A Simple Algorithm for Minimum Cuts in Near-Linear Time

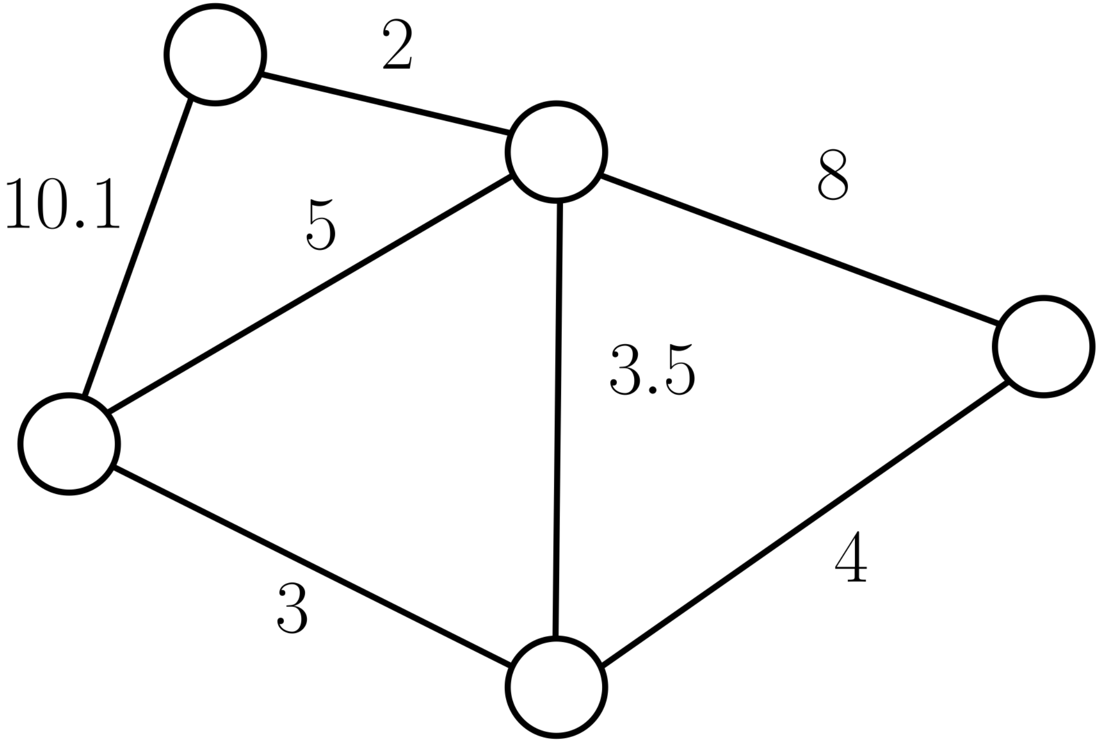
Nalin Bhardwaj – University of California San Diego

Antonio Molina Lovett – Princeton University

Bryce Sandlund – University of Waterloo

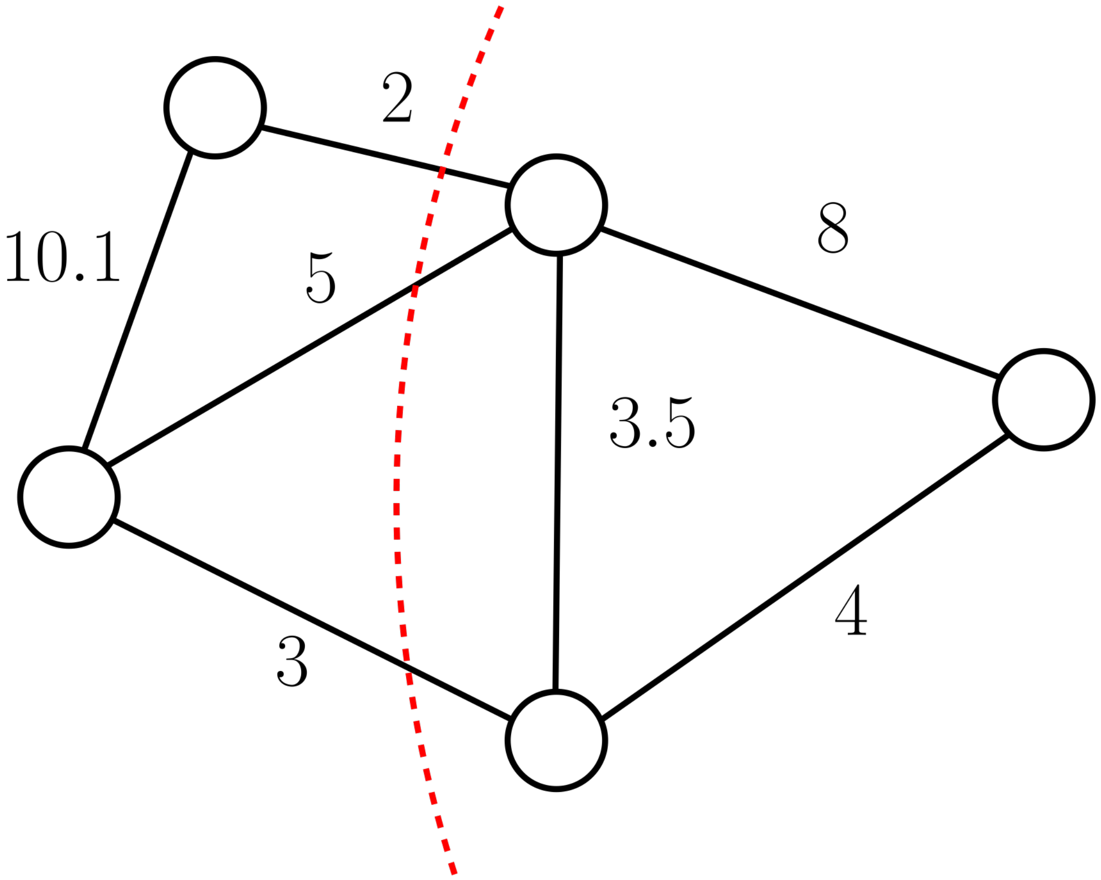
The Weighted Minimum Cut Problem

Given a graph G on vertex set V, determine a nonempty vertex subset $S \subseteq V$ such that the total weight of edges from S to $V \setminus S$ is minimized.



The Weighted Minimum Cut Problem

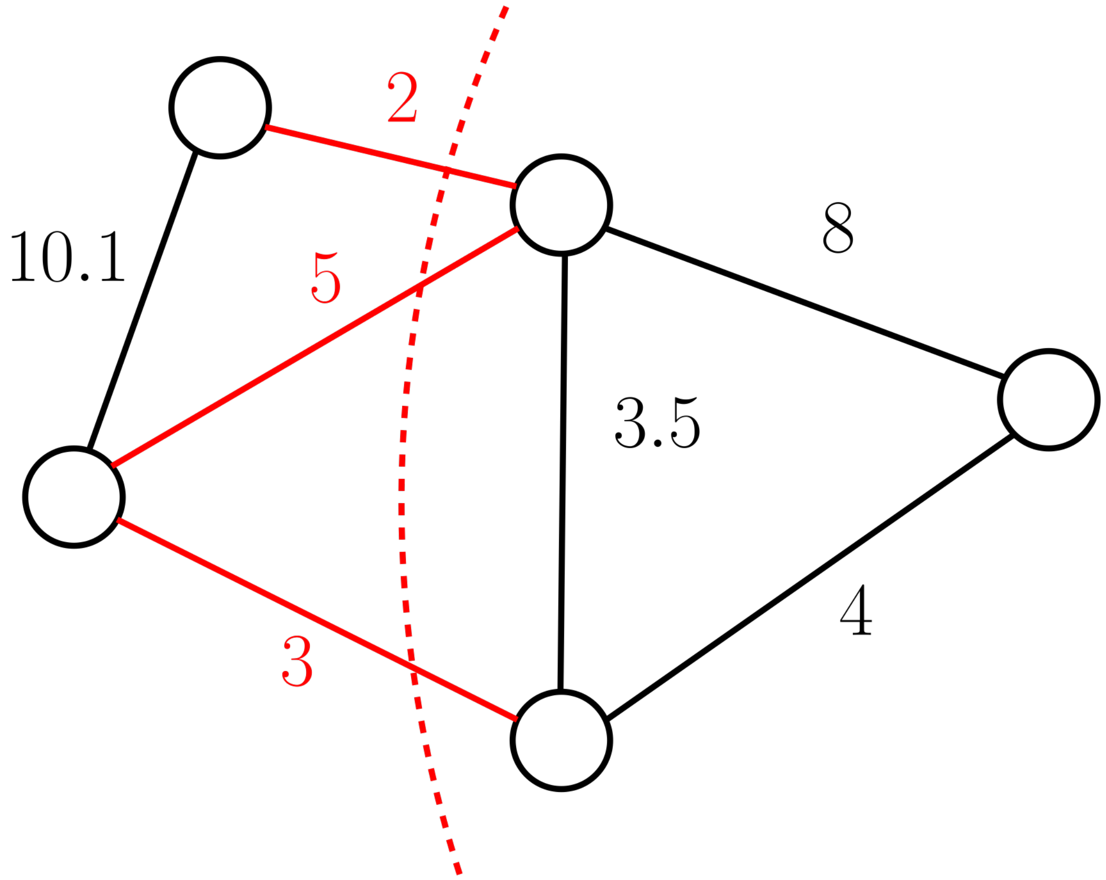
Given a graph G on vertex set V, determine a nonempty vertex subset $S \subseteq V$ such that the total weight of edges from S to $V \setminus S$ is minimized.



The Weighted Minimum Cut Problem

Given a graph G on vertex set V, determine a nonempty vertex subset $S \subseteq V$ such that the total weight of edges from S to $V \setminus S$ is minimized.

The weight of the minimum cut is the total weight of the edges crossing the cut.



History

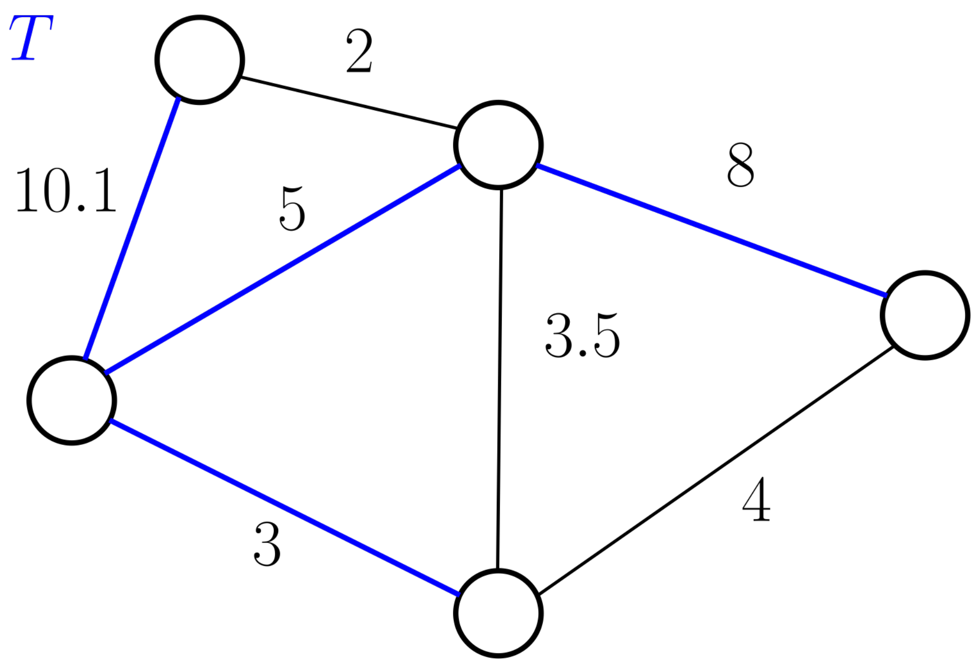
Previous Work	Result	Citation
$\binom{n}{2}$ flow computations	$O(n^2) \cdot flow =$ about $O(n^5)$ with push-relabel.	Naïve Attempt
Gomory-Hu tree	$O(n) \cdot flow =$ about $O(n^4)$ with push-relabel.	Gomory & Hu, SIAM J. Appl. Math ‘61
A faster algorithm for finding a minimum cut in a directed graph	$O(flow) =$ about $O(n^3)$ with push-relabel.	Hao & Orlin, J. Algorithms ‘94
Stoer-Wagner	Compute *arbitrary* min s-t cut. Contract s and t. Repeat. $O(nm + \mathrm{n}^2 \log n)$ with Fibonacci heap, $O(nm \log n)$ with a binary heap.	Stoer & Wagner, J. ACM ‘97
Karger’s randomized contraction algorithm	Pick a random edge and contract it. Repeat. $O(n^2 m \log n)$.	Karger, SODA ‘93
Karger-Stein algorithm	Branch Karger after $\frac{n}{\sqrt{2}}$ contractions. $O(n^2 \log^3 n)$.	Karger & Stein, J. ACM ‘96
Minimum cuts in near-linear time	Sample edges, pack trees, find minimum cuts that cut ≤ 2 tree edges. $O(m \log^3 n)$.	Karger, J. ACM ‘00

Flow

Edge Contraction

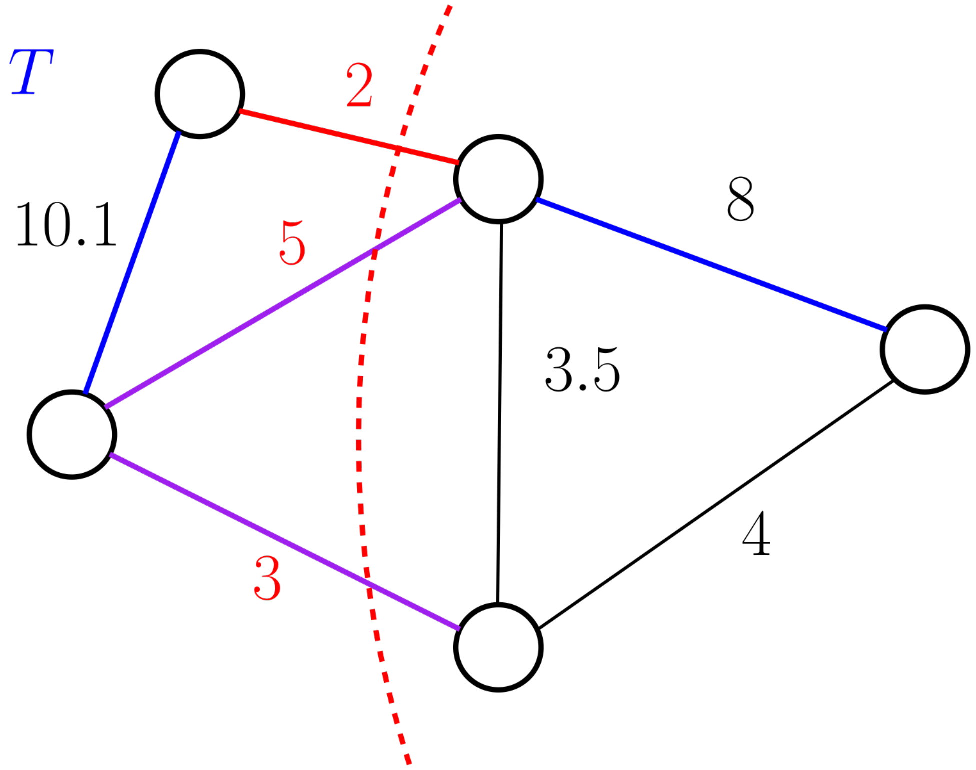
Our Results

1. A simplification of Karger's algorithm to find a smallest cut of G that 2-respects (cuts ≤ 2 edges of) a spanning tree T of G.



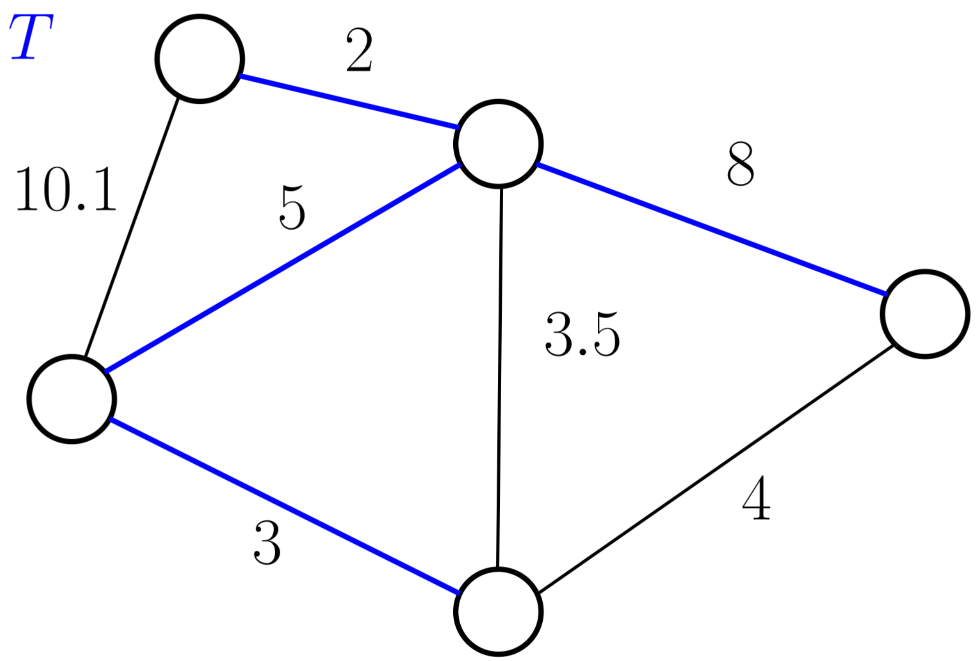
Our Results

1. A simplification of Karger's algorithm to find a smallest cut of G that 2-respects (cuts ≤ 2 edges of) a spanning tree T of G.



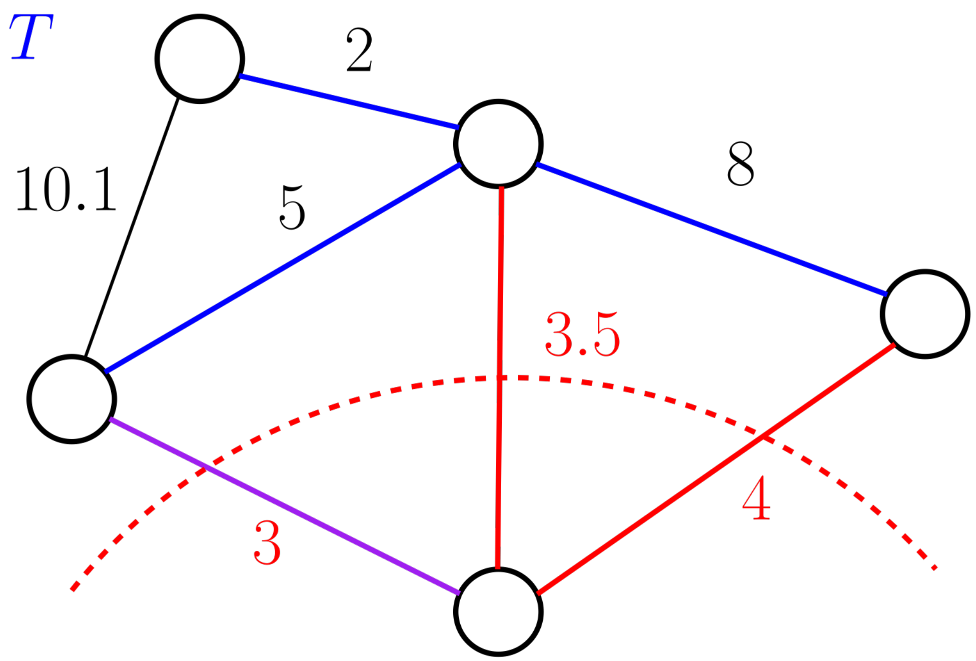
Our Results

1. A simplification of Karger's algorithm to find a smallest cut of G that 2-respects (cuts ≤ 2 edges of) a spanning tree T of G.



Our Results

1. A simplification of Karger's algorithm to find a smallest cut of G that 2-respects (cuts ≤ 2 edges of) a spanning tree T of G.



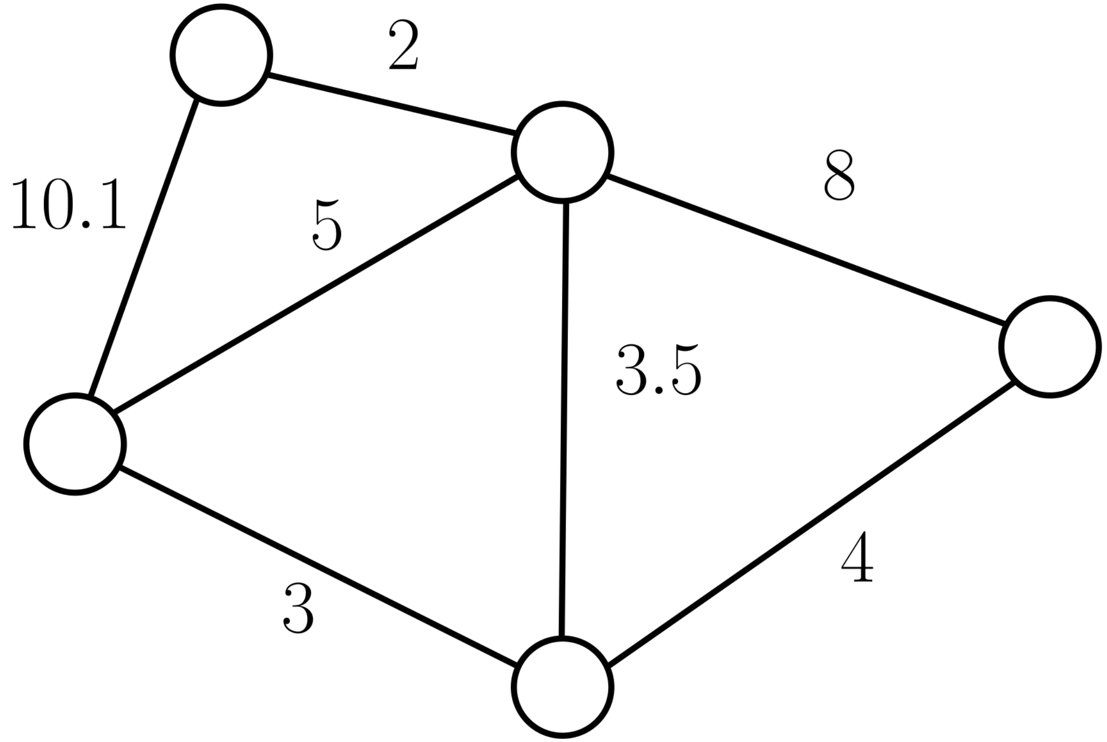
Our Results

1. A simplification of Karger's algorithm to find a smallest cut of G that 2-respects (cuts $\leq$ 2 edges of) a spanning tree T of G.
2. A self-contained version of Karger's algorithm.
3. An implementation of our version of Karger's algorithm.

Very Recent Work	Result	Citation
Minimum cut in $O(m \log^2 n)$ time	Improvement of 2-respect algorithm. $O(m \log^2 n)$.	Gawrychowski et al., ICALP '20
Weighted min-cut: sequential, cut-query and streaming algorithms	Improvement of 2-respect algorithm. $O\left(m \frac{\log^2 n}{\log \log n} + n \log^6 n\right)$.	Mukhopadhyay & Nanongkai, STOC '20

Karger's Near-Linear Time Algorithm

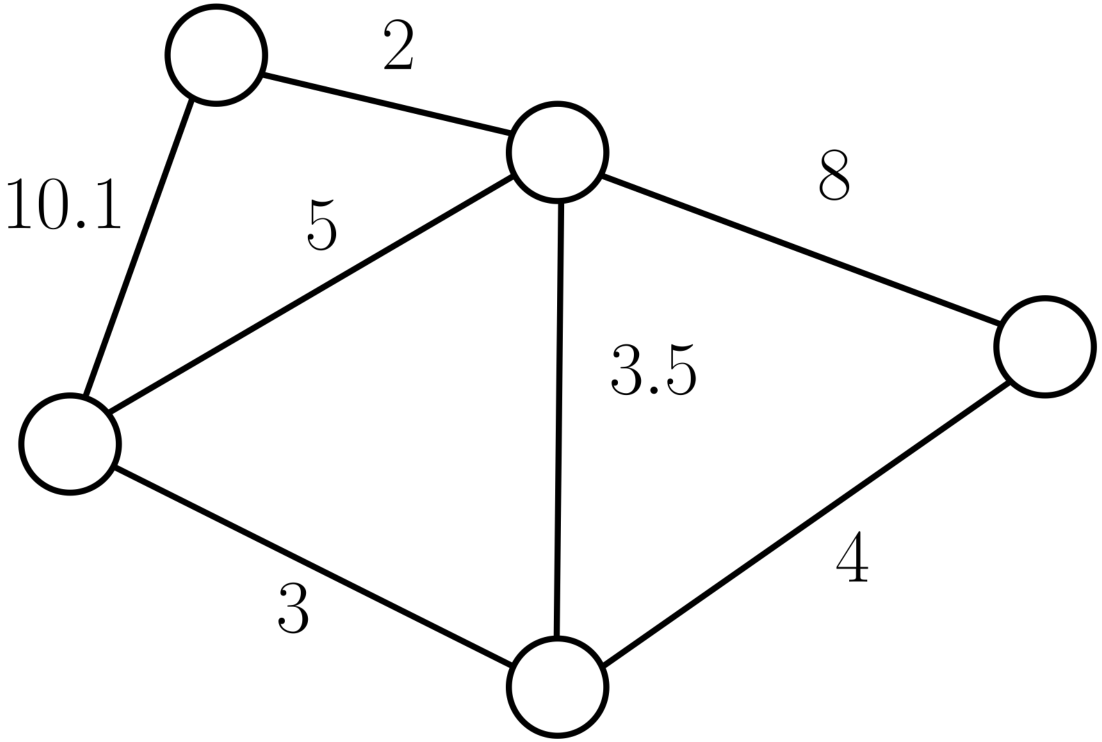
1. Sample edges of G.
2. Pack trees in the sampled graph.
3. Sample trees from the packing.
4. For each sampled tree T, determine a smallest cut of G that cuts at most two edges of T.



Tree Packing

Definition: A set of spanning trees, each with an assigned weight, so that the total weight of trees containing a given edge is no greater than the weight of that edge.

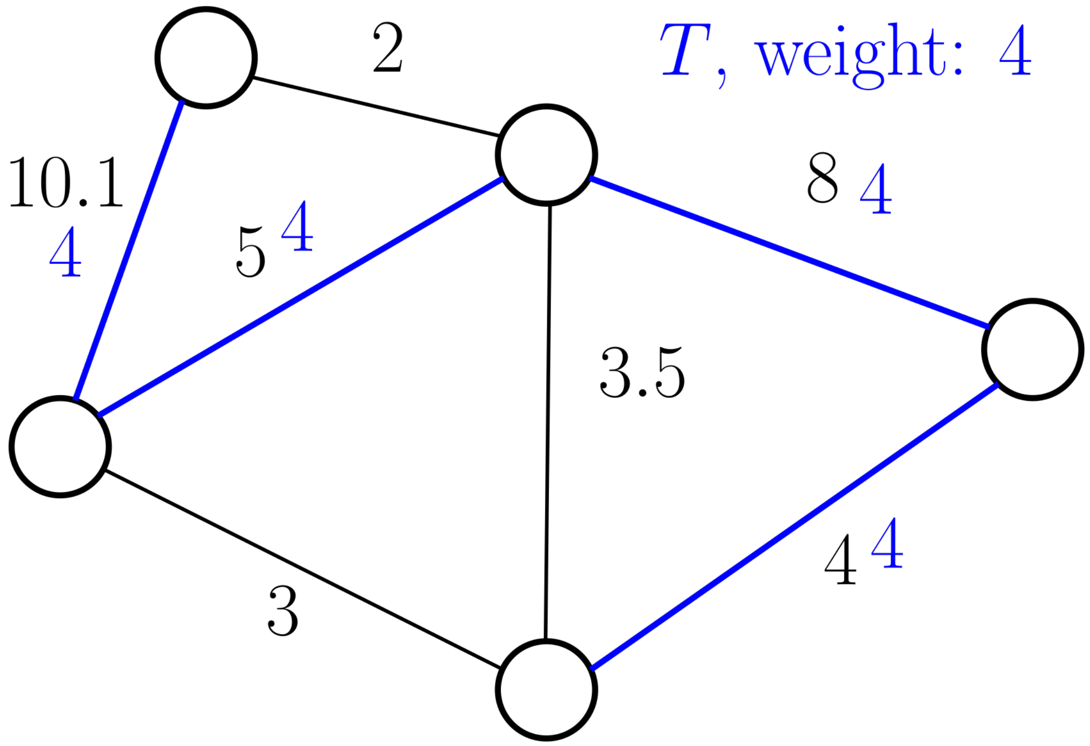
The weight of the packing is the total weight of the trees in it.



Tree Packing

Definition: A set of spanning trees, each with an assigned weight, so that the total weight of trees containing a given edge is no greater than the weight of that edge.

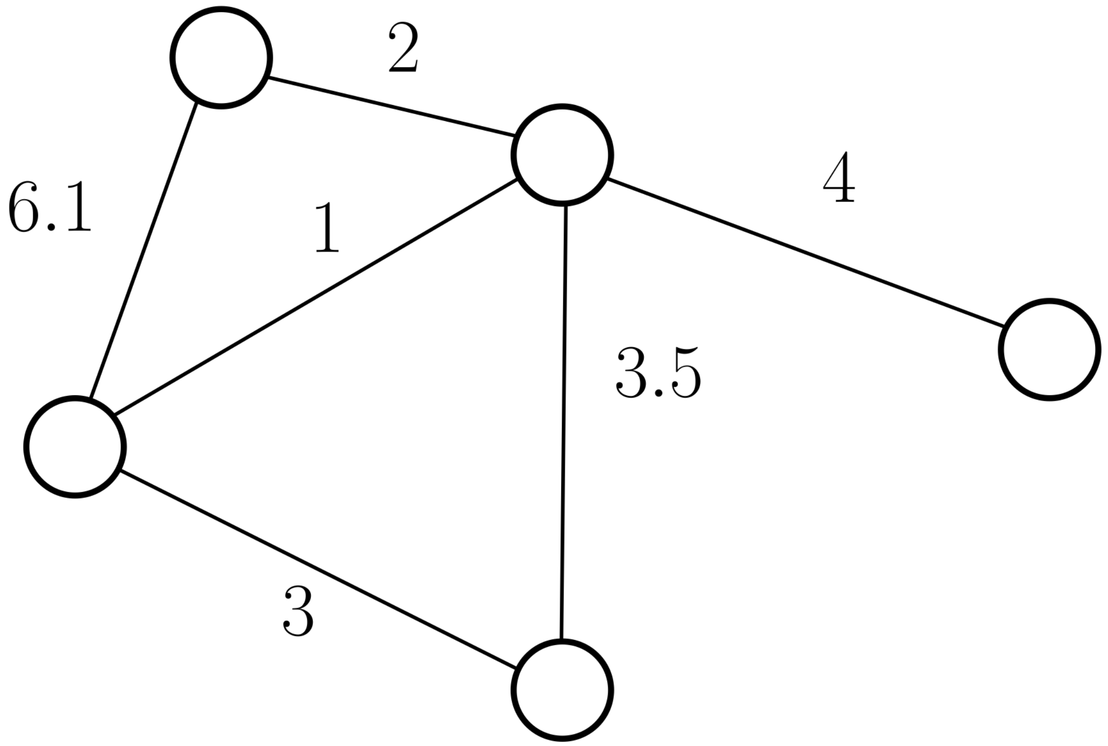
The weight of the packing is the total weight of the trees in it.



Tree Packing

Definition: A set of spanning trees, each with an assigned weight, so that the total weight of trees containing a given edge is no greater than the weight of that edge.

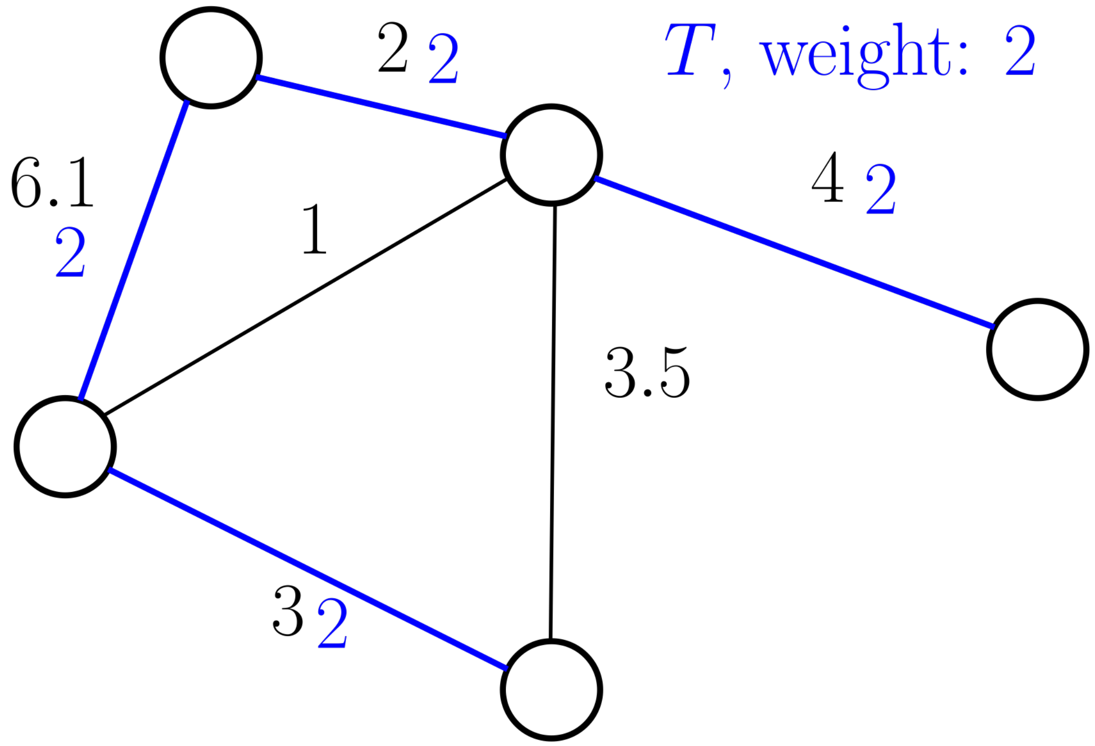
The weight of the packing is the total weight of the trees in it.



Tree Packing

Definition: A set of spanning trees, each with an assigned weight, so that the total weight of trees containing a given edge is no greater than the weight of that edge.

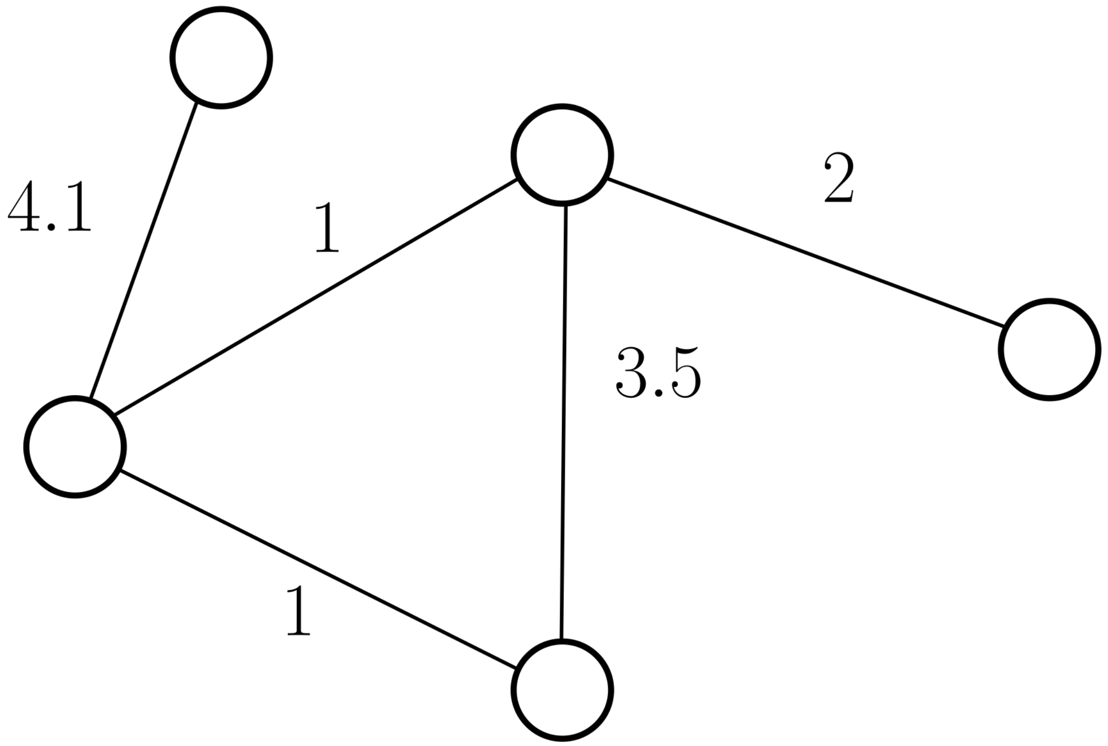
The weight of the packing is the total weight of the trees in it.



Tree Packing

Definition: A set of spanning trees, each with an assigned weight, so that the total weight of trees containing a given edge is no greater than the weight of that edge.

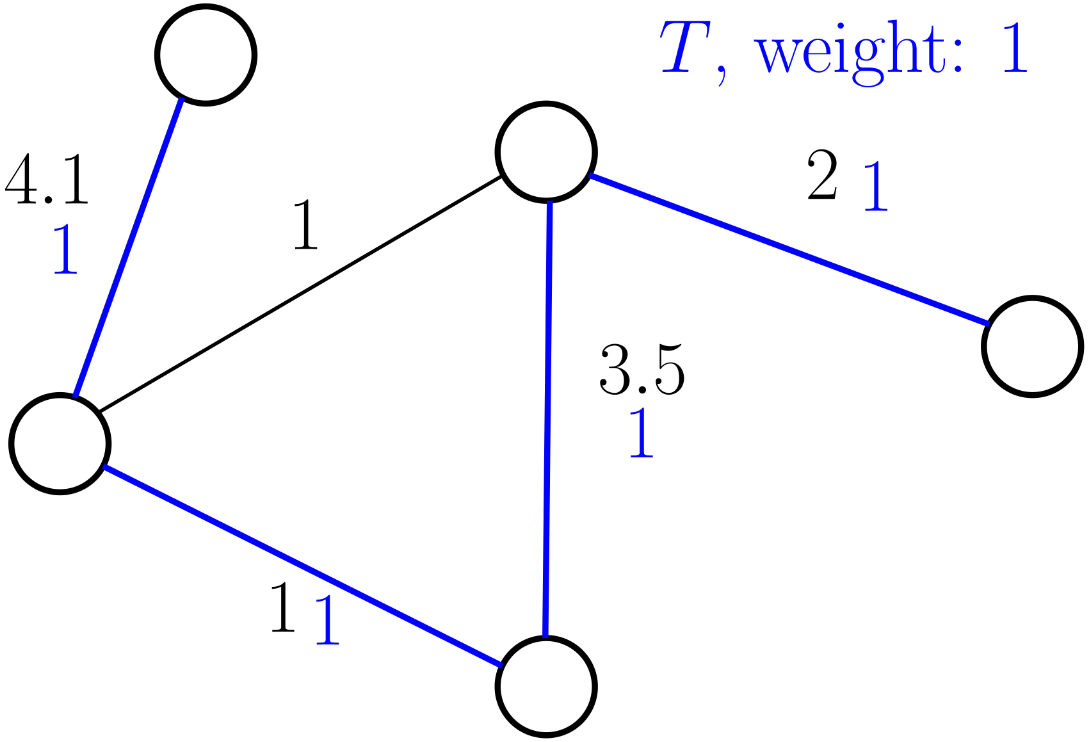
The weight of the packing is the total weight of the trees in it.



Tree Packing

Definition: A set of spanning trees, each with an assigned weight, so that the total weight of trees containing a given edge is no greater than the weight of that edge.

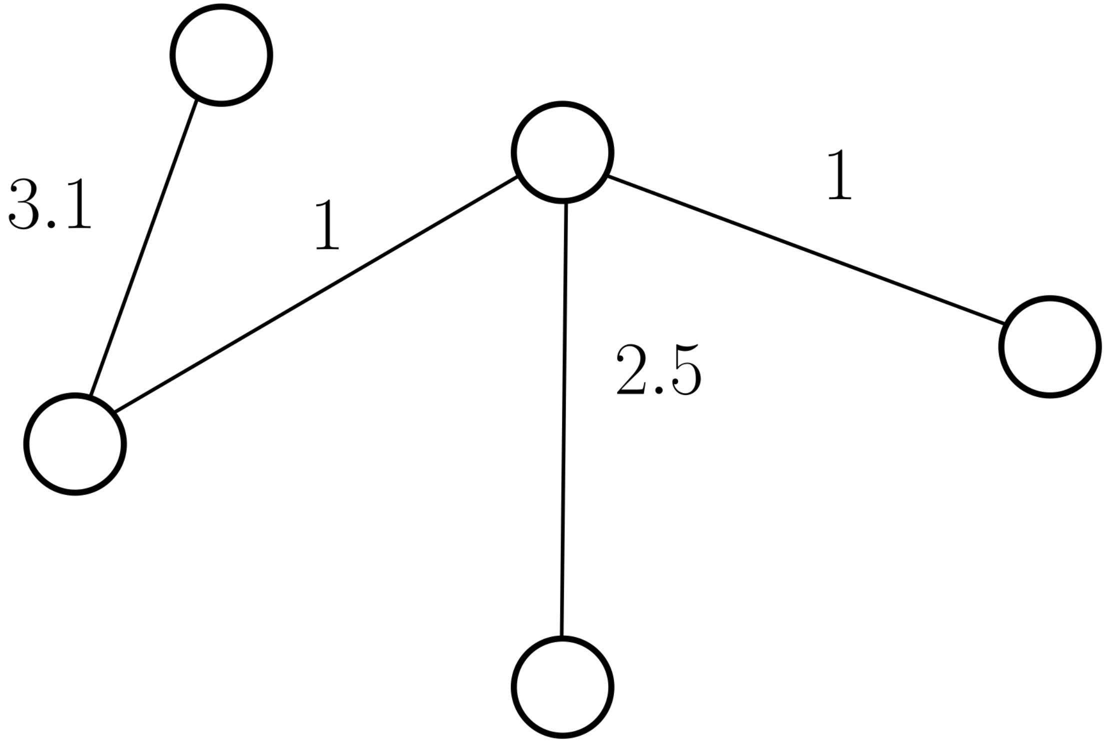
The weight of the packing is the total weight of the trees in it.



Tree Packing

Definition: A set of spanning trees, each with an assigned weight, so that the total weight of trees containing a given edge is no greater than the weight of that edge.

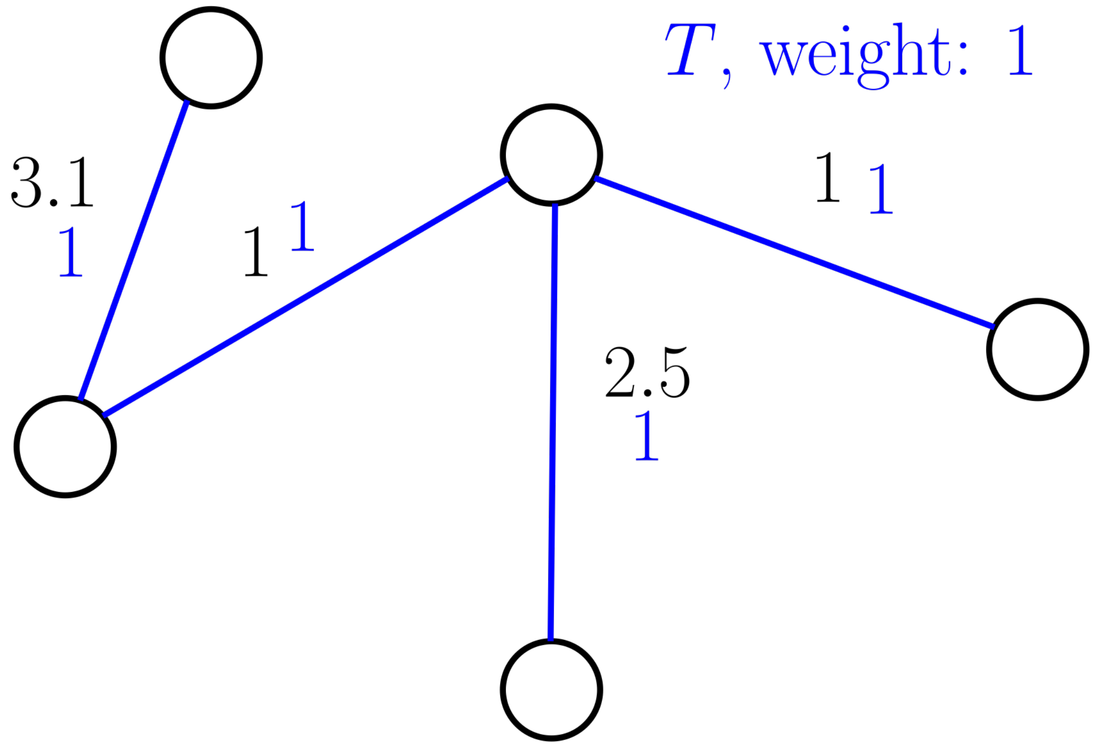
The weight of the packing is the total weight of the trees in it.



Tree Packing

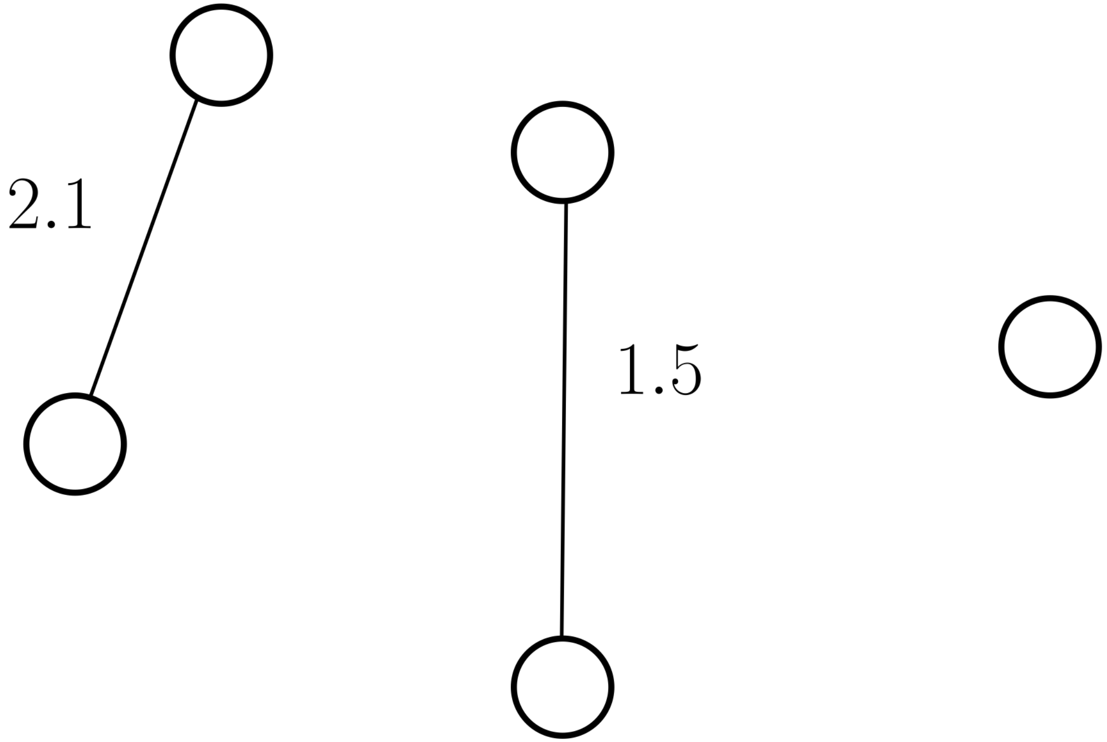
Definition: A set of spanning trees, each with an assigned weight, so that the total weight of trees containing a given edge is no greater than the weight of that edge.

The weight of the packing is the total weight of the trees in it.



Tree Packing

Definition: A set of spanning trees, each with an assigned weight, so that the total weight of trees containing a given edge is no greater than the weight of that edge.



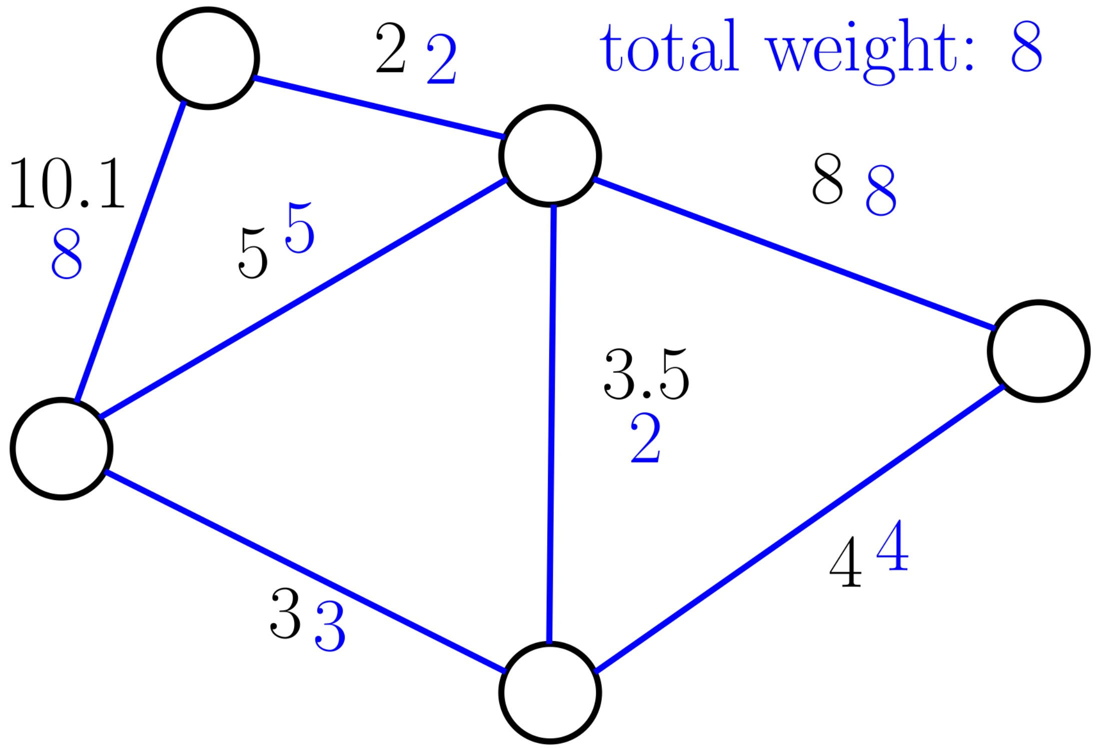
The weight of the packing is the total weight of the trees in it.

Tree Packing

Definition: A set of spanning trees, each with an assigned weight, so that the total weight of trees containing a given edge is no greater than the weight of that edge.

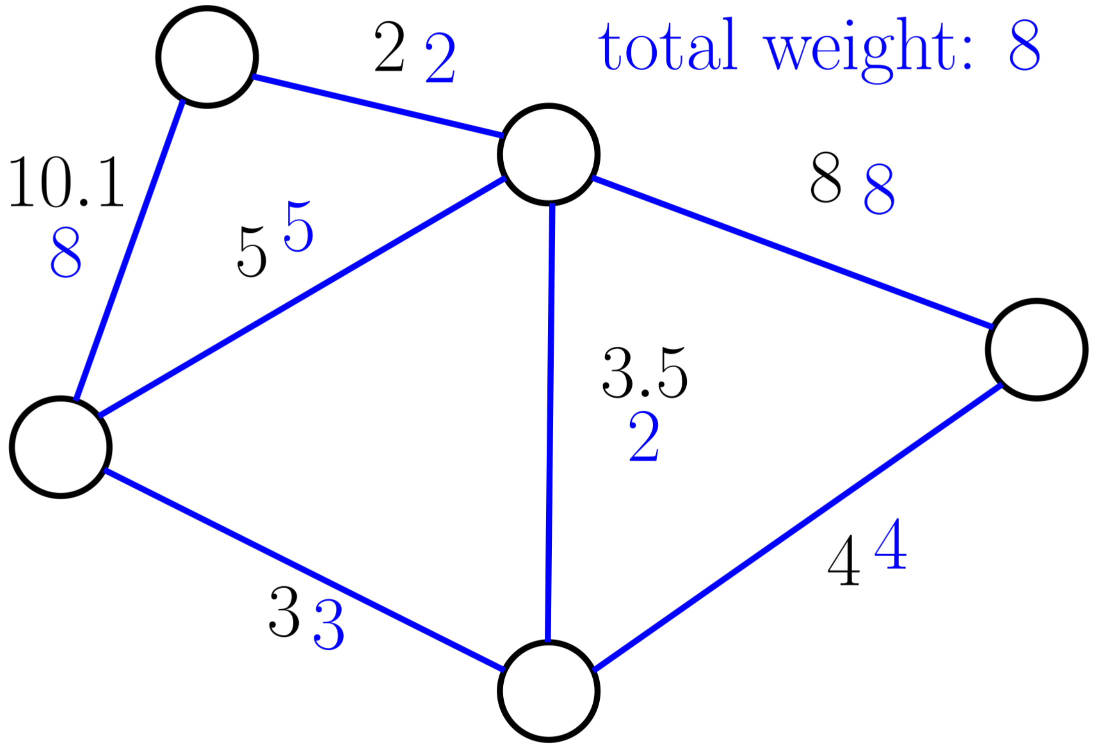
The weight of the packing is the total weight of the trees in it.

Theorem: a packing of weight at least c/2 exists, where c is the weight of the min cut.



Tree Packing

Definition: A set of spanning trees, each with an assigned weight, so that the total weight of trees containing a given edge is no greater than the weight of that edge.



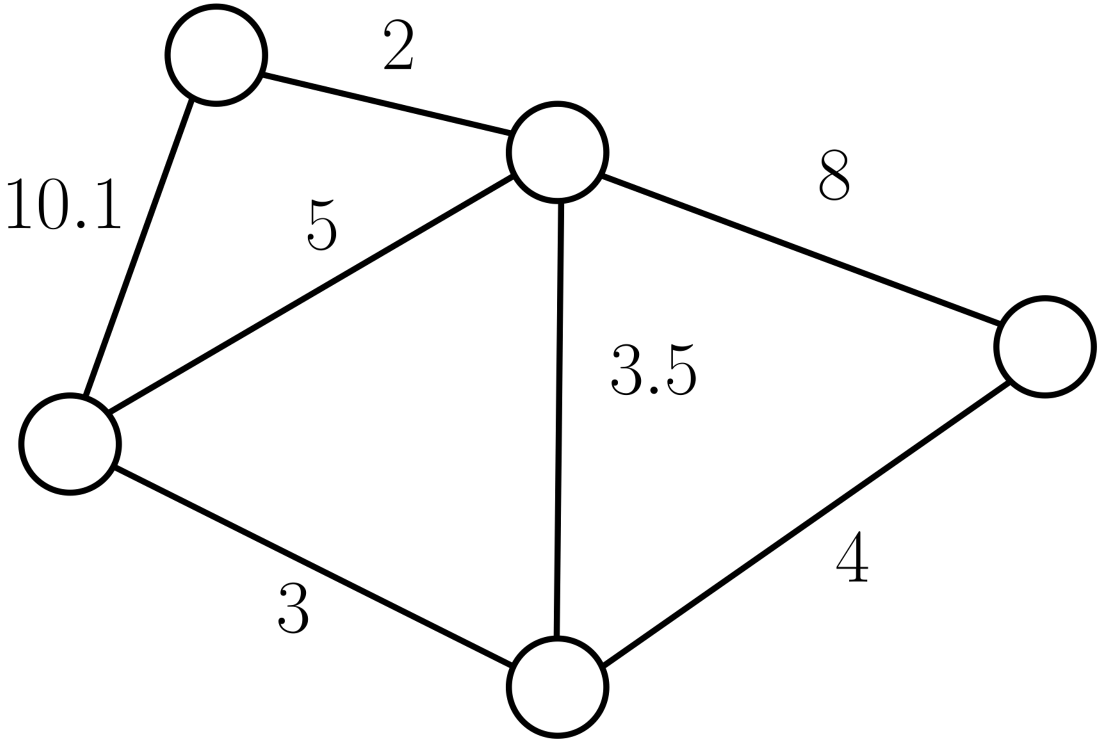
The weight of the packing is the total weight of the trees in it.

Theorem: a packing of weight at least c/2 exists, where c is the weight of the min cut.

Lemma: in a packing of weight $\geq c/2$, each tree crosses the min cut at most twice on average.

Karger's Near-Linear Time Algorithm

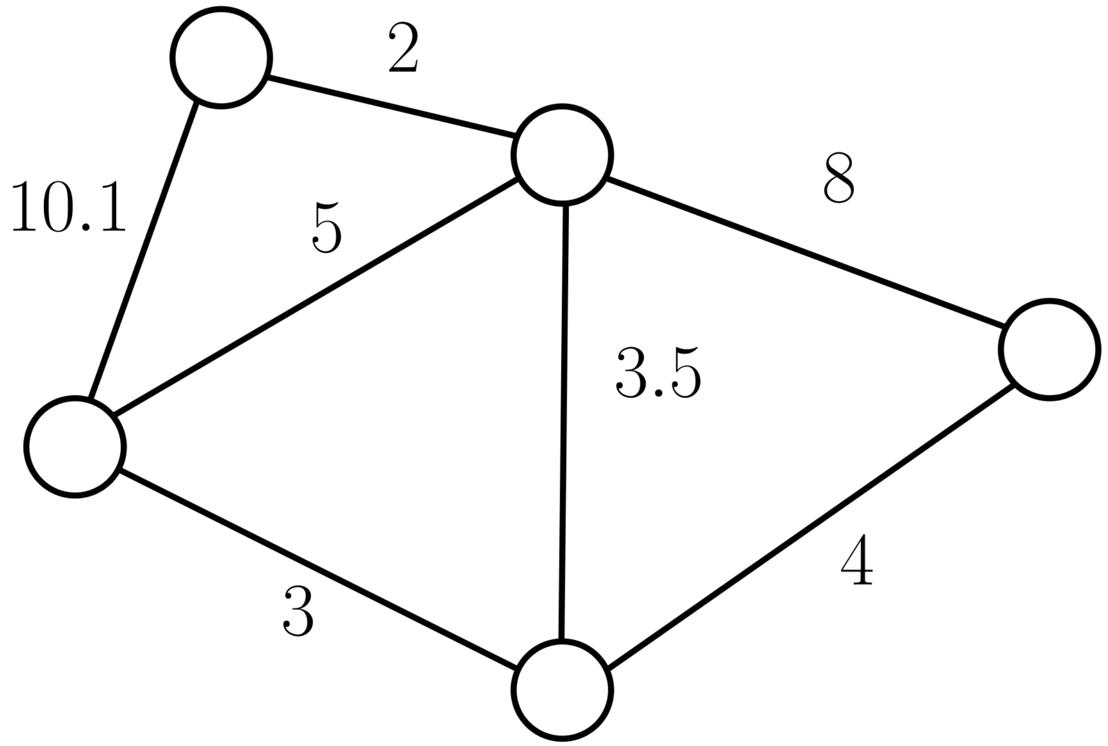
1. Sample edges of G.
2. Pack trees in the sampled graph.
3. Sample trees from the packing.
4. For each sampled tree T, determine a smallest cut of G that cuts at most two edges of T.



Karger’s Near-Linear Time Algorithm

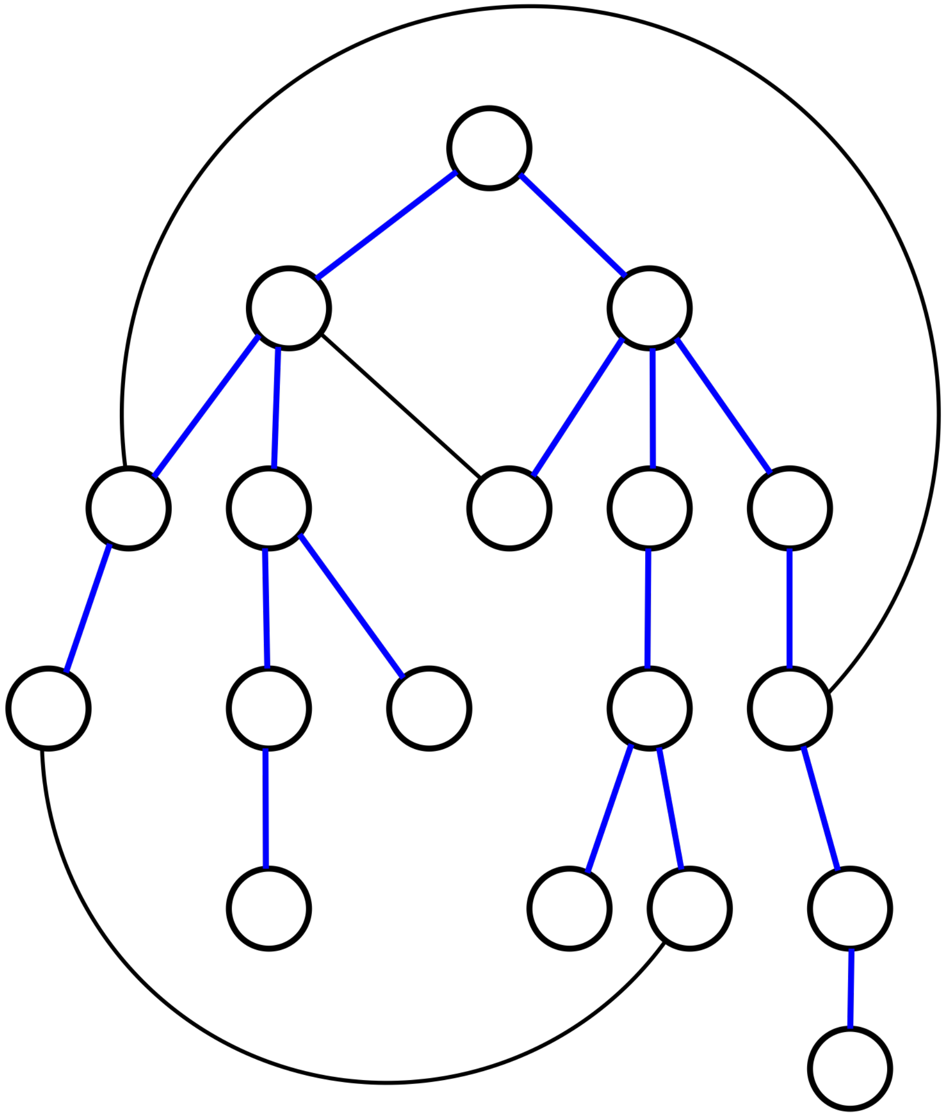
1. Sample edges of G.
2. Pack trees in the sampled graph.
3. Sample trees from the packing.
4. For each sampled tree T, determine a smallest cut of G that cuts at most two edges of T.

Can reduce to step 4.



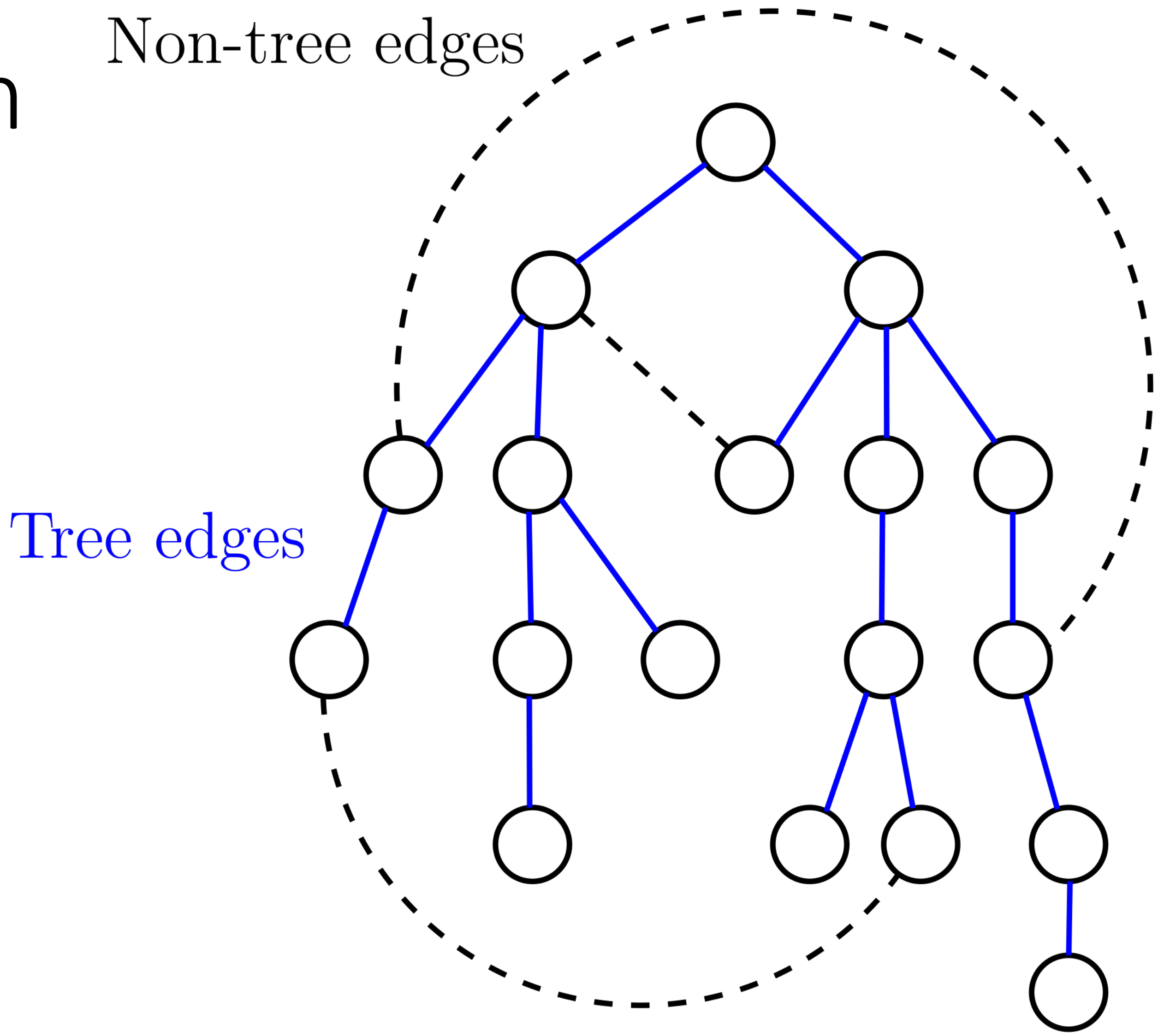
1-Respect Algorithm

Given spanning tree T of a graph G, find a smallest cut of G that cuts one edge of T.



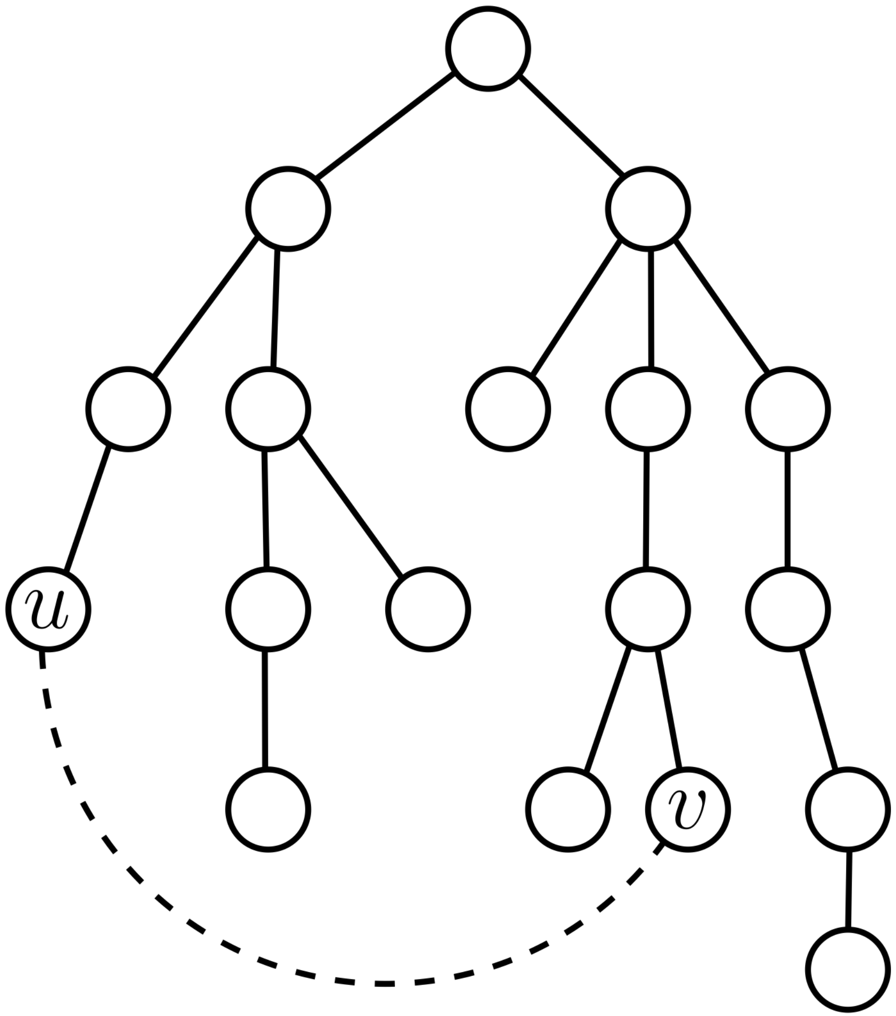
1-Respect Algorithm

Given spanning tree T of a graph G, find a smallest cut of G that cuts one edge of T.



1-Respect Algorithm

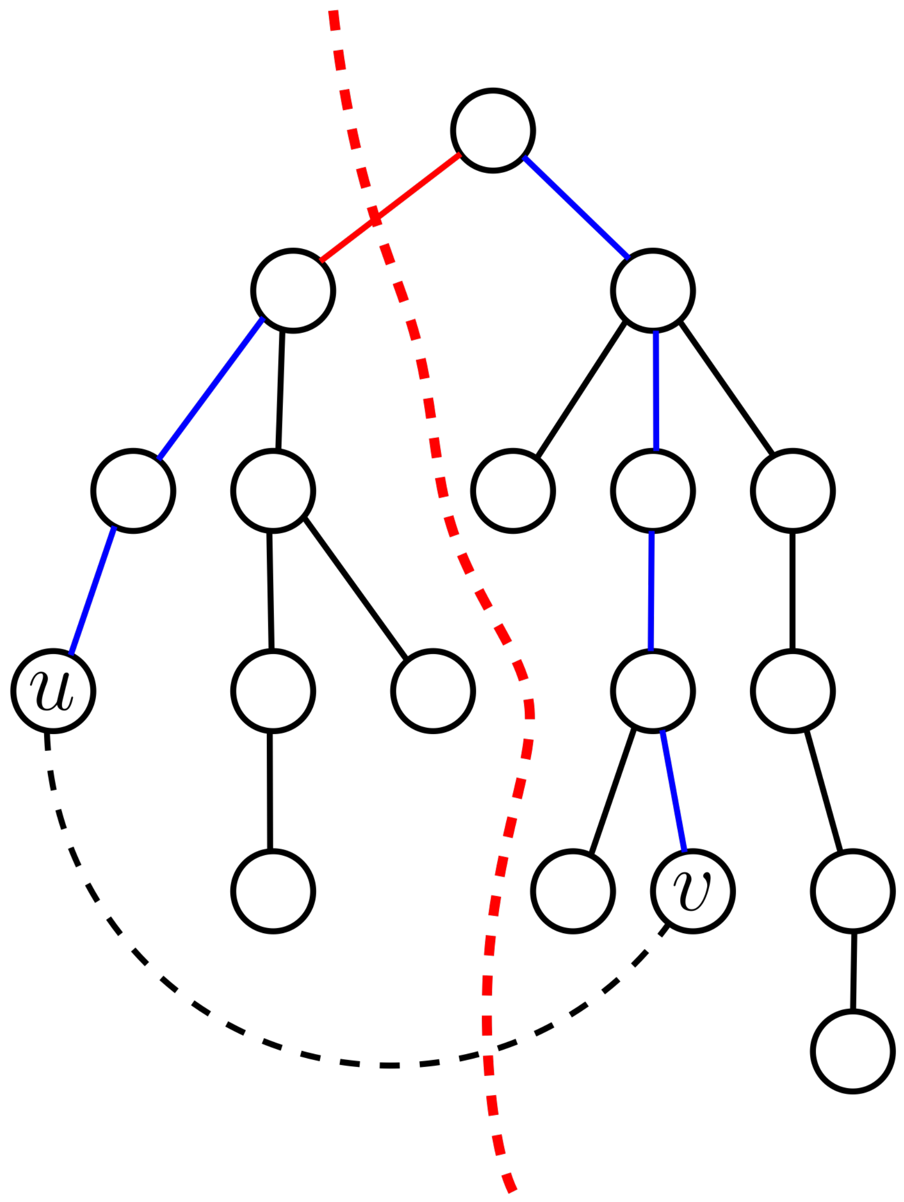
When is a non-tree edge uv cut?



1-Respect Algorithm

When is a non-tree edge uv cut?

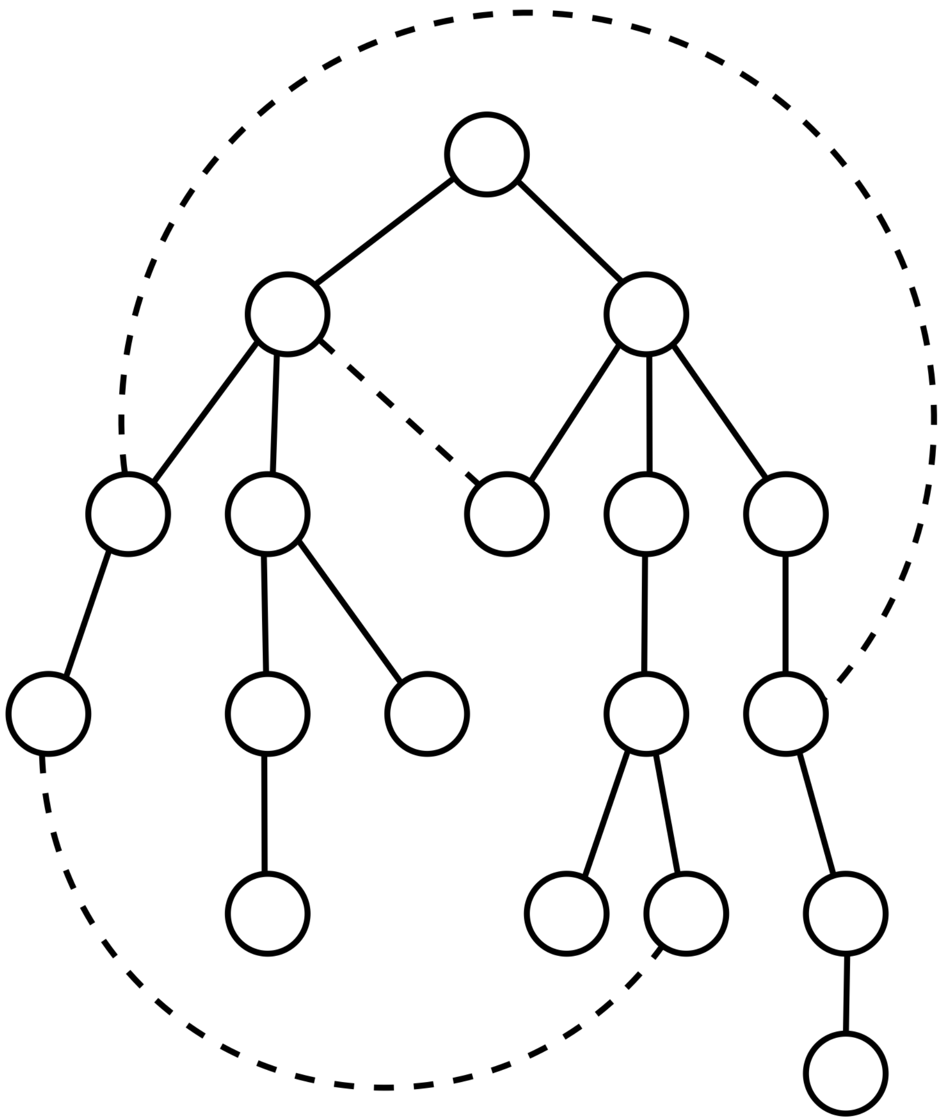
Non-tree edge uv is cut iff the cut in G cuts an edge on the uv-path in T.



1-Respect Algorithm

How to compute the size of all $n-1$ cuts?

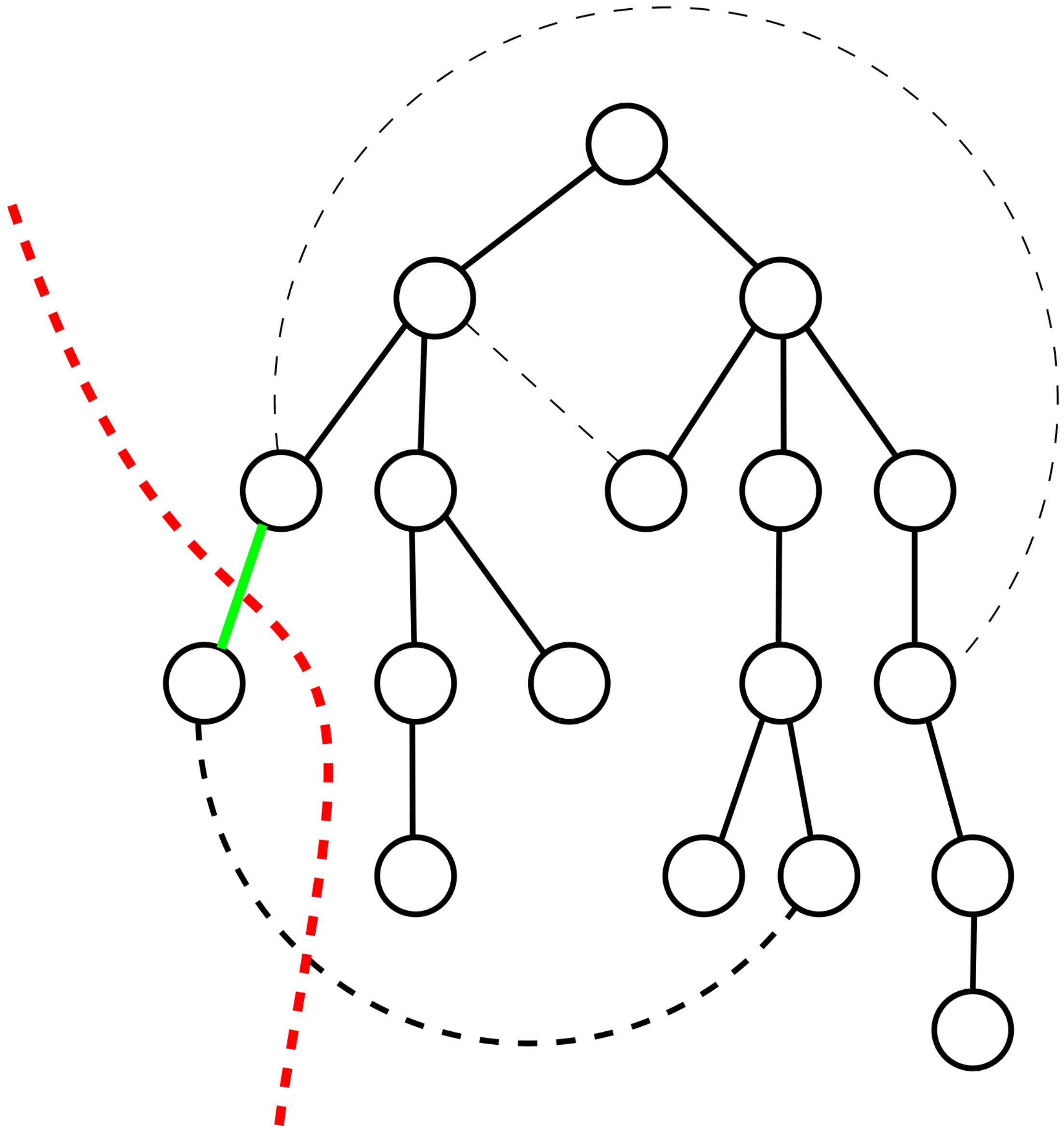
Idea: iterate an edge e through T, keeping track of non-tree edges that cross a cut at e.



1-Respect Algorithm

How to compute the size of all $n-1$ cuts?

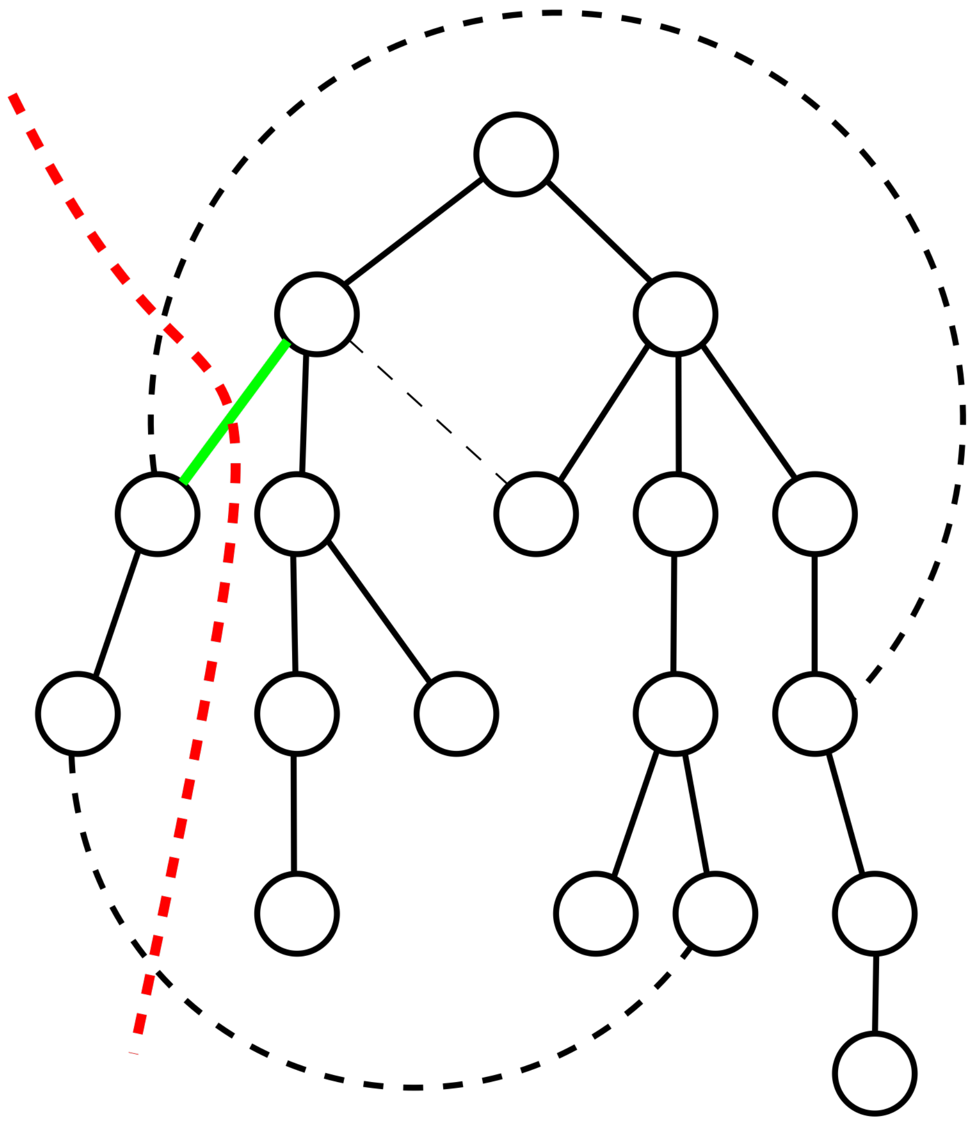
Idea: iterate an edge e through T, keeping track of non-tree edges that cross a cut at e.



1-Respect Algorithm

How to compute the size of all $n-1$ cuts?

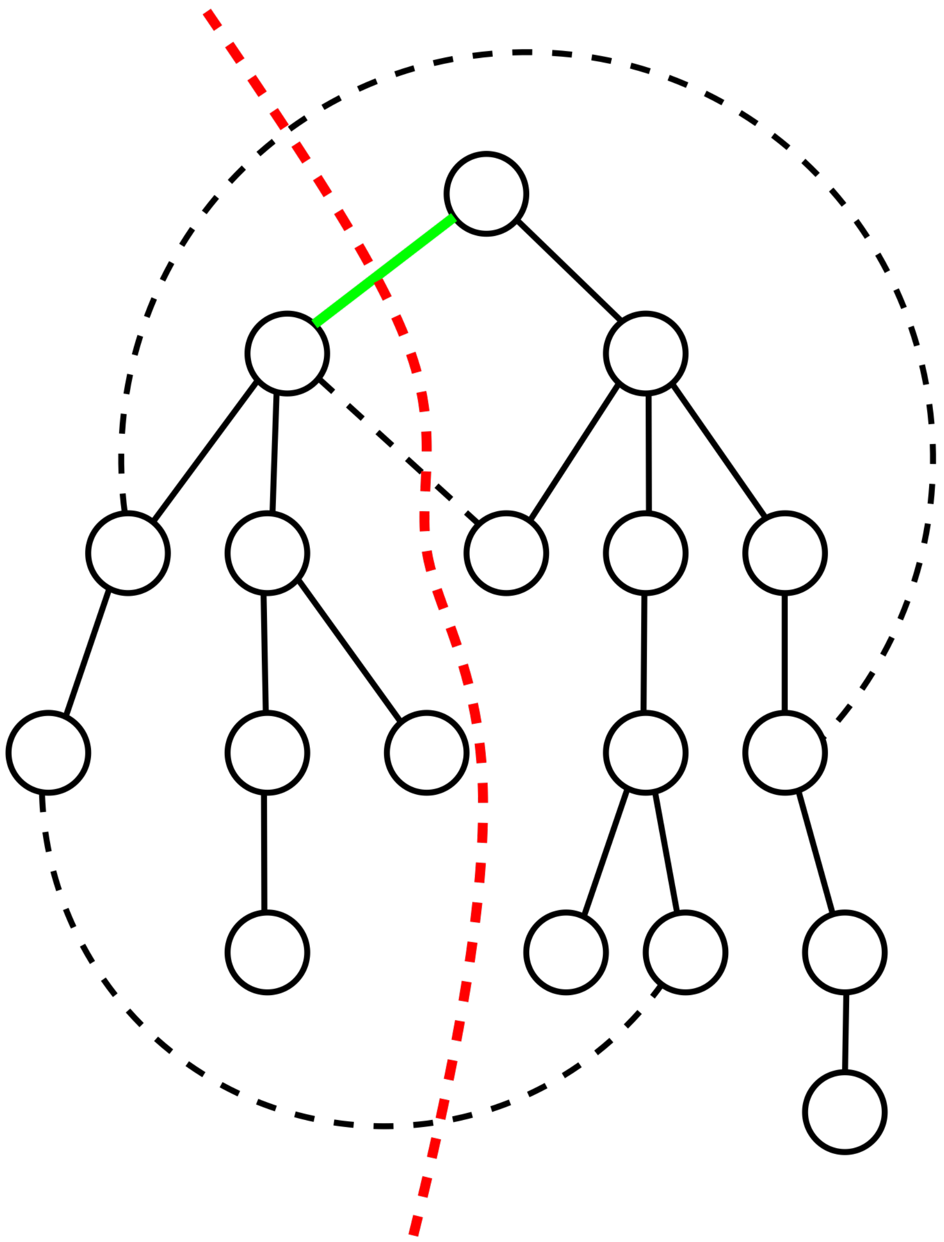
Idea: iterate an edge e through T, keeping track of non-tree edges that cross a cut at e.



1-Respect Algorithm

How to compute the size of all $n - 1$ cuts?

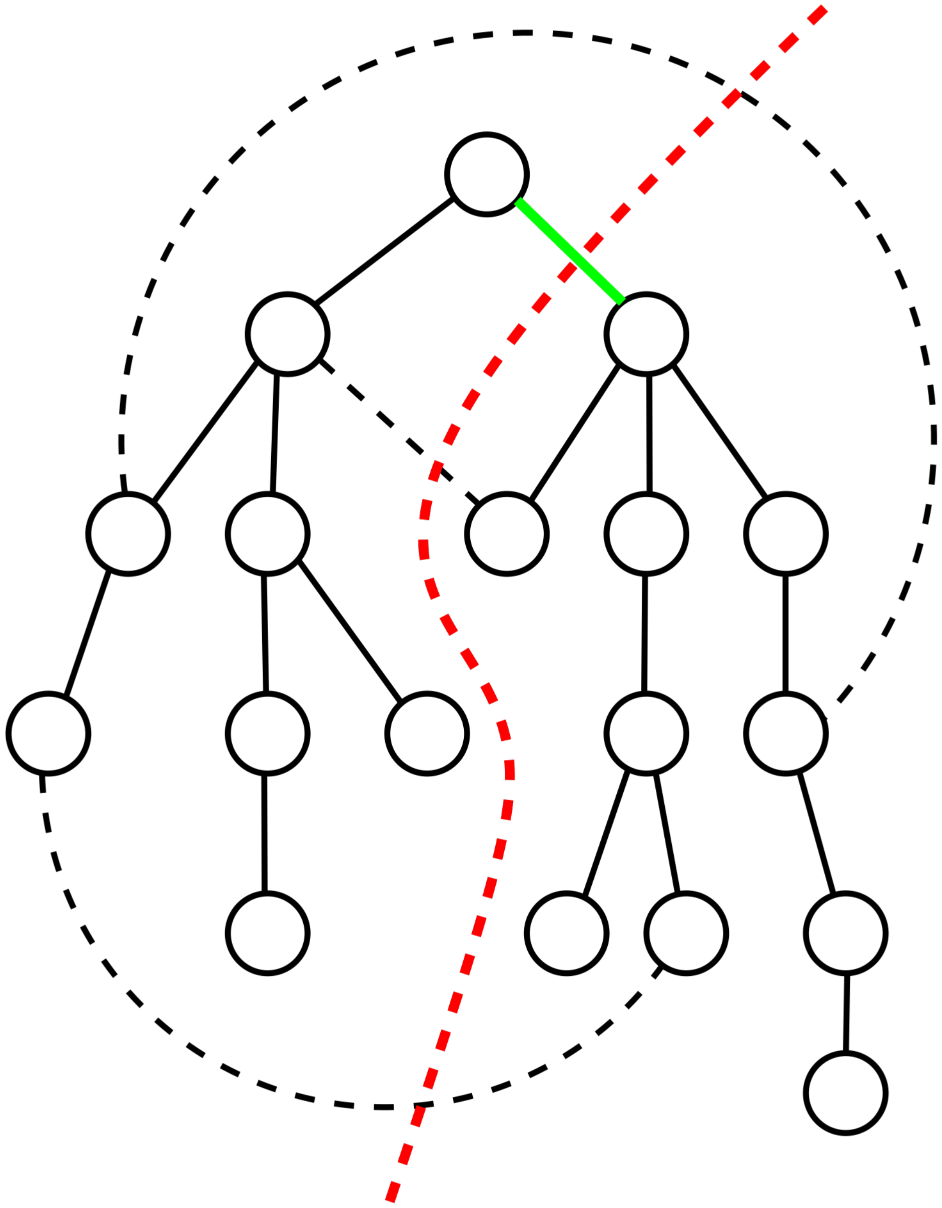
Idea: iterate an edge e through T, keeping track of non-tree edges that cross a cut at e.



1-Respect Algorithm

How to compute the size of all $n-1$ cuts?

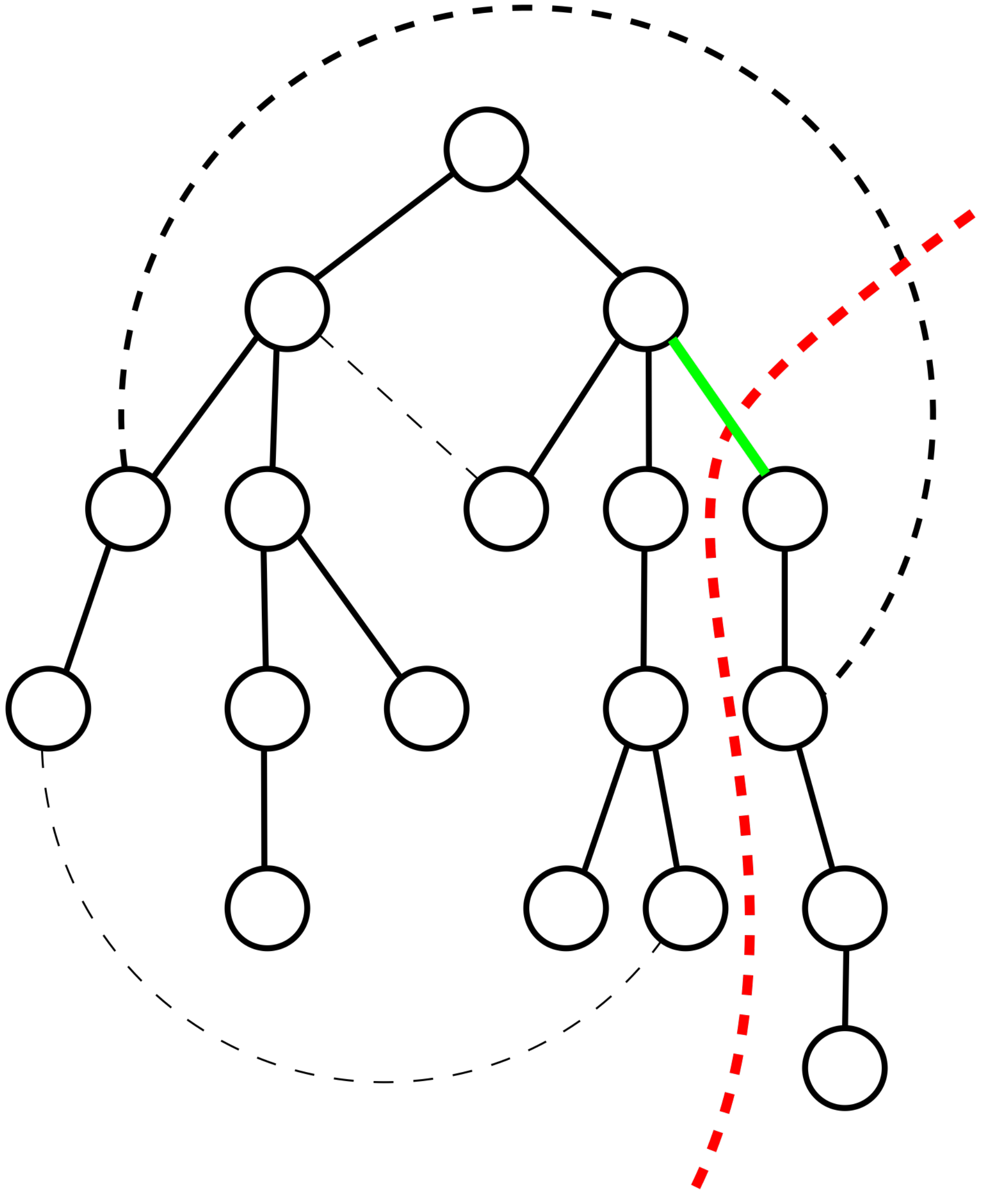
Idea: iterate an edge e through T, keeping track of non-tree edges that cross a cut at e.



1-Respect Algorithm

How to compute the size of all $n - 1$ cuts?

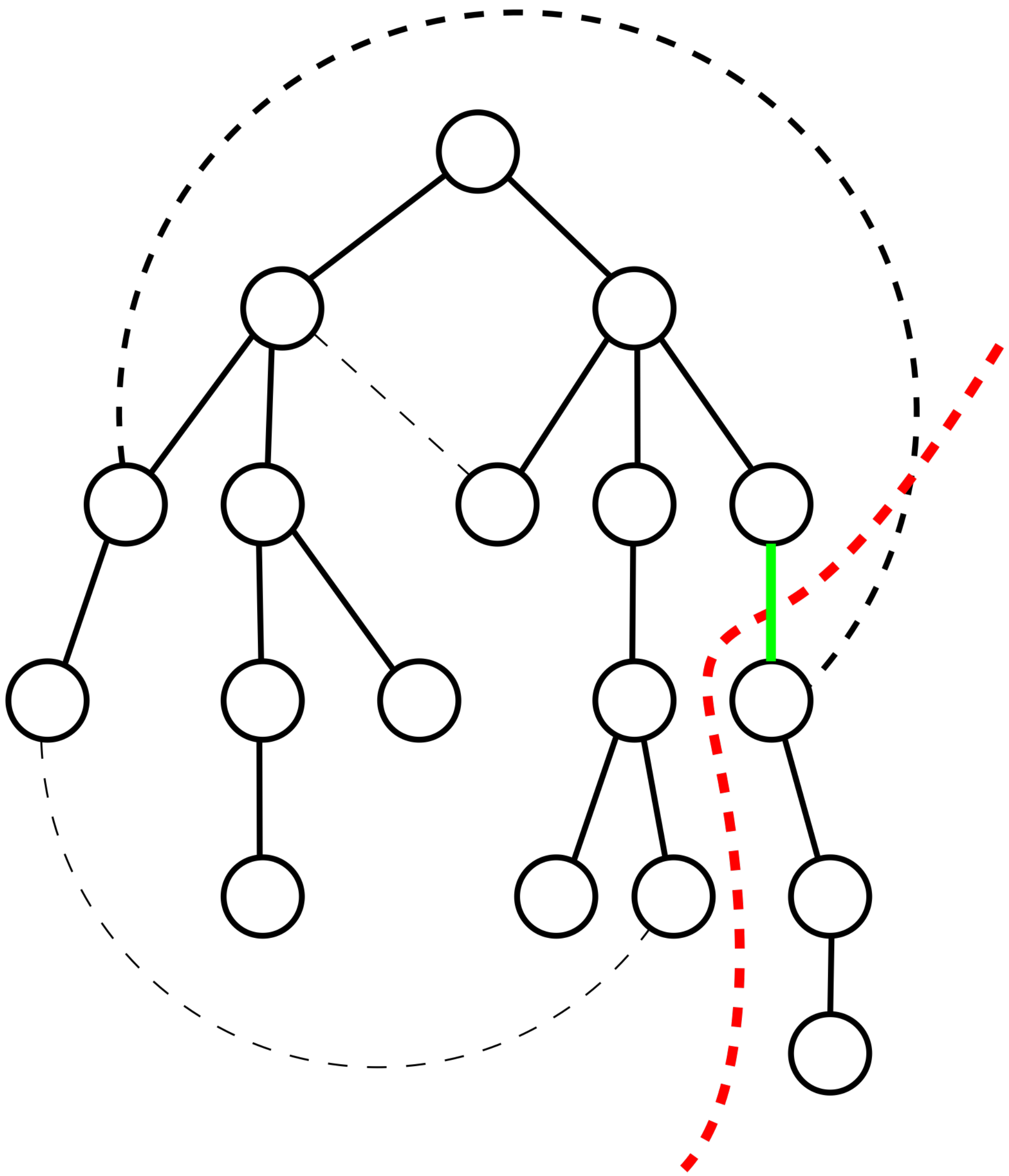
Idea: iterate an edge e through T, keeping track of non-tree edges that cross a cut at e.



1-Respect Algorithm

How to compute the size of all $n-1$ cuts?

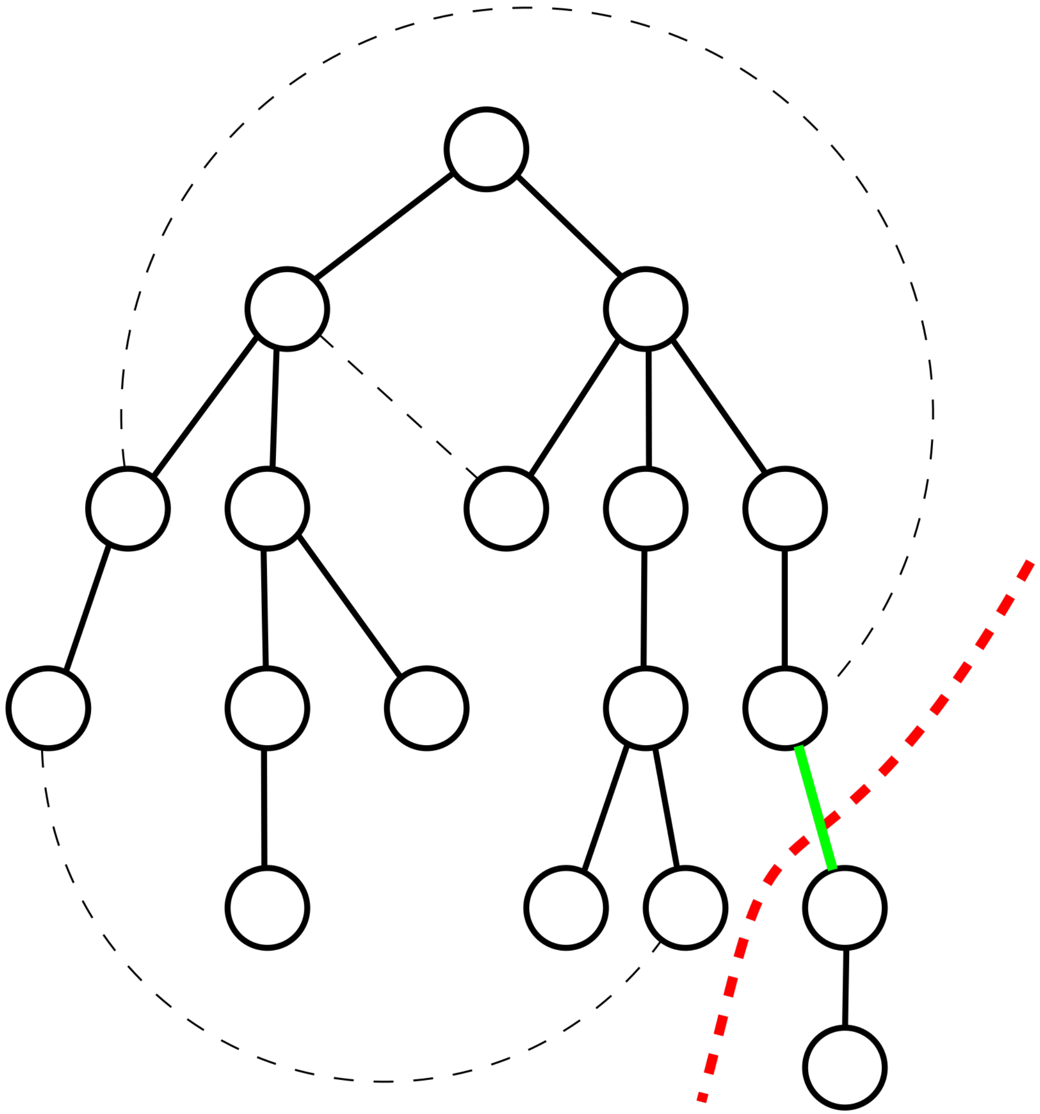
Idea: iterate an edge e through T, keeping track of non-tree edges that cross a cut at e.



1-Respect Algorithm

How to compute the size of all $n - 1$ cuts?

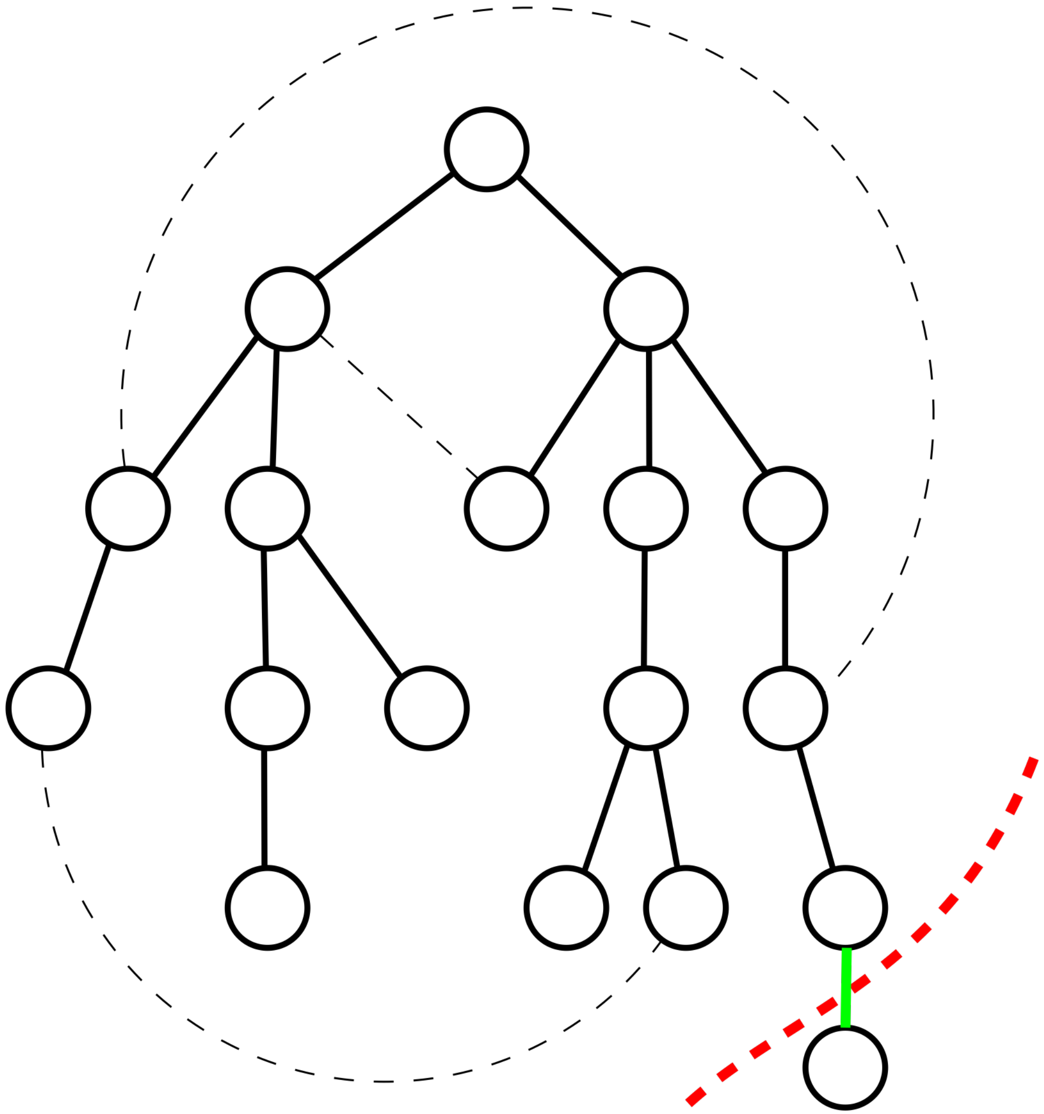
Idea: iterate an edge e through T, keeping track of non-tree edges that cross a cut at e.



1-Respect Algorithm

How to compute the size of all $n-1$ cuts?

Idea: iterate an edge e through T, keeping track of non-tree edges that cross a cut at e.

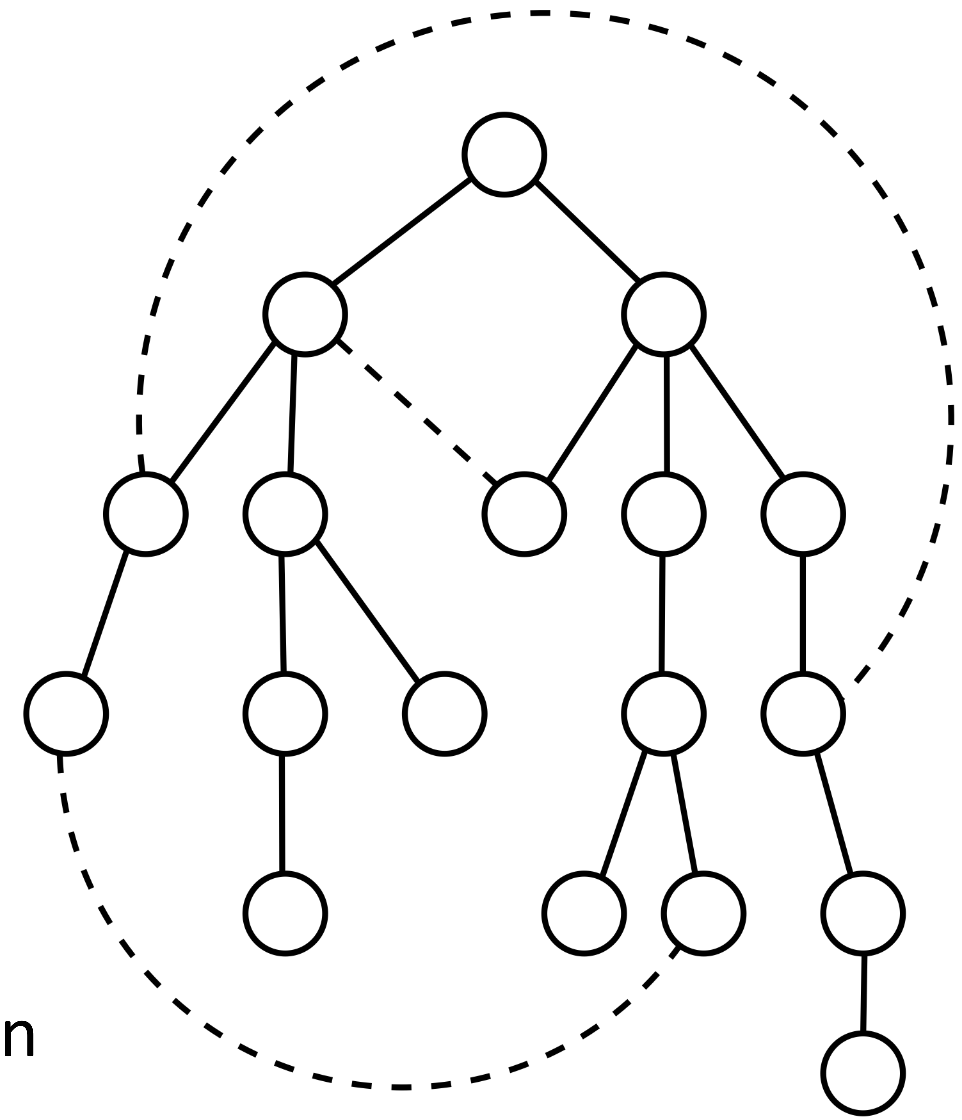


1-Respect Algorithm

How to compute the size of all $n-1$ cuts?

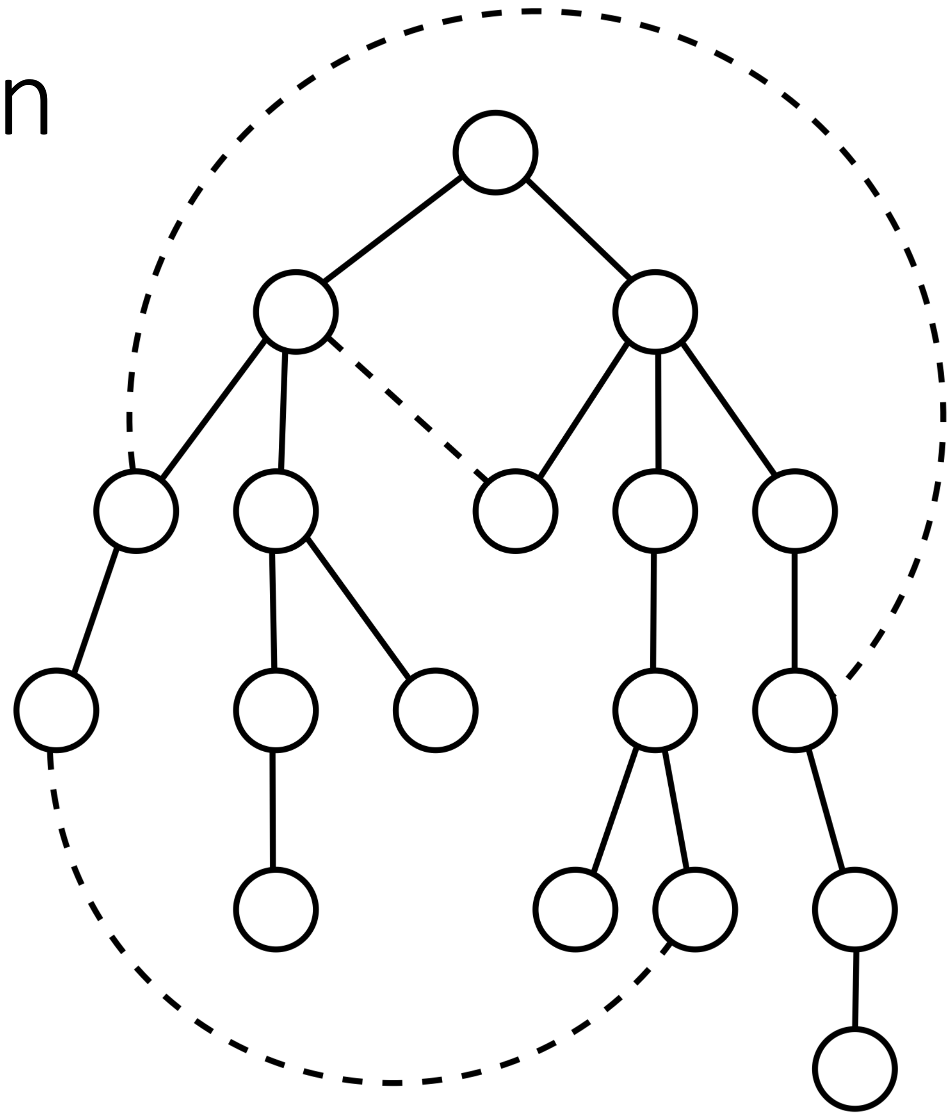
Idea: iterate an edge e through T, keeping track of non-tree edges that cross a cut at e.

Is there an order of edges e that results in non-tree edges transitioning on and off the current cut a small number of times?



Heavy-Light Decomposition

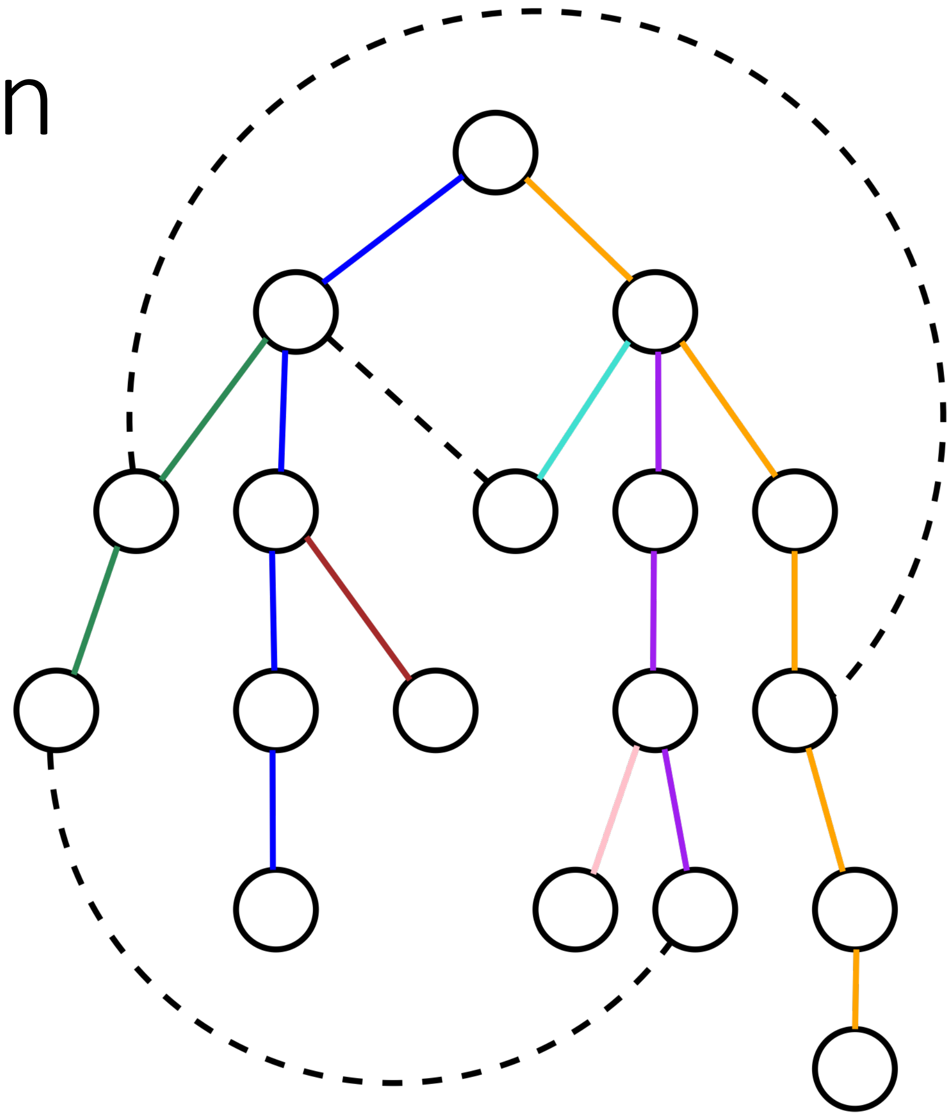
1. Split T into root-to-leaf paths.
2. Continue the path to the child with the most descendants.



Heavy-Light Decomposition

1. Split T into root-to-leaf paths.
2. Continue the path to the child with the most descendants.

Any root-to-leaf path requires at most $O(\log n)$ color changes.

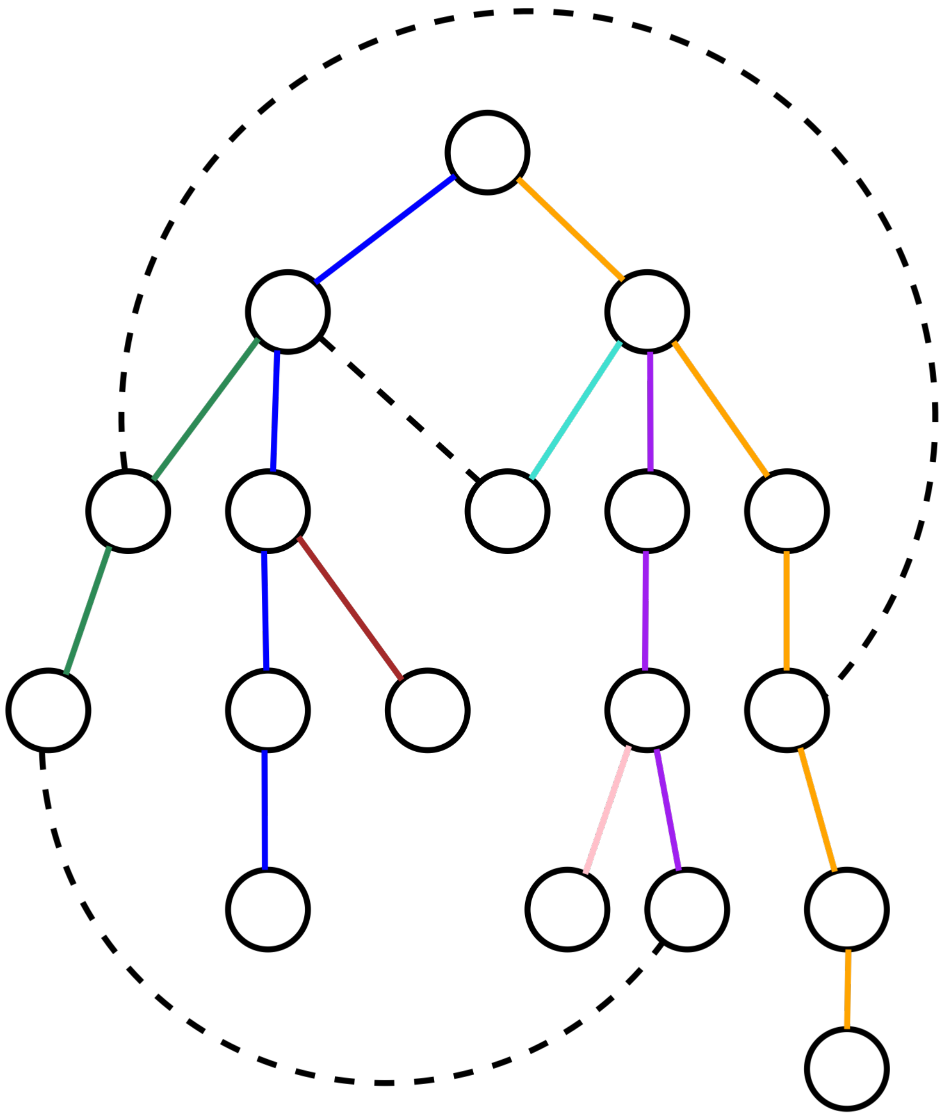


1-Respect Algorithm

1. Iterate edge e in heavy-light decomposition order.
2. Keep track of non-tree edges uv that cross a cut at e.

Non-tree edge uv will transition on or off the current cut $O(\log n)$ times.

$\Rightarrow O(m \log n)$ time.

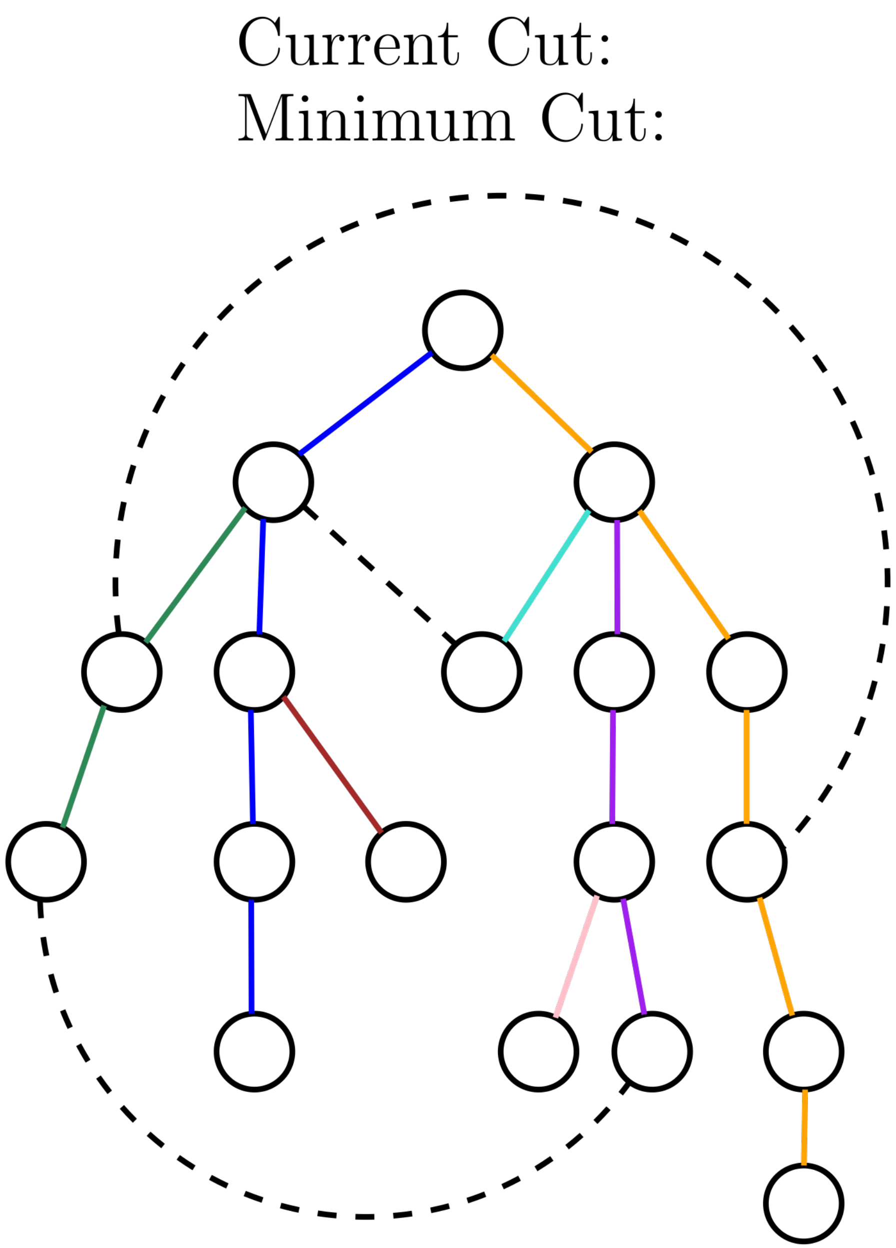


1-Respect Algorithm

1. Iterate edge e in heavy-light decomposition order.
2. Keep track of non-tree edges uv that cross a cut at e.

Non-tree edge uv will transition on or off the current cut $O(\log n)$ times.

$\Rightarrow O(m \log n)$ time.

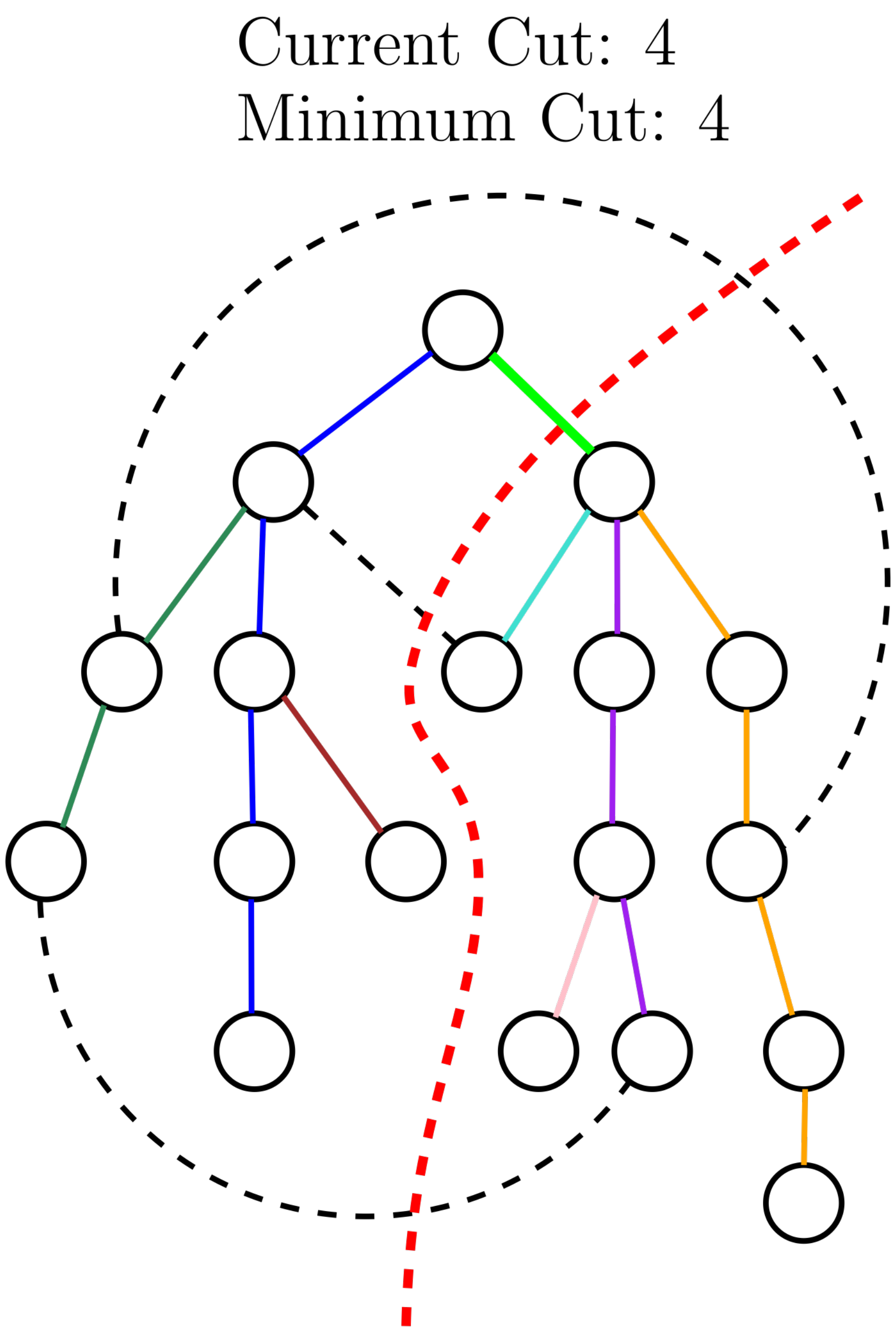


1-Respect Algorithm

1. Iterate edge e in heavy-light decomposition order.
2. Keep track of non-tree edges uv that cross a cut at e.

Non-tree edge uv will transition on or off the current cut $O(\log n)$ times.

$\Rightarrow O(m \log n)$ time.

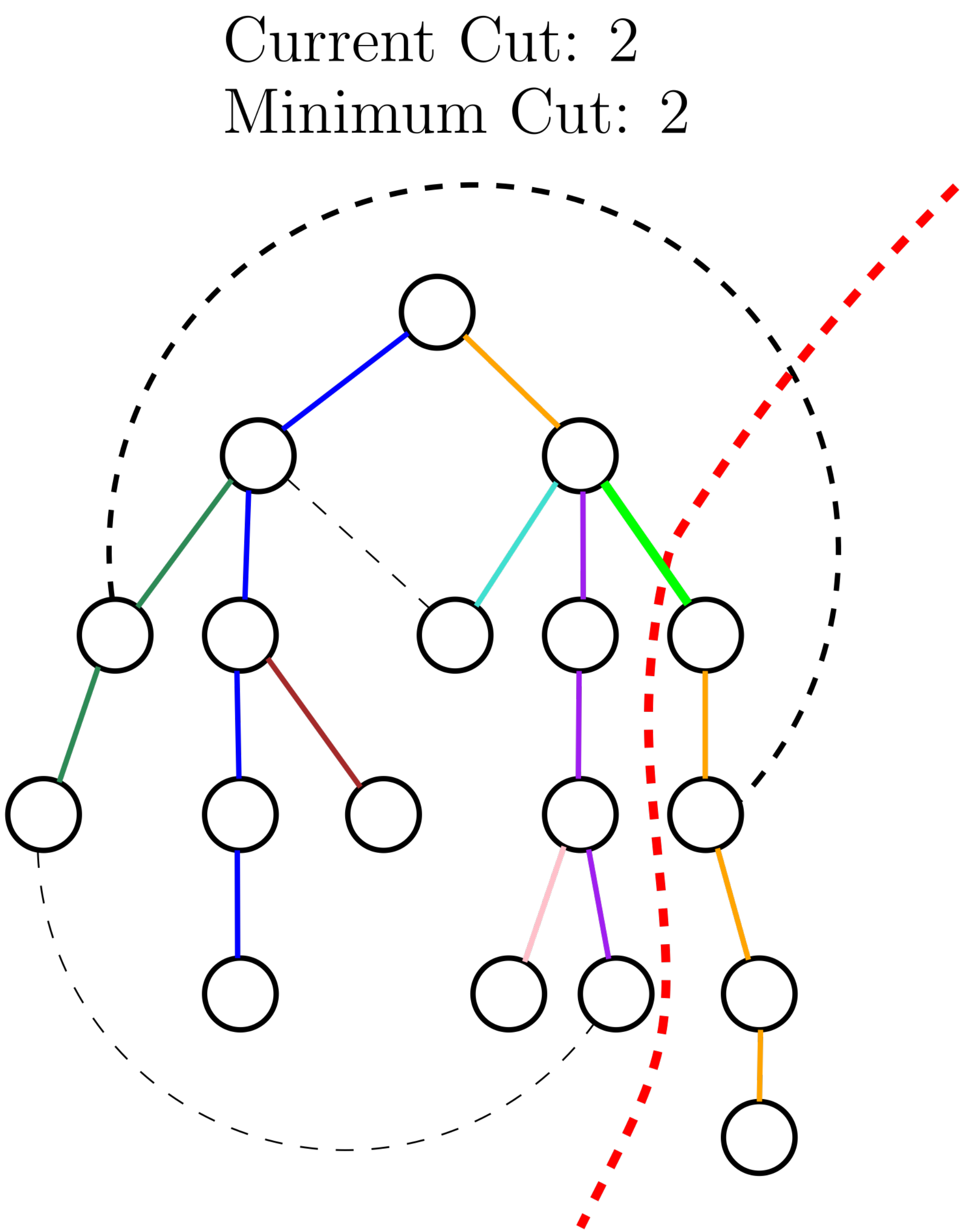


1-Respect Algorithm

1. Iterate edge e in heavy-light decomposition order.
2. Keep track of non-tree edges uv that cross a cut at e.

Non-tree edge uv will transition on or off the current cut $O(\log n)$ times.

$\Rightarrow O(m \log n)$ time.

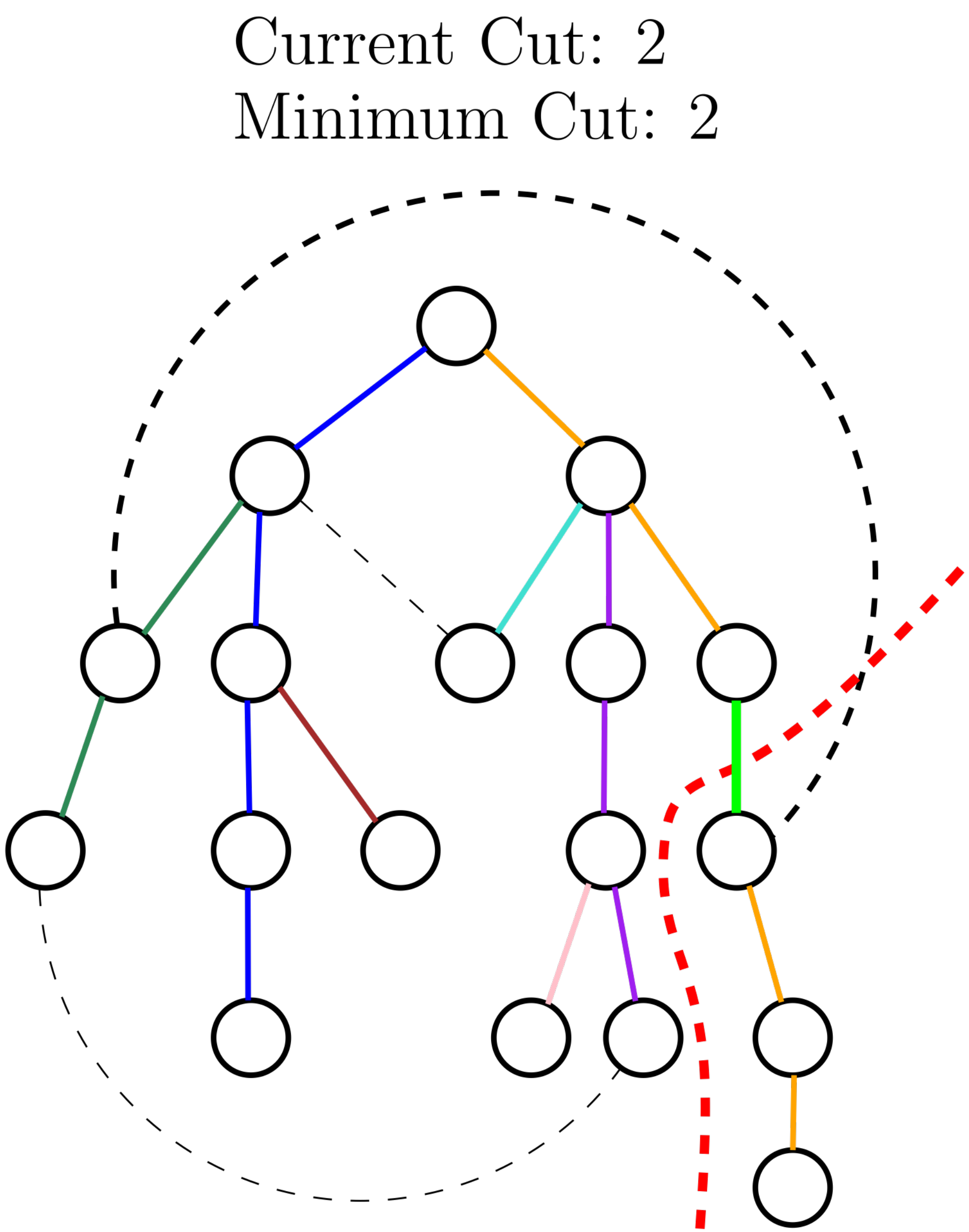


1-Respect Algorithm

1. Iterate edge e in heavy-light decomposition order.
2. Keep track of non-tree edges uv that cross a cut at e.

Non-tree edge uv will transition on or off the current cut $O(\log n)$ times.

$\Rightarrow O(m \log n)$ time.

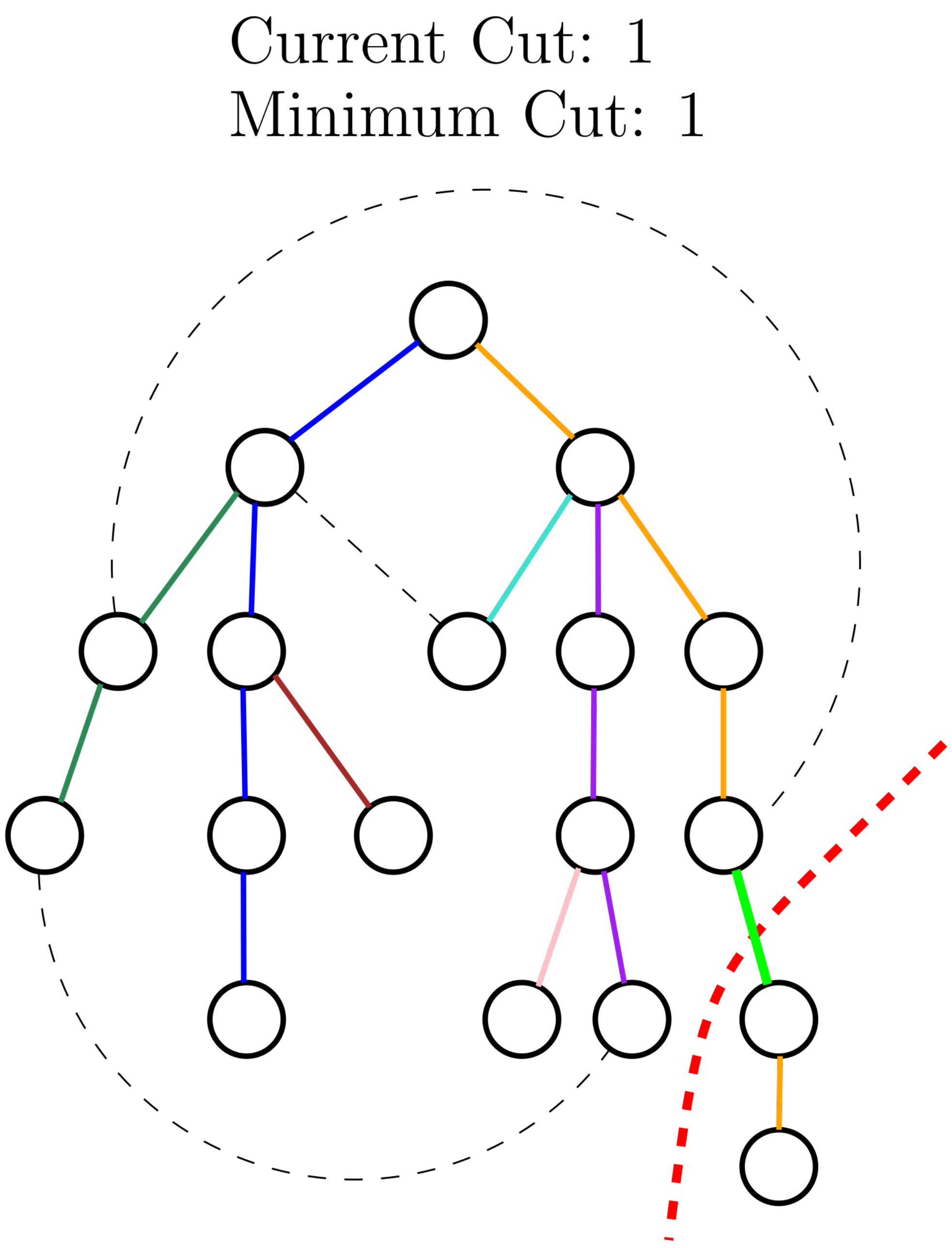


1-Respect Algorithm

1. Iterate edge e in heavy-light decomposition order.
2. Keep track of non-tree edges uv that cross a cut at e.

Non-tree edge uv will transition on or off the current cut $O(\log n)$ times.

$\Rightarrow O(m \log n)$ time.

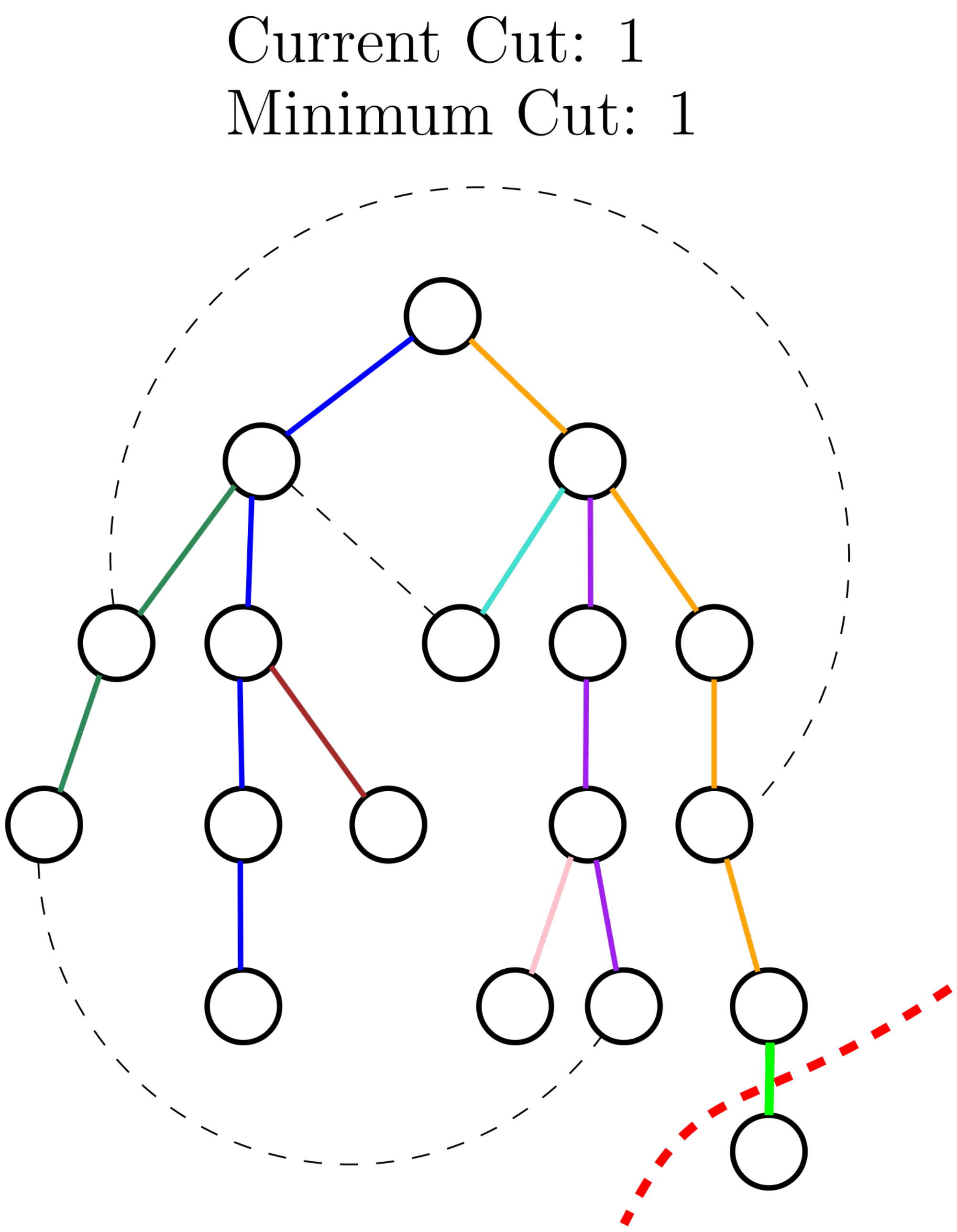


1-Respect Algorithm

1. Iterate edge e in heavy-light decomposition order.
2. Keep track of non-tree edges uv that cross a cut at e.

Non-tree edge uv will transition on or off the current cut $O(\log n)$ times.

$\Rightarrow O(m \log n)$ time.

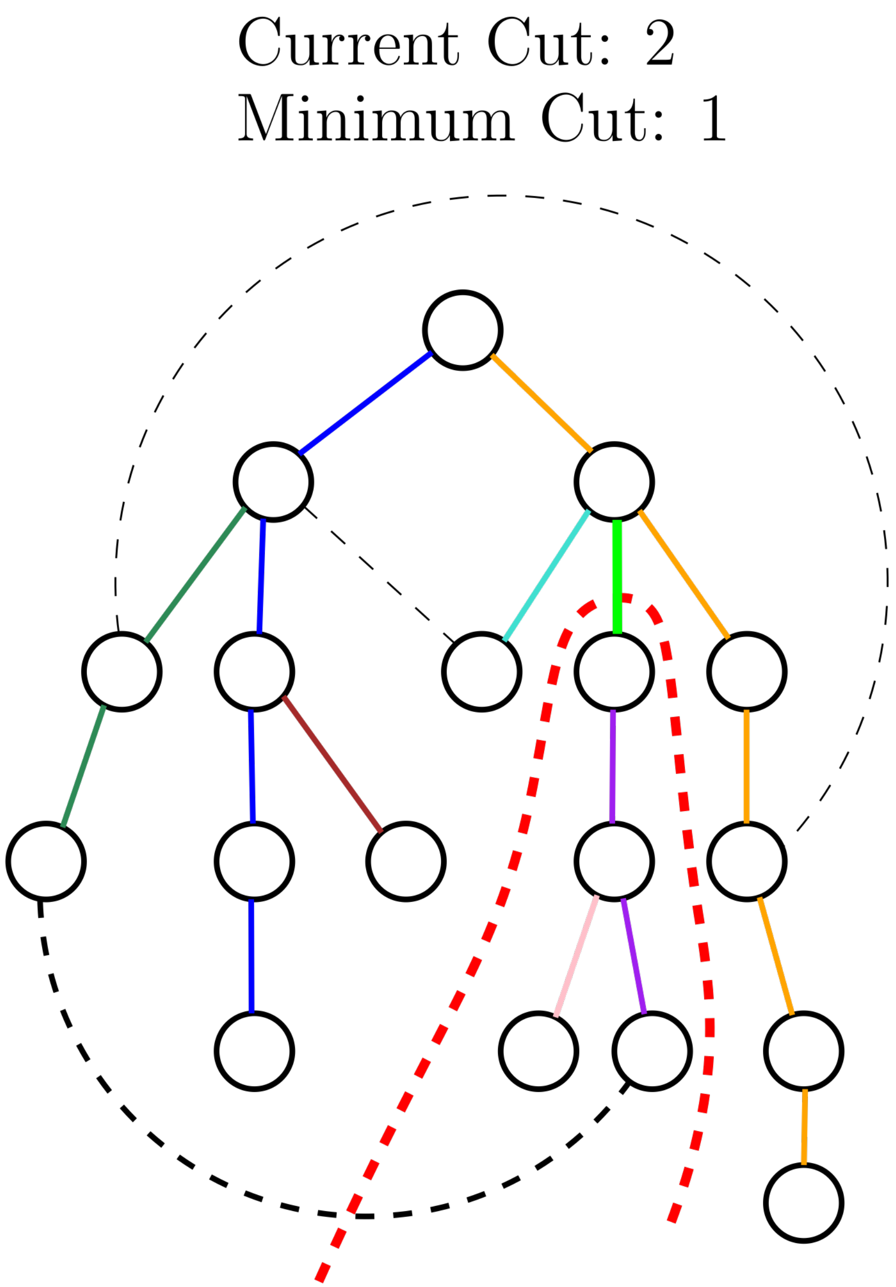


1-Respect Algorithm

1. Iterate edge e in heavy-light decomposition order.
2. Keep track of non-tree edges uv that cross a cut at e.

Non-tree edge uv will transition on or off the current cut $O(\log n)$ times.

$\Rightarrow O(m \log n)$ time.

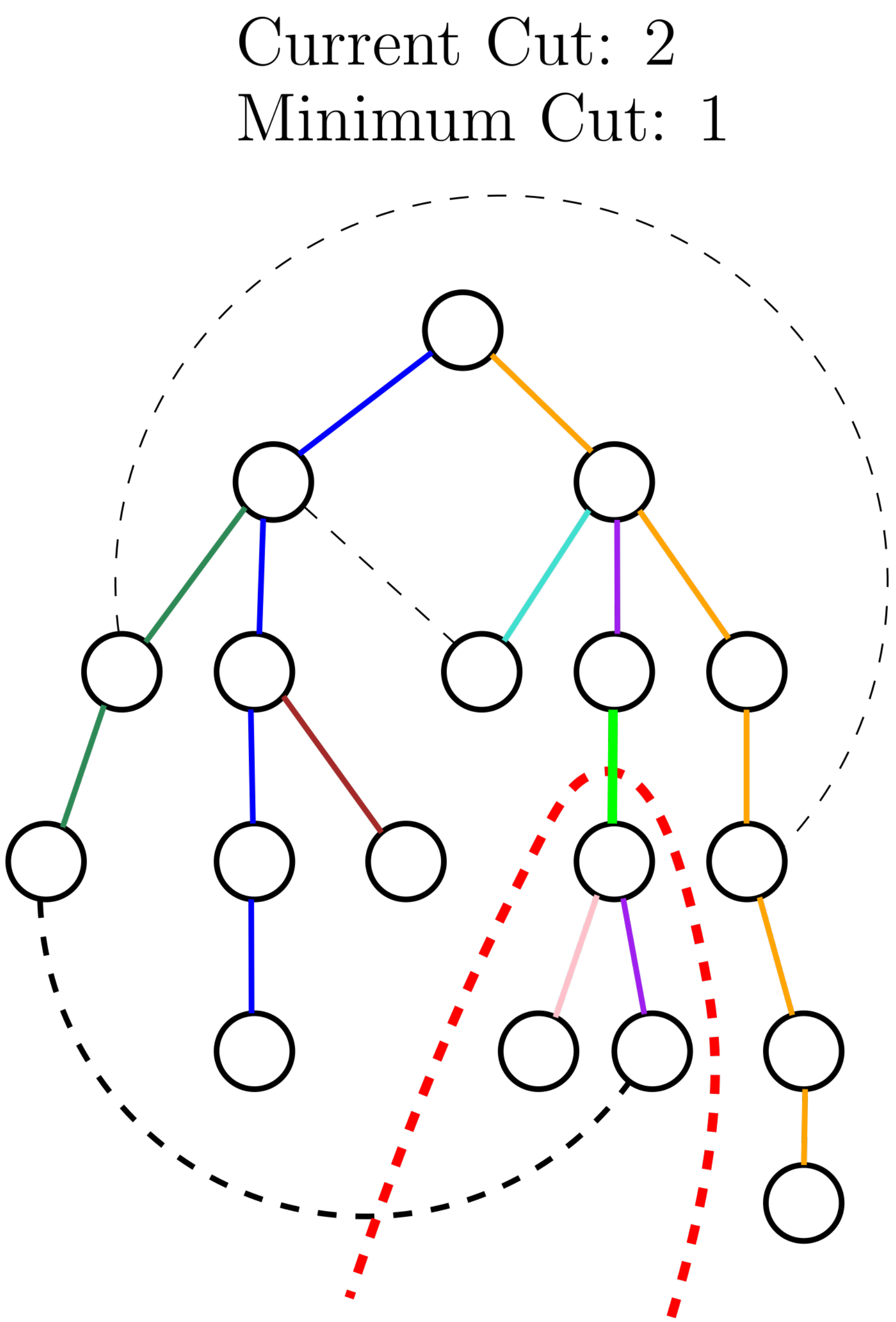


1-Respect Algorithm

1. Iterate edge e in heavy-light decomposition order.
2. Keep track of non-tree edges uv that cross a cut at e.

Non-tree edge uv will transition on or off the current cut $O(\log n)$ times.

$\Rightarrow O(m \log n)$ time.

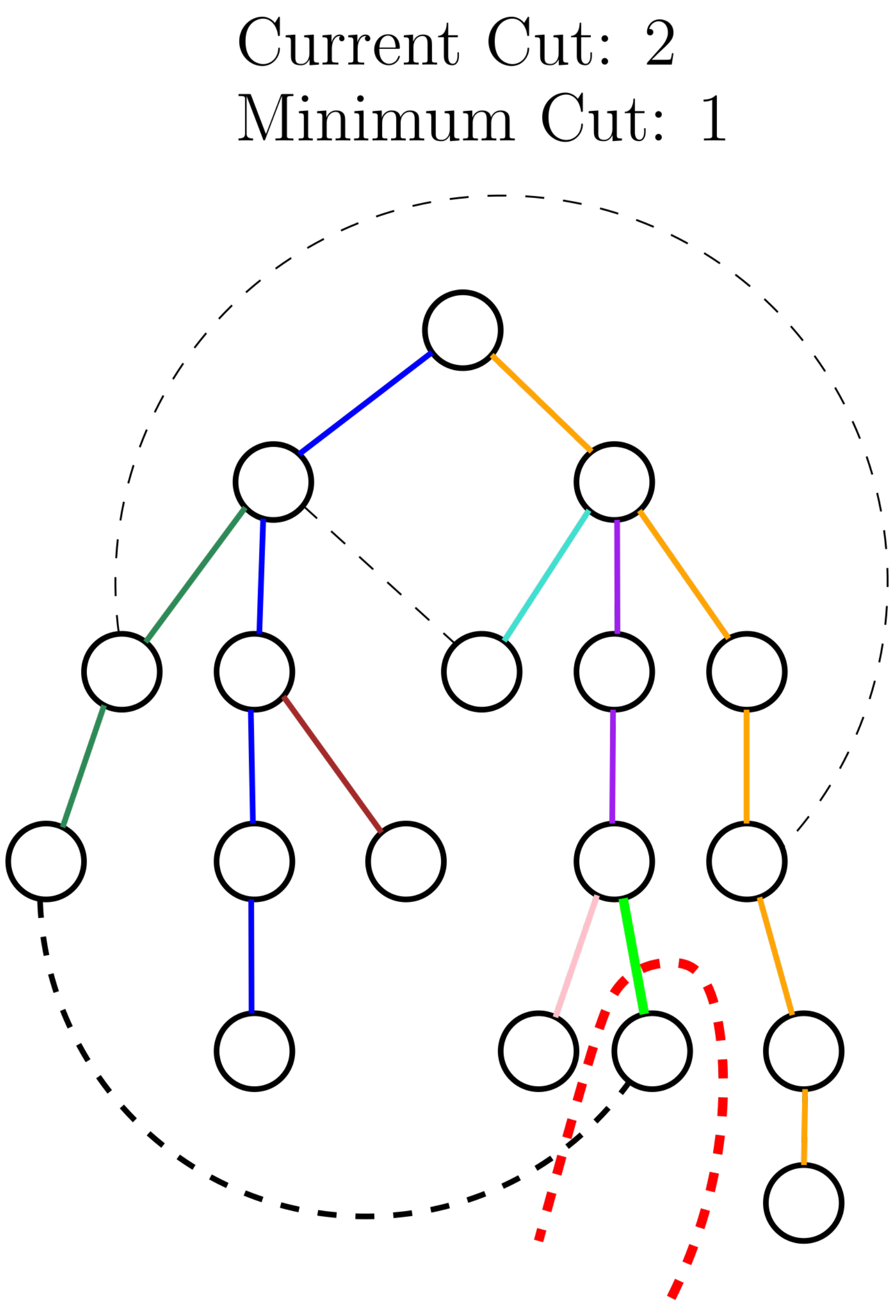


1-Respect Algorithm

1. Iterate edge e in heavy-light decomposition order.
2. Keep track of non-tree edges uv that cross a cut at e.

Non-tree edge uv will transition on or off the current cut $O(\log n)$ times.

$\Rightarrow O(m \log n)$ time.

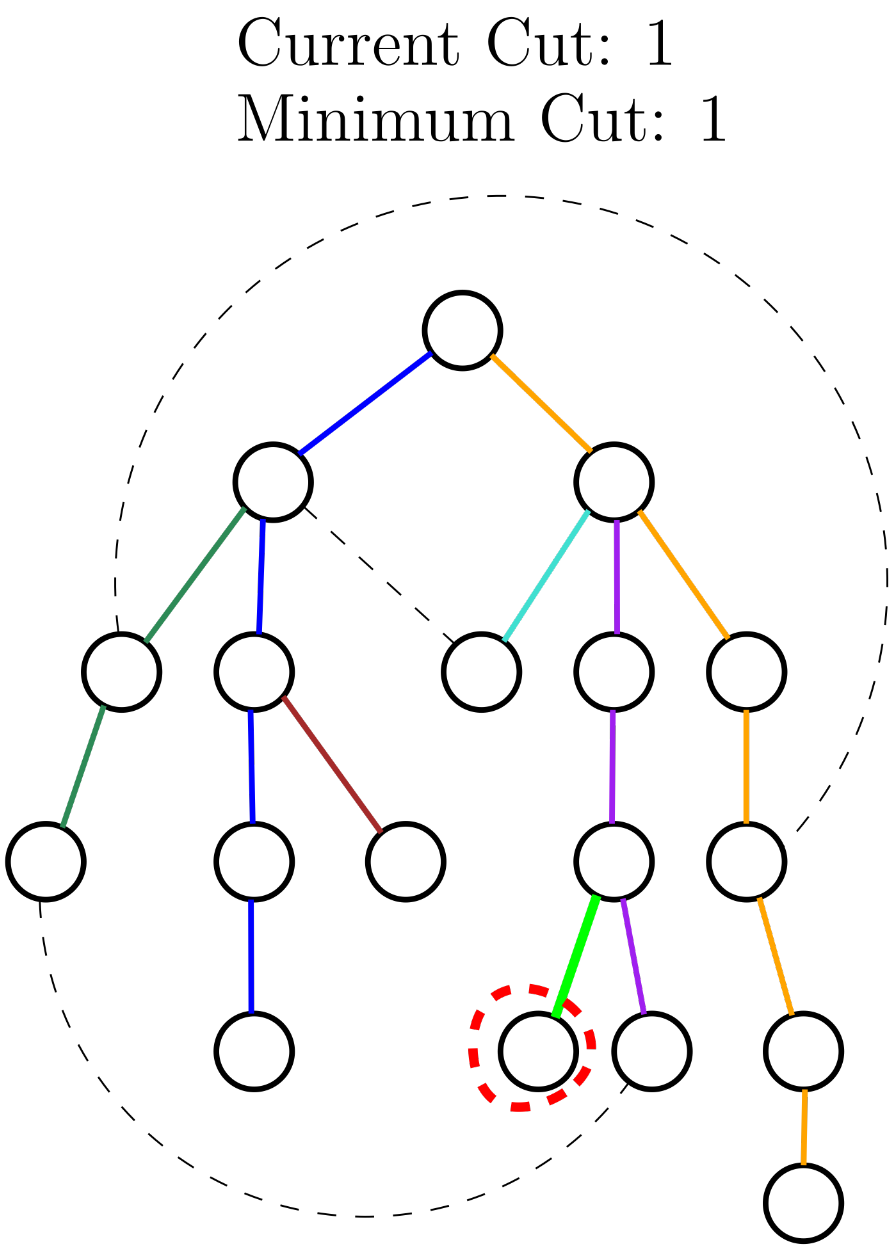


1-Respect Algorithm

1. Iterate edge e in heavy-light decomposition order.
2. Keep track of non-tree edges uv that cross a cut at e.

Non-tree edge uv will transition on or off the current cut $O(\log n)$ times.

$\Rightarrow O(m \log n)$ time.

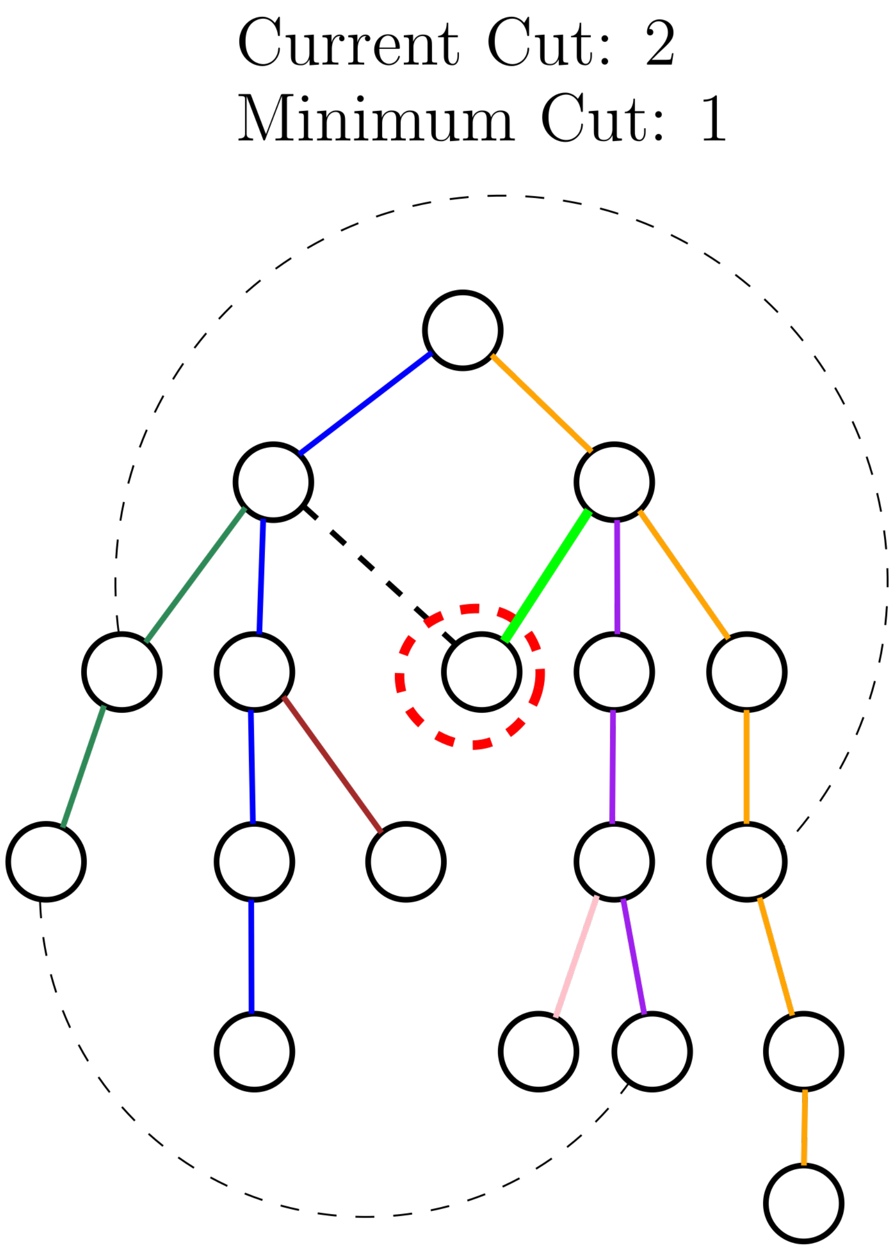


1-Respect Algorithm

1. Iterate edge e in heavy-light decomposition order.
2. Keep track of non-tree edges uv that cross a cut at e.

Non-tree edge uv will transition on or off the current cut $O(\log n)$ times.

$\Rightarrow O(m \log n)$ time.

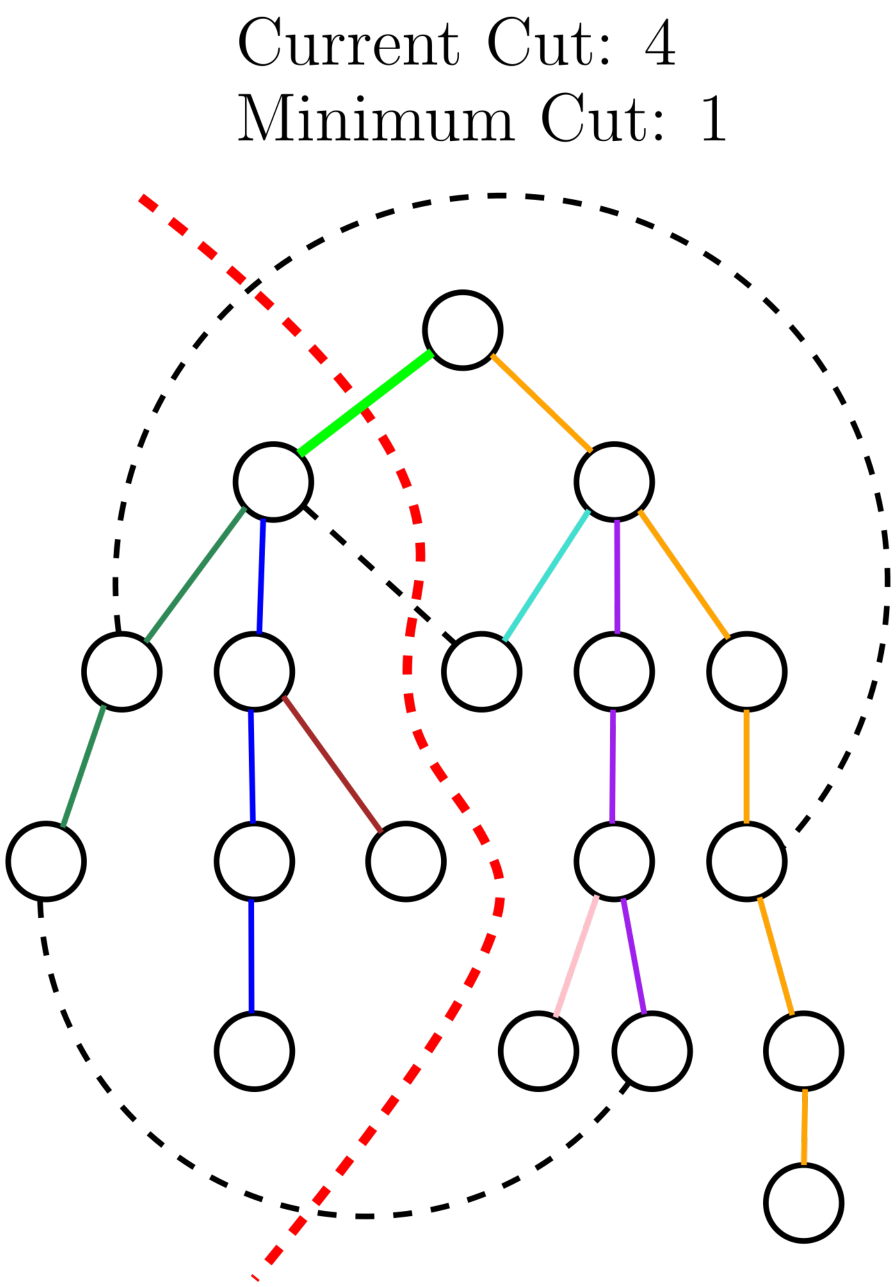


1-Respect Algorithm

1. Iterate edge e in heavy-light decomposition order.
2. Keep track of non-tree edges uv that cross a cut at e.

Non-tree edge uv will transition on or off the current cut $O(\log n)$ times.

$\Rightarrow O(m \log n)$ time.

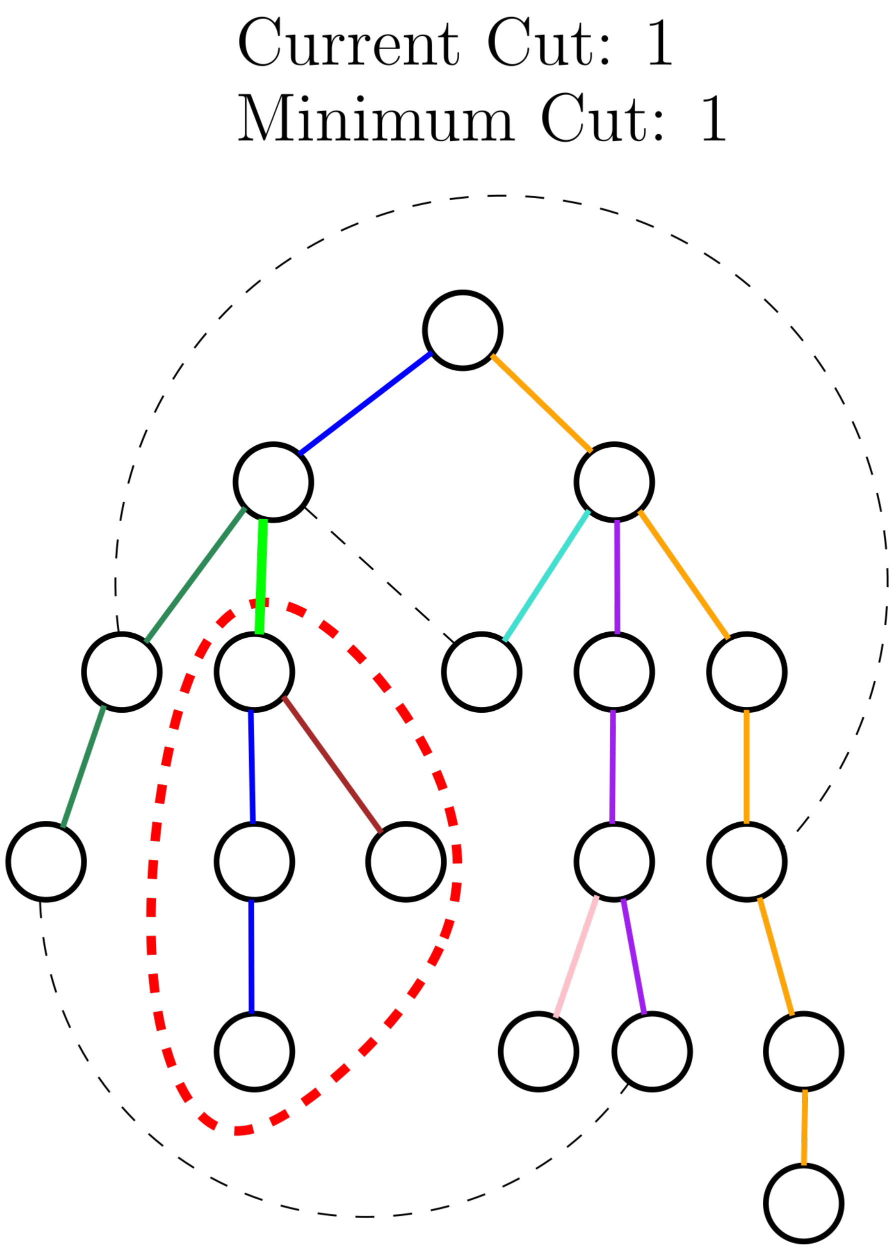


1-Respect Algorithm

1. Iterate edge e in heavy-light decomposition order.
2. Keep track of non-tree edges uv that cross a cut at e.

Non-tree edge uv will transition on or off the current cut $O(\log n)$ times.

$\Rightarrow O(m \log n)$ time.

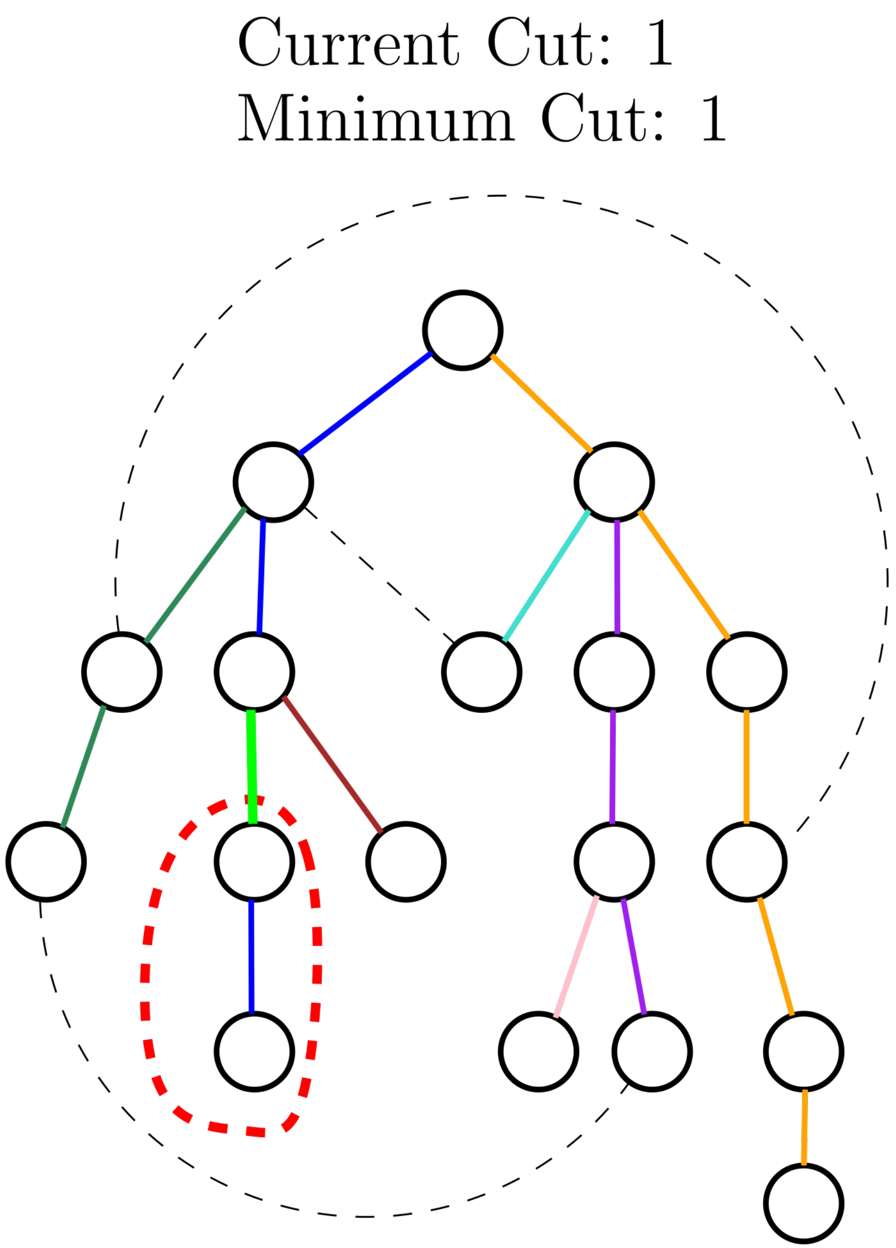


1-Respect Algorithm

1. Iterate edge e in heavy-light decomposition order.
2. Keep track of non-tree edges uv that cross a cut at e.

Non-tree edge uv will transition on or off the current cut $O(\log n)$ times.

$\Rightarrow O(m \log n)$ time.

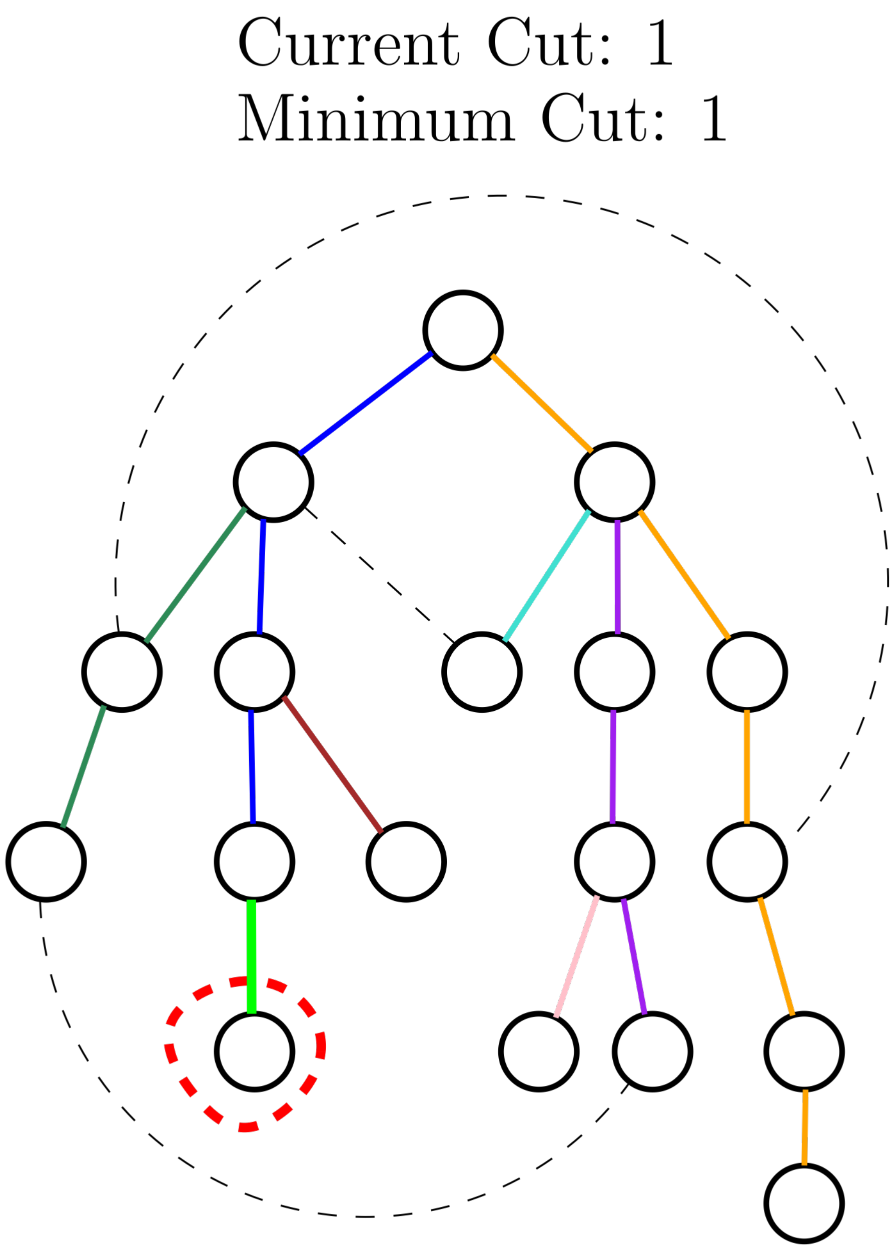


1-Respect Algorithm

1. Iterate edge e in heavy-light decomposition order.
2. Keep track of non-tree edges uv that cross a cut at e.

Non-tree edge uv will transition on or off the current cut $O(\log n)$ times.

$\Rightarrow O(m \log n)$ time.

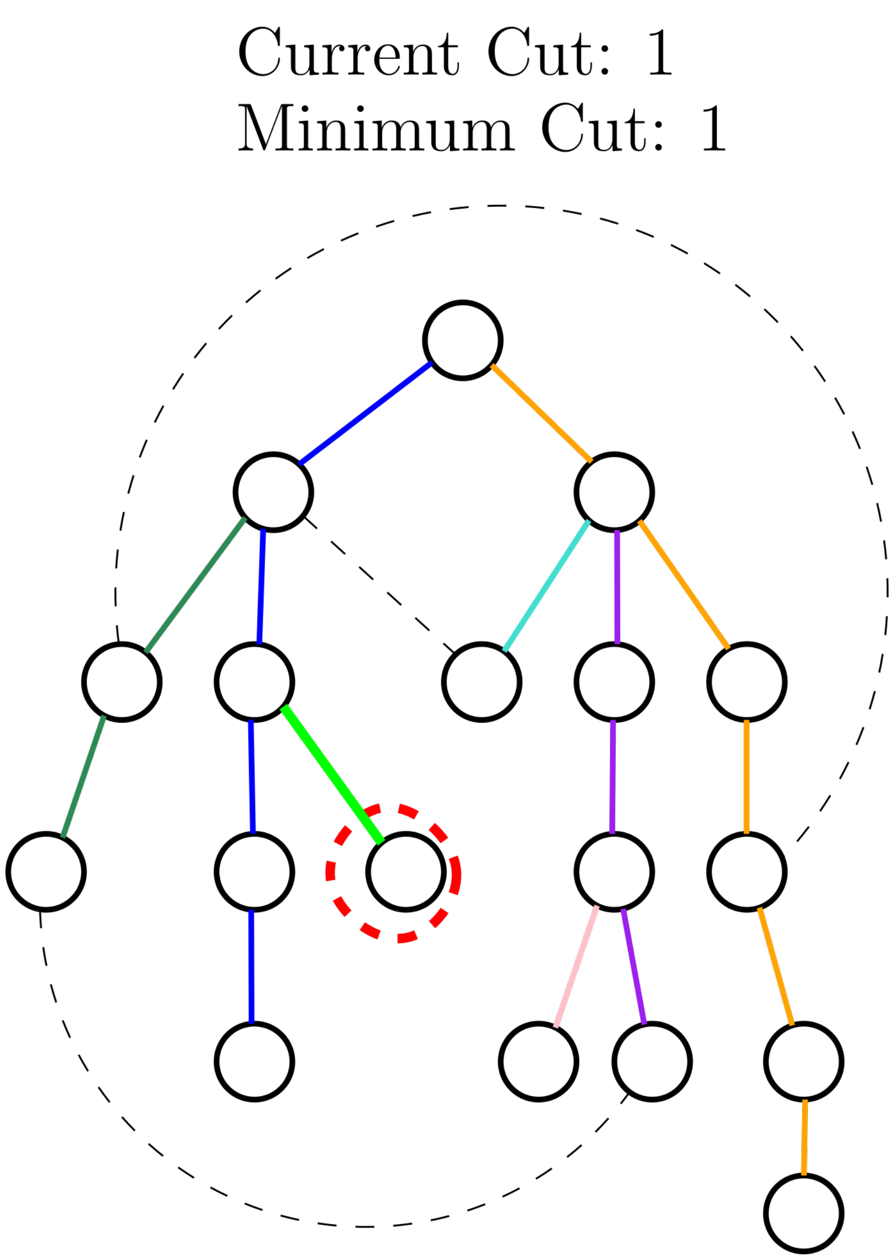


1-Respect Algorithm

1. Iterate edge e in heavy-light decomposition order.
2. Keep track of non-tree edges uv that cross a cut at e.

Non-tree edge uv will transition on or off the current cut $O(\log n)$ times.

$\Rightarrow O(m \log n)$ time.

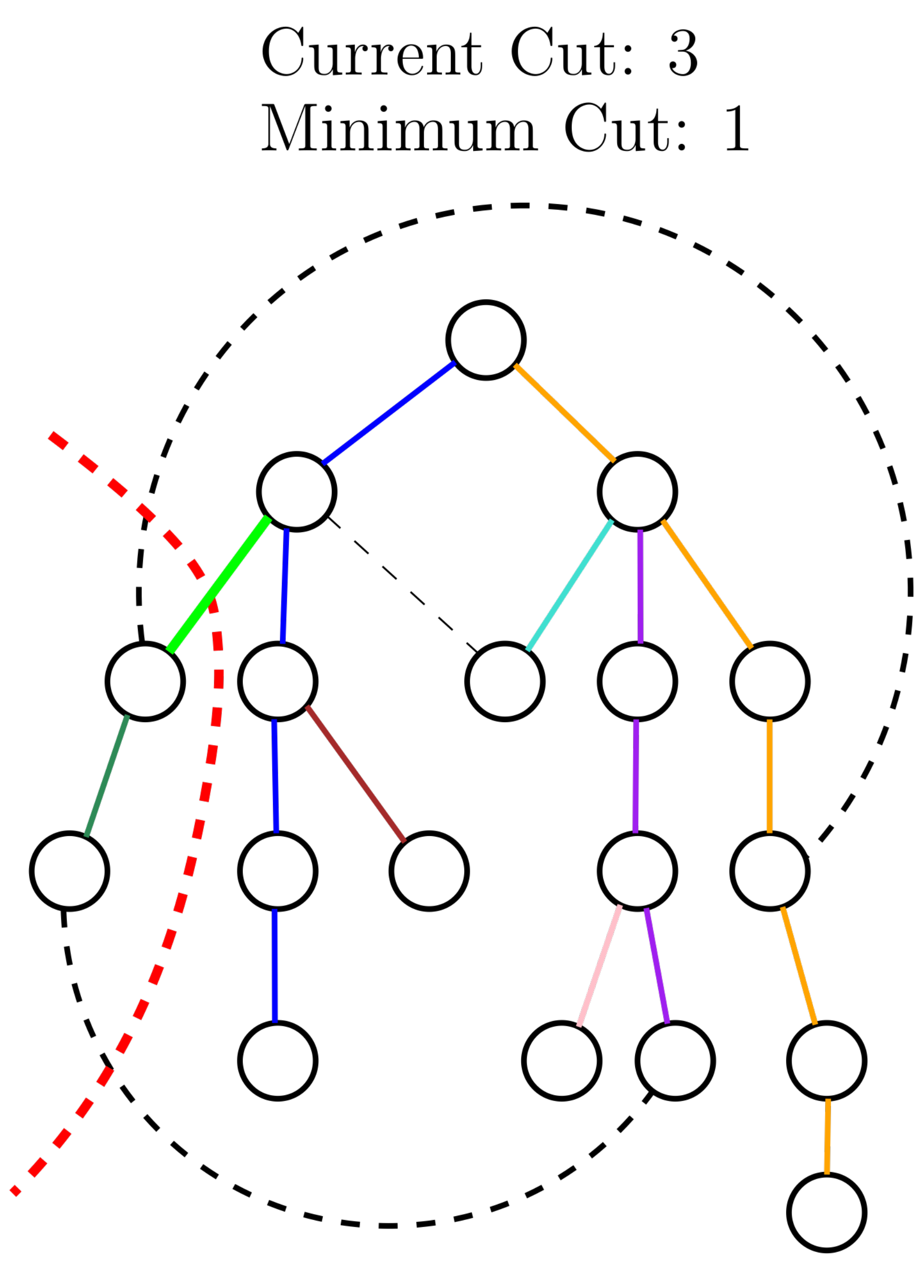


1-Respect Algorithm

1. Iterate edge e in heavy-light decomposition order.
2. Keep track of non-tree edges uv that cross a cut at e.

Non-tree edge uv will transition on or off the current cut $O(\log n)$ times.

$\Rightarrow O(m \log n)$ time.

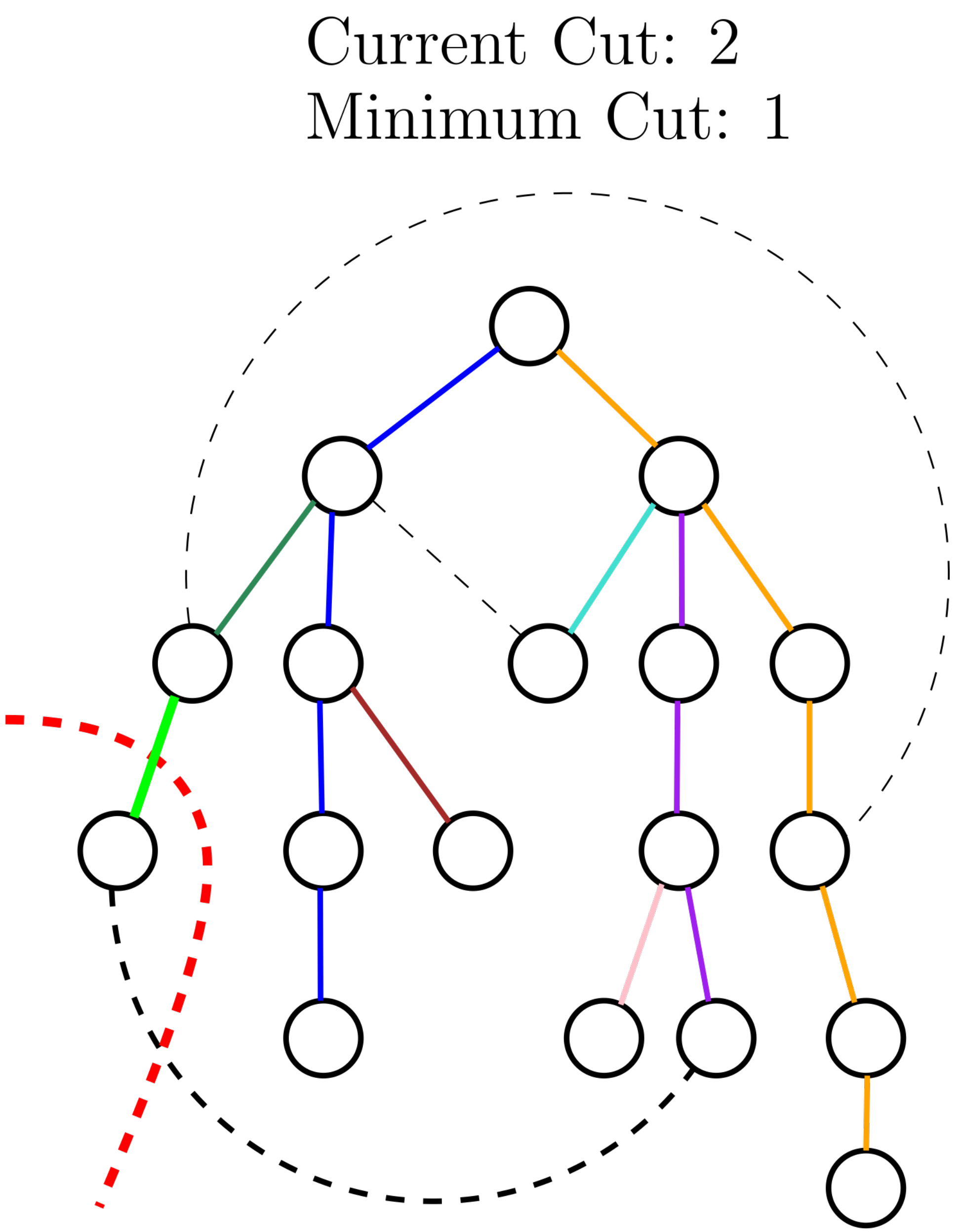


1-Respect Algorithm

1. Iterate edge e in heavy-light decomposition order.
2. Keep track of non-tree edges uv that cross a cut at e.

Non-tree edge uv will transition on or off the current cut $O(\log n)$ times.

$\Rightarrow O(m \log n)$ time.

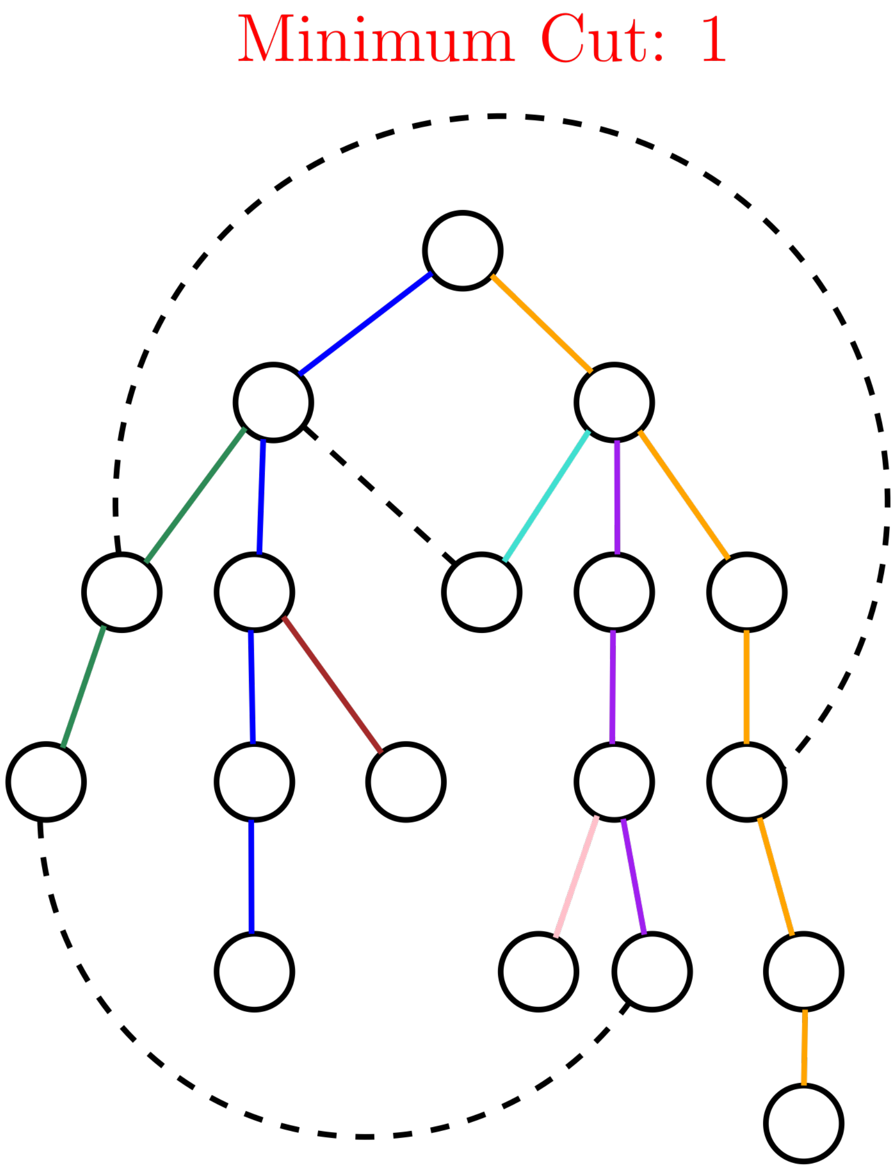


1-Respect Algorithm

1. Iterate edge e in heavy-light decomposition order.
2. Keep track of non-tree edges uv that cross a cut at e.

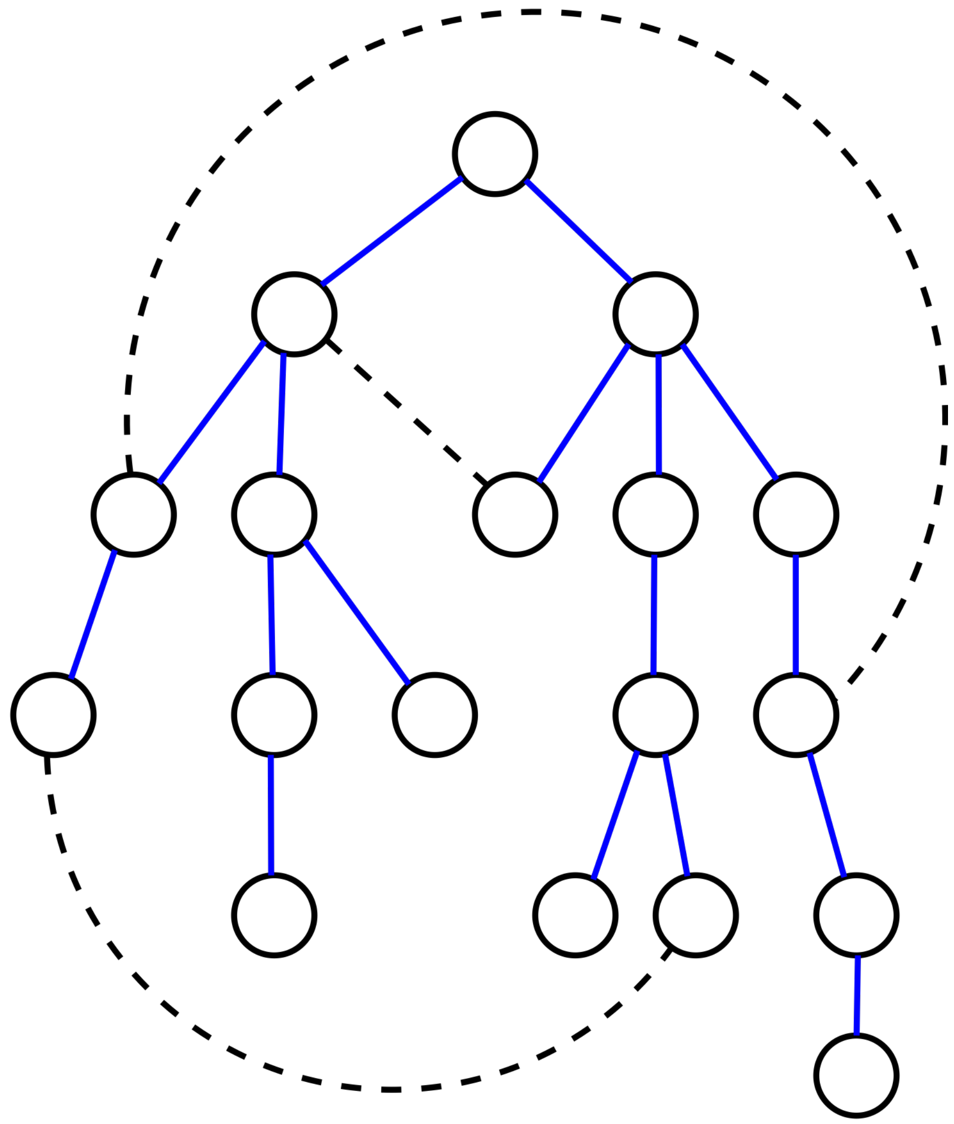
Non-tree edge uv will transition on or off the current cut $O(\log n)$ times.

$\Rightarrow O(m \log n)$ time.



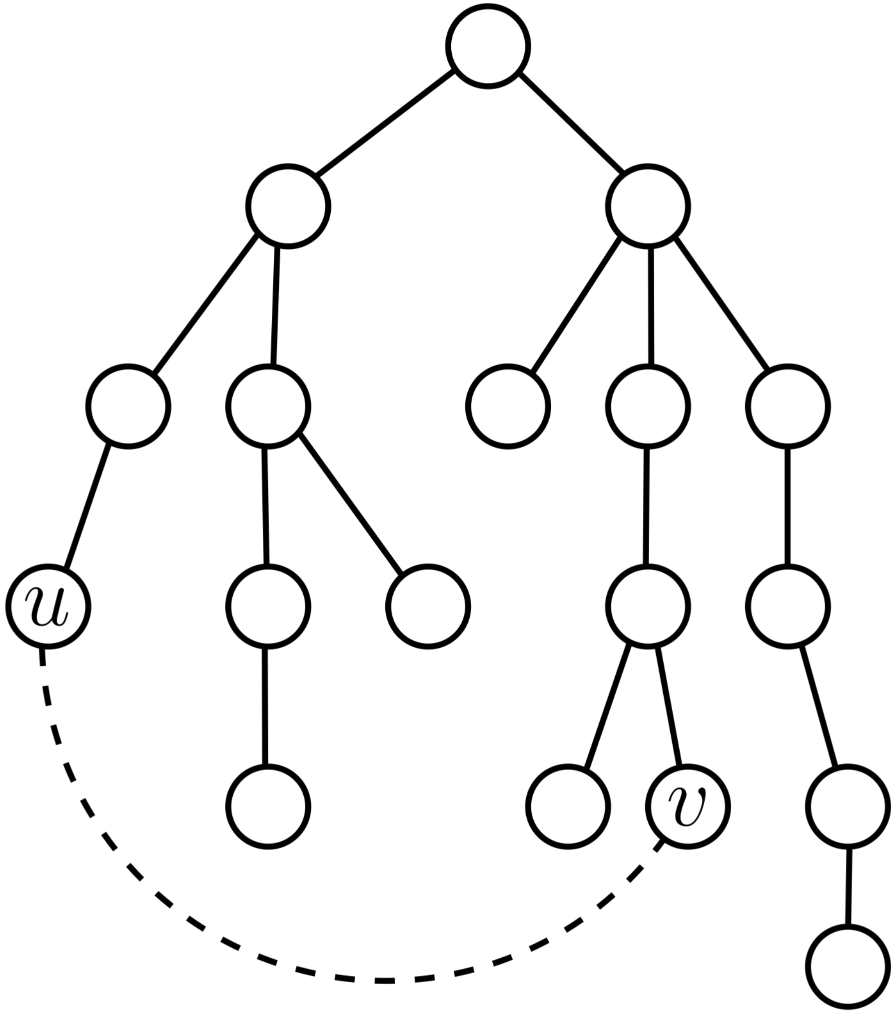
2-Respect Algorithm

Given spanning tree T of a graph G, find a smallest cut of G that cuts two edges of T.



2-Respect Algorithm

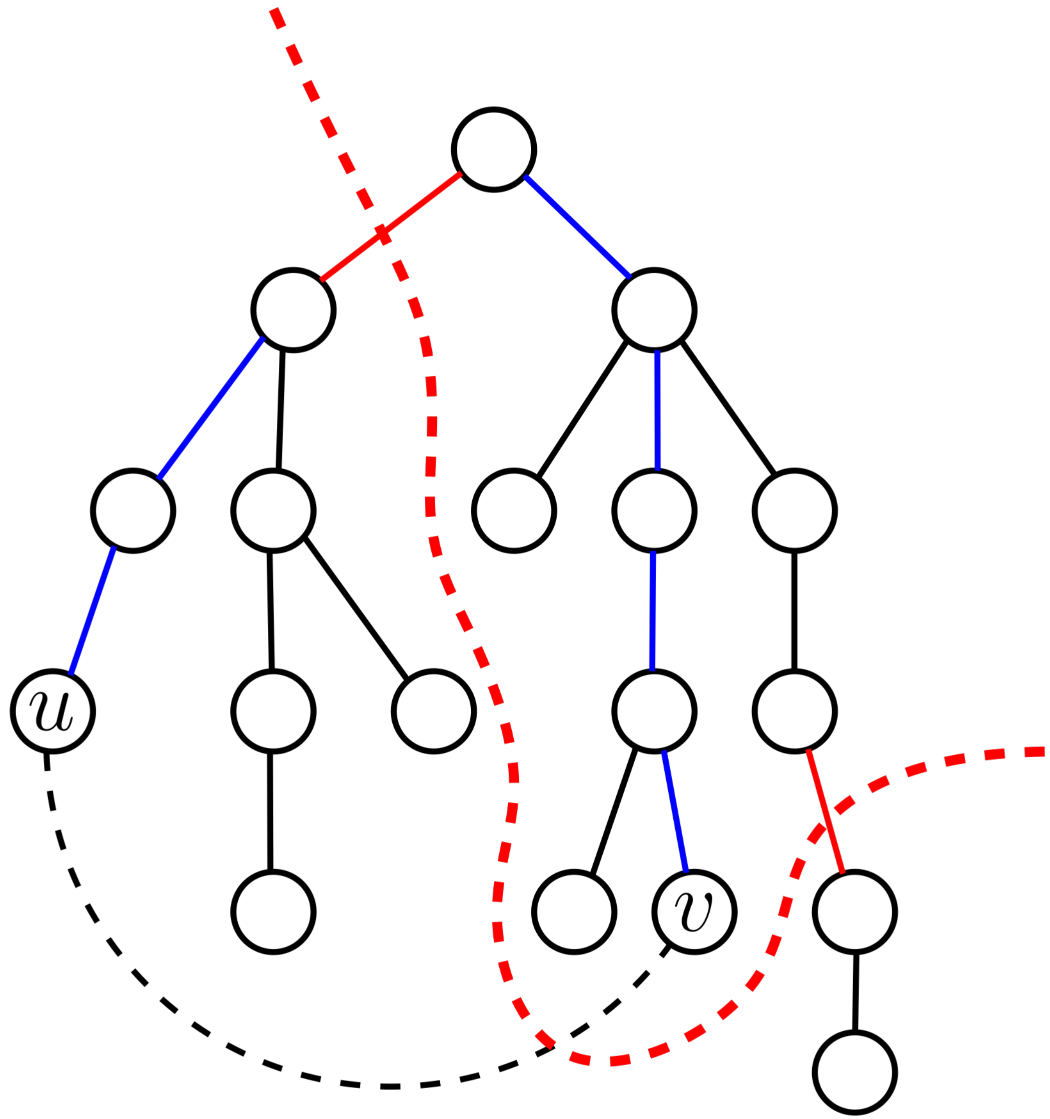
When two edges of T are cut, when does a non-tree edge uv cross the cut?



2-Respect Algorithm

When two edges of T are cut, when does a non-tree edge uv cross the cut?

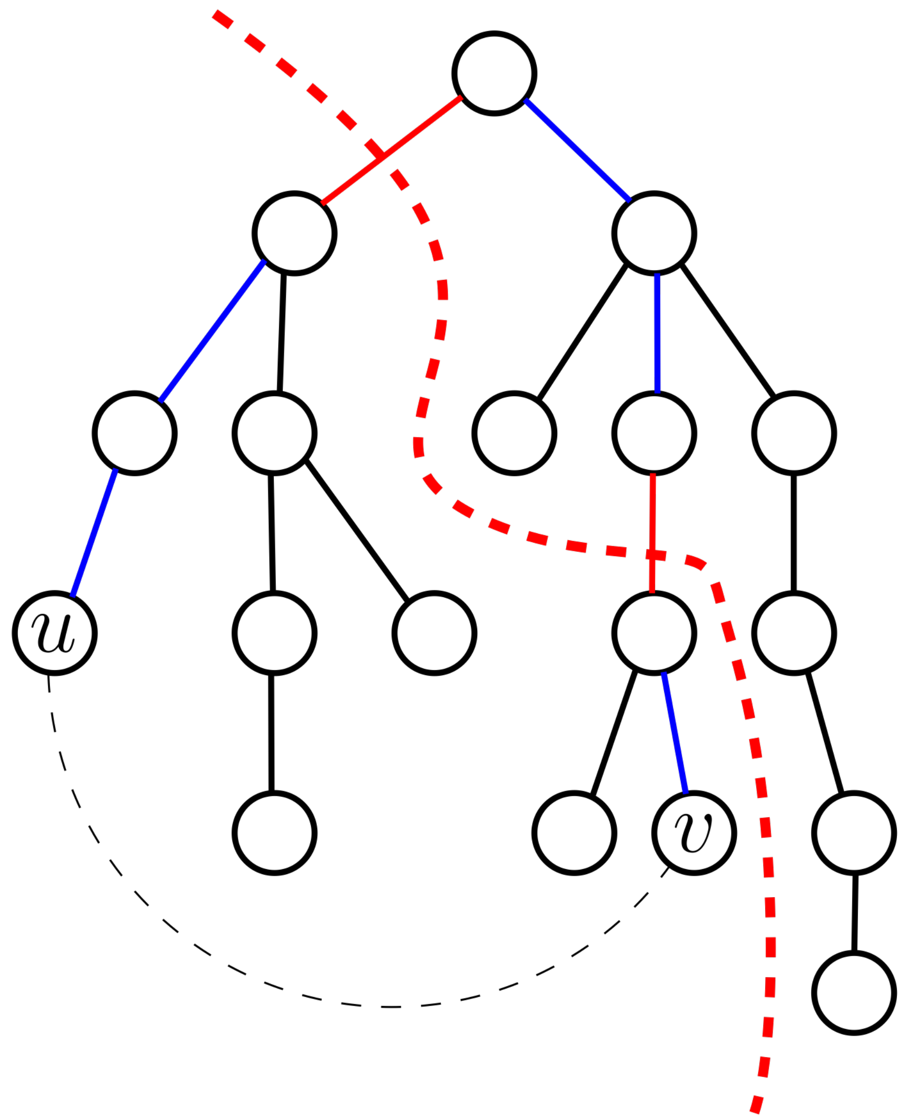
Non-tree edge uv is cut iff the cut in G cuts exactly one edge on the uv-path in T.



2-Respect Algorithm

When two edges of T are cut, when does a non-tree edge uv cross the cut?

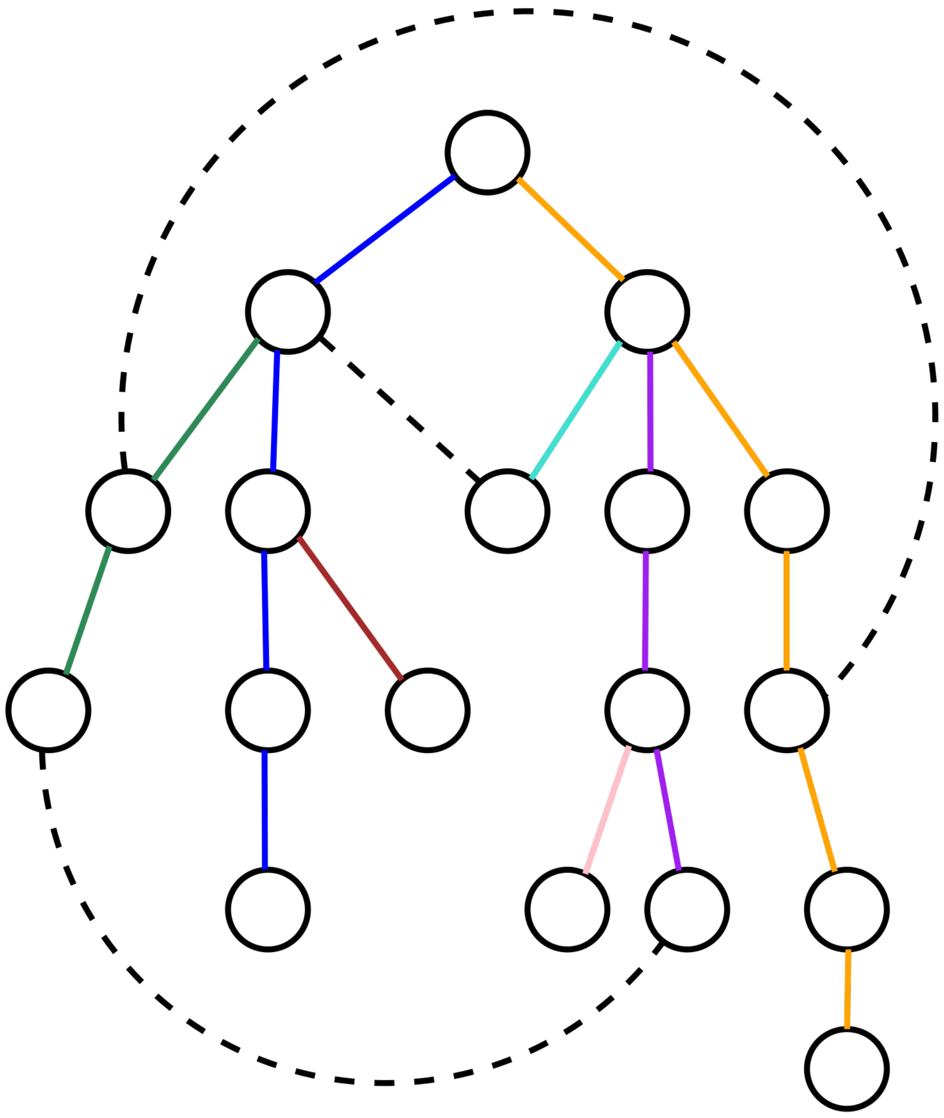
Non-tree edge uv is cut iff the cut in G cuts exactly one edge on the uv-path in T.



2-Respect Algorithm

How can we leverage our 1-respect strategy for cuts that cut two edges of T?

We cannot spend $\Omega(n^2)$ time checking all cuts.

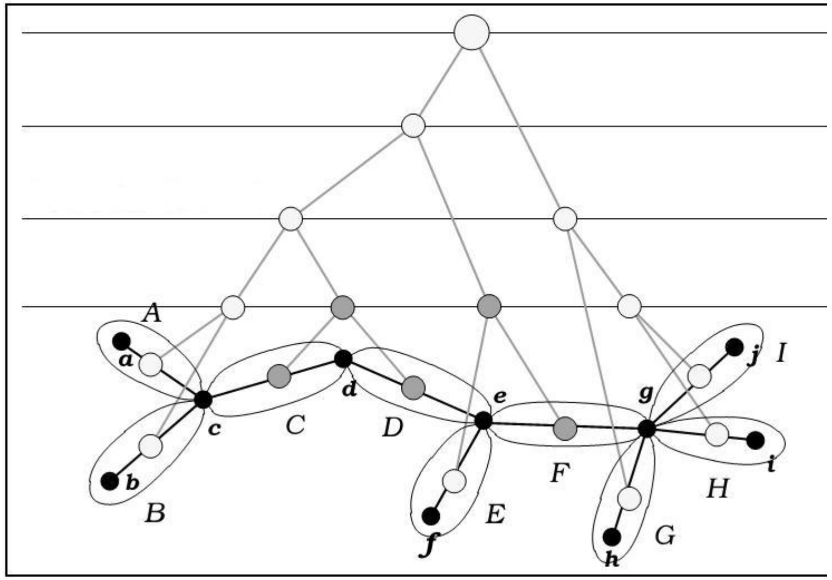


Top Tree Data Structure

Operations over a weighted tree T:

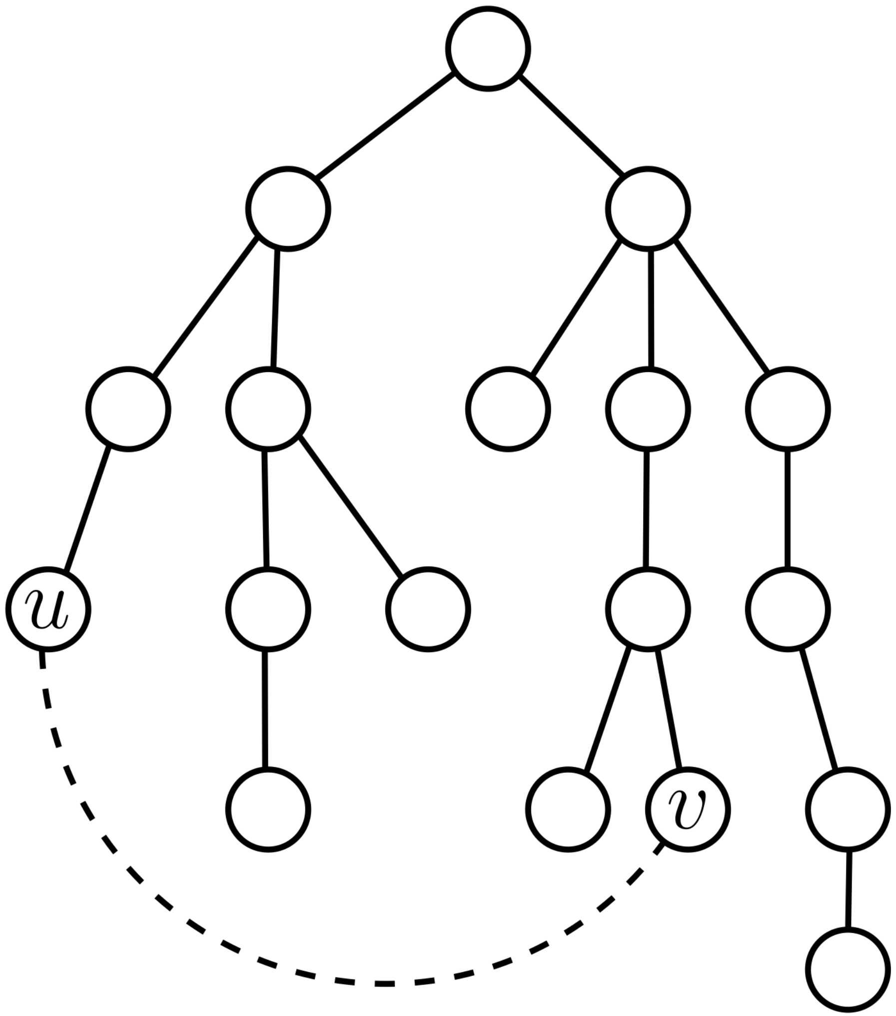
- $PathAdd(u, v, w)$:= add weight w to all edges on the uv-path in T.
- $NonPathAdd(u, v, w)$:= add weight w to all edges not on the uv-path in T.
- $QueryMinimum()$:= Return the minimum weight edge in T.

All operations take $O(\log n)$ time.



Idea

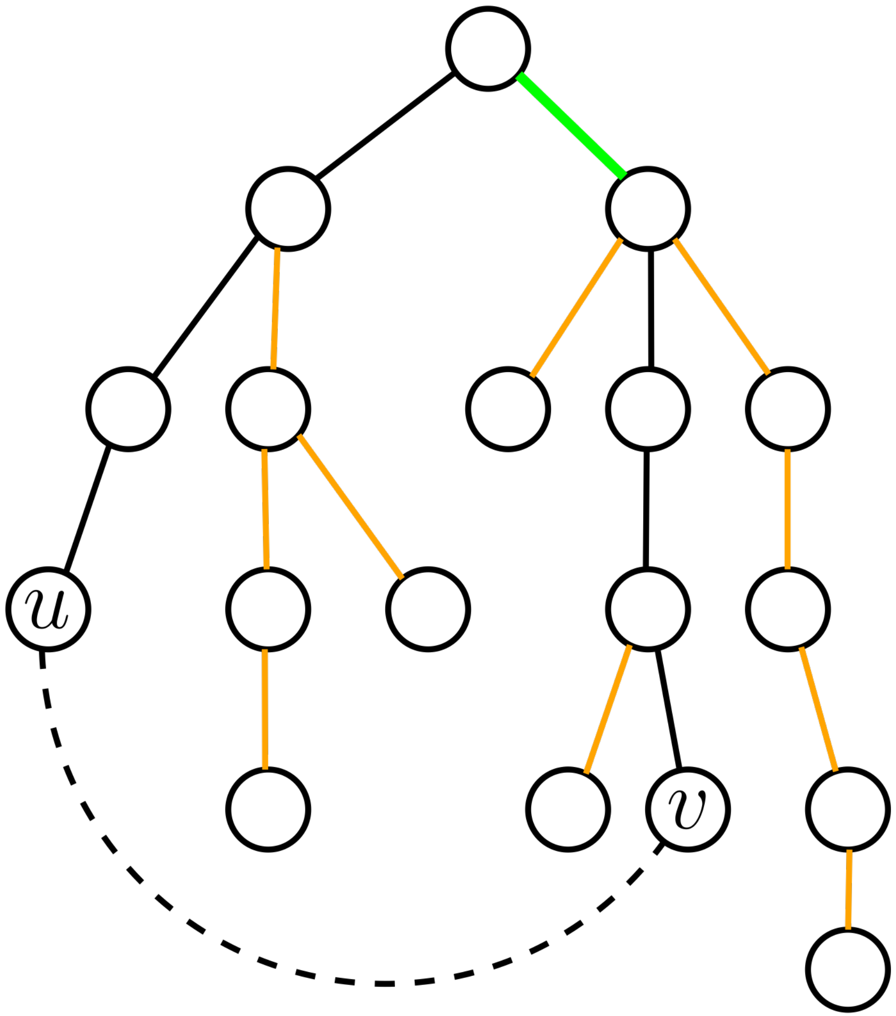
Call the two tree edges that we cut e and f. If we fix e, we can determine which f result in non-tree edge uv crossing the cut.



Idea

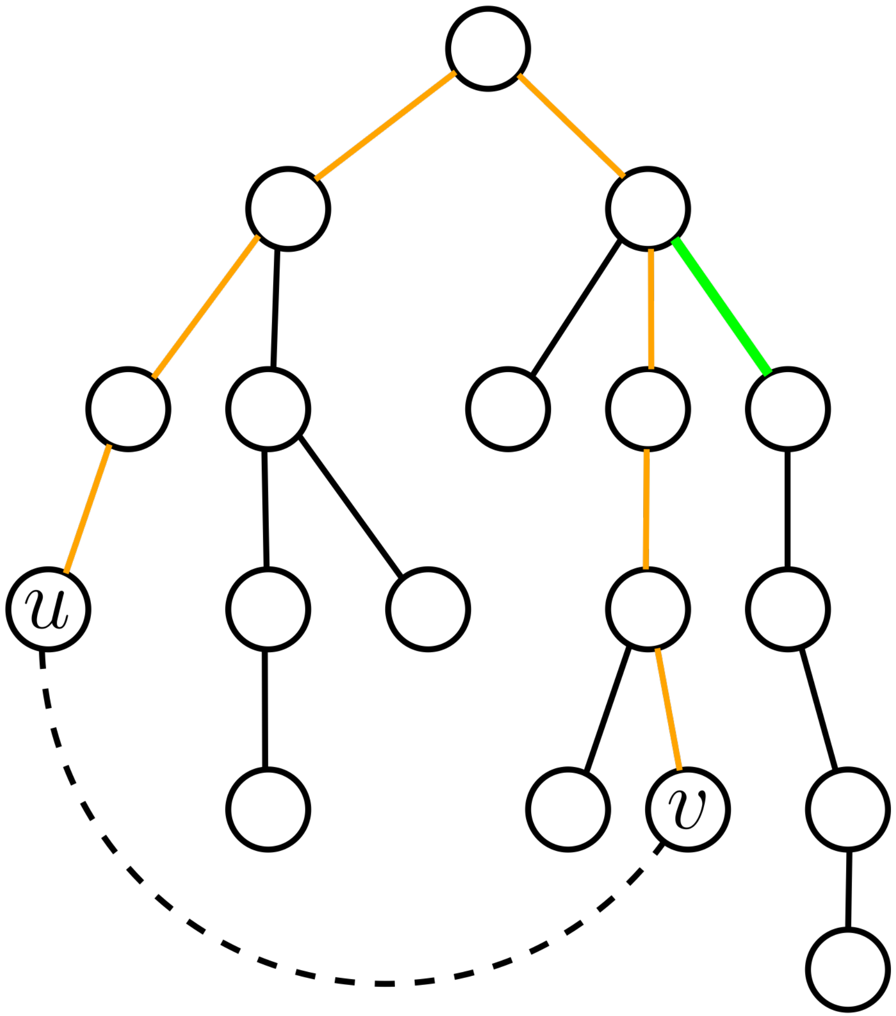
Call the two tree edges that we cut e and f. If we fix e, we can determine which f result in non-tree edge uv crossing the cut.

- If e is on the uv-path, any f off the uv-path cut uv.



Idea

Call the two tree edges that we cut e and f. If we fix e, we can determine which f result in non-tree edge uv crossing the cut.



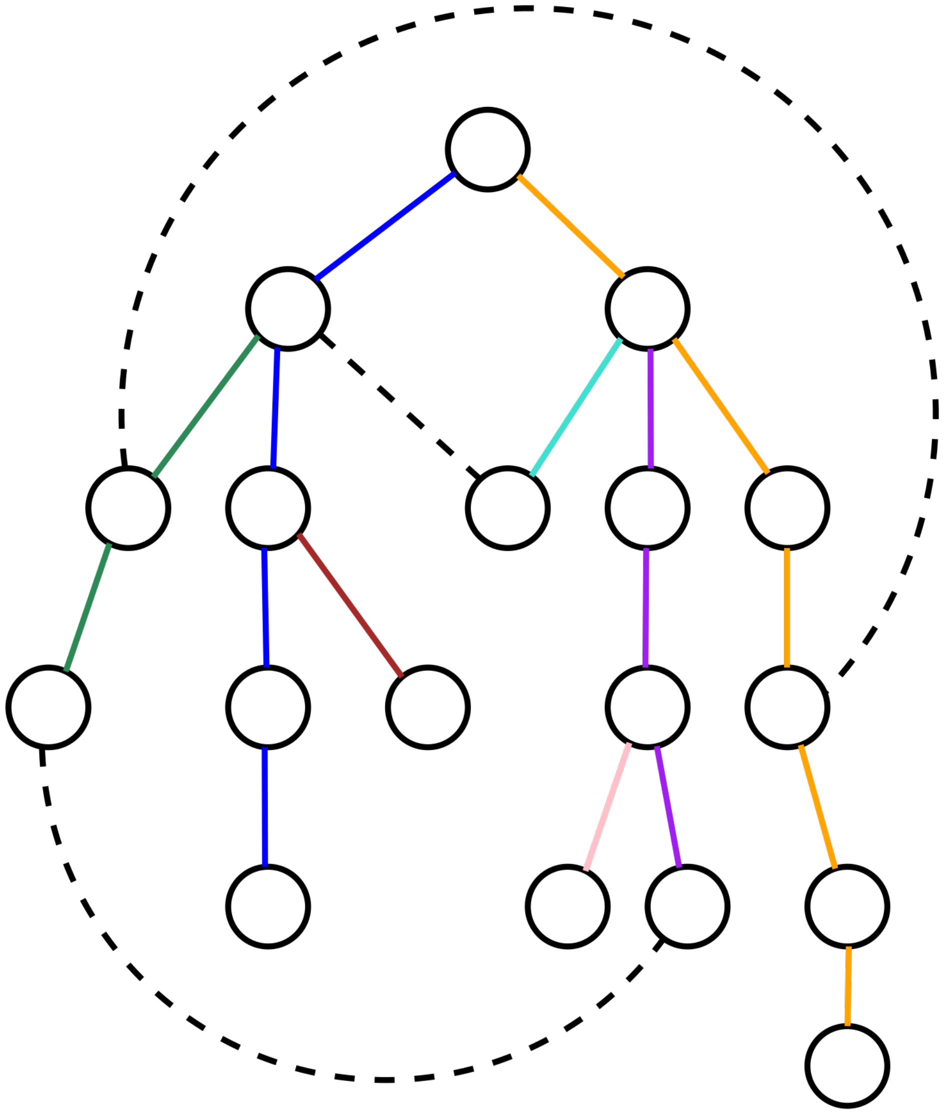
- If e is on the uv-path, any f off the uv-path cut uv.
- If e is off the uv-path, any f on the uv-path cut uv.

Use top tree to find best f!

2-Respect Algorithm

1. Iterate fixed tree-edge e in heavy-light decomposition order.
2. Keep track of the cost of cutting any other edge f in a top tree.
3. After e is moved and the top tree updated, query for best f.

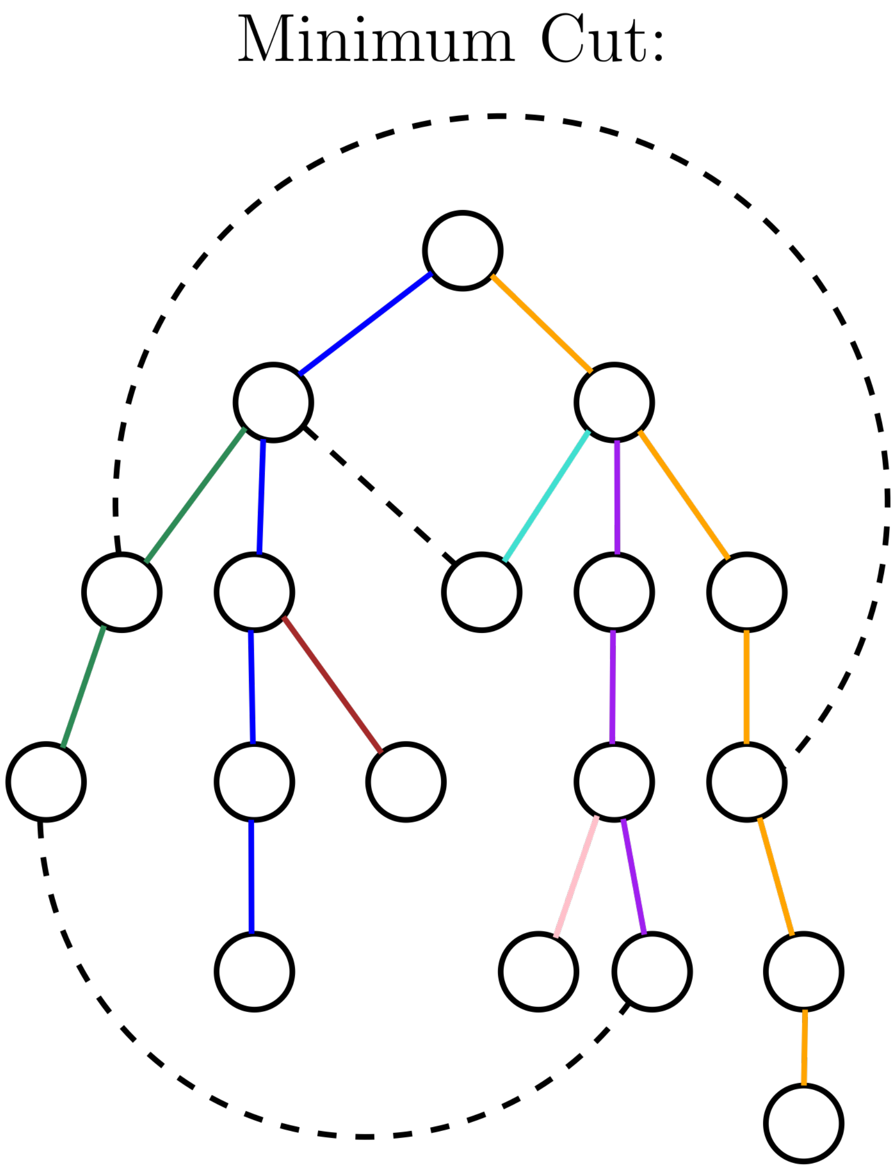
$\Rightarrow O(m \log^2 n)$ time.



2-Respect Algorithm

1. Iterate fixed tree-edge e in heavy-light decomposition order.
2. Keep track of the cost of cutting any other edge f in a top tree.
3. After e is moved and the top tree updated, query for best f.

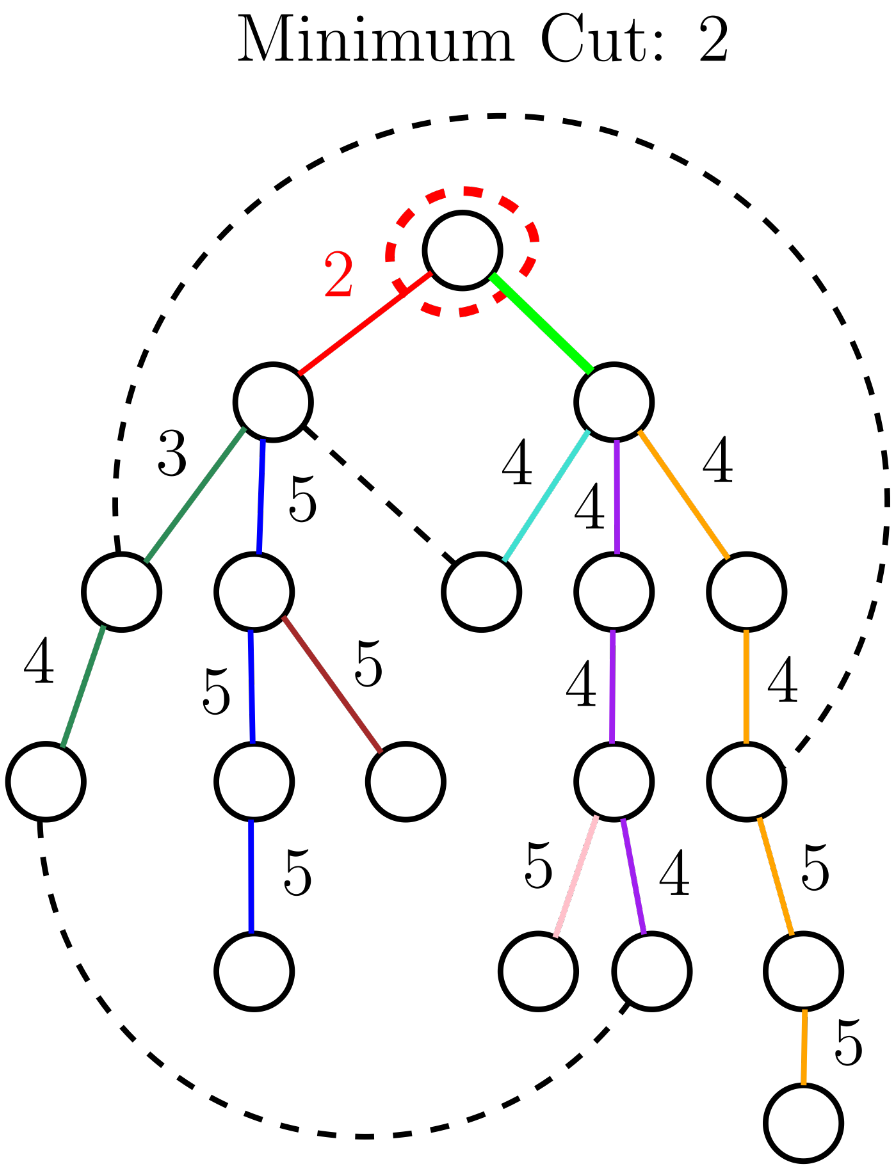
$\Rightarrow O(m \log^2 n)$ time.



2-Respect Algorithm

1. Iterate fixed tree-edge e in heavy-light decomposition order.
2. Keep track of the cost of cutting any other edge f in a top tree.
3. After e is moved and the top tree updated, query for best f.

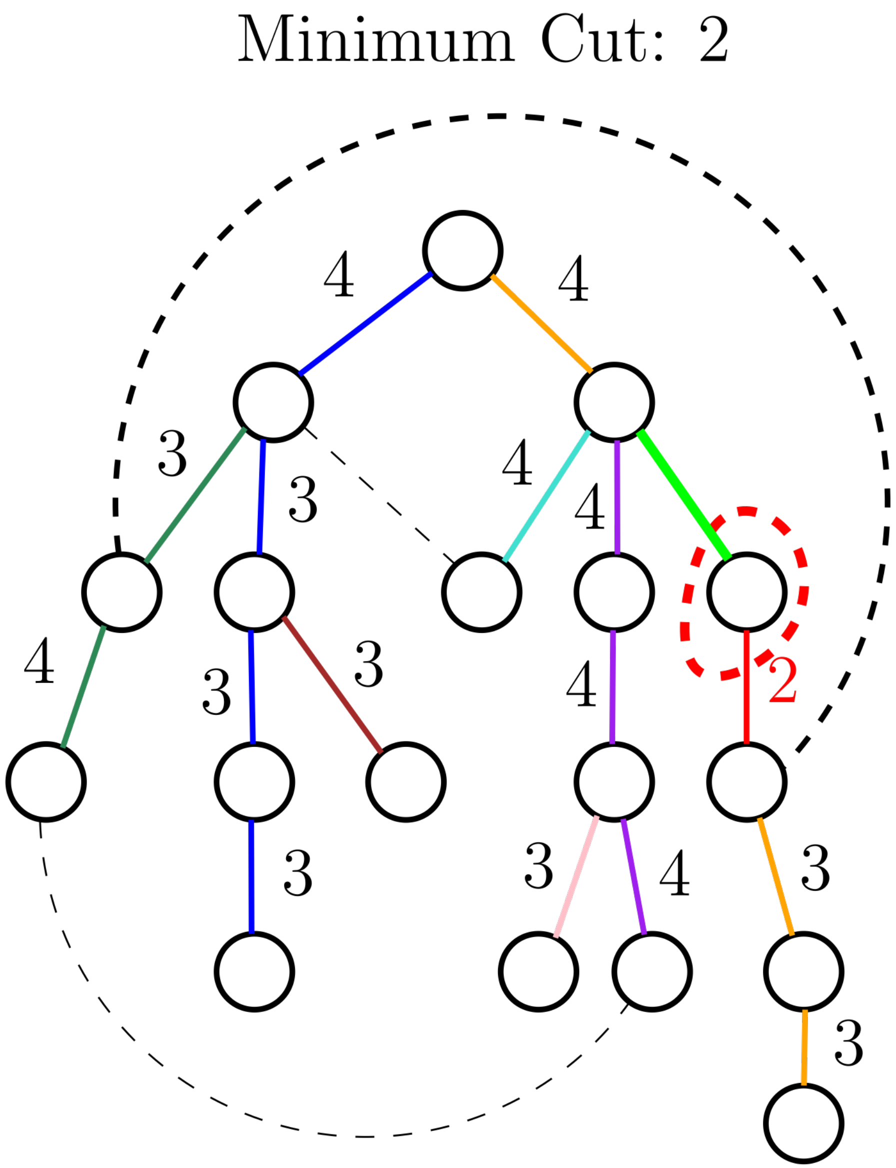
$\Rightarrow O(m \log^2 n)$ time.



2-Respect Algorithm

1. Iterate fixed tree-edge e in heavy-light decomposition order.
2. Keep track of the cost of cutting any other edge f in a top tree.
3. After e is moved and the top tree updated, query for best f.

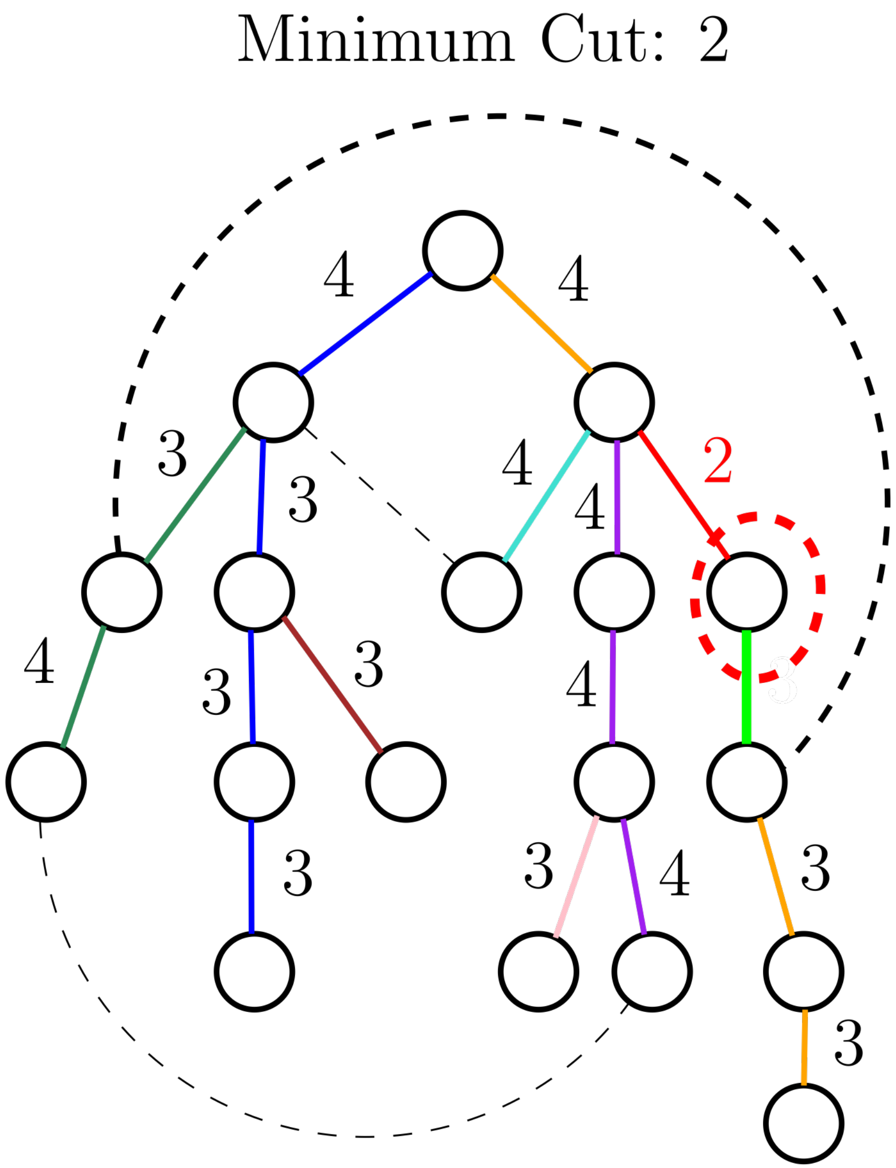
$\Rightarrow O(m \log^2 n)$ time.



2-Respect Algorithm

1. Iterate fixed tree-edge e in heavy-light decomposition order.
2. Keep track of the cost of cutting any other edge f in a top tree.
3. After e is moved and the top tree updated, query for best f.

$\Rightarrow O(m \log^2 n)$ time.

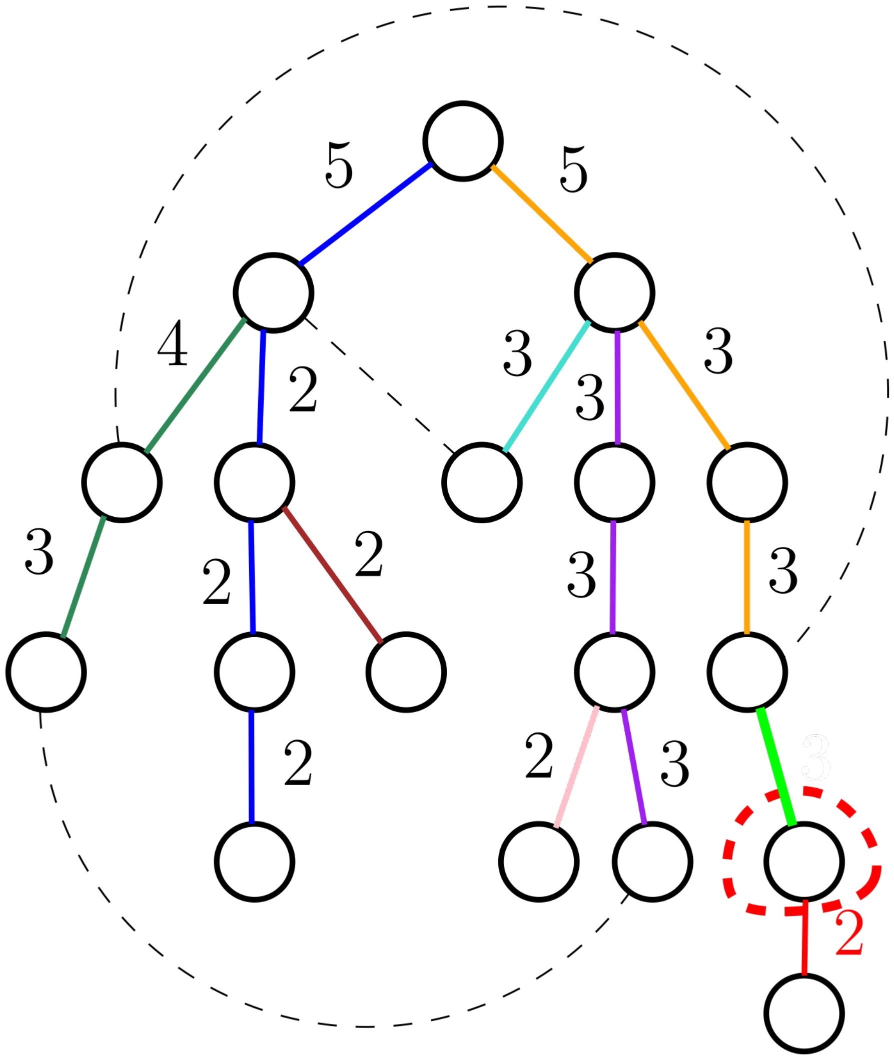


2-Respect Algorithm

1. Iterate fixed tree-edge e in heavy-light decomposition order.
2. Keep track of the cost of cutting any other edge f in a top tree.
3. After e is moved and the top tree updated, query for best f.

$\Rightarrow O(m \log^2 n)$ time.

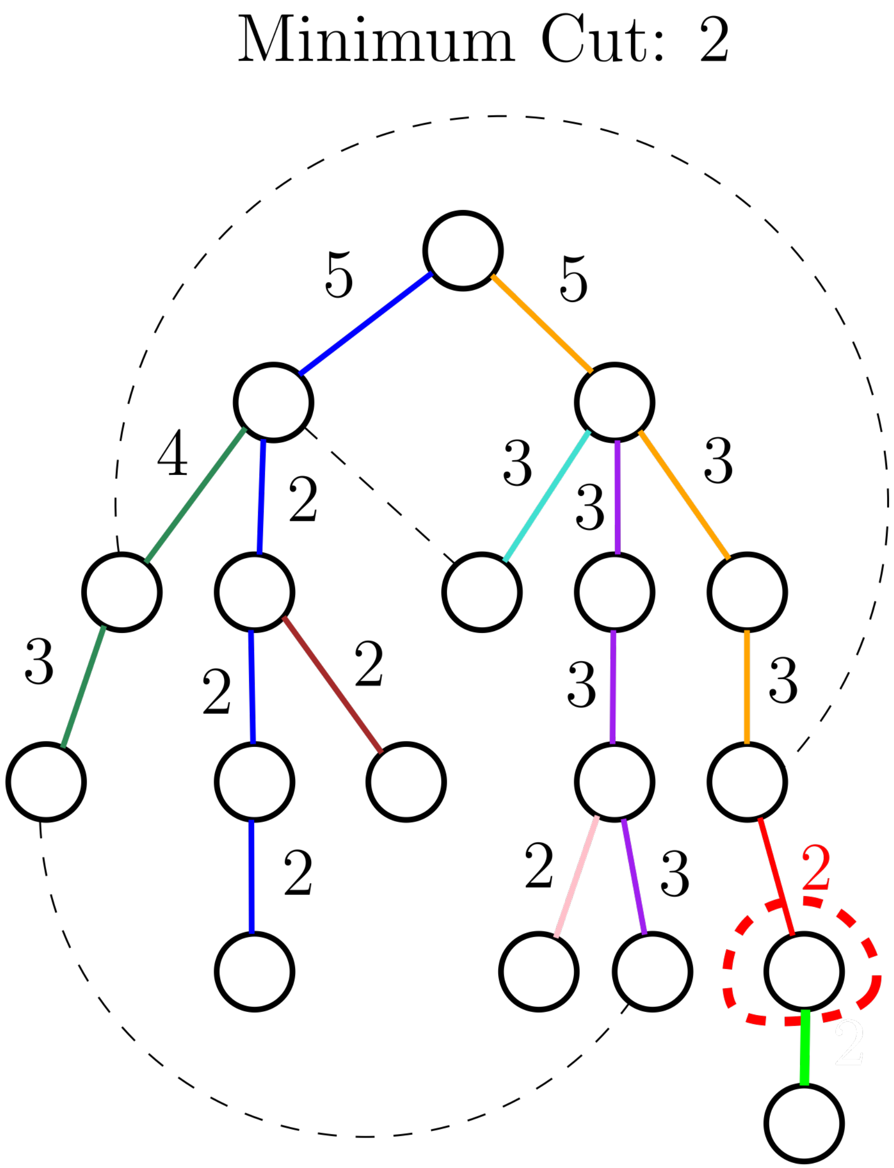
Minimum Cut: 2



2-Respect Algorithm

1. Iterate fixed tree-edge e in heavy-light decomposition order.
2. Keep track of the cost of cutting any other edge f in a top tree.
3. After e is moved and the top tree updated, query for best f.

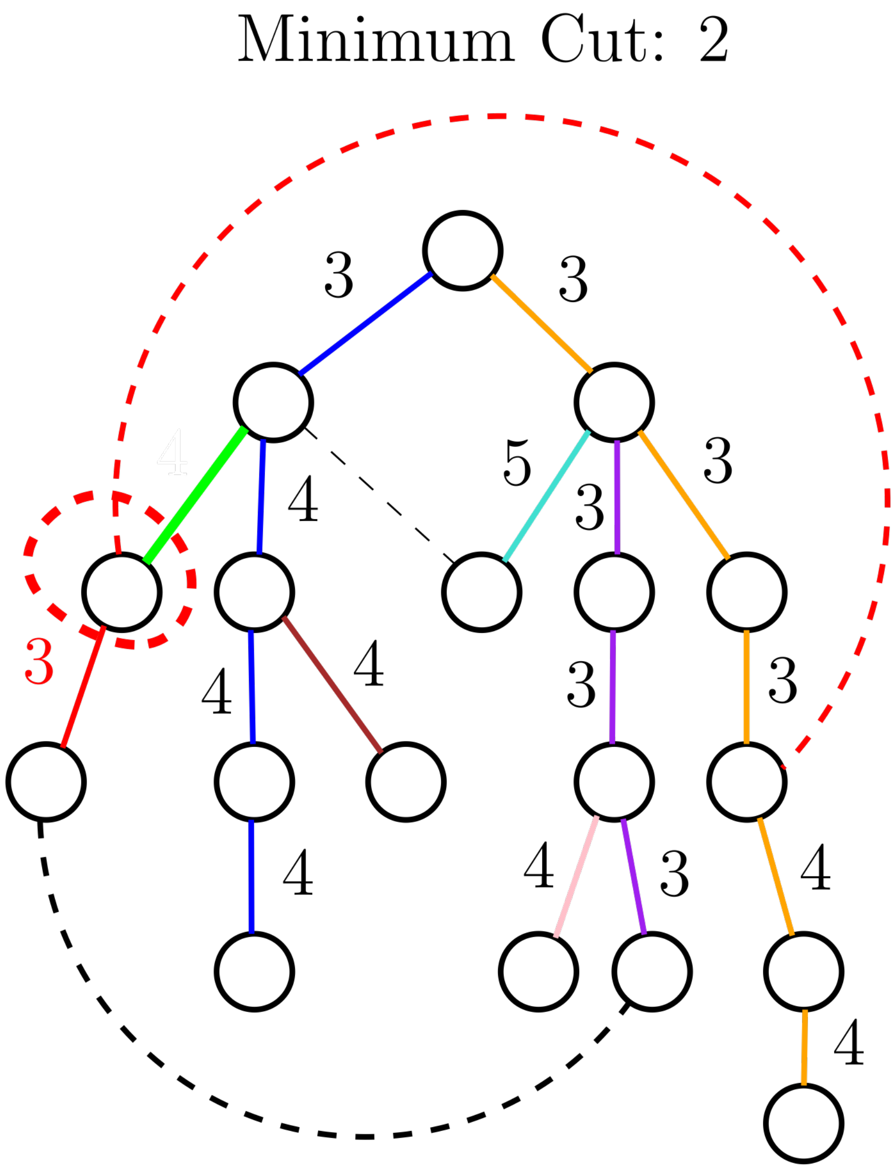
$\Rightarrow O(m \log^2 n)$ time.



2-Respect Algorithm

1. Iterate fixed tree-edge e in heavy-light decomposition order.
2. Keep track of the cost of cutting any other edge f in a top tree.
3. After e is moved and the top tree updated, query for best f.

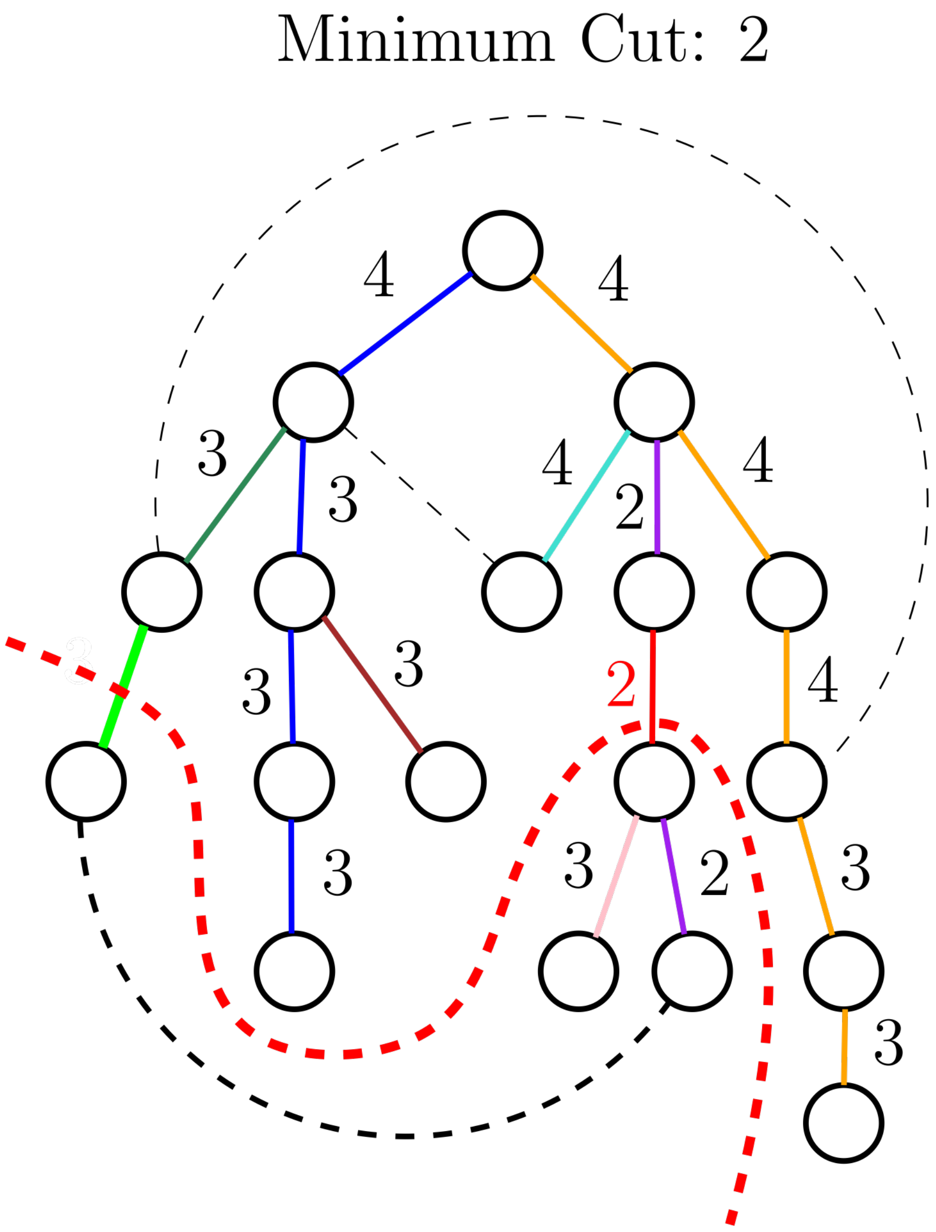
$\Rightarrow O(m \log^2 n)$ time.



2-Respect Algorithm

1. Iterate fixed tree-edge e in heavy-light decomposition order.
2. Keep track of the cost of cutting any other edge f in a top tree.
3. After e is moved and the top tree updated, query for best f.

$\Rightarrow O(m \log^2 n)$ time.



Implementation

- Available at: https://github.com/nalinbhardwaj/min-cut-paper.
- About ~200 lines of code for the 2-respect algorithm.

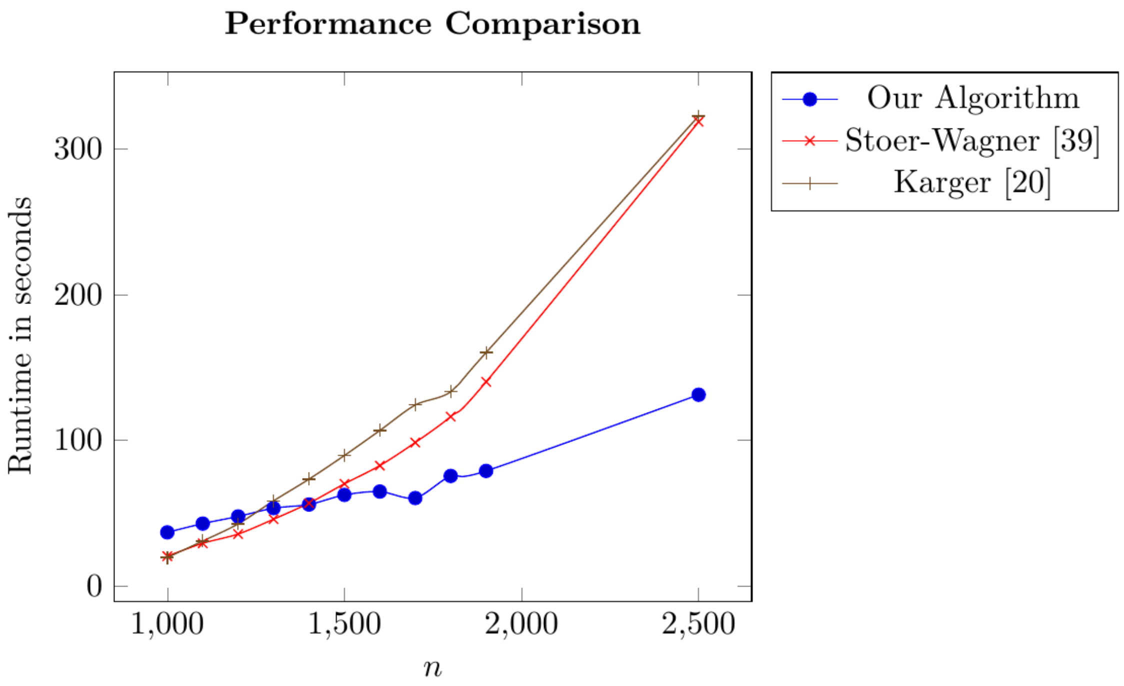


Figure 1 Performance comparison of an $O(m \log^4 n)$ implementation of our algorithm with an $O(n^3)$ Stoer-Wagner [39] and $O(n^3 \log n)$ Karger [20].

Thanks!

Questions?