에서는 탐색 우선순위가 같으면 작은 정점부터 방문

예제 입력 1 복사

```
4 5 1
1 2
1 3
1 4
2 4
3 4
```

예제 출력 1 복사

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

DFS와 BFS / 1260

C++

크기가 작은 정점부터
방문 해야 함
-> 간선 정렬

```cpp
int main(){
    ios::sync_with_stdio(0); // fastio
    cin.tie(0), cout.tie(0); // fastio

    cin >> n >> m >> k;
    while(m--){
        ll s, e; cin >> s >> e;
        add(s, e);
    }

    // 크기가 작은 순으로 방문 해야 함
    for(int i = 1;i <= n;i++) sort(a[i].begin(), a[i].end());

    dfs(k); cout << "\n";
    // 방문 배열 초기화
    for(int i = 1;i <= n;i++) v[i] = 0;
    bfs(k);

    return 0;
}
```

DFS와 BFS / 1260

Python

크기가 작은 정점부터
방문 해야 함
-> 간선 정렬

```python
n, m, k = list(map(int, input().rstrip().split()))
for _ in range(m):
    s, e = list(map(int, input().rstrip().split()))
    add(s, e)

# 크기가 작은 순으로 방문 해야 함
for i in range(1, n + 1):
    a[i].sort()

dfs(k)
print()
# 방문 배열 초기화
for i in range(1, n + 1):
    v[i] = 0
bfs(k)
```

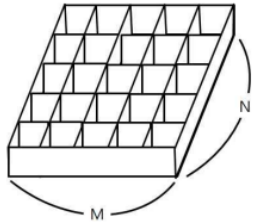
질문?

토마토 / 7576

백준 7576 / https://www.acmicpc.net/problem/7576

문제

철수의 토마토 농장에서는 토마토를 보관하는 큰 창고를 가지고 있다. 토마토는 아래의 그림과 같이 격자 모양 상자의 칸에 하나씩 넣어서 창고에 보관한다.



창고에 보관되는 토마토들 중에는 잘 익은 것도 있지만, 아직 익지 않은 토마토들도 있을 수 있다. 보관 후 하루가 지나면, 익은 토마토들의 인접한 곳에 있는 익지 않은 토마토들은 익은 토마토의 영향을 받아 익게 된다. 하나의 토마토의 인접한 곳은 왼쪽, 오른쪽, 앞, 뒤 네 방향에 있는 토마토를 의미한다. 대각선 방향에 있는 토마토들에게는 영향을 주지 못하며, 토마토가 혼자 저절로 익는 경우는 없다고 가정한다. 철수는 창고에 보관된 토마토들이 며칠이 지나면 다 익게 되는지, 그 최소 일수를 알고 싶어 한다.

토마토를 창고에 보관하는 격자모양의 상자들의 크기와 익은 토마토들과 익지 않은 토마토들의 정보가 주어졌을 때, 며칠이 지나면 토마토들이 모두 익는지, 그 최소 일수를 구하는 프로그램을 작성하라. 단, 상자의 일부 칸에는 토마토가 들어있지 않을 수도 있다.

입력

첫 줄에는 상자의 크기를 나타내는 두 정수 M,N이 주어진다. M은 상자의 가로 칸의 수, N은 상자의 세로 칸의 수를 나타낸다. 단, 2 ≤ M,N ≤ 1,000 이다. 둘째 줄부터는 하나의 상자에 저장된 토마토들의 정보가 주어진다. 즉, 둘째 줄부터 N개의 줄에는 상자에 담긴 토마토의 정보가 주어진다. 하나의 줄에는 상자 가로줄에 들어있는 토마토의 상태가 M개의 정수로 주어진다. 정수 1은 익은 토마토, 정수 0은 익지 않은 토마토, 정수 -1은 토마토가 들어있지 않은 칸을 나타낸다.

토마토가 하나 이상 있는 경우만 입력으로 주어진다.

출력

여러분은 토마토가 모두 익을 때까지의 최소 날짜를 출력해야 한다. 만약, 저장될 때부터 모든 토마토가 익어있는 상태이면 0을 출력해야 하고, 토마토가 모두 익지는 못하는 상황이면 -1을 출력해야 한다.

토마토 / 7576

-1에는 벽, 0에는 익지 않은 토마토, 1에는 익은 토마토가 있음

익은 토마토는 하루마다 상하좌우로 인접한
익지 않은 토마토를 익은 토마토로 바꿈

예제 입력 3 복사

```
6 4
1 -1 0 0 0 0
0 -1 0 0 0 0
0 0 0 0 -1 0
0 0 0 0 -1 1
```

예제 출력 3 복사

```
6
```

토마토 / 7576

익은 토마토는 하루마다 상하좌우로 인접한
익지 않은 토마토를 익은 토마토로 바꿈

모든 익지 않은 토마토를 익은 토마토로 바꿨을 때 걸리는 시간 출력

예제 입력 3 복사

```
6 4
1 -1 0 0 0 0
0 -1 0 0 0 0
0 0 0 0 -1 0
0 0 0 0 -1 1
```

예제 출력 3 복사

```
6
```

토마토 / 7576

0	-1	6	5	4	3
1	-1	5	4	3	2
2	3	4	5	-1	1
3	4	5	6	-1	0

예제 입력 3 복사

```
6 4
1 -1 0 0 0 0
0 -1 0 0 0 0
0 0 0 0 -1 0
0 0 0 0 -1 1
```

예제 출력 3 복사

```
6
```

토마토 / 7576

토마토들이 익기 위한 최단 시간 중 최댓값을 구해야 함

토마토들이 익기 위한 최단 시간은 익지 않은 토마토와
익은 토마토의 최단거리와 같음

최단거리?

-> BFS

토마토 / 7576

최단거리?
-> BFS

우선 익은 토마토들을 전부 큐에 넣음
익은 토마토들은 인접한 상하좌우 칸 중 벽이 아닌 칸으로 퍼져 나감
-> 모든 칸은 상하좌우로 연결 된 그래프로 생각 할 수 있음

토마토 / 7576

우선 익은 토마토들을 전부 큐에 넣음
익은 토마토들은 인접한 상하좌우 칸 중 벽이 아닌 칸으로 퍼져 나감
-> 모든 칸은 상하좌우로 연결 된 그래프로 생각 할 수 있음

방문 배열 대신 최단거리 배열을 사용하면 최단거리를 구할 수 있음

토마토 / 7576

C++

```cpp
#include <iostream>
#include <queue>
#include <algorithm>
using namespace std;
using ll = long long;

const ll MAX = 1010;
const ll INF = 1e9;
ll dx[4] = {0, 0, 1, -1}, dy[4] = {1, -1, 0, 0};
ll d[MAX][MAX], a[MAX][MAX], n, m;

class node{
public:
    ll y, x;
};
queue <node> q;
```

```cpp
// 다음에 볼 배열의 인덱스가 범위 밖이거나 벽인지 판단
bool outrange(ll cy, ll cx){
    if(cy <= 0 || cx <= 0 || cy > n || cx > m) return 1;
    return a[cy][cx] == -1;
}

void bfs(){
    while(!q.empty()){
        auto [cy, cx] = q.front(); q.pop();
        for(int i = 0;i < 4;i++){
            // 다음에 볼 인덱스
            ll ny = cy + dy[i], nx = cx + dx[i];

            // 다음에 볼 인덱스가 범위 밖이거나 벽이면 건너 뜀
            if(outrange(ny, nx)) continue;
            // 이미 방문 했으면 건너 뜀
            if(d[ny][nx] != INF) continue;
            // 다음에 볼 인덱스의 최단거리 배열을
            // 현재 최단거리 배열 +1로 바꿈
            d[ny][nx] = d[cy][cx] + 1;
            q.push({ny, nx});
        }
    }
}
```

```cpp
int main(){
    ios::sync_with_stdio(0); // fastio
    cin.tie(0), cout.tie(0); // fastio

    cin >> m >> n;
    for(int i = 1;i <= n;i++){
        for(int j = 1;j <= m;j++){
            cin >> a[i][j];
            // 매우 큰 값으로 초기화
            // 거리 배열이 INF가 아니면 방문 한 배열
            d[i][j] = INF;

            // 현재 인덱스가 익은 토마토면
            if(a[i][j] == 1){
                // 거리 배열을 0으로 바꾸고 큐에 삽입
                d[i][j] = 0;
                q.push({i, j});
            }
        }
    }

    bfs();
    ll result = 0;
    for(int i = 1;i <= n;i++){
        for(int j = 1;j <= m;j++){
            // 벽이면 건너 뜀
            if(a[i][j] == -1) continue;
            // 최단거리의 최댓값으로 갱신
            result = max(result, d[i][j]);
        }
    }

    // 모든 토마토가 익을 수 없으면 -1
    cout << (result == INF ? -1 : result);

    return 0;
}
```

토마토 / 7576

Python

```python
import sys
from collections import deque
input = sys.stdin.readline

inf = 1000000000
# 매우 큰 값으로 초기화
d = [[inf] * 1010 for i in range(1010)]
a = []
q = deque()

dx = [1, -1, 0, 0]
dy = [0, 0, 1, -1]
```

```python
# 다음에 볼 인덱스가 범위 밖이거나 벽인지 판단
def outrange(cy, cx):
    if cy < 0 or cx < 0 or cy >= n or cx >= m:
        return 1
    return a[cy][cx] == -1

def bfs():
    while len(q) != 0:
        cy, cx = q.popleft()
        for i in range(4):
            # 다음에 볼 인덱스
            ny = cy + dy[i]
            nx = cx + dx[i]

            # 다음에 볼 인덱스가 범위 밖이거나 벽이면 건너 뜀
            if(outrange(ny, nx)):
                continue
            # 이미 방문 했으면 건너 뜀
            if(d[ny][nx] != inf):
                continue
            # 다음에 볼 인덱스의 최단거리  배열을
            # 현재 최단거리 배열 +1로 바꿈
            d[ny][nx] = d[cy][cx] + 1
            q.append((ny, nx))
```

```python
m, n = list(map(int, input().rstrip().split()))
for i in range(n):
    b = list(map(int, input().rstrip().split()))
    a.append(b)

    for j in range(len(b)):
        if a[i][j] != 1:
            continue
        # 현재 인덱스가 익은 토마토면
        # 거리 배열을 0으로 바꾸고 큐에 삽입
        d[i][j] = 0
        q.append((i, j))

bfs()

result = 0
for i in range(n):
    for j in range(m):
        if a[i][j] == -1:
            continue
        result = max(result, d[i][j])

# 모든 토마토가 익을 수 없으면 -1
print(-1 if result == inf else result)
```

질문?

트리 / 1068

백준 1068 / https://www.acmicpc.net/problem/1068

문제

트리에서 리프 노드란, 자식의 개수가 0인 노드를 말한다.

트리가 주어졌을 때, 노드 하나를 지울 것이다. 그 때, 남은 트리에서 리프 노드의 개수를 구하는 프로그램을 작성하시오. 노드를 지우면 그 노드와 노드의 모든 자손이 트리에서 제거된다.

예를 들어, 다음과 같은 트리가 있다고 하자.

입력

첫째 줄에 트리의 노드의 개수 N이 주어진다. N은 50보다 작거나 같은 자연수이다. 둘째 줄에는 0번 노드부터 N-1번 노드까지, 각 노드의 부모가 주어진다. 만약 부모가 없다면 (루트) -1이 주어진다. 셋째 줄에는 지울 노드의 번호가 주어진다.

출력

첫째 줄에 입력으로 주어진 트리에서 입력으로 주어진 노드를 지웠을 때, 리프 노드의 개수를 출력한다.

트리 / 1068

M번 정점과 M번 정점의 자손을 삭제 했을 때 남은 트리에서 리프 노드의 개수를 구하는 문제

예제 입력 1 복사

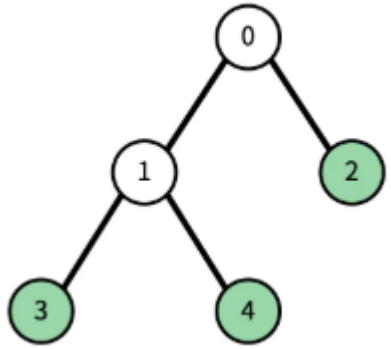
```
5
-1 0 0 1 1
2
```

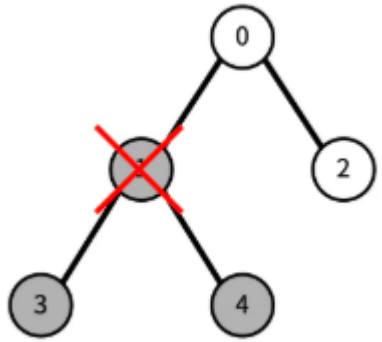
예제 출력 1 복사

```
2
```

트리 / 1068

다음 그림에서는 원래 트리는 2, 3, 4가 리프 노드로 3개
1번 정점을 삭제하면 2가 리프 노드로 1개로 바뀜





트리 / 1068

리프 노드
-> 자식 정점이 0개인 정점
-> DFS를 돌면서 자식 정점이 0개이면 정답 +1을 해주면 됨

이 문제에서는 정점 하나를 삭제 하기 때문에
DFS를 하는 현재 정점이 삭제한 정점이면 건너 뜀

자식 정점이 1개인데 삭제한 노드이면 자식 정점이 0개
-> 이것도 정답 +1을 해주면 됨

트리 / 1068

C++

```cpp
#include <iostream>
#include <vector>
using namespace std;
using ll = long long;

const ll MAX = 55;
vector <ll> a[MAX];
ll n, p[MAX], m, result, root;

void dfs(ll cur){
    // 현재 노드가 삭제된 노드면 return
    if(cur == m) return;
    for(auto& nxt : a[cur]) dfs(nxt);

    // 현재 노드의 자손이 없거나
    // 자손이 1개고 자손이 삭제된 노드면 정답 +1
    if(a[cur].empty() || (a[cur].size() == 1 && a[cur][0] == m)) result++;
}
```

```cpp
int main(){
    ios::sync_with_stdio(0); // fastio
    cin.tie(0), cout.tie(0); // fastio

    cin >> n;
    for(int i = 0;i < n;i++){
        cin >> p[i];
        // 부모가 -1이면 루트 노드
        if(p[i] == -1) root = i;
        // 아니면 간선 추가
        else a[p[i]].push_back(i);
    }
    cin >> m;

    dfs(root);
    cout << result;


    return 0;
}
```

트리 / 1068

Python

```python
import sys
input = sys.stdin.readline

n = int(input().rstrip())
p = list(map(int, input().rstrip().split()))
a = [[] for _ in range(55)]
root = 0
result = 0

def dfs(cur):
    global result
    # 현재 노드가 삭제된 노드면 return
    if cur == m:
        return
    for nxt in a[cur]:
        dfs(nxt)

    # 현재 노드의 자손이 없거나
    # 자손이 1개고 자손이 삭제된 노드면 정답 +1
    if len(a[cur]) == 0 or (len(a[cur]) == 1 and a[cur][0] == m):
        result += 1
```

```python
for i in range(len(p)):
    # 부모가 -1이면 루트 노드
    if p[i] == -1:
        root = i
    # 아니면 간선 추가
    else:
        a[p[i]].append(i)
m = int(input().rstrip())

dfs(root)
print(result)
```

질문?

기본 과제

DFS와 BFS – https://www.acmicpc.net/problem/1260

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

트리 – https://www.acmicpc.net/problem/1068

이분 그래프 - https://www.acmicpc.net/problem/1707

숨바꼭질 2 - https://www.acmicpc.net/problem/12851

심화 과제

경비행기 - https://www.acmicpc.net/problem/2585

트리의 높이와 너비 - https://www.acmicpc.net/problem/2250

고생하셨습니다

철로 / 13334

백준 13334 / https://www.acmicpc.net/problem/13334

문제

집과 사무실을 통근하는 n명의 사람들이 있다. 각 사람의 집과 사무실은 수평선 상에 있는 서로 다른 점에 위치하고 있다. 임의의 두 사람 A, B에 대하여, A의 집 혹은 사무실의 위치가 B의 집 혹은 사무실의 위치와 같을 수 있다. 통근을 하는 사람들의 편의를 위하여 일직선 상의 어떤 두 점을 잇는 철로를 건설하여, 기차를 운행하려고 한다. 제한된 예산 때문에, 철로의 길이는 d로 정해져 있다. 집과 사무실의 위치 모두 철로 선분에 포함되는 사람들의 수가 최대가 되도록, 철로 선분을 정하고자 한다.

양의 정수 d와 n 개의 정수쌍, (h_i, o_i), $1 \le i \le n$,이 주어져 있다. 여기서 h_i와 o_i는 사람 i의 집과 사무실의 위치이다. 길이 d의 모든 선분 L에 대하여, 집과 사무실의 위치가 모두 L에 포함되는 사람들의 최대 수를 구하는 프로그램을 작성하시오.

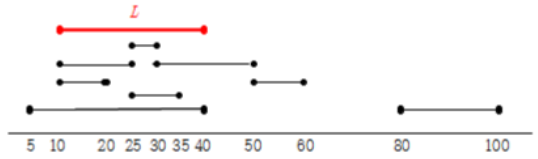


그림 1. 8 명의 집과 사무실의 위치

그림 1 에 있는 예를 고려해보자. 여기서 n = 8, $(h_1, o_1) = (5, 40)$, $(h_2, o_2) = (35, 25)$, $(h_3, o_3) = (10, 20)$, $(h_4, o_4) = (10, 25)$, $(h_5, o_5) = (30, 50)$, $(h_6, o_6) = (50, 60)$, $(h_7, o_7) = (30, 25)$, $(h_8, o_8) = (80, 100)$이고, d = 30이다. 이 예에서, 위치 10 과 40 사이의 빨간색 선분 L이, 가장 많은 사람들에 대하여 집과 사무실 위치 모두 포함되는 선분 중 하나이다. 따라서 답은 4 이다.

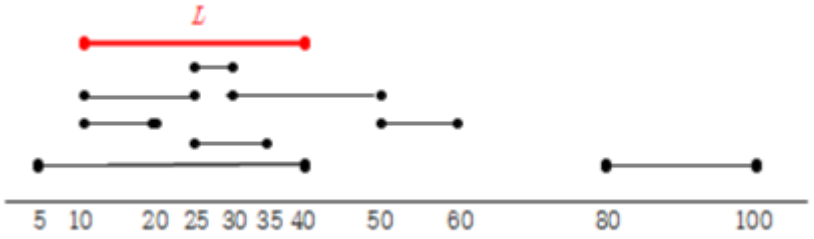
입력

입력은 표준입력을 사용한다. 첫 번째 줄에 사람 수를 나타내는 양의 정수 n ($1 \le n \le 100,000$)이 주어진다. 다음 n개의 각 줄에 정수 쌍 (h_i, o_i)가 주어진다. 여기서 h_i와 o_i는 −100,000,000이상, 100,000,000이하의 서로 다른 정수이다. 마지막 줄에, 철로의 길이를 나타내는 정수 d ($1 \le d \le 200,000,000$)가 주어진다.

출력

출력은 표준출력을 사용한다. 길이 d의 임의의 선분에 대하여, 집과 사무실 위치가 모두 그 선분에 포함되는 사람들의 최대 수를 한 줄에 출력한다.

철로 / 13334



예제 입력 1 복사

```
8
5 40
35 25
10 20
10 25
30 50
50 60
30 25
80 100
30
```

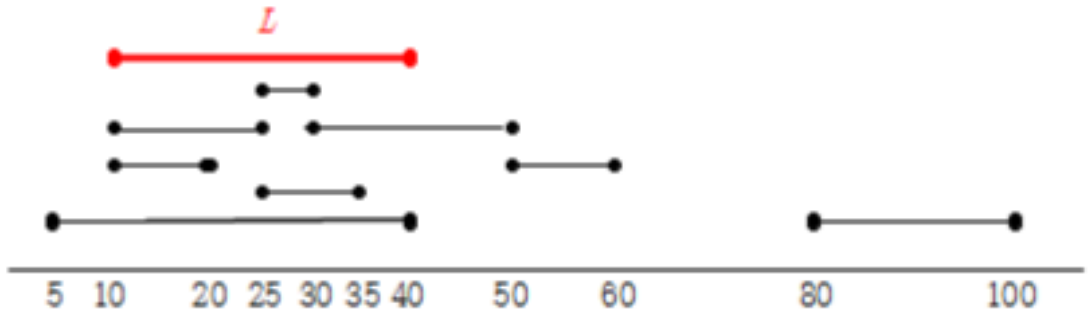
예제 출력 1 복사

```
4
```

철로 / 13334

각 구간의 시작점과 끝점이 L 안에 동시에 포함되는
개수의 최댓값을 구하면 됨

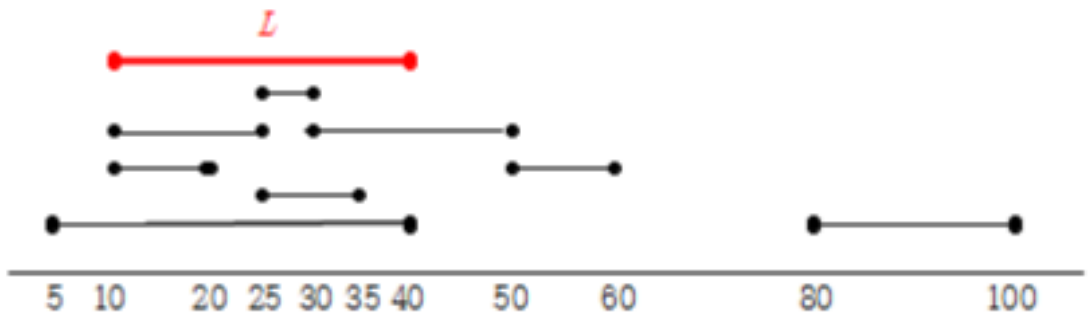
아래 그림은 예제 입력 1의 예시



철로 / 13334

우선 각 구간의 끝점을 기준으로 정렬을 해보자
항상 시작점은 끝점 보다 작음

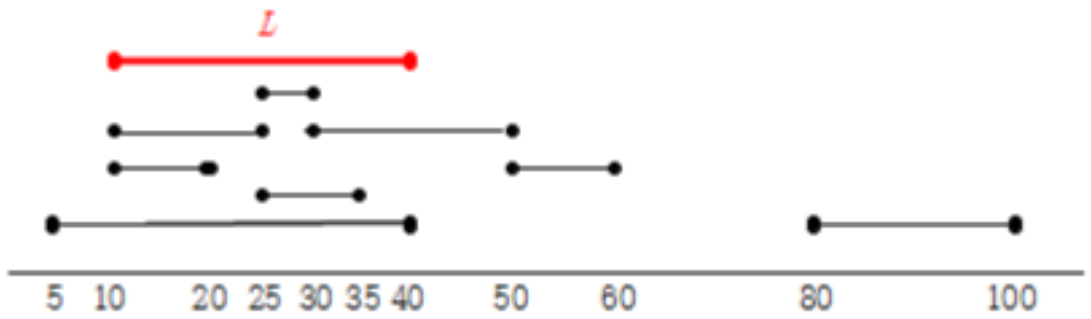
-> 현재 보고 있는 구간의 이전 값들은
-> 항상 현재 구간의 끝점 보다 작음



철로 / 13334

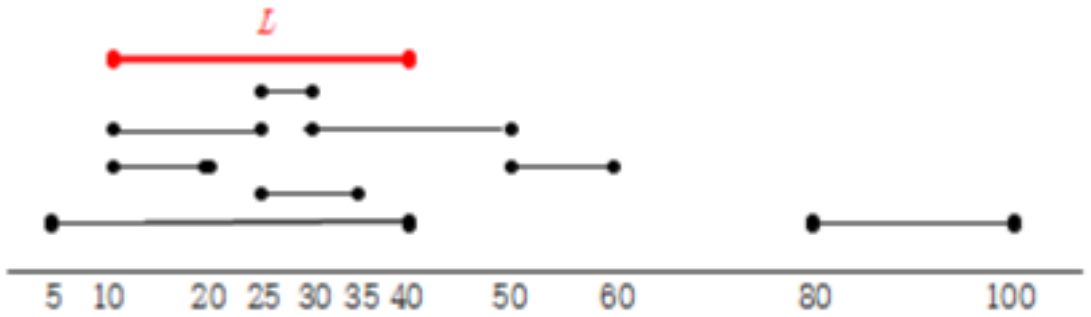
현재 보고 있는 구간의 이전 값들은
항상 현재 구간의 끝점 보다 작음

최소 우선순위 큐에 시작점을 넣어보자



철로 / 13334

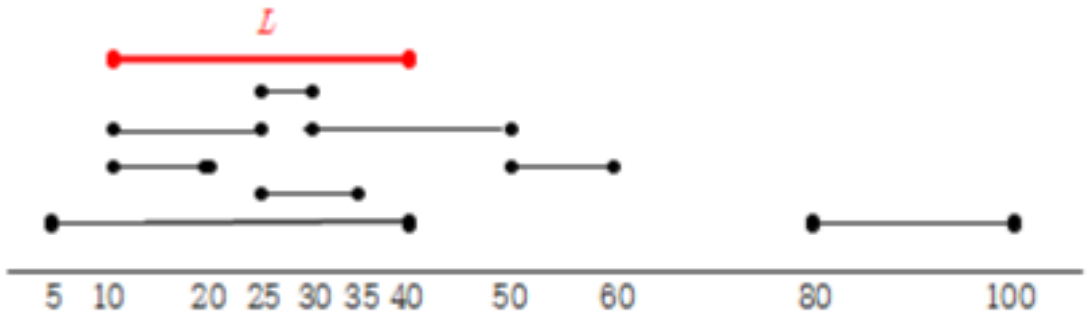
최소 우선순위 큐에 시작점을 넣어보자
-> 우선순위 큐의 값에는 시작점이 들어있음
-> 현재 보고 있는 구간의 끝점과의 차이가 L보다 작으면
-> 우선순위 큐의 값들의 끝점도 항상 L보다 작음



철로 / 13334

현재 보고 있는 구간의 끝점과의 차이가 L 이하이면
우선순위 큐의 값들의 끝점도 항상 L 이하

만약 차이가 L보다 크면 우선순위 큐에서 제거하면 됨
-> 항상 우선순위 큐에는 가능한 구간들만 들어 있음
-> 우선순위 큐의 크기의 최댓값이 정답



철로 / 13334

N = 8
(5 40) (35 25) (10 20) (10, 25)
(30 50) (50 60) (30 25) (80 100)

정답
4

끝점이 작은 순으로 정렬

철로 / 13334

N = 8
(10 20) (10 25) (30 25) (35 25)
(5 40) (30 50) (50 60) (80 100)
L = 30

정답
4

1 ~ N부터 차례대로 순회 해보자

우선순위 큐

철로 / 13334

N = 8

(10 20) (10 25) (30 25) (35 25)
(5 40) (30 50) (50 60) (80 100)

L = 30

우선순위 큐에 시작점을 넣음

10은 20과의 차이가 L 이하

-> 삭제 X, 최댓값 갱신

우선순위 큐

10

정답

1

철로 / 13334

N = 8

(10 20) (10 25) (30 25) (35 25)

(5 40) (30 50) (50 60) (80 100)

L = 30

정답

2

우선순위 큐에 시작점을 넣음

10은 25과의 차이가 L 이하

-> 삭제 X, 최댓값 갱신

우선순위 큐

철로 / 13334

N = 8

(10 20) (10 25) (30 25) (35 25)

(5 40) (30 50) (50 60) (80 100)

L = 30

정답

3

우선순위 큐에 시작점을 넣음

10은 30과의 차이가 L 이하

-> 삭제 X, 최댓값 갱신

우선순위 큐

철로 / 13334

N = 8

(10 20) (10 25) (30 25) (35 25)

(5 40) (30 50) (50 60) (80 100)

L = 30

우선순위 큐에 시작점을 넣음

10은 35과의 차이가 L 이하

-> 삭제 X, 최댓값 갱신

우선순위 큐

정답

4

철로 / 13334

N = 8

(10 20) (10 25) (30 25) (35 25)

(5 40) (30 50) (50 60) (80 100)

L = 30

정답

4

우선순위 큐에 시작점을 넣음

5는 40과의 차이가 L보다 큼

-> L의 선분 안에 시작점과 끝점이 동시에 존재 불가

우선순위 큐

철로 / 13334

N = 8

(10 20) (10 25) (30 25) (35 25)

(5 40) (30 50) (50 60) (80 100)

L = 30

우선순위 큐에 시작점을 넣음

10은 40과의 차이가 L 이하

-> 삭제 X, 최댓값 갱신

우선순위 큐

정답

4

철로 / 13334

N = 8

(10 20) (10 25) (30 25) (35 25)

(5 40) (30 50) (50 60) (80 100)

L = 30

정답

4

우선순위 큐에 시작점을 넣음

10은 50과의 차이가 L보다 큼

-> L의 선분 안에 시작점과 끝점이 동시에 존재 불가

우선순위 큐

철로 / 13334

N = 8

(10 20) (10 25) (30 25) (35 25)

(5 40) (30 50) (50 60) (80 100)

L = 30

정답

4

우선순위 큐에 시작점을 넣음

25는 60과의 차이가 L보다 큼

-> L의 선분 안에 시작점과 끝점이 동시에 존재 불가

우선순위 큐

철로 / 13334

N = 8

(10 20) (10 25) (30 25) (35 25)

(5 40) (30 50) (50 60) (80 100)

L = 30

정답

4

우선순위 큐에 시작점을 넣음

30은 100과의 차이가 L보다 큼

-> L의 선분 안에 시작점과 끝점이 동시에 존재 불가

우선순위 큐

철로 / 13334

N = 8

(10 20) (10 25) (30 25) (35 25)

(5 40) (30 50) (50 60) (80 100)

L = 30

정답

4

우선순위 큐에 시작점을 넣음

50은 100과의 차이가 L보다 큼

-> L의 선분 안에 시작점과 끝점이 동시에 존재 불가

우선순위 큐

철로 / 13334

N = 8

(10 20) (10 25) (30 25) (35 25)

(5 40) (30 50) (50 60) (80 100)

L = 30

우선순위 큐에 시작점을 넣음

80은 100과의 차이가 L 이하

-> 삭제 X, 최댓값 갱신

우선순위 큐

정답

4

철로 / 13334

C++

```cpp
#include <iostream>
#include <queue>
#include <algorithm>
using namespace std;
using ll = long long;

const ll MAX = 101010;
ll n, l, result;
pair <ll, ll> a[MAX];
priority_queue <ll, vector<ll>, greater<ll>> pq;
```

```cpp
int main(){
    ios::sync_with_stdio(0); // fastio
    cin.tie(0), cout.tie(0); // fastio

    cin >> n;
    for(int i = 1;i <= n;i++){
        // first -> 끝점
        // second -> 시작점
        cin >> a[i].first >> a[i].second;
        // 시작점이 끝점보다 크면 스왑
        if(a[i].second > a[i].first) swap(a[i].first, a[i].second);
    }

    cin >> l;
    // 끝점을 기준으로 정렬
    sort(a + 1, a + n + 1);

    for(int i = 1;i <= n;i++){
        // 현재 시작점을 우선순위 큐에 넣음
        pq.push(a[i].second);
        // 우선순위 큐의 값과 현재 구간의 끝점의 차이가
        // L보다 크면 값 삭제
        while(!pq.empty() && pq.top() < a[i].first - l) pq.pop();

        // 우선순위 큐의 크기 최댓값으로 정답 갱신
        result = max<ll>(result, pq.size());
    }

    cout << result;
    return 0;
}
```

철로 / 13334

Python

```python
import sys
import heapq
input = sys.stdin.readline

n = int(input().rstrip())
a = []
pq = []
```

```python
for _ in range(n):
    # e -> 끝점 s -> 시작점
    e, s = list(map(int, input().rstrip().split()))
    # 시작점이 끝점보다 크면 스왑
    if e < s:
        e, s = s, e
    a.append((e, s))

l = int(input().rstrip())
# 끝점을 기준으로 정렬
a.sort()
result = 0

for e, s in a:
    # 현재 시작점을 우선순위 큐에 넣음
    heapq.heappush(pq, s)
    # 우선순위 큐의 값과 현재 구간의 끝점의 차이가
    # L보다 크면 값 삭제
    while len(pq) != 0 and pq[0] < e - l:
        heapq.heappop(pq)
    # 우선순위 큐의 크기 최댓값으로 정답 갱신
    result = max(result, len(pq))

print(result)
```

질문?

소가 길을 건너간 이유 4 / 14464

백준 14464 / https://www.acmicpc.net/problem/14464

문제

농부 존의 소들은 효율적으로 길을 건너는 방법을 터득하고 있다. 그들은 길 건너기의 달인인 닭의 도움을 받기로 했다.

안타깝게도 닭은 매우 바쁜 동물이라, 소를 도와줄 시간이 별로 없다. 농장에 C마리($1 \le C \le 20,000$)의 닭이 있고, 1번부터 C번까지 번호가 붙어 있다. i번 닭은 정확히 T_i초에만 소를 도와줄 수 있다. 하지만 닭은 길 건너기의 달인이므로 소를 데리고도 순식간에 길을 건널 수 있다. 소는 할 일이 없어서 여유롭게 길을 건널 수 있다. 소는 총 N마리($1 \le N \le 20,000$)가 있고, 마찬가지로 1번부터 N번까지 번호가 붙어 있다. j번 소는 A_j초부터 B_j초까지 길을 건널 수 있다. j번 소가 i번 닭의 도움을 받아 길을 건너려면 $A_j \le T_i \le B_j$를 만족해야 한다.

소는 최대 한 마리의 닭에게만 도움을 받을 수 있고, 닭 역시 최대 한 마리의 소만 도와줄 수 있다. 도움을 받을 수 있는 소가 최대 몇 마리인지 구해보자.

입력

첫 줄에 C와 N이 주어진다. 다음 C줄에는 $T_1 \ldots T_C$가 주어지고, 그 다음 N줄에는 A_j와 $B_j (A_j \le B_j)$가 주어진다. A, B, T는 모두 최대 1,000,000,000인 음이 아닌 정수이고, 같을 수도 있다.

출력

도움을 받을 수 있는 소가 최대 몇 마리인지 출력한다.

소가 길을 건너간 이유 4 / 14464

N의 크기를 가진 배열 A, M개의 구간이 주어짐
각 배열의 값을 M개의 구간에 1대1 매칭을 시켰을 때
매칭의 최댓값을 구하면 됨

예제 입력 1 복사

```
5 4
7
8
6
2
9
2 5
4 9
0 3
8 13
```

예제 출력 1 복사

```
3
```

소가 길을 건너간 이유 4 / 14464

일단 배열 A와 M개의 구간을 정렬 해보자
구간은 시작점을 기준으로 정렬

일단 정렬을 하고 보면 A[i]를 구간에 매칭 시킬 때
매칭 시킬 수 있는 구간 중에서 끝 점이 빠른 구간과
매칭 시키는 게 이득임

소가 길을 건너간 이유 4 / 14464

매칭 시킬 수 있는 구간 중에서 끝 점이 빠른 구간과
매칭 시키는 게 이득임

끝 점이 느린 구간은 뒤에 값들과 매칭이 될 수 있음
EX) 2 5 의 구간은 7과 매칭 될 수 없지만
2 7 의 구간은 7과 매칭 될 수 있음

소가 길을 건너간 이유 4 / 14464

끝 점이 제일 빠른 구간과 매칭
-> 우선순위 큐로 관리 하면
-> 삽입/삭제 O(log N), 반환 O(1)

최대한 매칭을 많이 하는 법은 알았는데
매칭이 가능한지 판단하는 법?

소가 길을 건너간 이유 4 / 14464

최대한 매칭을 많이 하는 법은 알았는데
매칭이 가능한지 판단하는 법?

이미 배열과 구간 모두 정렬 함
-> 구간을 차례대로 보면서 시작 점이 배열의 값보다
-> 작으면 매칭이 가능 함
우선순위 큐에 끝 점 삽입

소가 길을 건너간 이유 4 / 14464

이미 배열과 구간 모두 정렬 함
-> 구간을 차례대로 보면서 시작 점이 배열의 값보다
-> 작으면 매칭이 가능 함
우선순위 큐에 끝 점 삽입

그러면 우선순위 큐에 있는 값들은
모두 배열의 값보다 시작 점이 빠름
-> 우선순위 큐의 값이 현재 배열 값보다 빠르면 제거
-> 아니면 매칭 할 수 있으므로 제일 끝 점이 빠른 값과 매칭

소가 길을 건너간 이유 4 / 14464

5 4
7 8 6 2
2 5, 4 9, 0 3, 8 13

출력
3

정렬을 해봅시다

소가 길을 건너간 이유 4 / 14464

5 4 정답
2 6 7 8 0
0 3, 2 5, 4 9, 8 13
현재 배열의 값이 2, 시작점이 0이므로 매칭 가능
-> 우선순위 큐에 삽입

우선순위 큐

소가 길을 건너간 이유 4 / 14464

5 4 정답

2 6 7 8 0

0 3, 2 5, 4 9, 8 13

현재 배열의 값이 2, 시작점이 2이므로 매칭 가능

-> 우선순위 큐에 삽입

우선순위 큐

소가 길을 건너간 이유 4 / 14464

5 4 정답

2 6 7 8 1

0 3, 2 5, 4 9, 8 13

현재 배열의 값이 2, 시작점이 4이므로 매칭 불가

-> 배열의 값과 현재 매칭 가능한 구간 중

-> 끝점이 가장 작은 값과 매칭

우선순위 큐

소가 길을 건너간 이유 4 / 14464

5 4 정답

2 6 7 8 1

0 3, 2 5, 4 9, 8 13

현재 배열의 값이 6, 시작점이 4이므로 매칭 가능

-> 우선순위 큐에 삽입

우선순위 큐

소가 길을 건너간 이유 4 / 14464

5 4 정답

2 6 7 8 1

0 3, 2 5, 4 9, 8 13

현재 배열의 값이 6, 시작점이 8이므로 매칭 불가

-> 우선순위 큐의 값과 현재 배열 값 매칭

-> 하지만 현재 우선순위 큐의 값이 배열의 값보다 작음

우선순위 큐

소가 길을 건너간 이유 4 / 14464

5 4 정답
2 6 7 8 2
0 3, 2 5, 4 9, 8 13
현재 배열의 값이 6, 시작점이 8이므로 매칭 불가
-> 우선순위 큐의 값과 현재 배열 값 매칭

우선순위 큐

소가 길을 건너간 이유 4 / 14464

5 4 정답
2 6 7 8 2
0 3, 2 5, 4 9, 8 13
현재 배열의 값이 7, 시작점이 8이므로 매칭 불가
-> 우선순위 큐의 값과 현재 배열 값 매칭
-> 우선순위 큐의 값이 없으니까 넘어 감
우선순위 큐

소가 길을 건너간 이유 4 / 14464

5 4 정답

2 6 7 8 2

0 3, 2 5, 4 9, 8 13

현재 배열의 값이 8, 시작점이 8이므로 매칭 가능

-> 우선순위 큐에 삽입

우선순위 큐

13

소가 길을 건너간 이유 4 / 14464

5 4 정답
2 6 7 8 3
0 3, 2 5, 4 9, 8 13
현재 배열의 값이 8, 시작점이 8이므로 매칭 가능
-> 우선순위 큐의 값과 현재 배열 값 매칭

우선순위 큐

소가 길을 건너간 이유 4 / 14464

C++

```cpp
#include <iostream>
#include <algorithm>
#include <queue>
using namespace std;
using ll = long long;

const ll MAX = 20101;
ll n, m, a[MAX];
pair <ll, ll> b[MAX];
priority_queue <ll, vector<ll>, greater<ll>> pq;
```

```cpp
int main(){
    ios::sync_with_stdio(0); // fastio
    cin.tie(0), cout.tie(0); // fastio

    cin >> n >> m;
    for(int i = 1;i <= n;i++) cin >> a[i];
    for(int i = 1;i <= m;i++) cin >> b[i].first >> b[i].second;
    // 정렬
    sort(a + 1, a + n + 1);
    sort(b + 1, b + m + 1);

    // idx -> 현재 보는 구간의 인덱스
    ll idx = 1, result = 0;
    for(int i = 1;i <= n;i++){
        // 현재 구간을 끝까지 안봤고
        // 구간의 시작점이 배열의 값 이하면
        while(idx <= m && b[idx].first <= a[i]){
            // 우선순위 큐에 끝점 삽입
            pq.push(b[idx].second);
            idx++; // 보는 구간의 인덱스 1 증가
        }

        // 우선순위 큐의 값이 현재 배열의 값보다 작으면
        // 매칭 할 수 없으므로 삭제
        while(!pq.empty() && pq.top() < a[i]) pq.pop();
        // 우선순위 큐가 비어있지 않으면
        if(!pq.empty()){
            // 매칭 가능
            result++; pq.pop();
        }
    }

    cout << result;

    return 0;
}
```

소가 길을 건너간 이유 4 / 14464

Python

```python
import sys
import heapq
input = sys.stdin.readline

n, m = list(map(int, input().rstrip().split()))
a = []
for _ in range(n):
    x = int(input().rstrip())
    a.append(x)

b = []
for _ in range(m):
    l, r = list(map(int, input().rstrip().split()))
    b.append((l, r))

# 정렬
a.sort()
b.sort()

idx = 0 # 현재 보는 구간의 인덱스
result = 0
pq = []
```

```python
for i in a:
    # 현재 구간을 끝까지 안봤고
    # 구간의 시작점이 배열의 값 이하면
    while idx < m and b[idx][0] <= i:
        # 우선순위 큐에 삽입
        heapq.heappush(pq, b[idx][1])
        idx += 1 # 보는 구간의 인덱스 1 증가

    # 우선순위 큐의 값이 현재 배열의 값보다 작으면
    # 매칭 할 수 없으므로 삭제
    while len(pq) != 0 and pq[0] < i:
        heapq.heappop(pq)

    # 우선순위 큐가 비어있지 않으면
    if len(pq) != 0:
        # 매칭 가능
        result += 1
        heapq.heappop(pq)

print(result)
```

질문?

고생하셨습니다