09. BFS, DFS

09. 너비 우선 탐색, 깊이 우선 탐색 (숭실대학교 박찬솔)

목차

그래프 순회

- 그래프의 각 정점을 방문하는 절차

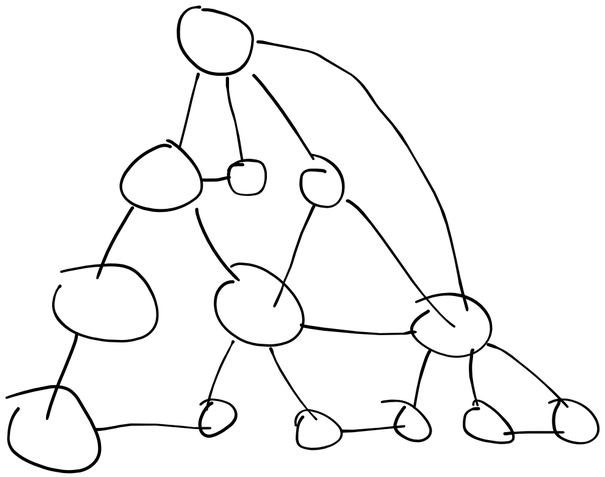
- 그래프 순회의 종류는 정점을 방문하는 순서에 따라 분류됨.

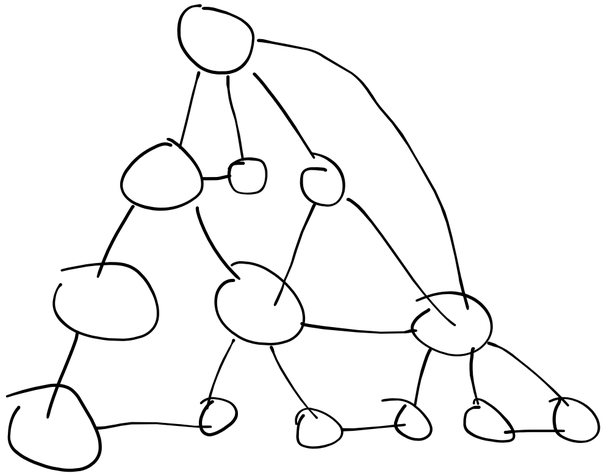
- 여러 번 방문하는 것을 방지하기 위해서 정점의 방문 여부를 관리하기도 함.
  - 일부 특별한 그래프에서는 명시적으로 정점의 방문 여부를 저장하지 않기도 함.
  - 이 경우에는 그래프의 구조가 정점을 중복으로 방문하지 않는 것을 보장함.

BFS, DFS

- **너비 우선 탐색, BFS** Breadth First Search
  - 한 정점과 인접한 정점들을 **모두 방문**하여 그래프를 순회

- **깊이 우선 탐색, DFS** Depth First Search
  - 한 정점에서 **깊이가 깊어지는 방향**으로 정점을 방문하여 그래프를 순회
  - 더 이상 나아갈 정점이 없다면 이전 정점으로 돌아옴





질문?

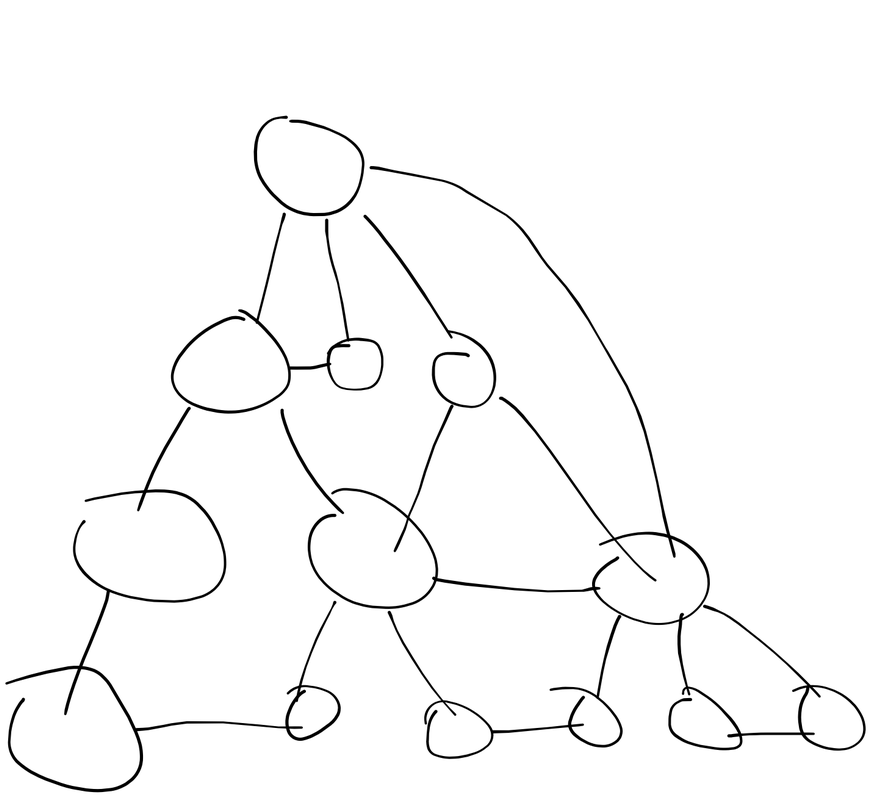
BFS

- 한 정점과 인접한 정점이 여러 개라면 어떤 정점을 우선해서 방문할까?
  - 따로 정해진 게 없다.
  - 아무 순서로 방문하면 됨.

- **Queue를 사용함.**

- 시작 정점을 Q에 넣음.
- Q가 빌 때까지 다음 과정을 반복
  - Q에서 정점 하나 u를 제거함.
  - u와 인접한 모든 정점 v에 대하여, v가 이전에 방문하지 않은 정점이면 Q를 v에 넣음.

BFS



BFS

```cpp
int N; // 정점 수

vector<vector<int>> graph(N); // 인접 그래프
vector<int> vis(N); // 정점 방문 여부
queue<int> Q; // 큐

Q.push(1);
vis[1] = 1;

while (!Q.empty()) {
    int u = Q.front(); Q.pop();
    vis[u] = 1;
    cout << u << " ";
    for (int v : graph[u]) {
        if (!vis[v]) Q.push(v);
    }
}
```

질문?

DFS

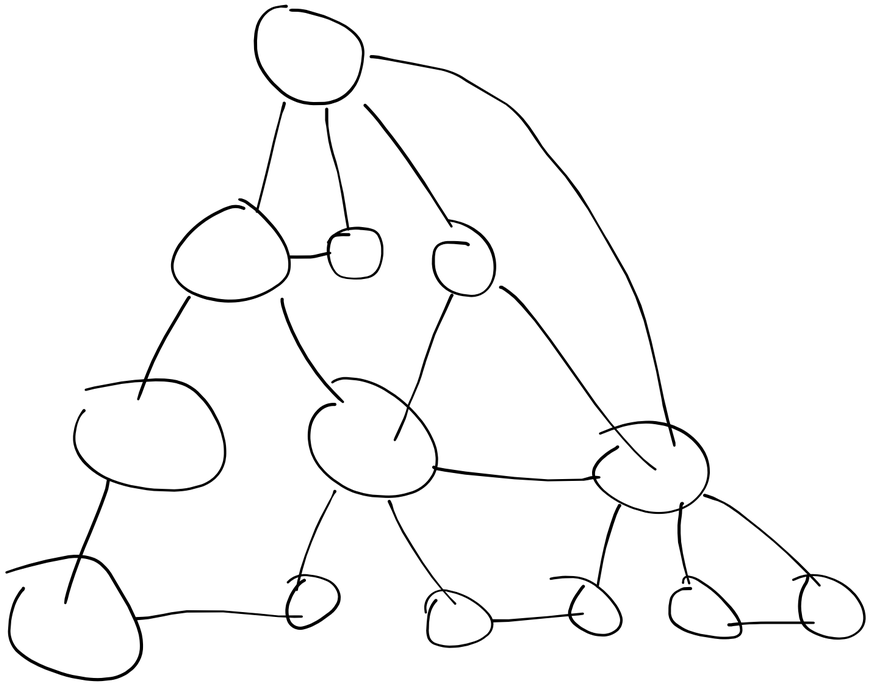
- 한 정점과 인접한 정점이 여러 개라면 어떤 정점을 우선해서 방문할까?
  - 따로 정해진 게 없다.
  - 아무 순서로 방문하면 됨.

- **Stack을 사용함.**
  - 구현의 편의를 위해 보통 재귀 함수를 사용함.

- **재귀 함수** F(u)를 정의하자. (단, u는 정점)
  - u와 인접한 정점 v에 대하여, v를 방문하지 않았으면 F(v)를 호출한다.

- **F(시작 정점)**을 호출한다.

DFS



Stack:

DFS

```cpp
int N; // 정점 수
vector<vector<int>> graph(N); // 인접 리스트
vector<int> vis(N); // 방문 여부

void dfs(int u) {
    vis[u] = 1;
    cout << u << " ";
    for (int v : graph[u]) if (!vis[v]) dfs(v);
}


...

dfs(1);
```

BFS/DFS – 시간 복잡도

- 시간 복잡도 : O(V + E)
  - 정점을 O(V)번 방문함
  - 총 O(E)개의 간선을 거침 **handshaking lemma**

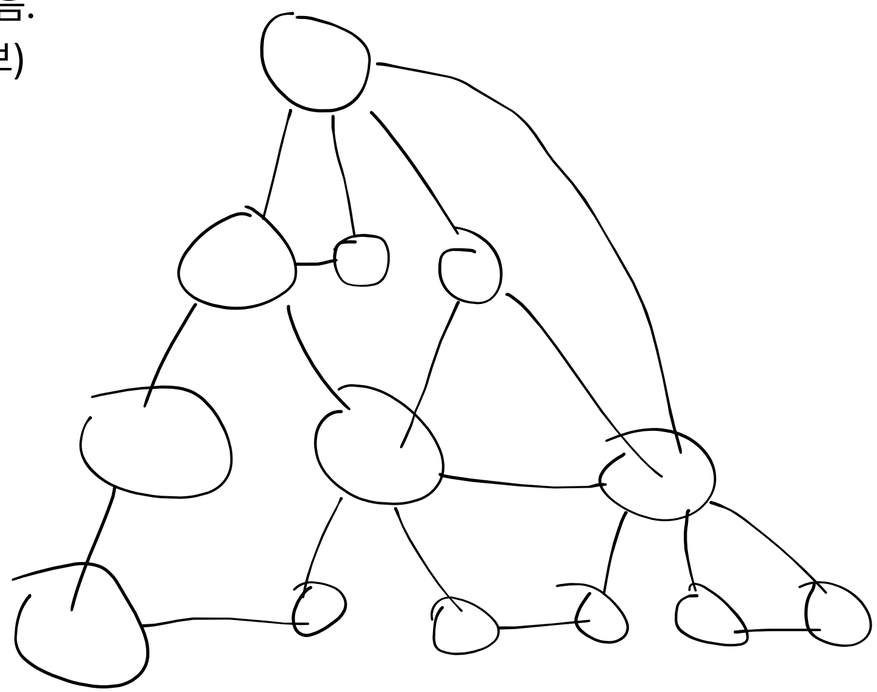
- 공간 복잡도 : O(V + E)
  - 인접 리스트 : O(V + E)
  - 방문 여부 배열 : O(V)

질문?

BFS – Shortest Path

- 모든 간선의 가중치가 같은 그래프에서 BFS를 사용하여 최단 경로 그래프를 구할 수 있다.

- 단순히 시작 정점에서 BFS를 돌려주는 것으로 모든 정점과의 최단 경로를 구할 수 있음.
- 여기서 관리할 정보는 현재 정점이 시작 정점과 얼마나 멀리 떨어져 있는가. (거리 정보)
  - 다음 정점으로 넘어갈 때, **현재 정점 거리 + 1**이 다음 정점의 최단 경로 거리가 된다.



DFS – Shortest Path?

- DFS도 최단 경로를 구하는 데 사용할 수 없을까?
  - 물론, 예시의 DFS 코드를 조금만 수정하면 DFS로도 최단 경로를 구할 수 있다.

- DFS는 정점 하나를 잡고 간선을 따라서 계속 이동한다.
- 최단 경로를 갱신하기 위하여 이미 방문한 정점을 다시 방문하여 처리해야 할 수도 있다.

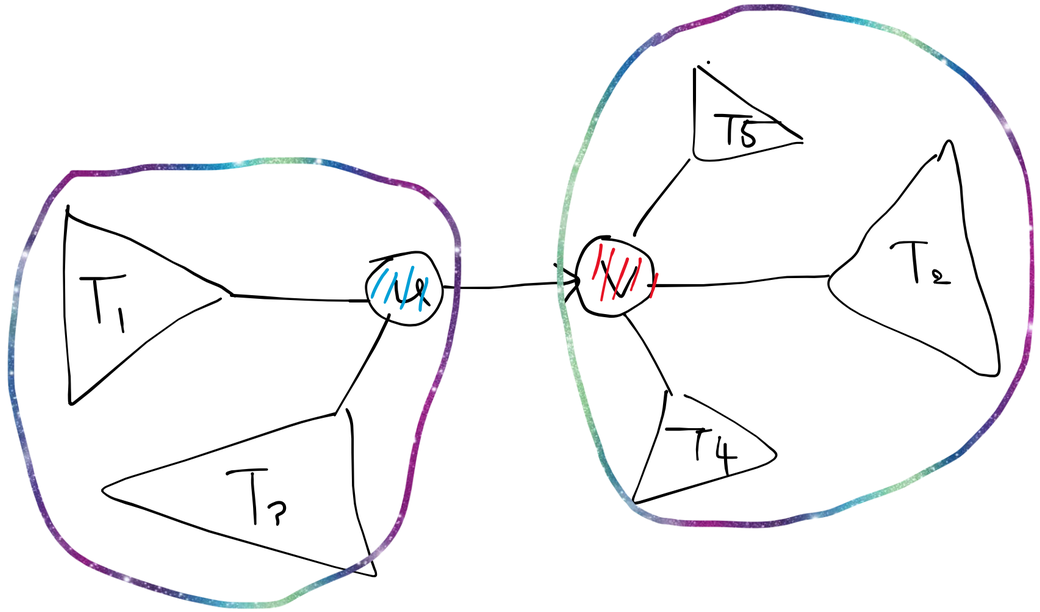
- $O(V^2)$ 시간이 걸릴 수 있음.
  - BFS보다 비효율적이라서 일반적으로 사용하지 않음.

질문?

DFS – Tree

- 트리에서 DFS 순회를 하면 방문 배열을 관리할 필요가 없다.
  - 이전에 방문한 정점이 무엇인지만 알면 된다.

- 현재 정점 v에 있고, 이전에 방문한 정점이 u라고 하자.
  - v에서는 u로 돌아가지만 않으면 u 쪽에 있는 정점으로 다시 돌아가는 길은 없다.



DFS – Tree

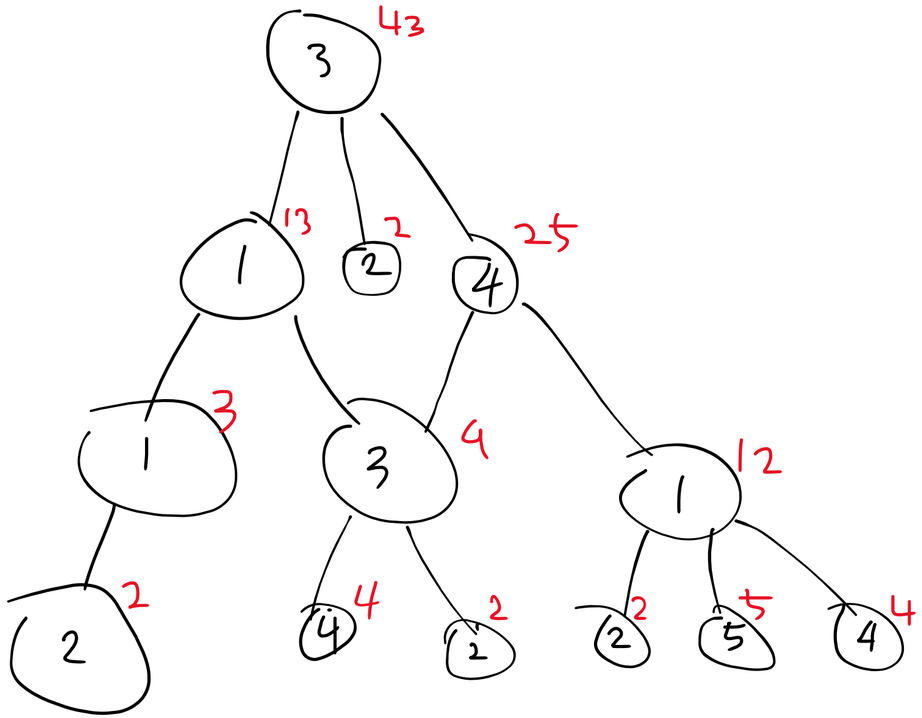
```cpp
int N;
vector<vector<int>> graph(N);
void dfs(int u, int prv) {
    cout << u << " ";
    for (int v : graph[u]) if (prv != u) dfs(v, u);
}

...

dfs(1, 1);
```

질문?

DFS – 정보 관리하기

- 모든 서브 트리에 대하여 트리에 있는 각 정점에 적힌 수의 합을 구해보는 문제

- 예)



DFS – 정보 관리하기

- int dfs(int u);

- 정점 u를 루트로 하는 서브 트리를 순회한다. 이때, 반환 값은 서브 트리의 모든 정점에 적힌 수의 합이다.

- 점화식:
- dfs(u) = cost[u] + sum(For all child v of u, dfs(v))

DFS – 정보 관리하기


```cpp
int N;
vector<int> point(N);
vector<vector<int>> graph(N);

int dfs(int u, int prv) {
    int ret = point[u];
    for (int v : graph[u]) if (prv != u) ret += dfs(v, u);
    cout << u << " " << ret << "\n";
    return ret;
}

...

dfs(1, 1);
```

질문?

DFS/BFS – 컴포넌트의 개수 세기

- BOJ 2667 (단지번호붙이기)
- 붙어 있는(상하좌우로 인접한) 단지의 개수를 세는 문제
- 오른쪽 그림에서 답은 3

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

<그림 1>

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

<그림 2>

DFS/BFS – 컴포넌트의 개수 세기

- 상하좌우로 인접한 그래프 탐색하기
- (x, y)에서 (x – 1, y), (x + 1, y), (x, y – 1), (x, y + 1)로 이동하면 된다.

- DFS를 한번 돌리면 상하좌우로 인접한 것들은 모두 방문 처리됨.
- 따라서, 제일 바깥에서 DFS를 호출한 횟수가 컴포넌트의 개수가 됨

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

<그림 1>

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

<그림 2>

DFS/BFS – 컴포넌트의 개수 세기


```cpp
#include <bits/stdc++.h>

using namespace std;
int vec[26][26], n;

int dfs(int r, int c) {
    vec[r][c] = 0;
    int ret = 1;
    for (const auto &[dx, dy]: vector<pair<int, int>>{
        {-1, 0},
        {1,  0},
        {0,  -1},
        {0,  1},
    }) {
        int x = r + dx, y = c + dy;
        if (x < 1 || y < 1 || x > n || y > n) continue;
        if (vec[x][y]) {
            ret += dfs(x, y);
        }
    }
    return ret;
}
```

```cpp
int main() {
    cin >> n;

    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++) {
            char c; cin >> c;
            vec[i][j] = c == '1';
        }

    vector<int> ans;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            if (vec[i][j]) {
                ans.push_back(dfs(i, j));
            }

    sort(ans.begin(), ans.end());
    cout << ans.size() << "\n";
    for (int i : ans) cout << i << "\n";
}
```

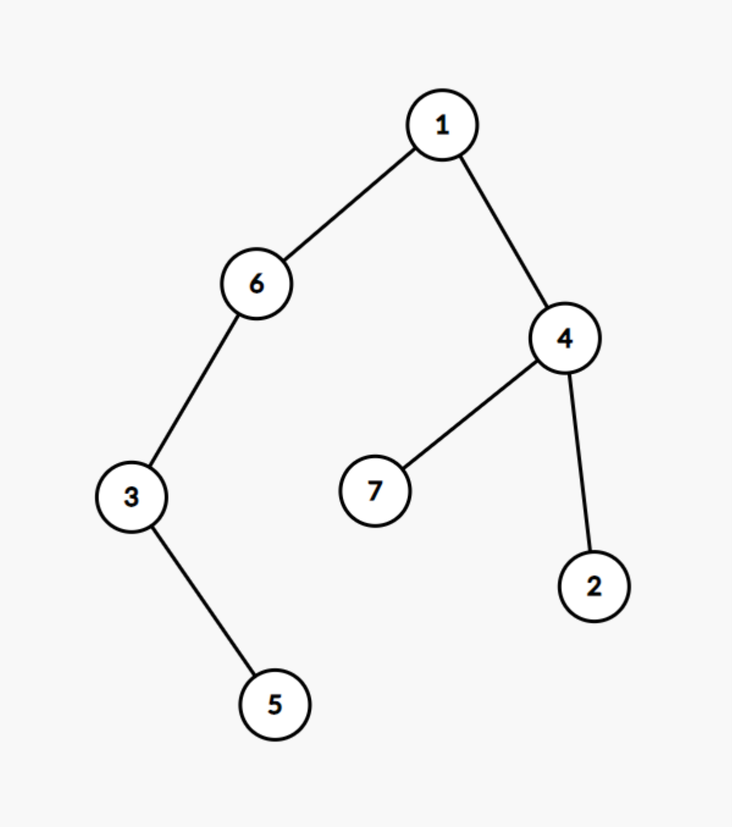
질문?

DFS – 각 노드의 부모 찾기

- BOJ 11725 (트리의 부모 찾기)

- 트리에서 dfs 돌리는 것을 그대로 해주면 됨.

- int dfs(v, u);
- 현재 정점이 v이고, 이전 정점이 u일 때의 정의를 그대로 사용하면 v의 부모 노드는 u임
- 별도의 배열을 만들어서 답을 저장해주면 됨.



DFS – 각 노드의 부모 찾기


```cpp
#include <bits/stdc++.h>

using namespace std;

int n, ans[100001];
vector<int> graph[100001];
void dfs(int u, int prv) {
    for (int v : graph[u]) {
        if (v != prv) {
            ans[v] = u;
            dfs(v, u);
        }
    }
}

int main() {
    cin >> n;
    for (int i = 1; i < n; i++) {
        int a, b; cin >> a >> b;
        graph[a].push_back(b);
        graph[b].push_back(a);
    }

    dfs(1, 1);

    for (int i = 2; i <= n; i++) cout << ans[i] << "\n";
}
```

질문?

문제

- **필수 문제**
- BOJ 2667 (단지번호붙이기)
- BOJ 11725 (트리의 부모 찾기)
- BOJ 1260 (DFS와 BFS)
- BOJ 16953 (A -> B)

- **연습/심화 문제**
- BOJ 12869 (뮤탈리스크)
- BOJ 10026 (적록색약)
- BOJ 7576 (토마토)