Graphs

지금까지 배운 것

- 지금까지 배운 자료구조를 떠올려 보자
- 배열, 스택, 큐, 덱, ...
- 이 자료구조들의 특징은 무엇일까?

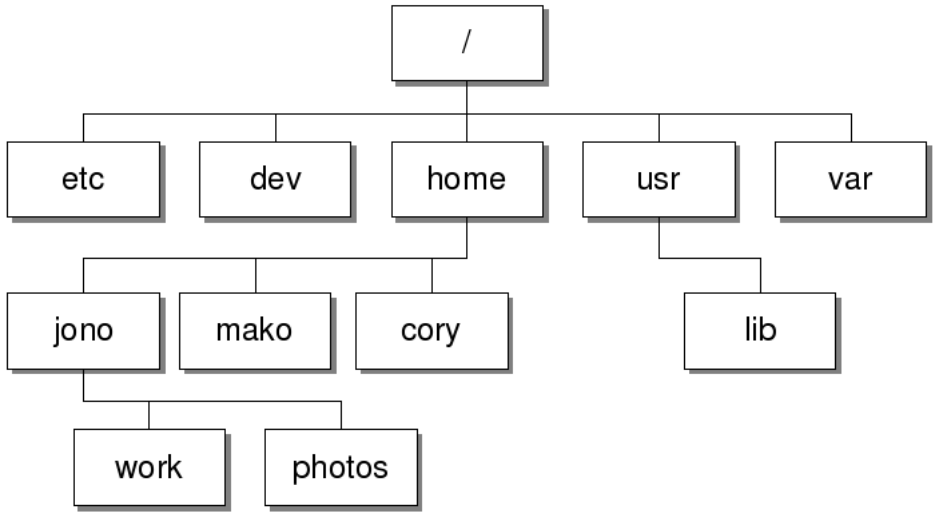
선형 자료구조와 비선형 자료구조

- **Linear Datastructure**
- 하나의 자료 앞-뒤로 한 개의 자료만 존재한다
- 스택, 큐, 덱 모두 하나의 자료 뒤에 한 개의 원소만 존재할 수 있다

- **Non-linear Datastructure**
- 한 개의 원소 뒤에 여러 개의 원소가 존재할 수 있을까? 어떤 것이 있을까?

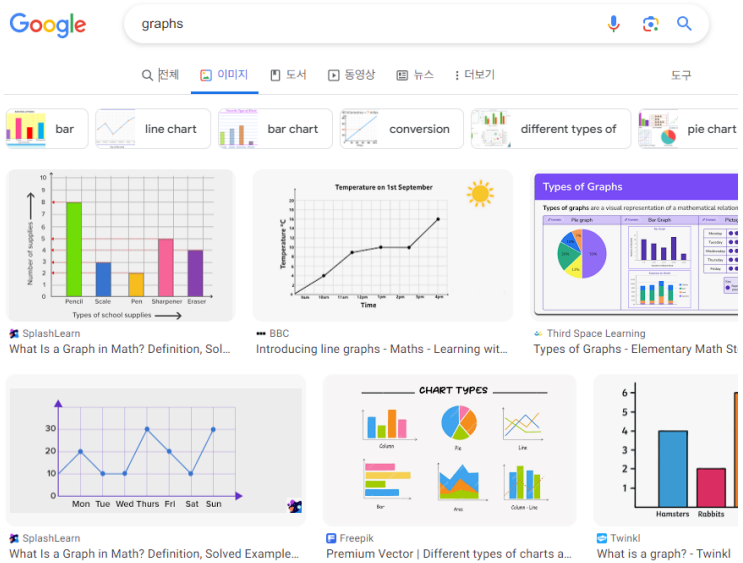
선형 자료구조와 비선형 자료구조

- 자료끼리 연관이 있지만, 그 관계가 1:1이 아닌 경우



그래프

- 컴퓨터 과학에서 그래프는 무엇을 의미할까?
- 기존에 우리가 알고있던 수학적인 그래프
- $f\colon X \to Y$인 함수 f의 그래프, $x \in X$인 x에 대해서 $\big(x, f(x)\big)$을 시각적으로 나타낸다

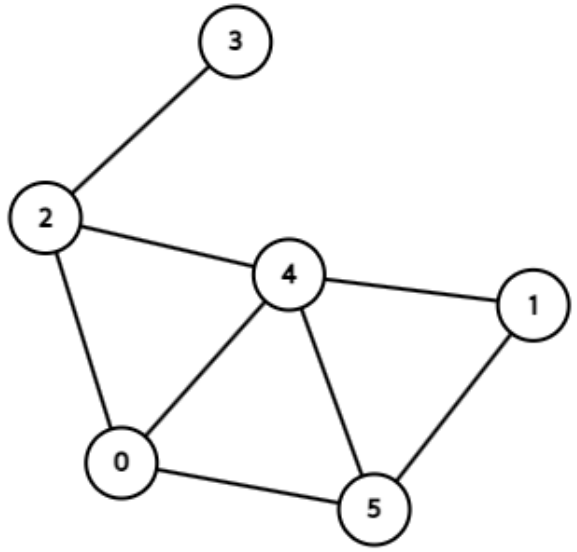


그래프

- 이산수학 및 컴퓨터 과학에서의 그래프

$$G(V, E)$$

- 그래프 G는 정점(Node, Vertex)의 집합 V와 간선(Edge)의 집합 E로 이루어져 있다
- 하나의 정점에는 여러 정점이 이어질 수 있음
- **비선형 자료구조**



그래프의 종류

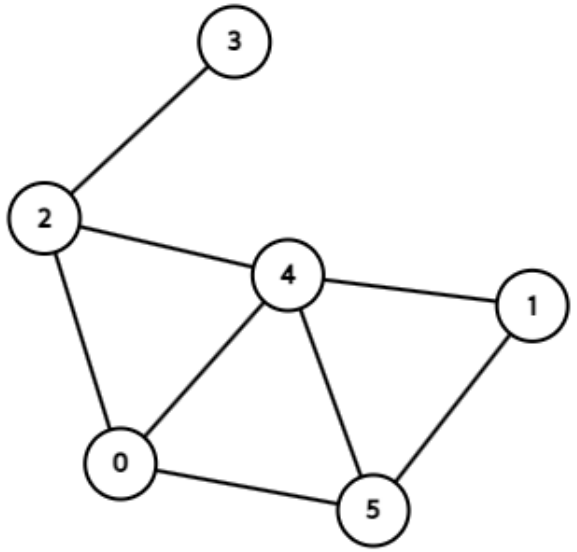
- **단방향 그래프**: 하나의 간선 (i, j)는 $i \to j$만을 의미하며, $j \to i$를 의미하지 않는다.
- **양방향 그래프**: 하나의 간선 (i, j)는 $i \to j$와 $j \to i$ 모두 의미한다.
- **가중치 그래프**: 간선에 **가중치**라는 정보가 추가된다, (i, j, w)는 $i \to j$인 가중치 w 간선이라는 의미이다. 이는 양방향, 단방향 그래프 모두 사용될 수 있다.
- **연결 그래프**: 모든 정점끼리 간선을 통해 이동할 수 있는 그래프

특별한 그래프의 종류

- **트리**: 사이클이 존재하지 않는 연결 그래프
- **DAG**: 사이클이 존재하지 않는 방향 그래프
- **포레스트**: 트리의 집합
- **이분 그래프**: 정점을 두 그룹으로 나눌 수 있는 그래프, 각 그룹은 다른 그룹으로만 간선을 이을 수 있다
- **완전 그래프**: 서로 다른 두 정점을 잇는 간선이 모두 존재하는 그래프

그래프를 나타내는 방법

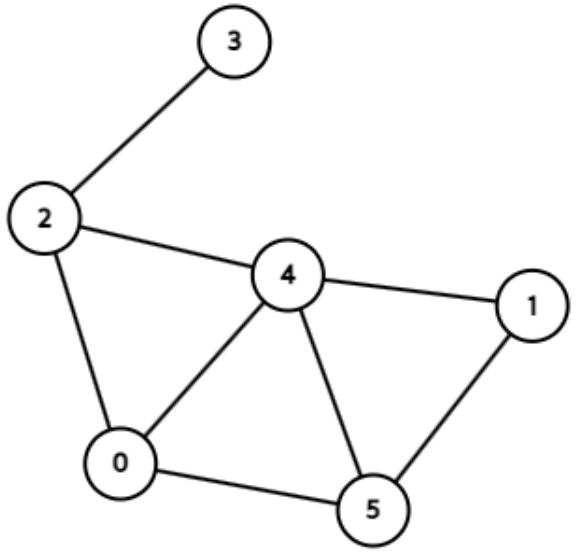
- **인접 행렬 (Adjacency matrix)**
- 그래프를 $N \times N$ 행렬 A로 나타내고, $A_{i,j}$가 1인 경우 $i \rightarrow j$로 이동할 수 있다는 의미
- 직관적이지만, 정점이 많아질수록 낭비되는 공간이 많아진다, $O(|V|^2)$의 공간 사용



$$\begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

그래프를 나타내는 방법

- **인접 리스트 (Adjacency list)**
- 노드를 나타내는 리스트에 연결된 다른 노드들을 저장
- 각 리스트를 가변 리스트로 구현해 메모리 낭비를 최소화, $O(|V| + |E|)$의 공간



0		2	4	5	
1		4	5		
2		0	3		
3		2			
4		0	1	2	5
5		0	1	4	

그래프를 나타내는 방법

```
int graph[1001][1001];
for(int i = 0; i < V; i++) {
    cin >> a >> b >> w;
    graph[a][b] = w;
    graph[b][a] = w;
}
```

```
vector<int> graph[1001];
for(int i = 0; i < V; i++) {
    // a - b 간 간선이 있음
    cin >> a >> b;
    graph[a].push_back(b);
    graph[b].push_back(a);
}
```

```
vector<pair<int, int>> graph[1001];
for(int i = 0; i < V; i++) {
    // a - b 간 가중치 w인 간선이 있음
    cin >> a >> b >> w;
    graph[a].emplace_back(b, w);
    graph[b].emplace_back(a, w);
}
```

그래프 순회, 탐색

- 한 정점에서 시작해서, 도달할 수 있는 정점들을 어떤 **전략에 따라 순서대로 방문**
- 전위 순회, 중위 순회, 후위 순회
- 깊이 우선 탐색, 너비 우선 탐색

- 배운 자료구조를 바탕으로 깊이 우선 탐색과 너비 우선 탐색의 원리를 알아보자

깊이 우선 탐색 (DFS)

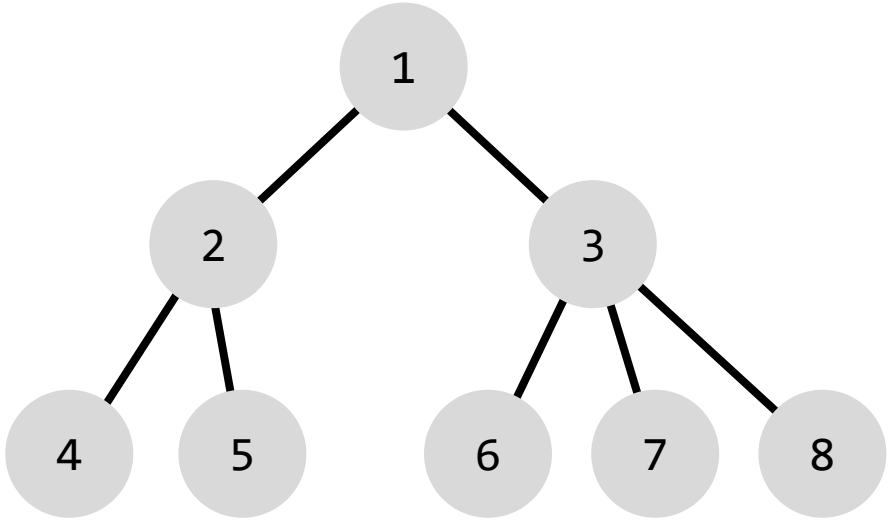
- 임의의 정점에서 시작해서, 인접한 정점 중 방문하지 않은 정점을 방문
- 방문한 정점에서 다시 탐색
- 인접한 정점 중 방문하지 않은 정점이 없다면, 이전 정점으로 돌아가기

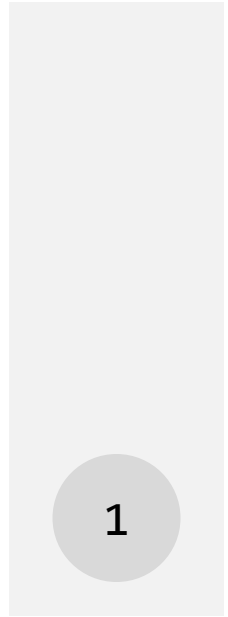
깊이 우선 탐색 (DFS)

- 임의의 정점에서 시작해서, 인접한 정점 중 방문하지 않은 정점을 방문
- 방문한 정점에서 다시 탐색
- 인접한 정점 중 방문하지 않은 정점이 없다면, 이전 정점으로 돌아가기

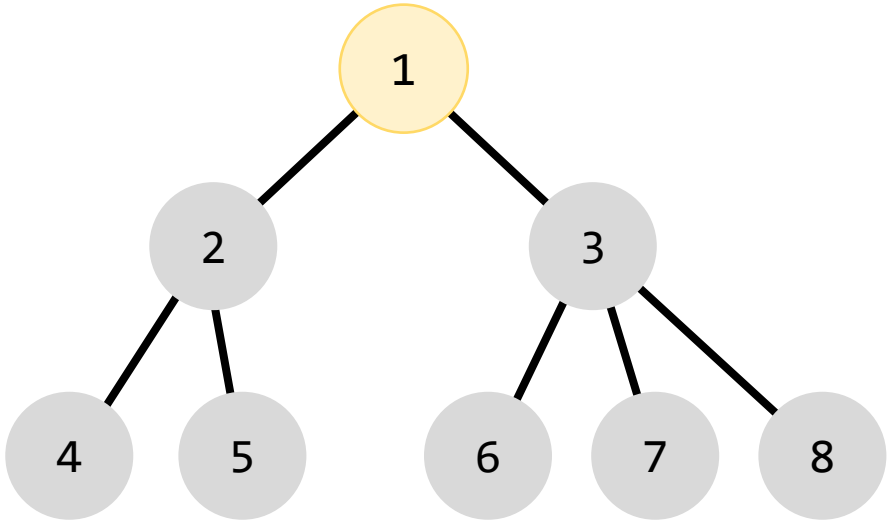
- 방문한 정점에서 똑같이 진행하기 때문에 재귀호출을 사용
- 재귀 스택과 동일하게, 자료구조 스택을 활용해 구현할 수도 있음

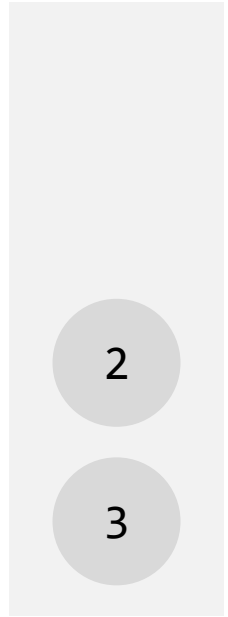
깊이 우선 탐색 (DFS)



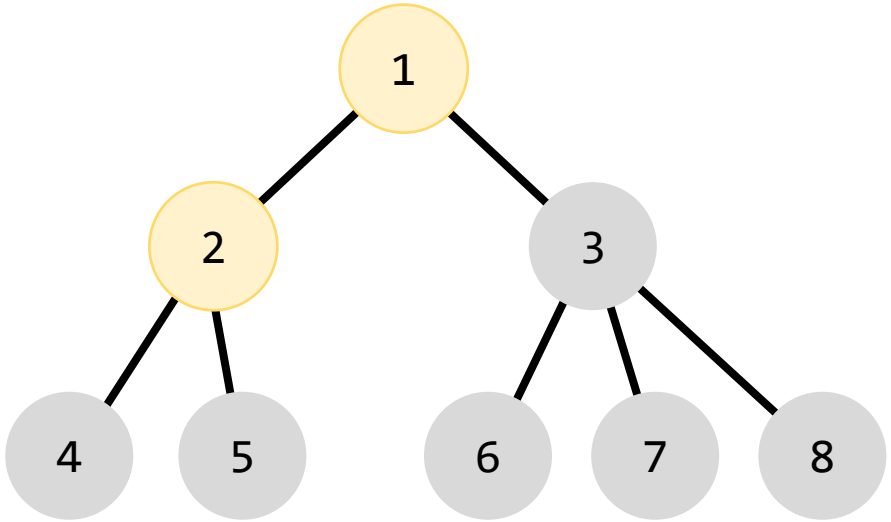


깊이 우선 탐색 (DFS)

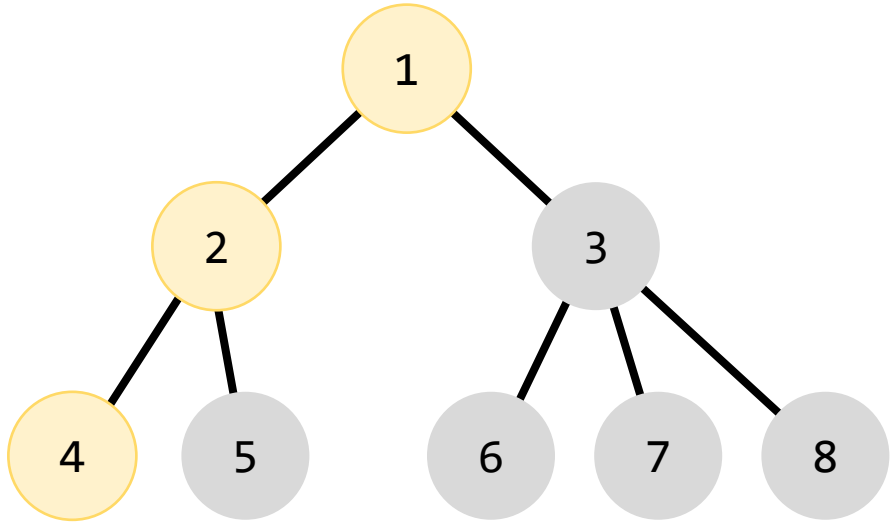


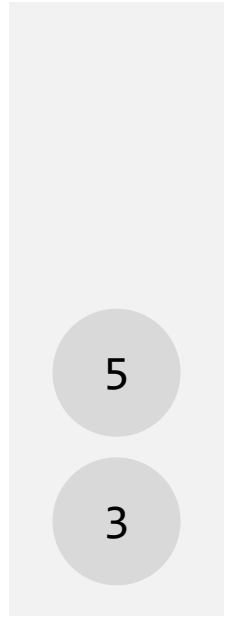


깊이 우선 탐색 (DFS)

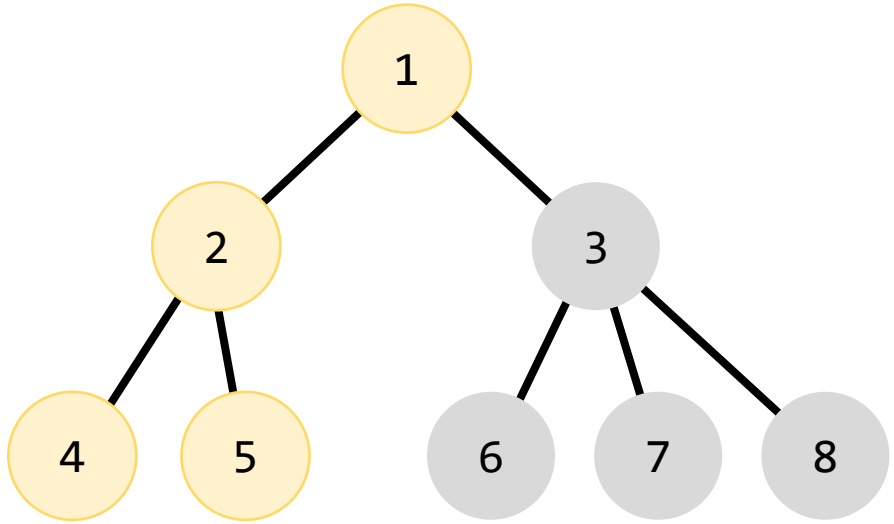


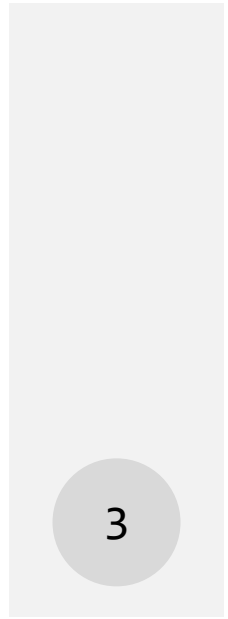
깊이 우선 탐색 (DFS)



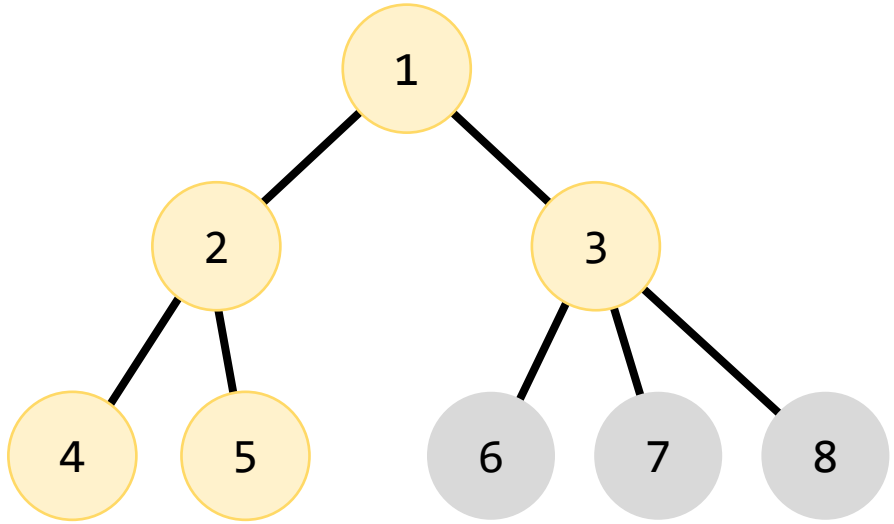


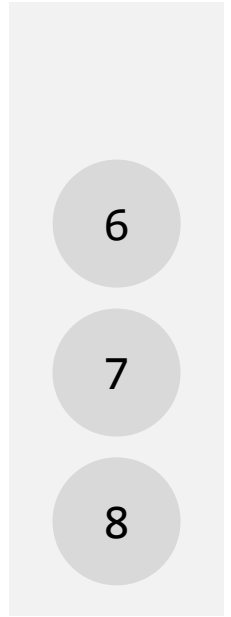
깊이 우선 탐색 (DFS)



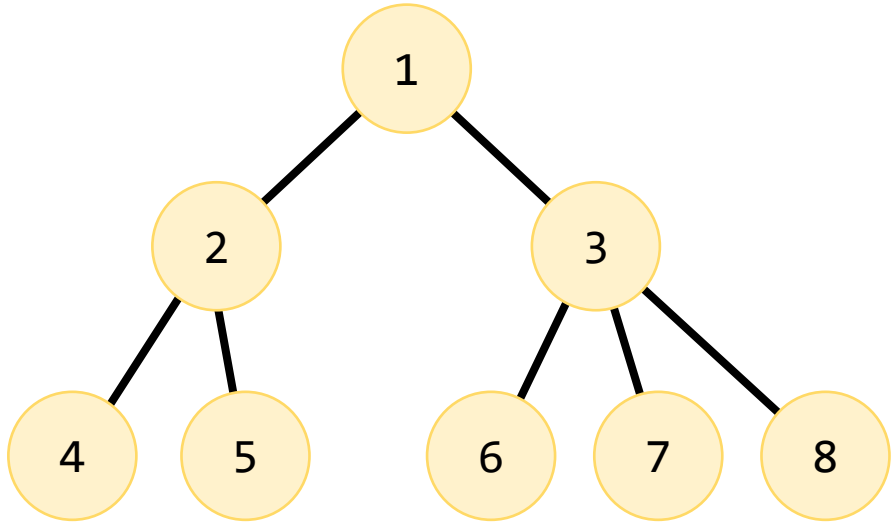


깊이 우선 탐색 (DFS)





깊이 우선 탐색 (DFS)



깊이 우선 탐색 (DFS)

```
void dfs(int cur){
    v[cur] = 1;
    for(auto nxt : g[cur]){
        if (v[nxt]) continue;
        dfs(nxt);
    }
}
```

```
void stack_dfs(int start){
    stk.push(start);
    while(!stk.empty()){
        int cur = stk.top(); stk.pop();
        v[cur] = 1;
        for(auto nxt : g[cur]){
            if (v[nxt]) continue;
            stk.push(nxt);
        }
    }
}
```

너비 우선 탐색 (BFS)

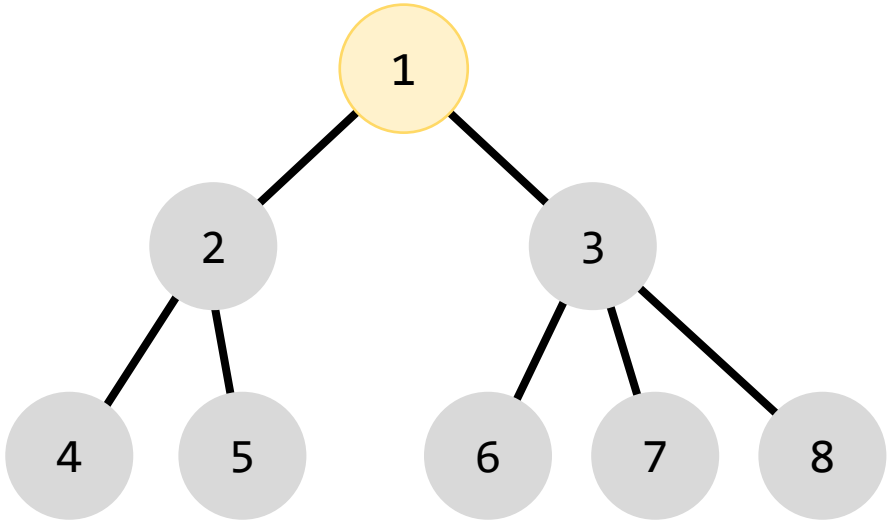
- 임의의 정점에서 시작해서, 인접한 정점 중 방문하지 않은 **모든 정점들을 방문**
- 이후에 방문할 정점들은 앞에서 방문해야하는 정점들을 모두 방문한 뒤 처리

너비 우선 탐색 (BFS)

- 임의의 정점에서 시작해서, 인접한 정점 중 방문하지 않은 **모든 정점들을 방문**
- 이후에 방문할 정점들은 앞에서 방문해야하는 정점들을 모두 방문한 뒤 처리

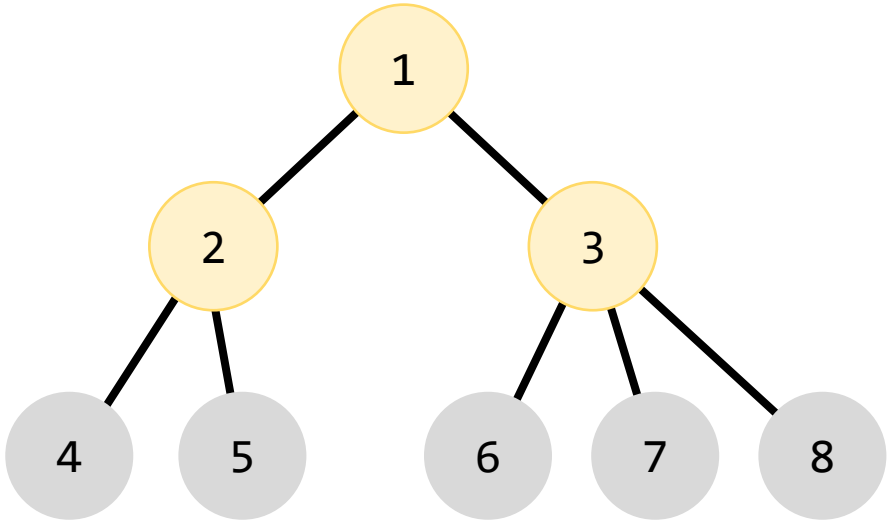
- 현재 정점과 인접한 정점 중 방문하지 않은 정점이 모두 다음에 방문해야 한다
- 단, 이미 방문해야 하는 게 있다면 기존 대기열 정점을 모두 완료한 다음 진행한다
- 자료구조 큐를 사용해서 구현

너비 우선 탐색 (BFS)

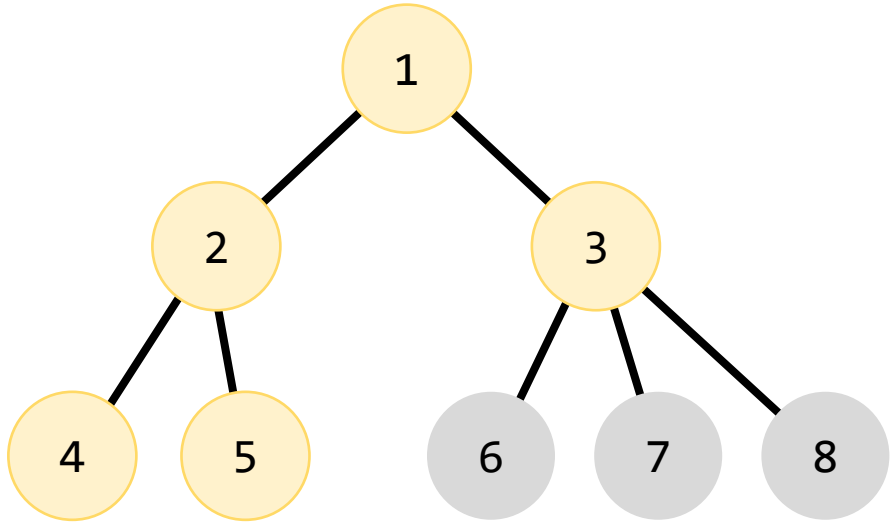


1

너비 우선 탐색 (BFS)



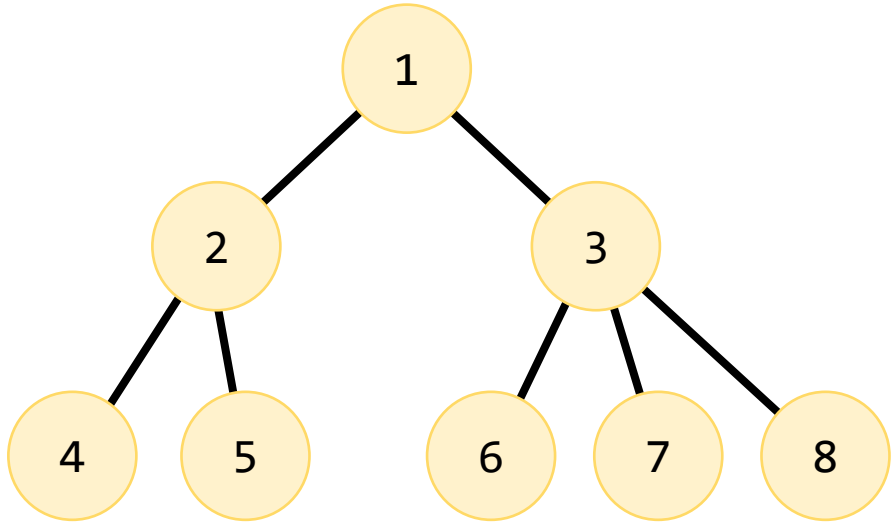
너비 우선 탐색 (BFS)



2

3 4 5

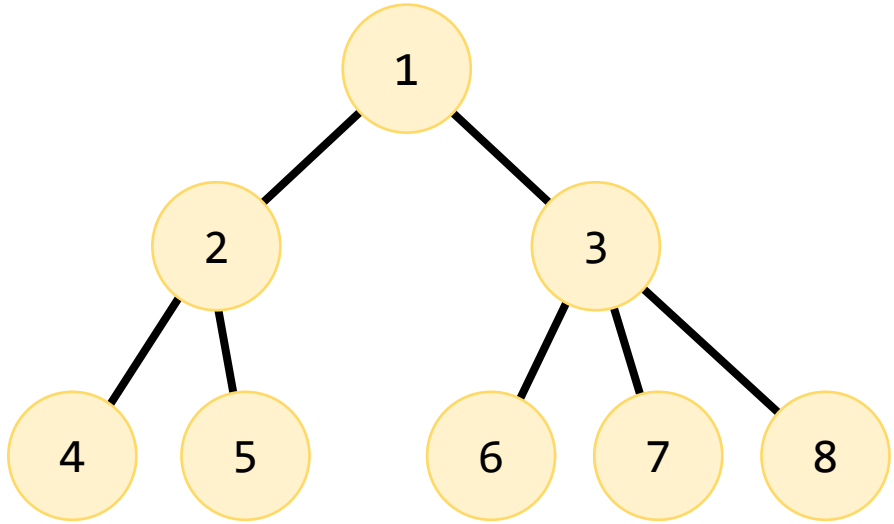
너비 우선 탐색 (BFS)



3

4 5 6 7 8

너비 우선 탐색 (BFS)

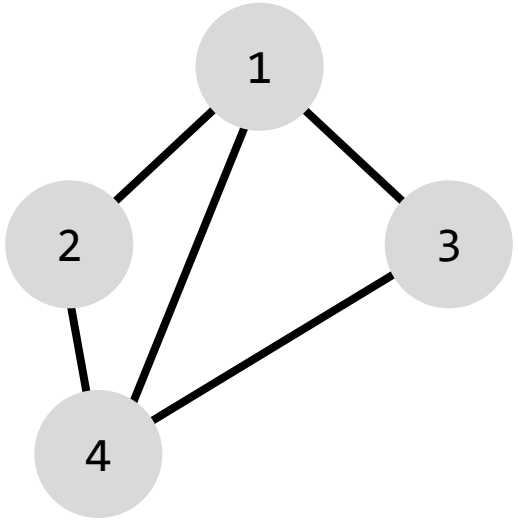


너비 우선 탐색 (BFS)

```
void bfs(int start){
    queue<int> q;
    v[start] = 1;
    q.push(start);
    while(!q.empty()){
        int cur = q.front(); q.pop();
        for(auto nxt : g[cur]){
            if (v[nxt]) continue;
            v[nxt] = 1;
            q.push(nxt);
        }
    }
}
```

큐에 넣을 때 방문 체크하자

- 사이클이 존재하는 그래프의 경우, 정점에 방문할 때 체크하면 문제가 발생
- 큐에 하나의 정점이 여러 개 들어가게 된다, 메모리 초과 가능성



바이러스 BOJ 2606

- 컴퓨터의 연결 상태가 주어졌을 때, 1번 컴퓨터가 바이러스에 걸릴 때, 최종적으로 바이러스에 감염되는 컴퓨터의 개수 구하기 (1번 컴퓨터 제외)

- DFS와 BFS를 통해 얻을 수 있는 정보에 대해서 알아보자

바이러스 BOJ 2606

- 컴퓨터의 연결 상태가 주어졌을 때, 1번 컴퓨터가 바이러스에 걸릴 때, 최종적으로 바이러스에 감염되는 컴퓨터의 개수 구하기 (1번 컴퓨터 제외)

- 탐색을 한 번 진행하면, 연결된 모든 정점을 한 번 방문한다
- 방문한 정점의 개수를 따로 기록할 수 있는 방법은 무엇일까?

바이러스 BOJ 2606

- 만약 1번 정점에서 시작하는 DFS가 종료됐을 때, 방문돼 있다면 서로 연결된 것이다

```
…
dfs(1);
for(int i = 1; i <= N; i++) ans += v[i];
cout << ans - 1 << '\n';
```

연결 요소의 개수 BOJ 11724

- 그래프가 총 몇 개의 연결 요소 (Connected Component)로 이루어져 있을까?
- 두 정점이 서로 연결돼 있다면, 같은 연결 요소에 속한다
- 한 번의 그래프 탐색을 통해 얻을 수 있는 정보는 무엇일까?

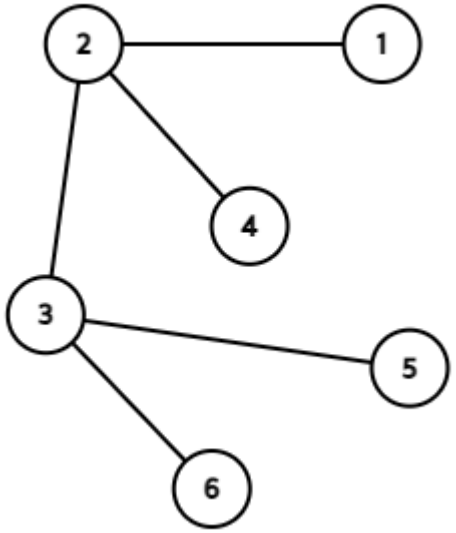
연결 요소의 개수 BOJ 11724

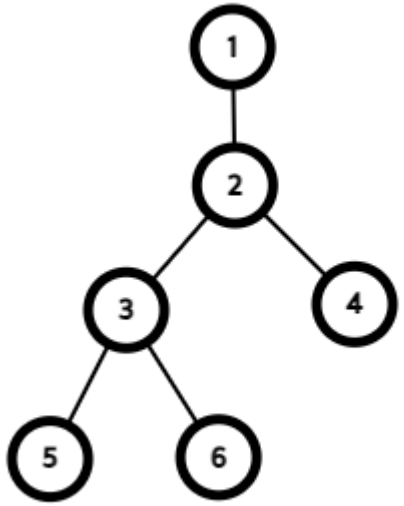
- 그래프가 총 몇 개의 연결 요소 (Connected Component)로 이루어져 있을까?
- 두 정점이 서로 연결돼 있다면, 같은 연결 요소에 속한다
- 그래프 탐색을 1회 진행하면 그 정점과 연결된 모든 노드를 방문한다
- 그래프 탐색을 진행한 횟수가 곧 연결 요소의 개수

```
for(int i = 1; i <= N; i++){
    if (v[i]) continue;
    ans++;
    bfs(i); // or dfs(i)
}
```

트리의 부모 찾기 BOJ 11725

- **트리**가 주어지고 1번 정점이 루트라고 했을 때, 각 정점들의 부모 구하기
- 트리의 위, 아래를 우리가 강제할 수 있을까?





격자를 그래프로 모델링하기 BOJ 1012

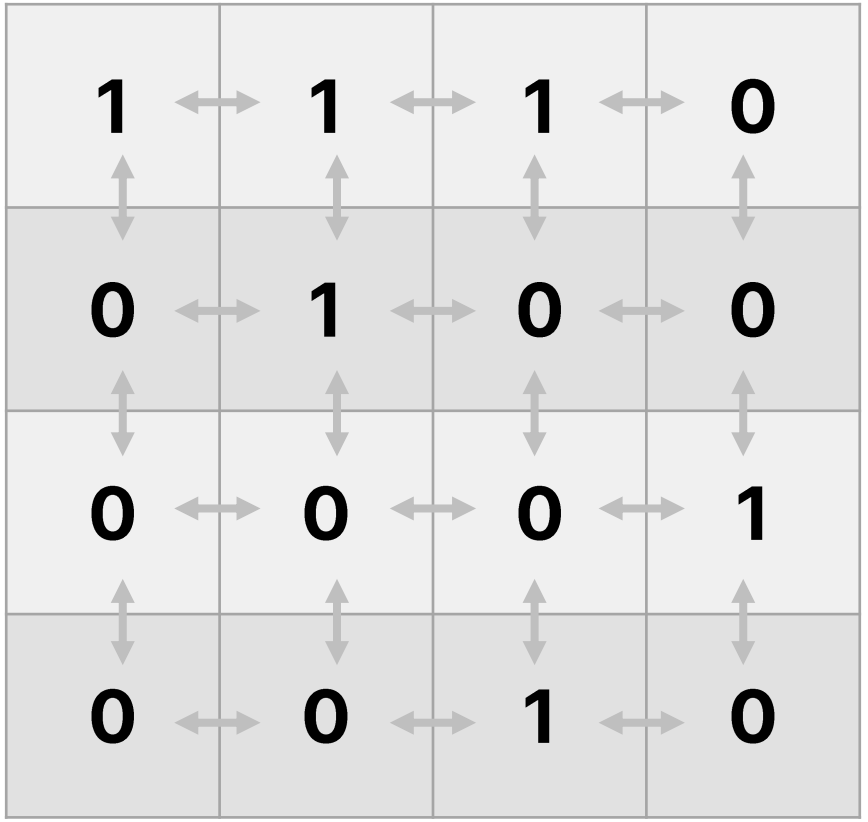
- 조금 더 실생활에 가까운 문제 지문 상황을 **그래프로 모델링**해 보자
- 서로 이어져 있는 1의 덩어리 수 (연결된 정점의 개수) 구하기

1	**1**	0	0	0	0	0	0	0	0
0	**1**	0	0	0	0	0	0	0	0
0	0	0	0	**1**	0	0	0	0	0
0	0	0	0	**1**	0	0	0	0	0
0	0	**1**	**1**	0	0	0	**1**	**1**	**1**
0	0	0	0	**1**	0	0	**1**	**1**	**1**

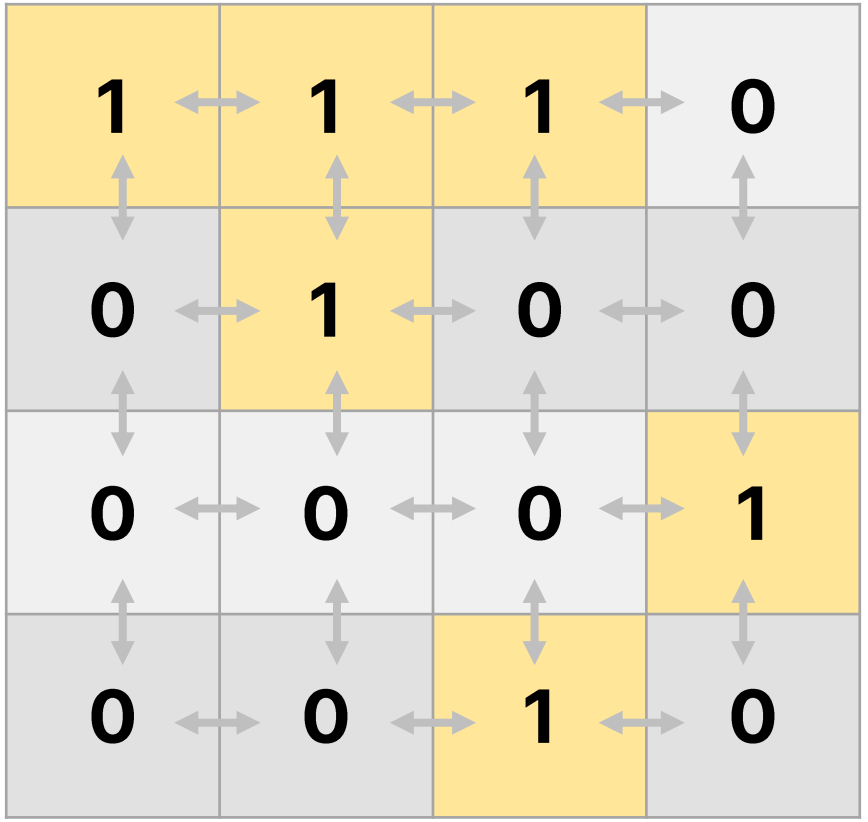
유기농 배추 BOJ 1012

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

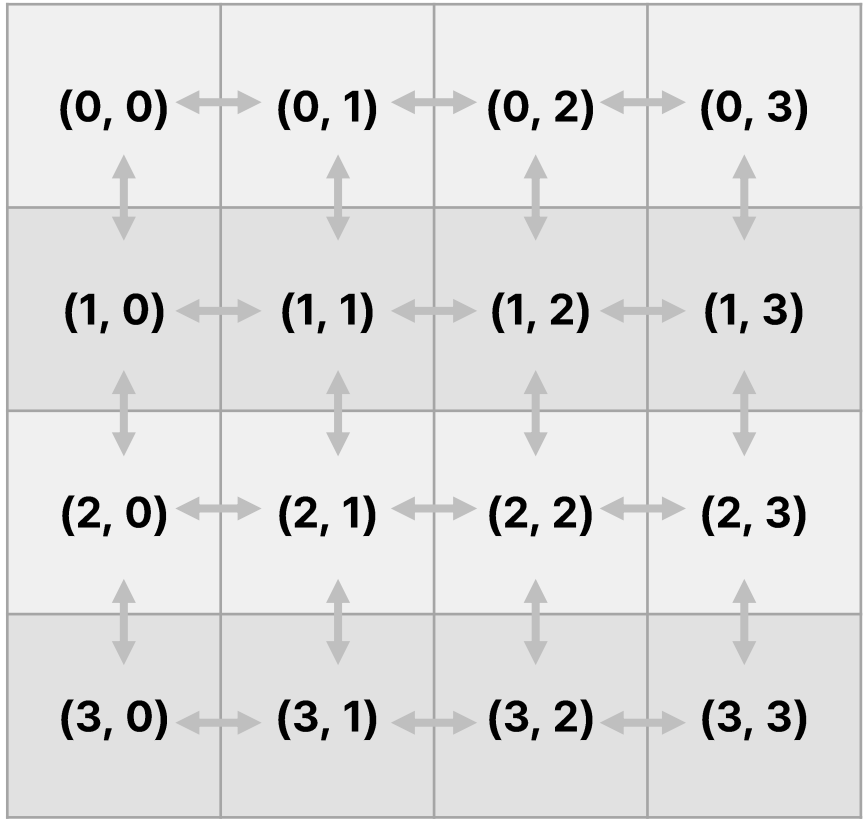
유기농 배추 BOJ 1012



유기농 배추 BOJ 1012

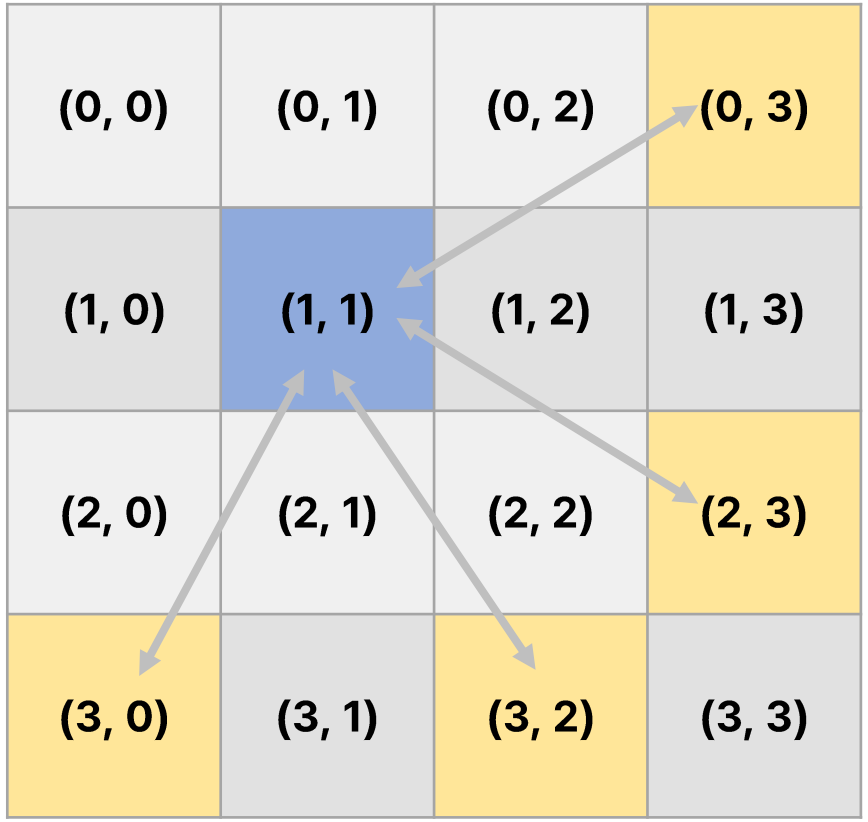


유기농 배추 BOJ 1012




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

for(int i = 0; i < 4; i++){
    int tx = x + dx[i];
    int ty = y + dy[i];
    if (tx < 0 || ty < 0 || tx >= M || ty >= N)
        continue;

    // further traversal below

}
```

나이트 이동 BOJ 7562



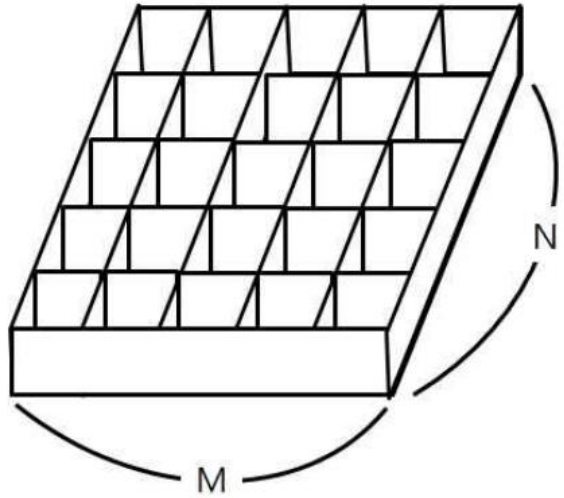

```
int dx[8] = {-2, -1, 1, 2, -2, -1, 1, 2};
int dy[4] = {-1, -2, -2, -1, 1, 2, 2, 1};

for(int i = 0; i < 8; i++){
    int tx = x + dx[i];
    int ty = y + dy[i];
    if (tx < 0 || ty < 0 || tx >= M || ty >= N)
        continue;

    // further traversal below

}
```

BFS와 최단 거리

- 주어진 간선의 **가중치가 모두 같을 때**, BFS를 사용해서 특정 정점으로부터 다른 정점까지의 최단거리를 구할 수 있다
- 그래프가 주어졌을 때, 정점 A에서 B까지 가는 데 거쳐야하는 간선의 개수
- BFS에서는 큐를 활용해서 방문 순서를 관리한다, 시작 정점부터 가까운 순서대로 방문
- 특정 시점에 큐에 넣게 되는 후보 정점들은 현재 정점보다 1만큼 먼 곳에 존재한다

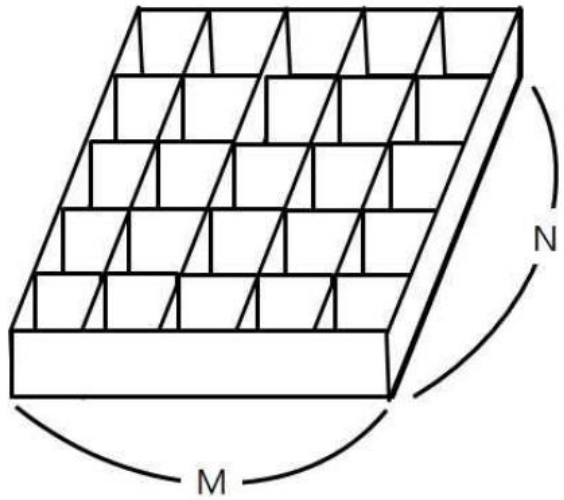
토마토 BOJ 7576

- 격자 모양의 토마토 판에서, 익은 토마토는 하루가 지나면 상하좌우로 인접한 다른 토마토들을 익게 한다
- 모든 토마토들이 익기 위해서는 얼마만큼의 시간이 걸릴까?
- 토마토가 들어있지 않은 칸이 존재할 수 있다
- 모든 토마토가 익는 시간이 존재하지 않을 수도 있다
- $N, M \leq 1\,000$



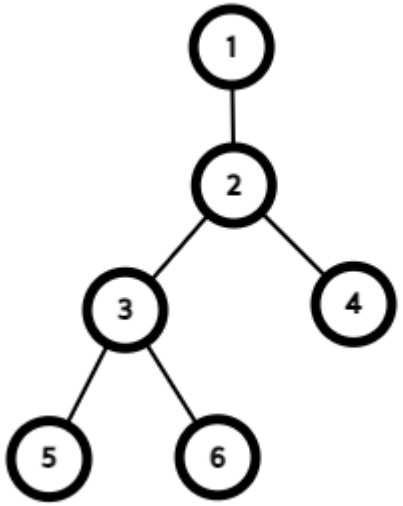
토마토 BOJ 7576

- 하나의 익은 토마토 칸에서 시작해서 퍼져 나가는 형태
- Multi-Source BFS
- 처음 큐에 익은 토마토를 모두 넣어 두고, BFS를 돌린다
- BFS의 특성 상, 시작 칸의 방문을 1로, 새로 방문하는 칸을 현재 칸의 방문 수 + 1로 기록하면 최단 거리를 구할 수 있다
- 방문 체크, 빈 칸인 경우를 잘 생각하자



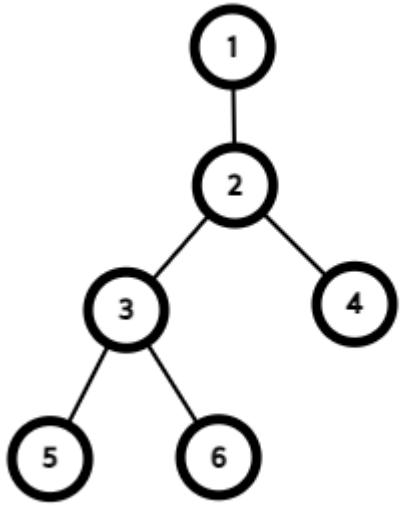
특정 거리의 도시 찾기 BOJ 18352

- **그래프**와 시작 정점 번호가 주어질 때, 최단 거리가 K인 정점들을 찾아보자
- 아래와 같은 그래프에서 1번 정점에서 출발한다면 $K = 2$인 경우, $3, 4$번 정점이 해당



트리의 지름 BOJ 1967

- **트리**가 주어질 때, 가장 먼 두 정점 쌍 구하기
- 아래 그래프에서는 $(1, 5)$가 정답이 될 수 있음, 지름 3
- 주어지는 그래프는 가중치 그래프



트리의 지름 BOJ 1967

- 어떤 점에서 가장 먼 점은 지름의 한 쪽 끝 점이다
- 지름의 한 쪽 끝에서 가장 먼 점은 다른 지름의 끝 점이다
- 두 명제가 성립한다면, 두 번의 그래프 탐색을 통해 지름을 구할 수 있다

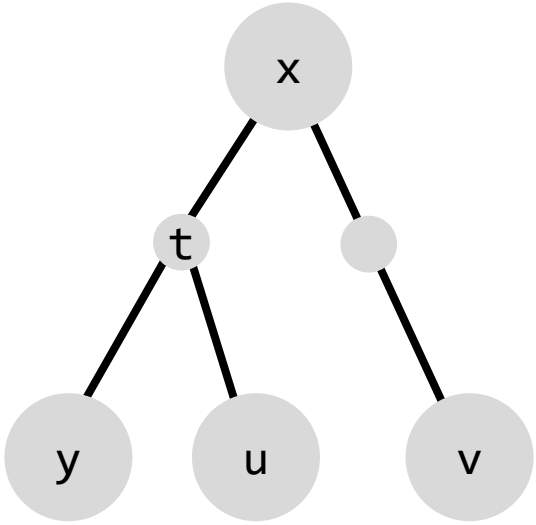
- 두 번째 문장은 자명하다, 첫 번째 문장이 성립함을 어떻게 보일 수 있을까?

트리의 지름 BOJ 1967

- 트리의 어떤 정점 x에서 **가장 먼 정점**을 y라고 하자
- 트리의 지름을 이루는 양 끝 정점을 u, v라고 하자
- 만약 y가 u, v에 속한다면 트리의 지름을 올바르게 구한다.
- 속하지 않는 경우, 경우의 수는 두 가지이다
- $x \leftrightarrow y$ 경로에 속하는 한 개 이상의 정점이 $u \leftrightarrow v$ 경로에 포함되는 경우
- $x \leftrightarrow y$ 경로에 속하는 정점 모두가 $u \leftrightarrow v$ 경로에 포함되지 않는 경우

트리의 지름 BOJ 1967

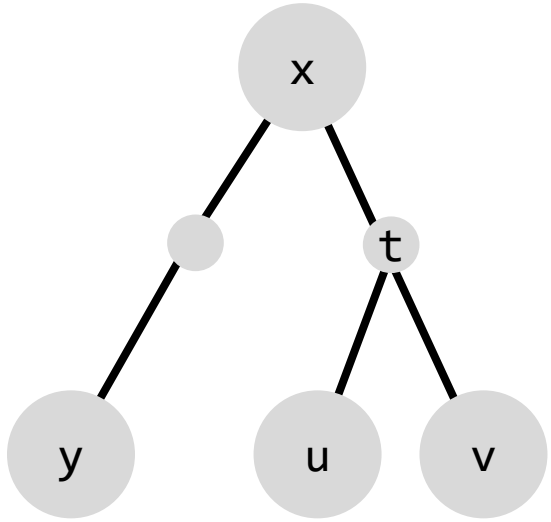
- $x \leftrightarrow y$ 경로 중 정점 한 개 이상이 $u \leftrightarrow v$ 경로에 포함되는 경우
- x에서 가장 먼 점이 y이다. 처음으로 갈라지는 정점 t에서,
- $d(t, y) \geq d(t, u)$이므로, $d(y, v)$가 $d(u, v)$보다 크다
- u, v가 지름의 양 끝 점이라는 것에 모순이다



- $d(a, b)$ = 정점 a에서 b까지의 최단 거리

트리의 지름 BOJ 1967

- $x \leftrightarrow y$ 경로에 속하는 정점이 $u \leftrightarrow v$ 경로에 포함되지 않는 경우
- $d(x, y) \geq d(x, u)$, x에서 가장 먼 정점이 y이다
- $d(y, t) \geq d(u, t)$, 작은 우변에서 떼어내서 좌변에 더했다
- $d(y, v) \geq d(u, v)$, u, v가 트리의 지름이라는 것에 모순이다



트리의 지름 BOJ 1967

- 따라서 트리가 주어지면, 임의의 정점에서 가장 먼 정점 u를 찾는다
- u에서 가장 먼 정점 v를 찾는다
- u, v는 트리의 지름의 양 끝 점이다

- 가중치 그래프이므로 BFS를 통한 접근이 불가능하다. DFS를 통해서 진행한다

트리의 지름 BOJ 1967

```
void dfs(int cur, int d) {
    v[cur] = d;
    for(auto [nxt, w] : g[cur]) {
        if (v[nxt]) continue;
        dfs(nxt, d+w);
    }
}
```

```
dfs(1, 1);
auto idx = max_element(v, v+N+1) - v;
memset(v, 0, sizeof v);
dfs(idx, 1);
cout << *max_element(v, v+N+1) - 1 << '\n';
```

벽 부수고 이동하기 BOJ 2206

- (N, M)의 공간에서 $(1, 1)$에서 출발할 때, (N, M)까지 도달하는 최단 거리를 구하자
- 단, 1로 표시된 공간은 벽이 있는 공간이며, 최대 한 개의 벽을 뚫을 수 있다
- ??

벽 부수고 이동하기 BOJ 2206

- (N, M)의 공간에서 $(1, 1)$에서 출발할 때, (N, M)까지 도달하는 최단 거리를 구하자
- 단, 1로 표시된 공간은 벽이 있는 공간이며, 최대 한 개의 벽을 뚫을 수 있다

- 그래프로 어떻게 모델링할 지 잘 생각해 보자
- 벽을 뚫은 시점을 기준으로 전과 후는 서로 같은 상태라고 볼 수 있을까?

벽 부수고 이동하기 BOJ 2206

- (N, M)의 공간에서 $(1, 1)$에서 출발할 때, (N, M)까지 도달하는 최단 거리를 구하자
- 단, 1로 표시된 공간은 벽이 있는 공간이며, 최대 한 개의 벽을 뚫을 수 있다

- 격자상에서 1을 만나는 경우, 벽을 뚫은 상태가 되며, 더 이상 뚫을 수 없게 된다
- 격자를 두 층으로 나타낸 뒤, 0층은 벽을 뚫지 않은 상태, 1층은 벽을 뚫지 않은 상태로 나타내면 다음과 같은 3차원 배열을 만들 수 있다: $v[k][i][j]$

벽 부수고 이동하기 BOJ 2206

- (N, M)의 공간에서 $(1, 1)$에서 출발할 때, (N, M)까지 도달하는 최단 거리를 구하자
- 단, 1로 표시된 공간은 벽이 있는 공간이며, 최대 한 개의 벽을 뚫을 수 있다

- 이동할 때 걸리는 가중치는 모두 1이므로, BFS를 사용해서 최단 거리를 구해낼 수 있다. 0층에 있을 때 1을 만나는 경우, 다음 층으로 이동하되 거리가 증가하지 않도록 해야 한다.

References

- https://github.com/justiceHui/SSU-SCCC-Study/blob/master/2023-summer-basic/slide/09-3-graph-traversal.pdf
- https://blog.naver.com/jinhan814/222084804068