SAMSUNG

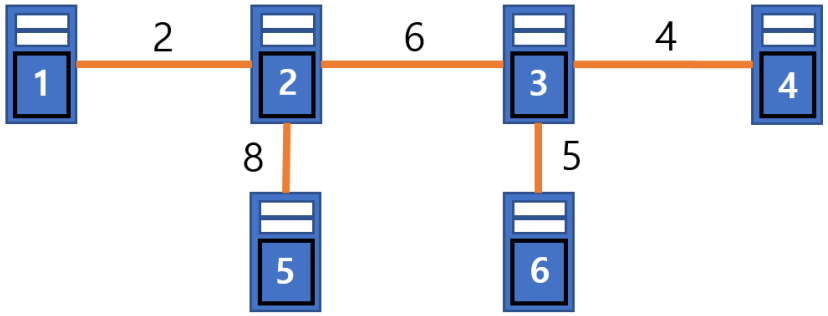
신소재 케이블2

문제 설명

신소재를 활용한 유선 통신용 케이블을 개발하였다.

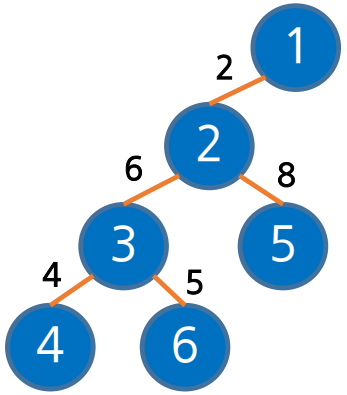
실제로 구성된 네트워크에서 케이블의 성능과 안정성을 테스트를 하기 위해 **n개의 통신 장비를 n - 1개의 케이블로 연결한다.**

(구성된 네트워크는 **트리 구조**를 가진다. 즉, 임의의 두 장비 간의 경로가 존재하고 유일하다.)



[Fig. 1]

Tree 구조 문제라는 Hint !



하나의 케이블은 두 대의 통신 장비가 양방향 통신을 할 수 있다. 각각의 케이블은 고유의 전송 시간을 가진다.

두 장비가 직접 연결되어 있지 않더라도 두 장비 사이에 케이블과 다른 장비로 구성된 경로가 존재할 경우 신호 전송이 가능하다.

이 때, 전송 시간은 경로상에 있는 케이블의 전송 시간의 합이 되고 신호는 이미 지나간 장비를 다시 지나가지 않는다.

[Fig. 1]에서 장비 1과 장비 4의 경우 장비 2와 장비 3을 통해 신호 전송이 가능하며 이때 신호 전송 시간은 2 + 6 + 4 = 12이다. ([Fig. 1]에서 장비 그림 안에 있는 수는 장비 ID를 의미하고 선 위에 있는 수는 전송 시간을 의미한다.) ➡ measure(1, 4) 함수 예시. return 값 12

모든 장비는 서로 통신이 가능하도록 연결이 되어 있다.

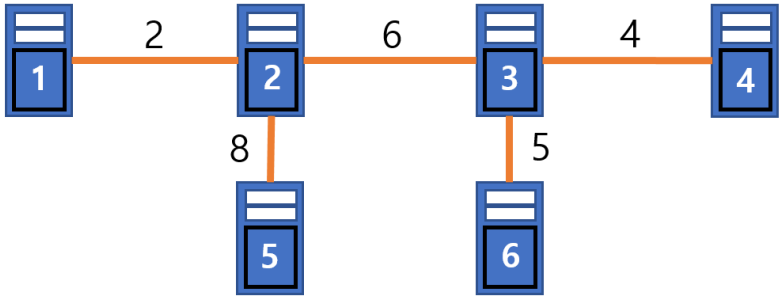
문제 설명

케이블의 테스트는 한 장비에서 다른 장비로 신호를 전송하고 전송 경로에 있는 한 장비가 신호를 모니터링을 하는 방식으로 진행된다.

테스트를 하기 전에 모니터링을 할 장비를 지정한다.

테스트의 신뢰도를 높이기 위해서 전송 시간이 최대가 되도록 보내는 장비와 받는 장비를 선택한다. 만약 전송 시간이 최대가 되는 경우가 여러 개인 경우 그 중 아무거나 선택한다.

모니터링을 하는 장비가 보내는 장비 또는 받는 장비가 되는 것도 가능하다.



[Fig. 1]

[Fig. 1]에서 장비 2에서 신호를 모니터링한다면 신호를 보내는 장비와 받는 장비로 장비 5와 장비 6을 선택한다. 이 두 장비가 장비 2에서 신호를 모니터링할 때 전송 시간이 최대가 되는 경우이고 전송 시간은 8 + 6 + 5 = 19이다. ➡ test(2) 함수 예시. return 값 19

장비 1에서 신호를 모니터링한다면 신호를 보내는 장비와 받는 장비로 장비 1과 장비 6을 선택한다. 이 두 장비가 장비 1에서 신호를 모니터링할 때 전송 시간이 최대가 되는 경우이고 전송 시간은 2 + 6 + 5 = 13이다. ➡ test(1) 함수 예시. return 값 13

문제/API 분석

void init(int mDevice)

테스트 케이스에 대한 초기화 함수. 각 테스트 케이스의 맨 처음 1회 호출된다.

초기에 장비 mDevice가 있다. mDevice는 장비 번호이다.

Parameters
mDevice : 초기 장비 번호 (1 ≤ mDevice ≤ 1,000,000,000)

[제약사항] 1. 각 테스트 케이스 시작 시 init() 함수가 한 번 호출된다.

- init() 함수
  - ✓ 초기 장비 번호 mDevice 를 트리의 첫 번째 노드로 구성

void connect(int mOldDevice, int mNewDevice, int mLatency)

새로운 장비 mNewDevice를 추가하고 장비 mNewDevice와 장비 mOldDevice를 전송 속도가 mLatency인 케이블로 연결한다.

mOldDevice는 이미 존재하는 장비 번호이다.

mNewDevice는 추가되는 새로운 장비 번호이다.

Parameters
mOldDevice : 이미 존재하는 장비 번호 (1 ≤ mOldDevice ≤ 1,000,000,000)
mNewDevice : 새롭게 추가되는 장비 번호 (1 ≤ mNewDevice ≤ 1,000,000,000)
mLatency : 케이블의 전송 속도 (1 ≤ mLatency ≤ 10,000)

[제약사항] 2. 각 테스트 케이스에서 connect() 함수의 호출 횟수는 10,000 이하이다.

- connect() 함수
  - ✓ 새로운 장비를 추가하는 함수
  - ✓ 최대 1만 번 호출 -> **최대 장비 개수 10,001개**
  - ✓ 장비 번호는 1~10억 범위의 정수로 표현 -> **Renumbering (Hash 사용)**

문제/API 분석

int measure(int mDevice1, int mDevice2)

장비 mDevice1에서 장비 mDevice2로 신호를 전송했을 때 전송 시간을 반환한다.

mDevice1와 mDevice2는 이미 존재하는 장비 번호이고 서로 다르다.

Parameters
mDevice1 : 장비 번호 (1 ≤ mDevice1 ≤ 1,000,000,000)
mDevice2 : 장비 번호 (1 ≤ mDevice2 ≤ 1,000,000,000, mDevice1 ≠ mDevice2)

Returns
장비 mDevice1에서 장비 mDevice2로 신호를 전송했을 때 전송 시간

[제약사항] 3. 각 테스트 케이스에서 measure() 함수의 호출 횟수는 1,000 이하이다.

- measure() 함수
  - ✓ 두 장비 사이의 전송 시간 (거리) 반환
  - ✓ Tree 구조에서 두 노드 사이의 거리 탐색 : DFS, BFS 사용
  - ✓ **Testcase 당 시간 복잡도 : 함수 호출 수 1,000 * 노드의 수 10,001 = 약 1천만**

-> 시간 복잡도를 고려했을 때 단순 DFS, BFS로도 해결 가능하다.

문제/API 분석

int test(int mDevice)

신호를 모니터링하는 장비를 장비 mDevice로 하고 테스트를 진행한다.

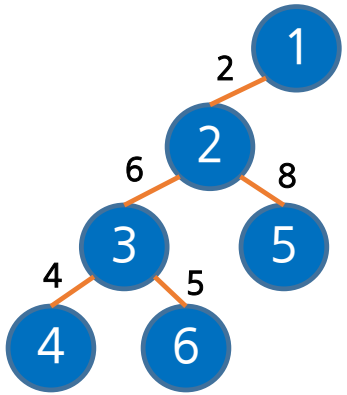
전송 신호가 장비 mDevice를 지나가고 전송 시간이 최대가 되도록 보내는 장비와 받는 장비를 선택하고 이때의 전송 시간을 반환한다. (본문 설명 참조)

mDevice는 이미 존재하는 장비 번호이다.

함수 호출 시, 이미 존재하는 장비는 2개 이상 있음을 보장한다.

Parameters
mDevice : 장비 번호 (1 ≤ mDevice ≤ 1,000,000,000)

Returns
장비 mDevice로 전송 신호가 지나갈 때 최대 전송 시간



[제약사항] 4. 각 테스트 케이스에서 test() 함수의 호출 횟수는 1,000 이하이다.

- test() 함수
  - ✓ 모니터링 장비를 지나가는 모든 경로 중 최대 전송 시간(거리) 반환
  - ✓ Tree 구조에서 두 노드 사이의 거리 탐색 : DFS, BFS 사용
    1. 모니터링 장비(mDevice)를 root로 생각하고
    2. mDevice의 자식 노드들을 기점으로 DFS 사용
    3. 가장 긴 2개의 거리의 합이 최대 전송 시간(거리)
  - ✓ Testcase 당 시간 복잡도 : 함수 호출 수 1,000 * 노드의 수 10,001 = 약 1천만

-> 시간 복잡도를 고려했을 때 단순 DFS, BFS로도 해결 가능하다.

[제약사항] 5. 임의의 두 장비의 전송 경로에 있는 장비의 수는 100 이하이다.

-> 탐색이 크게 길어지지 않는다.

-> 즉, DFS 또는 BFS 를 사용해서 구현 완성 시 큰 문제 없다!!

하지만, 이런 문제가 나오는 경우는 거의 드물다. 항상 최적화하는 연습 필요!!

구현 : DFS

앞서 문제 분석에서 확인했듯이 시간 복잡도 상 단순하게 DFS 또는 BFS로 구현해도 문제가 없다.

먼저 measure() 와 test() 를 DFS로 어떻게 구현할 수 있는지 알아보고,

최적화를 할 수 있는 방법에 대해 생각해보자.

1. 자료구조 선언

```
constexpr int MAX_DEVICE = 10005;
int root, id;

using pii = pair<int, int>;
vector<pii> adj[MAX_DEVICE];

unordered_map<int, int> idmap;
```

- root : init()에서 받은 첫 노드
- id : renumbering 된 device id
- pair<Device id, latency>를 데이터로 갖는 vector 선언 : 인접배열
- idmap : Renumbering을 위한 hash map 사용

구현 : DFS

2. init()

```
void init(int mDevice)
{
    for (int i = 0; i <= id; ++i)
    adj[i].clear();

    root = id = 1;
    idmap.clear();
    idmap[mDevice] = id;
}
```

3. connect()

```
void connect(int mOldDevice, int mNewDevice, int mLatency)
{
    int old_device_id = idmap[mOldDevice];
    int new_device_id = ++id;

    idmap[mNewDevice] = new_device_id;

    adj[old_device_id].push_back({ new_device_id, mLatency });
    adj[new_device_id].push_back({ old_device_id, mLatency });
}
```

- connect() : 기존 장비 mOldDevice 와 새로운 장비 mNewDevice를 전송 속도가 mLatency 인 케이블로 연결
- old device와 new device를 양방향으로 연결
- idmap으로 기존 장비 renumbering id인 old_device id를 얻어옴
- idmap에 새로운 장비 renumbering id 등록
- 인접배열 adj에 old device와 new device 연결해서 등록하여 양방향 그래프 구성

구현 : DFS

4. measure()

```
int getLatency(int pid, int sid, int eid, int dist) {
    if (sid == eid) return dist;

    for (auto& a : adj[sid]) {
        if (pid != a.first) {
            int ret = getLatency(sid, a.first, eid, dist + a.second);
            if (ret) return ret;
        }
    }
    return 0;
}
```

- getLatency() : dfs를 이용하여 eid까지 거리 구하기
- pid는 부모이므로 제외하고 탐색

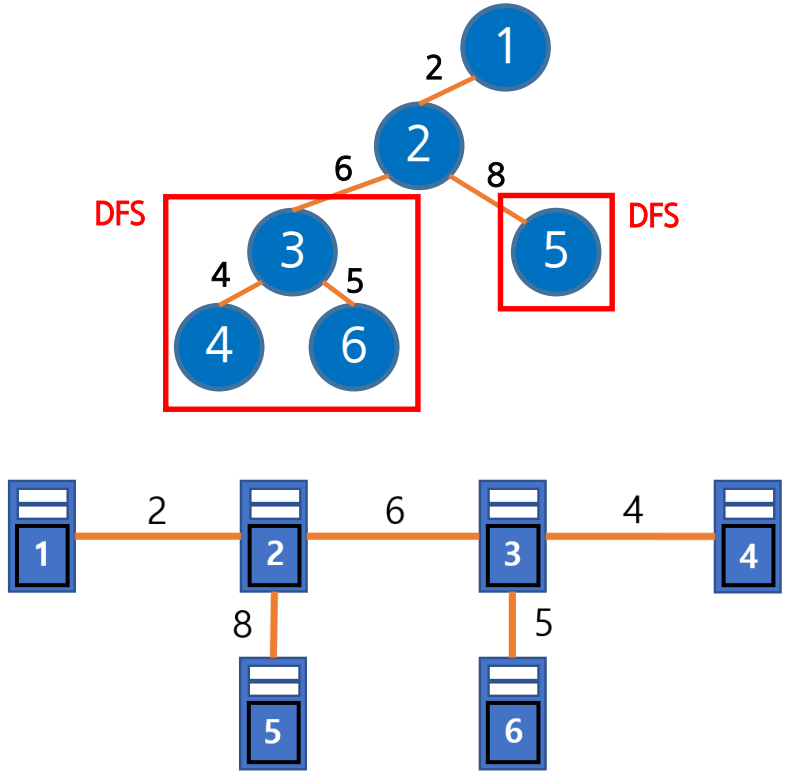
```
int measure(int mDevice1, int mDevice2)
{
    int sid = idmap[mDevice1];
    int eid = idmap[mDevice2];

    return getLatency(sid, sid, eid, 0);
}
```

- getLatency() 를 사용하여 sid(mDevice1)에서 eid(mDevice2)로 신호를 전송했을 때 latency 측정
- 최악의 경우 전체 노드 탐색 -> O(함수 호출 수 1,000 * 최대 노드 수 10,001) 약 1천만

구현 : DFS

5. test()

구현 방법 예시 아래 그림과 같이 주어졌을 때 test(2)인 경우



[Fig. 1]

모니터링 장비가 장비 2인 경우,

장비 2를 **가상의 root로 생각**하고 2의 **자식 노드인 3, 5를 기점으로 DFS를 사용**하여 최대 전송 시간을 구한다.

① 장비 3을 기점으로 DFS를 사용했을 때 얻을 수 있는 최대 latency : 5

② 장비 5를 기점으로 DFS를 사용했을 때 얻을 수 있는 최대 latency : 0 (5의 자식 X)

가장 큰 latency는 ① + 장비2와 장비3 사이의 latency = 11이 되고,

두번째로 큰 latency는 ② + 장비2와 장비 5사이의 latency = 8 이 된다.

따라서 장비 2에서 신호를 모니터링할 때 전송시간이 최대가 되는 경우는 11 + 8 = 19가 된다.

구현 : DFS

5. test()

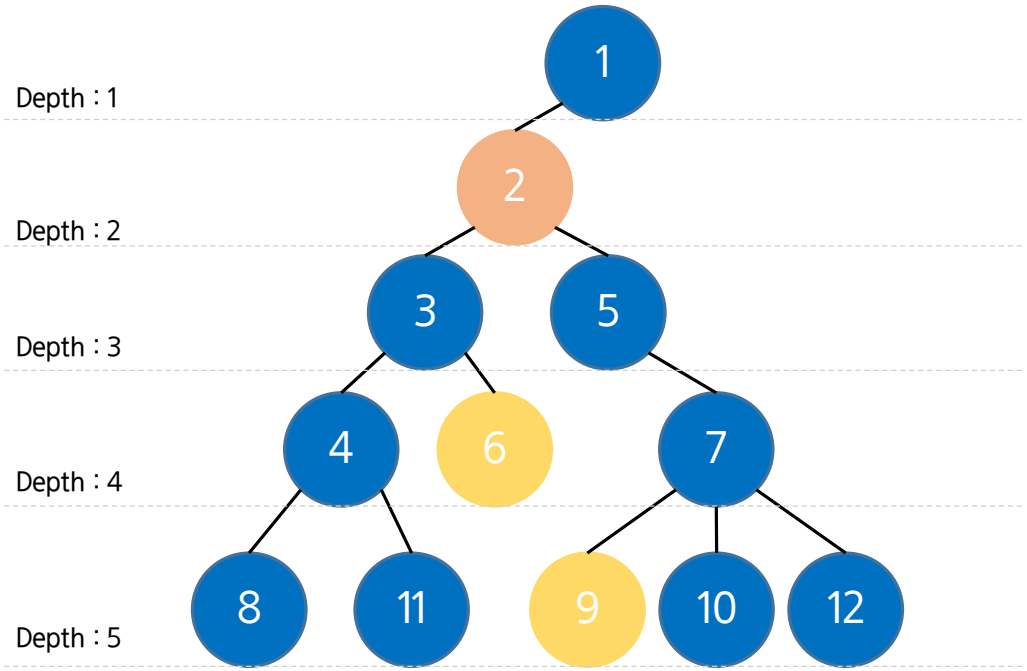
```
int getMaxLatency(int pid, int id) {
    int max_latency = 0;
    for (auto& a : adj[id]) {
         if (pid != a.first) {
             int ret = getMaxLatency(id, a.first) + a.second;
             if (max_latency < ret)
                  max_latency = ret;
         }
    }
        return max_latency;
}


int test(int mDevice)
{
    int id = idmap[mDevice];
    int first = 0, second = 0;
    for (auto& a : adj[id]) {
        int lat = getMaxLatency(id, a.first) + a.second;
        if (lat > first)
            second = first, first = lat;
        else if (lat > second)
            second = lat;
    }
        return first + second;
}
```

- getMaxLatency() : dfs를 이용하여 leaf node까지 거리 구하기
- 모든 adj (인접배열)의 node에 대해 leaf node까지 거리를 탐색하고, 탐색을 통해 얻은 latency가 더 크다면 Max latency 갱신
- pid는 부모이므로 제외하고 탐색

- 모니터링 장비인 mDevice에 대해 idmap으로 renumbering id를 얻어 옴
- **first** : 가장 큰 latency, **second** : 두번째로 큰 latency
- 모니터링 장비 id를 가상의 root로 생각하고 id의 자식을 시점으로 max latency를 구함 : getMaxLatency() 사용
- getMaxLatency + adj[id]의 latency 값이 first 또는 second 보다 크다면 갱신해 줌
- 1번째로 큰 latency + 2번째로 큰 latency 이 모니터링 장비 mDevice를 지나는 최대 전송시간

최적화 (1) LCA

LCA 란? **Lowest Common Ancestor** 의 약자로 **Tree 구조에서 최소 공통 조상을 찾는 알고리즘**이다. 즉, 두 정점에서 가장 가까운 공통 조상을 찾는 알고리즘이다.



정점 6, 9 의 최소 공통 조상은 무엇일까? 왼쪽 그림에서도 확인할 수 있듯이 LCA(6,9) = 2 이다.

각 노드의 parent정보와 depth를 기록해 놓으면 쉽게 구할 수 있다.

1) 두 노드의 depth를 확인한다.
2) 만약 두 노드의 depth가 다르다면, 더 깊은 노드의 depth를 -1를 해준다.(즉, 부모 노드로 이동한다)
3) 만약 두 노드의 depth가 같다면, 두 노드 모두 depth를 -1를 해준다.
4) depth를 1씩 줄여가면서 노드가 같아지는 지점을 찾는다.
5) 이 지점이 곧 최소 공통 조상이 된다.

예시)

1) node 6의 depth : 4, node 9의 depth : 5
2) node 9의 depth가 더 큰 값이므로 -1 하여 node 7 로 이동한다.
3) node 6과 node 7의 depth는 모두 4이다.
4) 따라서 두 노드 모두 depth -1로 node 6 은 node 3 으로, node 7 은 node 5 로 이동한다.
5) 두 노드가 같아질 때 까지 반복하면 결국 두 노드는 node 2로 같게 된다. node 2는 node 6과 node 9의 최소 공통 노드가 된다.

최적화 (1) LCA 를 사용하여 measure() 구현

[제약사항] 5. 임의의 두 장비의 전송 경로에 있는 장비의 수는 100 이하이다. 제약사항 5번을 생각해보면, 트리의 최대 깊이는 100이 된다는 것을 알 수 있다.

따라서, 두 장비 간 전송 시간을 반환하는 measure()에 LCA를 사용할 경우 시간복잡도는 O(함수 호출 수 1,000 * 트리의 최대 깊이 : 100 = 100,000)으로 단순 DFS를 사용했을 때의 시간 복잡도 1천만보다 빠르다는 것을 알 수 있다.

1. 자료구조 선언

```
constexpr int MAX_DEVICE = 10005;
int root, id;

int parent[MAX_DEVICE];
int distToParent[MAX_DEVICE];
int depth[MAX_DEVICE];

using pii = pair<int, int>;
vector<pii> adj[MAX_DEVICE];

unordered_map<int, int> idmap;
```

measure()에서 LCA를 위해 사용하는 변수 추가

- parent[] : 부모 노드 기록
- distToParent[] : 부모 노드와의 latency 기록
- depth : 노드 depth 기록

2. init() : 동일

최적화 (1) LCA 를 사용하여 measure() 구현

3. connect()

```
void connect(int mOldDevice, int mNewDevice, int mLatency)
{
    int old_device_id = idmap[mOldDevice];
    int new_device_id = ++id;

    idmap[mNewDevice] = new_device_id;

    parent[new_device_id] = old_device_id;
    depth[new_device_id] = depth[old_device_id] + 1;
    distToParent[new_device_id] = mLatency;

}
```

- 새로운 장비를 등록 할 때, 부모 장비 정보 입력
- 새로운 장비에 대한 depth 기록
- 부모 장비까지의 거리. 즉, latency 기록

최적화 (1) LCA 를 사용하여 measure() 구현

4. measure()

```
int measure(int mDevice1, int mDevice2)
{
    int sid = idmap[mDevice1];
    int eid = idmap[mDevice2];

①   if (depth[sid] > depth[eid]) swap(sid, eid);

    int diff = depth[eid] - depth[sid];
    int ret = 0;
②   while (diff--) {
        ret += distToParent[eid];
        eid = parent[eid];
    }

③   while (sid != eid) {
        ret += distToParent[sid] + distToParent[eid];
        sid = parent[sid];
        eid = parent[eid];
    }

    return ret;
}
```

① 두 노드를 비교하여 depth가 더 큰 값을 갖는 노드를 eid로 지정

② 두 노드의 depth 차이 값 만큼 반복하여 eid를 sid와 같은 depth위치까지 이동

③ 두 노드가 같을 때까지 부모 노드로 이동하면서, latency 더해줌

최적화 (2) LCA + max latency 기록

앞에서 우리는 measure() 를 LCA를 이용하여 최적화할 수 있었다. 그렇다면 test()는 어떻게 최적화할 수 있을까?

DFS 탐색을 사용해서 test()에서 첫번째로 큰 latency와 두번째로 큰 latency를 더해서 최종 값을 구할 수 있었다.

이 방법을 생각해보면 우리는 각 노드 별 첫번째로 큰 latency와 node , 두번째로 큰 latency와 node를 미리 기록하고 알 수 있다면 더 빠르게 test() 결과 값을 얻을 수 있다는 것을 알 수 있다.

1) 새로운 장비를 추가할 때마다 각 노드의 first max latency와 node, second max latency와 node를 업데이트 해준다.

2) 최적화(1)에서도 언급했듯이 제약사항 5번에 의해 트리의 최대 깊이는 100이므로 test()의 시간복잡도도 O(함수 호출 수 1,000 * 트리의 최대 깊이 100 = 100,000)이 된다.

최적화 (2) Max latency 관리하여 test() 구현

1. 자료구조 선언

```
constexpr int MAX_DEVICE = 10005;
int root, id;

int parent[MAX_DEVICE];
int distToParent[MAX_DEVICE];
int depth[MAX_DEVICE];

int firstID[MAX_DEVICE];
int secondID[MAX_DEVICE];
int firstLatency[MAX_DEVICE];
int secondLatency[MAX_DEVICE];

unordered_map<int, int> idmap;
```

- 인접 행렬을 사용하지 않고, 각 노드 별 Max Latency 1순위 노드와 latency, 2순위 노드와 latency를 기록

2. init()

```
void init(int mDevice)
{
    for (int i = 0; i <= id; ++i) {
        firstLatency[i] = secondLatency[i] = 0;
        firstID[i] = secondID[i] = 0;
    }

    root = id = 1;
    idmap.clear();
    idmap[mDevice] = id;
}
```

- 각 노드의 first max latency와 node, second max latency와 node 초기화

최적화 (2) Max latency 관리하여 test() 구현

3. connect()

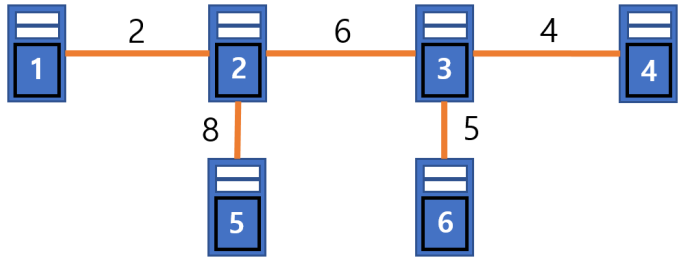
```
void update(int pid, int cid) {
    int distSum = 0;
    while (pid) {
        distSum += distToParent[cid];
①       if (firstID[pid] == cid) {
            if (firstLatency[pid] < distSum)
                firstLatency[pid] = distSum;
            else
                break;
        }
②       else if (firstLatency[pid] < distSum) {
            secondLatency[pid] = firstLatency[pid];
            secondID[pid] = firstID[pid];
            firstLatency[pid] = distSum;
            firstID[pid] = cid;
        }
③       else if (secondLatency[pid] < distSum) {
            secondLatency[pid] = distSum;
            secondID[pid] = cid;
        }
④       else
            break;

⑤       cid = pid;
        pid = parent[pid];
    }
}
```

- connect() 로 새로운 장비를 추가할 때마다 update() 를 호출하여 각 노드의 first max latency와 node, second max latency와 node를 업데이트
- 부모 노드(pid)의 first max latency, first node id, second max latency, second node id 업데이트
  - ① pid의 First ID가 cid 라면, first latency 갱신. 갱신할 내용이 없으면 break;
  - ② pid의 first ID가 cid가 아니고 first max latency가 갱신되는 경우 (즉, First ID가 바뀌는 경우) 이전 first max latency 값을 second may latency 로 변경하고, 새로운 값으로 first max latency 와 first ID 를 업데이트
  - ③ pid의 second may latency가 갱신되는 경우 (즉, second ID가 바뀌는 경우) second max latency와 second ID 값을 업데이트
  - ④ update할 내용이 없다면 break;
  - ⑤ 부모 노드로 이동하며 반복

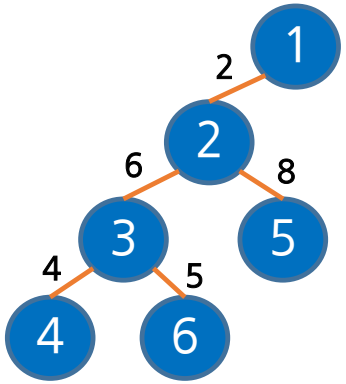
최적화 (2) Max latency 관리하여 test() 구현

3. connect()

```
void connect(int mOldDevice, int mNewDevice, int mLatency)
{
    int old_device_id = idmap[mOldDevice];
    int new_device_id = ++id;

    idmap[mNewDevice] = new_device_id;

    parent[new_device_id] = old_device_id;
    depth[new_device_id] = depth[old_device_id] + 1;
    distToParent[new_device_id] = mLatency;

    update(old_device_id, new_device_id);

}
```



[Fig. 1]



	[0]	[1]	[2]	[3]	[4]	…
firstID	0	2	3	6	0	…
secondID	0	0	5	4	0	…
firstLatency	0	13	11	5	0	…
secondLatency	0	0	8	4	0	…

[Fig. 1] 과 같이 연결되었을 때의 변수 값

4. measure() : 동일. LCA

최적화 (2) Max latency 관리하여 test() 구현

5. test()

```
int test(int mDevice)
{
    int mid = idmap[mDevice];
    int pid = parent[mid];
    int ret = firstLatency[mid] + secondLatency[mid];

    int distSum = firstLatency[mid];
    while (pid) {
        distSum += distToParent[mid];
①       if (secondLatency[pid]) {
            if (firstID[pid] == mid)
                ret = max(ret, distSum + secondLatency[pid]);
            else
                ret = max(ret, distSum + firstLatency[pid]);
        }
②       else {
            ret = max(ret, distSum);
        }

③       mid = pid;
        pid = parent[pid];
    }
    return ret;
}
```

- 모니터링 장비 mid의 자식들 방향으로 first max latency 와 second max latency 를 더한 값으로 초기값 지정
- 부모 노드 (pid)의 자식들의 latency 확인해서 업데이트 (부모 노드의 자식 노드들 중 다른 하나의 max latency 찾기)
  - ① 부모 노드 (pid)의 자식 노드가 2개 이상이라면, 부모 노드의 first ID가 현재 노드인지 확인. 현재노드라면 second max latency값을 더해주고, 아니라면 first max latency를 더해준 후 초기값 (ret)과 비교
  - ② 부모 노드 (pid) 자식 노드가 1개 뿐이라면 현재 결과로 비교 후 ret값 지정
  - ③ 부모 노드 방향으로 탐색

요약

- 실행시간 비교

DFS로 구현 + 최적화(1) LCA 를 사용하여 measure() 구현 + 최적화 (2) Max latency 관리하여 test() 구현	C++ 언어	13,960kb 메모리	42ms 시간	4,121B 코드길이	Pass 결과
DFS로 구현 + 최적화(1) LCA 를 사용하여 measure() 구현	C++ 언어	14,308kb 메모리	229ms 시간	2,768B 코드길이	Pass 결과
DFS로 구현	C++ 언어	14,188kb 메모리	367ms 시간	2,214B 코드길이	Pass 결과

- 함수 호출 수, 최대 노드 수를 고려했을 때 시간복잡도를 계산해보면 단순 DFS 또는 BFS로도 풀이할 수 있었던 문제
- 하지만, 우리는 항상 최적화하는 연습 필요!! 문제에서 주어진 제약사항을 활용해서 최적화 가능
- 노드 별 정보를 배열로 표현하는 방법 외에도 각 노드 별 정보를 하나의 노드로 저장하는 구현 방법 (struct, class..) 도 있음
- 다양한 방법으로 풀 수 있었던 문제로 BFS로도 풀어보세요! ☺