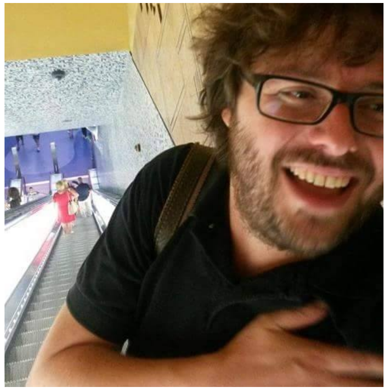
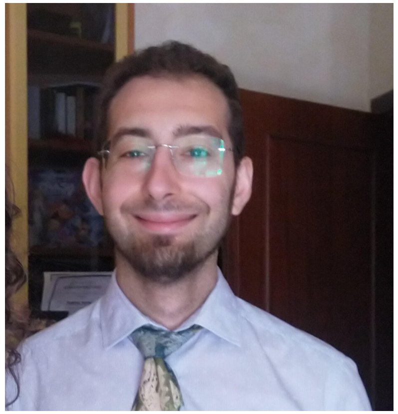
Finding Diameter-Reducing Shortcuts in Trees

Davide Bilò, Luciano Gualà, Stefano Leucci, Luca Pepè Sciarria





UNIVERSITÀ DEGLI STUDI DELL'AQUILA

Università di Roma Tor Vergata

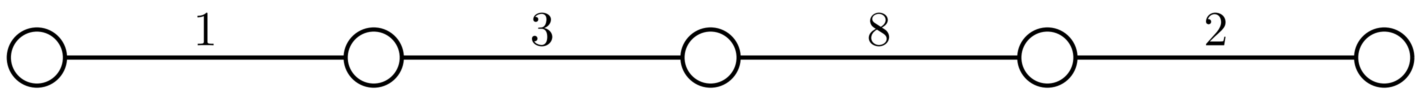
UNIVERSITÀ DEGLI STUDI DELL'AQUILA

Università di Roma Tor Vergata

Diameter-Optimally Augmenting a Path

Input:

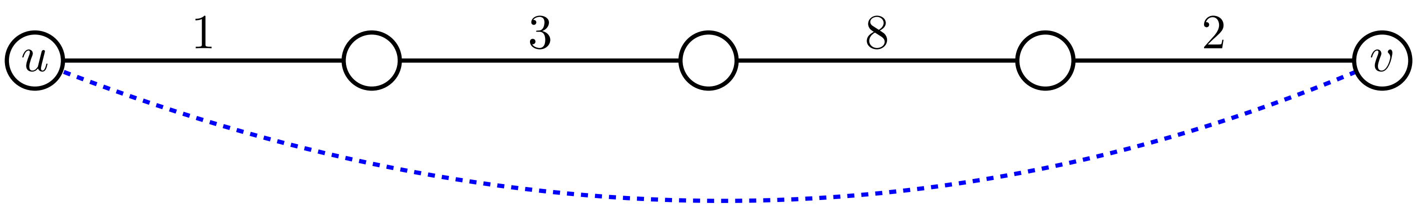
- A weighted path P with n vertices and non-negative edge costs



Diameter-Optimally Augmenting a Path

Input:

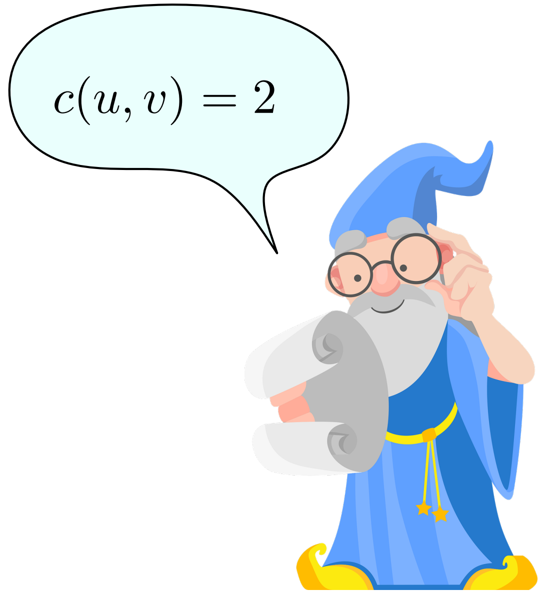
- A weighted path P with n vertices and non-negative edge costs
- Acccess to an oracle that can be queried with a *missing* edge (u, v) and reports the cost of (u, v)

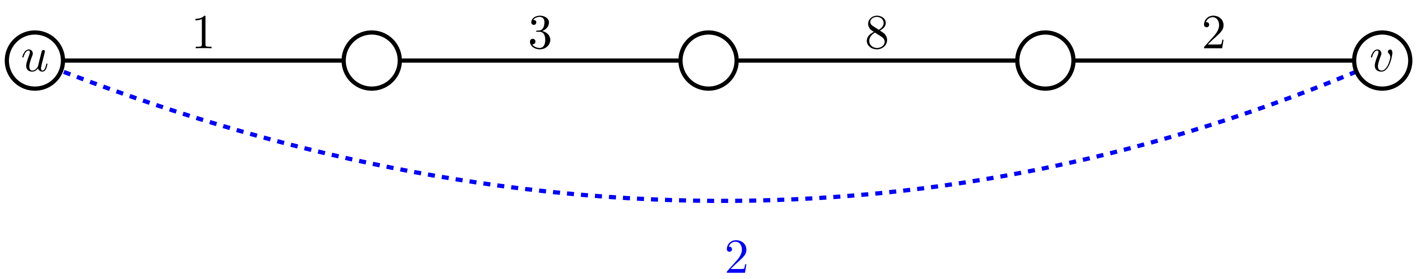


Diameter-Optimally Augmenting a Path

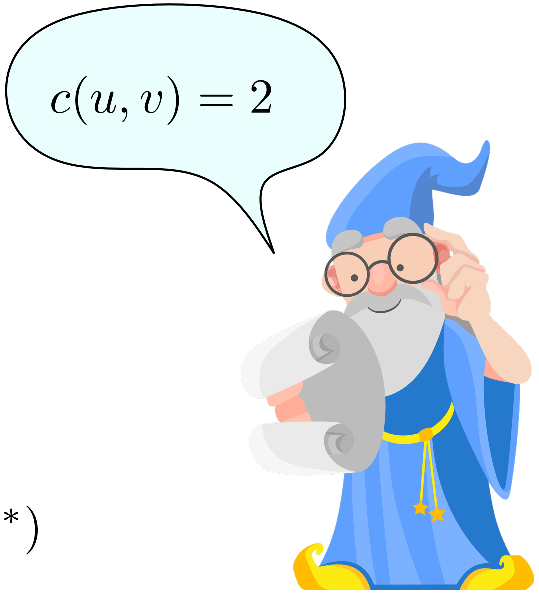
Input:

- A weighted path P with n vertices and non-negative edge costs
- Acccess to an oracle that can be queried with a *missing* edge (u, v) and reports the cost of (u, v)





Diameter-Optimally Augmenting a Path

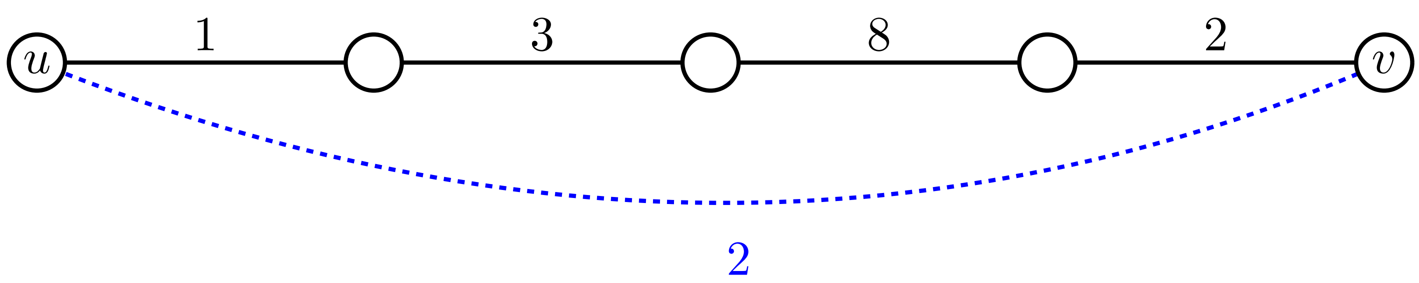


Input:

- A weighted path P with n vertices and non-negative edge costs
- Acccess to an oracle that can be queried with a *missing* edge (u, v) and reports the cost of (u, v)

Goal:

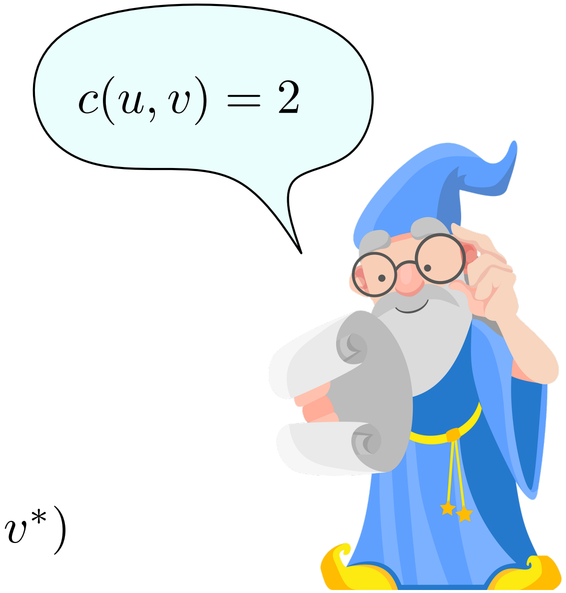
- Find a *shortcut* edge (u^*, v^*) that minimizes the diameter of $P + (u^*, v^*)$



Diameter-Optimally Augmenting a Path

Input:

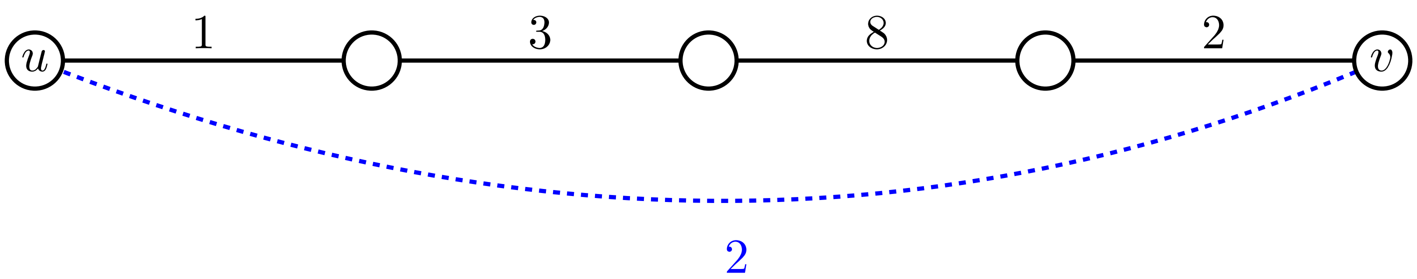
- A weighted path P with n vertices and non-negative edge costs
- Acccess to an oracle that can be queried with a *missing* edge (u, v) and reports the cost of (u, v)



Goal:

- Find a *shortcut* edge (u^*, v^*) that minimizes the diameter of $P + (u^*, v^*)$

diameter$(P + (u, v))$$= 8$



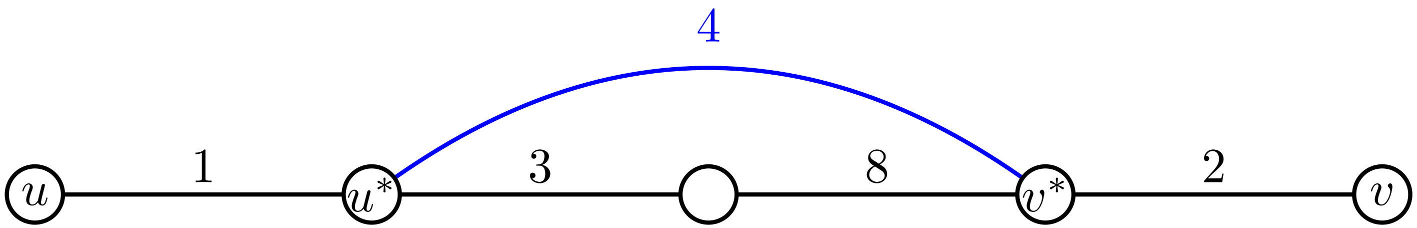
Diameter-Optimally Augmenting a Path

Input:

- A weighted path P with n vertices and non-negative edge costs
- Acccess to an oracle that can be queried with a *missing* edge (u, v) and reports the cost of (u, v)

Goal:

- Find a *shortcut* edge (u^*, v^*) that minimizes the diameter of $P + (u^*, v^*)$

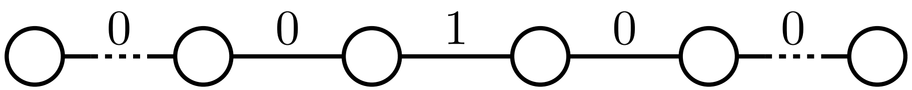


diameter($P + (u, v)$)$= 8$

diameter($P + (u^*, v^*)$)$= 7$

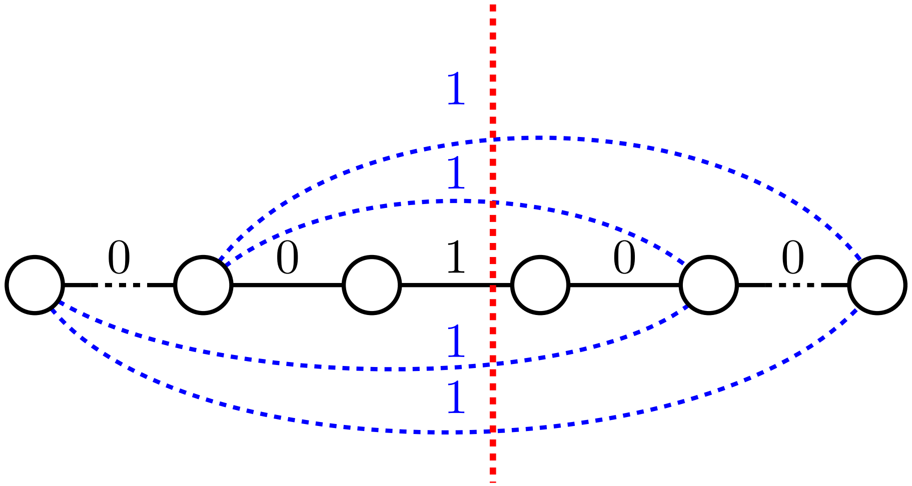
Diameter-Optimally Augmenting a Path

Requires $\Omega(n^2)$ queries/time in general:



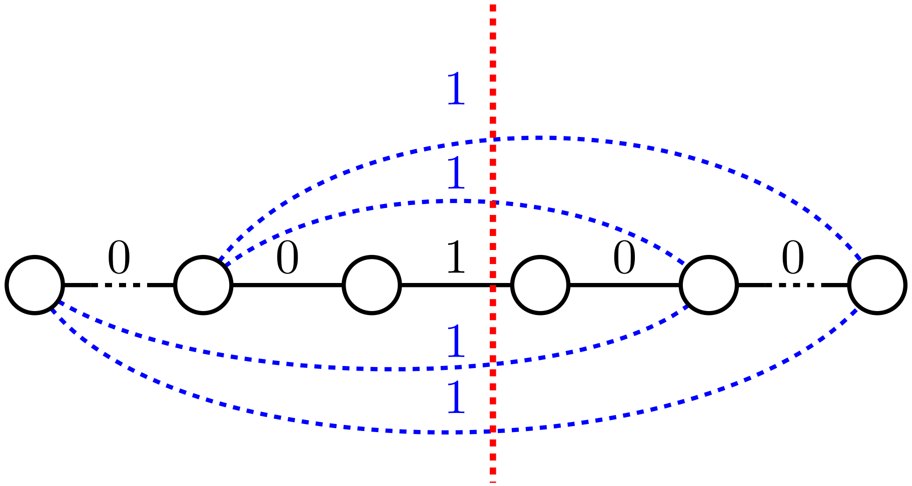
Diameter-Optimally Augmenting a Path

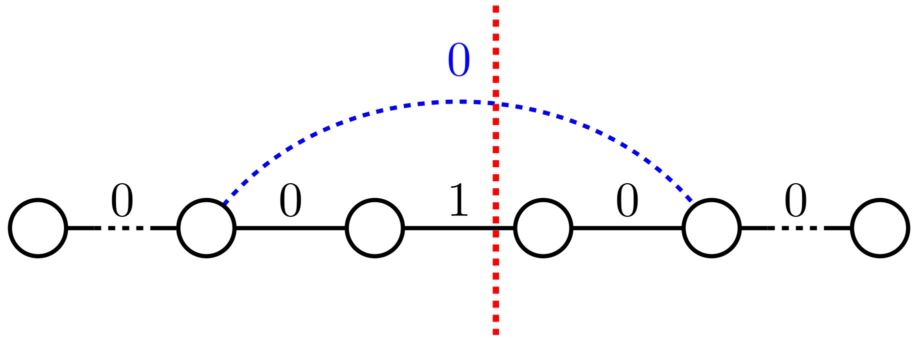
Requires $\Omega(n^2)$ queries/time in general:



Diameter-Optimally Augmenting a Path

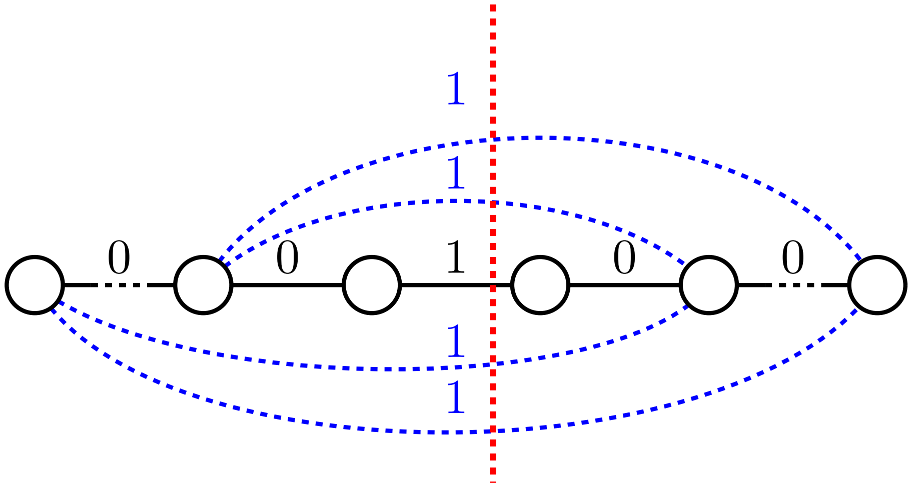
Requires $\Omega(n^2)$ queries/time in general:





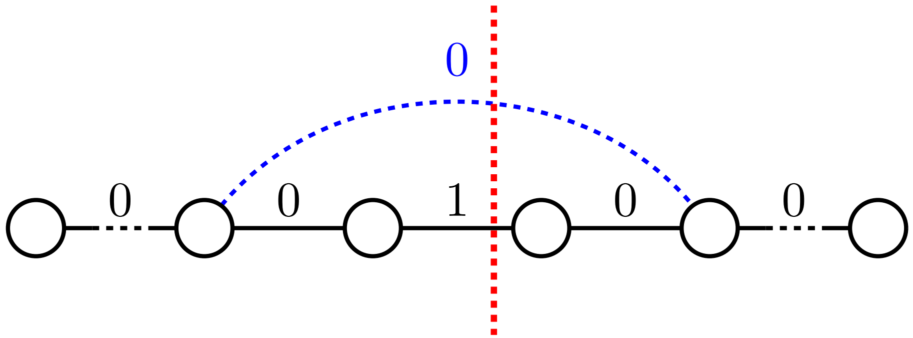
Diameter-Optimally Augmenting a Path

Requires $\Omega(n^2)$ queries/time in general:



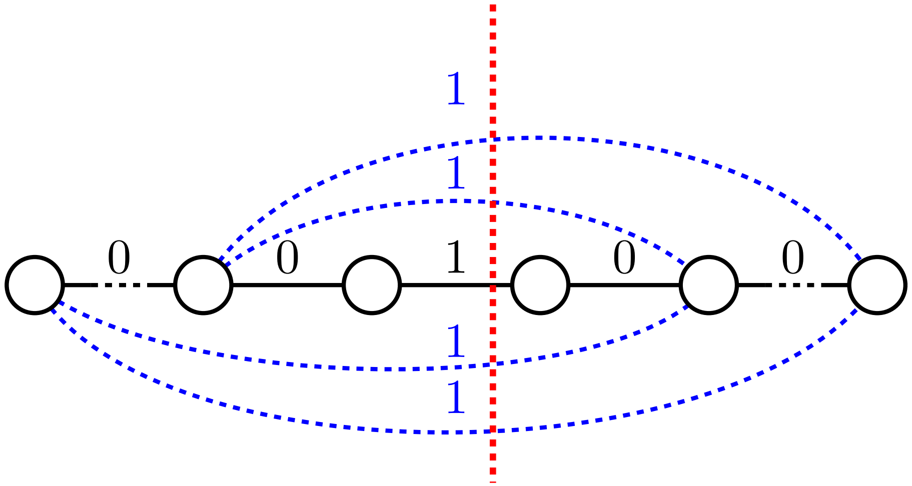
Can be solved in:

- $O(n^2 \log n)$ time, $O(n)$ space [Wang & Zhao TCS 2021]
- $O(n^2)$ time, $O(n \log n)$ space [Bilò, TCS 2022]



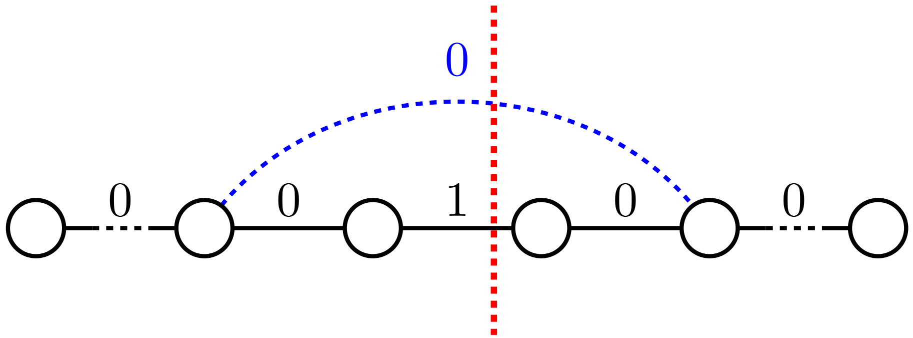
Diameter-Optimally Augmenting a Path

Requires $\Omega(n^2)$ queries/time in general:



Can be solved in:

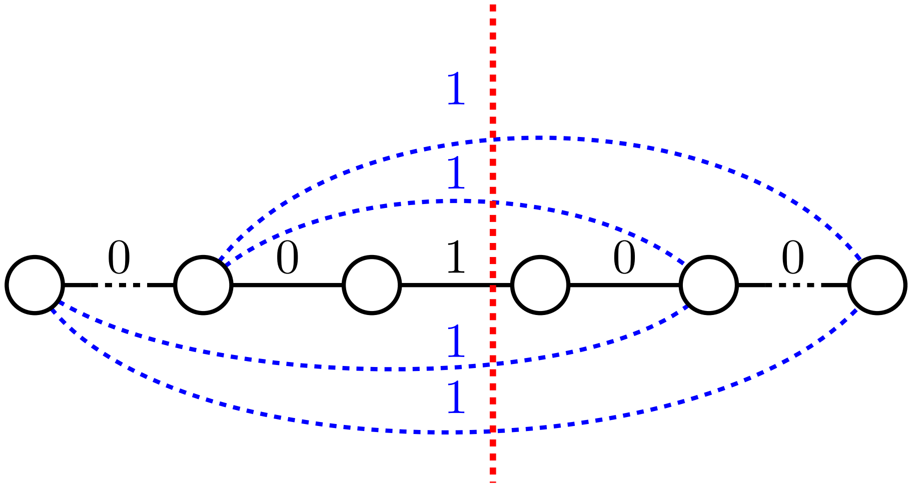
- $O(n^2 \log n)$ time, $O(n)$ space [Wang & Zhao TCS 2021]
- $O(n^2)$ time, $O(n \log n)$ space [Bilò, TCS 2022]



Metric opdimal diameter augmentation: The costs of all edges and non-edges satisfy the triangle inequality

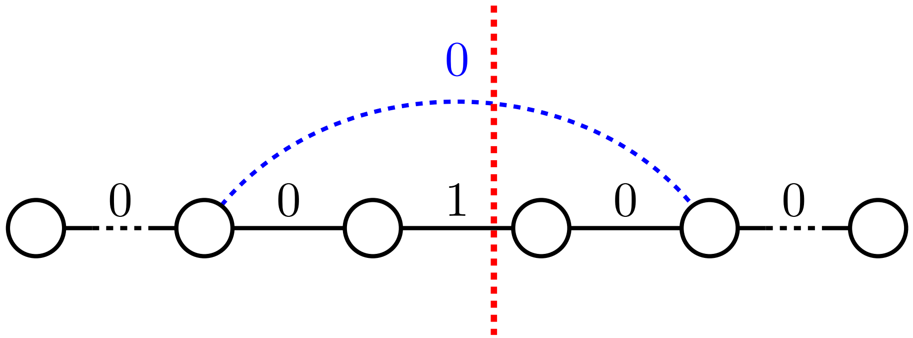
Diameter-Optimally Augmenting a Path

Requires $\Omega(n^2)$ queries/time in general:



Can be solved in:

- $O(n^2 \log n)$ time, $O(n)$ space [Wang & Zhao TCS 2021]
- $O(n^2)$ time, $O(n \log n)$ space [Bilò, TCS 2022]



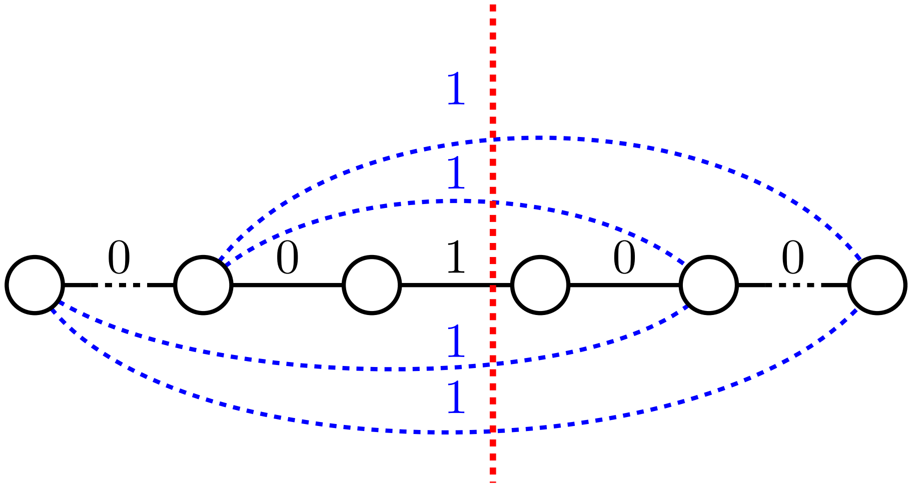
Metric opdimal diameter augmentation: The costs of all edges and non-edges satisfy the triangle inequality

Can be solved in:

- $O(n \log^3 n)$ time [Große et al., J. Found. Comput. Sci. 2019]
- $O(n \log n)$ time [Wang, Comput. Geom. 2018]

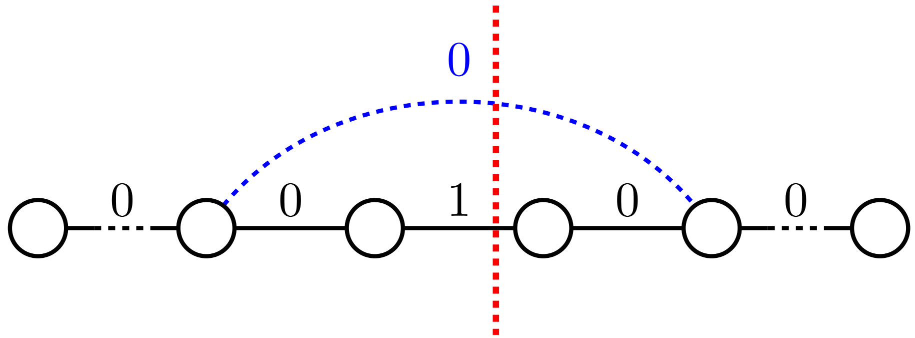
Diameter-Optimally Augmenting a Path

Requires $\Omega(n^2)$ queries/time in general:



Can be solved in:

- $O(n^2 \log n)$ time, $O(n)$ space [Wang & Zhao TCS 2021]
- $O(n^2)$ time, $O(n \log n)$ space [Bilò, TCS 2022]



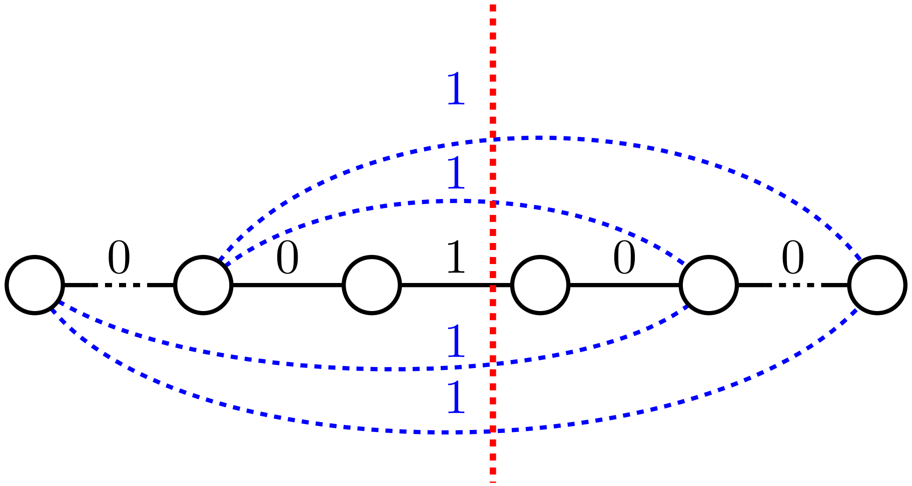
Metric opdimal diameter augmentation: The costs of all edges and non-edges satisfy the triangle inequality

Can be solved in:

- $O(n \log^3 n)$ time [Große et al., J. Found. Comput. Sci. 2019]
- $O(n \log n)$ time [Wang, Comput. Geom. 2018]
- $(1+\varepsilon)$-apx in time $O(n + \frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$ [Bilò, TCS 2022]

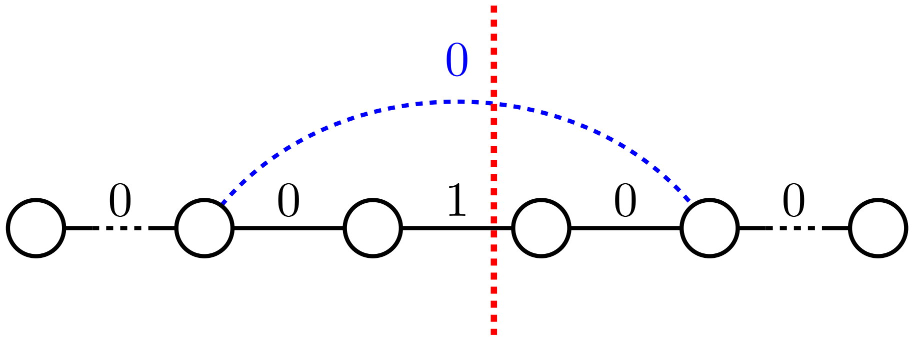
Diameter-Optimally Augmenting a Path

Requires $\Omega(n^2)$ queries/time in general:



Can be solved in:

- $O(n^2 \log n)$ time, $O(n)$ space [Wang & Zhao TCS 2021]
- $O(n^2)$ time, $O(n \log n)$ space [Bilò, TCS 2022]



Metric opdimal diameter augmentation: The costs of all edges and non-edges satisfy the triangle inequality

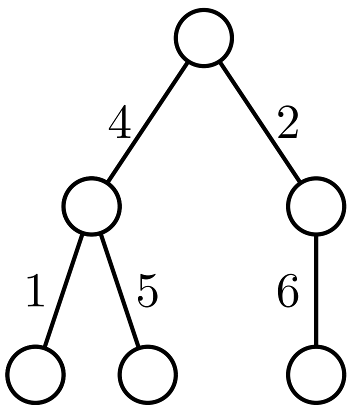
Can be solved in:

- $O(n \log^3 n)$ time [Große et al., J. Found. Comput. Sci. 2019]
- $O(n \log n)$ time [Wang, Comput. Geom. 2018]
- $(1+\varepsilon)$-apx in time $O(n + \frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$ [Bilò, TCS 2022] Also for trees!

k-Diameter-Optimally Augmenting a Tree

k-DOAT Input:

- A weighted tree T with n vertices and non-negative edge costs
- Acccess to an oracle that can be queried with a *missing* edge (u, v) and reports the cost of (u, v)



$k = 3$

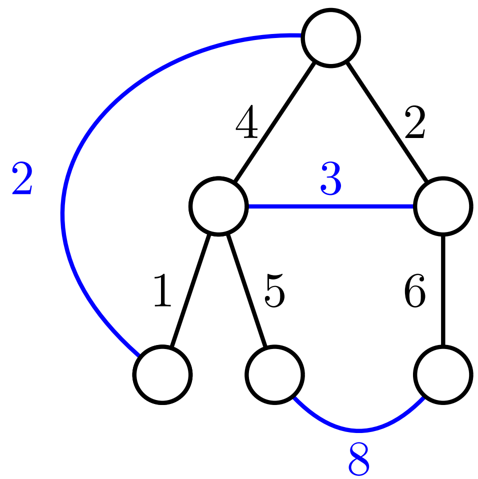
k-Diameter-Optimally Augmenting a Tree

k-DOAT Input:

- A weighted tree T with n vertices and non-negative edge costs
- Acccess to an oracle that can be queried with a *missing* edge (u, v) and reports the cost of (u, v)

Goal:

- Find a S set of at most k shortcuts that minimizes the diameter of $T + S$



$k = 3$

Our Results

Lower bound for metric k-DOAT:

- For $k \geq 3$: Any $< \frac{10}{9}$-approximation algorithm uses $\Omega(n^2)$ queries.

Our Results

Lower bound for metric k-DOAT:

- For $k \geq 3$: Any $< \frac{10}{9}$-approximation algorithm uses $\Omega(n^2)$ queries.

Approximation algorithms for metric k-DOAT:

- Linear-time 4-approximation algorithm for $k = O\left(\sqrt{\frac{n}{\log n}}\right)$
- Linear-time $(1+\varepsilon)$-approximation algorithm for $k = o(\sqrt{\log n})$ and trees with $O(n^{\frac{1}{(2k+2)^2}})$ leaves.

Our Results

Lower bound for metric k-DOAT:

- For $k \geq 3$: Any $< \frac{10}{9}$-approximation algorithm uses $\Omega(n^2)$ queries.

Approximation algorithms for metric k-DOAT:

- Linear-time 4-approximation algorithm for $k = O\left(\sqrt{\frac{n}{\log n}}\right)$
- Linear-time $(1+\varepsilon)$-approximation algorithm for $k = o(\sqrt{\log n})$ and trees with $O(n^{\frac{1}{(2k+2)^2}})$ leaves.

$\implies$ Paths admit a linear-time $(1+\varepsilon)$-apx algorithm. Dichotomy with trees!

Our Results

Lower bound for metric k-DOAT:

- For $k \geq 3$: Any $< \frac{10}{9}$-approximation algorithm uses $\Omega(n^2)$ queries.

Approximation algorithms for metric k-DOAT:

- Linear-time 4-approximation algorithm for $k = O\left(\sqrt{\frac{n}{\log n}}\right)$
- Linear-time $(1+\varepsilon)$-approximation algorithm for $k = o(\sqrt{\log n})$ and trees with $O(n^{\frac{1}{(2k+2)^2}})$ leaves.

  $\Longrightarrow$ Paths admit a linear-time $(1+\varepsilon)$-apx algorithm. Dichotomy with trees!

Exact algorithms (not necessarily metric):

- $O(nk \log n)$-time algorithm to find the diameter of a tree augmented with k edges

Our Results

Lower bound for metric k-DOAT:

- For $k \geq 3$: Any $< \frac{10}{9}$-approximation algorithm uses $\Omega(n^2)$ queries.

Approximation algorithms for metric k-DOAT:

- Linear-time 4-approximation algorithm for $k = O\left(\sqrt{\frac{n}{\log n}}\right)$
- Linear-time $(1+\varepsilon)$-approximation algorithm for $k = o(\sqrt{\log n})$ and trees with $O(n^{\frac{1}{(2k+2)^2}})$ leaves.

$\implies$ Paths admit a linear-time $(1+\varepsilon)$-apx algorithm. Dichotomy with trees!

Exact algorithms (not necessarily metric):

- $O(nk \log n)$-time algorithm to find the diameter of a tree augmented with k edges

$$\Downarrow$$

- $O(k \cdot n^{2k+1} \log n)$-time algorithm for k-DOAT

Our Results

Lower bound for metric k-DOAT:

- For $k \geq 3$: Any $< \frac{10}{9}$-approximation algorithm uses $\Omega(n^2)$ queries.

Approximation algorithms for metric k-DOAT:

- Linear-time 4-approximation algorithm for $k = O\left(\sqrt{\frac{n}{\log n}}\right)$
- Linear-time $(1+\varepsilon)$-approximation algorithm for $k = o(\sqrt{\log n})$ and trees with $O(n^{\frac{1}{(2k+2)^2}})$ leaves.

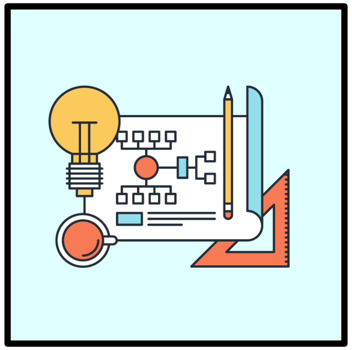
$\implies$ Paths admit a linear-time $(1+\varepsilon)$-apx algorithm. Dichotomy with trees!

Exact algorithms (not necessarily metric):

- $O(nk \log n)$-time algorithm to find the diameter of a tree augmented with k edges

$$\Downarrow$$

- $O(k \cdot n^{2k+1} \log n)$-time algorithm for k-DOAT



Our Results

Lower bound for metric k-DOAT:

- For $k \geq 3$: Any $< \frac{10}{9}$-approximation algorithm uses $\Omega(n^2)$ queries.

Approximation algorithms for metric k-DOAT:

- Linear-time 4-approximation algorithm for $k = O\left(\sqrt{\frac{n}{\log n}}\right)$
- Linear-time $(1+\varepsilon)$-approximation algorithm for $k = o(\sqrt{\log n})$ and trees with $O(n^{\frac{1}{(2k+2)^2}})$ leaves.

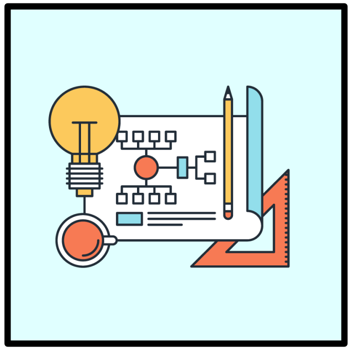
$\implies$ Paths admit a linear-time $(1+\varepsilon)$-apx algorithm. Dichotomy with trees!

Exact algorithms (not necessarily metric):

- $O(nk \log n)$-time algorithm to find the diameter of a tree augmented with k edges

$$\Downarrow$$

- $O(k \cdot n^{2k+1} \log n)$-time algorithm for k-DOAT



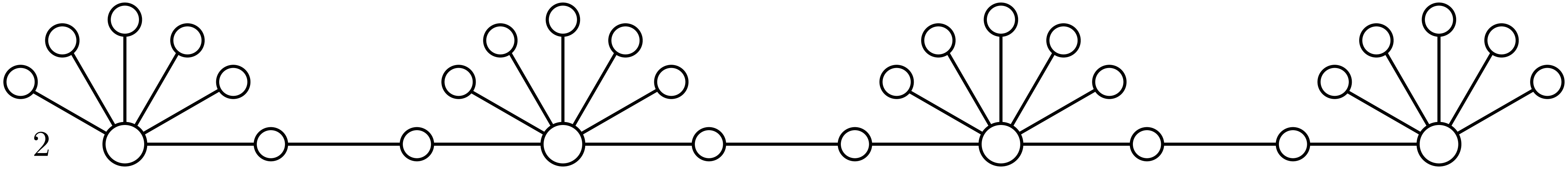
Our Lower Bound for $k \geq 3$

Our Lower Bound for $k \geq 3$

Consider $k = 3$ for simplicity

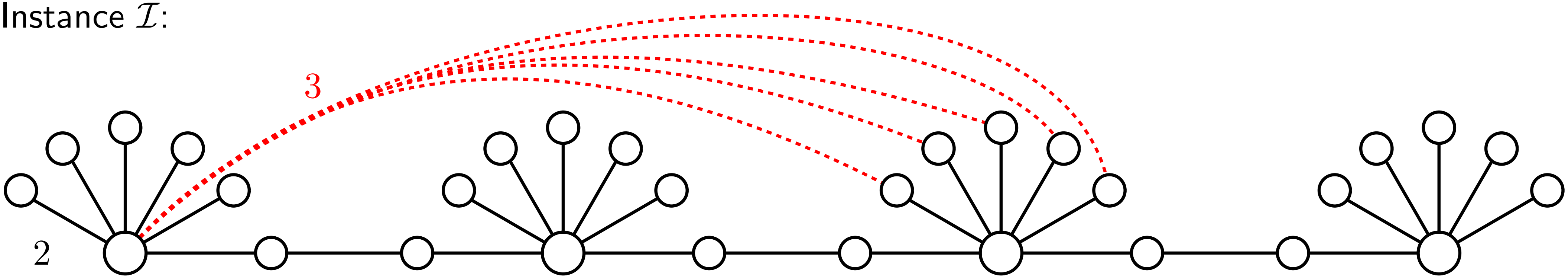
Our Lower Bound for $k = 3$

Instance $\mathcal{I}$:



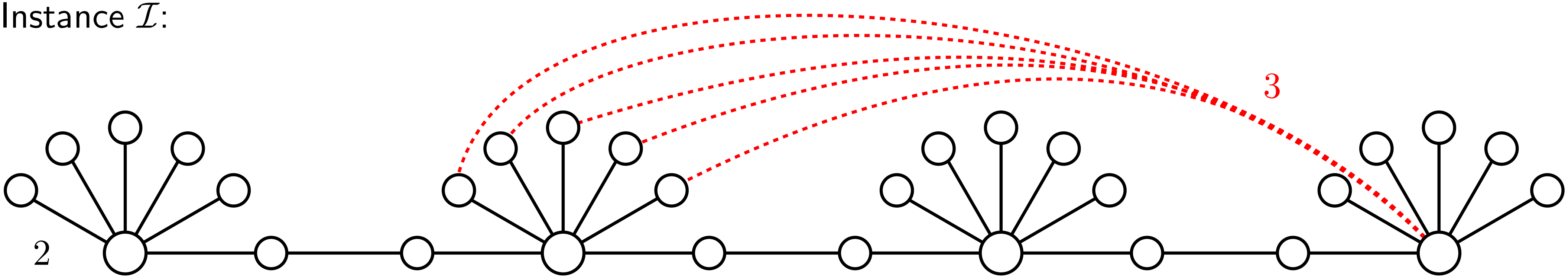
Our Lower Bound for $k = 3$

Instance $\mathcal{I}$:



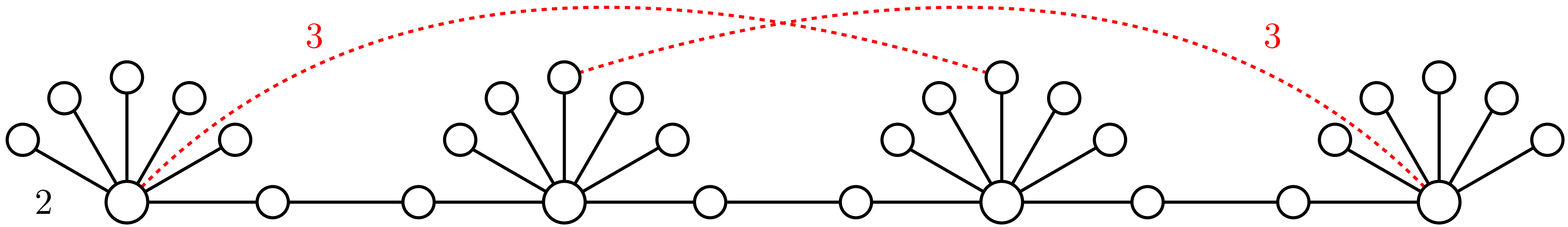
Our Lower Bound for $k = 3$

Instance $\mathcal{I}$:



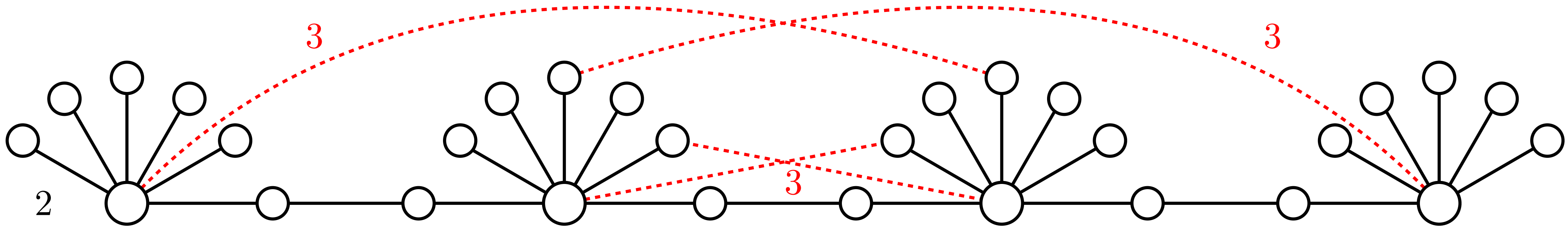
Our Lower Bound for $k = 3$

Instance $\mathcal{I}$:



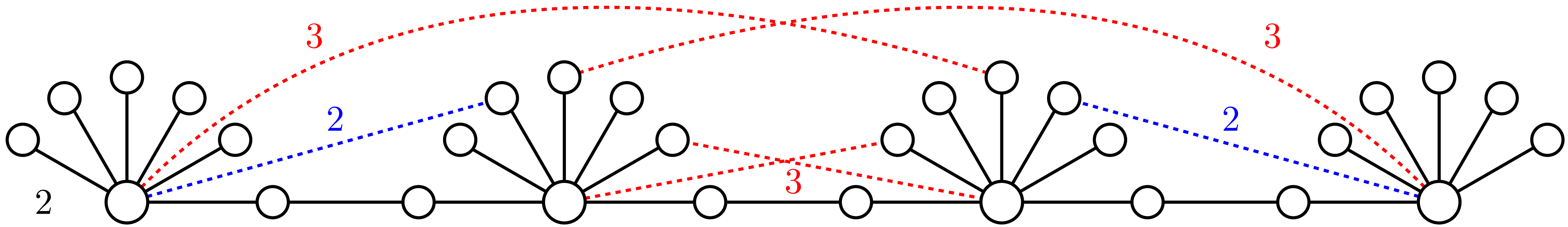
Our Lower Bound for $k = 3$

Instance $\mathcal{I}$:



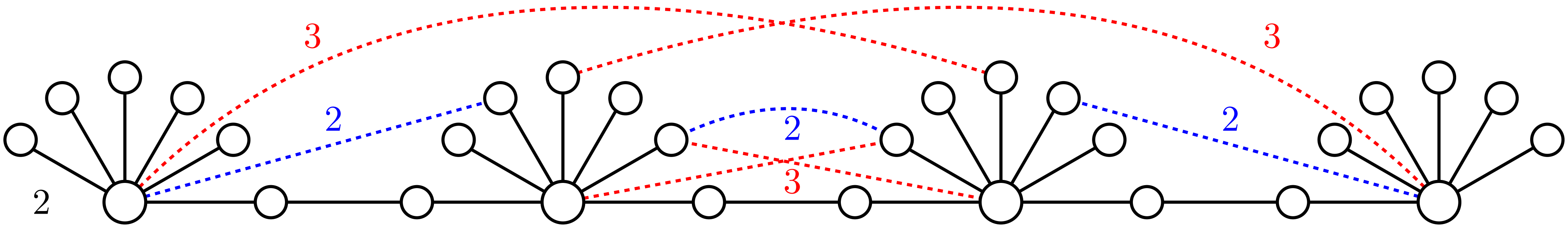
Our Lower Bound for $k = 3$

Instance $\mathcal{I}$:



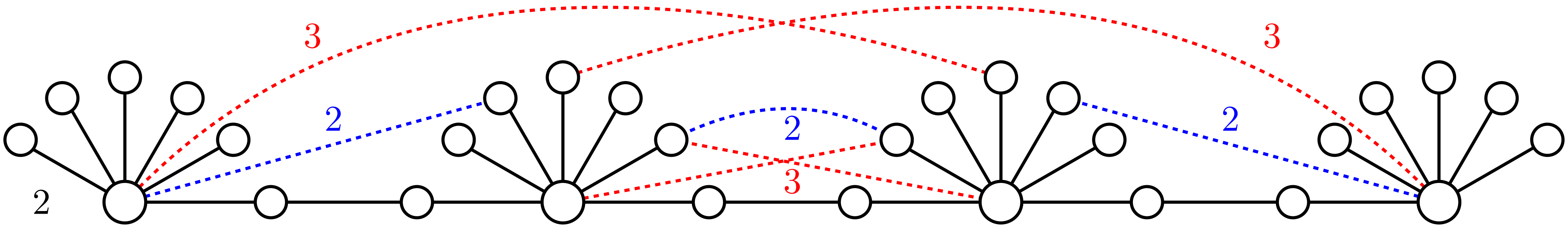
Our Lower Bound for $k = 3$

Instance $\mathcal{I}$:



Our Lower Bound for $k = 3$

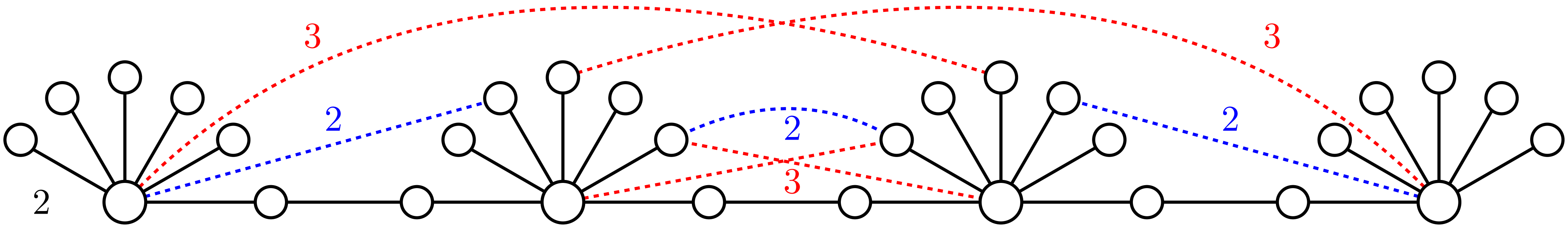
Instance $\mathcal{I}$:



$+$ Metric closure

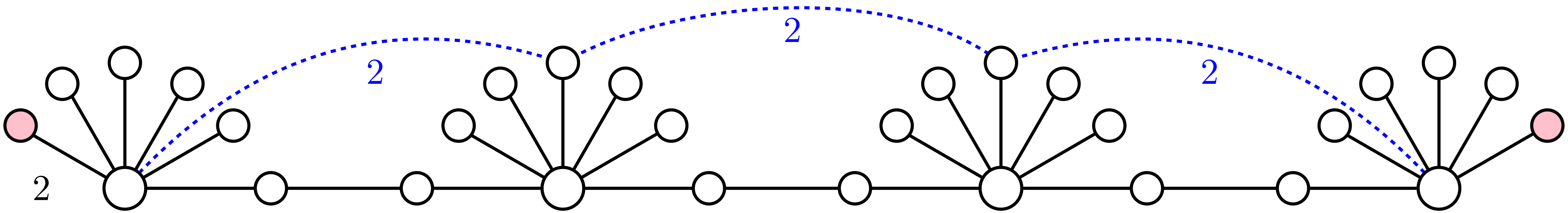
Our Lower Bound for $k = 3$

Instance $\mathcal{I}$:



$+$ Metric closure

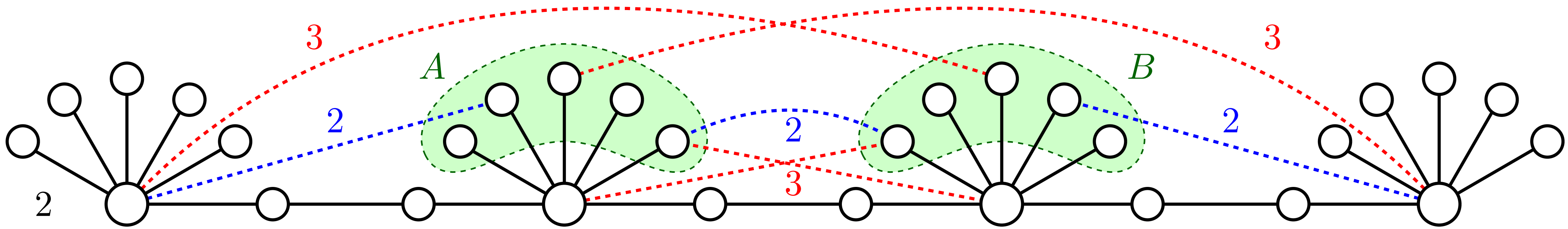
For any set S of ≤ 3 shortcuts, diam$(T + S) \geq 10$



Our Lower Bound for $k = 3$

Pick $a \in A$ and $b \in B$

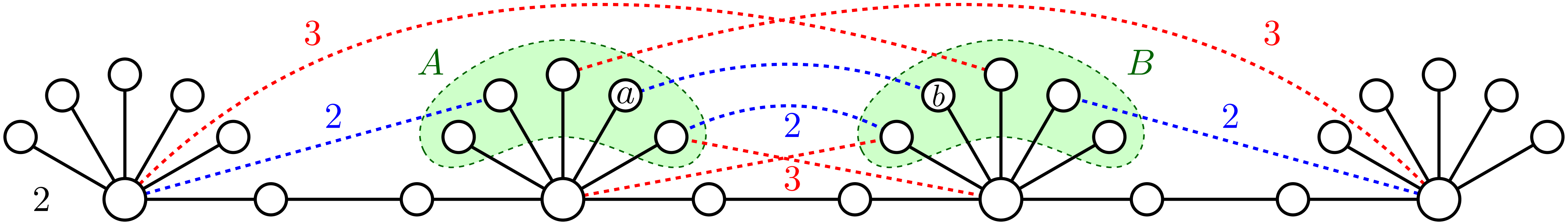
Instance $\mathcal{I}_{a,b}$:



Our Lower Bound for $k = 3$

Pick $a \in A$ and $b \in B$

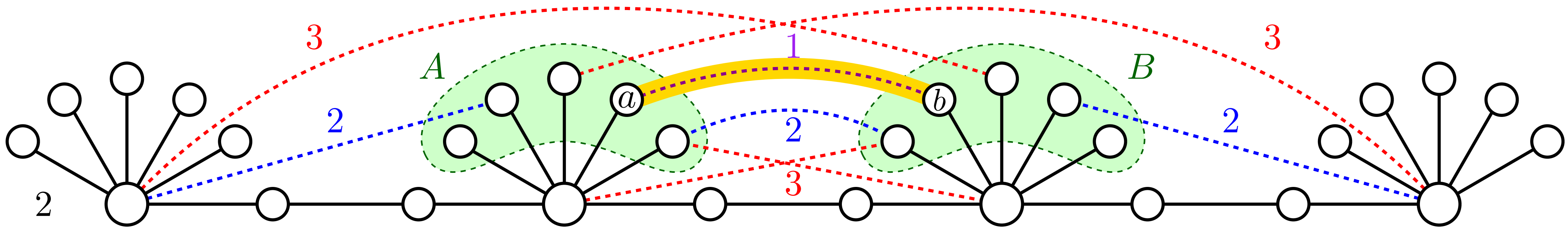
Instance $\mathcal{I}_{a,b}$:



Our Lower Bound for $k = 3$

Pick $a \in A$ and $b \in B$ and lower the cost of (a, b) to 1

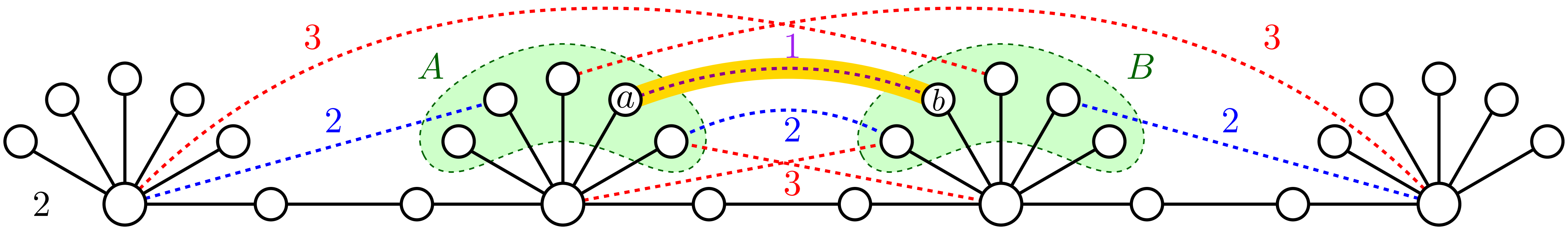
Instance $\mathcal{I}_{a,b}$:



Our Lower Bound for $k = 3$

Pick $a \in A$ and $b \in B$ and lower the cost of (a, b) to 1

Instance $\mathcal{I}_{a,b}$:

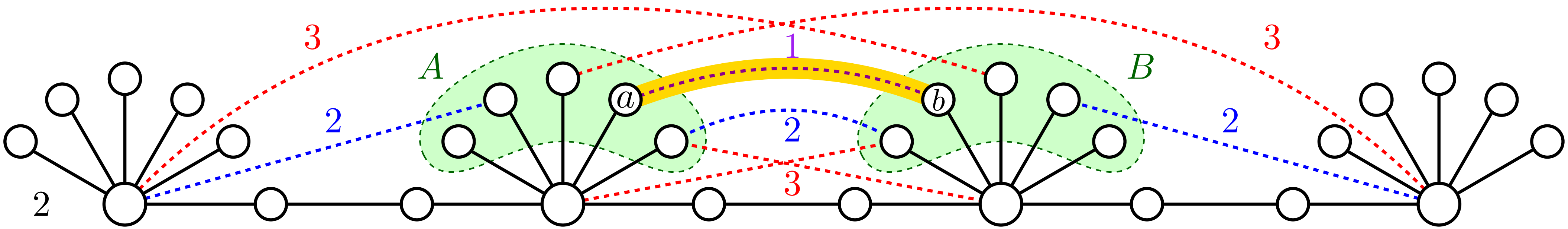


$+$ Metric closure

Our Lower Bound for $k = 3$

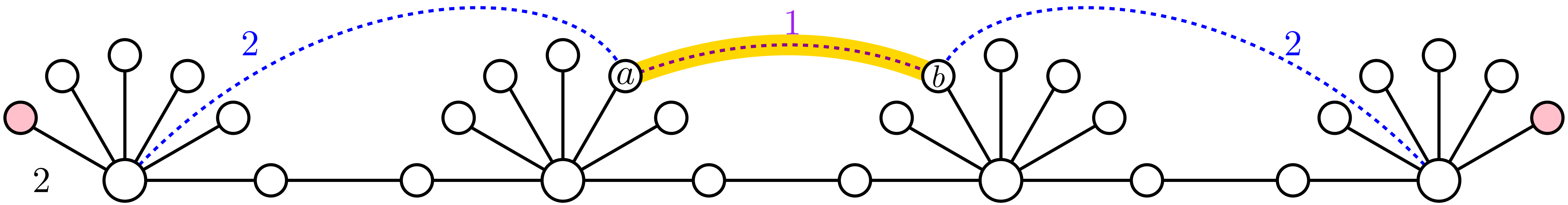
Pick $a \in A$ and $b \in B$ and lower the cost of (a, b) to 1

Instance $\mathcal{I}_{a,b}$:



$+$ Metric closure

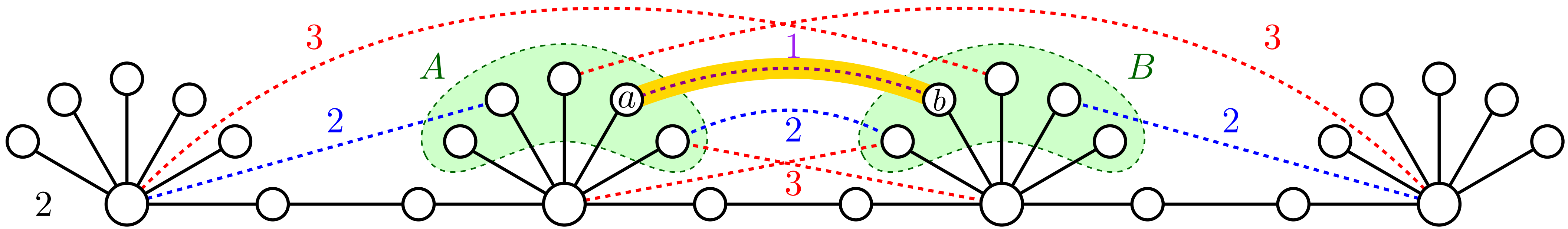
There is a set S of 3 shortcuts such that $\text{diam}(T + S) = 9$



Our Lower Bound for $k = 3$

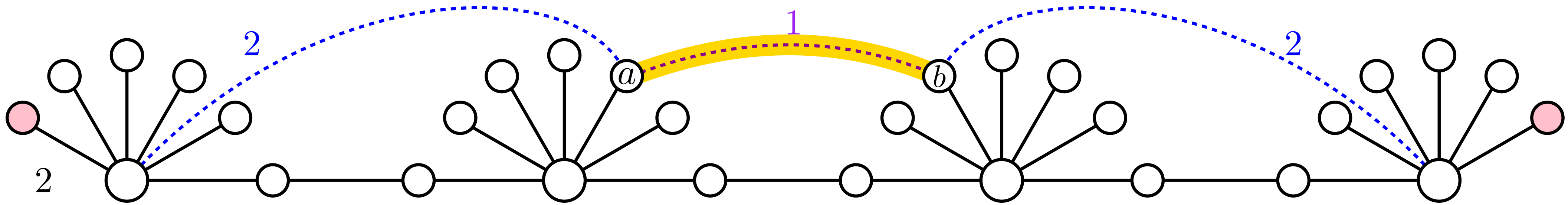
Pick $a \in A$ and $b \in B$ and lower the cost of (a, b) to 1

Instance $\mathcal{I}_{a,b}$:



$+$ Metric closure

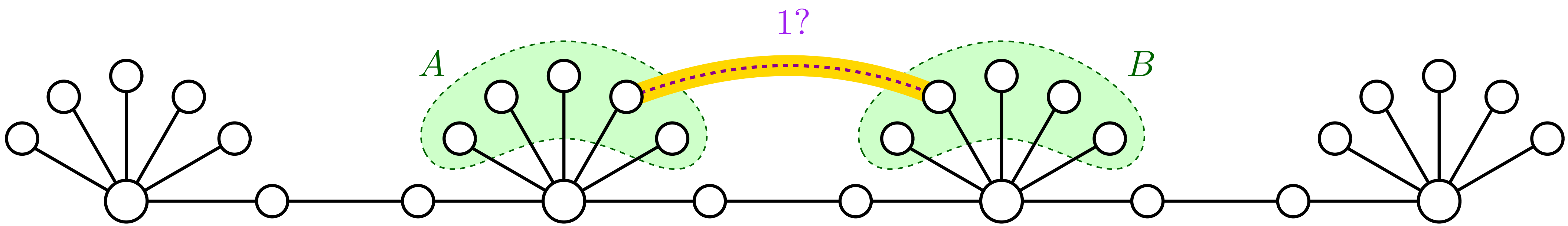
There is a set S of 3 shortcuts such that $\text{diam}(T + S) = 9$



Key Property: the cost all edges, except for (a, b) is the same in $\mathcal{I}$ and $\mathcal{I}_{a,b}$

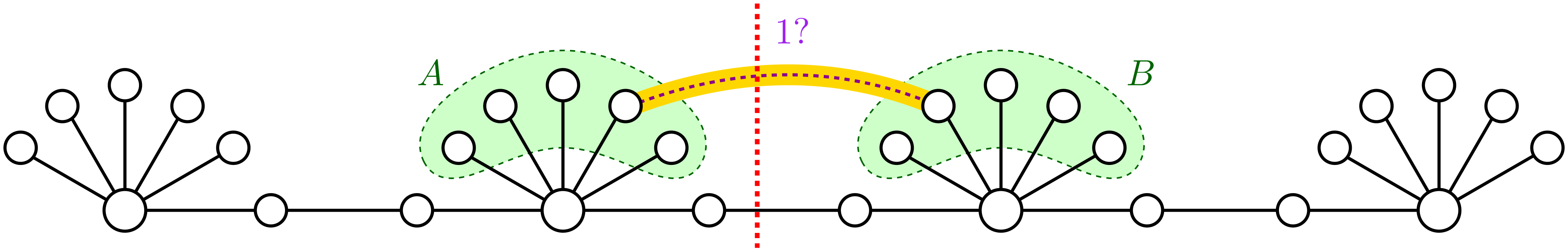
Our Lower Bound for $k = 3$

Input: Either $\mathcal{I}$ or $\mathcal{I}_{a,b}$ for some $(a, b) \in A \times B$



Our Lower Bound for $k = 3$

Input: Either $\mathcal{I}$ or $\mathcal{I}_{a,b}$ for some $(a, b) \in A \times B$

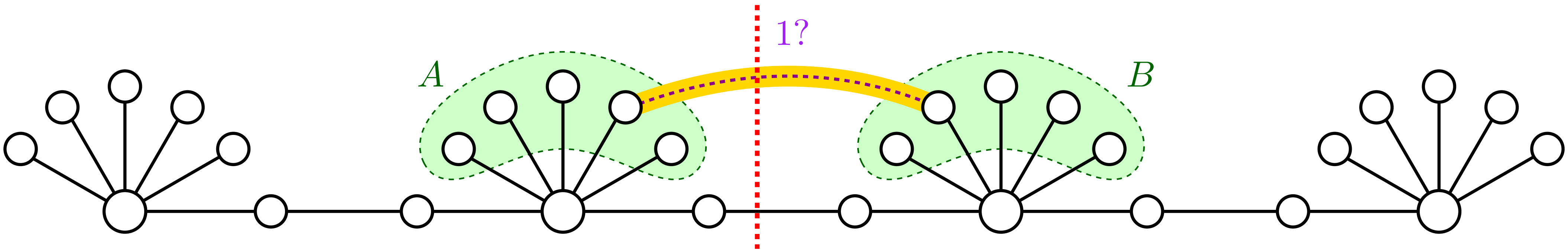


An algorithm that solves 3-DOAT needs to tell $\mathcal{I}$ apart from $\mathcal{I}_{a,b}$

... which requires querying all edges in $A \times B$

Our Lower Bound for $k = 3$

Input: Either $\mathcal{I}$ or $\mathcal{I}_{a,b}$ for some $(a, b) \in A \times B$



An algorithm that solves 3-DOAT needs to tell $\mathcal{I}$ apart from $\mathcal{I}_{a,b}$

... which requires querying all edges in $A \times B$

Actually shows: there is no $o(n^2)$-queries/time σ-approximation algorithm with $\sigma < \frac{10}{9}$

Our Exact Algorithm

(Speeding up the) Naive Strategy

- For every possible set S of k shortcuts: $O(n^{2k})$ choices
  - Compute the diameter of $T + S$ $\widetilde{O}(n^2)$ time

(Speeding up the) Naive Strategy

- For every possible set S of k shortcuts: $O(n^{2k})$ choices
  - Compute the diameter of $T + S$ $\widetilde{O}(n^2)$ time

Total running time: $\widetilde{O}(n^{2k+2})$

(Speeding up the) Naive Strategy

- For every possible set S of k shortcuts: $O(n^{2k})$ choices
  - Compute the diameter of $T + S$ $\widetilde{O}(n^2)$ time

Total running time: $\widetilde{O}(n^{2k+2})$

(Speeding up the) Naive Strategy

- For every possible set S of k shortcuts: $O(n^{2k})$ choices
    - Compute the diameter of $T + S$ ~~$\widetilde{O}(n^2)$ time~~ $\widetilde{O}(k \cdot n)$ time

Total running time: $\widetilde{O}(n^{2k+2})$

(Speeding up the) Naive Strategy

- For every possible set S of k shortcuts: $O(n^{2k})$ choices
  - Compute the diameter of $T + S$ $\cancel{\widetilde{O}(n^2)\text{ time}}$ $\widetilde{O}(k \cdot n)$ time

Total running time: $\cancel{\widetilde{O}(n^{2k+2})}$ $\widetilde{O}(k \cdot n^{2k+1})$ time

(Speeding up the) Naive Strategy

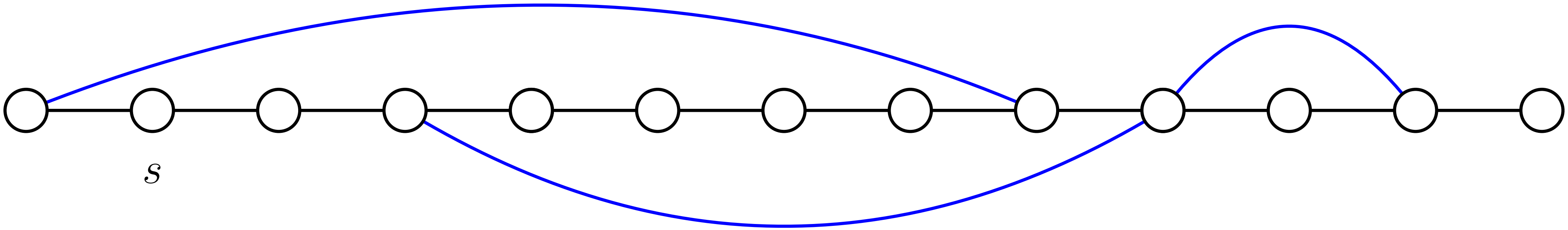
- For every possible set S of k shortcuts: $O(n^{2k})$ choices
  - Compute the diameter of $T + S$ $\cancel{\widetilde{O}(n^2) \text{ time}}$ $\widetilde{O}(k \cdot n)$ time

Total running time: $\cancel{\widetilde{O}(n^{2k+2})}$ $\widetilde{O}(k \cdot n^{2k+1})$ time

Some remarks:

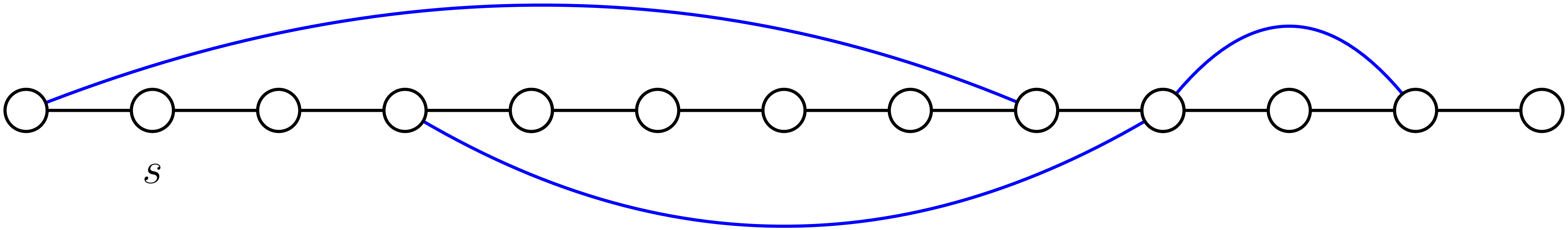
- Saves a $\Theta(n)$ factor
- Computing the diameter of a graph with $(n-1) + k$ edges is interesting in its own regard

Warm-up: Diameter of an Augmented Path $P + S$



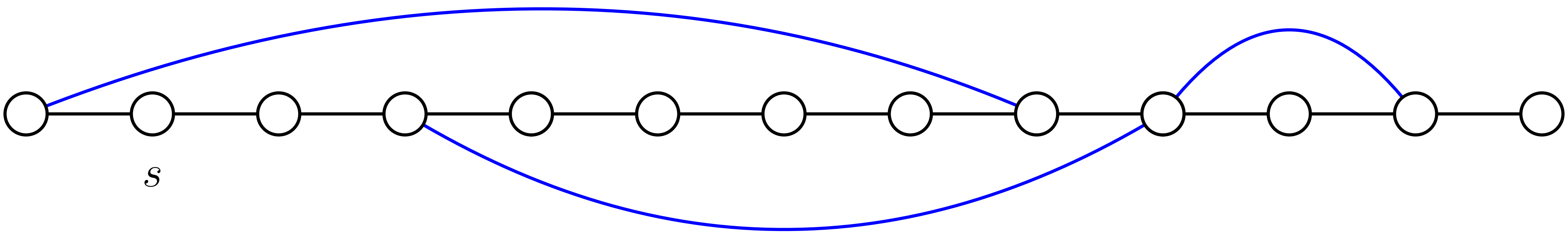
- For every *source* vertex s — $O(n)$ choices
  - Compute the eccentricity $\mathcal{E}(s)$ of s in $P + S$

Warm-up: Diameter of an Augmented Path $P + S$



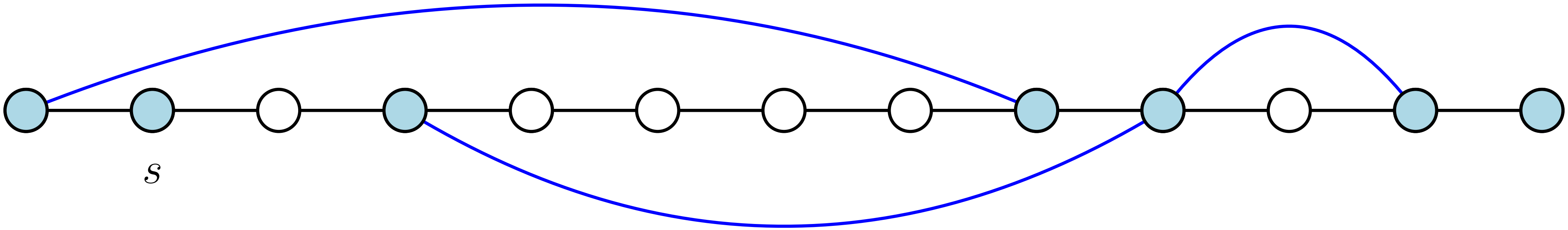
- For every *source* vertex s — $O(n)$ choices
  - Compute the eccentricity $\mathcal{E}(s)$ of s in $P + S$ — Want: $\widetilde{O}(k)$ time

Warm-up: Diameter of an Augmented Path $P + S$



- For every *source* vertex s — $O(n)$ choices
  - Compute the eccentricity $\mathcal{E}(s)$ of s in $P + S$ — Want: $\widetilde{O}(k)$ time — **How?**

Warm-up: Diameter of an Augmented Path $P + S$



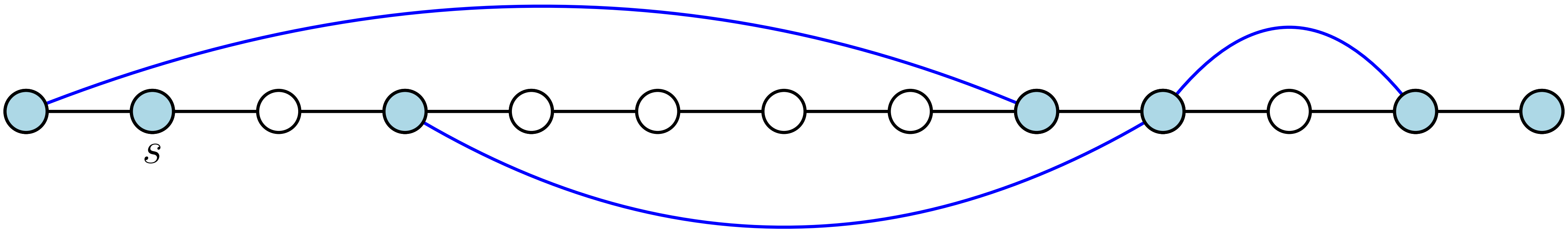
- For every *source* vertex s — $O(n)$ choices
  - Compute the eccentricity $\mathcal{E}(s)$ of s in $P + S$ — Want: $\widetilde{O}(k)$ time — **How?**

Mark s, the endpoints of the path, and all endpoints of some shortcut as **terminals**

Idea: if we know the distances from s to the **terminals**, we can quickly compute all other distances from s

Warm-up: Diameter of an Augmented Path $P + S$

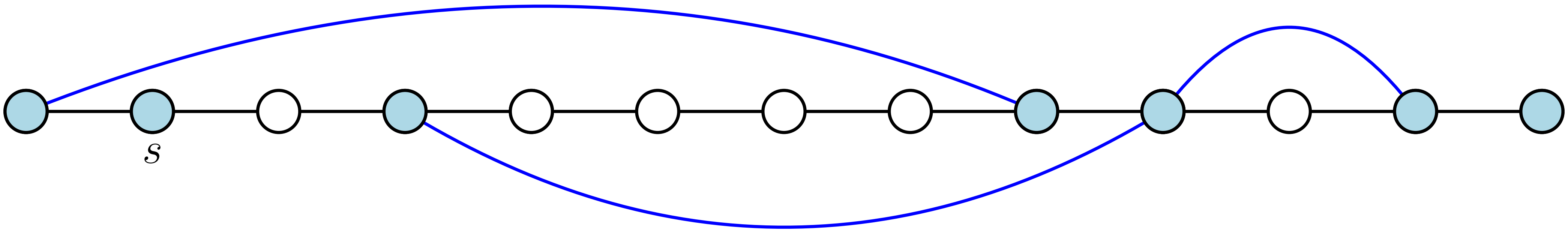
Subgoal: Quickly find all distances $\alpha_v = d_{P+S}(s, v)$ between s and all **terminals** v in $P + S$



Warm-up: Diameter of an Augmented Path $P + S$

Subgoal: Quickly find all distances $\alpha_v = d_{P+S}(s, v)$ between s and all **terminals** v in $P + S$

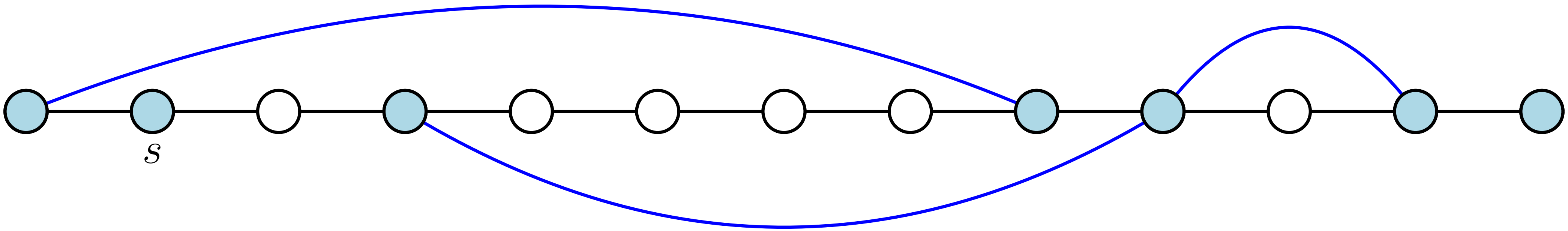
Assumption: constant time access to all-pairs distances in P (easy after a $O(n)$-time preprocessing)



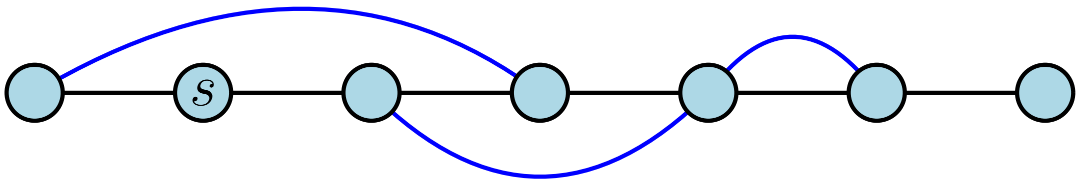
Warm-up: Diameter of an Augmented Path $P + S$

Subgoal: Quickly find all distances $\alpha_v = d_{P+S}(s, v)$ between s and all **terminals** v in $P + S$

Assumption: constant time access to all-pairs distances in P (easy after a $O(n)$-time preprocessing)



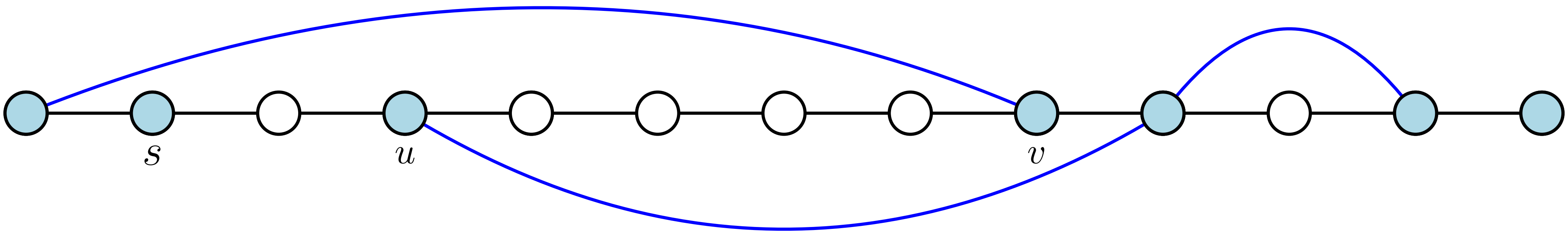
"**Shrink**" $P + S$ into H



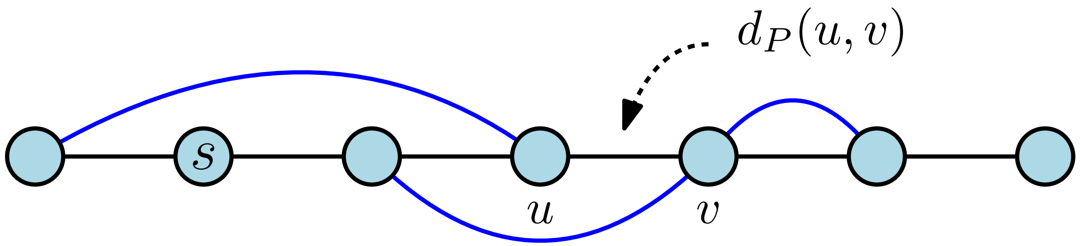
Warm-up: Diameter of an Augmented Path $P + S$

Subgoal: Quickly find all distances $\alpha_v = d_{P+S}(s, v)$ between s and all **terminals** v in $P + S$

Assumption: constant time access to all-pairs distances in P (easy after a $O(n)$-time preprocessing)



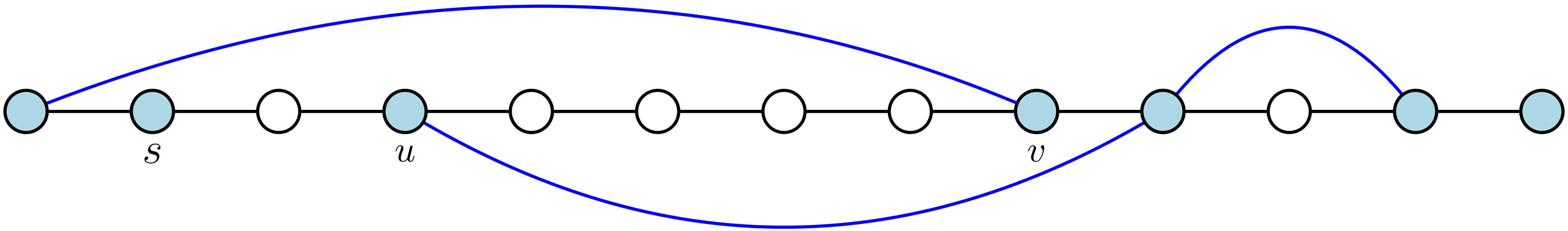
"**Shrink**" $P + S$ into H



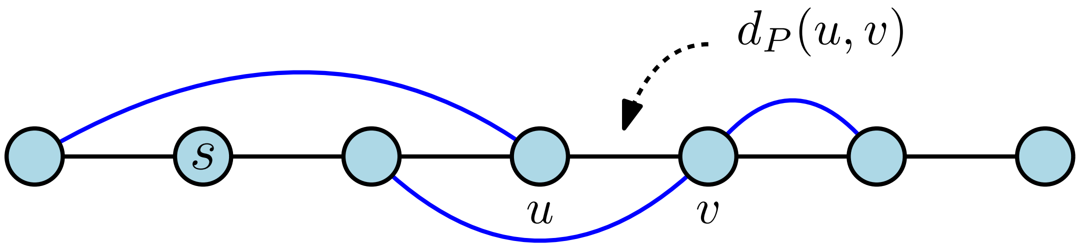
Warm-up: Diameter of an Augmented Path $P + S$

Subgoal: Quickly find all distances $\alpha_v = d_{P+S}(s, v)$ between s and all **terminals** v in $P + S$

Assumption: constant time access to all-pairs distances in P (easy after a $O(n)$-time preprocessing)



"**Shrink**" $P + S$ into H

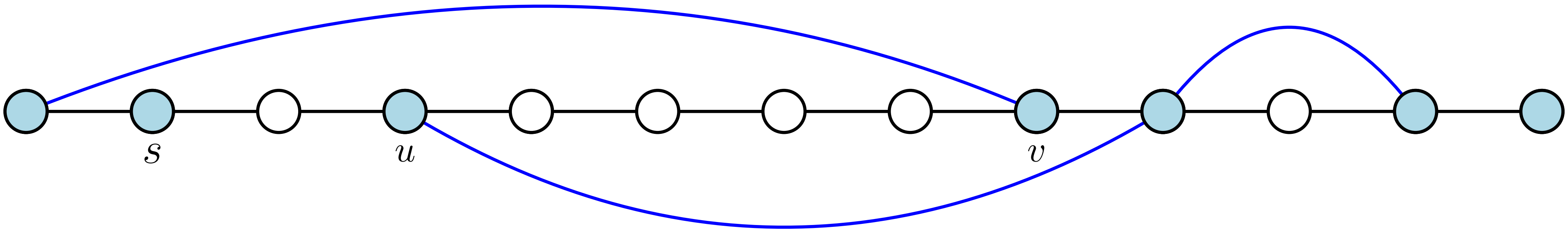


$O(k)$ vertices and $O(k)$ edges.

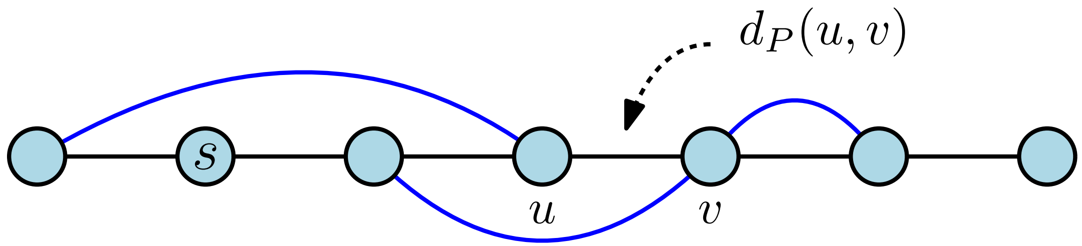
Warm-up: Diameter of an Augmented Path $P + S$

Subgoal: Quickly find all distances $\alpha_v = d_{P+S}(s, v)$ between s and all **terminals** v in $P + S$

Assumption: constant time access to all-pairs distances in P (easy after a $O(n)$-time preprocessing)



"**Shrink**" $P + S$ into H

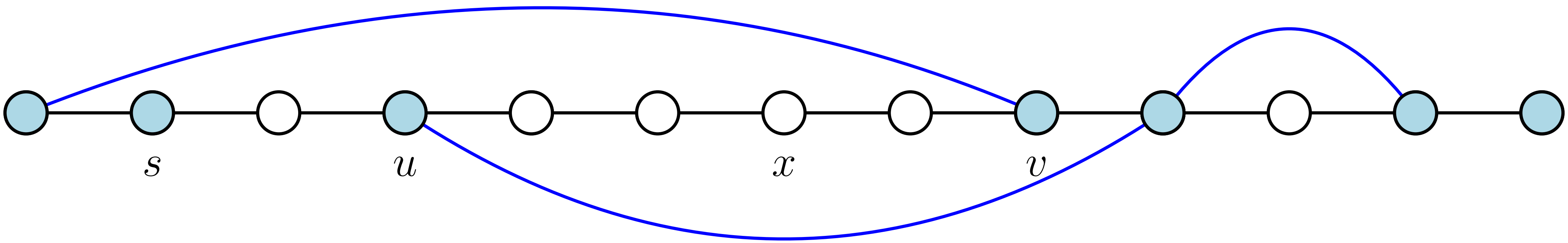


$O(k)$ vertices and $O(k)$ edges.

Compute $\alpha_v = d_H(s, v)$ for all terminals v in time $O(k \log k)$ using Dijkstra's algorithm .

Warm-up: Diameter of an Augmented Path $P + S$

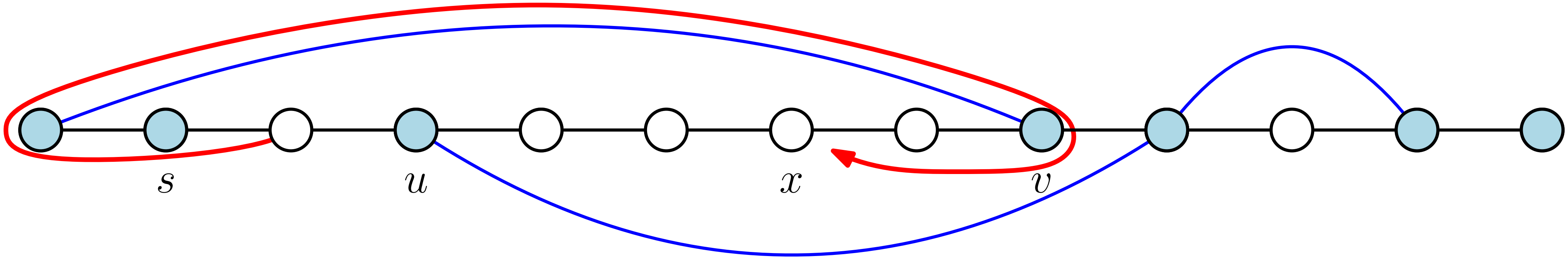
What about the other vertices?



The shortest path to a non-terminal vertex x must pass through one of its two neighboring terminal vertices u, v

Warm-up: Diameter of an Augmented Path $P + S$

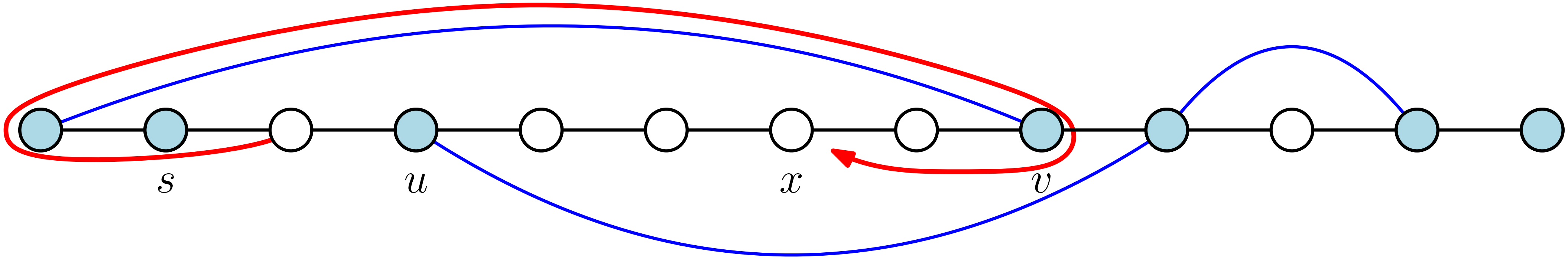
What about the other vertices?



The shortest path to a non-terminal vertex x must pass through one of its two neighboring terminal vertices u, v

Warm-up: Diameter of an Augmented Path $P + S$

What about the other vertices?

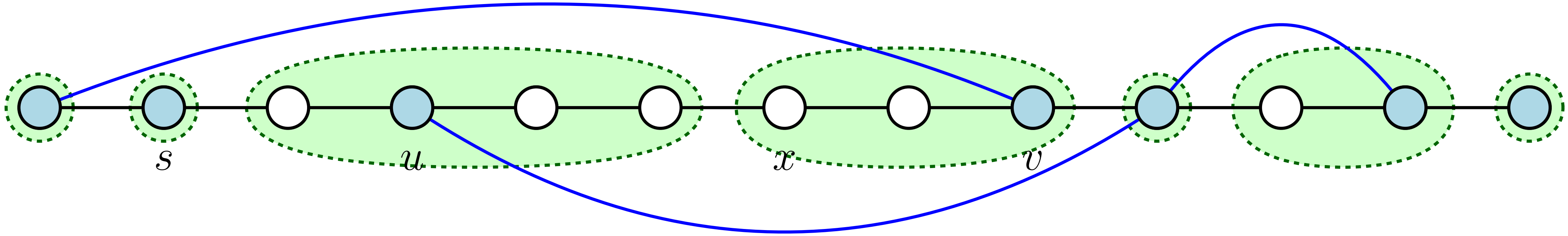


The shortest path to a non-terminal vertex x must pass through one of its two neighboring terminal vertices u, v

$$d_{P+S}(s, x) = \min \begin{cases} \alpha_u + d_P(u, x) \\ \alpha_v + d_P(v, x) \end{cases}$$

Warm-up: Diameter of an Augmented Path $P + S$

What about the other vertices?



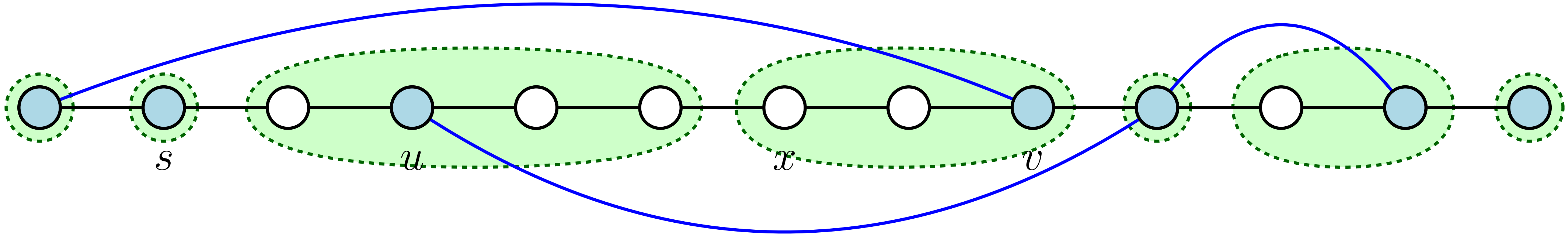
The shortest path to a non-terminal vertex x must pass through one of its two neighboring terminal vertices u, v

$$d_{P+S}(s,x) = \min \begin{cases} \alpha_u + d_P(u,x) \\ \alpha_v + d_P(v,x) \end{cases}$$

Assign each node of P to its "closest" terminal

Warm-up: Diameter of an Augmented Path $P + S$

What about the other vertices?



The shortest path to a non-terminal vertex x must pass through one of its two neighboring terminal vertices u, v

$$d_{P+S}(s, x) = \min \begin{cases} \alpha_u + d_P(u, x) \\ \alpha_v + d_P(v, x) \end{cases}$$

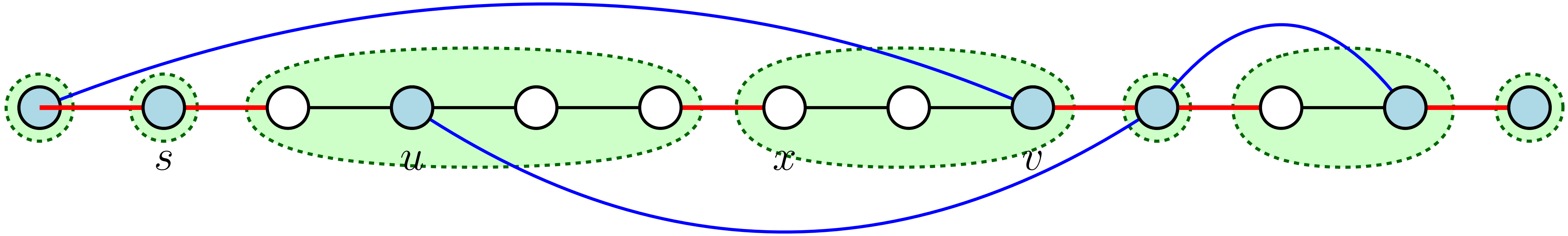
Assign each node of P to its "closest" terminal

For each terminal v, we are interested in the distance $\mathcal{E}_v$ from v to the farthest node assigned to v

The eccentricity of s is $\mathcal{E}(s) = \max_{\text{terminal } v} (\alpha_v + \mathcal{E}_v)$

Warm-up: Diameter of an Augmented Path $P + S$

What about the other vertices?



The shortest path to a non-terminal vertex x must pass through one of its two neighboring terminal vertices u, v

$$d_{P+S}(s, x) = \min \begin{cases} \alpha_u + d_P(u, x) \\ \alpha_v + d_P(v, x) \end{cases}$$

Assign each node of P to its "closest" terminal

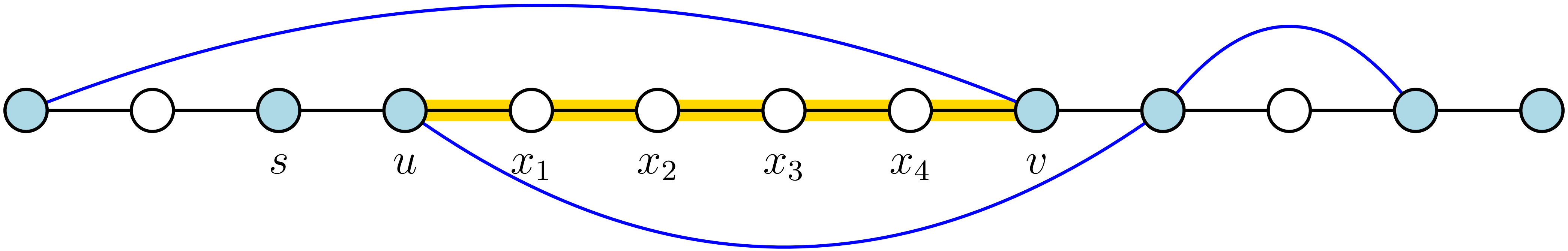
For each terminal v, we are interested in the distance $\mathcal{E}_v$ from v to the farthest node assigned to v

The eccentricity of s is $\mathcal{E}(s) = \max_{\text{terminal } v} (\alpha_v + \mathcal{E}_v)$

Observation: It suffices to **quickly** find the *boundary* edges

Warm-up: Diameter of an Augmented Path $P + S$

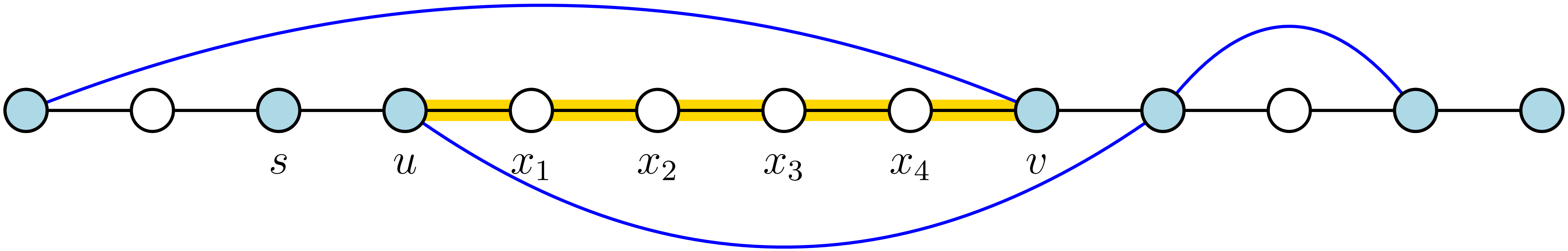
For each subpath between two consecutive terminal vertices u, v:



$$d_{P+S}(s, x_i) = \min \begin{cases} \alpha_u + d_P(u, x_i) \\ \alpha_v + d_P(v, x_i) \end{cases}$$

Warm-up: Diameter of an Augmented Path $P + S$

For each subpath between two consecutive terminal vertices u, v:



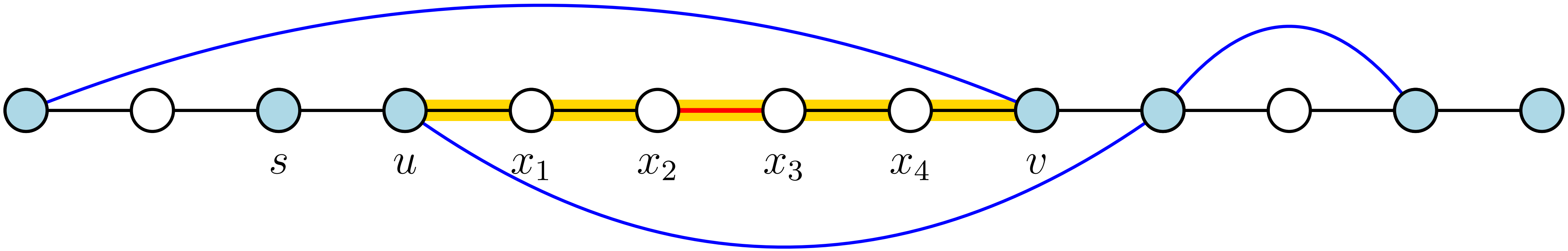
$$d_{P+S}(s, x_i) = \min \begin{cases} \alpha_u + d_P(u, x_i) \\ \alpha_v + d_P(v, x_i) \end{cases}$$

Monotonically non-decreasing w.r.t. i

Monotonically non-increasing w.r.t. i

Warm-up: Diameter of an Augmented Path $P + S$

For each subpath between two consecutive terminal vertices u, v:



$$d_{P+S}(s, x_i) = \min \begin{cases} \alpha_u + d_P(u, x_i) \\ \alpha_v + d_P(v, x_i) \end{cases}$$

Monotonically non-decreasing w.r.t. i

Monotonically non-increasing w.r.t. i

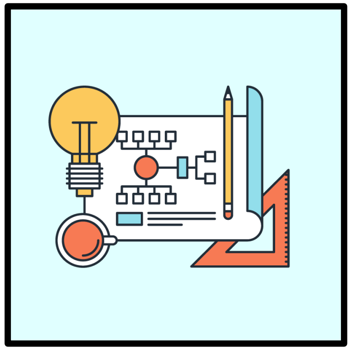
Find the cross-over point via binary search in time $O(\log n)$.

From Paths to Trees

- Mark terminals
- $H \leftarrow$ Compress graph
- Compute α_vs on H
- For each terminal v:
  - Find $\mathcal{E}_v$ via binary search
- Return $\max_v (\alpha_v + \mathcal{E}_v)$

From Paths to Trees

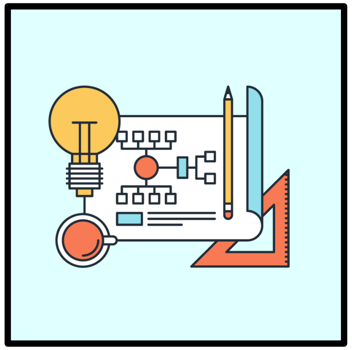
- Mark terminals
- $H \leftarrow$ Compress graph
- Compute α_vs on H
- For each terminal v:
  - Find $\mathcal{E}_v$ via binary search
- Return $\max_v (\alpha_v + \mathcal{E}_v)$



Auxiliar Data
Structure
DS

From Paths to Trees

- DS $\leftarrow$ `Build`(T)
- Mark terminals
- $H \leftarrow$ Compress graph
- Compute α_vs on H
- For each terminal v:
  - Find $\mathcal{E}_v$ via binary search
- Return $\max_v (\alpha_v + \mathcal{E}_v)$

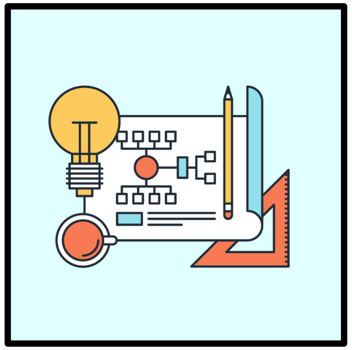


Auxiliar Data
Structure
DS

Build(T): Initializes the data structure on the tree T $O(n)$

From Paths to Trees

- DS $\leftarrow$ `Build`(T)
- `DS.MakeTerminal`(v_1) , `DS.MakeTerminal`(v_2), ...
- $H \leftarrow$ Compress graph
- Compute α_vs on H
- For each terminal v:
  - Find $\mathcal{E}_v$ via binary search
- Return $\max_v (\alpha_v + \mathcal{E}_v)$



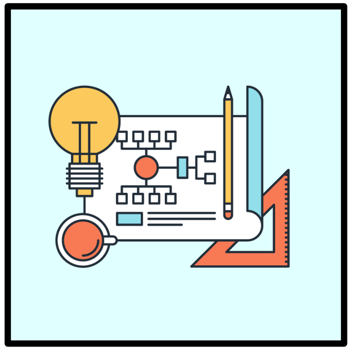
Auxiliar Data
Structure
DS

Build(T): Initializes the data structure on the tree T $O(n)$

MakeTerminal(v): Marks vertex v as a terminal vertex $O(\log n)$

From Paths to Trees

- DS $\leftarrow$ `Build`(T)
- `DS.MakeTerminal`(v_1) , `DS.MakeTerminal`(v_2), ...
- $H \leftarrow$ `DS.Shrink`$() + S$
- Compute α_vs on H
- For each terminal v:
  - Find $\mathcal{E}_v$ via binary search
- Return $\max_v (\alpha_v + \mathcal{E}_v)$



Auxiliar Data
Structure
DS

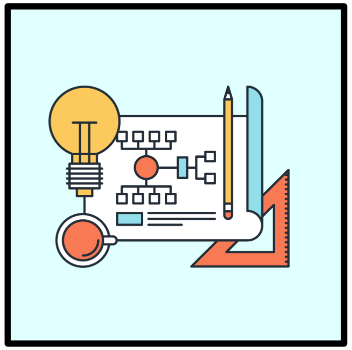
Build(T): Initializes the data structure on the tree T $O(n)$

MakeTerminal(v): Marks vertex v as a terminal vertex $O(\log n)$

Shrink$()$: Returns a compact representation of T that contains all terminals $O(\#$ terminals$)$

From Paths to Trees

- DS $\leftarrow$ `Build`(T)
- `DS.MakeTerminal`(v_1) , `DS.MakeTerminal`(v_2), ...
- $H \leftarrow$ `DS.Shrink`$() + S$
- Compute α_vs on H `DS.SetAlpha`(v_1, α_1), `DS.SetAlpha`(v_2, α_2), ...
- For each terminal v:
  - Find $\mathcal{E}_v$ via binary search
- Return $\max_v (\alpha_v + \mathcal{E}_v)$



Auxiliar Data
Structure
DS

Build(T): Initializes the data structure on the tree T $O(n)$

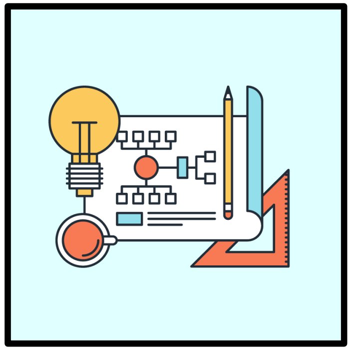
MakeTerminal(v): Marks vertex v as a terminal vertex $O(\log n)$

Shrink$()$: Returns a compact representation of T that contains all terminals $O(\#$ terminals$)$

SetAlpha(v, α_v): Assigns a weight $\alpha_v \geq 0$ to vertex v $O(1)$

From Paths to Trees

- DS $\leftarrow$ `Build`(T)
- `DS.MakeTerminal`(v_1) , `DS.MakeTerminal`$(v_2), \ldots$
- $H \leftarrow$ `DS.Shrink`$() + S$
- Compute α_vs on H `DS.SetAlpha`(v_1, α_1), `DS.SetAlpha`$(v_2, \alpha_2), \ldots$
- Return `DS.ReportFarthest`$()$

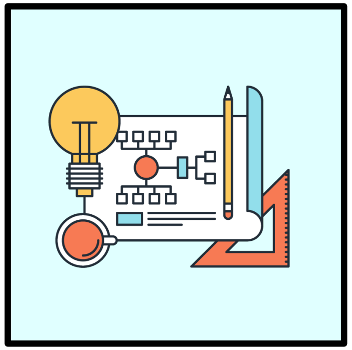


Auxiliar Data Structure DS

Build(T): Initializes the data structure on the tree T	$O(n)$
MakeTerminal(v): Marks vertex v as a terminal vertex	$O(\log n)$
Shrink$()$: Returns a compact representation of T that contains all terminals	$O(\#\text{ terminals})$
SetAlpha(v, α_v): Assigns a weight $\alpha_v \geq 0$ to vertex v	$O(1)$
ReportFarthest$()$: Return the vertex that is "farthest" from all terminals	$O(\#\text{ terminals} \cdot \log n)$

From Paths to Trees

- DS $\leftarrow$ `Build`(T)
- `DS.MakeTerminal`(v_1) , `DS.MakeTerminal`$(v_2), \ldots$
- $H \leftarrow$ `DS.Shrink`$() + S$
- Compute α_vs on H `DS.SetAlpha`(v_1, α_1), `DS.SetAlpha`$(v_2, \alpha_2), \ldots$
- Return `DS.ReportFarthest`$()$

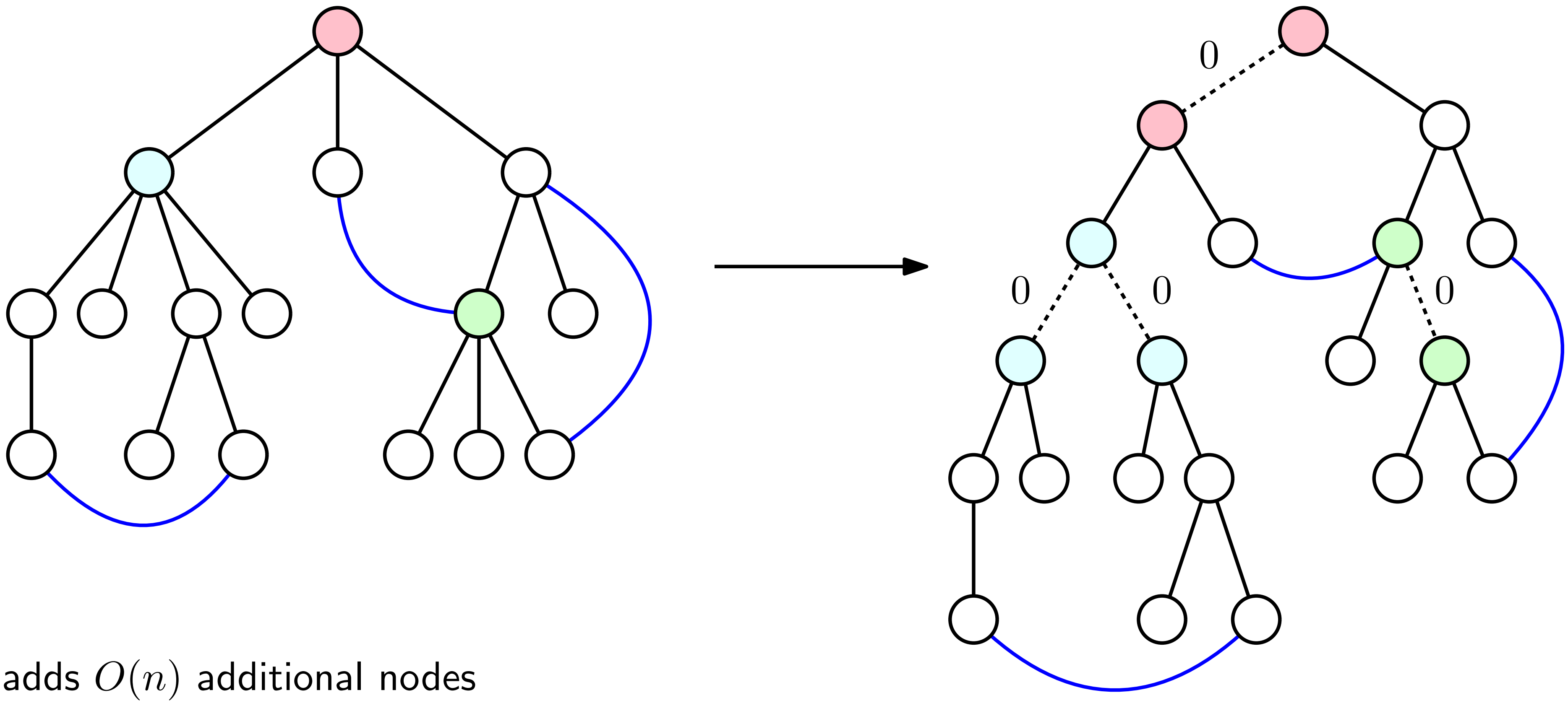


Auxiliar Data
Structure
DS

Build(T):	Initializes the data structure on the tree T	$O(n)$
MakeTerminal(v):	Marks vertex v as a terminal vertex	$O(\log n)$
Shrink$()$:	Returns a compact representation of T that contains all terminals	$O(\#$ terminals$)$
SetAlpha(v, α_v):	Assigns a weight $\alpha_v \geq 0$ to vertex v	$O(1)$
ReportFarthest$()$:	Return the vertex that is "farthest" from all terminals	$O(\#$ terminals $\cdot \log n)$

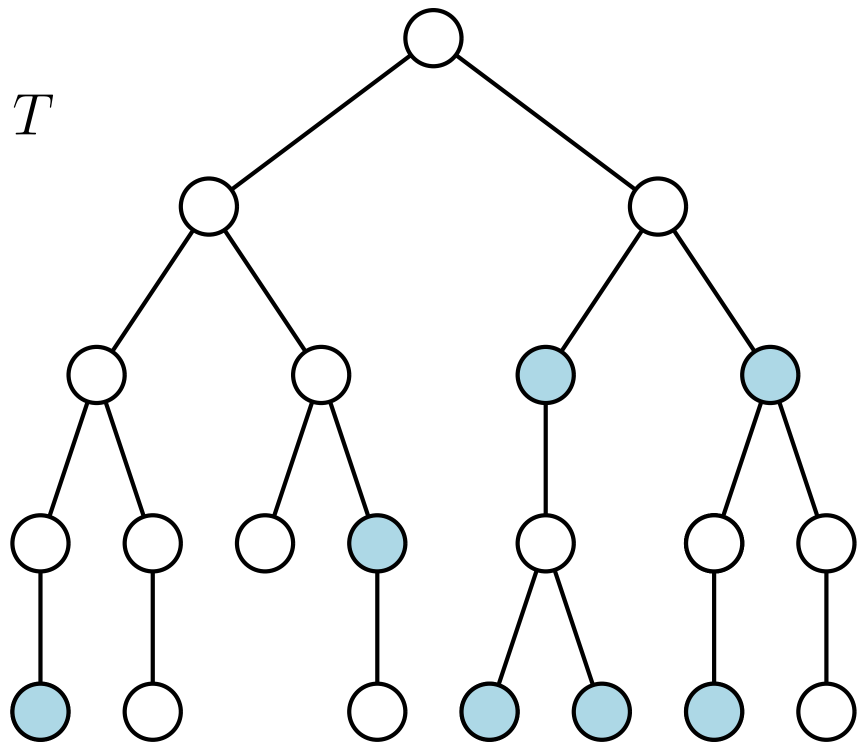
A simplifying assumption

We can assume that T is a binary tree



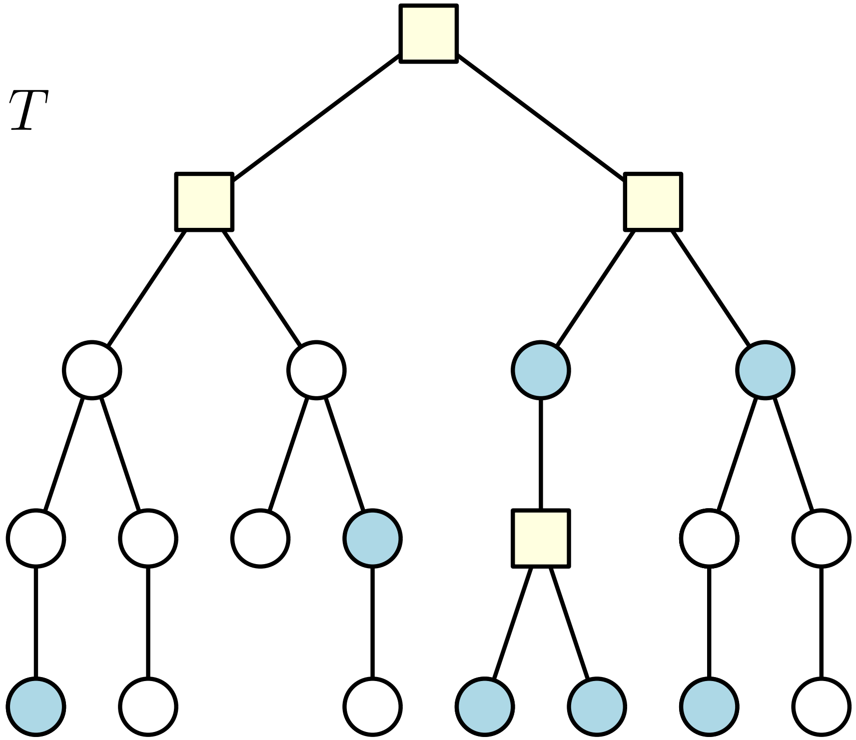
This adds $O(n)$ additional nodes

Shrink()



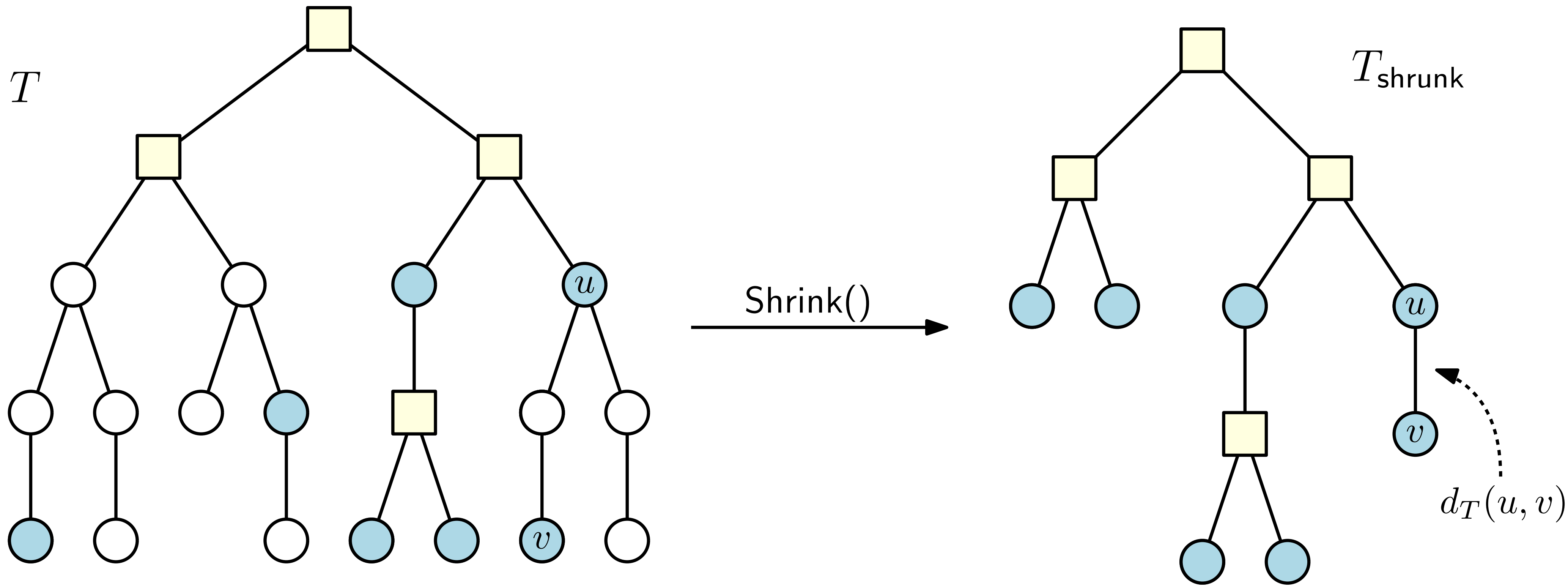
The vertex set V' of the shrunk tree are all the terminals, plus all the LCAs between pairs of terminals

Shrink()



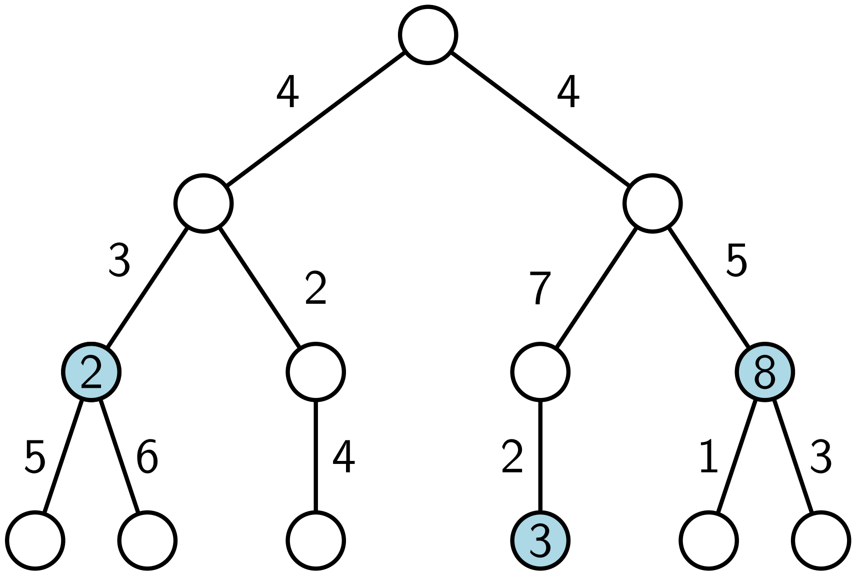
The vertex set V' of the shrunk tree are all the terminals, plus all the LCAs between pairs of terminals

Shrink()



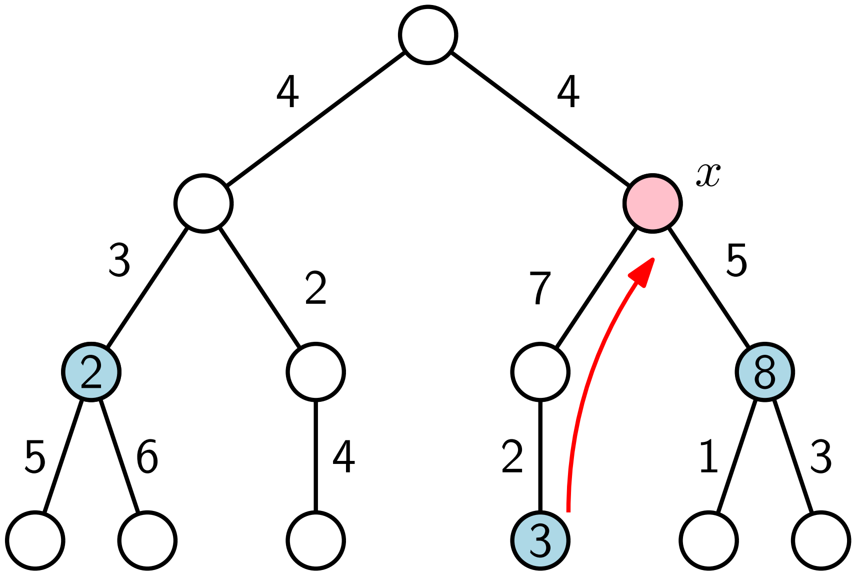
The vertex set V' of the shrunk tree are all the terminals, plus all the LCAs between pairs of terminals

ReportFarthest()



Define the "distance" of a vertex x as $\delta(x) = \min_{\text{terminal } v} \left(\alpha_v + d_T(v,x)\right)$

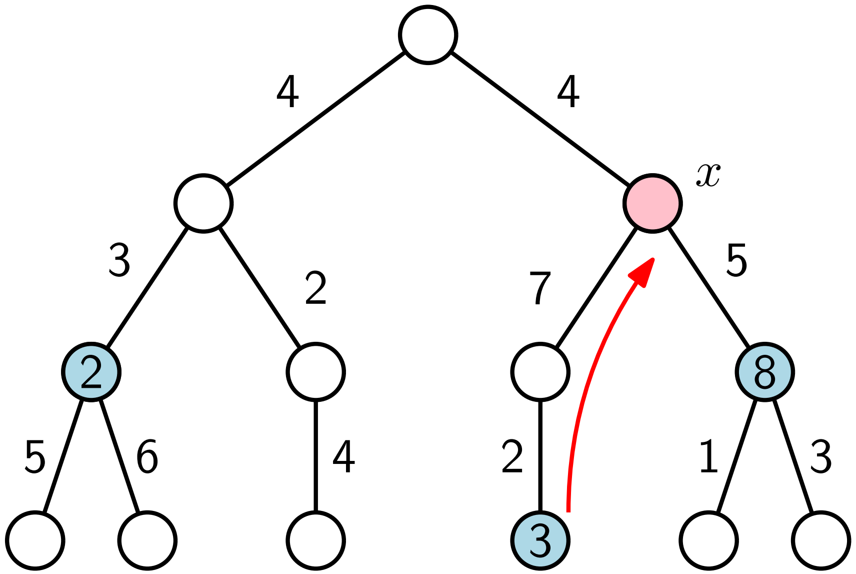
ReportFarthest()



$\delta(x) = 12$

Define the "distance" of a vertex x as $\delta(x) = \min_{\text{terminal } v} \left(\alpha_v + d_T(v, x)\right)$

ReportFarthest()

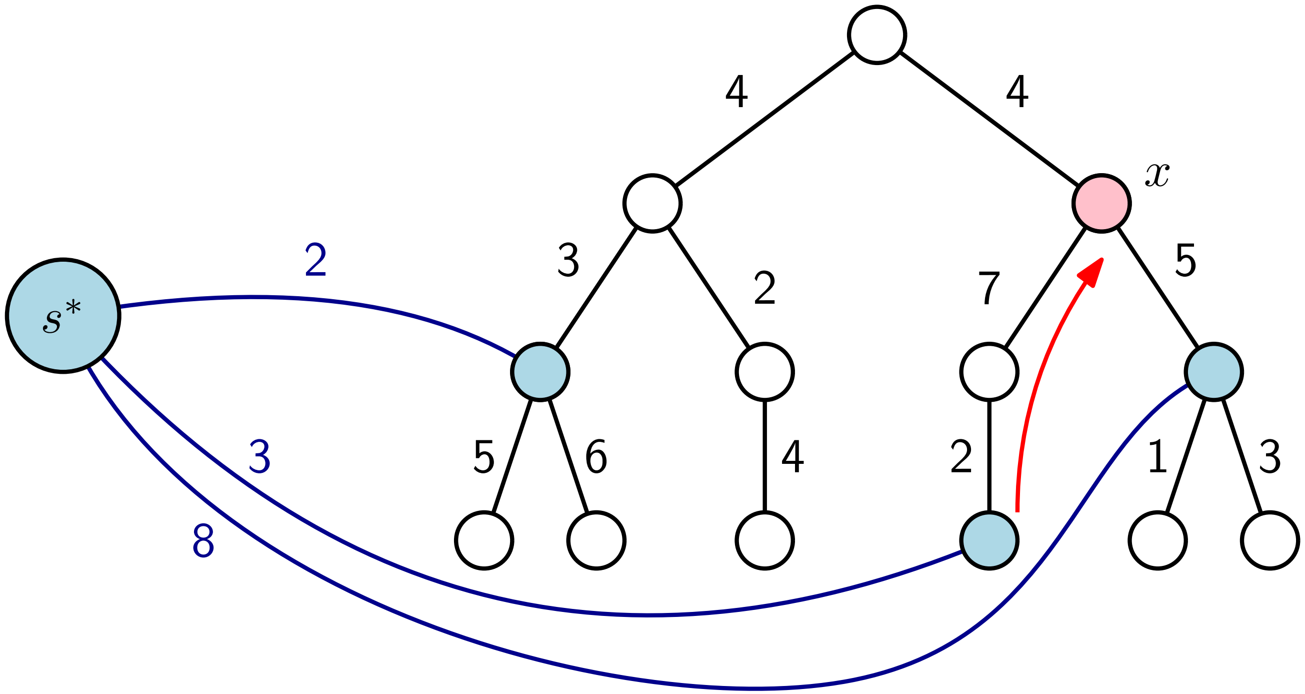


$\delta(x) = 12$

Define the "distance" of a vertex x as $\delta(x) = \min_{\text{terminal } v} \big(\alpha_v + d_T(v, x)\big)$

ReportFarthest() returns a vertex x that maximizes $\delta(x)$

ReportFarthest()



$\delta(x) = 12$ = Eccentricity of s^*

Define the "distance" of a vertex x as $\delta(x) = \min_{\text{terminal } v} \left(\alpha_v + d_T(v, x)\right)$

ReportFarthest() returns a vertex x that maximizes $\delta(x)$

Our Data Structure: Implementation

- The tree T is stored using a *top-tree* $\mathcal{T}$ time per op: $O(\log \#\text{vertices})$
  - Can add (link) and remove (cut) edges
  - Can mark/unmark vertices as terminals
  - Given a vertex v, it reports the closest ancestor of v that is a terminal
  - Given v, can report the eccentricity of v w.r.t. its tree

Our Data Structure: Implementation

- The tree T is stored using a *top-tree* $\mathcal{T}$ time per op: $O(\log \#\text{vertices})$
  - Can add (link) and remove (cut) edges
  - Can mark/unmark vertices as terminals
  - Given a vertex v, it reports the closest ancestor of v that is a terminal
  - Given v, can report the eccentricity of v w.r.t. its tree

- The shrunk tree T_{shrunk} is stored using a *link-cut tree*
  - Vertex additions and deletions
  - Link/cut operations

time per op: $O(\log \#\text{vertices})$

Our Data Structure: Implementation

- The tree T is stored using a *top-tree* $\mathcal{T}$ — time per op: $O(\log \#\text{vertices})$
  - Can add (link) and remove (cut) edges
  - Can mark/unmark vertices as terminals
  - Given a vertex v, it reports the closest ancestor of v that is a terminal
  - Given v, can report the eccentricity of v w.r.t. its tree

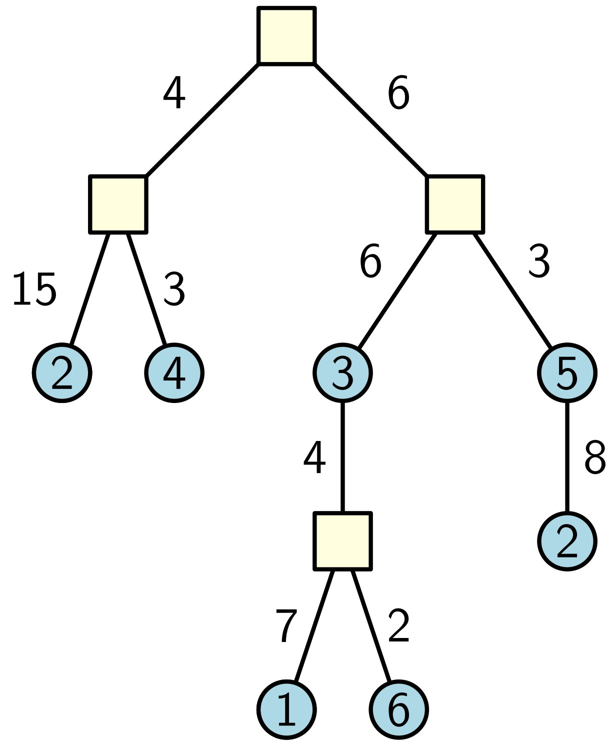
- The shrunk tree T_{shrunk} is stored using a *link-cut tree*
  - Vertex additions and deletions
  - Link/cut operations

  time per op: $O(\log \#\text{vertices})$

- Oracles with linear size that can report: — time per op: $O(1)$
  - The lowerst common ancestor of a pair of vertices in T
  - The level ancestor of a vertex in T
  - The distance/hop-disance between a pair of vertices in T

Implementing ReportFarthest()

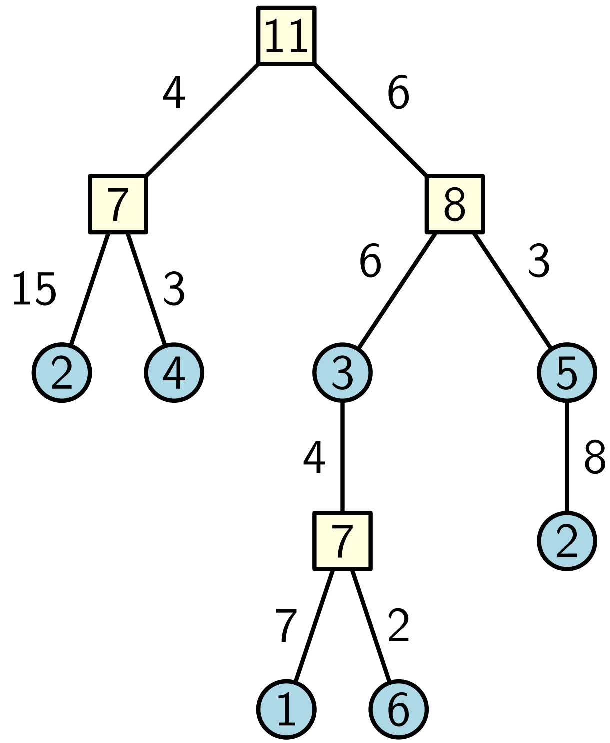
Compute the distance $\beta_v = \delta(v)$ to each vertex v in T_{shrunk}



Can be done in time $O(k)$ using a postorder DFS vits followed by a preoder DFS visit

Implementing ReportFarthest()

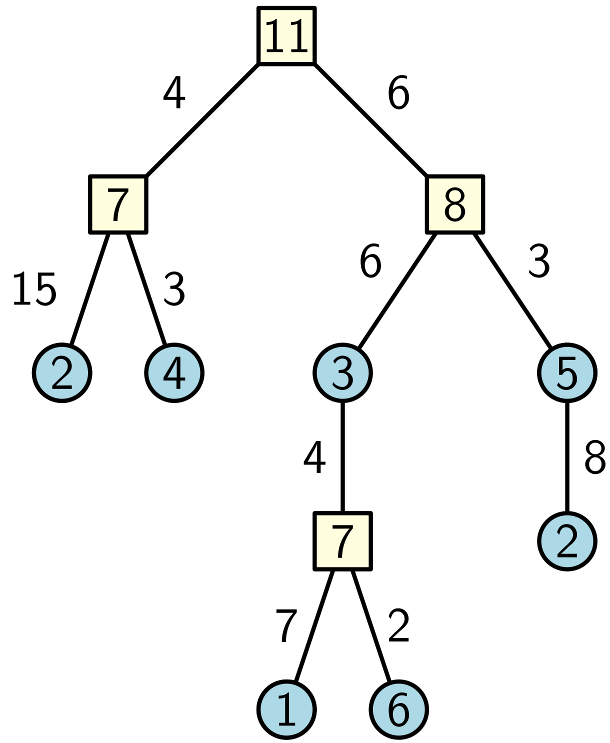
Compute the distance $\beta_v = \delta(v)$ to each vertex v in T_{shrunk}



Can be done in time $O(k)$ using a postorder DFS vits followed by a preoder DFS visit

Implementing ReportFarthest()

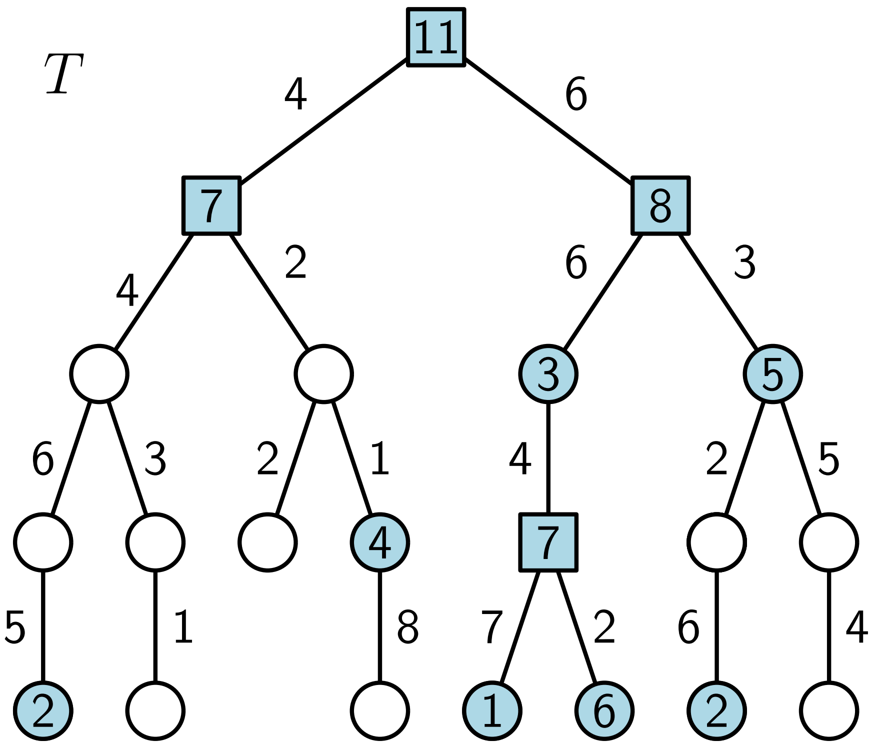
Compute the distance $\beta_v = \delta(v)$ to each vertex v in T_{shrunk}



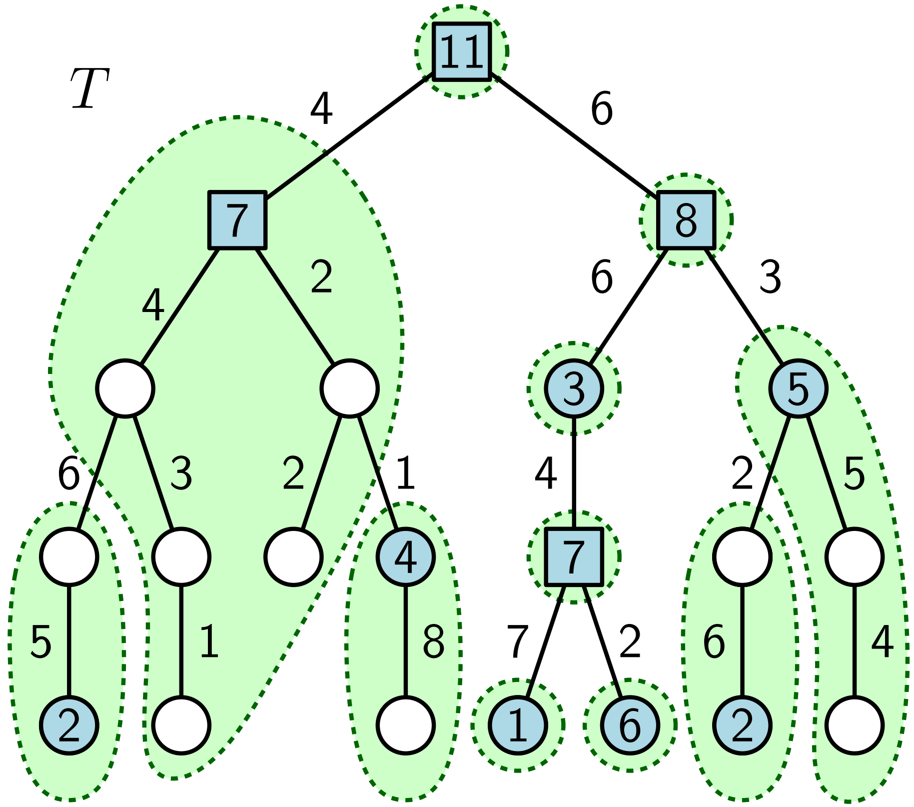
Can be done in time $O(k)$ using a postorder DFS vits followed by a preoder DFS visit

This allows us to treat all vertices in T_{shrunk} as if they were terminals

Implementing ReportFarthest()

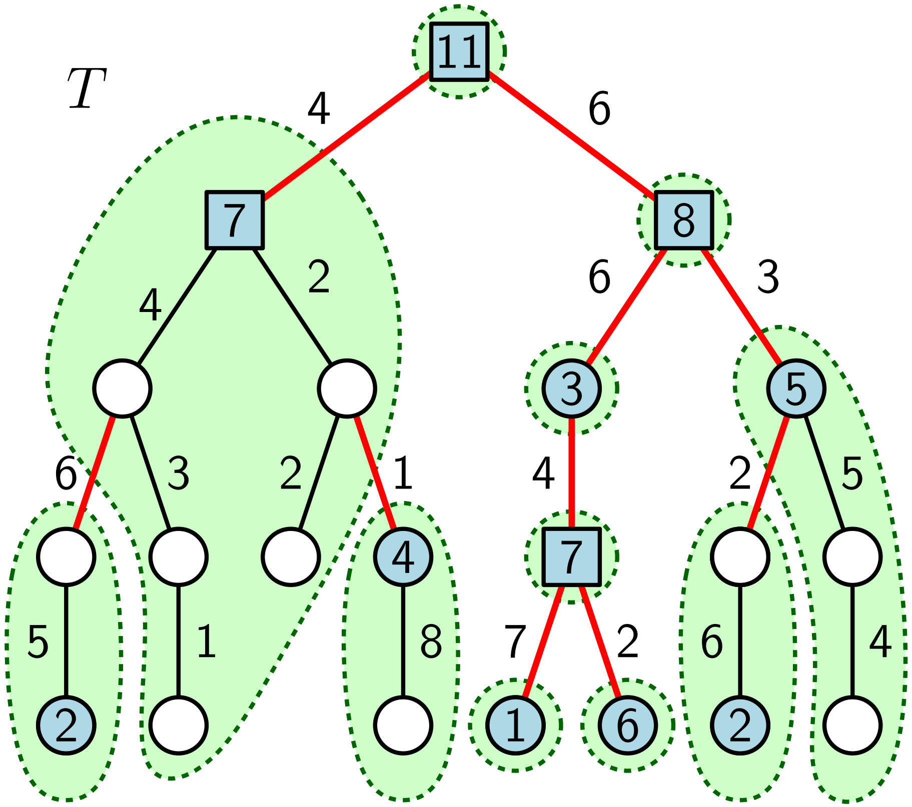


Implementing ReportFarthest()



Assign each node of T to the "closest" node of T_{shrunk}

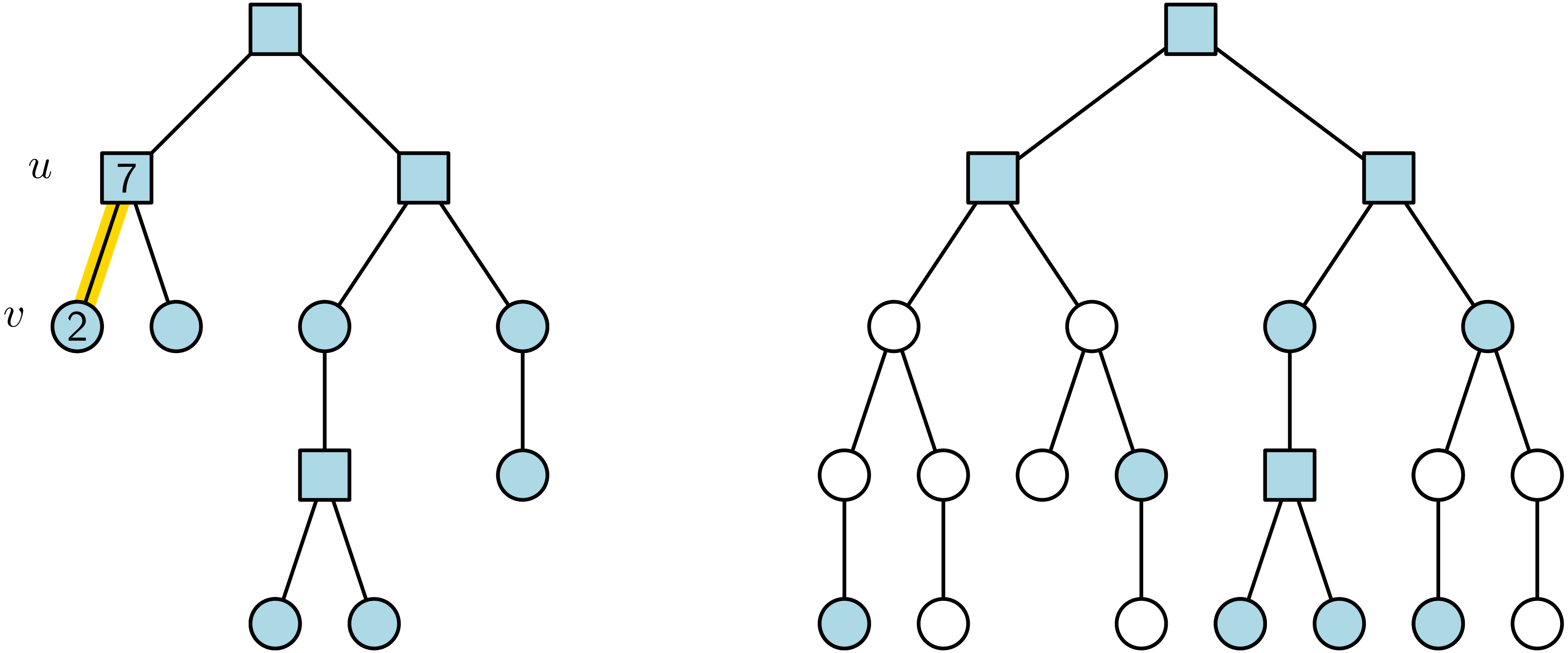
Implementing ReportFarthest()



Assign each node of T to the "closest" node of T_{shrunk}

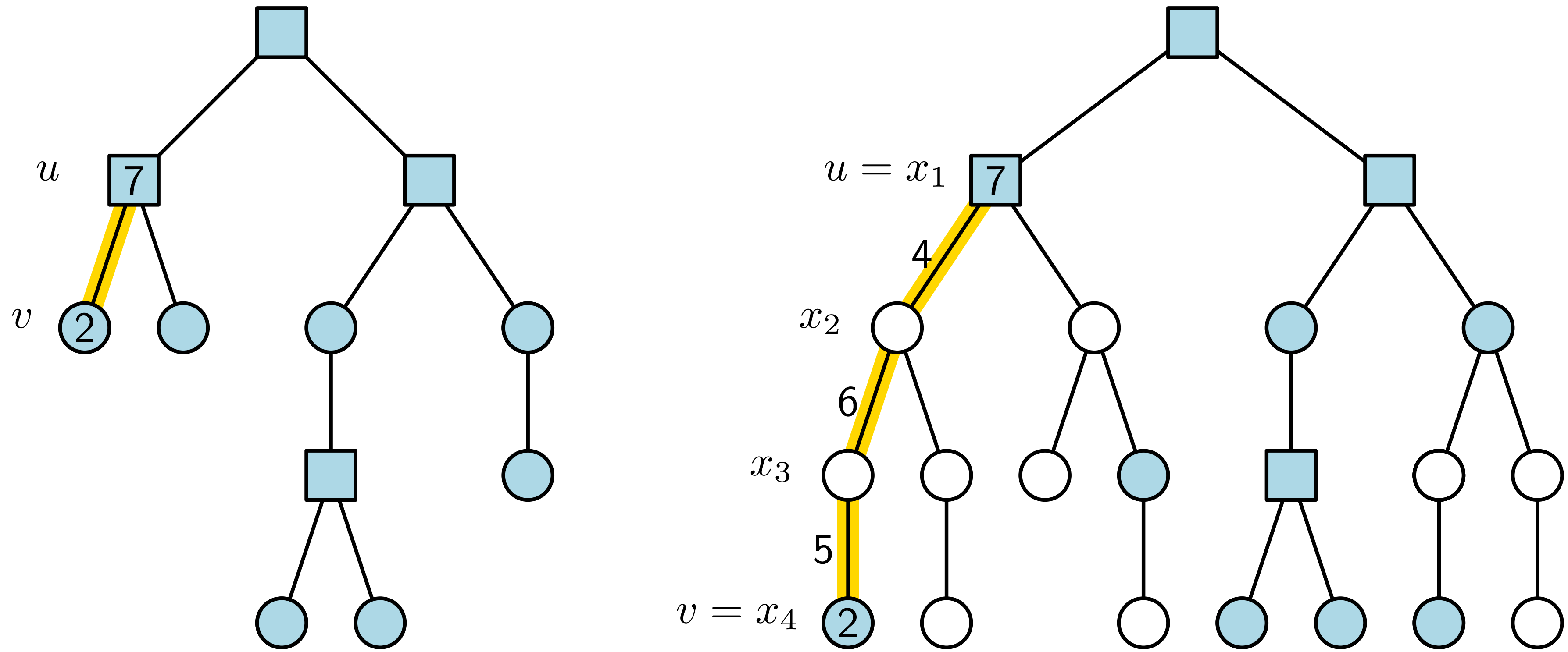
We want to **quickly** find the boundary edges

Implementing ReportFarthest()



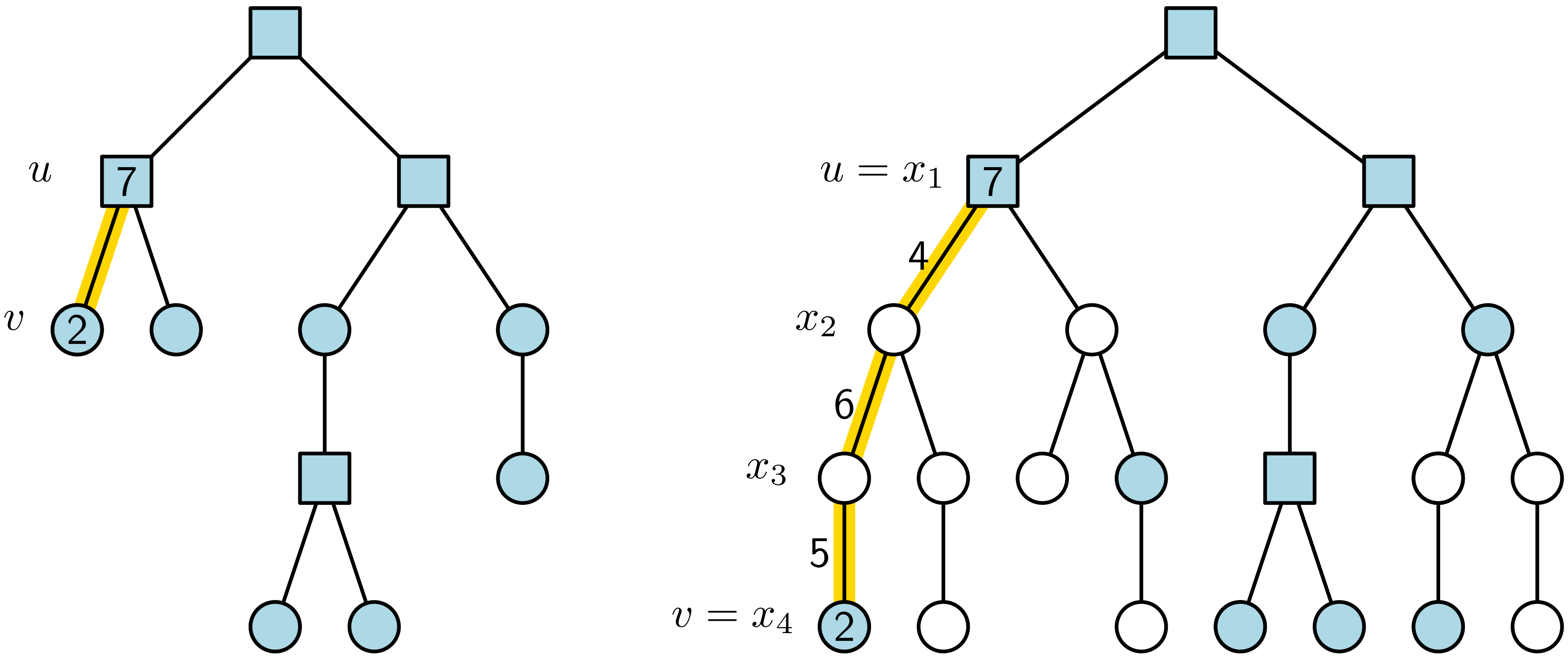
Each edge (u, v) in T_{shrunk} corresponds to a vertical path $\langle u = x_1, x_2, \ldots, x_k = v \rangle$ in T

Implementing ReportFarthest()



Each edge (u, v) in T_{shrunk} corresponds to a vertical path $\langle u = x_1, x_2, \ldots, x_k = v \rangle$ in T

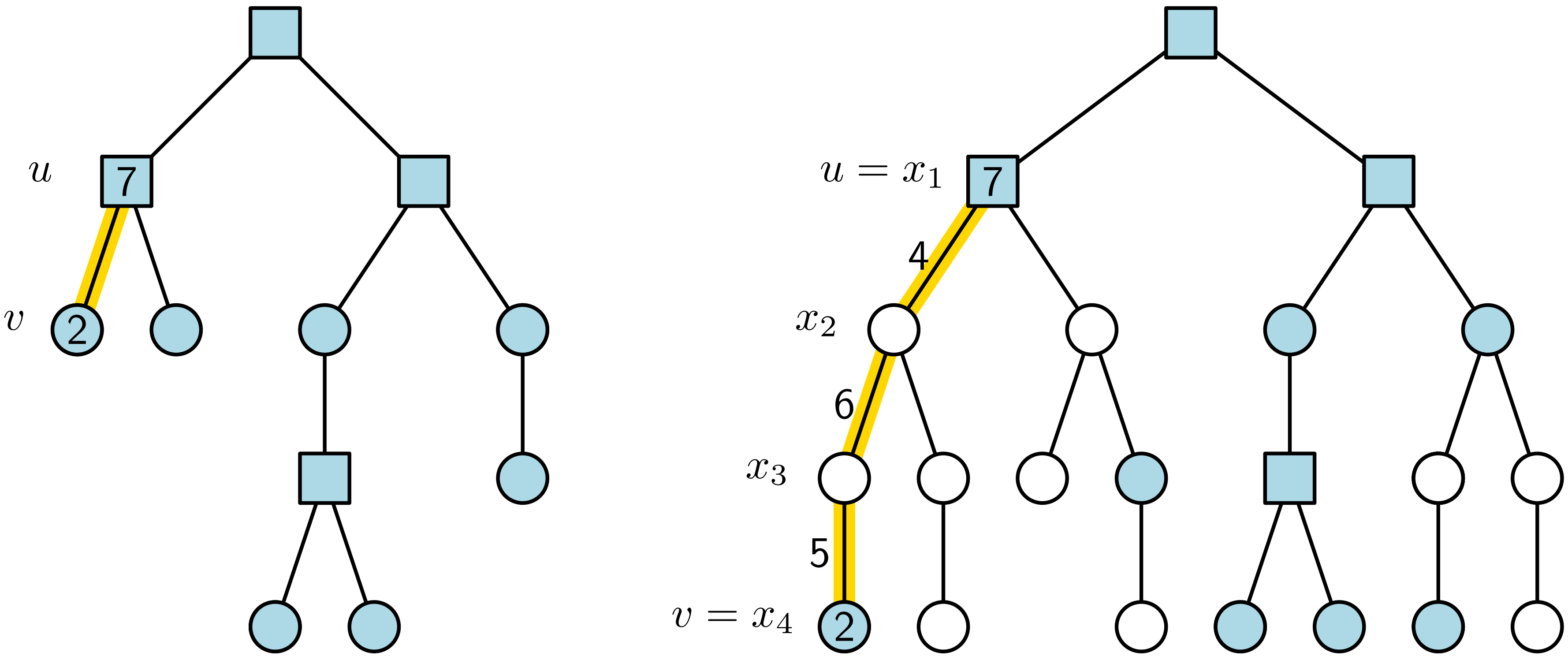
Implementing ReportFarthest()



Each edge (u, v) in T_{shrunk} corresponds to a vertical path $\langle u = x_1, x_2, \ldots, x_k = v \rangle$ in T

$$\delta(x_i) = \min \begin{cases} \beta_u + d_T(u, x_i) \\ \beta_v + d_T(v, x_i) \end{cases}$$

Implementing ReportFarthest()



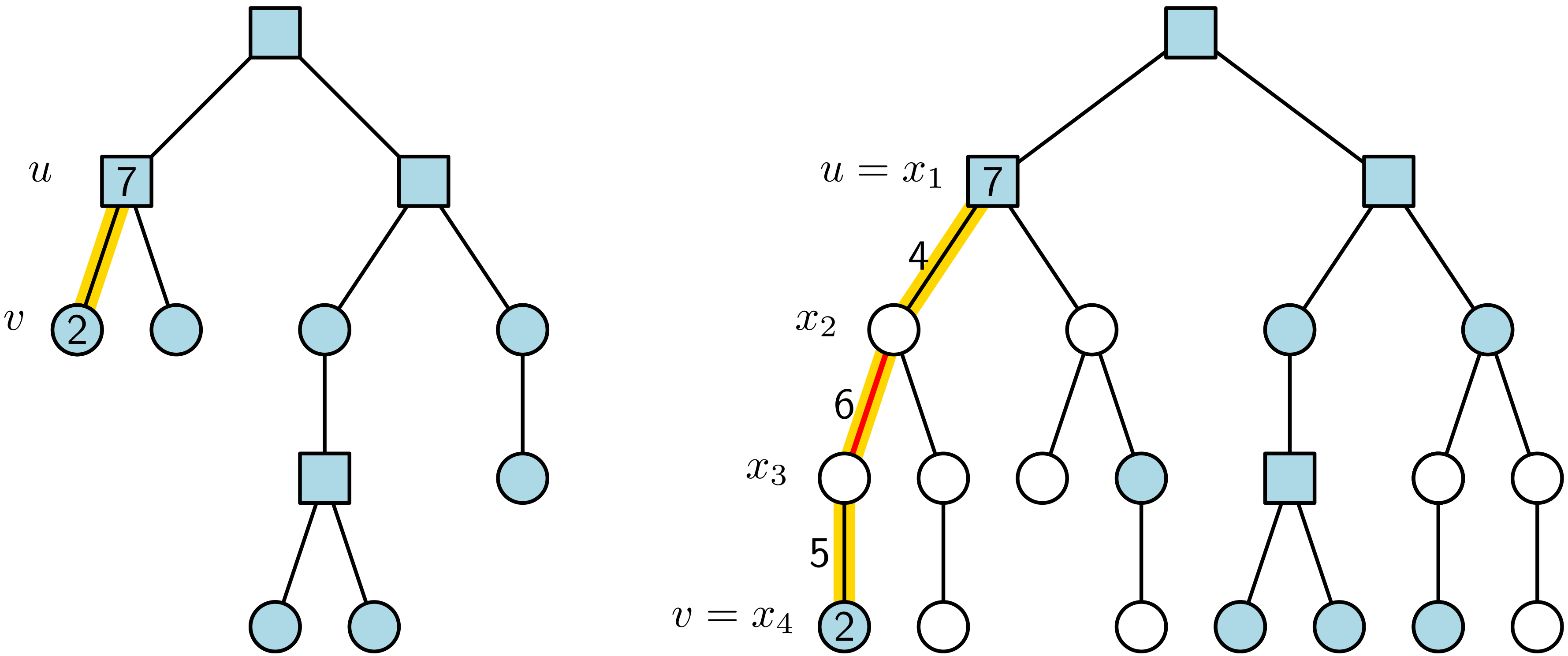
Each edge (u, v) in T_{shrunk} corresponds to a vertical path $\langle u = x_1, x_2, \ldots, x_k = v \rangle$ in T

$$\delta(x_i) = \min \begin{cases} \beta_u + d_T(u, x_i) \\ \beta_v + d_T(v, x_i) \end{cases}$$

Monotonically non-decreasing w.r.t. i

Monotonically non-increasing w.r.t. i

Implementing ReportFarthest()



Each edge (u, v) in T_{shrunk} corresponds to a vertical path $\langle u = x_1, x_2, \ldots, x_k = v \rangle$ in T

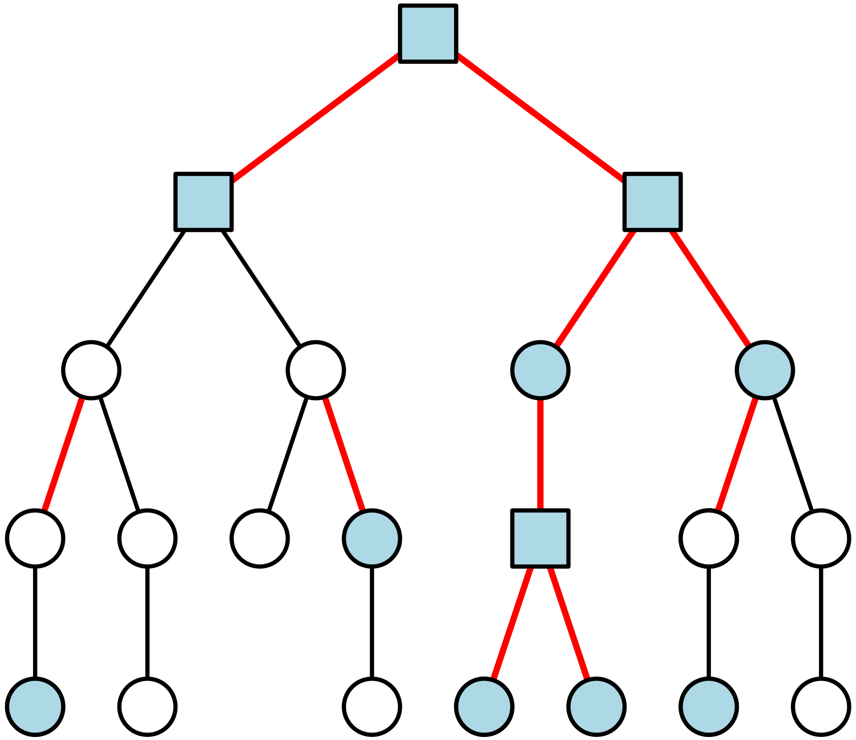
$$\delta(x_i) = \min \begin{cases} \beta_u + d_T(u, x_i) \\ \beta_v + d_T(v, x_i) \end{cases}$$

Monotonically non-decreasing w.r.t. i

Monotonically non-increasing w.r.t. i

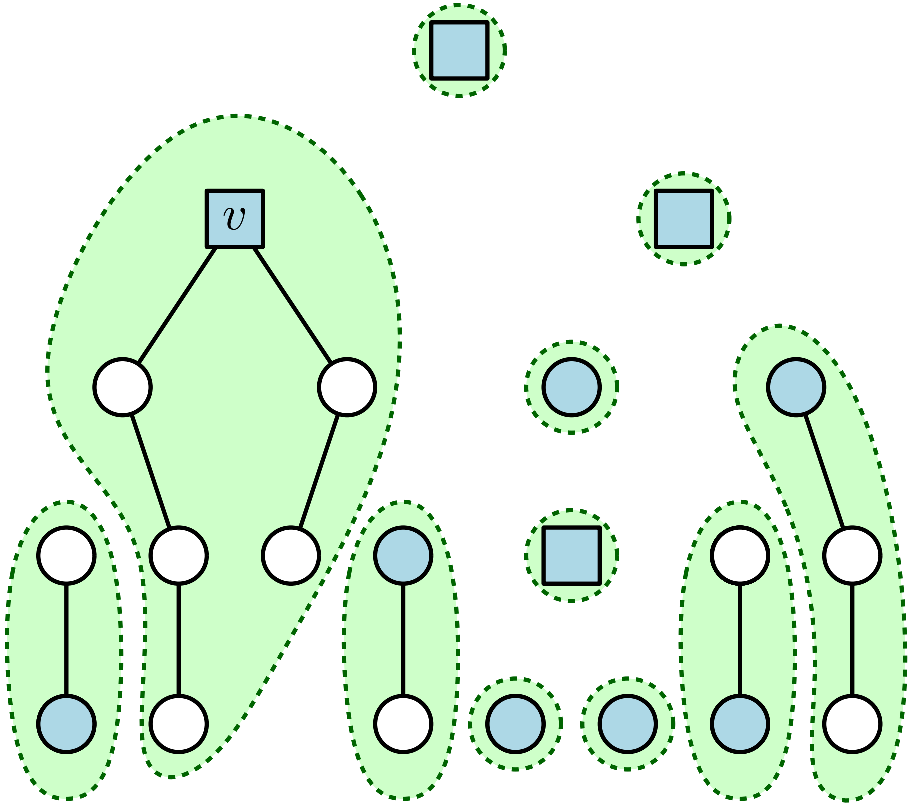
Binary search!

Implementing ReportFarthest()



Cut all bounday edges from $\mathcal{T}$

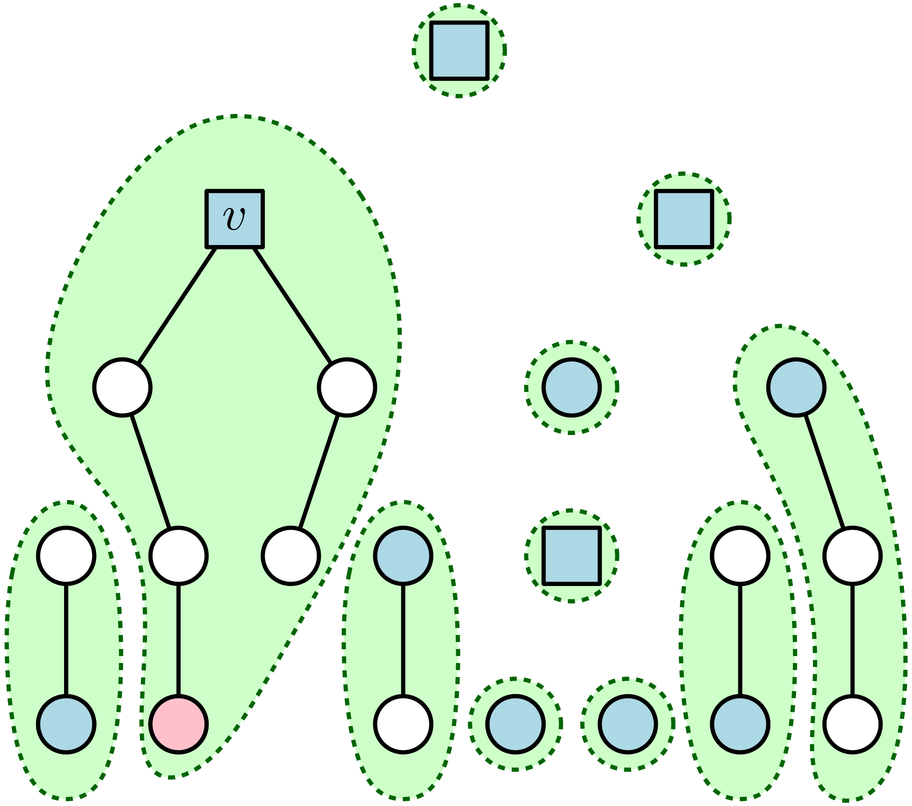
Implementing ReportFarthest()



Cut all bounday edges from $\mathcal{T}$

The resulting forest contains exactly one tree T_v for each vertex v in $T_{\textsf{shrunk}}$

Implementing ReportFarthest()

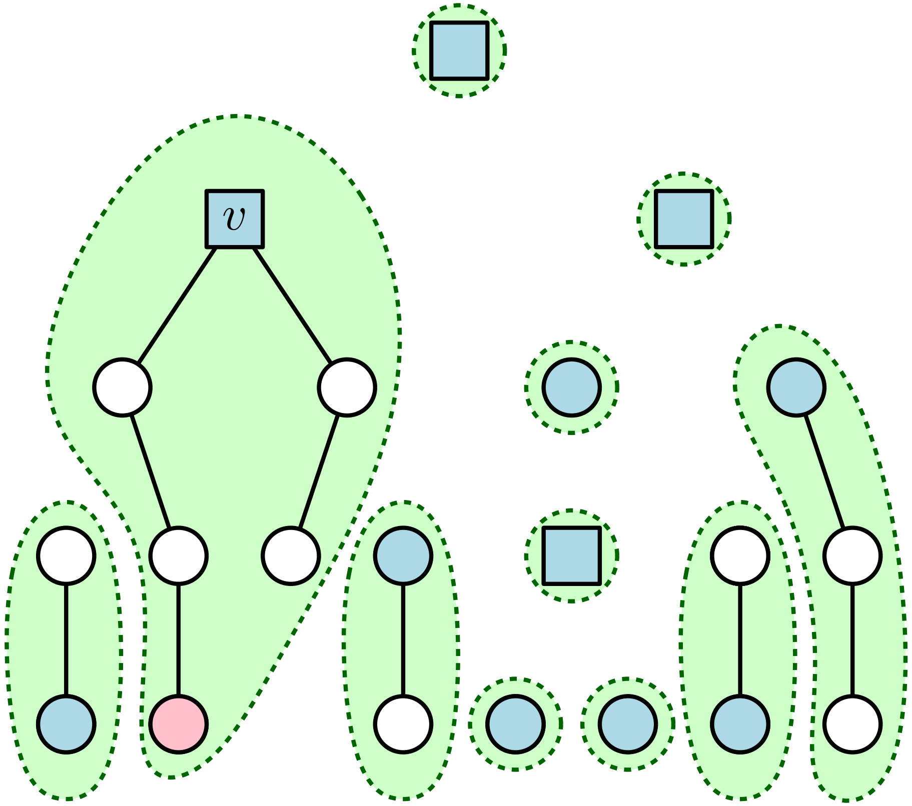


Cut all bounday edges from $\mathcal{T}$

The resulting forest contains exactly one tree T_v for each vertex v in T_{shrunk}

For each v: query $\mathcal{T}$ to find the eccentricity $\mathcal{E}_v$ of v in T_v

Implementing ReportFarthest()



Cut all bounday edges from $\mathcal{T}$

The resulting forest contains exactly one tree T_v for each vertex v in T_{shrunk}

For each v: query $\mathcal{T}$ to find the eccentricity $\mathcal{E}_v$ of v in T_v

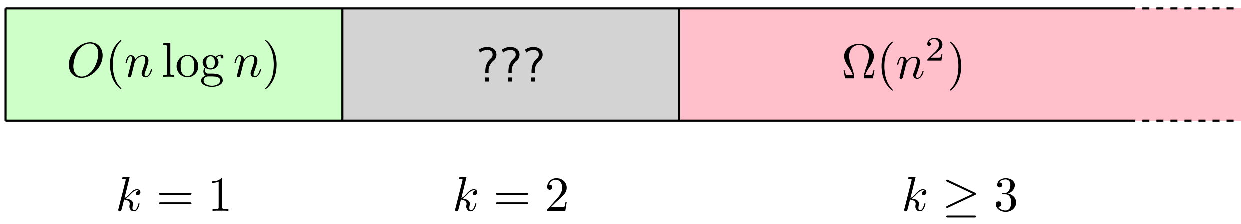
Return $\max_v (\beta_v + \mathcal{E}_v)$ ...and restore the orignal state of $\mathcal{T}$ (**link** the boundary edges)

Open Problems

Faster algorithms for <u>metric</u> k-DOAT?

- Avoid trying all possible shortcuts

Lower bound on the number of queries needed to solve metric 2-DOAT?



Open Problems

Faster algorithms for metric k-DOAT?

- Avoid trying all possible shortcuts

Lower bound on the number of queries needed to solve metric 2-DOAT?

$O(n \log n)$	???	$\Omega(n^2)$
$k = 1$	$k = 2$	$k \geq 3$

Thank you!

Questions?