Heavy Light Decomposition

나정휘

https://justiceHui.github.io/

목차

- Prerequisites
- Motivation
- Heavy Light Decomposition
- Implementation
- Applications

Prerequisites

Prerequisites

- Union Find
  - Union by Size
  - Small to Large Technique
- Segment Tree

Prerequisites

- Union Find – Union by Size
  - Union by Size
    - 두 트리를 합칠 때, 큰 트리 밑에 작은 트리를 붙임

  - Union by Size를 하면 트리의 높이가 O(log N)인 이유
    - 어떤 트리 T_1을 T_2 밑에 붙이면, T_1에 속한 정점들의 깊이가 1씩 증가
    - $|T_1| \leq |T_2|$ 이므로 $2|T_1| \leq |T_1 \cup T_2|$
    - 깊이가 1씩 증가할 때마다 정점이 속한 트리의 크기는 2배 이상이 됨
    - 따라서 각 정점마다 깊이는 최대 O(log N)번 증가할 수 있음

Prerequisites

- Segment Tree
  - 집합 T에 속한 원소들로 구성된 배열 A[1..N]이 있다고 하자.
  - 그리고 결합 법칙이 성립하고 항등원이 있는 이항 연산자 T×T → T가 있다고 하자.
    - Monoid (T, ×)
    - 결합 법칙이 성립하므로 a×b×c = (a×b)×c = a×(b×c)
    - 항등원이 존재하므로 임의의 a ∈ T에 대해 1×a = a×1 = a를 만족하는 원소 1 ∈ T가 존재
  - 세그먼트 트리를 사용하면 아래 두 작업을 각각 O(log N)번의 × 연산만으로 수행할 수 있음
    - update(x, v) : A[x]를 v로 바꾼다.
    - query(l, r) : A[l]×A[l+1]×...×A[r]을 구한다.

Prerequisites

- Segment Tree
  - Monoid의 예시
    - add, mul, min, max, gcd, ...
  - 결합 법칙이 성립하지만 항등원이 존재하지 않는 경우
    - semigroup ⊃ monoid
    - 항등원을 의미하는 값을 강제로 넣어주면 됨
  - 수열에서 다양한 연산을 매우 효율적으로 처리할 수 있음

질문?

Motivation

Motivation

- 주의
  - HLD가 처음 연구되었을 때의 motivation과 많이 다를 수 있습니다.
  - 여러분의 이해를 돕기 위한 과정이니 양해 부탁드립니다...

Motivation

- 선형 자료구조
  - 구간에 대한 여러 연산을 효율적으로 할 수 있는 선형 자료구조는 많이 있음
    - 누적 합 배열
    - 펜윅 트리, 세그먼트 트리
    - 스플레이 트리, 트립 등 각종 BBST
    - 평방 분할
  - 트리의 경로는 LCA를 기준으로 쪼개면 선형으로 볼 수 있음
  - 트리의 경로에 대한 쿼리도 세그먼트 트리로 처리할 수 있을까?

Motivation

- 기본적인 아이디어
  - 트리를 미리 몇 개의 선형 체인으로 분할
  - 각 체인을 한 개의 세그먼트 트리로 관리

  - 체인으로 어떻게 분할해야 하지?
    - 임의의 경로가 o(N)개의 체인으로 구성되도록
    - O(sqrt N), O(log N), ...

Motivation

- 기본적인 아이디어
  - 트리를 미리 몇 개의 선형 체인으로 분할
  - 각 체인을 한 개의 세그먼트 트리로 관리

  - 체인으로 어떻게 분할해야 하지?
    - 임의의 경로가 o(N)개의 체인으로 구성되도록
    - O(sqrt N), O(log N), ...

  - Union by Size의 아이디어를 이용
    - 전체 체인 개수는 O(N)개
    - 각 경로를 구성하는 체인은 O(log N)개

질문?

Heavy Light Decomposition

Heavy Light Decomposition

- 간선의 분류
  - 정점의 무게
    - w(v) = v를 루트로 하는 서브 트리에 속한 정점의 개수

  - 간선의 분류
    - 루트가 있는 트리에서 부모 정점 p에서 자식 정점 c로 가는 간선 (p, c)
    - heavy edge $w(c) > w(p)/2$ 인 간선
    - light edge $w(c) \leq w(p)/2$ 인 간선

    - 각 정점마다 아래로 내려가는 heavy edge는 하나만 존재

Heavy Light Decomposition

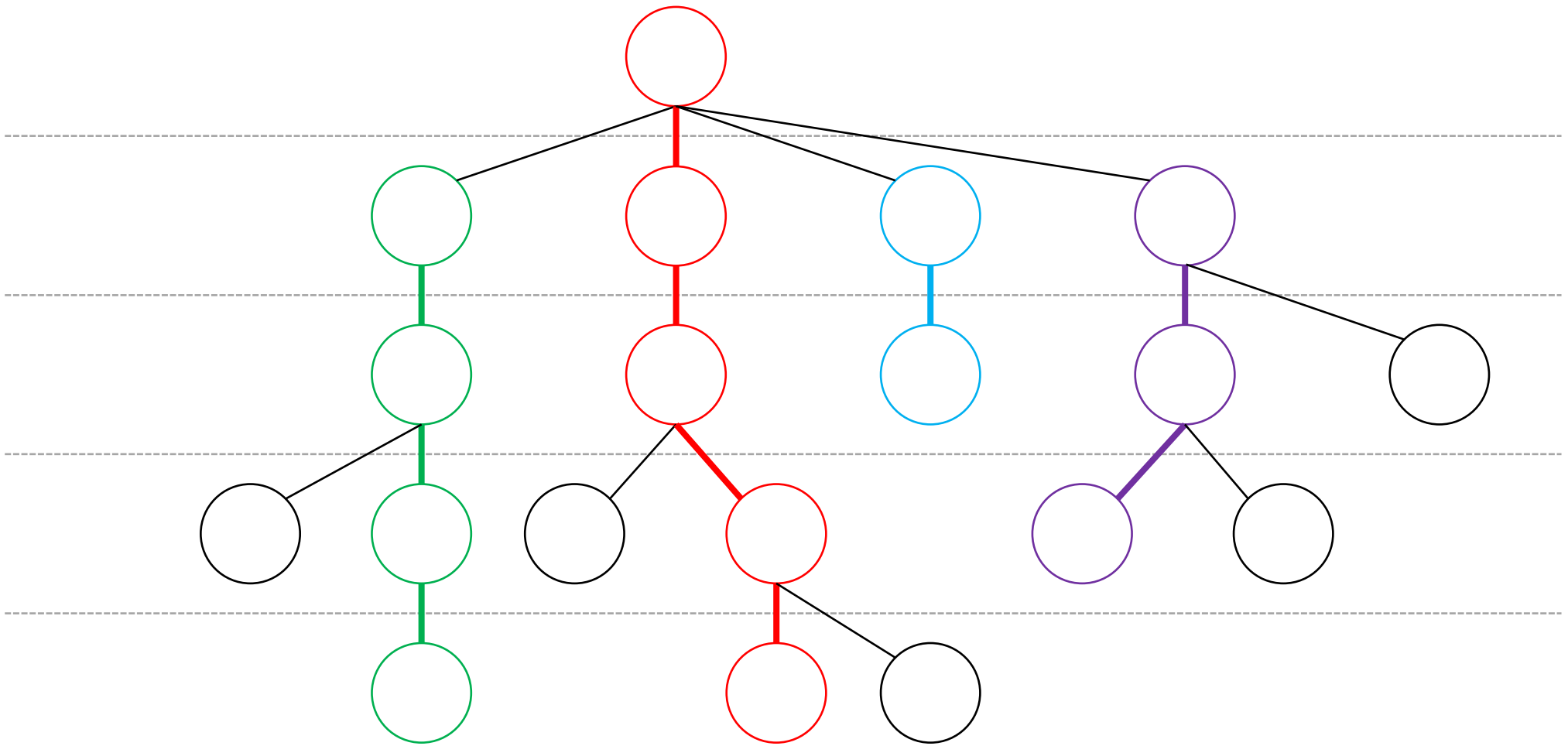
- 간선의 분류
  - 간선을 분류하면 좋은 점
    - light edge를 타고 올라갈 때마다 트리의 크기가 2배 이상이 됨
    - 임의의 정점에서 루트 정점까지 가는 경로에 light edge는 최대 O(log N)개
  - 구현
    - 실제로는 구현의 편의를 위해 서브 트리가 가장 큰 정점으로 가는 간선 하나를 heavy edge로 설정
    - 이렇게 해도 루트 정점까지 가는 경로에 있는 light edge의 개수는 O(log N)으로 동일함
      - size(c) > size(p)/2인 light edge 없음

Heavy Light Decomposition

- 간선의 분류
  - heavy chain
    - 무거운 간선들로 연결되어 있는 정점을 체인으로 묶음
    - 모든 경로는 최대 2*O(log N) = O(log N)개의 체인으로 구성

    - 경로에 대한 쿼리를 O(log N)번의 세그먼트 트리 연산으로 처리 가능

Heavy Light Decomposition



질문?

Implementation

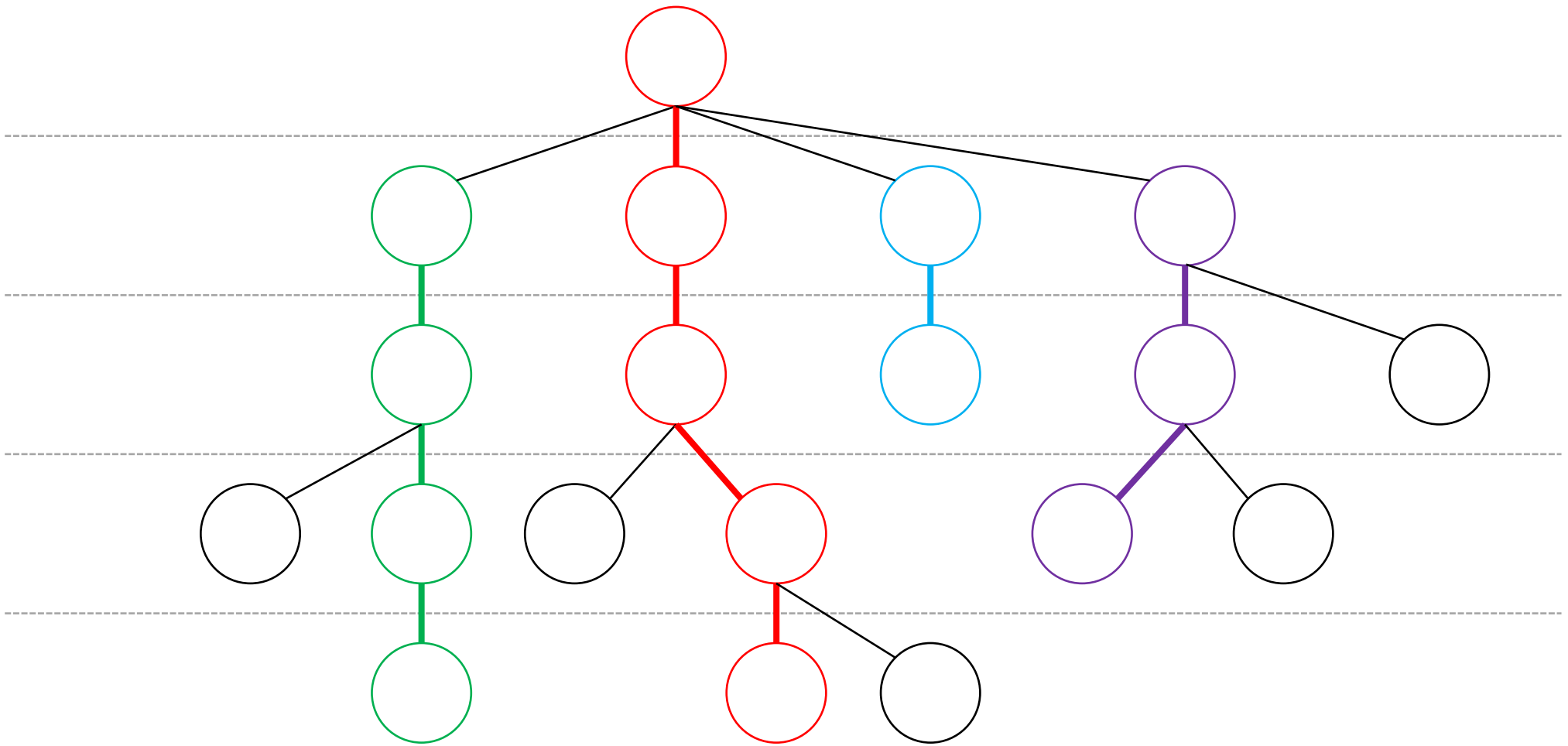
Implementation

- 구현해야 하는 것
  - 트리를 체인으로 분할
  - 트리의 정점을 세그먼트 트리의 리프 정점에 매핑
  - 경로 쿼리

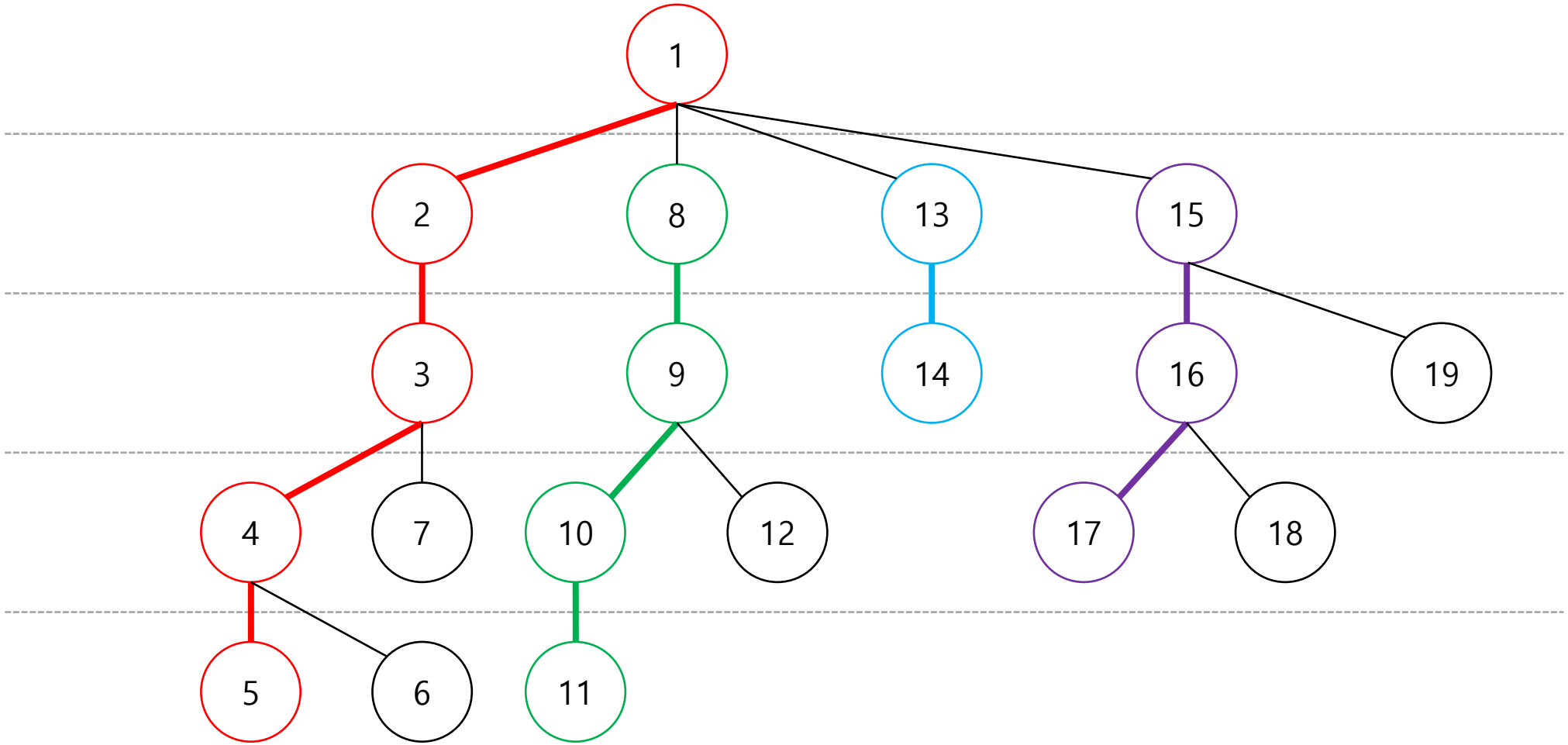
Implementation

- 분할
  - 가장 무거운 자식 정점이 인접 리스트의 맨 앞에 오도록 순서를 바꿈
    - G[v][0] = v에서 내려가는 heavy edge
  - 한 체인에 속한 정점은 dfs ordering 상에서 인접한 위치에 있음
    - 세그먼트 트리를 한 개만 관리해도 됨
    - 서브 트리 쿼리(euler tour technique)도 동시에 처리 가능

Heavy Light Decomposition



Heavy Light Decomposition



Heavy Light Decomposition

```cpp
/*
 * Sz[v] : v를 루트로 하는 서브 트리의 크기
 * Dep[v] : v의 깊이
 * Par[v] : v의 부모 정점
 * Top[v] : v가 속한 체인에서 맨 위에 있는 정점
 * In[v] : DFS에서 v에 들어가는 시간
 * Out[v] : DFS에서 v를 빠져 나오는 시간
 * G[v] : v의 자식 정점
 */
int Sz[MAX_V], Dep[MAX_V], Par[MAX_V], Top[MAX_V], In[MAX_V], Out[MAX_V];
vector<int> G[MAX_V];

void DFS1(int v){
    Sz[v] = 1;
    for(auto &i : G[v]){
        Dep[i] = Dep[v] + 1; Par[i] = v;
        DFS1(i); Sz[v] += Sz[i];
        if(Sz[i] > Sz[G[v][0]]) swap(i, G[v][0]); // 가장 무거운 정점이 맨 앞으로
    }
}

void DFS2(int v){
    static int cnt = 0;
    In[v] = ++cnt;
    for(auto i : G[v]){
        // 간선 (v, i)가 light edge면 i가 새로운 체인의 top, heavy edge면 v와 같은 체인
        Top[i] = i == G[v][0] ? Top[v] : i;
        DFS2(i);
    }
    Out[v] = cnt;
}
```

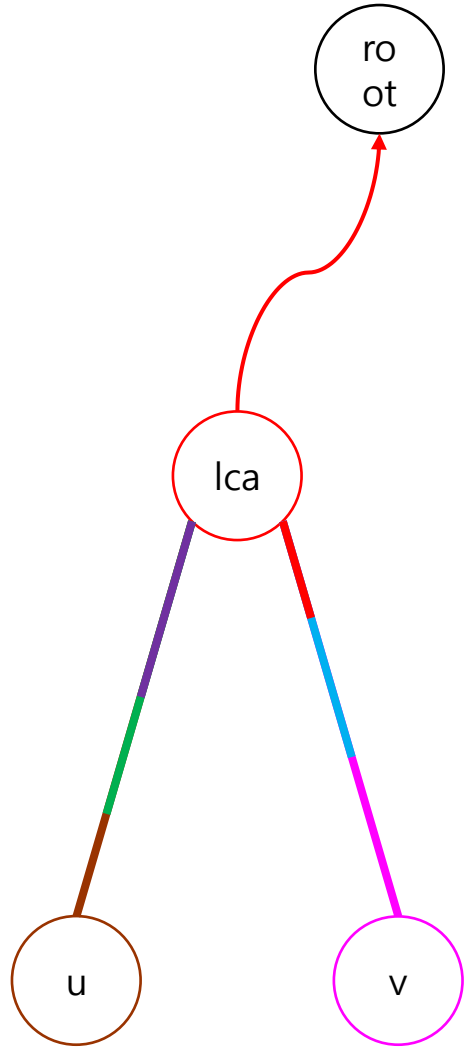
질문?

Heavy Light Decomposition

- 경로 쿼리
  - 경로를 LCA 기준으로 나눠서 생각하자.
    - u, v에서 시작해서 체인을 타고 올라가다가
    - LCA가 속한 체인에서 만나고 종료

    - Top[u]와 Top[v] 중 더 깊이 있는 정점의 체인부터 타고 올라가면 됨
    - Dep[Top[u]] > Dep[Top[v]] 이면 In[Top[u]]부터 In[u]까지 쿼리

    - Top[u] = Top[v]이면 u와 v가 LCA를 포함하는 같은 체인에 있는 상태
    - In[u] < In[v] 이면 In[u]부터 In[v]까지 쿼리



Heavy Light Decomposition

```cpp
int PathQuery(int u, int v){
    int ret = 0;
    for(; Top[u] != Top[v]; u=Par[Top[u]]){
        if(Dep[Top[u]] < Dep[Top[v]]) swap(u, v);
        ret += SegQuery(In[Top[u]], In[u]);
    }
    if(In[u] > In[v]) swap(u, v);
    ret += SegQuery(In[u], In[v]);
    return ret;
}
```

Heavy Light Decomposition

- 간선에 가중치가 붙은 경우
  - 지금까지 설명한 내용은 모두 정점에 가중치가 붙어있는 경우
  - 간선에 가중치가 붙어 있으면 어떻게 할까?

  - 루트를 제외한 모든 정점은 위로 올라가는 간선을 정확히 1개 갖고 있음
  - 정점에 가중치가 붙어있는 상황으로 바꿀 수 있음

  - 주의
    - LCA의 부모로 올라가는 간선은 포함하면 안 됨
    - 마지막에 SegQuery(In[u]+1, In[v])

질문?

Applications

Applications

- BOJ 11438 LCA 2
  - 트리에서 두 정점의 LCA를 구하는 문제

  - u, v가 같은 체인에 속한다면 둘 중 더 높이 있는 정점이 LCA
  - O(log N)에 구할 수 있고, 일반적으로 Sparse Table을 사용한 것보다 빠름

Applications

- BOJ 13510 트리와 쿼리 1
  - 간선의 가중치를 바꾸는 쿼리, 경로의 가중치 최댓값을 구하는 쿼리

  - 구현 연습 해보세요

Applications

- BOJ 13309 트리
  - 두 정점이 연결되어 있는지 확인하는 쿼리
  - 간선을 제거하는 쿼리

  - 트리에서 두 정점을 잇는 경로는 유일함
  - 경로를 구성하는 간선 중 하나라도 끊어져 있으면 연결 X

Applications

- BOJ 13512 트리와 쿼리 3
  - 정점의 색을 바꾸는 쿼리
  - 루트에서 v번 정점까지 가는 경로에서 처음으로 만나는 검은색 정점을 구하는 쿼리

  - 구간이 주어지면 가장 앞에 있는 검은색의 위치를 구하는 세그먼트 트리
    - 각 정점에서 구간의 합을 관리
    - sum(l, m) > 0이면 왼쪽 정점, sum(m+1, r)이면 오른쪽 정점

Applications

- BOJ 17429 국제 메시 기구
  - 경로 쿼리와 서브 트리 쿼리를 모두 해야 하는 문제
    - 서브 트리에 대한 쿼리는 Euler Tour Technique를 이용하면 구간에 대한 쿼리로 바꿀 수 있음
  - T[x]를 a * T[x] + b로 바꾸는 lazy propagation
    - BOJ 13925 수열과 쿼리 13 참고

Applications

- BOJ 13519 트리와 쿼리 10
  - BOJ 16993 연속합과 쿼리 문제를 트리에서 푸는 문제

  - 교환 법칙이 성립하지 않기 때문에 체인을 합치는 순서와 방향을 잘 고려해야 함
    - u에서 LCA로 올라가는 방향
    - LCA에서 v로 내려가는 방향

질문?

Applications

- Tree DP
  - 많은 Tree DP 문제는 두 정점의 DP값을 합치는 방식으로 점화식을 계산함
    - 두 트리 A, B의 DP 값을 합칠 때 |A|+|B| 만큼의 시간이 걸리는 경우
    - 전체 문제를 $O(N \log^2 N)$에 해결할 수 있음

  - 간단한 케이스
    - 트리가 일직선이면 분할 정복을 이용해 O(N log N)에 계산할 수 있음
      - merge sort를 생각해 보자.
    - 트리가 성게 그래프이면 분할 정복을 이용해 O(N log N)에 계산할 수 있음
      - 일직선과 동일한 방법

Applications

- Tree DP
  - Step 1
    - 트리의 간선을 heavy edge와 light edge로 구분
    - 루트가 속한 체인의 각 정점마다, light edge에 달려 있는 서브 트리에 대해 재귀적으로 해결
  - Step 2
    - 트리에 heavy chain과, 체인에 달려 있는 크기 1 짜리 light edge만 남아 있는 상태
    - 각 정점에 달려 있는 light edge만 고려하면 성게 그래프이므로 분할 정복하면 $O(N \log^2 N)$
  - Step 3
    - heavy chain만 남아 있는 상태
    - 체인은 일직선이므로 분할 정복하면 $O(N \log^2 N)$

Applications

- BOJ 18477 Jiry Matchings
  - 트리에서 크기가 1, 2, … , N-1인 매칭의 최대 가중치를 구하는 문제

  - 트리의 가중치 매칭은 MCMF로 모델링할 수 있음
    - Why? 트리는 이분 그래프
    - 정답은 볼록 함수 형태

Applications

- BOJ 18477 Jiry Matchings
  - D[v][k] = v를 루트로 하는 서브 트리에서 매칭을 k개 만들었을 때의 최대 가중치
    - $f_v(k)$ = D[v][k]는 볼록 함수
    - $D[p][k_1+k_2] \leftarrow D[u][k_1] + D[v][k_2]$
    - 기울기가 큰 것부터 하나씩 끼워 넣는다고 생각하면 O(|A|+|B|) 시간에 합칠 수 있음
      - merge sort를 생각해 보자.
  - 두 트리의 DP값을 |A|+|B| 시간에 합칠 수 있으므로 $O(N \log^2 N)$에 풀 수 있음

질문?