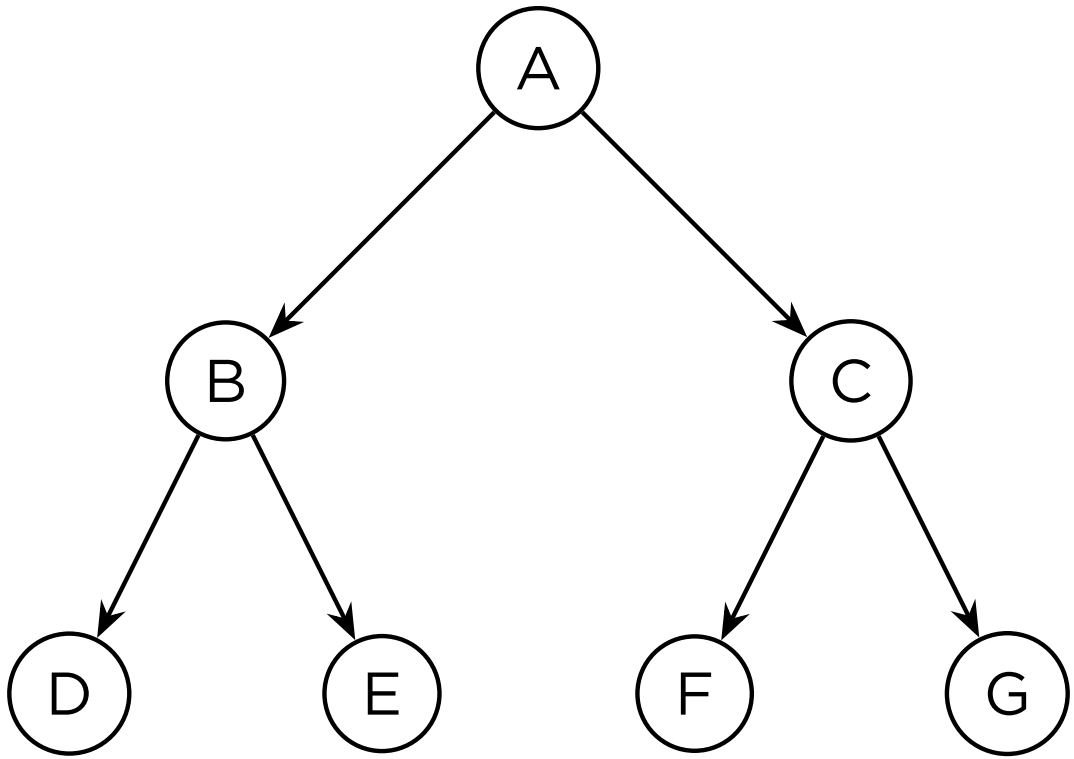
#5 트리의 탐색과 활용

2019 SCSC Summer Coding Workshop

서강대학교 컴퓨터공학과 박수현

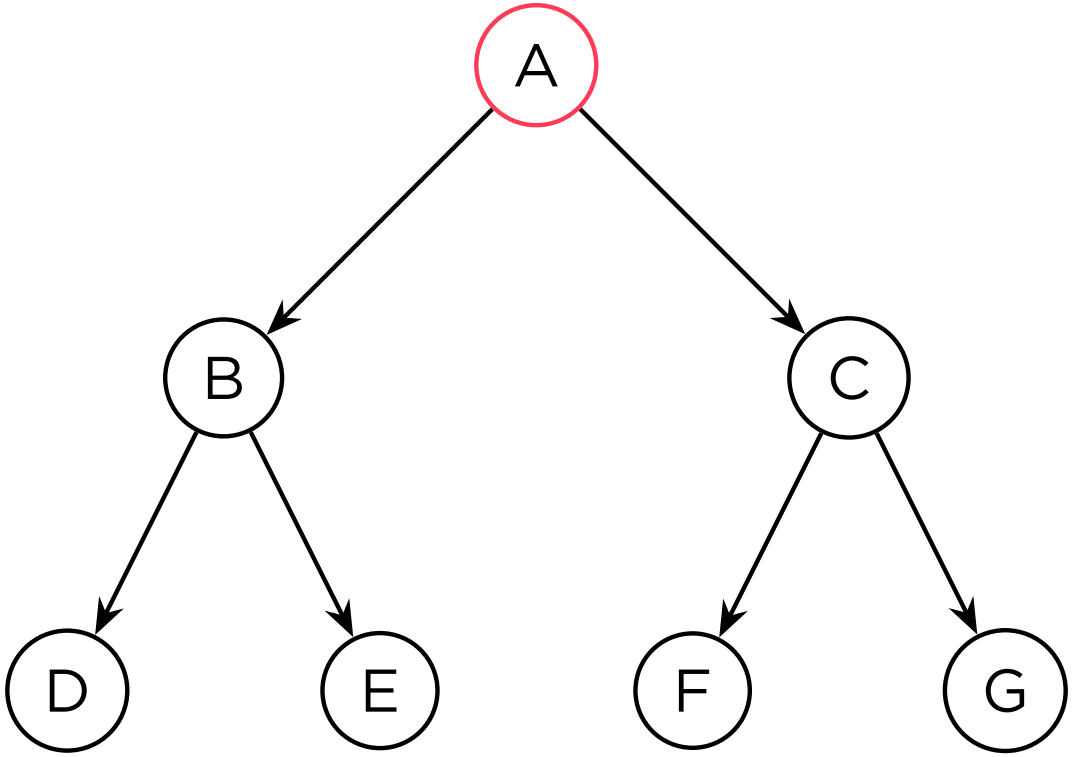
me@shiftpsh.com

트리



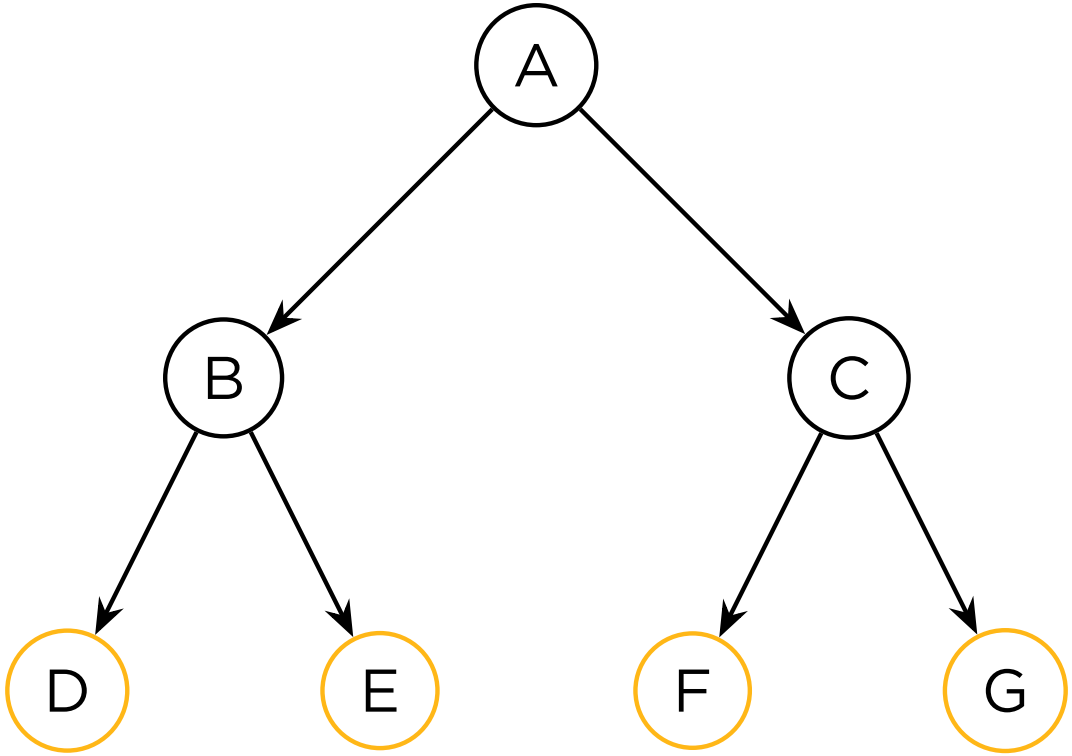
사이클cycle 이 없는 연결 그래프connected graph

트리



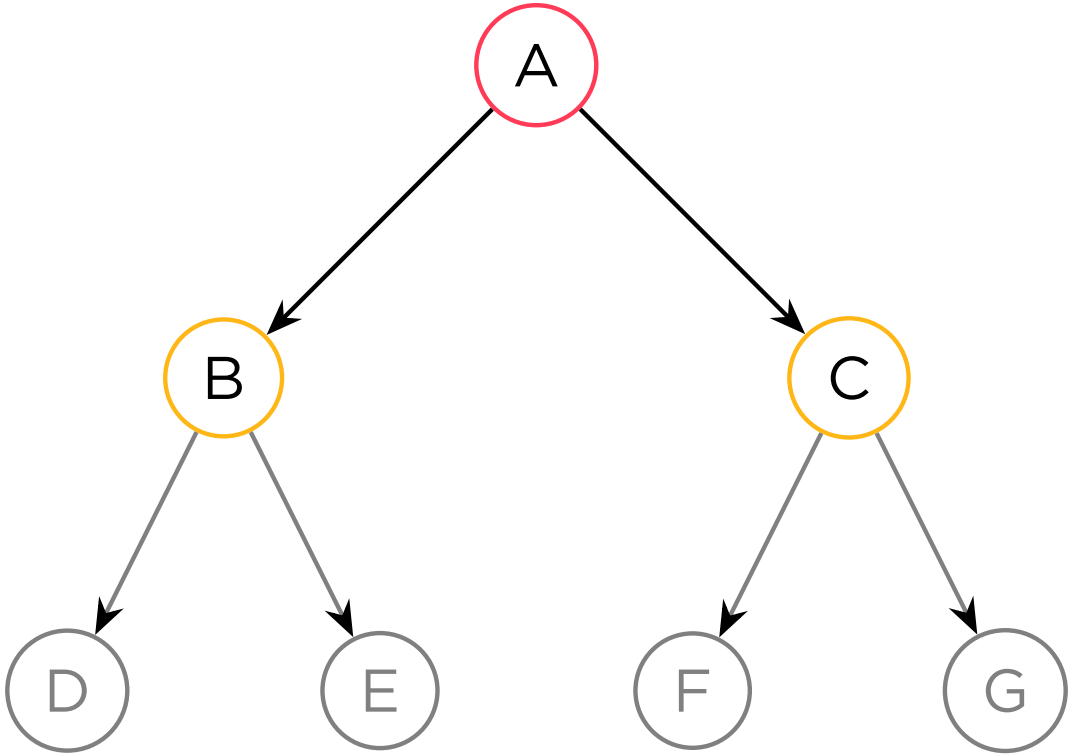
루트root 노드

트리



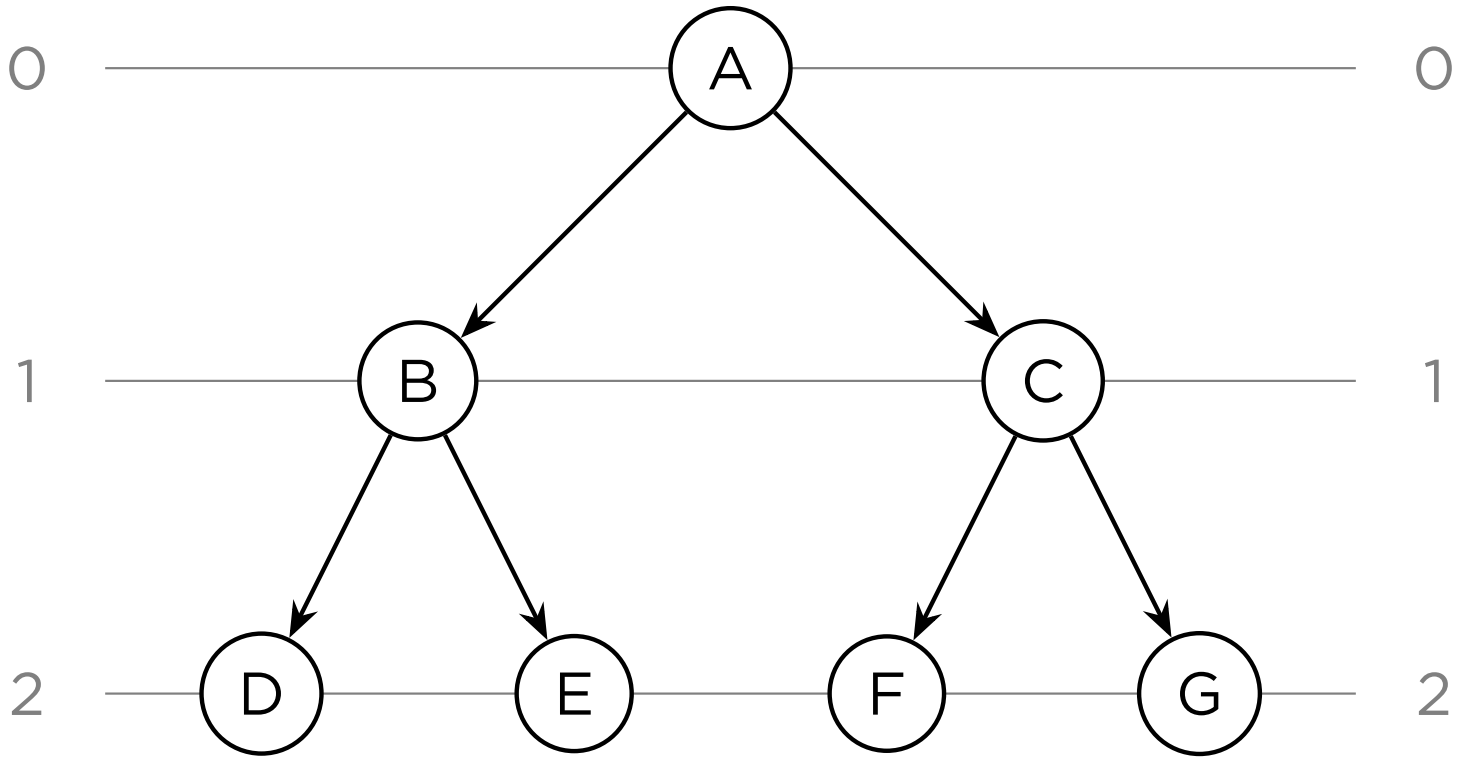
리프leaf 노드

트리



부모parent 노드와 자식child 노드

트리



레벨level 혹은 깊이depth

트리의 특징

- 노드가 N개 $\rightarrow$ 간선은 $N-1$개
- 임의의 두 노드 간의 경로는 유일
- 각 노드는 단 하나의 부모 노드를 가짐
- 그냥 그래프 다루듯이 다루면 된다!

트리의 저장

```
-1 0 0 1 1
```

‘0번 노드부터 $N-1$번 노드까지, 각 노드의 부모가 주어진다. 만약 부모가 없다면 (루트) -1이 주어진다.’

트리의 저장

```
-1 0 0 1 1
```

- 그냥 인접 리스트를 만들어도 루트부터 DFS/BFS 할 수 있다

루트는 0번 노드고, 간선 $0 \leftrightarrow 1, 0 \leftrightarrow 2, 1 \leftrightarrow 3, 1 \leftrightarrow 4$가 존재하는 트리

- 추가적으로 `parent` 배열을 만들면 좋다

$0, 1, 2, 3, 4$번 노드의 부모는 각각 $-1, 0, 0, 1, 1$번 노드

트리의 저장

```cpp
int n;
cin >> n;

for (int i = 0; i < n; i++) {
    cin >> parent[i];
    tree[i].emplace_back(parent[i]); // child → parent
    tree[parent[i]].emplace_back(i); // parent → child
}
```

문제 상황에 맞게 알아서 잘 판단해 저장하면 된다

트리의 탐색

트리는 어차피 그래프이기 때문에 DFS/BFS를 그대로 쓰면 된다,
다만 DFS의 경우 `visit` 배열이 필요 없다

- $\mathrm{dfs}(u, p)$ - u는 현재 확인하는 노드, p는 u의 부모 노드

트리의 저장

```
1  void dfs(int u, int p) {
2      for (int v : tree[u]) {
3          if (v == p) continue;
4          dfs(v, u);
5      }
6  }
```

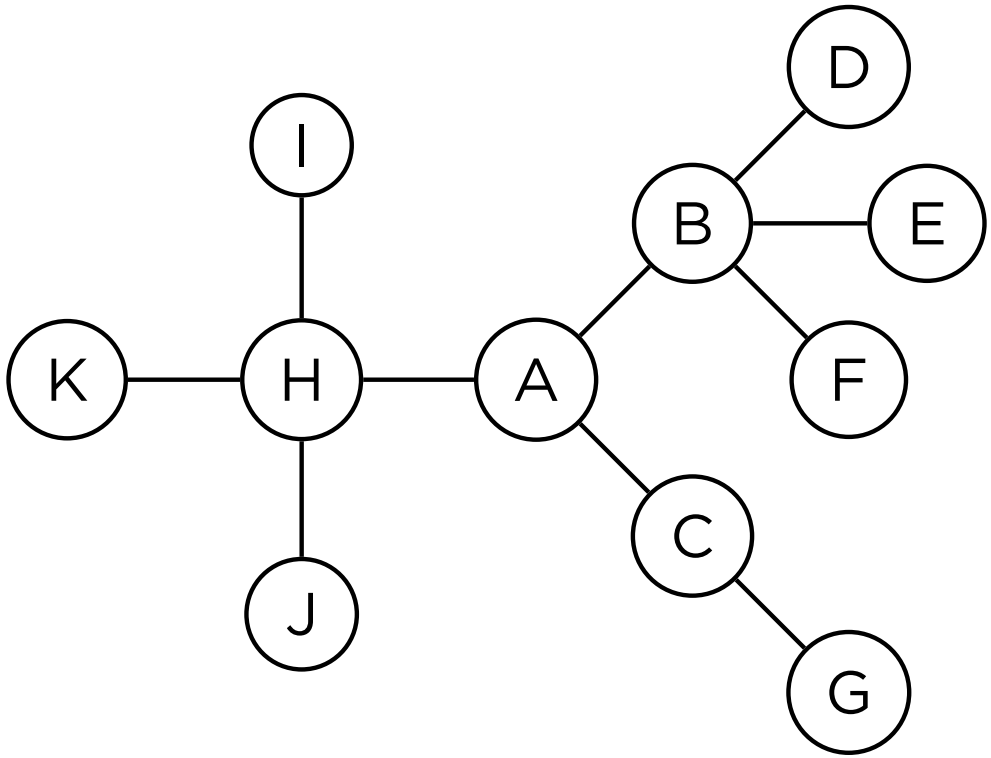
현재 확인하는 노드 u에 인접한 노드 v가 현재 확인하는 노드의 부모 p라면 그 방향으로 진행하지 않으면 된다

트리의 부모 찾기 BOJ #11725

루트가 정해지지 않은 트리가 주어진다. 트리의 루트를 1이라고 정했을 때, 각 노드의 부모를 구하는 프로그램을 작성하시오.

트리의 부모 찾기 BOJ #11725

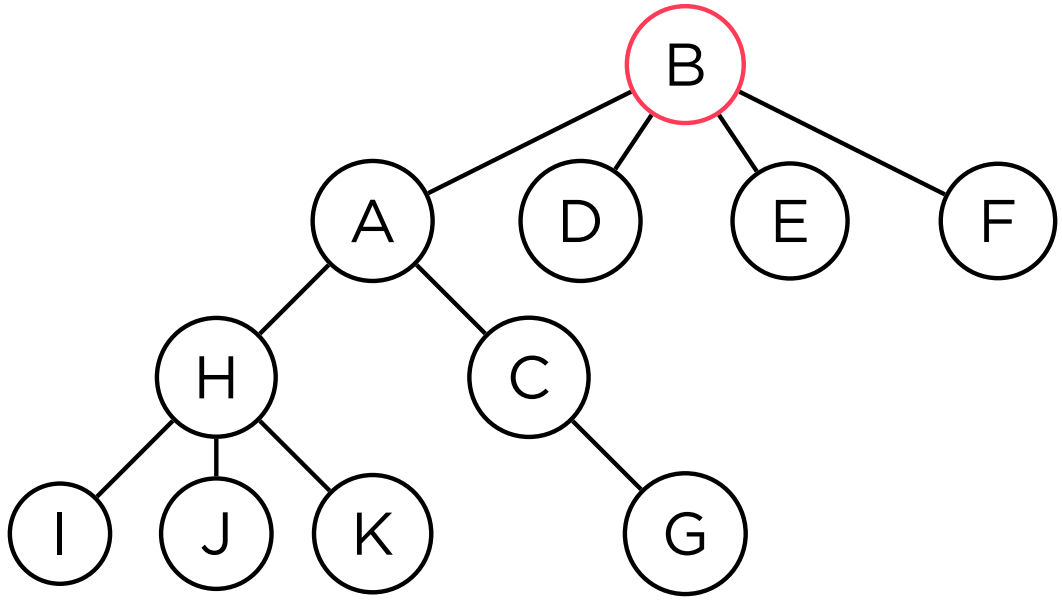
루트가 정해지지 않은 트리?



- 트리는 어떤 정점을 루트라고 정하더라도 여전히 트리다

트리의 부모 찾기 BOJ #11725

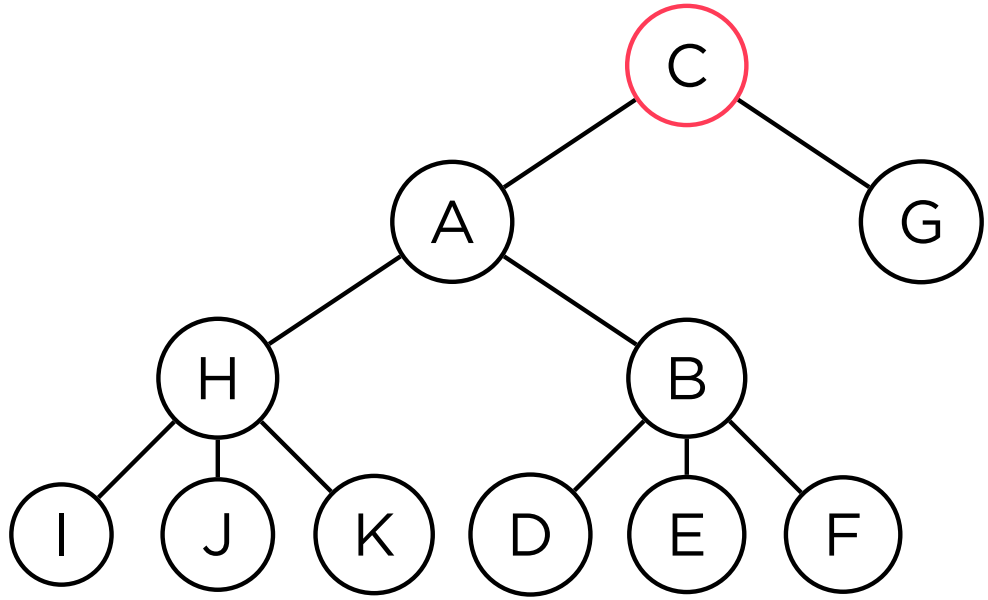
루트가 정해지지 않은 트리?



- 트리는 어떤 정점을 루트라고 정하더라도 여전히 트리다

트리의 부모 찾기 BOJ #11725

루트가 정해지지 않은 트리?



- 트리는 어떤 정점을 루트라고 정하더라도 여전히 트리다

트리의 부모 찾기 BOJ #11725

'트리의 루트를 1 이라고 정했을 때'

- 그래프의 연결관계가 주어지는데 그냥 루트를 우리 맘대로 1 이라고 놓고 탐색하겠다는 뜻
- 그래프를 받고 단순히 루트부터 DFS를 돌리면 된다

트리의 부모 찾기 BOJ #11725

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

int parent[100001];
vector<vector<int>> tree;

void dfs(int u, int p) {
    for (int v : tree[u]) {
        if (v == p) continue;
        parent[v] = u;
        dfs(v, u);
    }
}
```

트리를 그래프로 생각하고 인접 리스트로 저장

트리의 부모 찾기 BOJ #11725

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

int parent[100001];
vector<vector<int>> tree;

void dfs(int u, int p) {
    for (int v : tree[u]) {
        if (v == p) continue;
        parent[v] = u;
        dfs(v, u);
    }
}
```

$\mathrm{dfs}(u, p)$ - u는 현재 확인하는 노드, p는 u의 부모 노드.

트리에서 DFS는 항상 루트에서 리프 방향으로 진행된다

트리의 부모 찾기 BOJ #11725

```cpp
int main() {
    cin.tie(nullptr);
    cout.tie(nullptr);
    ios_base::sync_with_stdio(false);

    int n;
    cin >> n;
    tree.resize(n + 1);
    for (int i = 0; i < n - 1; i++) {
        int u, v;
        cin >> u >> v;
        tree[u].emplace_back(v);
        tree[v].emplace_back(u);
    }
```

$\mathrm{dfs}(u, p)$ - u는 현재 확인하는 노드, p는 u의 부모 노드.

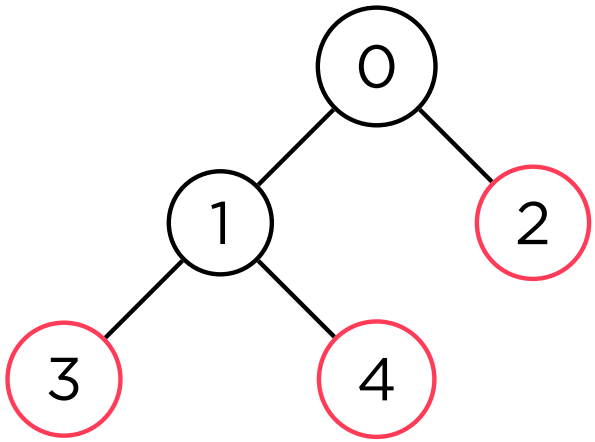
인접 리스트로 입력받는다

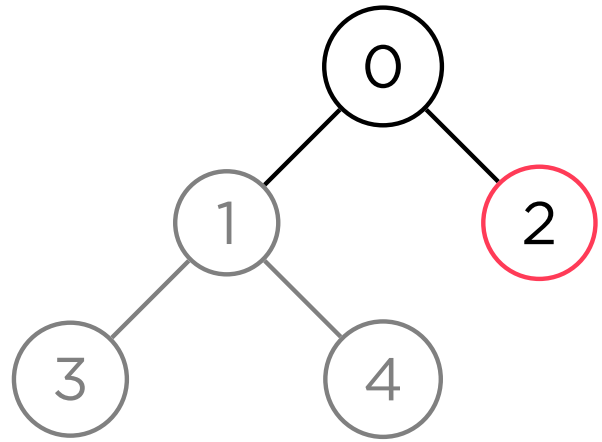
트리의 부모 찾기 BOJ #11725

```
31      dfs(1, -1);
32      for (int i = 2; i <= n; i++) {
33          cout << parent[i] << '\n';
34      }
35
36      return 0;
37  }
```

DFS를 한 번 돌리면 부모 노드 정보가 parent 배열에 저장된다

트리 BOJ #1068





트리가 주어졌을 때, 노드 중 하나를 제거할 것이다. 그 때, 남은 트리에서 리프 노드의 개수를 구하는 프로그램을 작성하시오.

트리 BOJ #1068

리프 노드의 조건?

- 자식 노드가 하나도 없으면 된다
- 제거한 노드를 무시하고 DFS를 돌리면서 각 노드마다 자식 개수를 센다
- 자식 개수가 0개라면 리프 노드이다

트리 BOJ #1068

```cpp
#include <iostream>
#include <vector>

using namespace std;

int n, root, del;
int leaves = 0;
vector<vector<int>> tree;
```

트리는 그래프로 취급 - 인접 리스트로 저장

트리 BOJ #1068

```
10  void dfs(int u, int p) {
11      int children = 0;
12      for (int v : tree[u]) {
13          if (v == p || v == del) continue;
14          dfs(v, u);
15          children++;
16      }
17      if (children == 0) {
18          leaves++;
19      }
20  }
```

$\mathrm{dfs}(u, p)$ - u는 현재 확인하는 노드, p는 u의 부모 노드

부모 노드이거나 제거한 노드일 경우 무시하고 진행

트리 BOJ #1068

```
10  void dfs(int u, int p) {
11      int children = 0;
12      for (int v : tree[u]) {
13          if (v == p || v == del) continue;
14          dfs(v, u);
15          children++;
16      }
17      if (children == 0) {
18          leaves++;
19      }
20  }
```

자식 노드가 하나도 없다면 리프 노드인 것이다

트리 BOJ #1068

```
22  int main() {
23      cin >> n;
24      tree.resize(n);
25
26      for (int i = 0; i < n; i++) {
27          int x;
28          cin >> x;
29          if (x == -1) {
30              root = i;
31          } else {
32              tree[i].emplace_back(x);
33              tree[x].emplace_back(i);
34          }
35      }
```

인접 리스트로 받고 루트의 경우 따로 처리해 준다

트리 BOJ #1068

```
37      cin >> del;
38      if (del != root) dfs(root, -1);
39
40      cout << leaves;
41
42      return 0;
43  }
```

루트를 삭제하는 경우 트리 전체가 삭제되므로 이 경우는 예외로 하고, DFS로 탐색 후 결과 출력

트리에서의 최단 경로

트리에서 노드와 노드 사이의 경로를 구하면 그게 바로 최단 경로이다

- 트리에서는 노드와 노드 사이의 경로가 유일하기 때문
- 역시 DFS(혹은 BFS) 한 번으로 구할 수 있다

트리에서의 최단 경로

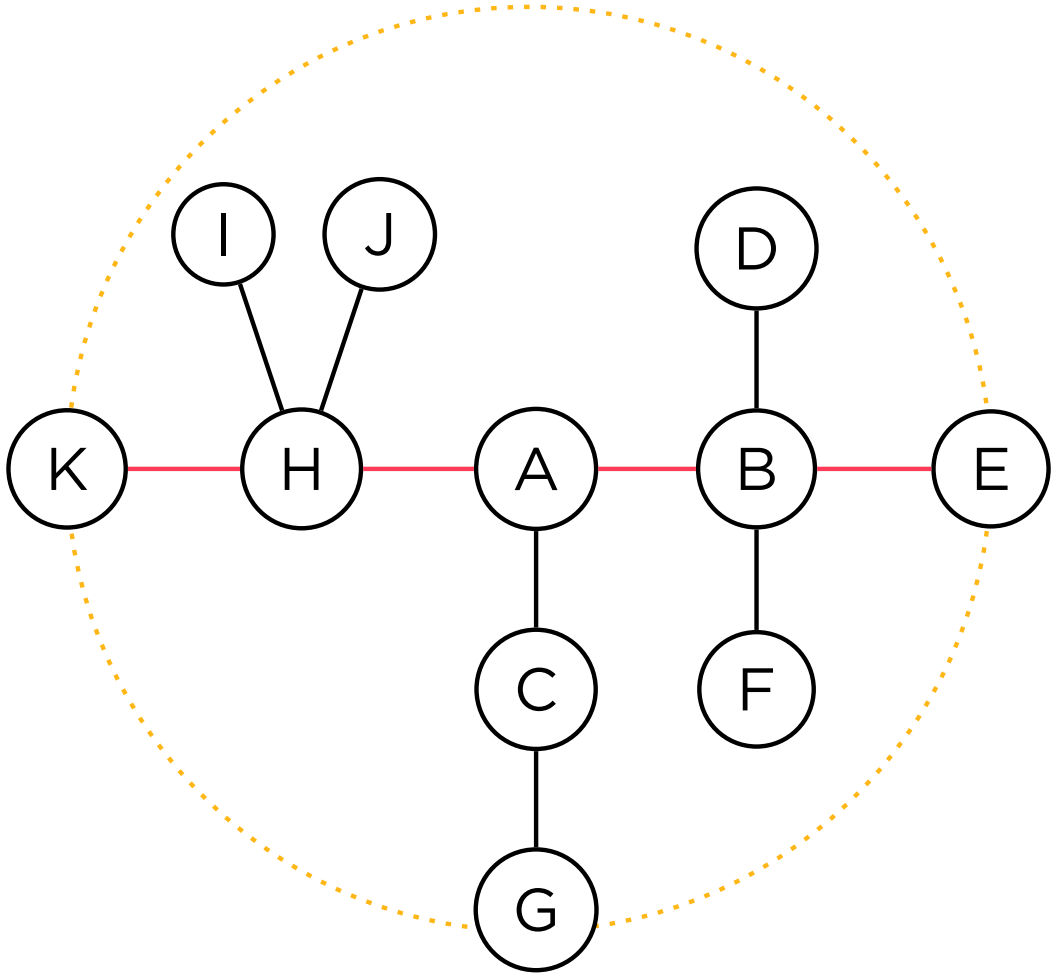
```cpp
int dist[MAXN];

void dfs(int u, int p) {
    for (auto vc : tree[u]) { // v, dist
        int v = vc.first, c = vc.second;
        if (v == p) continue;
        dist[v] = dist[u] + c;
        dfs(v, u);
    }
}
```

단일 시작점(루트)에서 모든 노드까지 최단 경로 구하기

트리의 지름 BOJ #1967

트리에서 어떤 두 노드를 선택해서 양쪽으로 쫙 당길 때, 가장 길게 늘어나는 경우가 있을 것이다. 이럴 때 트리의 모든 노드들은 이 두 노드를 지름의 끝 점으로 하는 원 안에 들어가게 된다.
이런 두 노드 사이의 경로의 길이를 **트리의 지름**이라고 한다. 정확히 정의하자면 트리에 존재하는 모든 경로들 중에서 가장 긴 것의 길이를 말한다.

트리의 지름 BOJ #1967



트리의 지름 BOJ #1967

입력으로 루트가 있는 트리를 가중치가 있는 간선들로 줄 때, 트리의 지름을 구해서 출력하는 프로그램을 작성하시오.

트리의 지름 BOJ #1967

- 루트 r에서 가장 먼 점 x를 고르고
- x에서 가장 먼 점 y를 고르면
- $x \leftrightarrow y$가 바로 트리의 지름이다

이게 왜 되지?!

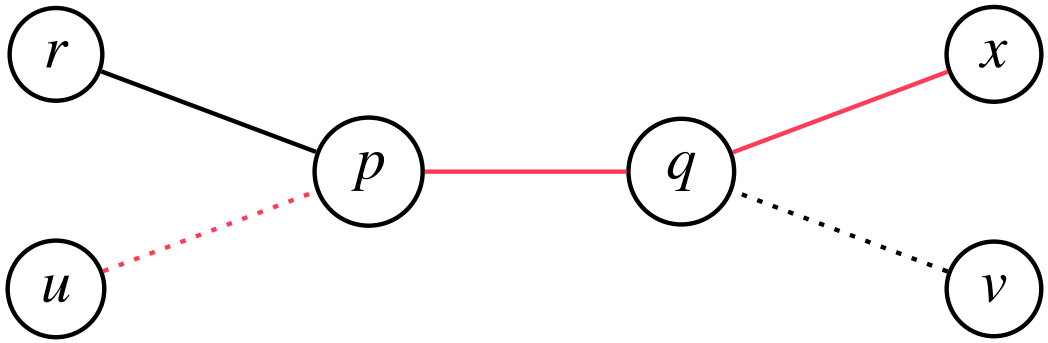
트리의 지름 BOJ #1967

트리의 지름을 이루는 경로의 시작점을 u, 끝점을 v라고 둔다면

- r이 u 또는 v인 경우? $\rightarrow x \leftrightarrow y$는 당연히 지름
- x가 u 또는 v인 경우? $\rightarrow x \leftrightarrow y$는 당연히 지름
- 만약에 둘 다 아니고...
  - $r \leftrightarrow x$의 경로가 $u \leftrightarrow v$의 경로와 겹친다면?
  - $r \leftrightarrow x$의 경로가 $u \leftrightarrow v$의 경로와 안 겹친다면?

트리의 지름 BOJ #1967

- $r \leftrightarrow x$의 경로가 $u \leftrightarrow v$의 경로와 겹친다면?

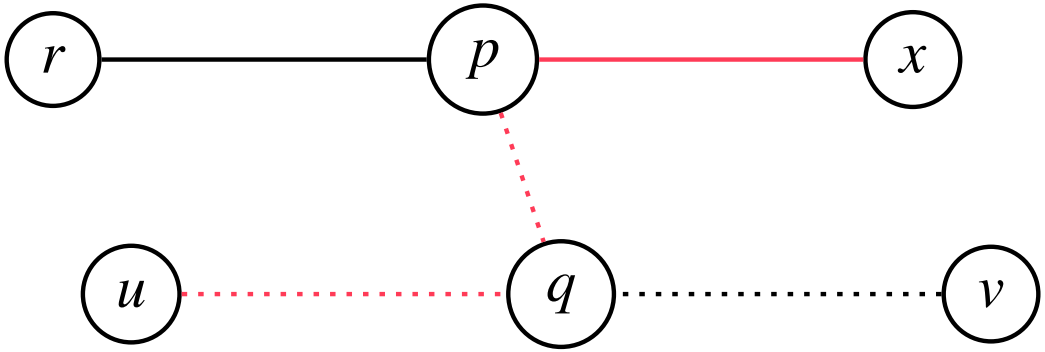


r에서 가장 먼 점은 x이므로 $q \leftrightarrow x > q \leftrightarrow v$

$\Rightarrow u \leftrightarrow x > u \leftrightarrow v$이므로 가정에 모순

트리의 지름 BOJ #1967

- $r \leftrightarrow x$의 경로가 $u \leftrightarrow v$의 경로와 안 겹친다면?



r에서 가장 먼 점은 x이므로 $q \leftrightarrow x > p \leftrightarrow x > p \leftrightarrow v > q \leftrightarrow v$

$\Rightarrow u \leftrightarrow x > u \leftrightarrow v$이므로 가정에 모순

트리의 지름 BOJ #1967

```cpp
#include <iostream>
#include <algorithm>
#include <vector>
#include <tuple>

using namespace std;
using pii = pair<int, int>;

vector<vector<pii>> tree; // destination, cost

int dist[10001];

void dfs(int u, int p) {
    for (auto vc : tree[u]) {
        int v = vc.first, c = vc.second;
        if (v == p) continue;
        dist[v] = dist[u] + c;
        dfs(v, u);
    }
}
```

최단 거리를 구해 주는 코드

트리의 지름 BOJ #1967

```
22  int main() {
23      cin.tie(nullptr);
24      cout.tie(nullptr);
25      ios_base::sync_with_stdio(false);
26
27      int n;
28      cin >> n;
29      tree.resize(n + 1);
30
31      for (int i = 0; i < n - 1; i++) {
32          int u, v, c;
33          cin >> u >> v >> c;
34          tree[u].emplace_back(pii(v, c));
35          tree[v].emplace_back(pii(u, c));
36      }
```

인접 리스트로 입력받는다

트리의 지름 BOJ #1967

```
38      // 1st dfs → get u
39      dist[1] = 0;
40      dfs(1, -1);
41      int mx = -1, u;
42      for (int i = 1; i ⩽ n; i++) {
43          if (mx < dist[i]) {
44              mx = dist[i];
45              u = i;
46          }
47      }
```

1번째 DFS: 1번 노드를 루트로 생각하고, 루트에서 가장 먼 노드 u 찾기

트리의 지름 BOJ #1967

```
49      // 2nd dfs → get v
50      dist[u] = 0;
51      dfs(u, -1);
52      mx = -1;
53      for (int i = 1; i ⩽ n; i++) {
54          mx = max(mx, dist[i]);
55      }
56
57      cout << mx;
58
59      return 0;
60  }
```

2번째 DFS: u번 노드를 루트로 생각하고, 루트에서 가장 먼 노드까지의 거리 출력

문제 풀어보고, 질문하는 시간 (~17시까지)