Binary Searching a Tree

Oren Weimann
MIT, CSAIL

Joint work with
Shay Mozes (Brown University)
Krzysztof Onak (MIT)
How old is Waldo?

How quickly can you learn Waldo’s age?
How old is Waldo?

How quickly can you learn Waldo’s age?

You can ask Waldo if he’s $x$ years old.
How old is Waldo?

How quickly can you learn Waldo’s age?

- You can ask Waldo if he’s $x$ years old.
- Possible answers:
  - “Yes, I’m $x$ years old.”
  - “No, I’m younger.”
  - “No, I’m older.”
How old is Waldo?

How quickly can you learn Waldo’s age?

- You can ask Waldo if he’s $x$ years old.
- Possible answers:
  - “Yes, I’m $x$ years old.”
  - “No, I’m younger.”
  - “No, I’m older.”

Waldo, are you 22?

17 18 19 20 21 22 23 24
How old is Waldo?

How quickly can you learn Waldo’s age?

- You can ask Waldo if he’s $x$ years old.
- Possible answers:
  - “Yes, I’m $x$ years old.”
  - “No, I’m younger.”
  - “No, I’m older.”

17  18  19  20  21  × × ×
How old is Waldo?

How quickly can you learn Waldo’s age?

- You can ask Waldo if he’s $x$ years old.
- Possible answers:
  - “Yes, I’m $x$ years old.”
  - “No, I’m younger.”
  - “No, I’m older.”

Waldo, are you 18?

17 18 19 20 21
How old is Waldo?

How quickly can you learn Waldo’s age?

- You can ask Waldo if he’s $x$ years old.
- Possible answers:
  - “Yes, I’m $x$ years old.”
  - “No, I’m younger.”
  - “No, I’m older.”

Possible answers:

- 19
- 20
- 21
- 3
- 4
How old is Waldo?

How quickly can you learn Waldo’s age?

- You can ask Waldo if he’s \( x \) years old.
- Possible answers:
  - “Yes, I’m \( x \) years old.”
  - “No, I’m younger.”
  - “No, I’m older.”

Waldo, are you 20?

✗ ✓ 19 20 21 ✗ ✗ ✗
How old is Waldo?

How quickly can you learn Waldo’s age?

- You can ask Waldo if he’s $x$ years old.
- Possible answers:
  - “Yes, I’m $x$ years old.”
  - “No, I’m younger.”
  - “No, I’m older.”

[Images and speech bubble indicating Waldo's response: No, I'm younger.]

[Options to select: 19, 45, 12, 3]
How old is Waldo?

How quickly can you learn Waldo’s age?

- You can ask Waldo if he’s $x$ years old.
- Possible answers:
  - “Yes, I’m $x$ years old.”
  - “No, I’m younger.”
  - “No, I’m older.”

You must be 19!
Binary search

- Optimal solution:
  - Always ask about the number in the middle of the range of potential solutions.
Binary search

Optimal solution:

- Always ask about the number in the middle of the range of potential solutions.
- $\lceil \log_2 n \rceil$ questions in the worst case, where $n$ is the size of the range.
Optimal solution:

- Always ask about the number in the middle of the range of potential solutions.

$$\lfloor \log_2 n \rfloor$$ questions in the worst case, where \( n \) is the size of the range.

The searching problem is easy:
Binary search

Optimal solution:
- Always ask about the number in the middle of the range of potential solutions.
- \( \lceil \log_2 n \rceil \) questions in the worst case, where \( n \) is the size of the range.

The searching problem is easy:
- Only two “directions”: greater and smaller numbers.
Binary search

Optimal solution:
- Always ask about the number in the middle of the range of potential solutions.
- $\lceil \log_2 n \rceil$ questions in the worst case, where $n$ is the size of the range.

The searching problem is easy:
- Only two “directions”: greater and smaller numbers.
- Potential solutions constitute a totally ordered set.
Binary search

Optimal solution:
- Always ask about the number in the middle of the range of potential solutions.
- \( \lceil \log_2 n \rceil \) questions in the worst case, where \( n \) is the size of the range.

The searching problem is easy:
- Only two “directions”: greater and smaller numbers.
- Potential solutions constitute a totally ordered set.

But . . .
Binary search

- Optimal solution:
  - Always ask about the number in the middle of the range of potential solutions.
  - \( \lfloor \log_2 n \rfloor \) questions in the worst case, where \( n \) is the size of the range.

- The searching problem is easy:
  - Only two “directions”: greater and smaller numbers.
  - Potential solutions constitute a totally ordered set.

- But there is a greater challenge to face!
WHERE IS WALDO?
Searching in caves

- Waldo hides in a cave.
Searching in caves

- Waldo hides in a cave.
- The cave consists of chambers and corridors.
Waldo hides in a cave.

The cave consists of chambers and corridors.

The graph of the cave is a tree.
Searching in caves

- Waldo hides in a cave.
- The cave consists of chambers and corridors.
- The graph of the cave is a tree.
- Goal: Figure out which chamber Waldo is in.
Two query models

1. Questions about vertices
Two query models

1. Questions about vertices
   - Ask about a vertex-chamber $v$. 

[Diagram of a network with a question mark and a character]
Two query models

1. Questions about vertices
   - Ask about a vertex-chamber $v$.
   - Learn either that Waldo is in $v$, or which corridor outgoing from $v$ leads to Waldo.
Two query models

1. Questions about vertices
   - Ask about a vertex-chamber $v$.
   - Learn either that Waldo is in $v$, or which corridor outgoing from $v$ leads to Waldo.
Two query models

1. Questions about vertices
2. Questions about edges
Two query models

1. Questions about vertices
2. Questions about edges
   - Ask about an edge-corridor $e$. 
Two query models

1. Questions about vertices
2. Questions about edges
   - Ask about an edge-corridor $e$.
   - Learn which endpoint of $e$ is closer to Waldo.
Two query models

1. Questions about vertices
2. Questions about edges
   - Ask about an edge-corridor $e$.
   - Learn which endpoint of $e$ is closer to Waldo.
Searching in partial orders

Given is a partial order $S$ (or its diagram).
Searching in partial orders

- Given is a partial order $S$ (or its diagram).
- Waldo secretly chooses $x \in S$. 

\[ x = e \]
Searching in partial orders

- Given is a partial order $S$ (or its diagram).
- Waldo secretly chooses $x \in S$.
- Goal: Find out $x$ by asking Waldo questions: “Is $x \leq y$?”
Searching in partial orders

- Given is a partial order $S$ (or its diagram).
- Waldo secretly chooses $x \in S$.
- Goal: Find out $x$ by asking Waldo questions: “Is $x \leq y$?”

![Graph showing searching in partial orders](image-url)
Searching in partial orders

- Given is a partial order $S$ (or its diagram).
- Waldo secretly chooses $x \in S$.
- Goal: Find out $x$ by asking Waldo questions: “Is $x \leq y$?”
- For some posets the problem is identical to searching in trees in the edge-query model.
Optimal strategies

By a strategy for a given problem we mean a decision tree for solving this problem.
Optimal strategies

By a strategy for a given problem we mean a decision tree for solving this problem.

By an optimal strategy for this problem we mean the shallowest decision tree for solving this problem.
By a **strategy** for a given problem we mean a decision tree for solving this problem.

By an **optimal strategy** for this problem we mean the shallowest decision tree for solving this problem.

A sample optimal strategy in the vertex-query model:
Previous work

- Hyafil, Rivest [IPL 1976]:
  - computing optimal decision trees is NP-hard for general structures
Previous work

- Hyafil, Rivest [IPL 1976]:
  - computing optimal decision trees is NP-hard for general structures

- Ben-Asher, Farchi, Newman [SIAM J. on Comp. 1997]:
  - edge-query model: optimal strategy in $O(n^4 \log^3 n)$
Previous work

- Hyafil, Rivest [IPL 1976]:
  - computing optimal decision trees is NP-hard for general structures
- Ben-Asher, Farchi, Newman [SIAM J. on Comp. 1997]:
  - edge-query model: optimal strategy in $O(n^4 \log^3 n)$
- Laber, Nogueira [ENDM 2001]:
  - edge-query model: 2-approximation in $O(n \log n)$
Previous work

- Hyafil, Rivest [IPL 1976]:
  - computing optimal decision trees is NP-hard for general structures
- Ben-Asher, Farchi, Newman [SIAM J. on Comp. 1997]:
  - edge-query model: optimal strategy in $O(n^4 \log^3 n)$
- Laber, Nogueira [ENDM 2001]:
  - edge-query model: 2-approximation in $O(n \log n)$
- Carmo, Donadelli, Kohayakawa, Laber [TCS 2004]:
  - finding optimal poset searching strategy is NP-hard
  - approximate strategies for random posets
Previous work

- Hyafil, Rivest [IPL 1976]:
  - computing optimal decision trees is NP-hard for general structures

- Ben-Asher, Farchi, Newman [SIAM J. on Comp. 1997]:
  - edge-query model: optimal strategy in $O(n^4 \log^3 n)$

- Laber, Nogueira [ENDM 2001]:
  - edge-query model: 2-approximation in $O(n \log n)$

- Carmo, Donadelli, Kohayakawa, Laber [TCS 2004]:
  - finding optimal poset searching strategy is NP-hard
  - approximate strategies for random posets

- Onak, Parys [FOCS 2006]:
  - edge-query model: optimal strategy in $O(n^3)$
  - vertex-query model: optimal strategy in $O(n)$
Our Results

- $O(n)$ in the edge-query model [SODA 2008]
- novel bottom-up construction algorithm
- a method for reusing parts of already computed subproblems
- from a solution in the form of an edge-weighed tree to a decision tree solution in $O(n)$
Our Results

- \(O(n)\) in the edge-query model [SODA 2008]
  - novel bottom-up construction algorithm
  - a method for reusing parts of already computed subproblems
  - from a solution in the form of an edge-weighed tree to a decision tree solution in \(O(n)\)

Applications
- file system synchronization
- bug detection
General technique [OP 2006]

Short overview:
General technique [OP 2006]

Short overview:

- Reduce the problem to optimizing a strategy function.
Short overview:

- Reduce the problem to optimizing a strategy function.
- Recursively construct an optimum strategy function.
General technique [OP 2006]

Short overview:

1. Reduce the problem to optimizing a strategy function.
2. Recursively construct an optimum strategy function.

We start with the vertex-query model.
Strategy functions

Strategy function:
Strategy functions

Strategy function:

A function on objects that we can ask about. In our case it goes from the set of vertices to nonnegative integers,

\[ f : V \rightarrow \{0, 1, 2, \ldots\}. \]
Strategy functions

Strategy function:
- A function on objects that we can ask about. In our case it goes from the set of vertices to nonnegative integers,
  \[ f : V \rightarrow \{0, 1, 2, \ldots\}. \]
- For any two different \( v \) and \( w \) such that \( f(v) = f(w) \), there is \( u \) on the path from \( v \) to \( w \) such that
  \[ f(u) > f(v) = f(w). \]
Strategy functions

Strategy function:

- A function on objects that we can ask about. In our case it goes from the set of vertices to nonnegative integers, \( f : V \rightarrow \{0, 1, 2, \ldots\} \).
- For any two different \( v \) and \( w \) such that \( f(v) = f(w) \), there is \( u \) on the path from \( v \) to \( w \) such that \( f(u) > f(v) = f(w) \).
Mutual correspondence

A strategy function bounded by $k$

$\Rightarrow$ a strategy of at most $k$ queries in the worst case
Mutual correspondence

A strategy function bounded by $k$

$\Rightarrow$ a strategy of at most $k$ queries in the worst case

Idea: Ask about the vertex of the greatest value in the subtree induced by the potential solutions
Mutual correspondence

A strategy function bounded by $k$
$\Rightarrow$ a strategy of at most $k$ queries in the worst case

Idea: Ask about the vertex of the greatest value in the subtree induced by the potential solutions
Mutual correspondence

A strategy of $k$ queries in the worst case
⇒ a strategy function bounded by $k$
Mutual correspondence

A strategy of \( k \) queries in the worst case
⇒ a strategy function bounded by \( k \)

Idea: If we ask about a vertex \( v \), let \( f(v) \) be the number of further questions we need to ask before we find the target.
Mutual correspondence

A strategy of $k$ queries in the worst case
⇒ a strategy function bounded by $k$

Idea: If we ask about a vertex $v$, let $f(v)$ be the number of further questions we need to ask before we find the target.

$f(v) = 3$:

$f(v) = 2$:

$f(v) = 1$:
Conclusion

It suffices to construct a strategy function of the least maximum!
Visibility

The value at a vertex $w$ is **visible** from a vertex $v$ if on the simple path from $v$ to $w$ there is no greater value.
Visibility

The value at a vertex $w$ is visible from a vertex $v$ if on the simple path from $v$ to $w$ there is no greater value.

Values visible from $v$: 3, 2, 5, 6
Visibility sequences

The visibility sequence from a vertex $v$ is the sequence of all values visible from $v$, enumerated from the greatest to the least.
The visibility sequence from a vertex \( v \) is the sequence of all values visible from \( v \), enumerated from the greatest to the least.

The visibility sequence from \( v \): \( (6, 5, 3, 2) \)
The **visibility sequence** from a vertex $v$ is the sequence of all values visible from $v$, enumerated from the greatest to the least.

The visibility sequence from $v$: $(6, 5, 3, 2)$

The visibility sequences are ordered lexicographically. For instance, $(8, 4, 3, 2) > (7, 6, 4, 2, 1)$. 
Extension operator

1. Root the input tree arbitrarily.
Extension operator

1. Root the input tree arbitrarily.
2. At each vertex $v$: 
Extension operator

1. Root the input tree arbitrarily.

2. At each vertex $v$:
   (a) Take recursively computed strategy functions on subtrees rooted at children of $v$. 
Extension operator

1. Root the input tree arbitrarily.

2. At each vertex $v$:
   (a) Take recursively computed strategy functions on subtrees rooted at children of $v$.
   (b) Extend them to the subtree rooted at $v$. In vertex-query model we only need to fix $f(v)$. 
Extension operator

1. Root the input tree arbitrarily.

2. At each vertex $v$:
   (a) Take recursively computed strategy functions on subtrees rooted at children of $v$.
   (b) Extend them to the subtree rooted at $v$. In vertex-query model we only need to fix $f(v)$.

   ! To get a correct strategy function, it suffices to know the visibility sequences from children of $v$ in their subtrees.
Extension operator

1. Root the input tree arbitrarily.

2. At each vertex \( v \):
   (a) Take recursively computed strategy functions on subtrees rooted at children of \( v \).
   (b) Extend them to the subtree rooted at \( v \). In vertex-query model we only need to fix \( f(v) \).

To get a correct strategy function, it suffices to know the visibility sequences from children of \( v \) in their subtrees.

An extension operator is a procedure that takes those visibility sequences, extends the function, and returns the visibility sequence from \( v \) in the subtree rooted at \( v \).
An Optimal Extension

A *minimizing* extension is one that gives the lexicographically smallest visibility sequence at $v$. *minimizing extensions* accumulate to an optimal solution [OP 2006].
An extension operator $\forall$ for a vertex $v$: 

\[
\begin{array}{c|c|c}
 s_1 & s_2 & s_3 \\
 5 & 2 & 3 \\
 1 & 1 & 2 \\
 0 & 0 & 0 \\
\end{array}
\]
An extension operator $\mathcal{V}$ for a vertex $v$:

1. Find the greatest value $q$ that occurs in more than one sequence.

$q = 2$
An extension operator $\mathcal{V}$ for a vertex $v$:

1. Find the greatest value $q$ that occurs in more than one sequence.

2. Let $f(v)$ be the least value greater than $q$ that does not occur in any visibility sequence.
An extension operator $\mathcal{V}$ for a vertex $v$:

1. Find the greatest value $q$ that occurs in more than one sequence.

2. Let $f(v)$ be the least value greater than $q$ that does not occur in any visibility sequence.
One can show that $\nabla$ is minimizing.
One can show that $V$ is minimizing.

The whole computation takes $O(n \log n)$ time, as in the vertex-query model the required vertex can always be located in at most $\lceil \log_2 n \rceil$ queries.
One can show that \( \nabla \) is minimizing.

The whole computation takes \( O(n \log n) \) time, as in the vertex-query model the required vertex can always be located in at most \( \lfloor \log_2 n \rfloor \) queries.

The running time can be improved to \( O(n) \) fairly simple.
Edge-query model
Edge-query model

Questions about edges.
Edge-query model

- Questions about edges.
- Ask about an edge $e$. 
Edge-query model

- Questions about edges.
  - Ask about an edge $e$.
  - Learn which endpoint of $e$ is closer to Waldo.
Edge-query model

An extension assigns all $f(e_i)$’s
An extension assigns all $f(e_i)$’s

$f(e_i) \neq f(e_j)$
Edge-query model

An extension assigns all $f(e_i)$’s

- $f(e_i) \neq f(e_j)$
- $f(e_i)$ is not in $s_i$
An extension assigns all $f(e_i)$’s

- $f(e_i) \neq f(e_j)$
- $f(e_i)$ is not in $s_i$
- $f(e_i)$ is in $s_j$ $\Rightarrow$ $f(e_j) > f(e_i)$
An extension assigns all $f(e_i)$’s

- $f(e_i) \neq f(e_j)$
- $f(e_i)$ is not in $s_i$
- $f(e_i)$ is in $s_j \Rightarrow f(e_i) > f(e_i)$
- $u$ is in $s_i$ and $s_j \Rightarrow \max\{f(e_i), f(e_i)\} > u$
Algorithm Outline

Table: Free values

<table>
<thead>
<tr>
<th>free values</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
</table>

Diagram:

- Node $v$ with edges to $f(e_1)$, $f(e_2)$, and $f(e_k)$
- Subtrees $s_1$, $s_2$, and $s_k$ with values:
  - $s_1$: $4$ and $0$
  - $s_2$: $1$ and $0$
  - $s_k$: $1$ and $0$
Algorithm Outline

set $u = \max \{ s_i \}$
Algorithm Outline

- set $u = \max\{s_i\}$
- while not all edges assigned

<table>
<thead>
<tr>
<th>free values</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diagram:
- $s_1$:
  - $f(e_1)?$
  - 4
  - 0
- $s_2$:
  - $f(e_2)?$
  - 1
  - 0
- $s_k$:
  - $f(e_k)?$
  - 1
  - 0
Algorithm Outline

- set $u = \max\{s_i\}$
- while not all edges assigned
  - if $u$ appears once mark $u$ as *not free*, move to next largest $u$
Algorithm Outline

- set $u = \max\{s_i\}$
- while not all edges assigned
  - if $u$ appears once mark $u$ as not free, move to next largest $u$

```
free values

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$s_2$</td>
<td></td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$s_k$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
```

$\text{v}$

$f(e_1)$?

$f(e_2)$?

$f(e_k)$?
Algorithm Outline

- set $u = \max\{s_i\}$
- while not all edges assigned
  - if $u$ appears once mark $u$ as not free, move to next largest $u$
  - otherwise:

<table>
<thead>
<tr>
<th>free values</th>
<th>$u$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

$\begin{align*}
\{s_1, s_2, \ldots, s_k\} & \\
\begin{array}{cccccc}
4 & f(e_1) & 1 & 1 & 1 & 4 & \ldots & 1 & 0 & 0 & 0
\end{array}
\end{align*}$
Algorithm Outline

- set $u = \max\{s_i\}$
- while not all edges assigned
  - if $u$ appears once mark $u$ as not free, move to next largest $u$
  - otherwise:
    - $w = $ smallest free value > $u$

<table>
<thead>
<tr>
<th>free values</th>
<th>$U$</th>
<th>$W$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

- $f(e_1)$? 
- $f(e_2)$? 
- $f(e_k)$?
### Algorithm Outline

- **set** $u = \max\{s_i\}$
- **while** not all edges assigned
  - if $u$ appears once mark $u$ as *not free*, move to next largest $u$
  - otherwise:
    - $w = \text{smallest free value} > u$
    - $S_j = \text{any maximal sequence w.r.t } w$

<table>
<thead>
<tr>
<th>Free values</th>
<th>$U$</th>
<th>$W$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

![Graph](image)
Algorithm Outline

- set $u = \max\{s_i\}$
- while not all edges assigned
  - if $u$ appears once mark $u$ as not free, move to next largest $u$
  - otherwise:
    - $w =$ smallest free value $> u$
    - $S_j =$ any maximal sequence w.r.t $w$

free values

<table>
<thead>
<tr>
<th></th>
<th>$U$</th>
<th>$W$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
Algorithm Outline

- set $u = \max\{s_i\}$
- while not all edges assigned
  - if $u$ appears once mark $u$ as not free, move to next largest $u$
  - otherwise:
    - $w = \text{smallest free value} > u$
    - $S_j = \text{any maximal sequence w.r.t } w$
    - mark $w$ as not free

<table>
<thead>
<tr>
<th>Free values</th>
<th>$u$</th>
<th>$w$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Diagram:
- $f(e_i)$, $f(e_2)$, $f(e_k)$
- $S_j$ values:
  - $s_1$: 0, 1, 4, 5
  - $s_2$: 0, 1, 4, 5
  - $s_k$: 0, 1, 4, 5
- $W$: 1, 1, 1
- $u$, $w$: 0, 1, 2, 3, 5, 6
Algorithm Outline

- set \( u = \max\{ s_i \} \)
- while not all edges assigned
  - if \( u \) appears once mark \( u \) as \textit{not free}, move to next largest \( u \)
  - otherwise:
    - \( w = \) smallest free value > \( u \)
    - \( S_j = \) any maximal sequence w.r.t \( w \)
    - mark \( w \) as \textit{not free}

\[
\begin{array}{cccccc}
\text{free values} & U & W \\
0 & 1 & 3 & 5 & 6 \\
\end{array}
\]
Algorithm Outline

- set $u = \max \{ s_i \}$
- while not all edges assigned
  - if $u$ appears once mark $u$ as not free, move to next largest $u$
  - otherwise:
    - $w =$ smallest free value $> u$
    - $S_j =$ any maximal sequence w.r.t $w$
    - mark $w$ as not free
    - set current $f(e_j) = w$

<table>
<thead>
<tr>
<th>free values</th>
<th>$U$</th>
<th>$W$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>
Algorithm Outline

- set \( u = \max\{ s_i \} \)
- while not all edges assigned
  - if \( u \) appears once mark \( u \) as not free, move to next largest \( u \)
  - otherwise:
    - \( w = \) smallest free value > \( u \)
    - \( S_j = \) any maximal sequence w.r.t \( w \)
    - mark \( w \) as not free
    - set current \( f(e_j) = w \)

<table>
<thead>
<tr>
<th>free values</th>
<th>( U )</th>
<th>( W )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>
set \( u = \max\{s_i\} \)

while not all edges assigned

if \( u \) appears once mark \( u \) as not free, move to next largest \( u \)
otherwise:

\( w = \) smallest free value > \( u \)
\( S_j = \) any maximal sequence w.r.t \( w \)
mark \( w \) as not free
set current \( f(e_j) = w \)
mark all \( S_j \) values between \( u \) and \( w \) as free
Algorithm Outline

1. set $u = \max\{s_i\}$
2. while not all edges assigned
   - if $u$ appears once mark $u$ as not free, move to next largest $u$
   - otherwise:
     - $w =$ smallest free value $>$ $u$
     - $S_j =$ any maximal sequence w.r.t $w$
     - mark $w$ as not free
     - set current $f(e_j) = w$
     - mark all $S_j$ values between $u$ and $w$ as free
     - remove all values $<$ $w$ from $S_j$

free values

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>3</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
</table>

Diagram:
- Set $u = \max\{s_i\}$
- While not all edges assigned
  - If $u$ appears once, mark $u$ as not free and move to next largest $u$
  - Otherwise:
    - $w =$ smallest free value $>$ $u$
    - $S_j =$ any maximal sequence w.r.t $w$
    - Mark $w$ as not free
    - Set current $f(e_j) = w$
    - Mark all $S_j$ values between $u$ and $w$ as free
    - Remove all values $<$ $w$ from $S_j$
Algorithm Outline

- set \( u = \max\{s_i\} \)
- while not all edges assigned
  - if \( u \) appears once mark \( u \) as *not free*, move to next largest \( u \)
  - otherwise:
    - \( w = \) smallest free value > \( u \)
    - \( S_j = \) any maximal sequence w.r.t \( w \)
    - mark \( w \) as *not free*
    - set current \( f(e_j) = w \)
    - mark all \( S_j \) values between \( u \) and \( w \) as *free*
    - remove all values < \( w \) from \( S_j \)

<table>
<thead>
<tr>
<th>free values</th>
<th>( U )</th>
<th>( W )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 0 )</td>
<td>( 1 )</td>
<td>( 3 )</td>
</tr>
<tr>
<td>( 5 )</td>
<td>( 6 )</td>
<td></td>
</tr>
</tbody>
</table>
Algorithm Outline

- set $u = \max\{ s_i \}$
- while not all edges assigned
  - if $u$ appears once mark $u$ as not free, move to next largest $u$
  - otherwise:
    - $w =$ smallest free value $> u$
    - $S_j =$ any maximal sequence w.r.t $w$
    - mark $w$ as not free
    - set current $f(e_j) = w$
    - mark all $S_j$ values between $u$ and $w$ as free
    - remove all values $< w$ from $S_j$

| free values | 0 | 1 | 3 | 5 | 6 |
Algorithm Outline

- set $u = \max\{s_i\}$
- while not all edges assigned
  - if $u$ appears once mark $u$ as not free, move to next largest $u$
  - otherwise:
    - $w = \text{smallest free value} > u$
    - $S_j = \text{any maximal sequence w.r.t } w$
    - mark $w$ as not free
    - set current $f(e_j) = w$
    - mark all $S_j$ values between $u$ and $w$ as free
    - remove all values < $w$ from $S_j$

```
free values  
---
<table>
<thead>
<tr>
<th>$u$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
</tbody>
</table>
```
Algorithm Outline

- set \( u = \text{max}\{s_i\} \)
- while not all edges assigned
  - if \( u \) appears once mark \( u \) as not free, move to next largest \( u \)
  - otherwise:
    - \( w = \text{smallest free value} > u \)
    - \( S_j = \text{any maximal sequence w.r.t} \ w \)
    - mark \( w \) as not free
    - set current \( f(e_j) = w \)
    - mark all \( S_j \) values between \( u \) and \( w \) as free
    - remove all values < \( w \) from \( S_j \)

<table>
<thead>
<tr>
<th>free values</th>
<th>( U )</th>
<th>( W )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 1</td>
<td>3</td>
</tr>
</tbody>
</table>

Diagram:
- Tree with vertices labeled 2, 4, 1, ..., 1
- \( S_j \) intervals
- \( f(e_2) \) and \( f(e_k) \)?
Algorithm Outline

- set \( u = \max\{s_i\} \)
- while not all edges assigned
  - if \( u \) appears once mark \( u \) as not free, move to next largest \( u \)
  - otherwise:
    - \( w = \) smallest free value > \( u \)
    - \( S_j = \) any maximal sequence w.r.t \( w \)
    - mark \( w \) as not free
    - set current \( f(e_j) = w \)
    - mark all \( S_j \) values between \( u \) and \( w \) as free
    - remove all values < \( w \) from \( S_j \)

**free values**

\[
\begin{array}{c|c|c|c}
\text{U} & \text{W} \\
\hline
0 & 1 & 5 & 6 \\
\end{array}
\]
Algorithm Outline

- set \( u = \max\{s_i\} \)

- while not all edges assigned
  - if \( u \) appears once mark \( u \) as *not free*, move to next largest \( u \)
  - otherwise:
    - \( w \) = smallest free value > \( u \)
    - \( S_j \) = any maximal sequence w.r.t \( w \)
    - mark \( w \) as *not free*
    - set current \( f(e_j) = w \)
    - mark all \( S_j \) values between \( u \) and \( w \) as *free*
    - remove all values < \( w \) from \( S_j \)

<table>
<thead>
<tr>
<th>free values</th>
<th>( U )</th>
<th>( W )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ S_j = s_1 s_2 s_3 \ldots \]
Algorithm Outline

set $u = \text{max}\{s_i\}$

while not all edges assigned
  if $u$ appears once mark $u$ as not free, move to next largest $u$
  otherwise:
    $w = \text{smallest free value} > u$
    $S_j = \text{any maximal sequence w.r.t } w$
    mark $w$ as not free
    set current $f(e_j) = w$
    mark all $S_j$ values between $u$ and $w$ as free
    remove all values $< w$ from $S_j$

<table>
<thead>
<tr>
<th>free values</th>
<th>$U$</th>
<th>$W$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>
Algorithm Outline

- set $u = \max \{ s_i \}$
- while not all edges assigned
  - if $u$ appears once mark $u$ as not free, move to next largest $u$
  - otherwise:
    - $w =$ smallest free value $> u$
    - $S_j =$ any maximal sequence w.r.t $w$
    - mark $w$ as not free
    - set current $f(e_j) = w$
    - mark all $S_j$ values between $u$ and $w$ as free
    - remove all values $< w$ from $S_j$

<table>
<thead>
<tr>
<th>free values</th>
<th>$U$</th>
<th>$W$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

- remove all values $< w$ from $S_j$
Algorithm Outline

- set \( u = \max\{s_i\} \)
- while not all edges assigned
  - if \( u \) appears once mark \( u \) as *not free*, move to next largest \( u \)
  - otherwise:
    - \( w = \) smallest free value > \( u \)
    - \( S_j = \) any maximal sequence w.r.t \( w \)
    - mark \( w \) as *not free*
    - set current \( f(e_j) = w \)
    - mark all \( S_j \) values between \( u \) and \( w \) as *free*
    - remove all values < \( w \) from \( S_j \)

free values

\[
\begin{array}{cccccc}
0 & 1 & 2 & & 5 & 6
\end{array}
\]
Algorithm Outline

- set $u = \max\{s_i\}$
- while not all edges assigned
  - if $u$ appears once mark $u$ as *not free*, move to next largest $u$
  - otherwise:
    - $w = \text{smallest free value} > u$
    - $S_j = \text{any maximal sequence w.r.t} \ w$
    - mark $w$ as *not free*
    - set current $f(e_j) = w$
    - mark all $S_j$ values between $u$ and $w$ as *free*
    - remove all values $< w$ from $S_j$

<table>
<thead>
<tr>
<th>free values</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
</table>

*Abbreviation*

$v$

$s_1$

$v$

$s_2$

$v$

$s_k$
Algorithm Outline

- set $u = \max\{s_i\}$
- while not all edges assigned
  - if $u$ appears once mark $u$ as not free, move to next largest $u$
  - otherwise:
    - $w =$ smallest free value $> u$
    - $S_j =$ any maximal sequence w.r.t $w$
    - mark $w$ as not free
    - set current $f(e_j) = w$
    - mark all $S_j$ values between $u$ and $w$ as free
    - remove all values $< w$ from $S_j$

### Free Values

<table>
<thead>
<tr>
<th>Free Values</th>
<th>$U$</th>
<th>$W$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>
**Algorithm Outline**

- set $u = \max\{s_i\}$

- while not all edges assigned
  - if $u$ appears once mark $u$ as *not free*, move to next largest $u$
  - otherwise:
    - $w = \text{smallest free value} > u$
    - $S_j = \text{any maximal sequence w.r.t } w$
    - mark $w$ as *not free*
    - set current $f(e_j) = w$
    - mark all $S_j$ values between $u$ and $w$ as *free*
    - remove all values $< w$ from $S_j$

<table>
<thead>
<tr>
<th>free values</th>
<th>$U$</th>
<th>$W$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>
Algorithm Outline

- set \( u = \max\{s_i\} \)
- while not all edges assigned
  - if \( u \) appears once mark \( u \) as not free, move to next largest \( u \)
  - otherwise:
    - \( w = \) smallest free value > \( u \)
    - \( S_j = \) any maximal sequence w.r.t \( w \)
    - mark \( w \) as not free
    - set current \( f(e_j) = w \)
    - mark all \( S_j \) values between \( u \) and \( w \) as free
    - remove all values < \( w \) from \( S_j \)

<table>
<thead>
<tr>
<th>free values</th>
<th>( U )</th>
<th>( W )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Algorithm Outline

1. **set** \( u = \max\{s_i\} \)
2. **while** not all edges assigned
   - **if** \( u \) appears once mark \( u \) as *not free*, move to next largest \( u \)
   - **otherwise:**
     - \( w = \) smallest free value > \( u \)
     - \( S_j = \) any maximal sequence w.r.t \( w \)
     - mark \( w \) as *not free*
     - set current \( f(e_j) = w \)
     - mark all \( S_j \) values between \( u \) and \( w \) as *free*
     - remove all values < \( w \) from \( S_j \)

<table>
<thead>
<tr>
<th>free values</th>
<th>( u )</th>
<th>( w )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Diagram:

- Tree: \( v \rightarrow s_1, s_2, s_k \)
- Path: \( f(e_k) \)
Algorithm Outline

- set $u = \max \{ s_i \}$
- while not all edges assigned
  - if $u$ appears once mark $u$ as not free, move to next largest $u$
  - otherwise:
    - $w = \text{smallest free value} > u$
    - $S_j = \text{any maximal sequence w.r.t } w$
    - mark $w$ as not free
    - set current $f(e_j) = w$
    - mark all $S_j$ values between $u$ and $w$ as free
    - remove all values $< w$ from $S_j$

```
free values
| 0 | 1 |   | 5 | 6 |
```
Algorithm Outline

1. set \( u = \max\{s_i\} \)
2. while not all edges assigned
   1. if \( u \) appears once mark \( u \) as not free, move to next largest \( u \)
   2. otherwise:
      1. \( w = \) smallest free value > \( u \)
      2. \( S_j = \) any maximal sequence w.r.t \( w \)
      3. mark \( w \) as not free
      4. set current \( f(e_j) = w \)
      5. mark all \( S_j \) values between \( u \) and \( w \) as free
      6. remove all values < \( w \) from \( S_j \)

free values

\[
\begin{array}{c|c|c|c|c}
  & \text{free values} \\
\hline
  u & 0 & 1 & 5 & 6 \\
\end{array}
\]
Algorithm Outline

set \( u = \max\{s_i\} \)

while not all edges assigned

- if \( u \) appears once mark \( u \) as \textit{not free}, move to next largest \( u \)
- otherwise:
  - \( w = \) smallest free value > \( u \)
  - \( S_j = \) any maximal sequence w.r.t \( w \)
  - mark \( w \) as \textit{not free}
  - set current \( f(e_j) = w \)
  - mark all \( S_j \) values between \( u \) and \( w \) as \textit{free}
  - remove all values < \( w \) from \( S_j \)
Algorithm Outline

- set $u = \max\{s_i\}$
- while not all edges assigned
  - if $u$ appears once mark $u$ as *not free*, move to next largest $u$ and $u \neq 0$
  - otherwise:
    - $w$ = smallest free value $> u$
    - $S_j$ = any maximal sequence w.r.t $w$
    - mark $w$ as *not free*
    - set current $f(e_j) = w$
    - mark all $S_j$ values between $u$ and $w$ as *free*
    - remove all values $< w$ from $S_j$

<table>
<thead>
<tr>
<th>free values</th>
<th>0</th>
<th></th>
<th></th>
<th>5</th>
<th>6</th>
</tr>
</thead>
</table>
Algorithm Outline

- set \( u = \max \{ s_i \} \)
- while not all edges assigned
  - if \( u \) appears once, mark \( u \) as not free, move to next largest \( u \) and \( u \neq 0 \)
  - otherwise:
    - \( w = \) smallest free value > \( u \)
    - \( S_j = \) any maximal sequence w.r.t \( w \)
    - mark \( w \) as not free
    - set current \( f(e_j) = w \)
    - mark all \( S_j \) values between \( u \) and \( w \) as free
    - remove all values < \( w \) from \( S_j \)
Algorithm Outline

- set $u = \max\{s_i\}$
- while not all edges assigned
  - if $u$ appears once, mark $u$ as not free, move to next largest $u$
  - otherwise:
    - $w = $ smallest free value $> u$
    - $S_j = $ any maximal sequence w.r.t $w$
    - mark $w$ as not free
    - set current $f(e_j) = w$
    - mark all $S_j$ values between $u$ and $w$ as free
    - remove all values $< w$ from $S_j$

free values

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diagram:

- $v$
- $w$
- $s_1$
- $s_2$
- $s_k$
- $f(e_k)$?
- $4$
- $3$
- $2$
- $\ldots$
- $1$
- $0$
Algorithm Outline

1. Set $u = \max\{s_i\}$
2. While not all edges assigned
   - If $u$ appears once, mark $u$ as not free, move to next largest $u$ and $u \neq 0$
   - Otherwise:
     1. $w = \text{smallest free value} > u$
     2. $S_j = \text{any maximal sequence wrt } w$
     3. Mark $w$ as not free
     4. Set current $f(e_j) = w$
     5. Mark all $S_j$ values between $u$ and $w$ as free
     6. Remove all values $< w$ from $S_j$

Free values:

<table>
<thead>
<tr>
<th>$U$</th>
<th></th>
<th></th>
<th></th>
<th>$W$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>6</td>
</tr>
</tbody>
</table>
Algorithm Outline

- set $u = \max\{s_i\}$
- while not all edges assigned
  - if $u$ appears once mark $u$ as not free, move to next largest $u$
  - otherwise:
    - $w = $ smallest free value > $u$
    - $S_j = $ any maximal sequence w.r.t. $w$
    - mark $w$ as not free
    - set current $f(e_j) = w$
    - mark all $S_j$ values between $u$ and $w$ as free
    - remove all values < $w$ from $S_j$

<table>
<thead>
<tr>
<th>free values</th>
<th>$U$</th>
<th>$W$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

\[ f(e_k) = ? \]
Algorithm Outline

- set $u = \max\{s_i\}$
- while not all edges assigned
  - if $u$ appears once, mark $u$ as not free, move to next largest $u$
  - otherwise:
    - $w =$ smallest free value > $u$
    - $S_j =$ any maximal sequence w.r.t. $w$
    - mark $w$ as not free
    - set current $f(e_j) = w$
    - mark all $S_j$ values between $u$ and $w$ as free
    - remove all values < $w$ from $S_j$

free values

<table>
<thead>
<tr>
<th>$U$</th>
<th>0</th>
<th>3</th>
<th>4</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Algorithm Outline

- set \( u = \max\{s_i\} \)

- while not all edges assigned
  - if \( u \) appears once mark \( u \) as *not free*, move to next largest \( u \) and \( u \neq 0 \)
  - otherwise:
    - \( w = \) smallest free value > \( u \)
    - \( S_j = \) any maximal sequence w.r.t \( w \)
    - mark \( w \) as *not free*
    - set current \( f(e_j) = w \)
    - mark all \( S_j \) values between \( u \) and \( w \) as *free*
    - remove all values < \( w \) from \( S_j \)

<table>
<thead>
<tr>
<th>free values</th>
<th>0</th>
<th>3</th>
<th>4</th>
<th>6</th>
</tr>
</thead>
</table>
Algorithm Outline

1. **set** $u = \max\{s_i\}$
2. **while** not all edges assigned
   - if $u$ appears once, mark $u$ as *not free*, move to next largest $u$
   - otherwise:
     1. $w = \text{smallest free value} > u$
     2. $S_j = \text{any maximal sequence w.r.t } w$
     3. mark $w$ as *not free*
     4. set current $f(e_j) = w$
     5. mark all $S_j$ values between $u$ and $w$ as *free*
     6. remove all values $< w$ from $S_j$

<table>
<thead>
<tr>
<th>free values</th>
<th>U</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>
Algorithm Outline

set \( u = \max\{s_i\} \)

while not all edges assigned

- if \( u \) appears once, mark \( u \) as not free, move to next largest \( u \)
- otherwise:
  - \( w = \) smallest free value \( > u \)
  - \( S_j = \) any maximal sequence w.r.t \( w \)
  - mark \( w \) as not free
  - set current \( f(e_j) = w \)
  - mark all \( S_j \) values between \( u \) and \( w \) as free
  - remove all values \( < w \) from \( S_j \)

<table>
<thead>
<tr>
<th>free values</th>
<th>( U )</th>
<th>( W )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>4</td>
</tr>
</tbody>
</table>
**Algorithm Outline**

- **set** $u = \text{max}\{s_i\}$
- **while not all edges assigned**
  - if $u$ appears once, mark $u$ as *not free*, move to next largest $u$ (and $u \neq 0$)
  - otherwise:
    - $w = \text{smallest free value} > u$
    - $S_j = \text{any maximal sequence w.r.t } w$
    - mark $w$ as *not free*
    - set current $f(e_j) = w$
    - mark all $S_j$ values between $u$ and $w$ as *free*
    - remove all values $< w$ from $S_j$

<table>
<thead>
<tr>
<th>Free values</th>
<th>$U$</th>
<th>$W$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>4</td>
</tr>
</tbody>
</table>

**Diagram**:
- Node $v$ with children 5, 3, 2, 1, 0
- $S_j$ values
- $f(e_k)$?
Algorithm Outline

- set $u = \max\{s_i\}$
- while not all edges assigned
  - if $u$ appears once, mark $u$ as not free, move to next largest $u$ and $u \neq 0$
  - otherwise:
    - $w =$ smallest free value $> u$
    - $S_j =$ any maximal sequence w.r.t $w$
    - mark $w$ as not free
    - set current $f(e_j) = w$
    - mark all $S_j$ values between $u$ and $w$ as free
    - remove all values $< w$ from $S_j$

<table>
<thead>
<tr>
<th>free values</th>
<th>$U$</th>
<th>$W$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>
Algorithm Outline

- set $u = \max\{s_i\}$
- while not all edges assigned
  - if $u$ appears once mark $u$ as not free, move to next largest $u$
  - otherwise:
    - $w =$ smallest free value $> u$
    - $S_j =$ any maximal sequence w.r.t $w$
    - mark $w$ as not free
    - set current $f(e_j) = w$
    - mark all $S_j$ values between $u$ and $w$ as free
    - remove all values $< w$ from $S_j$

<table>
<thead>
<tr>
<th>free values</th>
<th>$U$</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>
Algorithm Outline

- set $u = \max\{s_i\}$

- while not all edges assigned

  - if $u$ appears once mark $u$ as not free, move to next largest $u$
  - otherwise:

    - $w = \text{smallest free value} > u$
    - $S_j = \text{any maximal sequence w.r.t } w$
    - mark $w$ as not free
    - set current $f(e_j) = w$
    - mark all $S_j$ values between $u$ and $w$ as free
    - remove all values $< w$ from $S_j$

**free values**

| 0 | 2 | 4 | 6 |
Algorithm Outline

- set \( u = \max\{s_i\} \)
- while not all edges assigned
  - if \( u \) appears once mark \( u \) as not free, move to next largest \( u \) and \( u \neq 0 \)
  - otherwise:
    - \( w = \) smallest free value > \( u \)
    - \( S_j = \) any maximal sequence w.r.t \( w \)
    - mark \( w \) as not free
    - set current \( f(e_j) = w \)
    - mark all \( S_j \) values between \( u \) and \( w \) as free
    - remove all values < \( w \) from \( S_j \)

\[ \text{free values} \begin{array}{|c|c|c|c|c|} 
\hline
0 & 2 & 4 & 6 \\
\hline
\end{array} \]
**Algorithm Outline**

- set $u = \max\{s_i\}$
- while not all edges assigned
  - if $u$ appears once, mark $u$ as *not free*, move to next largest $u$
  - otherwise:
    - $w =$ smallest free value $> u$
    - $S_j =$ any maximal sequence w.r.t $w$
    - mark $w$ as *not free*
    - set current $f(e_j) = w$
    - mark all $S_j$ values between $u$ and $w$ as *free*
    - remove all values $< w$ from $S_j$

<table>
<thead>
<tr>
<th>free values</th>
<th>$U$</th>
<th>$W$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u$</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

Diagram:
- Set $u = \max\{s_i\}$
- While not all edges assigned:
  - If $u$ appears once, mark $u$ as *not free*, move to next largest $u$
  - Otherwise:
    - $w =$ smallest free value $> u$
    - $S_j =$ any maximal sequence w.r.t $w$
    - Mark $w$ as *not free*
    - Set current $f(e_j) = w$
    - Mark all $S_j$ values between $u$ and $w$ as *free*
    - Remove all values $< w$ from $S_j$
Algorithm Outline

- set $u = \max\{s_i\}$
- while not all edges assigned
  - if $u$ appears once mark $u$ as not free, move to next largest $u$
  - otherwise:
    - $w = \text{smallest free value} > u$
    - $S_j = \text{any maximal sequence w.r.t } w$
    - mark $w$ as not free
    - set current $f(e_j) = w$
    - mark all $S_j$ values between $u$ and $w$ as free
    - remove all values $< w$ from $S_j$

| free values | 0 | 1 | 4 | 6 |
Algorithm Outline

- set \( u = \max \{ s_i \} \)
- while not all edges assigned
  - if \( u \) appears once mark \( u \) as not free, move to next largest \( u \)
  - otherwise:
    - \( w = \) smallest free value > \( u \)
    - \( S_j = \) any maximal sequence w.r.t \( w \)
    - mark \( w \) as not free
    - set current \( f(e_j) = w \)
    - mark all \( S_j \) values between \( u \) and \( w \) as free
    - remove all values < \( w \) from \( S_j \)

### free values

<table>
<thead>
<tr>
<th>( u )</th>
<th>( w )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
</tr>
</tbody>
</table>
Algorithm Outline

- set $u = \max\{s_i\}$
- while not all edges assigned
  - if $u$ appears once mark $u$ as *not free*, move to next largest $u$ and $u \neq 0$
  - otherwise:
    - $w =$ smallest free value $> u$
    - $S_j =$ any maximal sequence w.r.t $w$
    - mark $w$ as *not free*
    - set current $f(e_j) = w$
    - mark all $S_j$ values between $u$ and $w$ as *free*
    - remove all values $< w$ from $S_j$

### free values

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$u$</td>
<td>$w$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>4</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

Diagram:

- Vertex $v$ with edges to vertices 5, 3, 2
- Vertices 5, 3, 2 connected to vertices with values 5, 3, 2
- $S_j$ values: $s_1, s_2, s_k, \ldots, s_5$
Algorithm Outline

1. set \( u = \max\{s_i\} \)
2. while not all edges assigned
   - if \( u \) appears once, mark \( u \) as not free, move to next largest \( u \) and \( u \neq 0 \)
   - otherwise:
     1. \( w = \) smallest free value > \( u \)
     2. \( S_j = \) any maximal sequence w.r.t \( w \)
     3. mark \( w \) as not free
     4. set current \( f(e_j) = w \)
     5. mark all \( S_j \) values between \( u \) and \( w \) as free
     6. remove all values < \( w \) from \( S_j \)

<table>
<thead>
<tr>
<th>free values</th>
<th>( U )</th>
<th>( W )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>
Algorithm Outline

- set $u = \max\{s_i\}$
- while not all edges assigned
  - if $u$ appears once mark $u$ as *not free*, move to next largest $u$
  - otherwise:
    - $w =$ smallest free value $> u$
    - $S_j =$ any maximal sequence w.r.t $w$
    - mark $w$ as *not free*
    - set current $f(e_j) = w$
    - mark all $S_j$ values between $u$ and $w$ as *free*
    - remove all values $< w$ from $S_j$

That's it!

| free values | 0 | 1 | 4 | 6 |
Algorithm Outline

- Set $u = \max\{s_i\}$

- While not all edges assigned
  - If $u$ appears once, mark $u$ as not free, move to next largest $u$
  - Otherwise:
    - $w =$ smallest free value $> u$
    - $S_j =$ any maximal sequence w.r.t $w$
    - Mark $w$ as not free
    - Set current $f(e_j) = w$
    - Mark all $S_j$ values between $u$ and $w$ as free
    - Remove all values $< w$ from $S_j$

That’s it!
Running Time

\[ v \]

\[ s_1 \quad s_2 \quad s_k \]

\[ 4 \quad 1 \quad 1 \quad 1 \]

\[ 1 \quad 1 \quad 1 \]

\[ 0 \quad 0 \quad 0 \]
Running Time

$|S_1| + |S_2| + \ldots + |S_k|$ is not a lower bound!
| $S_1$| $S_2$| ...| $S_k$| is not a lower bound!

In many cases, the largest values of the largest visibility sequence are unchanged at $v$ itself.
$|S_1| + |S_2| + \ldots + |S_k|$ is not a lower bound!

In many cases, the largest values of the largest visibility sequence are unchanged at $v$ itself.

$k(v) = \#v$’s children
| $S_1| + |S_2| + ... + |S_k| \text{ is not a lower bound!}

in many cases, the largest values of the largest visibility sequence are unchanged at $v$ itself

$k(v) = \#v$’s children

$q(v) = |S_2| + ... + |S_k|
Running Time

| $S_1| + |S_2| + \ldots + |S_k| \text{ is not a lower bound}! |
\hline
in many cases, the largest values of the largest visibility sequence are unchanged at $v$ itself
\hline
$k(v) = \#v$’s children
\hline
$q(v) = |S_2| + \ldots + |S_k|
\hline
t(v) = \text{largest value that appears in } S_v \text{ but not in } S_I
\hline
\begin{array}{c}
4 \\
1 \\
0
\end{array} \quad \begin{array}{c}
1 \\
1 \\
0
\end{array} \quad \ldots \quad \begin{array}{c}
1 \\
0
\end{array}
Running Time

$|S_1| + |S_2| + ... + |S_k|$ is not a lower bound!

In many cases, the largest values of the largest visibility sequence are unchanged at $v$ itself.

- $k(v) = \#v$’s children
- $q(v) = |S_2| + ... + |S_k|
- $t(v) =$ largest value that appears in $S_v$ but not in $S_I$

An extension can be computed in $O(k(v) + q(v) + t(v))$.
Running Time

\[ |S_1| + |S_2| + ... + |S_k| \text{ is not a lower bound!} \]

in many cases, the largest values of the largest visibility sequence are unchanged at \( v \) itself

- \( k(v) = \#v's \) children
- \( q(v) = |S_2| + ... + |S_k| \)
- \( t(v) = \) largest value that appears in \( S_v \) but not in \( S_1 \)

an extension can be computed in \( O( k(v) + q(v) + t(v) ) \)

\[ \sum_v k(v) + q(v) + t(v) = O(n) \]
From Strategy Function to Decision Tree in $O(n)$ Time
From Strategy Function to Decision Tree in $O(n)$ Time
From Strategy Function to Decision Tree in $O(n)$ Time

For all edges $e$

- let $s =$ visibility sequence at $\text{bottom}(e)$
- if $s$ contains no values smaller than $f(e)$
  - set $\text{bottom}(e)$ as the solution when the query on $e$ returns $\text{bottom}(e)$
- else, let $v_1 < \ldots < v_k < f(e)$ in $s$, let $e_i$ be the edge $v_i$ is assigned to
  - set $e_k$ as the solution when the query on $e$ returns $\text{bottom}(e)$
  - for every $1 \leq i < k$ set $e_i$ as the solution when the query on $e_{i+1}$ returns $\text{top}(e_{i+1})$
  - set $\text{top}(e_1)$ as the solution when the query on $e_1$ returns $\text{top}(e_1)$
Thank you !!