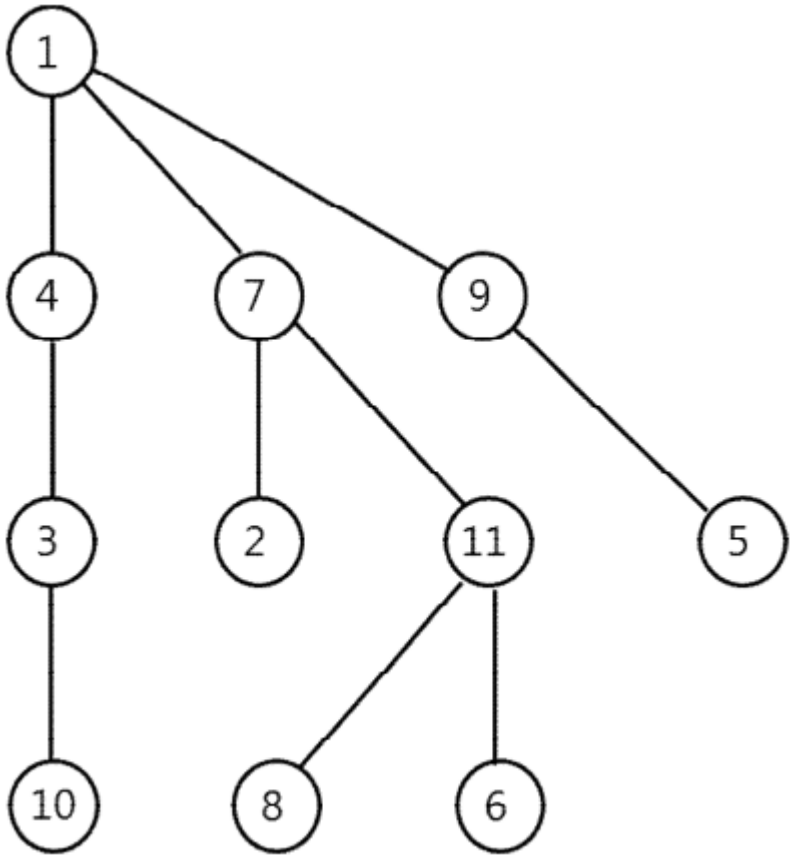
06. Tree

2019 Summer / 20141574 임지환

Tree

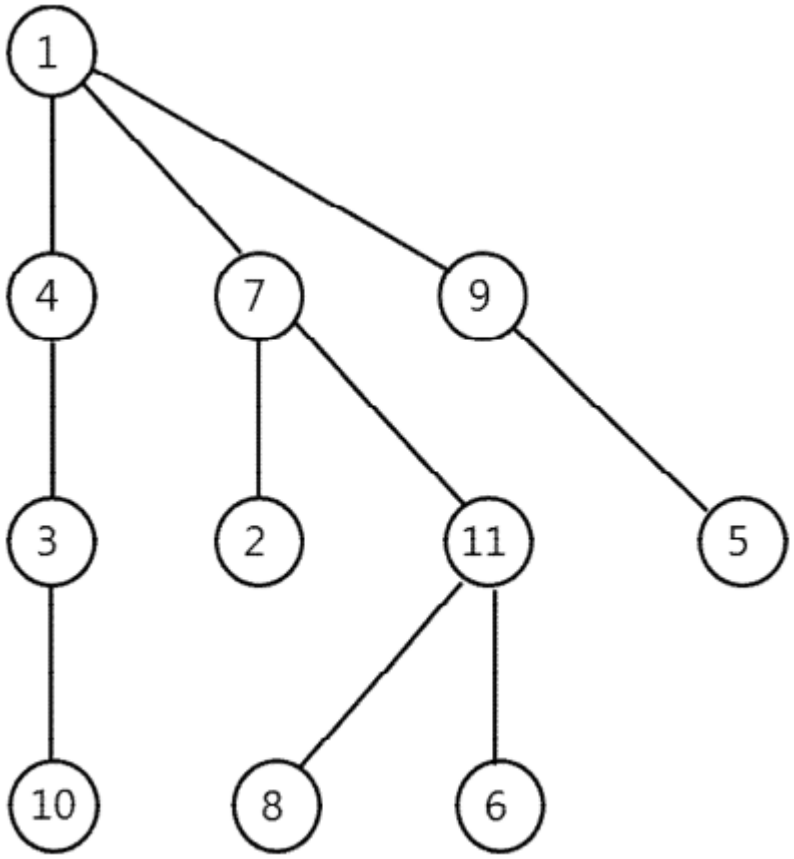
Tree

- kind of Graph,
- 각 노드들이 서로 다른 자식을 가짐
- 노드 순서쌍 (u,v)에 대해 path(u,v)가 유일
- 사이클을 가지는 노드 집합x
- 반드시 하나의 root가 존재



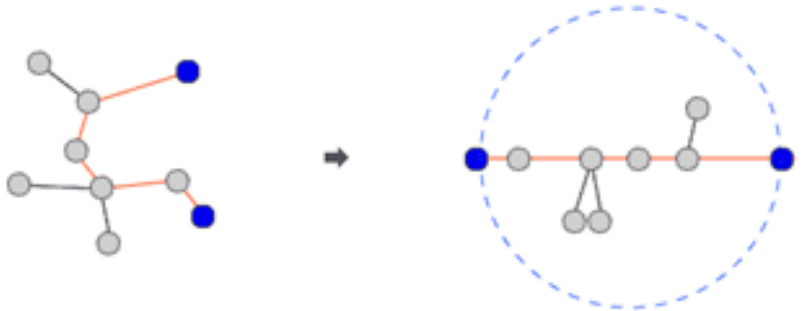
Tree

- kind of Graph,
- 각 노드들이 서로 다른 자식을 가짐
- 노드 순서쌍 (u,v)에 대해 path(u,v)가 유일
- 사이클을 가지는 노드 집합x
- 반드시 하나의 root가 존재



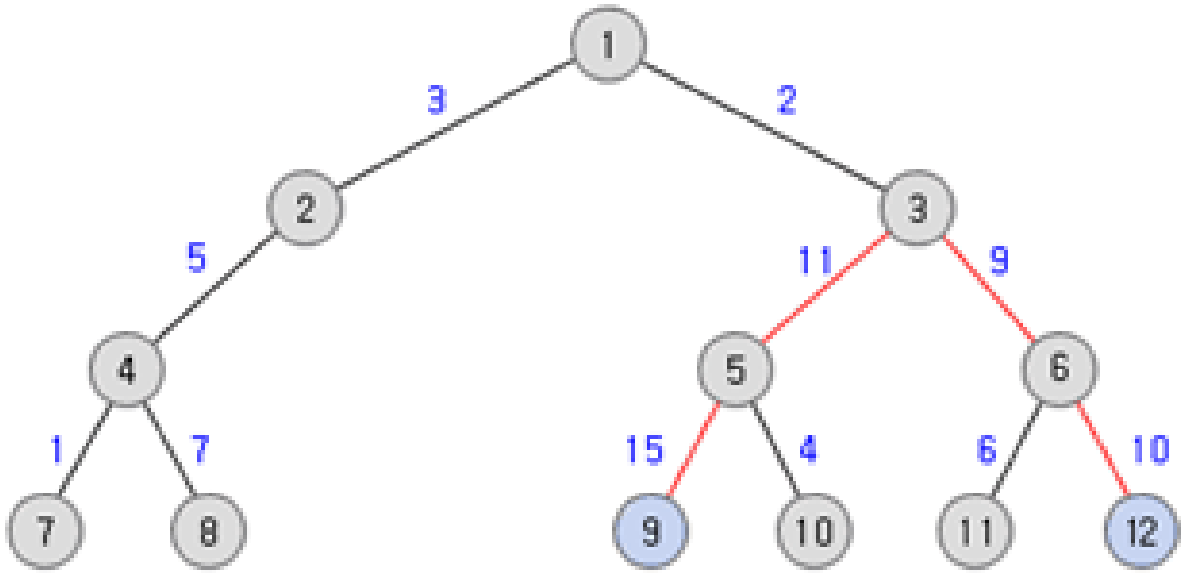
#1967 트리의 지름 3

트리(tree)는 사이클이 없는 무방향 그래프이다. 트리에서는 어떤 두 노드를 선택해도 둘 사이에 경로가 항상 하나만 존재하게 된다. 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 지름이라고 한다. 정확히 정의하자면 트리에 존재하는 모든 경로들 중에서 가장 긴 것의 길이를 말한다.

#1967 트리의 지름 3



#1967 트리의 지름 

1. 트리에서 임의의 정점 x 설정
2. 정점 x에서 가장 먼 정점 y 찾기
3. 정점 y에서 가장 먼 정점 z 찾기

트리의 지름 : $path(y, z)$

pf) https://blog.myungwoo.kr/112

```
8       vector<ii> adj[10001];
9
10      void dfs(int cur, int w, int prev = -1) {
11          if (ans < w) {
12              ans = w, x = cur;
13          }
14          for (ii it : adj[cur]) {
15              int next = it.first, d = it.second;
16              if (next == prev) continue;
17              dfs(next, w + d, cur);
18          }
19      }
20
```

11) ans : root부터 가장 멀리 떨어져 있는 점까지의 거리

12) x : ans를 만족하는 점

```
21  int main() {
22      ios_base::sync_with_stdio(false);
23      cin.tie(0);
24      cin >> N;
25      for (int i = 1; i<N; i++) {
26          int u, v, w;
27          cin >> u >> v >> w;
28          adj[u].push_back(ii(v, w));
29          adj[v].push_back(ii(u, w));
30      }
31      dfs(1, 0); ans = 0; dfs(x, 0);
32      cout << ans;
33  }
34
```

31) 최초 임의의 점을 1로 설정
1과 가장 멀리 떨어진 지점 x를
기준으로 dfs

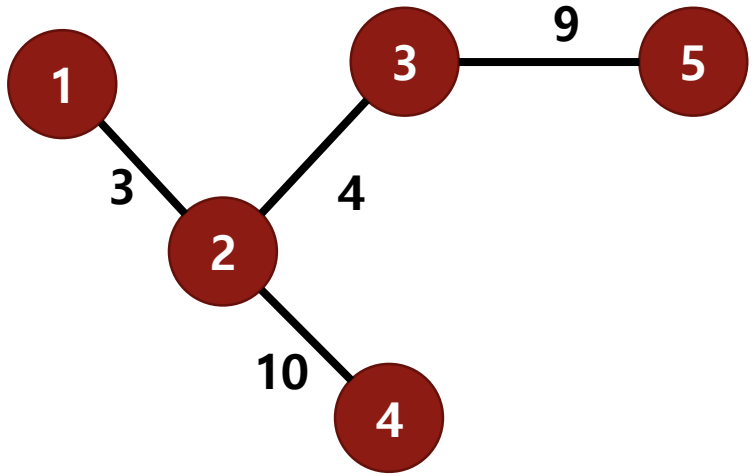
#13016 내 왼손에는 흑염룡이 잠들어 있다

- $N(\leq 50{,}000)$개의 노드가 트리 형태로 만들어져 있을 때,

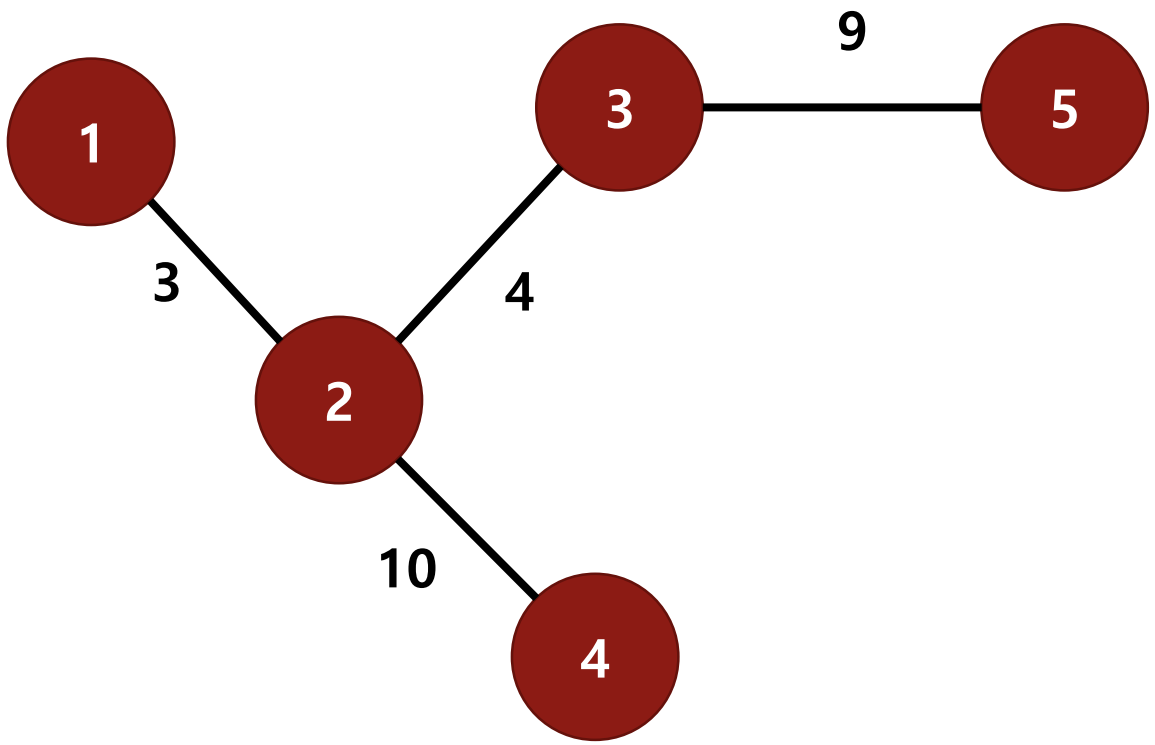
  모든 $i(1 \leq i \leq N)$에서 가장 멀리 떨어진 지점까지의 거리를 구하여라.

예시)

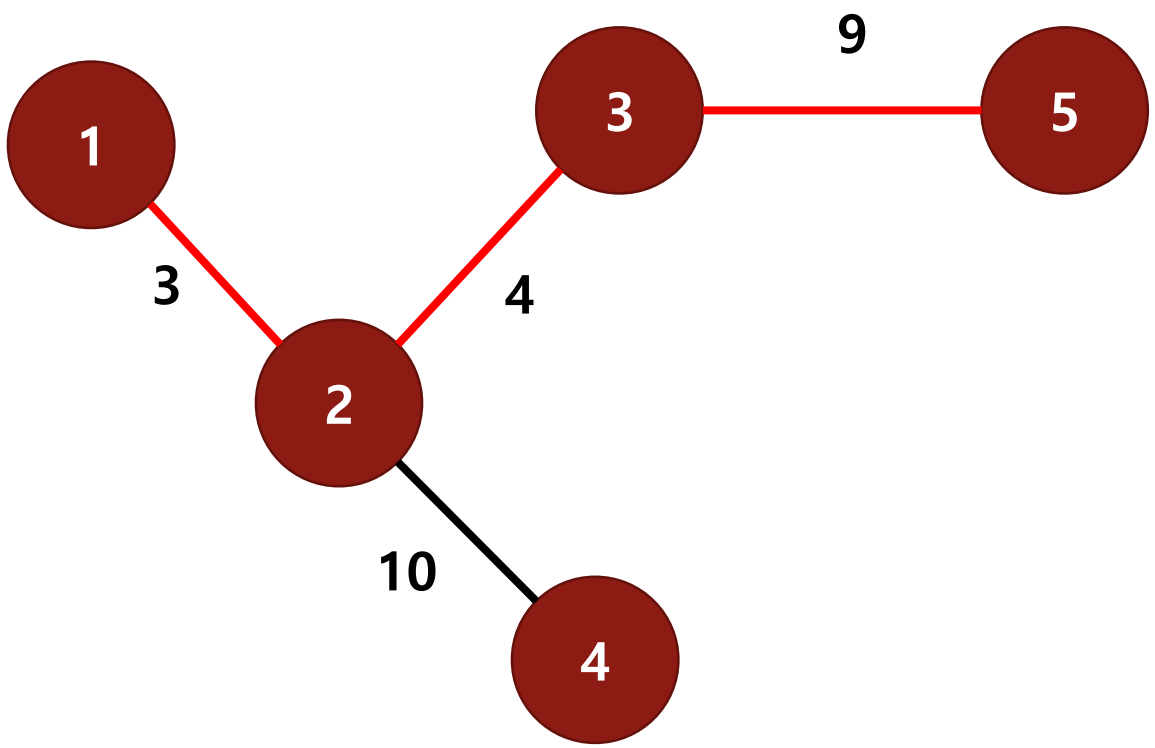
```
5
2 1 3
2 4 10
2 3 4
3 5 9
```



#13016 내 왼손에는 흑염룡이 잠들어 있다

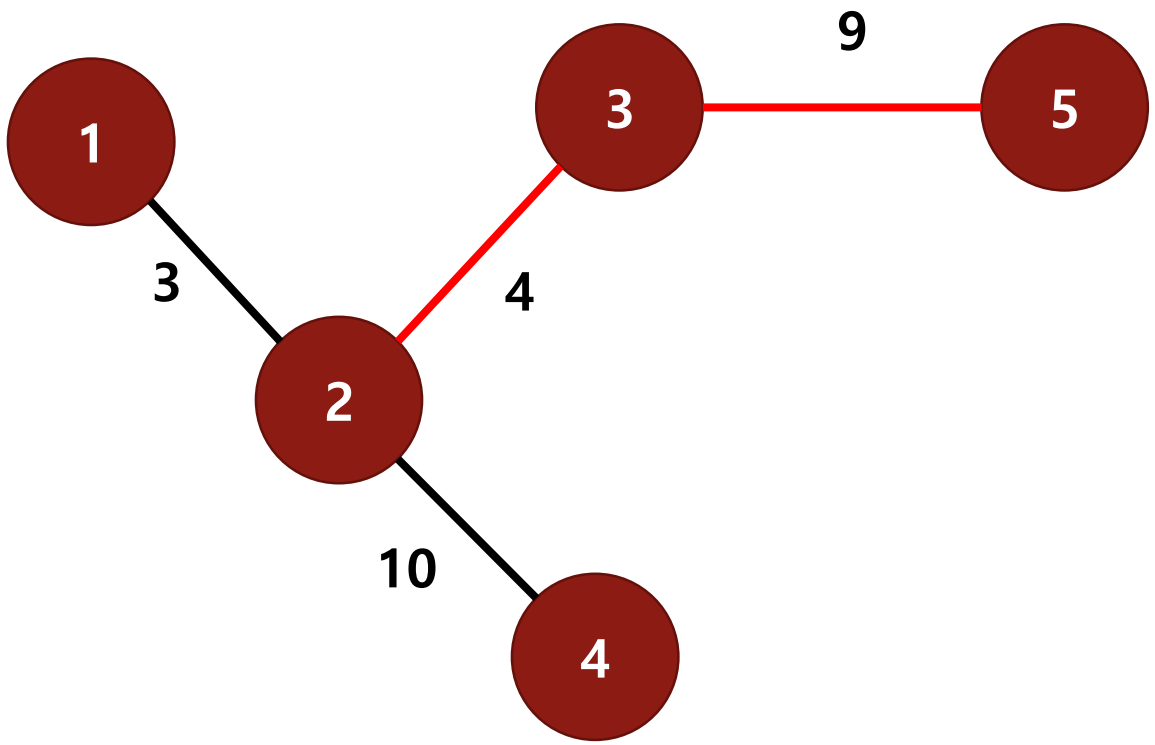


#13016 내 왼손에는 흑염룡이 잠들어 있다



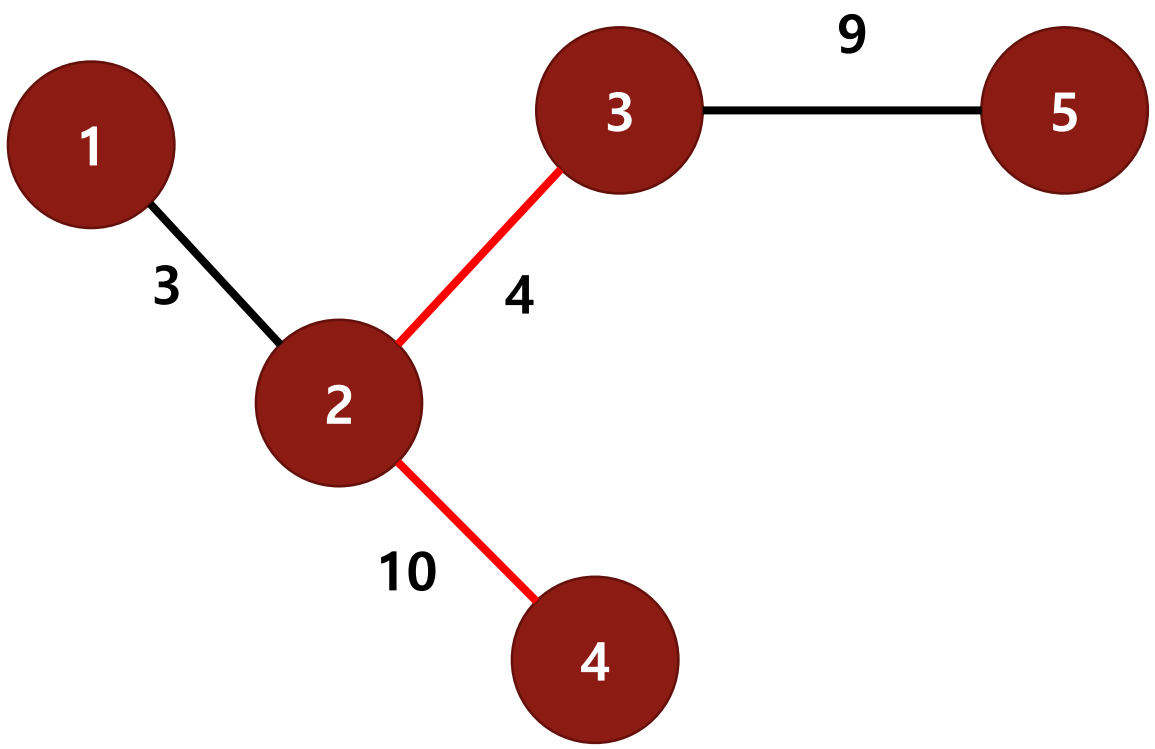
dist[1] = 16

#13016 내 왼손에는 흑염룡이 잠들어 있다



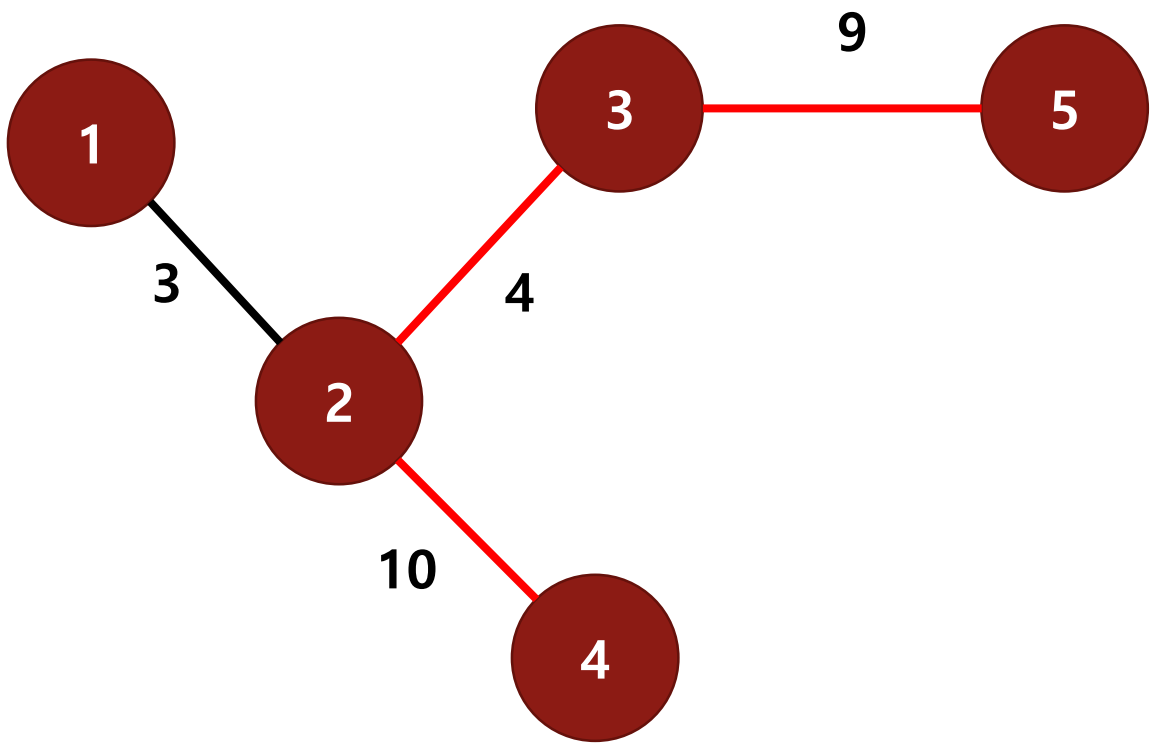
dist[1] = 16
dist[2] = 13

#13016 내 왼손에는 흑염룡이 잠들어 있다



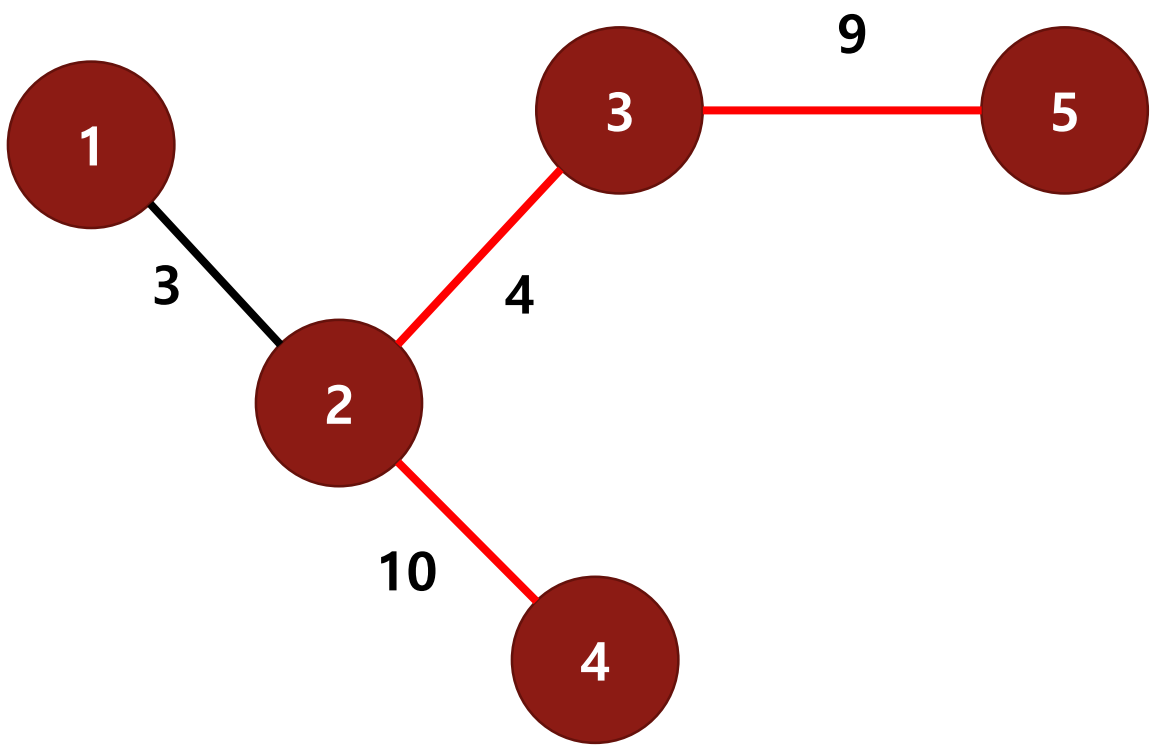
dist[1] = 16
dist[2] = 13
dist[3] = 14

#13016 내 왼손에는 흑염룡이 잠들어 있다



dist[1] = 16
dist[2] = 13
dist[3] = 14
dist[4] = 23

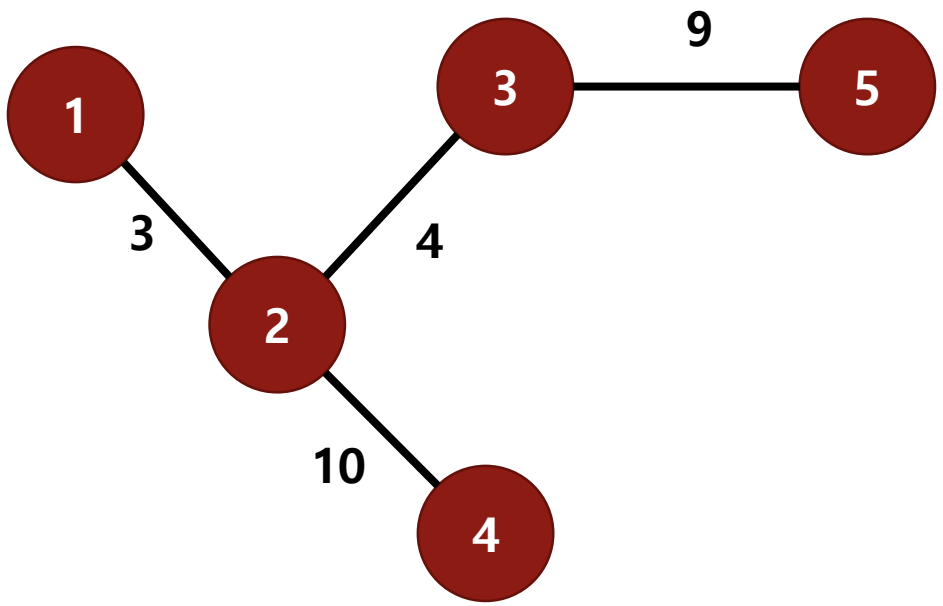
#13016 내 왼손에는 흑염룡이 잠들어 있다



dist[1] = 16
dist[2] = 13
dist[3] = 14
dist[4] = 23
dist[5] = 23

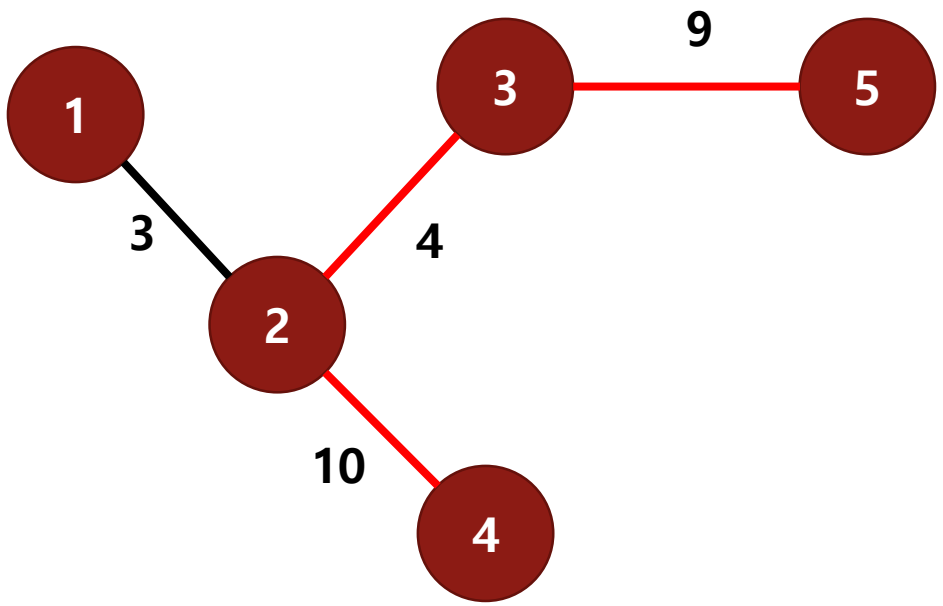
#13016 내 왼손에는 흑염룡이 잠들어 있다

- 트리의 지름 : 23 ($path(4,5)$)



#13016 내 왼손에는 흑염룡이 잠들어 있다

- 트리의 지름 : 23 ($path(4,5)$)



- 트리의 지름 상에 있는 정점

  ⇒2, 3, 4, 5

- 트리의 지름 상에 있지 않은 정점

  ⇒1

#13016 내 왼손에는 흑염룡이 잠들어 있다

- 트리의 지름 상에 있는 정점

1. 트리의 지름 양 끝에 있는 정점

#13016 내 왼손에는 흑염룡이 잠들어 있다

- 트리의 지름 상에 있는 정점

1. 트리의 지름 양 끝에 있는 정점

   ⇒반대쪽 끝점까지의 거리

#13016 내 왼손에는 흑염룡이 잠들어 있다

- 트리의 지름 상에 있는 정점

1. 트리의 지름 양 끝에 있는 정점

   ⇒반대쪽 끝점까지의 거리

2. 양 끝이 아닌 정점

#13016 내 왼손에는 흑염룡이 잠들어 있다

- 트리의 지름 상에 있는 정점

1. 트리의 지름 양 끝에 있는 정점

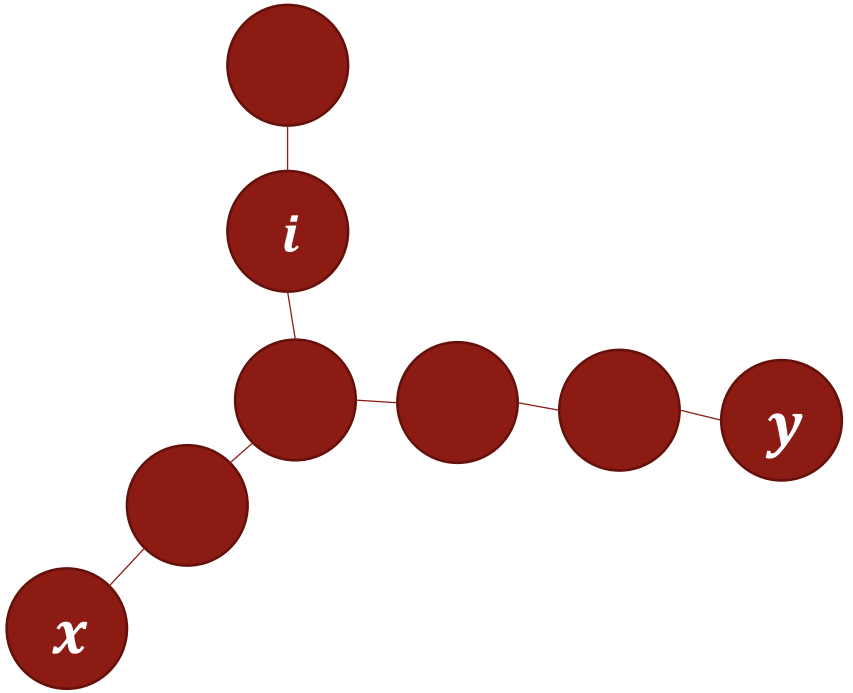
   ⇒반대쪽 끝점까지의 거리

2. 양 끝이 아닌 정점

   ⇒양 끝점까지 거리 중 최대

#13016 내 왼손에는 흑염룡이 잠들어 있다

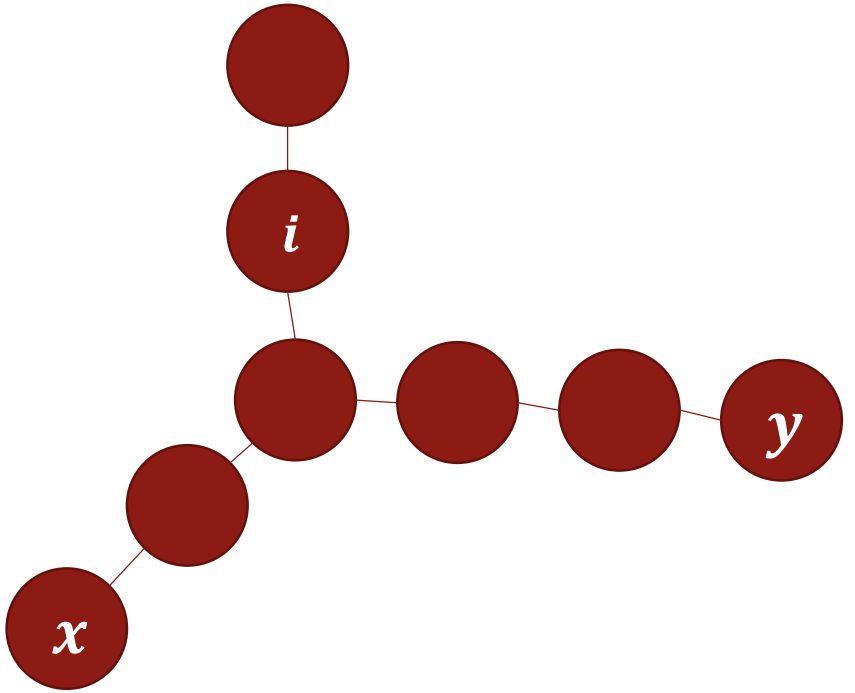
- 트리의 지름 상에 있지 않은 정점



#13016 내 왼손에는 흑염룡이 잠들어 있다

- 트리의 지름 상에 있지 않은 정점

  $\Rightarrow path(i,x), path(i,y)$ 중 큰 값



```
 8      vector<ii> adj[mxn];
 9      vector<int> xdist, ydist;
10      int n;
11      int Max, x, y;
12
13      void dfs(int curr, int prev, int dist) {
14          xdist[curr] = dist;
15          if (Max < dist) {
16              Max = dist; x = curr;
17          }
18          for (auto it : adj[curr]) {
19              int next = it.first, w = it.second;
20              if (next != prev) dfs(next, curr, w + dist);
21          }
22      }
23
```

14) xdist[i] : 지름의 양 끝점 중 한 곳에서 i까지의 거리

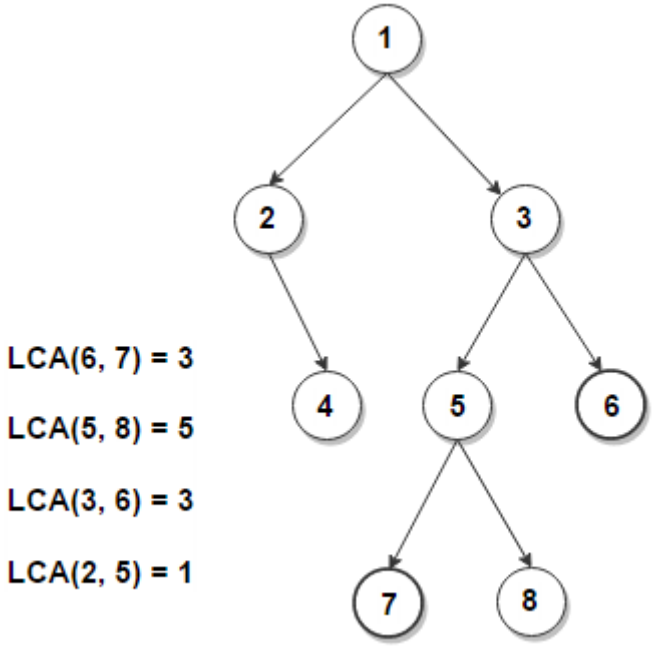
```cpp
int main() {
    cin >> n;
    xdist.resize(n + 1);
    for (int i = 1; i < n; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        adj[u].push_back(ii(v, w));
        adj[v].push_back(ii(u, w));
    }

    dfs(1, 0, 0);
    y = x;
    dfs(y, 0, 0);
    ydist = xdist;
    dfs(x, 0, 0);
    for (int i = 1; i <= n; i++)
        cout << max(ydist[i], xdist[i]) << '\n';
    return 0;
}
```

35) y : 1에서 가장 멀리 떨어진 점
37) ydist[i] : y에서 i까지의 거리

최소 공통 조상

- LCA (Lowest Common Ancestor)

루트가 정해진 트리에서 두 노드 u, v를 모두 자손으로 갖는 노드(조상) 중 가장 아래 있는 노드



#11438 LCA 2

- 루트가 1이고 $N\,(2 \le N \le 100{,}000)$개의 정점으로 이루어진 트리에서 두 노드의 쌍(u, v)가 $M\,(1 \le M \le 100{,}000)$개가 주어졌을 때 $LCA(u, v)$를 구하여라.

#11438 LCA 2

- LCA를 구하는 알고리즘
  1. Segment tree를 통한 range minimum query
  2. Sparse table 구성을 통한 path compression

#11438 LCA 2

- LCA를 구하는 알고리즘
  1. Segment tree를 통한 range minimum query
  2. Sparse table 구성을 통한 path compression

Sparse Table

- 구간 쿼리 처리를 $O(1)$ 시간에 할 수 있게 하는 자료구조

1. Static data
2. Associativity

Sparse Table

- 기본 형태

$Table[i][j] :=\ i \sim i + 2^j - 1$번째 까지의 값

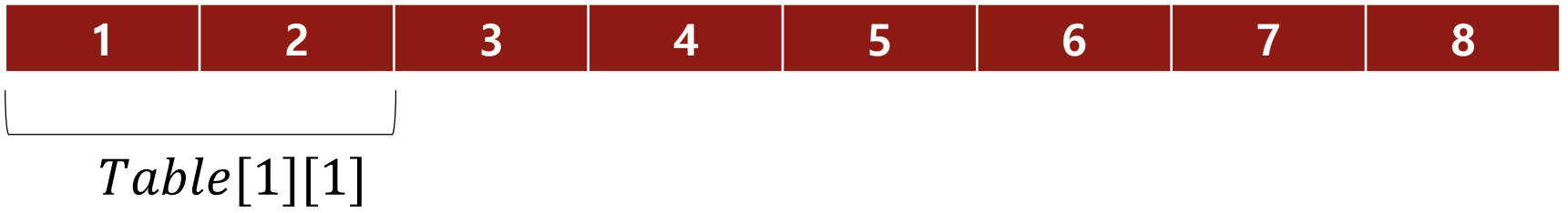
$Table[i][j] = f(arr[i], arr[i+1], \ldots, arr[i + 2^j - 1])$

Sparse Table

- 기본 형태

$Table[i][j] :=$ $i \sim i + 2^j - 1$번째 까지의 값

$Table[i][j] = f(arr[i], arr[i+1], \ldots, arr[i + 2^j - 1])$

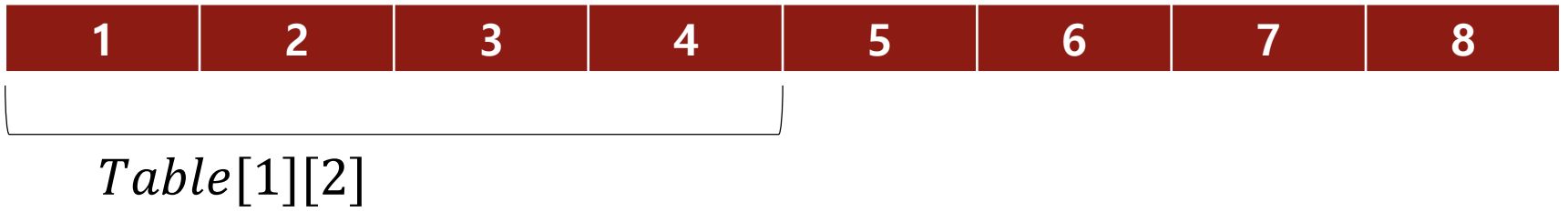


Sparse Table

- 기본 형태

$Table[i][j] :=$ $i \sim i + 2^j - 1$번째 까지의 값

$Table[i][j] = f(arr[i], arr[i+1], \ldots, arr[i + 2^j - 1])$

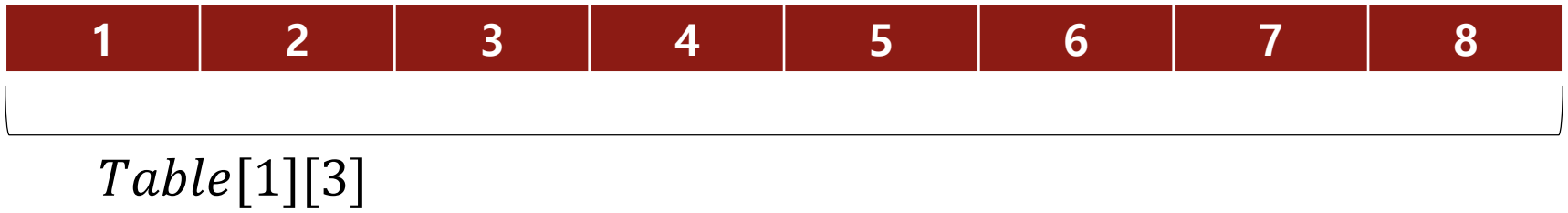


Sparse Table

- 기본 형태

$Table[i][j] :=$ $i \sim i + 2^j - 1$번째 까지의 값

$Table[i][j] = f(arr[i], arr[i+1], \ldots, arr[i + 2^j - 1])$



Sparse Table

$$Table[i][j] = f(arr[i], arr[i+1], \ldots, arr[i+2^j-1])$$

Sparse Table

$$Table[i][j] = f(arr[i], arr[i+1], \ldots, arr[i+2^j-1])$$

$$Table[i][j-1] = f(arr[i], arr[i+1], \ldots, arr[i+2^{j-1}-1])$$

Sparse Table

$$Table[i][j] = f(arr[i], arr[i+1], \ldots, arr[i+2^j-1])$$

$$Table[i][j-1] = f(arr[i], arr[i+1], \ldots, arr[i+2^{j-1}-1])$$

$$Table[i+2^{j-1}][j-1] = f(arr[i+2^{j-1}], arr[i+2^{j-1}+1], \ldots, arr[i+2^j-1])$$

Sparse Table

$$Table[i][j] = f(arr[i], arr[i+1], \ldots, arr[i+2^j-1])$$

$$Table[i][j-1] = f(arr[i], arr[i+1], \ldots, arr[i+2^{j-1}-1])$$

$$Table[i+2^{j-1}][j-1] = f(arr[i+2^{j-1}], arr[i+2^{j-1}+1], \ldots, arr[i+2^j-1])$$

by associativity,

$$Table[i][j] = f(Table[i][j-1], Table[i+2^{j-1}][j-1])$$

#11438 LCA 2

- Sparse Table을 LCA에 적용한다면?

$parent[i][j] :=$ i번 노드의 2^j번째 조상

$$parent[i][j] = parent\Big[\, parent[i][j-1]\Big][j-1]$$

#11438 LCA 2

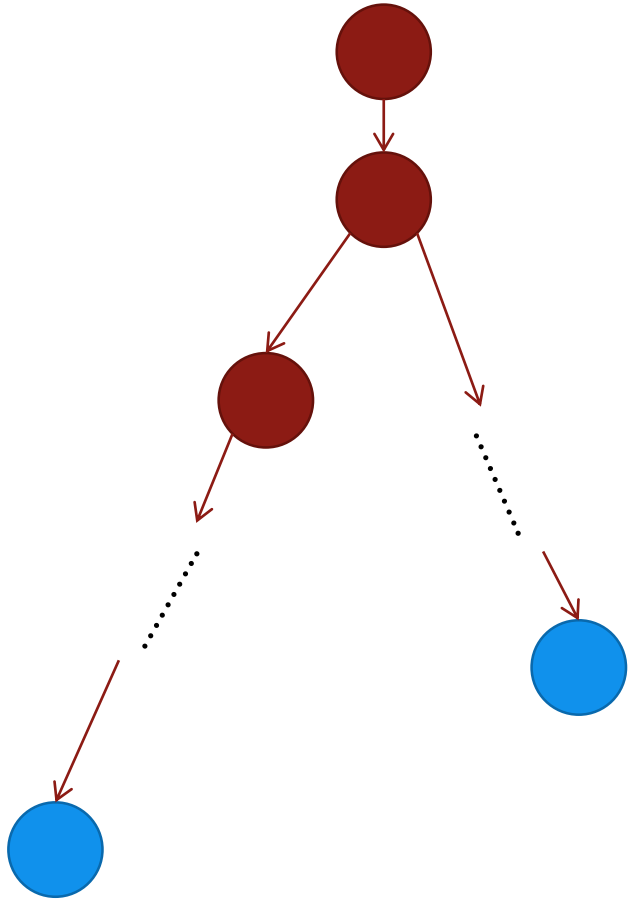
- 노드 바로 위의 부모는 어떻게..?

  $\Rightarrow$ dfs를 통한 순서 탐색, 특정 노드의 깊이 구하기

  $\Rightarrow parent[child][0] = par$

#11438 LCA 2

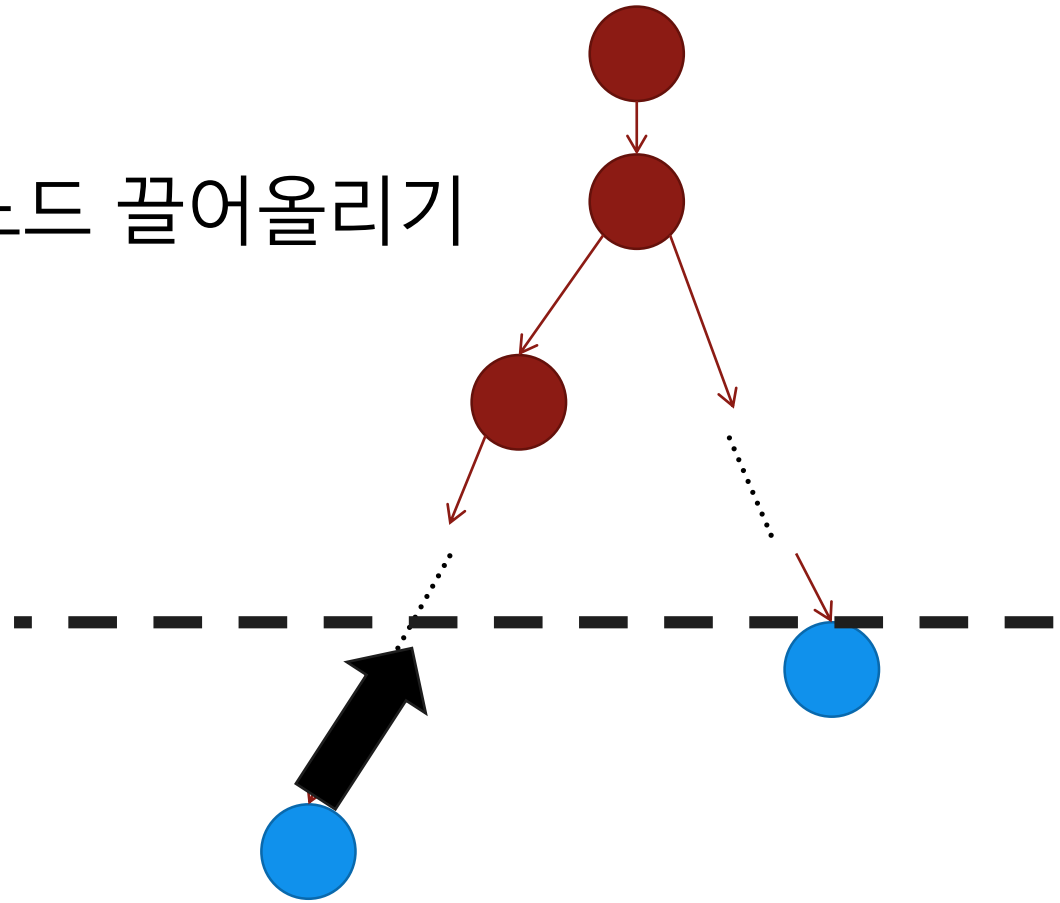
- LCA 구하기



#11438 LCA 2

- LCA 구하기

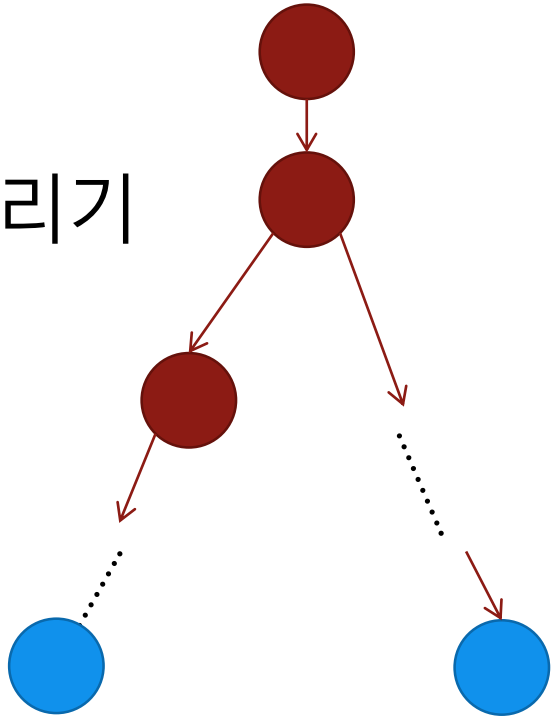
1. 깊이가 낮은 노드 끌어올리기



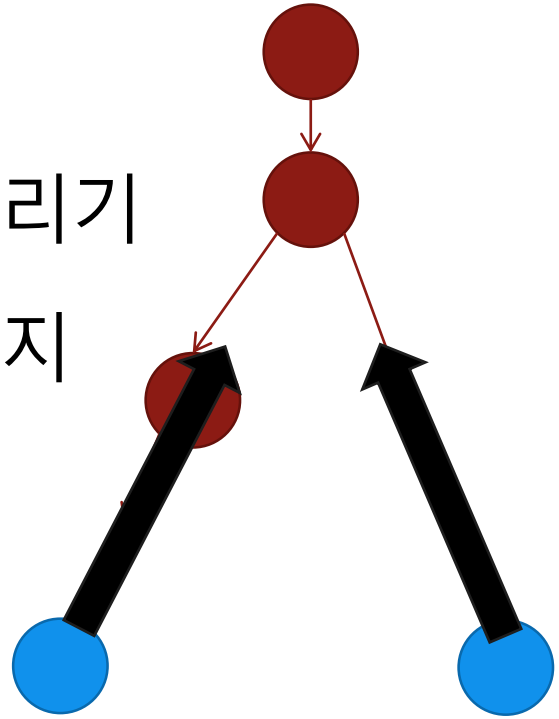
#11438 LCA 2

- LCA 구하기

1. 깊이가 낮은 노드 끌어올리기



#11438 LCA 2



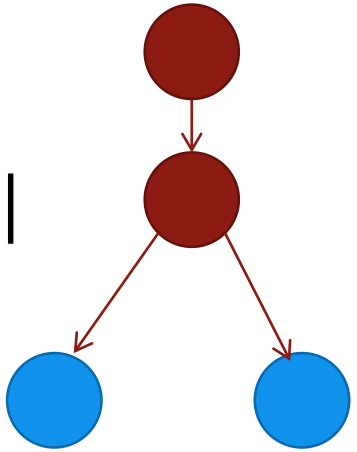
- LCA 구하기

1. 깊이가 낮은 노드 끌어올리기
2. 직계 부모가 같아질 때까지
   두 노드 끌어올리기

#11438 LCA 2

- LCA 구하기

1. 깊이가 낮은 노드 끌어올리기
2. 직계 부모가 같아질 때까지
   두 노드 끌어올리기

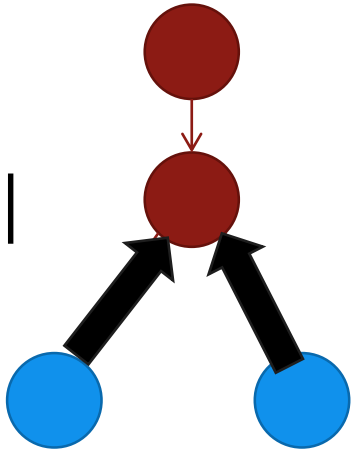


#11438 LCA 2

- LCA 구하기

1. 깊이가 낮은 노드 끌어올리기
2. 직계 부모가 같아질 때까지

   두 노드 끌어올리기
3. return parent[u][0]



#11438 LCA 2

1. dfs를 통한 각 노드의 깊이, 2^0번째 조상 구하기
2. sparse table 채우기
3. LCA 구하기

```
13  void get_first_par(int curr, int depth) {
14      vis[curr] = true;
15      depth_of[curr] = depth;
16
17      maxdepth = max(maxdepth, depth);
18
19      for (int next : adj_list[curr]) {
20          if (!vis[next]) {
21              par[next][0] = curr;
22              get_first_par(next, depth + 1);
23          }
24      }
25  }
26
```

15) depth_of[i] := i번째 노드의 깊이
depth_of[root] = 0

21) par[next][0] := next번째 노드의 1번째 부모

```
27  void get_every_par() {
28      while (maxdepth) {
29          maxexpon++;
30          maxdepth /= 2;
31      }
32
33      for (int i = 1; i < maxexpon; i++) {
34          for (int j = 1; j <= No_of_v; j++)
35              par[j][i] = par[par[j][i - 1]][i - 1];
36      }
37  }
38
```

indexing error 주의!

```
39  int LCA(int u, int v) {
40      if (depth_of[u] < depth_of[v])
41          swap(u, v);
42      for (int i = maxexpon; i >= 0; i--) {
43          if (depth_of[u] - depth_of[v] >= (1 << i))
44              u = par[u][i];
45      }
46
47      if (u == v) return u;
48      for (int i = maxexpon; i >= 0; i--) {
49          if (par[u][i] != par[v][i]) {
50              u = par[u][i];
51              v = par[v][i];
52          }
53      }
54      return par[u][0];
55  }
56
```

40) 노드 u가 항상 깊도록 고정

42~45) 깊이가 깊은 노드를 v의 깊이 까지 끌어올리기

48~53) u와 v의 부모가 같아질 때 까지 끌어 올리기

Tree Dynamic Programming

- 기본적으로 Top-Down 방식으로 전개

  ⇒ dfs를 통해 Leaf node까지 순회한 뒤 그 값들을 통해 상위 값 도출

  ⇒재귀적인 형태로 정의하는 것이 용이

#2533 사회망 서비스(SNS)

- $N\,(2 \le N \le 1{,}000{,}000)$개의 노드
- early adaptor가 아닌 노드는 인접한 모든 노드가 early adaptor여야 함
- 모든 노드가 새로운 아이디어를 수용하도록 하는 최소 early adaptor의 수를 구하여라.

#2533 사회망 서비스(SNS)

- 편의상 early adaptor인 경우를 1, 아닌 경우를 0으로 표현

#2533 사회망 서비스(SNS)

- 편의상 early adaptor인 경우를 1, 아닌 경우를 0으로 표현

1. 특정 노드가 0이라면

#2533 사회망 서비스(SNS)

- 편의상 early adaptor인 경우를 1, 아닌 경우를 0으로 표현

1. 특정 노드가 0이라면

   ⇒자식노드들은 반드시 1이 되어야 함

#2533 사회망 서비스(SNS)

- 편의상 early adaptor인 경우를 1, 아닌 경우를 0으로 표현

1. 특정 노드가 0이라면

   ⇒자식노드들은 반드시 1이 되어야 함

2. 특정 노드가 1이라면

#2533 사회망 서비스(SNS)

- 편의상 early adaptor인 경우를 1, 아닌 경우를 0으로 표현

1. 특정 노드가 0이라면

   ⇒자식노드들은 반드시 1이 되어야 함

2. 특정 노드가 1이라면

   ⇒자식노드들은 0이든 1이든 상관 없음

#2533 사회망 서비스(SNS)

- $dp[i][0] :=$ i번째 노드가 0일 때 i를 루트로 하는 서브트리에서의 최소 early adaptor의 수
- $dp[i][1] :=$ i번째 노드가 1일 때 i를 루트로 하는 서브트리에서의 최소 early adaptor의 수

```
11  void dfs(int curr) {
12      vis[curr] = true;
13      dp[curr][0] = 0;
14      dp[curr][1] = 1;
15      for (int next : g[curr]) {
16          if (vis[next]) continue;
17          dfs(next);
18          dp[curr][0] += dp[next][1];
19          dp[curr][1] += min(dp[next][0], dp[next][1]);
20      }
21  }
22
```

17) 자식노드들에 대한 정보를 먼저 처리한 후 상위노드에 대해 처리

18) current 노드가 0이라면 자식노드들은 반드시 1이 되어야 함

19) current 노드가 1이라면 자식노드들은 0이든 1이든 상관없음

Problem Set

- 3 1967 트리의 지름
- 5 13016 내 왼손에는 흑염룡이 잠들어 있다
- 2 8872 빌라봉
- 1 11438 LCA 2
- 1 1761 정점들의 거리
- 4 3176 도로 네트워크
- 3 13511 트리와 쿼리 2
- 1 2533 사회망 서비스 (SNS)
- 5 10273 고대 동굴 탐사
- 3 1289 트리의 가중치
- 1 9013 Parents

ㅎㅎ