"택시 (taxi)" 문제 풀이

작성자: 윤창기

부분문제 1

$N \leq 20$ 으로 지수 시간 알고리즘이 동작하는 제한이다. 가장 마지막으로 방문한 도시와, 지금까지 방문한 도시의 집합을 저장하는 동적 계획법으로 해결할 수 있다.

부분문제 3

$N \leq 2\,000$인 경우로, 임의의 두 점 i, j에 대해 i에서 j로 가는 $B_i\text{dist}(i,j)$ 비용의 간선을 이어주면 Dijkstra's algorithm 등을 통해 $O(N^2)$ 시간 안에 해결할 수 있다. 여기서 $\text{dist}(i,j)$는 입력에 주어진 트리에서 두 점을 잇는 최단경로의 길이로 정의한다. 이후에는 Dijkstra's algorithm을 보다 효율적으로 subquadratic 시간에 해결하는 방법을 찾아야 한다.

부분문제 2

주어진 트리가 직선인 경우, Dijkstra's algorithm을 보다 최적화한 동적 계획법을 생각할 수 있다. D_i를 1번 도시에서 i번 도시로 이동하는 최소 비용으로 정의하면, $D_1 = 0$, $i \neq 1$에 대해서는 다음이 성립한다.

$$\begin{aligned} D_i &= \min_{j \neq i} D_j + B_j|C_i - C_j| \\ &= \min\left(\min_{j<i}\left[D_j - C_jB_j + B_jC_i\right], \min_{j>i}\left[D_j + C_jB_j - B_jC_i\right]\right) \end{aligned}$$

여기서 $C_i = \text{dist}(1,i)$로 정의한다. 이 때, i에서 j로 이동하는 데 이어 j에서 k로 이동한다면, $B_i > B_j$가 성립한다고 가정해도 좋다. 그렇지 않은 경우 i에서 k로 직접 이동하는 것에 비해 비용이 더 낮을 수 없기 때문이다. 따라서 $1, \cdots, n$의 모든 정점을 B_i가 내림차순이 되는 순서로 정렬하고, 위 동적 계획법 식을 볼록 껍질 최적화 (Convex Hull Optimization) 등을 통해 $O(N \log N)$ 시간에 해결할 수 있다.

부분문제 4

$B_i \leq 30$이 성립하는 경우이다. 위의 경우에 적용했던 동적 계획법을 트리에 대해 다시 쓰면 아래와 같다. 편의상 $B_i < B_1$인 정점 i에 대해서만 생각하자. 그렇지 않은 정점들에 대해서는 답이 자명하다.

$$D_i = \min_{B_j > B_i} D_j + B_j \cdot \text{dist}(i,j) \tag{1}$$

고정된 b와 $B_i = b$를 만족하는 정점 i의 목록, 그러한 정점 i의 목록이 주어진 경우 이 정점들에 의한 식 **??**의 갱신을 Multi-source Dijkstra's algorithm을 활용하여 한꺼번에 $O(N \log N)$ 에 수행할 수 있다. 따라서 b를 내림차순으로 순회하며 이를 반복하면 전체 문제를 $O(\max B_i \cdot N \log N)$ 시간에 해결할 수 있다.

부분문제 5

$B_i > 0$을 만족하는 정점의 수 K가 적은 경우이다. $B_i > 0$인 정점의 택시만 이용하는 답은 부분문제 3과 같이 Dijkstra's algorithm을 활용하여 $O(K^2)$ 시간 안에 구할 수 있다. $B_i = 0$인 정점에 도착하는 최소 비용은 트리를 여러 직선으로 분할하는 Heavy-Light Decomposition 등을 이용하여 계산할 수 있다. $B_i > 0$

인 정점의 조상에 위치한 Heavy Chain은 최대 $\log N$개 존재할 수 있으므로, Heavy Chain h 중 i에 가장 가까운 정점 $h(i)$에 초기 비용 $D_i + B_i\text{dist}(h(i), i)$, 거리당 비용 B_i짜리 택시가 있다고 가정해도 좋다. 이와 같이 heavy chain에 등록되는 새로운 정점의 개수가 최대 $K \log N$개이고, 부분문제 2와 같이 Convex Hull Optimization을 이용하여 $K^2 + N \log N$ 시간에 문제를 해결할 수 있다.

부분문제 6

추가 제약 조건이 없는 경우로, 전체 문제를 해결하기 위해서는 B_i가 감소하는 순서대로 탐색하면 된다는 사실과 함께 Centroid Decomposition 등의 자료구조를 활용할 수 있다.

정점들을 B_i가 감소하는 순서로 정렬한다. 어떤 정점 i에 대해 D_i가 결정되어, 다른 점으로 이동하는 경우를 생각해보자. 만약 i가 트리의 루트라면, 다른 모든 정점 j에 대해 $D_j \leftarrow \min(D_j, D_i + B_i\text{depth}(j)$ 와 같이 갱신하는 것으로 충분하다. 이는 y절편이 D_i, 기울기가 B_i인 직선을 $x = \text{depth}(j)$에서 평가한 것으로 생각할 수 있다.

실제로는 i가 루트가 아니므로, i의 모든 부모 p에 대해 p의 서브트리에 속하는 정점 v를 잡고 $D_v \leftarrow \min(D_v, D_i + B_i(\text{depth(i)} + \text{depth(v)} - \text{depth(p)})$ 와 같이 갱신해줄 수 있다. 이는 y절편이 $D_i + B_i(\text{depth}(v) - \text{depth}(p))$, 기울기가 B_i인 직선으로 생각할 수 있다.

어떤 D_j를 계산하기 위해서는 과거에 생성된 직선들을 $x = \text{depth}(j)$에서 계산한 값 중 최솟값을 찾음으로써 계산할 수 있고, Convex Hull Optimization을 사용하면 직선의 개수가 L개일 때 총 $O(L \log L)$ 시간 내에 새로운 직선을 삽입하거나, 임의의 점에서 직선들 중 최솟값을 찾는 자료구조를 지원할 수 있다.

단순한 방법으로 이를 구현할 경우 직선이 총 $O(N^2)$개가 되므로, 이를 원래 트리가 아닌 Centroid Decomposition 과정을 나타낸 트리에서 수행하는 경우 생성되는 직선의 개수를 $O(N \log N)$ 개로 줄일 수 있으며, 전체 문제를 $O(N \log^2 N)$ 시간에 해결할 수 있다.