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?

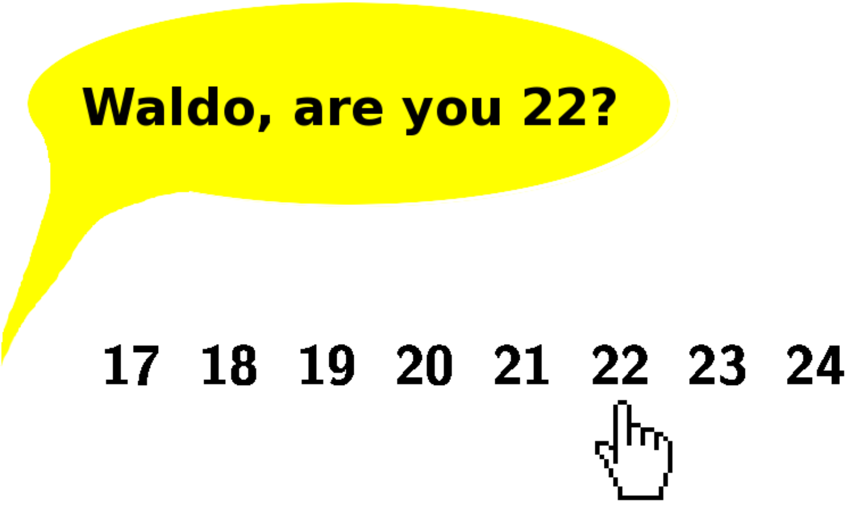
How quickly can you learn Waldo's age?

- 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?

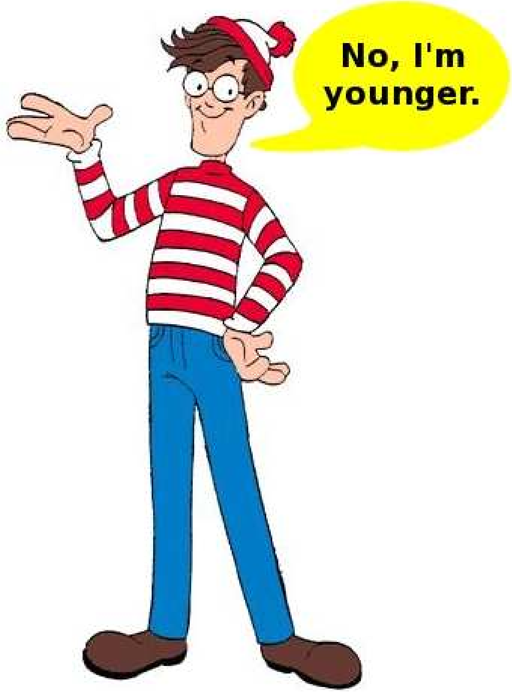
- You can ask Waldo if he’s x years old.
- Possible answers:
  - “Yes, I’m x years old.”
  - “No, I’m younger.”
  - “No, I’m older.”



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."

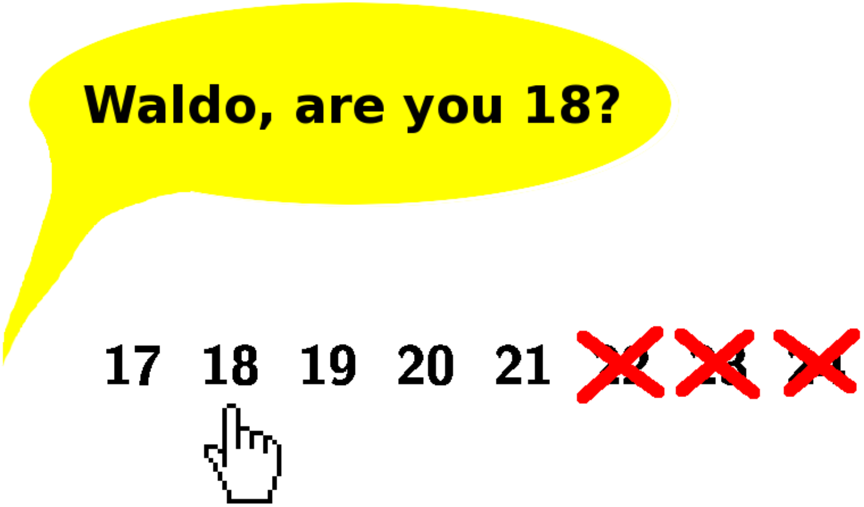


17 18 19 20 21 ~~22~~ ~~23~~ ~~24~~

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.”



How old is Waldo?

How quickly can you learn Waldo’s age?

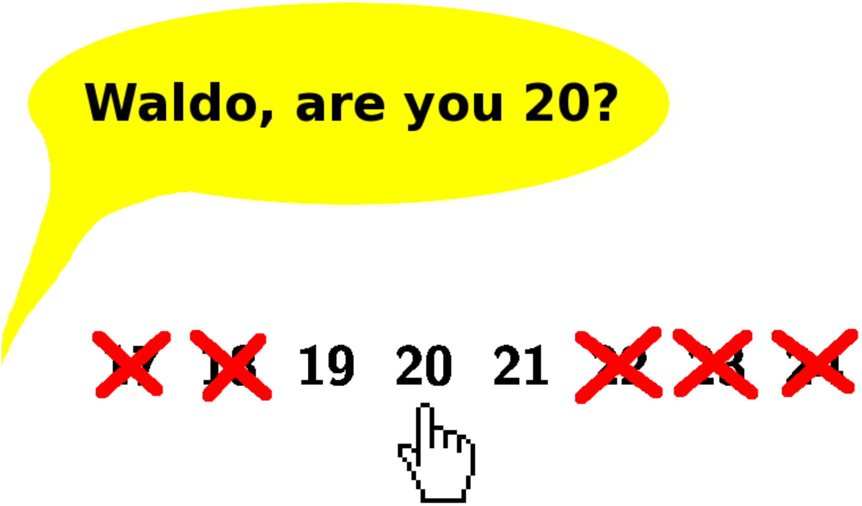
- You can ask Waldo if he’s x years old.
- Possible answers:
    - “Yes, I’m x years old.”
    - “No, I’m younger.”
    - “No, I’m older.”

~~17~~ ~~18~~ 19 20 21 ~~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.”



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~~ ~~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.”

You must be 19!

~~17~~ ~~18~~ 19 ~~20~~ ~~21~~ ~~22~~ ~~23~~ ~~24~~

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

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:

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.

Binary search

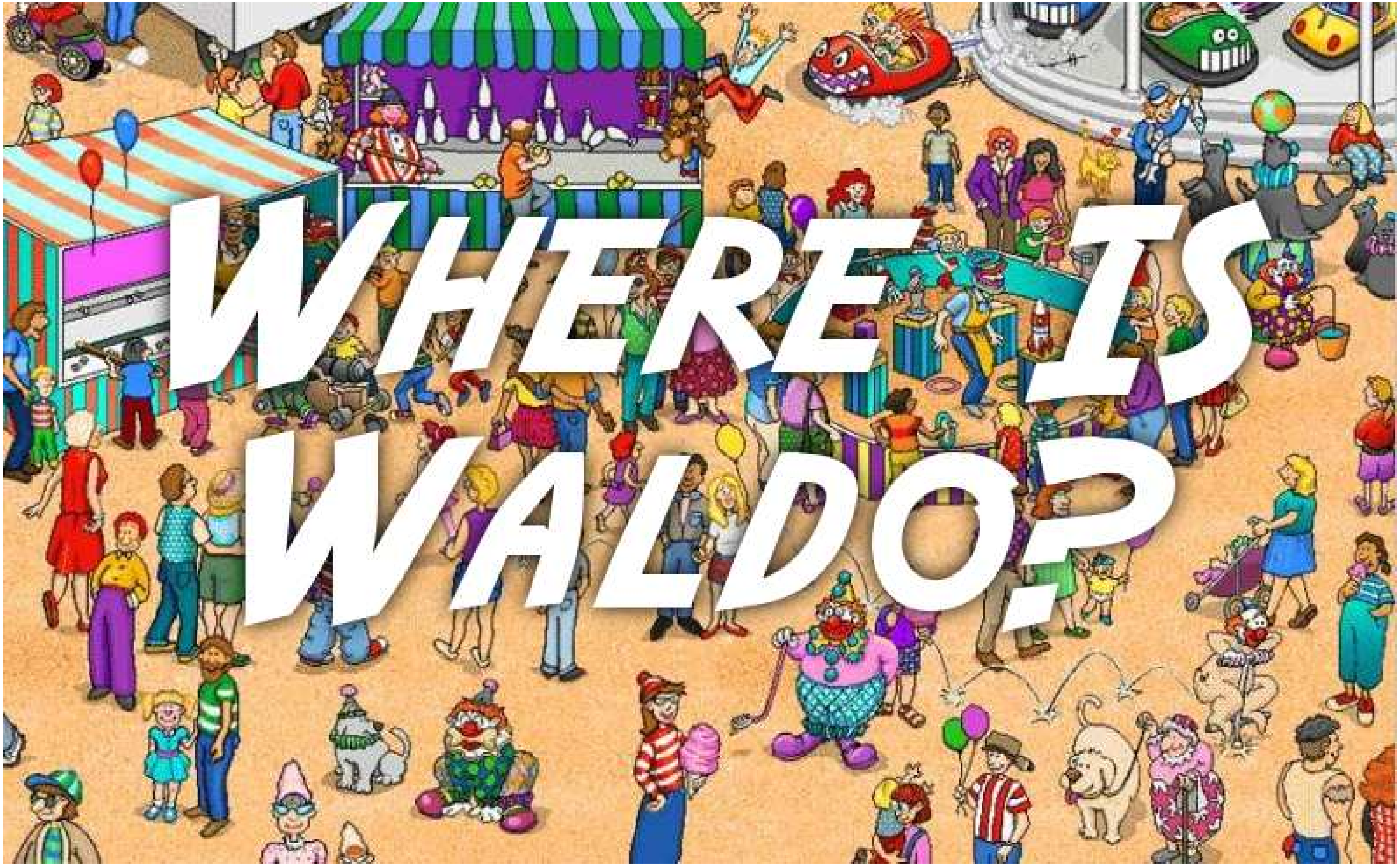
- Optimal solution:
  - Always ask about the number in the middle of the range of potential solutions.
  - $\lfloor \log_2 n \rfloor$ questions in the worst case, where n is the size of the range.
- The searching problem is easy:
  - Only two “directions”: greater and smaller numbers.
  - Potential solutions constitute a totally ordered set.

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 . . .

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!



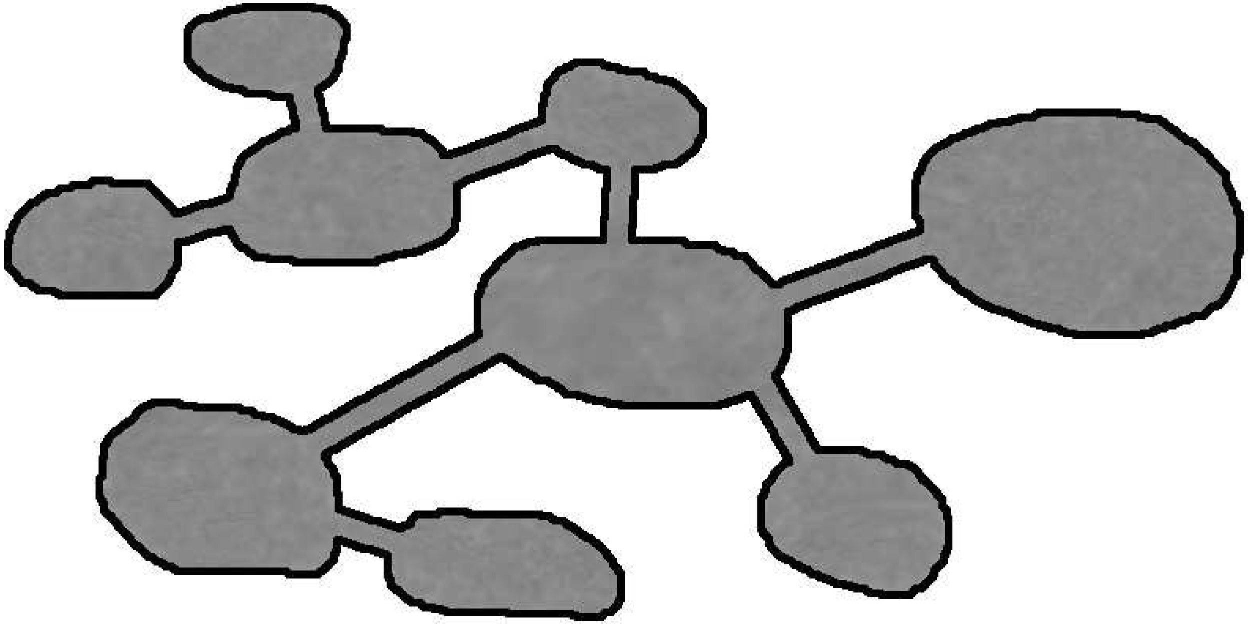
WHERE IS WALDO?

Searching in caves

- Waldo hides in a cave.

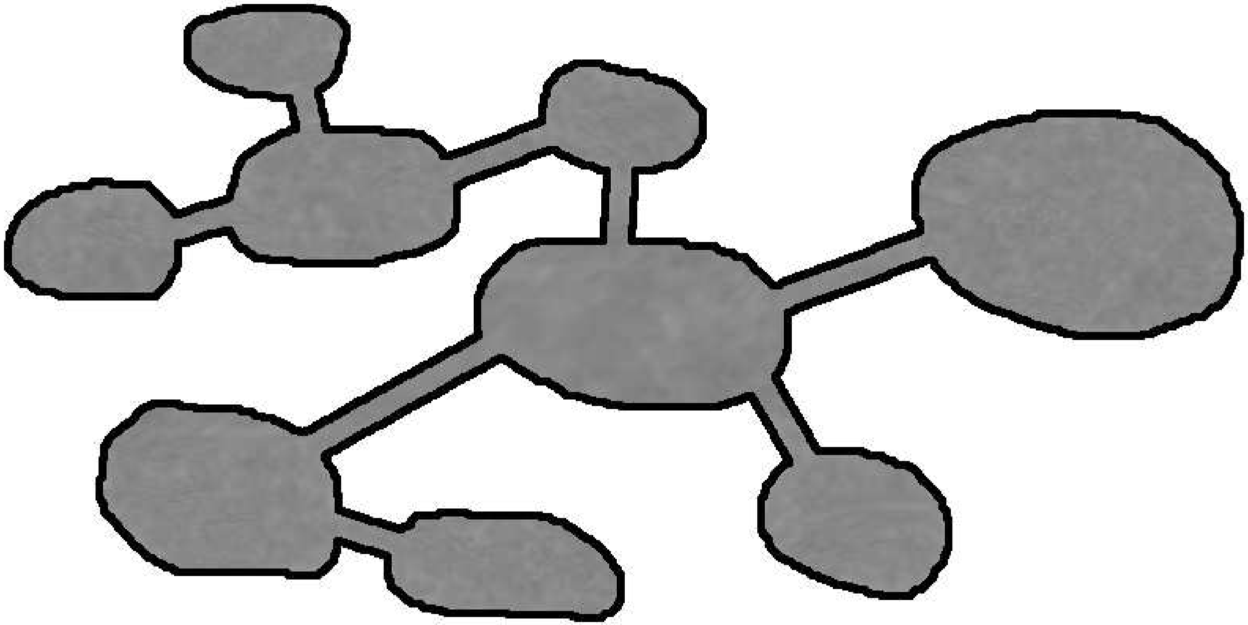
Searching in caves

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



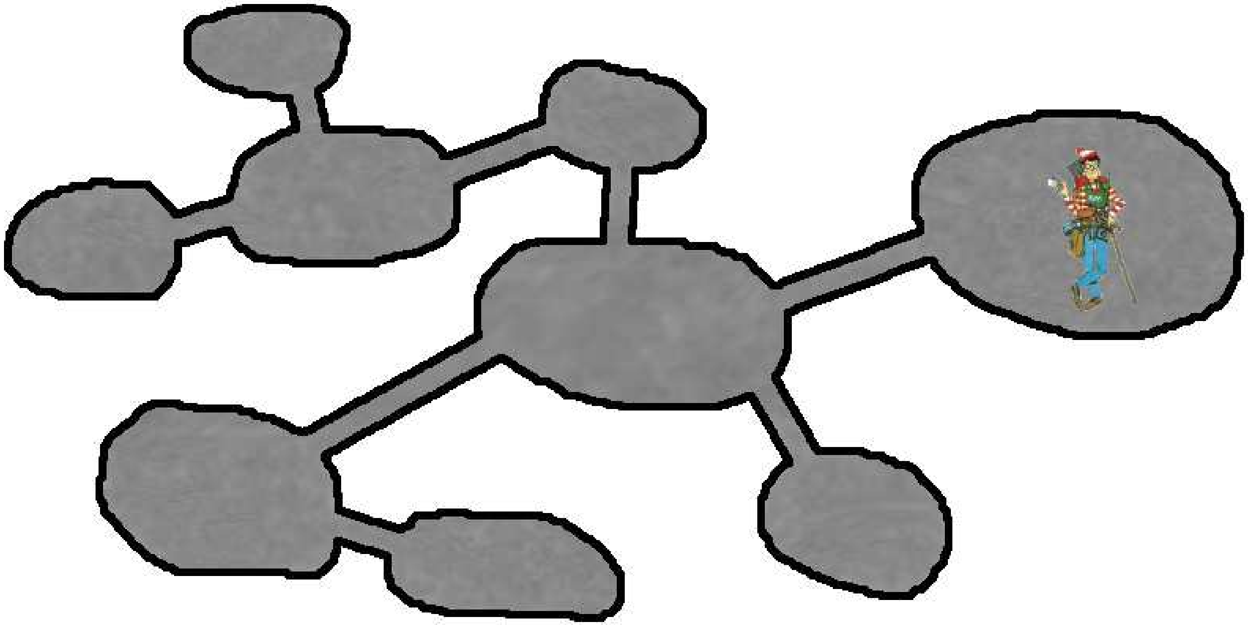
Searching in ~~caves~~ trees

- Waldo hides in a cave.
- The cave consists of chambers and corridors.
- The graph of the cave is a tree.



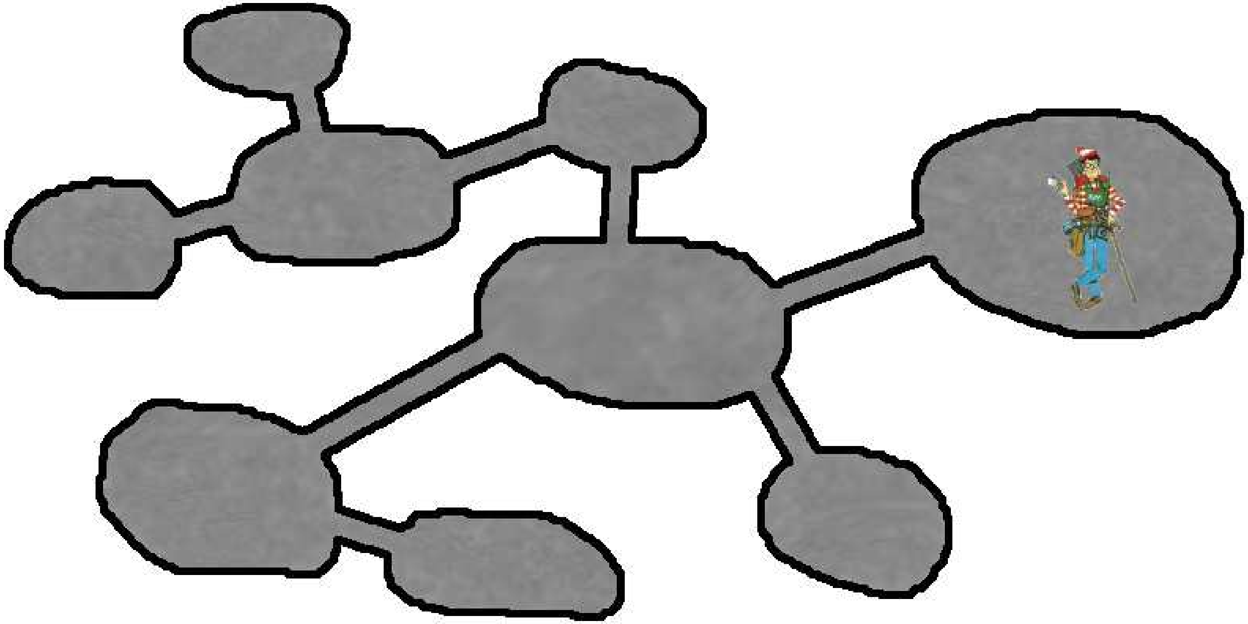
Searching in ~~caves~~ trees

- 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.



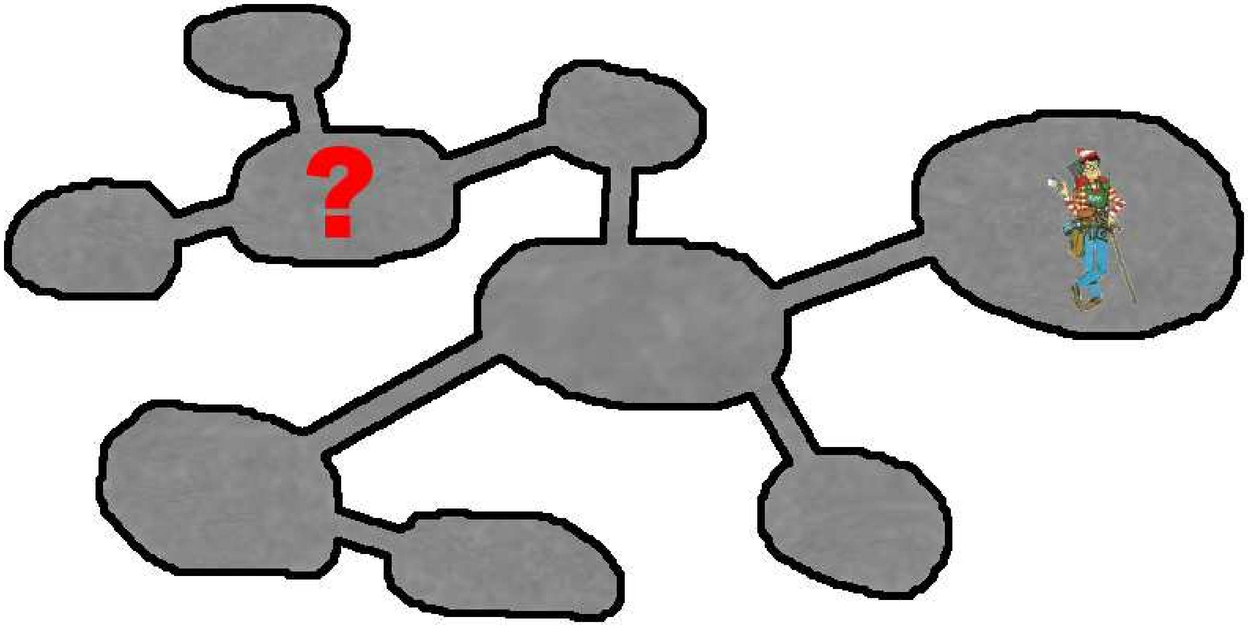
Two query models

1. Questions about vertices



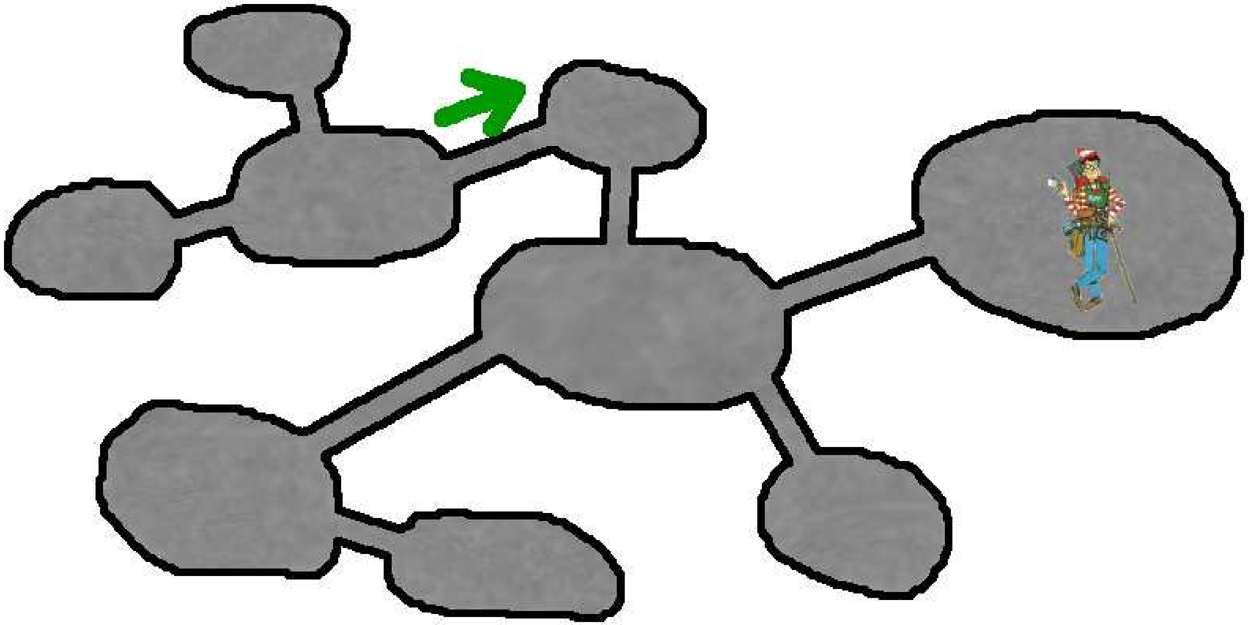
Two query models

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



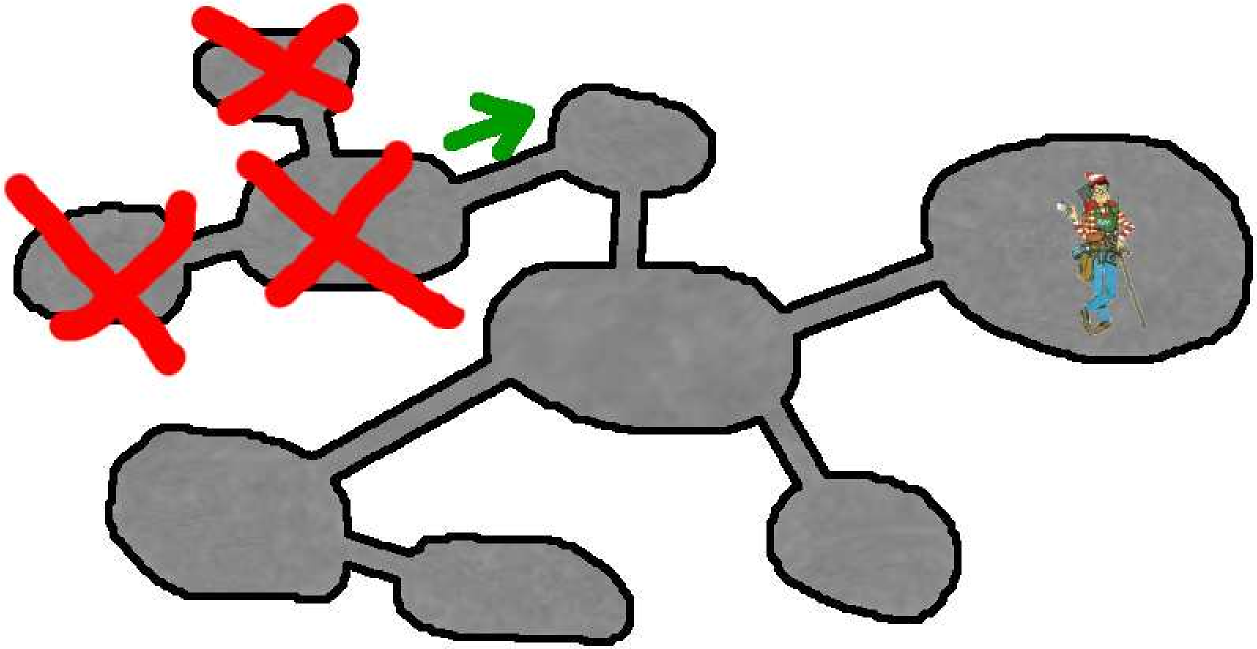
Two query models

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.



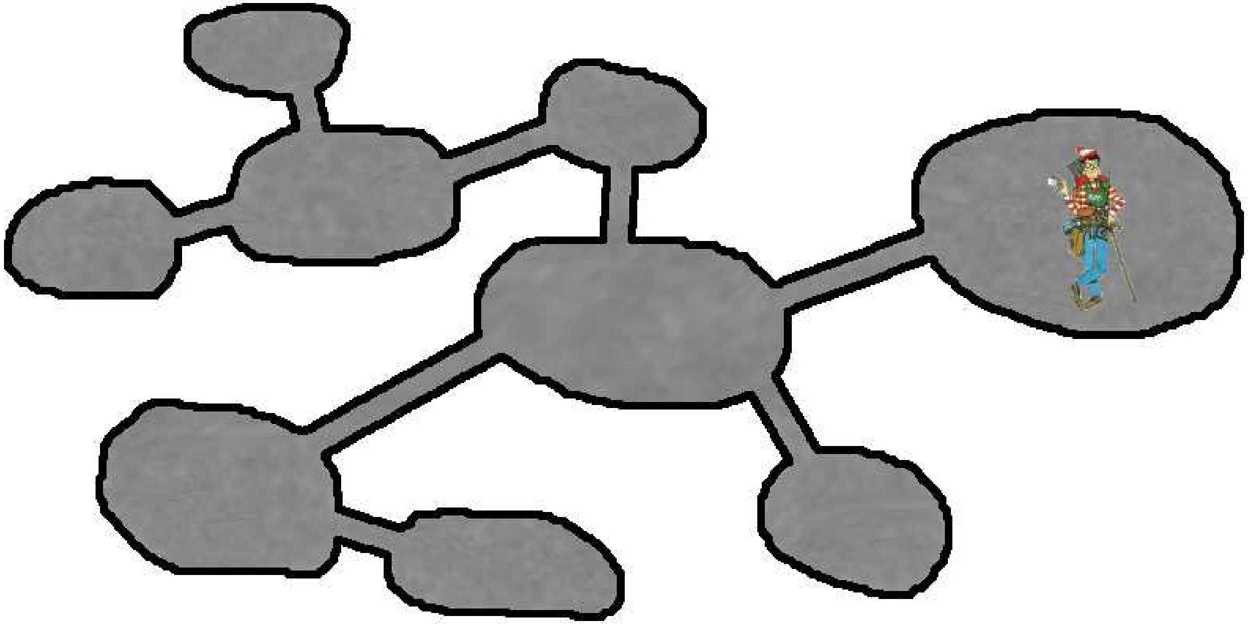
Two query models

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.



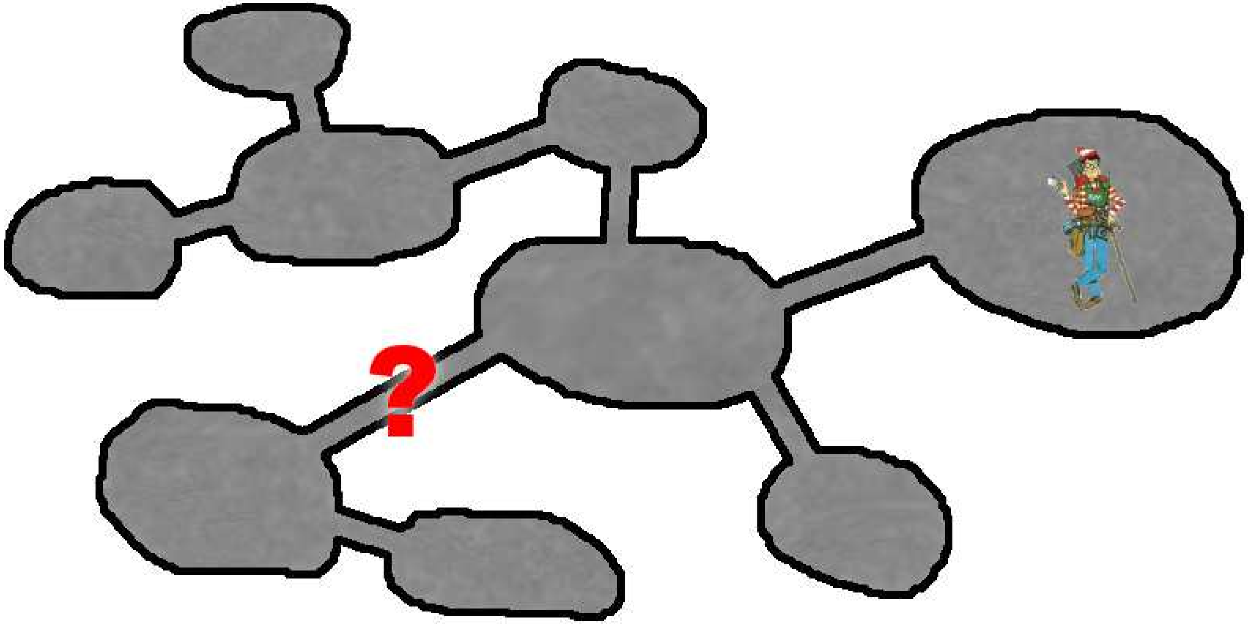
Two query models

1. Questions about vertices
2. Questions about edges



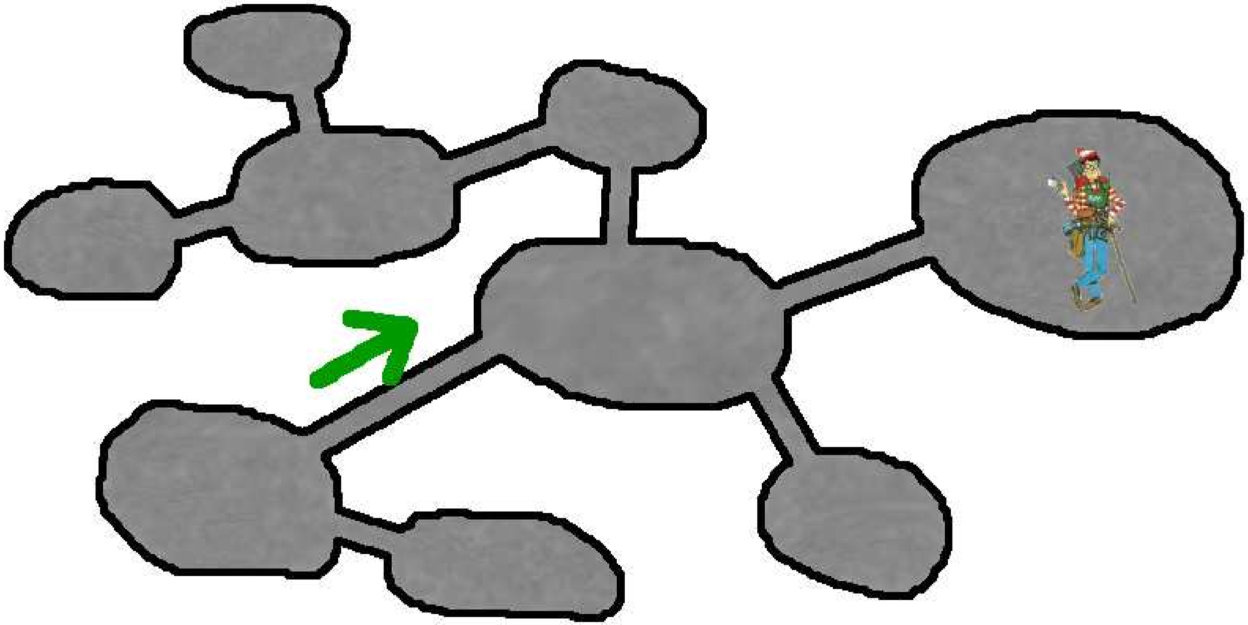
Two query models

1. Questions about vertices
2. Questions about edges
   - Ask about an edge-corridor e.



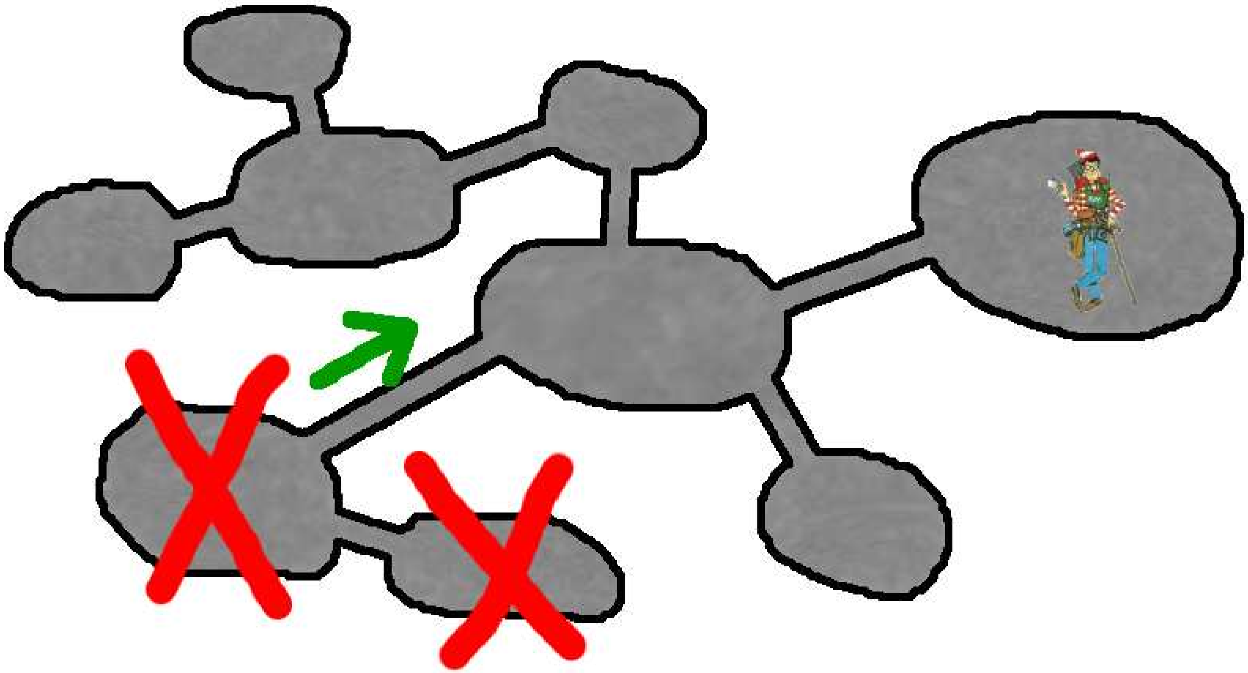
Two query models

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



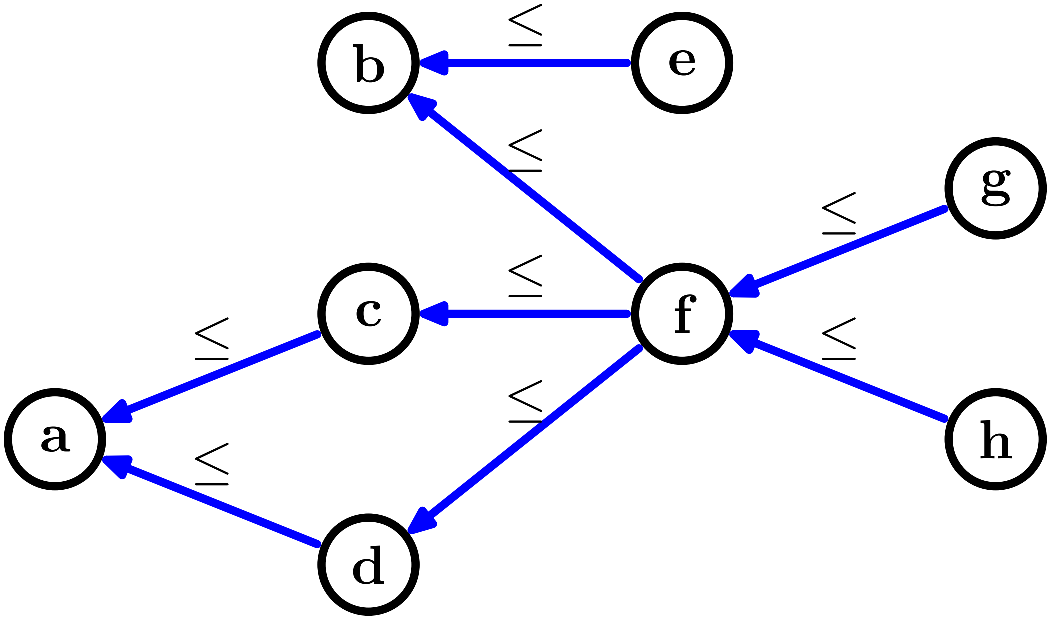
Two query models

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



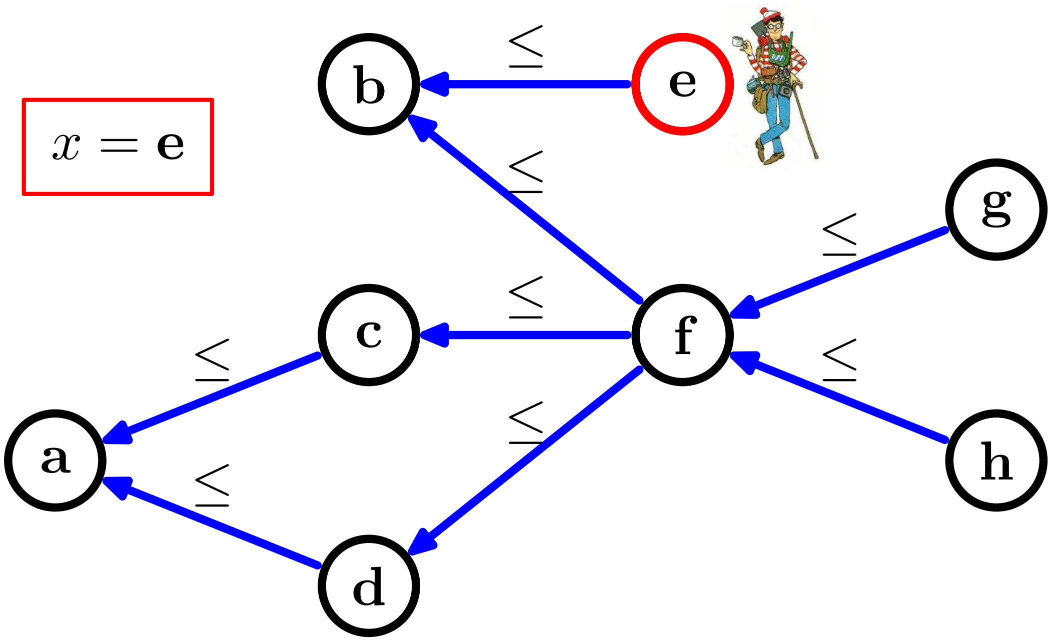
Searching in partial orders

- Given is a partial order S (or its diagram).



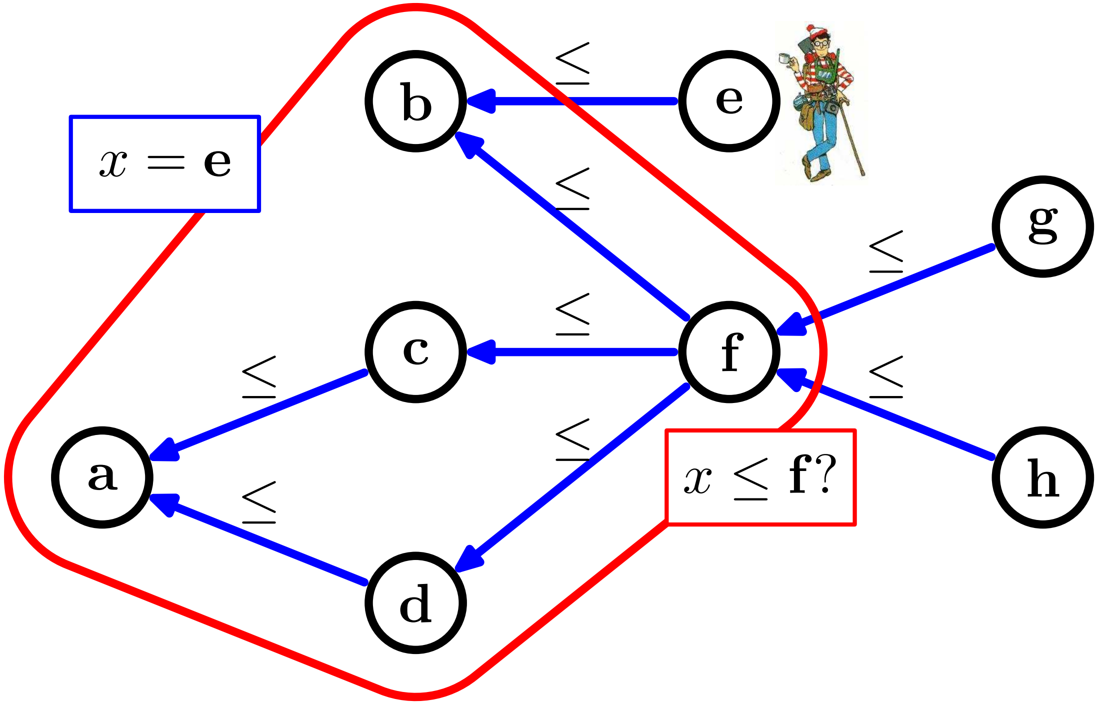
Searching in partial orders

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



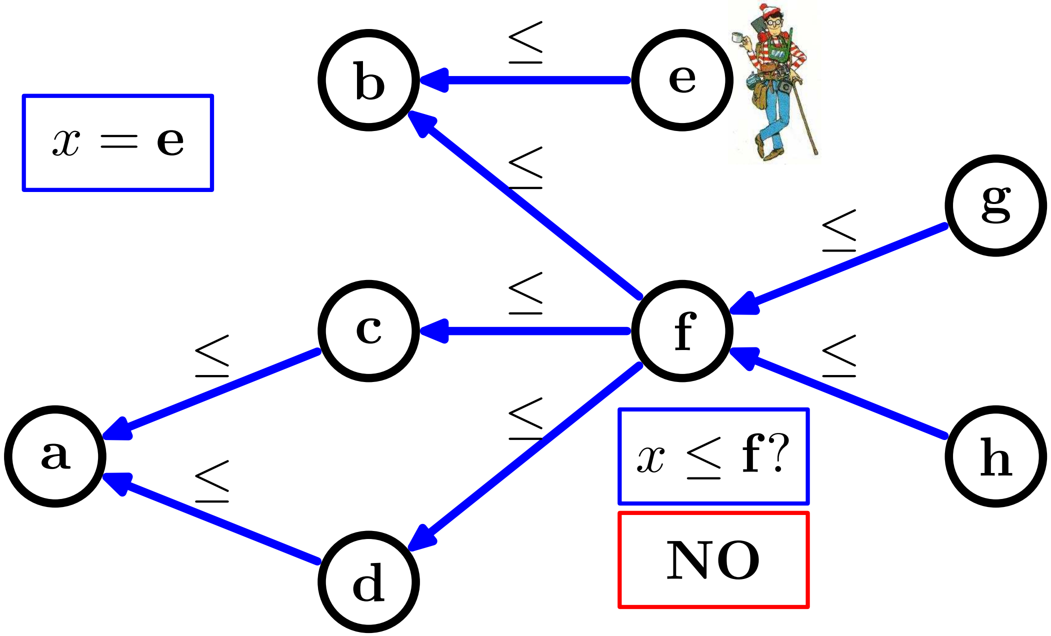
Searching in partial orders

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



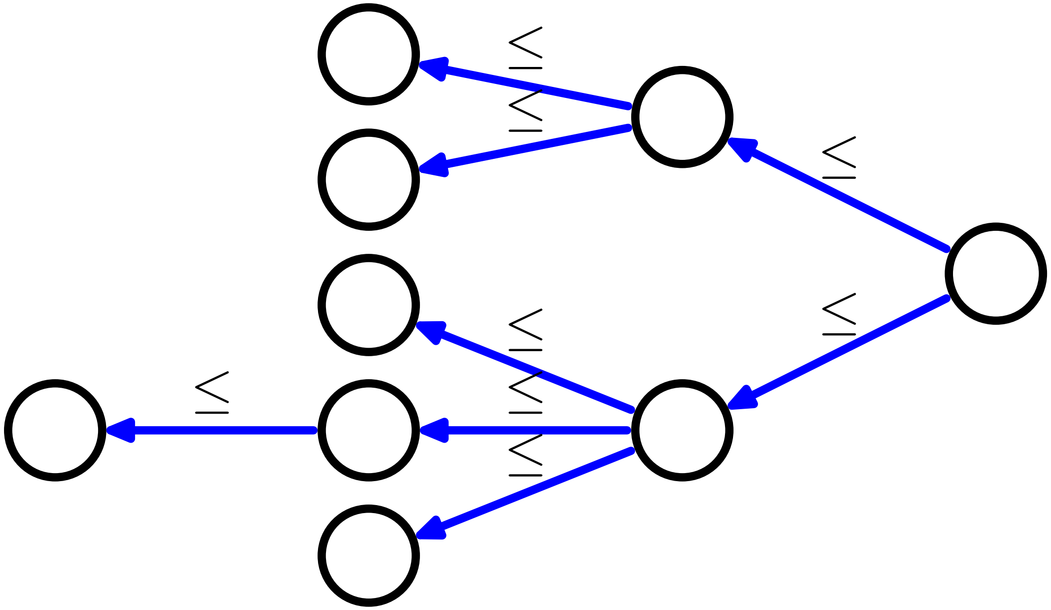
Searching in partial orders

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



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

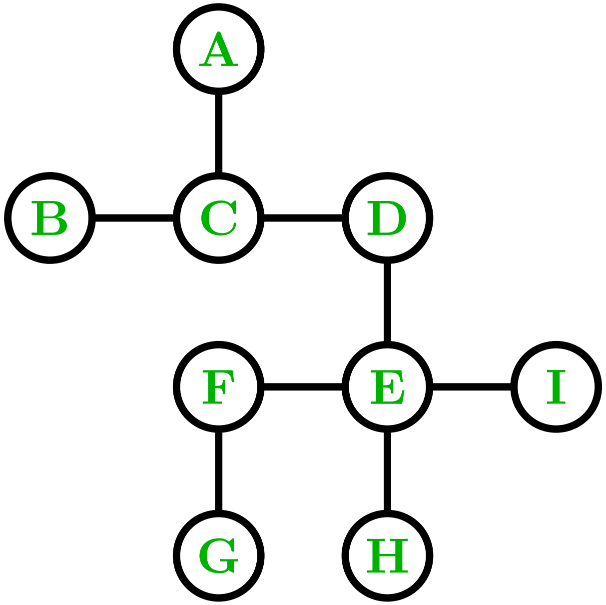
- By a strategy for a given problem we mean a decision tree for solving this problem.

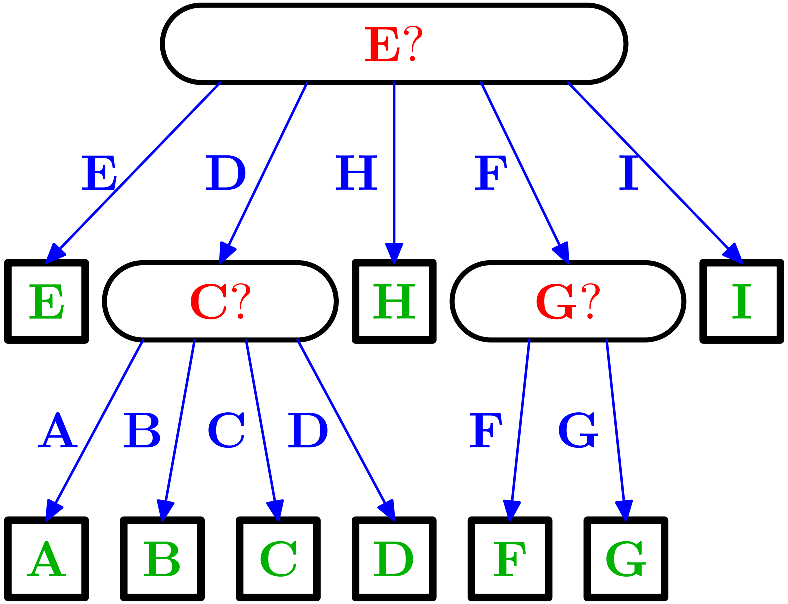
Optimal strategies

- 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.

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.
- 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.

General technique [OP 2006]

Short overview:

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

General technique [OP 2006]

Short overview:

- Reduce the problem to optimizing a strategy function.
- 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,
$$f : V \to \{0, 1, 2, \ldots\}.$$

Strategy functions

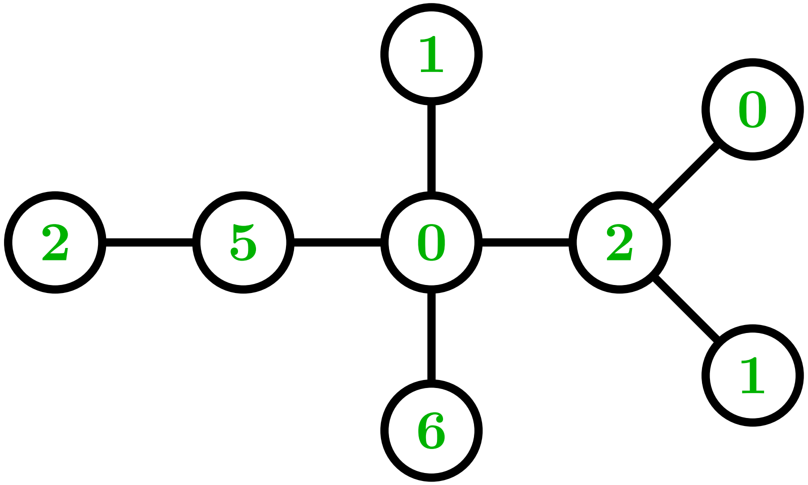
Strategy function:

- 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,
$$f : V \to \{0, 1, 2, \ldots\}.$$
- 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

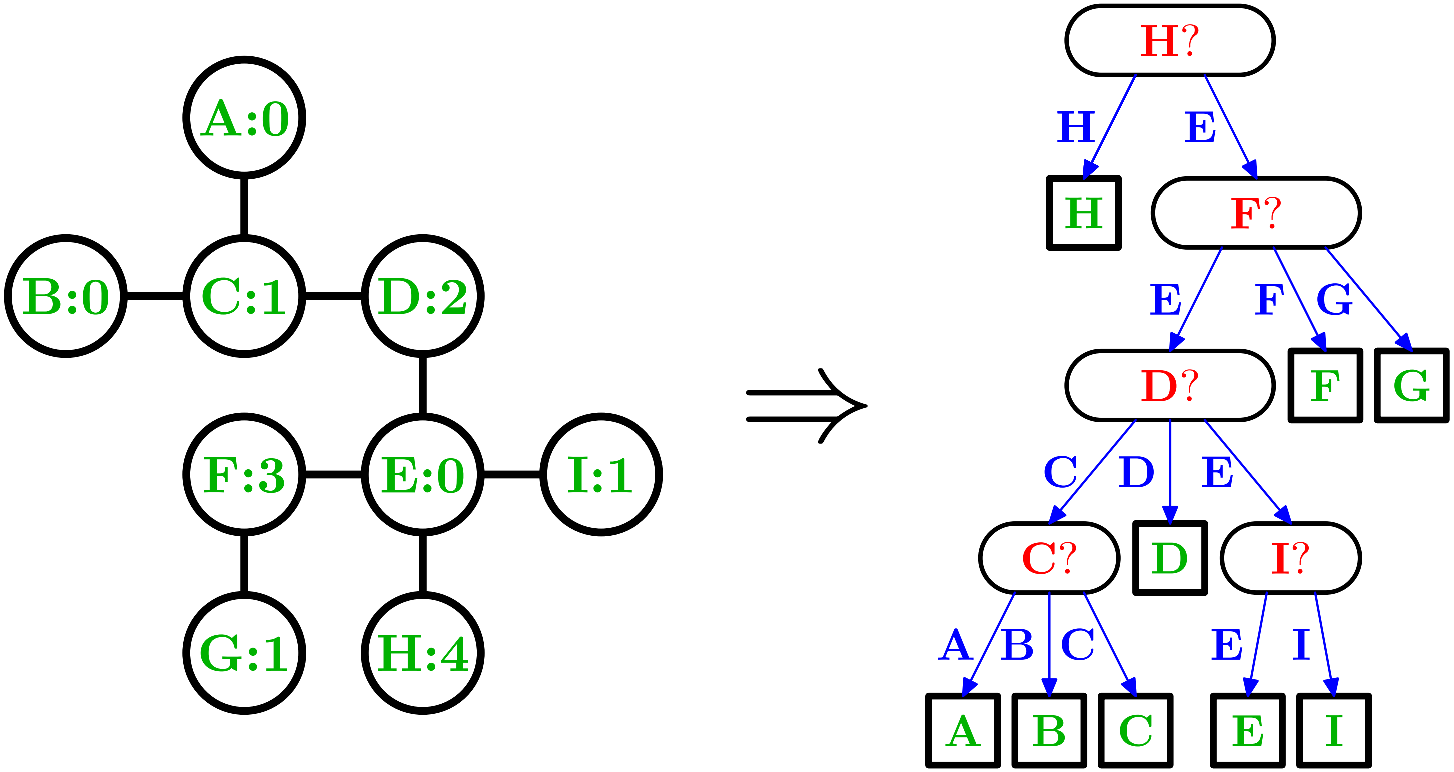
Idea: Ask about the vertex of the greatest value in the subtree induced by the potential solutions

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

$\Rightarrow$ a strategy function bounded by k

Mutual correspondence

A strategy of k queries in the worst case

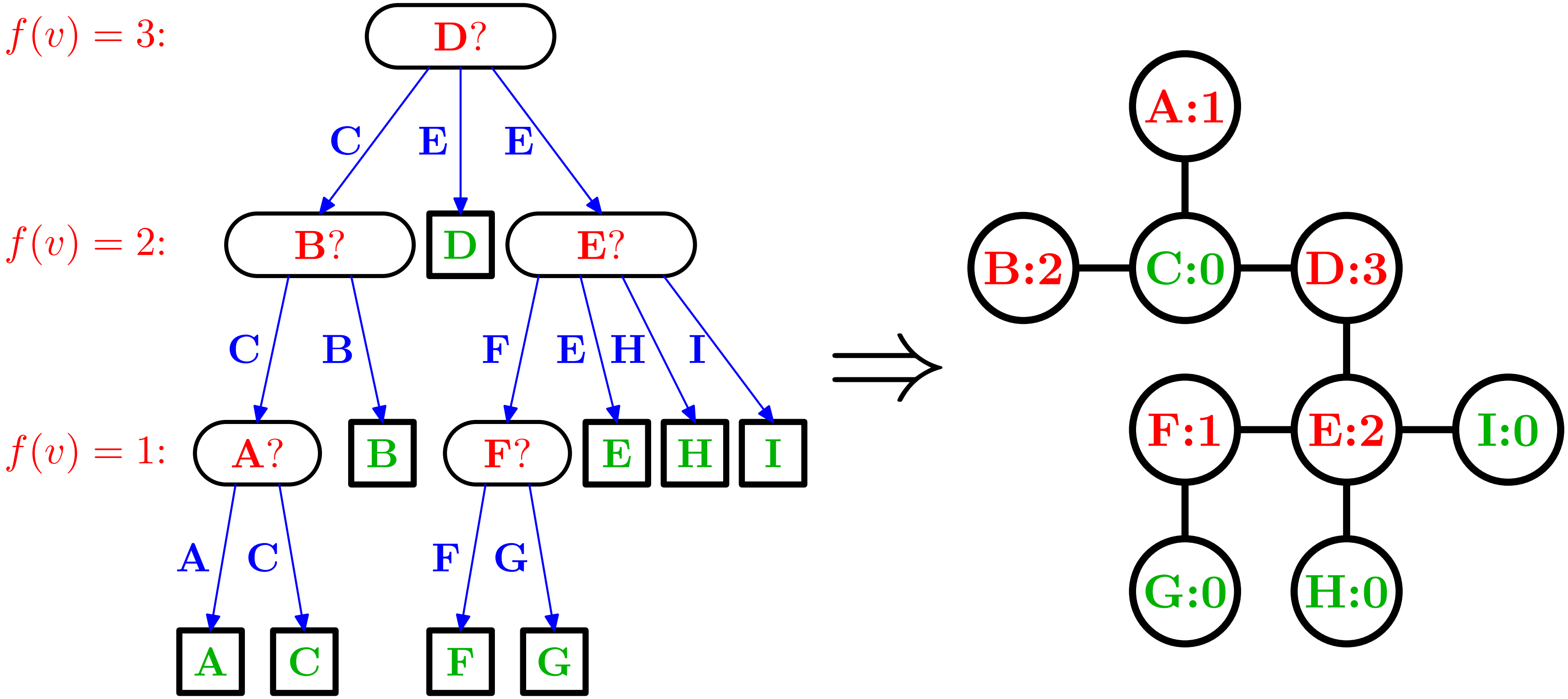
$\Rightarrow$ a strategy function bounded by k

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
$\Rightarrow$ 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.



Conclusion

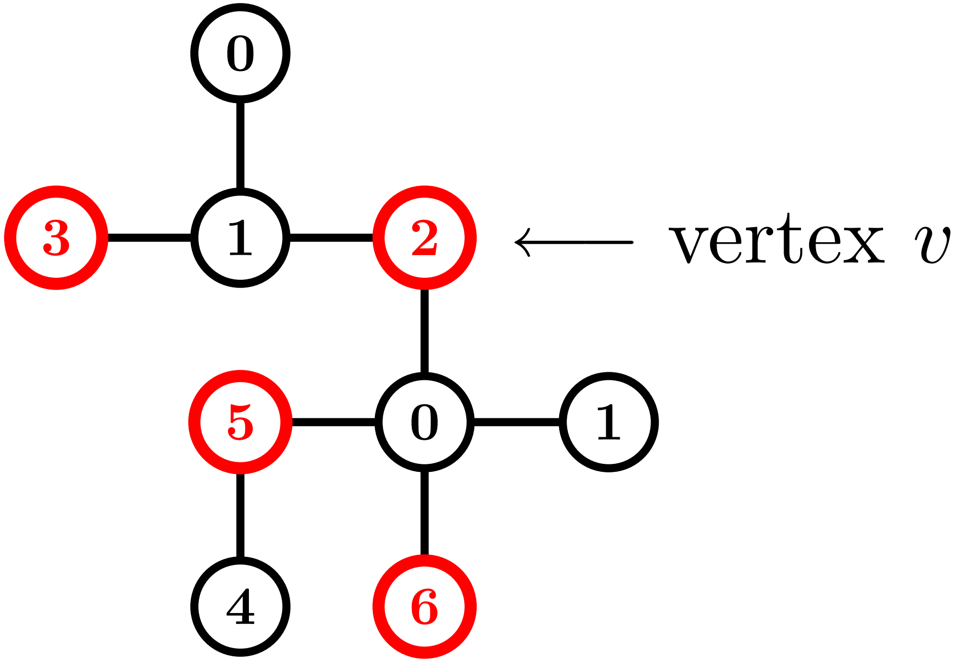
It suffices to construct a strategy function of the least maximum!

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.



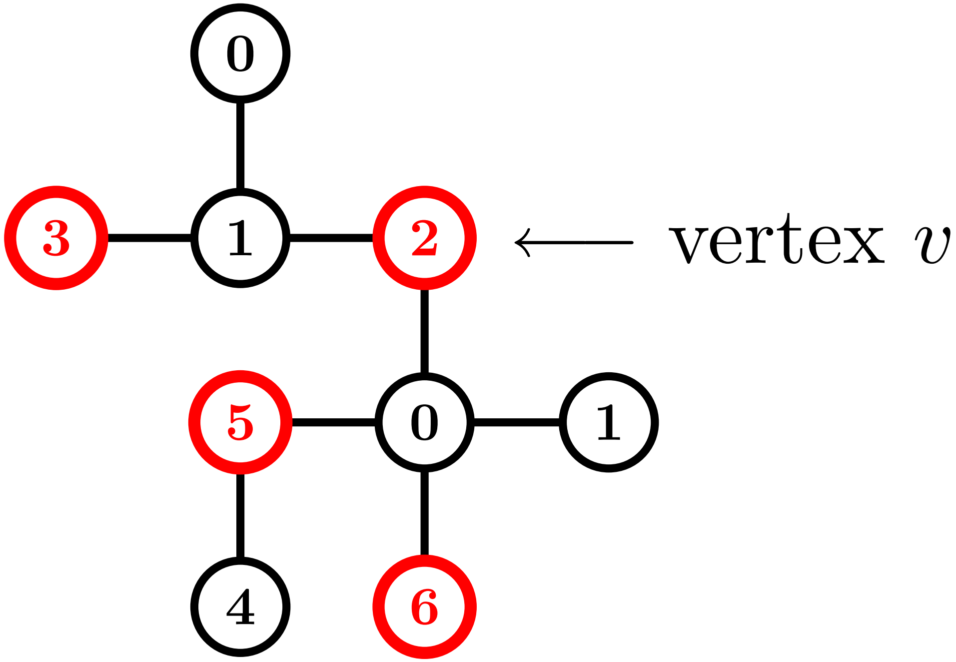
Values visible from v: 3, 2, 5, 6

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.

Visibility sequences

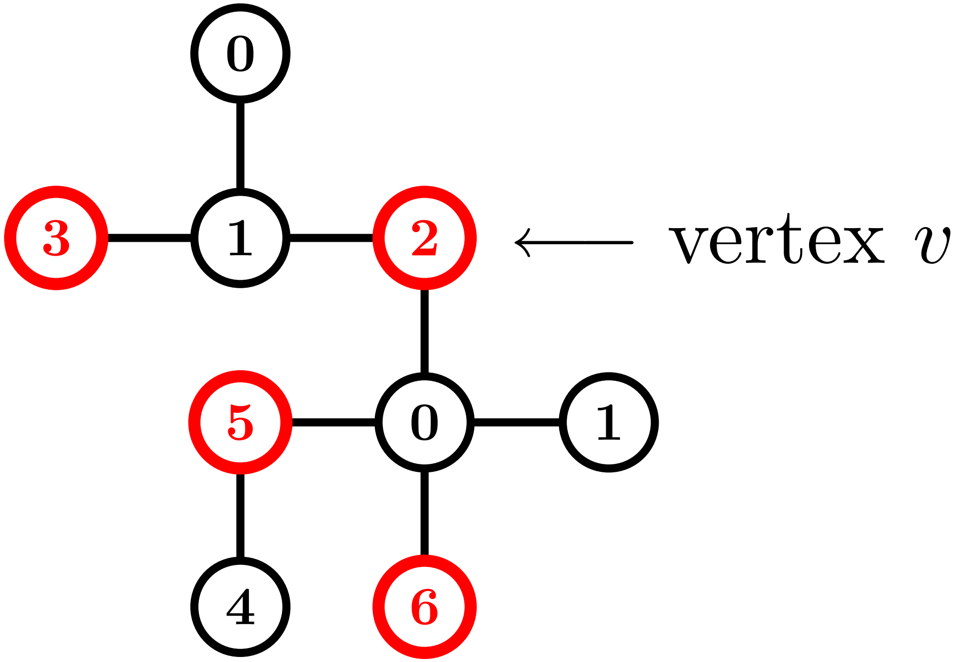
The visibility sequence from a vertex v is the sequence of all values visible from v, enumerated from the greatest to the least.



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

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 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 $f(v)$.

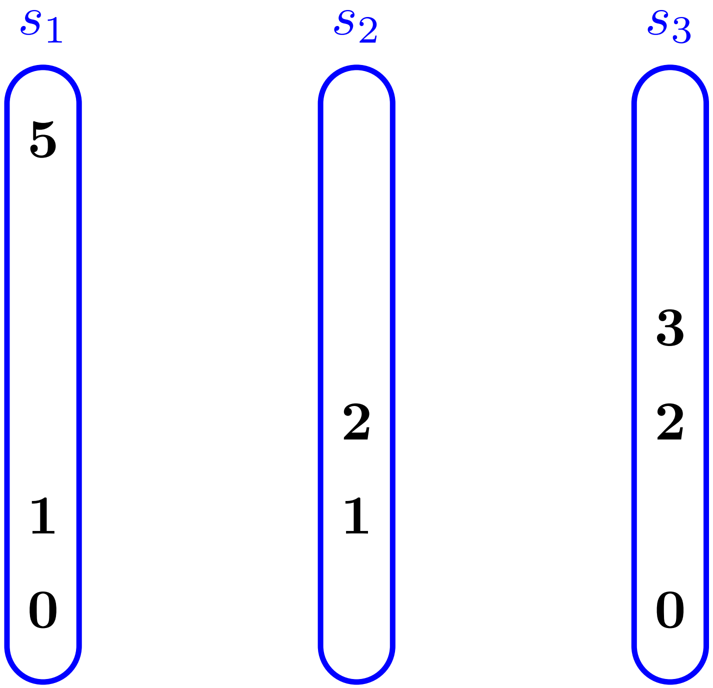
- To get a correct strategy function, it suffices to know the visibility sequences from children of v in their subtrees.
- An extension operator is a procedure that takes those visibility sequences, extends the function, and returns the visibility sequence from v in the subtree rooted at v.

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].

Vertex-query model

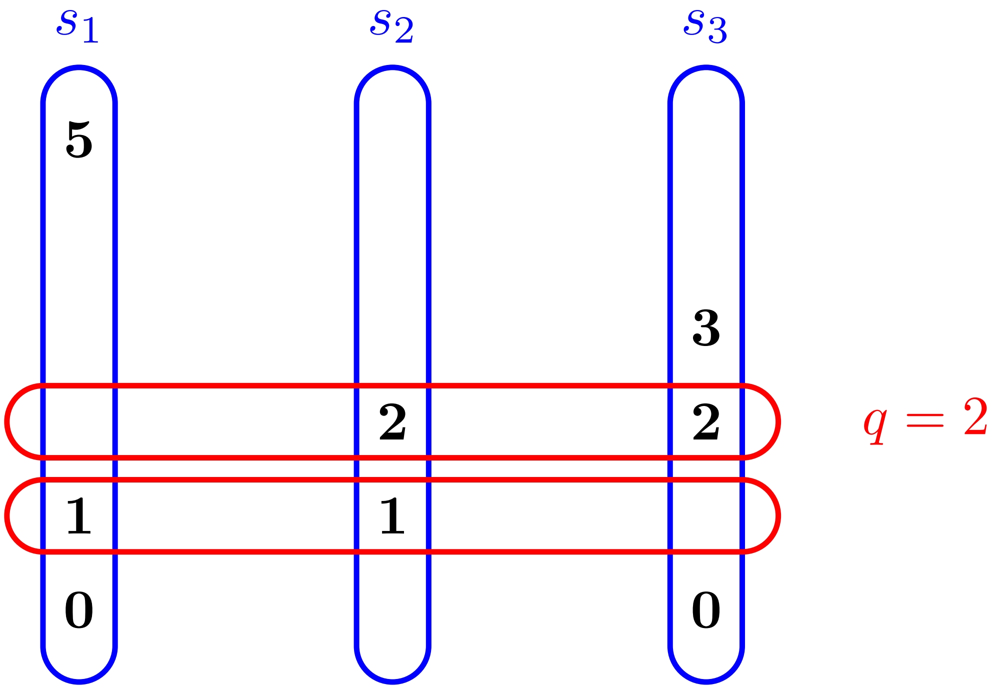
- An extension operator $\mathbb{V}$ for a vertex v:



Vertex-query model

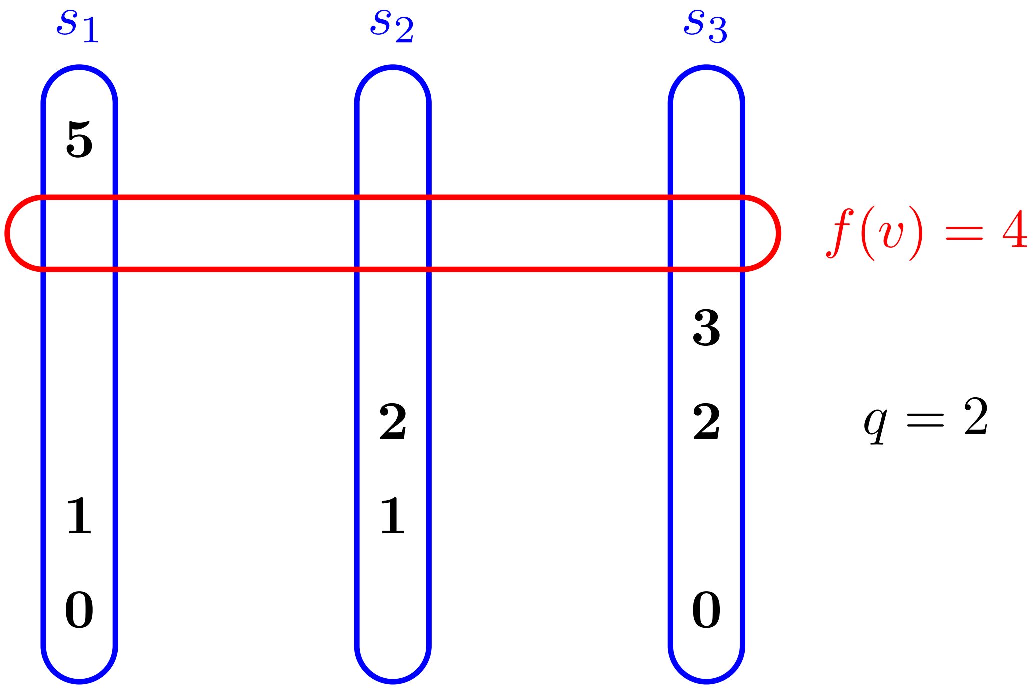
- An extension operator $\mathbb{V}$ for a vertex v:

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



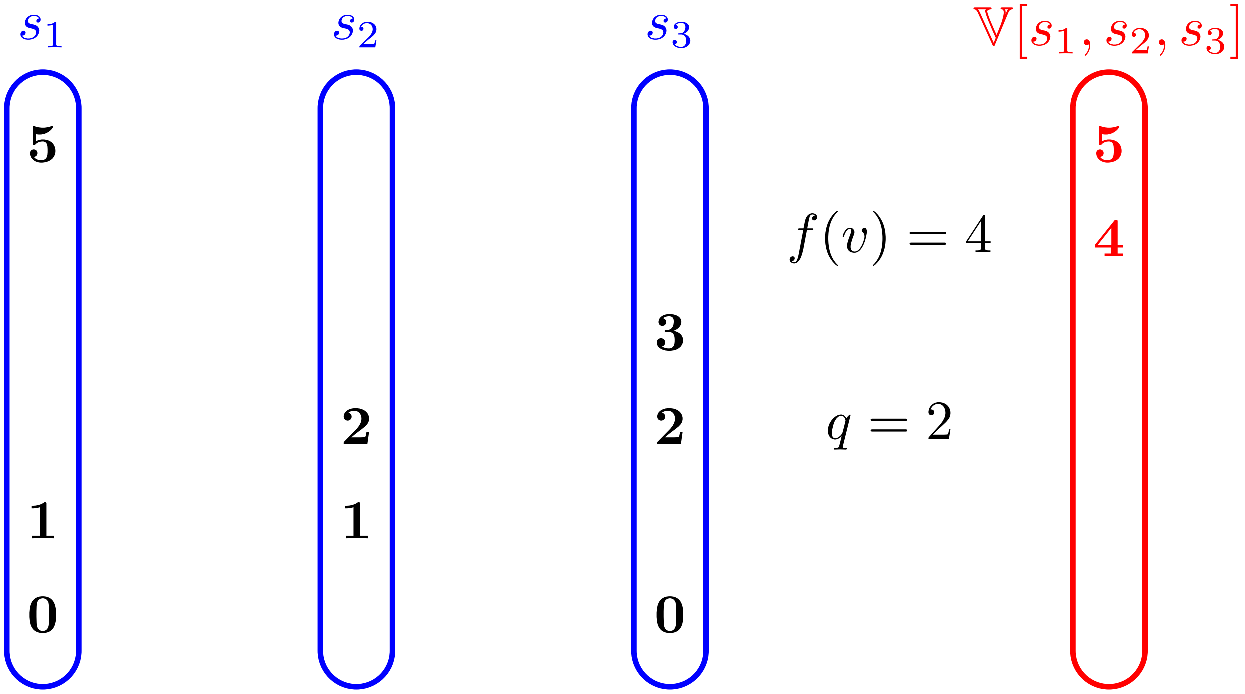
Vertex-query model

- An extension operator $\mathbb{V}$ 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.



Vertex-query model

- An extension operator $\mathbb{V}$ 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.



s_1	s_2	s_3		$\mathbb{V}[s_1, s_2, s_3]$
5				**5**
			$f(v) = 4$	**4**
		3		
	2	**2**	$q = 2$	
1	**1**			
0		**0**		

Vertex-query model

- One can show that $\mathbb{V}$ is minimizing.

Vertex-query model

- One can show that $\mathbb{V}$ 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 $\lfloor \log_2 n \rfloor$ queries.

Vertex-query model

- One can show that $\mathbb{V}$ is minimizing.

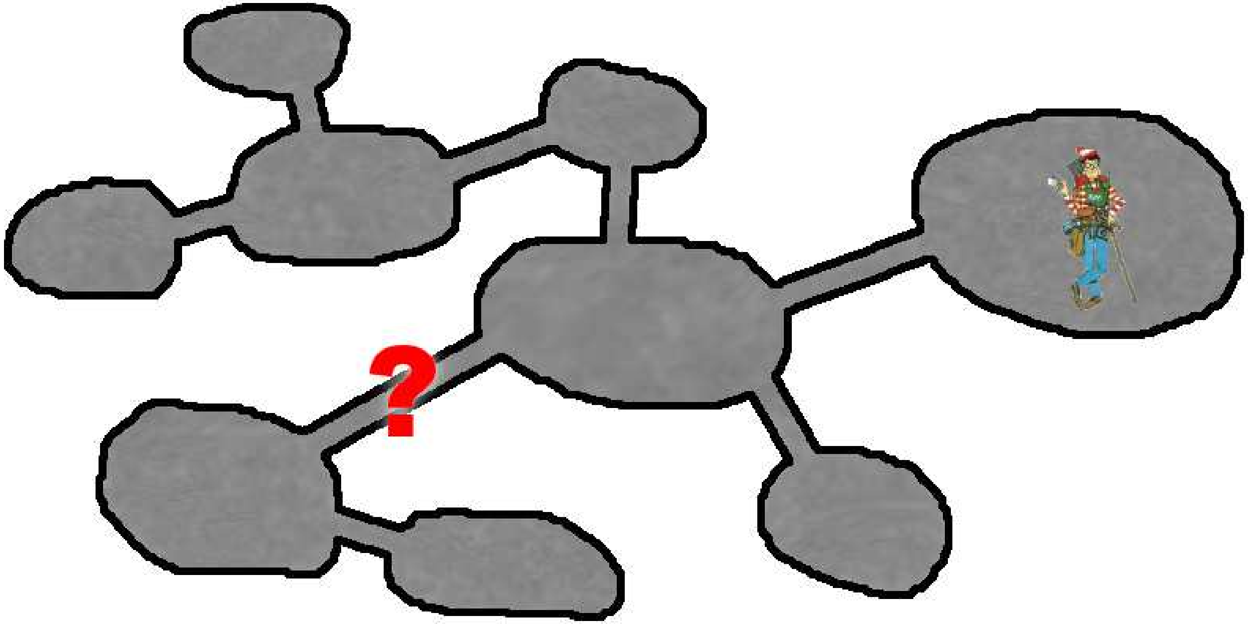
- The whole computation takes $O(n \log n)$ time, as in the vertex-query model the required vertex can always be located in at most $\lfloor \log_2 n \rfloor$ queries.

- The running time can be improved to $O(n)$ fairly simple.

Edge-query model

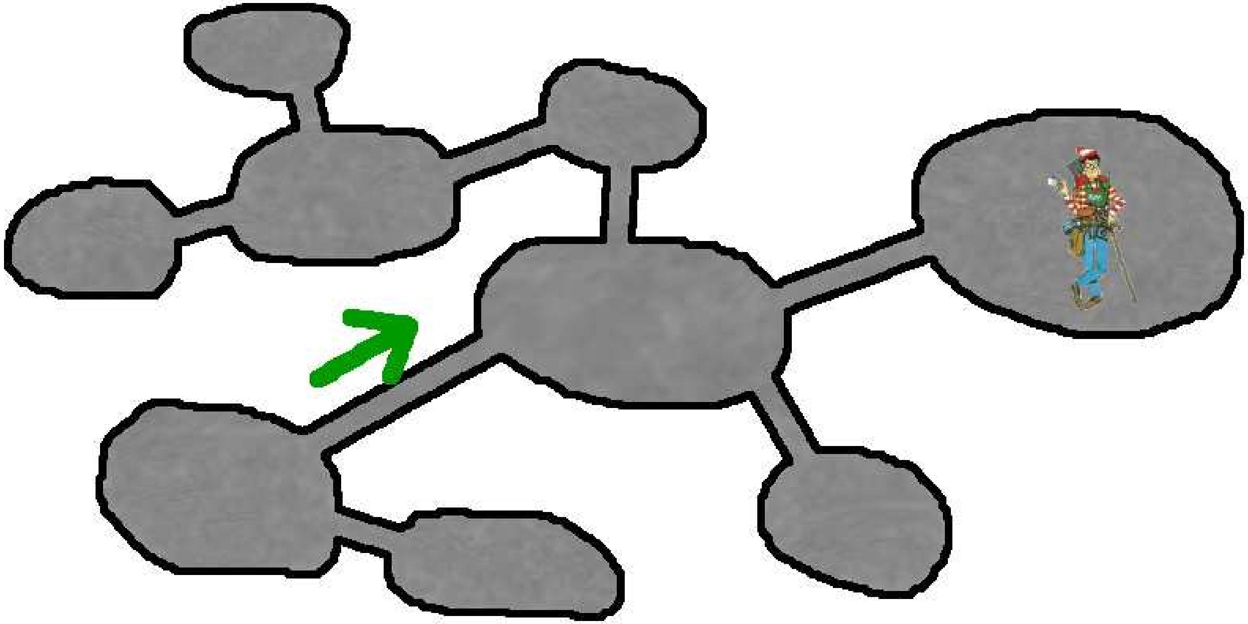
Edge-query model

- Questions about edges.



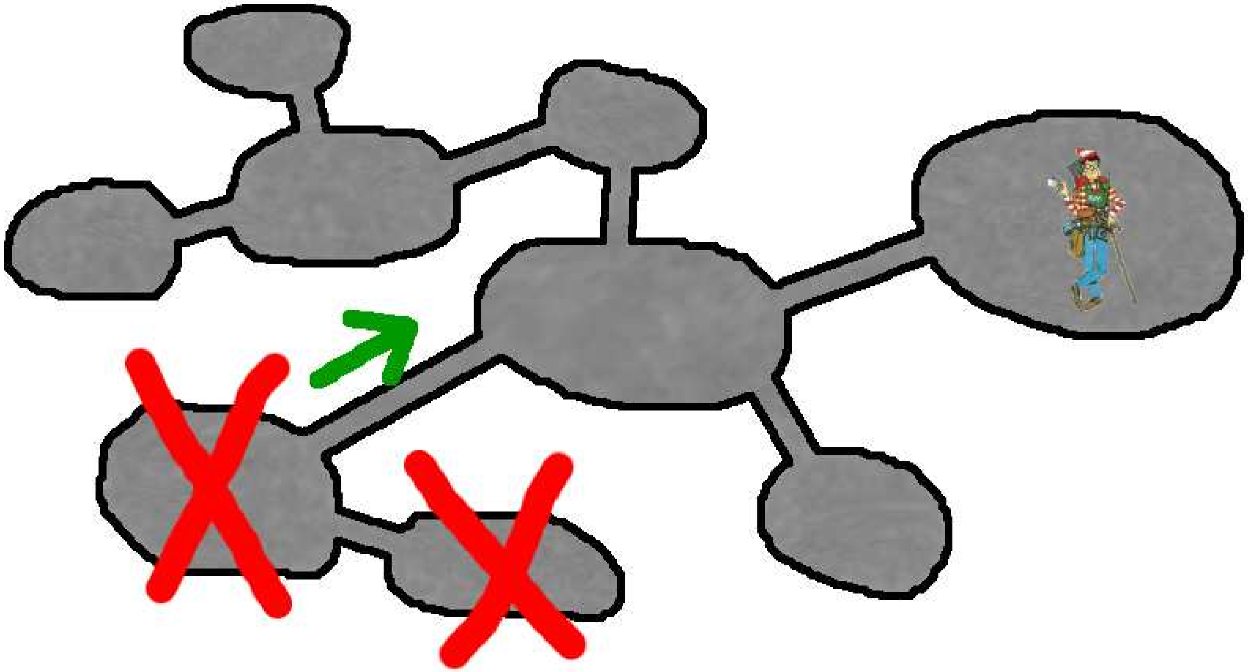
Edge-query model

- Questions about edges.
    - Ask about an edge e.



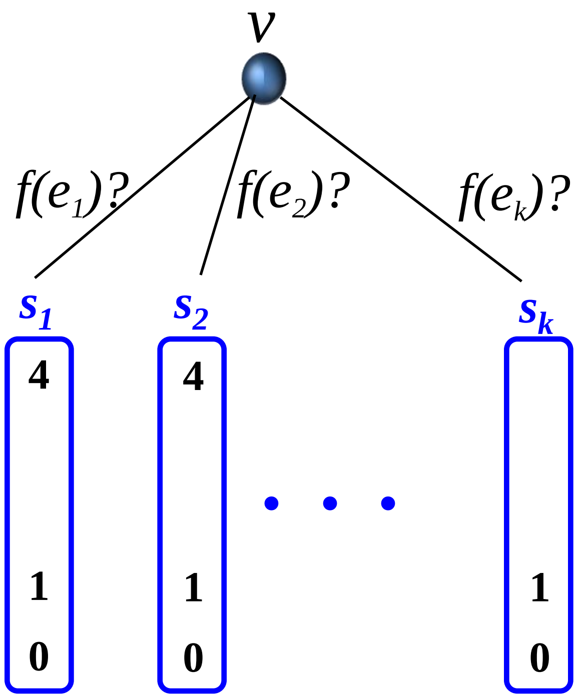
Edge-query model

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



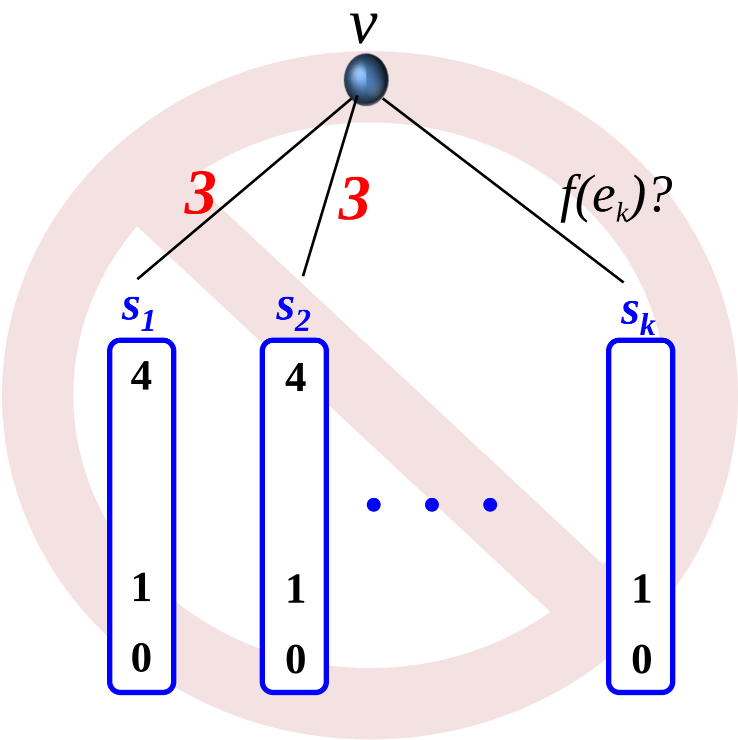
Edge-query model

- An extension assigns all $f(e_i)$'s



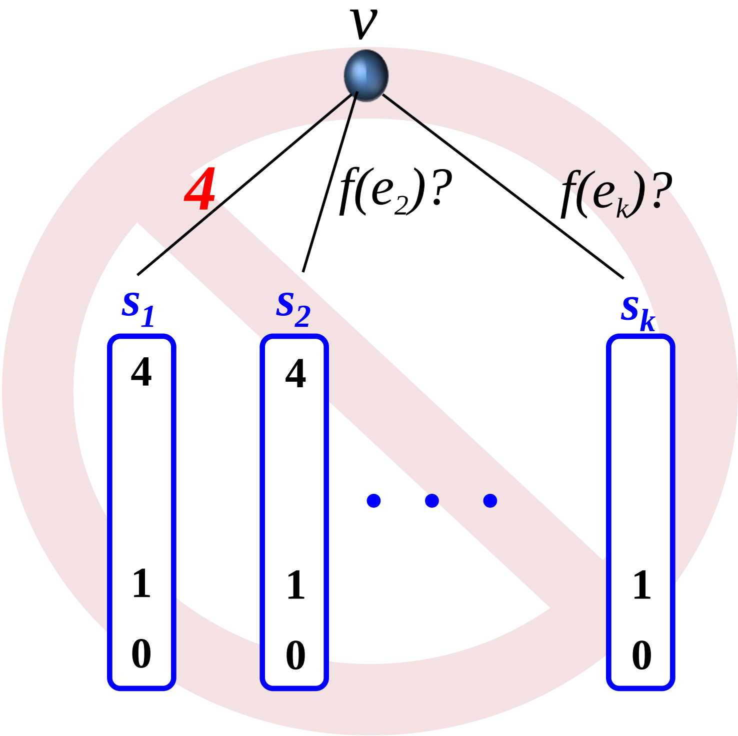
Edge-query model

- An extension assigns all $f(e_i)$'s
  - $f(e_i) \neq f(e_j)$



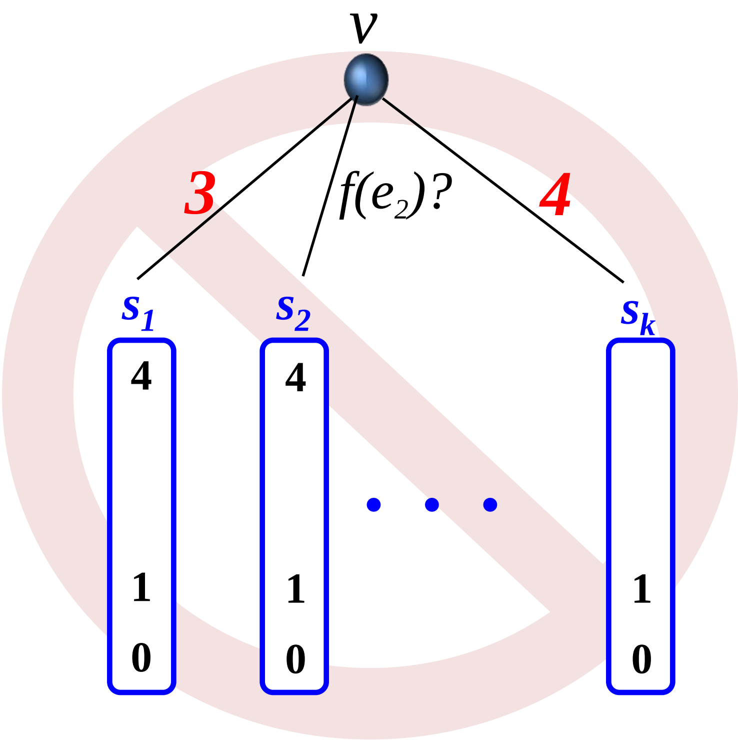
Edge-query model

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



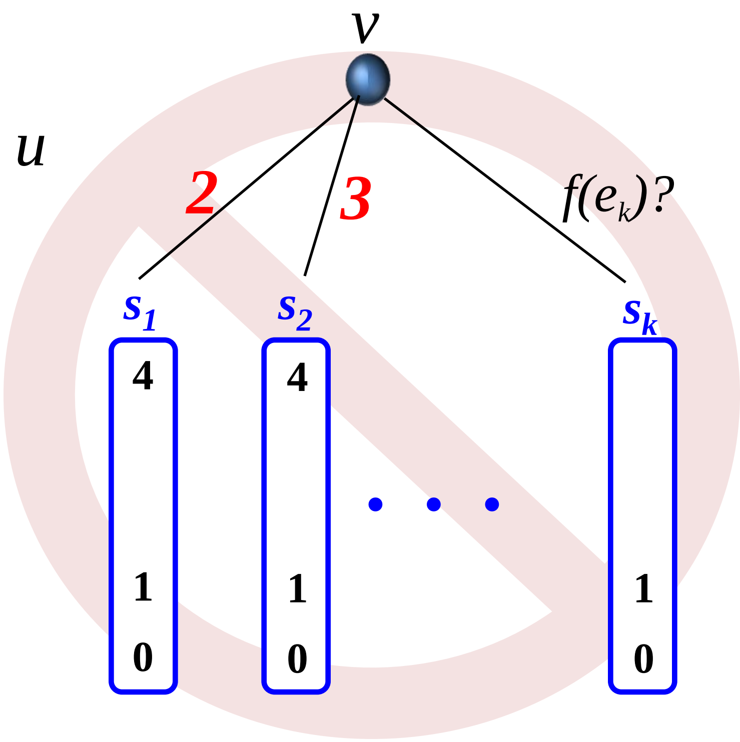
Edge-query model

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

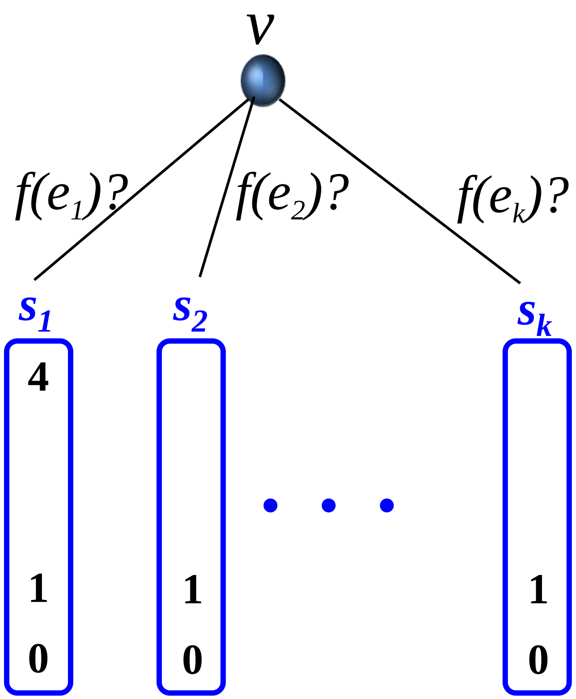


Edge-query model

- An extension assigns all $f(e_i)$'s
  - $f(e_i) \neq f(e_j)$
  - $f(e_i)$ is not in s_i
  - $f(e_i)$ is in s_j ⇨ $f(e_j) > f(e_i)$
  - u is in s_i and s_j ⇨ $max\{f(e_i), f(e_j)\} > u$



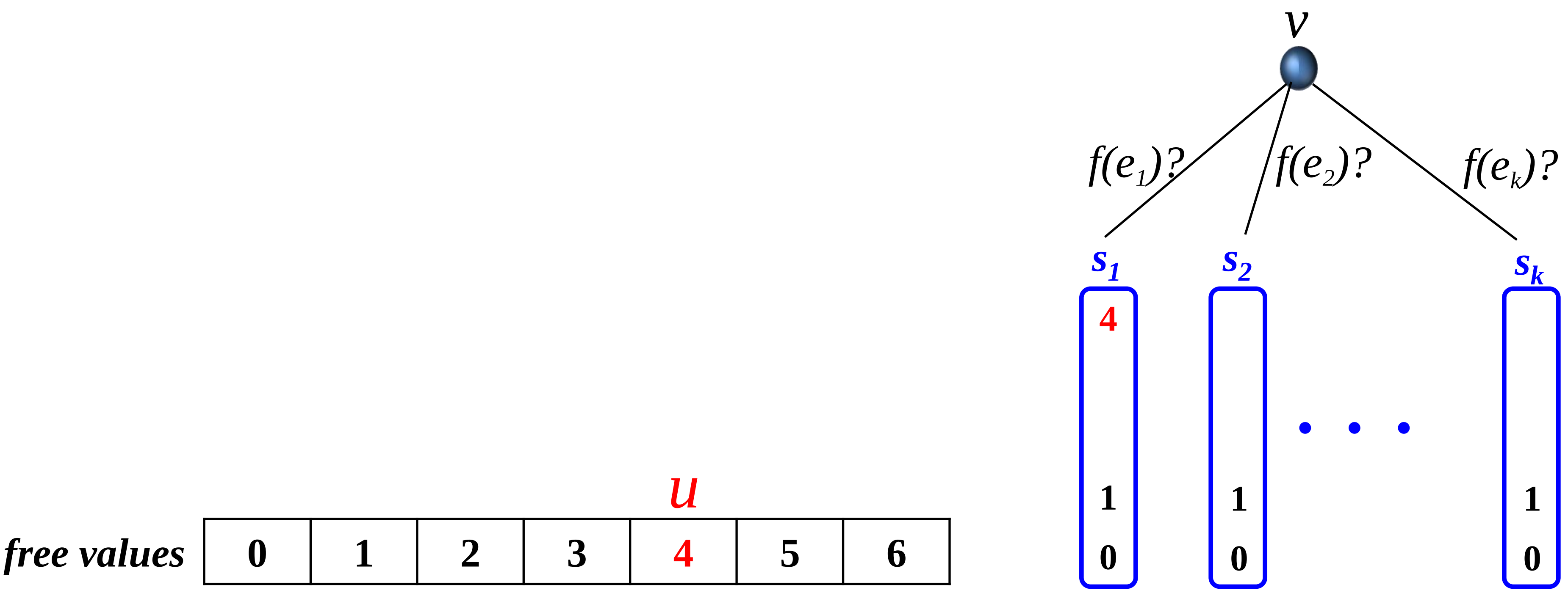
Algorithm Outline



free values	0	1	2	3	4	5	6

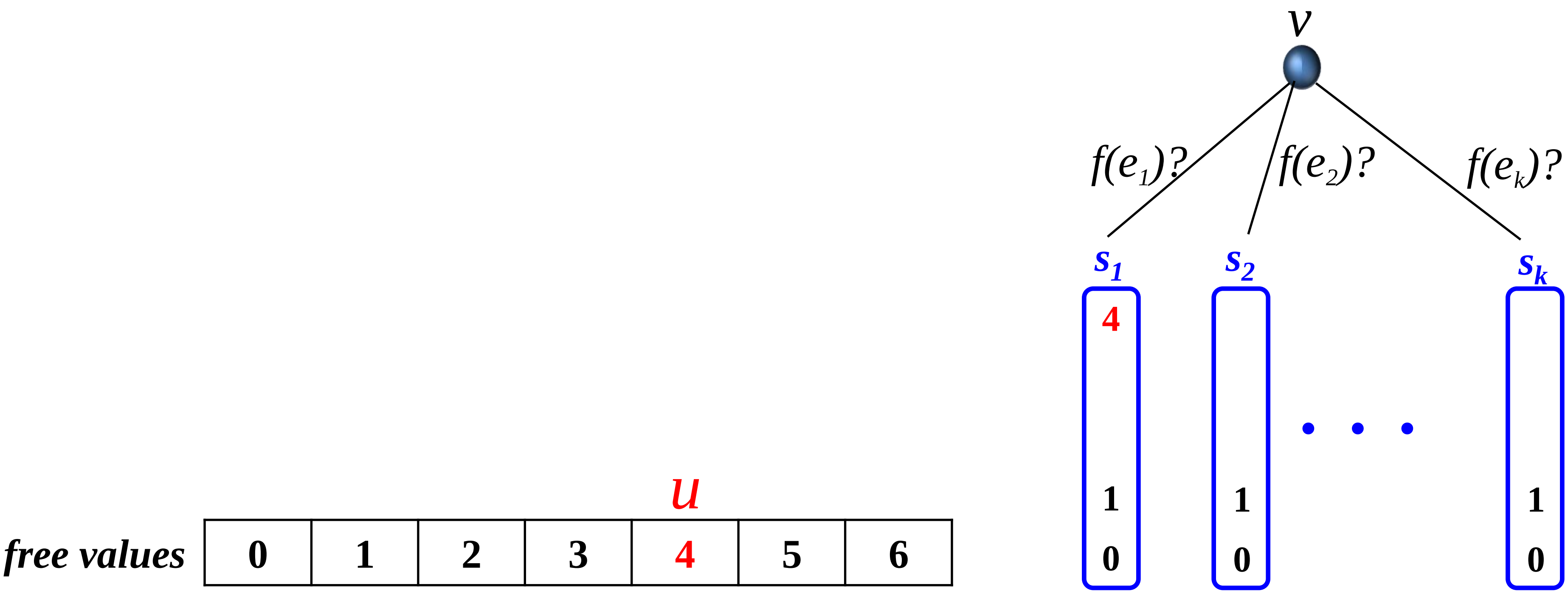
Algorithm Outline

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



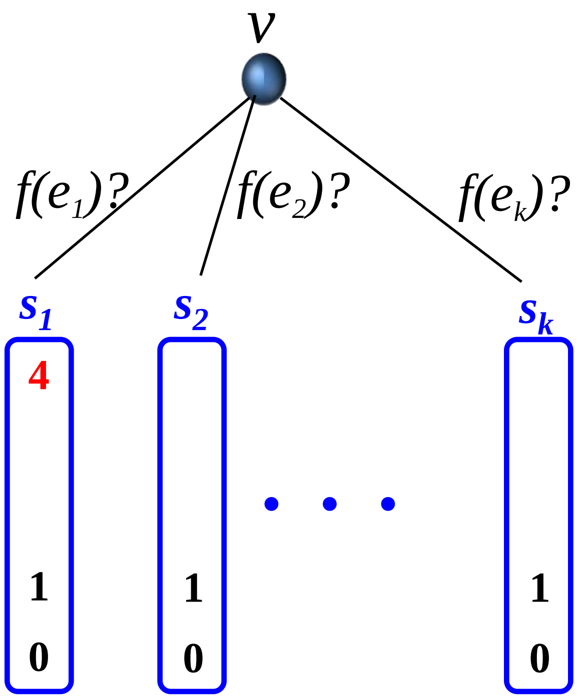
Algorithm Outline

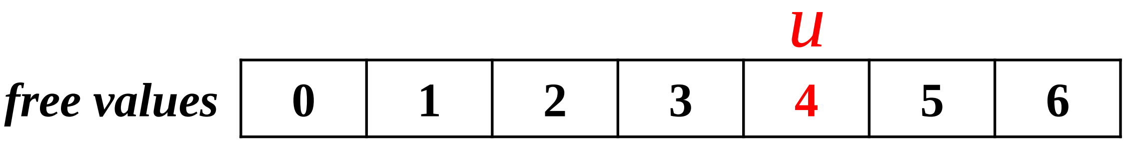
- set $u = max\{s_i\}$
- while not all edges assigned



Algorithm Outline

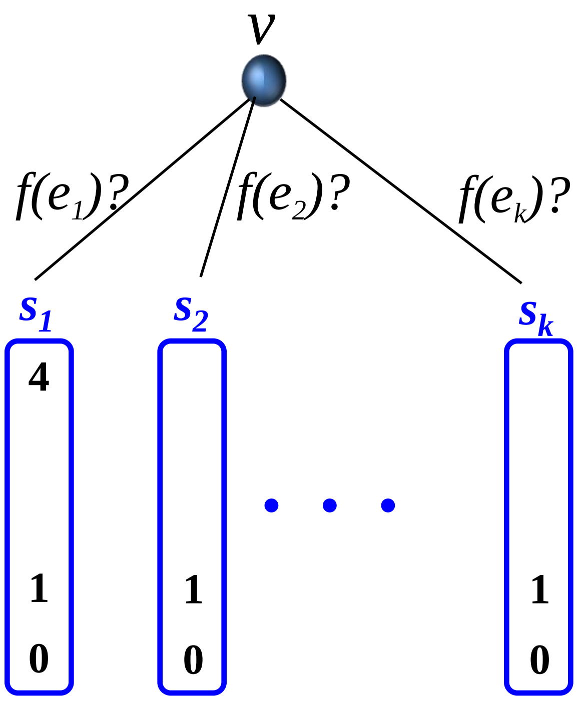
- set $u = max\{s_i\}$
- while not all edges assigned
  - if u appears once mark u as *not free*, move to next largest u





Algorithm Outline

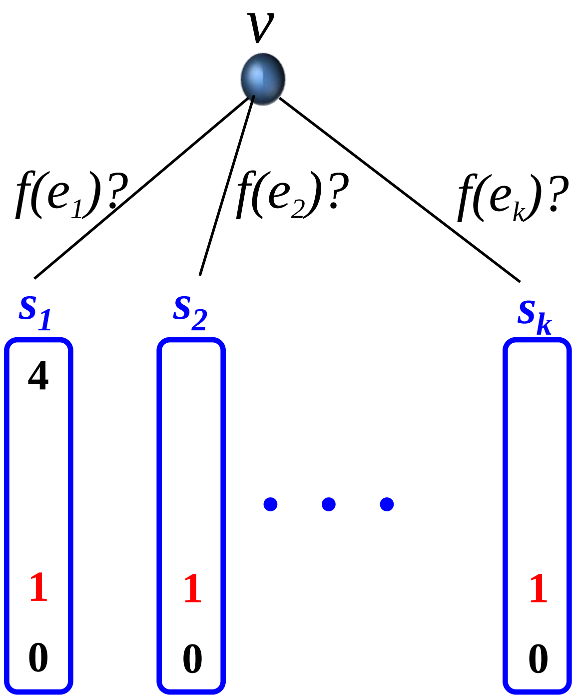
- set $u = max\{s_i\}$
- while not all edges assigned
  - if u appears once mark u as *not free*, move to next largest u



free values	**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 *not free*, move to next largest u
  - otherwise:

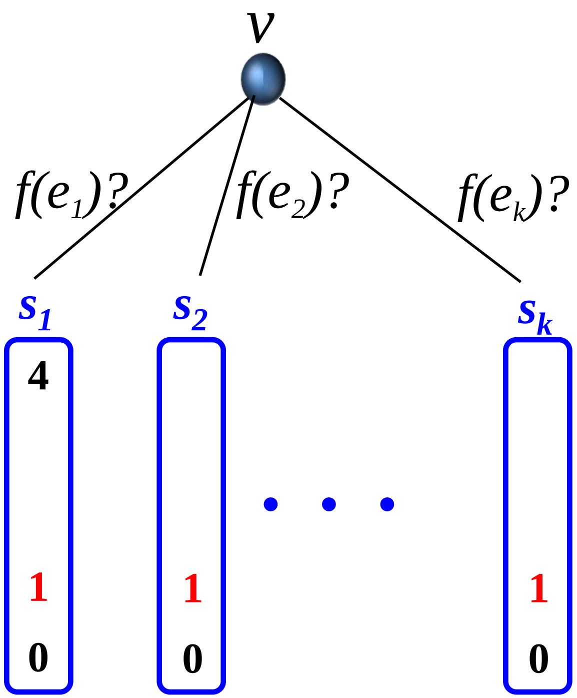


u

free values	**0**	**1**	**2**	**3**		**5**	**6**

Algorithm Outline

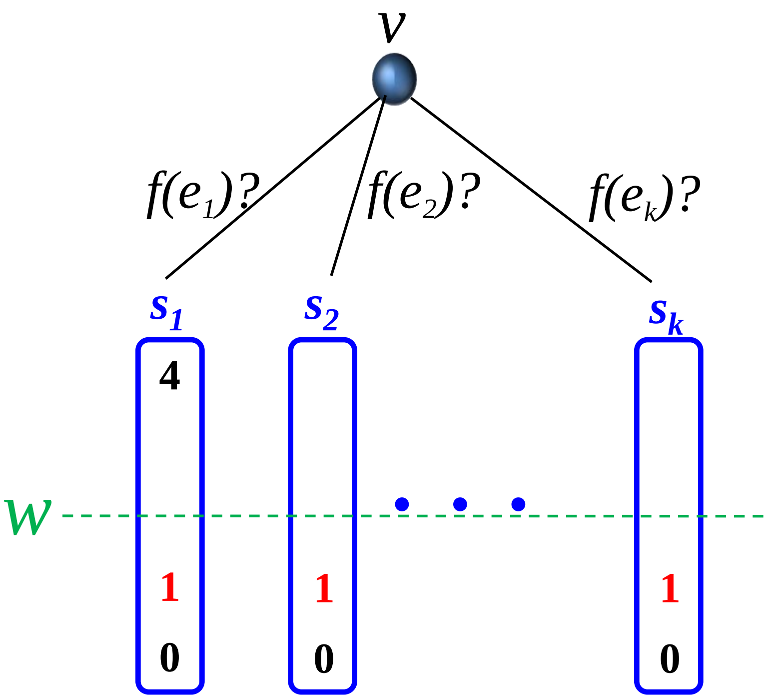
- set $u = max\{s_i\}$
- while not all edges assigned
  - if u appears once mark u as *not free*, move to next largest u
  - otherwise:
    - w = smallest free value > u



		u	w				
free values	**0**	**1**	**2**	**3**		**5**	**6**

Algorithm Outline

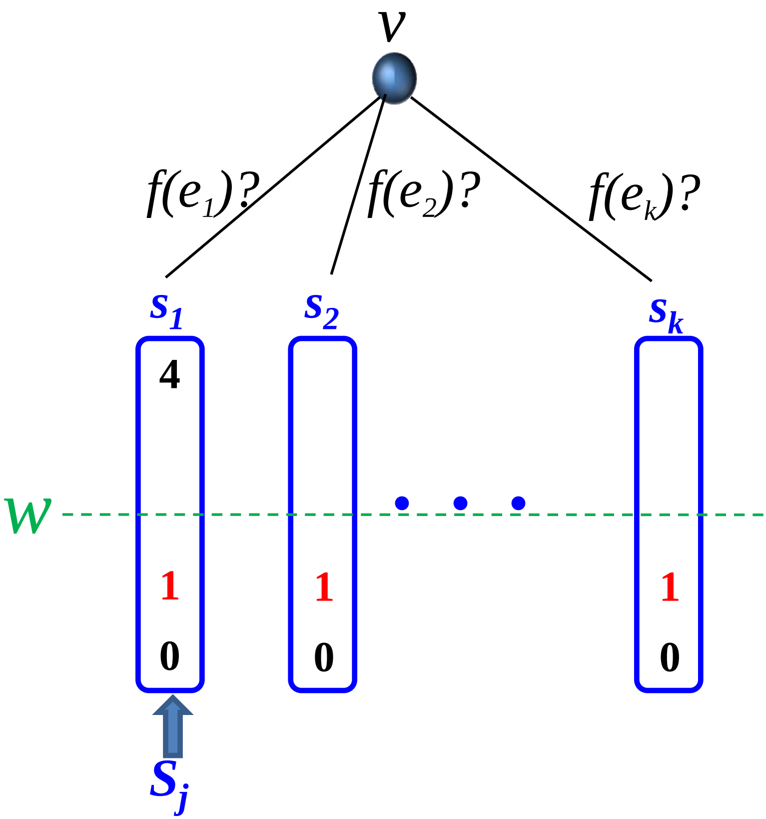
- set $u = max\{s_i\}$
- while not all edges assigned
  - if u appears once mark u as *not free*, move to next largest u
  - otherwise:
    - w = smallest free value > u
    - S_j = **any** maximal sequence w.r.t w



		u	*w*				
free values	0	1	2	3		5	6

Algorithm Outline

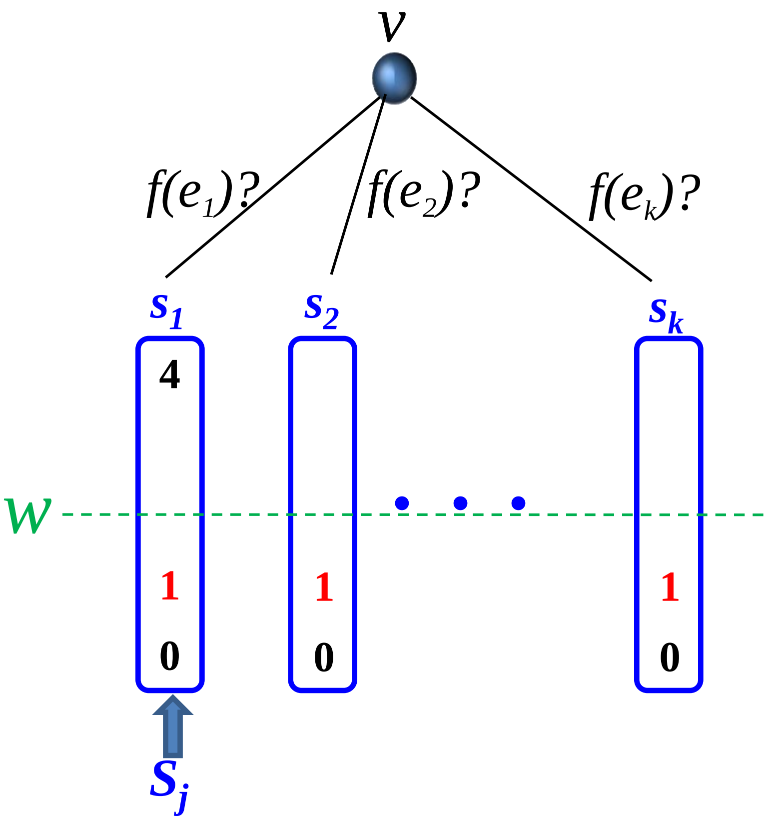
- set $u = max\{s_i\}$
- while not all edges assigned
  - if u appears once mark u as *not free*, move to next largest u
  - otherwise:
    - w = smallest free value > u
    - S_j = any maximal sequence w.r.t w



		u	w				
free values	**0**	**1**	**2**	**3**		**5**	**6**

Algorithm Outline

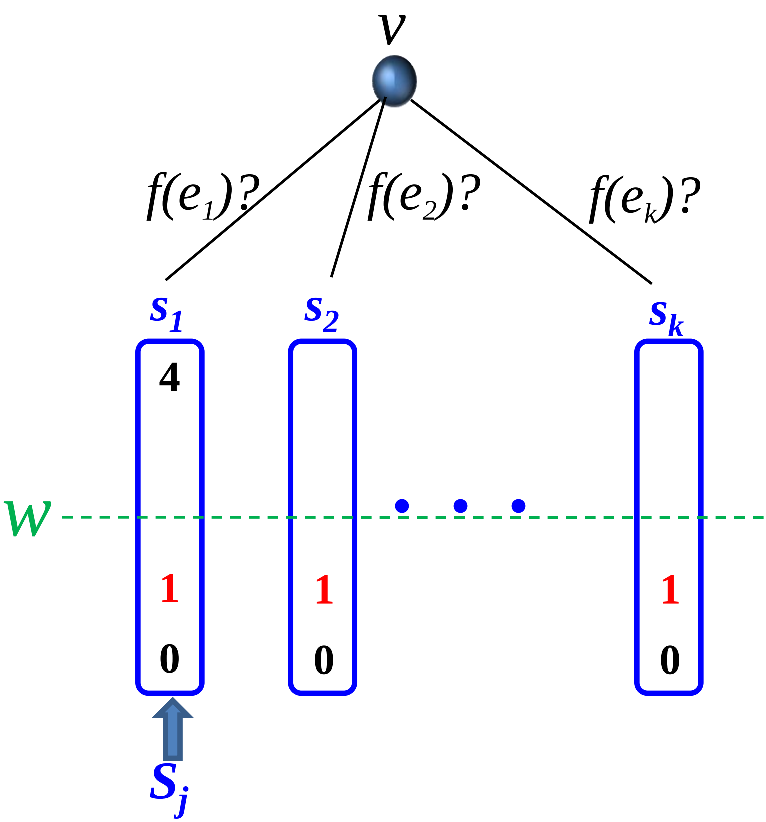
- set $u = max\{s_i\}$
- while not all edges assigned
  - if u appears once mark u as *not free*, move to next largest u
  - otherwise:
    - w = smallest free value > u
    - S_j = any maximal sequence w.r.t w
    - mark w as *not free*



		u	w				
free values	0	1	2	3		5	6

Algorithm Outline

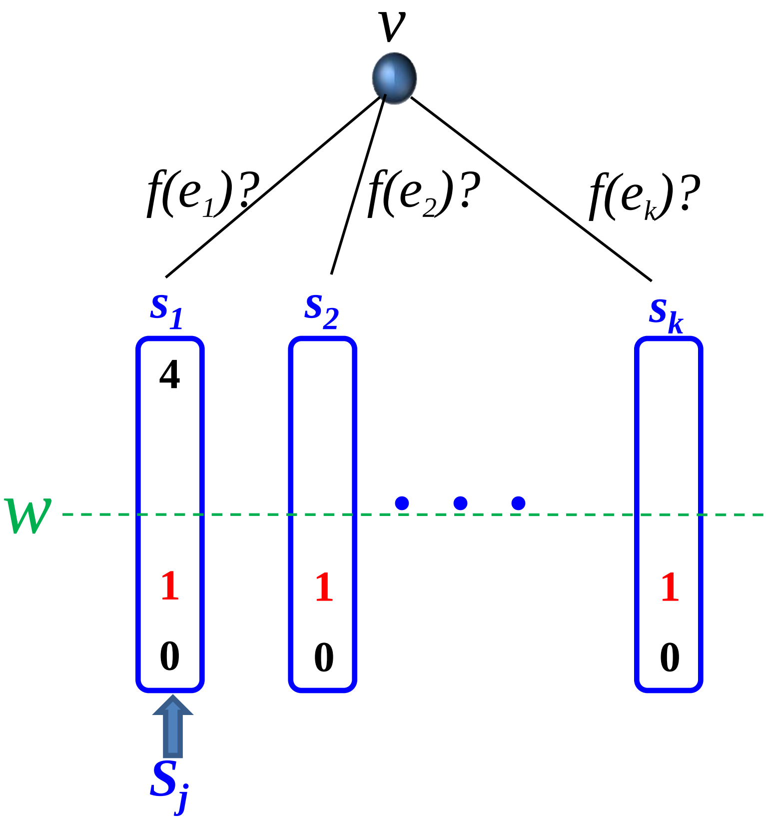
- set $u = max\{s_i\}$
- while not all edges assigned
  - if u appears once mark u as *not free*, move to next largest u
  - otherwise:
    - w = smallest free value > u
    - S_j = any maximal sequence w.r.t w
    - mark w as *not free*



		u	*w*				
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 = smallest free value > u
    - S_j = any maximal sequence w.r.t w
    - mark w as *not free*
    - set **current** $f(e_j) = w$

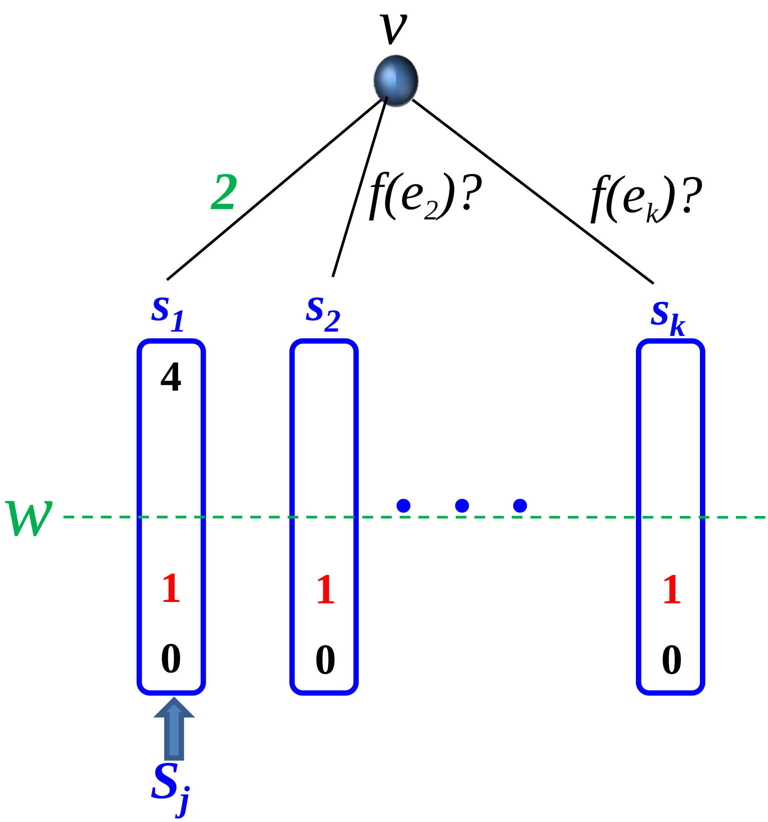


		u	w				
free values	**0**	**1**		**3**		**5**	**6**

Algorithm Outline

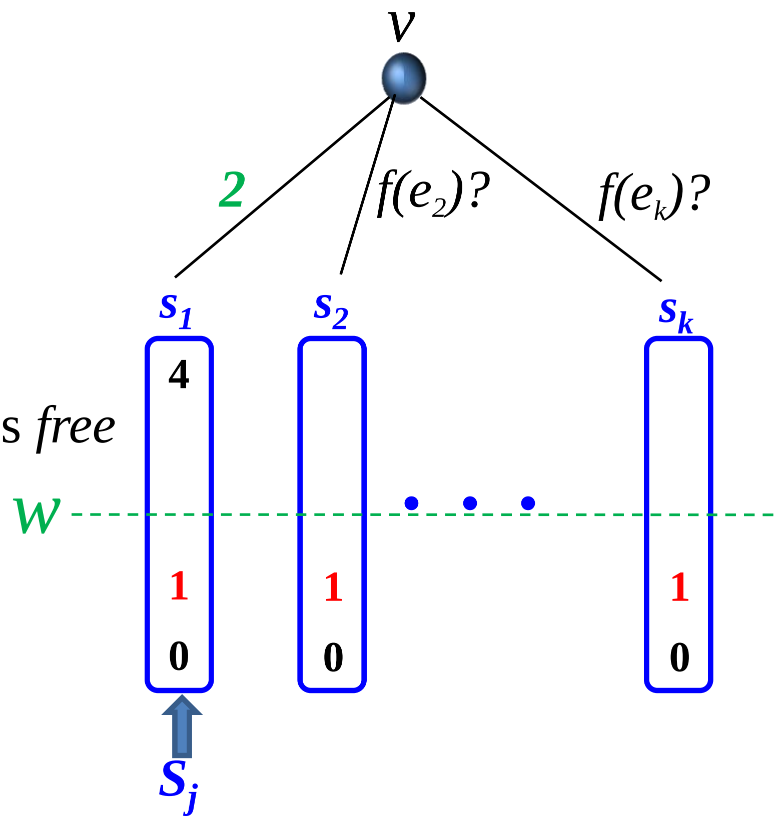
- set $u = max\{s_i\}$
- while not all edges assigned
  - if u appears once mark u as *not free*, move to next largest u
  - otherwise:
    - w = smallest free value > u
    - S_j = any maximal sequence w.r.t w
    - mark w as *not free*
    - set **current** $f(e_j) = w$

		u	w				
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 = 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*

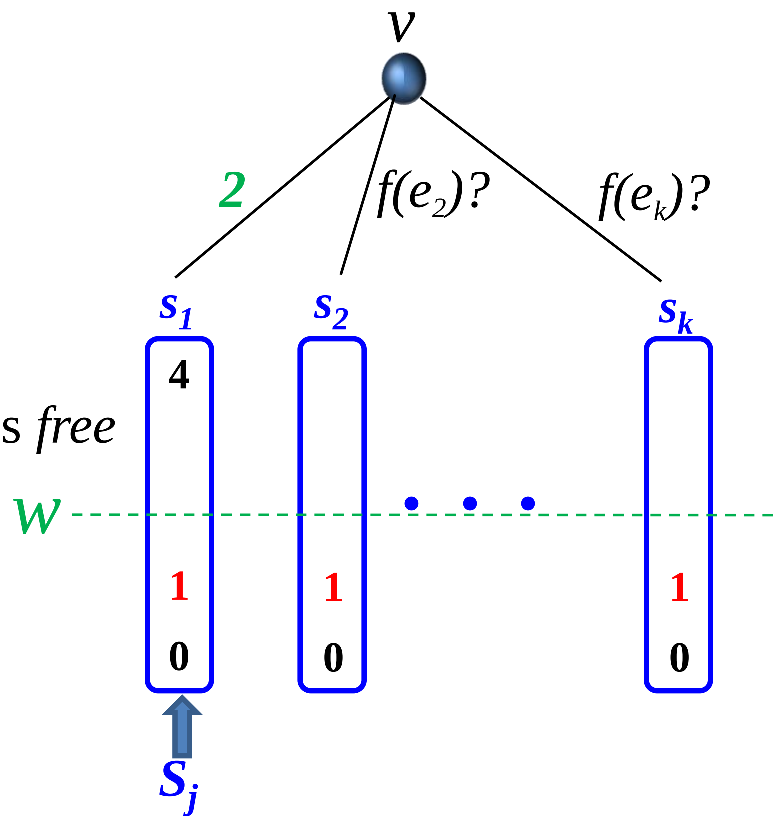


		u	w				
free values	**0**	**1**		**3**		**5**	**6**

Algorithm Outline

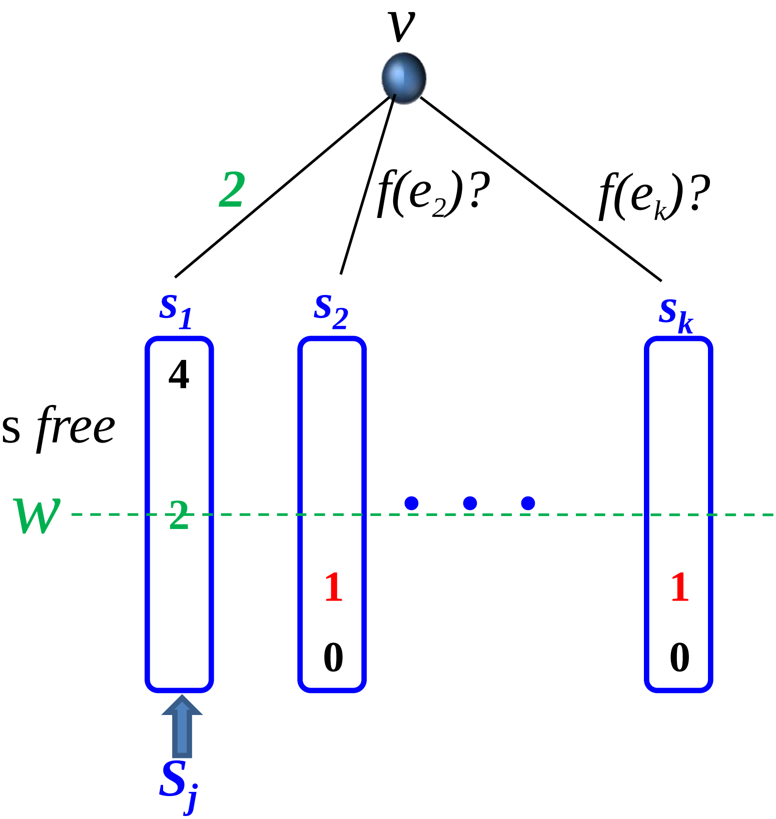
- set $u = max\{s_i\}$
- while not all edges assigned
  - if u appears once mark u as *not free*, move to next largest u
  - otherwise:
    - w = smallest free value > u
    - S_j = any maximal sequence w.r.t w
    - mark w as *not free*
    - set current $f(e_j) = w$
    - mark all S_j values between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

		u	w				
free values	0	1		3		5	6



Algorithm Outline

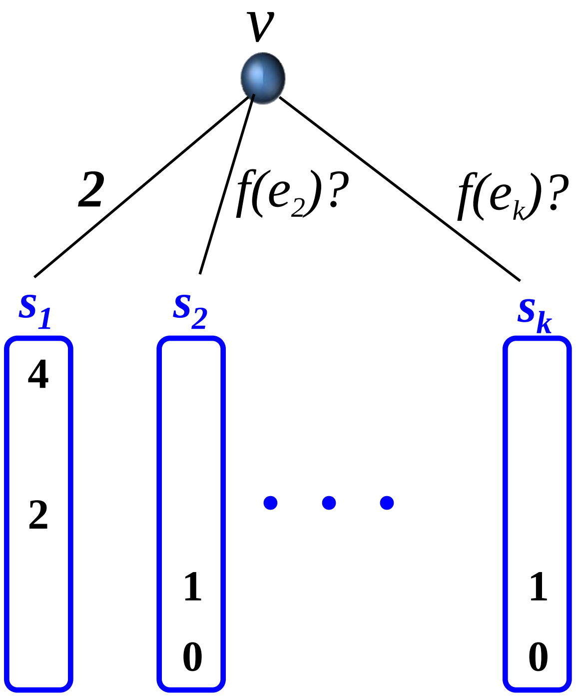
- set $u = max\{s_i\}$
- while not all edges assigned
  - if u appears once mark u as *not free*, move to next largest u
  - otherwise:
    - w = smallest free value > u
    - S_j = any maximal sequence w.r.t w
    - mark w as *not free*
    - set current $f(e_j) = w$
    - mark all S_j values between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



		u	w				
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 = 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

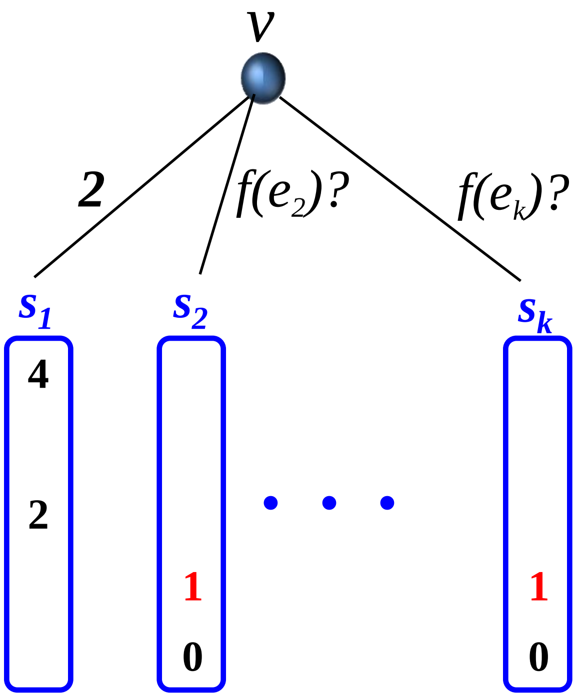


free values

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 = 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

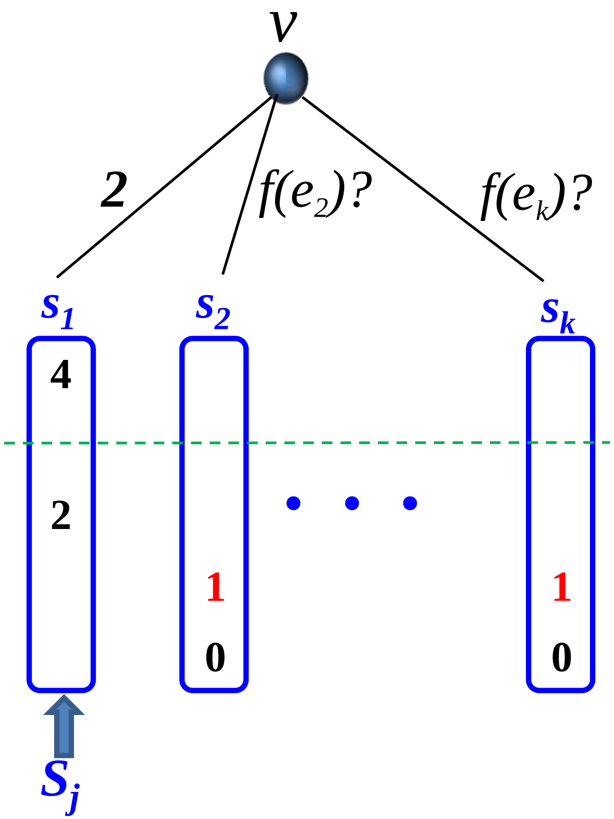


u

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 = 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

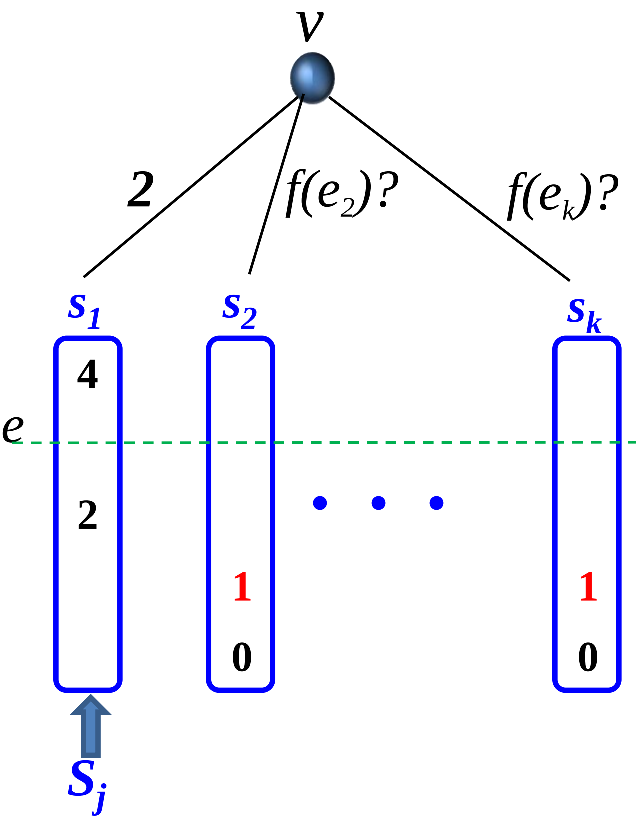


		u		w			
free values	**0**	**1**		**3**		**5**	**6**

Algorithm Outline

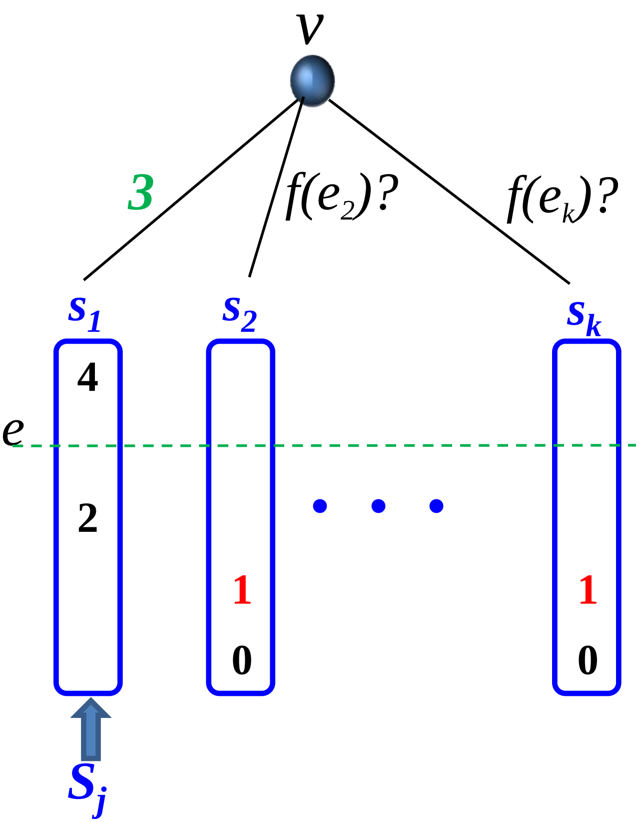
- set $u = max\{s_i\}$
- while not all edges assigned
  - if u appears once mark u as *not free*, move to next largest u
  - otherwise:
    - w = smallest free value > u
    - S_j = any maximal sequence w.r.t w
    - mark w as *not free*
    - set current $f(e_j) = w$
    - mark all S_j values between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

		u		w			
free values	**0**	**1**				**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 = 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

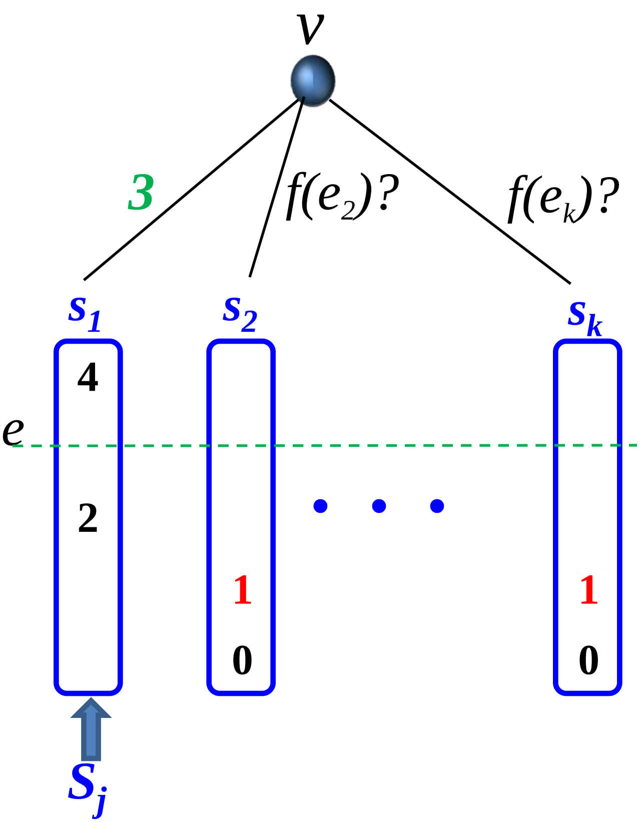


		u		w			
free values	**0**	**1**				**5**	**6**

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 = 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

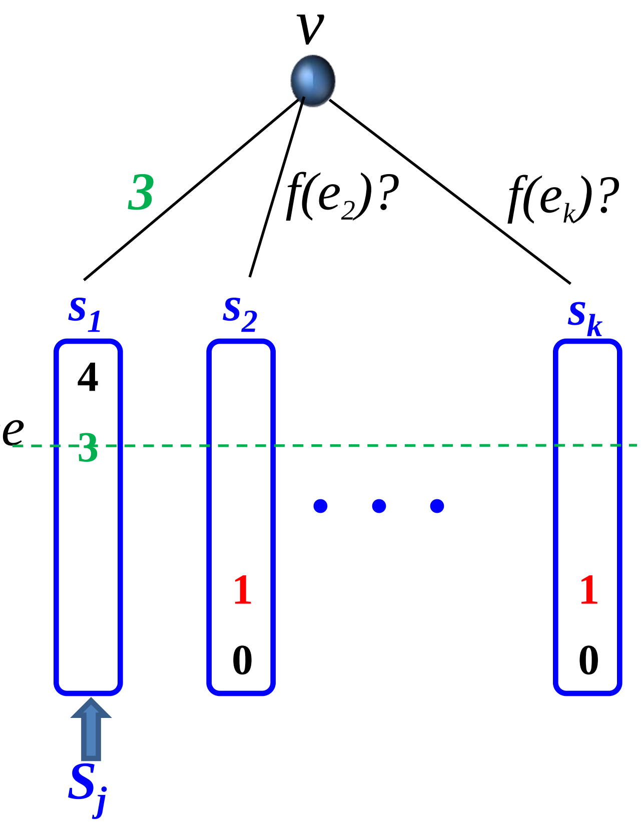
		u		w			
free values	0	1	2			5	6



Algorithm Outline

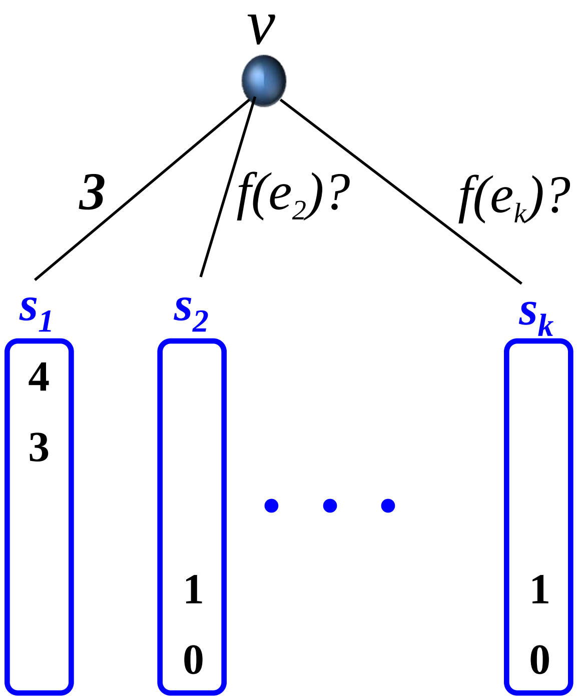
- set $u = max\{s_i\}$
- while not all edges assigned
  - if u appears once mark u as *not free*, move to next largest u
  - otherwise:
    - w = smallest free value > u
    - S_j = any maximal sequence w.r.t w
    - mark w as *not free*
    - set current $f(e_j) = w$
    - mark all S_j values between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

		u		w			
free values	**0**	**1**	**2**			**5**	**6**



Algorithm Outline

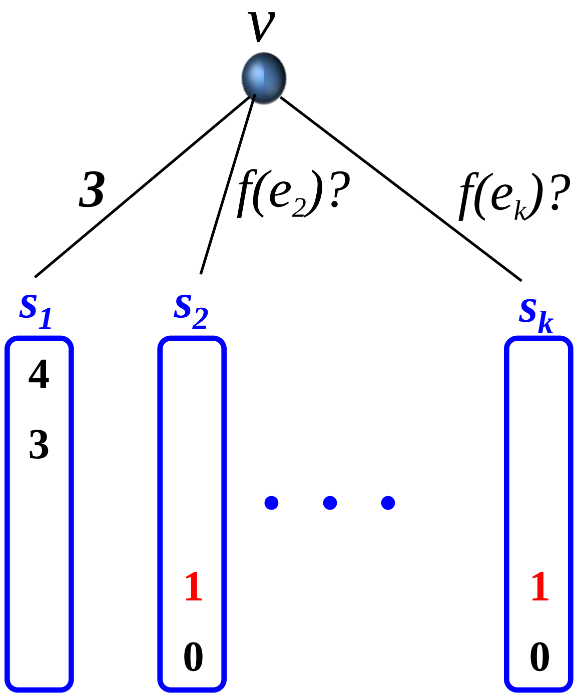
- set $u = max\{s_i\}$
- while not all edges assigned
  - if u appears once mark u as *not free*, move to next largest u
  - otherwise:
    - w = smallest free value > u
    - S_j = any maximal sequence w.r.t w
    - mark w as *not free*
    - set current $f(e_j) = w$
    - mark all S_j values between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



free values	**0**	**1**	**2**			**5**	**6**

Algorithm Outline

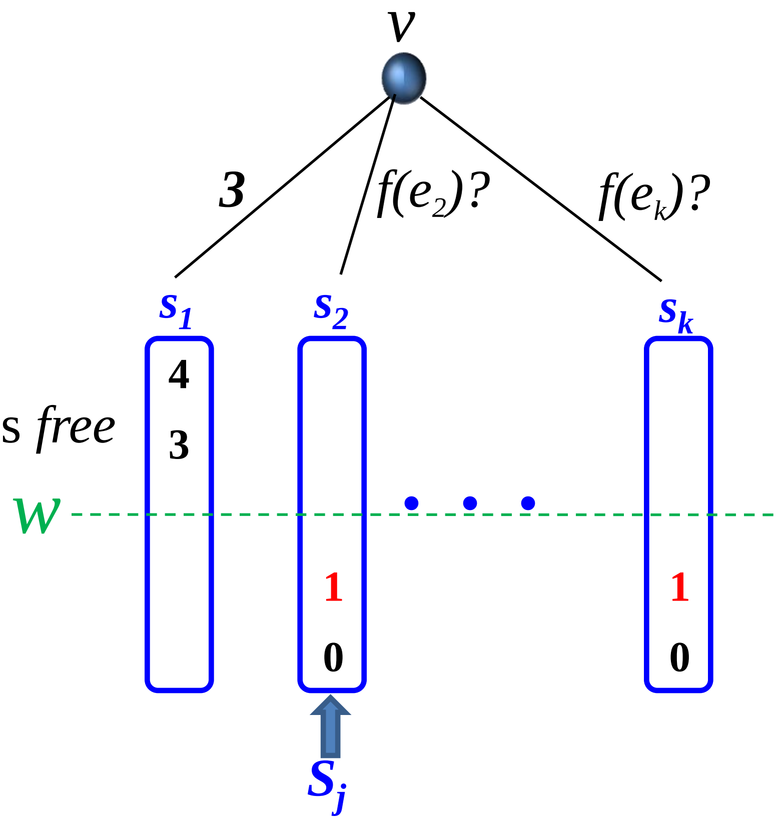
- set $u = max\{s_i\}$
- while not all edges assigned
  - if u appears once mark u as *not free*, move to next largest u
  - otherwise:
    - w = smallest free value > u
    - S_j = any maximal sequence w.r.t w
    - mark w as *not free*
    - set current $f(e_j) = w$
    - mark all S_j values between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



		u					
free values	0	1	2			5	6

Algorithm Outline

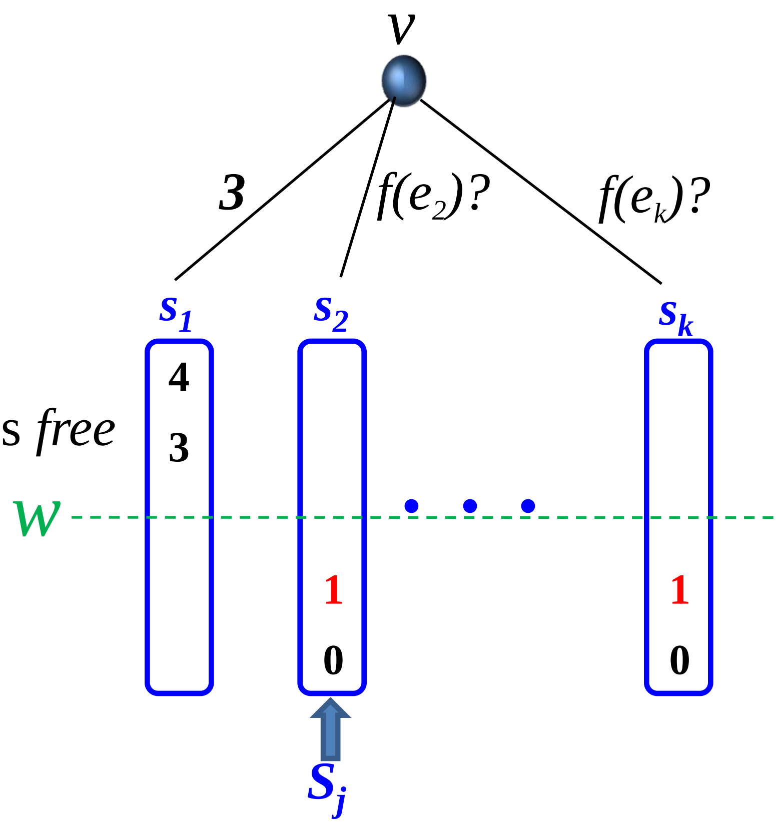
- set $u = max\{s_i\}$
- while not all edges assigned
  - if u appears once mark u as *not free*, move to next largest u
  - otherwise:
    - w = smallest free value > u
    - S_j = any maximal sequence w.r.t w
    - mark w as *not free*
    - set current $f(e_j) = w$
    - mark all S_j values between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



		u	w				
free values	**0**	**1**	**2**			**5**	**6**

Algorithm Outline

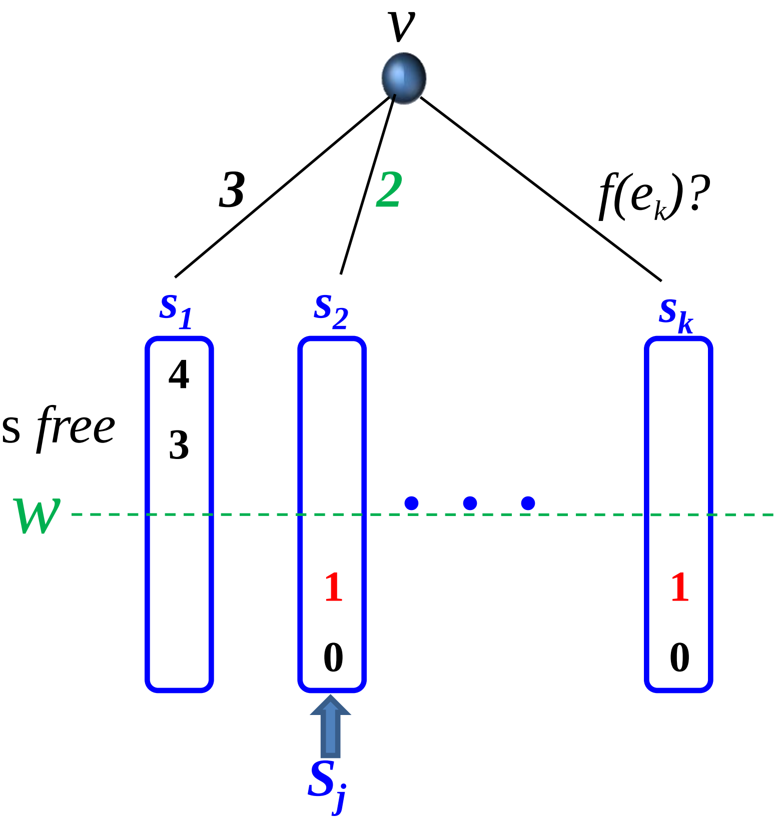
- set $u = max\{s_i\}$
- while not all edges assigned
  - if u appears once mark u as *not free*, move to next largest u
  - otherwise:
    - w = smallest free value > u
    - S_j = any maximal sequence w.r.t w
    - mark w as *not free*
    - set current $f(e_j) = w$
    - mark all S_j values between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



		u	w				
free values	**0**	**1**				**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 = 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

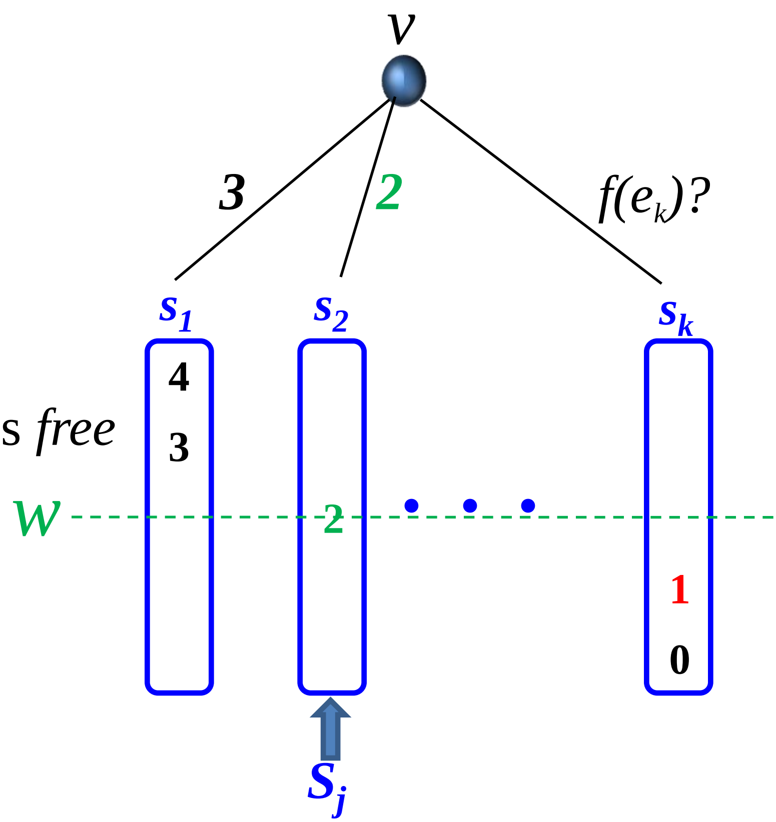


		u	w				
free values	**0**	**1**				**5**	**6**

Algorithm Outline

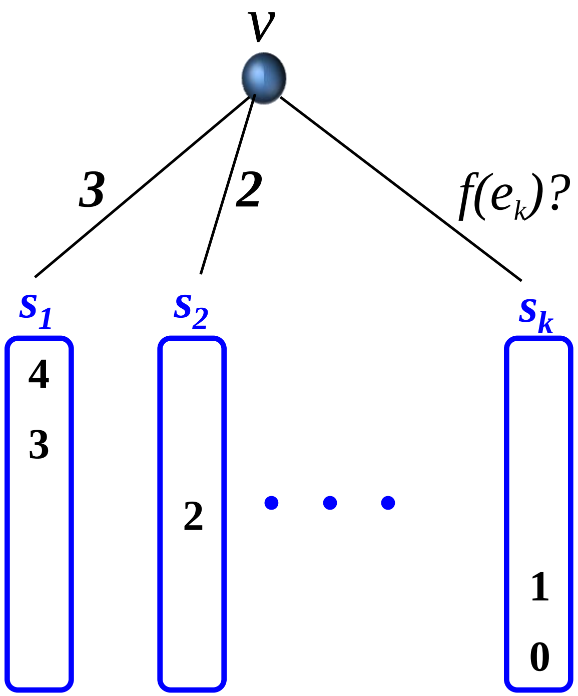
- set $u = max\{s_i\}$
- while not all edges assigned
  - if u appears once mark u as *not free*, move to next largest u
  - otherwise:
    - w = smallest free value > u
    - S_j = any maximal sequence w.r.t w
    - mark w as *not free*
    - set current $f(e_j) = w$
    - mark all S_j values between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

		u	w				
free values	**0**	**1**				**5**	**6**



Algorithm Outline

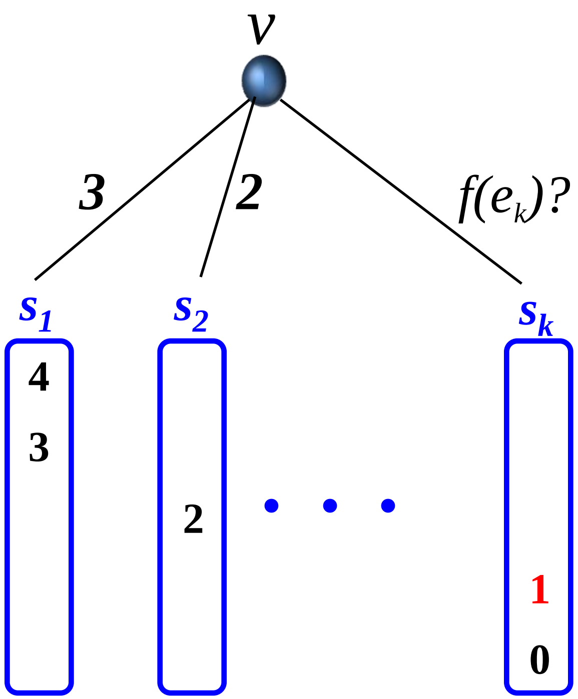
- set $u = max\{s_i\}$
- while not all edges assigned
  - if u appears once mark u as *not free*, move to next largest u
  - otherwise:
    - w = smallest free value > u
    - S_j = any maximal sequence w.r.t w
    - mark w as *not free*
    - set current $f(e_j) = w$
    - mark all S_j values between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



free values	**0**	**1**				**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 = 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



u

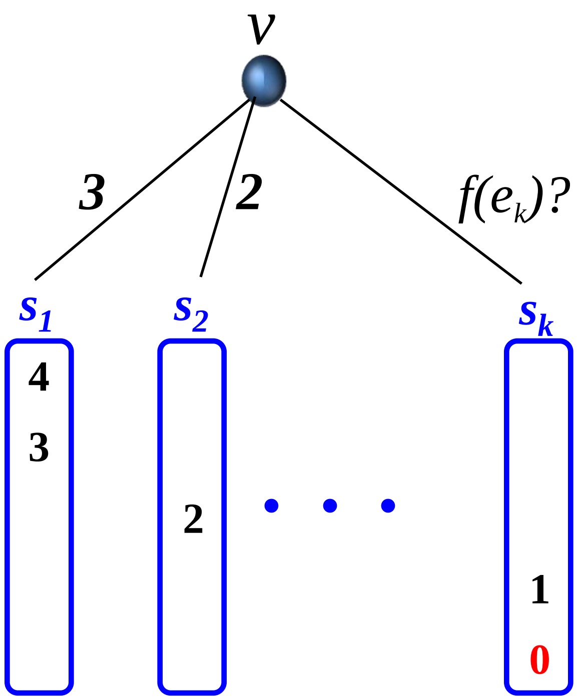
free values	**0**	**1**				**5**	**6**

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 = 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

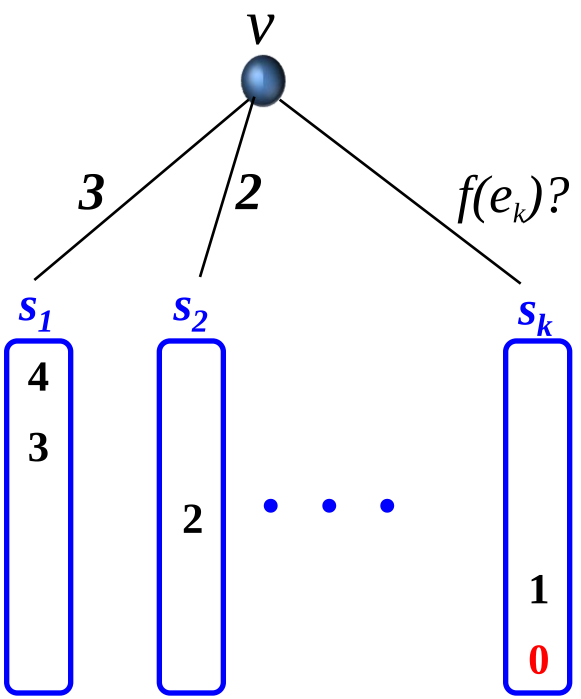
u

free values	**0**					**5**	**6**



Algorithm Outline

- set $u = max\{s_i\}$
- while not all edges assigned
  - if u appears $\underbrace{\text{once mark}}_{\text{and } u \neq 0}$ 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



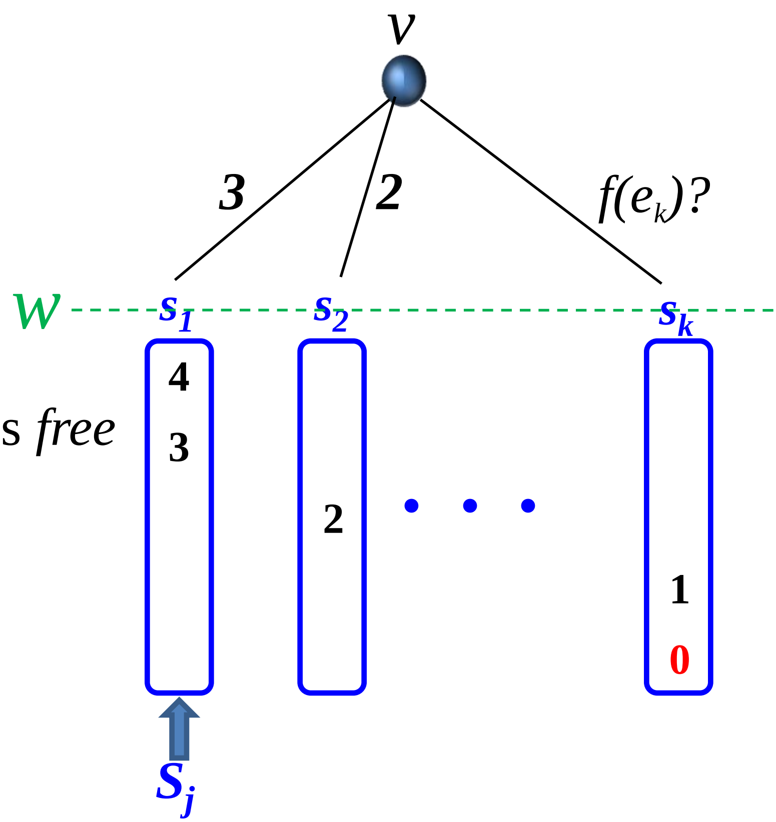
u

free values	**0**					**5**	**6**

Algorithm Outline

- set $u = max\{s_i\}$
- while not all edges assigned
  - if u appears once $\underbrace{\ }_{\text{and } u \neq 0}$ 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

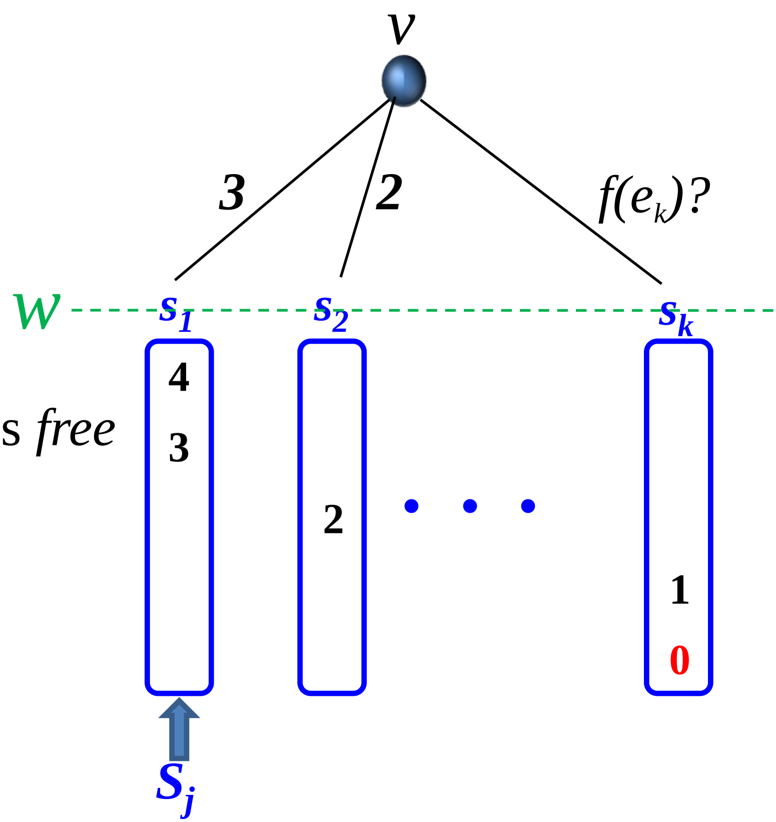
	u					w	
free values	**0**					**5**	**6**



Algorithm Outline

- set $u = max\{s_i\}$
- while not all edges assigned
  - if u appears $\underbrace{\text{once mark}}_{\text{and } u \neq 0}$ 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

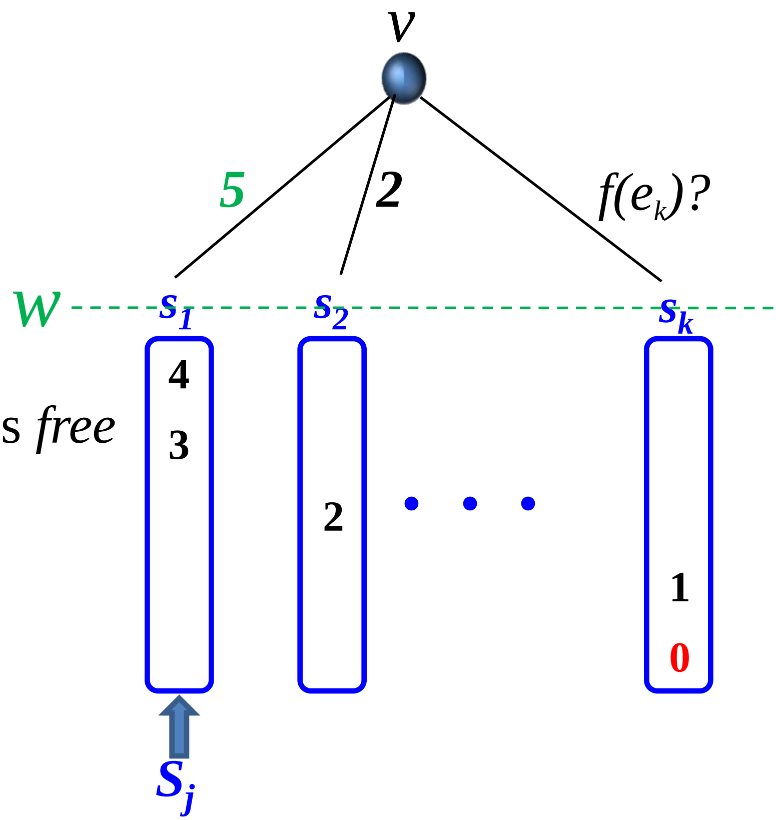
	u					w	
free values	**0**						**6**



Algorithm Outline

- set $u = max\{s_i\}$
- while not all edges assigned
  - if u appears once $\underbrace{\ }_{\text{and } u \neq 0}$ 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

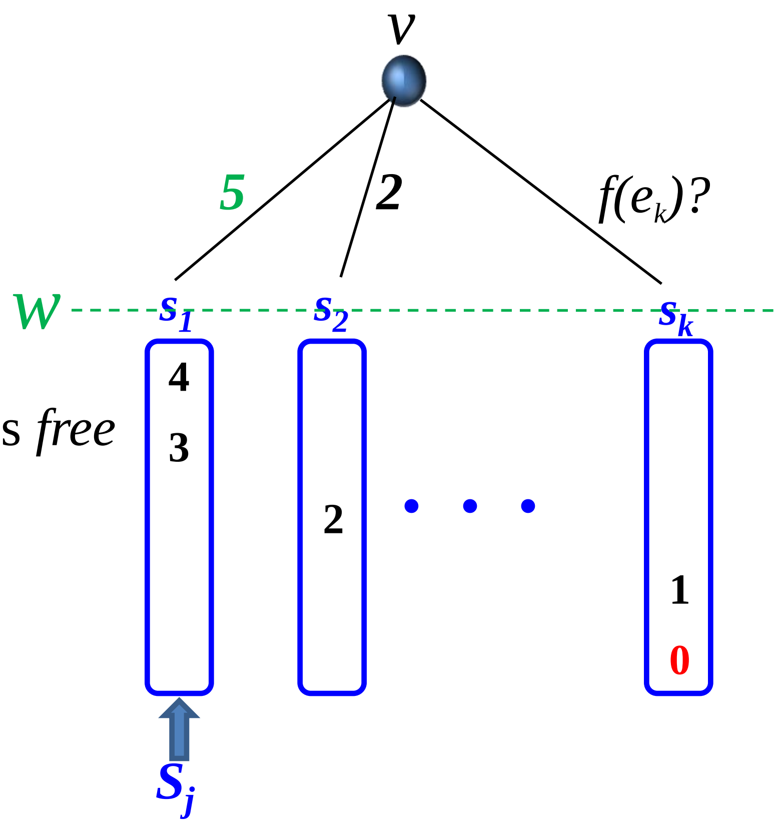
	u					w	
free values	**0**						**6**



Algorithm Outline

- set $u = max\{s_i\}$
- while not all edges assigned
  - if u appears once $\underbrace{\ }_{\text{and } u \neq 0}$ 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

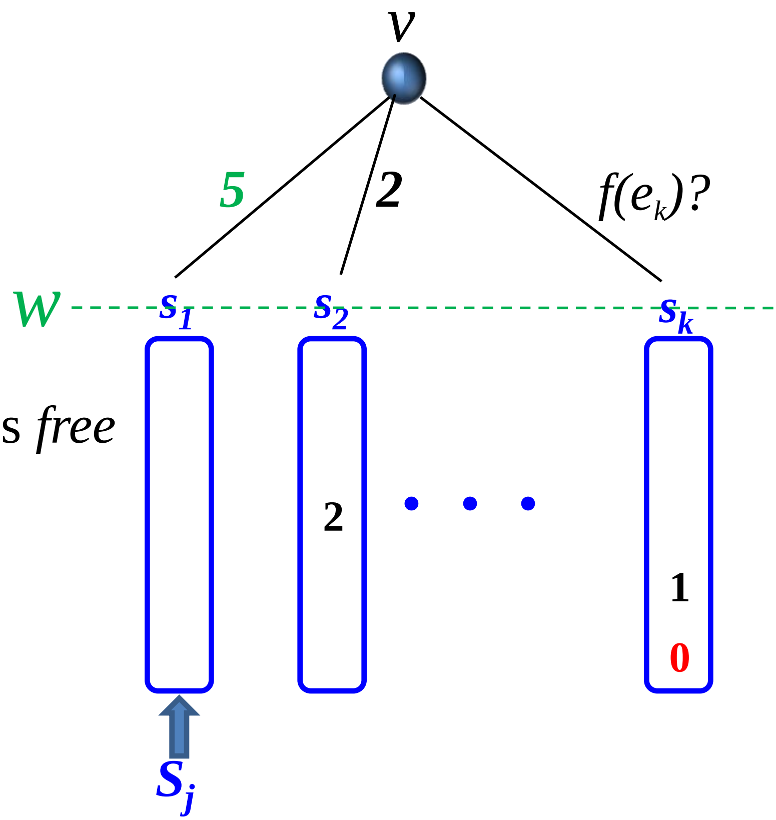
	u					w	
free values	**0**			**3**	**4**		**6**



Algorithm Outline

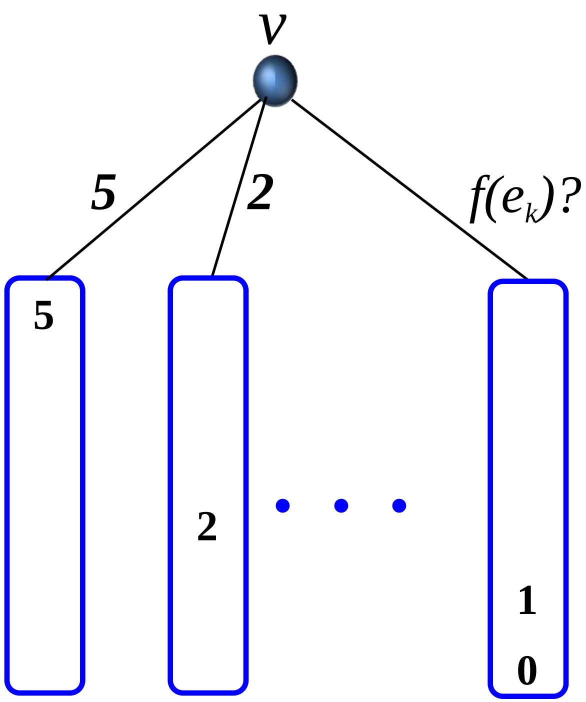
- set $u = max\{s_i\}$
- while not all edges assigned
  - if u appears $\underbrace{\text{once}}_{\text{and } u \neq 0}$ mark u as *not free*, move to next largest u
  - otherwise:
    - w = smallest free value > u
    - S_j = any maximal sequence w.r.t w
    - mark w as *not free*
    - set current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

	u					w	
free values	**0**			**3**	**4**		**6**



Algorithm Outline

- set $u = max\{s_i\}$
- while not all edges assigned
  - if u appears once $\underbrace{}_{\text{and } u \neq 0}$ 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

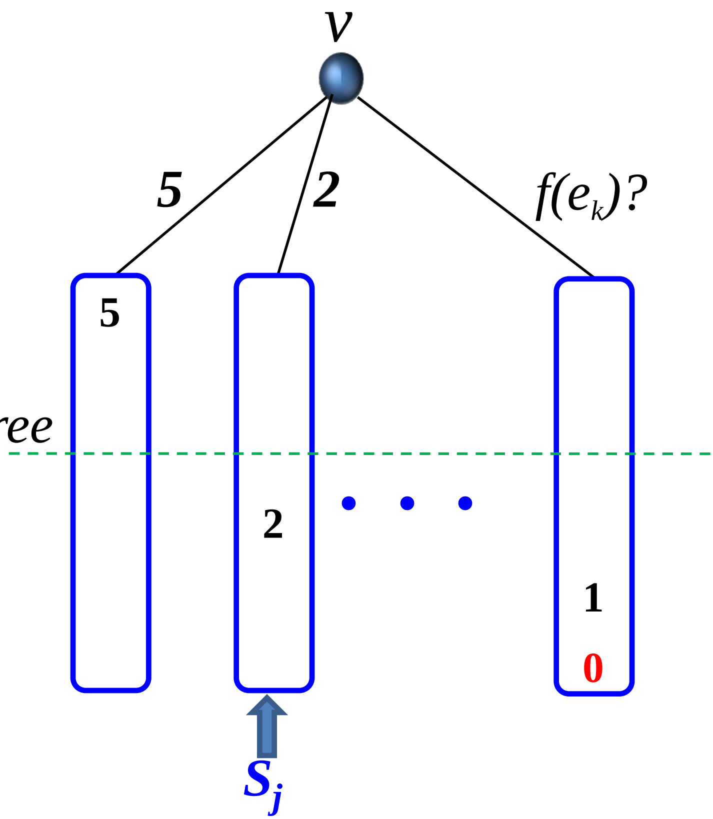


free values	**0**			**3**	**4**		**6**

Algorithm Outline

- set $u = max\{s_i\}$
- while not all edges assigned
  - if u appears $\underbrace{\text{once}}_{\text{and } u \neq 0}$ 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

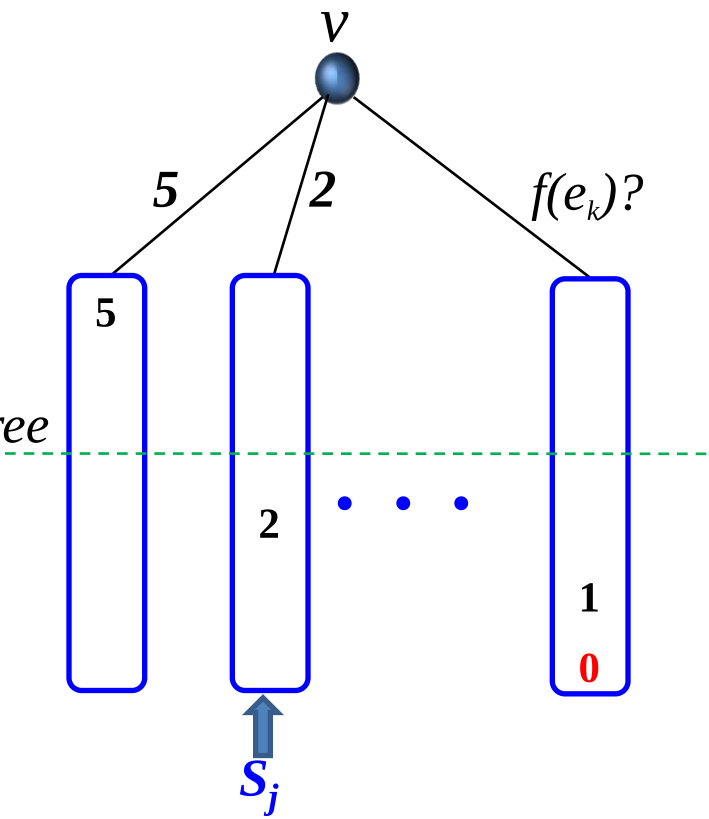
	u			w			
free values	**0**			**3**	**4**		**6**



Algorithm Outline

- set $u = max\{s_i\}$
- while not all edges assigned
  - if u appears $\underbrace{\text{once}}_{\text{and } u \neq 0}$ 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

	u			w			
free values	**0**				**4**		**6**

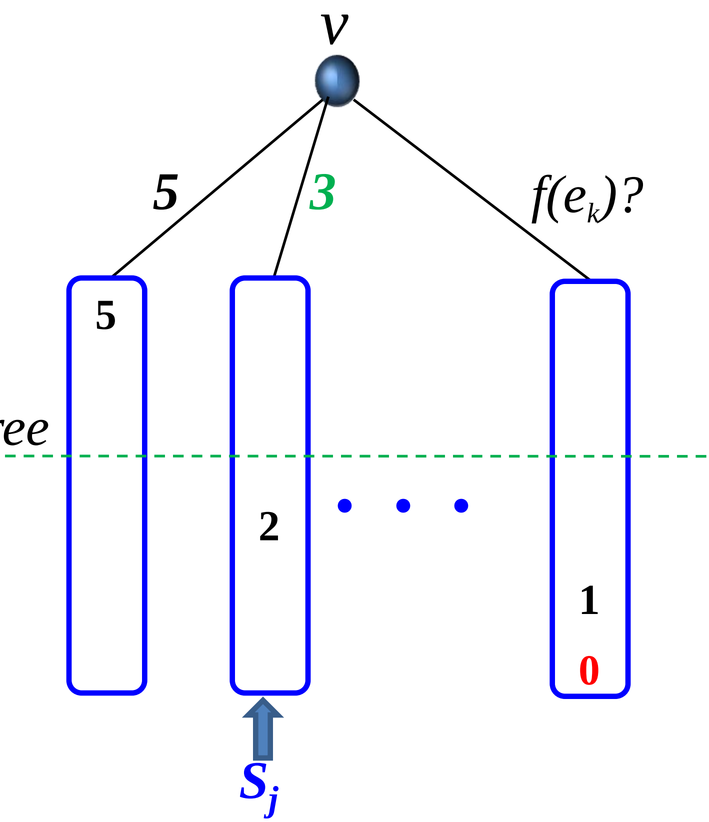


Algorithm Outline

- 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

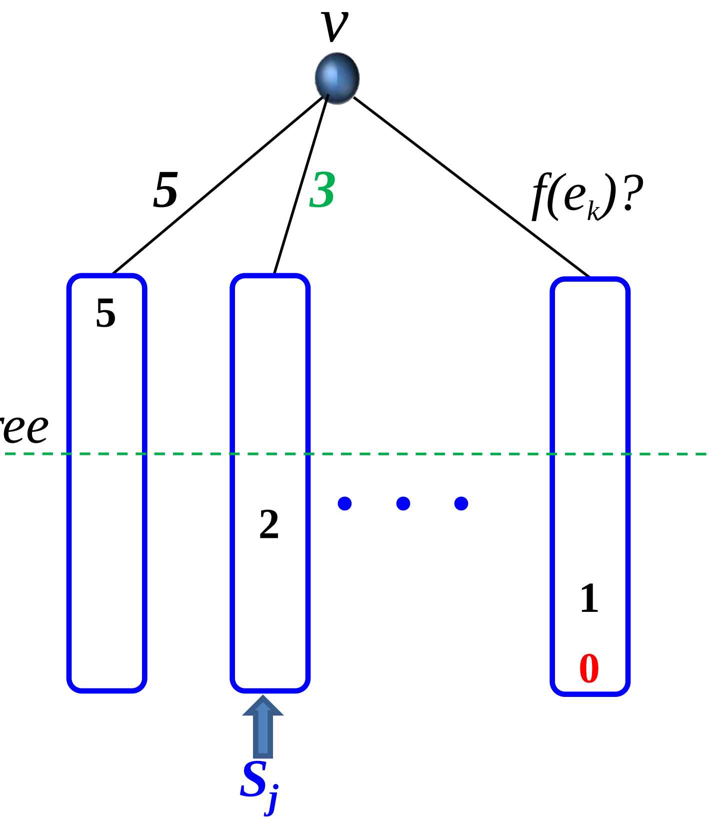
	u			w			
free values	**0**				**4**		**6**



Algorithm Outline

- set $u = max\{s_i\}$
- while not all edges assigned
  - if u appears $\underbrace{\text{once}}_{\text{and } u \neq 0}$ 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

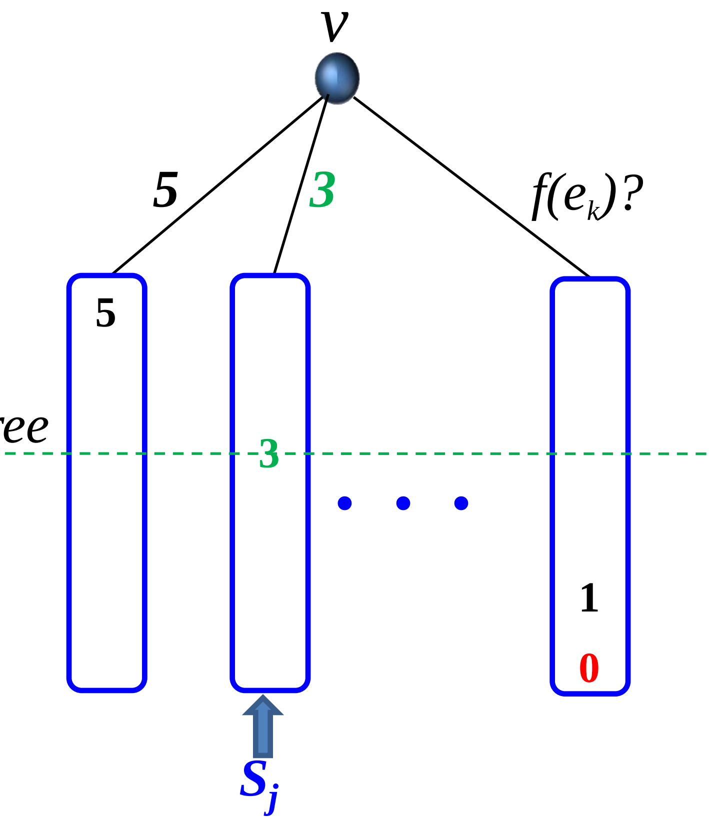
	u			w			
free values	**0**		**2**		**4**		**6**



Algorithm Outline

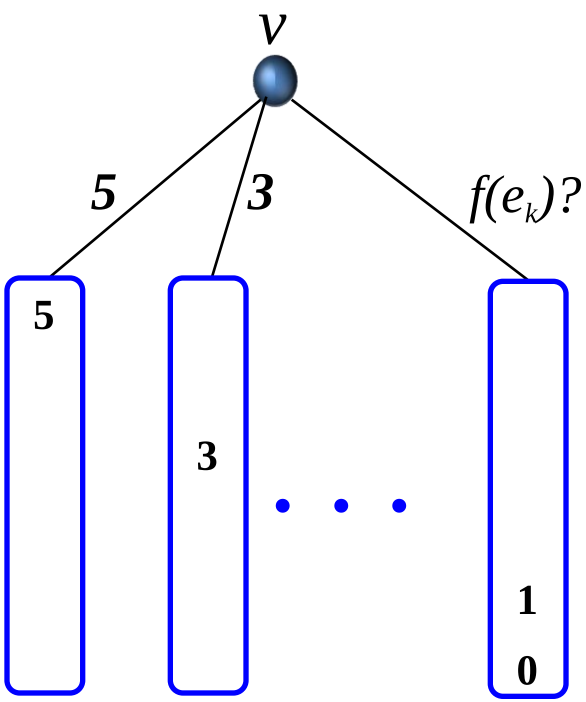
- set $u = max\{s_i\}$
- while not all edges assigned
  - if u appears once mark u as *not free*, move to next largest u
    and $u \neq 0$
  - otherwise:
    - w = smallest free value > u
    - S_j = any maximal sequence w.r.t w
    - mark w as *not free*
    - set current $f(e_j) = w$
    - mark all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

	u			w			
free values	**0**		**2**		**4**		**6**



Algorithm Outline

- set $u = max\{s_i\}$
- while not all edges assigned
  - if u appears $\underbrace{\text{once}}_{\text{and } u \neq 0}$ 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



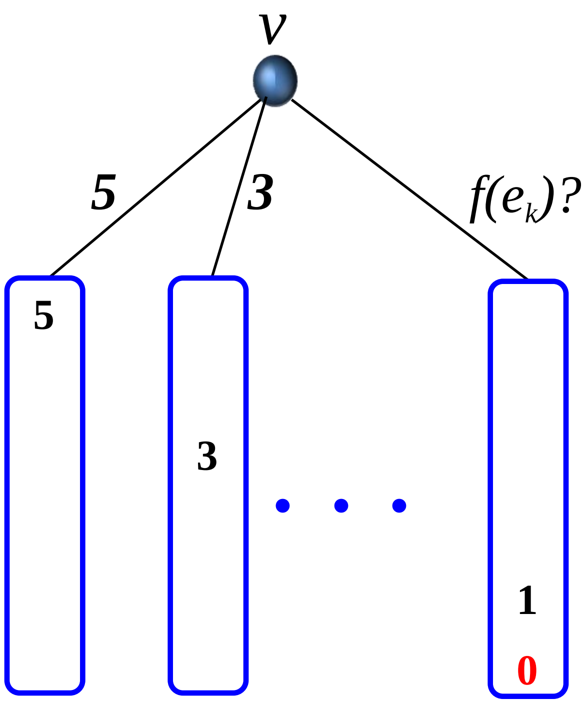
free values	**0**		**2**		**4**		**6**

Algorithm Outline

- set $u = max\{s_i\}$
- while not all edges assigned
  - if u appears $\underbrace{\text{once}}_{\text{and } u \neq 0}$ 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

u

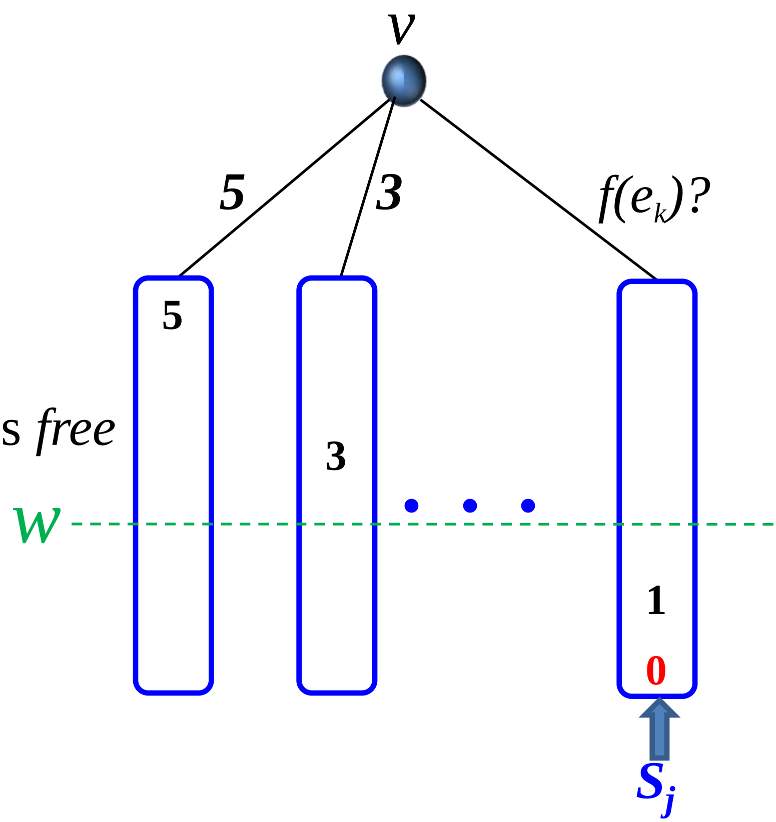
free values	**0**		**2**		**4**		**6**



Algorithm Outline

- set $u = max\{s_i\}$
- while not all edges assigned
  - if u appears once $\underbrace{\,}_{\text{and } u \neq 0}$ 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

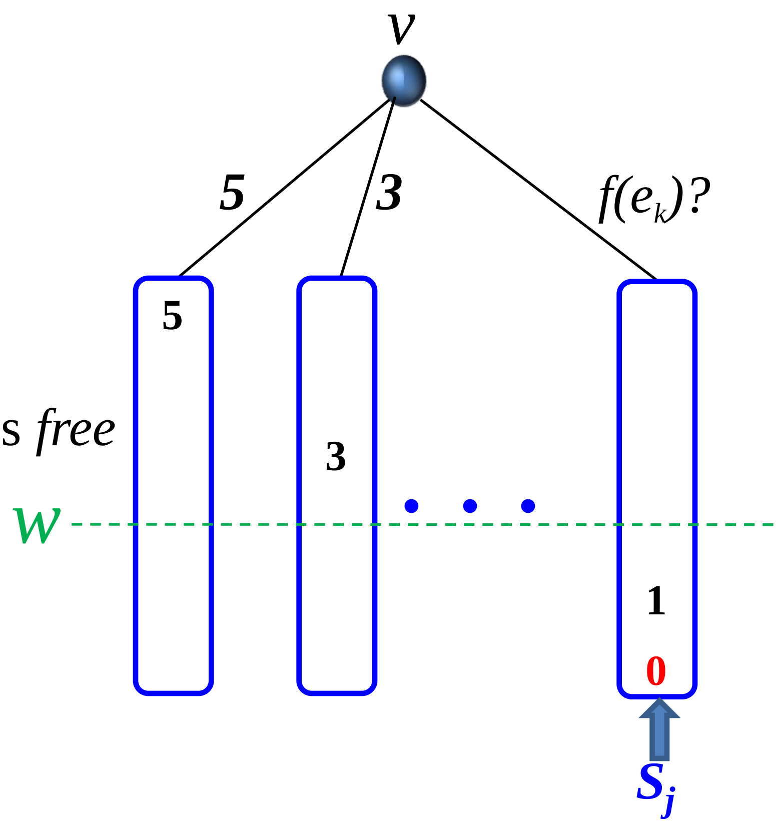
	u		w				
free values	**0**		**2**		**4**		**6**



Algorithm Outline

- set $u = max\{s_i\}$
- while not all edges assigned
  - if u appears $\underbrace{\text{once mark}}_{\text{and } u \neq 0}$ 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

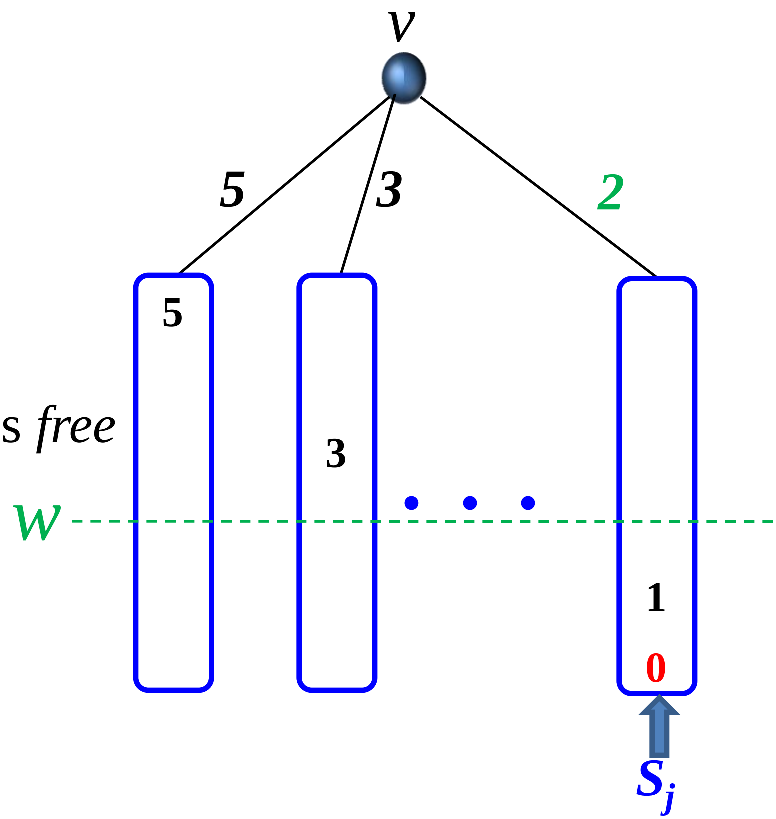
	u		w				
free values	**0**				**4**		**6**



Algorithm Outline

- set $u = max\{s_i\}$
- while not all edges assigned
  - if u appears $\underbrace{\text{once}}_{\text{and } u \neq 0}$ 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

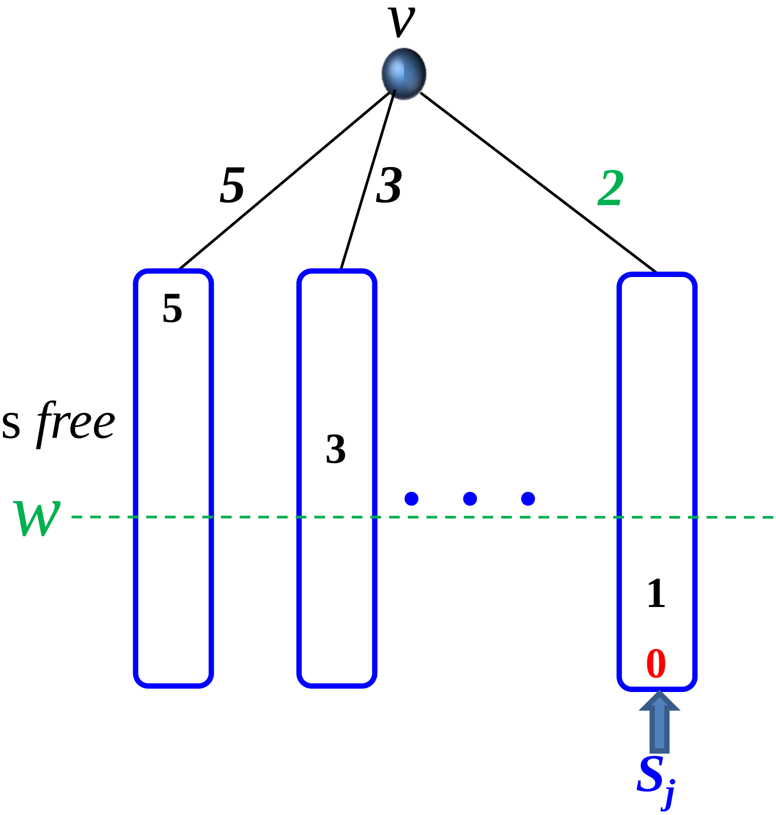
	u		w				
free values	**0**				**4**		**6**



Algorithm Outline

- set $u = max\{s_i\}$
- while not all edges assigned
  - if u appears once $\underbrace{}_{\text{and } u \neq 0}$ 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

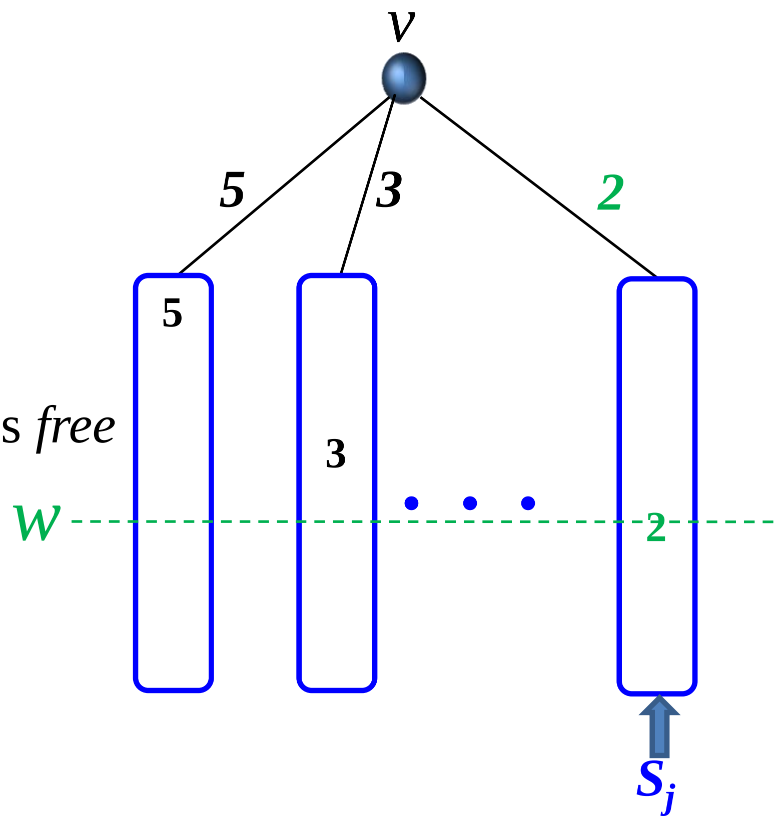
	u		w				
free values	**0**	**1**			**4**		**6**



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
    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

	u		w				
free values	**0**	**1**			**4**		**6**

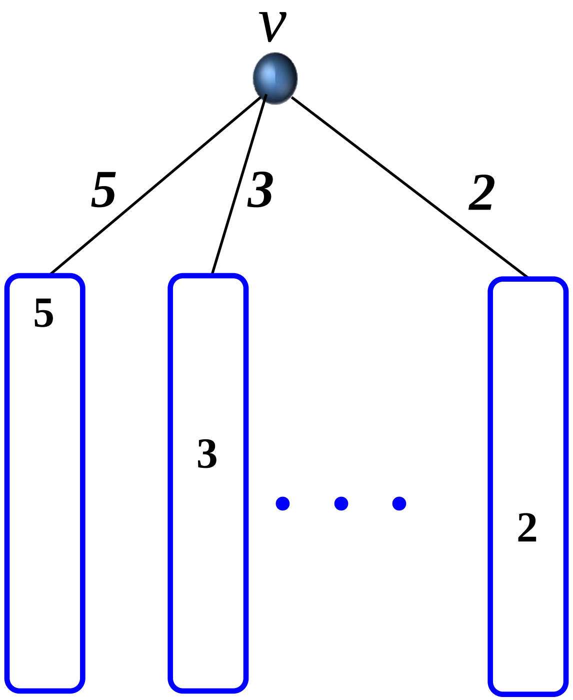


Algorithm Outline

That's it!

- 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



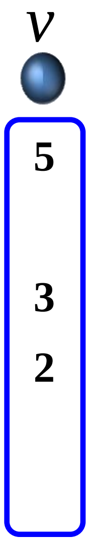
free values	**0**	**1**			**4**		**6**

Algorithm Outline

That's it!

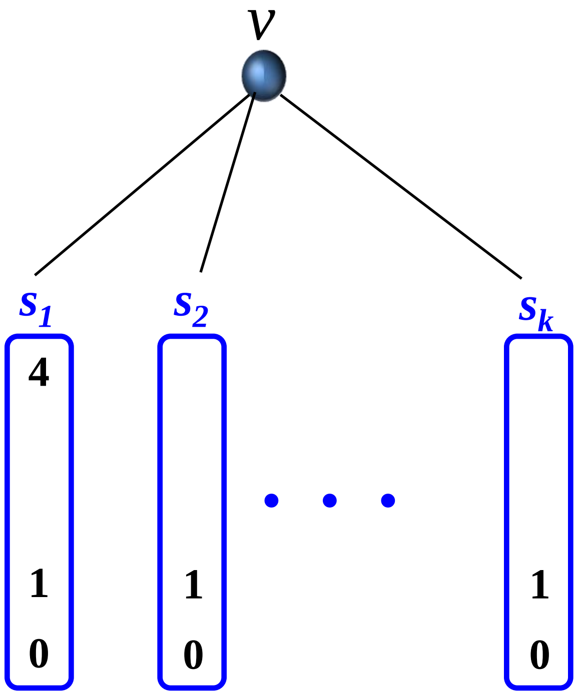
- set $u = max\{s_i\}$
- while not all edges assigned
  - if u appears once mark u as *not free*, move to next largest u

    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



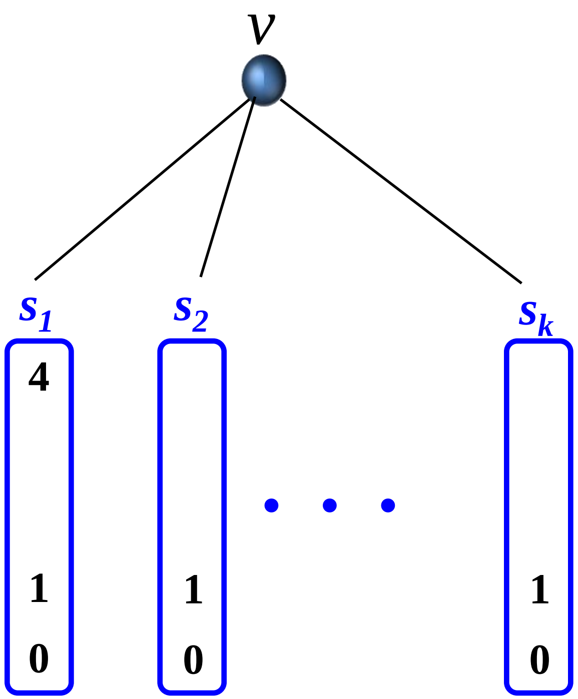
free values	**0**	**1**			**4**		**6**

Running Time



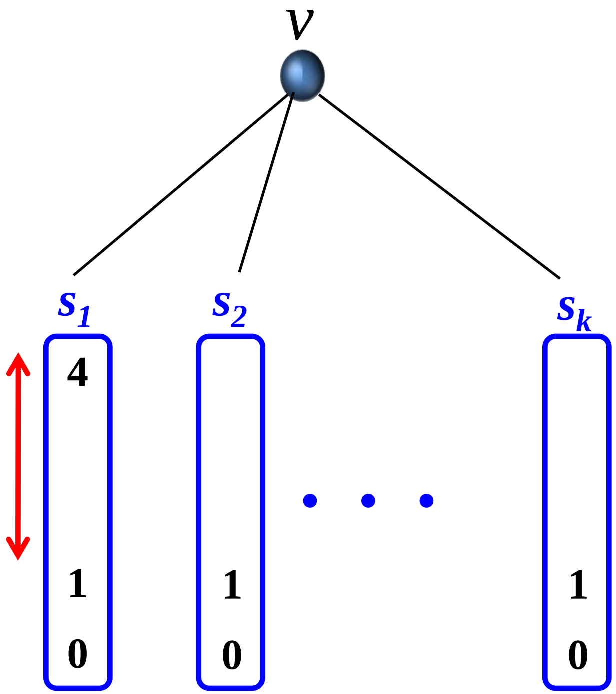
Running Time

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



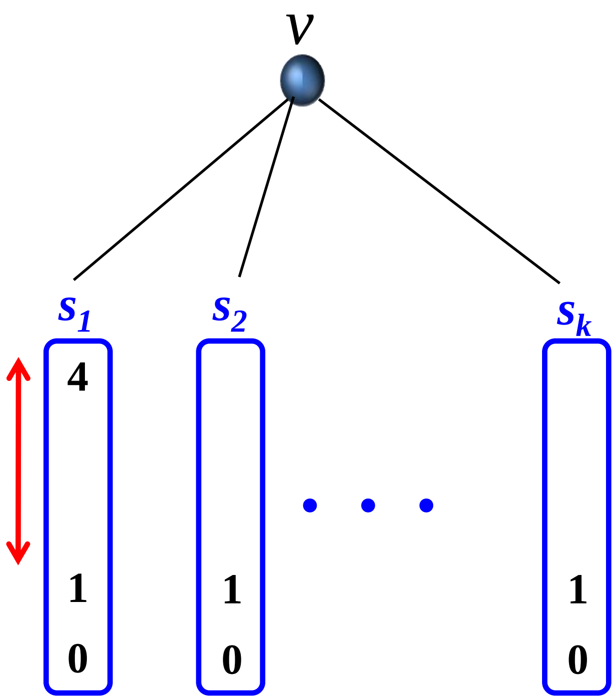
Running Time

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



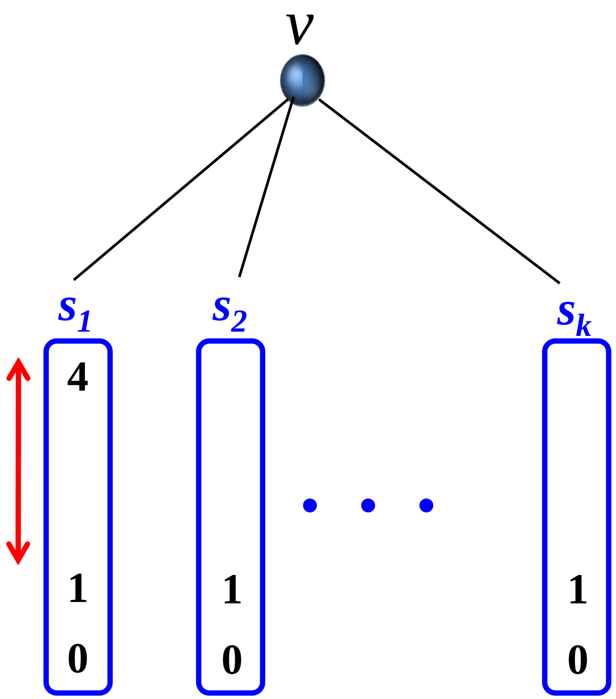
Running Time

- $|S_1| + |S_2| + \ldots + |S_k|$ 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



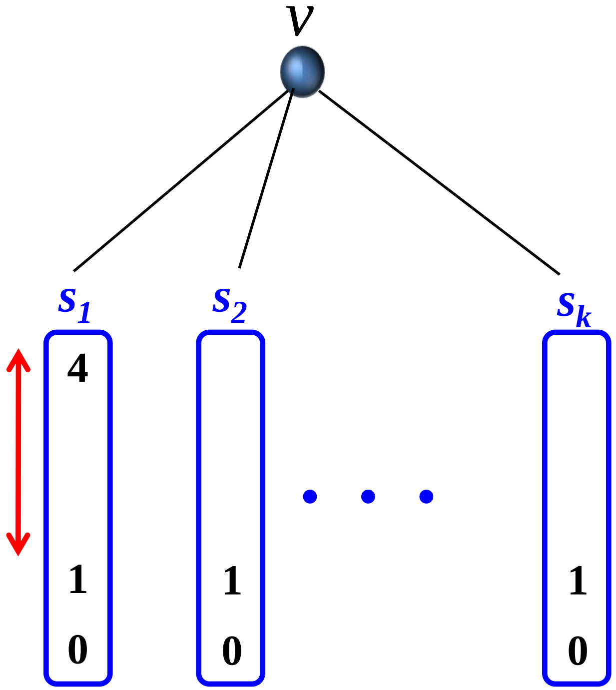
Running Time

- $|S_1| + |S_2| + \ldots + |S_k|$ 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
  - $q(v) = |S_2| + \ldots + |S_k|$



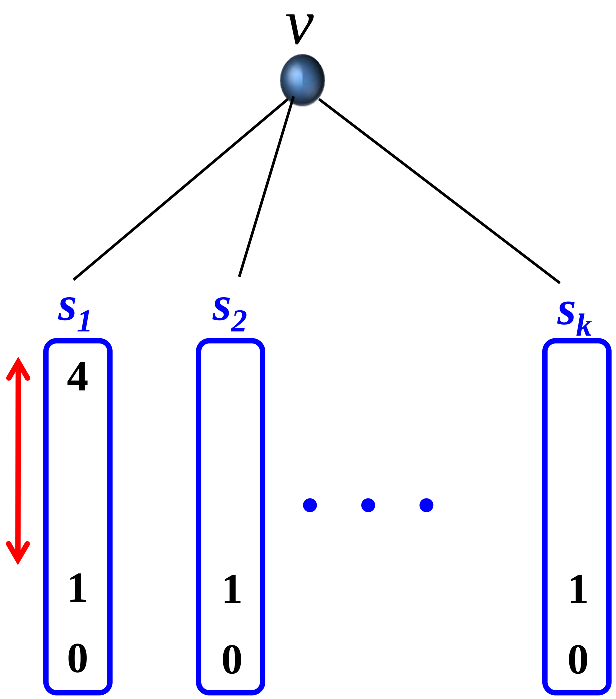
Running Time

- $|S_1| + |S_2| + \ldots + |S_k|$ 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
  - $q(v) = |S_2| + \ldots + |S_k|$
  - $t(v)$ = largest value that appears in S_v but not in S_1



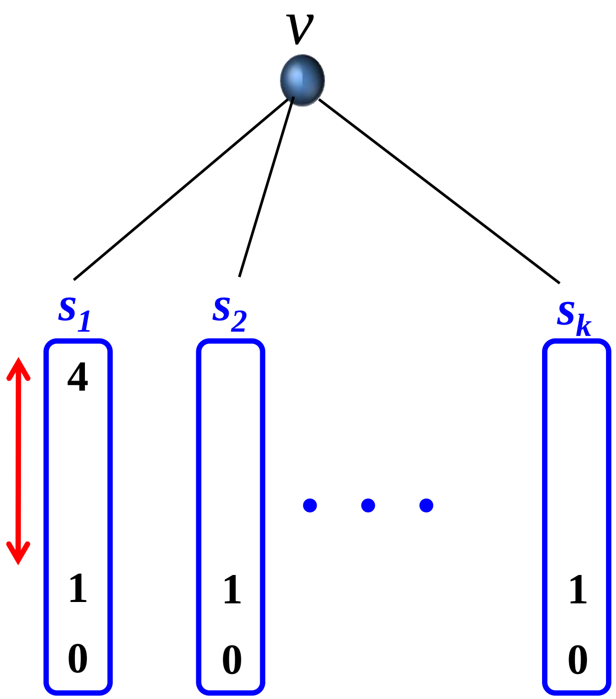
Running Time

- $|S_1| + |S_2| + \ldots + |S_k|$ 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
  - $q(v) = |S_2| + \ldots + |S_k|$
  - $t(v)$ = largest value that appears in S_v but not in S_1
- an extension can be computed in $O(\, k(v)+q(v)+t(v)\,)$



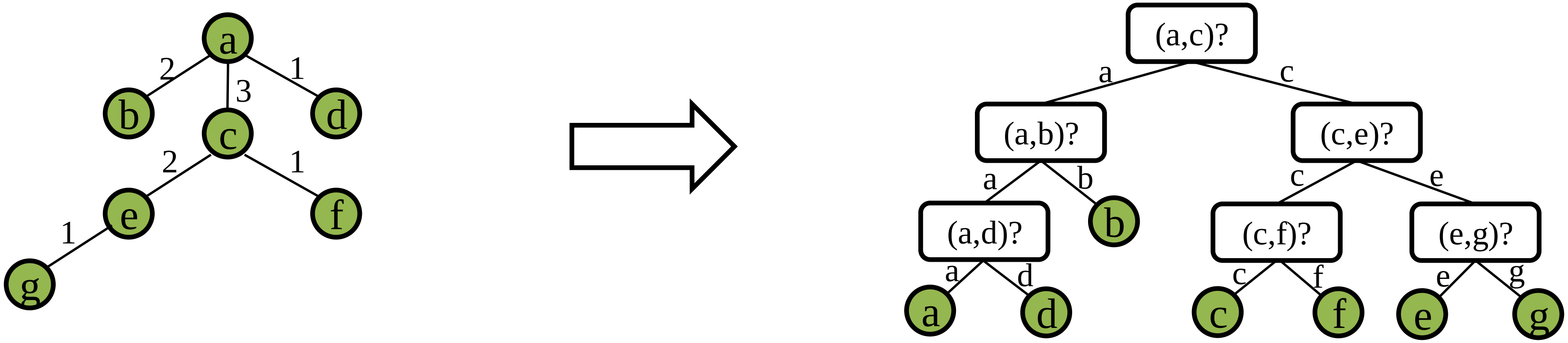
Running Time

- $|S_1| + |S_2| + \ldots + |S_k|$ 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
  - $q(v) = |S_2| + \ldots + |S_k|$
  - $t(v)$ = largest value that appears in S_v but not in S_1
- an extension can be computed in $O(\ k(v)+q(v)+t(v)\ )$
- $\sum_v k(v)+q(v)+t(v) = O(n)$

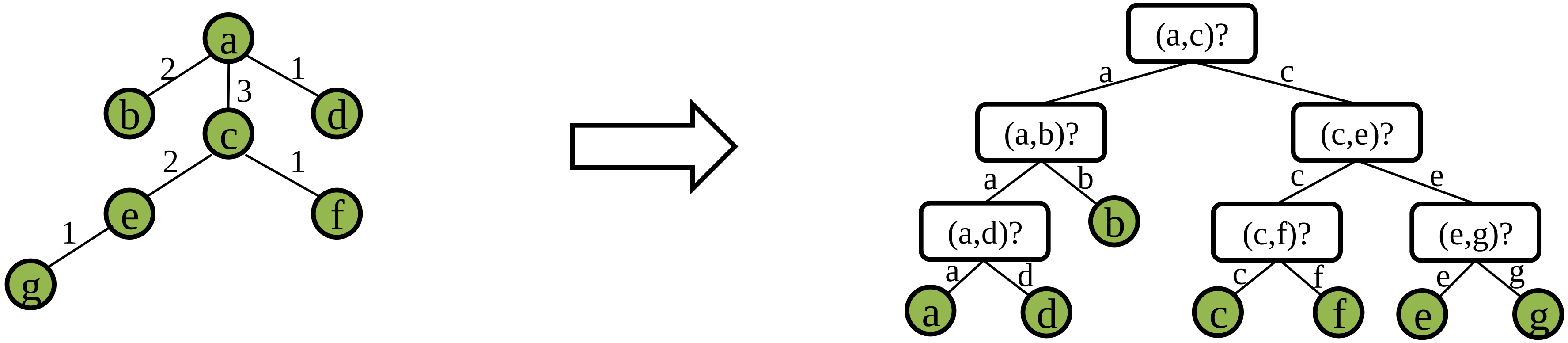


From Strategy Function to Decision Tree in $O(n)$ Time

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 *bottom(e)*
  - if *s* contains no values smaller than *f(e)*
    - set *bottom(e)* as the solution when the query on *e* returns *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 *bottom(e)*
    - for every $1 \le i < k$ set e_i as the solution when the query on e_{i+1} returns $top(e_{i+1})$
    - set $top(e_1)$ as the solution when the query on e_1 returns $top(e_1)$

Thank you !!