Geometric Spanning Tree

d 차원 좌표공간 상에 크기 N 의 점 집합 S 가 주어졌을 때, 집합 S 를 정점 집합으로 하며 각 정점 쌍마다 두 점의 거리를 가중치로 한 그래프를 생각할 수 있다. 이러한 그래프에서 최소 스패닝 트리를 찾는 문제를 *Minimum Geometric Spanning Tree* 라고 한다 (Encyclopedia of Algorithms 533p). 두 정점의 "거리" 를 지정하는 방법은 여러 가지가 존재하고, 이 거리계를 어떻게 지정하느냐에 따라서 문제를 해결하는 양상이 많이 달라진다. 일반적으로 사용하는 거리계는 L_1 metric (Taxicab distance), L_2 metric (Euclidean distance), L_∞ metric (Chebyshev distance) 이다.

L_1 metric에서 Minimum Geometric Spanning Tree를 찾는 문제는 *Rectilinear Spanning Tree* 라는 다른 이름으로 알려져 있다 (Encyclopedia 754p). $d = 2$ 인 경우, 이 문제에 대한 $O(n \log n)$ 풀이가 잘 알려져 있으며 가끔 경시대회 심화 문제로도 나올 수 있는 수준이다. 하지만 $d > 2$ 인 경우에 대해서는 간단한 풀이가 알려져 있지 않다. Tarjan과 Gabow는 *Scaling and related techniques for geometry problems (1984)* 을 통해서 일반적인 d 에 대한 $O(n \log^d n \log \log n)$ 알고리즘을 제안하였지만, 다양한 테크닉에 기반한 알고리즘으로 간단하지는 않다. 이 글에서는 d 차원 좌표 공간 상에서 S 에 대한 최소 스패닝 트리를 $O(n \log^{d-1} n)$ 시간에 해결하는 새로운 알고리즘을 소개한다.

이 알고리즘에 대한 아이디어는 KAIST 2019 ICPC Mock Contest 를 준비하면서 출제한 문제인 "Wind of Change" 에서 고안하였다. 문제 "Wind of Change" 에서 고려한 거리계는, 같은 정점 집합을 가지는 K 개의 가중치 있는 트리가 주어질 때, 두 정점과의 거리를 각 트리에서 두 노드의 거리들의 합으로 정의한 거리계이다. 이러한 경우의 거리계는 K 차원 L_1 거리계를 일반화하는 것으로 간주할 수 있다. 또한 이 글에서 설명할 분할 정복 접근은 트리에서도 동일한 분할 정복 (Centroid decomposition) 을 사용하면 할 수 있다. 트리는 아니지만, 좋은 Vertex separator를 가진 다른 그래프 군에 대해서도 이러한 거리계를 적용하여 문제를 해결할 수 있는지 고민해 보는 것도 좋은 연구 주제가 될 수 있다.

Algorithm

점 집합이 $S = \{(v_{1,1}, v_{1,2}, \ldots, v_{1,d}), (v_{2,1}, v_{2,2}, \ldots, v_{2,d}), \ldots, (v_{n,1}, v_{n,2}, \ldots, v_{n,d})\}$ 로 표현된다고 하면, 각각의 d 개의 축에 대해서, 각 축의 좌표 $(v_{x,i})$ 를 기준으로 정렬한 d 개의 정렬된 리스트를 $O(dn \log n)$ 시간에 만들 수 있다. 이 각각의 리스트에 대해서 우리는 다음과 같은 크기 N 의 "분할 정복 트리" 를 만들 수 있다.

각 구간 $[L, R]$ 과 $M = (L + R)/2$ 에 대해, X_M 으로 인덱스된 노드를 처음에 만들고, 구간 $[L, M-1], [M+1, R]$ 이 각각 비지 않았다면, 각각에 대해 재귀적으로 분할 정복 트리를 만든 후 그들의 루트를 M 에 합쳐준다. 이렇게 되면 각 구간에 대해서 $R - L + 1$ 크기의 트리를 형성할 수 있고, 루트는 X_M 이 된다. 아래 그림에 이러한 트리의 예시가 있다. 이 트리의 깊이가 $\log N$ 임은 쉽게 보일 수 있다.

노드 V 에 대해, 각 d 개의 트리에 대해서 가능한 조상의 개수는 $O(\log N)$ 개이다. 이들의 모든 조합 $(O(\log^d N))$ 을 고려해 보자. 각 조합에 대해, 크기 $k + 2$ 의 튜플 $(n_1, n_2, \ldots, n_d, V, D)$ 를 구성할 수 있다. 첫 d 개의 원소 $n_1, n_2, \cdots, n_d$ 는 각 트리에서의 조상에 대응되고, 마지막에서 두 번째 원소 V 는 현재 노드의 인덱스, 그리고 마지막 원소는 $D = \sum_{i=1}^{d} v_{i,x} - v_{i,n_i}$ 가 된다. 이러한 $O(N \log^k N)$ 개의 정보를 저장해 두자.

- **Lemma 1.** 임의의 두 노드 i, j 와 d번 리스트에 대해, $V_{d,i} - V_{d,j} = V_{d,i} - V_{d,lca(i,j)} + V_{d,j} - V_{d,lca(i,j)}$ 가 성립한다. 이 때 $lca_d(i, j)$ 는 d 번 분할 정복 트리에서 i, j 의 LCA 정점을 뜻한다.
    - 증명: d 번 리스트 상에서 $lca_d(i, j)$ 가 i, j 위치 사이에 있다.

- **Lemma 2.** 임의의 두 노드 i, j 에 대해, $(lca_1(i,j), lca_2(i,j), \cdots, lca_k(i,j))$ 는 i의 조상 집합이기도 하며, j 의 조상 집합이기도 하다. (증명은 자명하여 생략)
- **Lemma 3.** $x, y, z \in R_n$ 에 대해, $x - y + y - z \geq x - z$ (삼각 부등식)

이제, 같은 조상 집합을 가진 각 튜플을 고려하자. 첫 번째 튜플이 $(n_1, n_2, \ldots, n_d, v, f_v)$, 두 번째 튜플이 $(n_1, n_2, \ldots, n_d, w, f_w)$ 라고 하면, v, w 를 잇는 가중치 $f_v + f_w$ 의 간선을 추가한다. 이렇게 할 경우, Lemma 3에 의해서, 임의의 정점 i, j 를 잇는 간선 중 이들의 실제 거리보다 작은 간선은 추가되지 않는다. 또한, Lemma 2에 의해서, i, j 를 잇는 간선 중 이들의 실제 거리를 가지는 간선 역시 추가된다. 고로, 이렇게 만들어낸 그래프에서 최소 스패닝 트리를 구하면, 이는 원래 그래프에서의 최소 스패닝 트리와 동일하다. 이러한 알고리즘은 느리니, 다음과 같은 Lemma를 사용한다.

- **Lemma 4.** N 개의 정점을 가진 완전 그래프가 있고, 어떠한 함수 $f : V \to R_{\geq 0}$ 이 존재하여 두 정점 v, w 를 잇는 간선의 가중치가 $f(v) + f(w)$ 라고 하자. 이 때의 최소 스패닝 트리는, $f(v)$ 를 최소화하는 정점 v 를 중심으로 한 star graph이다.
    - 증명: 일반성을 잃지 않고 모든 f 의 값이 다르다고 하자. v 를 지나지 않는 간선 $\{x, y\}$ 가 MST 상에 있다고 하자. Kruskal identity에 의해 $f(x) + f(y) < max(f(x) + f(v), f(y) + f(v))$ 이다. 고로 v 가 최소라는 가정에 모순이다.

고로, 각각의 튜플에 대해서 간선을 이어줄 후보는, 이 후보와 같은 조상 집합을 가지며 최소 f_i 를 가지는 정점이다. 튜플들은 Counting sort를 사용하여 $O(dN \log^d N)$ 시간에 정렬할 수 있으니, 이 후보를 찾는 것은 O(1) amortized 시간에 가능하다. 이제 다음과 같은 알려진 결과를 사용하자.

- **Theorem 1.** $m \geq n \log\log\log n$ 일 경우, $O(m)$ 시간에 최소 스패닝 트리를 찾을 수 있다 (Fredman and Tarjan, 1987)

고로, 선형 시간에 이 정점들에 대한 스패닝 트리를 찾아줄 수 있고, 최종 시간 복잡도는 $O(N \log^d N)$ 이 된다 (d는 상수라 가정).

Faster Algorithm

이 알고리즘을 $O(N \log^{d-1} N)$ 으로 최적화하는 방법을 소개한다. 이 결과는 Yuta Takaya가 인터넷으로 밝힌 다음과 같은 결과에 기반한다.

- **Theorem 2**. N 개의 정점을 가진 완전 그래프가 있고, 어떠한 가중치 있는 트리 T와 함수 $f : V \to R_{\geq 0}$ 이 존재하여 두 정점 v, w 를 잇는 간선의 가중치가 $f(v) + f(w) + dist_T(v, w)$ 라고 하자. T, f 가 주어졌을 때, 이 때의 최소 스패닝 트리는 $O(N)$ 시간에 찾을 수 있다. (Takaya 2017 (링크 참조))

이제 정확히 하나의 트리 T_d 를 제외하고, 조상들의 모든 조합 ($O(\log^{d-1} N)$) 을 고려해 보자. 각 조합에 대해, 크기 $d+1$ 의 튜플 $(n_1, n_2, \ldots, n_{d-1}, V, D)$ 를 구성할 수 있다. 첫 $d-1$ 개의 원소 $n_1, n_2, \cdots, n_{d-1}$ 는 각 트리에서의 조상에 대응되고, 마지막에서 두 번째 원소 V 는 현재 노드의 인덱스, 그리고 마지막 원소는 $D = \sum_{i=1}^{d-1} v_{i,x} - v_{i,n_i}$ 가 된다. 이러한 $O(N \log^{d-1} N)$ 개의 정보를 저장해 두자.

위와 같이, 같은 조상 집합을 가진 튜플을 고려한 후, 첫 번째 튜플이 $(n_1, n_2, \ldots, n_{d-1}, v, f_v)$, 두 번째 튜플이 $(n_1, n_2, \ldots, n_{d-1}, w, f_w)$ 라고 하면, v, w 를 잇는 가중치 $f_v + f_w + dist(v, w)$ 의 간선을 추가한다. 이렇게 한 후 Theorem 2의 결과를 그대로 사용하면 될 것 같으나 (등장하지 않은 정점에는 가중치 $f(v) = \infty$ 배정), 실제로는 이 때 소모되는 시간이 T_d 의 크기에 정확히 비례하기 때문에, 추가적으로 $O(N)$ factor가 더 붙게 되어 전혀 효율적이지 않다. 고로, 이 튜플에서 등장한 정점의 개수에 비례한 압축된 트리를 만든 후, 이 트리에서 최소 스패닝 트리를 찾아야만 한다. 압축된 트리는, 튜플에서 등장한 정점의 선형 개수에 비례하는 정점을 가지며, 튜플에서 등장한 정점 간 거리를 그대로 보존하는 성질을 가져야 한다.

각각의 튜플들을 Counting sort를 사용하여 $O(dN \log^{d-1} N)$ 시간에 정렬해 주자. 이 때, 앞 $d-1$ 개의 조상 뿐만 아니라, 정점 번호 V 역시 정렬하는데, 이 때의 정렬 기준은 마지막 트리에서의 V 의 DFS preorder 순이다. 이렇게 되었을 경우, 이 preorder를 순회해 주면, $O(N)$ 번의 LCA 쿼리만으로 "압축된" 트리를 구성할 수 있음이 알려져 있다. LCA 쿼리는 $O(N)$ 혹은 $O(N \log N)$ 시간에 전처리 해 주면 쉽게 구할 수 있으니, 이 압축된 트리에서 Theorem 2를 사용하면 최소 스패닝 트리의 superset을 찾을 수 있다. 이 간선들에서 스패닝 트리를 찾는 것은 Theorem 1을 사용하여 선형 시간에 가능하고, 최종 시간 복잡도는 $O(N \log^{d-1} N)$ 이 된다.