Top Tree 간단하게 알아보기

Lawali

ingu9981@naver.com

August 18, 2022

Overview

Top Tree란 무엇인가?

- top tree는 dynamic tree structure의 일종입니다.
- dynamic tree structure은 forest에서 간선 추가 제거 및 여러가지 쿼리를 효율적으로 처리할 수 있는 자료구조입니다.
- 잘 알려진 dynamic tree structure로는 대표적으로 link cut tree가 있습니다.

왜 Top Tree를 사용하는가?

- 대부분의 경우 link cut tree를 사용해서 여러 가지 쿼리를 amortized $O(\log N)$에 해결 가능합니다.
- 하지만 link cut tree는 임의의 정점에서 root까지의 경로를 하나의 배열로 연결하는 것을 통해 관리하므로 path query에 적합하고, subtree query에 불리합니다.
- subtree query가 불가능한 것은 아니지만 고도의 테크닉이 필요하고, 더 느립니다.

Top Tree는 무엇을 할 수 있는가?

- top tree는 간선 추가 제거를 $O(\log N)$에 가능합니다.
- top tree는 path query, subtree query를 $O(\log N)$에 가능합니다.
- 위를 활용하여 dynamic tree dp 또한 해결가능합니다.

Top Tree - 사전지식

- Link-Cut Tree의 개념 및 가능한 쿼리의 범위

Top Tree

- top tree는 **간선**에 대한 정보를 관리하는 것을 통해 전체 트리 구조를 관리합니다.
- 인접한 두 간선을 union-find을 통해 전체 트리를 하나의 cluster로 압축하는 방식으로 전체 구조가 표현되고, 최종적으로 하나의 간선으로 압축됩니다.
- 위 과정을 union-find tree로 나타내면 binary tree가 되고, 이는 splay tree와 같은 bbst를 통해 관리할 수 있습니다.

Top Tree

- union연산은 rake와 compress라는 연산을 통해서만 이루어집니다.
- 트리의 간선은 union find tree의 leaf node와 대응됩니다.
- rake는 leaf인 간선을 인접한 간선에 합쳐주는 연산입니다.
- compress는 degree가 2인 정점에서 나와있는 두 간선을 합쳐주는 연산입니다.
- 트리의 정점은 compress 연산을 통해 제거되는 정점과 대응되는 union find tree상의 정점으로 표현합니다.

Top Tree

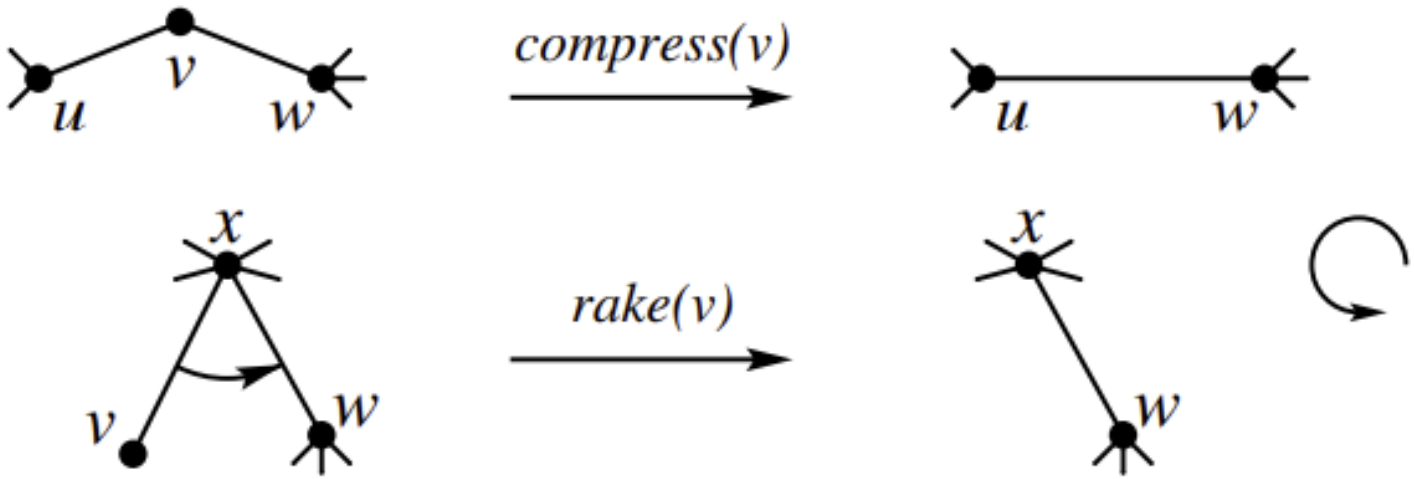

Figure 1: Basic operations.

Top Tree

- Top Tree를 build하는 과정은 다음과 같습니다.
- 먼저 아무 두 leaf node를 잡고, 그 둘 사이의 경로를 compress path 라고 합시다.
- compress path 밖에 있는 간선들을 통해 재귀적으로 top tree를 build 해 하나의 간선으로 압축합니다.
- 압축된 간선들을 compress path에 rake연산을 통해 합쳐줍니다.
- compress path의 간선들을 compress연산을 통해 합쳐줍니다.

Top Tree

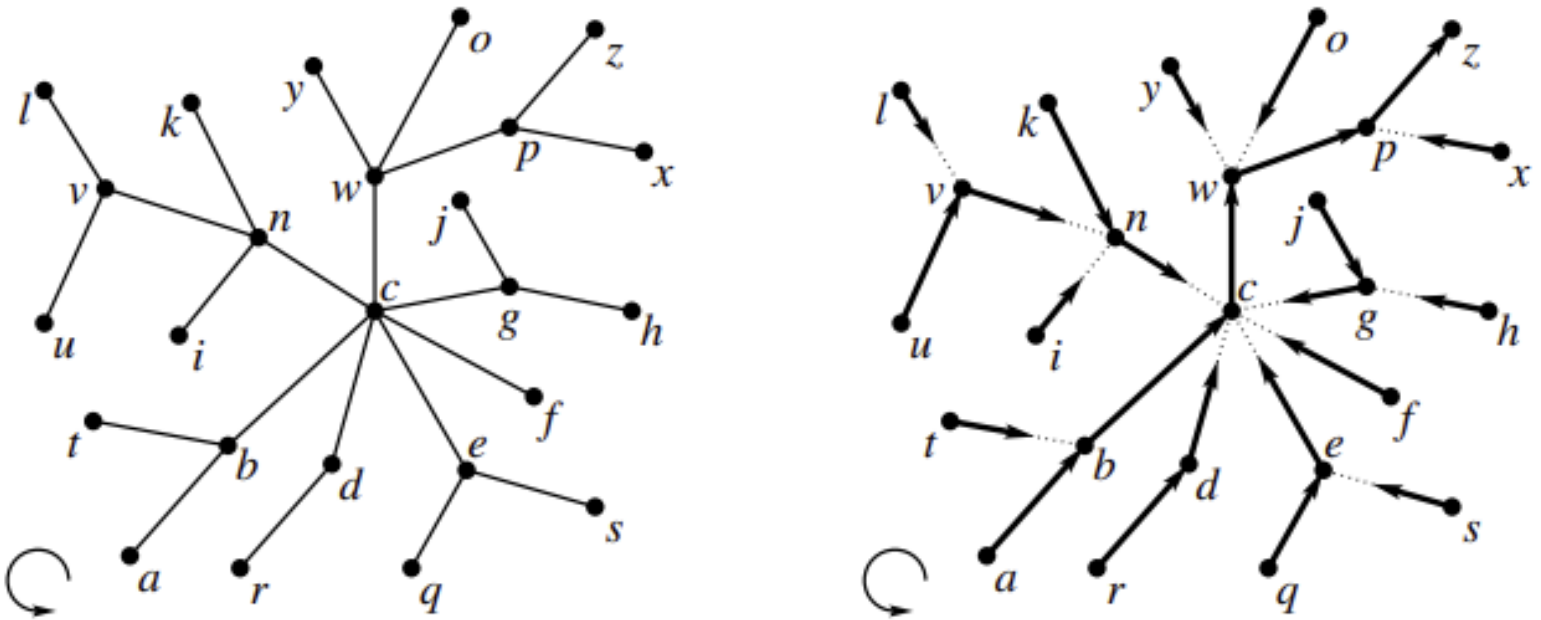

Figure 11: Example: Original tree and directed version (rooted at z and partitioned).

Top Tree

- 이제 Top Tree의 기본 operation 중 하나인 soft expose에 대해서 알아보겠습니다.
- soft expose연산은 Link Cut Tree의 access연산에 대응돠는 연산으로 여러 쿼리를 처리할 때 필수적으로 호출되는 연산입니다.
- soft_expose(u,v) 또는 soft_expose(u)가 호출되고 나면 $u-v$ path 또는 u가 Top Tree의 가장 위에 있는 compress path tree에 속하게 됩니다.
- 이중 soft_expose(u)를 설명드리겠습니다.

Top Tree

- Top Tree가 splay tree를 통해 관리된다고 가정하면 임의의 노드를 compress tree의 root로 올리는 것은 단순 splay연산으로 가능합니다.
- 현재 compress tree가 main compress tree가 아니라면 splice 연산을 통해 상위 compress tree로 올려줄 수 있습니다.
- 이는 Link Cut Tree의 access연산 중 부모 path와 연결해주는 부분과 유사합니다.

Top Tree

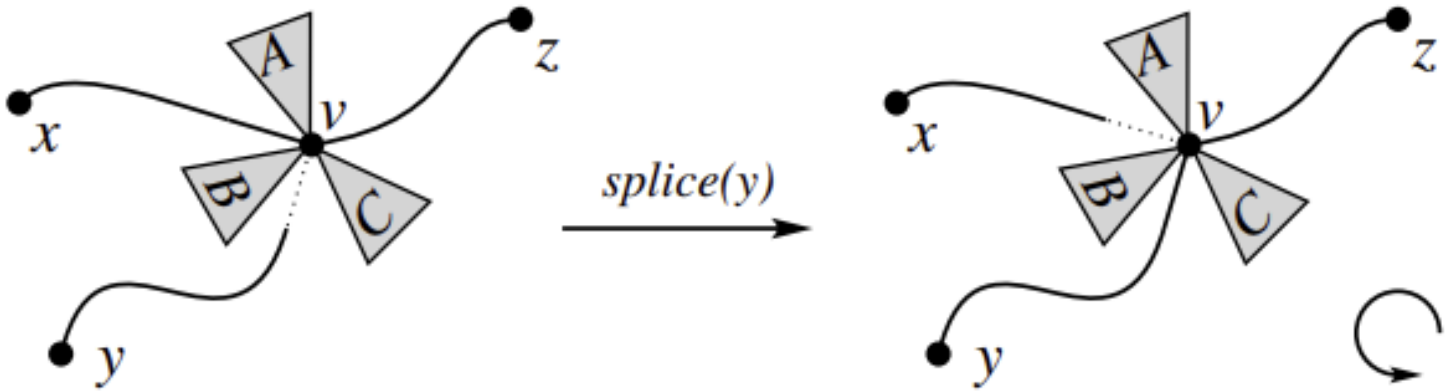

Figure 6: Splice: $y \cdots v \cdots z$ replaces $x \cdots v \cdots z$ as the exposed path.

Top Tree

- 즉 soft_expose는 splay와 splice를 반복하여 main compress tree의 루트로 올라갈 때 까지 반복합니다.
- soft_expose(u,v)는 soft_expose(u)를 통해 u를 main compress tree의 루트로 만든 뒤, u를 main compress tree의 루트로 고정한 상태로 soft_expose(v)를 통해 v를 u의 자식으로 만들어주면 됩니다.

Top Tree

- link(u,v)는 soft_expose(u)와 soft_expose(v)를 호출 뒤 두 main compress tree를 합쳐주면 됩니다.
- cut(u,v)는 soft_expose(u,v)를 호출 후 u-v간선에 해당하는 leaf node를 제거 후 두 splay tree를 분리해주면 됩니다.
- path_query(u,v)는 soft_expose(u,v)를 호출 후 splay tree상에서 구간 쿼리를 통해 구하면 됩니다.
- subtree_query(root,u)는 soft_expose(root,u)를 호출 후 u의 root방향 부모 v를 구해준 뒤 cut(u,v)를 하고 u의 트리 전체에 대한 쿼리를 구하면 됩니다.
- 위 모든 연산은 amortized $O(\log N)$입니다.

Top Tree 구현

- 실질적으로 쿼리를 구할 때 간선을 통해서 구하는 경우보다 정점 기준으로 구하는 경우가 더 많습니다.
- u-v-w가 있을 때 compress연산을 해주면 가운데 노드 v가 삭제되는데, 이 노드를 가리키는 top tree상의 노드를 v의 handle($= H_v$)이라고 부릅니다.
- 이 핸들은 각 정점마다 유일하기 때문에, 이것을 통해 정점의 정보를 저장할 수 있습니다.

Top Tree 구현

- 이 경우 리프노드의 핸들이 존재하지 않습니다.
- 포레스트에 1, 2, ..., N번 정점까지 N개의 정점이 있다고 하면, 각 정점에 대응되는 $1'$, $2'$, ... N'번 정점을 만들고, i번 정점과 i'번 정점 사이에 간선을 만들어줍니다.
- 이렇게 하면 N개의 정점 모두 리프노드가 아니게 되므로 핸들이 존재합니다.

Top Tree 구현

- 잘 생각해보면 Top Tree에서 하나의 간선에 해당하는 노드는 union find tree의 특성상 splay를 통해 루트로 올려줄 수 없고, 해당 노드가 저장하는 정보도 딱히 없습니다.
- 모든 간선에 해당하는 노드를 지워도 딱히 문제가 없습니다.

Top Tree 구현

- soft_expose(u)과정에서 splay(u) 후에 compress tree의 오른쪽 자식을 rake child로 만들어도 전체복잡도에는 문제가 없습니다.
- soft_expose(u)와 Link Cut Tree의 access(u)연산에 차이가 없어졌습니다.
- 나머지 추가적 연산은 Link Cut Tree인것마냥 구현이 가능합니다.

References

- Tarjan, Robert Werneck, Renato. (2005). Self-adjusting top trees. Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms. 813-822. 10.1145/1070432.1070547.

감사합니다