2021 신촌 연합 여름캠프 초급반 9회차

DFS & BFS

서강대학교 박재형

그래프 탐색

그래프 탐색이란?

- 그래프의 한 정점으로부터 출발해서 모든 정점을 1번씩 방문하여 그래프에 대한 정보를 얻는 것

- 정점 방문 순서에 따라 크게 두 방식으로 나뉨

- 깊이 우선 탐색 / 너비 우선 탐색

깊이 우선 탐색 (DFS)

- Depth First Search (DFS)

- 다음 분기(brunch)로 넘어가기 전에 현재 분기를 완전히 탐색하는 방법

- 트리의 전위(preorder)탐색이 그래프로 확장된 개념

- 재귀 함수 or 스택으로 구현

깊이 우선 탐색 (DFS)

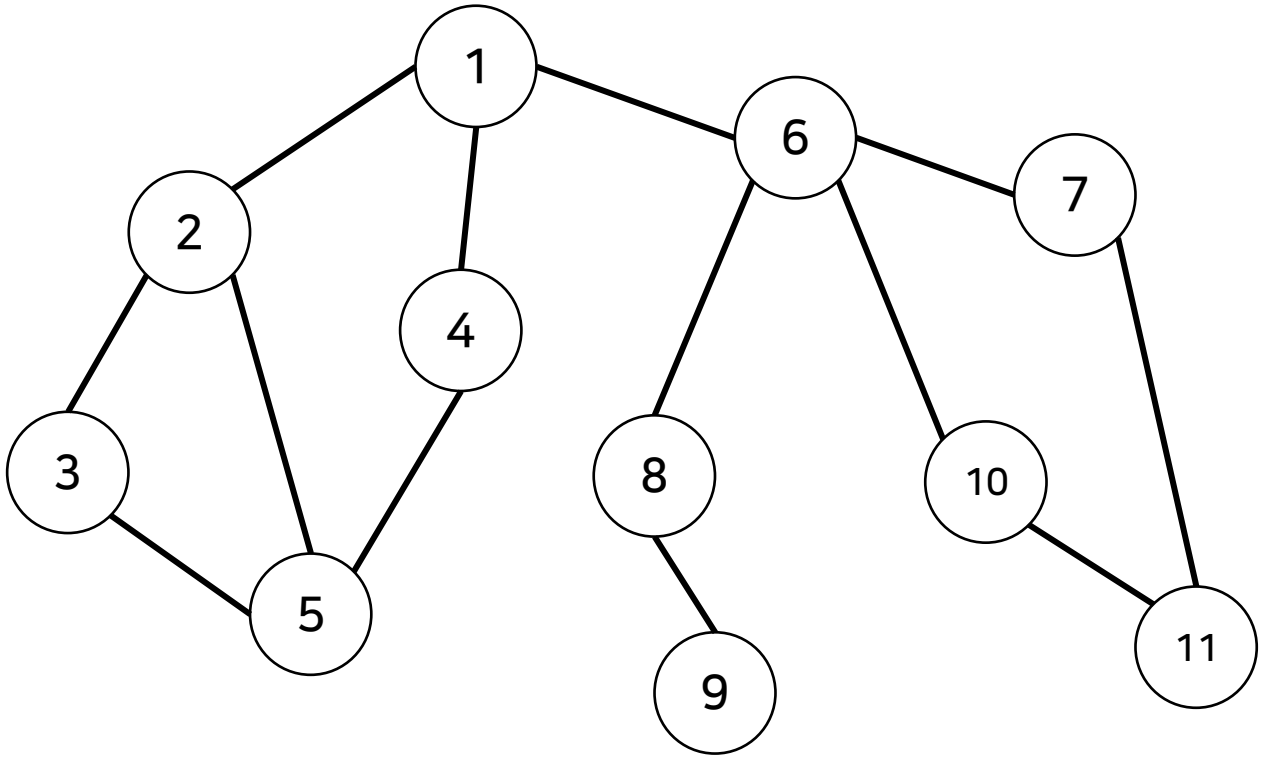
1. 정점 1개를 선택한다.

2. 현재 정점과 이웃한 정점 중 아직 방문하지 않은 정점 중 하나를 방문한다.

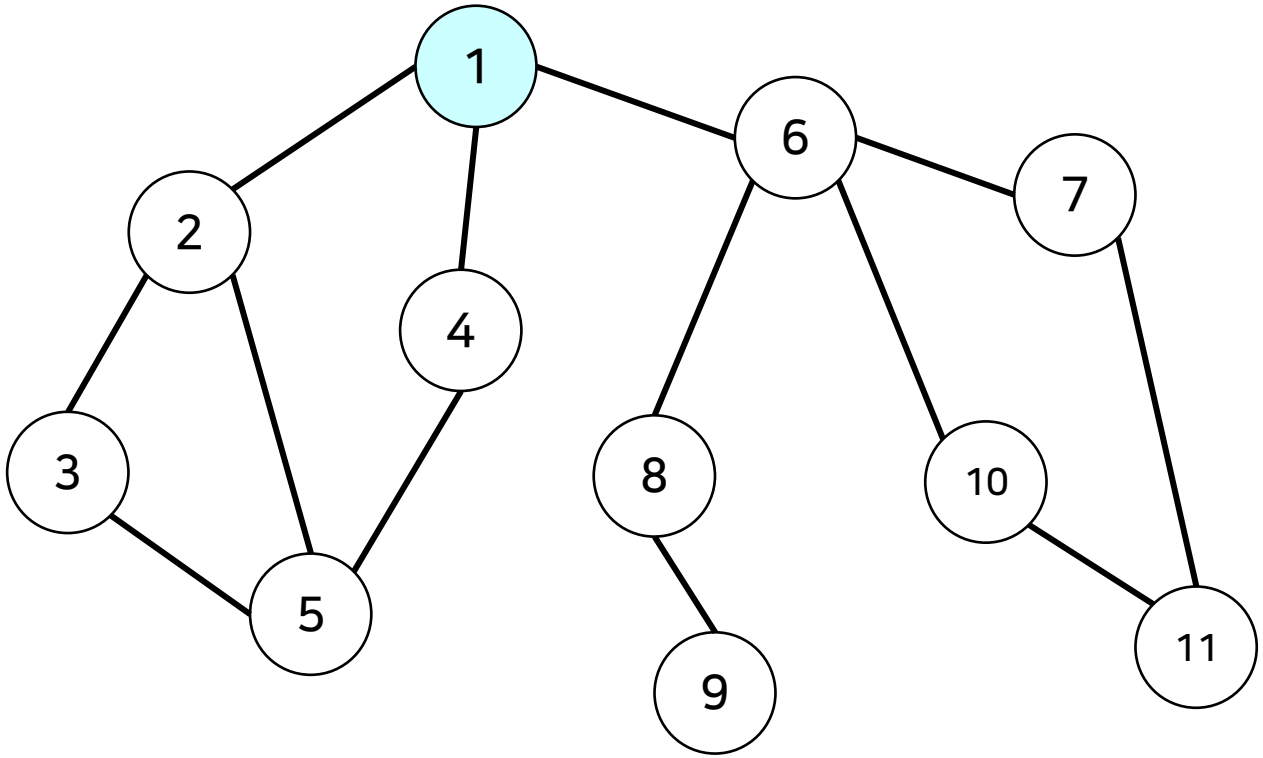
3. 더 이상 방문할 수 있는 정점이 없는 경우, 현재 정점의 이전 정점으로 돌아간다.

4. 모든 정점을 방문할 때까지 2~3을 반복한다.

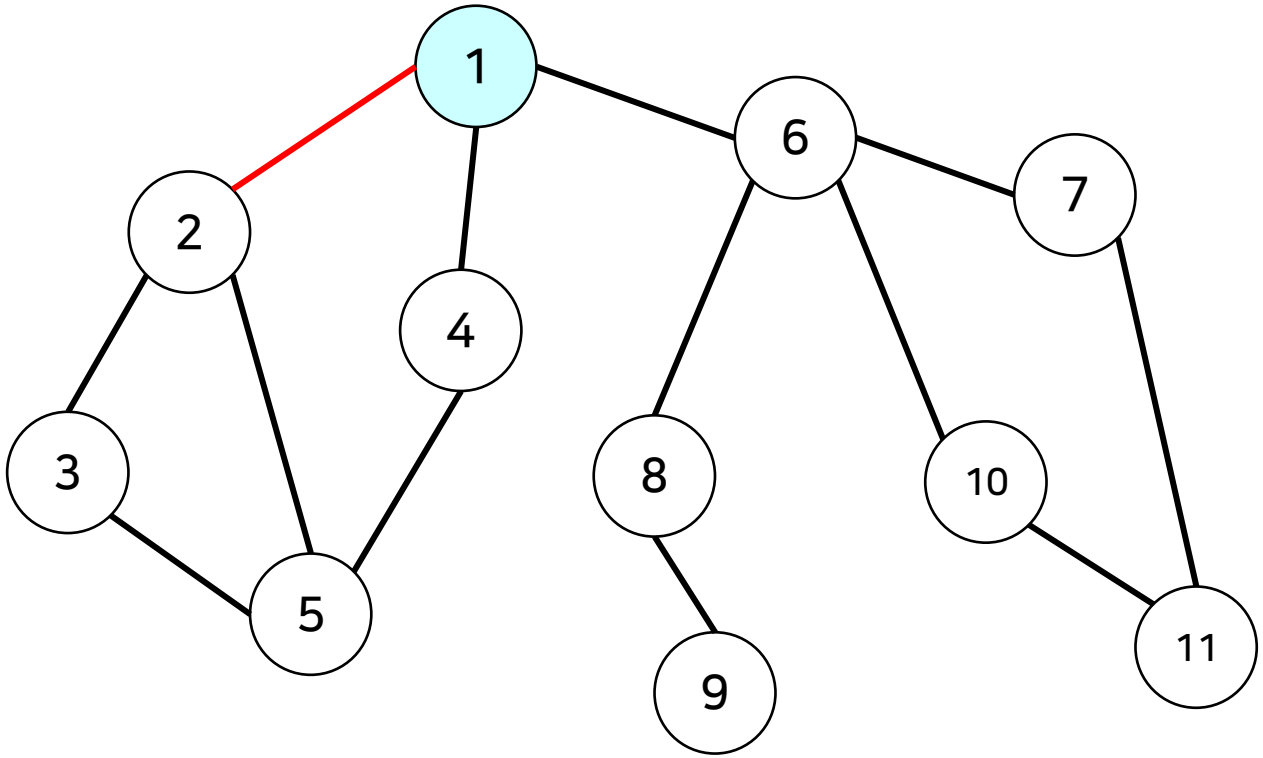
깊이 우선 탐색 (DFS)



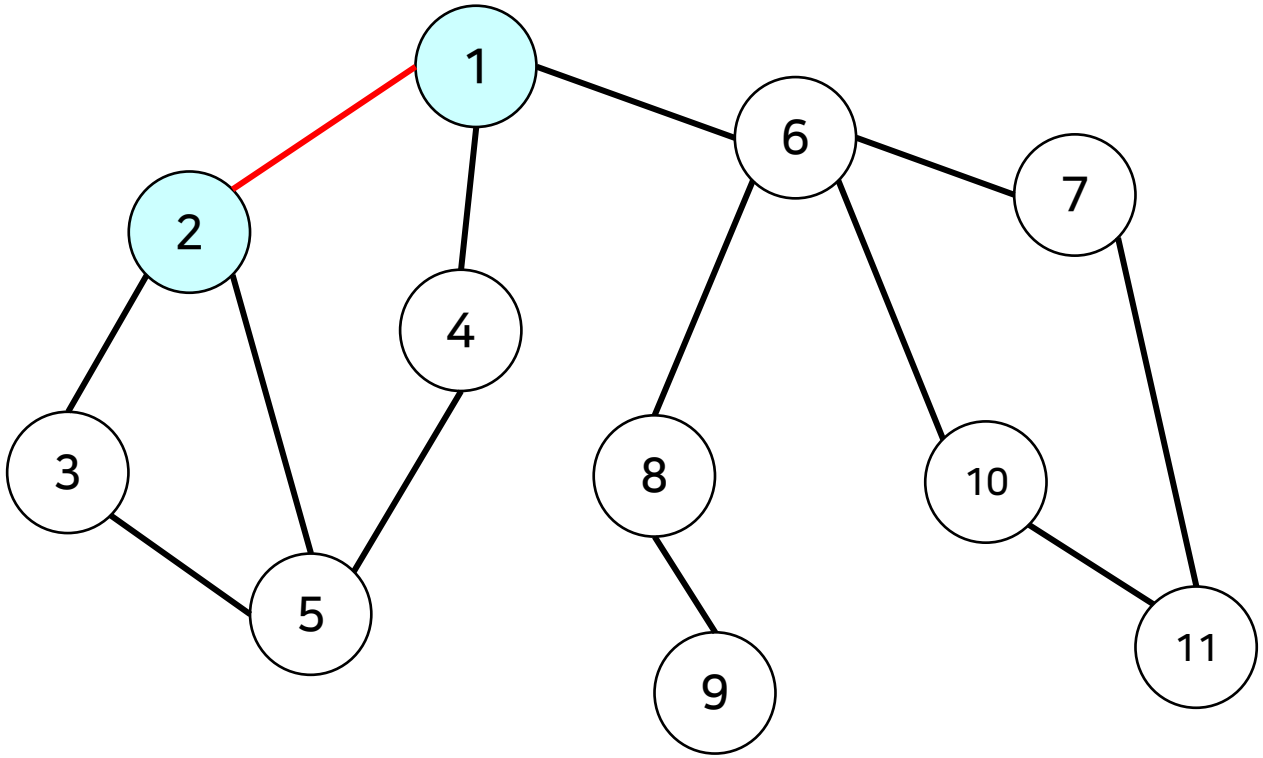
깊이 우선 탐색 (DFS)



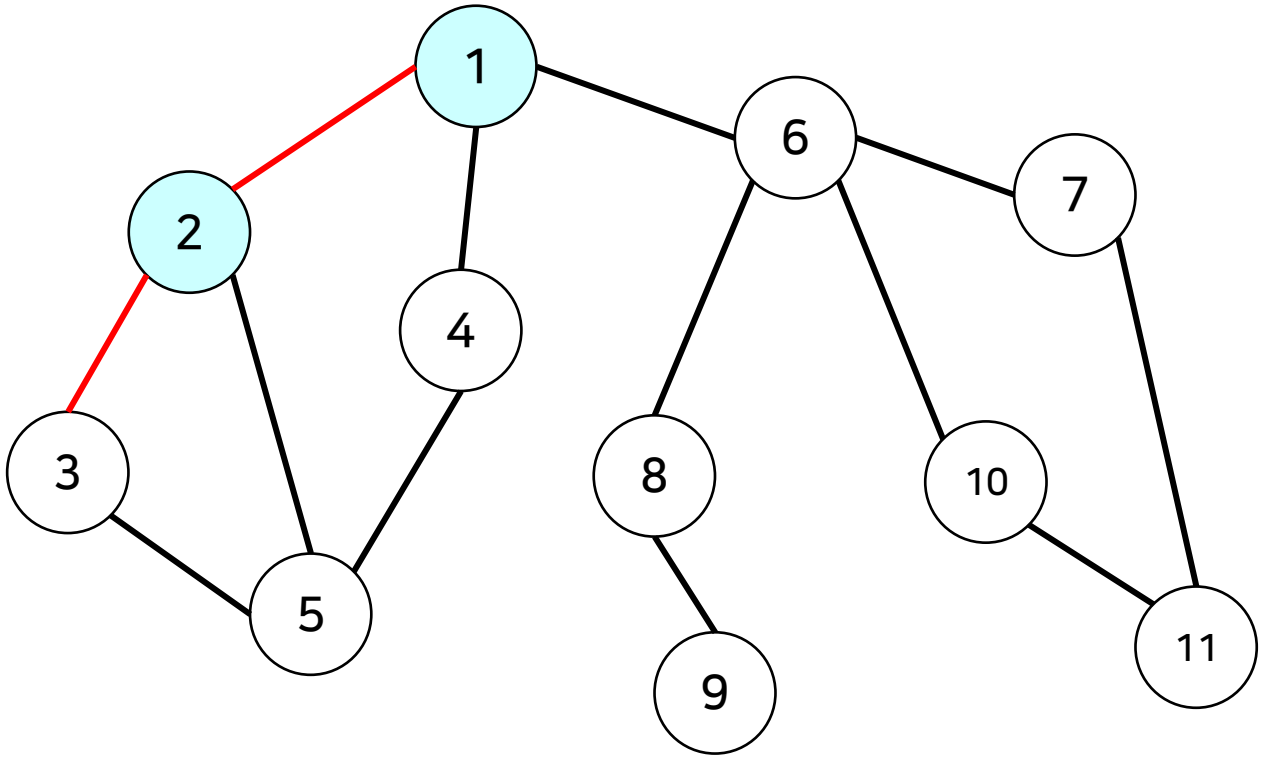
깊이 우선 탐색 (DFS)



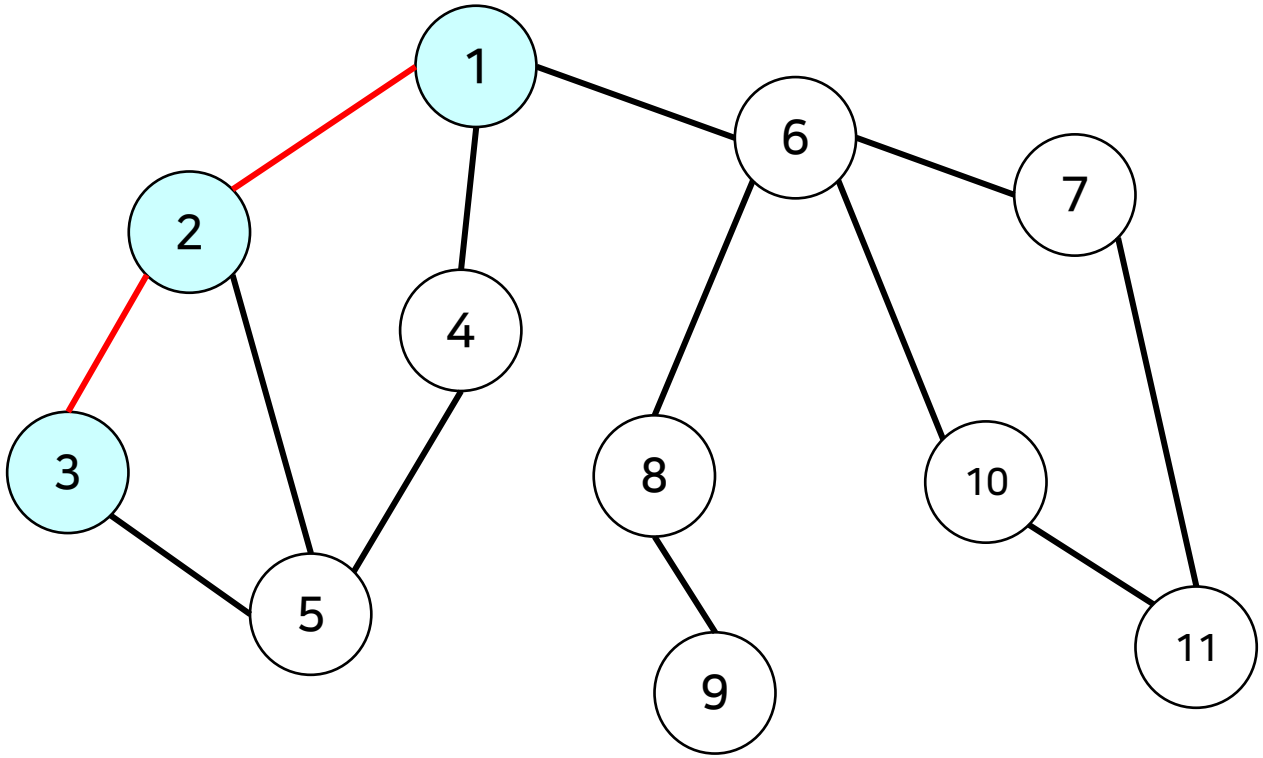
깊이 우선 탐색 (DFS)



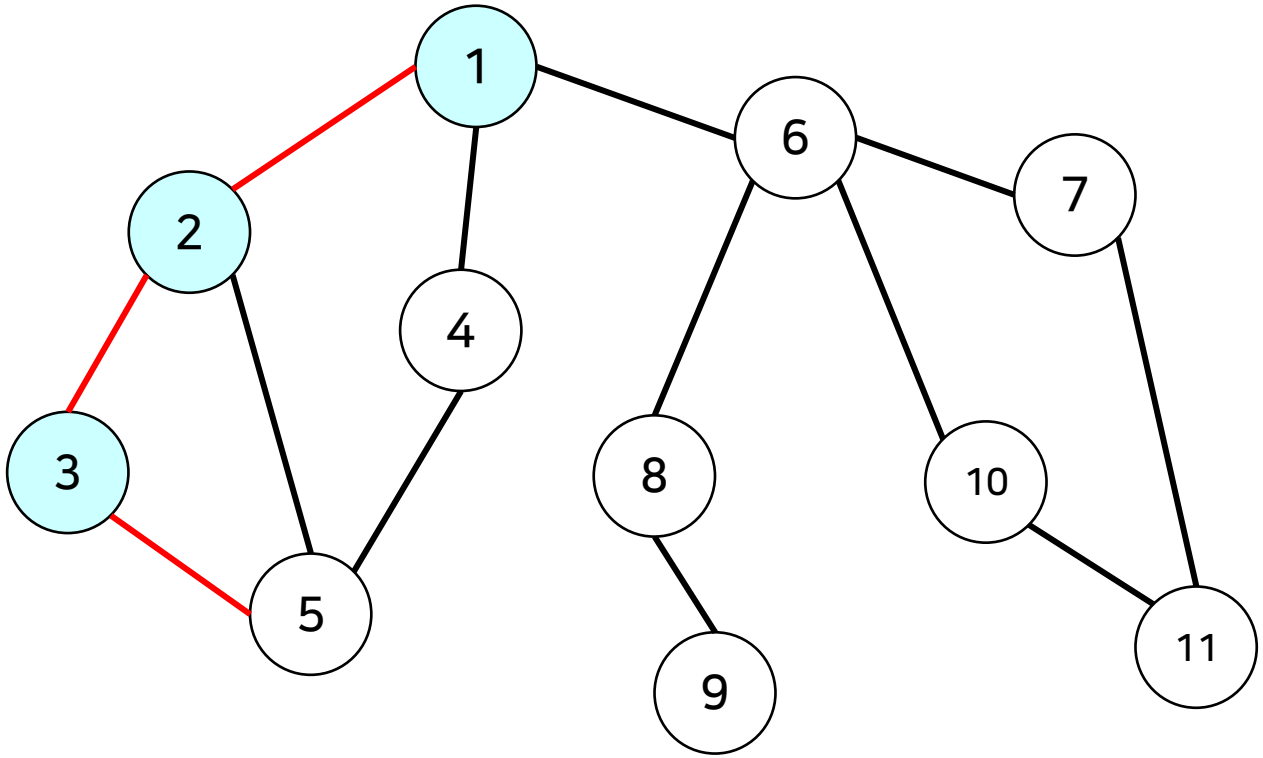
깊이 우선 탐색 (DFS)



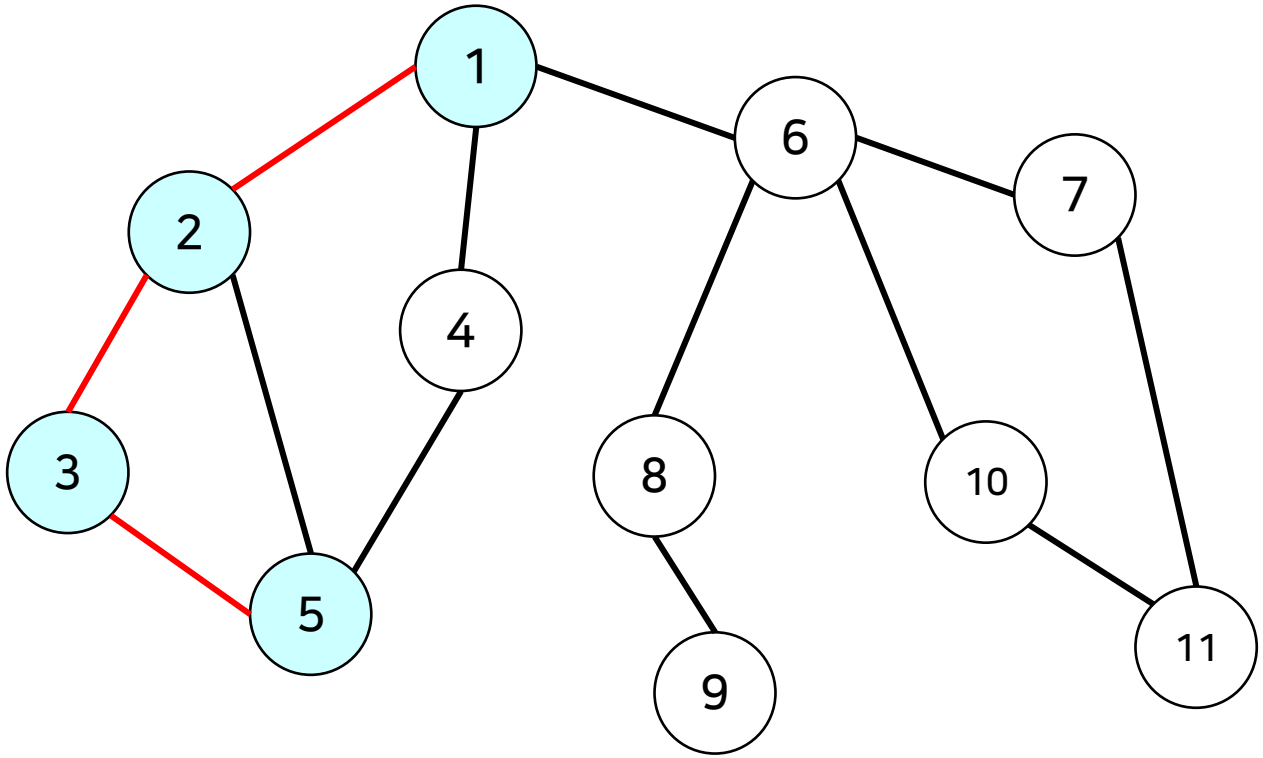
깊이 우선 탐색 (DFS)



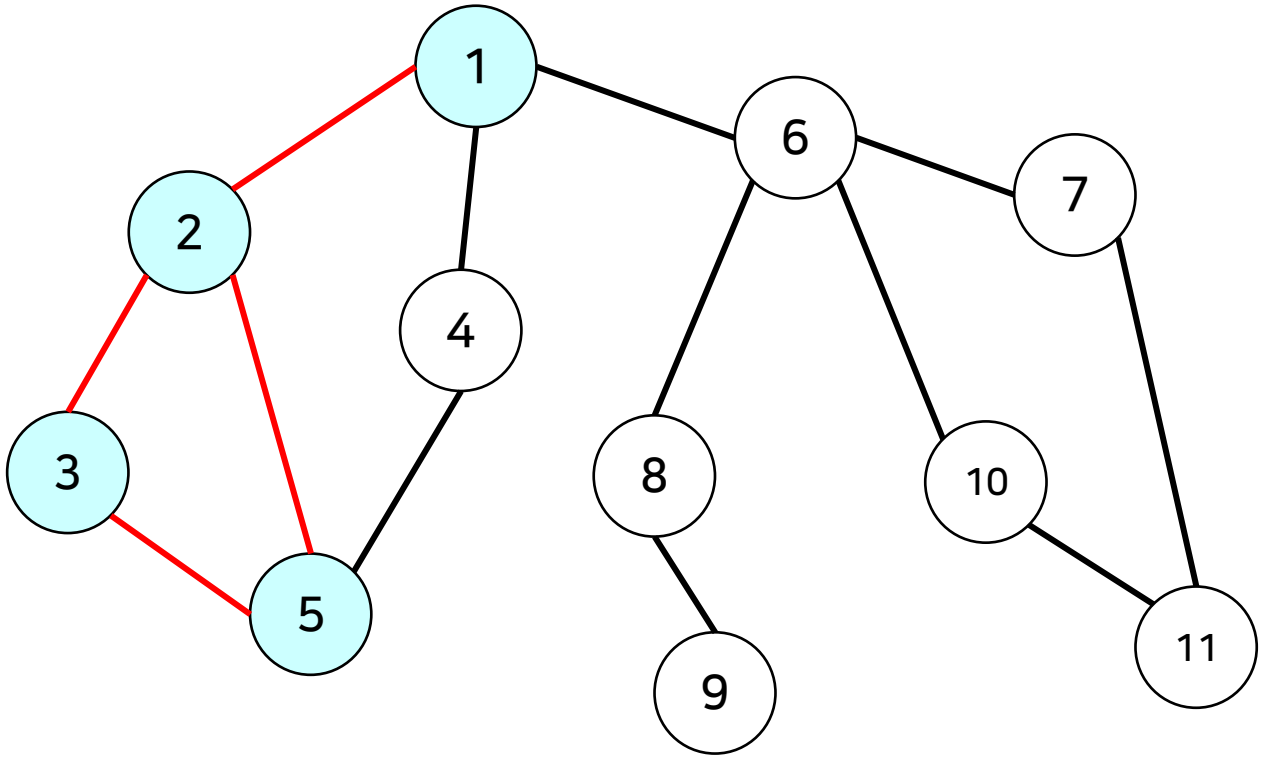
깊이 우선 탐색 (DFS)



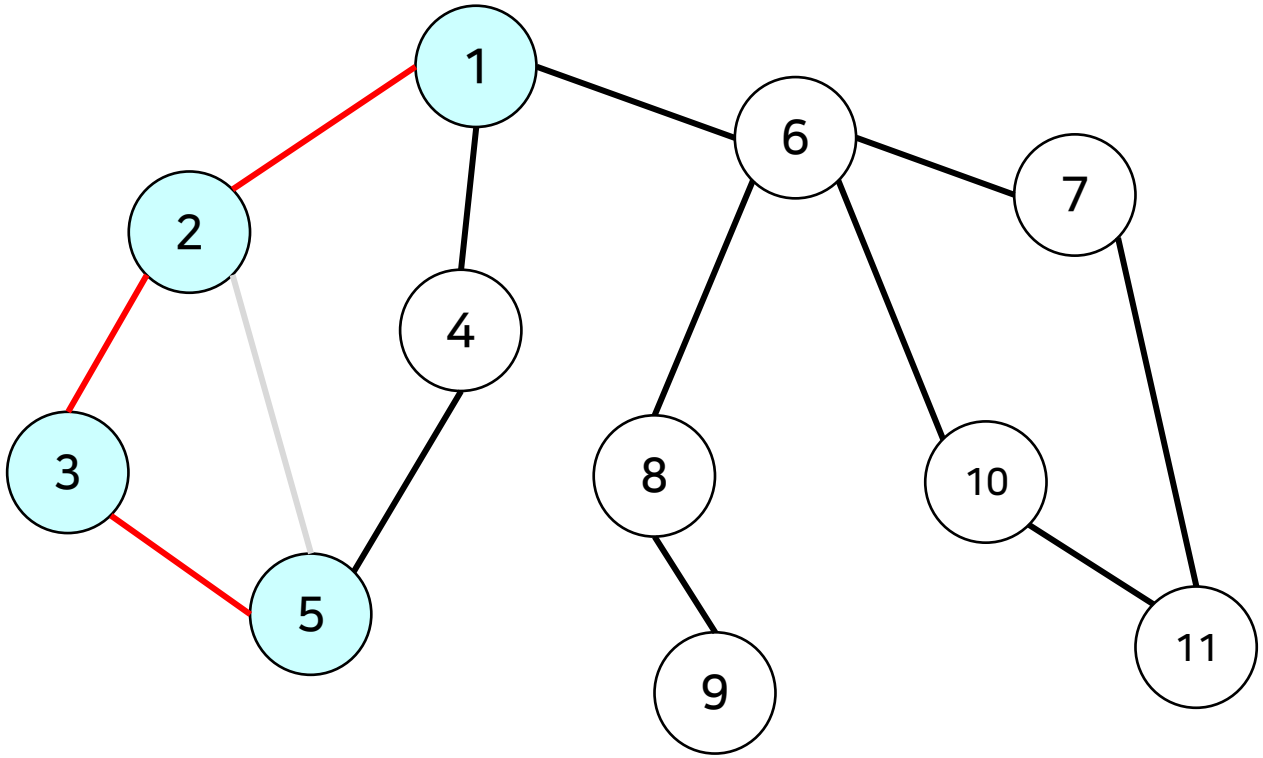
깊이 우선 탐색 (DFS)



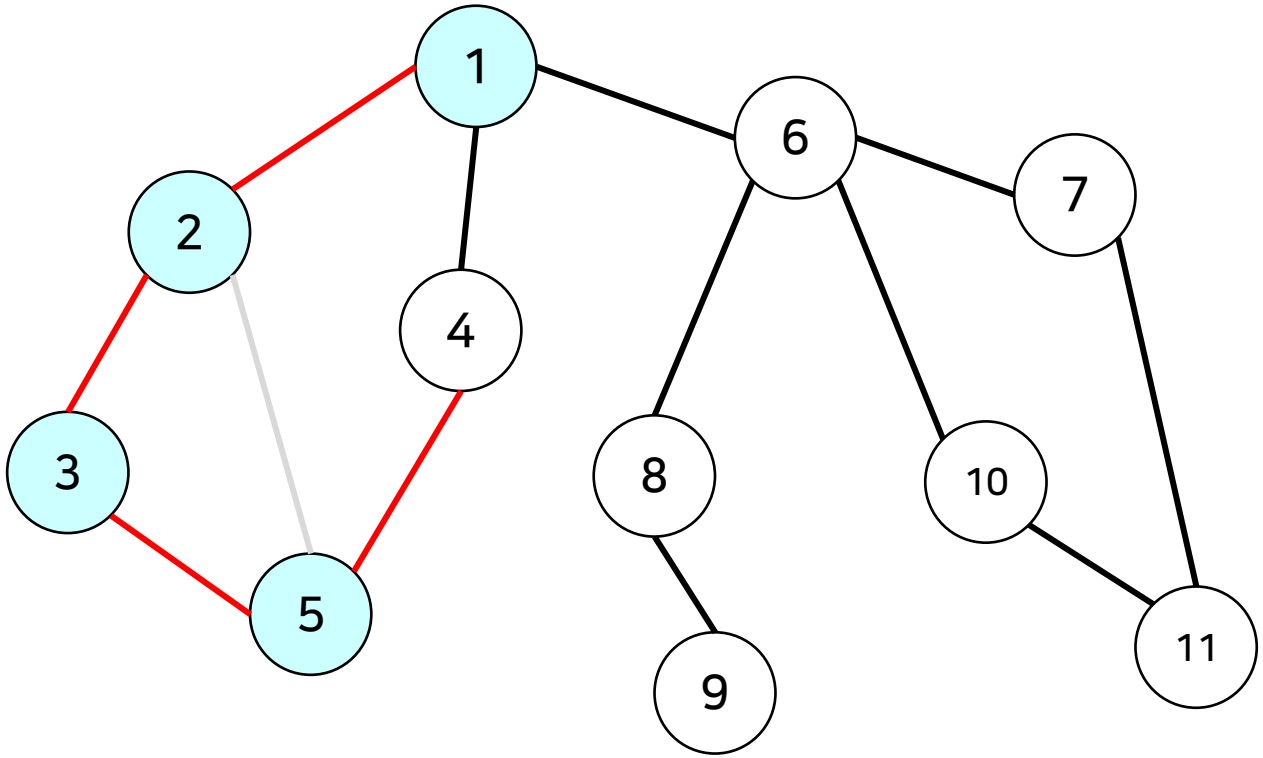
깊이 우선 탐색 (DFS)



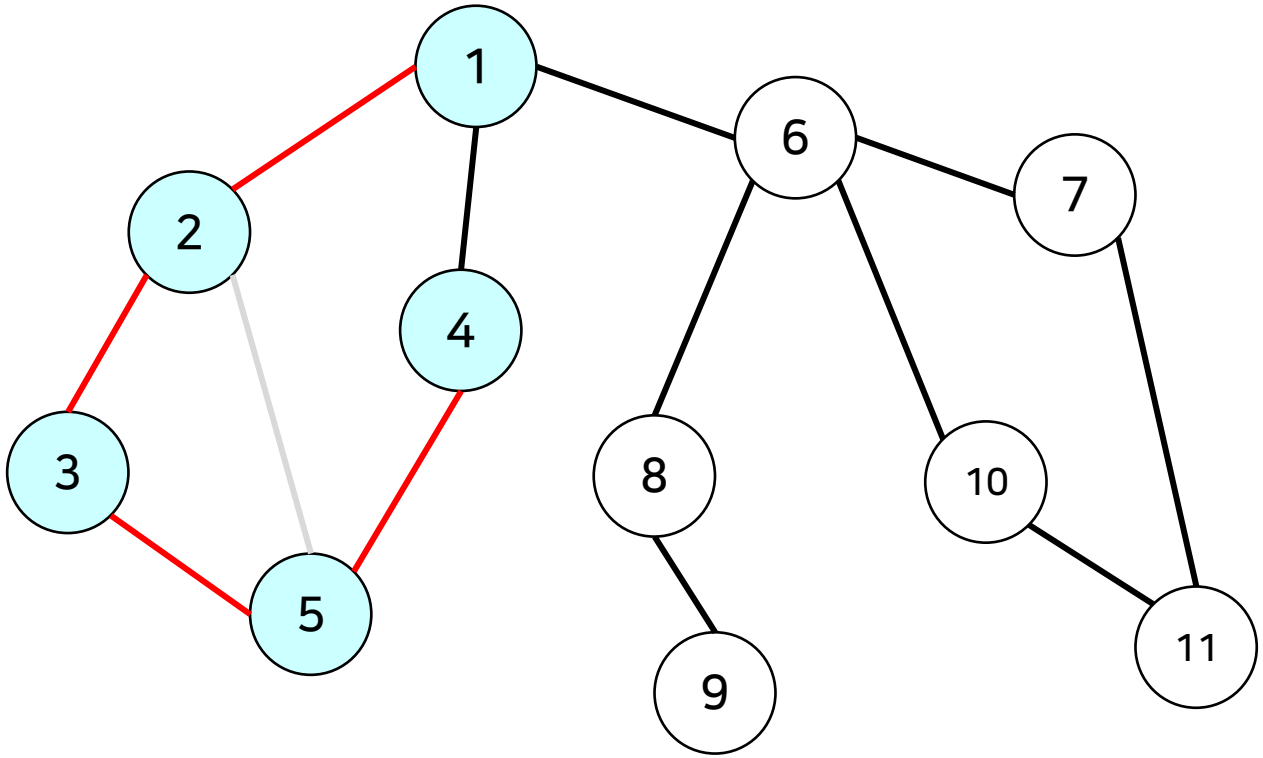
깊이 우선 탐색 (DFS)



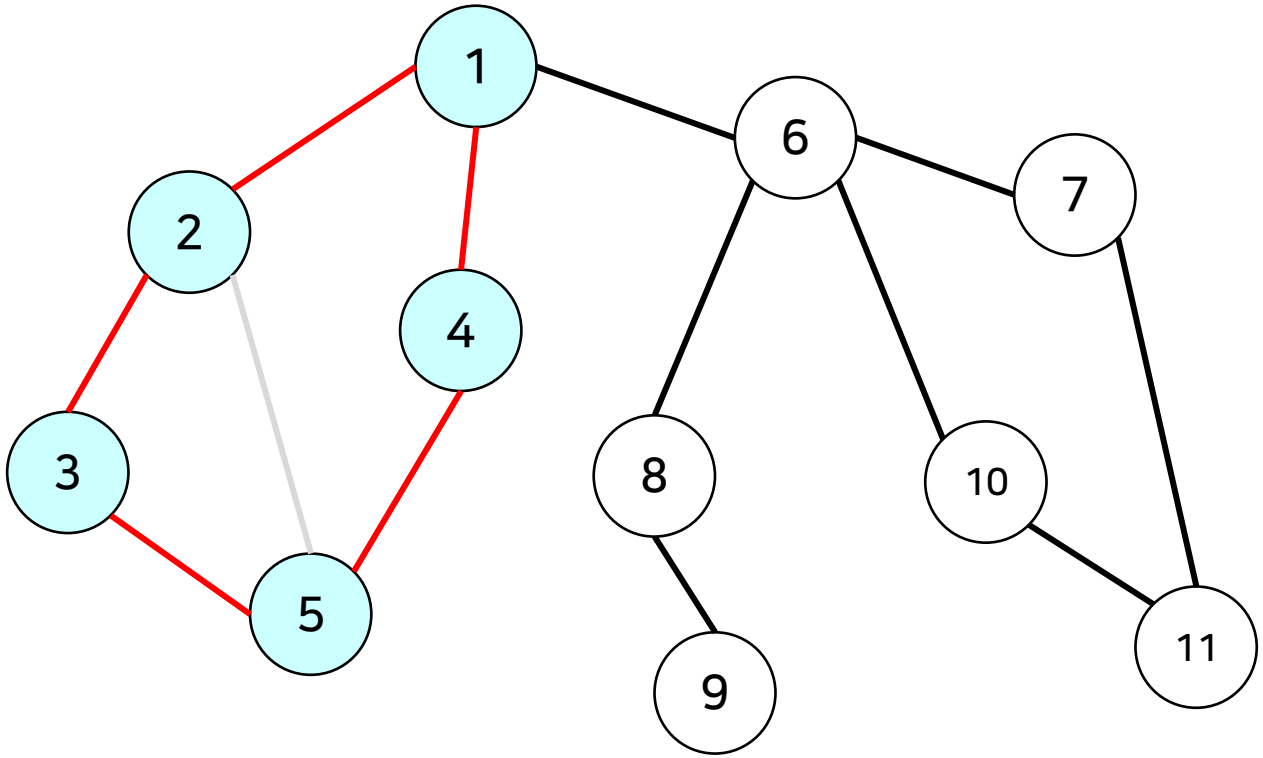
깊이 우선 탐색 (DFS)



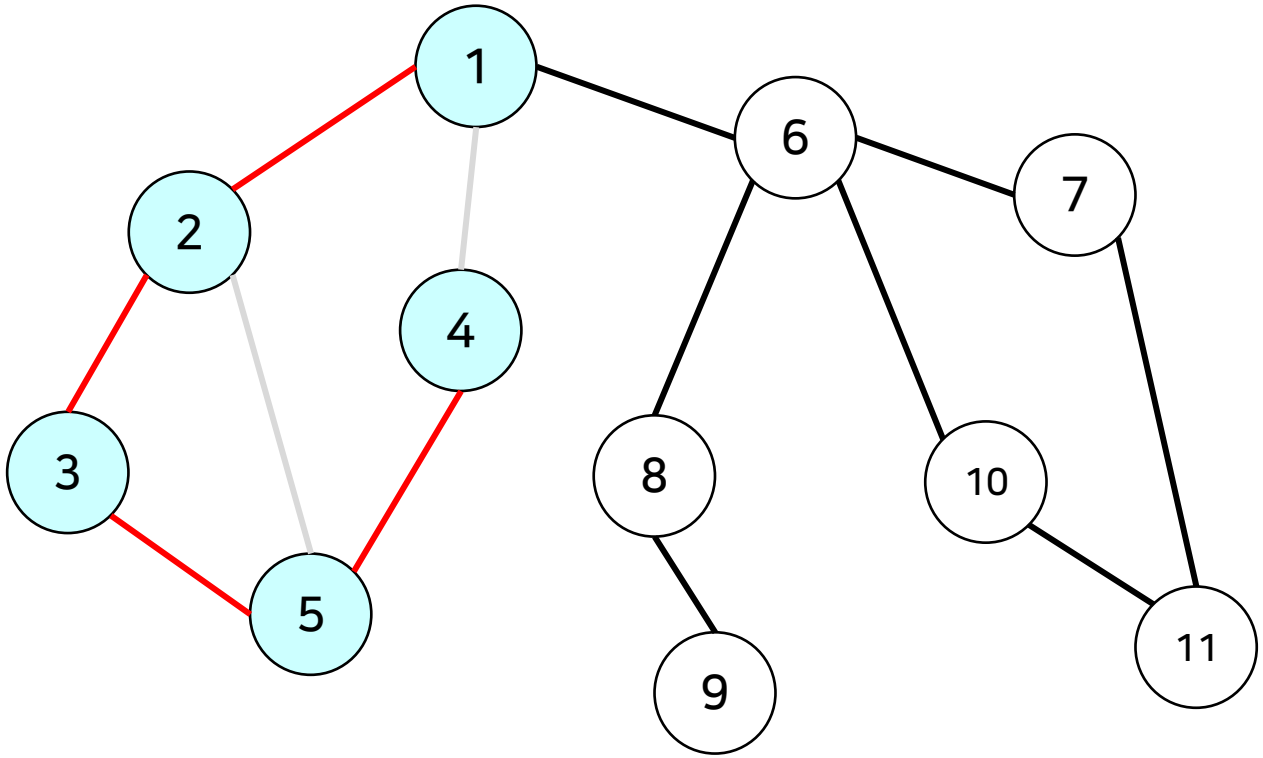
깊이 우선 탐색 (DFS)



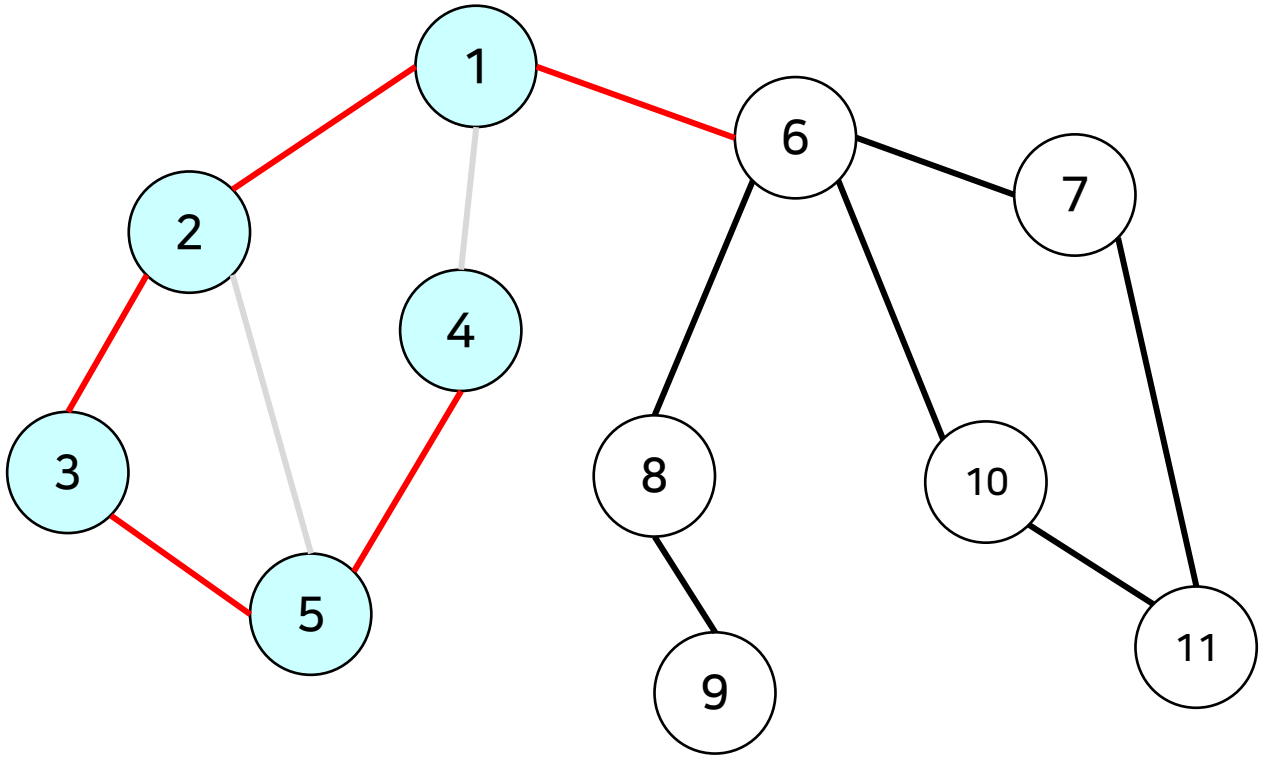
깊이 우선 탐색 (DFS)



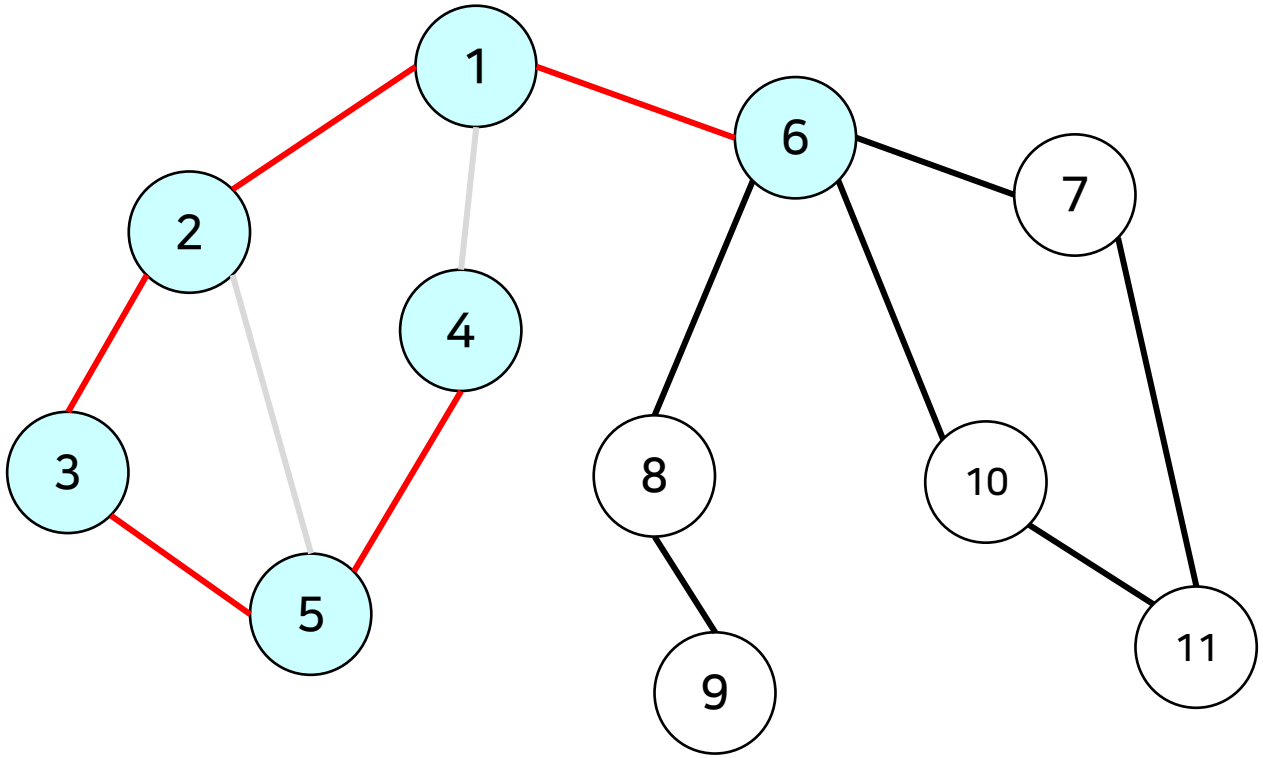
깊이 우선 탐색 (DFS)



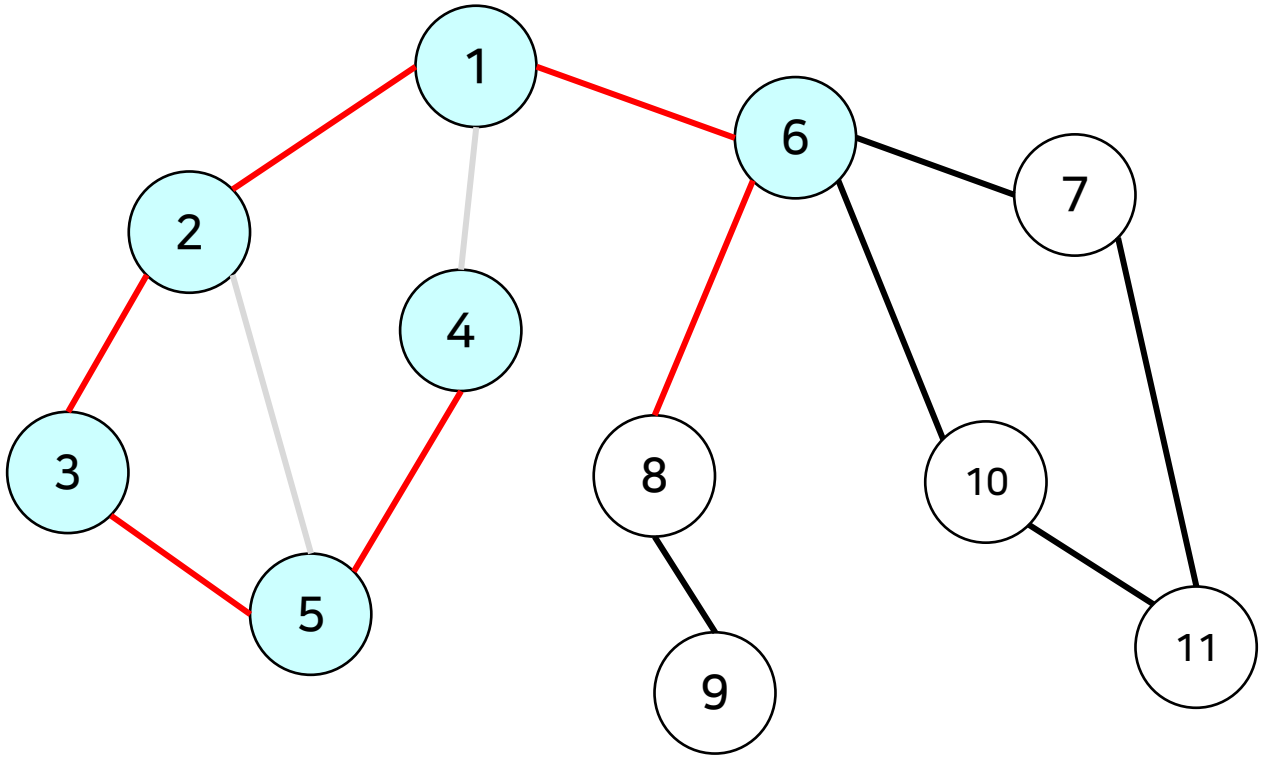
깊이 우선 탐색 (DFS)



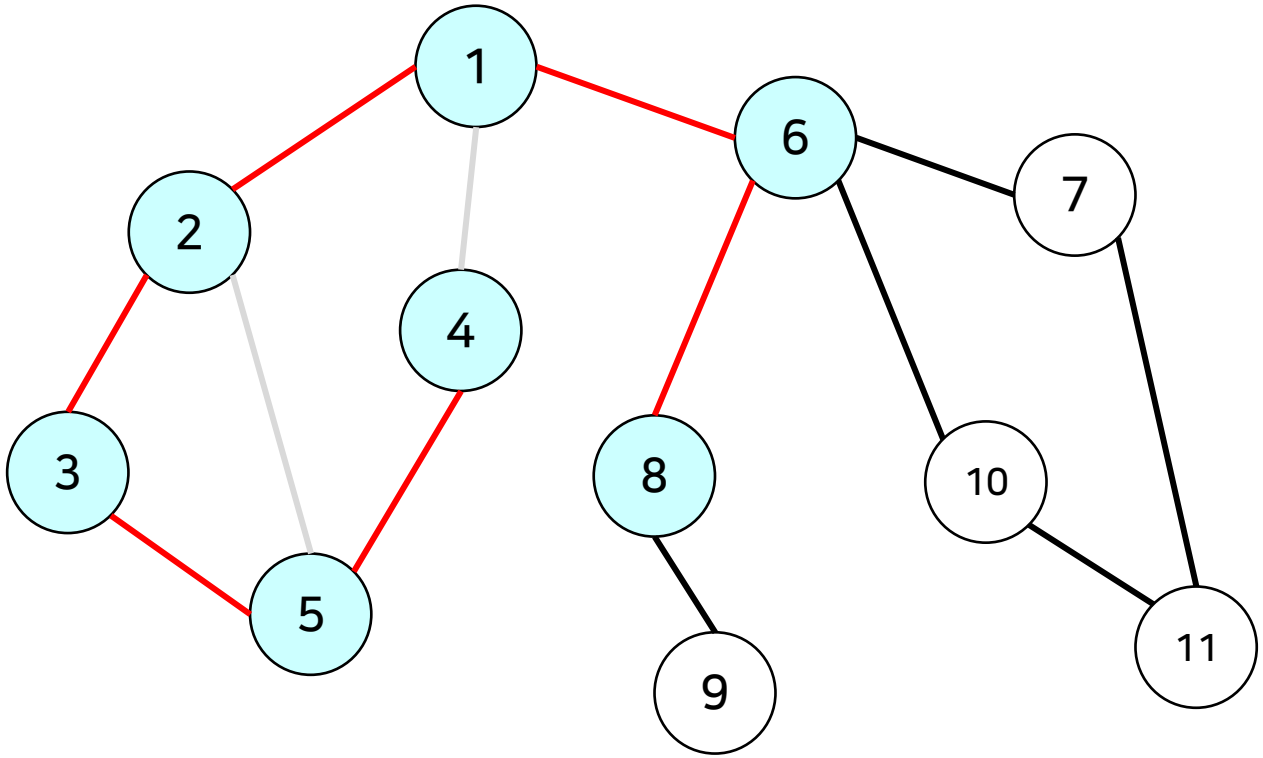
깊이 우선 탐색 (DFS)



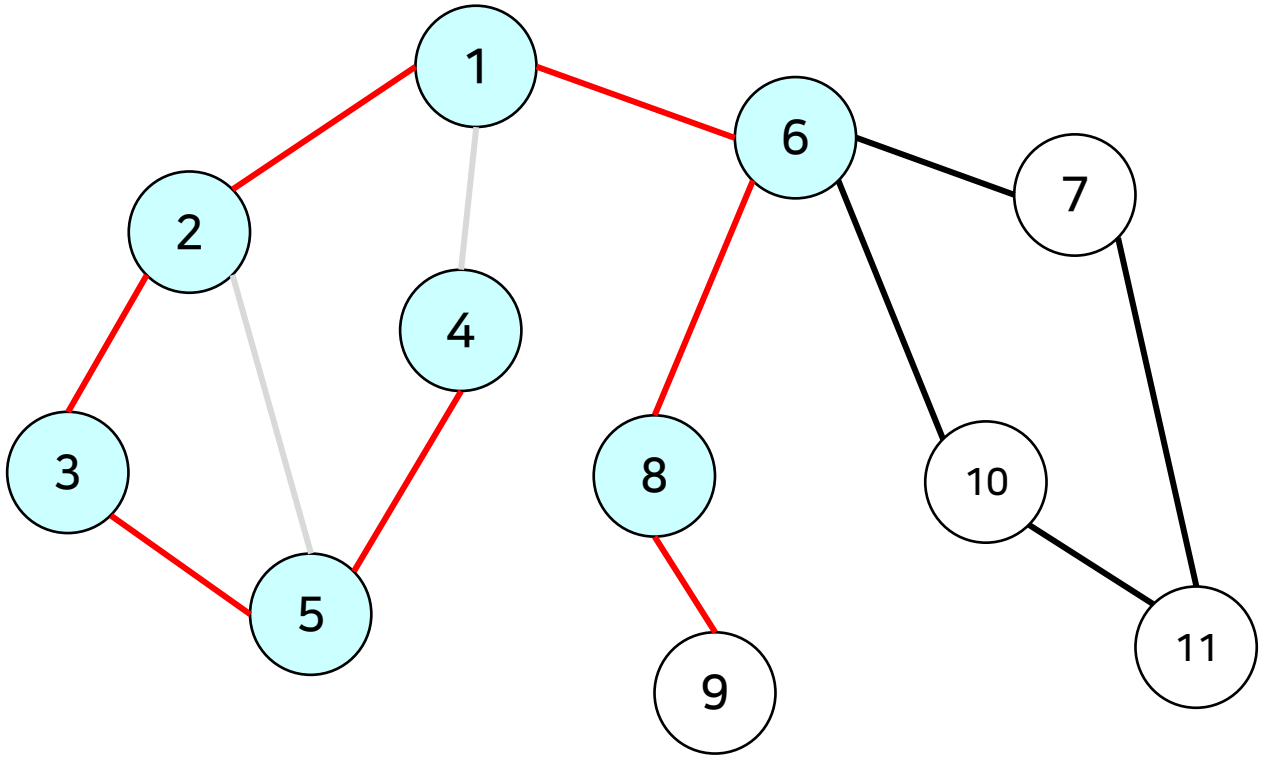
깊이 우선 탐색 (DFS)



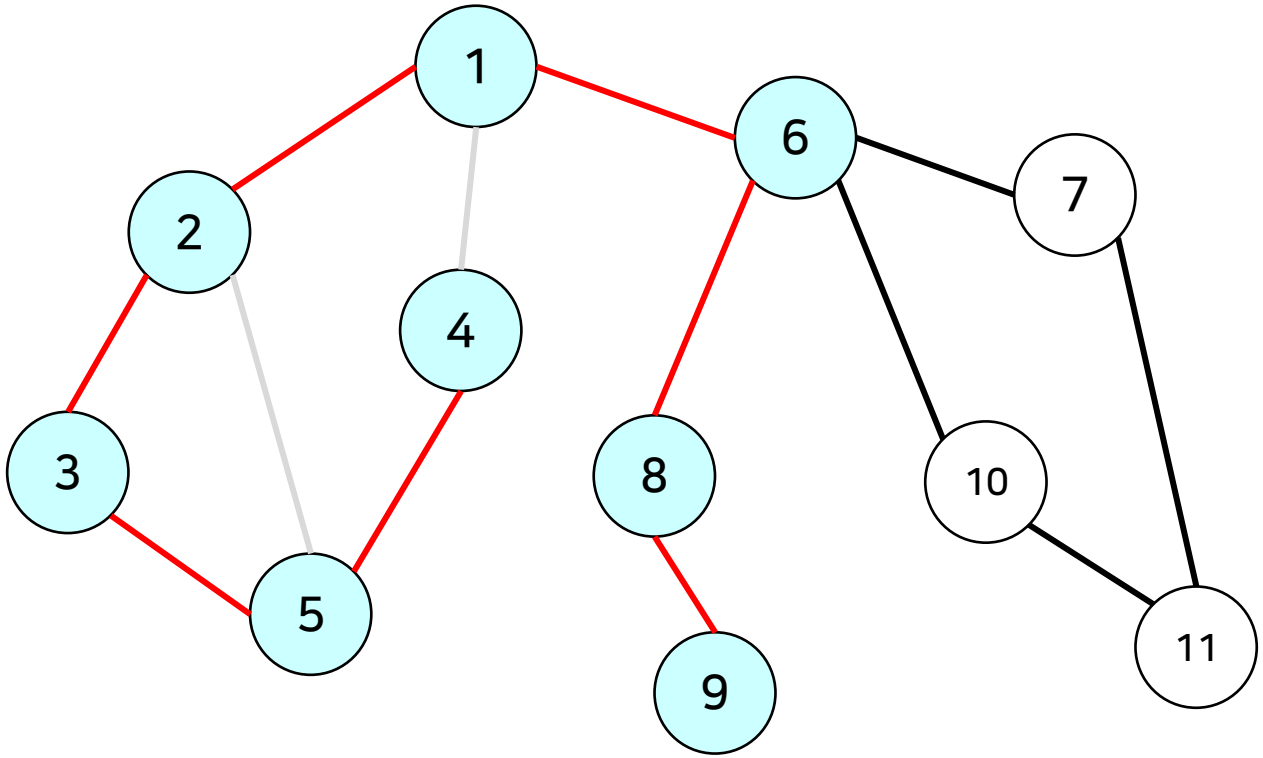
깊이 우선 탐색 (DFS)



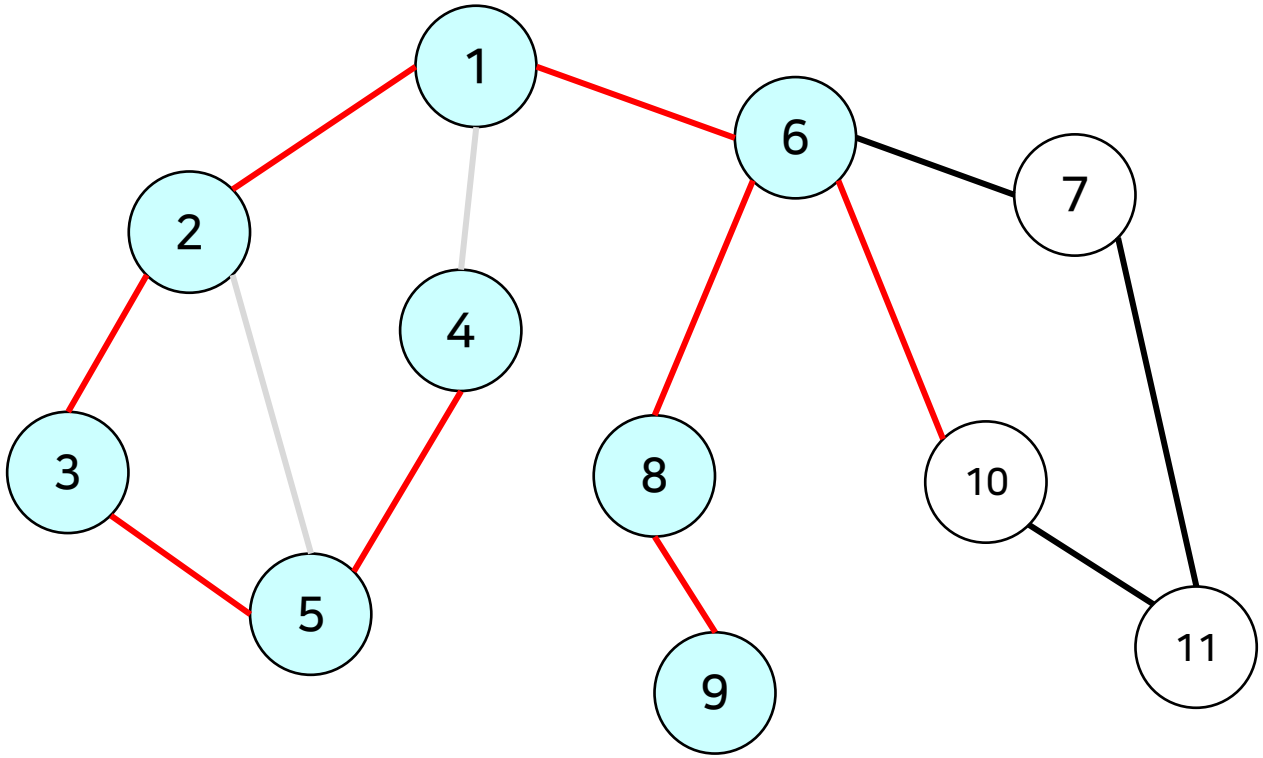
깊이 우선 탐색 (DFS)



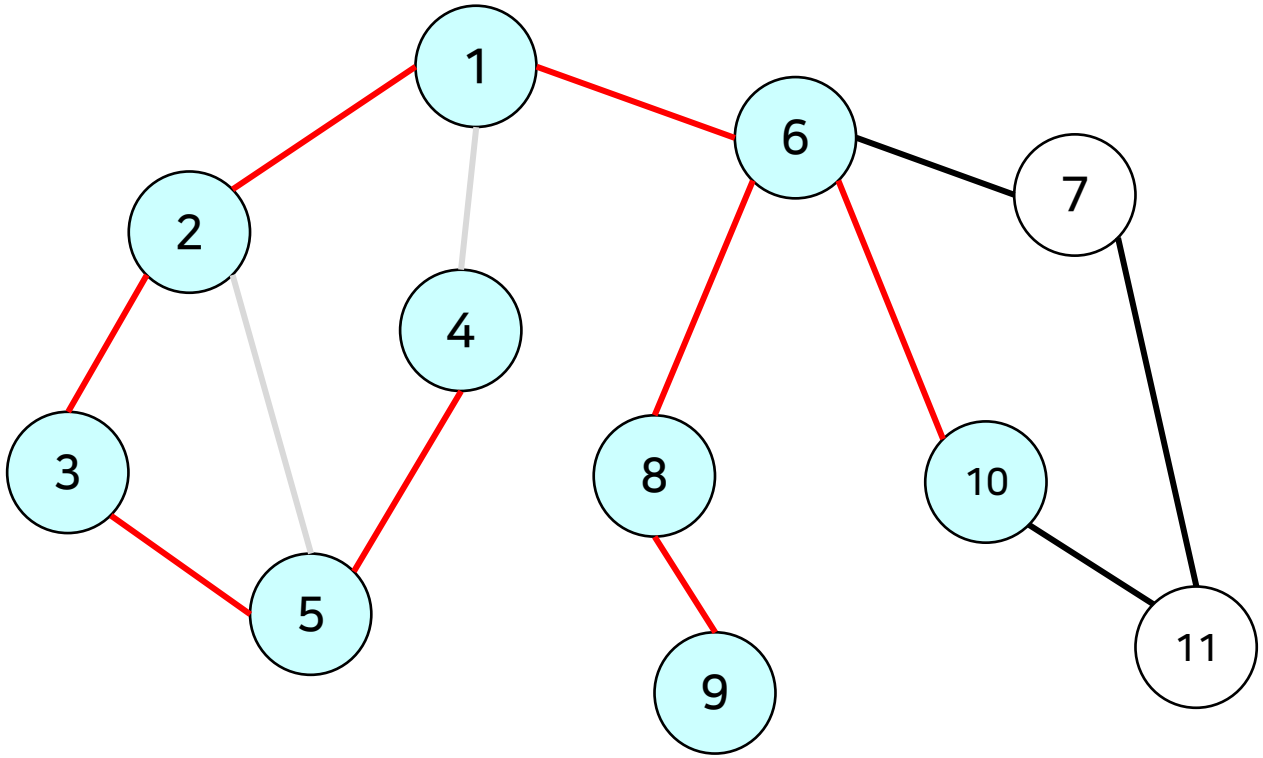
깊이 우선 탐색 (DFS)



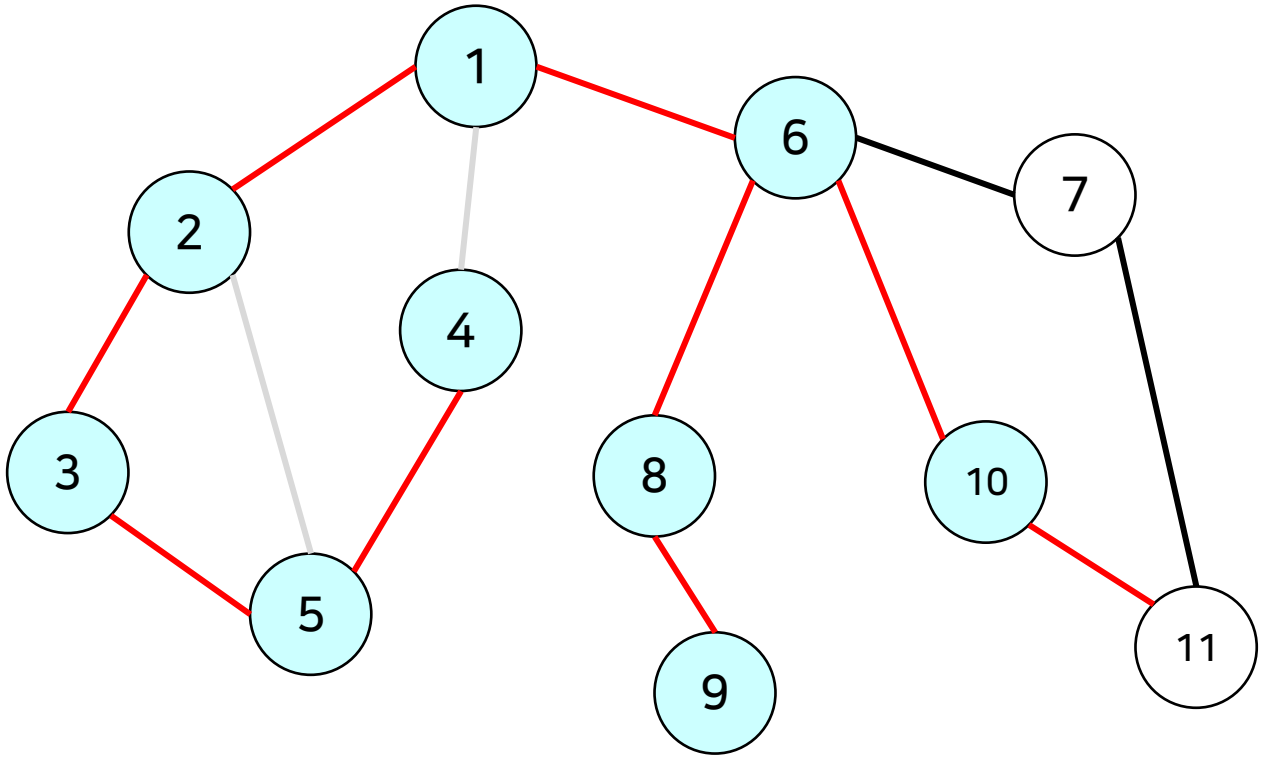
깊이 우선 탐색 (DFS)



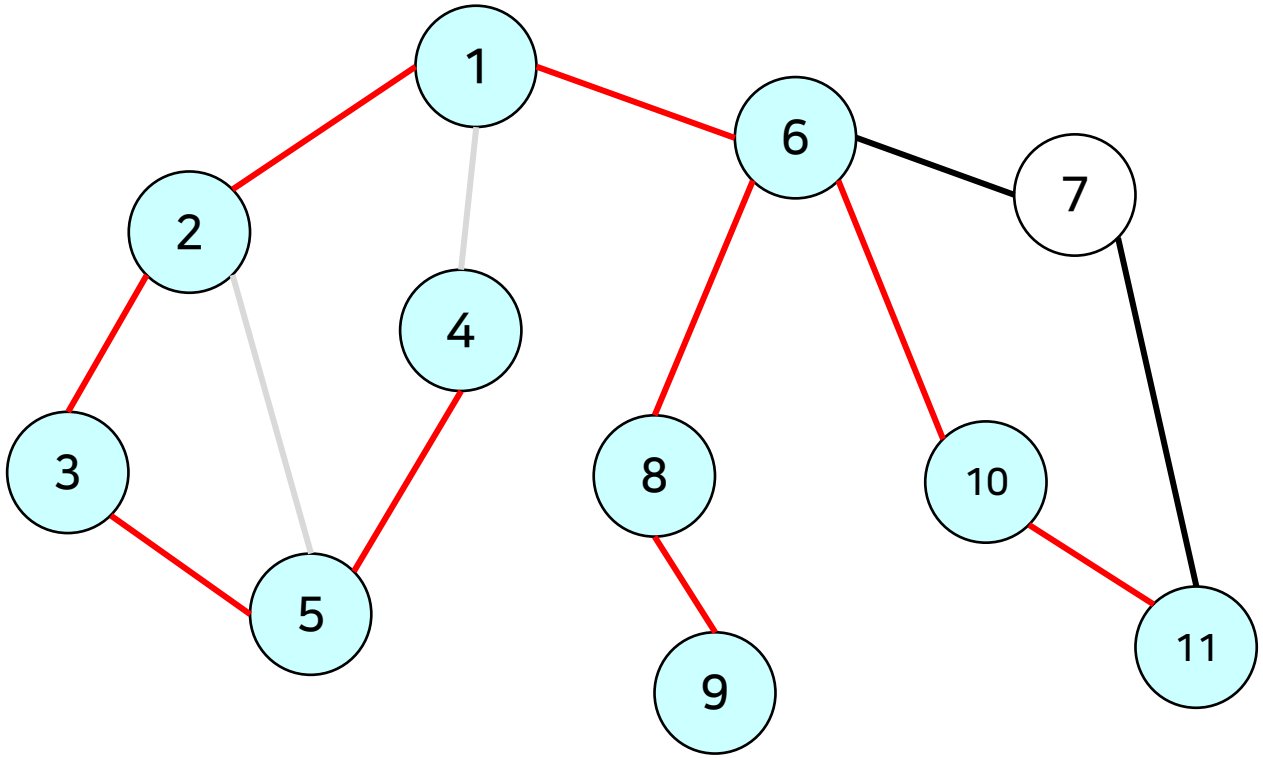
깊이 우선 탐색 (DFS)



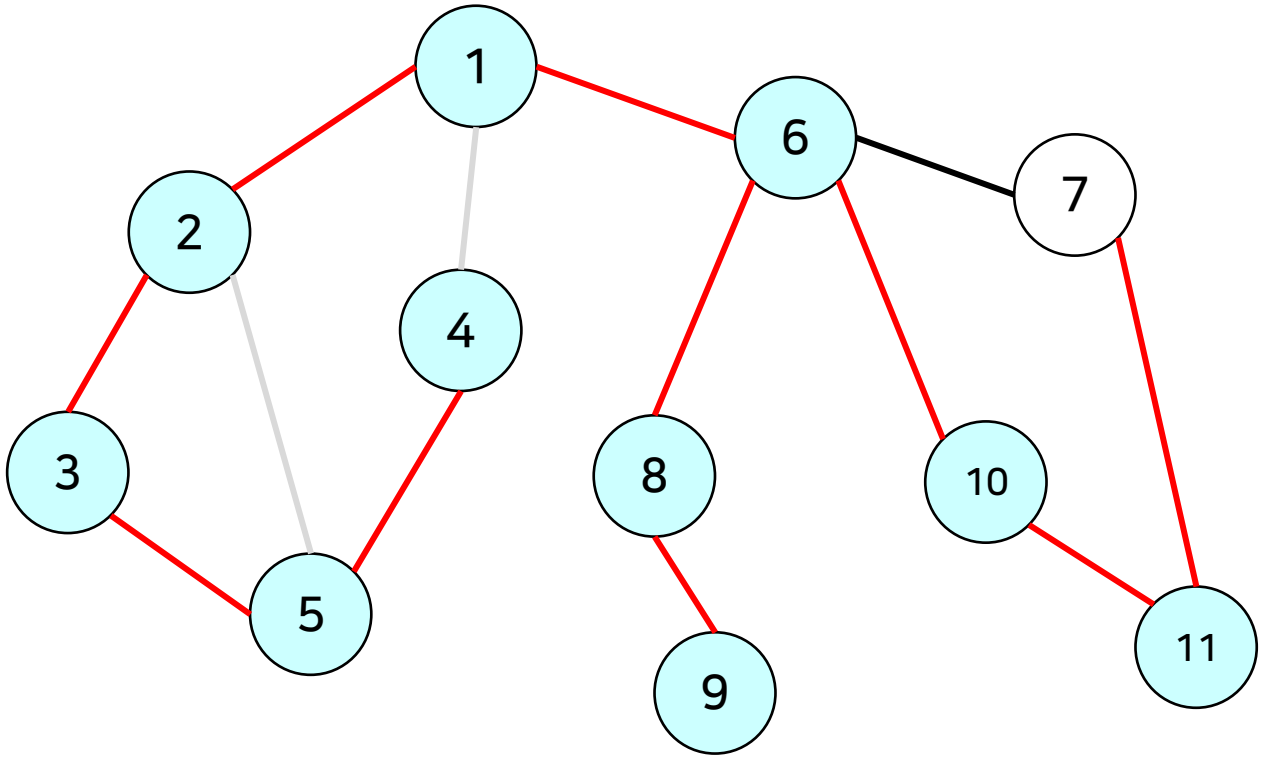
깊이 우선 탐색 (DFS)



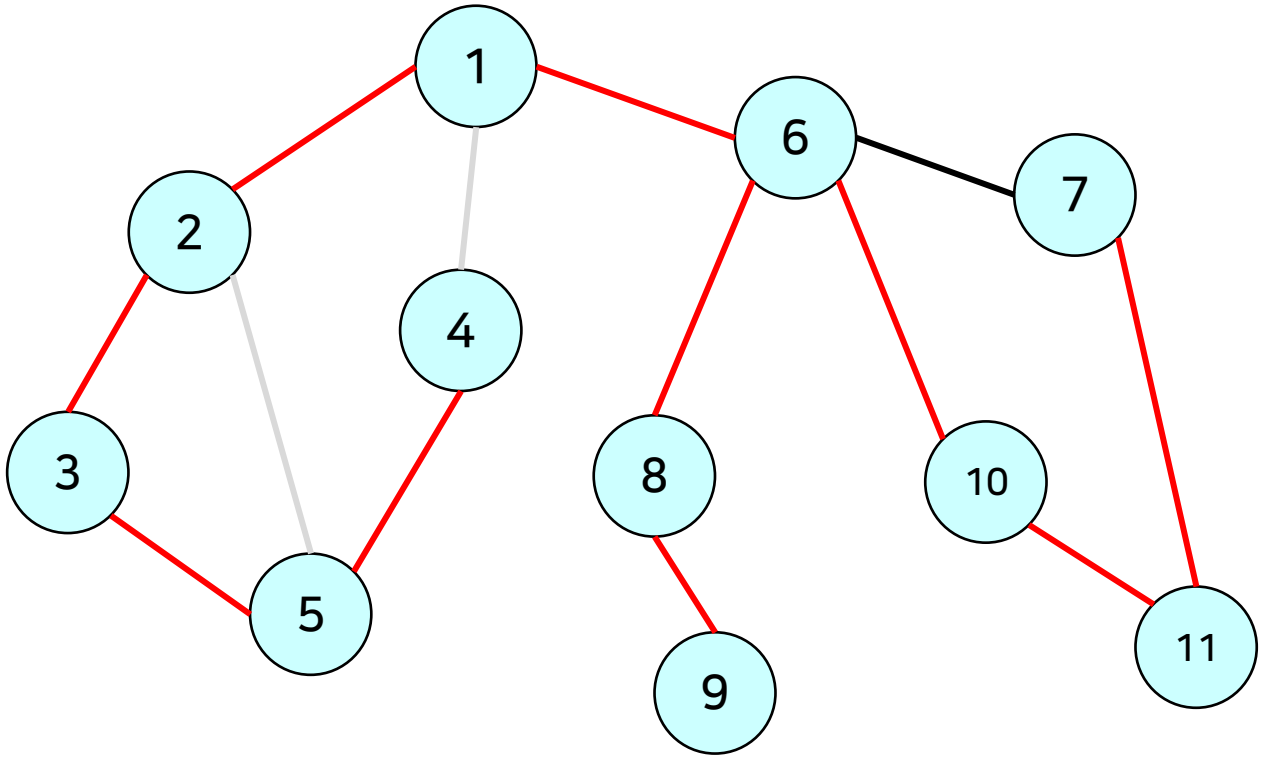
깊이 우선 탐색 (DFS)



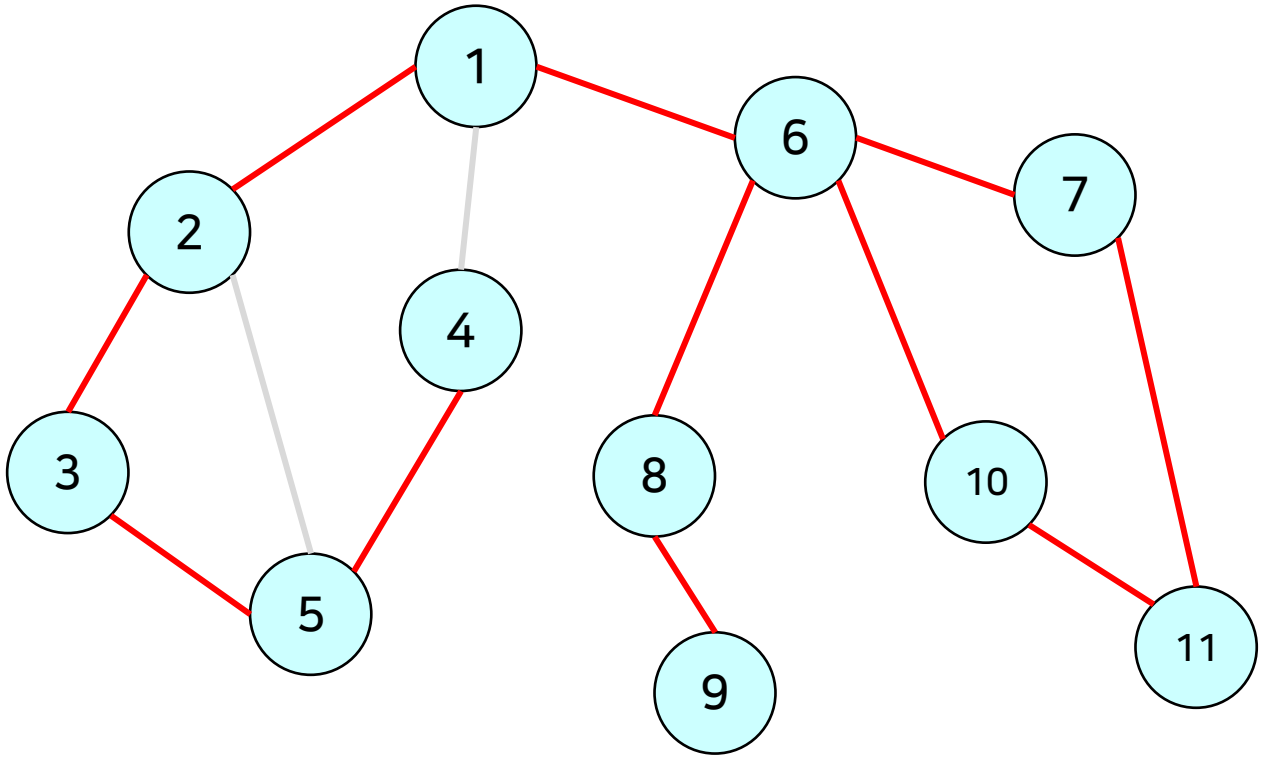
깊이 우선 탐색 (DFS)



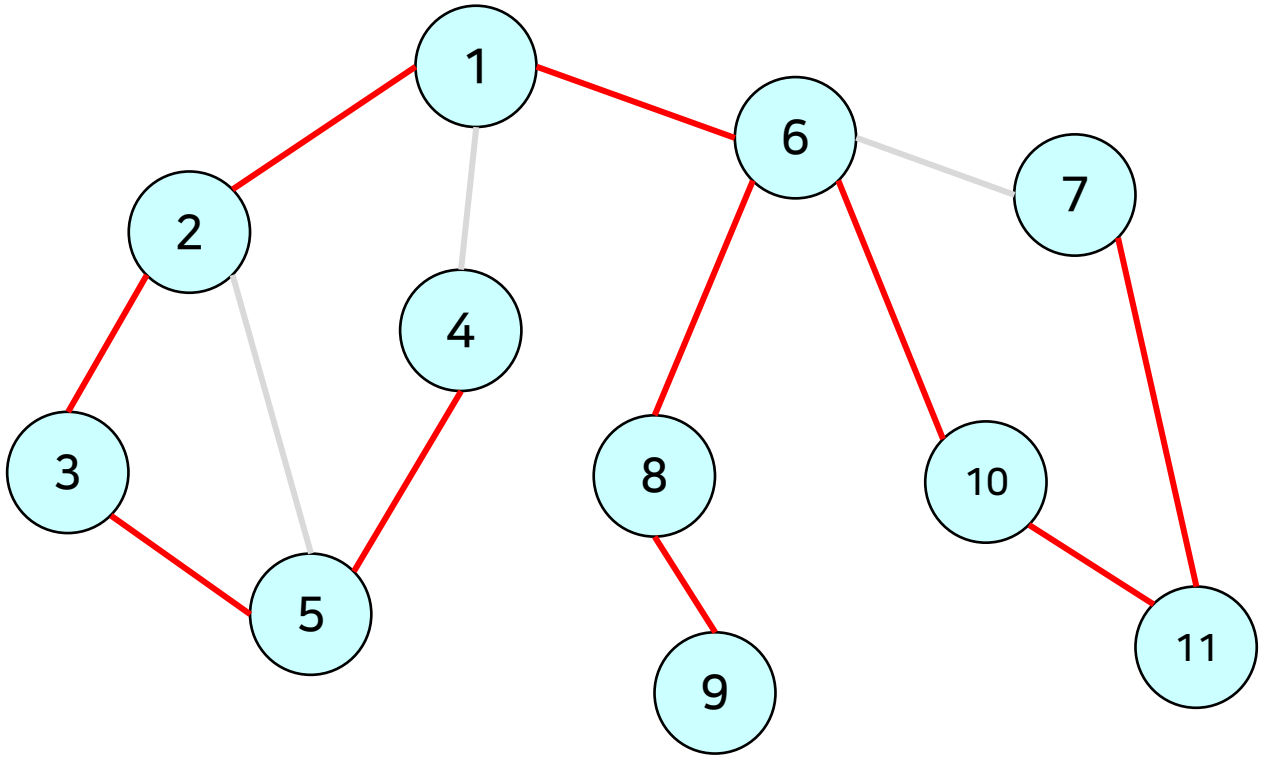
깊이 우선 탐색 (DFS)



깊이 우선 탐색 (DFS)



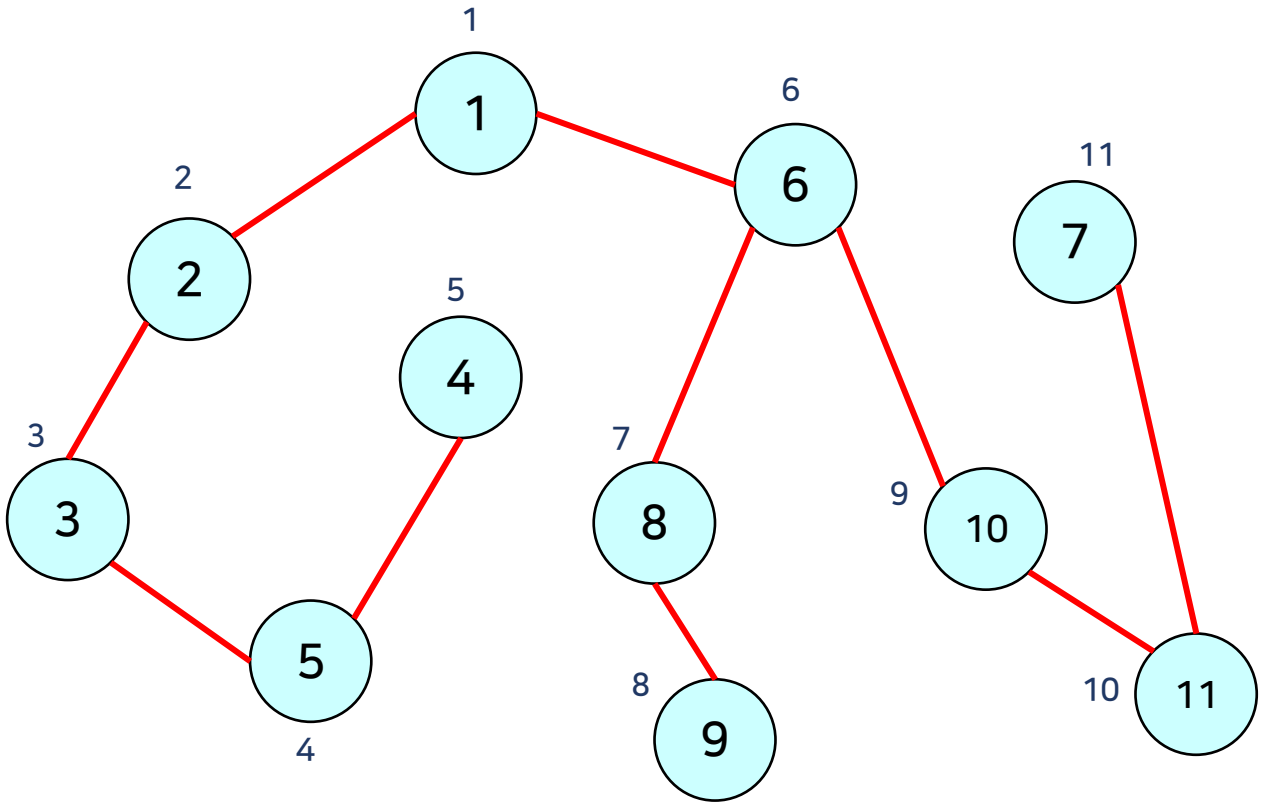
깊이 우선 탐색 (DFS)



END

깊이 우선 탐색 (DFS)

<DFS Tree>



깊이 우선 탐색 (DFS)

```cpp
#include<iostream>
#include<vector>
const int MAX = 10000;
using namespace std;

bool visited[MAX];
vector<int> adj[MAX];
void dfs(int u) {
    visited[u] = true;
    for (int i = 0; i < adj[u].size(); i++) {
        if (!visited[adj[u][i]]) {
            dfs(adj[u][i]);
        }
    }
}
```

```cpp
void dfs(int u) {
    visited[u] = true;
    for (int v : adj[u]) {
        if (!visited[v]) {
            dfs(v);
        }
    }
}
```

- Range-based for loop (since C++11)

- auto 키워드도 알아보기! (since C++11)

너비 우선 탐색 (BFS)

- Breadth First Search (BFS)

- 인접한 정점들을 먼저 탐색하는 방법

- 가까운 정점을 먼저, 먼 정점을 나중에

- 큐(queue)로 구현

너비 우선 탐색 (BFS)

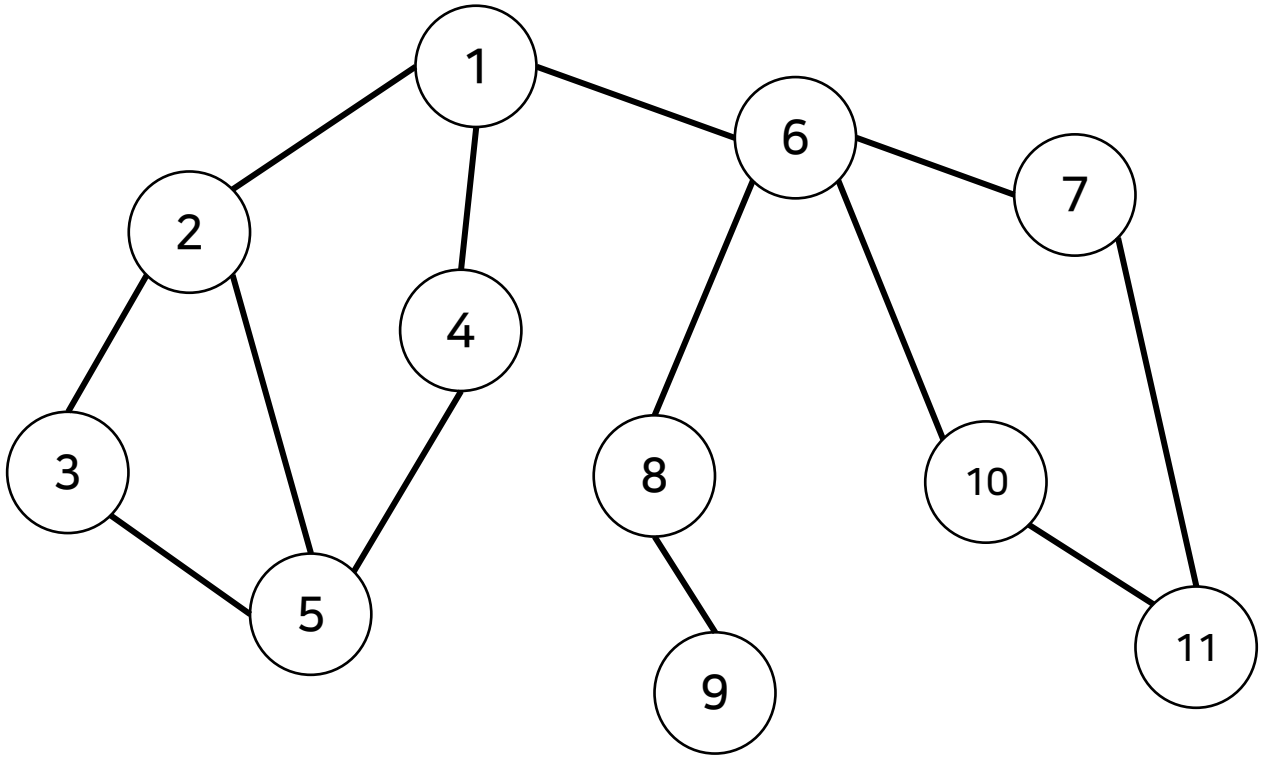
1. 정점 1개를 선택한다.

2. 현재 정점(u)과 이웃한 정점(v) 중 방문하지 않은 정점을 모두 방문한다.

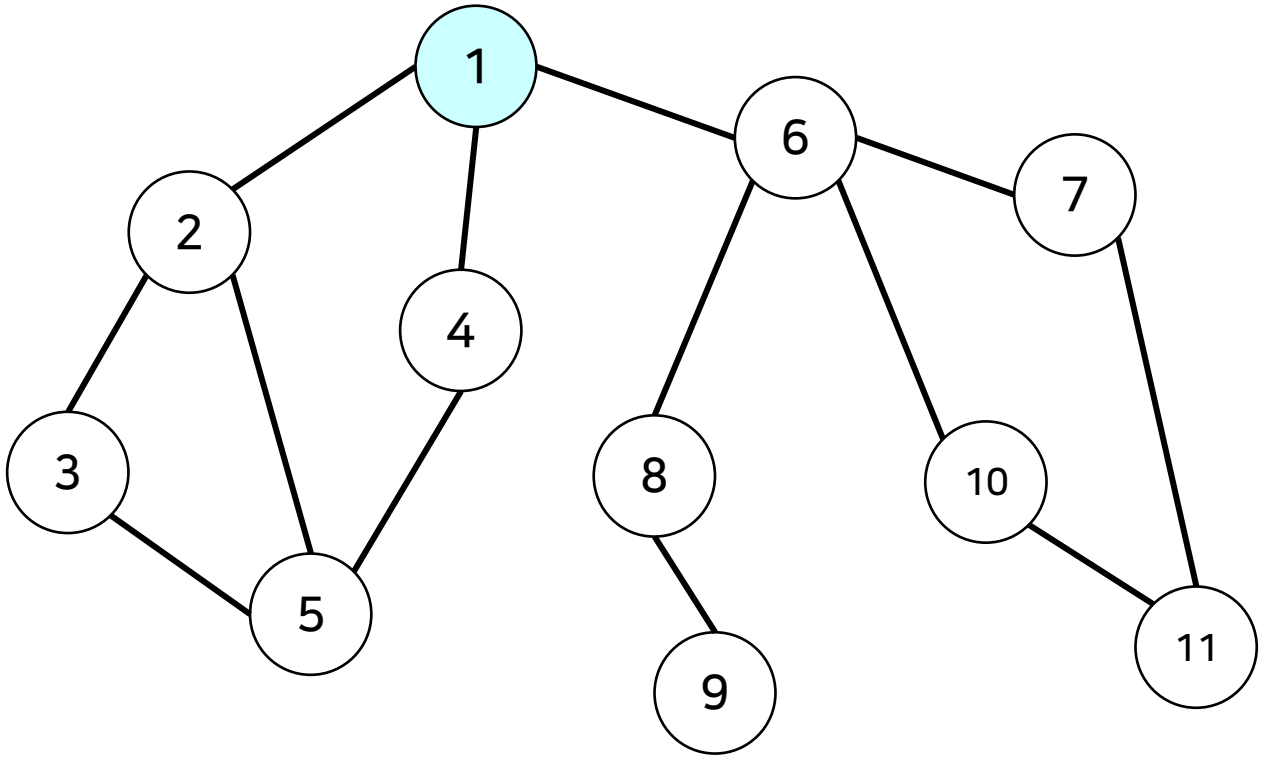
3. v 중 가장 먼저 방문한 이웃한 정점으로 이동한다.

4. 모든 정점을 방문할 때까지 2~3을 반복한다.

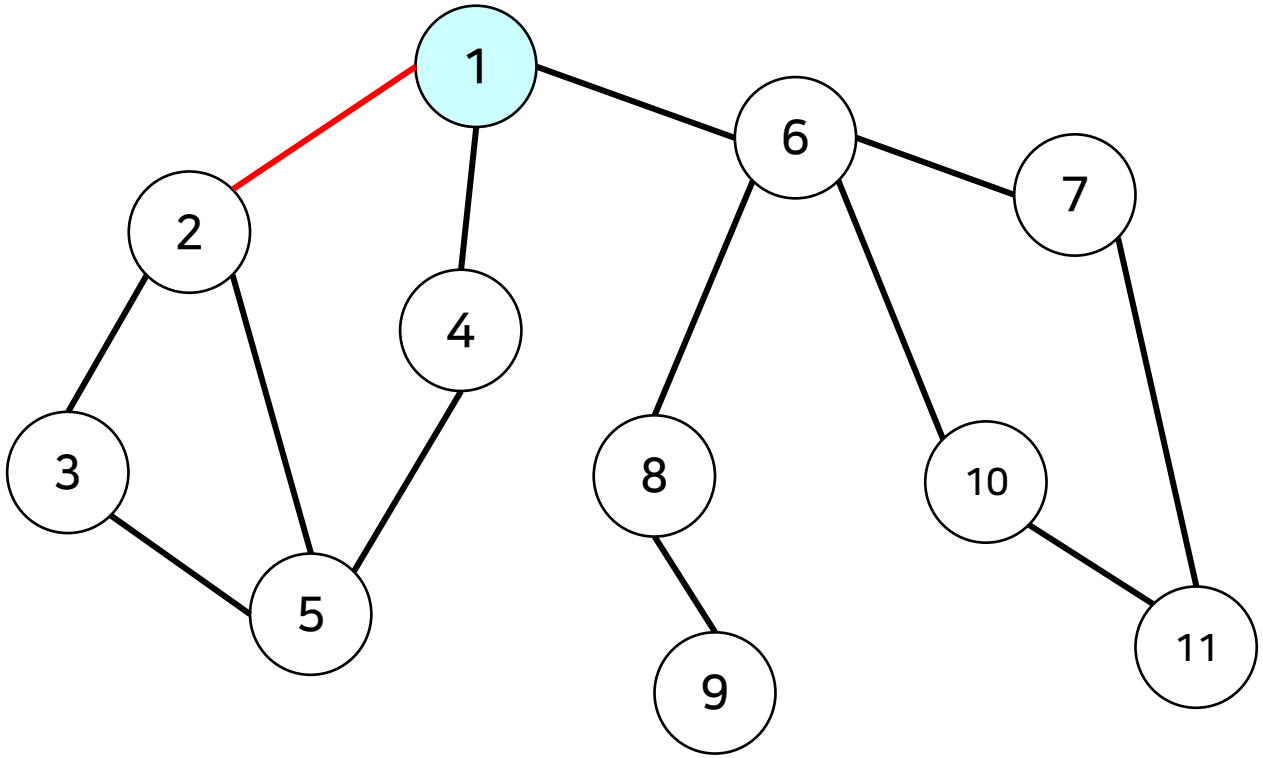
너비 우선 탐색 (BFS)



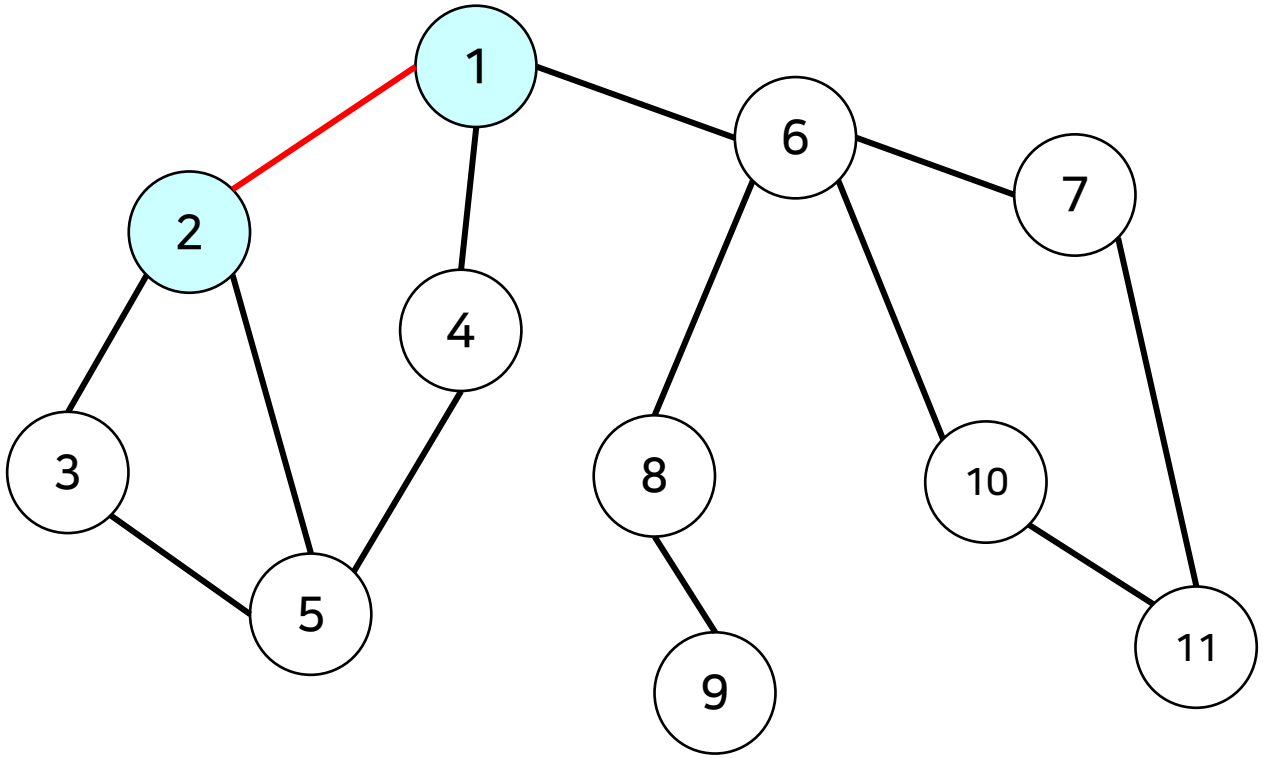
너비 우선 탐색 (BFS)



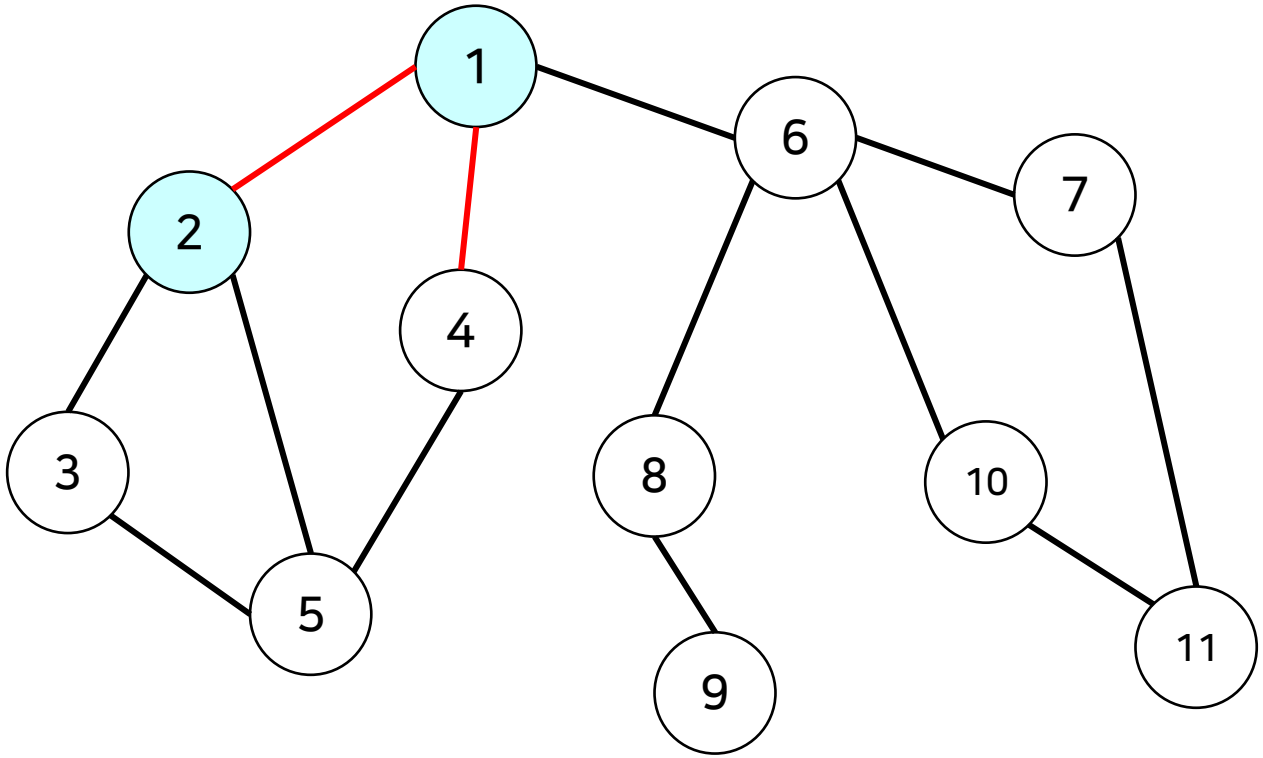
너비 우선 탐색 (BFS)



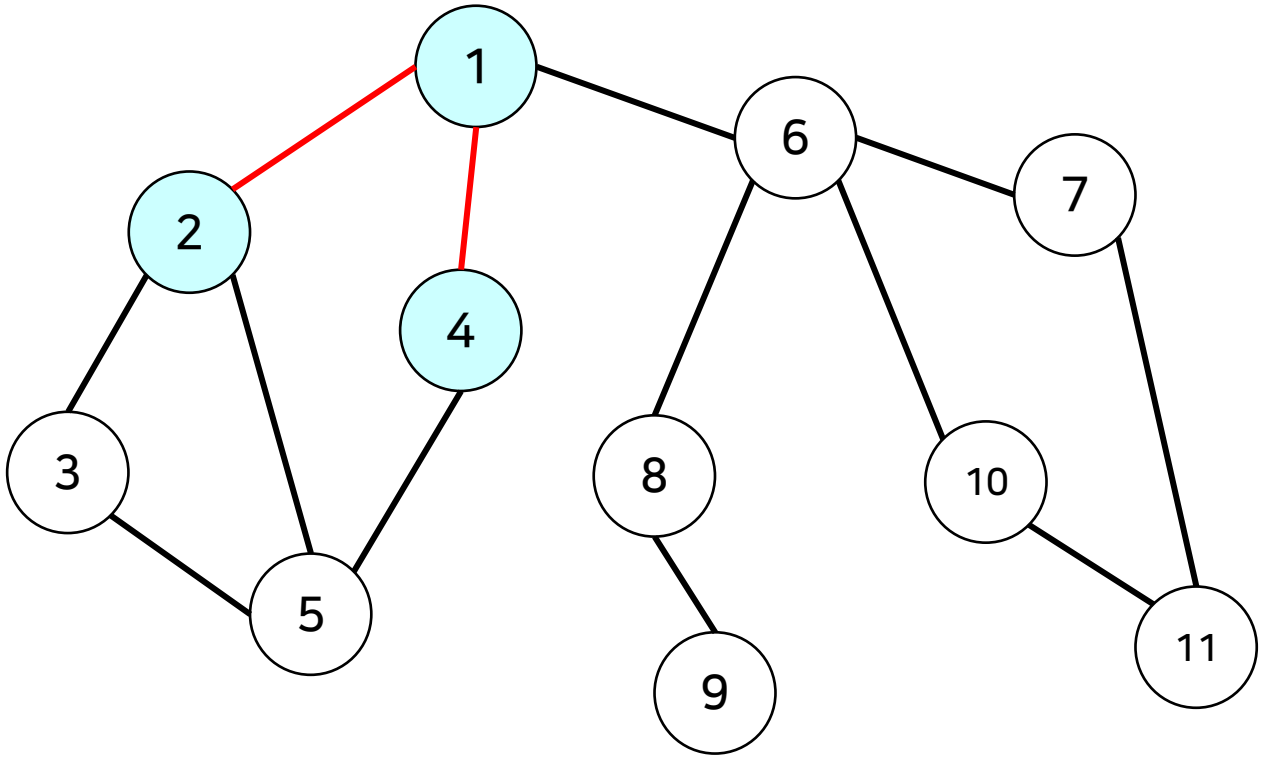
너비 우선 탐색 (BFS)



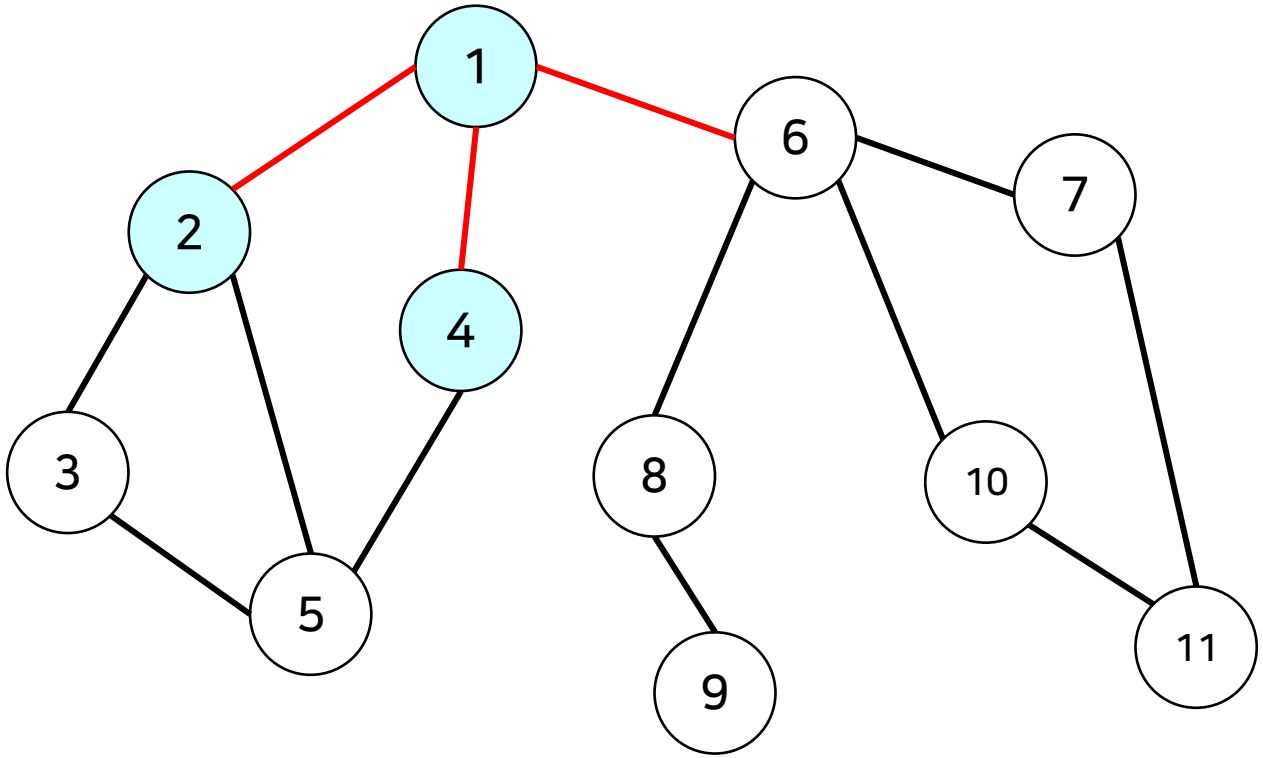
너비 우선 탐색 (BFS)



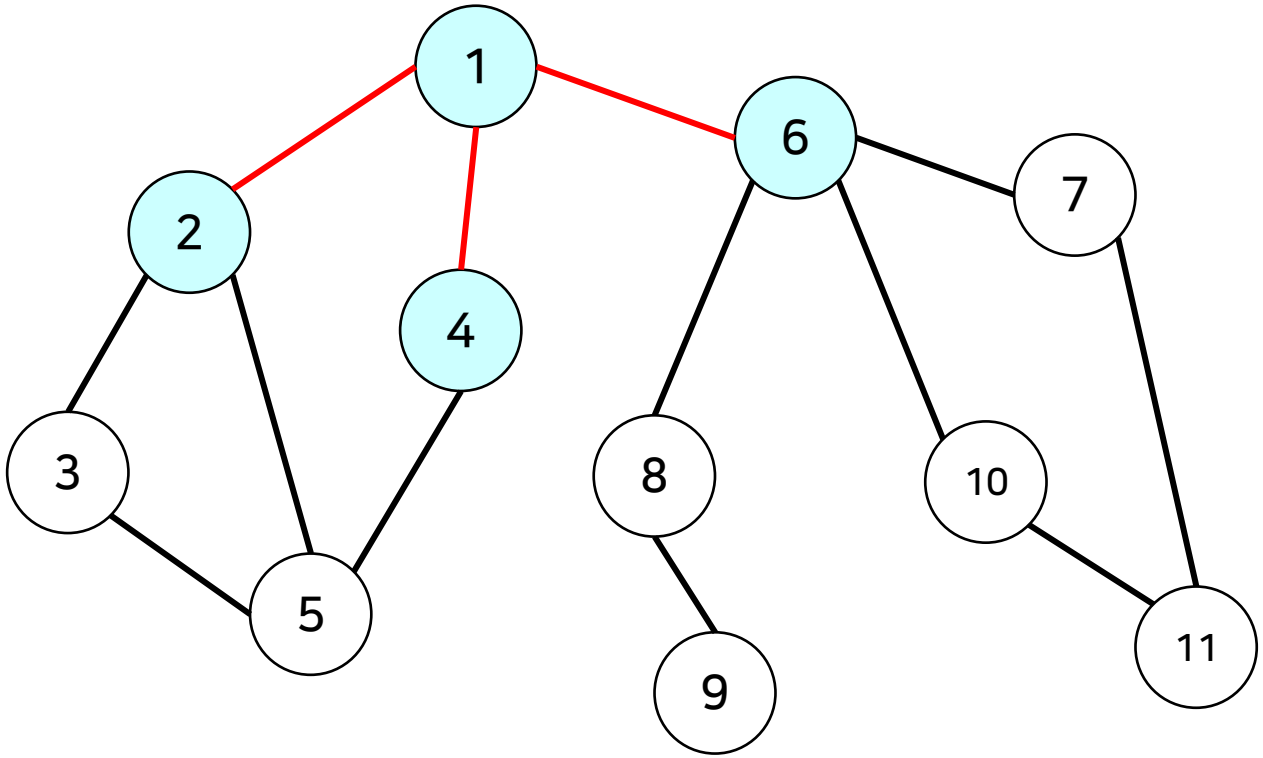
너비 우선 탐색 (BFS)



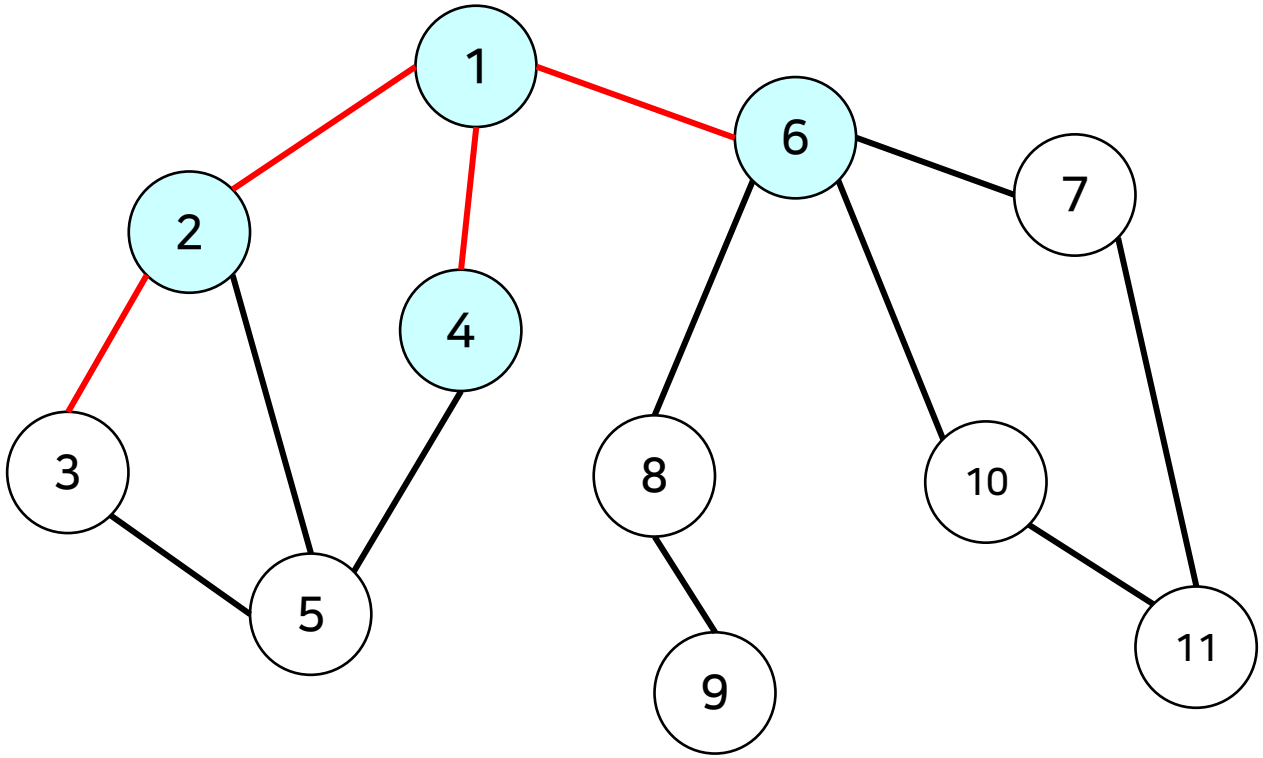
너비 우선 탐색 (BFS)



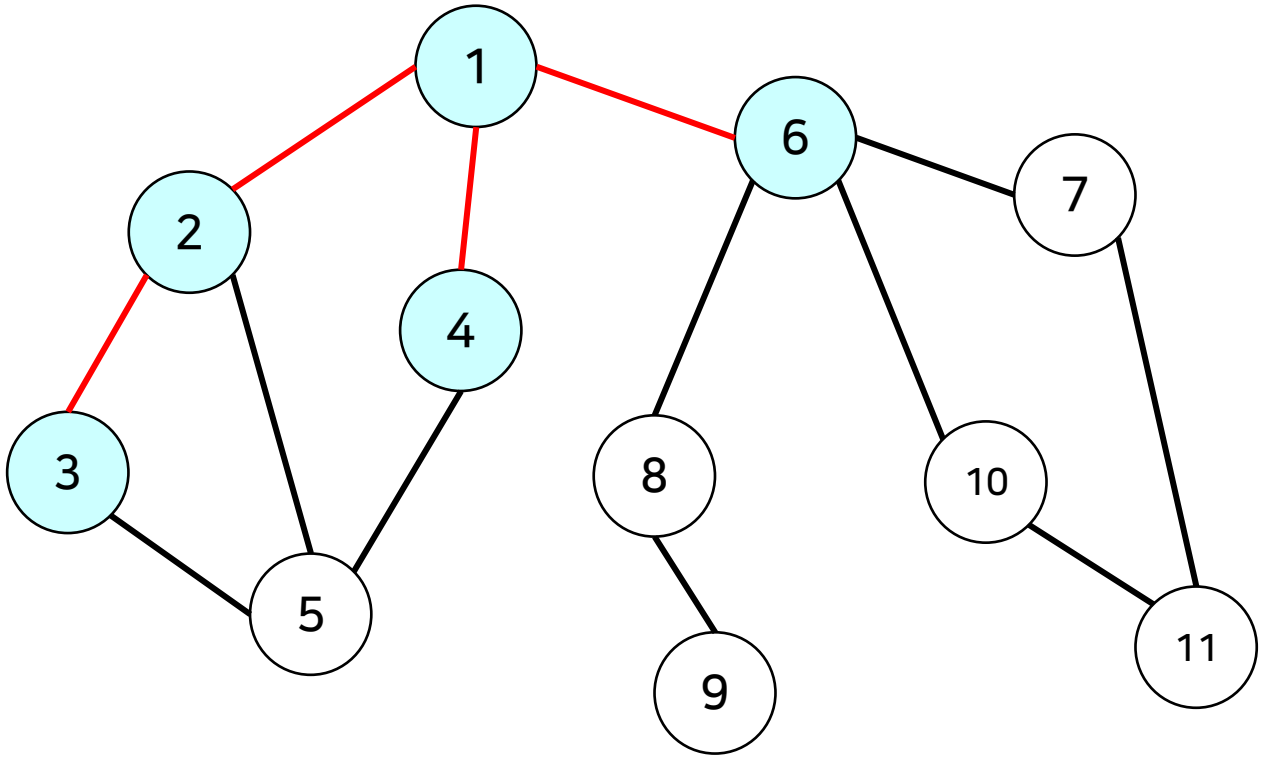
너비 우선 탐색 (BFS)



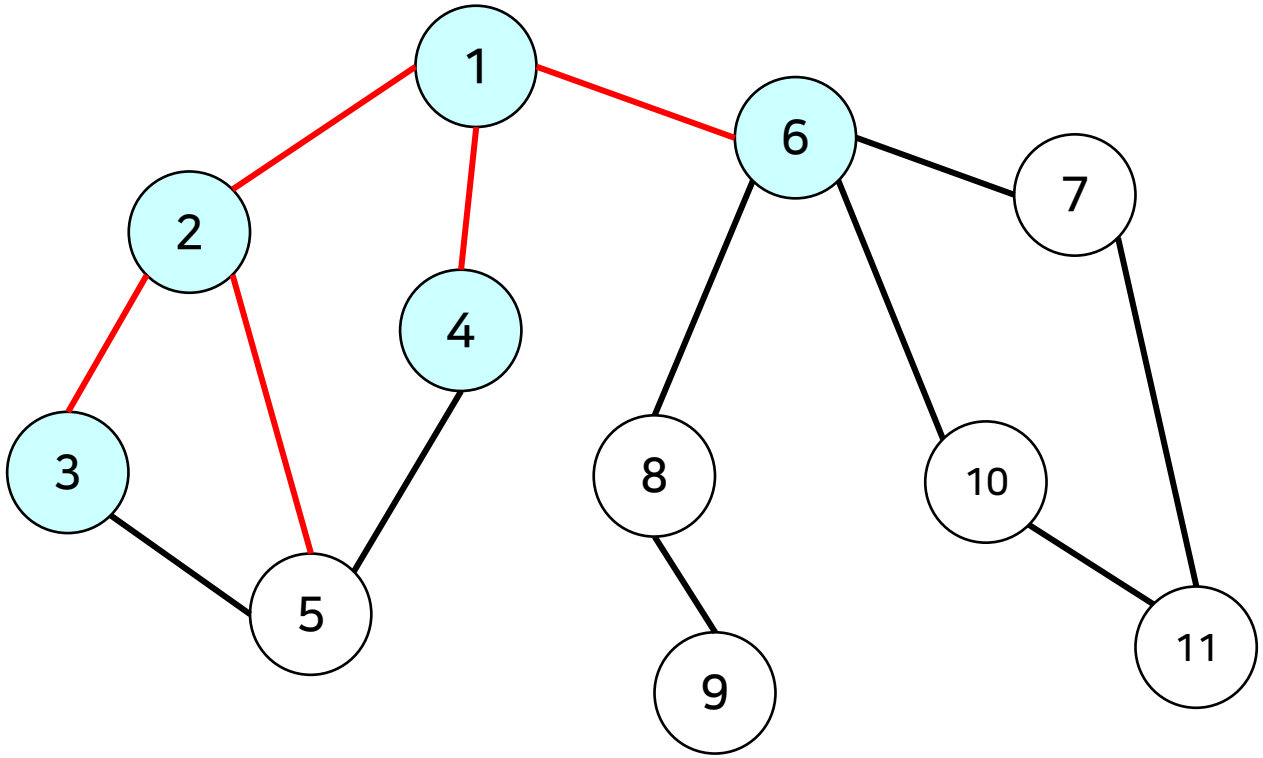
너비 우선 탐색 (BFS)



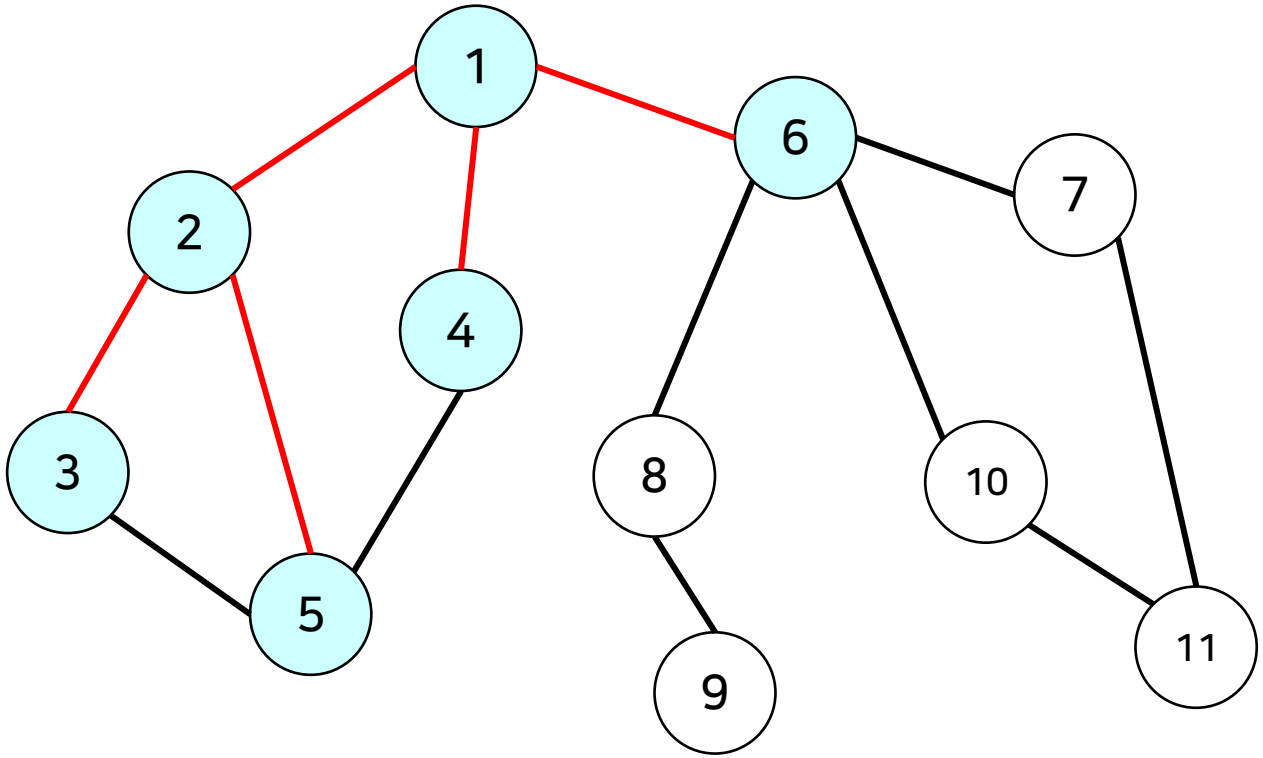
너비 우선 탐색 (BFS)



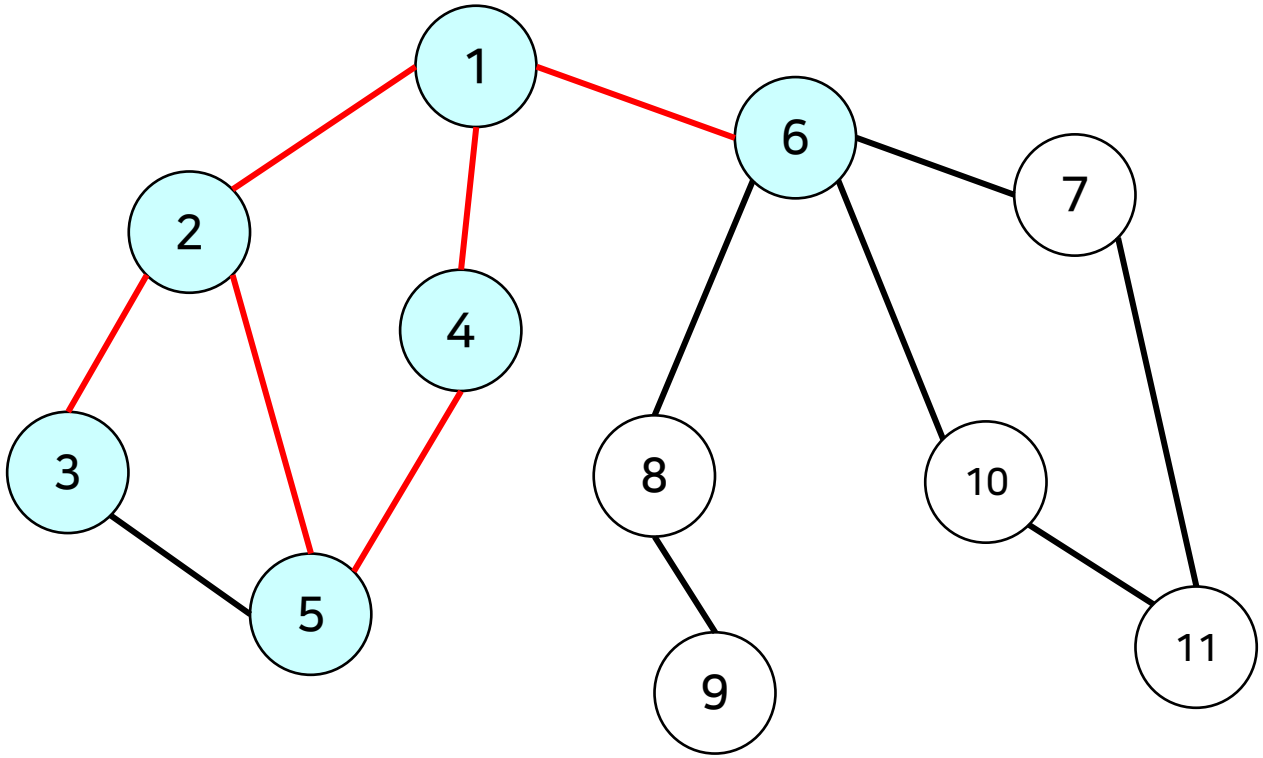
너비 우선 탐색 (BFS)



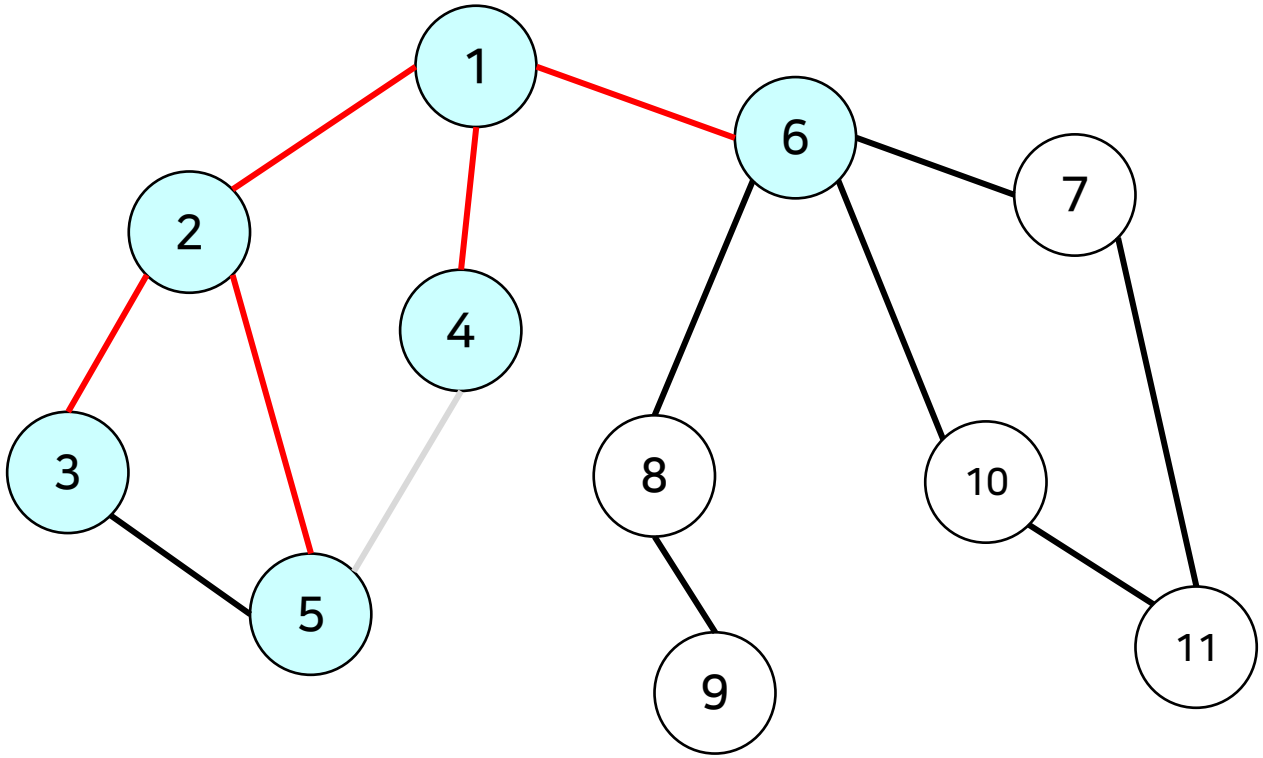
너비 우선 탐색 (BFS)



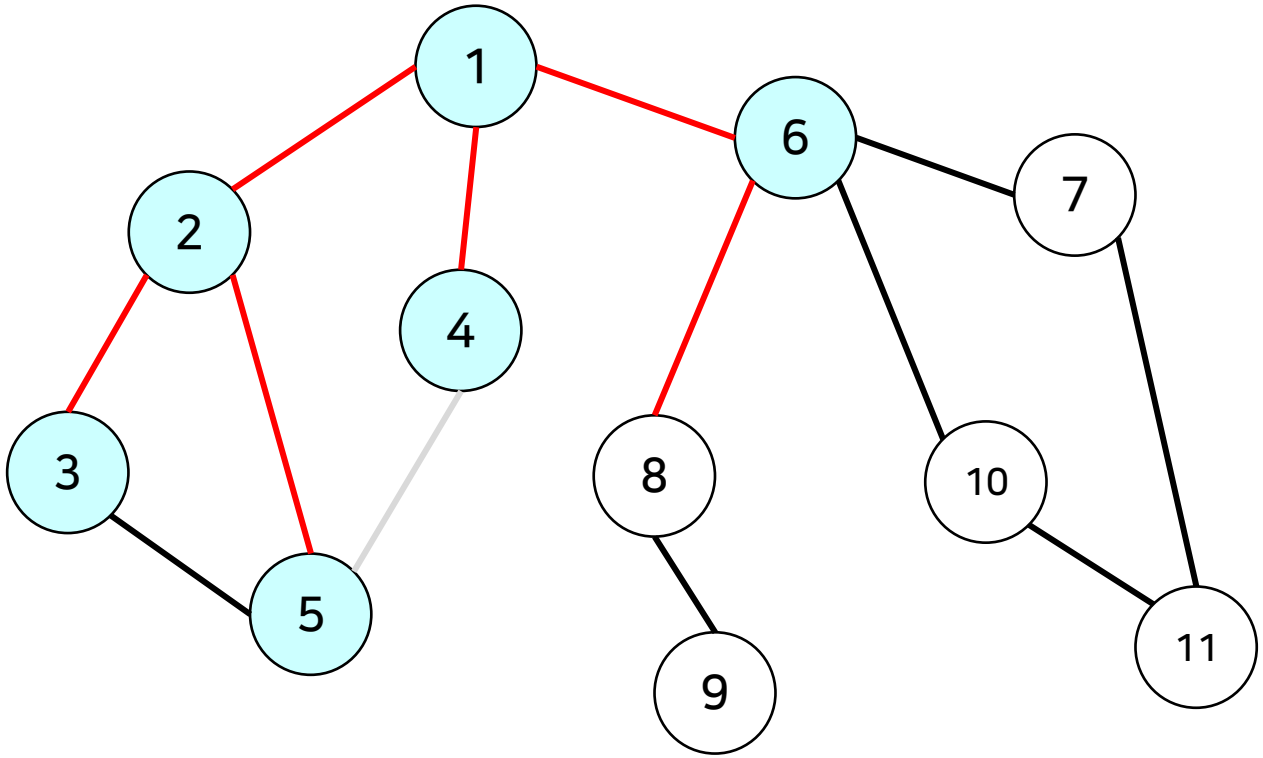
너비 우선 탐색 (BFS)



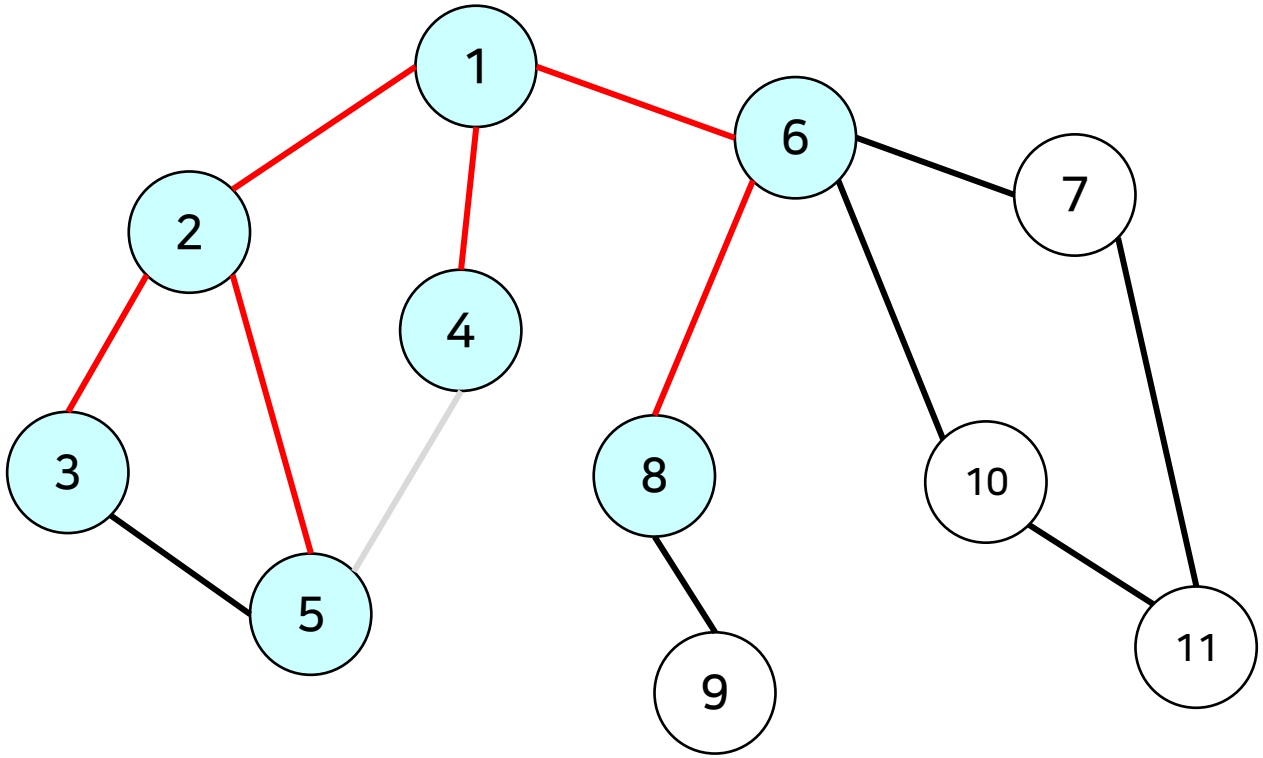
너비 우선 탐색 (BFS)



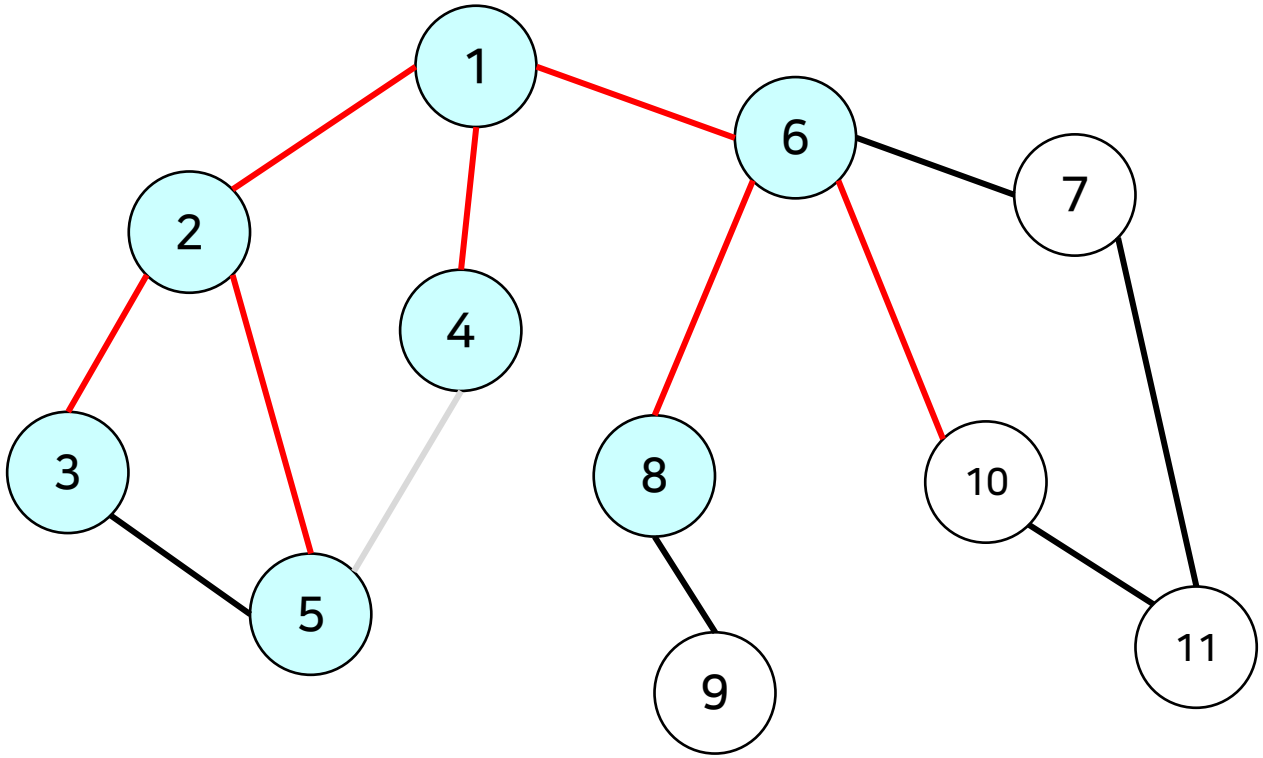
너비 우선 탐색 (BFS)



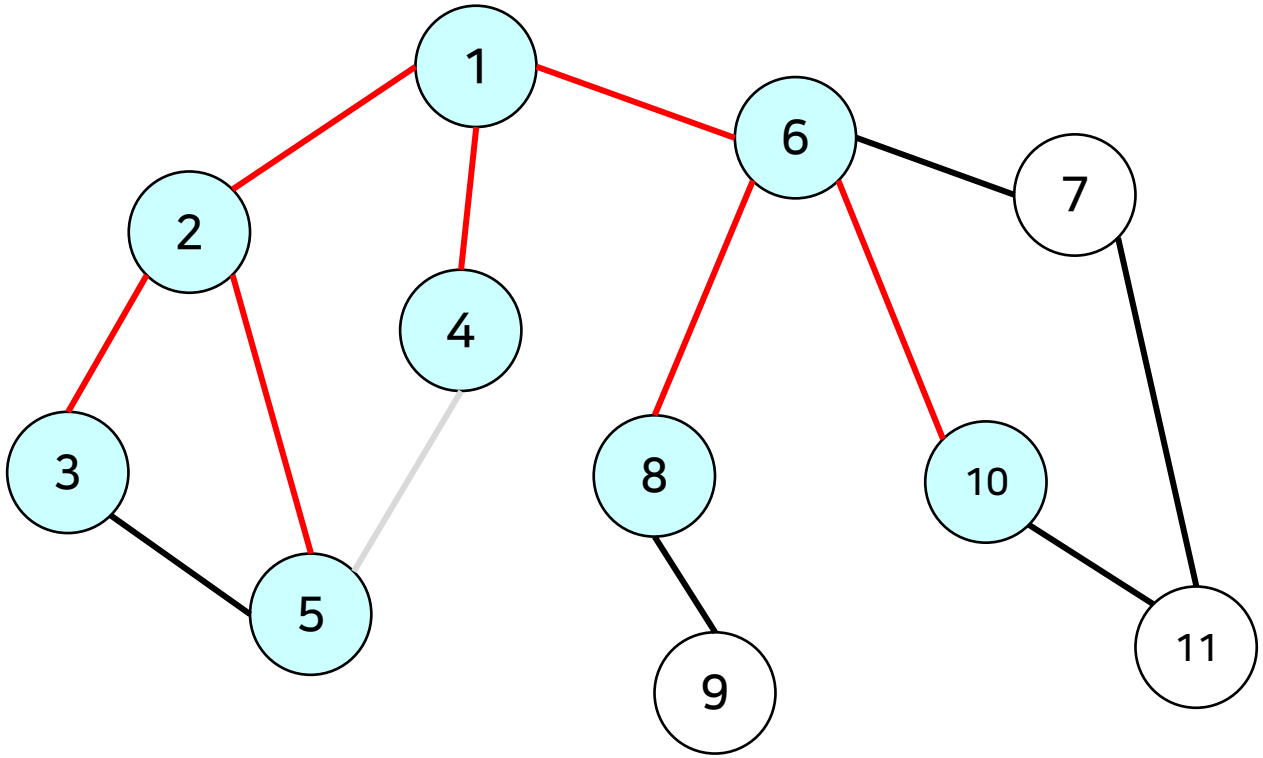
너비 우선 탐색 (BFS)



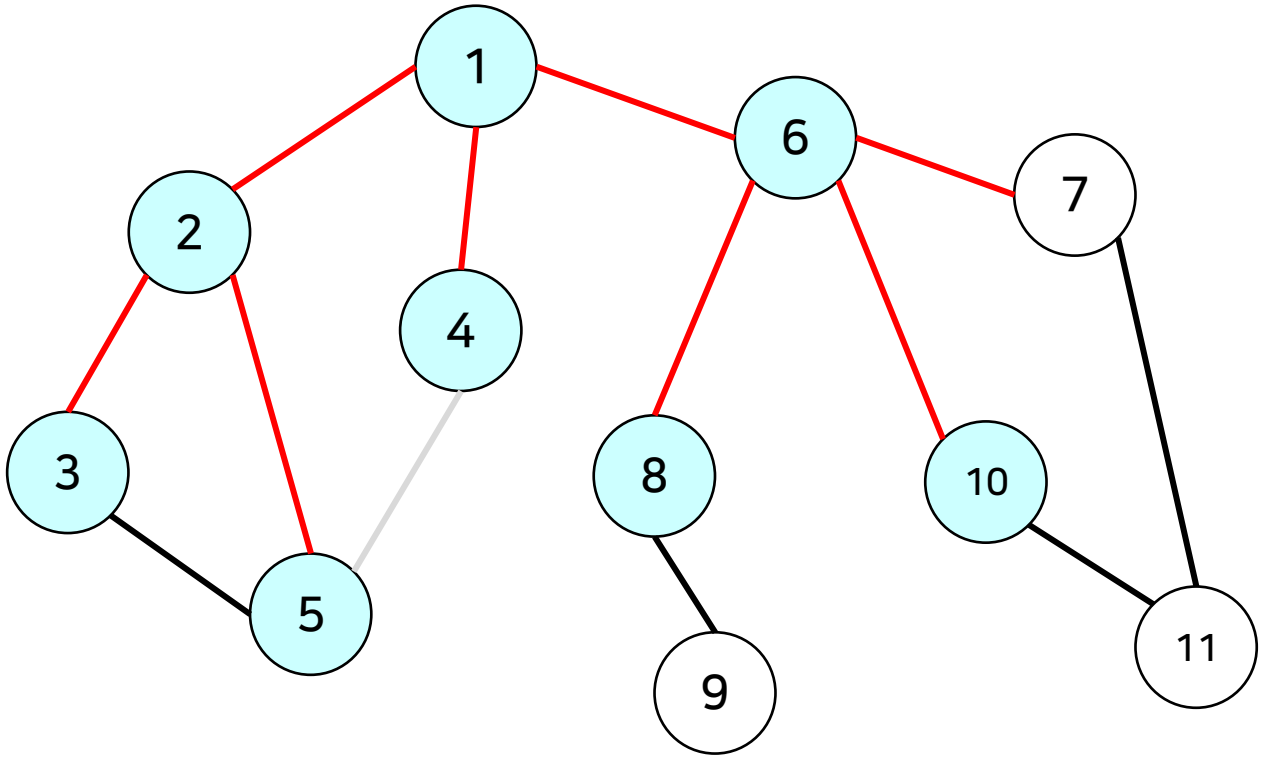
너비 우선 탐색 (BFS)



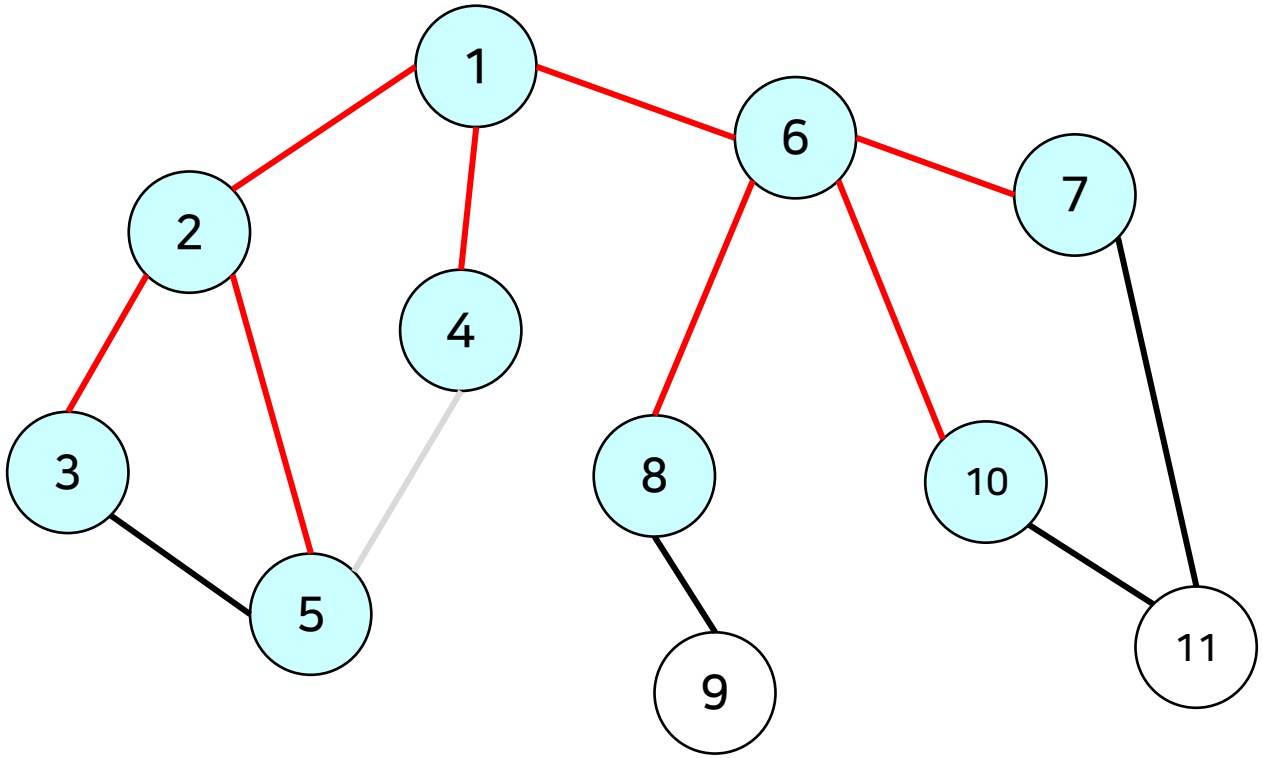
너비 우선 탐색 (BFS)



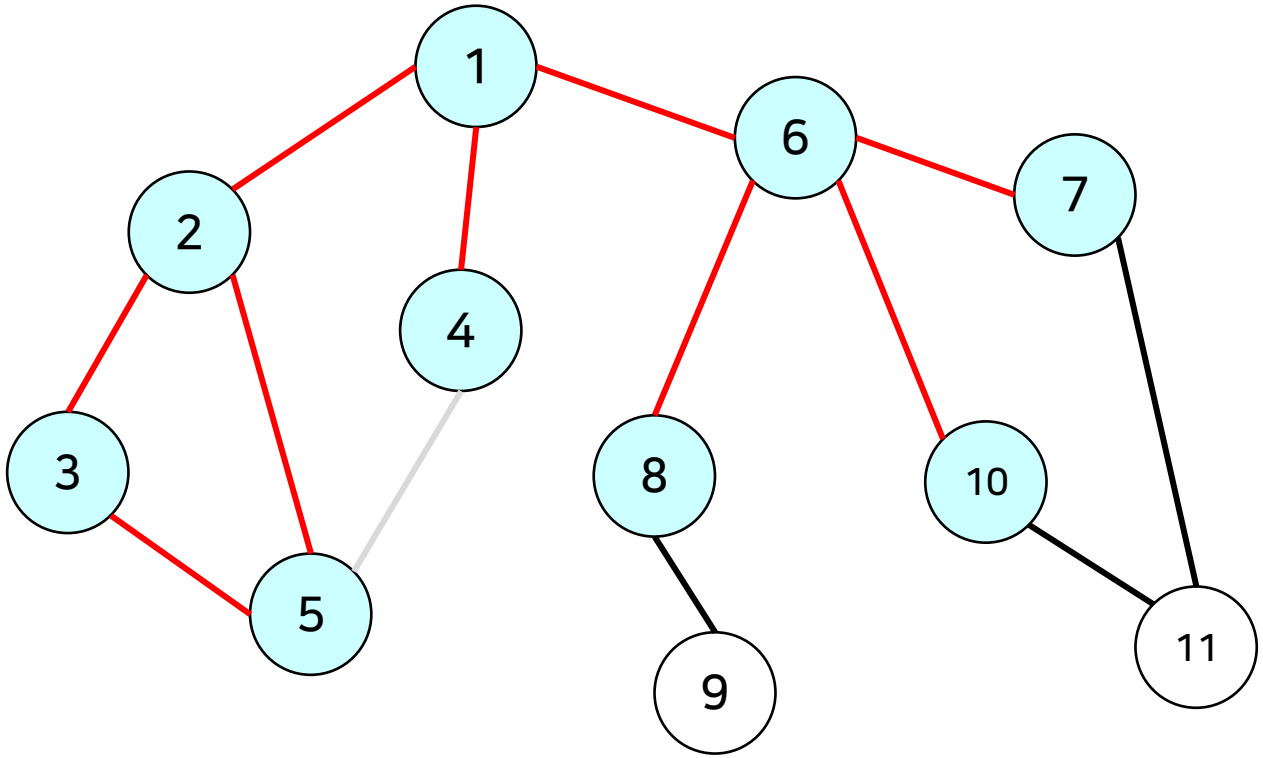
너비 우선 탐색 (BFS)



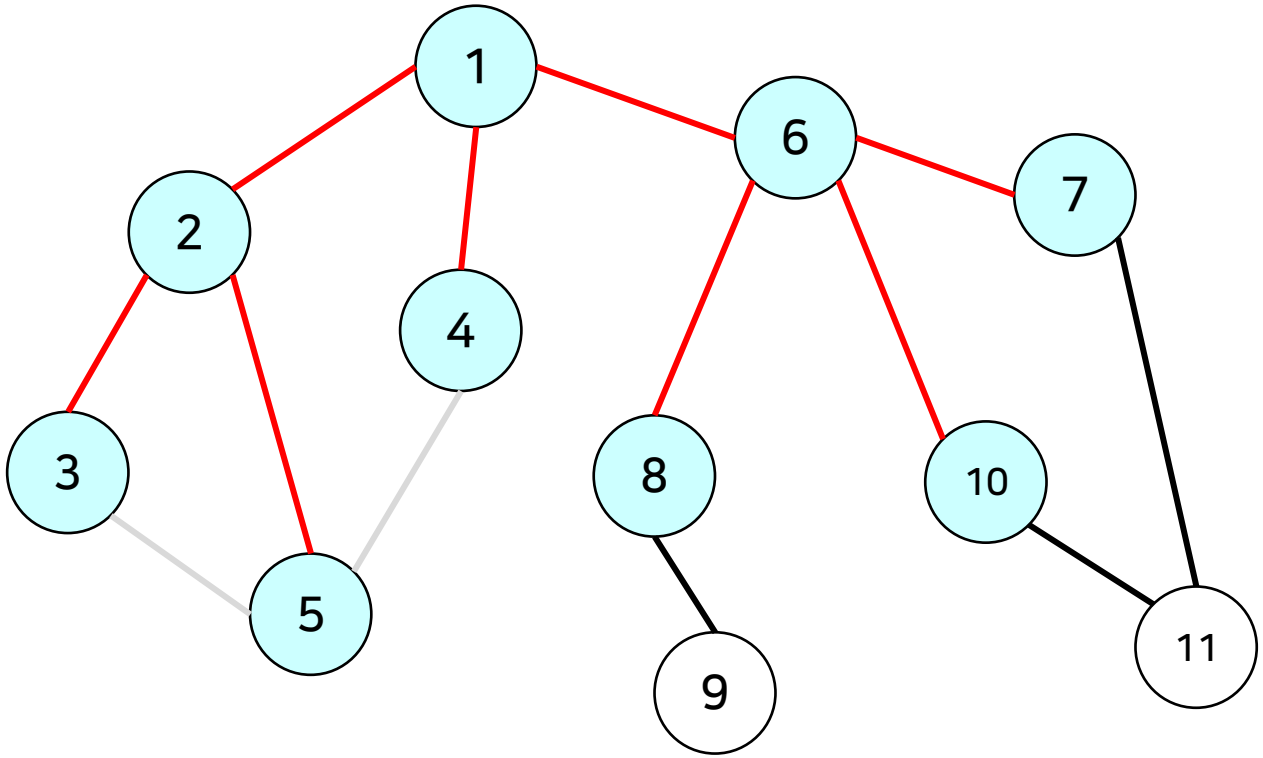
너비 우선 탐색 (BFS)



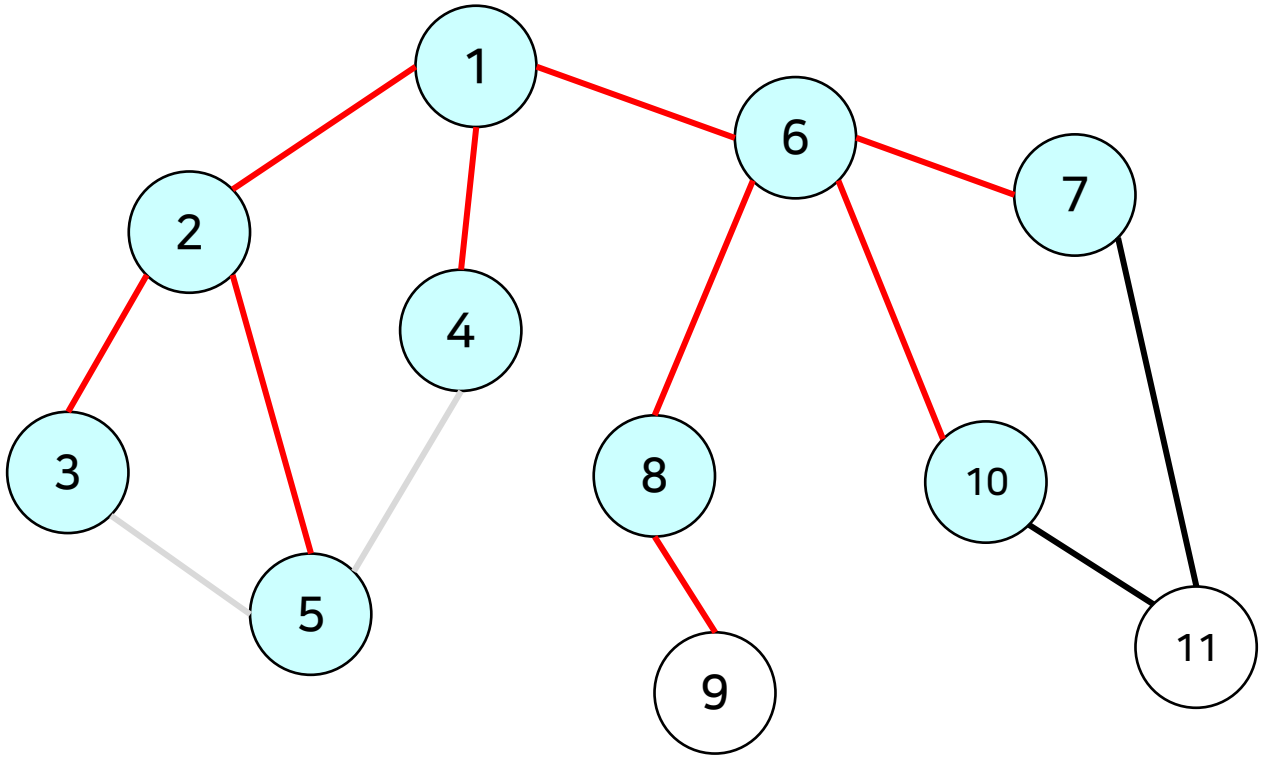
너비 우선 탐색 (BFS)



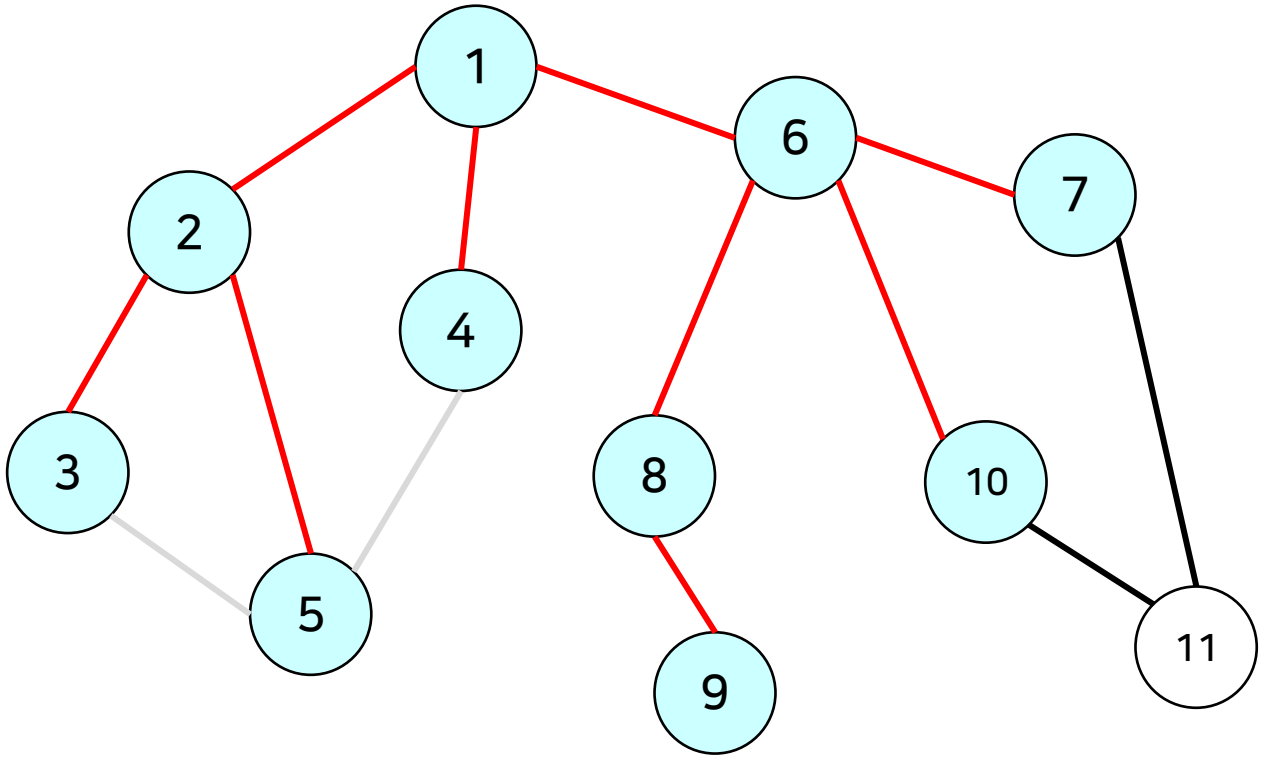
너비 우선 탐색 (BFS)



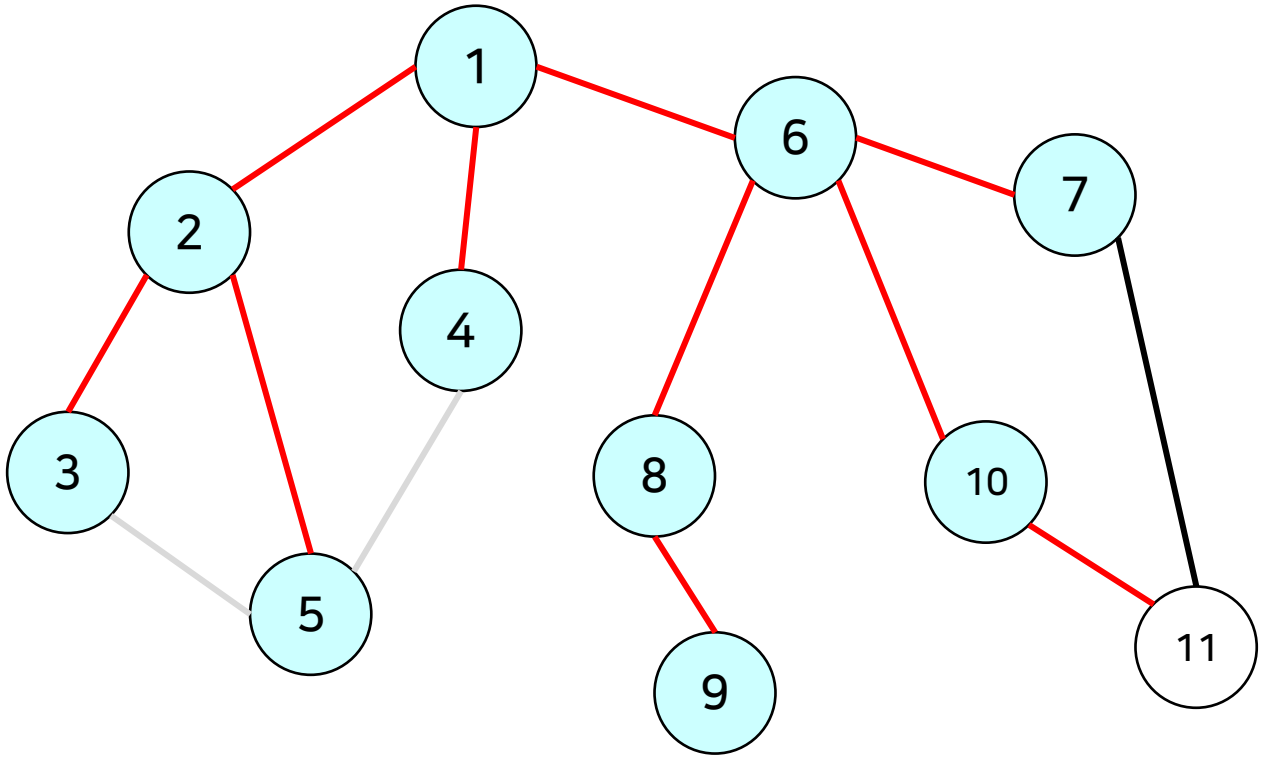
너비 우선 탐색 (BFS)



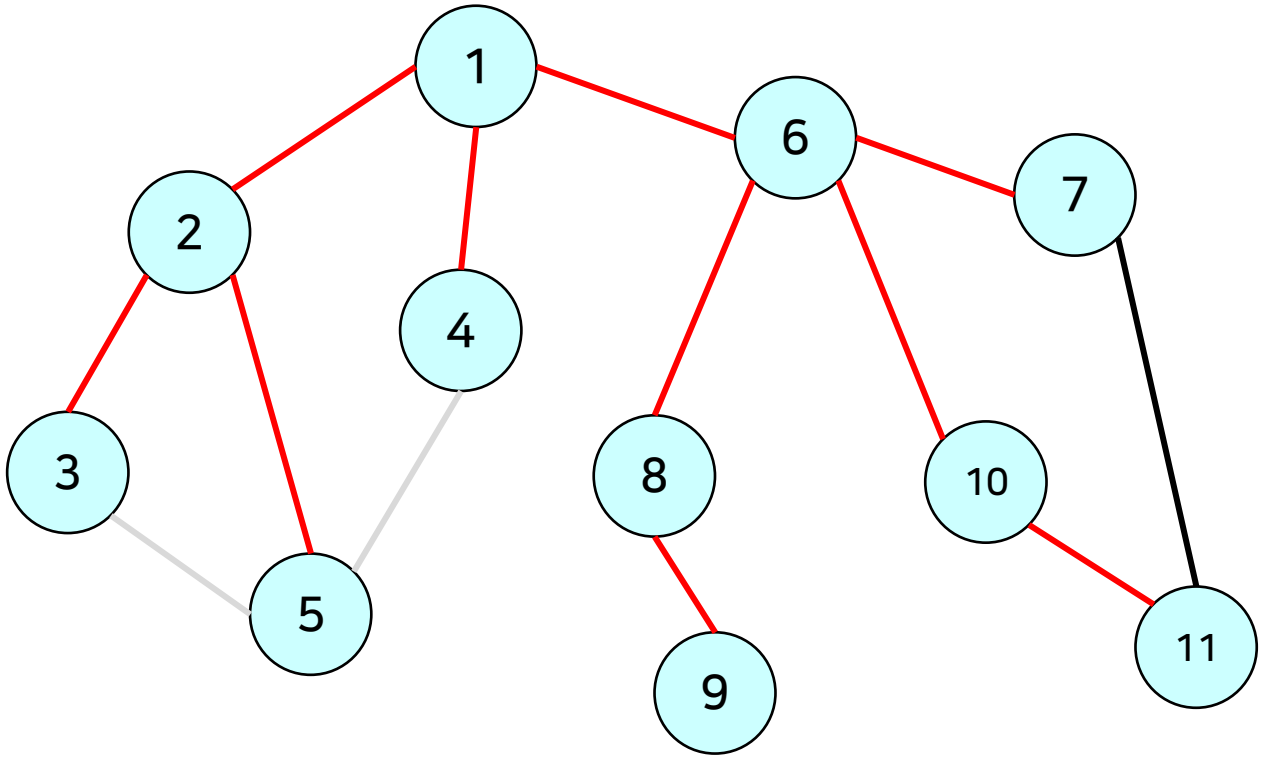
너비 우선 탐색 (BFS)



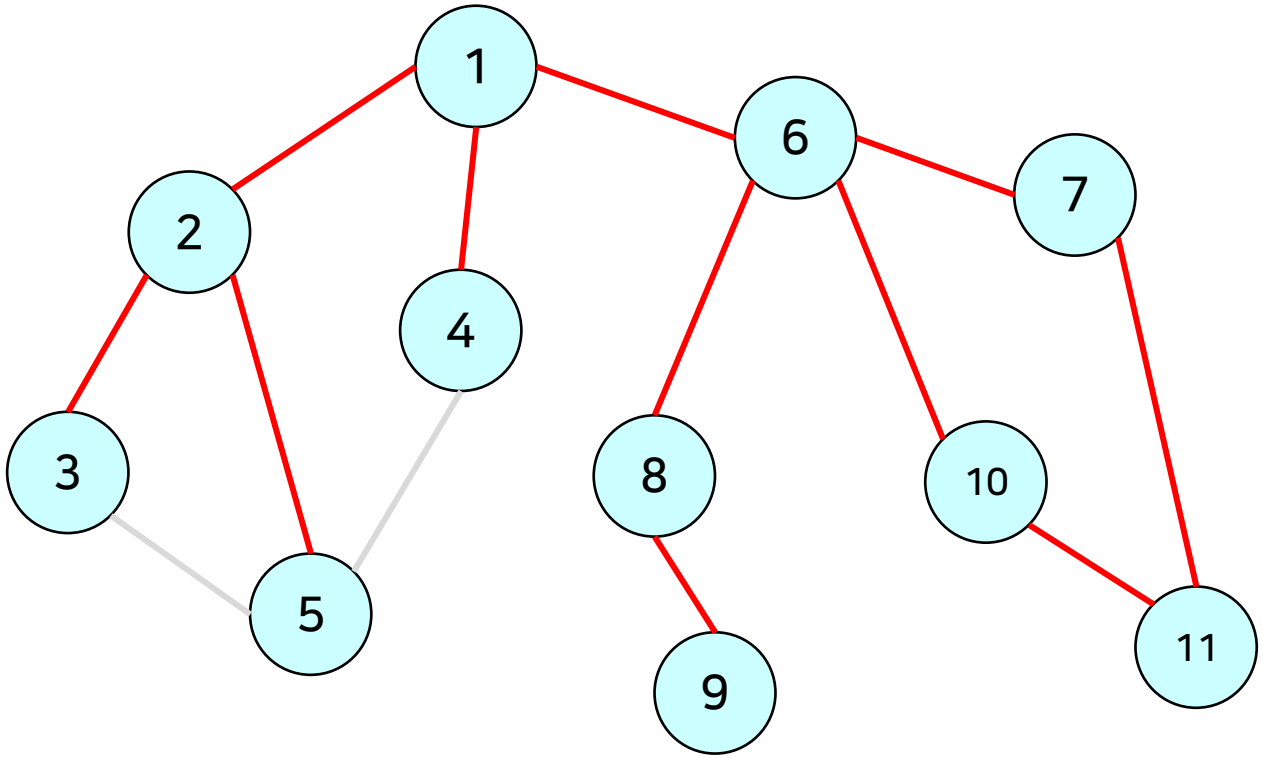
너비 우선 탐색 (BFS)



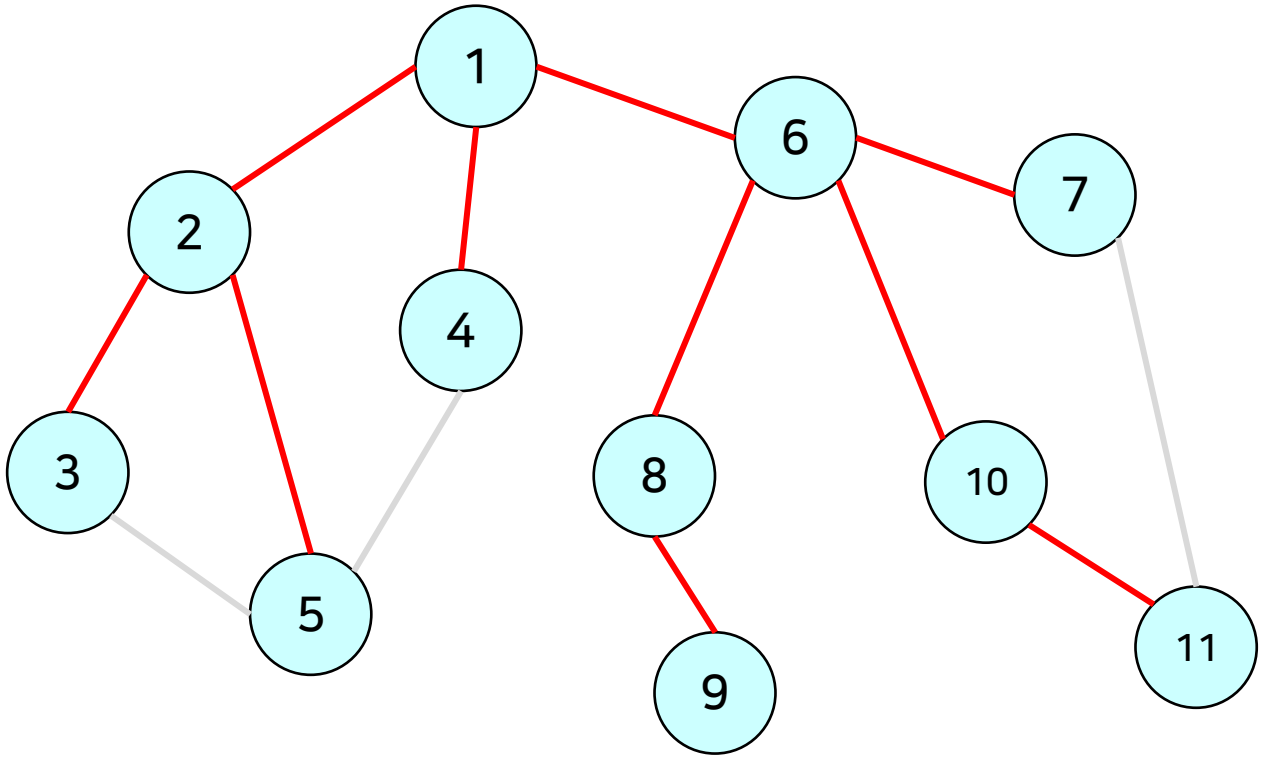
너비 우선 탐색 (BFS)



너비 우선 탐색 (BFS)



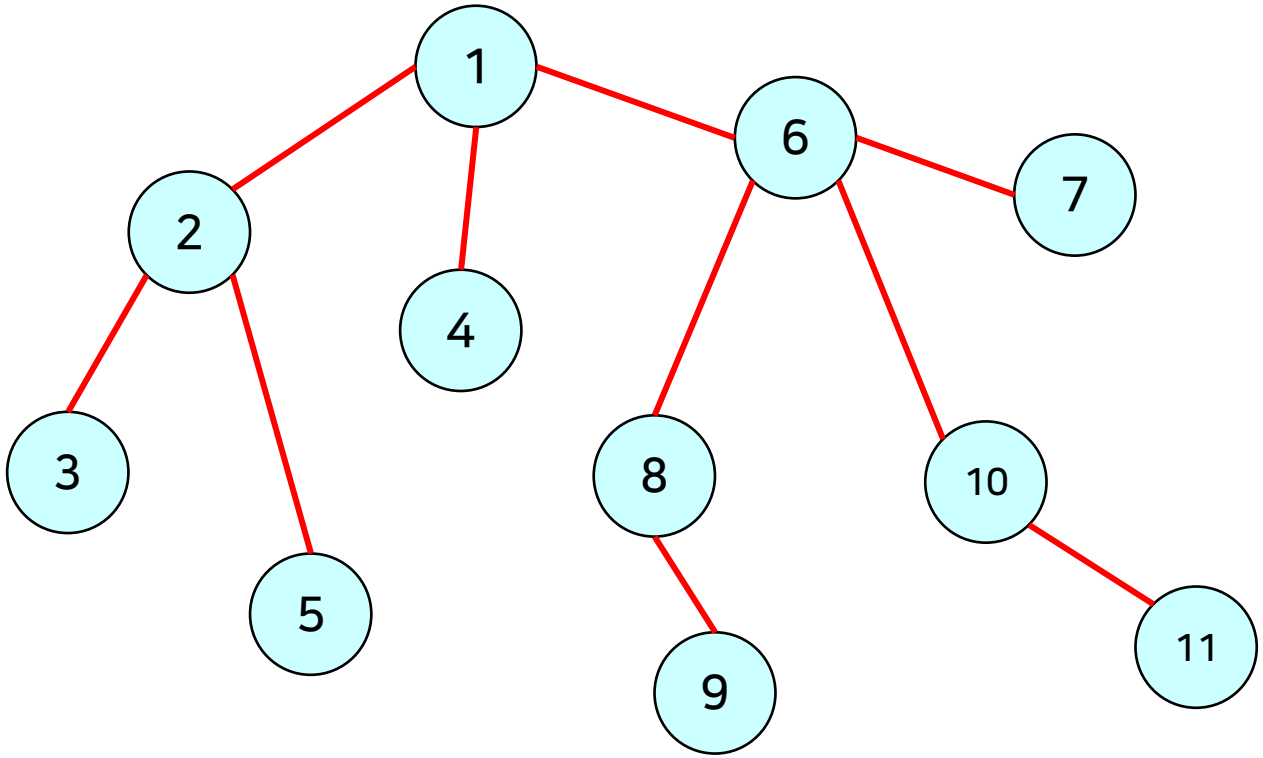
너비 우선 탐색 (BFS)



END

너비 우선 탐색 (BFS)

<BFS Tree>



너비 우선 탐색 (BFS)

```
vector<int> adj[MAX];
bool visited[MAX];

int main(void) {
    queue<int> q;
    q.push(root);
    visited[root] = true;
    while (!q.empty()) {
        int u = q.front(); q.pop();
        for (int v : adj[u]) {
            if (!visited[v]) {
                visited[v] = true;
                q.push(v);
            }
        }
    }
}
```

DFS vs BFS

<공통점>

- 그래프의 모든 정점을 1번씩 방문할 수 있음

- 시간복잡도가 동일

인접 행렬 : $O(N^2)$ 인접 리스트 : $O(N+E)$

DFS vs BFS

<차이점>

[DFS]

- 재귀 호출 or 스택 사용
- 재귀 호출로 인한 시간 소모
- 지나온 경로를 알아야 할 때 주로 사용

[BFS]

- 큐 사용
- 다음에 방문할 정점 저장으로 인한 메모리 소모
- 최단 거리를 구할 때 주로 사용

BOJ 1012 (유기농 배추)

문제

차세대 영농인 한나는 강원도 고랭지에서 유기농 배추를 재배하기로 하였다. 농약을 쓰지 않고 배추를 재배하려면 배추를 해충으로부터 보호하는 것이 중요하기 때문에, 한나는 해충 방지에 효과적인 배추흰지렁이를 구입하기로 결심한다. 이 지렁이는 배추근처에 서식하며 해충을 잡아 먹음으로써 배추를 보호한다. 특히, 어떤 배추에 배추흰지렁이가 한 마리라도 살고 있으면 이 지렁이는 인접한 다른 배추로 이동할 수 있어, 그 배추들 역시 해충으로부터 보호받을 수 있다. 한 배추의 상하좌우 네 방향에 다른 배추가 위치한 경우에 서로 인접해있는 것이다.

한나가 배추를 재배하는 땅은 고르지 못해서 배추를 군데군데 심어 놓았다. 배추들이 모여있는 곳에는 배추흰지렁이가 한 마리만 있으면 되므로 서로 인접해있는 배추들이 몇 군데에 퍼져있는지 조사하면 총 몇 마리의 지렁이가 필요한지 알 수 있다. 예를 들어 배추밭이 아래와 같이 구성되어 있으면 최소 5마리의 배추흰지렁이가 필요하다. 0은 배추가 심어져 있지 않은 땅이고, 1은 배추가 심어져 있는 땅을 나타낸다.

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**

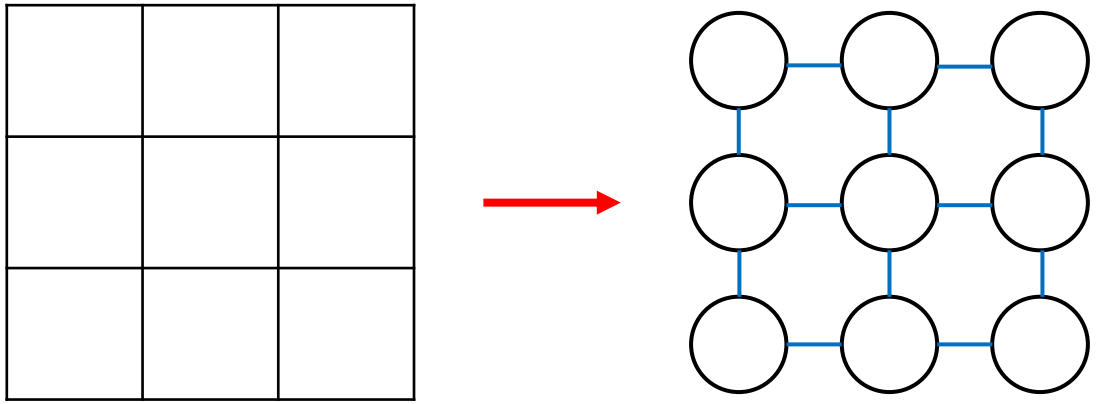
BOJ 1012 (유기농 배추)

✓ 필요한 지렁이의 수 = 붙어 있는 배추의 그룹의 수

✓ 그래프화 시키기

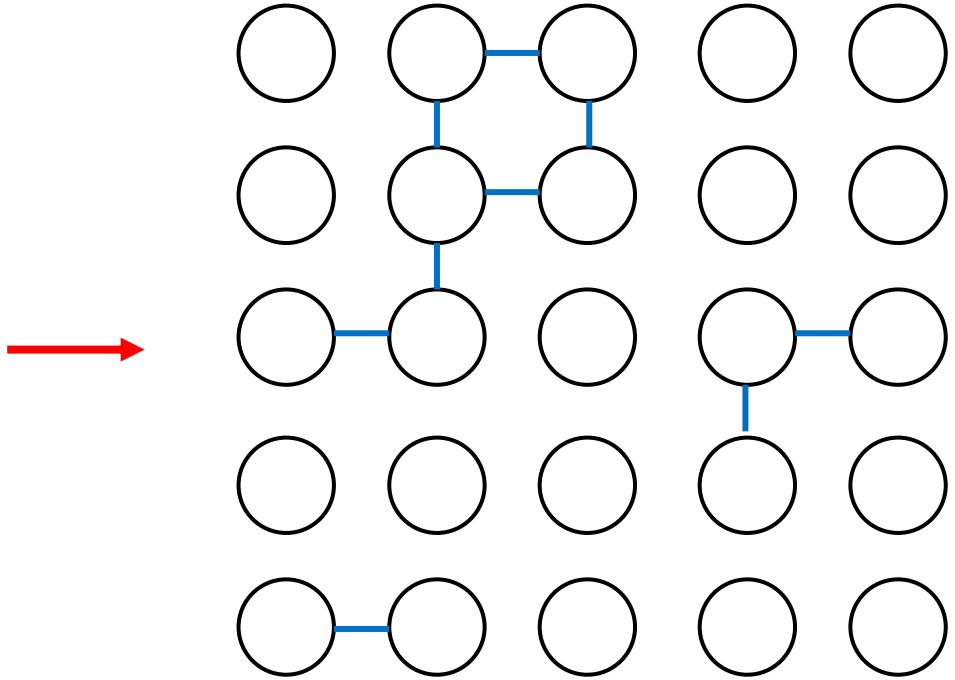
- 정점 : 하나의 격자칸

- 간선 : 인접한 배추로 이동하는 경로



BOJ 1012 (유기농 배추)

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



Flood Fill Algorithm : 어떤 칸과 연결된 영역을 찾는 알고리즘

BOJ 1012 (유기농 배추)

<DFS solution>

```
#include<iostream>
#include<cstring>
using namespace std;
int A[55][55];
int dx[4] = { 0, 1, 0, -1 };
int dy[4] = { 1, 0, -1, 0 };
int ans, m, n, k;
void dfs(int x, int y) {
    A[x][y] = 0;
    for (int i = 0; i < 4; i++) {
        int nx = x + dx[i];
        int ny = y + dy[i];
        if (nx >= 0 && nx < n && ny >= 0 && ny < m) {
            if(A[nx][ny] == 1) dfs(nx, ny);
        }
    }
}
```

```
int main(void) {
    int T; cin >> T;
    while (T--) {
        ans = 0;
        memset(A, 0, sizeof(A));
        cin >> m >> n >> k;
        for (int i = 0; i < k; i++) {
            int y, x; cin >> y >> x;
            A[x][y] = 1;
        }
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                if (A[i][j] == 1) {
                    ans++;
                    dfs(i, j);
                }
            }
        }
        cout << ans << '\n';
    }
}
```

BOJ 1012 (유기농 배추)

<BFS solution>

```
void bfs(int X, int Y) {
    queue<pair<int, int>> q;
    q.push({ X, Y });
    A[X][Y] = 0;
    while (!q.empty()) {
        int x = q.front().first;
        int y = q.front().second;
        // auto [x, y] = q.front();
        q.pop();
        for (int i = 0; i < 4; i++) {
            int nx = x + dx[i];
            int ny = y + dy[i];
            if (nx >= 0 && nx < n && ny >= 0 && ny < m) {
                if (A[nx][ny] == 1) {
                    A[nx][ny] = 0;
                    q.push({ nx, ny });
                }
            }
        }
    }
}
```

BOJ 2206 (벽 부수고 이동하기)

문제

N×M의 행렬로 표현되는 맵이 있다. 맵에서 0은 이동할 수 있는 곳을 나타내고, 1은 이동할 수 없는 벽이 있는 곳을 나타낸다. 당신은 (1, 1)에서 (N, M)의 위치까지 이동하려 하는데, 이때 최단 경로로 이동하려 한다. 최단경로는 맵에서 가장 적은 개수의 칸을 지나는 경로를 말하는데, 이때 시작하는 칸과 끝나는 칸도 포함해서 센다.

만약에 이동하는 도중에 한 개의 벽을 부수고 이동하는 것이 좀 더 경로가 짧아진다면, 벽을 한 개 까지 부수고 이동하여도 된다.

한 칸에서 이동할 수 있는 칸은 상하좌우로 인접한 칸이다.

맵이 주어졌을 때, 최단 경로를 구해 내는 프로그램을 작성하시오.

예제 입력 1 복사

```
6 4
0100
1110
1000
0000
0111
0000
```

예제 출력 1 복사

```
15
```

BOJ 2206 (벽 부수고 이동하기)

- ✓ 격자판을 그래프화
- ✓ (1, 1) → (N, M)으로 간선을 통해 이동하는 경로 중 최단 거리 → BFS
- ✓ 최단 거리는 어떻게 구하는가?

<Before> (단순 방문 체크)

bool visited[1001][1001]

<After> (최단 시간)

int dist[1001][1001]

BOJ 2206 (벽 부수고 이동하기)

- ✓ 벽을 한 번 부술 수 있는 기회 존재
- ✓ 지금까지 벽을 부순 적이 있는가? True or False → 0 or 1
- ✓ dist[1001][1001][2]

BOJ 1697 (숨바꼭질)

문제

수빈이는 동생과 숨바꼭질을 하고 있다. 수빈이는 현재 점 N($0 \le N \le 100,000$)에 있고, 동생은 점 K($0 \le K \le 100,000$)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.

입력

첫 번째 줄에 수빈이가 있는 위치 N과 동생이 있는 위치 K가 주어진다. N과 K는 정수이다.

BOJ 1697 (숨바꼭질)

- ✓ Naive 풀이
  - K에 도달하는 모든 경우 중 최소 시간
  - 4 → 3 → 2 → 4 → … (Cycle)
- ✓ 매 초마다 K에 도착하는 경우가 있는지 확인해보자!

BOJ 1697 (숨바꼭질)

- ✓ 그래프화 시키기

  - 정점 : 각 위치

  - 간선 : X → X+1 , X → X-1 , X → X*2

- ✓ 최단거리 → BFS

- ✓ 시작점 = N / 도착점 = K

필수 / 연습문제

[필수문제]

[BOJ 1987] 알파벳

[BOJ 2210] 숫자판 점프

[BOJ 1697] 숨바꼭질

[BOJ 14940] 쉬운 최단거리

[BOJ 22352] 항체 인식

[BOJ 20924] 트리의 기둥과 가지

[연습문제]

[BOJ 7576] 토마토

[BOJ 5859] 거짓말쟁이

[BOJ 16928] 뱀과 사다리 게임

[BOJ 2206] 벽 부수고 이동하기

[BOJ 7562] 나이트의 이동

[BOJ 14466] 소가 길을 건너간 이유 6

[BOJ 21738] 얼음깨기 펭귄