Fast LCA calculating algorithm

Hyunsoo Kim (15036)

Naïve ($O(n)$)

LCA(node u, node v)

- 두 노드(u, v)의 레벨이 다르면...
  레벨이 같을때까지 큰 것을 한 칸씩 위로 올림

- 두 노드(u v)의 레벨이 같으면 ....
  같은 노드가 될때까지 한 칸씩 위로 올림

It’s too simple.

Basic idea

LCA(node u, node v)

- 두 노드(u, v)의 레벨이 다르면...
  레벨이 같을때까지 큰 것을 한 칸씩 위로 올림

- 두 노드(u v)의 레벨이 같으면 ....
  같은 노드가 될때까지 한 칸씩 위로 올림

굳이 한 칸씩 볼 필요는 없다.

Sparse table

- D[k][i] = 정점 i의 2^k번째 부모
- D[k][i] = D[k-1][D[k-1][i]]

  (2^{k-1}번째 부모의 2^{k-1}번째 부모는 2^k번째 부모)

```
for(int k=1;k<20;k++) {
    for(int i=1;i<=n;i++) {
        par[k][i] = par[k-1][par[k-1][i]];
    }
}
```

LCA(node u, node v)

- 두 노드(u, v)의 레벨이 다르면...
  레벨이 같을때까지 큰 것을 한 칸씩 위로 올림 … ①

- 두 노드(u v)의 레벨이 같으면 ....
  같은 노드가 될때까지 한 칸씩 위로 올림 … ②

Sparse table을 통해 ①과 ②를 $O(\log n)$에 계산할 수 있다.

How to calculate ① in $O(\log n)$

- 두 노드(u, v)의 레벨이 다르면...
  레벨이 같을때까지 큰 것을 한 칸씩 위로 올림 … ①

두 노드 사이의 레벨 차이를 이진수로 나타낼 수 있다.

ex) $19 = 10011_{(2)}$

이 때, 이진수로 나타낸 각 자릿수가 1인지 아닌지 판단하고

i번째 자리수가 1이라면 2^i 만큼 올려주면 된다.

ex) 19번째 부모 : 2^4번째 부모의 2^1번째 부모의 2^0번째 부모

How to calculate ① in $O(\log n)$

- 두 노드(u, v)의 레벨이 다르면...
  레벨이 같을때까지 큰 것을 한 칸씩 위로 올림 … ①

이 때, 두 노드 간의 레벨 차이는 최대 n이므로

봐야 하는 이진수의 자릿수는 최대 $O(\log_2 n)$ 개이다.

코드 :

```
if(dep[A] < dep[B]) swap(A, B);
int tmp = dep[A] - dep[B];
for(int i=0;i<20;i++) {
    if((tmp & (1<<i)) > 0) {
        A = par[i][A];
    }
}
```

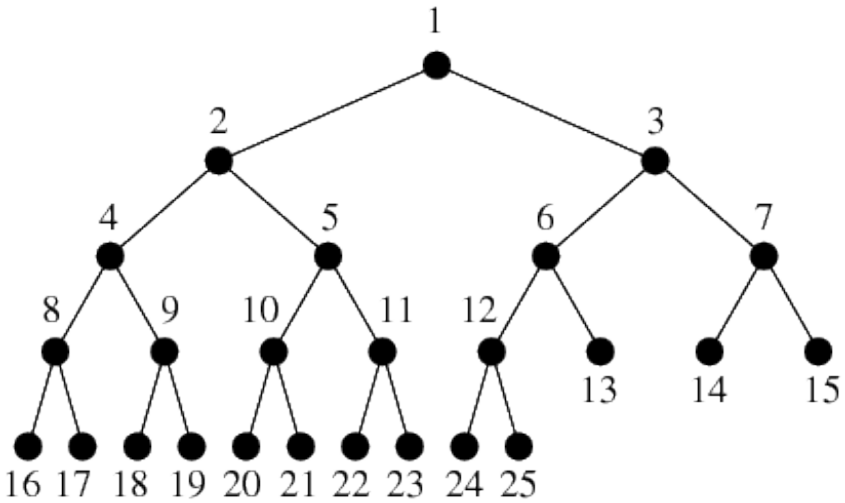
How to calculate ② in $O(\log n)$

- 두 노드(u v)의 레벨이 같으면 ….
  같은 노드가 될때까지 한 칸씩 위로 올림 … ②

한 가지 관찰을 하면 이것을 빠르게 계산할 수 있다.

두 노드를 한 칸씩 계속 올리다 보면, 두 노드가 계속 다르다가
어떤 시점부터 계속 같은 노드를 가리키게 된다.
(처음으로 같아지는 노드가 LCA이다.)

How to calculate ② in $O(\log n)$



ex) 21과 23의 경우

0	1	2	3	4
21	10	5	2	1
23	11	5	2	1

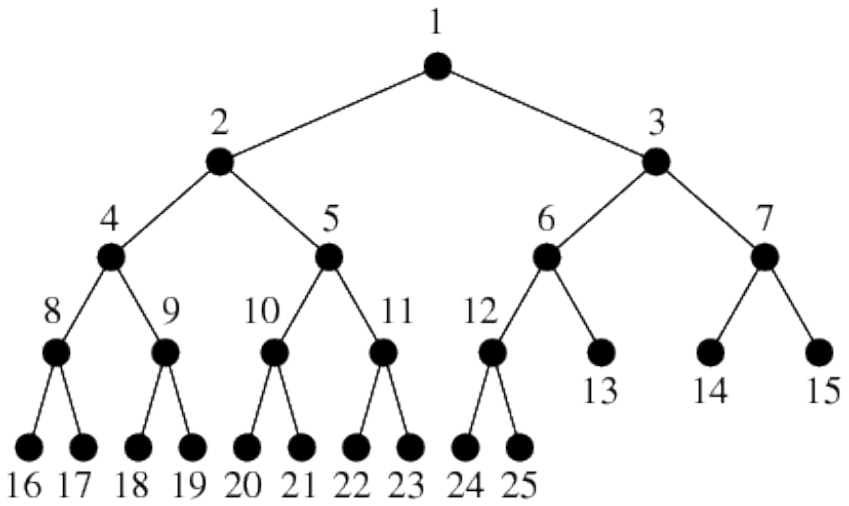
번째 부모

처음으로 같아지는 지점 (LCA)

How to calculate ② in $O(\log n)$

위의 성질을 이용하면 이진 탐색으로 LCA를 찾을 수 있다.

이진 탐색으로 두 노드가 다른 값을 가리키는 마지막 노드를 찾자.



ex) 21과 23의 경우

LCA인 5 바로 직전의 노드인

10과 11을 찾는다.

How to calculate ② in $O(\log n)$

- 두 노드(u v)의 레벨이 같으면 ….
  같은 노드가 될때까지 한 칸씩 위로 올림 … ②

"a번째 부모부터 b번째 부모까지 LCA 직전의 점이 될 수 있다."
고 할 때, 탐색 구간이 [a, b]라고 하자.

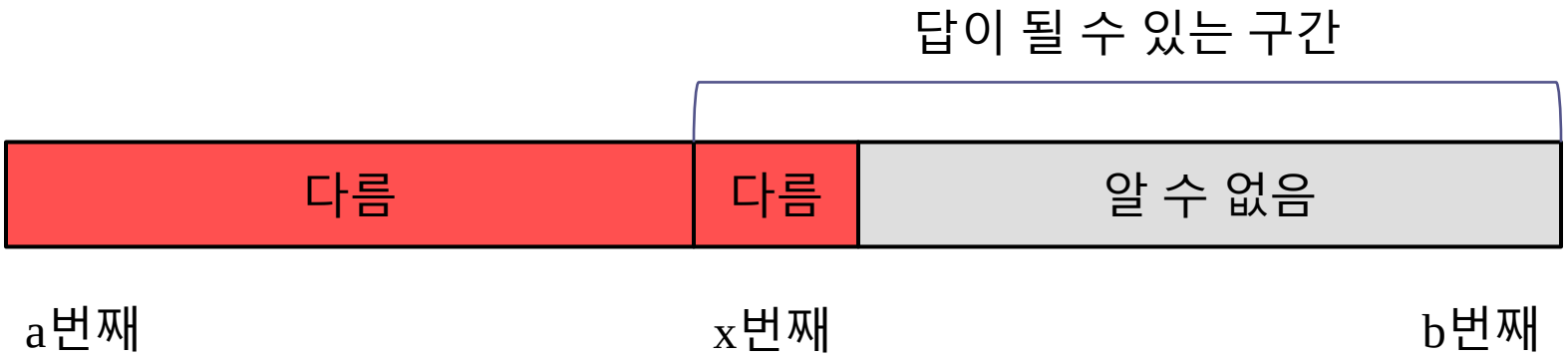
$x \in [a, b]$인 x에 대하여 u, v의 x번째 부모가 가리키고 있는 노드가 같은지의 여부를 알 수 있다면 탐색 구간을 줄일 수 있다.

How to calculate ② in $O(\log n)$

- 두 노드(u v)의 레벨이 같으면 ….
  같은 노드가 될때까지 한 칸씩 위로 올림 … ②

먼저, u와 v의 x번째 부모가 서로 다르다면,

LCA 직전 정점이 있을 수 있는 구간은 [x, b]가 된다.

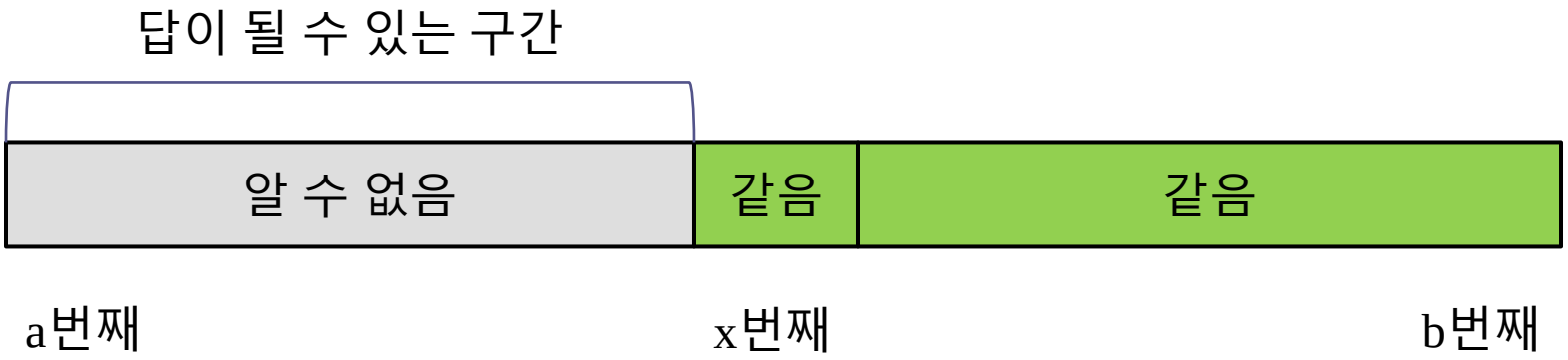


How to calculate ② in $O(\log n)$

- 두 노드(u v)의 레벨이 같으면 ….
  같은 노드가 될때까지 한 칸씩 위로 올림 … ②

비슷하게, u와 v의 x번째 부모가 서로 같다면,

LCA 직전 정점이 있을 수 있는 구간은 [a, x]가 된다.



How to calculate ② in $O(\log n)$

- 두 노드(u v)의 레벨이 같으면 ….
  같은 노드가 될때까지 한 칸씩 위로 올림 … ②

그래서 검사할 값 x를 적절히 정하면 구간을 반씩 줄여나갈 수 있다.

예) x = (a+b)/2로 잡으면 [a, b]에서 [a, (a+b)/2-1]

또는 [(a+b)/2, b]로 줄일 수 있다.

이 때, 구간을 반씩 줄이는 연산을 $O(\log n)$ 번 해야 답이 정해지므로

이 과정을 $O(\log n)$ 에 하려면 x번째 부모가 어떤 노드인지를

상수 시간에 구할 수 있어야 한다.

How to calculate ② in $O(\log n)$

- 두 노드(u v)의 레벨이 같으면 ….
  같은 노드가 될때까지 한 칸씩 위로 올림 … ②

Sparse table을 잘 이용하면 이것을 해결할 수 있다.

우선, 초기 탐색 구간을 $[0, 2^k-1]$로 잡자.

(k는 2^k이 n 이상이 되는 최소의 정수 k)

우리는 탐색 구간을 $[0, 2^i-1]$꼴로 유지하면서

최종적으로는 $[0, 2^0-1] = [0,0]$으로 만들 수 있다.

How to calculate ② in $O(\log n)$

현재 탐색 구간이 $[0, 2^i - 1]$ 이라고 하자. 이 때, $x = 2^{i-1}$이라고 두면, 우리는 D[i-1][u], D[i-1][v]를 통해 x번째 부모를 빠르게 구할 수 있다.

이 때, D[i-1][u] == D[i-1][v]라면 탐색 구간은 $[0, 2^{i-1} - 1]$이 되고 D[i-1][u] != D[i-1][v]라면 탐색 구간은 $[2^{i-1}, 2^i - 1]$이 된다.

그런데 D[i-1][u] != D[i-1][v]일 때 u' = D[i-1][u], v' = D[i-1][v]로 두면 다시 u', v'에 대하여 탐색 구간이 $[0, 2^{i-1} - 1]$라고 생각할 수 있다.

How to calculate ② in $O(\log n)$

위의 과정을 거치면 항상 탐색 구간을 $[0, 2^i - 1]$에서

$[0, 2^{i-1} - 1]$로 상수 시간에 한 차수 줄일 수 있다.

따라서 $O(\log n)$ 시간에 LCA 직전 정점을 구할 수 있다.

코드 :

```
if(A == B) return A;
for(int i=19;i>=0;i--) {
    if(par[i][A] != par[i][B]) {
        A = par[i][A];
        B = par[i][B];
    }
}
return par[0][A];
```

Full solution ($O(\log n)$)

```
int getlca (int A, int B) {
    if(dep[A] < dep[B]) swap(A, B);
    int tmp = dep[A] - dep[B];
    for(int i=0;i<20;i++) {
        if((tmp & (1<<i)) > 0) {
            A = par[i][A];
        }
    }
    if(A == B) return A;
    for(int i=19;i>=0;i--) {
        if(par[i][A] != par[i][B]) {
            A = par[i][A];
            B = par[i][B];
        }
    }
    return par[0][A];
}
```

It’s too simple.