"외곽 순환 도로" 문제 풀이

작성자: 구재현

서술의 편의를 위해 풀이에서는 교차로는 정점, 도로는 간선이라고 부른다. $dist(x, y)$ 를 x 와 y 의 최단 경로의 길이라고 정의한다.

부분문제 1

Dijkstra Algorithm을 사용하면 1번 정점에서 시작해서 임의의 정점에 도달하는 데 걸리는 최단 시간을 모두 계산할 수 있다. 문제에서 주어진 그래프는 정점이 N 개이고 간선이 최대 $2N$ 개 이다. Heap 자료구조를 사용한 Dijkstra Algorithm을 사용할 시, $O(N \log N)$ 에 필요한 정보를 모두 계산할 수 있고, 각 쿼리에서는 계산된 정보를 출력해 주면 된다.

시간 복잡도는 $O(N \log N + Q)$ 이다.

부분문제 2

두 정점을 오가는 경로의 경우의 수는 크게 세 가지가 있다.

- 1 번 정점을 거쳐서 가는 경우
- 1 번 정점을 거치지 않고, 외곽 순환 도로를 시계방향으로 돌아가는 경우
- 1 번 정점을 거치지 않고, 외곽 순환 도로를 반시계방향으로 돌아가는 경우

첫 번째 경우는 부분 문제 1과 같이, 1번 정점에서 시작하는 Dijkstra's Algorithm으로 $O(N \log N)$ 에 계산할 수 있다. $dist(s, 1) + dist(1, t) = dist(1, s) + dist(1, t)$ 이기 때문이다. 1번 정점에서 직접 연결된 간선을 사용하면 Dijkstra's Algorithm이 필요 없다고 생각할 수 있지만, 다른 간선으로 외곽 순환 도로에 진입한 후, 외곽 순환 도로를 통해서 목적지에 도착하는 것이 더 효율적일 수 있다.

두 번째와 세 번째 경우는, w_i 배열의 원형 구간 합을 구하는 문제가 된다. 이는 w_i 배열의 부분 합 (Prefix sum) 을 전처리해두면, $O(1)$ 시간에 계산할 수 있다.

시간 복잡도는 $O(N \log N + Q)$ 이다.

부분문제 3

외곽 순환 도로의 가중치가 매우 크기 때문에, 전혀 사용하지 않는 것이 이득이다. 외곽 순환 도로를 모두 무시하면, 문제에서 주어진 그래프는 트리 형태를 띈다.

고로, 문제는 트리 상에서 두 정점 간의 거리를 빠르게 계산하는 것으로 환원된다. 이는 여러 가지 방법으로 해결할 수 있는데, 가장 간단한 방법은 Sparse Table을 사용하여 두 정점의 LCA (Lowest Common Ancestor) 를 계산하고, $dist(1, s) + dist(1, t) - 2 \times dist(1, LCA(s, t))$ 를 출력하는 것이다. $dist(1, *)$ 는 DFS를 통해서 계산할 수 있다.

위 방법을 사용했을 때, 시간 복잡도는 $O((N + Q) \log N)$ 이다.

부분문제 4

어떠한 경로가 외곽 순환 도로를 사용하지 않는 경우는 부분 문제 3과 동일하게 해결할 수 있다.

어떠한 경로가 외곽 순환 도로를 사용하는 경우를 생각해 보자. 외곽 순환 도로 상에 속하는 정점 사이는 비용 없이 오갈 수 있으니, 하나의 정점이라고 생각할 수 있다.

고로, 외곽 순환 도로의 임의의 정점에서 시작해서, 그래프 상의 정점에 도달하는 최단 거리를 계산하면 된다. 외곽 순환 도로 상에 속하는 아무 정점에서 Dijkstra's Algorithm을 사용해도, 결과는 항상 같다. 고로 한 번의 Dijkstra's Algorithm으로 문제를 해결할 수 있다.

시간 복잡도는 $O((N+Q)\log N)$ 이다.

부분문제 5

외곽 순환 도로가 없는 원래 트리를 기준으로, Centroid decomposition을 하자. 현재 보고 있는 Centroid를 c 라고 하자. c 의 자식은 최대 3개이고, 각 자식에 서브트리가 하나씩 있으니, 트리는 최대 3개의 서브트리로 분해됨을 알 수 있다.

어떠한 최적 경로의 형태는 다음 세 가지 중 하나이다:

- Centroid c 를 지난다.
- Centroid c 를 지나지 않으나, c 의 서로 다른 서브트리를 잇는 간선을 사용한다.
- 두 경우 모두 아니다.

첫 번째 경우는 Dijkstra's Algorithm을 사용하여 c 를 시작점으로 한 최단 경로를 계산하면 해결할 수 있다.

두 번째 경우에서는 아주 중요한 관찰을 할 수 있다. c 의 서로 다른 서브트리를 잇는 간선은 외곽 순환 도로일 것이고, 그러한 간선은 c 의 자식 개수보다 많지 않다는 것이다. 초기 그래프에서 이 사실은 명백히 참이다. 이 사실이 Subproblem에서도 성립하는 이유는, Subproblem으로 재귀적으로 내려갈 때, 최대 하나의 간선을 추가하여서 초기 그래프와 동일한 환경을 구성해 줄 수 있음을 관찰하면 된다.

고로 c 의 서로 다른 서브트리를 잇는 간선은 최대 3개이다. 이러한 간선을 찾는 것은 단순 DFS로 가능하다. 간선을 찾은 후, 간선의 한쪽 끝에서 Dijkstra's Algorithm을 사용하면 된다.

세 번째 경우가 최적 경로의 형태가 되려면, 쿼리로 주어진 두 정점이 c 가 아니며, 같은 서브트리에 속해야 한다. 이에 해당되는 부분문제가 유일하게 정해지니, 재귀적으로 해결하면 된다.

각 쿼리는 최대 $O(\log N)$ 개의 부분문제에 속하며, 각 레이어에서 가장 계산이 많은 연산은 $O(N\log N)$ 시간에 Dijkstra's Algorithm을 4번 호출하는 것이다. 종합하면, $O(N\log^2 N + Q\log N)$ 시간에 전체 문제가 해결된다.

풀이가 *사이클이 감싸고 있는 트리* 라는 특수한 구조에 독립적인 편이라, 구현이 일반적인 트리 문제에 비해 크게 어렵지 않다는 점을 참고하면 좋다.

부분문제 6

입력 정점의 차수에 특별한 제한이 없더라도, 차수가 3인 인스턴스로 변환할 수 있다. 어떠한 정점 p 의 자식이 현재 2개고, 내가 새로운 자식 i 를 추가하려고 하는 상황이라고 하자. 단순히 추가할 경우, p 의 차수가 4가 될 수 있지만, 다음과 같은 식으로 추가할 경우 차수가 4인 정점을 만들지 않을 수 있다.

- p' 이라는 새로운 정점을 만든다.
- p 에 마지막으로 추가한 정점을 j 라고 하자. p와 j 간의 연결을 끊고, p 의 새로운 자식을 p' 으로 두자.
- p' 에 두 자식 i, j 를 이어준다.
- 이후 p 에 대한 포인터를 p' 으로 바꿔준다.

이 외에도 다양한 방법이 있다. $p_i \leq i$ 인 형태로 입력이 주어지기 때문에, 10줄 내외의 짧은 코드 추가로 차수가 3인 인스턴스로 변환할 수 있다. 정점의 개수가 최대 2배가 되지만, 복잡도에는 영향이 없다.

시간 복잡도는 $O(N \log^2 N + Q \log N)$ 이다.