# Longest common extensions in trees

Philip Bille[1] Paweł Gawrychowski[2] Inge Li Gørtz[1]
Gad M. Landau[3,4] Oren Weimann[3]

[1]DTU Compute

[2]University of Warsaw (supported by WCMCS)

[3]University of Haifa

[4]NYU Polytechnic

July 1, 2015

# Longest common extensions in strings

Preprocess a given string $s[1..n]$ for computing the longest common prefix of $s[i..n]$ and $s[j..n]$.

`bbababbaaaaababababaaababababababababbbabababab`

$\mathrm{LCE}(i, j) = 5$

# Longest common extensions in strings

Preprocess a given string $s[1..n]$ for computing the longest common prefix of $s[i..n]$ and $s[j..n]$.

`bbababbaaaaababababaaabababababababbbabababab`

$\mathrm{LCE}(i, j) = 5$

# Longest common extensions in strings

Preprocess a given string $s[1..n]$ for computing the longest common prefix of $s[i..n]$ and $s[j..n]$.

bbababbaaaaabababaaababababababababbbabababab

$i$ $j$

$\mathrm{LCE}(i, j) = 5$

# Longest common extensions in strings

Preprocess a given string $s[1..n]$ for computing the longest common prefix of $s[i..n]$ and $s[j..n]$.

`bbababbaaaaababababaaabababababababbbabababab`

$i$ $j$

$\mathrm{LCE}(i, j) = 5$

# Longest common extensions in strings

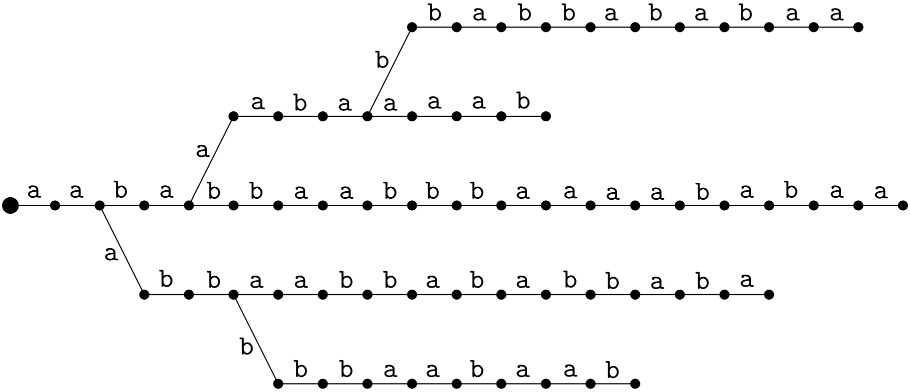
Preprocess a given string $s[1..n]$ for computing the longest common prefix of $s[i..n]$ and $s[j..n]$.

`bbababbaaaaababababaaababababababbbabababab`

$i$ $j$

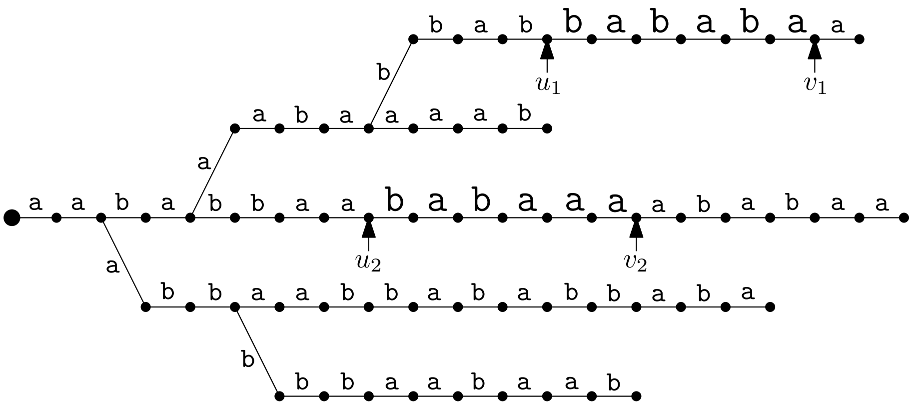
$\text{LCE}(i, j) = 5$

# What if there are multiple strings?



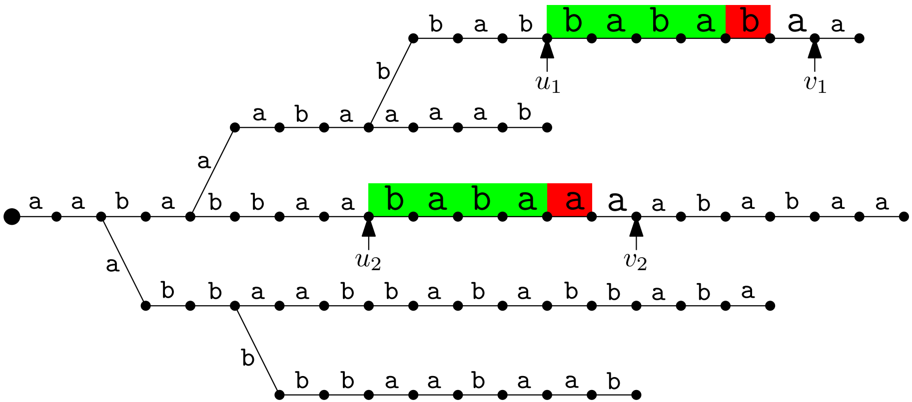

$$\mathrm{LCE}_{PP}(u_1, v_1, u_2, v_2) = 4$$

# What if there are multiple strings?



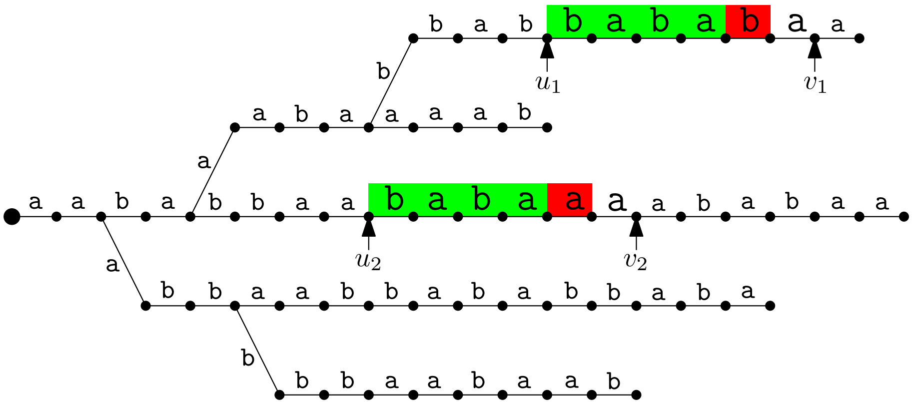

$\mathrm{LCE}_{PP}(u_1, v_1, u_2, v_2) = 4$

# What if there are multiple strings?




$\mathrm{LCE}_{PP}(u_1, v_1, u_2, v_2) = 4$

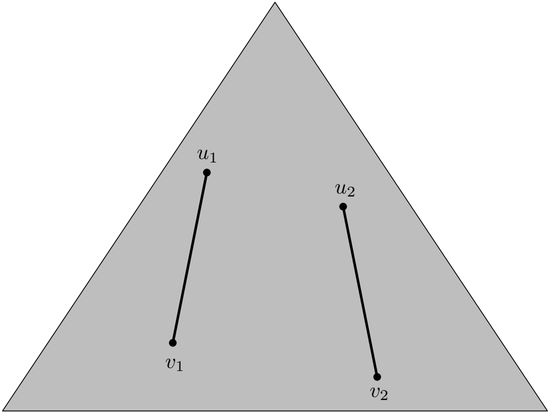
# What if there are multiple strings?



$$\mathrm{LCE}_{PP}(u_1, v_1, u_2, v_2) = 4$$

# Path-path queries

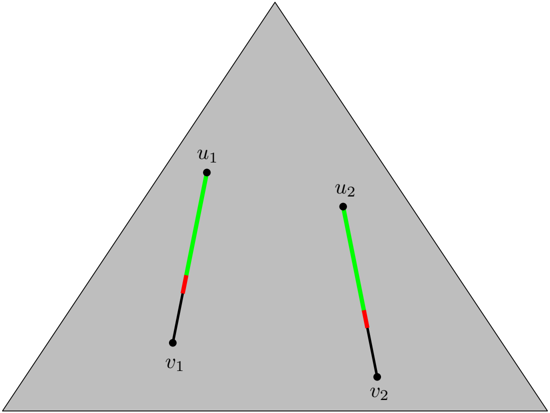
Given nodes $u_1, v_1, u_2, v_2$ such that $u_1$ is an ancestor of $v_1$ and $u_2$ is an ancestor of $v_2$, report the longest matching prefix of paths $u_1 \rightsquigarrow v_1$ and $u_2 \rightsquigarrow v_2$.




Without losing the generality, the paths are of the same length.

# Path-path queries

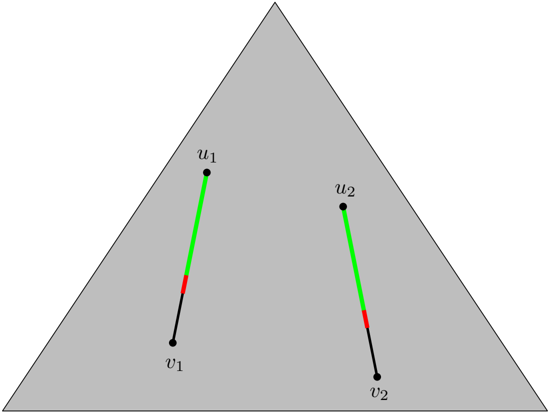
Given nodes $u_1, v_1, u_2, v_2$ such that $u_1$ is an ancestor of $v_1$ and $u_2$ is an ancestor of $v_2$, report the longest matching prefix of paths $u_1 \rightsquigarrow v_1$ and $u_2 \rightsquigarrow v_2$.




Without losing the generality, the paths are of the same length.

# Path-path queries

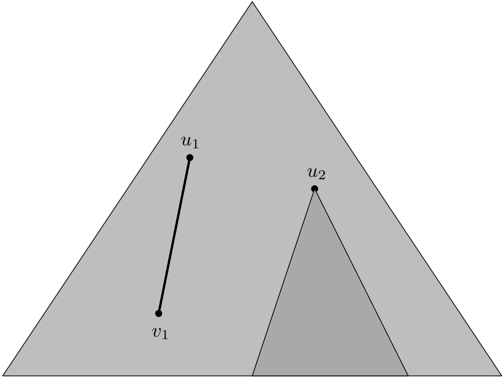
Given nodes $u_1, v_1, u_2, v_2$ such that $u_1$ is an ancestor of $v_1$ and $u_2$ is an ancestor of $v_2$, report the longest matching prefix of paths $u_1 \rightsquigarrow v_1$ and $u_2 \rightsquigarrow v_2$.




Without losing the generality, the paths are of the same length.

# Path-tree queries

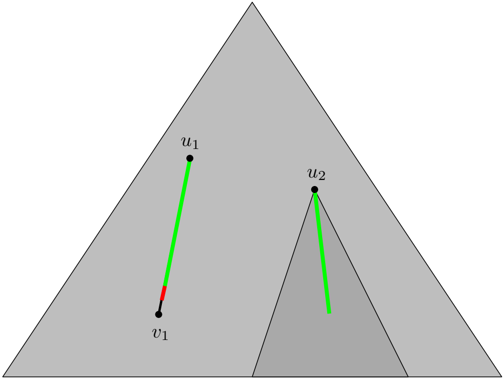
Given nodes $u_1, v_1, u_2$ such that $u_1$ is an ancestor of $v_1$, report the longest matching prefix of paths $u_1 \rightsquigarrow v_1$ and $u_2 \rightsquigarrow v_2$, where $v_2$ is a descendant of $u_2$.




Without losing the generality, edges outgoing from the same node have distinct labels.

# Path-tree queries

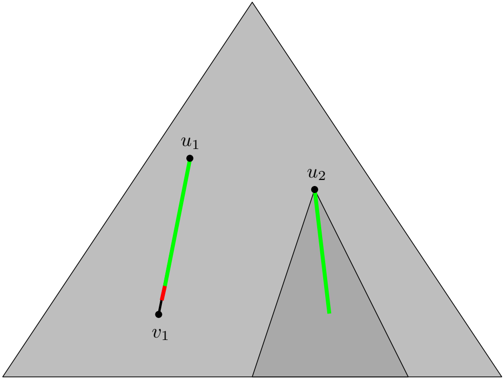
Given nodes $u_1, v_1, u_2$ such that $u_1$ is an ancestor of $v_1$, report the longest matching prefix of paths $u_1 \rightsquigarrow v_1$ and $u_2 \rightsquigarrow v_2$, where $v_2$ is a descendant of $u_2$.




Without losing the generality, edges outgoing from the same node have distinct labels.

# Path-tree queries

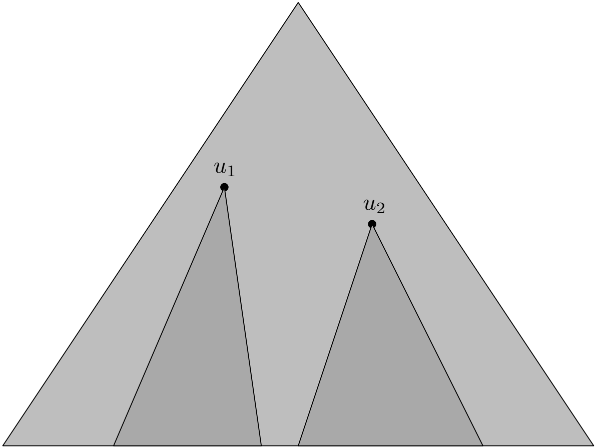
Given nodes $u_1, v_1, u_2$ such that $u_1$ is an ancestor of $v_1$, report the longest matching prefix of paths $u_1 \rightsquigarrow v_1$ and $u_2 \rightsquigarrow v_2$, where $v_2$ is a descendant of $u_2$.




Without losing the generality, edges outgoing from the same node have distinct labels.

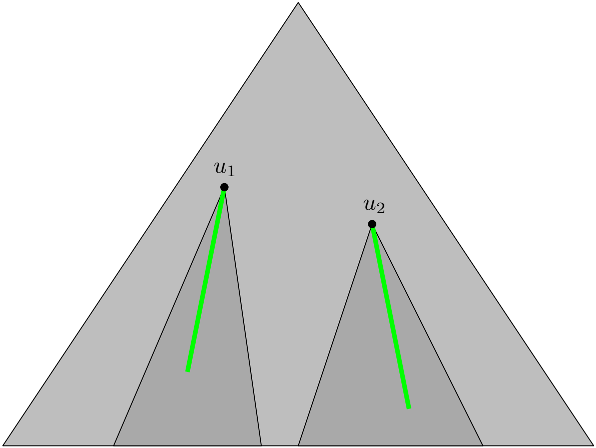
# Tree-tree queries

Given nodes $u_1, u_2$, report the longest matching prefix of paths $u_1 \leadsto v_1$ and $u_2 \leadsto v_2$, where $v_1$ is a descendant of $u_1$ and $v_2$ is a descendant of $u_2$.



# Tree-tree queries

Given nodes $u_1, u_2$, report the longest matching prefix of paths $u_1 \rightsquigarrow v_1$ and $u_2 \rightsquigarrow v_2$, where $v_1$ is a descendant of $u_1$ and $v_2$ is a descendant of $u_2$.



# Results

| problem | query | space | lowerbound |
|---|---|---|---|
| path-path | $\mathcal{O}(\log^* n)$ | $\mathcal{O}(n)$ | |
| path-tree | $\mathcal{O}((\log \log n)^2)$ | $\mathcal{O}(n)$ | predecessor hard |
| tree-tree | $\mathcal{O}(n/\tau)$ | $\mathcal{O}(n \cdot \tau)$ | set-intersection hard |

# Simple solution for path-path queries

We start with a simple $\mathcal{O}(1)$ query $\mathcal{O}(n \log n)$ space solution. For every $k = 0, 1, \ldots, \log n$, we build a separate structure of size $\mathcal{O}(n)$ allowing us to answer queries for paths of length $2^k$.

## Structure for paths of length $2^k$

There are only $n$ such paths. We sort them (lexicographically) and store the longest common prefix between any two paths adjacent in the sorted order. Longest common prefix of any two paths can be computing with a range minimum query on the stored numbers.

# Simple solution for path-path queries

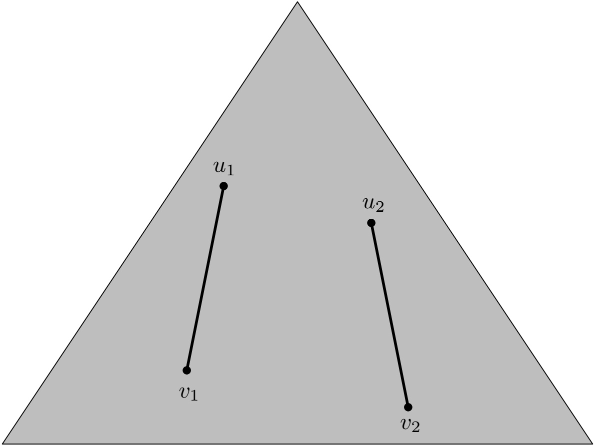
We start with a simple $\mathcal{O}(1)$ query $\mathcal{O}(n \log n)$ space solution. For every $k = 0, 1, \ldots, \log n$, we build a separate structure of size $\mathcal{O}(n)$ allowing us to answer queries for paths of length $2^k$.

### Structure for paths of length $2^k$

There are only $n$ such paths. We sort them (lexicographically) and store the longest common prefix between any two paths adjacent in the sorted order. Longest common prefix of any two paths can be computing with a range minimum query on the stored numbers.

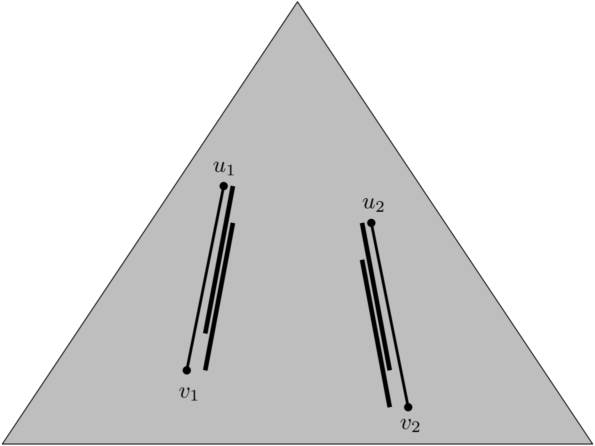
# Simple solution for path-path queries

Now, given an arbitrary query where the paths are of (the same) length $\ell$, we can reduce it to two queries where the paths are of length $2^k$, where $k = \lfloor \log \ell \rfloor$.



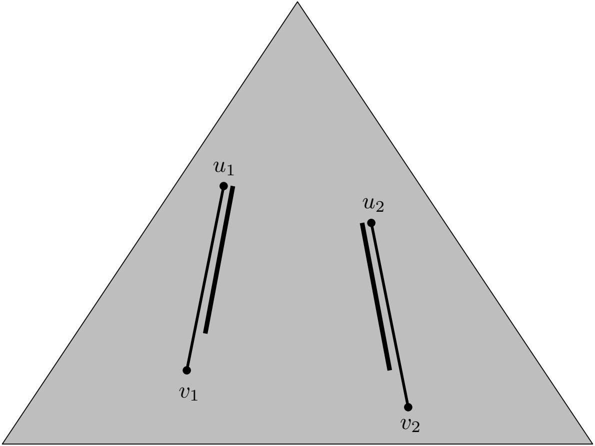
# Simple solution for path-path queries

Now, given an arbitrary query where the paths are of (the same) length $\ell$, we can reduce it to two queries where the paths are of length $2^k$, where $k = \lfloor \log \ell \rfloor$.



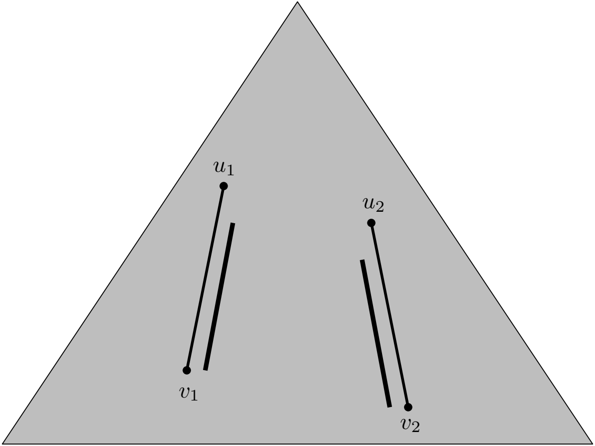
# Simple solution for path-path queries

Now, given an arbitrary query where the paths are of (the same) length $\ell$, we can reduce it to two queries where the paths are of length $2^k$, where $k = \lfloor \log \ell \rfloor$.



# Simple solution for path-path queries

Now, given an arbitrary query where the paths are of (the same) length $\ell$, we can reduce it to two queries where the paths are of length $2^k$, where $k = \lfloor \log \ell \rfloor$.



# Reducing the space

The natural approach is to store just some of the paths of length $2^k$, for every $k$. To choose which paths to store, we introduce the notion of **difference covers for trees**.

## Difference covers for trees

For any tree on $n$ nodes and a parameter $x$, it is possible to mark $\frac{2n}{x}$ nodes, so that for any $u, v$ at depths $\geq x^2$, there exists $\Delta \leq x^2$ such that the $\Delta$-th ancestors of both $u$ and $v$ are marked.

# Reducing the space

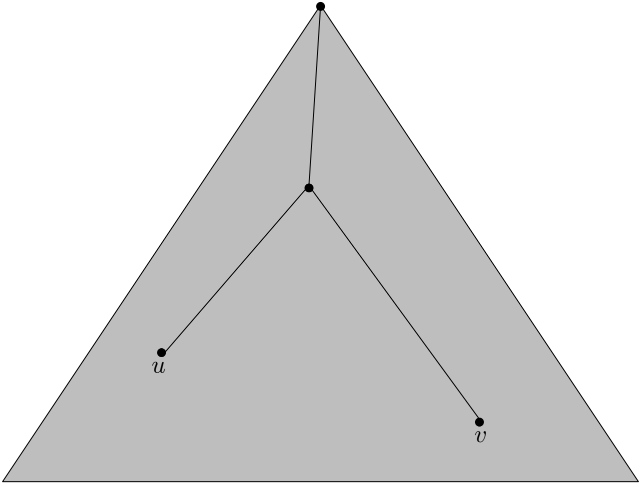
The natural approach is to store just some of the paths of length $2^k$, for every $k$. To choose which paths to store, we introduce the notion of **difference covers for trees**.

### Difference covers for trees

For any tree on $n$ nodes and a parameter $x$, it is possible to mark $\frac{2n}{x}$ nodes, so that for any $u, v$ at depths $\geq x^2$, there exists $\Delta \leq x^2$ such that the $\Delta$-th ancestors of both $u$ and $v$ are marked.

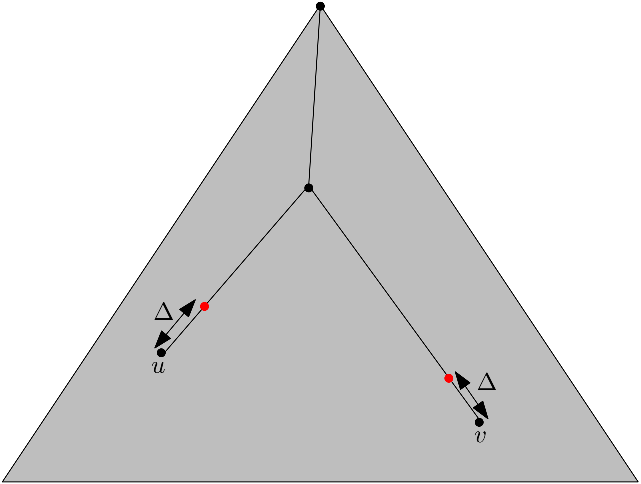
# Difference covers for trees

$\frac{2n}{x}$ marked nodes, $\Delta \leq x^2$



# Difference covers for trees

$\frac{2n}{x}$ marked nodes, $\Delta \leq x^2$



# Reducing the space

Find a difference cover with $x = \log n$. Then, for every $k = 0, 1, \ldots, \log n$ preprocess all paths of length $2^k$ ending at marked nodes. Additionally, preprocess all paths of length $\log^2 n$.

Any query can be reduced in $\mathcal{O}(1)$ time to computing the longest common prefix of two paths of length $\leq \log^2 n$.

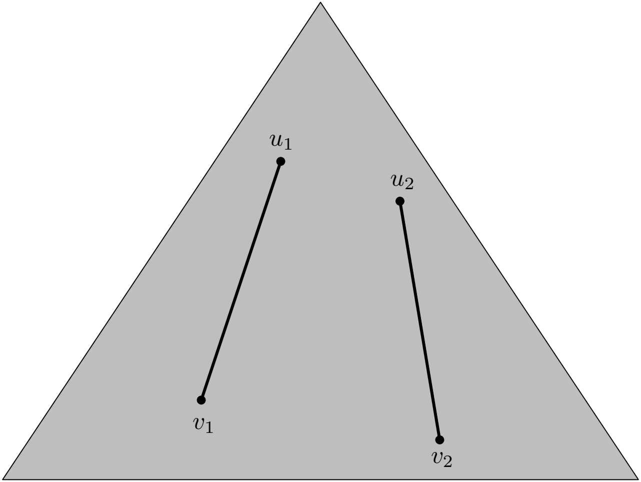
# Reducing the space

Find a difference cover with $x = \log n$. Then, for every $k = 0, 1, \ldots, \log n$ preprocess all paths of length $2^k$ ending at marked nodes. Additionally, preprocess all paths of length $\log^2 n$.

Any query can be reduced in $\mathcal{O}(1)$ time to computing the longest common prefix of two paths of length $\leq \log^2 n$.

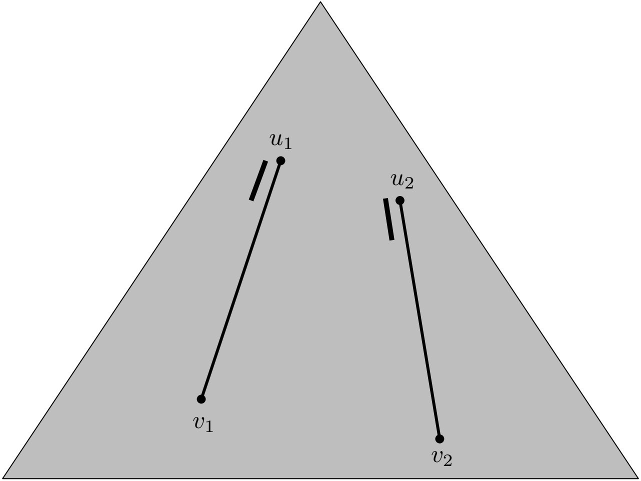
# Reducing the space

Compare prefixes of length $\log^2 n$.



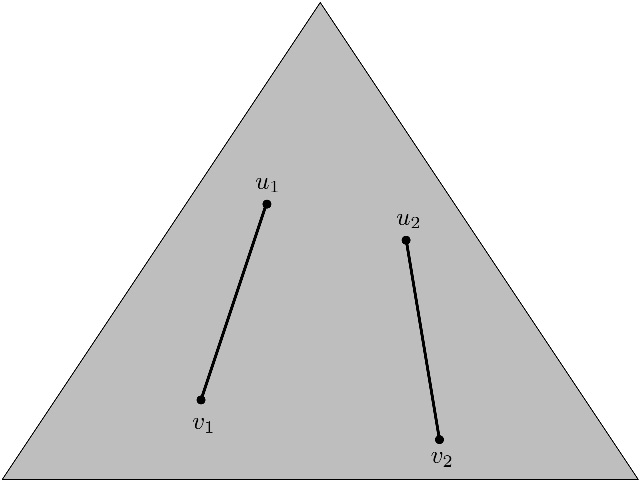
# Reducing the space

Compare prefixes of length $\log^2 n$.



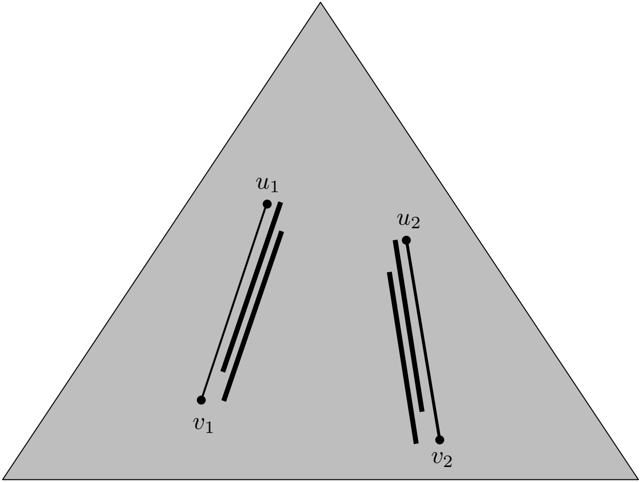
# Reducing the space

Cover the remaining part with paths of length $2^k$.



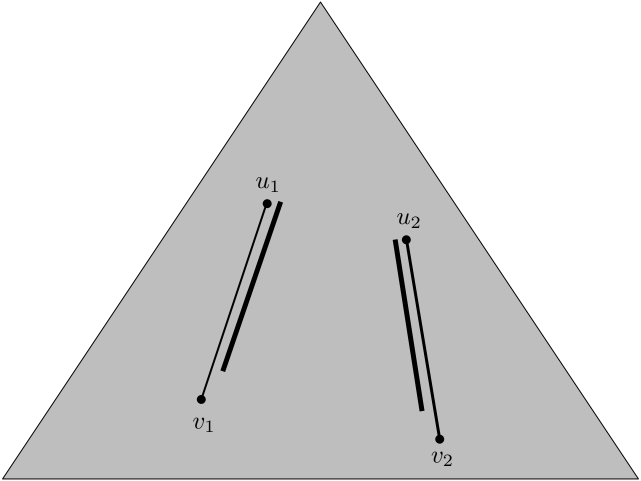
# Reducing the space

Cover the remaining part with paths of length $2^k$.



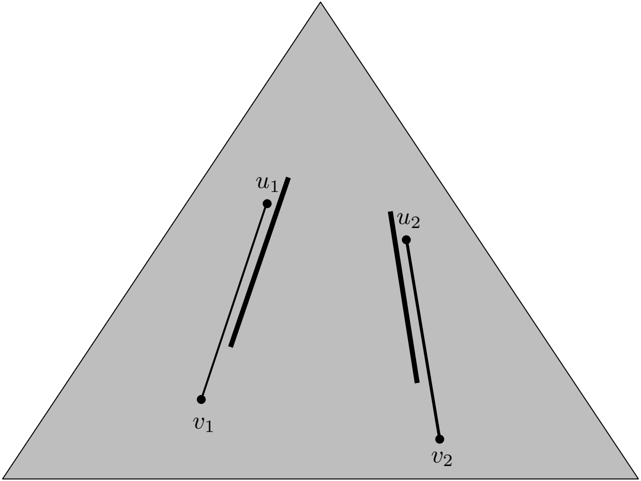
# Reducing the space

Slide the first pair of paths up so that they end at marked nodes.



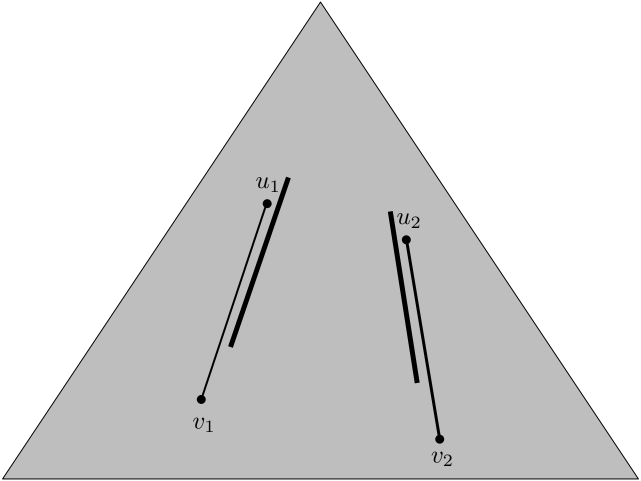
# Reducing the space

Slide the first pair of paths up so that they end at marked nodes.



# Reducing the space

Process the second pair of paths similarly.



# Reducing the space

After $\mathcal{O}(n)$ preprocessing we can reduce any query to computing the longest common prefix of two paths of length $\leq \log^2 n$. Now the reasoning can be iterated, so in the next step we get two paths of length $\leq \log^2(\log^2 n)$, and so on.

The number of iterations is at most $\mathcal{O}(\log^* n)$, so after $\mathcal{O}(n \log^* n)$ preprocessing we can answer any query in $\mathcal{O}(\log^* n)$ time.

Space can be decreased to $\mathcal{O}(n)$ by adding $\mathcal{O}(\log^* n)$ to the query (which is absorbed anyway).

# Reducing the space

After $\mathcal{O}(n)$ preprocessing we can reduce any query to computing the longest common prefix of two paths of length $\leq \log^2 n$. Now the reasoning can be iterated, so in the next step we get two paths of length $\leq \log^2(\log^2 n)$, and so on.

The number of iterations is at most $\mathcal{O}(\log^* n)$, so after $\mathcal{O}(n \log^* n)$ preprocessing we can answer any query in $\mathcal{O}(\log^* n)$ time.

Space can be decreased to $\mathcal{O}(n)$ by adding $\mathcal{O}(\log^* n)$ to the query (which is absorbed anyway).

# Reducing the space

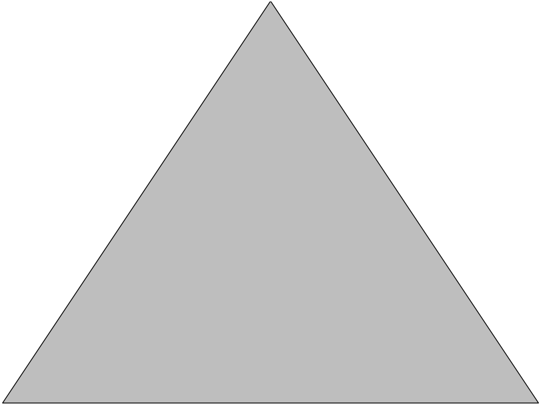
After $\mathcal{O}(n)$ preprocessing we can reduce any query to computing the longest common prefix of two paths of length $\leq \log^2 n$. Now the reasoning can be iterated, so in the next step we get two paths of length $\leq \log^2(\log^2 n)$, and so on.

The number of iterations is at most $\mathcal{O}(\log^* n)$, so after $\mathcal{O}(n \log^* n)$ preprocessing we can answer any query in $\mathcal{O}(\log^* n)$ time.

Space can be decreased to $\mathcal{O}(n)$ by adding $\mathcal{O}(\log^* n)$ to the query (which is absorbed anyway).

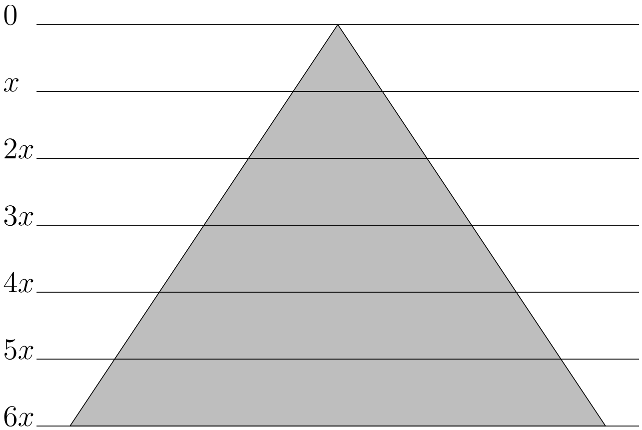
# Difference covers for trees

Generalizing the construction used for strings, the first step is to mark every node at depth $0, x, 2x, 3x, \ldots$.



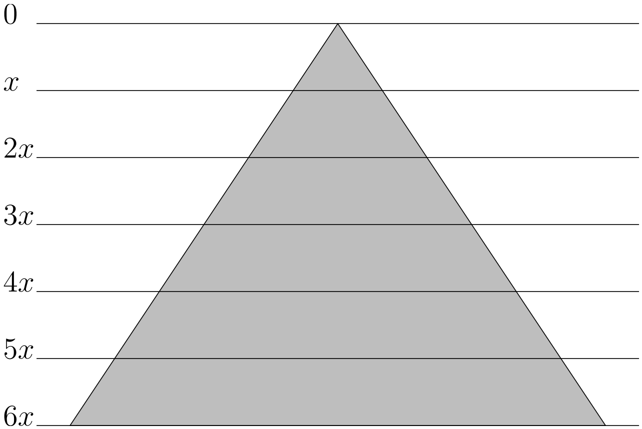
# Difference covers for trees

Generalizing the construction used for strings, the first step is to mark every node at depth $0, x, 2x, 3x, \ldots$.



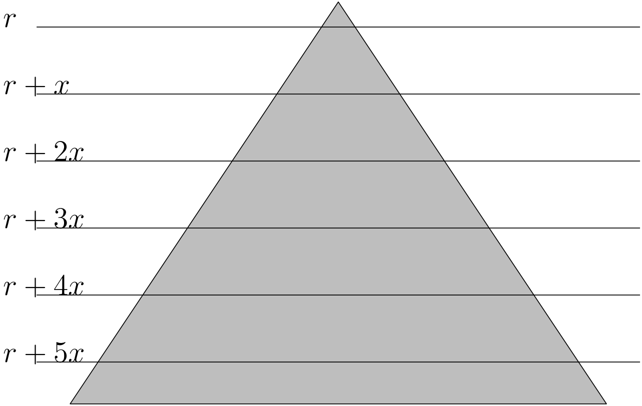
# Difference covers for trees

Doesn't quite work, because we cannot guarantee that there are $\frac{n}{x}$ such nodes.



# Difference covers for trees

But marking every node at depth $r, r+x, r+2x, r+3x, \ldots$ is also enough for our purposes, where $r \in \{0, 1, \ldots, x-1\}$.



# Difference covers for trees

**Trick**

For at least one $r \in \{0, 1, \ldots, x-1\}$ the number of marked nodes is at most $\frac{n}{x}$.

The construction essentially mimics the usual construction for strings, repeating the above trick twice, hence the total number of marked nodes is $\frac{2n}{x}$.

# Difference covers for trees

**Trick**

For at least one $r \in \{0, 1, \ldots, x-1\}$ the number of marked nodes is at most $\frac{n}{x}$.

The construction essentially mimics the usual construction for strings, repeating the above trick twice, hence the total number of marked nodes is $\frac{2n}{x}$.

# Path-tree queries

Also uses difference covers for trees, but now there are $\log\log n$ iterations, and every of them needs predecessor search, hence the query time is $\mathcal{O}((\log\log n)^2)$.

**Lowerbound**

By reduction from the predecessor problem, we show that any structure of size $\tilde{\mathcal{O}}(n)$ needs $\Omega(\log\log n)$ time to answer a query.

# Path-tree queries

Also uses difference covers for trees, but now there are $\log\log n$ iterations, and every of them needs predecessor search, hence the query time is $\mathcal{O}((\log\log n)^2)$.

## Lowerbound

By reduction from the predecessor problem, we show that any structure of size $\tilde{\mathcal{O}}(n)$ needs $\Omega(\log\log n)$ time to answer a query.

# Tree-tree queries

Completely different idea! Micro-macro decomposition into fragments of size $\tau$ gives a trade-off between space and query.

**Lowerbound**

We show a reduction from the set-intersection problem, hence if a popular folklore conjecture holds, answering the queries in constant time requires $\tilde{\Omega}(n^2)$ space.

# Tree-tree queries

Completely different idea! Micro-macro decomposition into fragments of size $\tau$ gives a trade-off between space and query.

## Lowerbound

We show a reduction from the set-intersection problem, hence if a popular folklore conjecture holds, answering the queries in constant time requires $\tilde{\Omega}(n^2)$ space.

# Open problems

1. Remove $\mathcal{O}(\log^* n)$ from the path-path complexity.
2. Decrease $\mathcal{O}((\log\log n)^2)$ to $\mathcal{O}(\log\log n)$ for path-tree queries.
3. Reduce tree-tree queries to set-intersection queries.
4. Other applications of difference covers for trees?

# Open problems

1. Remove $\mathcal{O}(\log^* n)$ from the path-path complexity.
2. Decrease $\mathcal{O}((\log\log n)^2)$ to $\mathcal{O}(\log\log n)$ for path-tree queries.
3. Reduce tree-tree queries to set-intersection queries.
4. Other applications of difference covers for trees?

# Open problems

1. Remove $\mathcal{O}(\log^* n)$ from the path-path complexity.
2. Decrease $\mathcal{O}((\log\log n)^2)$ to $\mathcal{O}(\log\log n)$ for path-tree queries.
3. Reduce tree-tree queries to set-intersection queries.
4. Other applications of difference covers for trees?

# Open problems

1. Remove $\mathcal{O}(\log^* n)$ from the path-path complexity.
2. Decrease $\mathcal{O}((\log\log n)^2)$ to $\mathcal{O}(\log\log n)$ for path-tree queries.
3. Reduce tree-tree queries to set-intersection queries.
4. Other applications of difference covers for trees?

Questions?