Competitive Programming
Winter Term 23/24

Trees

Kirill Simonov
Algorithm Engineering Group
HPI

Michael Zündorf
Scalable Algorithms Group
KIT
Topics today

- today
  - lowest common ancestor
  - binary lifting
  - heavy-light decomposition
  - centroid decomposition

- not covered
  - fast LCA
  - tree rerooting
In this lecture
In this lecture

Queries on arrays: max, sum, update etc.
In this lecture

Queries on arrays:

Queries on trees:
In this lecture

Queries on arrays:

Queries on trees:
Problem: We are given a tree. Answer queries: given $u$ and $v$, what is the distance between the two nodes?
Lowest Common Ancestor

**Problem:** We are given a tree. Answer queries: given $u$ and $v$, what is the distance between the two nodes?

**Idea:** split path queries into two paths
Lowest Common Ancestor

**Problem:** We are given a tree. Answer queries: given $u$ and $v$, what is the distance between the two nodes?

**Idea:** split path queries into two paths

$LCA(u, v)$: lowest node on both paths to the root
Lowest Common Ancestor

**Problem:** We are given a tree. Answer queries: given \( u \) and \( v \), what is the distance between the two nodes?

**Idea:** split path queries into two paths

\( LCA(u, v) \): lowest node on both paths to the root

How to find LCA? Build *Euler tour* of the tree:
Lowest Common Ancestor

**Problem:** We are given a tree. Answer queries: given $u$ and $v$, what is the distance between the two nodes?

**Idea:** split path queries into two paths

$LCA(u, v)$: lowest node on both paths to the root

How to find LCA? Build *Euler tour* of the tree:
Lowest Common Ancestor

**Problem**: We are given a tree. Answer queries: given \( u \) and \( v \), what is the distance between the two nodes?

**Idea**: split path queries into two paths

\( LCA(u, v) \): lowest node on both paths to the root

How to find LCA? Build *Euler tour* of the tree:
Problem: We are given a tree. Answer queries: given \( u \) and \( v \), what is the distance between the two nodes?

Idea: split path queries into two paths

LCA(\( u, v \)): lowest node on both paths to the root

How to find LCA? Build Euler tour of the tree:

\[ 1 \, 2 \, 1 \]
**Problem:** We are given a tree. Answer queries: given $u$ and $v$, what is the distance between the two nodes?

**Idea:** split path queries into two paths

$LCA(u, v)$: lowest node on both paths to the root

How to find LCA? Build *Euler tour* of the tree:

1 2 1 3
Lowest Common Ancestor

**Problem:** We are given a tree. Answer queries: given \( u \) and \( v \), what is the distance between the two nodes?

**Idea:** split path queries into two paths

\[ \text{LCA}(u, v): \text{lowest node on both paths to the root} \]

How to find LCA? Build *Euler tour* of the tree:
Lowest Common Ancestor

**Problem:** We are given a tree. Answer queries: given $u$ and $v$, what is the distance between the two nodes?

**Idea:** split path queries into two paths

$LCA(u, v)$: lowest node on both paths to the root

How to find LCA? Build *Euler tour* of the tree:

1 2 1 3 4 7 4 8 4 3 5 3 6 9 6 3 1
Lowest Common Ancestor

**Problem:** We are given a tree. Answer queries: given $u$ and $v$, what is the distance between the two nodes?

**Idea:** split path queries into two paths

$LCA(u, v)$: lowest node on both paths to the root

How to find LCA? Build *Euler tour* of the tree:

```plaintext
1 2 1 3 4 7 4 8 4 3 5 3 6 9 6 3 1
```

Length?
Lowest Common Ancestor

**Problem:** We are given a tree. Answer queries: given $u$ and $v$, what is the distance between the two nodes?

**Idea:** split path queries into two paths

$LCA(u, v)$: lowest node on both paths to the root

How to find LCA? Build *Euler tour* of the tree:

Length? $2n - 1$
Lowest Common Ancestor

**Problem:** We are given a tree. Answer queries: given $u$ and $v$, what is the distance between the two nodes?

**Idea:** split path queries into two paths

$LCA(u, v)$: lowest node on both paths to the root

How to find LCA? Build *Euler tour* of the tree:

Length? $2n - 1$

Subtrees?
Lowest Common Ancestor

**Problem:** We are given a tree. Answer queries: given $u$ and $v$, what is the distance between the two nodes?

**Idea:** split path queries into two paths

$LCA(u, v)$: lowest node on both paths to the root

How to find LCA? Build *Euler tour* of the tree:

Length? $2n - 1$

Subtrees? Subtrees are subarrays
**Problem:** We are given a tree. Answer queries: given $u$ and $v$, what is the distance between the two nodes?

**Idea:** split path queries into two paths

$LCA(u, v)$: lowest node on both paths to the root

How to find LCA? Build *Euler tour* of the tree:

Length? $2n - 1$

Subtrees? Subtrees are subarrays

Paths?
Lowest Common Ancestor

**Problem:** We are given a tree. Answer queries: given \( u \) and \( v \), what is the distance between the two nodes?

**Idea:** split path queries into two paths

\[ LCA(u, v): \text{lowest node on both paths to the root} \]

How to find LCA? Build *Euler tour* of the tree:

Length? \( 2n - 1 \)

Subtrees? Subtrees are subarrays

Paths? Subarray contains path (plus subtrees)
Lowest Common Ancestor

**Problem:** We are given a tree. Answer queries: given $u$ and $v$, what is the distance between the two nodes?

**Idea:** split path queries into two paths

$LCA(u, v)$: lowest node on both paths to the root

How to find LCA? Build *Euler tour* of the tree:

Length? $2n - 1$

Subtrees? Subtrees are subarrays

Paths? Subarray contains path (plus subtrees)

LCA?
Lowest Common Ancestor

**Problem:** We are given a tree. Answer queries: given $u$ and $v$, what is the distance between the two nodes?

**Idea:** split path queries into two paths

$LCA(u, v)$: lowest node on both paths to the root

How to find LCA? Build *Euler tour* of the tree:

- **Length?** $2n - 1$

- **Subtrees?** Subtrees are subarrays

- **Paths?** Subarray contains path (plus subtrees)

$LCA(7, 6) = 3$, $LCA(7, 8) = 3$, $LCA(3, 8) = 3$
**Problem:** We are given a tree. Answer queries: given $u$ and $v$, what is the distance between the two nodes?

**Idea:** split path queries into two paths

$LCA(u, v)$: lowest node on both paths to the root

How to find LCA? Build *Euler tour* of the tree:

Subtrees? Subtrees are subarrays

Paths? Subarray contains path (plus subtrees)

$LCA? \ 0 \ 1 \ 0 \ 1 \ 2 \ 3 \ 2 \ 3 \ 2 \ 1 \ 2 \ 1 \ 2 \ 3 \ 2 \ 1 \ 0$

Length? $2n - 1$
Lowest Common Ancestor

- Build Euler tour
Lowest Common Ancestor

- Build Euler tour

- Build data structure on tour
Lowest Common Ancestor

- Build Euler tour

- Build data structure on tour
- Answer LCA queries with data structure
Lowest Common Ancestor

- Build Euler tour
  ```cpp
  void dfs(Graph &adj, int v, int p = -1, int d = 0) {
    depth[v] = d;
    first[v] = euler.size();
    euler.push_back(v);
    for (auto nei : adj[node])
      if (nei != p) {
        dfs(adj, nei, v, d + 1);
        euler.push_back(v);
      }
  }
  ```

- Build data structure on tour

- Answer LCA queries with data structure
Lowest Common Ancestor

- Build Euler tour

```cpp
void dfs(Graph &adj, int v, int p = -1, int d = 0) {
    depth[v] = d;
    first[v] = euler.size();
    euler.push_back(v);
    for (auto nei : adj[node])
        if (nei != p) {
            dfs(adj, nei, v, d + 1);
            euler.push_back(v);
        }
}
```

Range minimum queries on static array

- Build data structure on tour

- Answer LCA queries with data structure
Lowest Common Ancestor

- Build Euler tour
  
  ```cpp
  void dfs(Graph &adj, int v, int p = -1, int d = 0) {
    depth[v] = d;
    first[v] = euler.size();
    euler.push_back(v);
    for (auto nei : adj[node])
      if (nei != p) {
        dfs(adj, nei, v, d + 1);
        euler.push_back(v);
      }
  }
  ```

- Build data structure on tour

- Answer LCA queries with data structure

Range minimum queries on static array
e.g., segment tree, sparse table
Lowest Common Ancestor

- Build Euler tour

```cpp
void dfs(Graph &adj, int v, int p = -1, int d = 0) {
    depth[v] = d;
    first[v] = euler.size();
    euler.push_back(v);
    for (auto nei : adj[node])
        if (nei != p) {
            dfs(adj, nei, v, d + 1);
            euler.push_back(v);
        }
}
```

- Build data structure on tour
- Answer LCA queries with data structure

\[ \text{LCA}(u, v) = \text{RMQ}(\text{first}[u], \text{first}[v]) \]
Lowest Common Ancestor

- Build Euler tour

\[
\text{void dfs(Graph &adj, int v, int p = -1, int d = 0) \{ }
\]
\[
\quad \text{depth[v] = d;}
\]
\[
\quad \text{first[v] = euler.size();}
\]
\[
\quad \text{euler.push_back(v);}
\]
\[
\quad \text{for (auto nei : adj[node])}
\]
\[
\qquad \text{if (nei != p) \{}
\]
\[
\quad \quad \text{dfs(adj, nei, v, d + 1);}
\]
\[
\quad \quad \text{euler.push_back(v);}
\]
\[
\quad \}
\]
\[
\text{\}}
\]

- Build data structure on tour

- Answer LCA queries with data structure

\[
\text{LCA(u, v) = RMQ(first[u], first[v])}
\]
\[
\text{dist(u, v) = depth[u] + depth[v] - 2 \cdot depth[LCA(u, v)]}
\]

Range minimum queries on static array
e.g., segment tree, sparse table
Lowest Common Ancestor

- Build Euler tour

```cpp
void dfs(Graph &adj, int v, int p = -1, int d = 0) {
    depth[v] = d;
    first[v] = euler.size();
    euler.push_back(v);
    for (auto nei : adj[node])
        if (nei != p) {
            dfs(adj, nei, v, d + 1);
            euler.push_back(v);
        }
}
```

- Build data structure on tour

- Answer LCA queries with data structure

\[
\text{LCA}(u, v) = \text{RMQ}(\text{first}[u], \text{first}[v])
\]

\[
\text{dist}(u, v) = \text{depth}[u] + \text{depth}[v] - 2 \cdot \text{depth}[	ext{LCA}(u, v)]
\]

Range minimum queries on static array

e.g., segment tree, sparse table

\(O(n)\)
Lowest Common Ancestor

- Build Euler tour

  ```
  void dfs(Graph &adj, int v, int p = -1, int d = 0) {
    depth[v] = d;
    first[v] = euler.size();
    euler.push_back(v);
    for (auto nei : adj[node])
      if (nei != p) {
        dfs(adj, nei, v, d + 1);
        euler.push_back(v);
      }
  }
  ```

  $O(n)$

  Range minimum queries on static array
  e.g., segment tree, sparse table
  $O(n \log n)$

- Build data structure on tour

- Answer LCA queries with data structure

  \[
  \text{LCA}(u, v) = \text{RMQ}(\text{first}[u], \text{first}[v])
  \]
  \[
  \text{dist}(u, v) = \text{depth}[u] + \text{depth}[v] - 2 \cdot \text{depth}[\text{LCA}(u, v)]
  \]
Lowest Common Ancestor

- Build Euler tour

  ```cpp
  void dfs(Graph &adj, int v, int p = -1, int d = 0) {
    depth[v] = d;
    first[v] = euler.size();
    euler.push_back(v);
    for (auto nei : adj[node])
      if (nei != p) {
        dfs(adj, nei, v, d + 1);
        euler.push_back(v);
      }
  }
  ```

  \( O(n) \)

- Build data structure on tour

- Answer LCA queries with data structure

  \[ \text{LCA}(u, v) = \text{RMQ}(\text{first}[u], \text{first}[v]) \]

  \[ \text{dist}(u, v) = \text{depth}[u] + \text{depth}[v] - 2 \cdot \text{depth}[\text{LCA}(u, v)] \]

  \( O(\log n) / O(1) \)

  e.g., segment tree, sparse table

  \( O(n \log n) \)
Binary lifting
Binary lifting

- How to find LCA naively?
Binary lifting

- How to find LCA naively?
  - Calculate depths
  - Move lower node \((u, v)\) to its parent
  - Repeat until \(u = v\)
Binary lifting

- How to find LCA naively?
  - Calculate depths
  - Move lower node \((u, v)\) to its parent
  - Repeat until \(u = v\)

- How to make fast jumps of arbitrary size?
Binary lifting

- How to find LCA naively?
  - Calculate depths
  - Move lower node \((u, v)\) to its parent
  - Repeat until \(u = v\)

- How to make fast jumps of arbitrary size?
  
  Make binary jumps!
Binary lifting

- How to find LCA naively?
  - Calculate depths
  - Move lower node \((u, v)\) to its parent
  - Repeat until \(u = v\)

- How to make fast jumps of arbitrary size?
  Make binary jumps!
  - 5 jumps? \(4 + 1\) jumps!
Binary lifting

- How to find LCA naively?
  - Calculate depths
  - Move lower node \((u, v)\) to its parent
  - Repeat until \(u = v\)

- How to make fast jumps of arbitrary size?
  Make binary jumps!
  - 5 jumps? \(4 + 1\) jumps! \(O(\log n)\)
Binary lifting

- How to find LCA naively?
  - Calculate depths
  - Move lower node \((u, v)\) to its parent
  - Repeat until \(u = v\)

- How to make fast jumps of arbitrary size?
  Make binary jumps!
  - 5 jumps? 4 + 1 jumps! \(O(\log n)\)

- Calculate depths with DFS
- Calculate ancestors in powers of two
Binary lifting

- How to find LCA naively?
  - Calculate depths
  - Move lower node \((u, v)\) to its parent
  - Repeat until \(u = v\)

- How to make fast jumps of arbitrary size?
  Make binary jumps!
  - 5 jumps? 4 + 1 jumps! \(O(\log n)\)

- Calculate depths with DFS
- Calculate ancestors in powers of two

For queries:
- Binary jumps from lower node until depth\([u]\) = depth\([v]\)
- Binary jumps (decreasing size) until same ancestor
Binary lifting

- How to find LCA naively?
  - Calculate depths
  - Move lower node \((u, v)\) to its parent
  - Repeat until \(u = v\)

- How to make fast jumps of arbitrary size?
  Make binary jumps!
  - 5 jumps? 4 + 1 jumps! \(O(\log n)\)

- Calculate depths with DFS \(O(n)\)
- Calculate ancestors in powers of two \(O(n \log n)\)

For queries:
- Binary jumps from lower node until \(\text{depth}[u] = \text{depth}[v]\)
- Binary jumps (decreasing size) until same ancestor

\(O(n)\)
Binary lifting

How to find LCA naively?
- Calculate depths
- Move lower node \((u, v)\) to its parent
- Repeat until \(u = v\)

How to make fast jumps of arbitrary size?
Make binary jumps!
- 5 jumps? 4 + 1 jumps! \(O(\log n)\)

Calculate depths with DFS \(O(n)\)
Calculate ancestors in powers of two \(O(n \log n)\)

For queries:
- Binary jumps from lower node until \(\text{depth}[u] = \text{depth}[v]\) \(O(\log n)\)
- Binary jumps (decreasing size) until same ancestor \(O(\log n)\)
Binary lifting

- How to find LCA naively?
  - Calculate depths
  - Move lower node \((u, v)\) to its parent
  - Repeat until \(u = v\)

- How to make fast jumps of arbitrary size?
  
  Make binary jumps!
  
  - 5 jumps? 4 + 1 jumps! \(O(\log n)\)

- Calculate depths with DFS \(O(n)\)
- Calculate ancestors in powers of two \(O(n \log n)\)

For queries:
  - Binary jumps from lower node until depth\([u]\) = depth\([v]\) \(O(\log n)\)
  - Binary jumps (decreasing size) until same ancestor \(O(\log n)\)
Binary lifting

- How to find LCA naively?
  - Calculate depths
  - Move lower node \((u, v)\) to its parent
  - Repeat until \(u = v\)

- How to make fast jumps of arbitrary size?
  Make binary jumps!
  - 5 jumps? \(4 + 1\) jumps! \(\mathcal{O}(\log n)\)

- Calculate depths with DFS \(\mathcal{O}(n)\)
- Calculate ancestors in powers of two \(\mathcal{O}(n \log n)\)

For queries:
- Binary jumps from lower node until depth\([u]\) = depth\([v]\) \(\mathcal{O}(\log n)\)
- Binary jumps (decreasing size) until same ancestor \(\mathcal{O}(\log n)\)
Binary lifting

- How to find LCA naively?
  - Calculate depths
  - Move lower node \((u, v)\) to its parent
  - Repeat until \(u = v\)

- How to make fast jumps of arbitrary size?
  Make binary jumps!
  - 5 jumps? 4 + 1 jumps! \(O(\log n)\)

- Calculate depths with DFS \(O(n)\)
- Calculate ancestors in powers of two \(O(n \log n)\)

For queries:
- Binary jumps from lower node until depth\([u]\) = depth\([v]\) \(O(\log n)\)
- Binary jumps (decreasing size) until same ancestor \(O(\log n)\)
Binary lifting

- How to find LCA naively?
  - Calculate depths
  - Move lower node \((u, v)\) to its parent
  - Repeat until \(u = v\)

- How to make fast jumps of arbitrary size?
  Make binary jumps!
  - 5 jumps? 4 + 1 jumps! \(O(\log n)\)

- Calculate depths with DFS \(O(n)\)
- Calculate ancestors in powers of two \(O(n \log n)\)

For queries:
- Binary jumps from lower node until depth\([u]\) = depth\([v]\) \(O(\log n)\)
- Binary jumps (decreasing size) until same ancestor \(O(\log n)\)
Ancestors for binary lifting

- Alternative LCA finding

```
1 2 1 3 4 7 4 8 4 3 5 3 6 9 6 3 1
```

```
Ancestors for binary lifting

- Alternative LCA finding
- In Euler tour, let’s save first[v] and last[v]
Ancestors for binary lifting

- Alternative LCA finding
- In Euler tour, let’s save `first[v]` and `last[v]`
- `u` is ancestor of `v` ⇔ `first[u] ≤ first[v] ≤ last[u]`
Ancestors for binary lifting

- Alternative LCA finding
- In Euler tour, let’s save $first[v]$ and $last[v]$
- $u$ is ancestor of $v \Leftrightarrow first[u] \leq first[v] \leq last[u]$

- Binary jumps (decreasing size) to find node below LCA
Ancestors for binary lifting

- Alternative LCA finding
- In Euler tour, let’s save \( \text{first}[v] \) and \( \text{last}[v] \)
- \( u \) is ancestor of \( v \) \( \iff \) \( \text{first}[u] \leq \text{first}[v] \leq \text{last}[u] \)

- Binary jumps (decreasing size) to find node below LCA
  - Jump starting at \( u \) if not ancestor of \( v \)
Ancestors for binary lifting

- Alternative LCA finding
- In Euler tour, let’s save $first[v]$ and $last[v]$
- $u$ is ancestor of $v \iff first[u] \leq first[v] \leq last[u]$

- Binary jumps (decreasing size) to find node below LCA
  - Jump starting at $u$ if not ancestor of $v$
Alternative LCA finding

In Euler tour, let’s save first[v] and last[v]

$u$ is ancestor of $v \iff$ first[$u$] $\leq$ first[$v$] $\leq$ last[$u$]

Binary jumps (decreasing size) to find node below LCA

Jump starting at $u$ if not ancestor of $v$
Ancestors for binary lifting

- Alternative LCA finding
- In Euler tour, let’s save first[v] and last[v]
  
- u is ancestor of v ⇔ first[u] ≤ first[v] ≤ last[u]

- Binary jumps (decreasing size) to find node below LCA
  
  - Jump starting at u if not ancestor of v
Ancestors for binary lifting

- Alternative LCA finding
- In Euler tour, let’s save $first[v]$ and $last[v]$
- $u$ is ancestor of $v$ $\iff$ $first[u] \leq first[v] \leq last[u]$

- Binary jumps (decreasing size) to find node below LCA
  - Jump starting at $u$ if not ancestor of $v$
Ancestors for binary lifting

- Alternative LCA finding
- In Euler tour, let’s save $first[v]$ and $last[v]$
- $u$ is ancestor of $v \iff first[u] \leq first[v] \leq last[u]$

- Binary jumps (decreasing size) to find node below LCA
  - Jump starting at $u$ if not ancestor of $v$
  - Parent of ending node is LCA
Ancestors for binary lifting

- Alternative LCA finding
- In Euler tour, let’s save first[v] and last[v]
  
\[
\begin{align*}
\text{first}[v] \leq \text{last}[u]
\end{align*}
\]

- u is ancestor of v ⇔ first[u] ≤ first[v] ≤ last[u]

- Binary jumps (decreasing size) to find node below LCA
  
  - Jump starting at u if not ancestor of v
  - Parent of ending node is LCA

- Binary lifting in \(O(n)\) memory is possible
  
  - https://codeforces.com/blog/entry/74847
LCA/binary lifting applications
LCA/binary lifting applications

- $LCA(u, v)$?
LCA/binary lifting applications

- \( LCA(u, v) \)  
  Euler tour/binary lifting
LCA/binary lifting applications

- $LCA(u, v)$?
- $dist(u, v)$?

Euler tour/binary lifting
LCA/binary lifting applications

- \( LCA(u, v)? \) Euler tour/binary lifting
- \( dist(u, v)? \) Euler tour/binary lifting + depths
LCA/binary lifting applications

- $\text{LCA}(u, v)$? Euler tour/binary lifting
- $\text{dist}(u, v)$? Euler tour/binary lifting + depths
- sum of edge weights on $(u, v)$ path?
LCA/binary lifting applications

- \( LCA(u, v)? \) Euler tour/binary lifting
- \( dist(u, v)? \) Euler tour/binary lifting + depths
- sum of edge weights on \((u, v)\) path?
LCA/binary lifting applications

- $LCA(u, v)$? Euler tour/binary lifting
- $\text{dist}(u, v)$? Euler tour/binary lifting + depths
- sum of edge weights on $(u, v)$ path? Euler tour with two entry types (up + down)!
LCA/binary lifting applications

- $LCA(u, v)$? Euler tour/binary lifting
- $dist(u, v)$? Euler tour/binary lifting + depths
- sum of edge weights on $(u, v)$ path? Euler tour with two entry types (up + down)!
- max of edge weights on $(u, v)$ path?
LCA/binary lifting applications

- $LCA(u, v)$? Euler tour/binary lifting
- $dist(u, v)$? Euler tour/binary lifting + depths
- sum of edge weights on $(u, v)$ path?
  Euler tour with two entry types (up + down)!
- max of edge weights on $(u, v)$ path?
  Binary lifting with max information
LCA/binary lifting applications

- $LCA(u, v)$? Euler tour/binary lifting
- $dist(u, v)$? Euler tour/binary lifting + depths
- sum of edge weights on $(u, v)$ path?
  
  Euler tour with two entry types (up + down)!
- max of edge weights on $(u, v)$ path?
  
  Binary lifting with max information
- max of edge weights with updates?
LCA/binary lifting applications

- $LCA(u, v)$?
  Euler tour/binary lifting

- $dist(u, v)$?
  Euler tour/binary lifting + depths

- sum of edge weights on $(u, v)$ path?
  Euler tour with two entry types (up + down)!

- max of edge weights on $(u, v)$ path?
  Binary lifting with max information

- max of edge weights with updates?
  Binary lifting: too much to update
LCA/binary lifting applications

- \( LCA(u, v)? \)  Euler tour/binary lifting
- \( dist(u, v)? \)  Euler tour/binary lifting + depths
- sum of edge weights on \((u, v)\) path?  
  Euler tour with two entry types (up + down)!
- max of edge weights on \((u, v)\) path?  
  Binary lifting with max information
- max of edge weights with updates?  
  Binary lifting: too much to update  
  Heavy-light decomposition!
Heavy light decomposition
Heavy light decomposition

**Idea:** Partition tree into paths, such that we traverse $O(\log n)$ paths per $(u, v)$ query.
**Heavy light decomposition**

**Idea:** Partition tree into paths, such that we traverse $O(\log n)$ paths per $(u, v)$ query.

Where should the path continue?
Heavy light decomposition

**Idea:** Partition tree into paths, such that we traverse $O(\log n)$ paths per $(u, v)$ query.

Where should the path continue?

The largest child!
Heavy light decomposition

**Idea:** Partition tree into paths, such that we traverse $O(\log n)$ paths per $(u, v)$ query.

Where should the path continue?

The largest child!
Heavy light decomposition

**Idea:** Partition tree into paths, such that we traverse $O(\log n)$ paths per $(u, v)$ query.

Where should the path continue?

The largest child!

- **Light** edge
- **Heavy** edge
Heavy light decomposition

**Idea:** Partition tree into paths, such that we traverse $O(\log n)$ paths per $(u, v)$ query.

Where should the path continue?

The largest child!

How many paths on the way from the root to any node?
Heavy light decomposition

**Idea:** Partition tree into paths, such that we traverse $O(\log n)$ paths per $(u, v)$ query.

Where should the path continue?

- **The largest child!**

How many paths on the way from the root to any node?

- New path means light edge
Heavy light decomposition

Idea: Partition tree into paths, such that we traverse $O(\log n)$ paths per $(u, v)$ query.

Where should the path continue?

The largest child!

- New path means light edge
- Light edge means not largest child
Heavy light decomposition

**Idea:** Partition tree into paths, such that we traverse $O(\log n)$ paths per $(u, v)$ query.

Where should the path continue?

The largest child!

How many paths on the way from the root to any node?
- New path means light edge
- Light edge means not largest child
- Tree size is at least halved
Heavy light decomposition

Idea: Partition tree into paths, such that we traverse $O(\log n)$ paths per $(u, v)$ query.

Where should the path continue?

The largest child!

- Heavy edge
- Light

How many paths on the way from the root to any node?

- New path means light edge
- Light edge means not largest child
- Tree size is at least halved
- $O(\log n)$ paths
Heavy light decomposition

Every \((u, v)\) path consists of \(O(\log n)\) (parts of) paths.
Heavy light decomposition

Every \((u, v)\) path consists of \(\mathcal{O}(\log n)\) (parts of) paths.

How does this help for queries?
Heavy light decomposition

Every \((u, v)\) path consists of \(O(\log n)\) (parts of) paths.

How does this help for queries? One data structure per path!
Heavy light decomposition

Every \((u, v)\) path consists of \(O(\log n)\) (parts of) paths.

How does this help for queries? One data structure per path!

- Update single node/edge?
Heavy light decomposition

Every \((u, v)\) path consists of \(O(\log n)\) (parts of) paths.

How does this help for queries? One data structure per path!

- Update single node/edge?
  - Point update in 1 path
Heavy light decomposition

Every \((u, v)\) path consists of \(O(\log n)\) (parts of) paths.

How does this help for queries? One data structure per path!

- Update single node/edge?
  Point update in 1 path

- Update some \((u, v)\) path?
Heavy light decomposition

Every \((u, v)\) path consists of \(\mathcal{O}(\log n)\) (parts of) paths.

How does this help for queries? One data structure per path!

- Update single node/edge?
  - Point update in 1 path

- Update some \((u, v)\) path?
  - Range update in \(\mathcal{O}(\log n)\) paths
Heavy light decomposition

Every \((u, v)\) path consists of \(O(\log n)\) (parts of) paths.

How does this help for queries? One data structure per path!

- Update single node/edge?
  - Point update in 1 path

- Update some \((u, v)\) path?
  - Range update in \(O(\log n)\) paths

- Query some single node/edge?
Heavy light decomposition

Every \((u, v)\) path consists of \(O(\log n)\) (parts of) paths.

How does this help for queries? One data structure per path!

- Update single node/edge?
  Point update in 1 path

- Update some \((u, v)\) path?
  Range update in \(O(\log n)\) paths

- Query some single node/edge?
  Point query in 1 path
Heavy light decomposition

Every \((u, v)\) path consists of \(\mathcal{O}(\log n)\) (parts of) paths.

How does this help for queries? One data structure per path!

- Update single node/edge?
  - Point update in 1 path

- Update some \((u, v)\) path?
  - Range update in \(\mathcal{O}(\log n)\) paths

- Query some single node/edge?
  - Point query in 1 path

- Query some \((u, v)\) path aggregate?
Heavy light decomposition

Every \((u, v)\) path consists of \(\mathcal{O}(\log n)\) (parts of) paths.

How does this help for queries? One data structure per path!

- Update single node/edge?
  - Point update in 1 path

- Update some \((u, v)\) path?
  - Range update in \(\mathcal{O}(\log n)\) paths

- Query some single node/edge?
  - Point query in 1 path

- Query some \((u, v)\) path aggregate?
  - Range query+combine of \(\mathcal{O}(\log n)\) paths
Heavy light decomposition
Heavy light decomposition

- One DS per path?
Heavy light decomposition

One DS per path? vs One DS to rule them all!

Diagram showing a tree structure with different colored paths.
Heavy light decomposition

- One DS per path?
- One DS to rule them all!
- How to number the nodes?
Heavy light decomposition

- One DS per path?
- One DS to rule them all!
- How to number the nodes?
Heavy light decomposition

- One DS per path?  One DS to rule them all!
- How to number the nodes?
One DS per path? One DS to rule them all!
How to number the nodes?
Heavy light decomposition

- One DS per path? One DS to rule them all!
- How to number the nodes?
Heavy light decomposition

- One DS per path?
- One DS to rule them all!
- How to number the nodes?
Heavy light decomposition

- One DS per path?
- How to number the nodes?

One DS to rule them all!
Heavy light decomposition

- One DS per path?
- How to number the nodes?

One DS to rule them all!
Heavy light decomposition

- One DS per path?
- One DS to rule them all!
- How to number the nodes?
Heavy light decomposition

- One DS per path?
- One DS to rule them all!
- How to number the nodes?
Heavy light decomposition

One DS per path? One DS to rule them all!

How to number the nodes?
Heavy light decomposition

- One DS per path? One DS to rule them all!
- How to number the nodes?
  - DFS order, but visit heavy child first
Heavy light decomposition

- One DS per path?
- One DS to rule them all!
- How to number the nodes?
  - DFS order, but visit heavy child first

```
pos
```

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
```
Heavy light decomposition

- One DS per path?
- How to number the nodes?
  - DFS order, but visit heavy child first

One DS to rule them all!
Heavy light decomposition

One DS per path? One DS to rule them all!

How to number the nodes?
DFS order, but visit heavy child first

vector<int> parent, depth, heavy, head, pos;
int cur_pos = 0;

// TODO Run DFS first (parent, depth, heavy)
void decompose(int v, int h, Graph& adj) {
    head[v] = h, pos[v] = cur_pos++;
    if (heavy[v] != -1)
        decompose(heavy[v], h, adj);
    for (int c : adj[v])
        if (c != parent[v] && c != heavy[v])
            decompose(c, c, adj);
}
Heavy light decomposition

How can we answer \((u, v)\) queries?
Heavy light decomposition

How can we answer \((u, v)\) queries?
How can we answer \((u, v)\) queries?

```c
void query(int u, int v) {
    for (; head[u] != head[v]; v = parent[head[v]]) {
        if (depth[head[u]] > depth[head[v]])
            swap(u, v);
        ds_query(pos[head[v]], pos[v]); (inclusive)
    }
    if (depth[u] > depth[v])
        swap(u, v);
    ds_query(pos[u], pos[v]); (inclusive)
}
```
How can we answer \((u, v)\) queries?

```c
void query(int u, int v) {
    for (; head[u] != head[v]; v = parent[head[v]]) {
        if (depth[head[u]] > depth[head[v]])
            swap(u, v);
        ds_query(pos[head[v]], pos[v]);  // (inclusive)
    }
    if (depth[u] > depth[v])
        swap(u, v);
    ds_query(pos[u], pos[v]);  // (inclusive)
}
```

Heavy light decomposition
How can we answer \((u, v)\) queries?

```c
void query(int u, int v) {
    for (; head[u] != head[v]; v = parent[head[v]]) {
        if (depth[head[u]] > depth[head[v]])
            swap(u, v);
        ds_query(pos[head[v]], pos[v]);  // (inclusive)
    }
    if (depth[u] > depth[v])
        swap(u, v);
    ds_query(pos[u], pos[v]);  // (inclusive)
}
```

Heavy light decomposition
Heavy light decomposition

How can we answer \((u, v)\) queries?

```c
void query(int u, int v) {
    for (; head[u] != head[v]; v = parent[head[v]]) {
        if (depth[head[u]] > depth[head[v]])
            swap(u, v);
        ds_query(pos[head[v]], pos[v]); (inclusive)
    }
    if (depth[u] > depth[v])
        swap(u, v);
    ds_query(pos[u], pos[v]); (inclusive)
}
```

(inclusive)
How can we answer \((u, v)\) queries?

```c
void query(int u, int v) {
    for (; head[u] != head[v]; v = parent[head[v]]) {
        if (depth[head[u]] > depth[head[v]])
            swap(u, v);
        ds_query(pos[head[v]], pos[v]); (inclusive)
    }
    if (depth[u] > depth[v])
        swap(u, v);
    ds_query(pos[u], pos[v]); (inclusive)
}
```
How can we answer \((u, v)\) queries?

```c
void query(int u, int v) {
    for (; head[u] != head[v]; v = parent[head[v]]) {
        if (depth[head[u]] > depth[head[v]])
            swap(u, v);
        ds_query(pos[head[v]], pos[v]); // (inclusive)
    }
    if (depth[u] > depth[v])
        swap(u, v);
    ds_query(pos[u], pos[v]); // (inclusive)
}
```

Heavy light decomposition
Heavy light decomposition

How can we answer \((u,v)\) queries?

```c
void query(int u, int v) {
    for (; head[u] != head[v]; v = parent[head[v]]) {
        if (depth[head[u]] > depth[head[v]])
            swap(u, v);
        ds_query(pos[head[v]], pos[v]);  // (inclusive)
    }
    if (depth[u] > depth[v])
        swap(u, v);
    ds_query(pos[u], pos[v]);  // (inclusive)
}
```

Be careful with final edge!
Weights on edges vs. nodes
Heavy light decomposition

**Bonus question:** What about subtree updates/queries?
Heavy light decomposition

**Bonus question:** What about subtree updates/queries?

E.g., add 42 to all nodes in subtree of node 2
Heavy light decomposition

**Bonus question:** What about subtree updates/queries?

E.g., add 42 to all nodes in subtree of node 2

- Our numbering is basically a DFS
- A subtree is a single segment in our DS!
**Heavy light decomposition**

**Bonus question:** What about subtree updates/queries?

- E.g., add 42 to all nodes in subtree of node 2
- Our numbering is basically a DFS
- A subtree is a single segment in our DS!

```c
ds_update(pos[u], pos[u]+size[u]-1, x); (inclusive)
```

A subtree is a single segment in our DS! The subtree size can be calculated as:

```
subtree size
```

The diagram shows a tree structure with nodes numbered from 1 to 16, illustrating the concept of heavy light decomposition.
Heavy light decomposition

**Bonus question:** What about subtree updates/queries?

E.g., add 42 to all nodes in subtree of node 2

- Our numbering is basically a DFS
- A subtree is a single segment in our DS!

```c
ds_update(pos[u], pos[u]+size[u]-1, x); // (inclusive)
```

https://codeforces.com/blog/entry/53170
Centroid decomposition
Centroid decomposition

**Problem template:** Count number of paths in a tree with *insert random property here*.
Centroid decomposition

**Problem template:** Count number of paths in a tree with *insert random property here*.

- prime path length
Centroid decomposition

**Problem template:** Count number of paths in a tree with *insert random property here*.

- prime path length
- square number of red nodes
Centroid decomposition

**Problem template:** Count number of paths in a tree with *insert random property here*.

- prime path length
- square number of red nodes
- edge weight sum zero
Centroid decomposition

**Problem template:** Count number of paths in a tree with *insert random property here.*

- prime path length
- square number of red nodes
- edge weight sum zero

Sometimes solvable with tree DP, smaller-into-larger
Centroid decomposition

**Problem template:** Count number of paths in a tree with *insert random property here*.

- prime path length
- square number of red nodes
- edge weight sum zero

Sometimes solvable with tree DP, smaller-into-larger

Typical tree DP:
Centroid decomposition

**Problem template:** Count number of paths in a tree with *insert random property here*.

- prime path length
- square number of red nodes
- edge weight sum zero

Sometimes solvable with tree DP, smaller-into-larger

Typical tree DP:

- In every node, we take care of paths to/through it
Centroid decomposition

Problem template: Count number of paths in a tree with insert random property here.

- prime path length
- square number of red nodes
- edge weight sum zero

Sometimes solvable with tree DP, smaller-into-larger

Typical tree DP:

- In every node, we take care of paths to/through it
- Combine data from children
Centroid decomposition

**Problem template:** Count number of paths in a tree with *insert random property here*.

- prime path length
- square number of red nodes
- edge weight sum zero

Sometimes solvable with tree DP, smaller-into-larger

Typical tree DP:

- In every node, we take care of paths to/through it
- Combine data from children

Complicated and expensive!
Centroid decomposition

**Problem template:** Count number of paths in a tree with *insert random property here.*

- prime path length
- square number of red nodes
- edge weight sum zero

Sometimes solvable with tree DP, smaller-into-larger

Typical tree DP:

- In every node, we take care of paths to/through it
- Combine data from children

Complicated and expensive!

How would you count paths with edge weight sum zero?
Centroid decomposition

**Problem template:** Count number of paths in a tree with *insert random property here*.

- prime path length
- square number of red nodes
- edge weight sum zero

Sometimes solvable with tree DP, smaller-into-larger

Typical tree DP:

- In every node, we take care of paths to/through it
- Combine data from children

Complicated and expensive!

How would you count paths with edge weight sum zero?

- $O(size \text{ of subtree})$ is too expensive
Centroid decomposition

**Problem template:** Count number of paths in a tree with *insert random property here*.

- prime path length
- square number of red nodes
- edge weight sum zero

Sometimes solvable with tree DP, smaller-into-larger

Typical tree DP:

- In every node, we take care of paths to/through it
- Combine data from children

Complicated and expensive!

How would you count paths with edge weight sum zero?

- $O(\text{size of subtree})$ is too expensive
  
  ... but only if we have large subtrees!
Centroid decomposition

Let’s divide and conquer!
Centroid decomposition

Let’s divide and conquer!

- Split at some node
Centroid decomposition

Let’s divide and conquer!

- Split at some node
- process all paths that go over split node
Centroid decomposition

Let's divide and conquer!

- Split at some node
- process all paths that go over split node
- Independently+recursively solve for subtrees
Centroid decomposition

Let’s divide and conquer!

- Split at some node
- process all paths that go over split node
- Independently+recursively solve for subtrees

Why does this cover every path once?
Centroid decomposition

Let's divide and conquer!

- Split at some node
- process all paths that go over split node
- Independently + recursively solve for subtrees

Why does this cover every path once?

How to choose the split node?
Centroid decomposition

Let’s divide and conquer!

- Split at some node
- process all paths that go over split node
- Independently + recursively solve for subtrees

Why does this cover every path once?

How to choose the split node?

- Highest degree?
- Half-point of diameter path?
- Node with smallest subtrees?
Centroid decomposition

The **centroid** of a tree is the node whose removal partitions the tree into components of size at most $n/2$. 
Centroid decomposition

The **centroid** of a tree is the node whose removal partitions the tree into components of size at most \( n/2 \).

Why is there always such a node?
Centroid decomposition

The **centroid** of a tree is the node whose removal partitions the tree into components of size at most $n/2$.

Why is there always such a node? How can we find it?
Centroid decomposition

The **centroid** of a tree is the node whose removal partitions the tree into components of size at most $n/2$.

Why is there always such a node? How can we find it?

Per centroid in $\mathcal{O}(\text{size of current tree})$. Overall run time?
The **centroid** of a tree is the node whose removal partitions the tree into components of size at most $n/2$.

Why is there always such a node? How can we find it?

Per centroid in $O$(size of current tree). Overall run time? $O(n \log n)$!
The **centroid** of a tree is the node whose removal partitions the tree into components of size at most $n/2$.

Why is there always such a node? How can we find it?

Per centroid in $O(\text{size of current tree})$. Overall run time? $O(n \log n)$!

Count Zero-Sum paths over centroid:
Centroid decomposition

The centroid of a tree is the node whose removal partitions the tree into components of size at most $n/2$.

Why is there always such a node? How can we find it?

Per centroid in $O(\text{size of current tree})$. Overall run time? $O(n \log n)$!

Count Zero-Sum paths over centroid:
Centroid decomposition

The **centroid** of a tree is the node whose removal partitions the tree into components of size at most \( n/2 \).

Why is there always such a node? How can we find it?

Per centroid in \( \mathcal{O}(\text{size of current tree}) \). Overall run time? \( \mathcal{O}(n \log n)! \)

Count Zero-Sum paths over centroid:

Merged list: -3, -2, -1, 0, +2, +3, +4, +5
The **centroid** of a tree is the node whose removal partitions the tree into components of size at most $n/2$.

Why is there always such a node? How can we find it?

Per centroid in $\mathcal{O}(\text{size of current tree})$. Overall run time? $\mathcal{O}(n \log n)$!

Count Zero-Sum paths over centroid:

Merged list: -3, -2, -1, 0, +2, +3, +4, +5
Centroid decomposition

The **centroid** of a tree is the node whose removal partitions the tree into components of size at most \( n/2 \).

Why is there always such a node? How can we find it?

Per centroid in \( \mathcal{O}(\text{size of current tree}) \). Overall run time? \( \mathcal{O}(n \log n) \)!

Count Zero-Sum paths over centroid:

Merged list: -3, -2, -1, 0, +2, +3, +4, +5

-2, +2, +3

-3, 0, +4

-1, +5
Centroid decomposition

The **centroid** of a tree is the node whose removal partitions the tree into components of size at most \( n/2 \).

Why is there always such a node? How can we find it?

Per centroid in \( \mathcal{O}(\text{size of current tree}) \). Overall run time? \( \mathcal{O}(n \log n) \)!

Count Zero-Sum paths over centroid:

Merged list: -3, -2, -1, 0, +2, +3, +4, +5
The **centroid** of a tree is the node whose removal partitions the tree into components of size at most $n/2$.

Why is there always such a node? How can we find it?

Per centroid in $O(size$ of current tree$)$. Overall run time? $O(n \log n)$!

**Count Zero-Sum paths over centroid:**

Merged list: -3, -2, -1, 0, +2, +3, +4, +5
Centroid decomposition

The **centroid** of a tree is the node whose removal partitions the tree into components of size at most $n/2$.

Why is there always such a node? How can we find it?

Per centroid in $O$(size of current tree). Overall run time? $O(n \log n)!$

Count Zero-Sum paths over centroid:

Merged list: -3, -2, -1, 0, +2, +3, +4, +5

Generate stats from combined path stats
Centroid decomposition

The centroid of a tree is the node whose removal partitions the tree into components of size at most \( n/2 \).

Why is there always such a node?  How can we find it?

Per centroid in \( O(\text{size of current tree}) \). Overall run time?  \( O(n \log n) \)!

Count Zero-Sum paths over centroid:

Merged list: -3, -2, -1, 0, +2, +3, +4, +5

- Generate stats from combined path stats
- Subtract stats within subtrees
Centroid decomposition

The **centroid** of a tree is the node whose removal partitions the tree into components of size at most $n/2$.

Why is there always such a node? How can we find it?

Per centroid in $O(\text{size of current tree})$. Overall run time? $O(n \log n)$!

Count Zero-Sum paths over centroid:

Merged list: -3, -2, -1, 0, +2, +3, +4, +5

- Generate stats from combined path stats
- Subtract stats within subtrees

**Problem**: Given some $x$, count the number of paths with $x$ edges in the tree.
Centroid decomposition

The **centroid** of a tree is the node whose removal partitions the tree into components of size at most $n/2$.

Why is there always such a node? How can we find it?

Per centroid in $O(\text{size of current tree})$. Overall run time? $O(n \log n)$!

Count Zero-Sum paths over centroid:

- Merged list: -3, -2, -1, 0, +2, +3, +4, +5

- Generate stats from combined path stats
- Subtract stats within subtrees

**Problem**: Given some $x$, count the number of paths with $x$ edges in the tree.

Edge weights 1, target $x$ instead of 0
Centroid decomposition

- Calculate sizes via DFS
- Find centroid
- Calculate subtree stats through another DFS
- Combine stats
- Remove centroid from graph
- Recursively call for every neighbor
Centroid tree
Centroid tree

**Radius queries**: Given a vertex \( v \), calculate *some property* of all nodes in distance \( k \) to \( v \).
Centroid tree

**Radius queries:** Given a vertex $v$, calculate *some property* of all nodes in distance $k$ to $v$.

- Given node $v$, find sum of all node weights in radius $k$ around $v$. 
Centroid tree

**Radius queries:** Given a vertex $v$, calculate *some property* of all nodes in distance $k$ to $v$.

- Given node $v$, find sum of all node weights in radius $k$ around $v$.
- Given node $v$, find the closest node with value above $x$. 
Centroid tree

**Radius queries:** Given a vertex \( v \), calculate *some property* of all nodes in distance \( k \) to \( v \).

- Given node \( v \), find sum of all node weights in radius \( k \) around \( v \).
- Given node \( v \), find the closest node with value above \( x \).

**Problem:** Given a tree with red and gray nodes, execute two types of queries.
1. Paint a node \( v \) red.
2. Given a node \( v \), calculate the distance to the closest red node.
Centroid tree

**Radius queries:** Given a vertex $v$, calculate *some property* of all nodes in distance $k$ to $v$.

- Given node $v$, find sum of all node weights in radius $k$ around $v$.
- Given node $v$, find the closest node with value above $x$.

**Problem:** Given a tree with red and gray nodes, execute two types of queries.
1. Paint a node $v$ red.
2. Given a node $v$, calculate the distance to the closest red node.
**Centroid tree**

**Radius queries:** Given a vertex \( v \), calculate *some property* of all nodes in distance \( k \) to \( v \).

- Given node \( v \), find sum of all node weights in radius \( k \) around \( v \).
- Given node \( v \), find the closest node with value above \( x \).

**Problem:** Given a tree with red and gray nodes, execute two types of queries.
1. Paint a node \( v \) red.
2. Given a node \( v \), calculate the distance to the closest red node.
Centroid tree

**Radius queries**: Given a vertex $v$, calculate *some property* of all nodes in distance $k$ to $v$.

- Given node $v$, find sum of all node weights in radius $k$ around $v$.
- Given node $v$, find the closest node with value above $x$.

**Problem**: Given a tree with red and gray nodes, execute two types of queries.
1. Paint a node $v$ red.
2. Given a node $v$, calculate the distance to the closest red node.
Centroid tree

**Radius queries**: Given a vertex $v$, calculate *some property* of all nodes in distance $k$ to $v$.

- Given node $v$, find sum of all node weights in radius $k$ around $v$.
- Given node $v$, find the closest node with value above $x$.

**Problem**: Given a tree with red and gray nodes, execute two types of queries.
1. Paint a node $v$ red.
2. Given a node $v$, calculate the distance to the closest red node.
Centroid tree

**Radius queries:** Given a vertex $v$, calculate *some property* of all nodes in distance $k$ to $v$.
- Given node $v$, find sum of all node weights in radius $k$ around $v$.
- Given node $v$, find the closest node with value above $x$.

**Problem:** Given a tree with red and gray nodes, execute two types of queries.
1. Paint a node $v$ red.
2. Given a node $v$, calculate the distance to the closest red node.
Centroid tree

**Radius queries:** Given a vertex \( v \), calculate some property of all nodes in distance \( k \) to \( v \).

- Given node \( v \), find sum of all node weights in radius \( k \) around \( v \).
- Given node \( v \), find the closest node with value above \( x \).

**Problem:** Given a tree with red and gray nodes, execute two types of queries.
1. Paint a node \( v \) red.
2. Given a node \( v \), calculate the distance to the closest red node.
Centroid tree

**Radius queries:** Given a vertex $v$, calculate *some property* of all nodes in distance $k$ to $v$.

- Given node $v$, find sum of all node weights in radius $k$ around $v$.
- Given node $v$, find the closest node with value above $x$.

**Problem:** Given a tree with red and gray nodes, execute two types of queries.
1. Paint a node $v$ red.
2. Given a node $v$, calculate the distance to the closest red node.
Centroid tree

**Radius queries**: Given a vertex \( v \), calculate *some property* of all nodes in distance \( k \) to \( v \).

- Given node \( v \), find sum of all node weights in radius \( k \) around \( v \).
- Given node \( v \), find the closest node with value above \( x \).

**Problem**: Given a tree with red and gray nodes, execute two types of queries.
1. Paint a node \( v \) red.
2. Given a node \( v \), calculate the distance to the closest red node.

Tree (T)  
Centroid tree (CT)

- A node \( v \) belongs to the components of all its CT ancestors.
Centroid tree

**Radius queries:** Given a vertex $v$, calculate *some property* of all nodes in distance $k$ to $v$.

- Given node $v$, find sum of all node weights in radius $k$ around $v$.
- Given node $v$, find the closest node with value above $x$.

**Problem:** Given a tree with red and gray nodes, execute two types of queries.
1. Paint a node $v$ red.
2. Given a node $v$, calculate the distance to the closest red node.
Centroid tree

**Radius queries:** Given a vertex $v$, calculate some property of all nodes in distance $k$ to $v$.
- Given node $v$, find sum of all node weights in radius $k$ around $v$.
- Given node $v$, find the closest node with value above $x$.

**Problem:** Given a tree with red and gray nodes, execute two types of queries.
1. Paint a node $v$ red.
2. Given a node $v$, calculate the distance to the closest red node.

Tree (T)  
```
15 -- 14 -- 11 -- 9
    |  |  |   |
   13 6 7 8
```

Centroid tree (CT)  
```
3
|
/|
/ | 
11 1 7
 |
/ \
/  
12 10
```

- A node $v$ belongs to the components of all its CT ancestors.
- Any $(u, v)$ path goes through $LCA_{CT}(u, v)$.
- A node $v$ has $O(\log n)$ CT ancestors.
**Centroid tree**

**Radius queries:** Given a vertex \( v \), calculate *some property* of all nodes in distance \( k \) to \( v \).
- Given node \( v \), find sum of all node weights in radius \( k \) around \( v \).
- Given node \( v \), find the closest node with value above \( x \).

**Problem:** Given a tree with red and gray nodes, execute two types of queries.
1. Paint a node \( v \) red.
2. Given a node \( v \), calculate the distance to the closest red node.

---

**Tree (T)**

![Tree diagram]

**Centroid tree (CT)**

- A node \( v \) belongs to the components of all its CT ancestors.
- Any \((u, v)\) path goes through \( LCA_{CT}(u, v) \).
- A node \( v \) has \( O(\log n) \) CT ancestors.

How does this help for queries?
Centroid tree

**Problem:** Given a tree with red and gray nodes, execute two types of queries.
1. Paint a node $v$ red.
2. Given a node $v$, calculate the distance to the closest red node.
Centroid tree

**Problem:** Given a tree with red and gray nodes, execute two types of queries. 
1. Paint a node $v$ red.
2. Given a node $v$, calculate the distance to the closest red node.

Tree (T)  

Centroid tree (CT)

- **Distance in T to closest red node in CT subtree**
Problem: Given a tree with red and gray nodes, execute two types of queries.
1. Paint a node \( v \) red.
2. Given a node \( v \), calculate the distance to the closest red node.

Tree (T)  
Centroid tree (CT)

- Distance in T to closest red node in CT subtree
- Query type 1: Update all CT ancestors
Problem: Given a tree with red and gray nodes, execute two types of queries.
1. Paint a node \( v \) red.
2. Given a node \( v \), calculate the distance to the closest red node.

Tree (T) and Centroid tree (CT)

- Distance in T to closest red node in CT subtree
- Query type 1: Update all CT ancestors
**Problem:** Given a tree with red and gray nodes, execute two types of queries.
1. Paint a node $v$ red.
2. Given a node $v$, calculate the distance to the closest red node.

- Distance in $T$ to closest red node in CT subtree
- Query type 1: Update all CT ancestors
Problem: Given a tree with red and gray nodes, execute two types of queries.
1. Paint a node $v$ red.
2. Given a node $v$, calculate the distance to the closest red node.

Tree (T) | Centroid tree (CT)
---|---
15—14—11—9 | 23—3
13—6 | 1
5 | 17
8
4
7—12—10
3

- Distance in T to closest red node in CT subtree
- Query type 1: Update all CT ancestors
**Problem:** Given a tree with red and gray nodes, execute two types of queries.
1. Paint a node $v$ red.
2. Given a node $v$, calculate the distance to the closest red node.
**Problem:** Given a tree with red and gray nodes, execute two types of queries.
1. Paint a node $v$ red.
2. Given a node $v$, calculate the distance to the closest red node.
Centroid tree

**Problem:** Given a tree with red and gray nodes, execute two types of queries.
1. Paint a node $v$ red.
2. Given a node $v$, calculate the distance to the closest red node.
**Problem:** Given a tree with red and gray nodes, execute two types of queries.
1. Paint a node $v$ red.
2. Given a node $v$, calculate the distance to the closest red node.
**Problem:** Given a tree with red and gray nodes, execute two types of queries.
1. Paint a node \( v \) red.
2. Given a node \( v \), calculate the distance to the closest red node.
**Problem:** Given a tree with red and gray nodes, execute two types of queries.
1. Paint a node $v$ red.
2. Given a node $v$, calculate the distance to the closest red node.

**Tree (T)**

**Centroid tree (CT)**

- Distance in $T$ to closest red node in CT subtree
- Query type 1: Update all CT ancestors
- Query type 2: Query all CT ancestors

Run time?
Centroid tree

**Problem:** Given a tree with red and gray nodes, execute two types of queries.
1. Paint a node $v$ red.
2. Given a node $v$, calculate the distance to the closest red node.

- Distance in $T$ to closest red node in $CT$ subtree
- Query type 1: Update all $CT$ ancestors
- Query type 2: Query all $CT$ ancestors

Run time? $O(\log^2 n)$
**Problem:** Given a tree with red and gray nodes, execute two types of queries.
1. Paint a node \( v \) red.
2. Given a node \( v \), calculate the distance to the closest red node.

Tree (T)  
Centroid tree (CT)  

- Distance in T to closest red node in CT subtree
- Query type 1: Update all CT ancestors
- Query type 2: Query all CT ancestors

Run time? \( \mathcal{O}(\log^2 n) \)

Beware of “false LCAs” (non-simple paths)!

- e.g., path from 12 via 3 to 8 with length 4+2 is not simple
Inclusion-Exclusion over CT-tree

- Given node $v$, find sum of all node weights in radius $k$ around $v$.
- Given node $v$, find the closest node with value above $x$. 
Inclusion-Exclusion over CT-tree

- Given node $v$, find sum of all node weights in radius $k$ around $v$.
- Given node $v$, find the closest node with value above $x$.  

**CT tree**
Inclusion-Exclusion over CT-tree

- Given node $v$, find sum of all node weights in radius $k$ around $v$.
- Given node $v$, find the closest node with value above $x$. 

CT tree

- for a query on node $v$:
Inclusion-Exclusion over CT-tree

- Given node $v$, find sum of all node weights in radius $k$ around $v$.
- Given node $v$, find the closest node with value above $x$.

CT tree

- for a query on node $v$:
  - iterate over CT ancestors $c$

Given node $v$, find sum of all node weights in radius $k$ around $v$.
Given node $v$, find the closest node with value above $x$. 
Inclusion-Exclusion over CT-tree

- Given node $v$, find sum of all node weights in radius $k$ around $v$.
- Given node $v$, find the closest node with value above $x$.

**CT tree**

- for a query on node $v$:
  - iterate over CT ancestors $c$
  - handle paths from $v$ over $c$ to the part of $c$'s subtree that is behind $c$
Inclusion-Exclusion over CT-tree

- Given node $v$, find sum of all node weights in radius $k$ around $v$.
- Given node $v$, find the closest node with value above $x$.

For a query on node $v$:
- Iterate over CT ancestors $c$
- Handle paths from $v$ over $c$ to the part of $c$’s subtree that is behind $c$

CT tree

Given node $v$, find sum of all node weights in radius $k$ around $v$.

Given node $v$, find the closest node with value above $x$.
Inclusion-Exclusion over CT-tree

- Given node $v$, find sum of all node weights in radius $k$ around $v$.
- Given node $v$, find the closest node with value above $x$.

For a query on node $v$:
- Iterate over CT ancestors $c$.
- Handle paths from $v$ over $c$ to the part of $c$’s subtree that is behind $c$.
Inclusion-Exclusion over CT-tree

- Given node $v$, find sum of all node weights in radius $k$ around $v$.
- Given node $v$, find the closest node with value above $x$.

CT tree

- For a query on node $v$:
  - Iterate over CT ancestors $c$
  - Handle paths from $v$ over $c$ to the part of $c$'s subtree that is behind $c$

  Example: $c_1$ handles paths to yellow and red

+ subtree of $c_1$
Inclusion-Exclusion over CT-tree

- Given node $v$, find sum of all node weights in radius $k$ around $v$.
- Given node $v$, find the closest node with value above $x$.

CT tree

- for a query on node $v$:
- iterate over CT ancestors $c$
- handle paths from $v$ over $c$ to the part of $c$’s subtree that is behind $c$

Given node $v$, find sum of all node weights in radius $k$ around $v$.

Given node $v$, find the closest node with value above $x$.

- $c_1$ handles paths to $v$ in the subtree of $c_1$
- $c_2$ handles paths to $v$ in the subtree of $c_2$
Inclusion-Exclusion over CT-tree

- Given node \( v \), find sum of all node weights in radius \( k \) around \( v \).
- Given node \( v \), find the closest node with value above \( x \).

CT tree

- for a query on node \( v \):
  - iterate over CT ancestors \( c \)
  - handle paths from \( v \) over \( c \) to the part of \( c \)’s subtree that is behind \( c \)

\[ \text{e.g., } c_1 \text{ handles paths to } \]

+ [subtree of \( c_1 \)]
+ [subtree of \( c_2 \)]
Inclusion-Exclusion over CT-tree

- Given node \( v \), find sum of all node weights in radius \( k \) around \( v \).
- Given node \( v \), find the closest node with value above \( x \).

For a query on node \( v \):
- Iterate over CT ancestors \( c \)
- Handle paths from \( v \) over \( c \) to the part of \( c \)’s subtree that is behind \( c \)

E.g., \( c_1 \) handles paths to the green area.
Inclusion-Exclusion over CT-tree

- Given node $v$, find sum of all node weights in radius $k$ around $v$.
- Given node $v$, find the closest node with value above $x$.

Given node $v$, find the closest node with value above $x$.

- for a query on node $v$:
  - iterate over CT ancestors $c$
  - handle paths from $v$ over $c$ to the part of $c$’s subtree that is behind $c$
    
    e.g., $c_1$ handles paths to  
    
    to cover every path exactly once, we aggregate
    
    $+ - + - + + + + - + - - + + = + +$
Inclusion-Exclusion over CT-tree

- Given node \( v \), find sum of all node weights in radius \( k \) around \( v \).
- Given node \( v \), find the closest node with value above \( x \).

**CT tree**

- for a query on node \( v \):
  - iterate over CT ancestors \( c \)
  - handle paths from \( v \) over \( c \) to the part of \( c \)’s subtree that is behind \( c \)
    - e.g., \( c_1 \) handles paths to
    - to cover every path exactly once, we aggregate
      \[
      + \quad - \quad + \quad - \quad + \quad + \quad - \quad + \quad = \quad -
      \]
    - so each centroid needs to keep aggregates of it’s tree and each subtree

\[
\begin{align*}
+ &\quad - \quad\text{subtree of } c_1 \\
+ &\quad - \quad\text{subtree of } c_2 \\
+ &\quad\text{subtree of } v
\end{align*}
\]
Inclusion-Exclusion over CT-tree

- Given node $v$, find sum of all node weights in radius $k$ around $v$.
- Given node $v$, find the closest node with value above $x$.

For a query on node $v$:
- Iterate over CT ancestors $c$.
- Handle paths from $v$ over $c$ to the part of $c$’s subtree that is behind $c$.
  
  e.g., $c_1$ handles paths to $\text{subtree of } c_1$
  $c_2$ handles paths to $\text{subtree of } c_2$
  $v$ handles paths to $\text{subtree of } v$

To cover every path exactly once, we aggregate:

\[
\begin{align*}
\text{+} & \quad \text{−} & \quad \text{−} & \quad \text{+} & \quad \text{−} & \quad \text{+} & \quad \text{+} & \quad \text{=} & \quad \text{=} \\
\end{align*}
\]

So each centroid needs to keep aggregates of it’s tree and each subtree.

\[\rightarrow O(\text{size}[c])\]