Graph

Sogang ICPC Tea

2020 Winter 초급

서강대학교 강효규

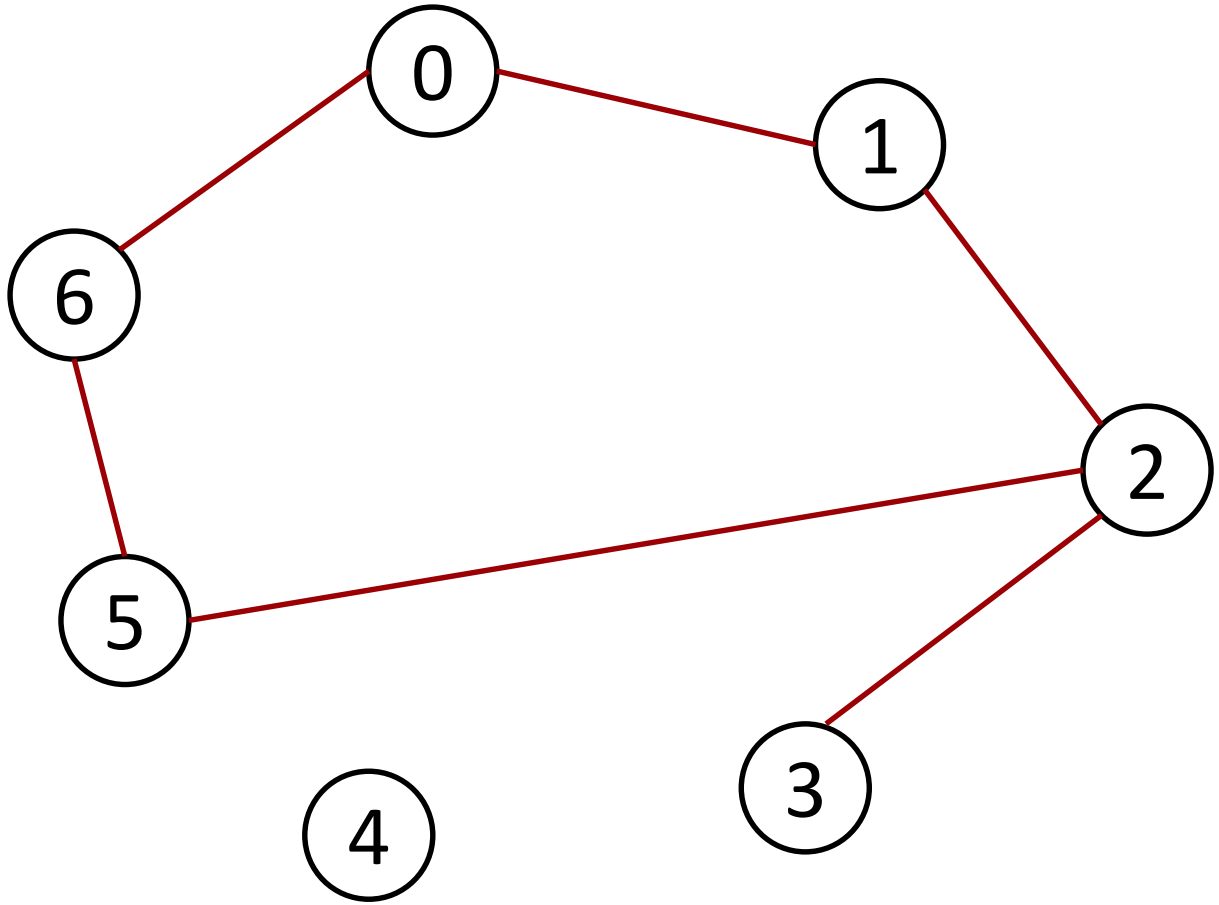
Graph

✓ 정점($vertex$)과 간선($edge$)의 집합으로 정의된 자료구조

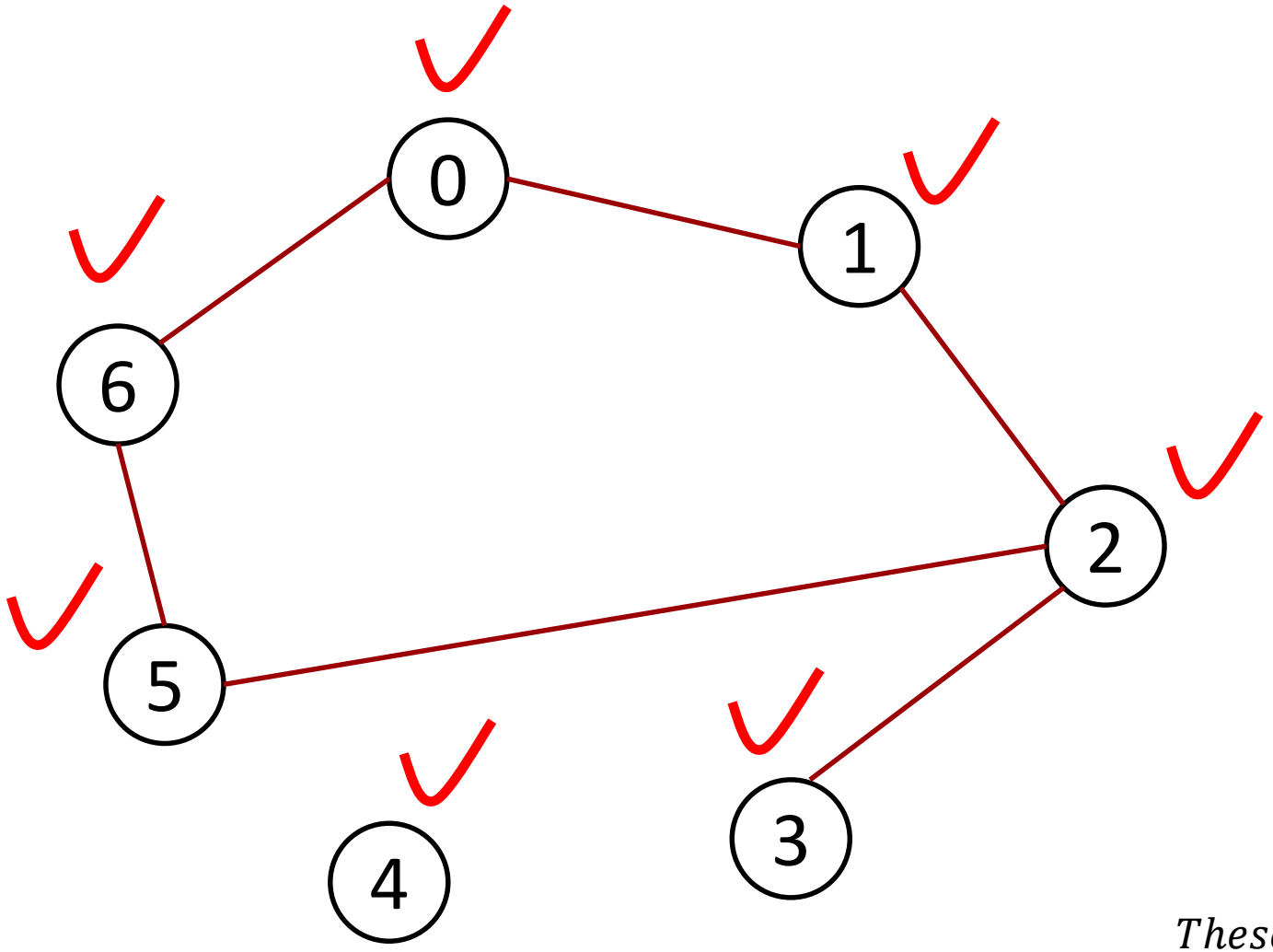
✓간선은 정점과 정점을 연결해주고 간선을 통해 연결된 정점은 '인접'해 있다고 한다.

✓$G = (V, E)$로 표현

Graph

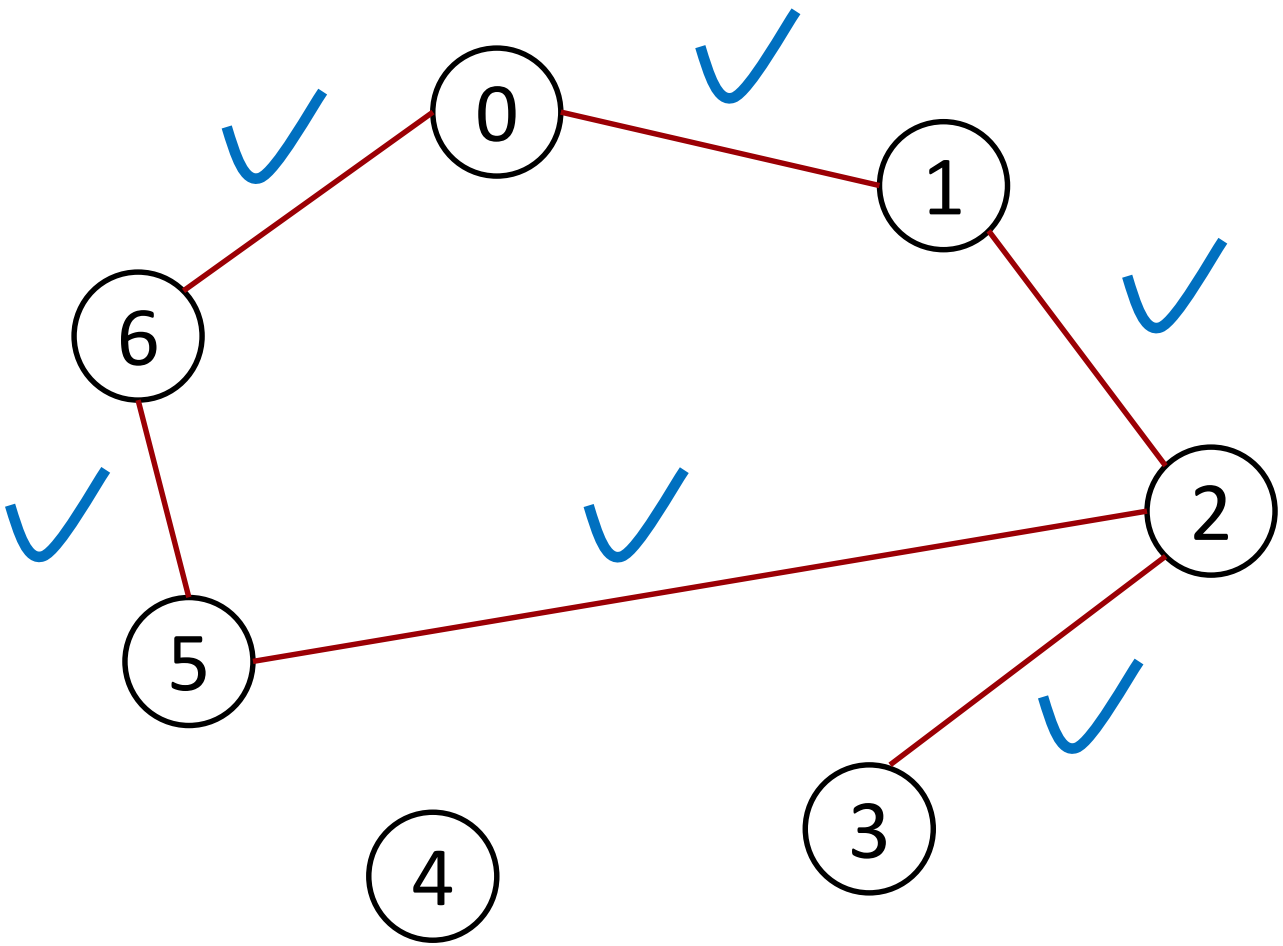


Graph



These are vertexes

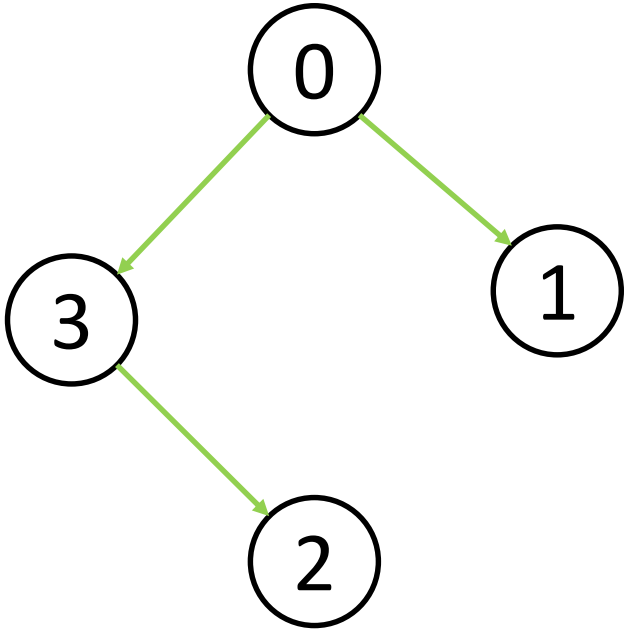
Graph



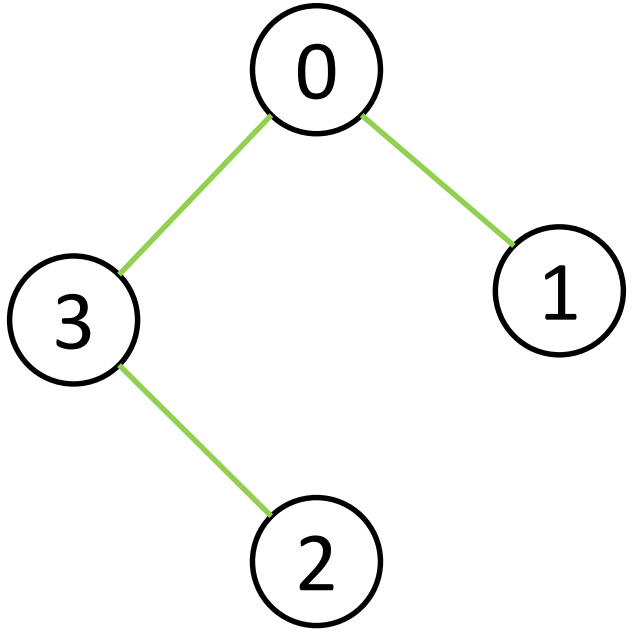
These are edges

Graph

그래프는 크게 두 가지로 분류된다.



directed graph (단방향 그래프)



undirected graph (양방향 그래프)

Graph

✓ $degree$(차수)

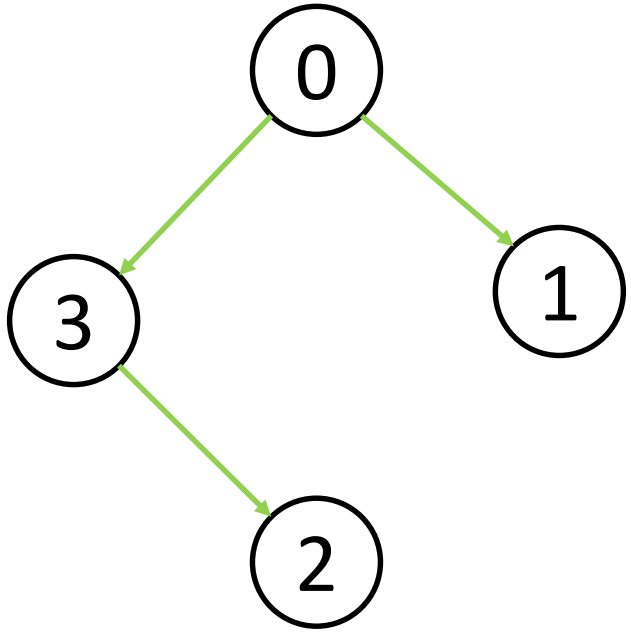
정점에 연결된 간선의 수
방향그래프에서는 indegree, outdegree로 나뉜다.

✓ $indegree$

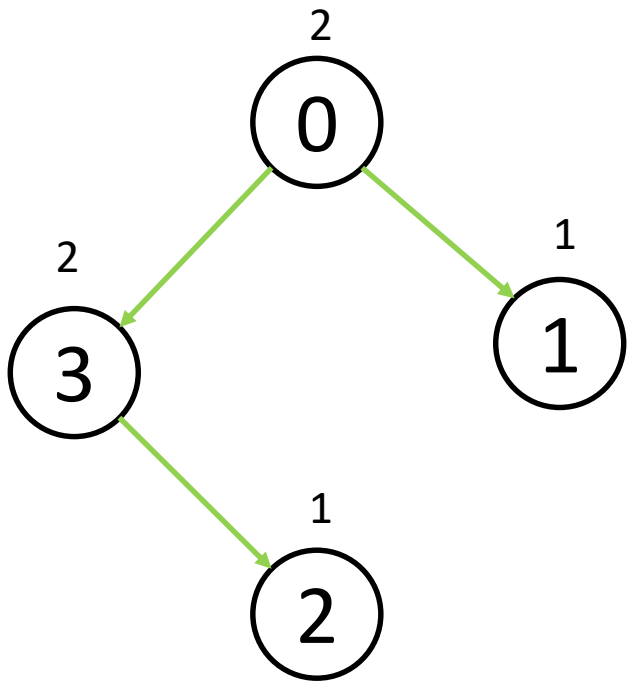
방향 그래프에서 해당 정점으로 들어오는 간선의 수

✓ $Outdegree$

방향 그래프에서 해당 정점에서 나가는 간선의 수

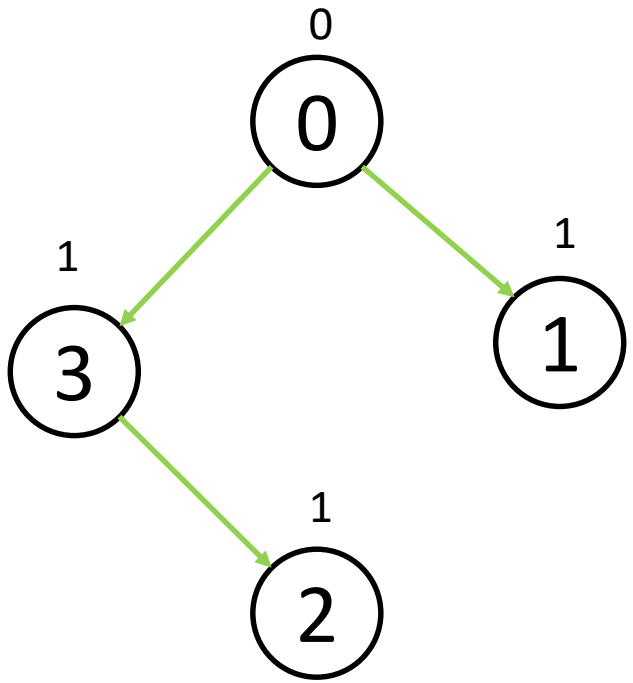


Graph



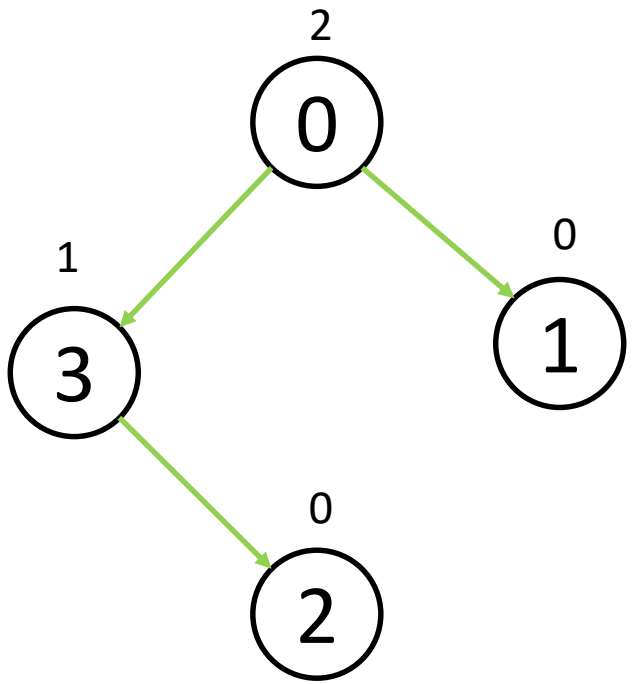
$degrees\ of\ graph$

Graph



$indegrees\ of\ graph$

Graph



$outdegrees\ of\ graph$

Graph

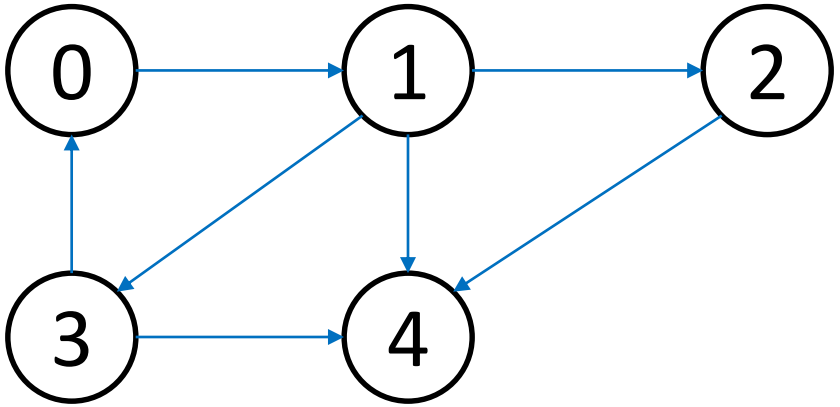
✓ $\sum indegree = \sum outdegree$

✓ $\sum degree = \sum indegree + \sum outdegree$

✓ 그래프의 $degree$를 이용해 $cycle$ 판정과 위상정렬을 할 수 있다.

Graph

✓ $path$(경로)

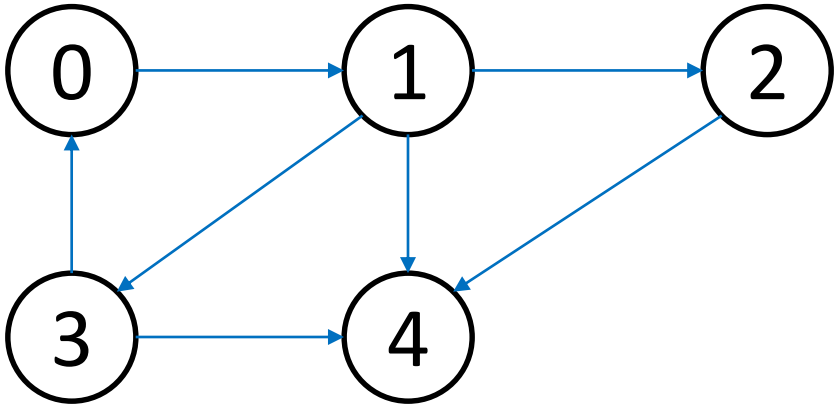


0번 정점에서 4번 정점으로 가는 경로들

- $0 \rightarrow 1 \rightarrow 4$
- $0 \rightarrow 1 \rightarrow 2 \rightarrow 4$
- $0 \rightarrow 1 \rightarrow 3 \rightarrow 4$

Graph

✓ *cycle*



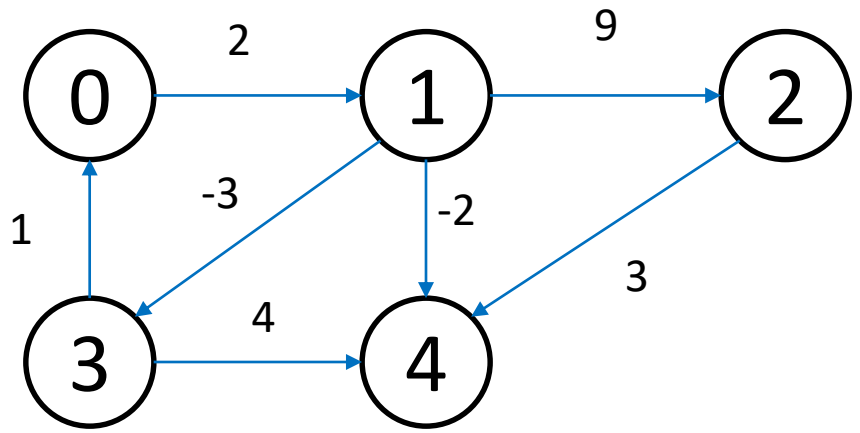
한 정점에서 $path$를 따라 동일한 정점으로 돌아올 수 있는 경우

- $0 \rightarrow 1 \rightarrow 3 \rightarrow 0$

Graph

✓*weight*(가중치)

- 간선에 할당되는 무게.



- 간선의 거리와 비용에 사용한다.

- 가중치가 존재하는 그래프를 *weighted graph*(가중치 그래프)라고 한다.

Graph

✓ 그래프는 알고리즘 & PS분야에서 가장 폭 넓고 깊은 주제 중 하나이다.

✓지금 배운 것들 외에도 정말 다양한 많은 주제와 개념이 있으니 끊임없이 공부해야 한다.

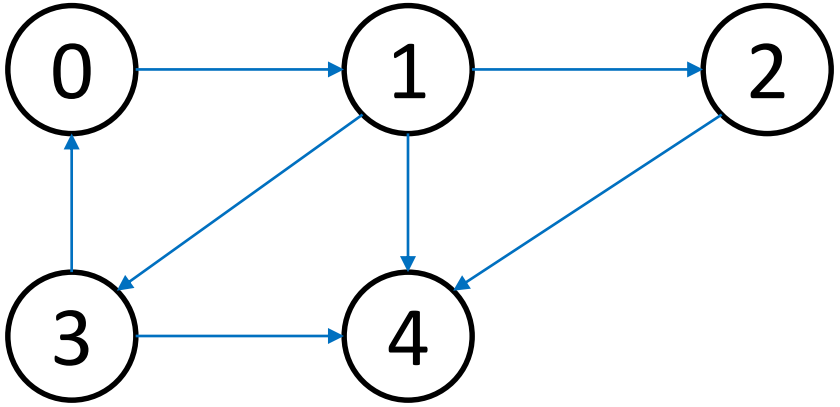
Graph의 표현

✓인접 행렬, Adjacent Matrix

- 두 정점 간의 간선 연결관계를 $|V| * |V|$크기의 행렬(배열) 형태로 표현
- 배열로 행렬을 구현하므로 정점 $i - j$간의 연결관계를 파악하고 싶을 때, i행 j열을 참조하면 $O(1)$에 파악가능
- 구현이 간단한 대신에, 공간복잡도가 $O(V^2)$이라 정점이 많아지면 사용 불가

Graph

✓ *Adjacent Matrix*

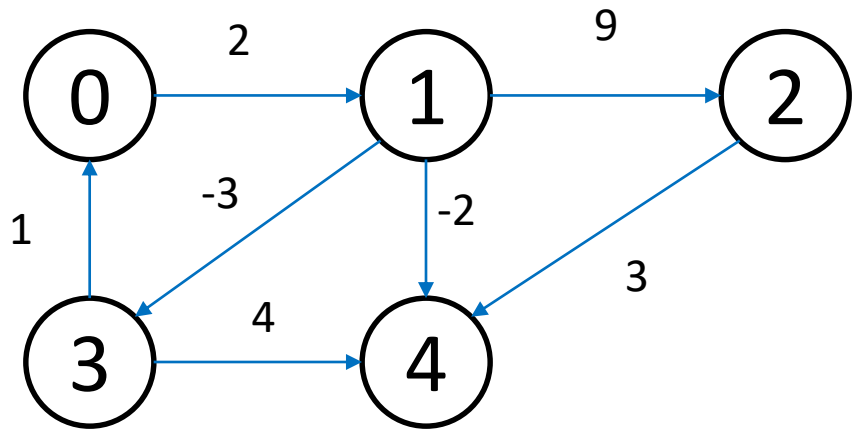


$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Graph

✓ *Adjacent Matrix*

$$\begin{bmatrix} inf & 2 & inf & inf & inf \\ inf & inf & 9 & -3 & -2 \\ inf & inf & inf & inf & 3 \\ 1 & inf & inf & inf & 4 \\ inf & inf & inf & inf & inf \end{bmatrix}$$



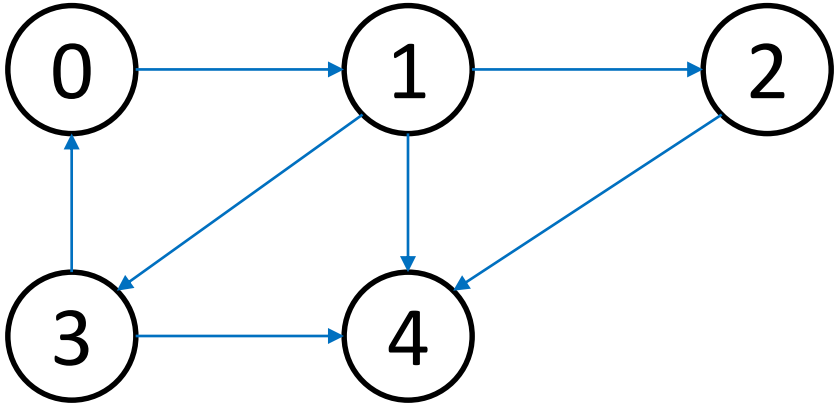
Graph의 표현

✓인접 리스트, Adjacent List

- 각 정점에 연결된 정점들을 리스트에 담아 보관
- vector stl을 사용해 구현하고, 임의의 두 정점의 연결관계를 체크하는데, $O(|V|)$에 파악가능
- 공간복잡도가 $O(E)$.

Graph

✓ *Adjacent List*



$$0 \rightarrow [1]$$
$$1 \rightarrow [2,3,4]$$
$$2 \rightarrow [4]$$
$$3 \rightarrow [4]$$
$$4 \rightarrow [0]$$

Graph

✓그래프의 정의와 표현을 배웠다.

✓그래프를 순회하는 방법을 배워볼 것이다.

✓우리가 구하고자 하는 최적해를 그래프에서 찾아내는 것을 목적으로 한다.

Graph

✓깊이 우선 탐색($depth\ first\ search$)

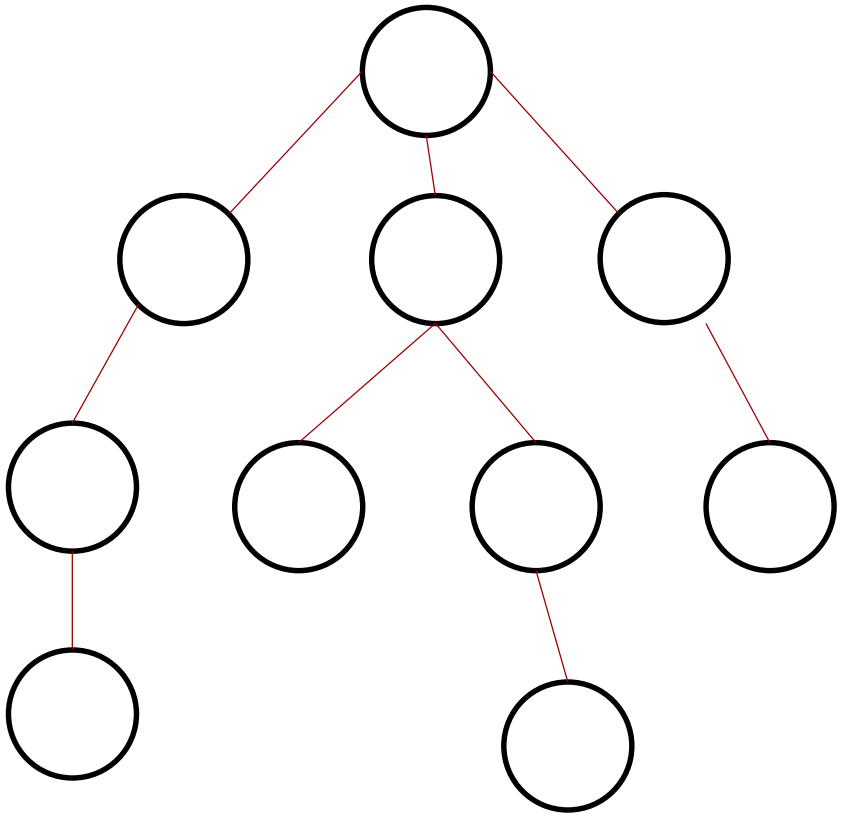
✓한 우물만 파는 방식

✓현재 정점에서 갈 수 있는 정점으로 들어가 계속해서 탐색하고 그 들어갔던 정점에서의 탐색이 종료되면, 나머지 정점을 탐색한다.

✓방문체크를 할 배열이 필요하고 보통 재귀함수로 구현한다.

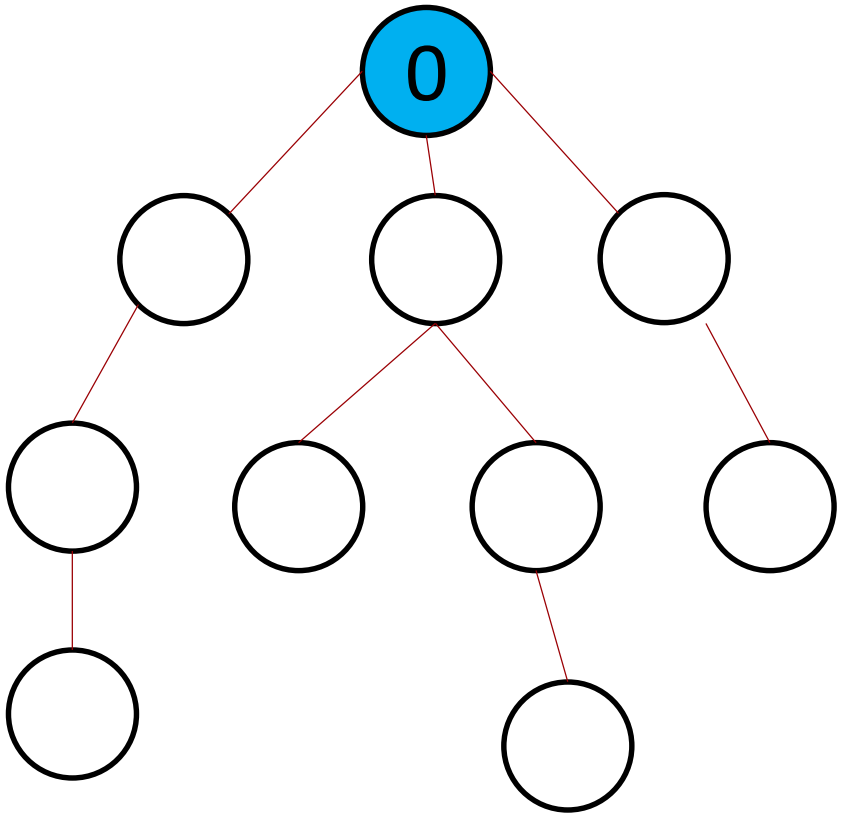
Graph

✓DFS(*depth first search*)



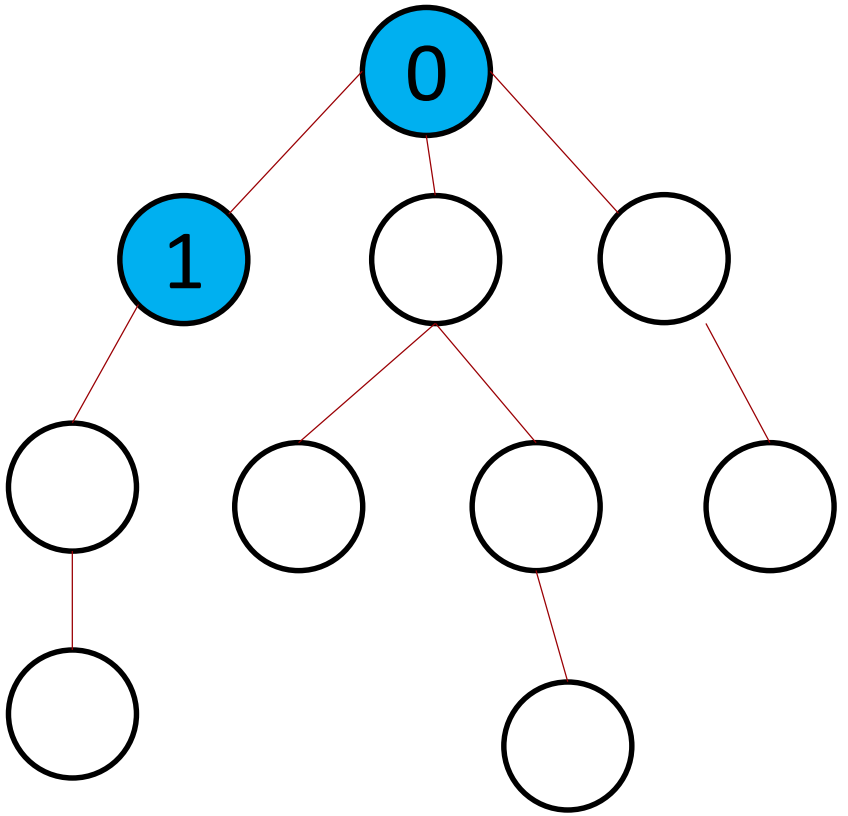
Graph

✓DFS(*depth first search*)



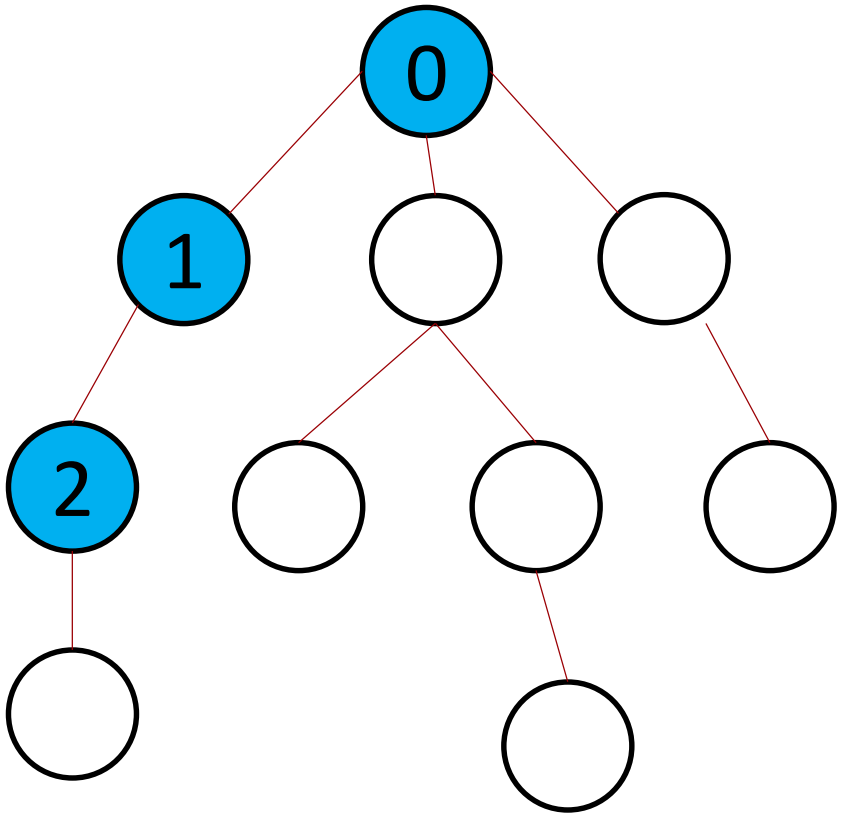
Graph

✓DFS(*depth first search*)



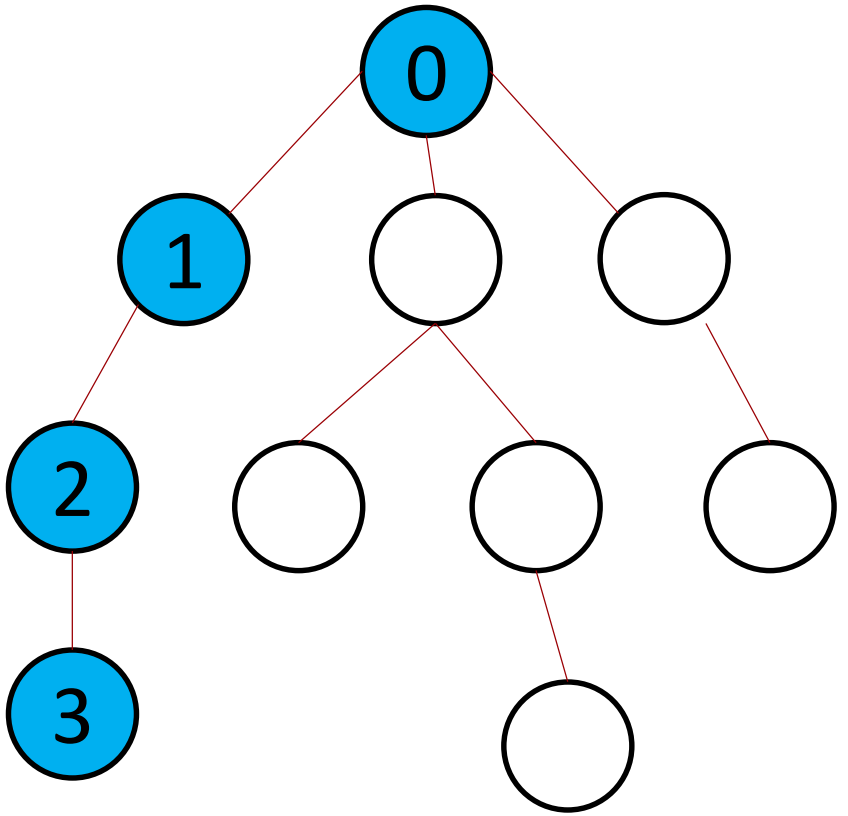
Graph

✓DFS(*depth first search*)



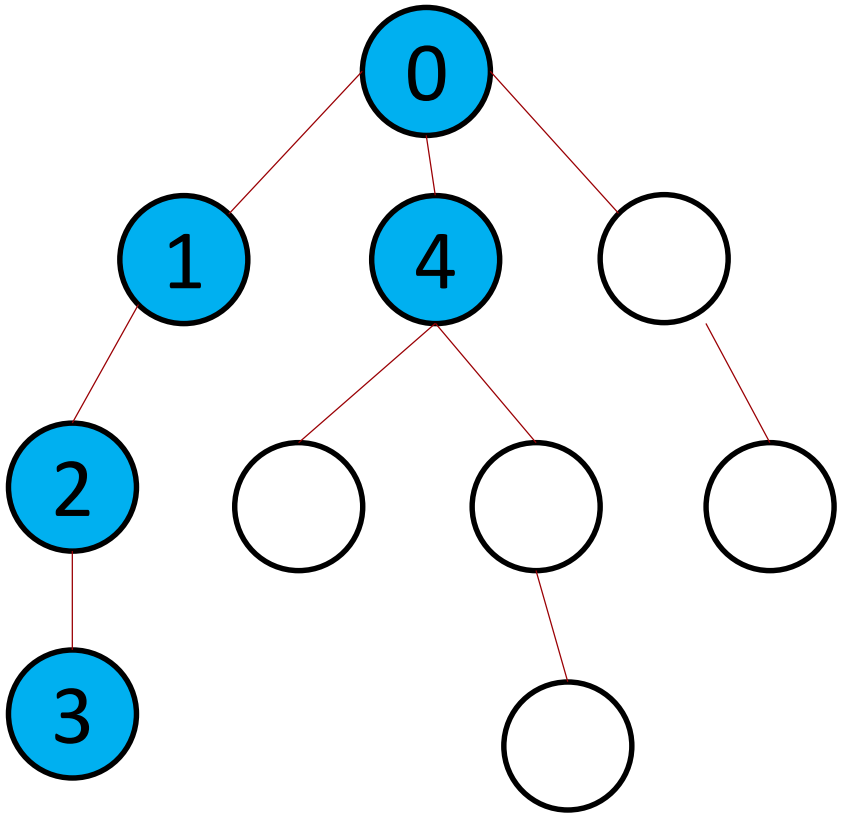
Graph

✓DFS(*depth first search*)



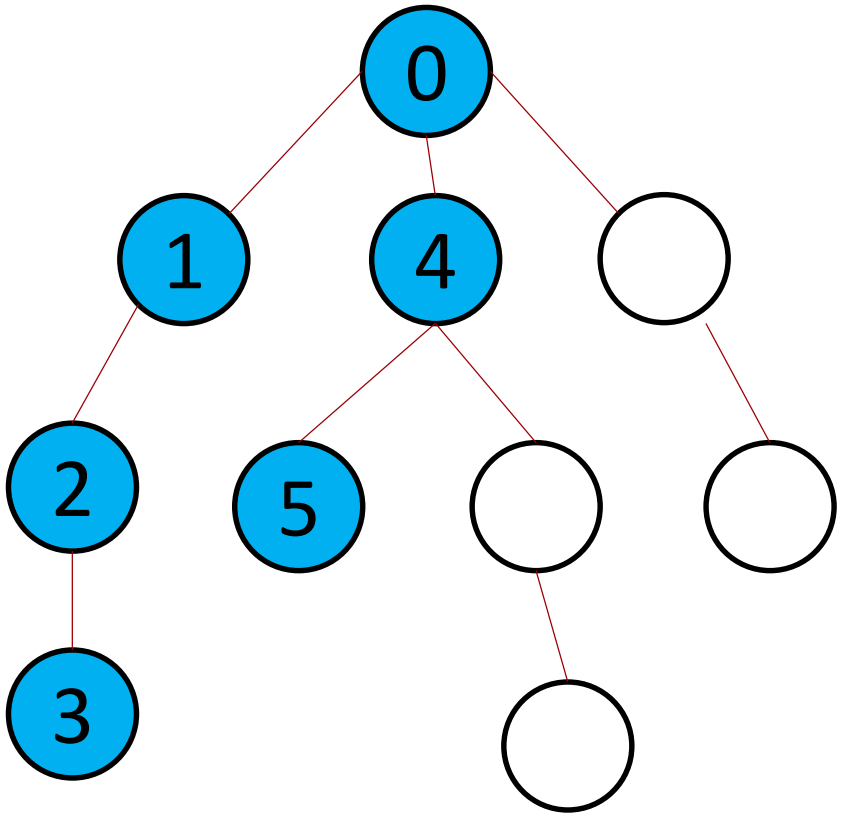
Graph

✓DFS(*depth first search*)



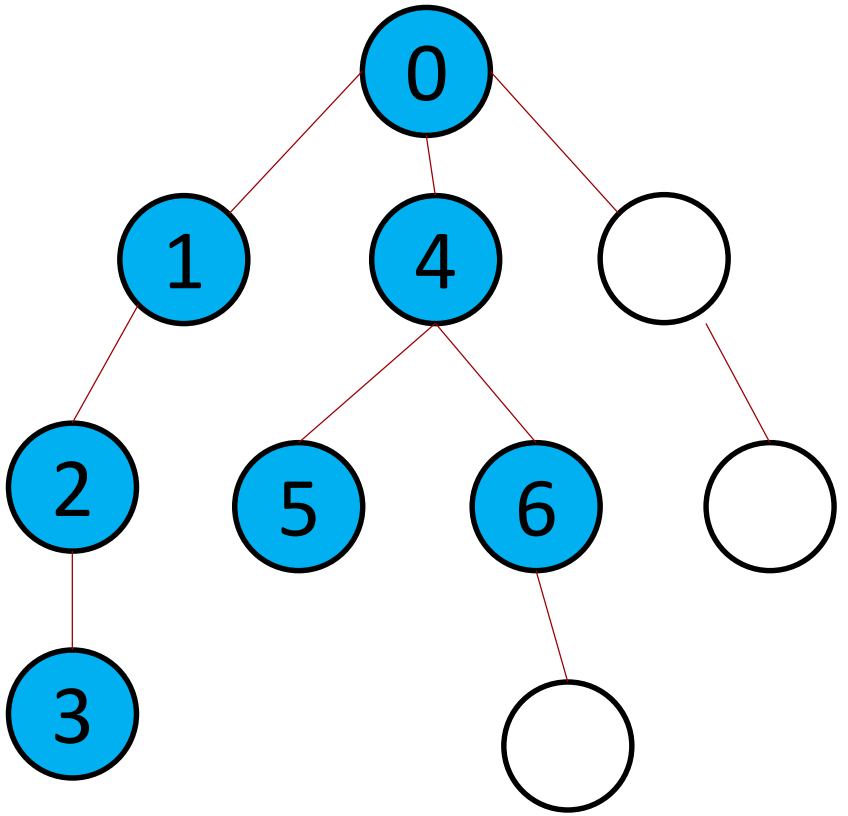
Graph

✓DFS(*depth first search*)



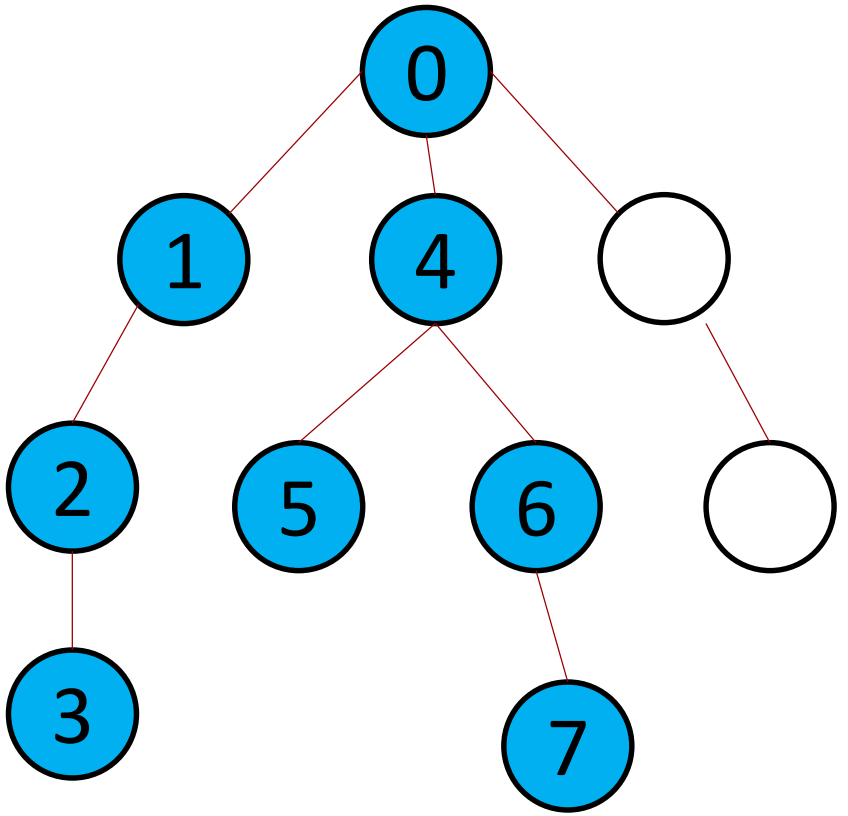
Graph

✓DFS(*depth first search*)



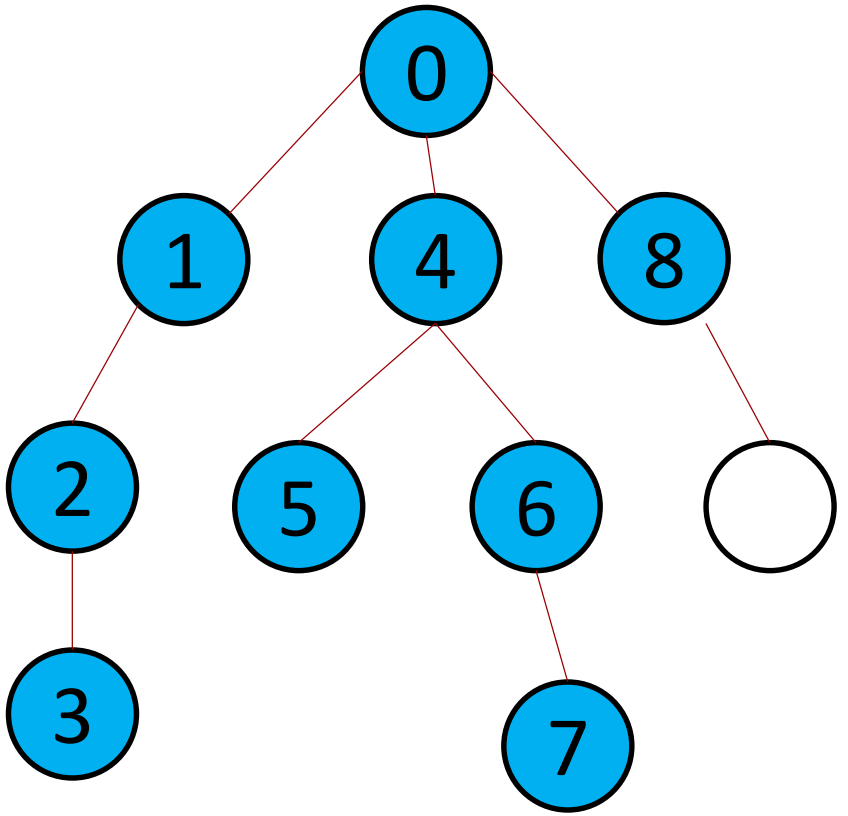
Graph

✓DFS(*depth first search*)



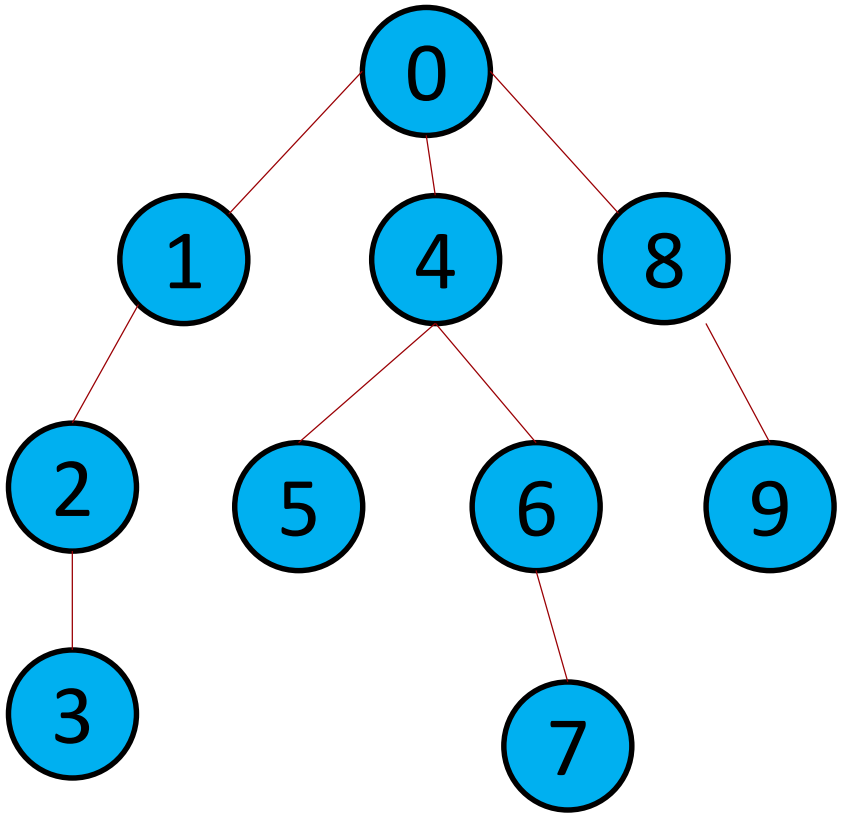
Graph

✓DFS(*depth first search*)



Graph

✓DFS(*depth first search*)



Graph

```
2  void dfs(ll x) {
3      checked[x] = true;
4      cout << x << ' ';
5      for (ll nxt : v[x]) {
6          if (checked[nxt])continue;
7          dfs(nxt);
8      }
9  }
```

인접 리스트로 구현한 코드

```
2  void dfs(ll x) {
3      checked[x] = true;
4      cout << x << ' ';
5      for (int i = 0; i < n_; i++) {
6          if (connected[x][i] && !checked[i])
7          dfs(i);
8      }
9  }
```

인접 행렬로 구현한 코드

✓너비 우선 탐색(*breadth first search*)

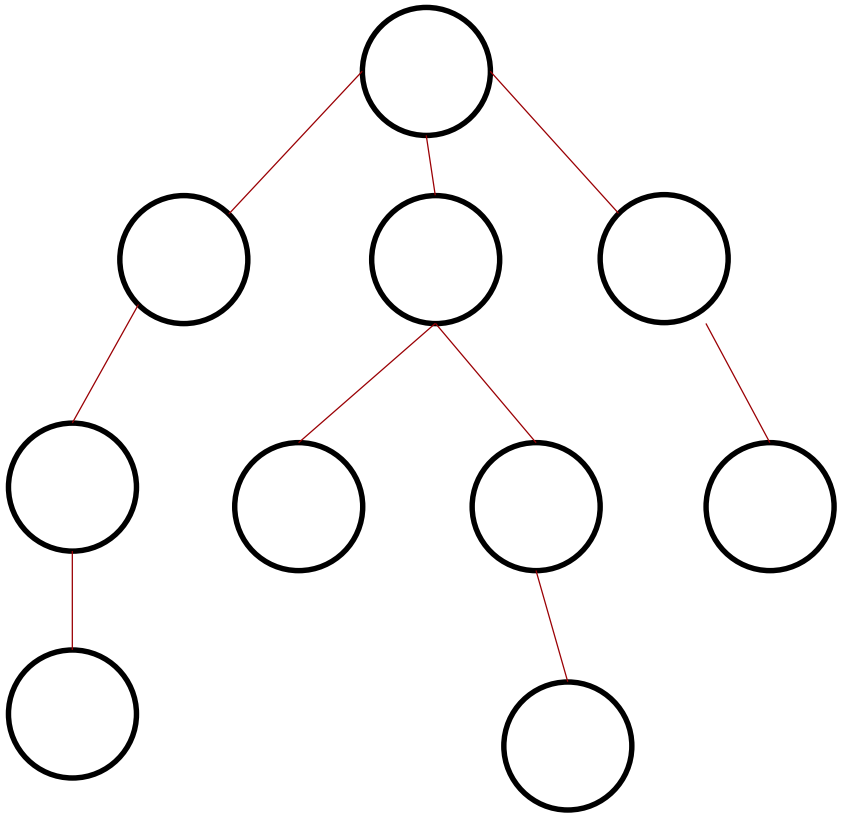
✓여러 우물을 얕게 얕게 탐색하는 방식

✓현재 정점에서 인접한 정점들을 우선적으로 탐색하는 방식

✓방문체크를 할 배열과 방문순서를 관리할 큐가 필요

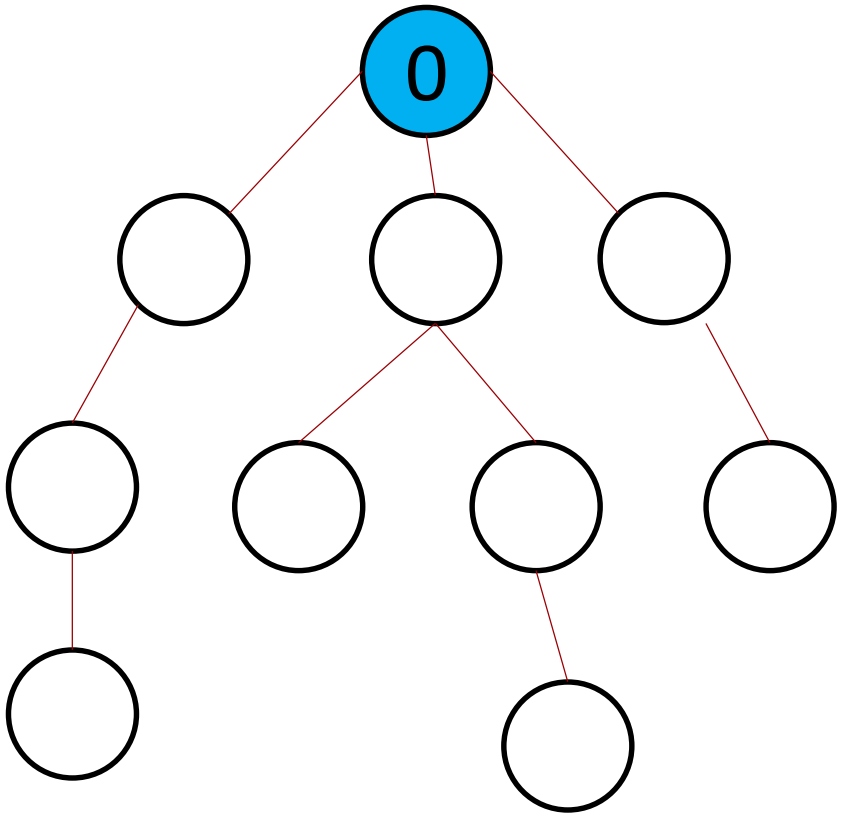
Graph

✓BFS(*breadth first search*)



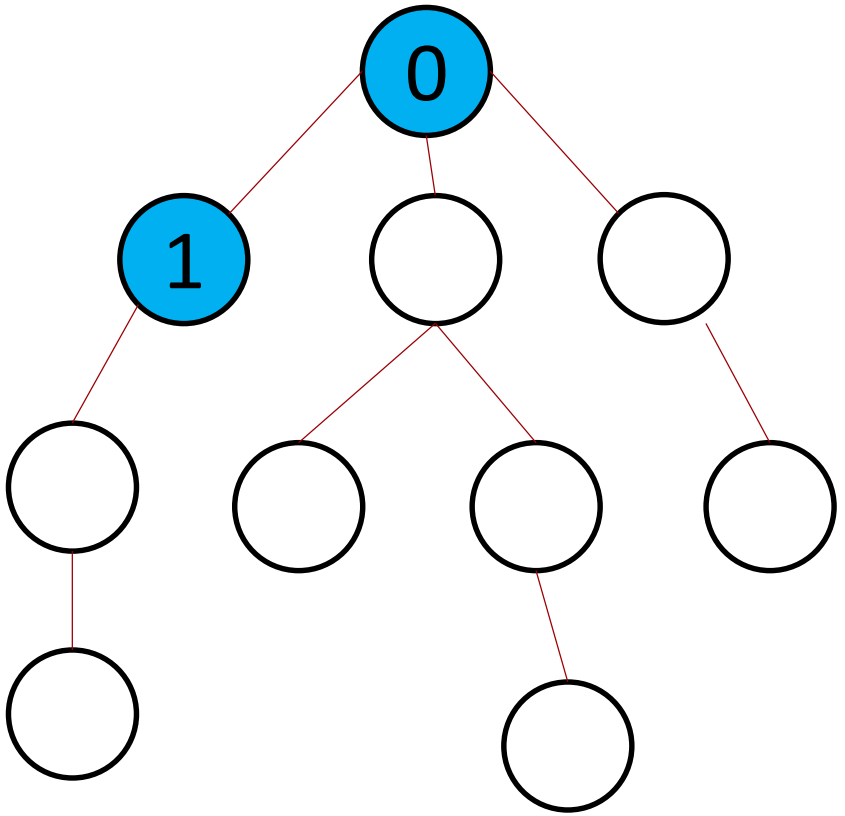
Graph

✓BFS(*breadth first search*)



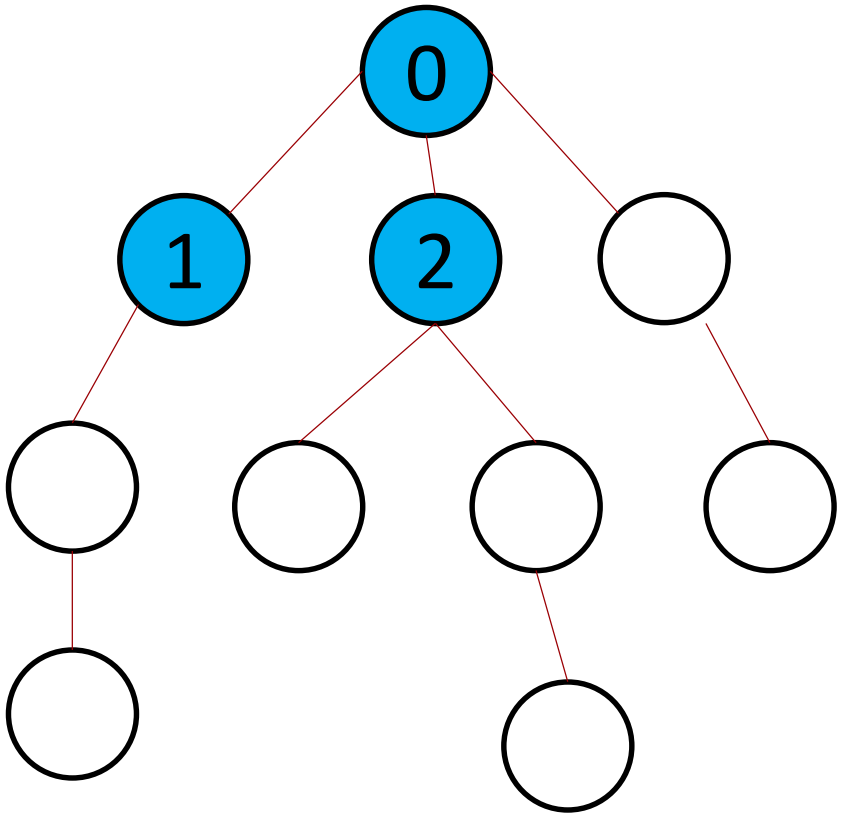
Graph

✓BFS(*breadth first search*)

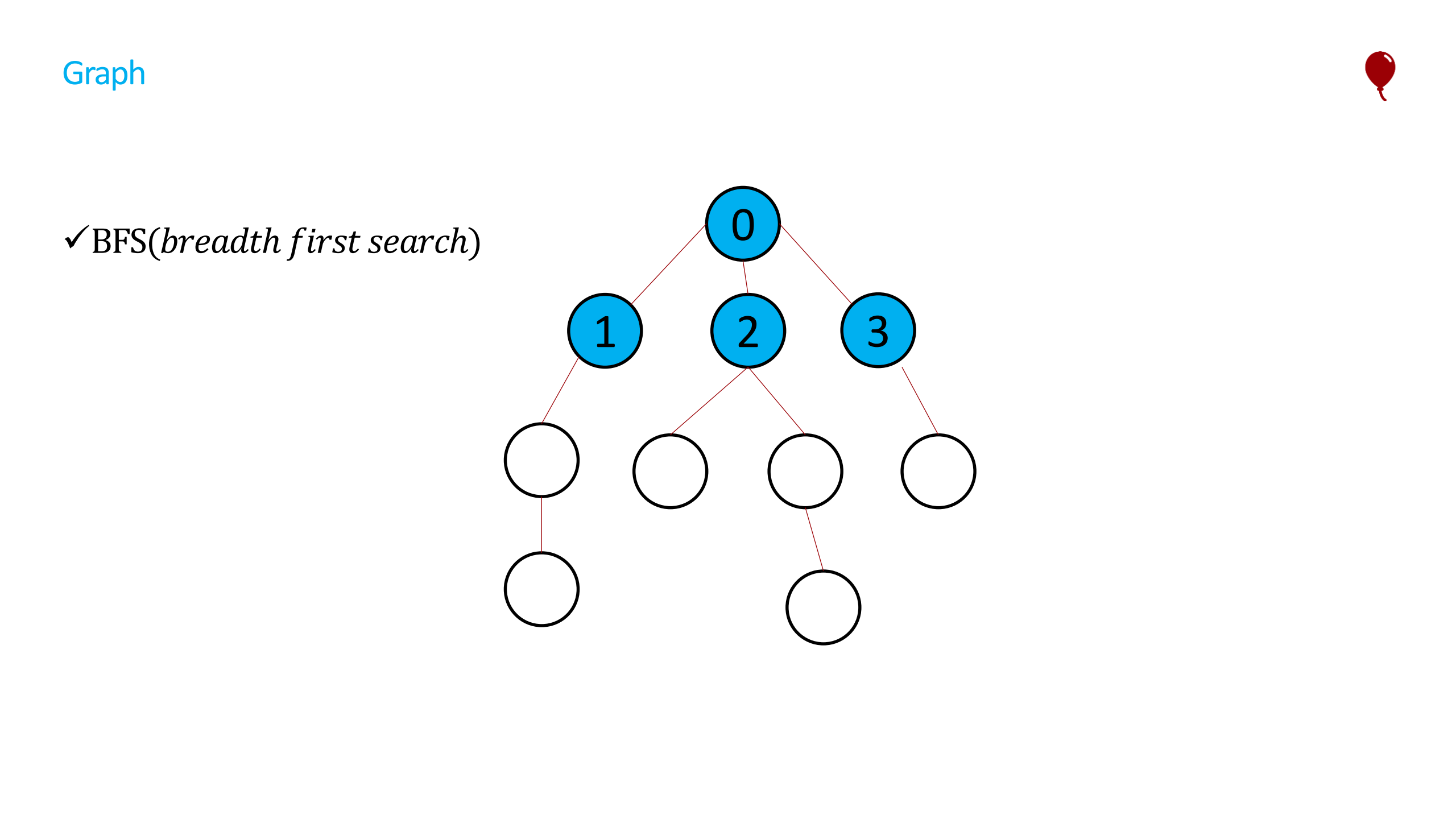


Graph

✓BFS(*breadth first search*)



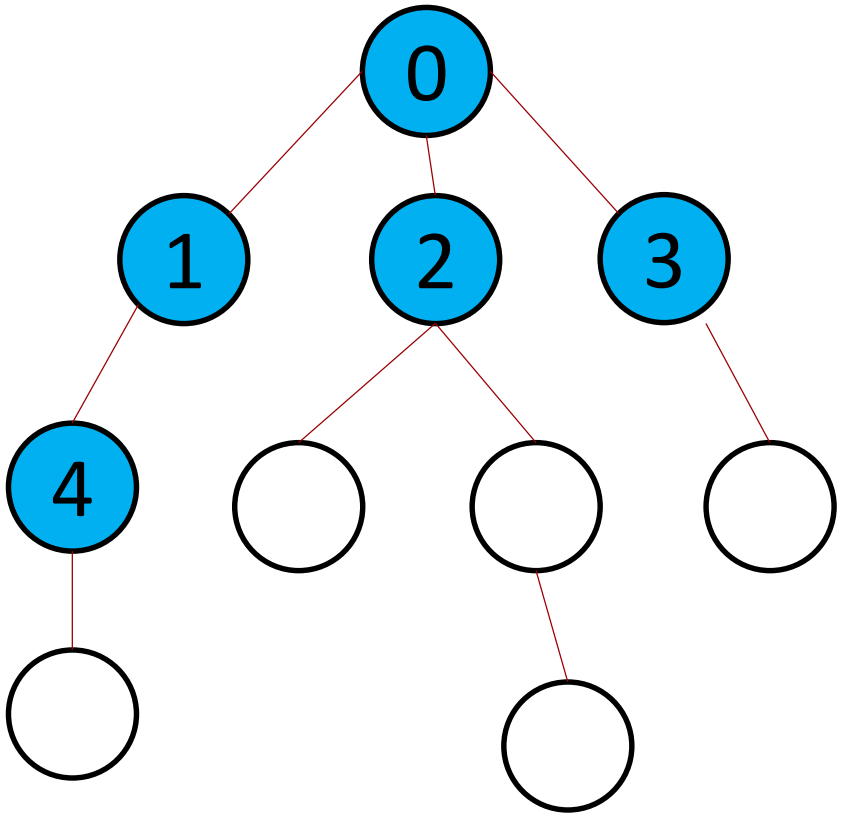
Graph



✓BFS(*breadth first search*)

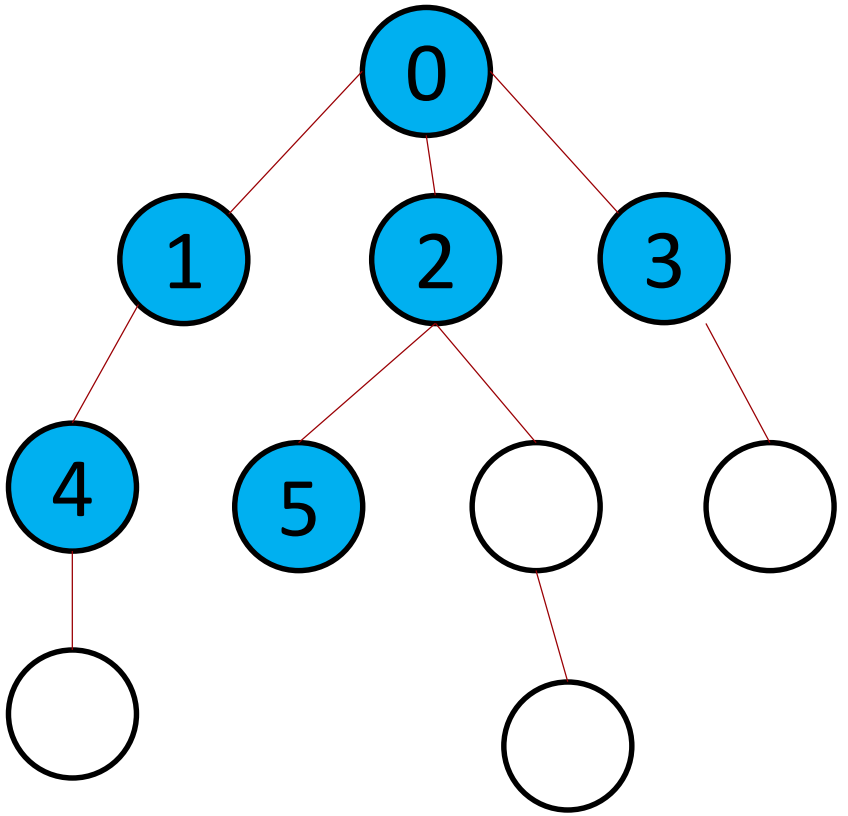
Graph

✓BFS(*breadth first search*)



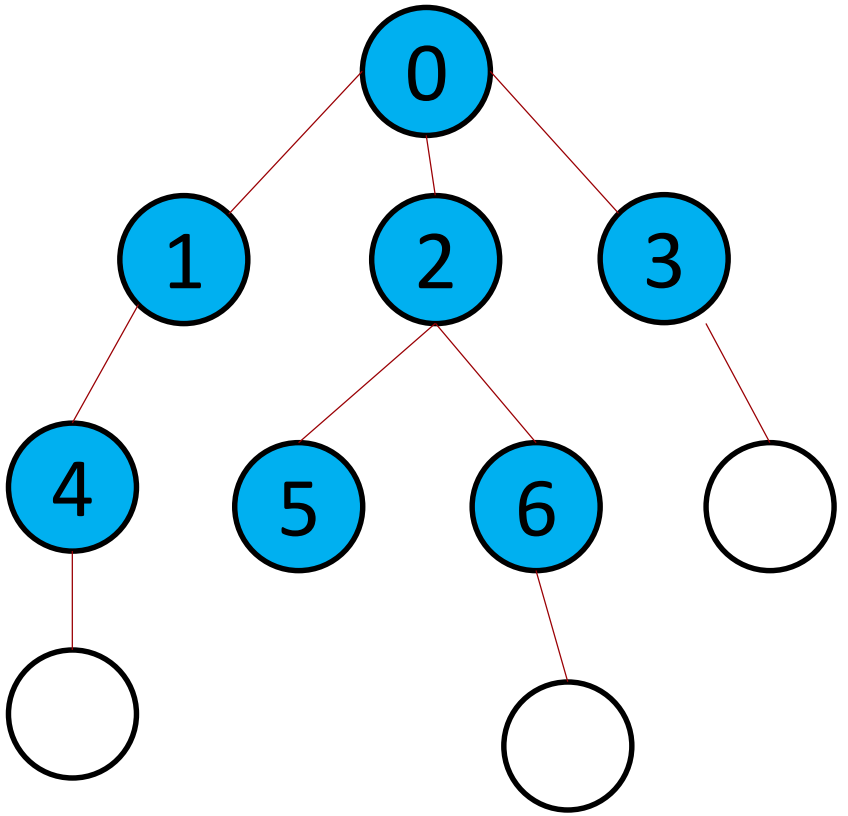
Graph

✓BFS(*breadth first search*)



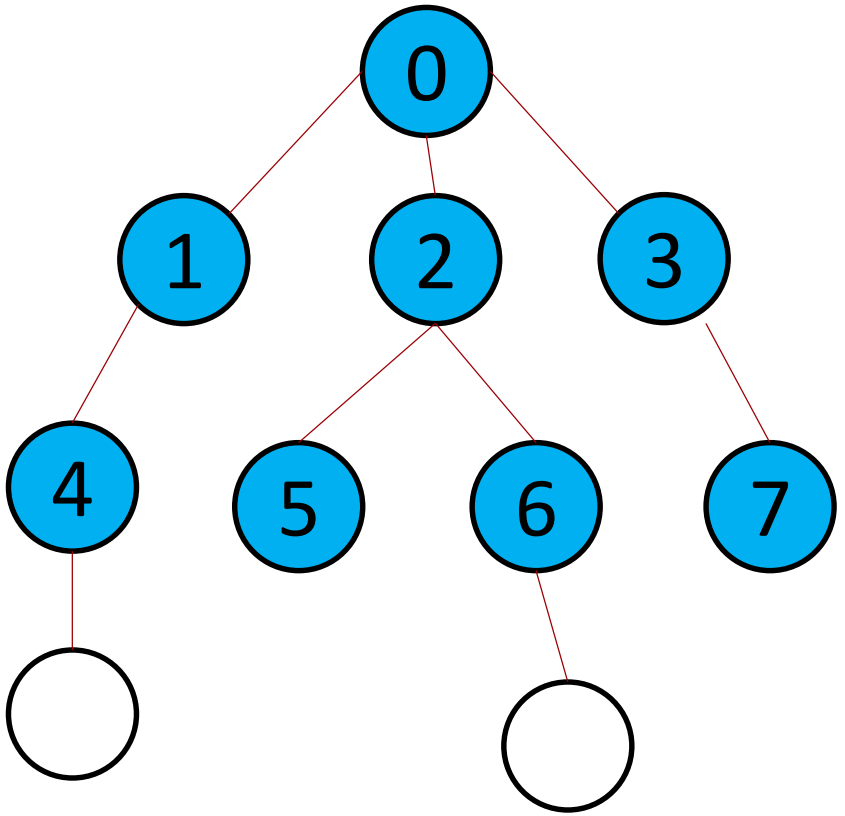
Graph

✓BFS(*breadth first search*)



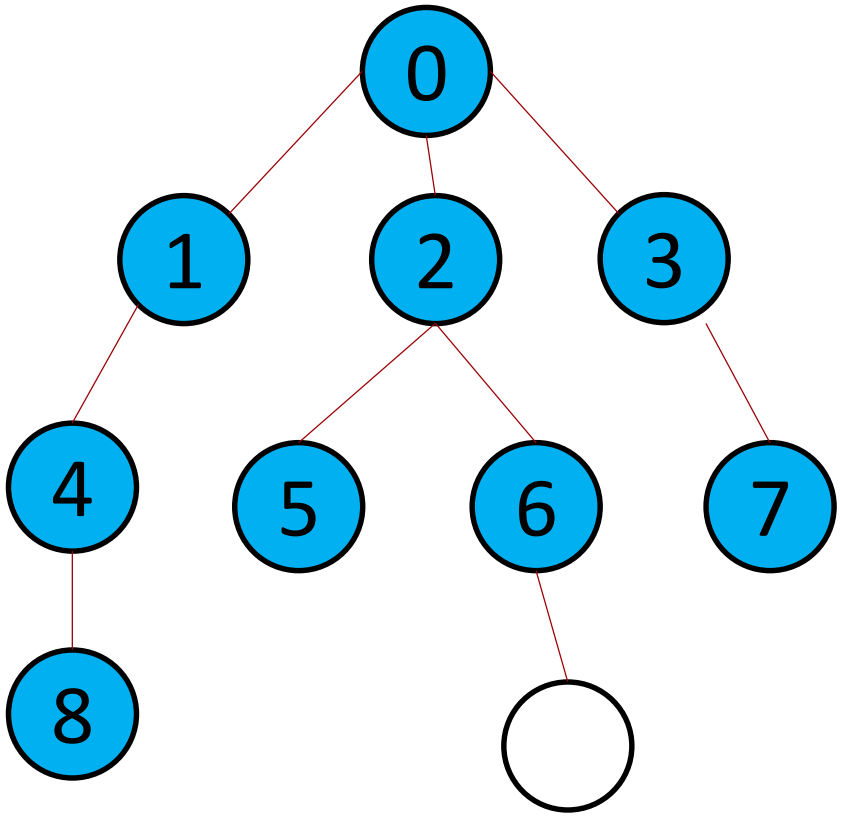
Graph

✓BFS(*breadth first search*)



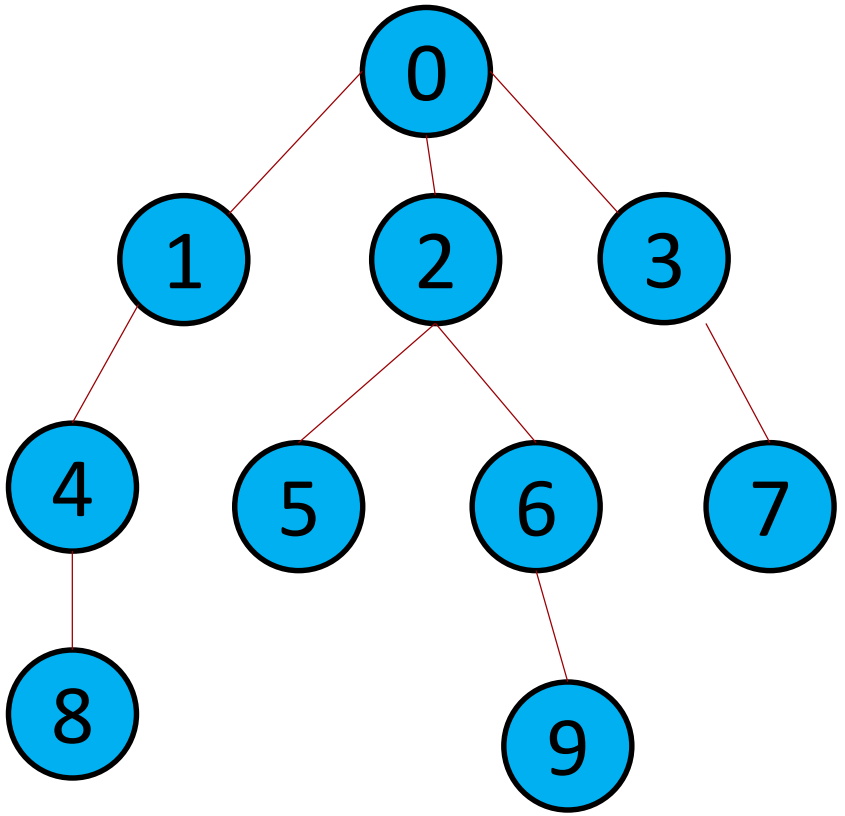
Graph

✓BFS(*breadth first search*)



Graph

✓BFS(*breadth first search*)



Graph

```cpp
void bfs(ll x) {
    queue<ll>q;
    q.push(x);
    checked[x] = true;
    while (q.size()) {
        ll temp = q.front();
        cout << temp << ' ';
        q.pop();
        for (ll nxt : v[temp]) {
            if (checked[nxt])continue;
            q.push(nxt);
            checked[nxt] = true;
        }
    }
}
```

인접 리스트로 구현한 코드

```cpp
void bfs(ll x) {
    queue<ll>q;
    q.push(x);
    checked[x] = true;
    while (q.size()) {
        ll temp = q.front();
        cout << temp << ' ';
        q.pop();
        for (int i = 0; i < n_; i++)
            if (connected[temp][i] && !checked[i])
                q.push(i), checked[i] = true;
    }
}
```

인접 행렬로 구현한 코드

#1260 DFS와 BFS

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.

#16953 A → B

문제

정수 A를 B로 바꾸려고 한다. 가능한 연산은 다음과 같은 두 가지이다.

- 2를 곱한다.
- 1을 수의 가장 오른쪽에 추가한다.

A를 B로 바꾸는데 필요한 연산의 최솟값을 구해보자.

입력

첫째 줄에 A, B ($1 \leq A < B \leq 10^9$)가 주어진다.

출력

A를 B로 바꾸는데 필요한 연산의 최솟값에 1을 더한 값을 출력한다. 만들 수 없는 경우에는 -1을 출력한다.

#16953 A → B

✓우리는 dp를 배웠기 때문에 왠지 dp로 풀고 싶어지는 느낌이 든다!

✓$dp[x]$를 x를 만드는데 필요한 연산의 최솟값으로 정의하면, 짝수일때는 $dp[x/2]$를 참조하고 x를 10으로 나눈 나머지가 1이라면, $dp[(x-1)/10]$을 참조하여 $dp\ table$을 채워 나갈 수 있을 것 같다.

✓근데 문제가 A, B의 범위가 10^9 까지라 배열에 저장을 하게 되면 메모리 초과가 나게 된다.

✓따라서 이 문제는 dp가 아니라 bfs를 이용해서 해결해야 한다.

#16953 A → B

✓문제를 조금만 바꿔서 그래프 처럼 생각해보자.

✓우리는 1번 정점에서 B번 정점까지의 경로의 길이의 최솟값을 찾는 것으로 바꿀 수 있다.

✓우리는 $\times 2$를 하거나 $\times 10$을 하고 $+1$을 하는 두가지 방법을 통해서만 수가 변화할 수 있음을 알고 있다. 즉 인접행렬이나 리스트를 만들지 않고도 그래프에서 어떤 정점끼리 연결되어 있을지를 미리 알고있다는 점을 이용해서, 10^9 개의 배열을 만들지 않고도 그래프 탐색을 통해 문제를 해결 할 수 있다

#16953 A → B

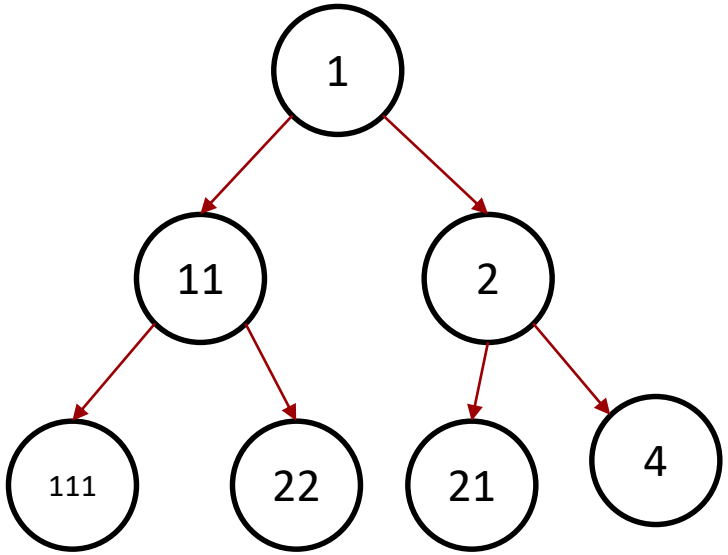
✓dfs로 탐색을 하게 되면, 최악의 경우 모든 경우를 전부 탐색해봐야 한다.

✓하지만 최악의 경우 또한 몇 가지가 안된다.

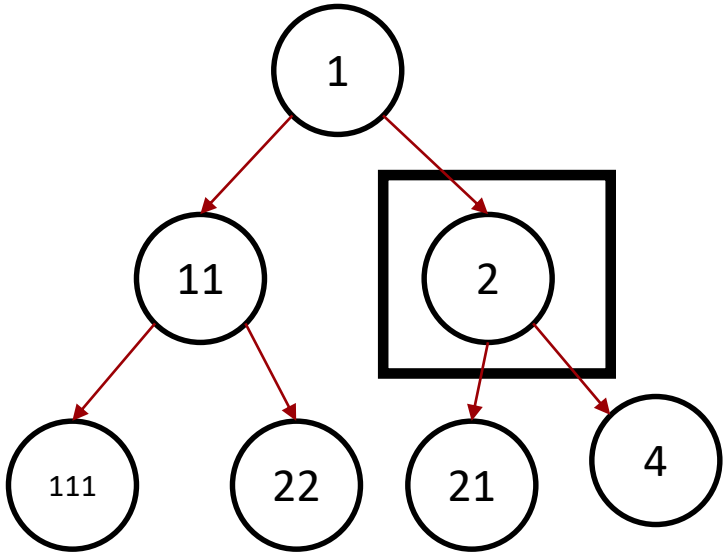
✓따라서 dfs, bfs 둘 다 가능하다.

✓예시를 통해 살펴보자.

#16953 A → B

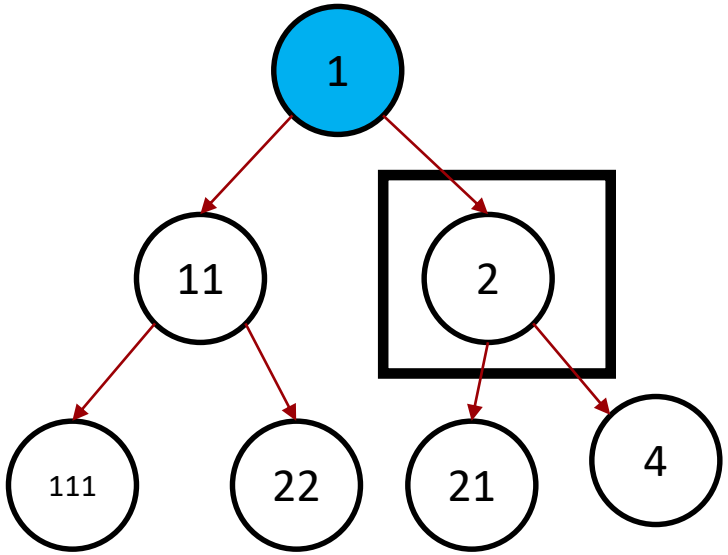


#16953 A → B

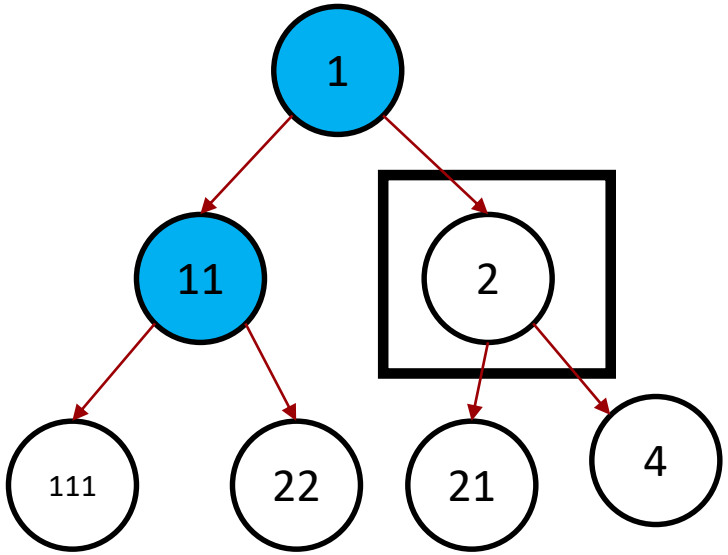


우리가 찾고자 하는 답은 2까지 가는 최솟값

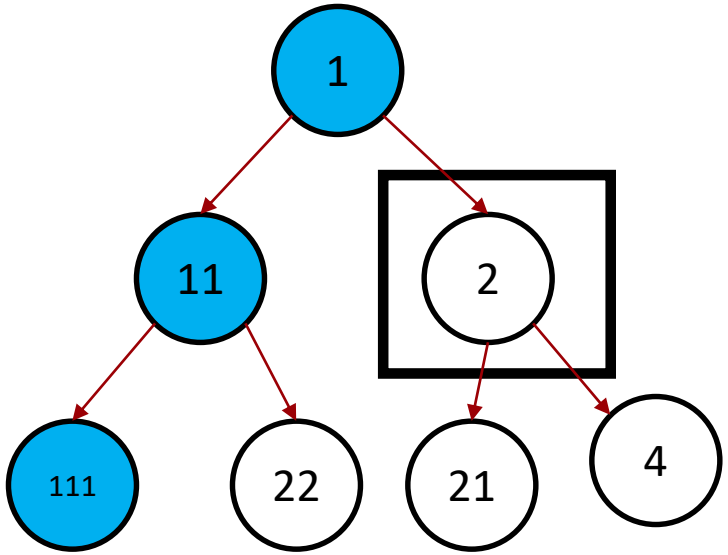
#16953 A → B



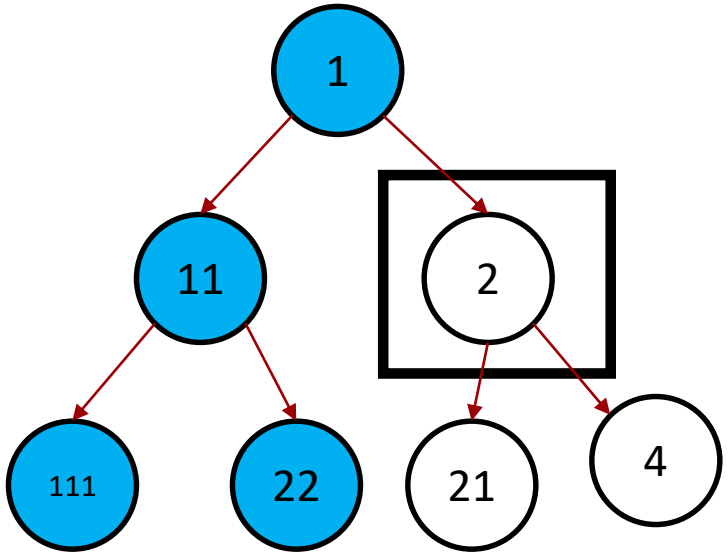
#16953 A → B



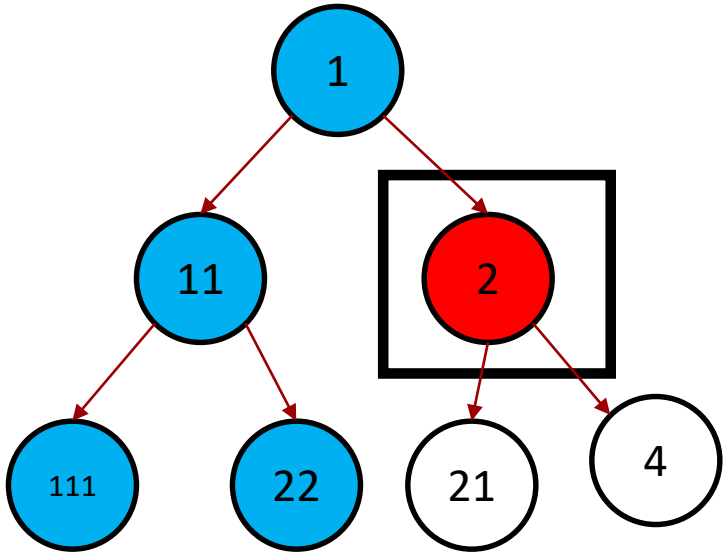
#16953 A → B



#16953 A → B



#16953 A → B



bfs일때는 111, 22를 탐색하지 않아도 발견할 수 있다!

#16953 A → B

```
cin >> a >> b;
    queue<pair<ll,ll>>q;
    q.push({ a,1 });
    while (q.size()) {
        pair<ll,ll> cur = q.front();
        q.pop();
        if (cur.first > b)continue;
        if (cur.first == b) {
            cout << cur.second;
            return 0;
        }
        q.push({ cur.first * 10 + 1,cur.second + 1 }), q.push({ cur.first * 2,cur.second + 1 });
    }
    cout << "-1";
```

#16953 A → B

```
2  void f(ll x, ll cnt) {
3      if (x > b)return;
4      if (x == b) {
5          ans = cnt;
6          return;
7      }
8      f(x * 10 + 1, cnt+1);
9      f(x * 2, cnt+1);
10 }
```

✓dfs는 항상 최적해를 보장해주지 못한다는 단점이 있다.

✓ 예시를 통해 살펴보자.

#1697 숨바꼭질

문제

수빈이는 동생과 숨바꼭질을 하고 있다. 수빈이는 현재 점 N(0 ≤ N ≤ 100,000)에 있고, 동생은 점 K(0 ≤ K ≤ 100,000)에 있다. 수빈이는 걷거나 순간이동을 할 수 있다. 만약, 수빈이의 위치가 X일 때 걷는다면 1초 후에 X-1 또는 X+1로 이동하게 된다. 순간이동을 하는 경우에는 1초 후에 2*X의 위치로 이동하게 된다.

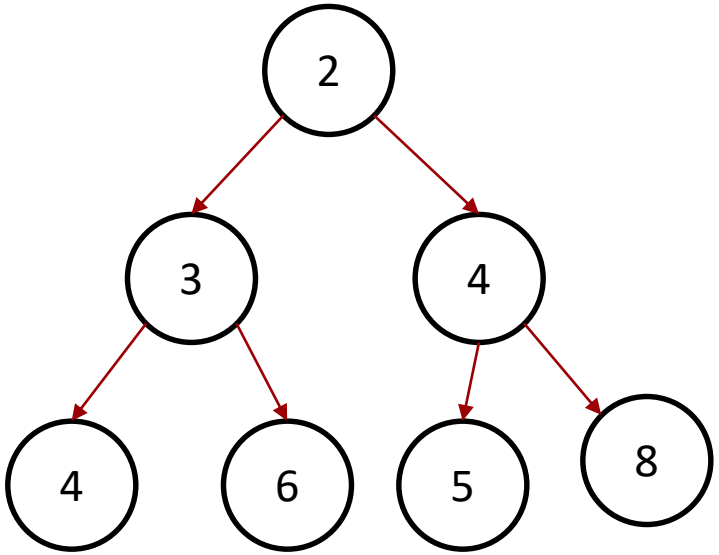
수빈이와 동생의 위치가 주어졌을 때, 수빈이가 동생을 찾을 수 있는 가장 빠른 시간이 몇 초 후인지 구하는 프로그램을 작성하시오.

#1697 숨바꼭질

✓문제가 아까와 굉장히 비슷하다.

✓이번엔 그리디한 성질이 만족하지 않는다. 따라서 서로 다른 경로로 같은 한 정점에 도달하는 것이 가능하다.

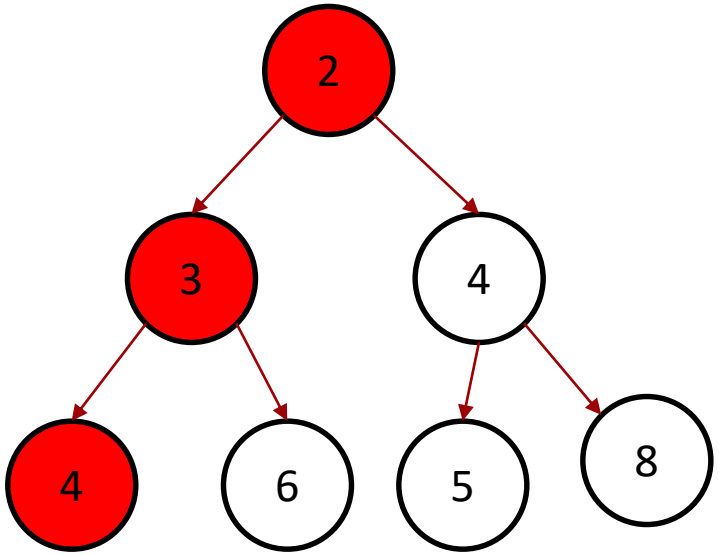
#1697 숨바꼭질



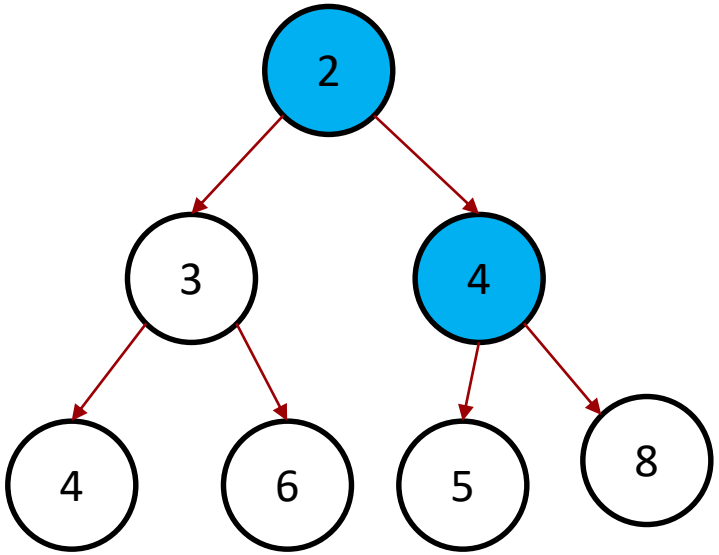
편의상 -1 연산은 잠시 삭제했다.

만약 4까지 최소 경로를 구해야 한다면?

#1697 숨바꼭질



#1697 숨바꼭질



#추천문제

필수문제

1260	DFS와 BFS
16953	A → B
1697	숨바꼭질
19538	루머
5829	Luxury River Cruise

연습문제

2178	미로 탐색
2606	바이러스
7576	토마토
19641	중첩 집합 모델
10026	적록색약
2023	신기한 소수
1987	알파벳
1240	노드사이의 거리
16940	BFS 스페셜 저지
18170	두 동전 언리미티드
17942	알고리즘 공부
20119	클레어와 물약
8111	0과 1