

“기지 간소화 (base)” 문제 풀이

작성자: 김동현

풀이 설명의 편의상, 문제에서 정의된 ‘기지’는 ‘정점’, ‘통로’는 ‘간선’으로 서술하였다.

부분문제 1

$i, i+1, \dots, j$ 번 정점만 남기기로 했다고 할 때, 어떤 간선이 ‘유지 비용에 관여하는지’는 다음과 동치임을 우선 알 수 있다.

- 해당 간선을 기준으로 양 쪽 컴포넌트 각각에 $\{i, i+1, \dots, j\}$ 에 속하는 정점이 존재한다.

트리 구조의 특성상 위 조건을 만족하는 간선은 반드시 포함시켜야 하며, 반대로 위 조건을 만족하지 않는 간선은 포함시킬 필요가 없음을 알 수 있다.

부분문제 1을 해결하기 위해서는 각 (i, j) 쌍에 대해 유지 비용에 관여하는 간선을 $O(N)$ 에 구하면 된다. 이는 정점 하나를 루트로 잡고 DFS를 하면서 각 간선 아래쪽 서브트리에 있는 $\{i, i+1, \dots, j\}$ 에 속하는 정점 개수를 세 주면 $O(N)$ 만에 각 간선이 유지 비용에 관여하는지 아닌지를 알 수 있다.

부분문제 2

각 간선에 대해, 해당 간선이 유지 비용에 관여하게 되는 (i, j) 쌍의 개수를 세는 방식으로 접근하자. 쌍의 개수와 간선 길이를 곱한 값을 모든 간선에 대해 더하면 된다.

간선을 고정했을 때, 해당 간선 기준으로 나뉘는 컴포넌트 둘 중 한 쪽을 검은색, 반대쪽을 하얀색으로 색칠했다고 하자. 이제 이 간선이 관여하는 (i, j) 쌍은 $[i, j]$ 구간에 검은색과 하얀색 정점이 모두 존재하는 쌍이다.

간선을 고정한 뒤 색칠하는 것은 DFS로 $O(N)$ 에 할 수 있고, 각 i 에 대해 검은색과 하얀색 정점을 모두 포함하게 되는 최소 j 를 구해나가는 것 역시 $O(N)$ 에 수행할 수 있다.

부분문제 3

0번 노드를 루트로 잡게 되면, 각 간선에 대해 아래쪽 서브트리에 속하는 점들의 번호가 구간의 형태로 나타남을 알 수 있다. 즉, 0번부터 $N-1$ 번 정점까지의 색깔이 하얀색 구간 - 검은색 구간 - 하얀색 구간의 형태로 나타난다. 각각의 길이를 A, B, C 라 하면 검은색과 하얀색을 모두 포함하는 (i, j) 쌍의 개수는 $\frac{N(N+1)}{2} - \frac{A(A+1)}{2} - \frac{B(B+1)}{2} - \frac{C(C+1)}{2}$ 와 같은 수식으로 구할 수 있다.

부분문제 4

다음 연산을 지원하는 자료구조를 생각하자.

- 정점 하나를 검은색으로 칠한다. (초기에는 모든 정점이 하얀색이다)
- 검은색과 하얀색 정점을 모두 포함하는 (i, j) 쌍의 개수를 구한다.

위 연산을 지원하는 자료구조를 빠르게 작동하도록 구현할 수 있다면, 트리가 선형 구조이므로 점을 하나씩 검은색으로 칠하면서 모든 간선에 대해 해당 간선이 관여하는 (i, j) 쌍의 개수를 구할 수 있다.

검은색으로 칠해진 연속한 정점 번호의 구간들을 관리하는 `std::set` 자료구조를 이용하는 것이 가장 간단하게 구현할 수 있는 방법이며, 이 방법을 사용할 시 업데이트(검은색 칠하기)에 $O(\log N)$ 시간, 쌍 개수 구하기에 $O(1)$ 시간이 걸리게 되어 총 시간복잡도 $O(N \log N)$ 에 부분문제를 해결할 수 있다.

부분문제 5

부분문제 4에서 구현한 자료구조를 일반적인 트리에서도 적용할 수 있다. HLD(Heavy-Light Decomposition) 내지는 Small-to-Large Trick으로 불리는 테크닉을 사용하면 된다.

한 정점을 루트로 잡고 DFS를 수행하면서, 각 정점마다 “해당 정점이 루트인 서브트리”에 해당하는 자료구조를 구한다고 하자. 각 정점 v 에서 다음과 같은 작업을 수행한다.

- v 의 자식들 각각에 해당하는 자료구조를 모두 구한다. 이들 중 검은색 점이 가장 많이 칠해진 자식을 c_{max} 라고 하자.
- v 에 해당하는 자료구조는 우선 c_{max} 에 해당하는 자료구조를 그대로 가져온 뒤 c_{max} 를 제외한 다른 자식들에서 칠해진 검은 점을 v 에 해당하는 자료구조에 하나씩 옮겨 칠해 준다.
 - 자료구조를 “그대로 가져온다”라는 것은 복사를 하는 것이 아니라, `std::swap` 내지는 포인터 등을 이용해 reference 자체를 $O(1)$ 만에 가져오는 것을 의미한다.
- 자료구조에 v 를 새로 검은색으로 칠한다.

이런 식으로 DFS를 수행하면 검은색 점을 칠하게 되는 횟수가 $O(N \log N)$ 이 된다. 검은색 점을 처음에 칠하는 횟수는 정점당 한 번이므로 $O(N)$ 이고, 검은색 점이 한 자료구조에서 다른 자료구조로 옮겨갈 때 그 자료구조의 크기가 2배 이상 증가하므로 옮겨다니는 횟수는 $O(\log N)$ 이 되기 때문이다. 즉, 시간복잡도 $O(N \log^2 N)$ 에 문제를 해결할 수 있다.