Heavy-Light Decompositoin

천민호 kesakiyo@naver.com

HLD란?

HLD란?

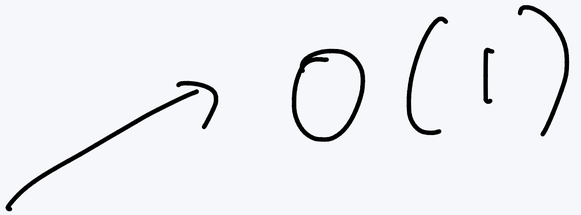
Heavy-Light Decomposition

- 최근 프로그래밍 대회에 자주 출제되고 있는 트리 알고리즘이다.
- 트리를 적절히 분해해 특정 쿼리를 빠르게 해결할 수 있게 도와준다.
- 실제 문제를 통해 HLD를 배워보자.

트리와 쿼리1

https://www.acmicpc.net/problem/13510

- 크기가 N인 트리가 주어진다.
- 각각의 간선에는 가중치가 있다.
- 이 때 두 종류의 쿼리가 들어온다.
- 1 i c : i번 간선의 비용을 c로 바꾼다.
- 2 u v : u에서 v로 가는 경로에 존재하는 비용 중에서 가장 큰 것을 출력한다.



(DFS or BFS)

트리와 쿼리1

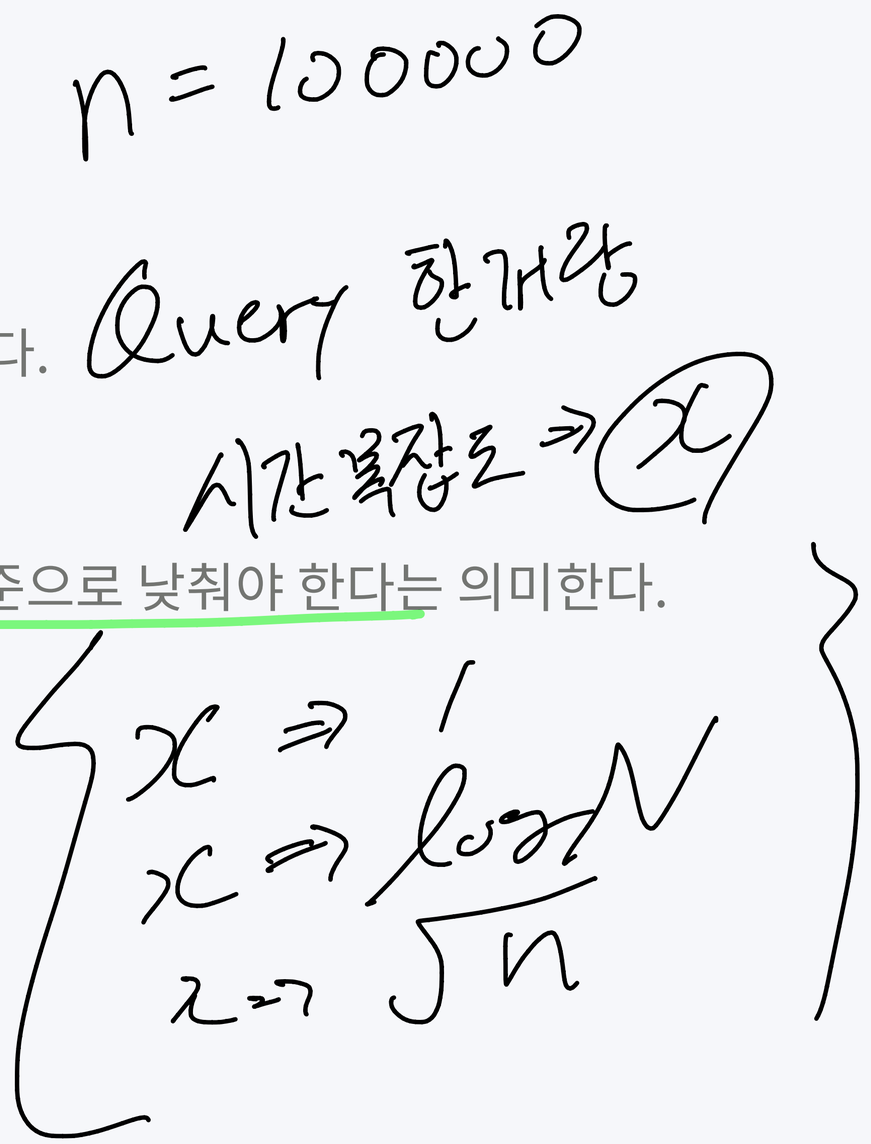
https://www.acmicpc.net/problem/13510

- 알고리즘을 막 공부하기 시작한 사람에게는 너무 쉬운 문제이다.
- 단순하게 BFS혹은 DFS로 문제를 해결할 수 있다.

트리와 쿼리1

https://www.acmicpc.net/problem/13510

- 이 때 쿼리 하나의 시간복잡도는 O(N + M)이다.
- 하지만 문제는 쿼리의 수가 10만개!!
- 총 시간복잡도는 O(Q(N + M))으로 약 100억이다.
- TL나기 딱 좋은 시간복잡도다.
- 즉 쿼리 하나의 시간복잡도를 log 혹은 root 수준으로 낮춰야 한다는 의미한다.

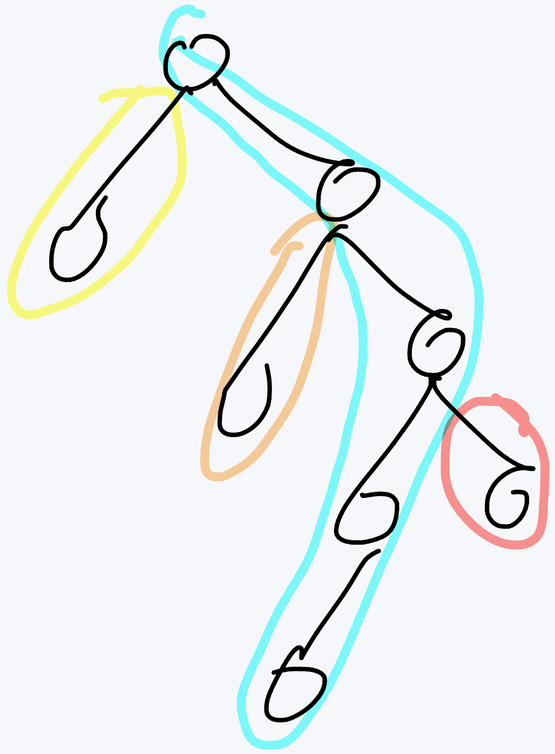


트리의 분해

트리의 분해

Heavy-Light Decomposition

- 만약 트리를 여러 개의 1차원 선분으로 분해할 수 있다면?
- 분해된 1차원 선분의 개수가 O(logN)이라면?
- 한 선분에서 최대값을 찾는 작업의 시간복잡도는 O(logN)이다.
- 선분의 개수가 O(logN)이니 쿼리 하나의 시간복잡도는 O(logN * logN)이다.
- 쿼리의 수까지 같이 계산해봐도 O(Q * logN * logN)이다.
- 그렇다면 어떻게 트리를 분해할 수 있을까?

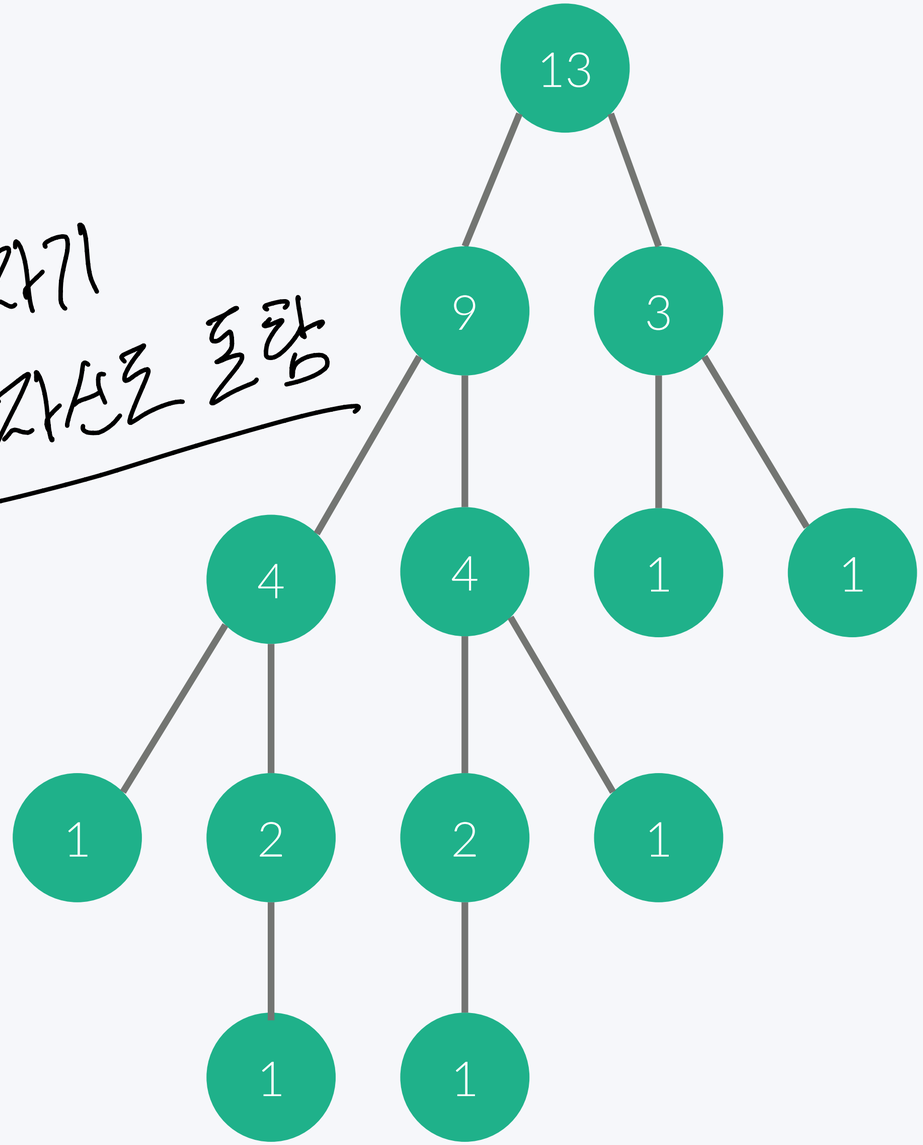


간선의 분류

Heavy-Light Decomposition

- 모든 간선은 무거운 간선과 가벼운 간선 두 종류로 분류할 수 있다.
- 한 정점의 무게를 해당 정점을 루트로 하는 서브트리에 속한 정점의 수라고 하자.
- 오른쪽 트리에서 정점의 숫자는 각 정점의 무게를 의미한다.

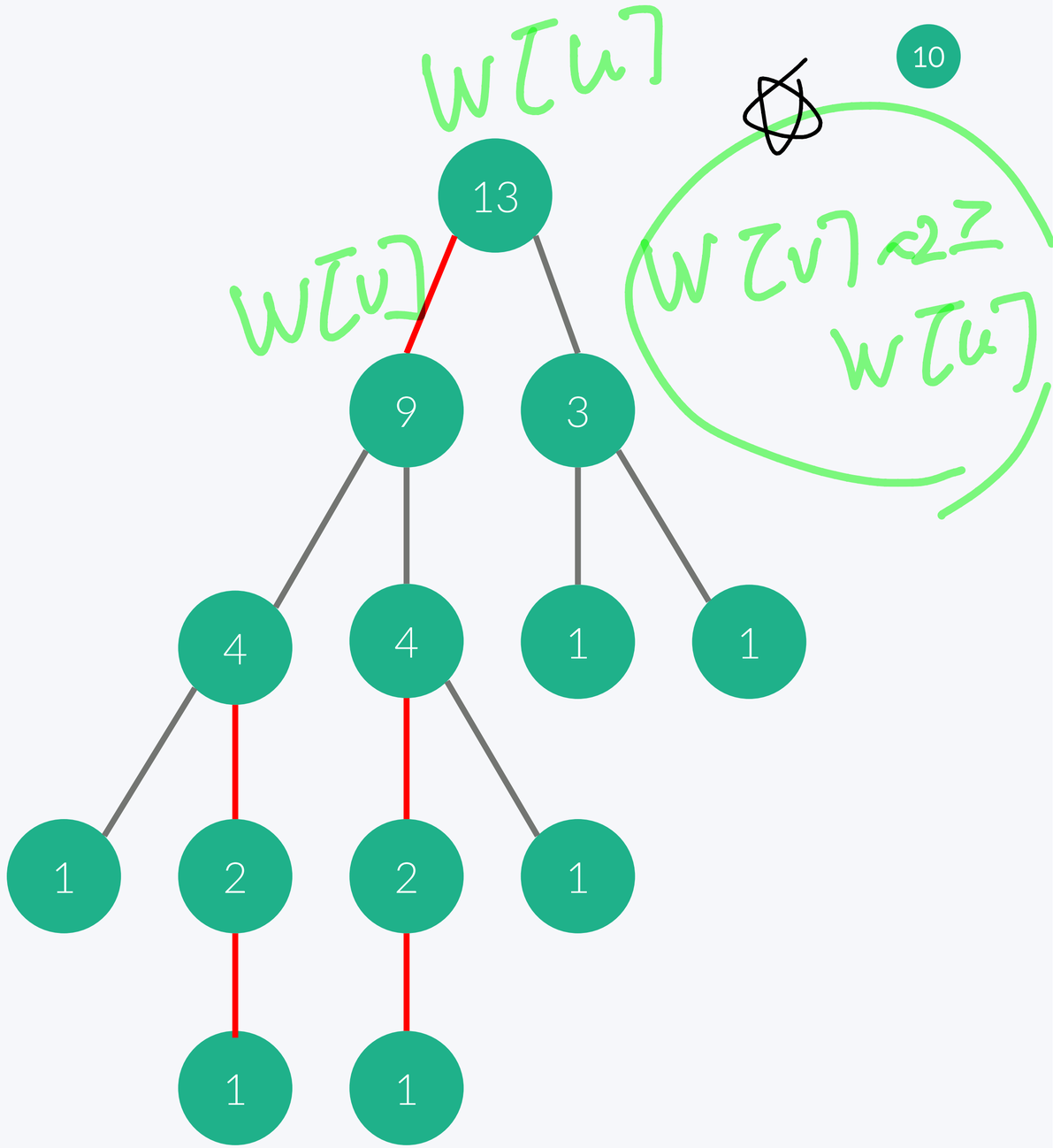
자기 자신도 포함



간선의 분류

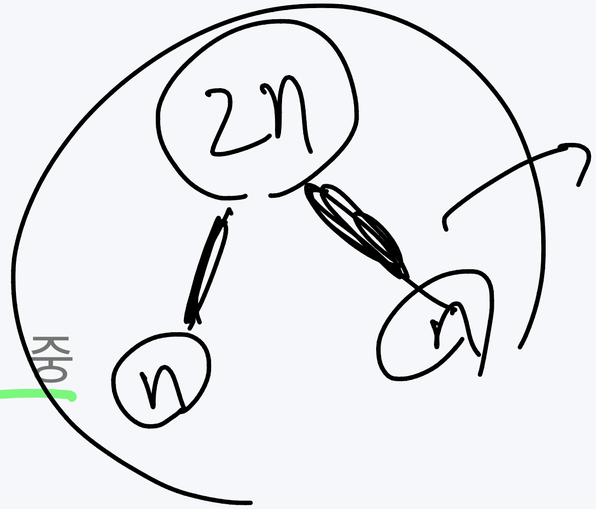
Heavy-Light Decomposition

- 한 정점 u의 자식들 중 하나인 v로 가는 간선이 있을 때 v의 무게가 u의 무게보다 절반 이하로 줄어든다면 무거운 간선이다.
- 이 경우가 아니라면 가벼운 간선이다.
- 오른쪽 트리에서 붉은 간선은 무거운 간선이고 회색 간선은 가벼운 간선이다.
- 이렇게 분류된 간선들을 보면 몇 가지 사실을 알 수 있다.



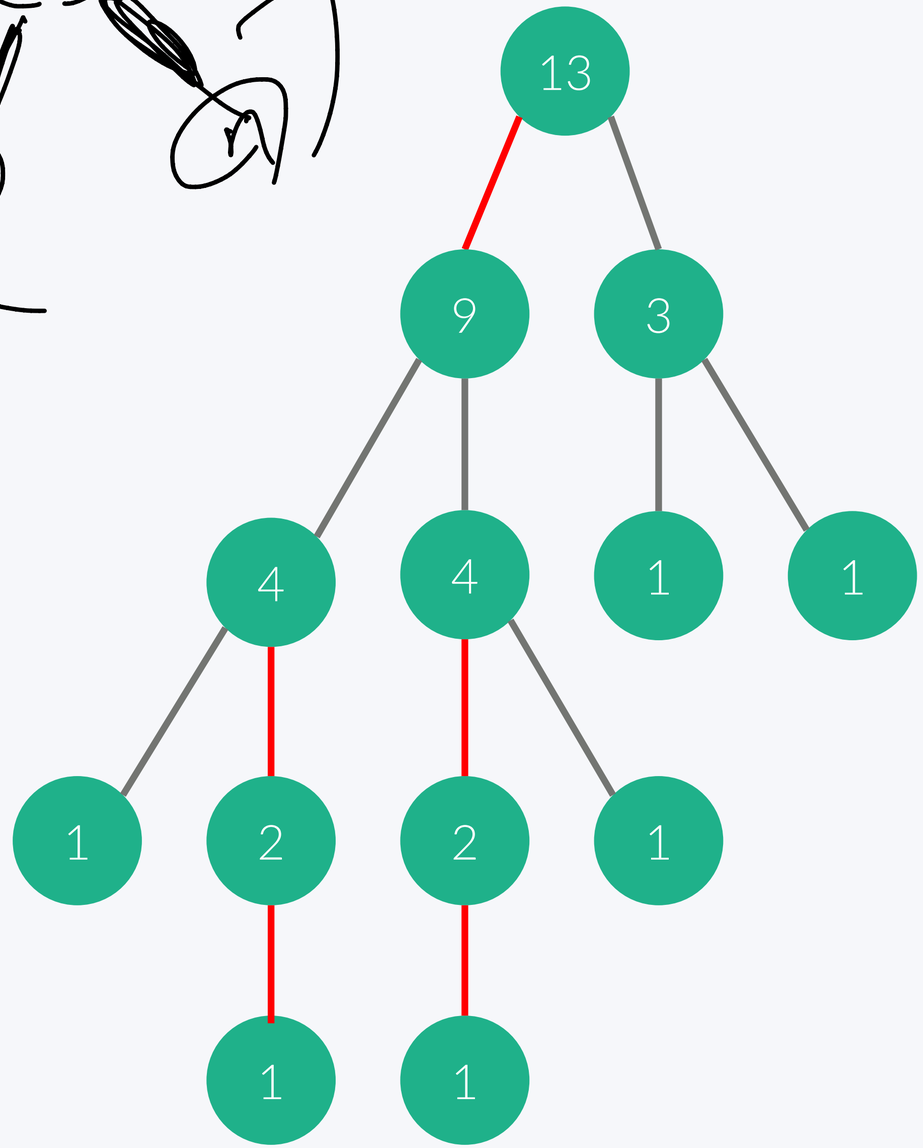
간선의 분류

Heavy-Light Decomposition



- 한 정점에서 자식으로 내려가는 간선들 중 무거운 간선은 두 개 이상일 수 없다.
- 부모에서 자식으로 갈 때 가벼운 간선을 따라 내려가면 정점의 무게는 항상 절반 넘게 줄어든다.
- 즉 루트에서 리프노드로 갈 때 너무 많은 가벼운 간선을 만날 수 없다.

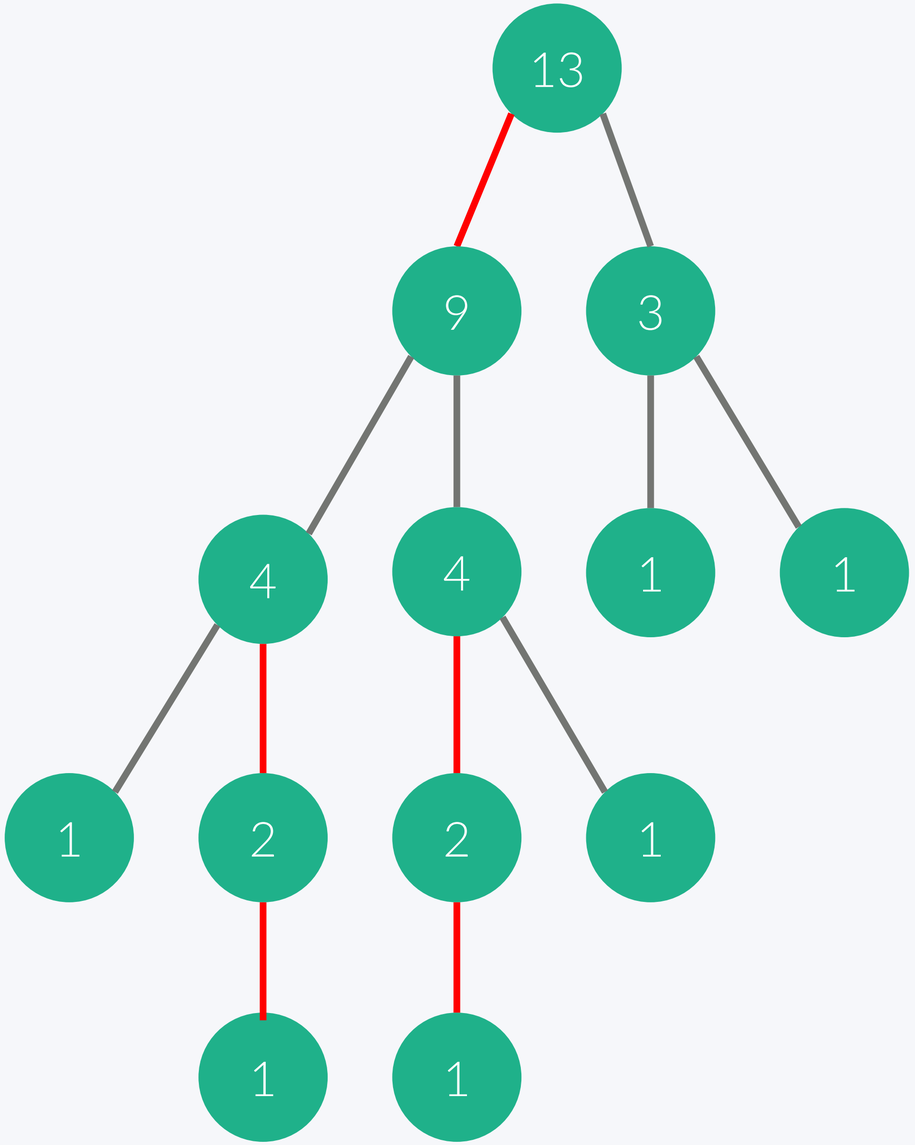
↳ 많아야 O(logN)



간선의 분류

Heavy-Light Decomposition

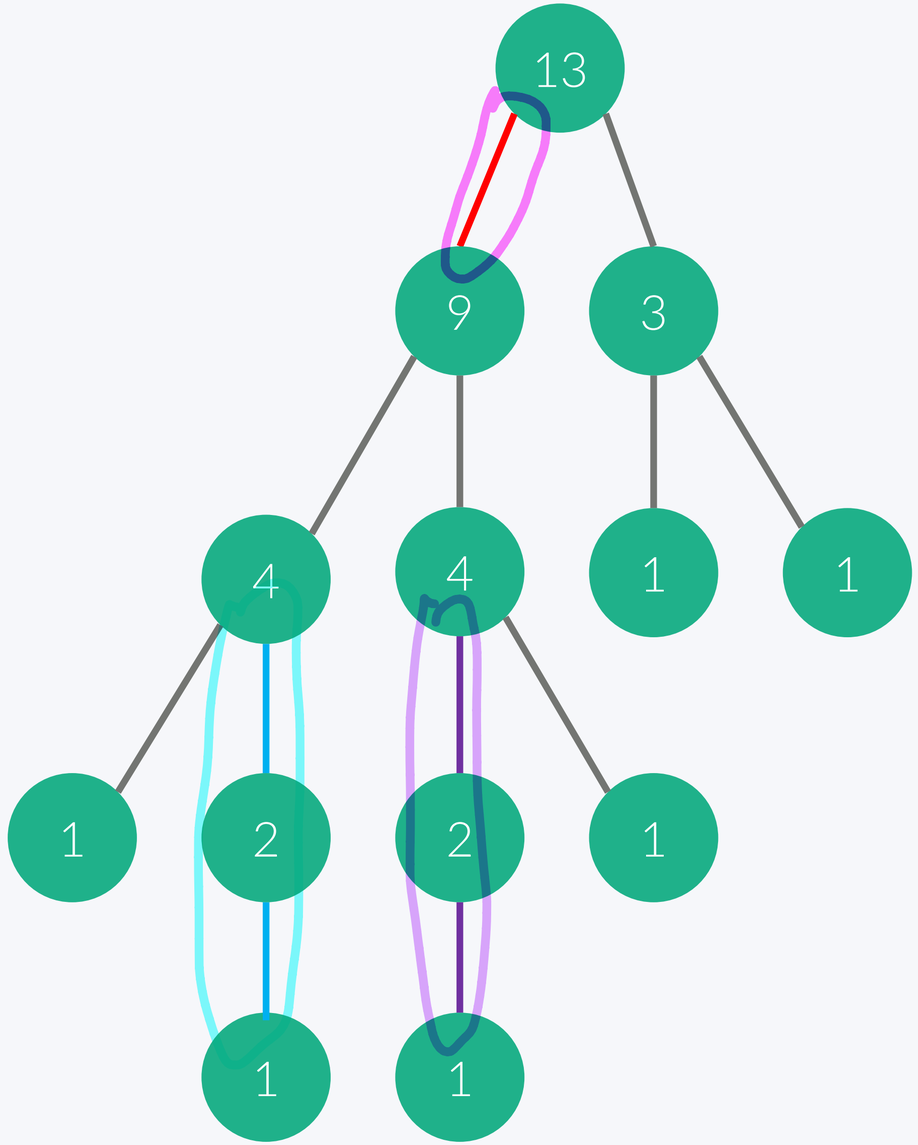
- 사실 1: 한 정점에서 자식으로 내려가는 간선들 중 무거운 간선은 두 개 이상일 수 없다.
- 사실 2: 부모에서 자식으로 갈 때 가벼운 간선을 따라 내려가면 정점의 무게는 항상 절반 넘게 줄어든다.



트리의 분해

Heavy-Light Decomposition

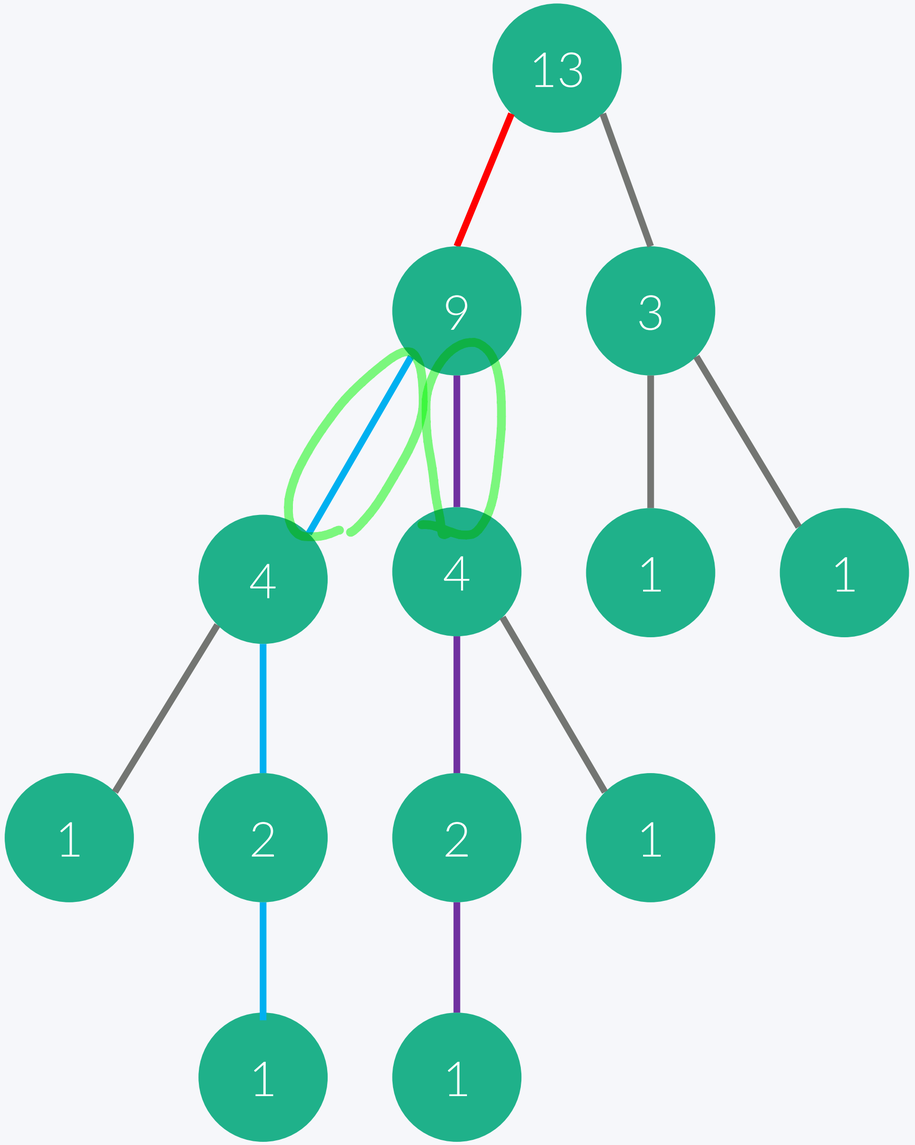
- 무거운 간선과 가벼운 간선으로 간선을 분류하면 트리를 분해할 수 있다.
- 인접한 무거운 간선을 한 집합으로 묶으면 사실 1에 의해 1차원 선분이 된다.
- 이를 무거운 경로라고 한다.



트리의 분해

Heavy-Light Decomposition

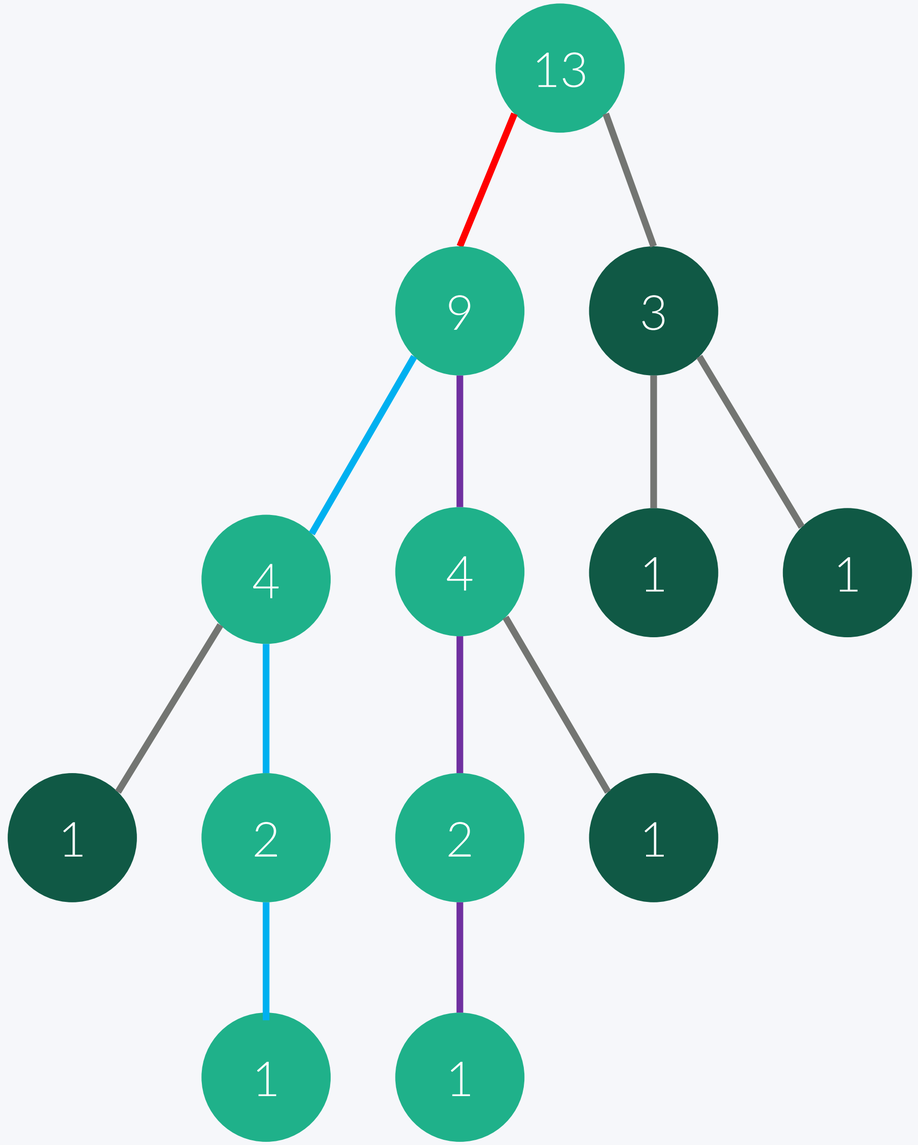
- 엄밀하게 정의에는 어긋나지만 무거운 경로의 가장 윗 정점에서 부모로 올라가는 간선도 해당 경로에 포함시켜준다.
- 나머지 간선들은 어떻게 해줄까?



트리의 분해

Heavy-Light Decomposition

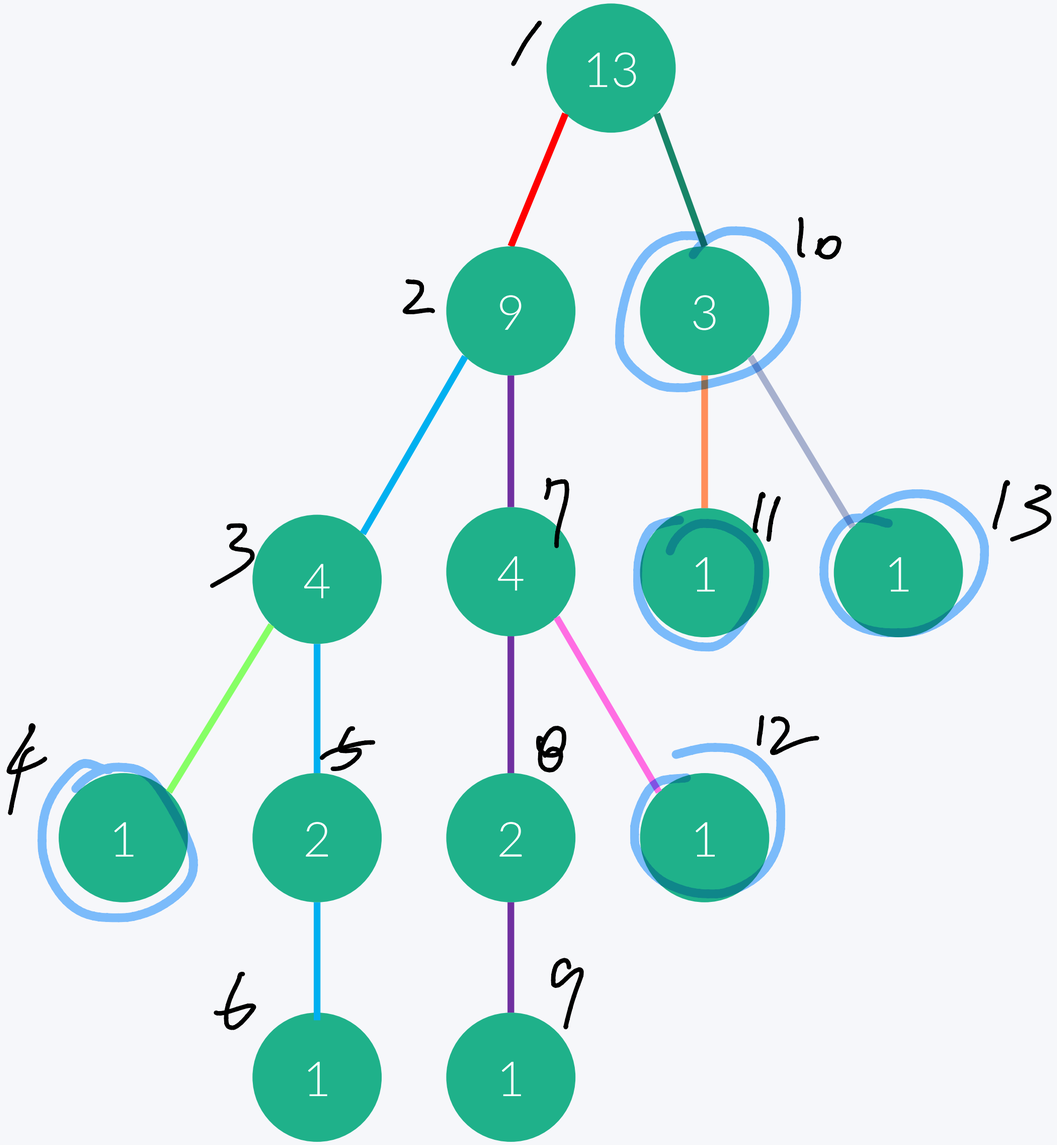
- 무거운 간선에 인접해 있지 않은 정점은 길이가 0인 무거운 경로라고 가정하자.
- 진한 초록색으로 색칠된 정점들이 그러한 정점들이다.



트리의 분해

Heavy-Light Decomposition

- 무거운 경로의 맨 윗 정점에서 부모로 가는 간선도 해당 경로에 포함시기로 했다.
- 그렇다면 모든 간선들은 정확히 하나의 무거운 경로에 포함된다.



(1, 2) (2, 7, 8, 9)

(2, 3) (7, 12)

(3, 4) (1, 10)

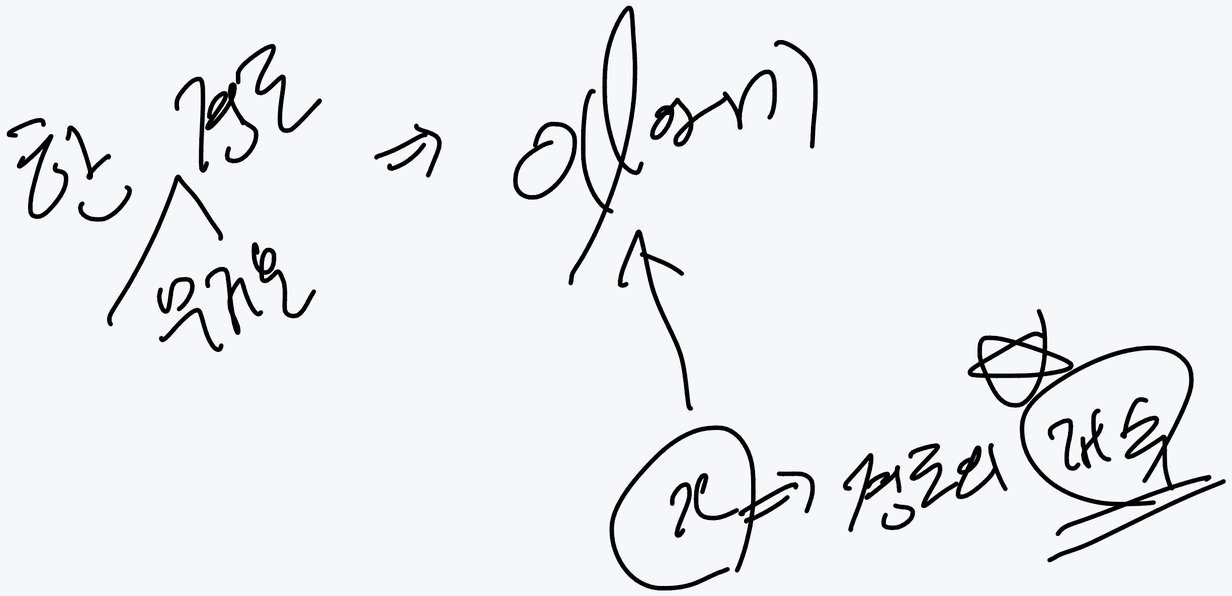
(3, 5, 6) (10, 11)

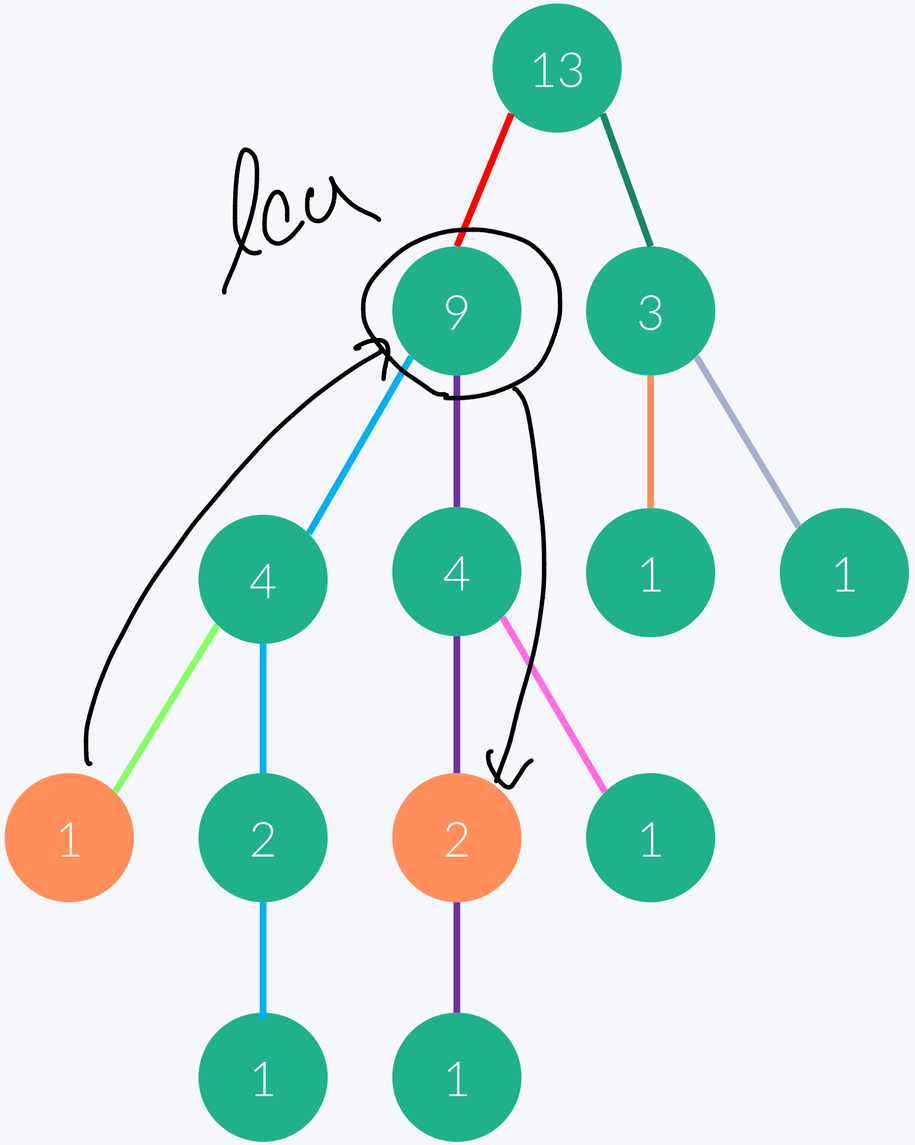
(10, 13)

트리의 분해

Heavy-Light Decomposition

- 이 때 임의의 두 정점은 몇 개의 무거운 경로들로 표현할 수 있다.
- 주황색으로 표현된 두 정점사이의 경로는 연두색 경로, 하늘색 경로의 일부, 보라색 경로의 일부로 표현할 수 있다.





트리의 분해

Heavy-Light Decomposition

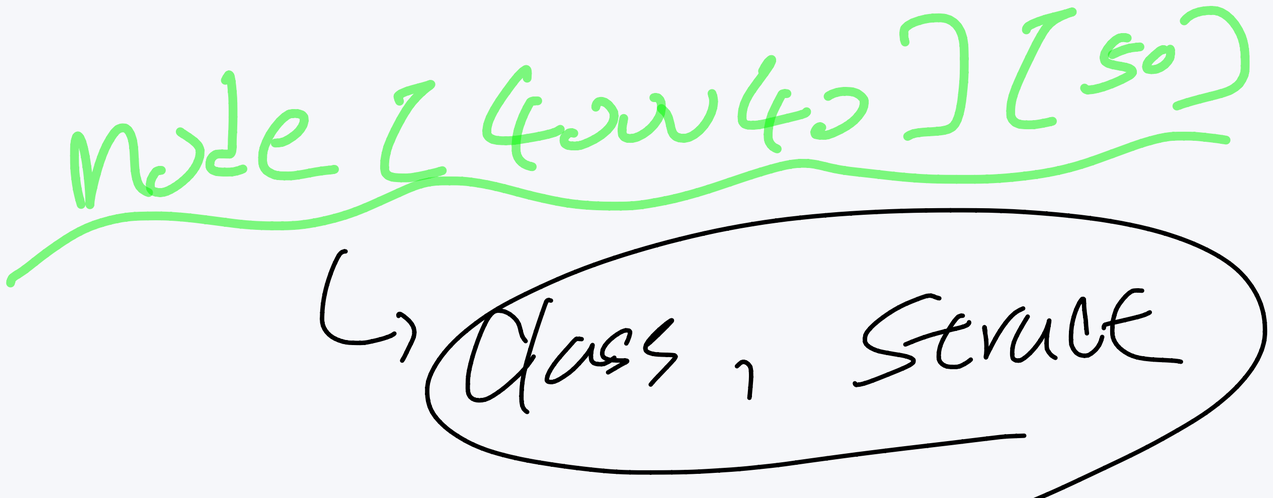
- 임의의 두 정점을 잡았을 때 두 정점의 경로에 무거운 경로는 몇 개가 있을까?
- 임의의 두 정점 u, v의 최소 공통 조상을 t라고 하자.
- 이 때 t에서 u 또는 v로 내려갈 때 몇 개의 서로 다른 무거운 경로를 만날까?
- 무거운 경로가 끊어지는 경우는 가벼운 간선을 만날때이다.
- 전에 언급한 사실 2에 의해 가벼운 간선은 루트에서 잎으로 갈 때 O(logN)를 넘게 만날 수 없다.
- 즉 임의의 두 정점 u, v의 경로에는 무거운 경로가 O(logN)개가 있다.

HLD로 문제 풀기

HLD로 문제 풀기

Heavy-Light Decomposition

- 이제 다시 문제로 돌아와서 문제에 어떻게 HLD를 적용할지 생각을 해보자.
- 주어진 트리의 간선들을 무거운 경로들의 집합으로 분해한다.
- 각각의 무거운 경로는 1차원 선분이다.
- 하나의 무거운 경로마다 구간 트리를 만들어 간선의 가중치를 저장하면 나중에 쿼리를 처리할 때 사용할 수 있다.



HLD로 문제 풀기

Heavy-Light Decomposition

- 업데이트 연산은 어떻게 해결할 수 있을까?
- 해당 간선의 어떤 무거운 경로에 속하는지 찾고 그 경로안에 몇 번째에 위치해 있는지 찾은 뒤 구간 트리를 갱신해주면 된다.
- 이 때 시간복잡도는 O(logN)이다.

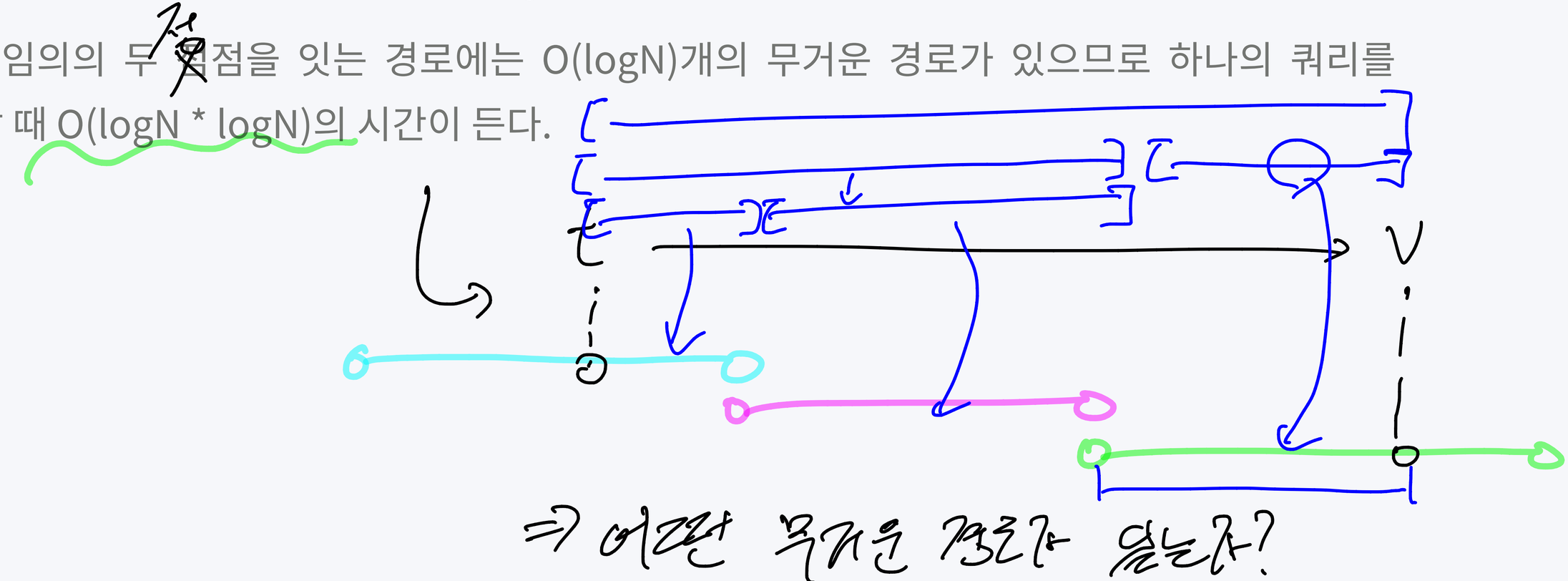
1. 찾는다. index ⇒ O(1)

2. 갱신한다. value ⇒ O(logn)

HLD로 문제 풀기

Heavy-Light Decomposition

- 임의의 두 정점을 잇는 경로에 존재하는 간선들의 비용 중 가장 큰 비용은 어떻게 찾을 수 있을까?
- 두 정점을 잇는 경로에 포함된 무거운 경로들에 대해서 구간 트리로 최대값을 구한다.
- 한 구간 트리에서 최대값을 구할 때 O(logN)의 시간이 든다.
- 이 때 임의의 두 정점을 잇는 경로에는 O(logN)개의 무거운 경로가 있으므로 하나의 쿼리를 처리할 때 O(logN * logN)의 시간이 든다.



트리와 쿼리1

https://www.acmicpc.net/problem/13510

- 예제 문제라서 HLD로 쉽게 해결 가능
- 소스: http://boj.kr/6cf2404f58794603a55ed59952f47666

트리와 쿼리2

https://www.acmicpc.net/problem/13511

- 1. u에서 v로 가는 경로에 있는 간선 비용의 합을 출력한다.
- 2. u에서 v로 가는 경로에 있는 정점 중 k번째 정점을 출력한다.
- 1번 쿼리는 구간합을 계산할 수 있는 구간 트리를 사용한다면 쉽게 계산할 수 있다.
  - query(u, lca(u, v)) + query(v, lca(u, v))
- 2번 쿼리는 lca를 구할 때 사용한 exp_parent 배열을 사용해 구할 수 있다.
- 2번 쿼리를 처리하는 과정은 최소 공통 조상을 구하는 과정과 유사하다.
- 소스: http://boj.kr/dc032ffb1b144043b957be00f0cb8f7d

트리와 쿼리3

https://www.acmicpc.net/problem/13512

- 1. i번 정점의 색을 바꾼다.(흰색이면 검은색, 검은색이면 흰색)
- 2. 1번 정점에서 v번 정점으로 가는 경로 중 가장 처음 만나는 검은색 정점을 출력한다.
- 기존의 HLD문제와는 다른 유형이다.
- 문제를 해결하려면 구간 합을 구할 수 있는 구간 트리가 필요하다.
- 흰색을 0, 검은색을 1로 표현한다.

트리와 쿼리3

https://www.acmicpc.net/problem/13512

- query(1, x)는 1번 정점에서 x번 정점까지 가는데 만나는 검은 정점의 수로 정의된다.
- 1번 정점에서 x번 정점까지 가는데 만나는 검은 정점의 수는 단조 증가한다.
- 즉 이분 탐색을 이용해 1번 정점에서 최소 얼만큼은 내려와야 검은 정점을 만나는지 찾을 수 있다.
- 1번 점점에서 k만큼 내려온 정점을 빠르게 찾는 방법이 필요하다.
- 최소 공통 조상을 찾는 코드의 exp_parent 배열을 사용하면 손쉽게 구할 수 있다.
- 소스: http://boj.kr/9ac54548fcda4469bb5207537a170aea

트리

https://www.acmicpc.net/problem/13309

- 1. u와 v가 이어졌는지 판단한다.
- 2. u와 v가 이어졌는지 판단하고 u 또는 v에서 부모로 올라가는 간선을 제거한다.
- 트리의 연결성을 판단하는 문제로 얼핏 보면 HLD와는 관련이 없을 것 같다.
- 하지만 임의의 두 정점 사이의 경로는 한가지밖에 없다는 트리의 성질을 파악하면 HLD로 문제를 해결할 수 있다는 점을 깨달을 수 있다.

트리

https://www.acmicpc.net/problem/13309

- 트리 간선의 비용이 1이면 두 정점이 이어져 있다는 것을 의미한다.
- 간선의 비용이 0이라면 두 정점은 끊어져 있다는 것을 의미한다.
- query(u, lca(u, v)) + query(v, lca(u, v))는 u와 v 사이에 있는 간선의 개수를 의미한다.
- u와 v사이에 실제 존재하는 간선의 수와 연결되어 있다면 존재해야 하는 간선의 수를 비교해 연결성을 확인할 수 있다.
- 연결되어 있을 때 존재해야 하는 간선의 수는 depth 배열을 이용해 구할 수 있다.
- 소스: http://boj.kr/ec205a62e6e549ac8d8fa7921bf71014

Grass Planting

https://www.acmicpc.net/problem/5916

- 1. u와 v사이에 존재하는 모든 간선에 1을 더한다.
- 2. u와 v의 경로안에 있는 모든 간선의 비용의 합을 출력한다.
- 하나의 원소가 아니라 구간을 업데이트하는 쿼리가 있어 단순 구간 트리로는 해결이 어렵다.
- 구간을 업데이트 하는 Lazy Propagation이 가능한 구간 트리를 사용한다면 해결할 수 있다.
- 소스: http://boj.kr/763d722f08b54892867fc85bf884025e

남극 탐험

https://www.acmicpc.net/problem/2927

- 1. u와 v를 연결하는 간선을 추가한다. 단 u에서 v로 이동할 수 있는 경우에는 간선을 추가하지 않는다.
- 2. 정점 u의 값을 업데이트한다.
- 3. u에서 v로 이동하는 경로상에 있는 정점의 값의 합을 구한다.
- 1번 쿼리에서 문제의 힌트를 얻을 수 있다.
- 이동할 수 있는 경우에는 간선을 추가하지 않는다는 것을 봤을 때 각각의 정점들은 하나의 트리에 속한다는 것을 알 수 있다.
- 모든 쿼리를 다 받았을 때 완성되는 트리의 모습을 알 수 있다.

남극 탐험

https://www.acmicpc.net/problem/2927

- 즉 동적으로 트리를 만들어 나가지 말고 이미 완성된 트리에 HLD를 적용하면 된다.
- 간선을 제거하는 작업이 없고 추가만 있기 때문에 두 정점의 연결성은 disjoin-set 알고리즘을 사용해 관리하면 편리하다.
- 트리가 여러 개로 나눠져 있을 수 있다.
- 이 경우는 각각의 트리의 루트들을 일렬로 연결해 하나의 트리로 생각하면 한 번의 HLD만으로 문제를 해결할 수 있다.
- 소스: http://boj.kr/57d9a65115994295a51a59a1b9fc3ffa