

# Operating on Data Efficiently with Common Algorithms



Rasmus Resen Amossen

<http://rasmus.resen.org>

---

# Algorithms

# Algorithms

Graph traversal

Brute force  
Greedy algorithms

Divide  
and  
conquer

Dynamic  
programming

Branch  
and bound

# Algorithms

Graph traversal

Brute force  
Greedy algorithms

Divide  
and  
conquer

Dynamic  
programming

Branch  
and  
bound

# Algorithms

Graph traversal

Divide  
and  
conquer

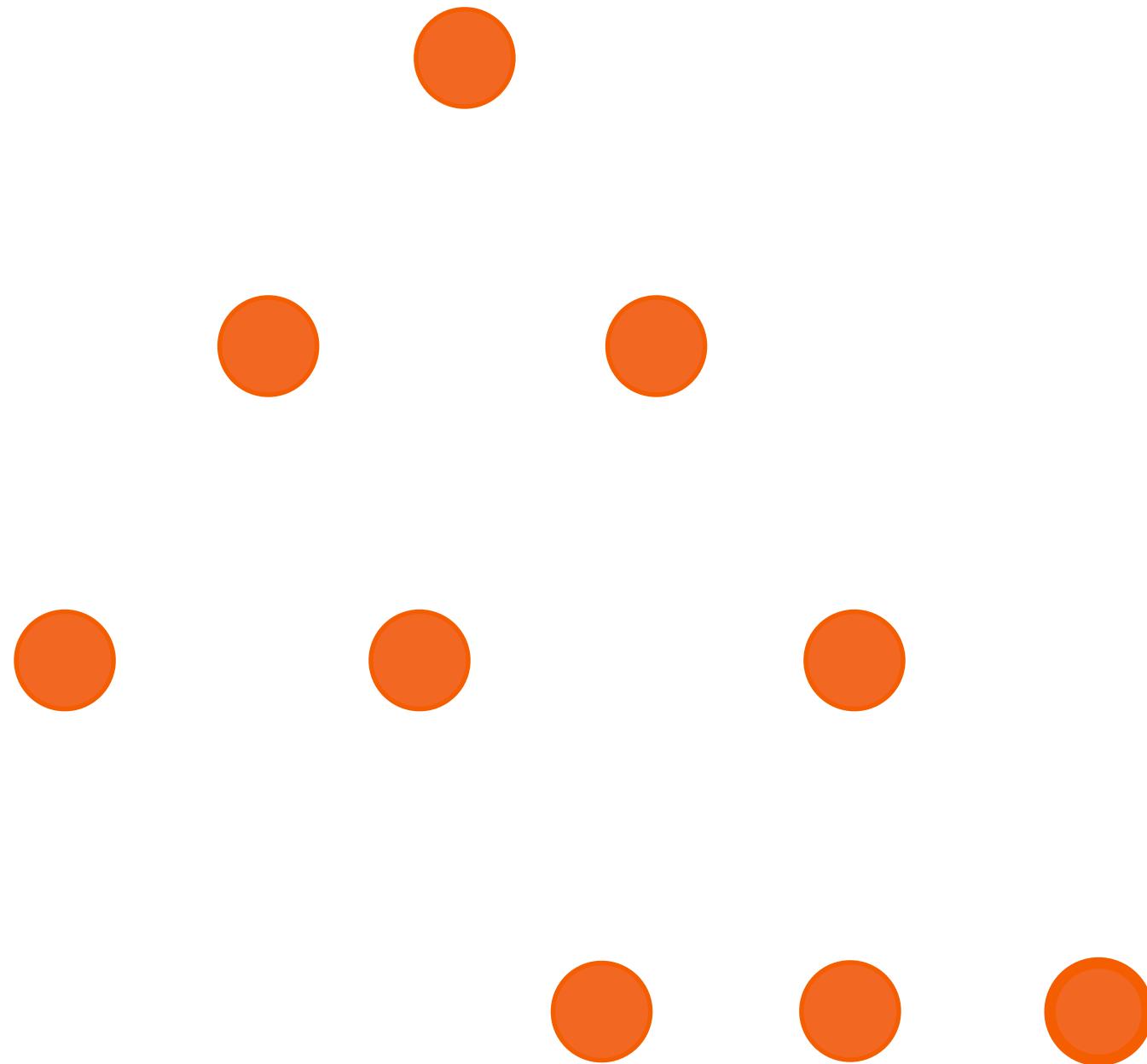
Dynamic  
programming

Brute force  
greedy algorithms

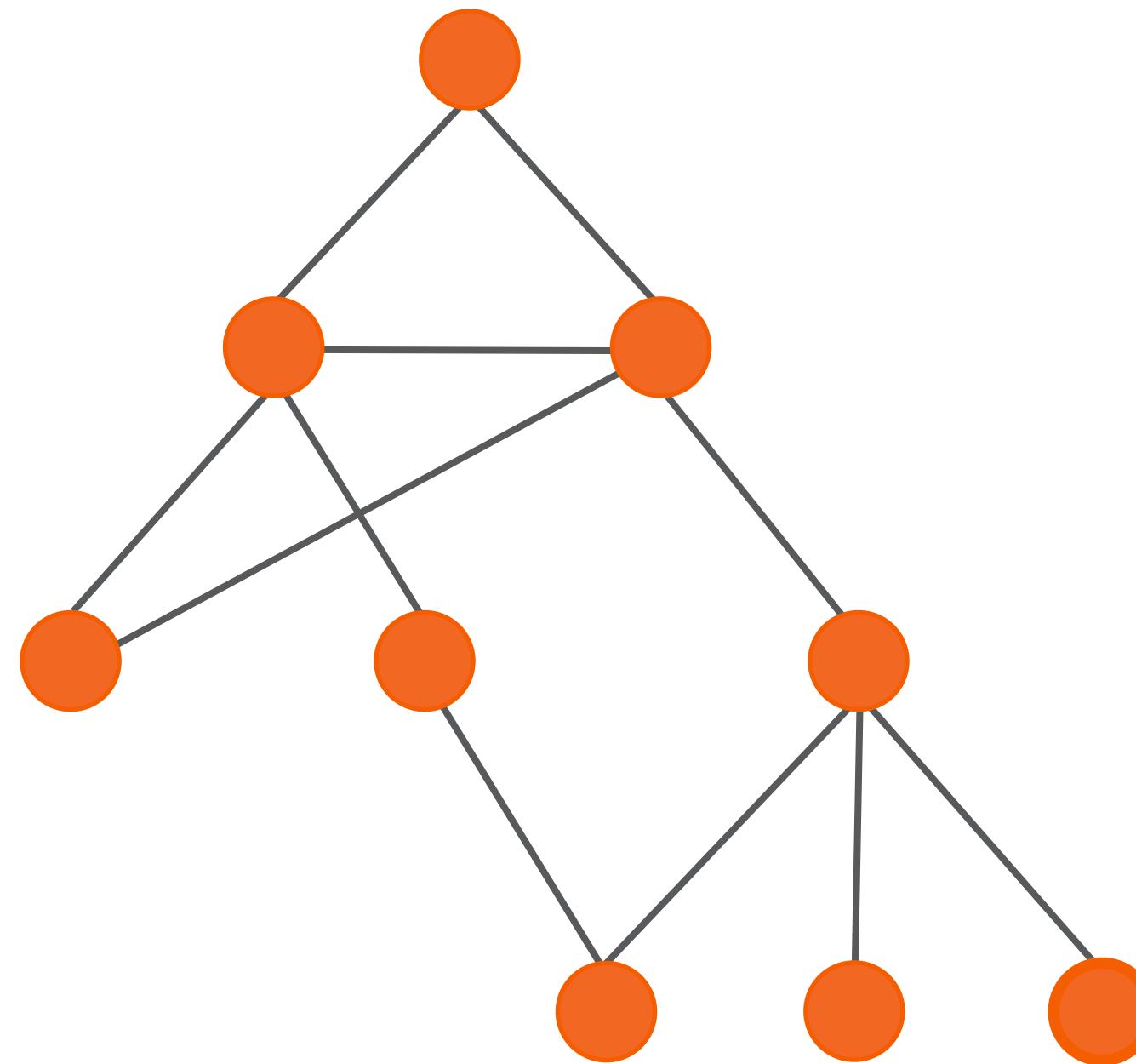
Branch  
and  
bound

# Graphs and Trees

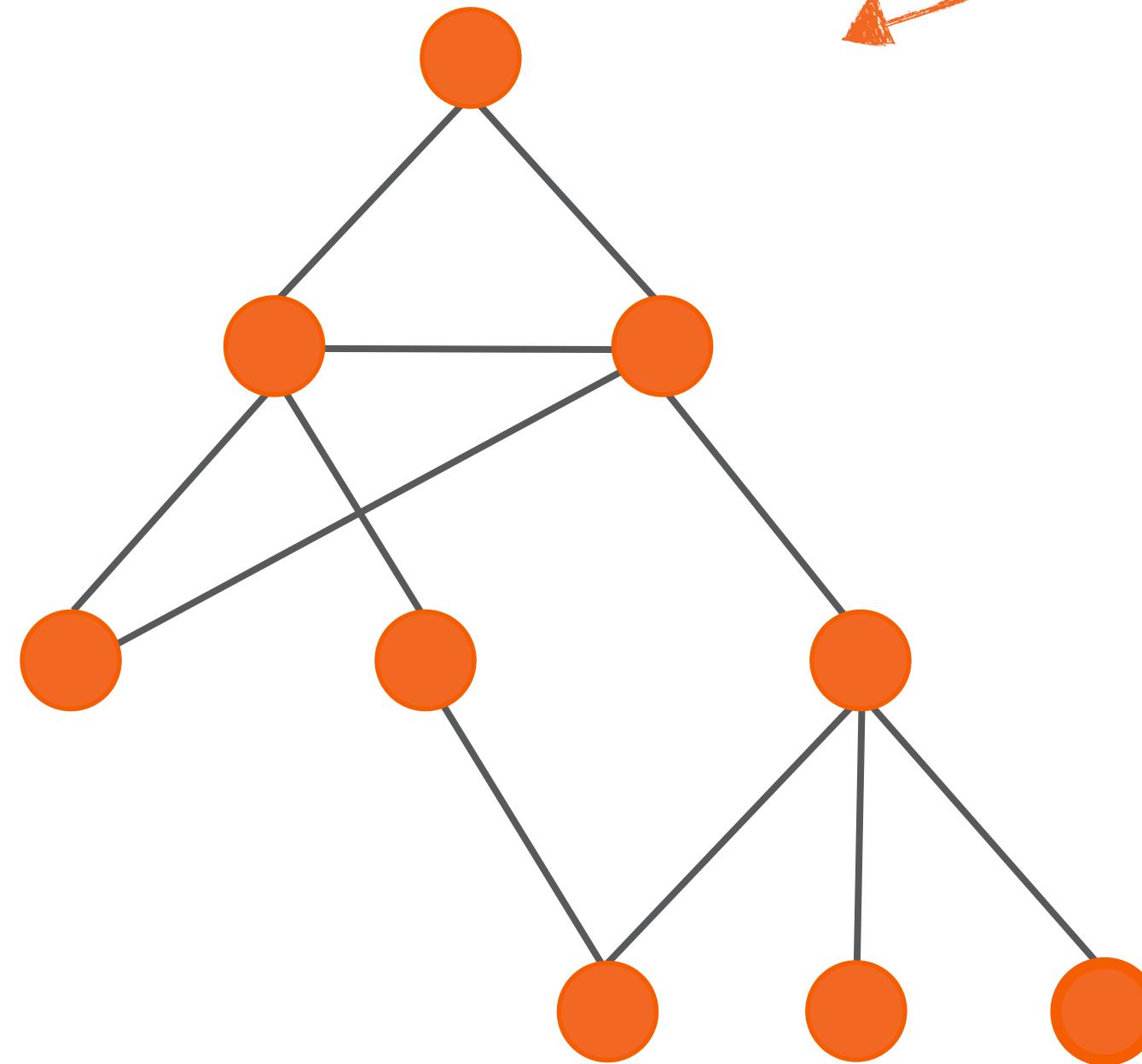
# Graphs and Trees



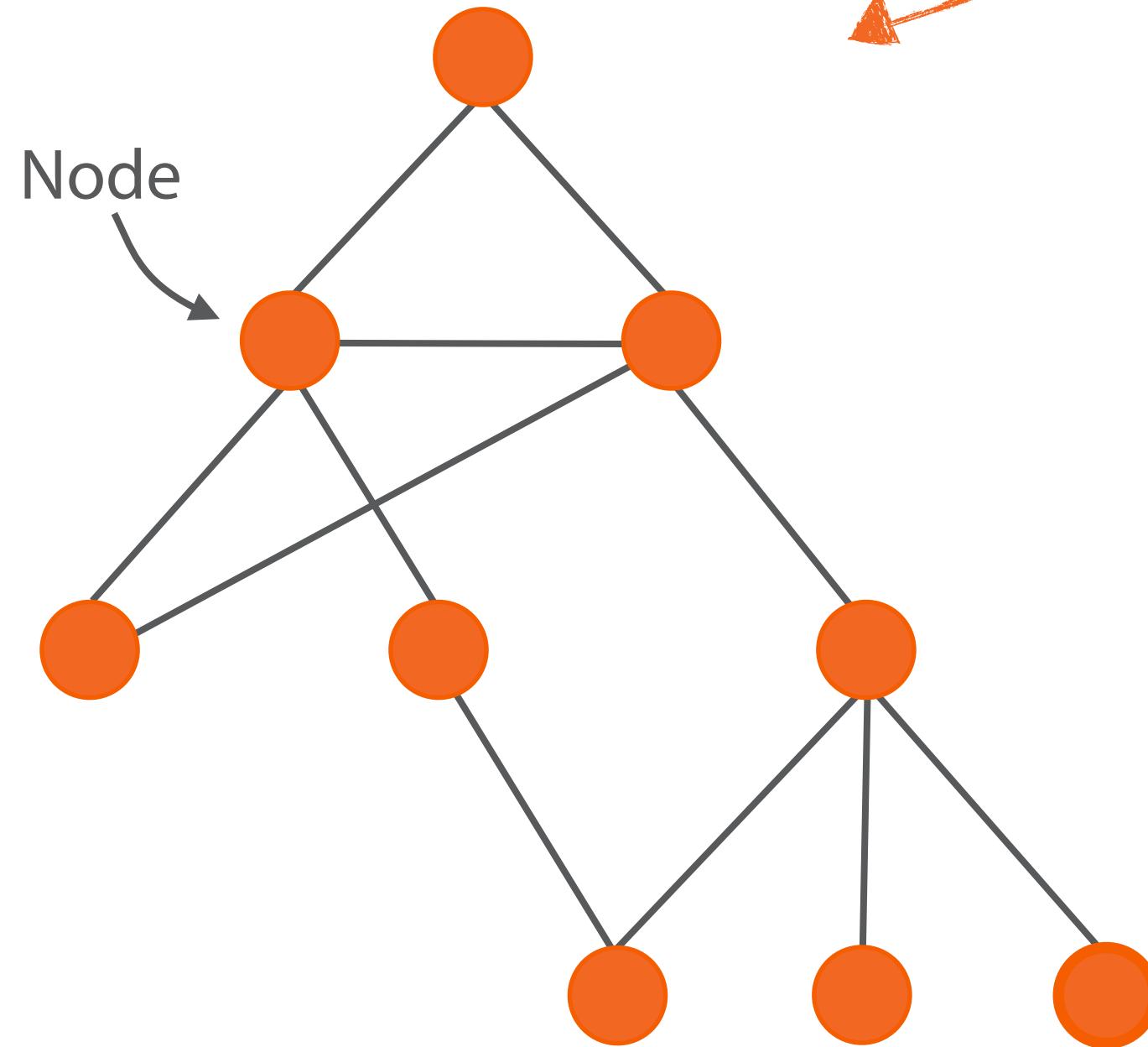
# Graphs and Trees



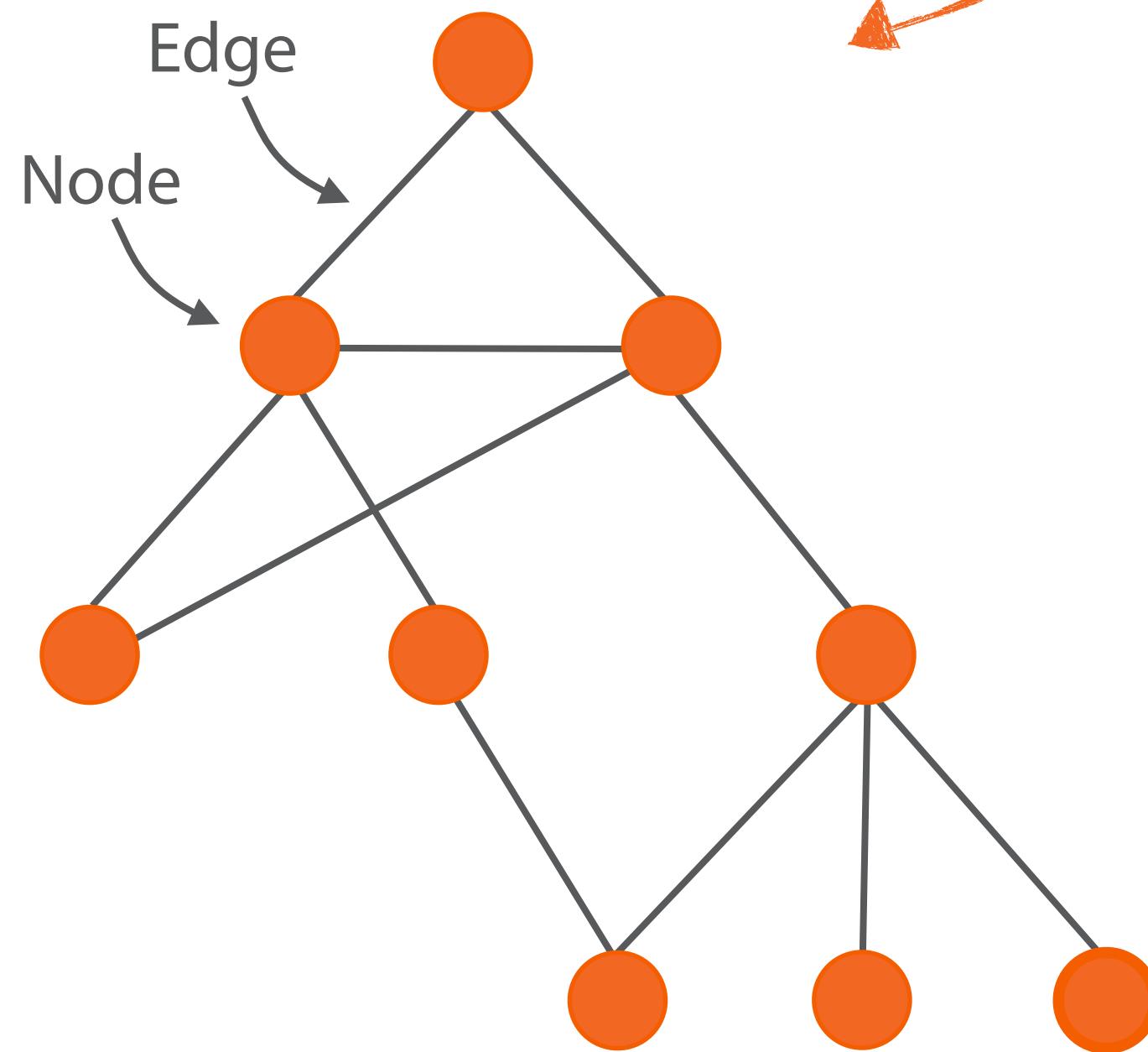
# Graphs and Trees



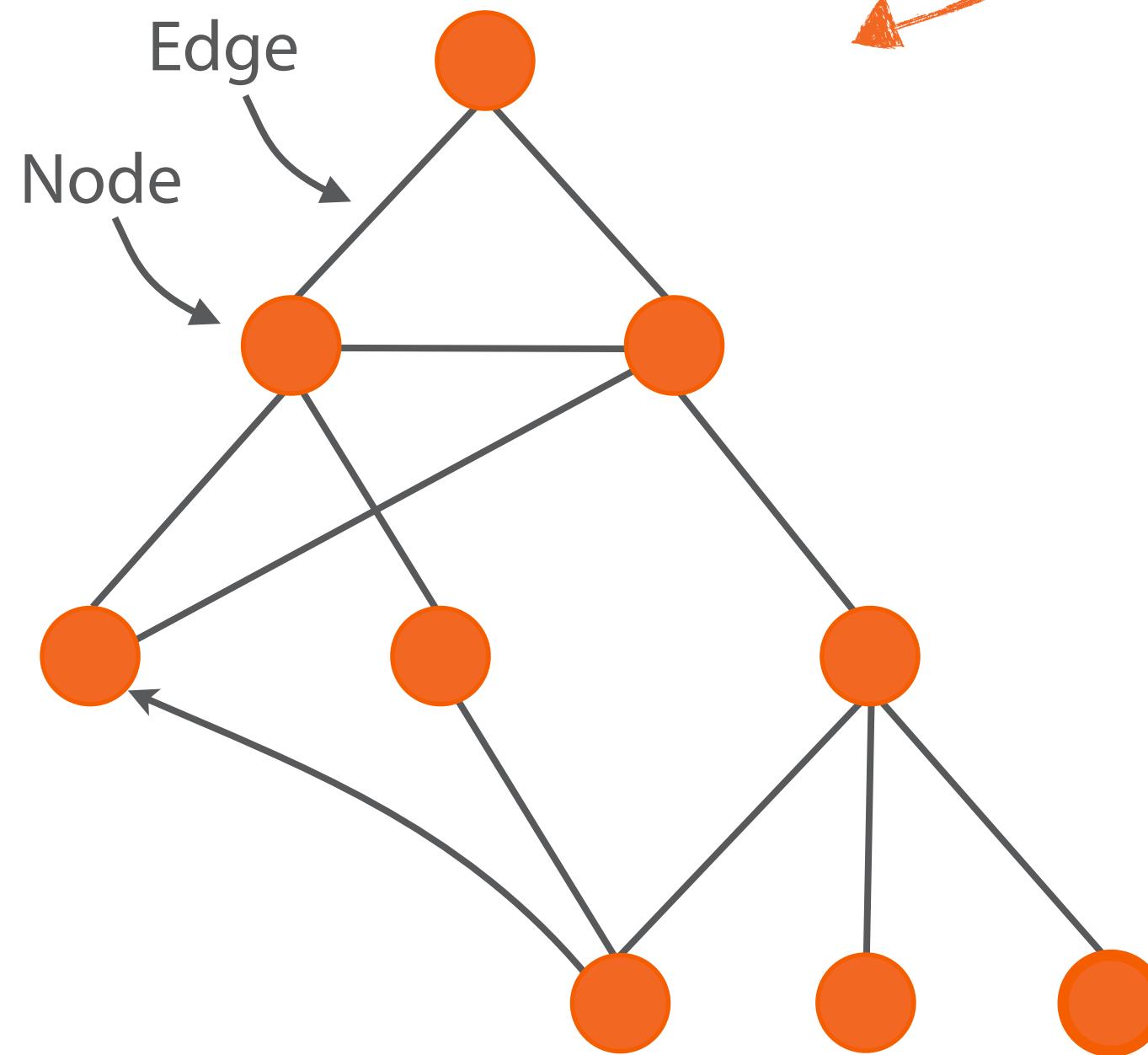
# Graphs and Trees



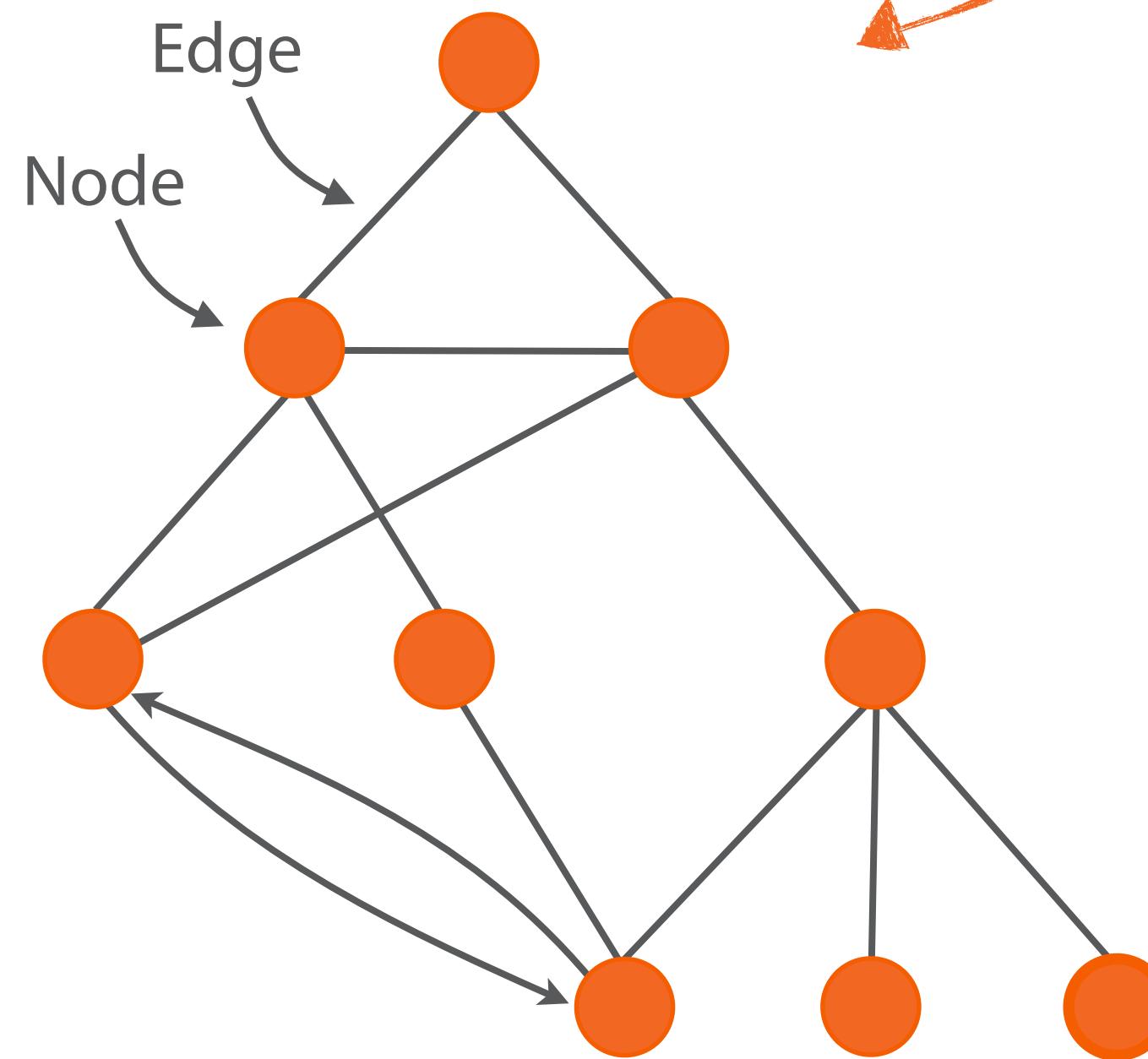
# Graphs and Trees



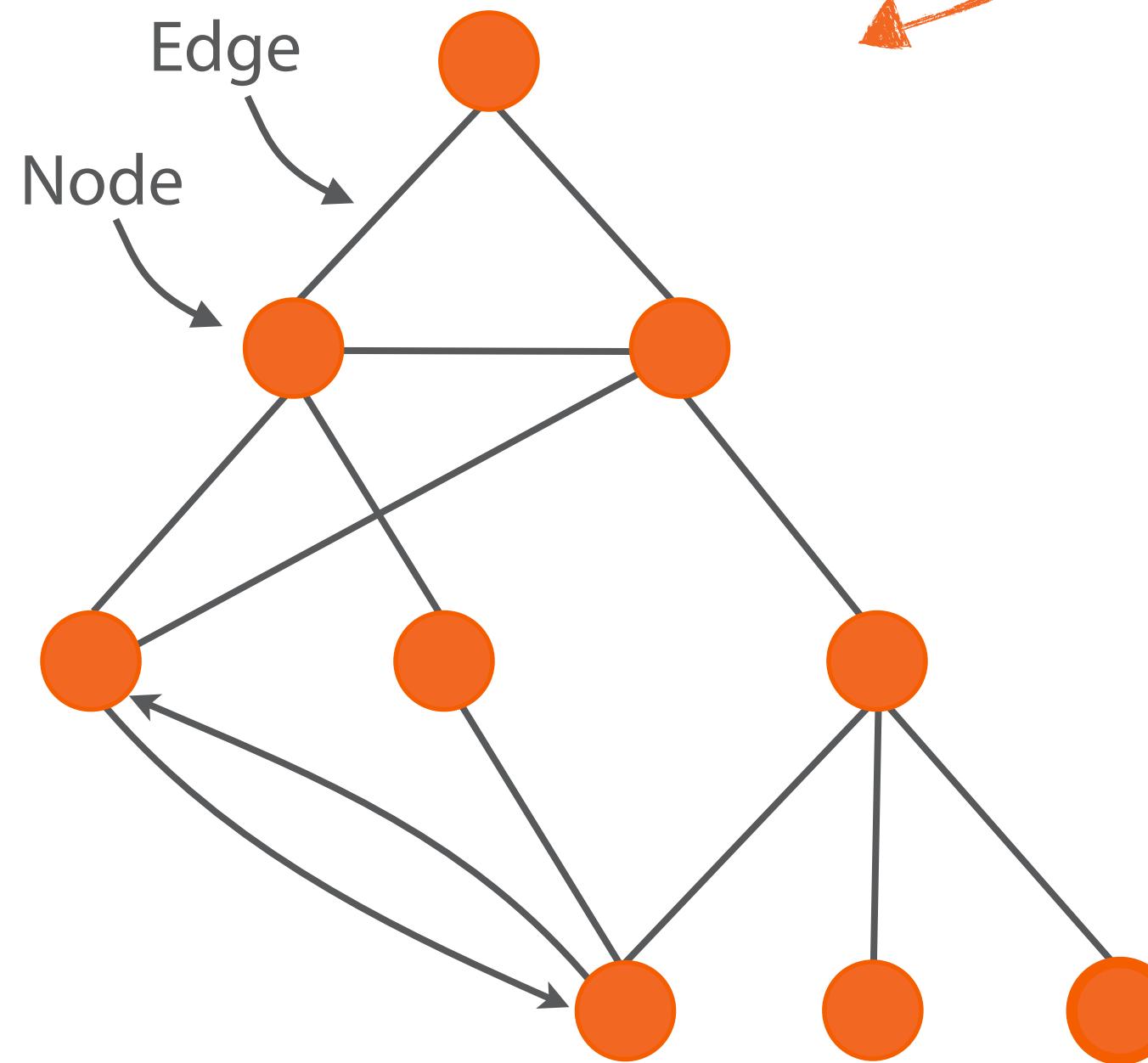
# Graphs and Trees



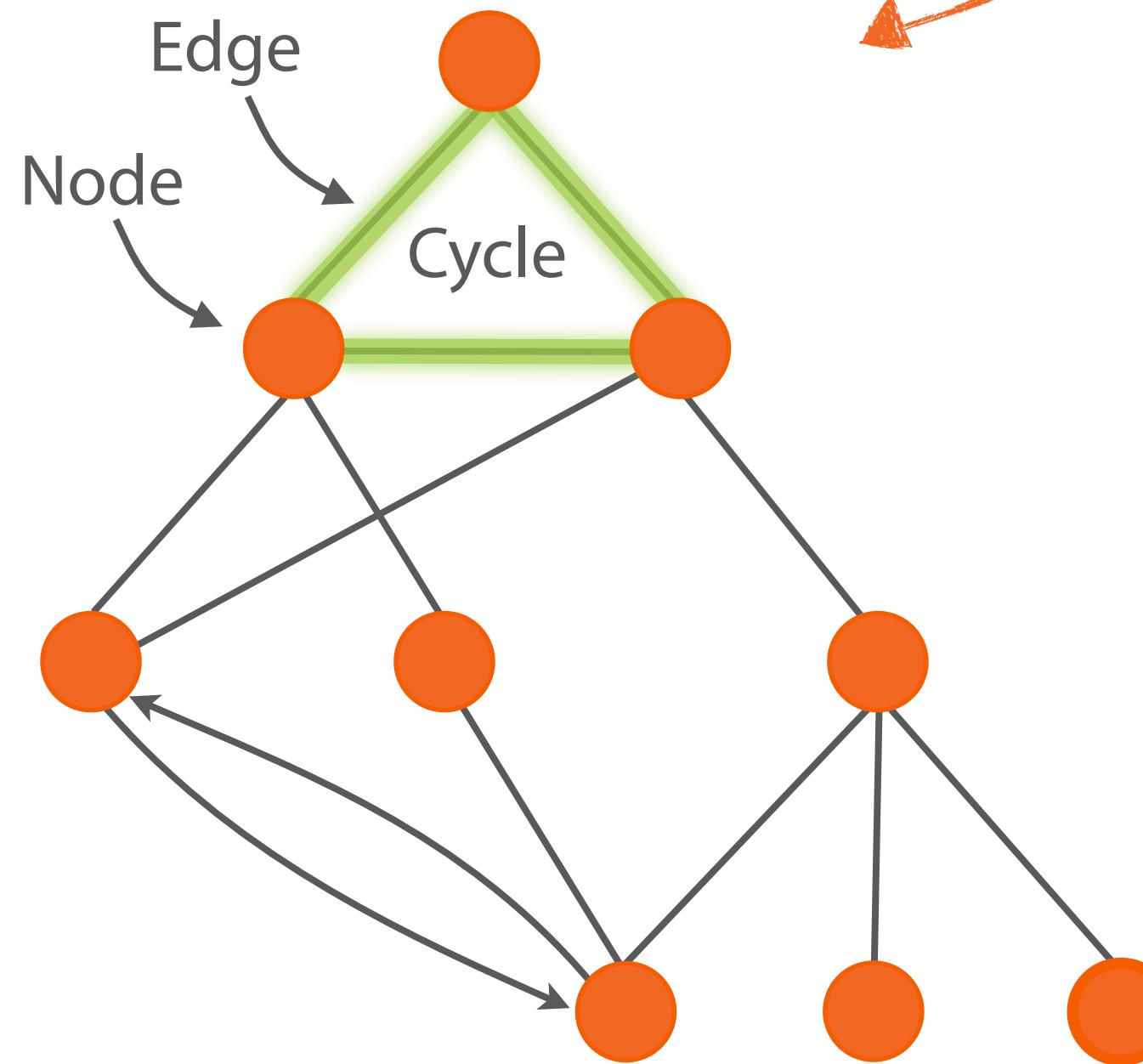
# Graphs and Trees



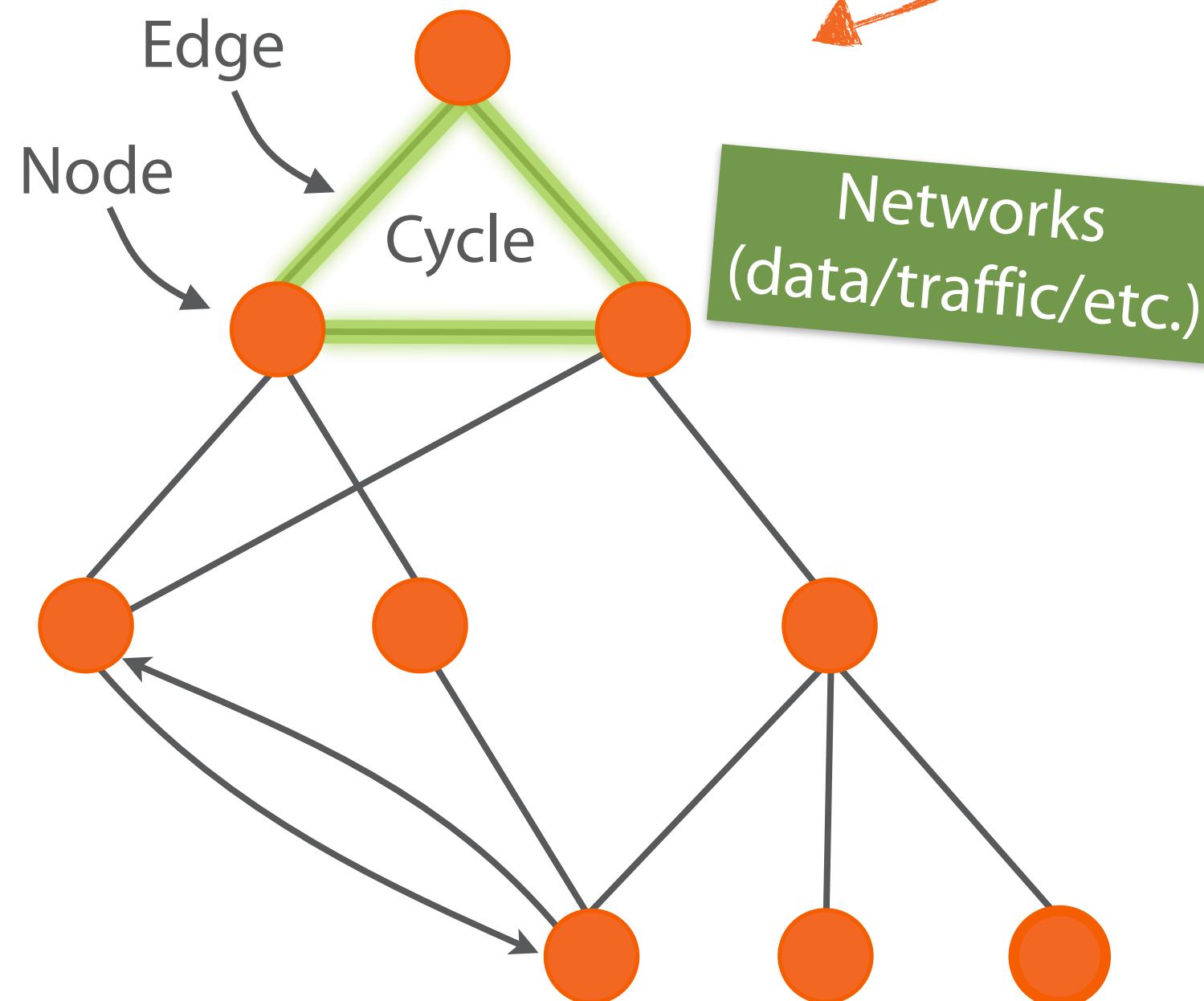
# (Directed) Graphs and Trees



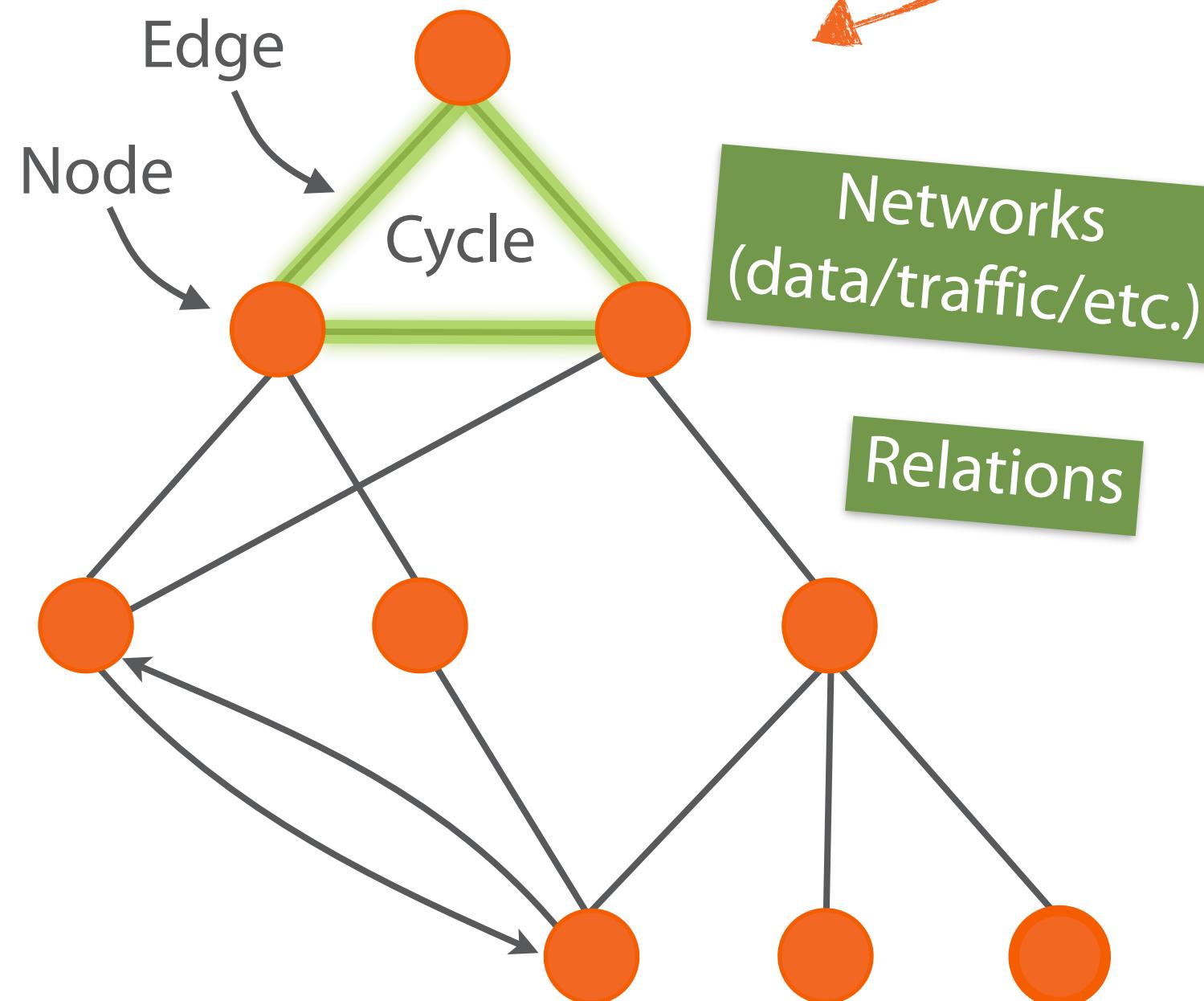
# (Directed) Graphs and Trees



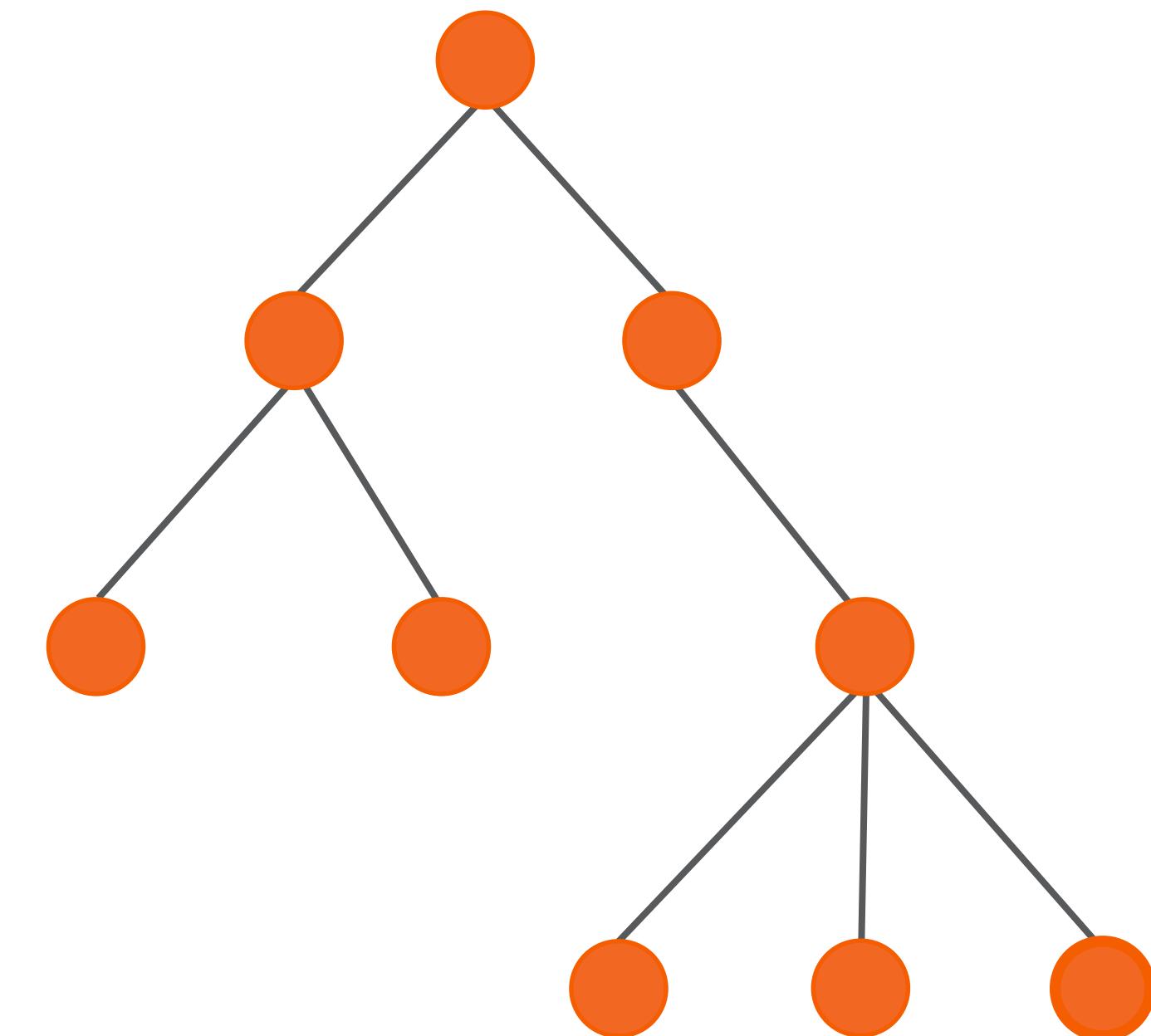
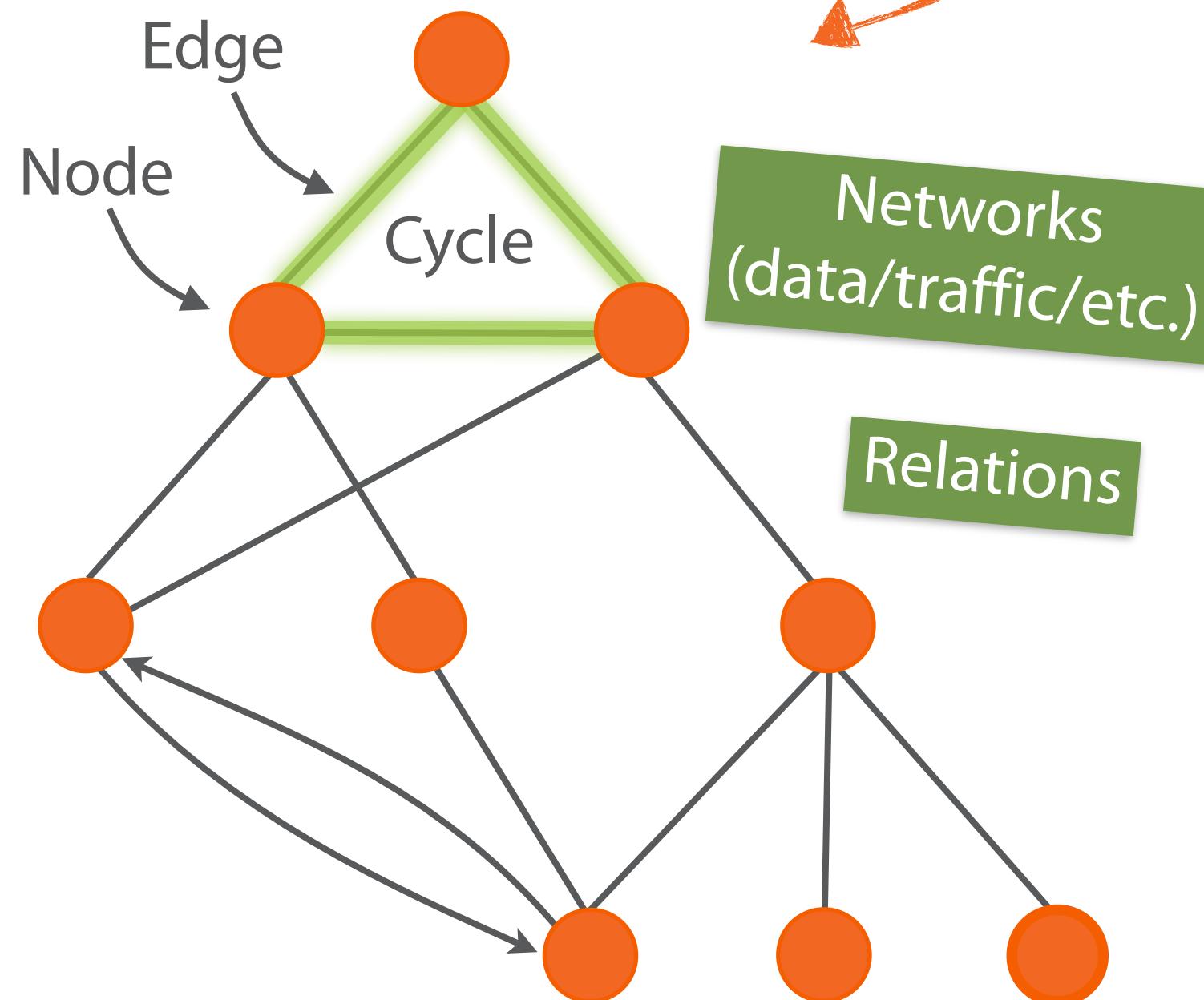
# (Directed) Graphs and Trees



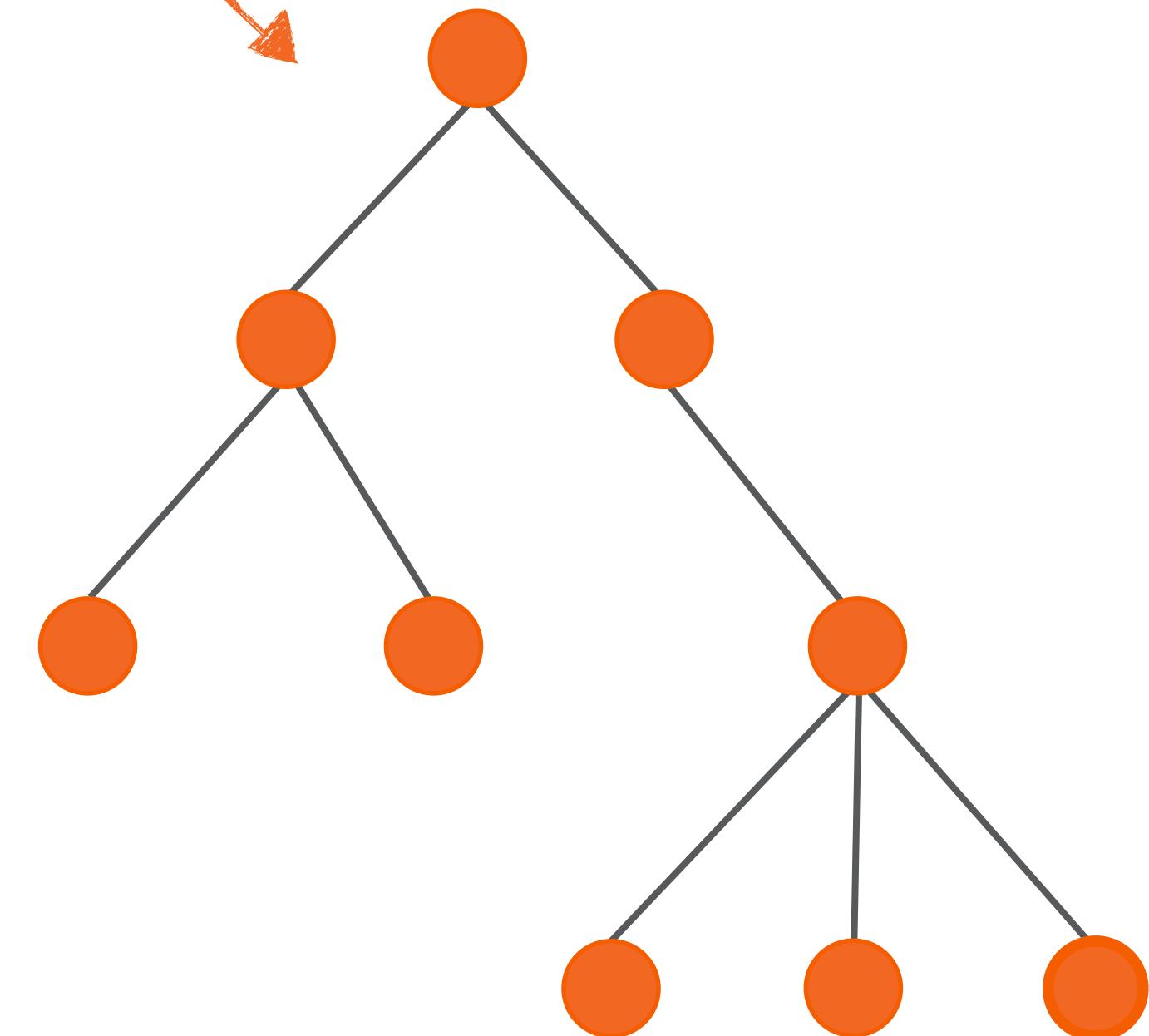
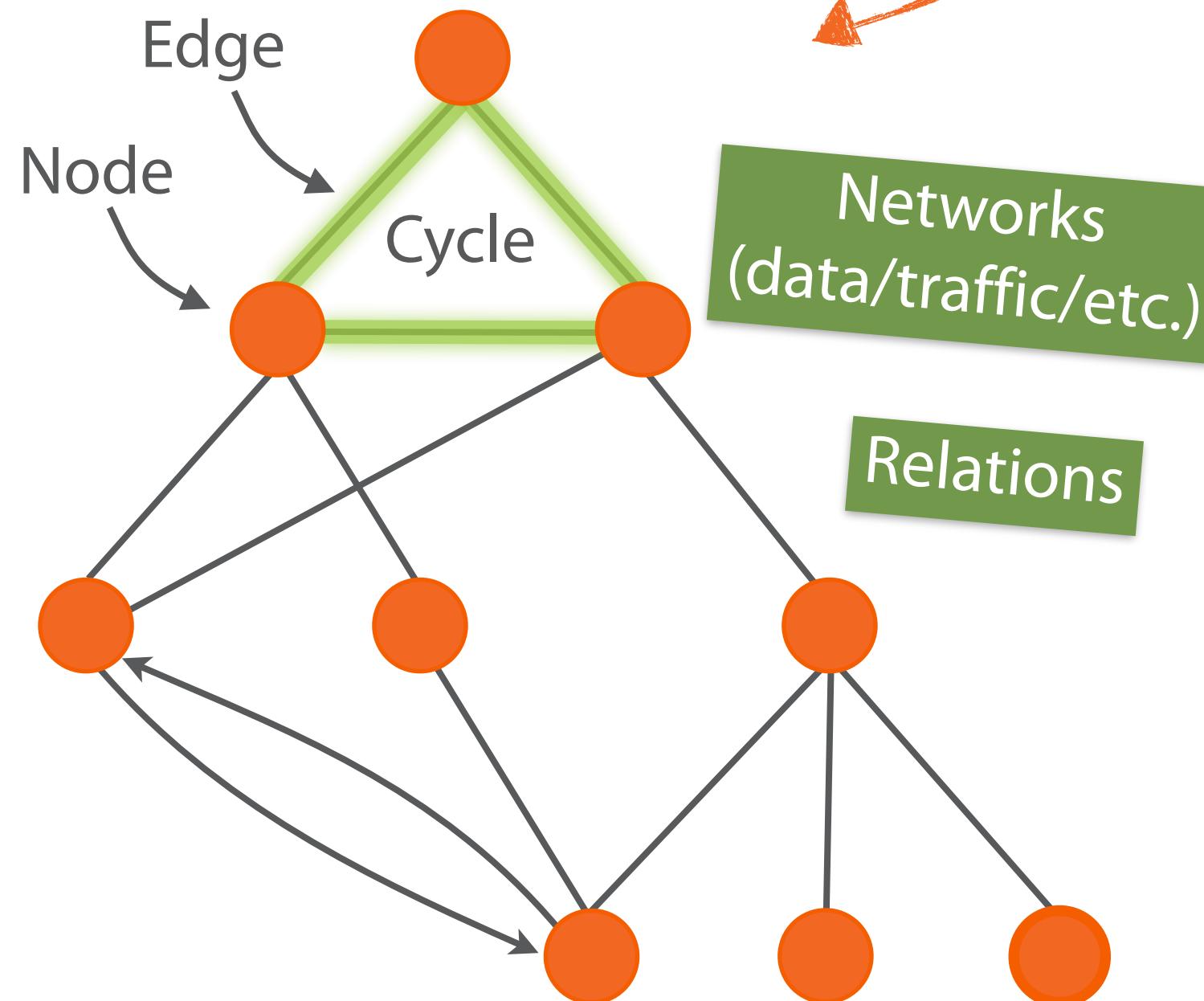
# (Directed) Graphs and Trees



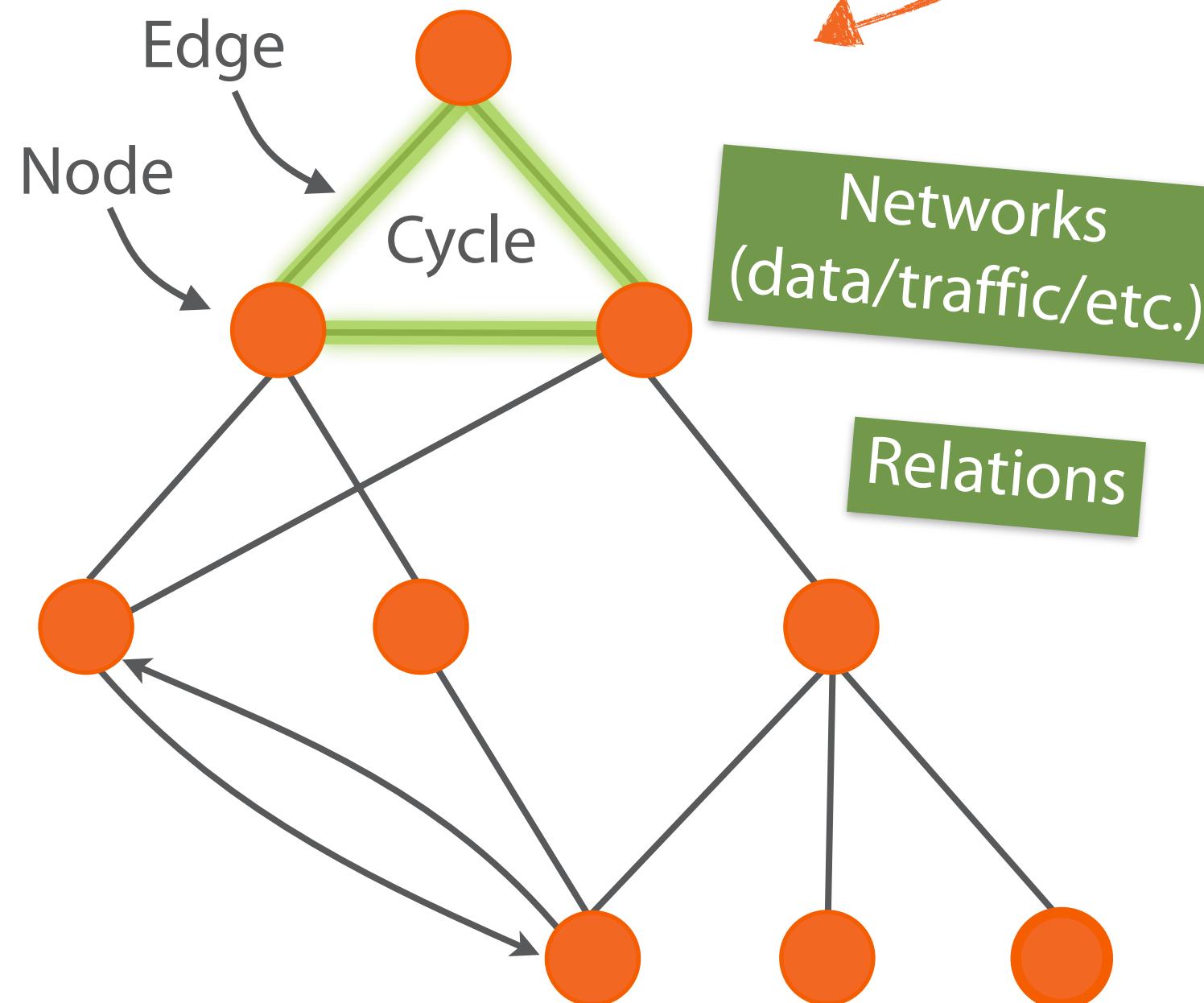
# (Directed) Graphs and Trees



# (Directed) Graphs and Trees

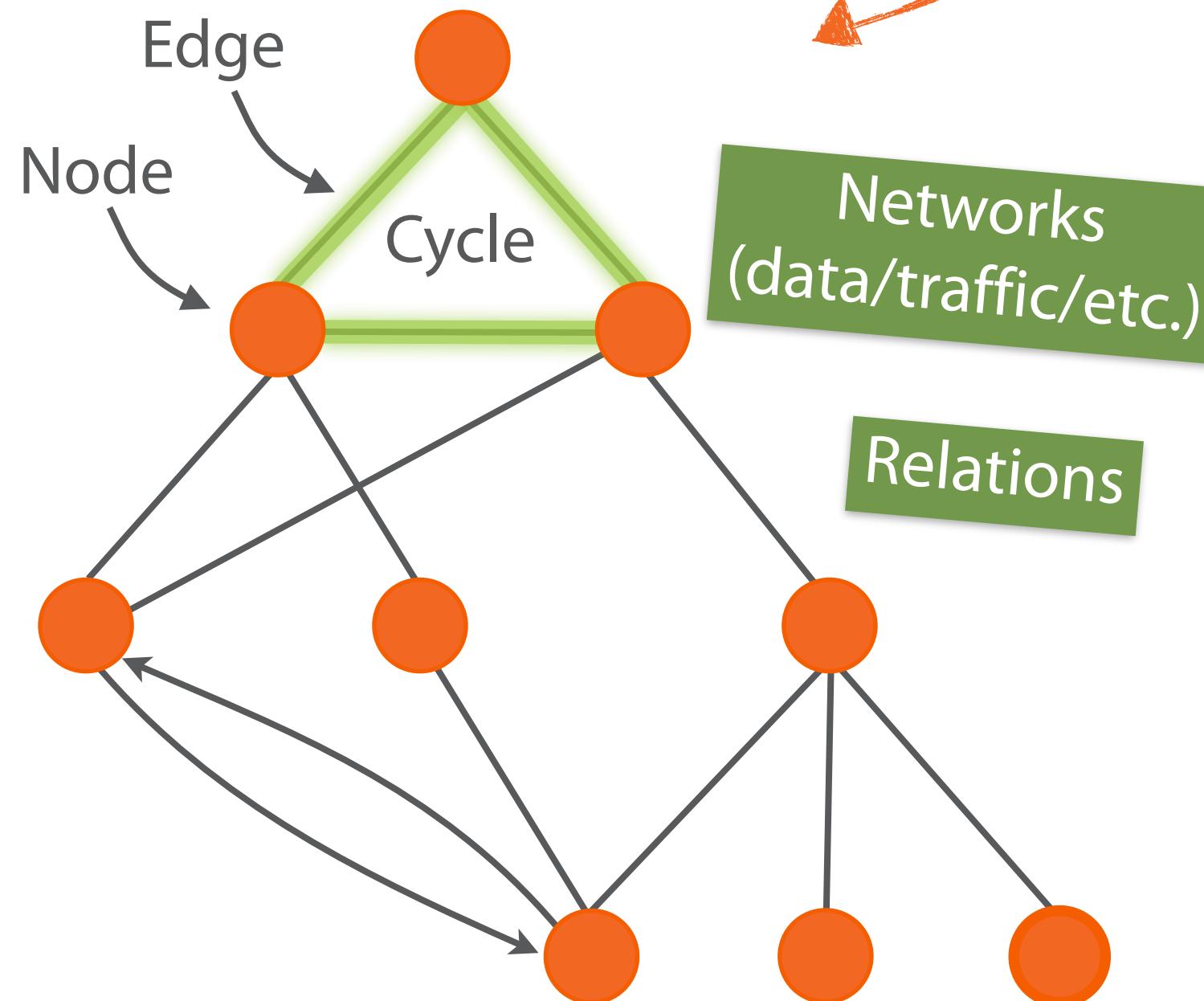


# (Directed) Graphs and Trees



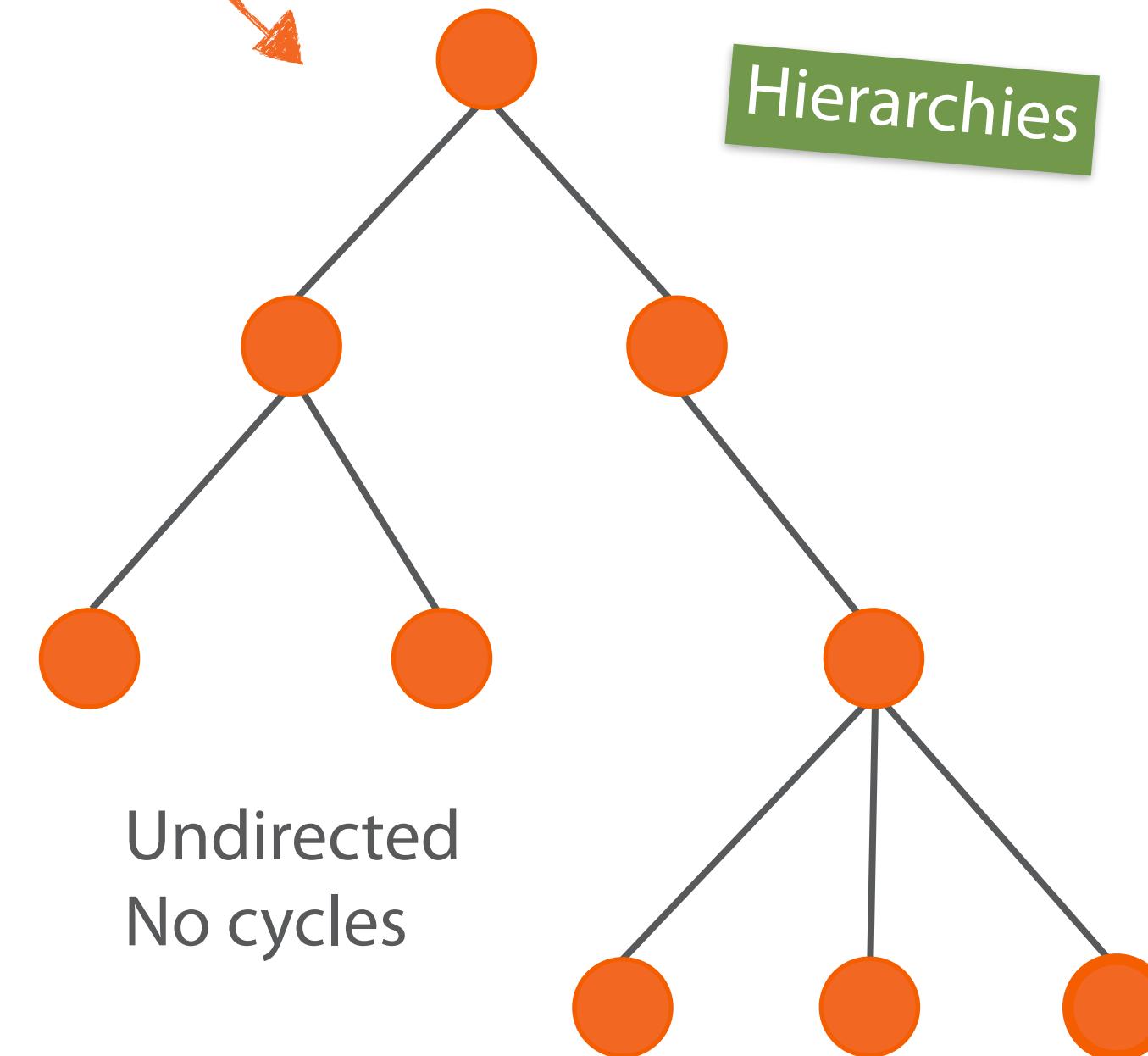
Undirected  
No cycles

# (Directed) Graphs and Trees

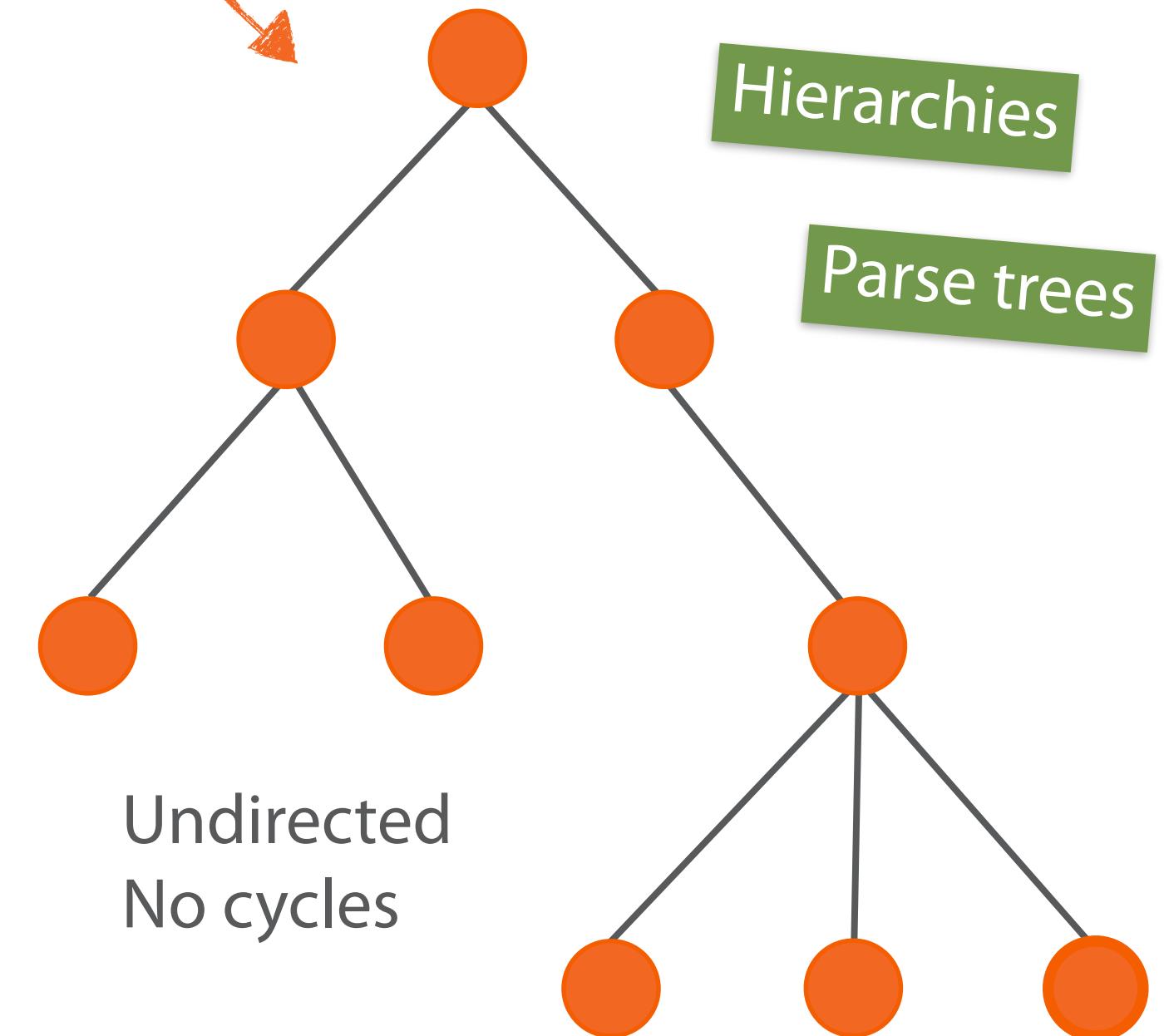
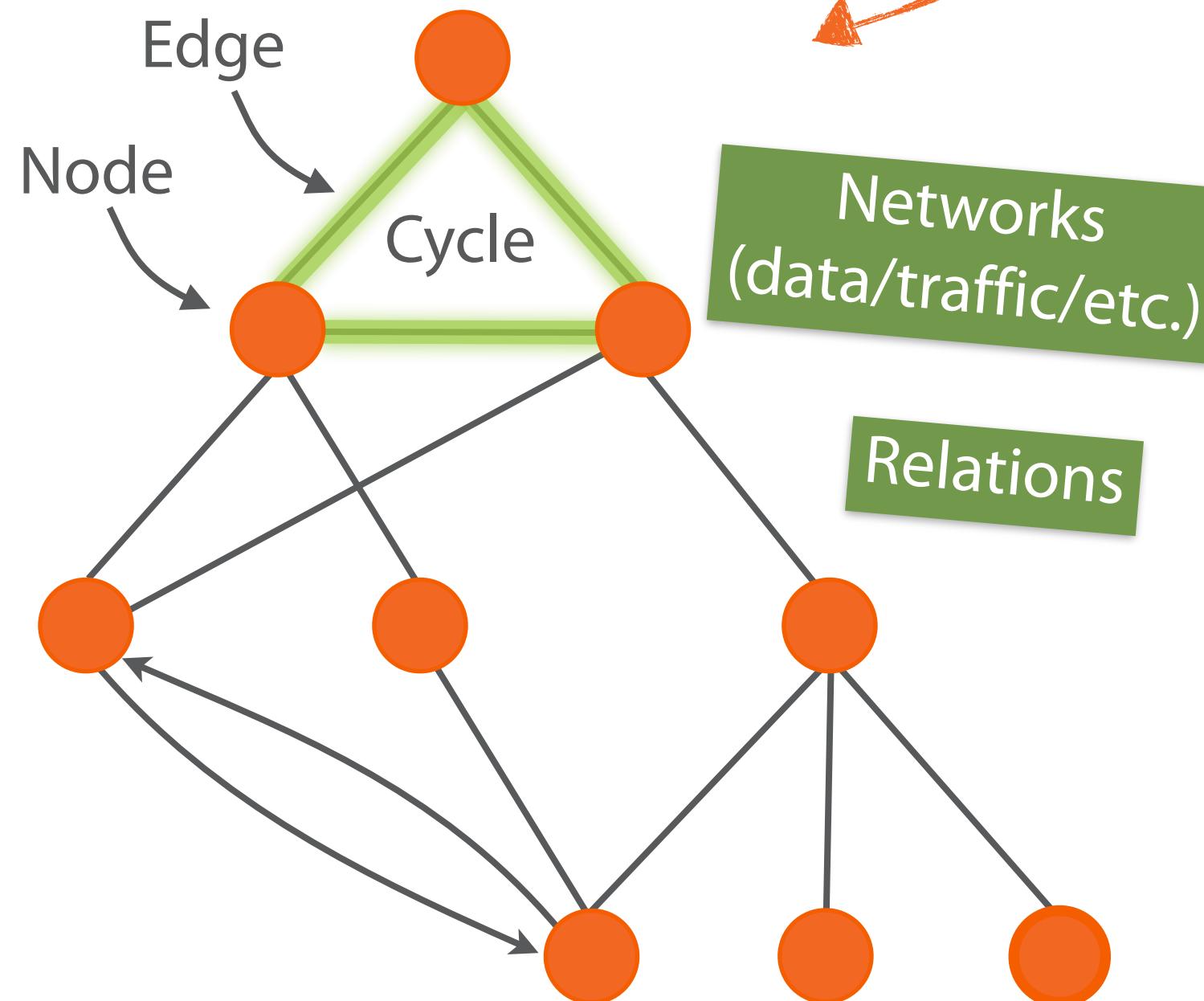


Networks  
(data/traffic/etc.)

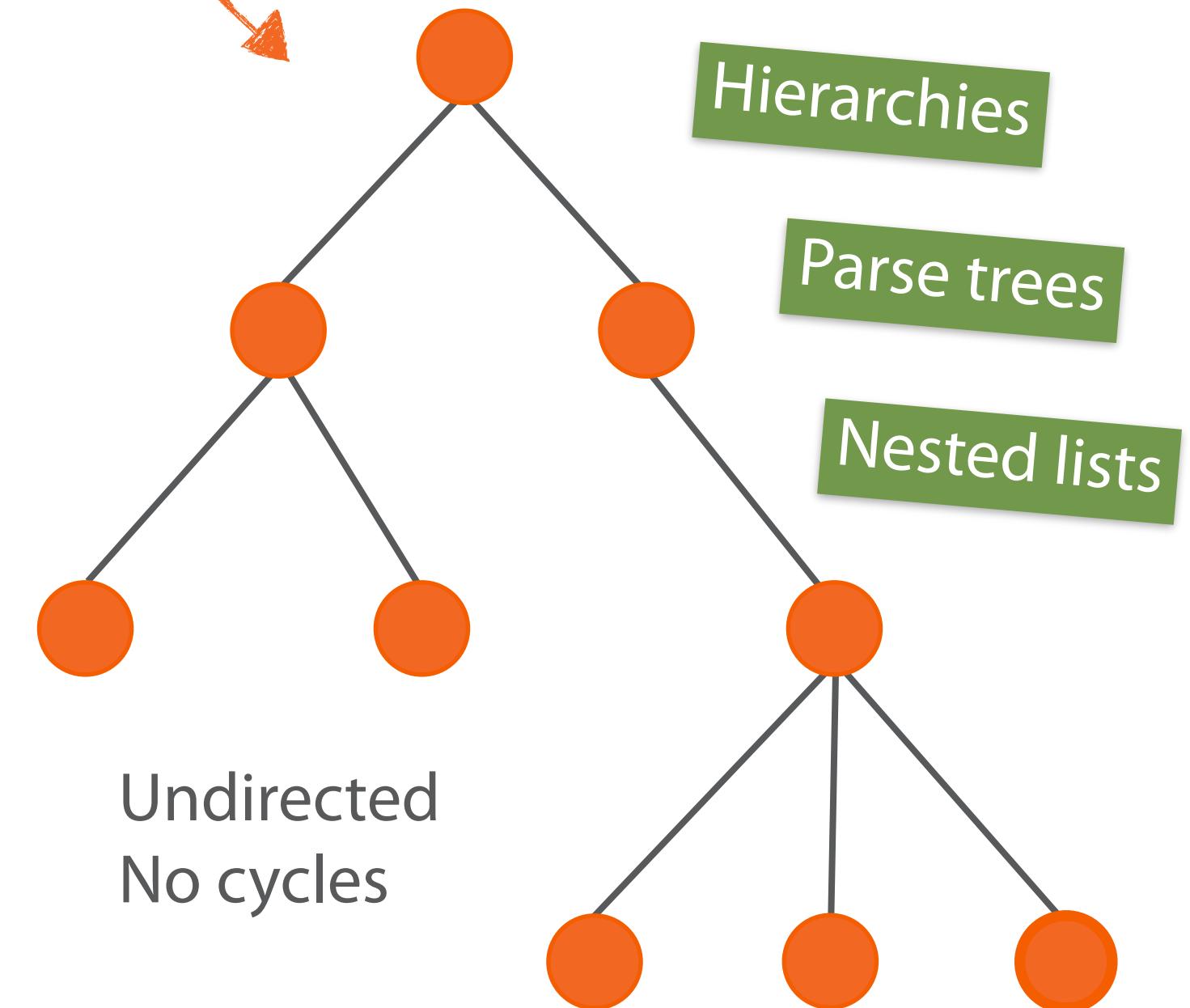
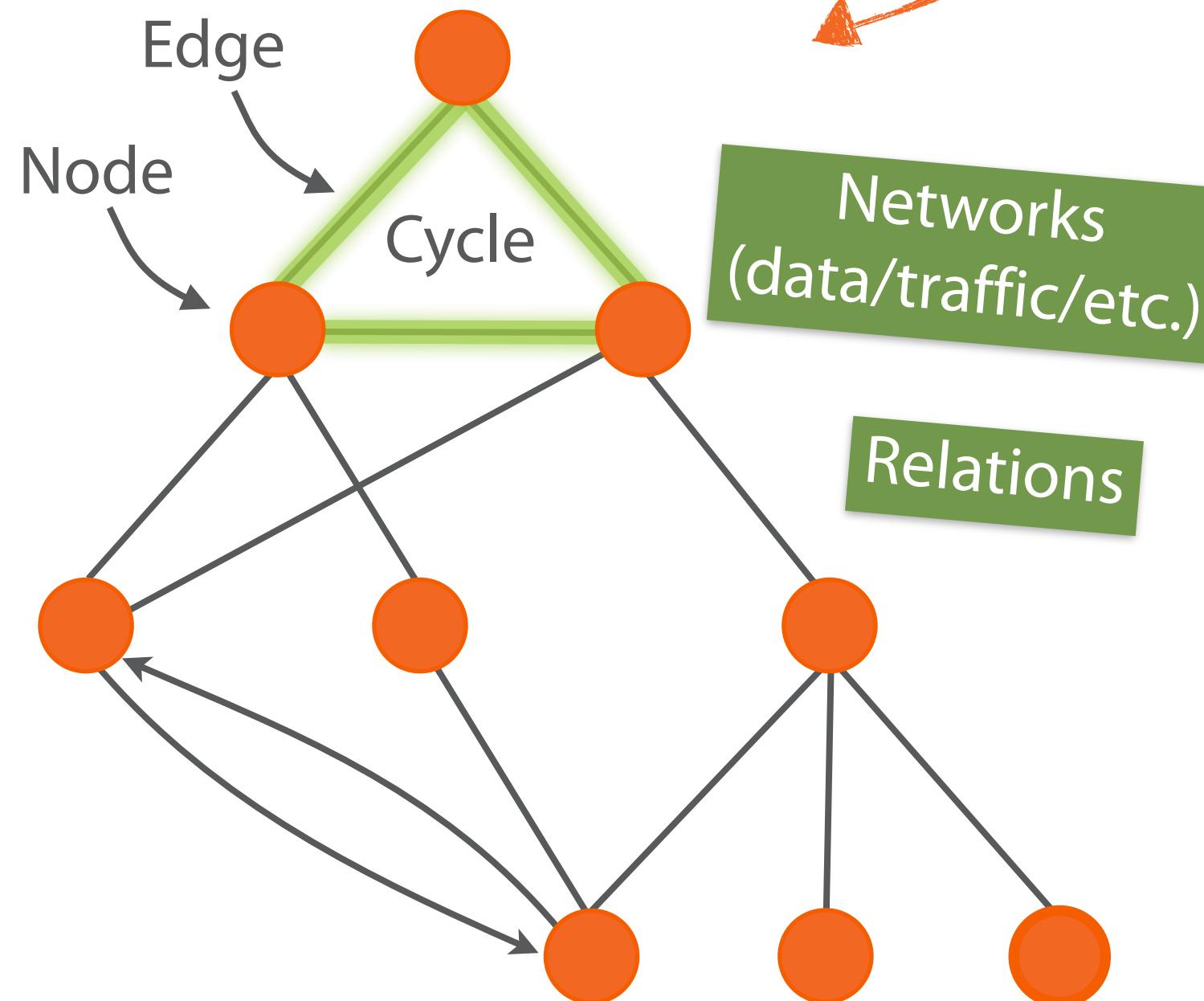
Relations



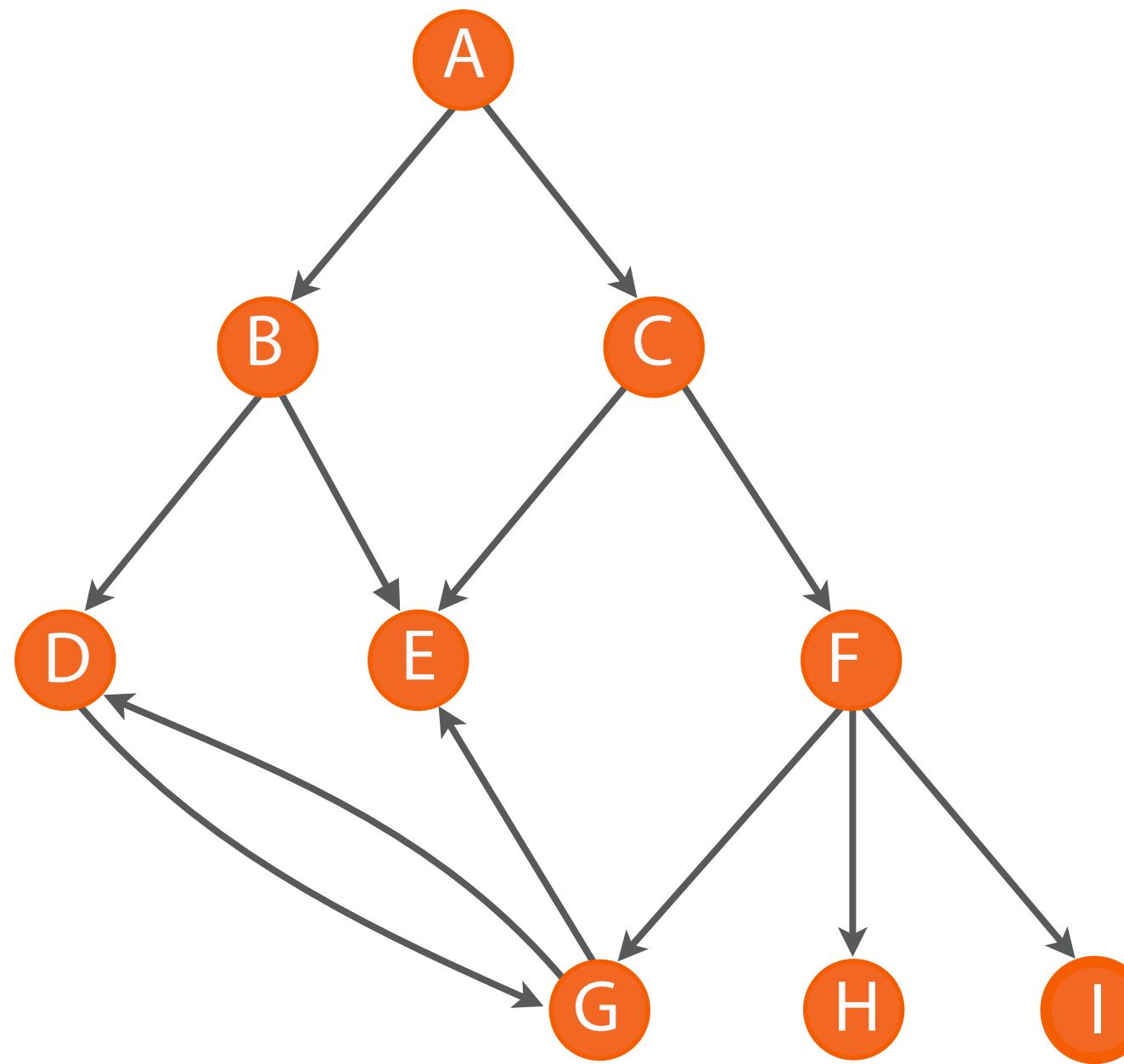
# (Directed) Graphs and Trees



# (Directed) Graphs and Trees

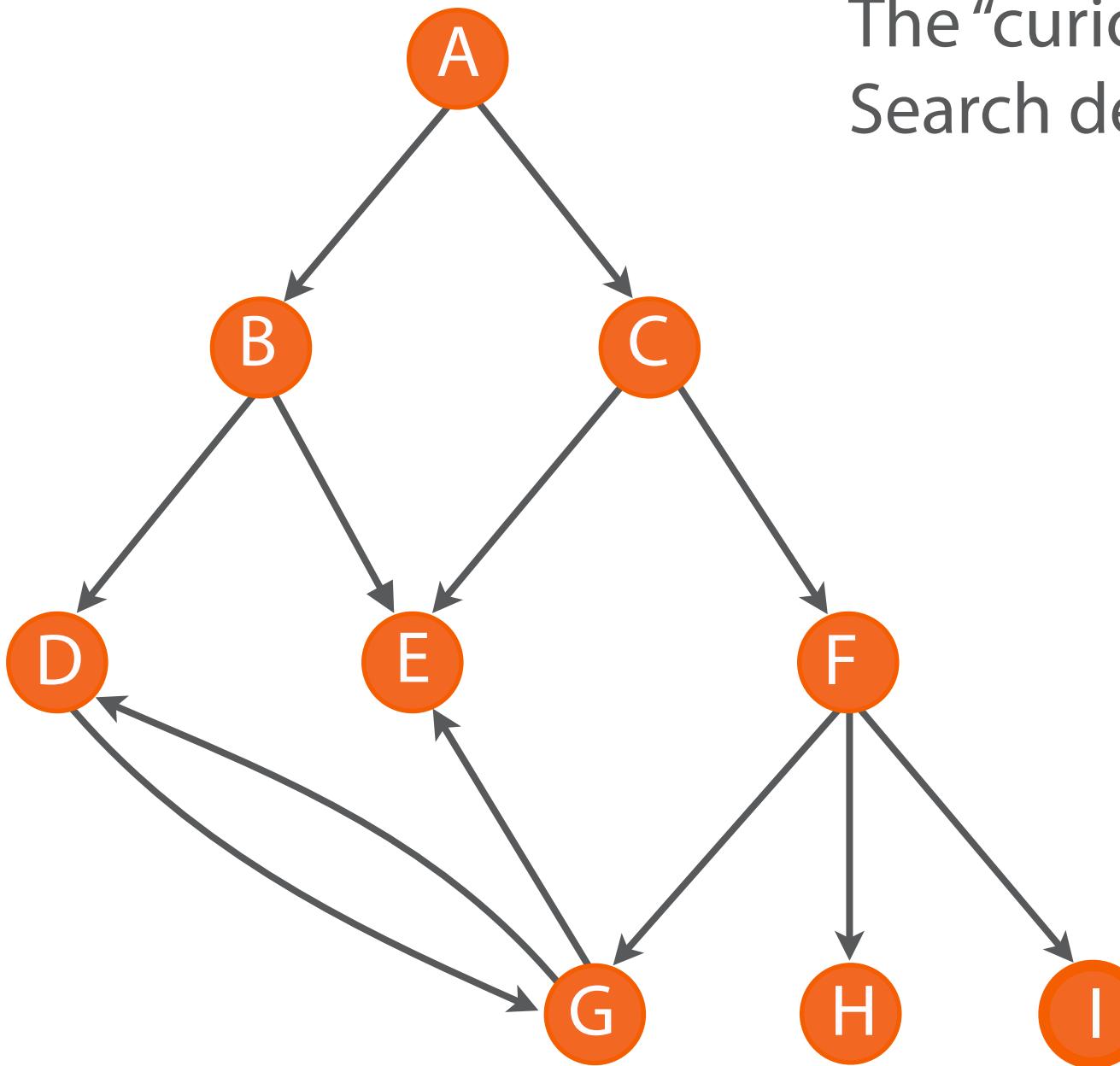


# Depth First Search (DFS)



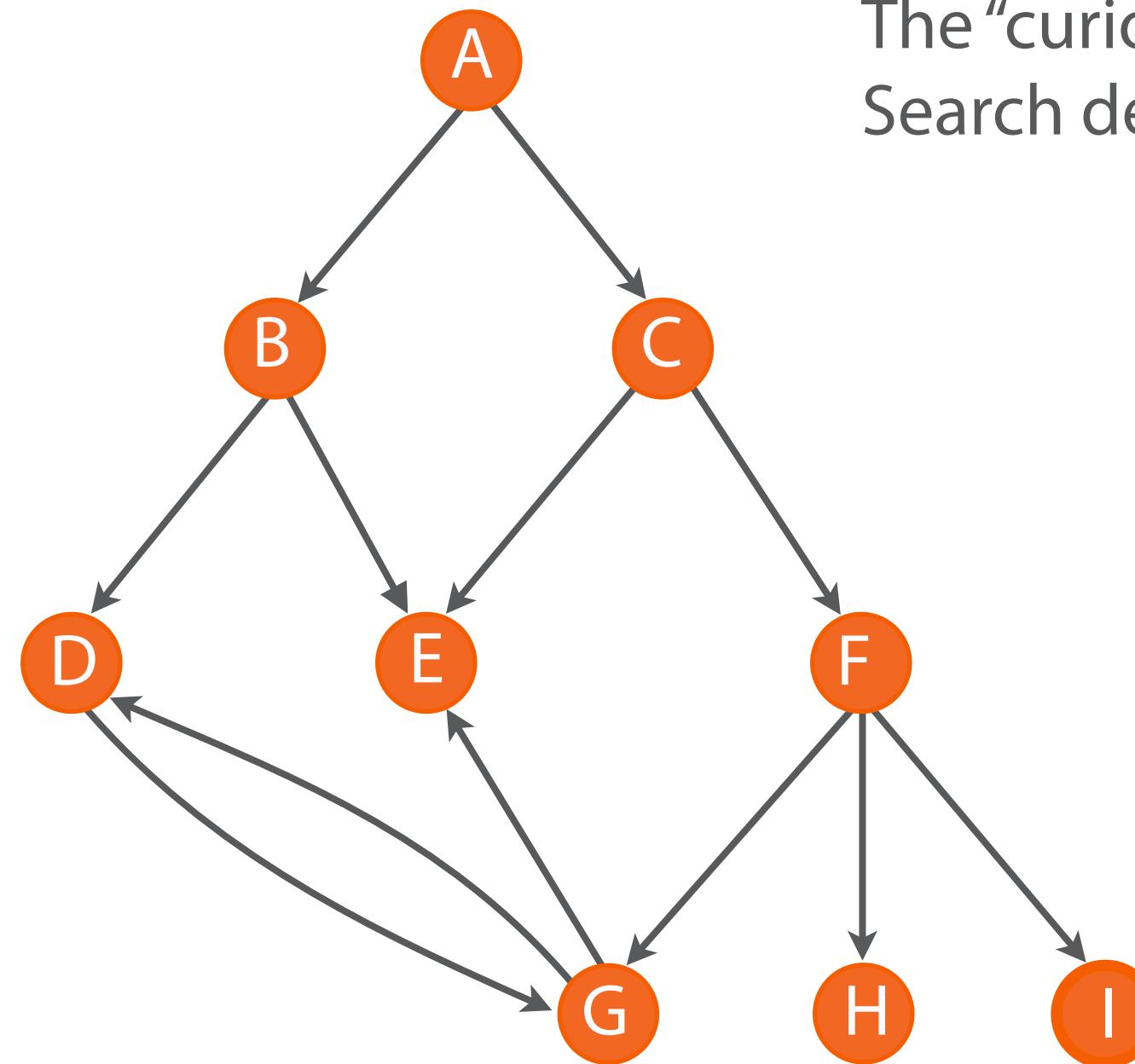
# Depth First Search (DFS)

The “curious” strategy:  
Search deeper nodes first.



# Depth First Search (DFS)

Typical default graph traversal strategy.



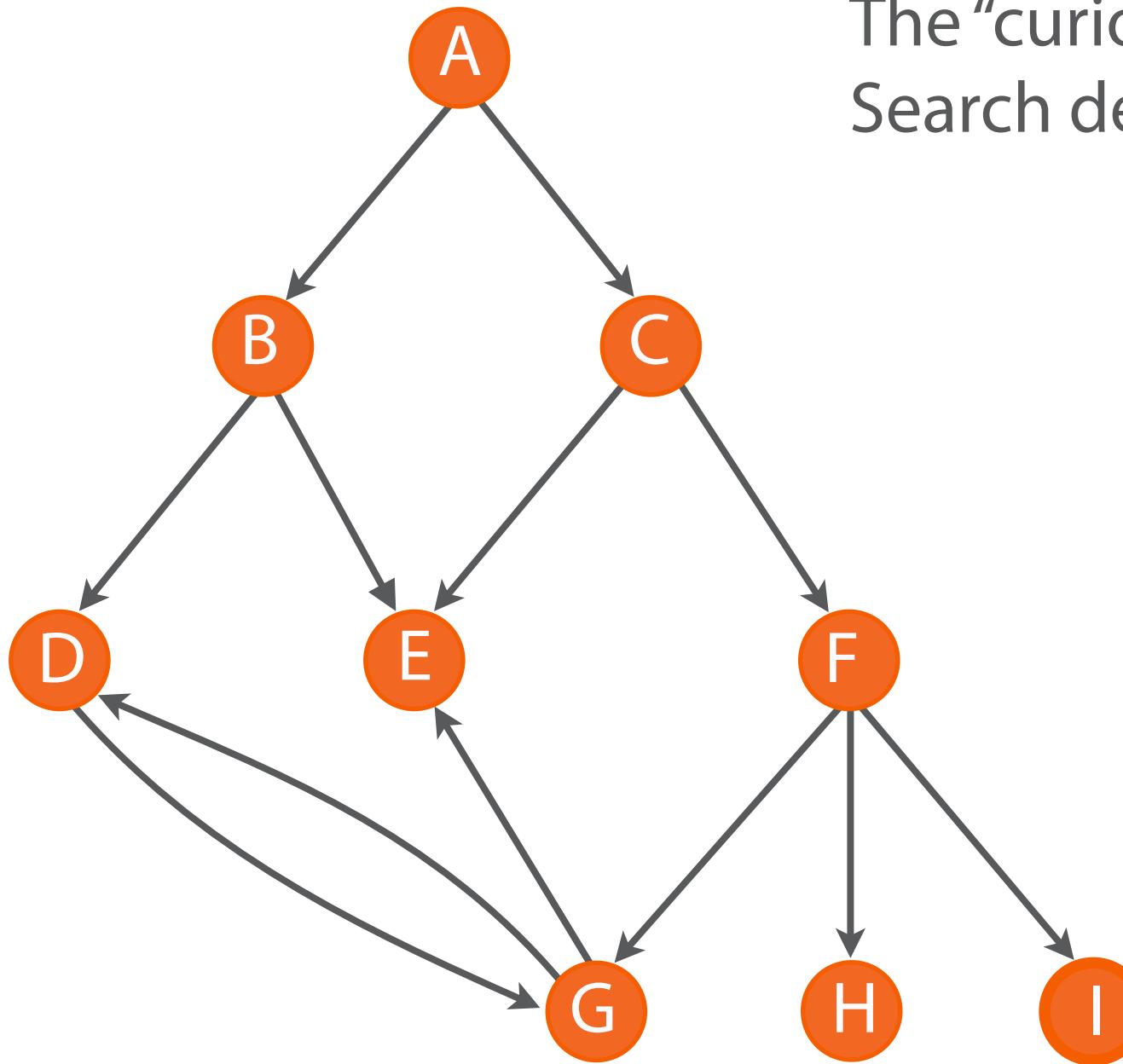
The “curious” strategy:  
Search deeper nodes first.

# Depth First Search (DFS)

Typical default graph traversal strategy.

Easy recursive implementation.

The “curious” strategy:  
Search deeper nodes first.

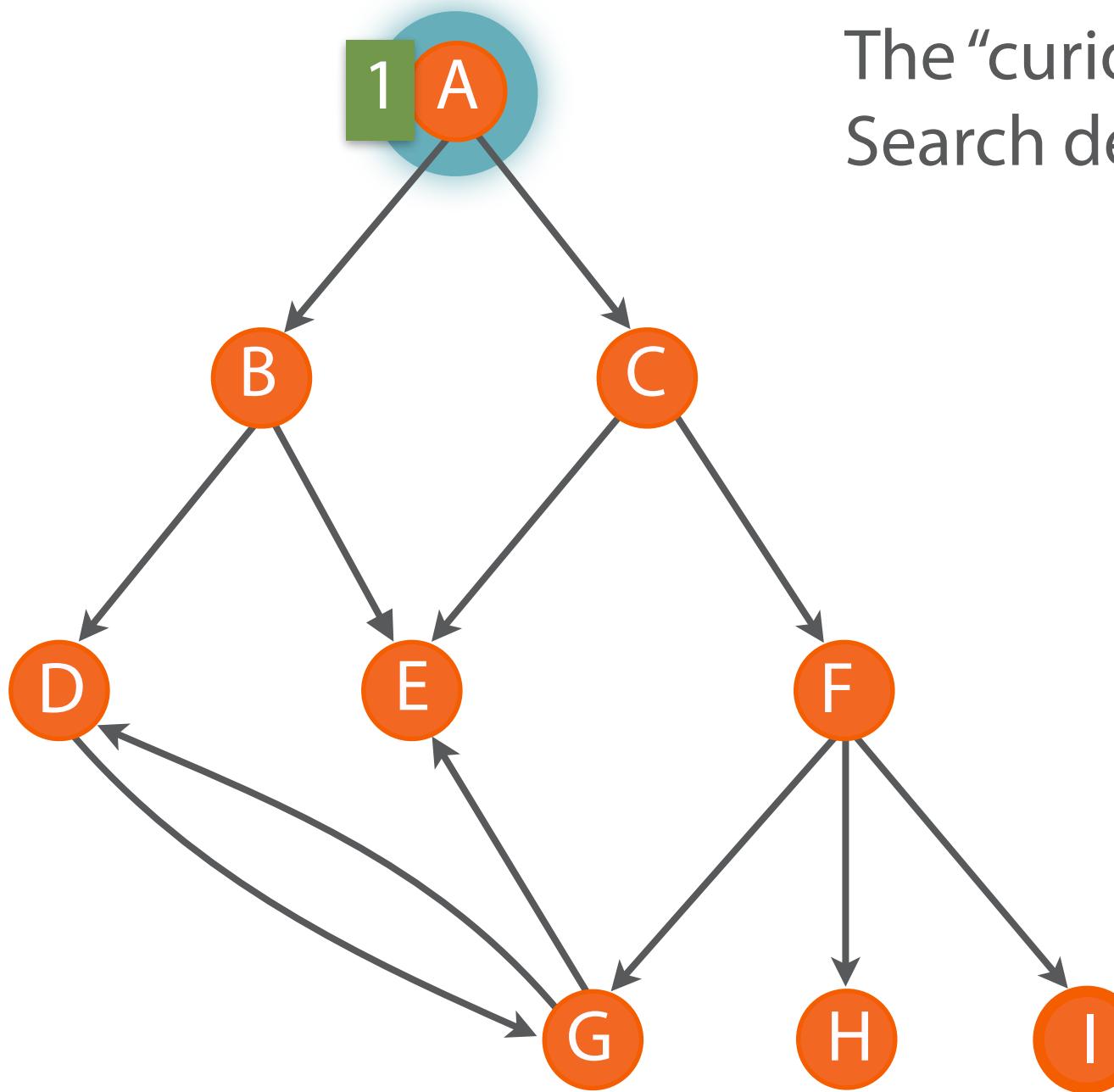


# Depth First Search (DFS)

Typical default graph traversal strategy.

Easy recursive implementation.

The “curious” strategy:  
Search deeper nodes first.

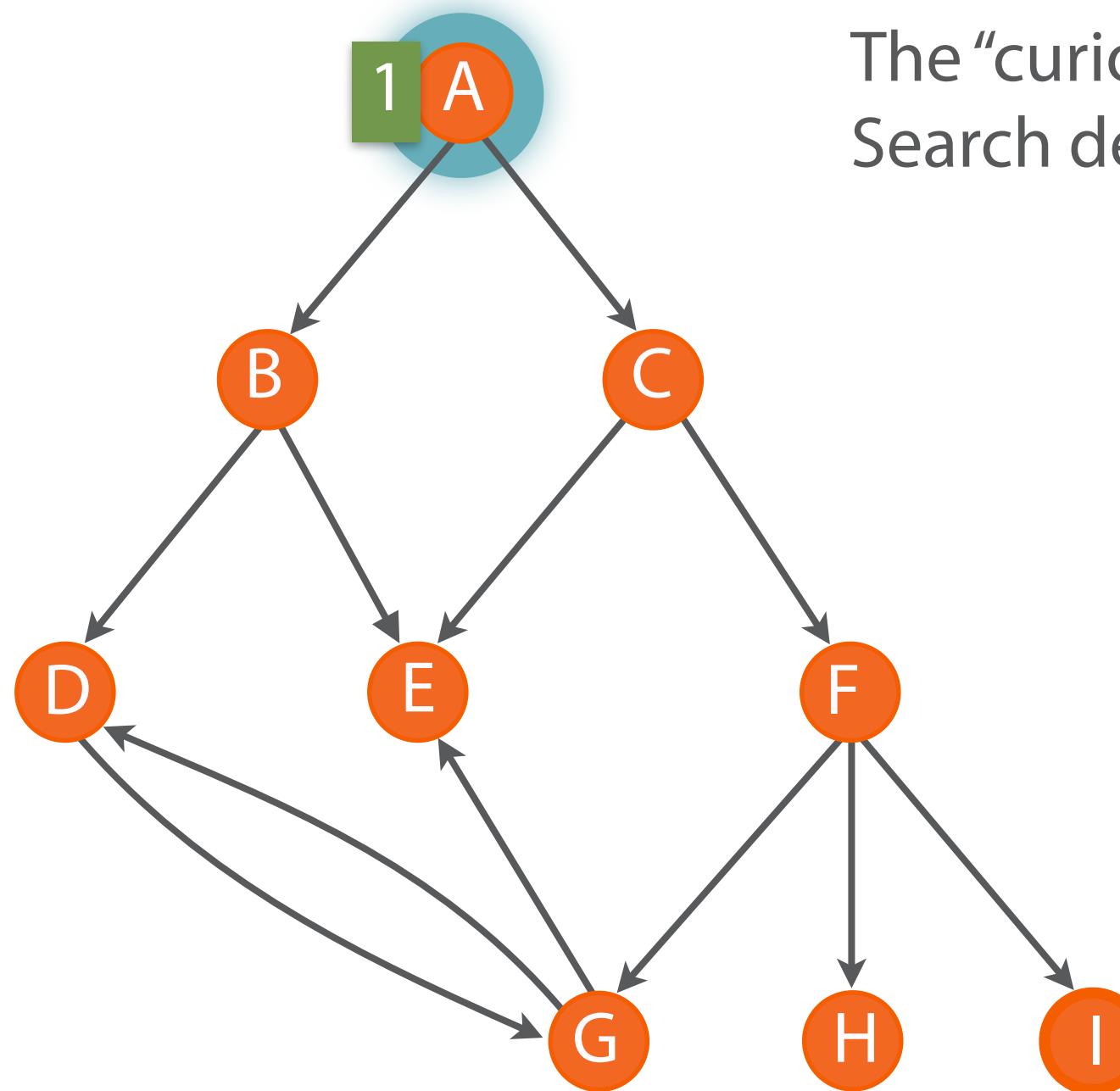


# Depth First Search (DFS)

Typical default graph traversal strategy.

Easy recursive implementation.

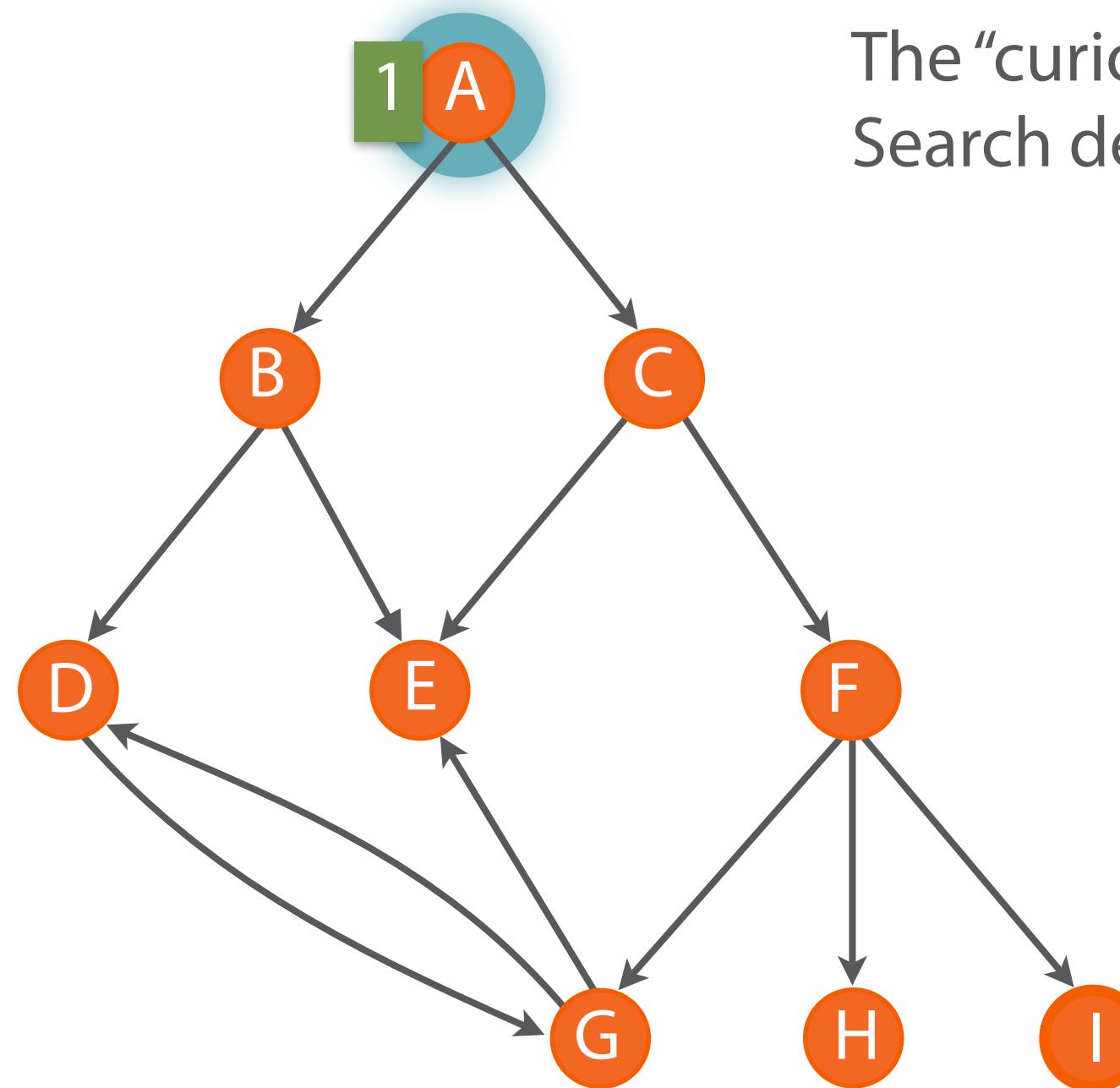
The “curious” strategy:  
Search deeper nodes first.



# Depth First Search (DFS)

Typical default graph traversal strategy.

Easy recursive implementation.



The “curious” strategy:  
Search deeper nodes first.

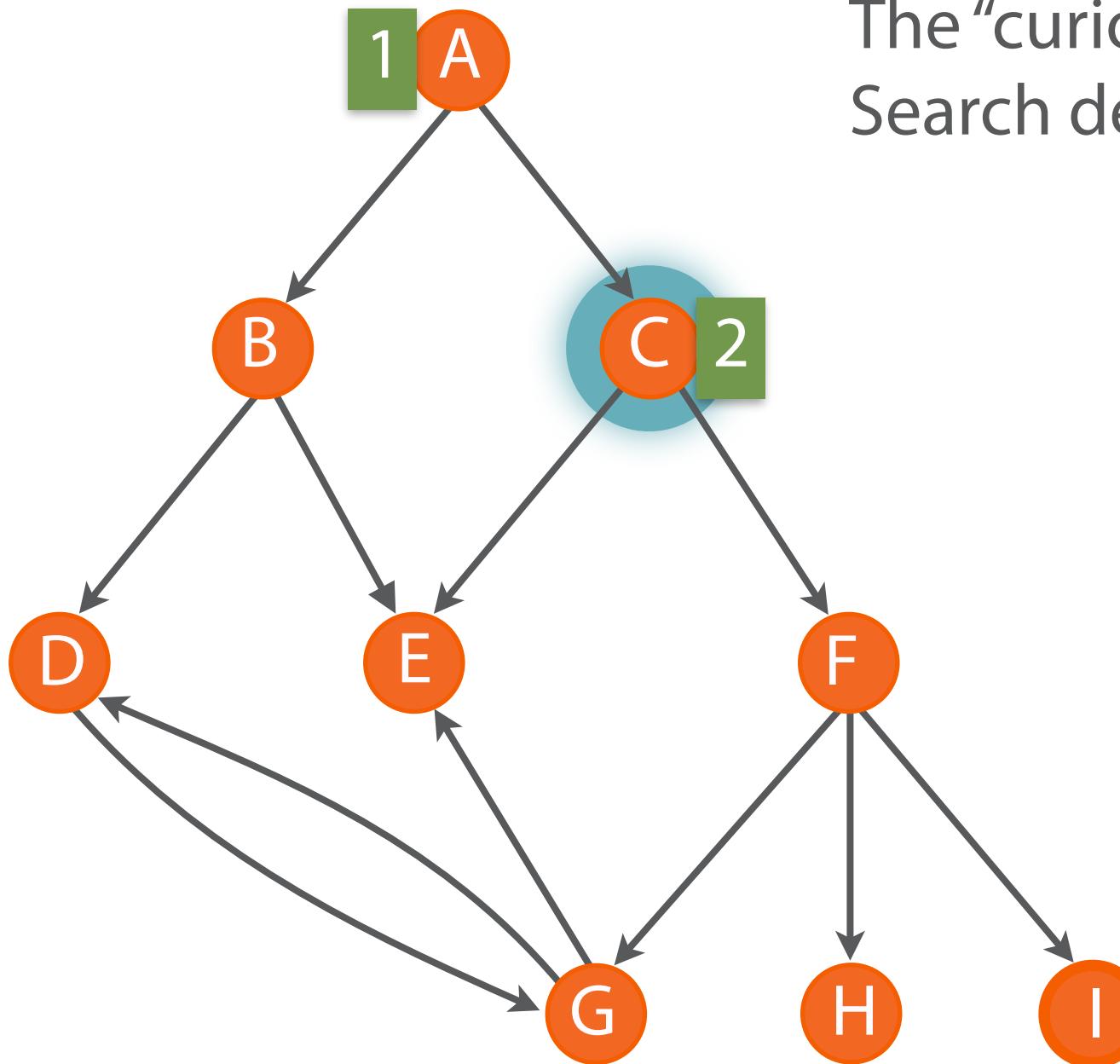


# Depth First Search (DFS)

Typical default graph traversal strategy.

Easy recursive implementation.

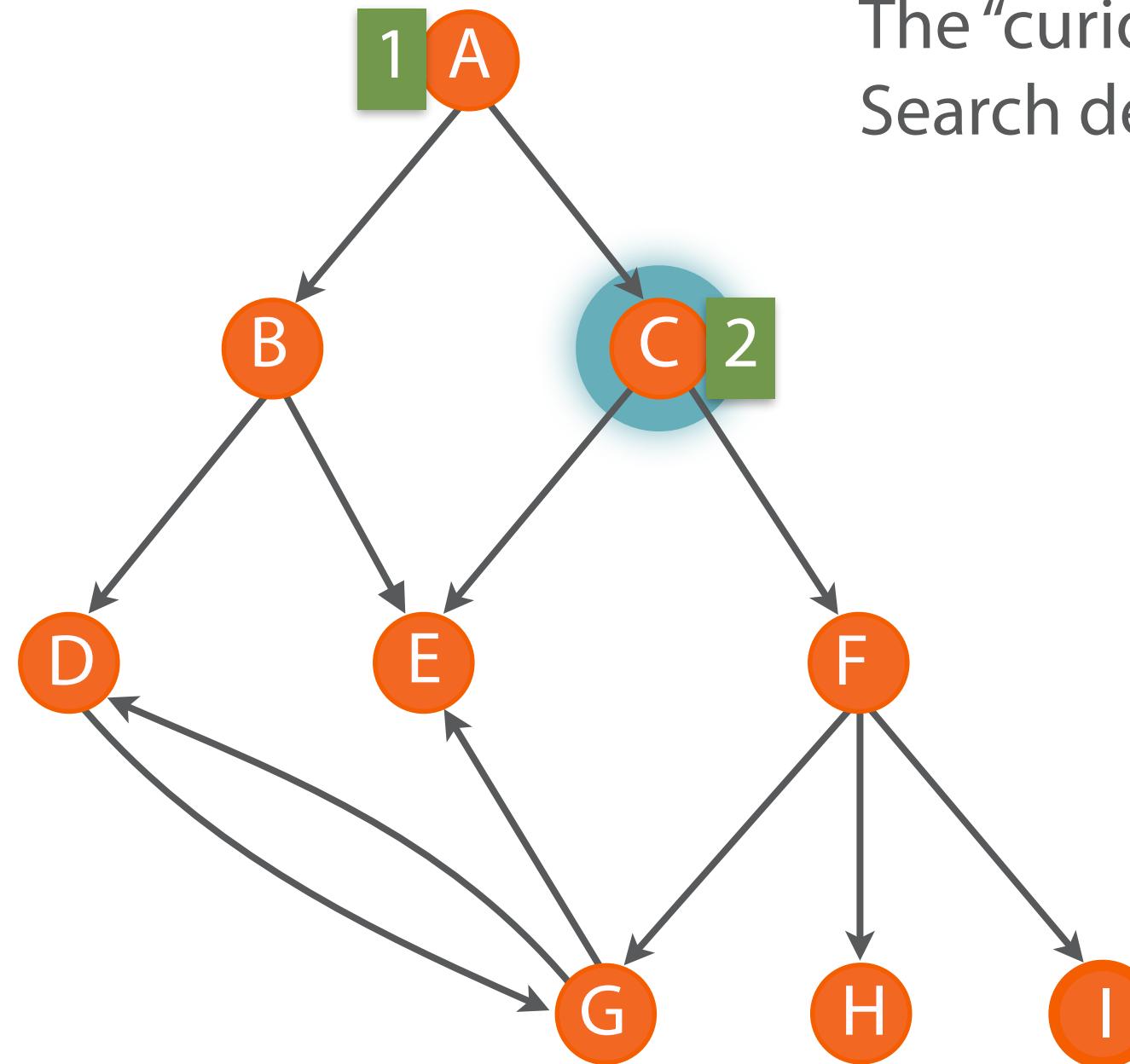
The “curious” strategy:  
Search deeper nodes first.



# Depth First Search (DFS)

Typical default graph traversal strategy.

Easy recursive implementation.



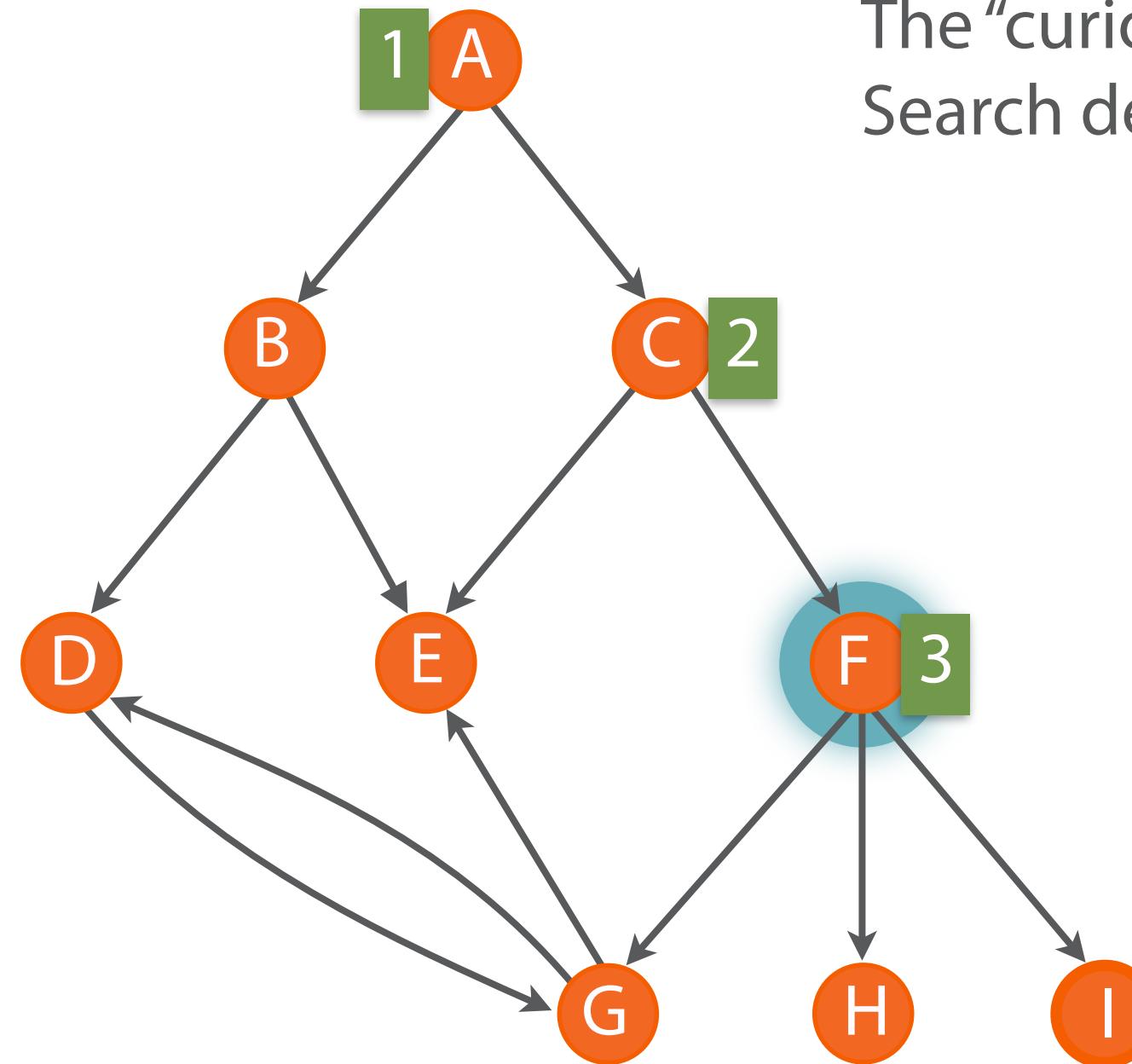
The “curious” strategy:  
Search deeper nodes first.



# Depth First Search (DFS)

Typical default graph traversal strategy.

Easy recursive implementation.



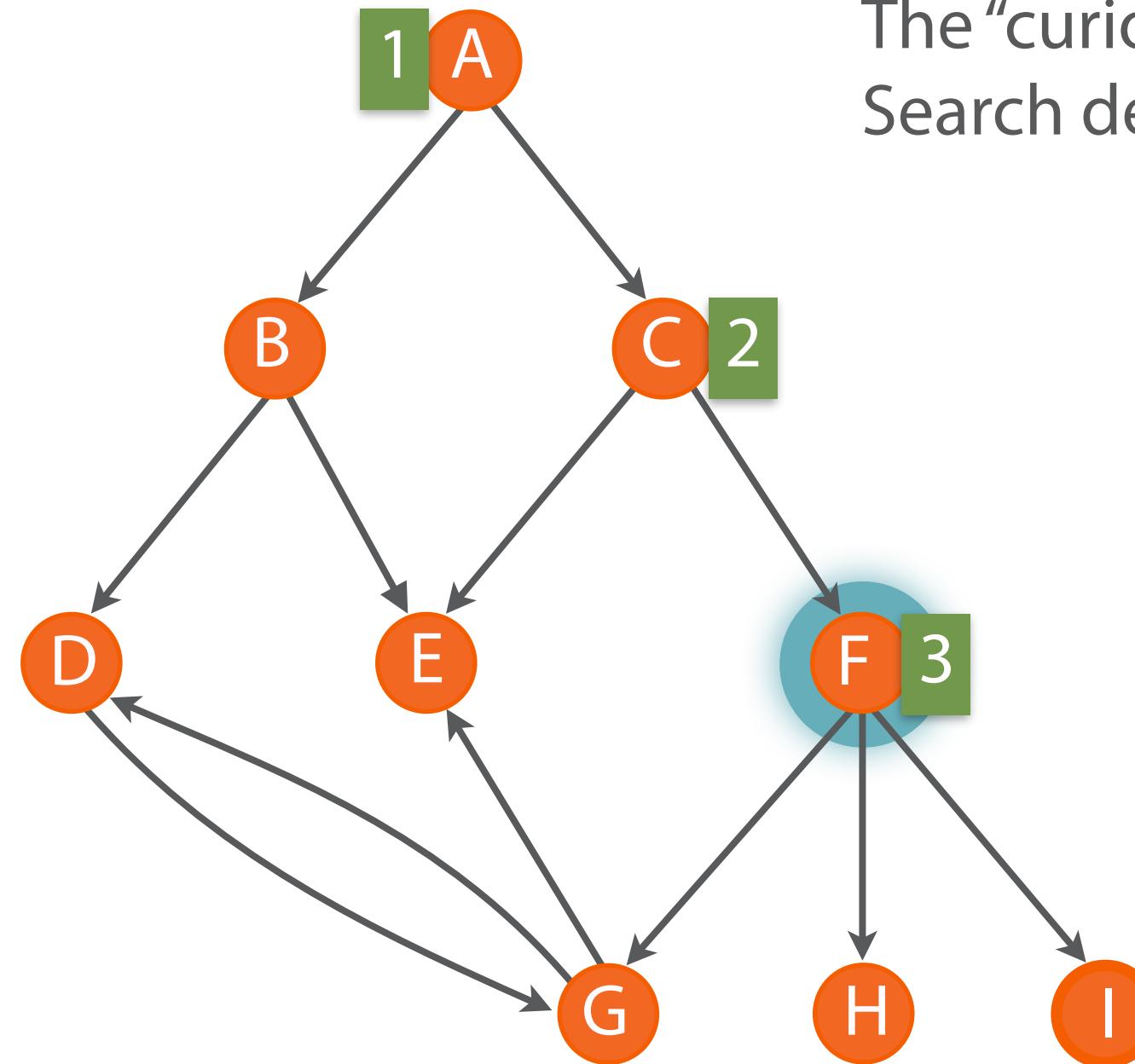
The “curious” strategy:  
Search deeper nodes first.



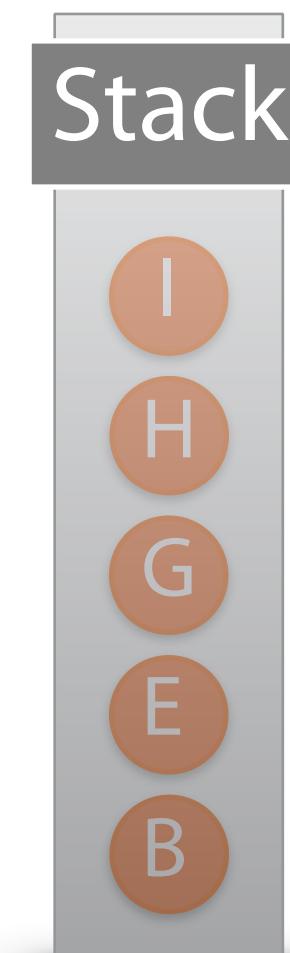
# Depth First Search (DFS)

Typical default graph traversal strategy.

Easy recursive implementation.



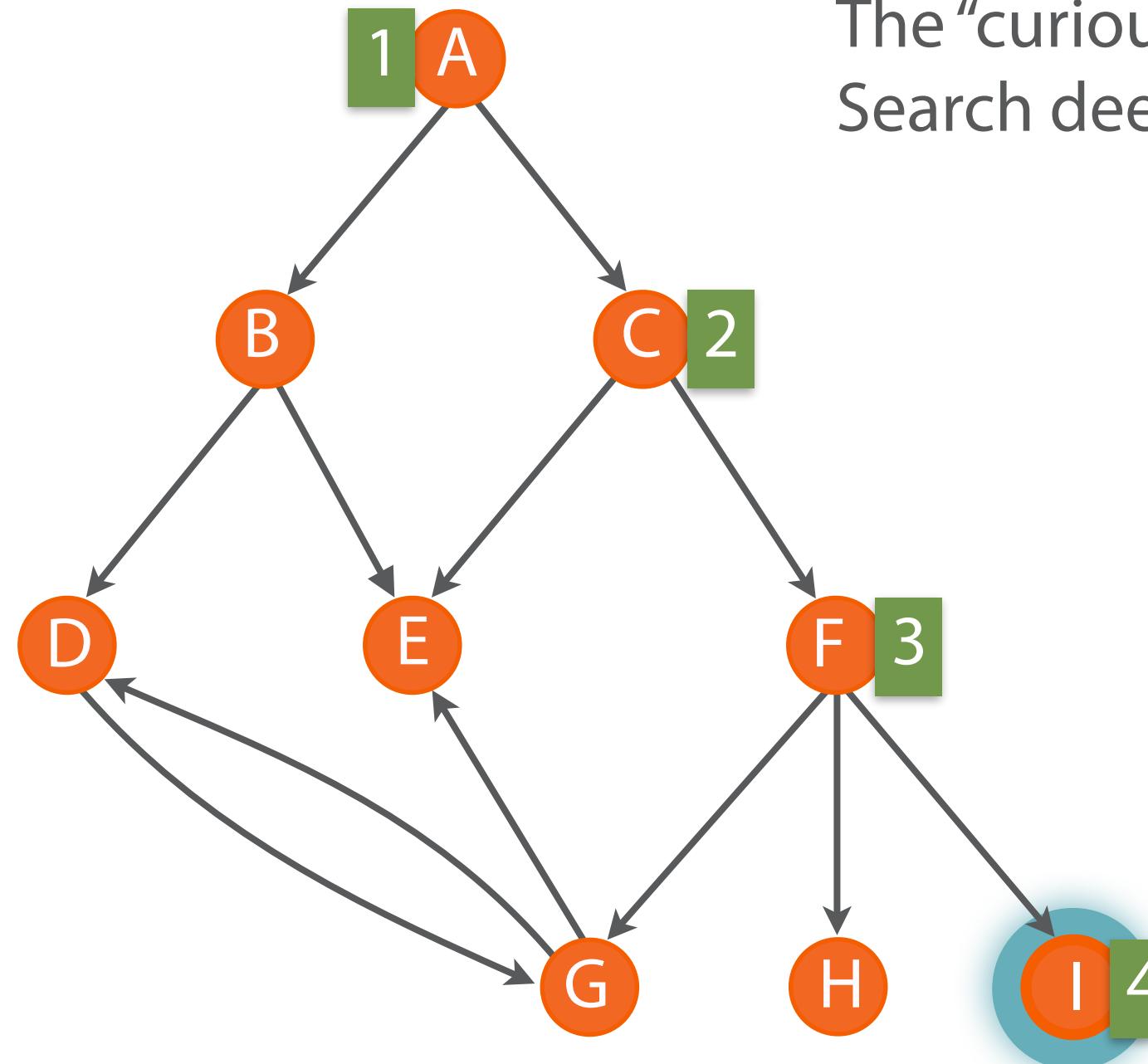
The “curious” strategy:  
Search deeper nodes first.



# Depth First Search (DFS)

Typical default graph traversal strategy.

Easy recursive implementation.



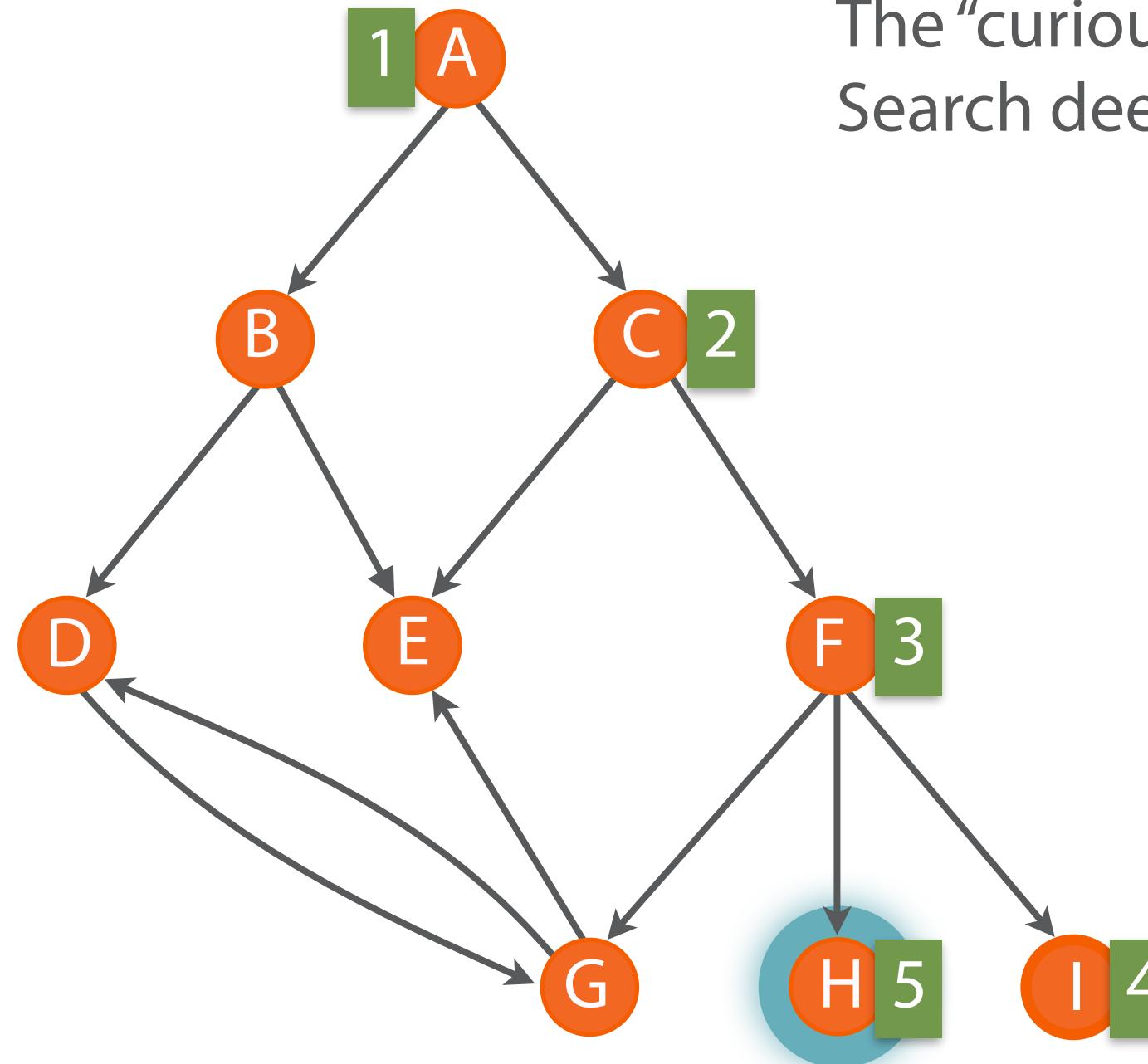
The “curious” strategy:  
Search deeper nodes first.



# Depth First Search (DFS)

Typical default graph traversal strategy.

Easy recursive implementation.



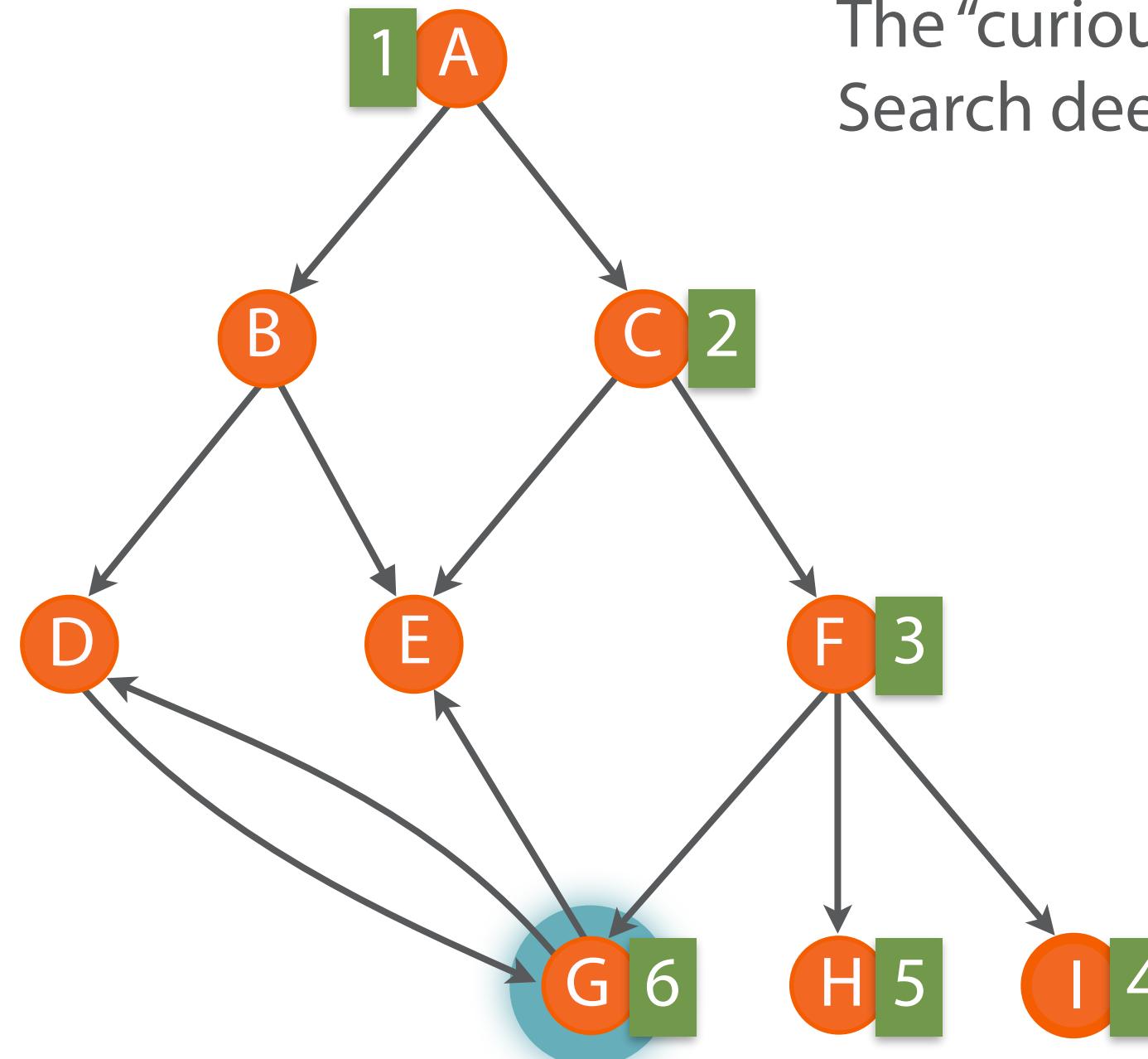
The “curious” strategy:  
Search deeper nodes first.



# Depth First Search (DFS)

Typical default graph traversal strategy.

Easy recursive implementation.



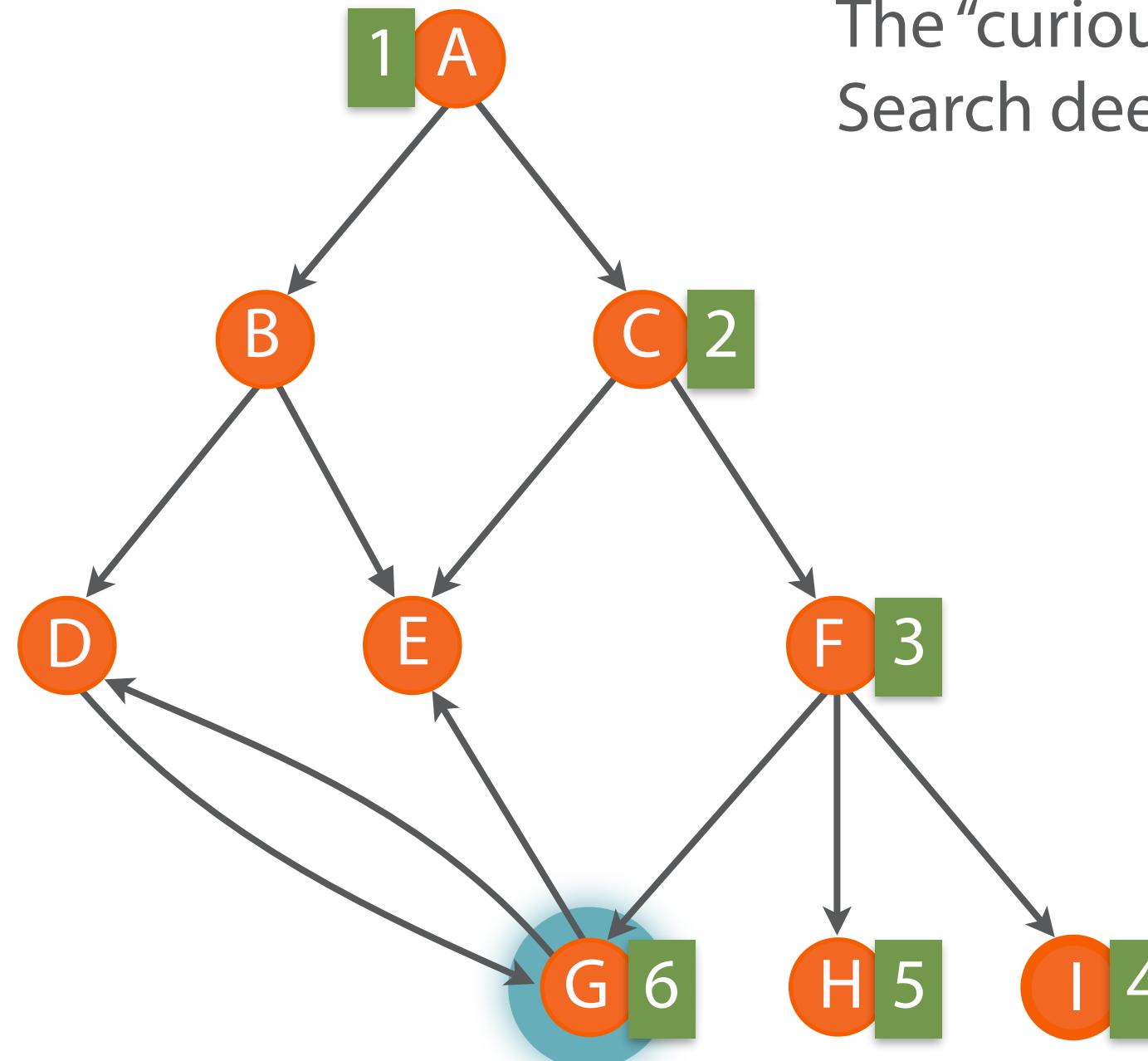
The “curious” strategy:  
Search deeper nodes first.



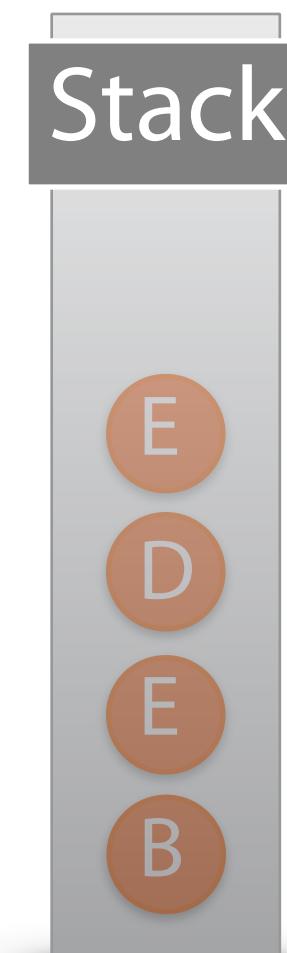
# Depth First Search (DFS)

Typical default graph traversal strategy.

Easy recursive implementation.



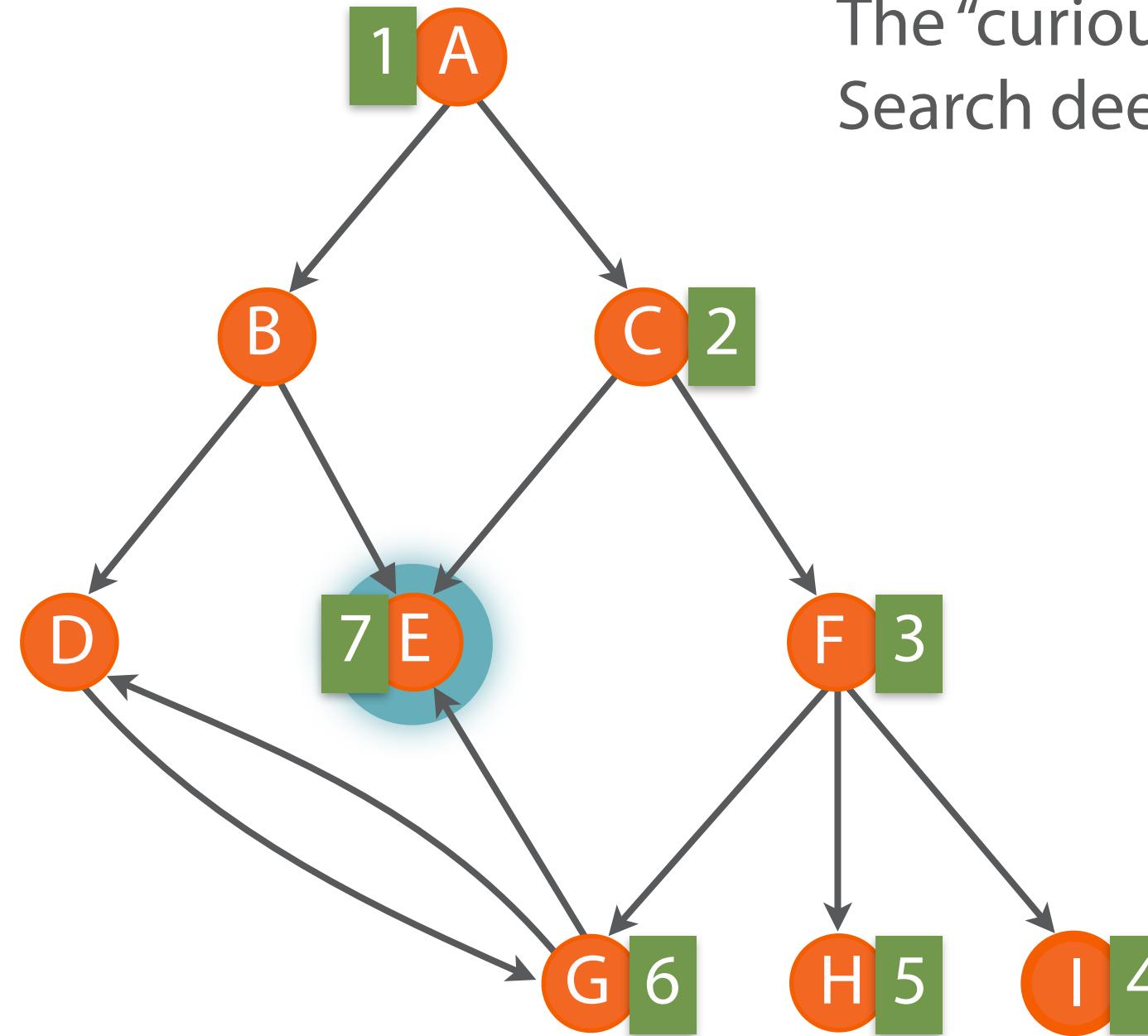
The “curious” strategy:  
Search deeper nodes first.



# Depth First Search (DFS)

Typical default graph traversal strategy.

Easy recursive implementation.



The “curious” strategy:  
Search deeper nodes first.

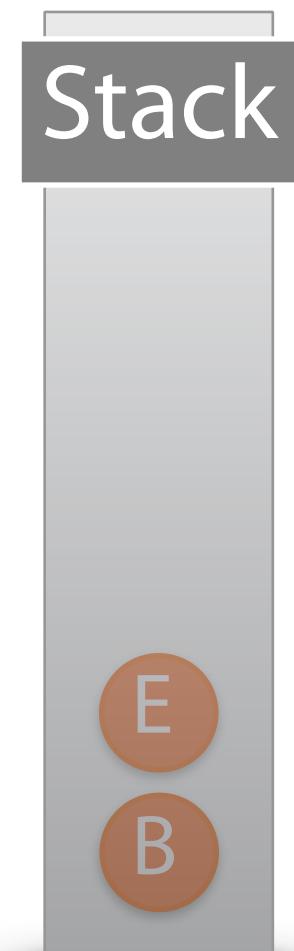
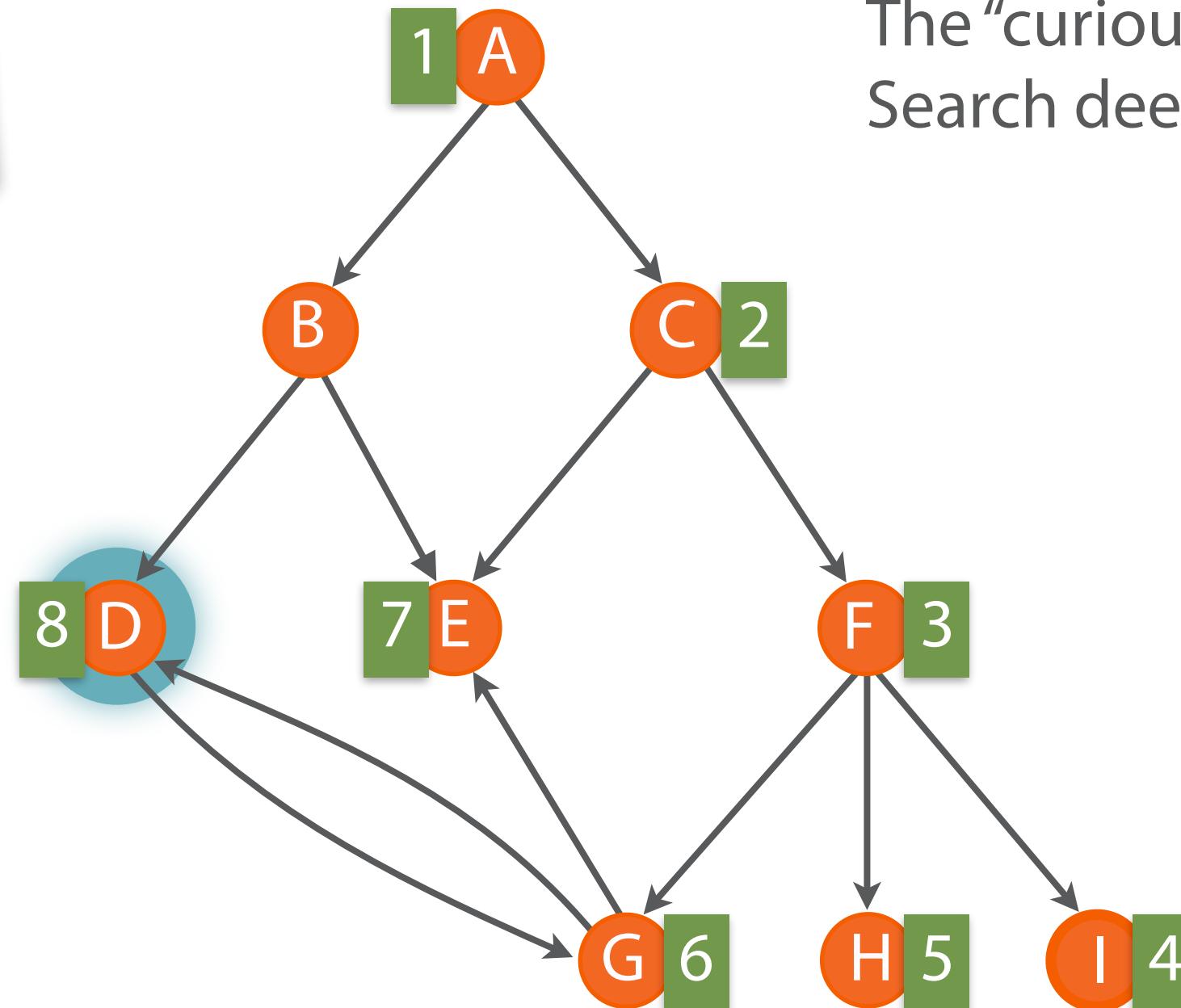


# Depth First Search (DFS)

Typical default graph traversal strategy.

Easy recursive implementation.

The “curious” strategy:  
Search deeper nodes first.

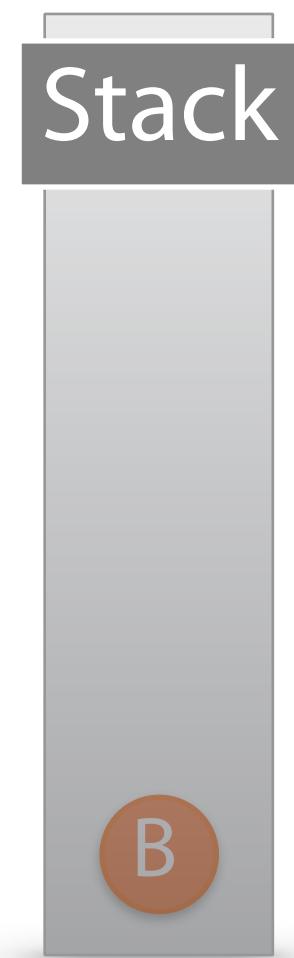
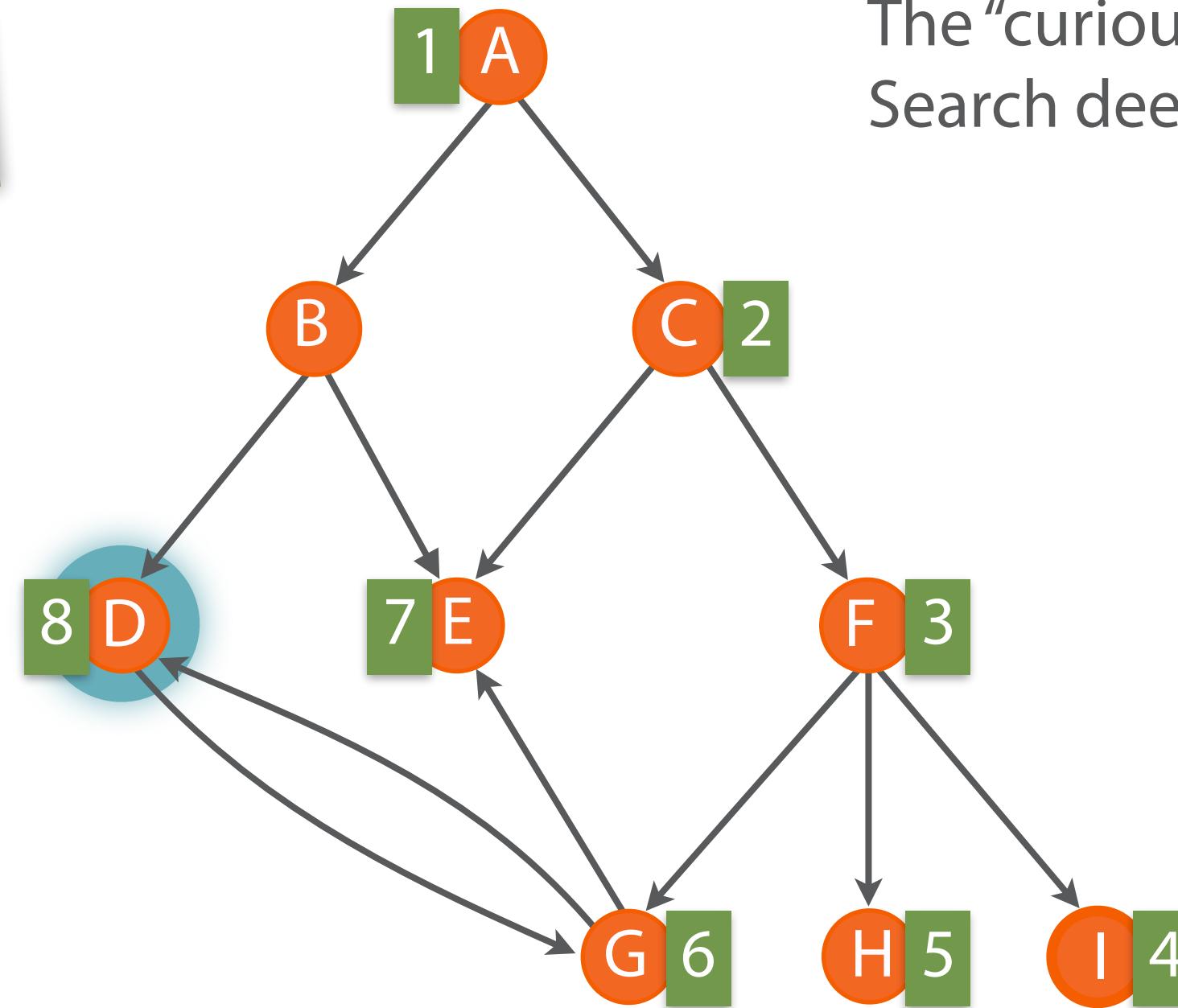


# Depth First Search (DFS)

Typical default graph traversal strategy.

Easy recursive implementation.

The “curious” strategy:  
Search deeper nodes first.

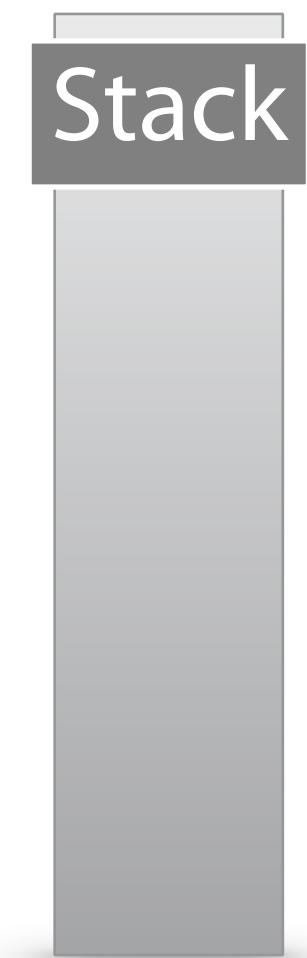
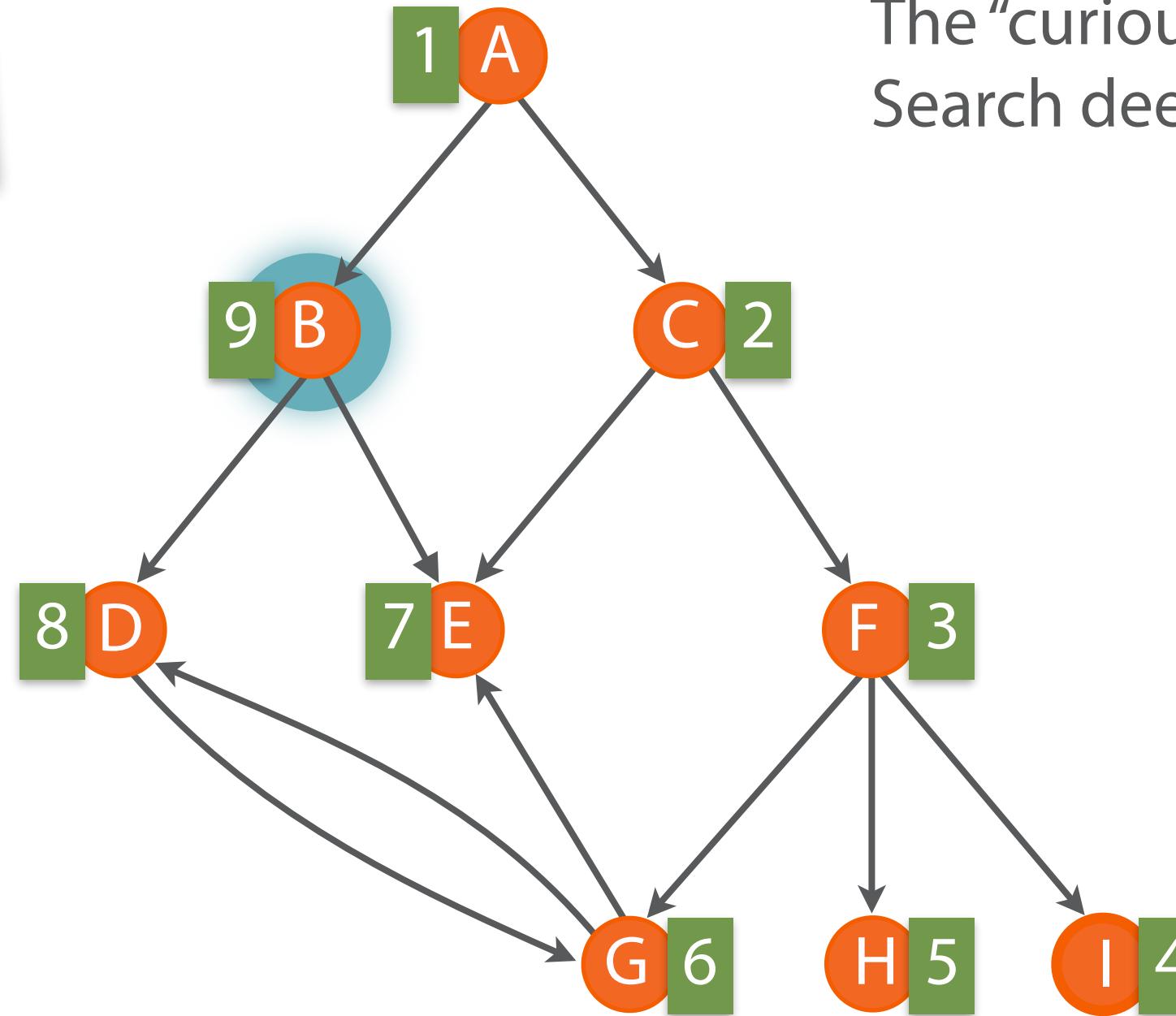


# Depth First Search (DFS)

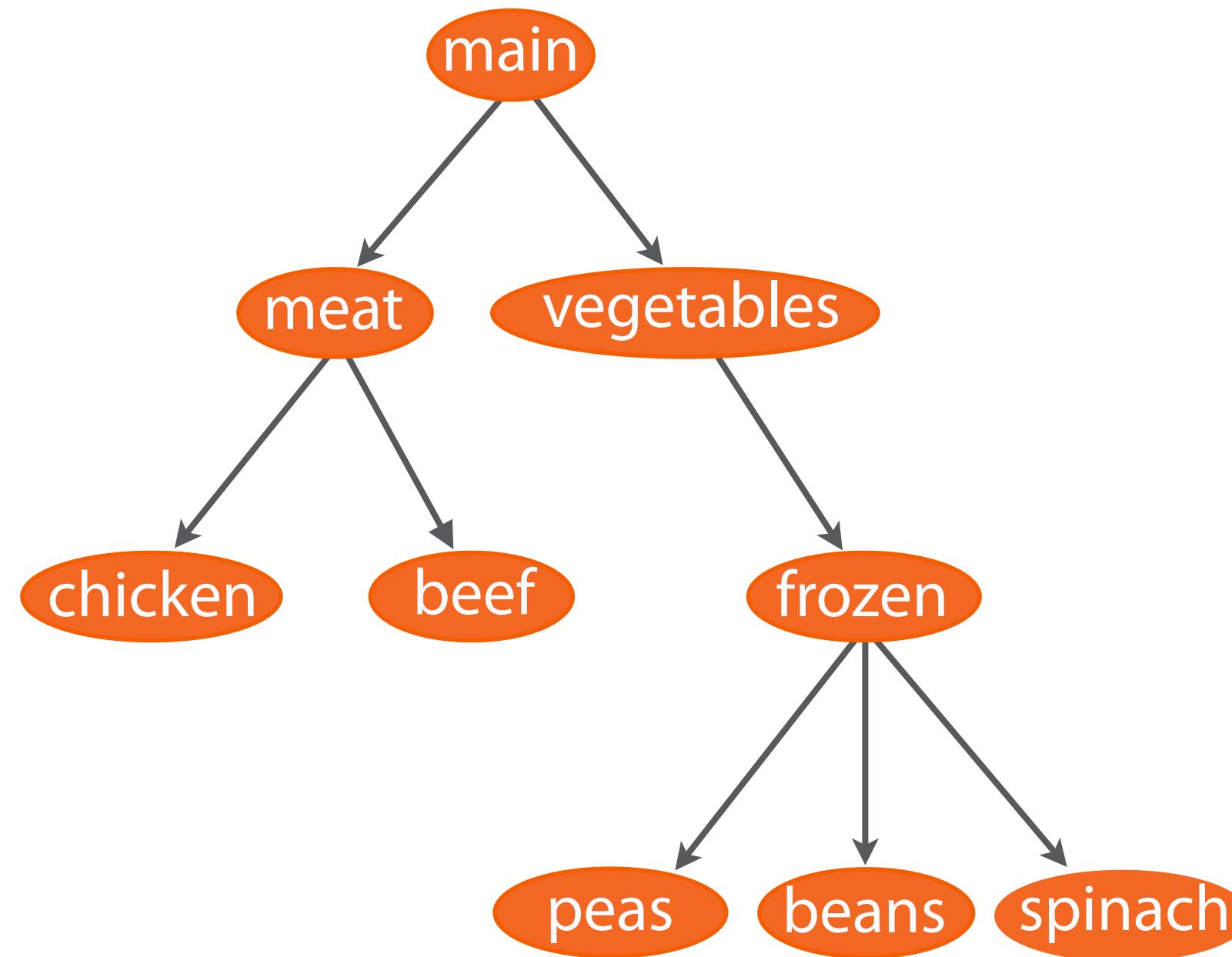
Typical default graph traversal strategy.

Easy recursive implementation.

The “curious” strategy:  
Search deeper nodes first.

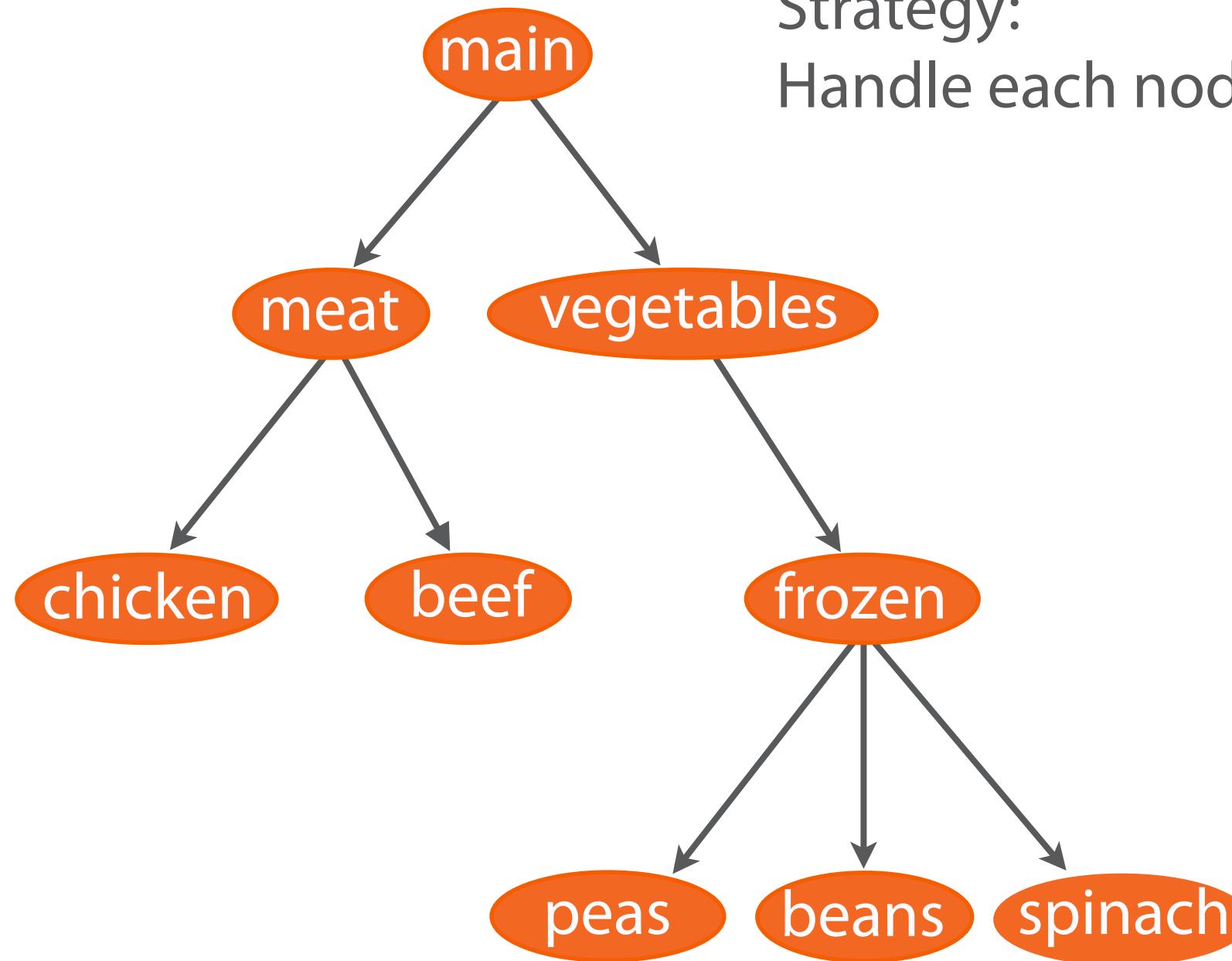


# DFS: Preorder Traversal



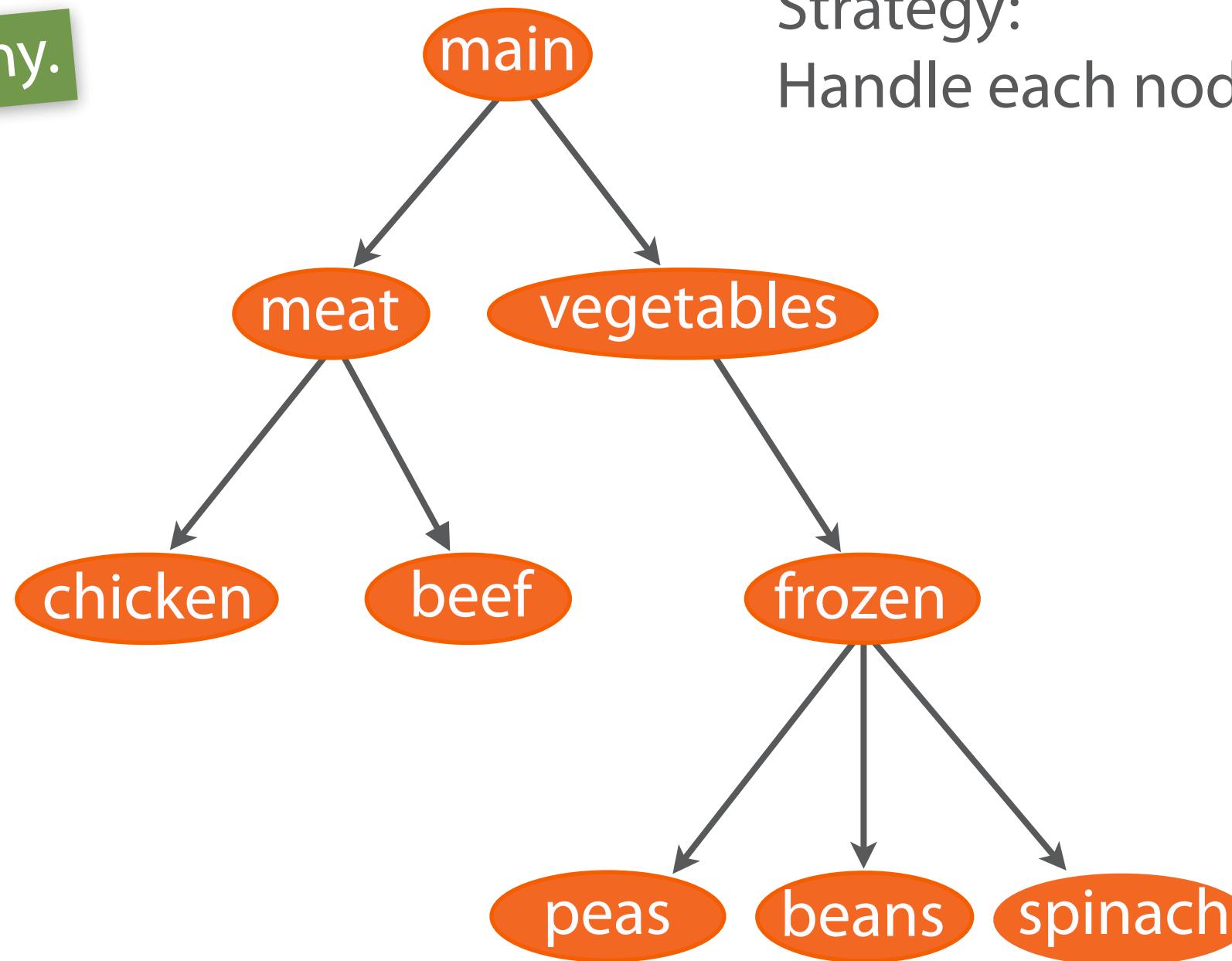
# DFS: Preorder Traversal

Strategy:  
Handle each node *before* its children.



# DFS: Preorder Traversal

For printing a hierarchy.



Strategy:  
Handle each node *before* its children.

# DFS: Preorder Traversal

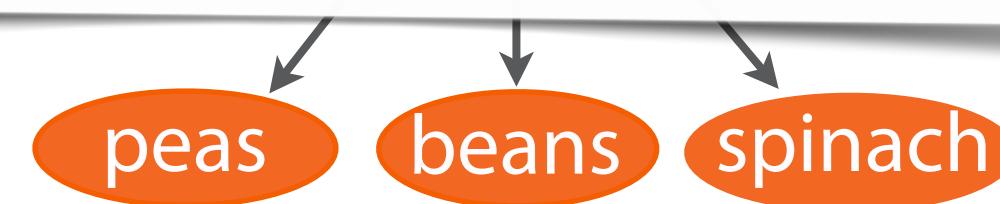
For printing a hierarchy.



Strategy:  
Handle each node *before* its children.

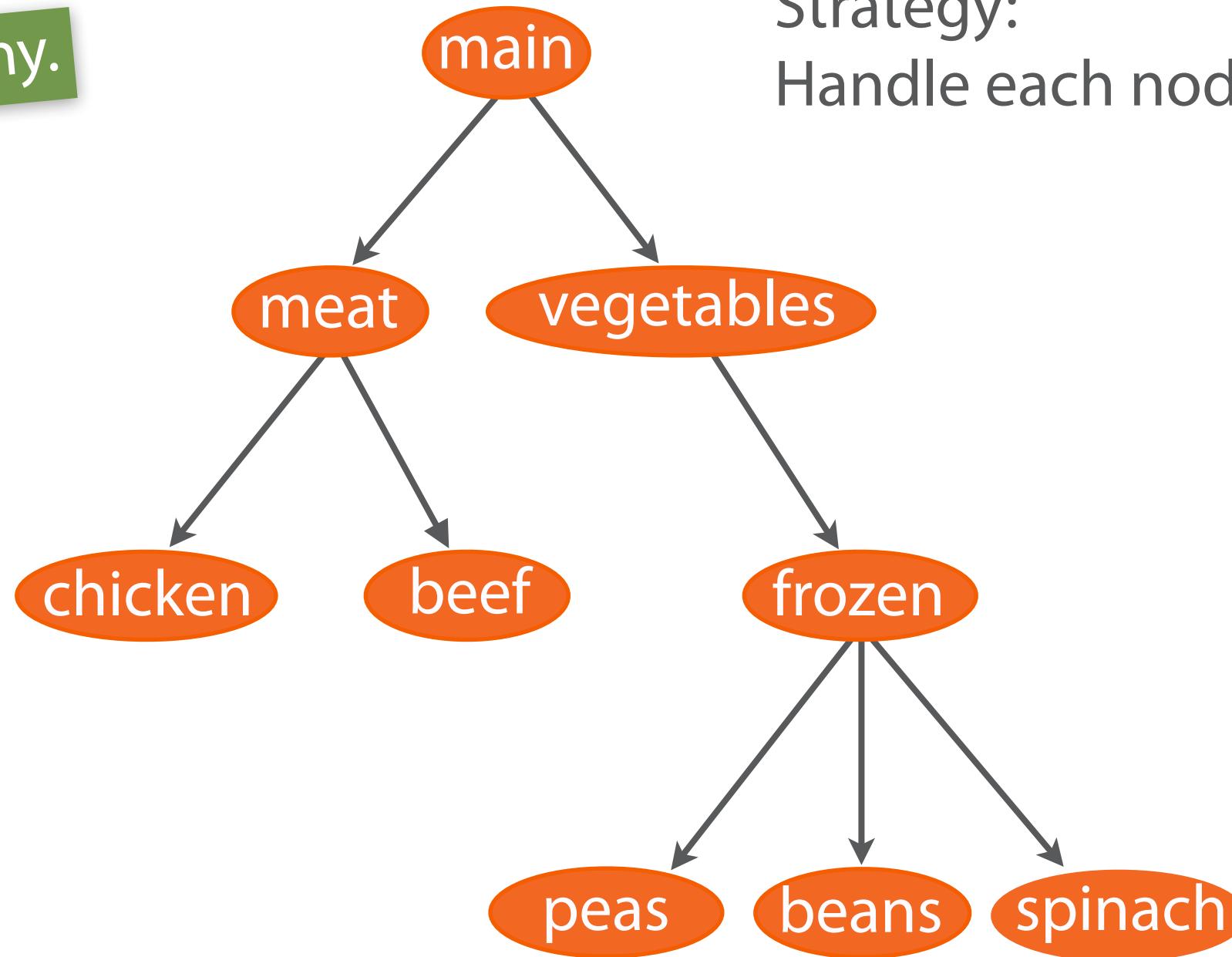
```
public void Preorder(Node<T> node, int level = 0)
{
    var indent = new String(' ', 3 * level);
    Console.WriteLine(indent + node.Value);

    foreach (var child in node.Children)
        Preorder(child, level + 1);
}
```



# DFS: Preorder Traversal

For printing a hierarchy.

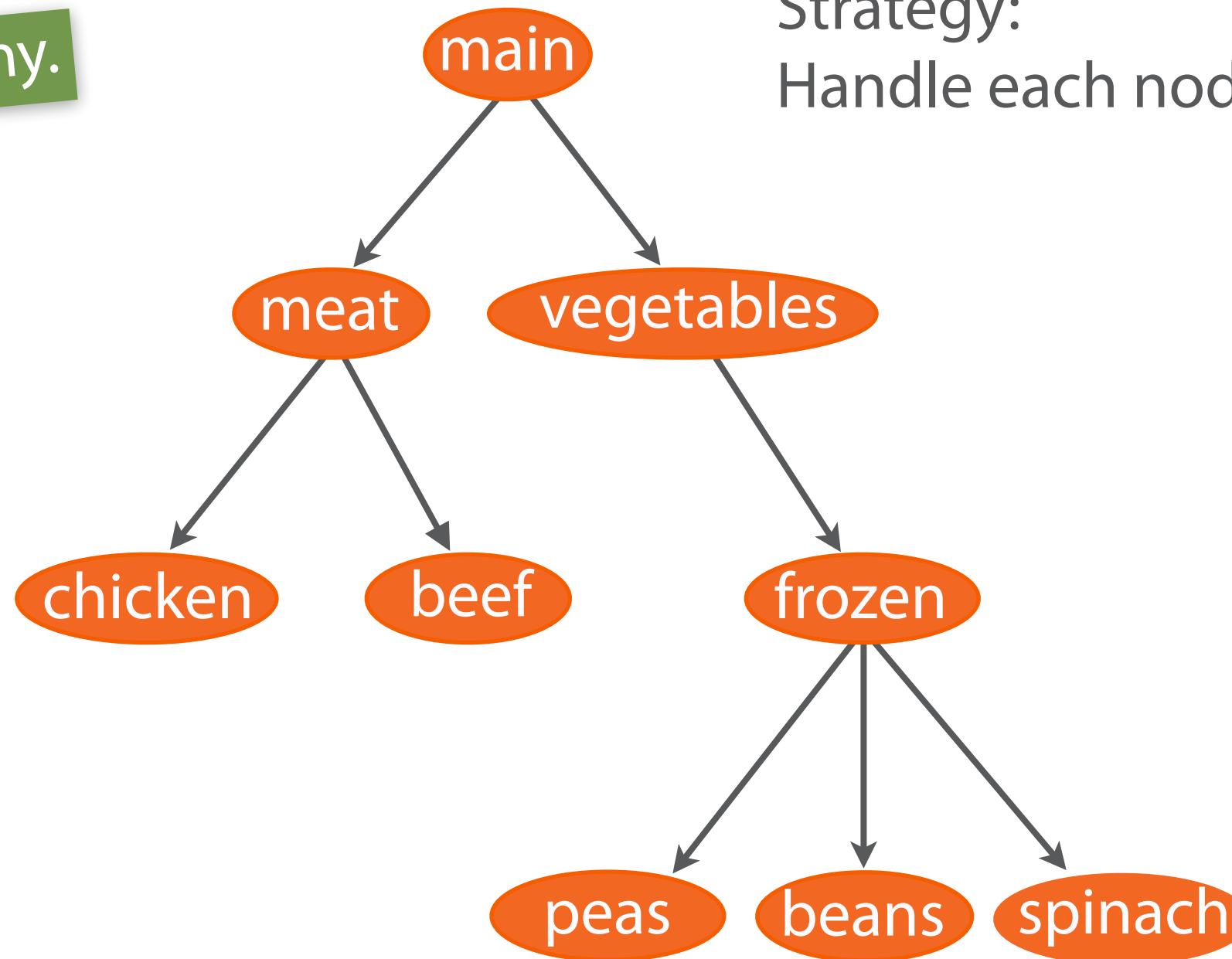


Strategy:  
Handle each node *before* its children.

# DFS: Preorder Traversal

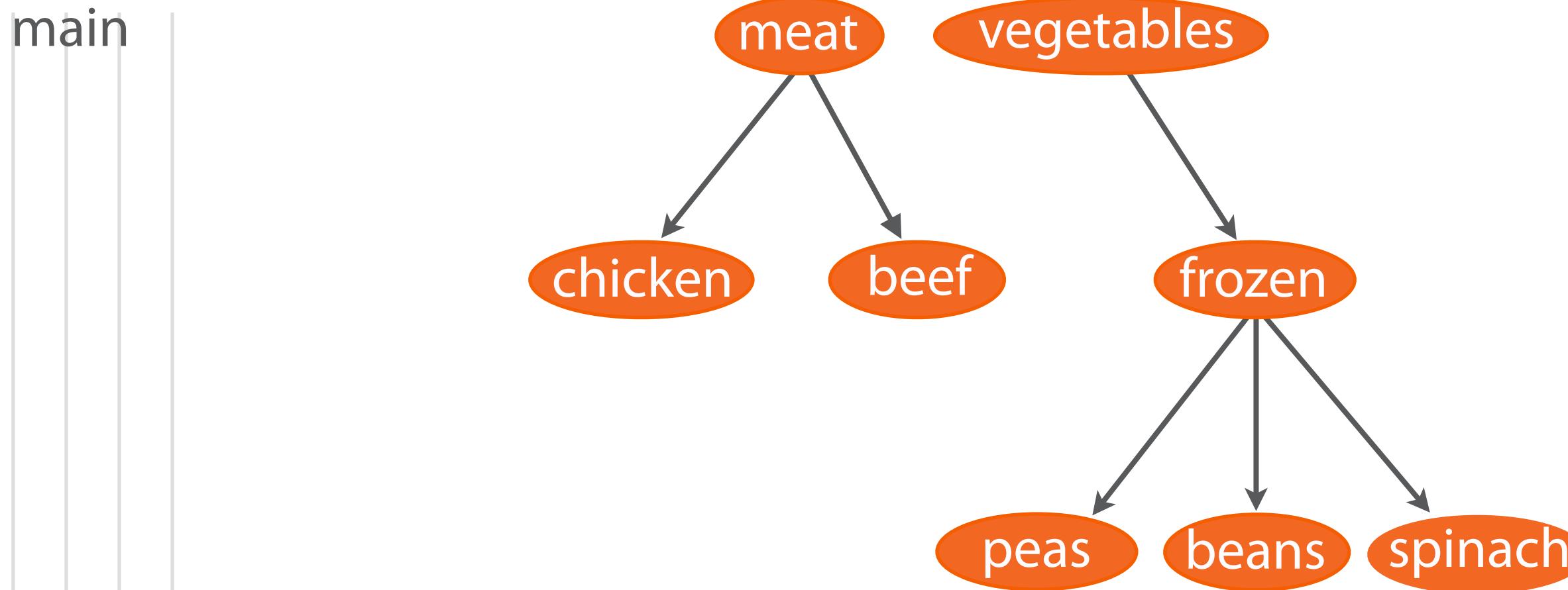
For printing a hierarchy.

Strategy:  
Handle each node *before* its children.



# DFS: Preorder Traversal

For printing a hierarchy.



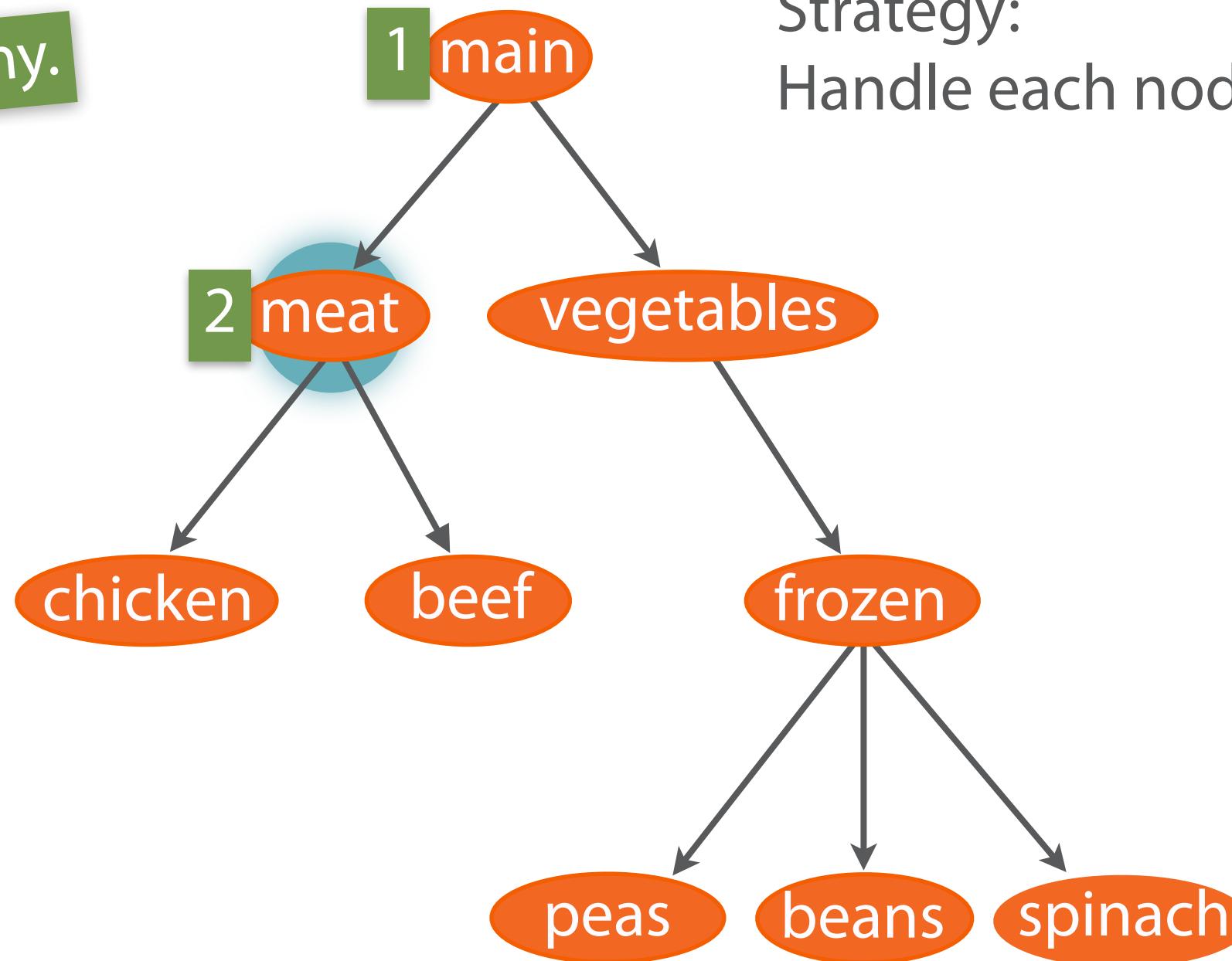
Strategy:  
Handle each node *before* its children.

# DFS: Preorder Traversal

For printing a hierarchy.

main  
meat

Strategy:  
Handle each node *before* its children.

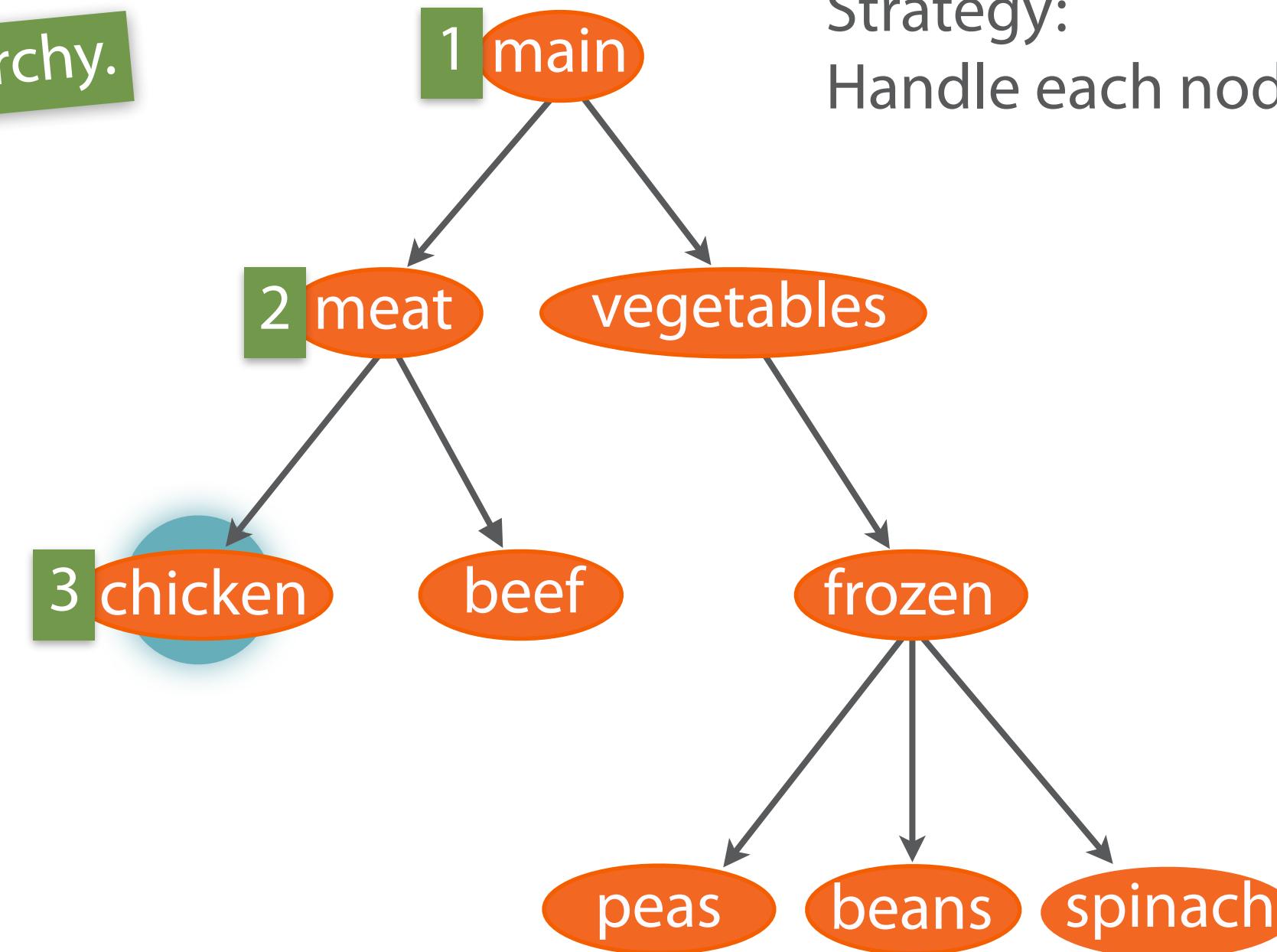


# DFS: Preorder Traversal

For printing a hierarchy.

main  
meat  
chicken

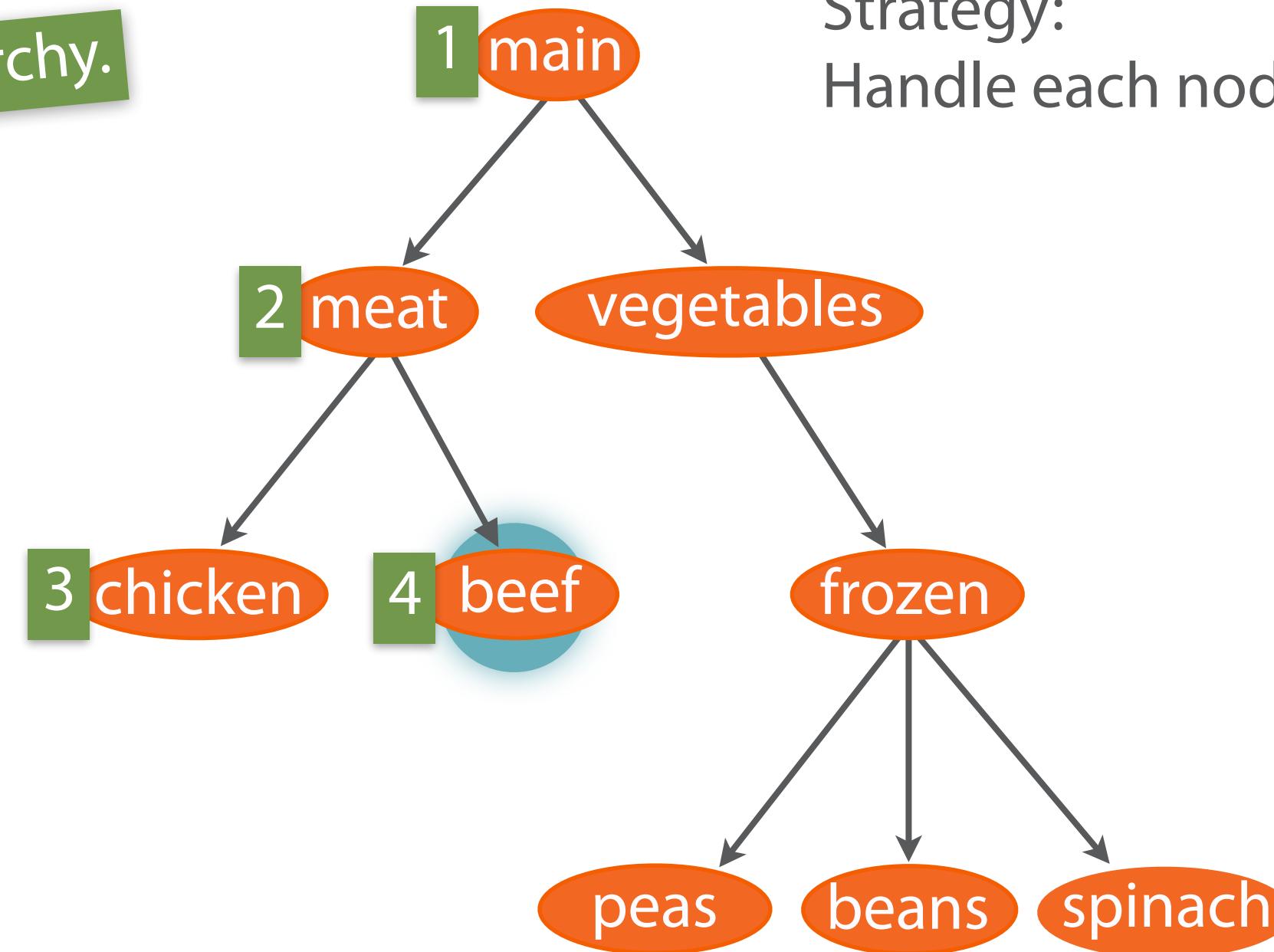
Strategy:  
Handle each node *before* its children.



# DFS: Preorder Traversal

For printing a hierarchy.

main  
meat  
chicken  
beef

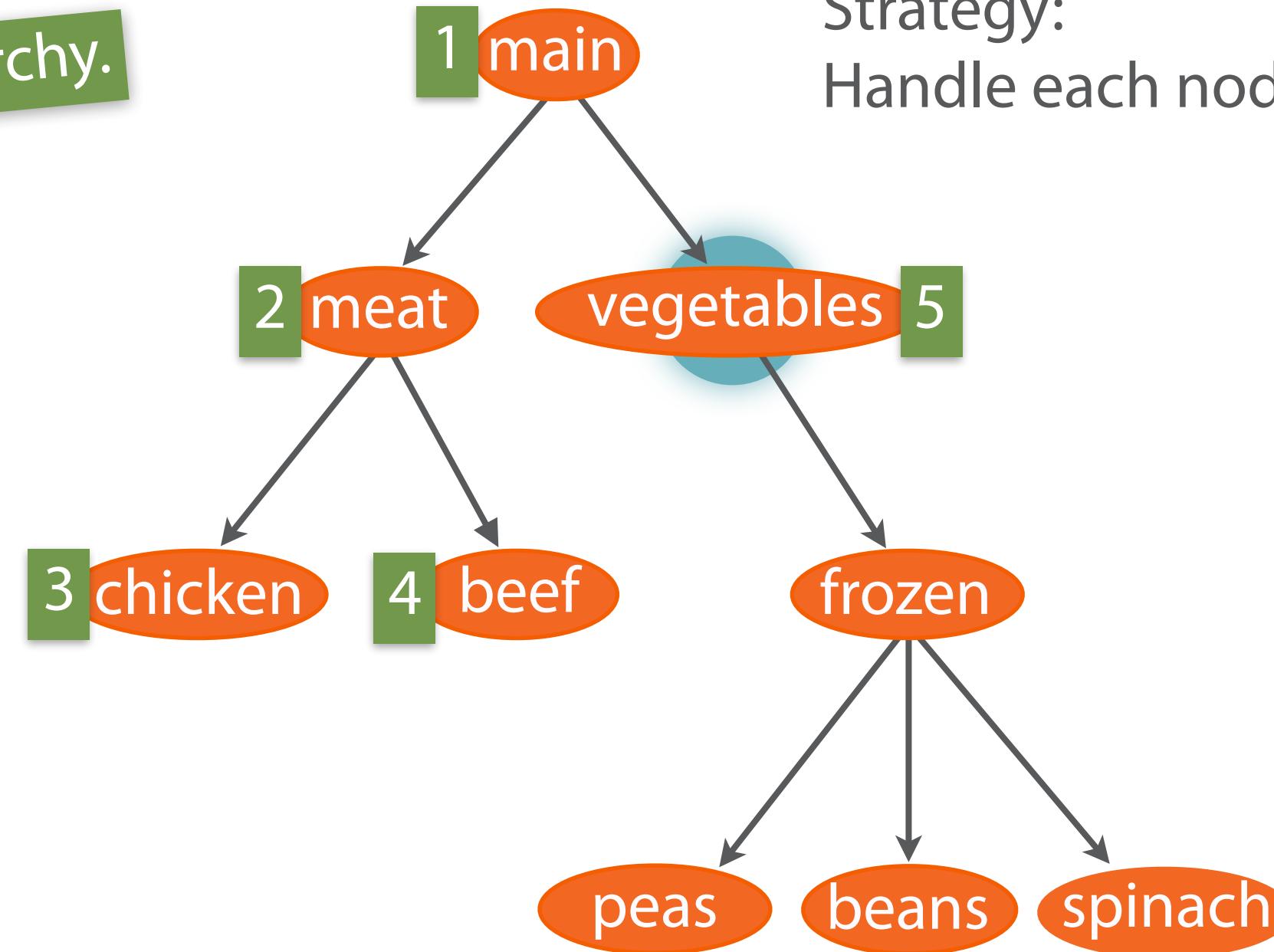


Strategy:  
Handle each node *before* its children.

# DFS: Preorder Traversal

For printing a hierarchy.

main  
meat  
chicken  
beef  
vegetables

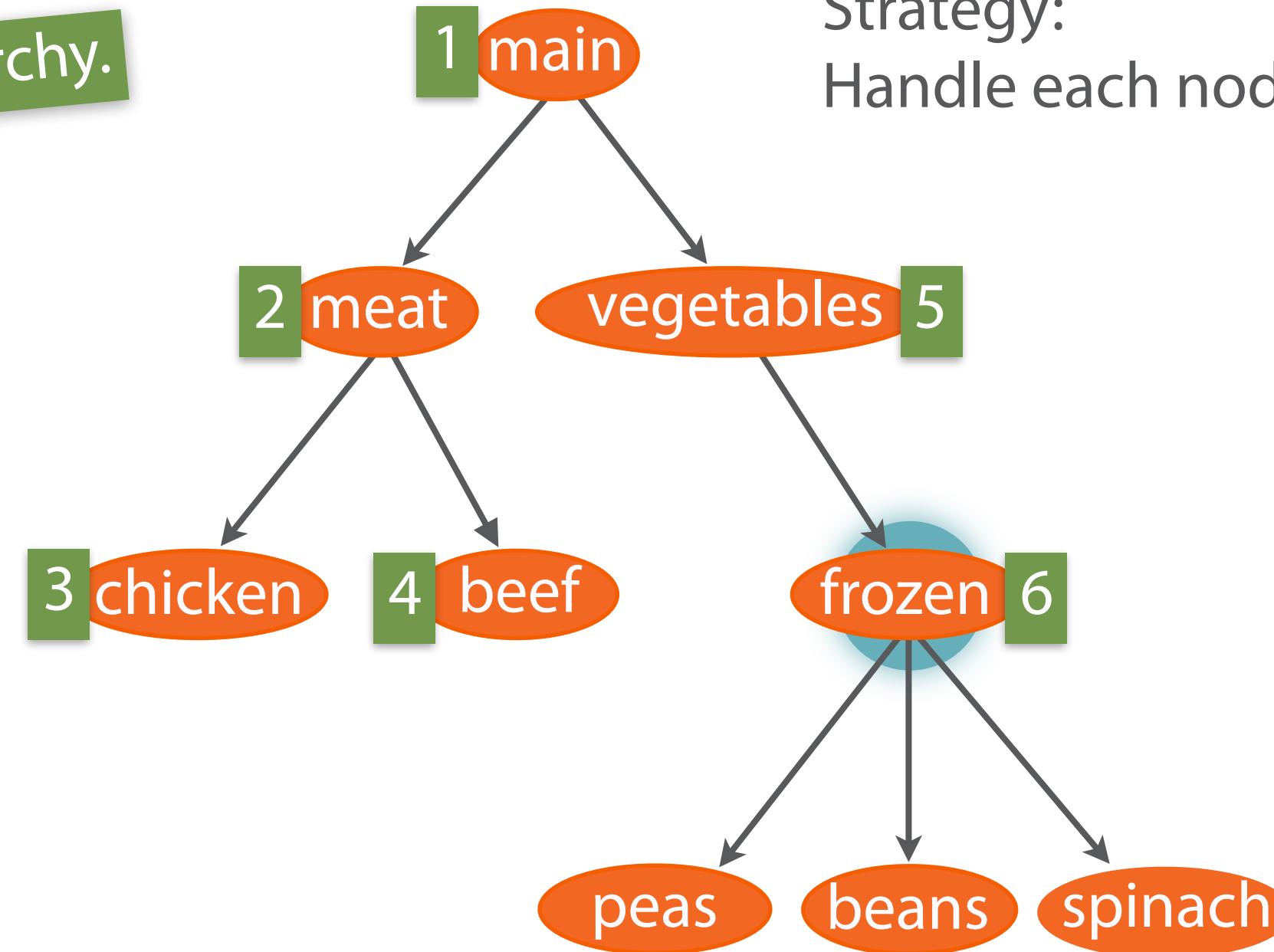


Strategy:  
Handle each node *before* its children.

# DFS: Preorder Traversal

For printing a hierarchy.

main  
meat  
chicken  
beef  
vegetables  
frozen

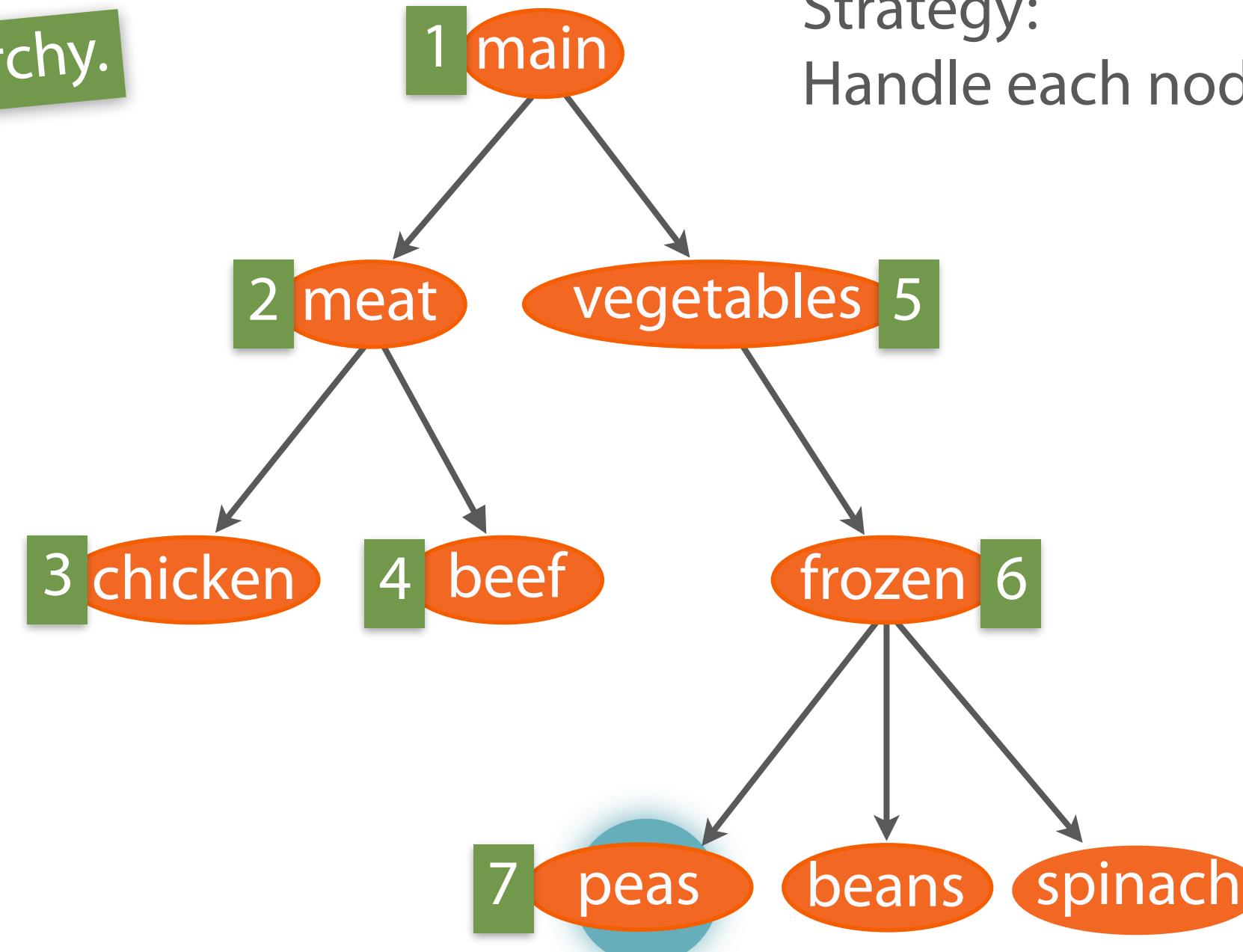


Strategy:  
Handle each node *before* its children.

# DFS: Preorder Traversal

For printing a hierarchy.

main  
meat  
chicken  
beef  
vegetables  
frozen  
peas

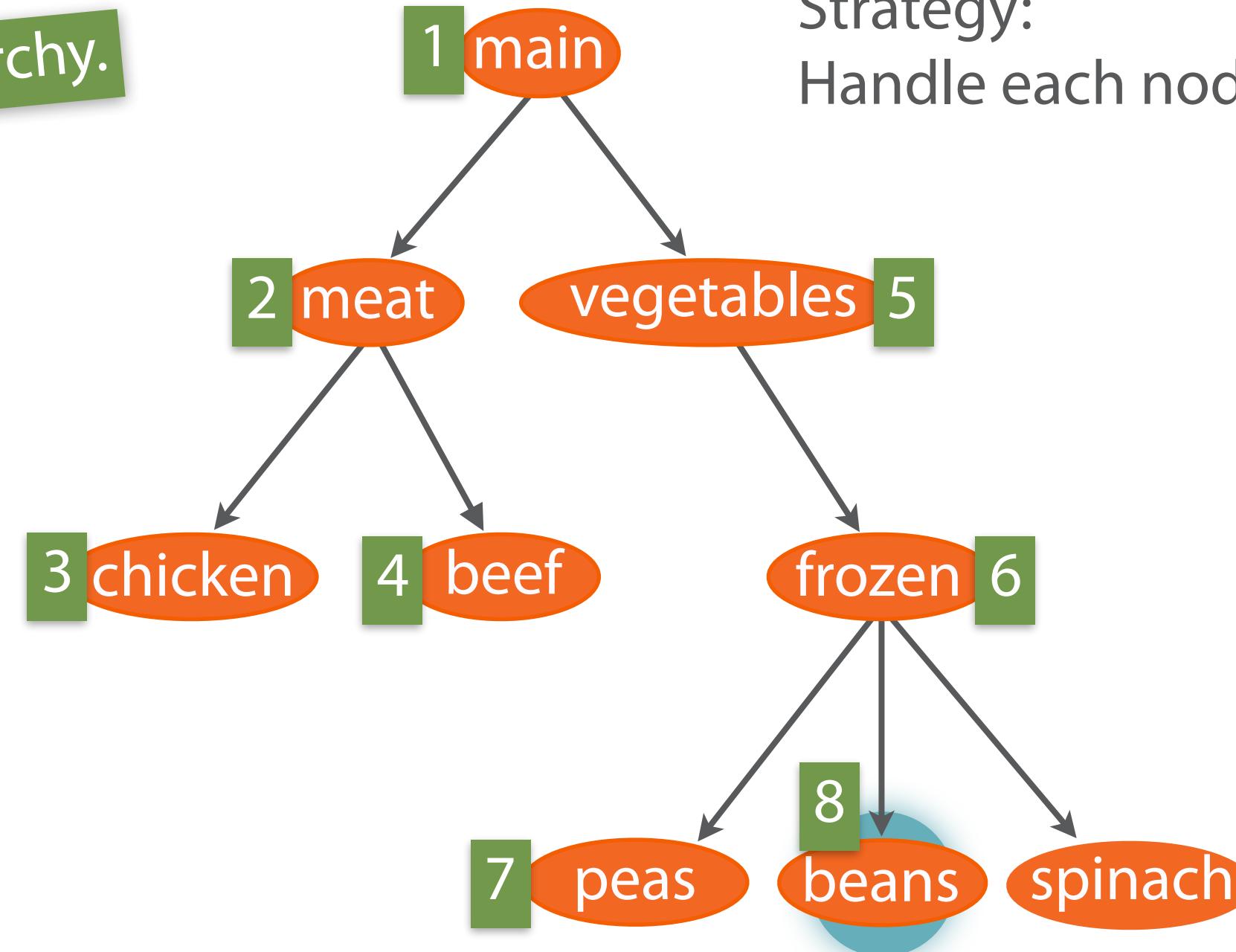


Strategy:  
Handle each node *before* its children.

# DFS: Preorder Traversal

For printing a hierarchy.

main  
meat  
chicken  
beef  
vegetables  
frozen  
peas  
beans

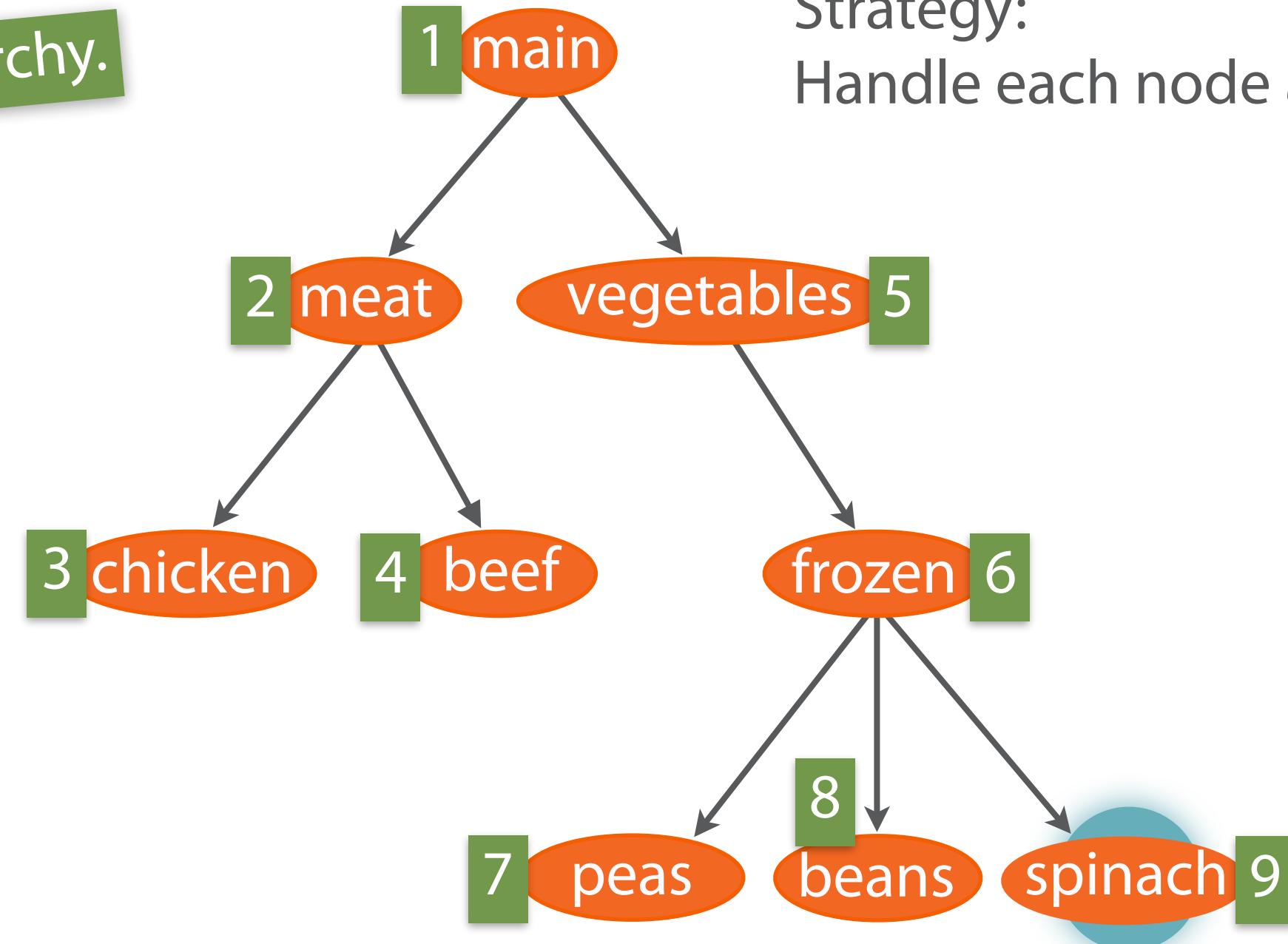


Strategy:  
Handle each node *before* its children.

# DFS: Preorder Traversal

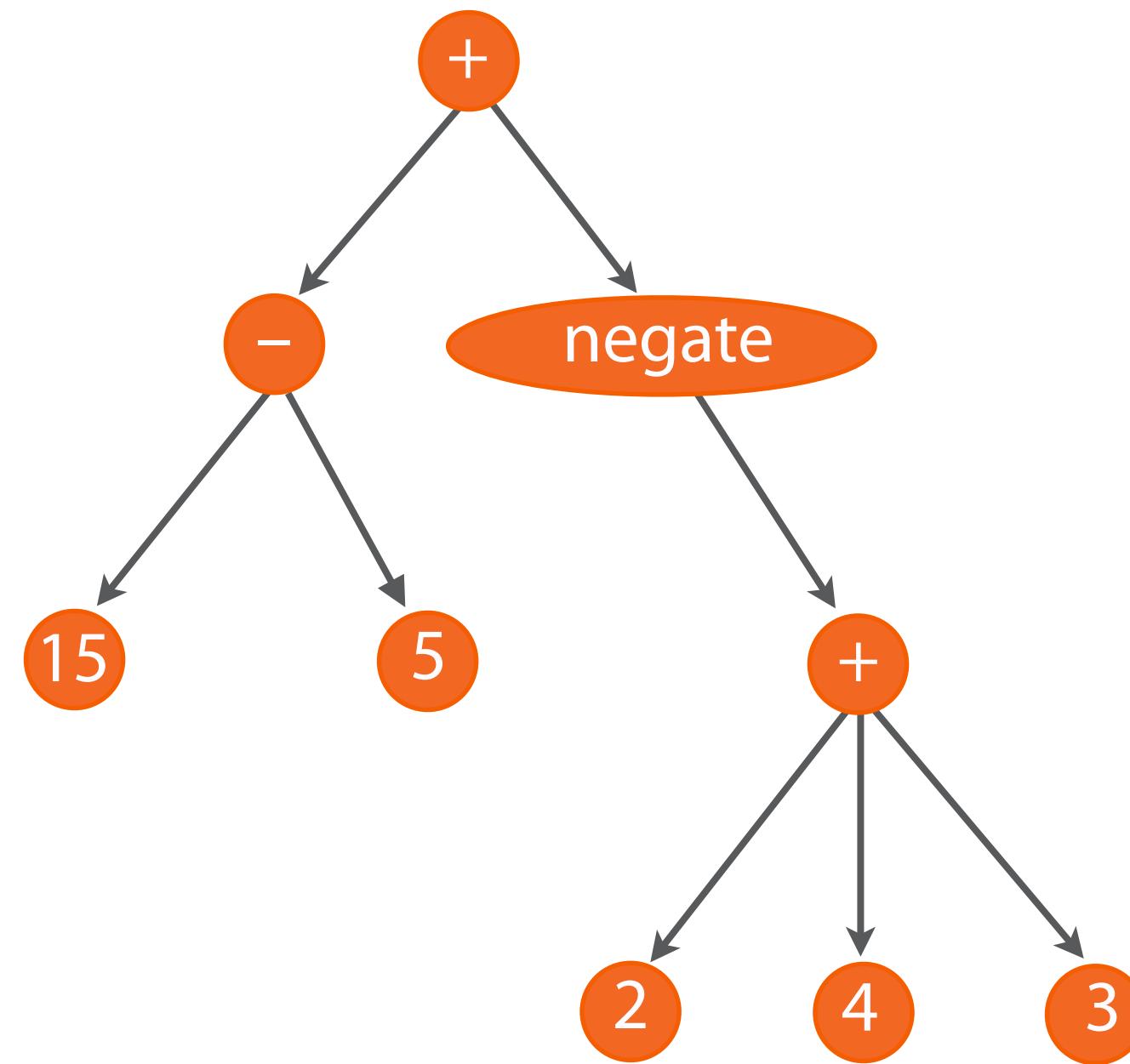
For printing a hierarchy.

main  
meat  
  chicken  
  beef  
vegetables  
frozen  
  peas  
  beans  
  spinach



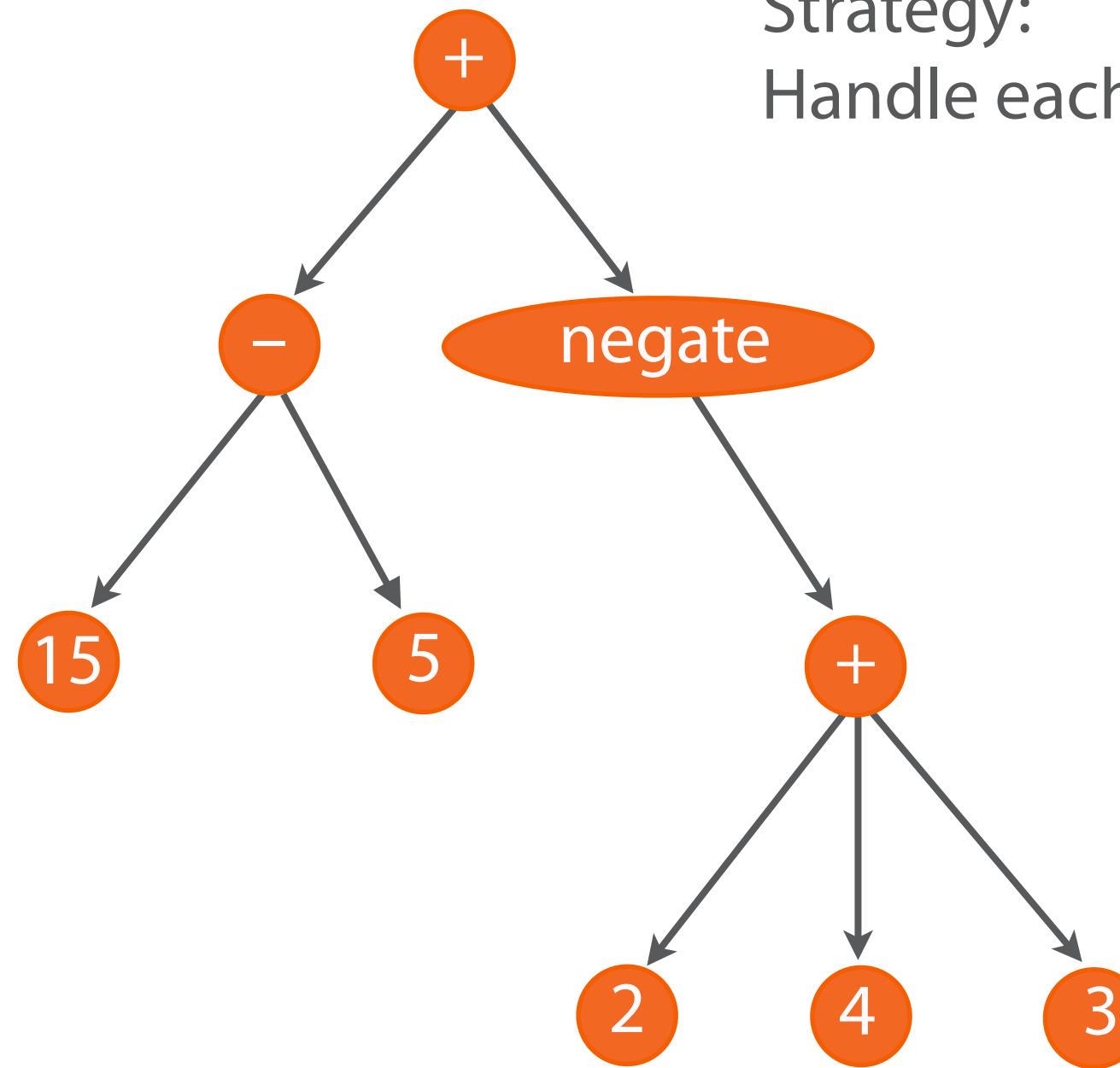
Strategy:  
Handle each node *before* its children.

# DFS: Postorder Traversal



# DFS: Postorder Traversal

Strategy:  
Handle each node *after* its children.



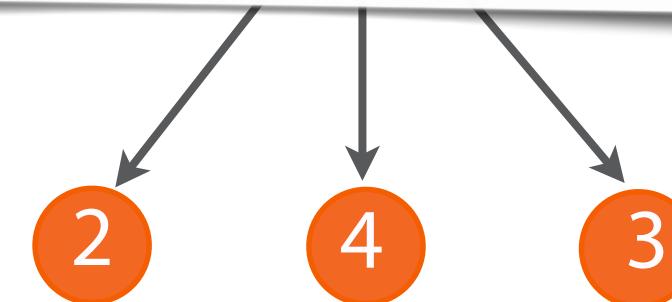
# DFS: Postorder Traversal



Strategy:  
Handle each node *after* its children.

```
public void Postorder(Node<T> node, int level = 0)
{
    foreach (var child in node.Children)
        Postorder(child, level + 1);

    var indent = new String(' ', 3 * level);
    Console.WriteLine(indent + node.Value);
}
```



# DFS: Postorder Traversal

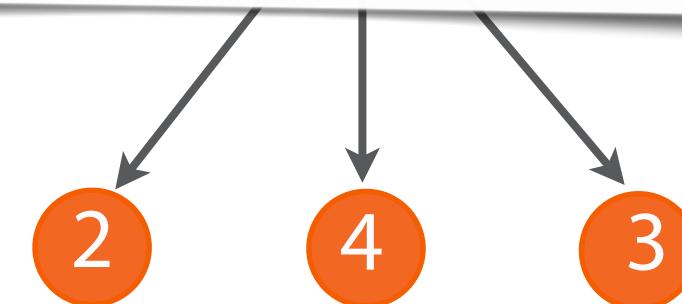
For executing a hierarchy.



Strategy:  
Handle each node *after* its children.

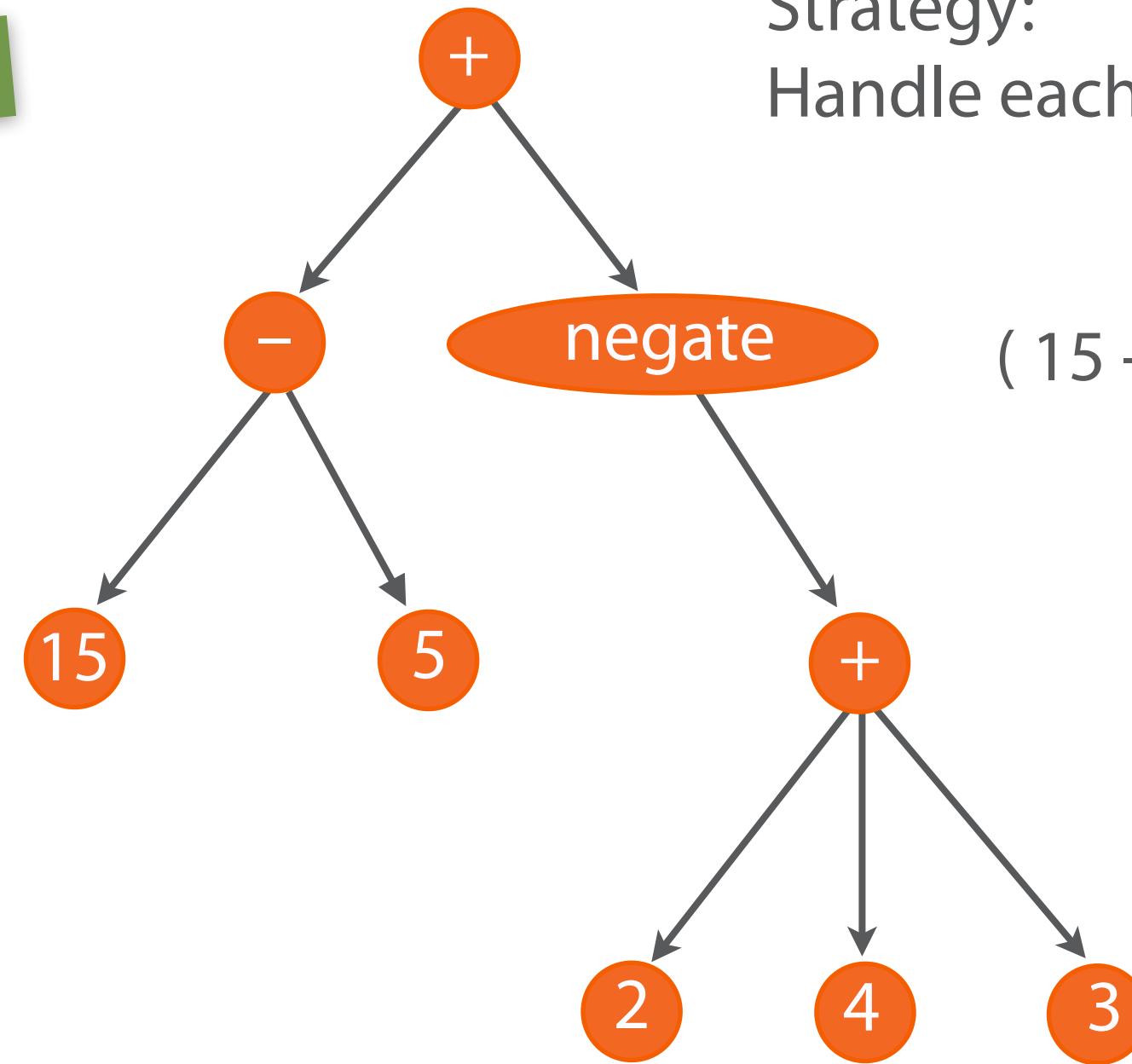
```
public void Postorder(Node<T> node, int level = 0)
{
    foreach (var child in node.Children)
        Postorder(child, level + 1);

    var indent = new String(' ', 3 * level);
    Console.WriteLine(indent + node.Value);
}
```



# DFS: Postorder Traversal

For executing a hierarchy.

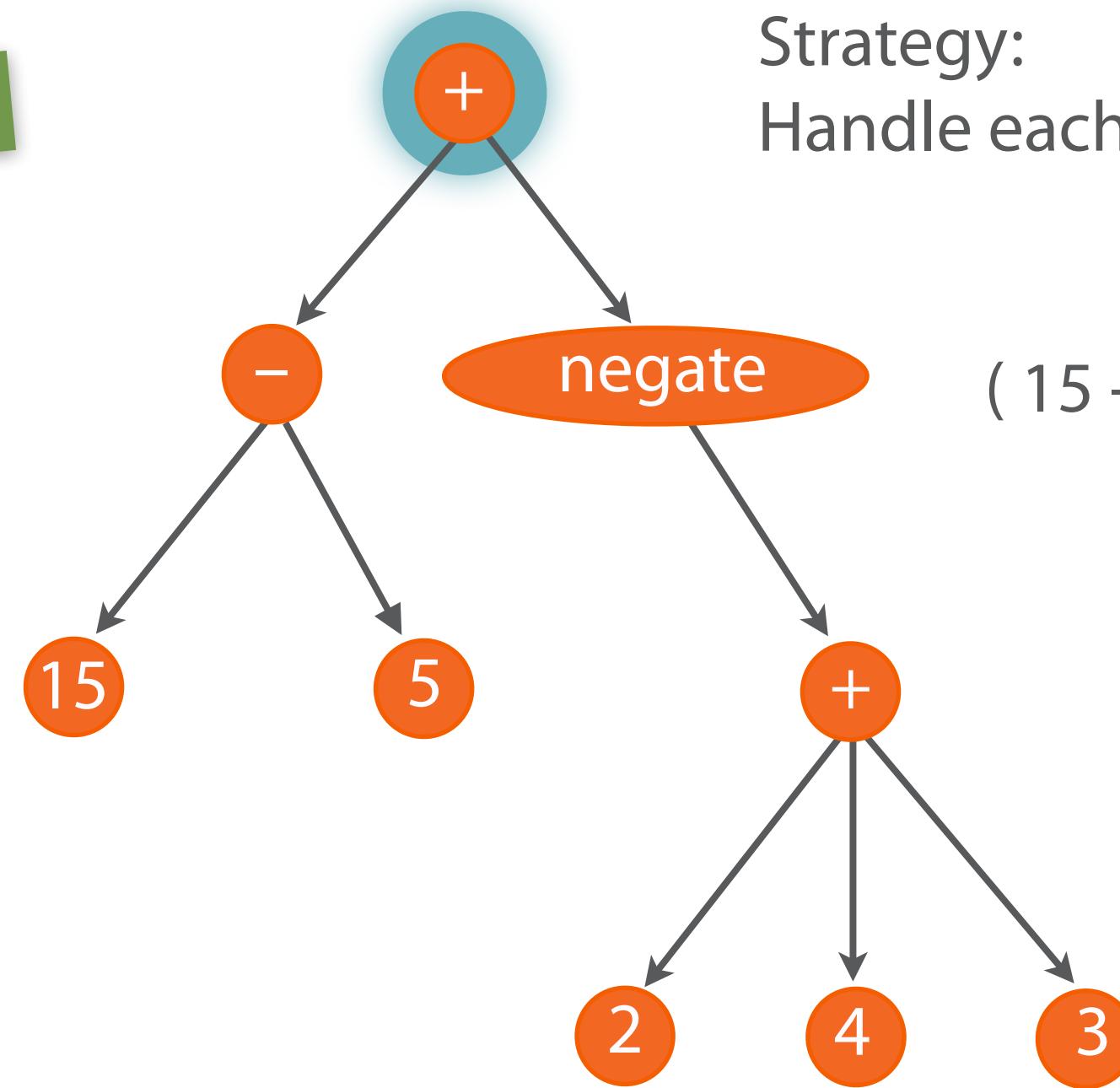


Strategy:  
Handle each node *after* its children.

$$(15 - 5) + (-1 \cdot (2 + 4 + 3))$$

# DFS: Postorder Traversal

For executing a hierarchy.

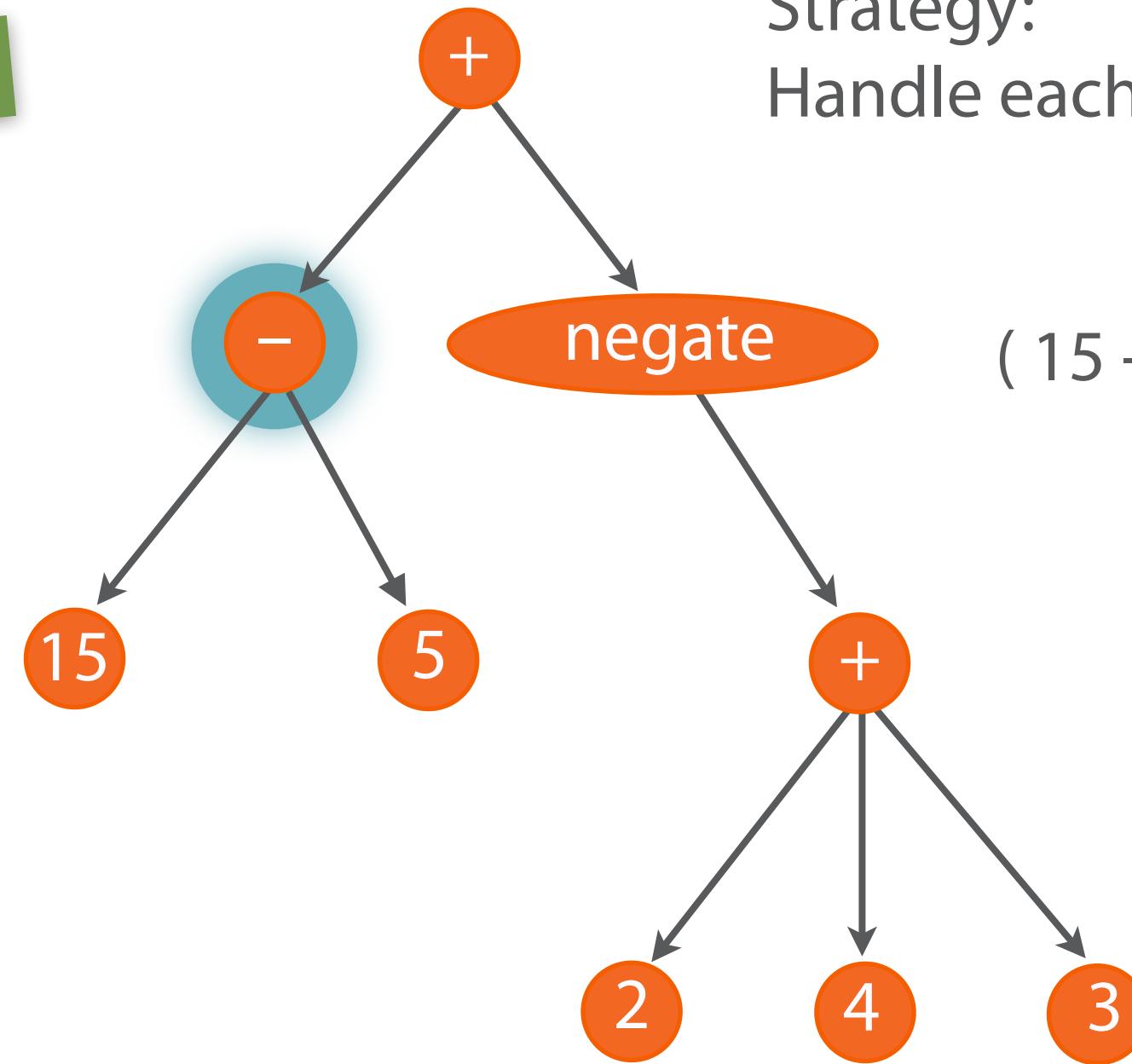


Strategy:  
Handle each node *after* its children.

$$(15 - 5) + (-1 \cdot (2 + 4 + 3))$$

# DFS: Postorder Traversal

For executing a hierarchy.

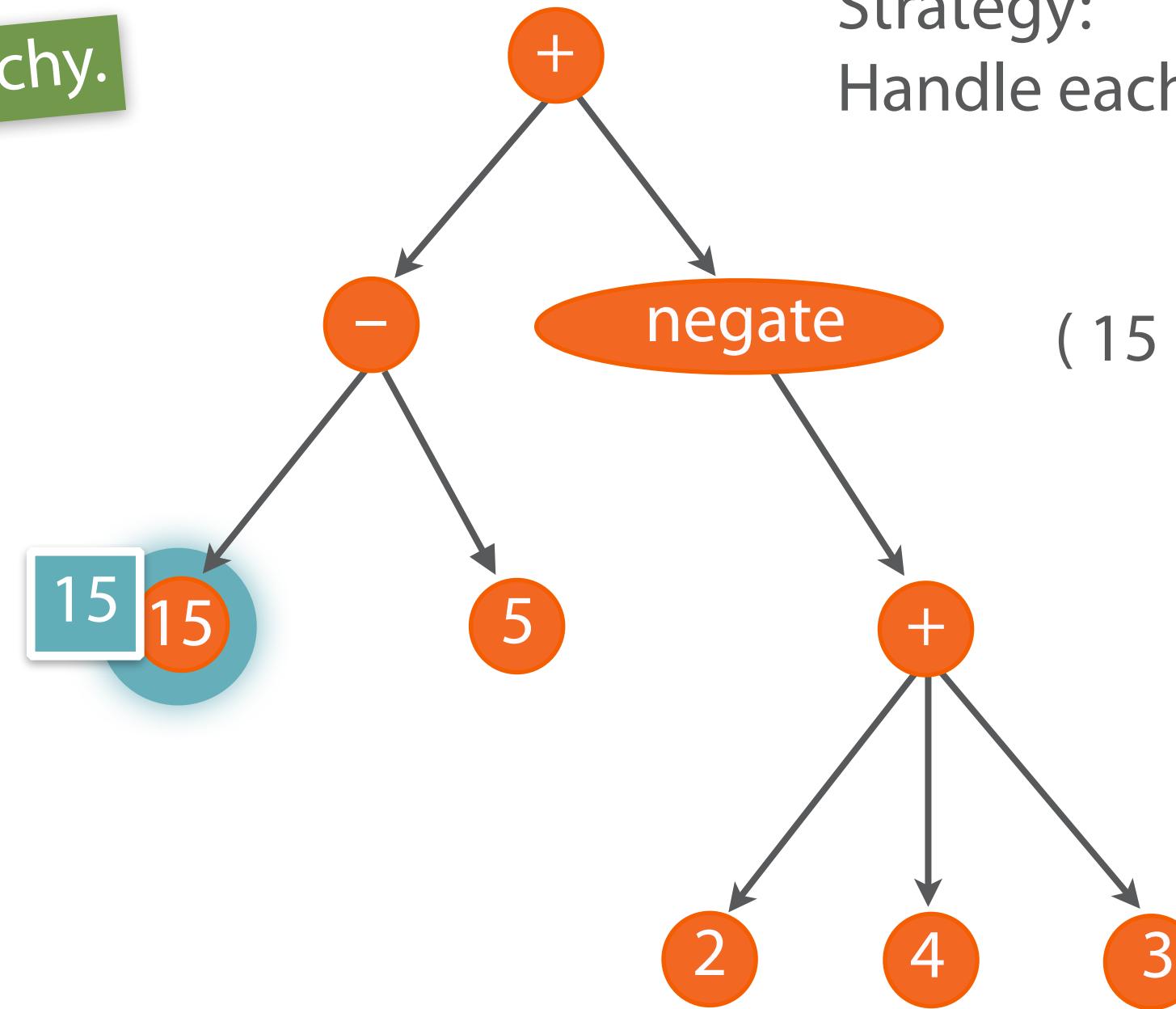


Strategy:  
Handle each node *after* its children.

$$(15 - 5) + (-1 \cdot (2 + 4 + 3))$$

# DFS: Postorder Traversal

For executing a hierarchy.

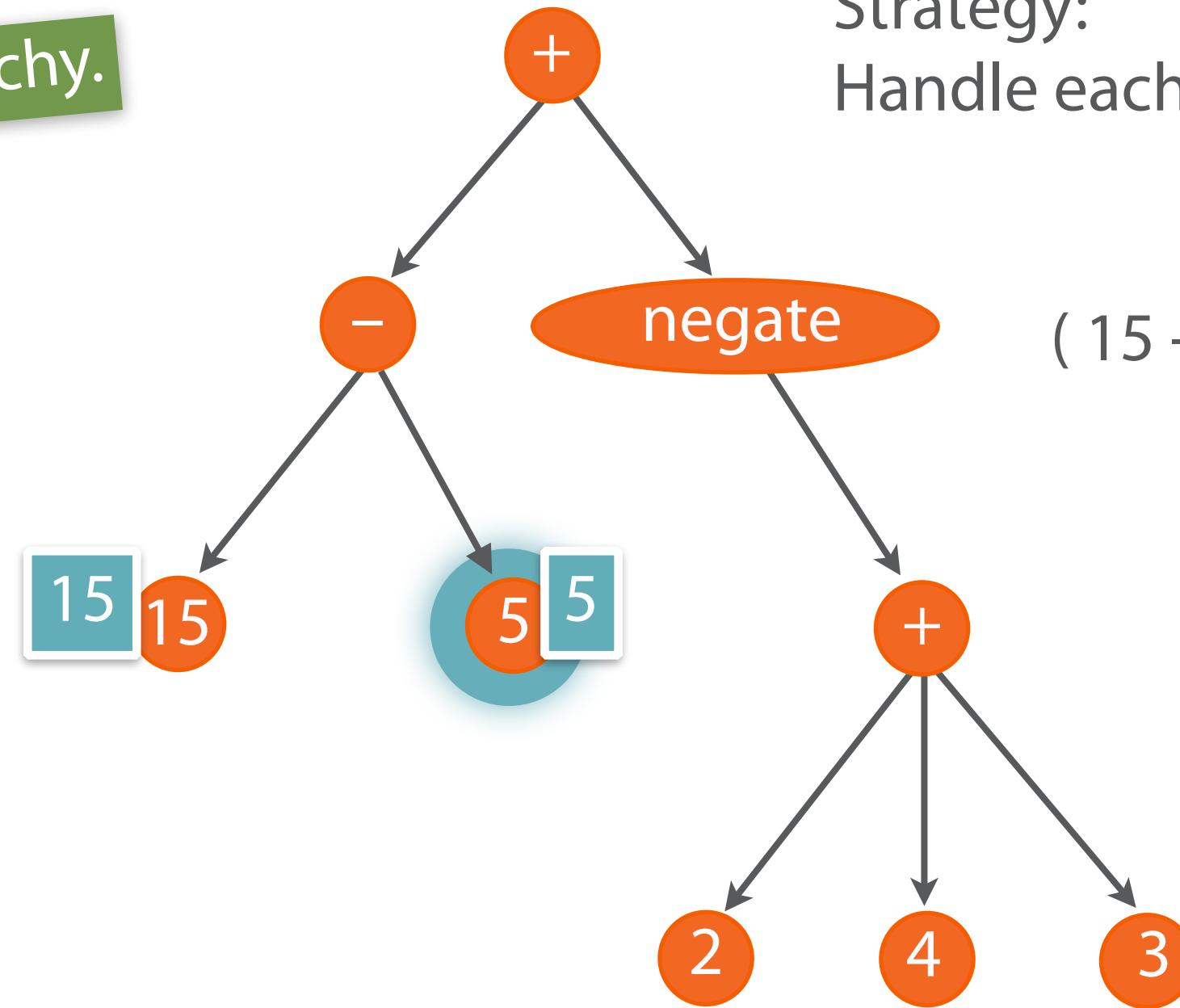


Strategy:  
Handle each node *after* its children.

$$(15 - 5) + (-1 \cdot (2 + 4 + 3))$$

# DFS: Postorder Traversal

For executing a hierarchy.

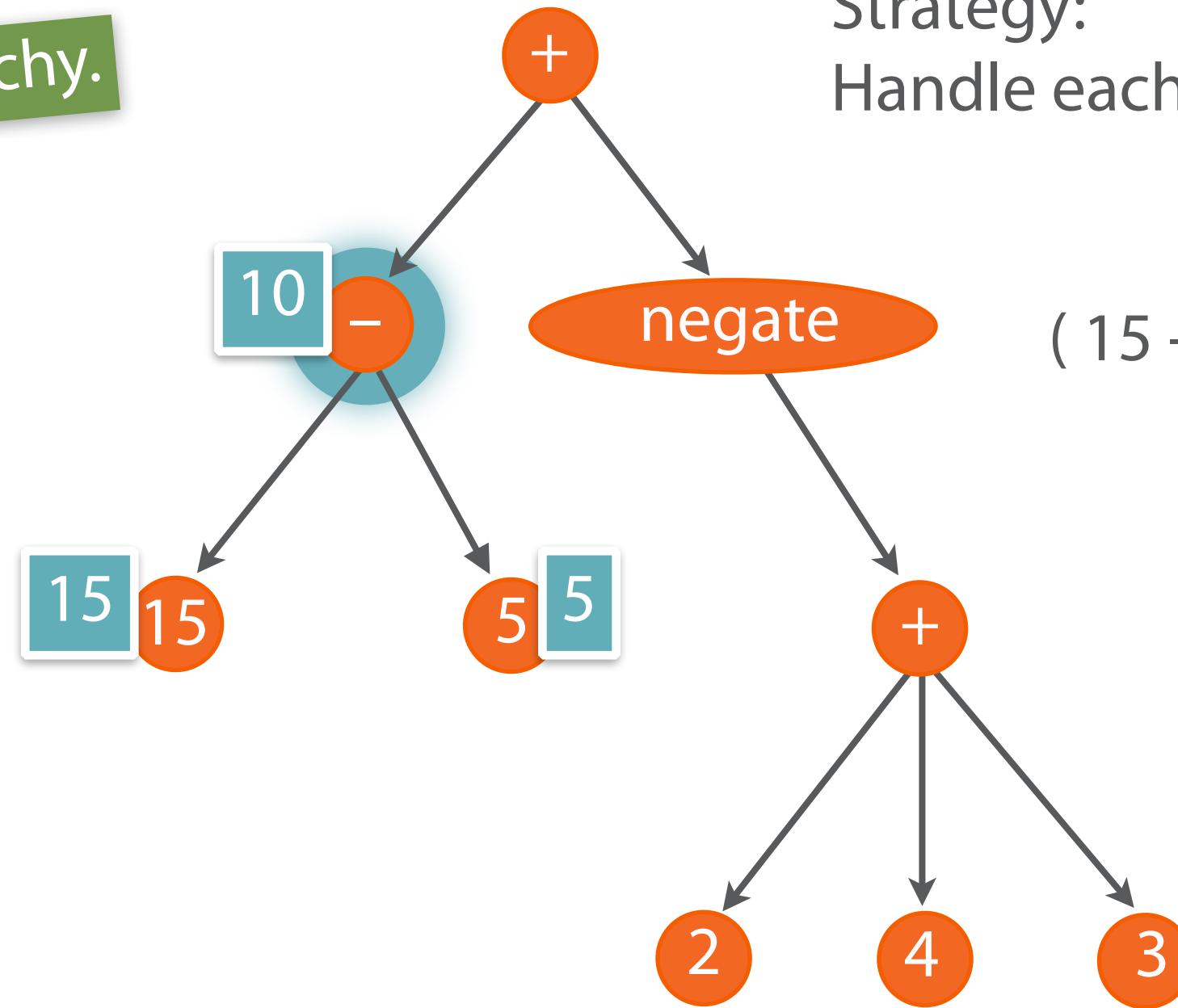


Strategy:  
Handle each node *after* its children.

$$(15 - 5) + (-1 \cdot (2 + 4 + 3))$$

# DFS: Postorder Traversal

For executing a hierarchy.

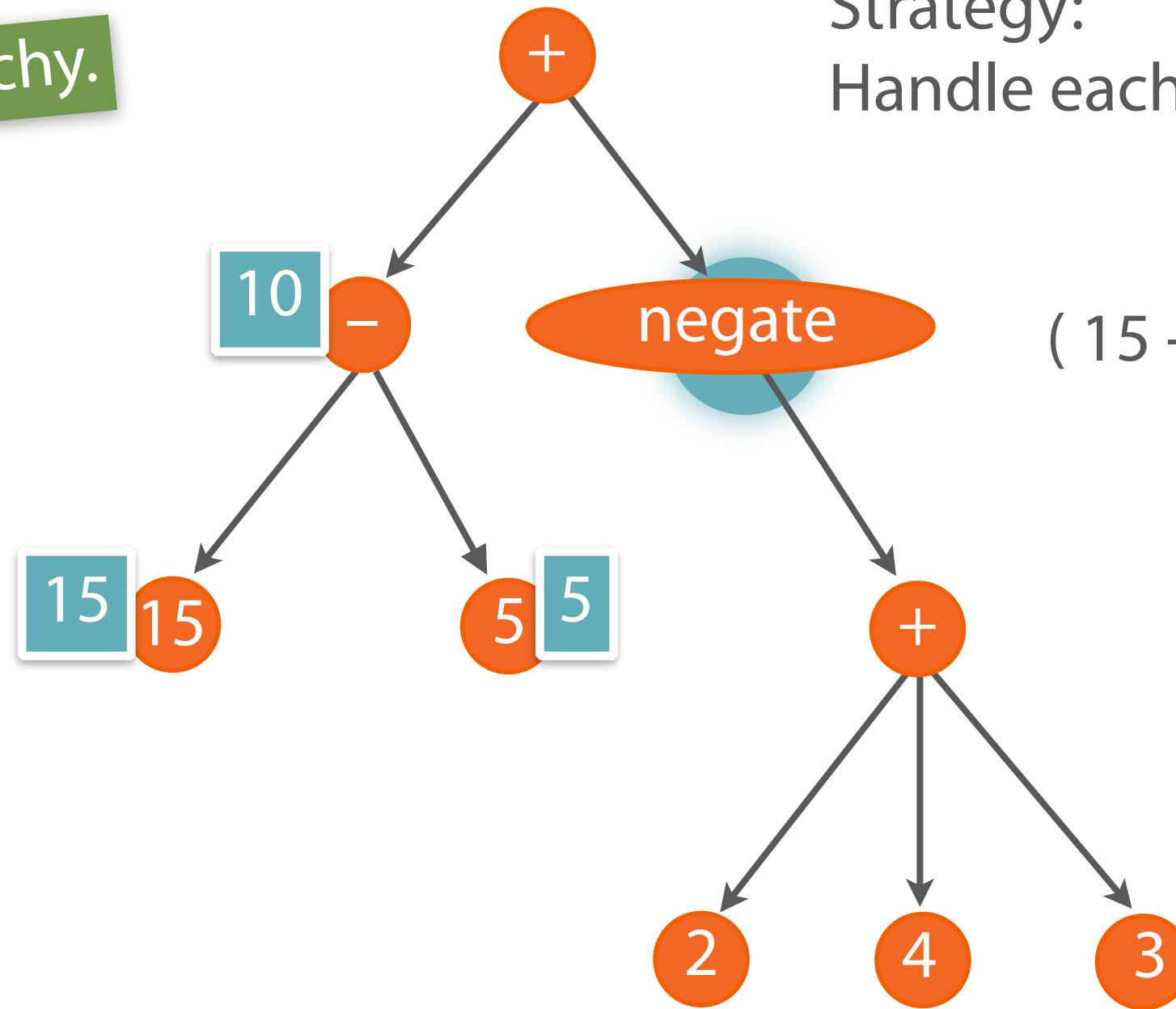


Strategy:  
Handle each node *after* its children.

$$(15 - 5) + (-1 \cdot (2 + 4 + 3))$$

# DFS: Postorder Traversal

For executing a hierarchy.

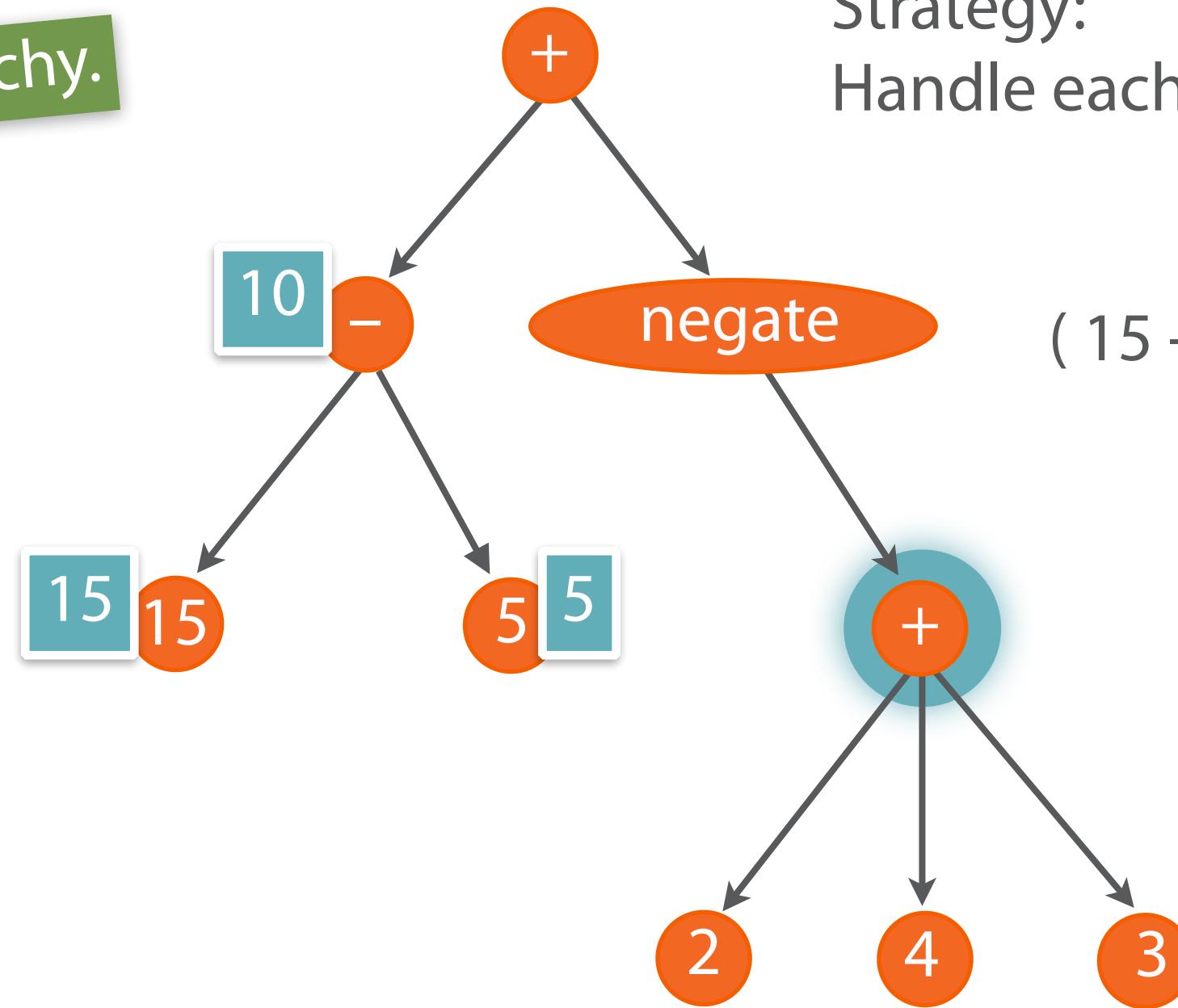


Strategy:  
Handle each node *after* its children.

$$(15 - 5) + (-1 \cdot (2 + 4 + 3))$$

# DFS: Postorder Traversal

For executing a hierarchy.

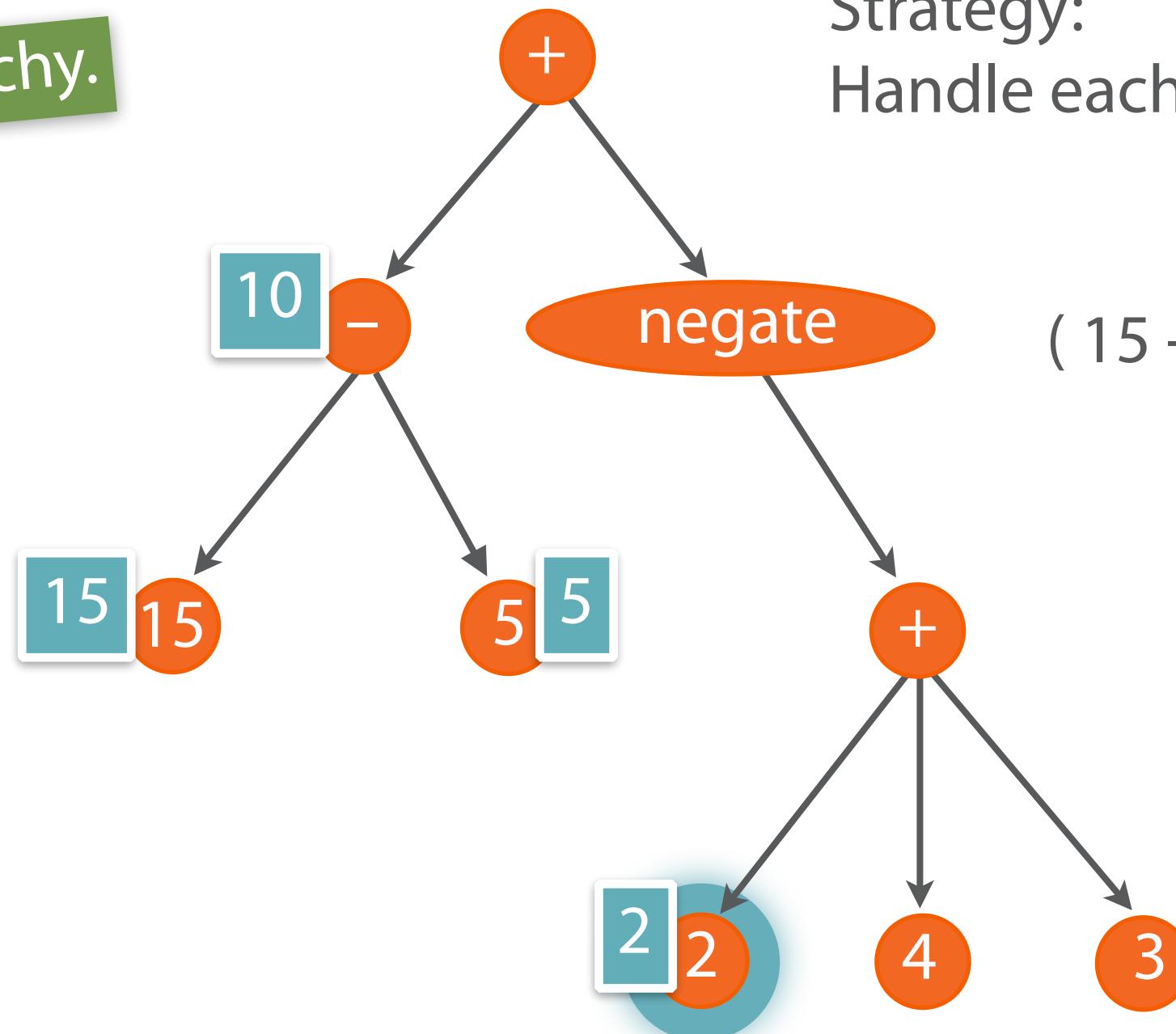


Strategy:  
Handle each node *after* its children.

$$(15 - 5) + (-1 \cdot (2 + 4 + 3))$$

# DFS: Postorder Traversal

For executing a hierarchy.

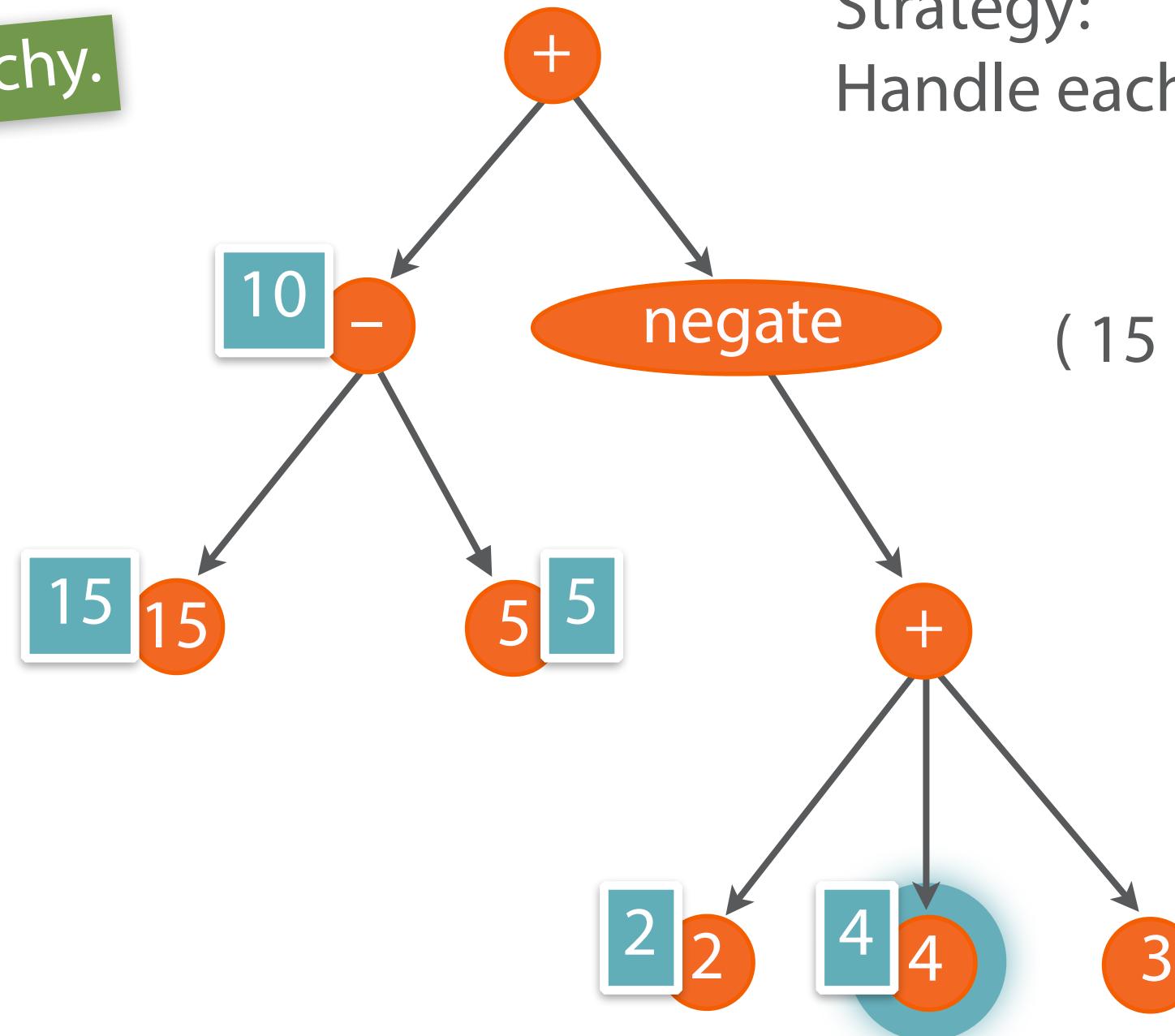


Strategy:  
Handle each node *after* its children.

$$(15 - 5) + (-1 \cdot (2 + 4 + 3))$$

# DFS: Postorder Traversal

For executing a hierarchy.

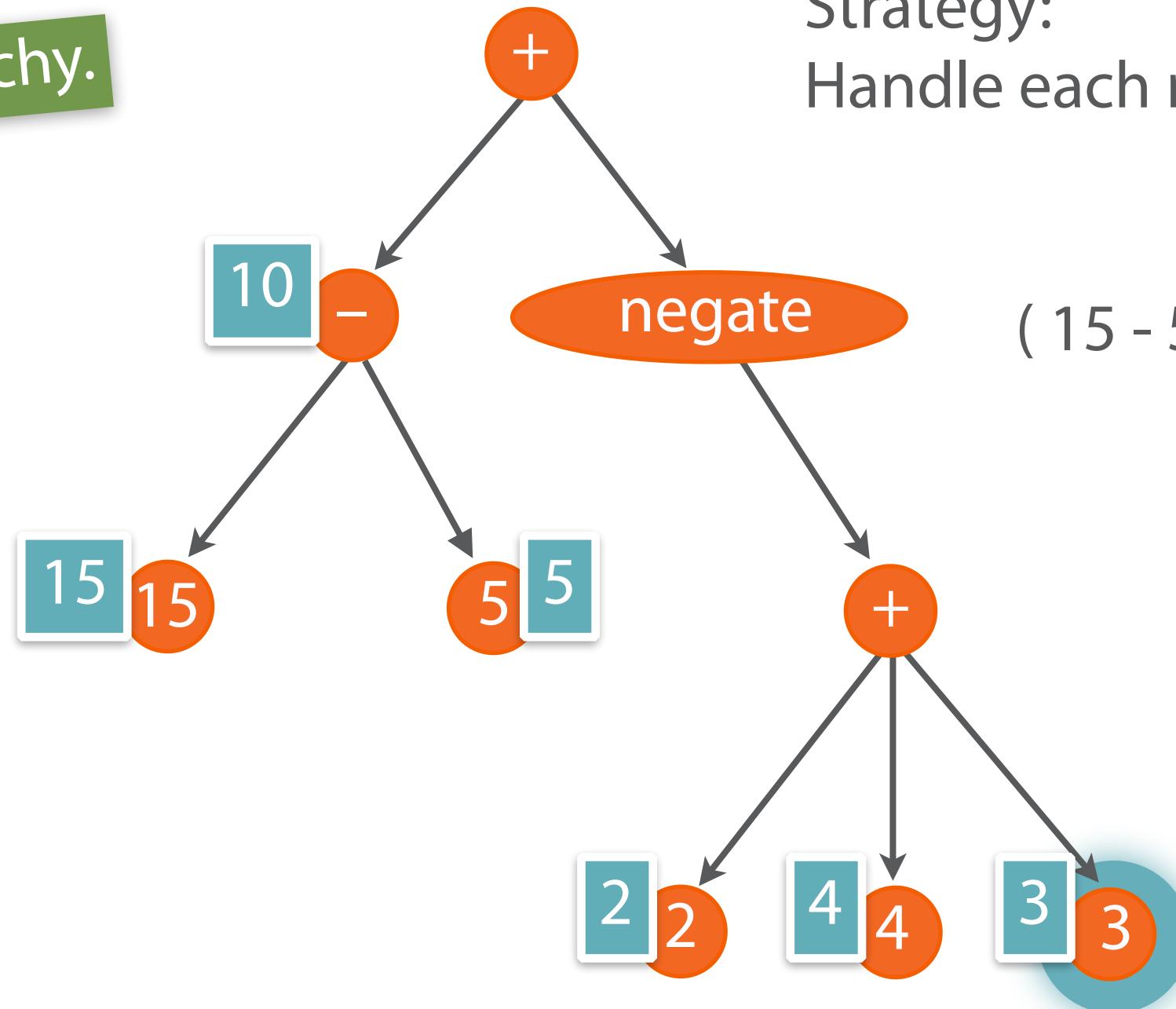


Strategy:  
Handle each node *after* its children.

$$(15 - 5) + (-1 \cdot (2 + 4 + 3))$$

# DFS: Postorder Traversal

For executing a hierarchy.

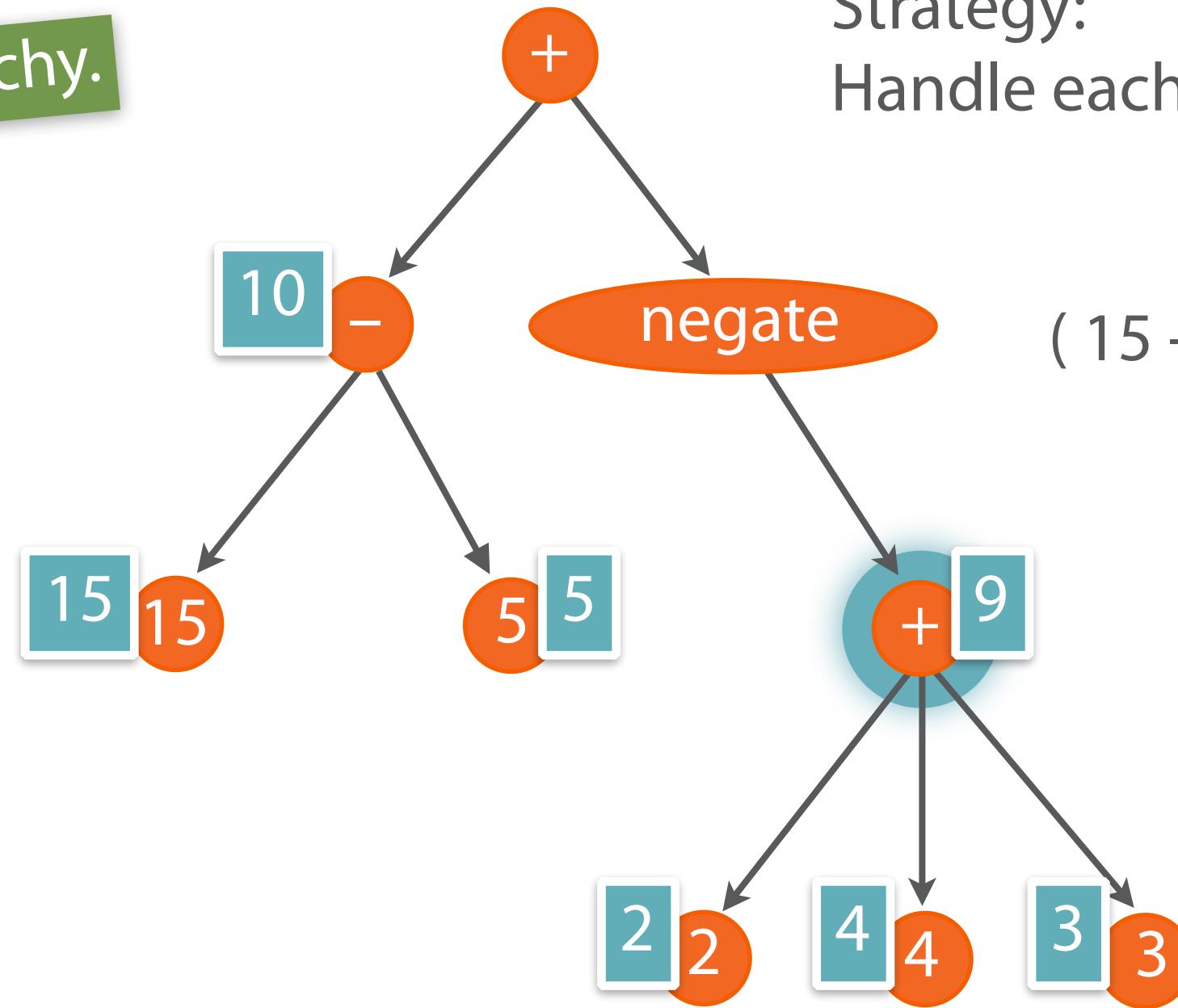


Strategy:  
Handle each node *after* its children.

$$(15 - 5) + (-1 \cdot (2 + 4 + 3))$$

# DFS: Postorder Traversal

For executing a hierarchy.

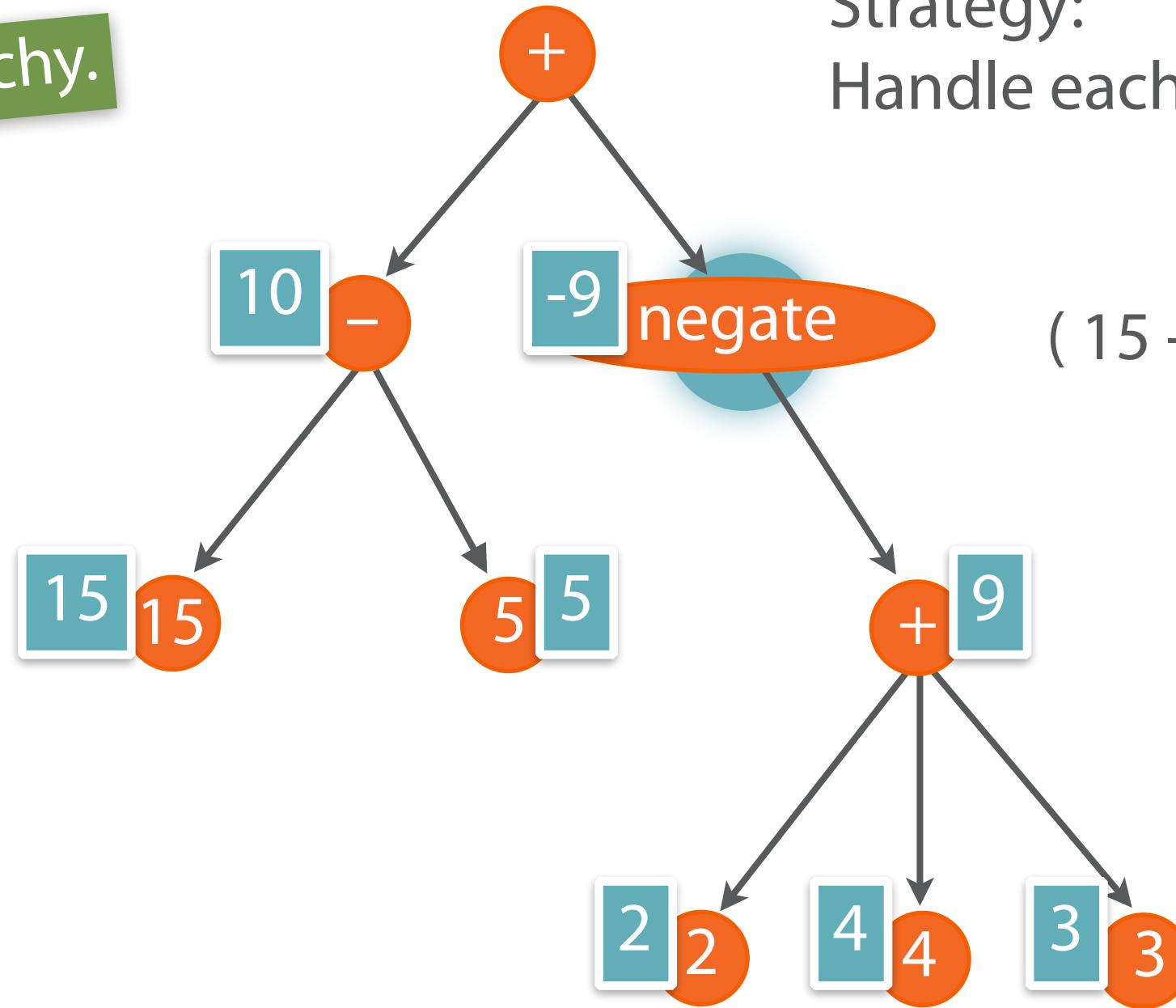


Strategy:  
Handle each node *after* its children.

$$(15 - 5) + (-1 \cdot (2 + 4 + 3))$$

# DFS: Postorder Traversal

For executing a hierarchy.

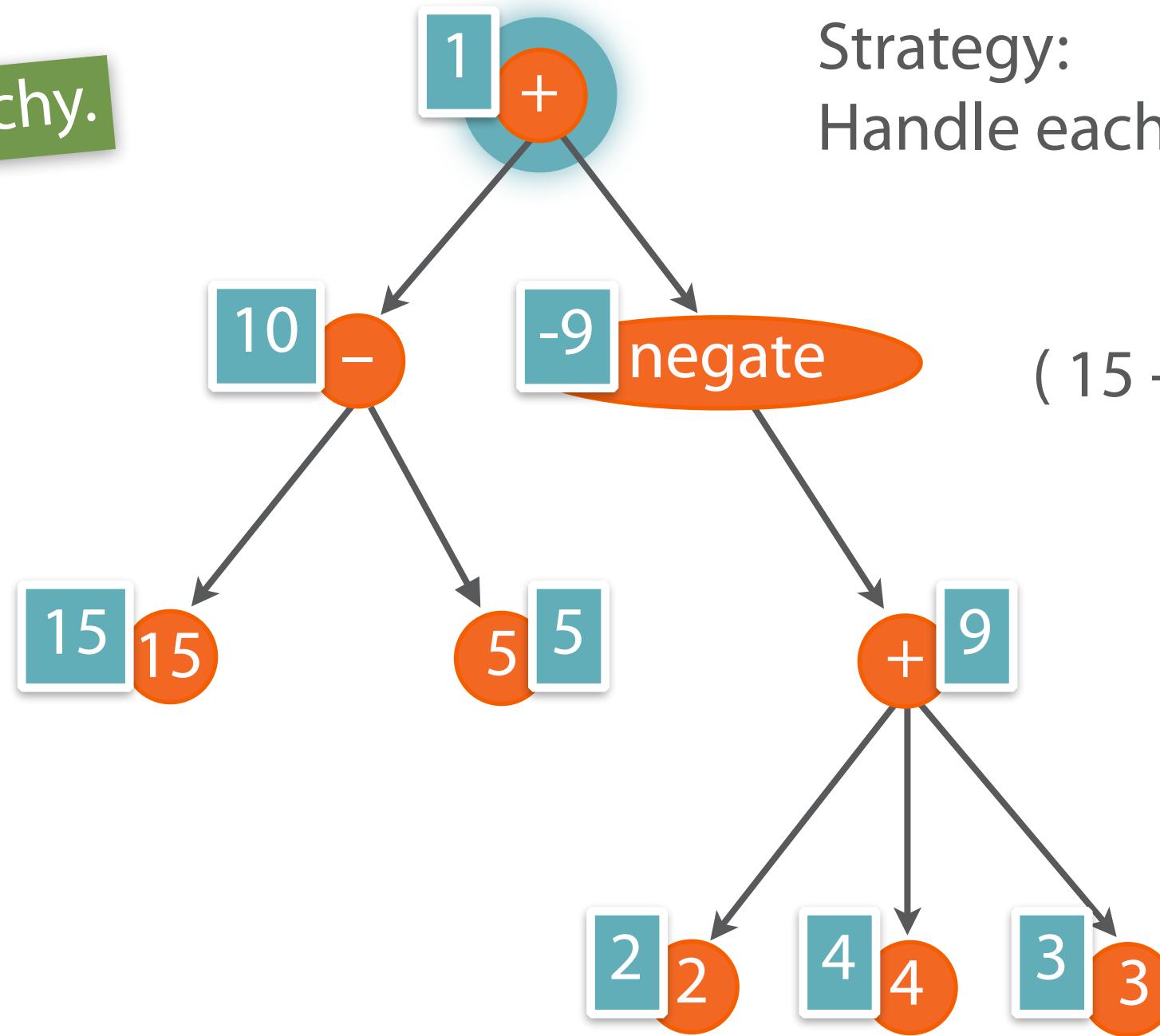


Strategy:  
Handle each node *after* its children.

$$(15 - 5) + (-1 \cdot (2 + 4 + 3))$$

# DFS: Postorder Traversal

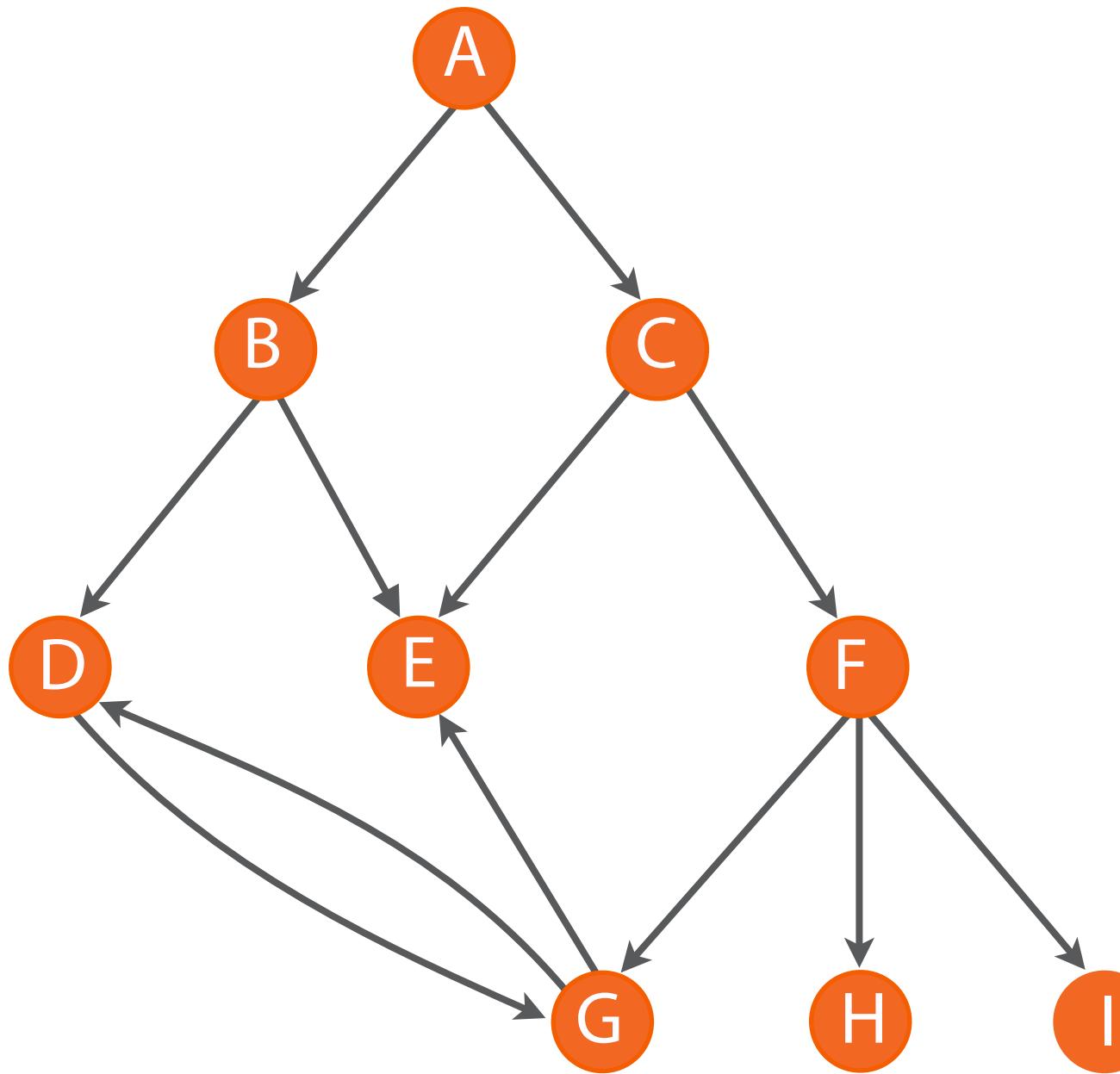
For executing a hierarchy.



Strategy:  
Handle each node *after* its children.

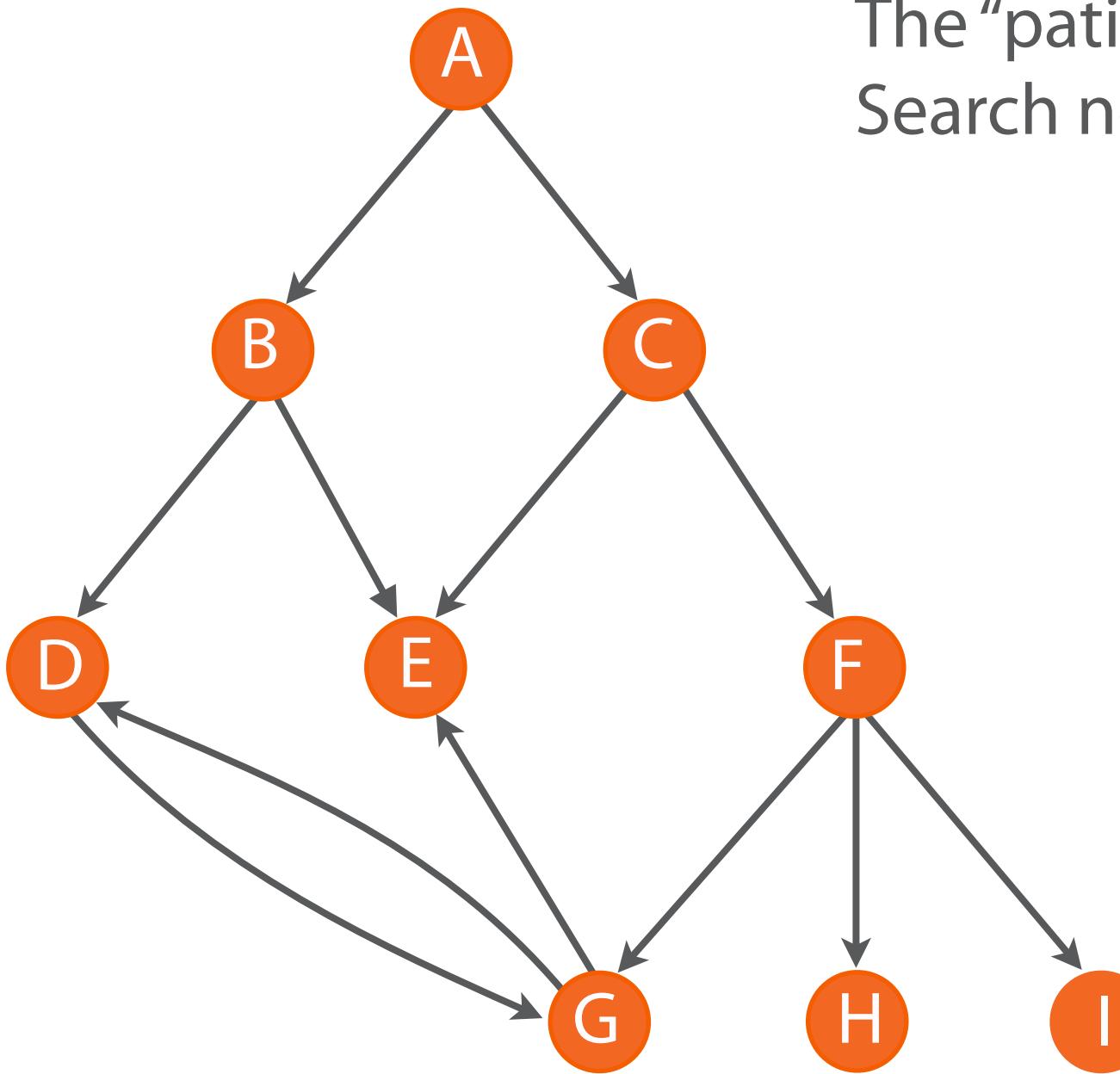
$$(15 - 5) + (-1 \cdot (2 + 4 + 3))$$

# Breadth First Search (BFS)



# Breadth First Search (BFS)

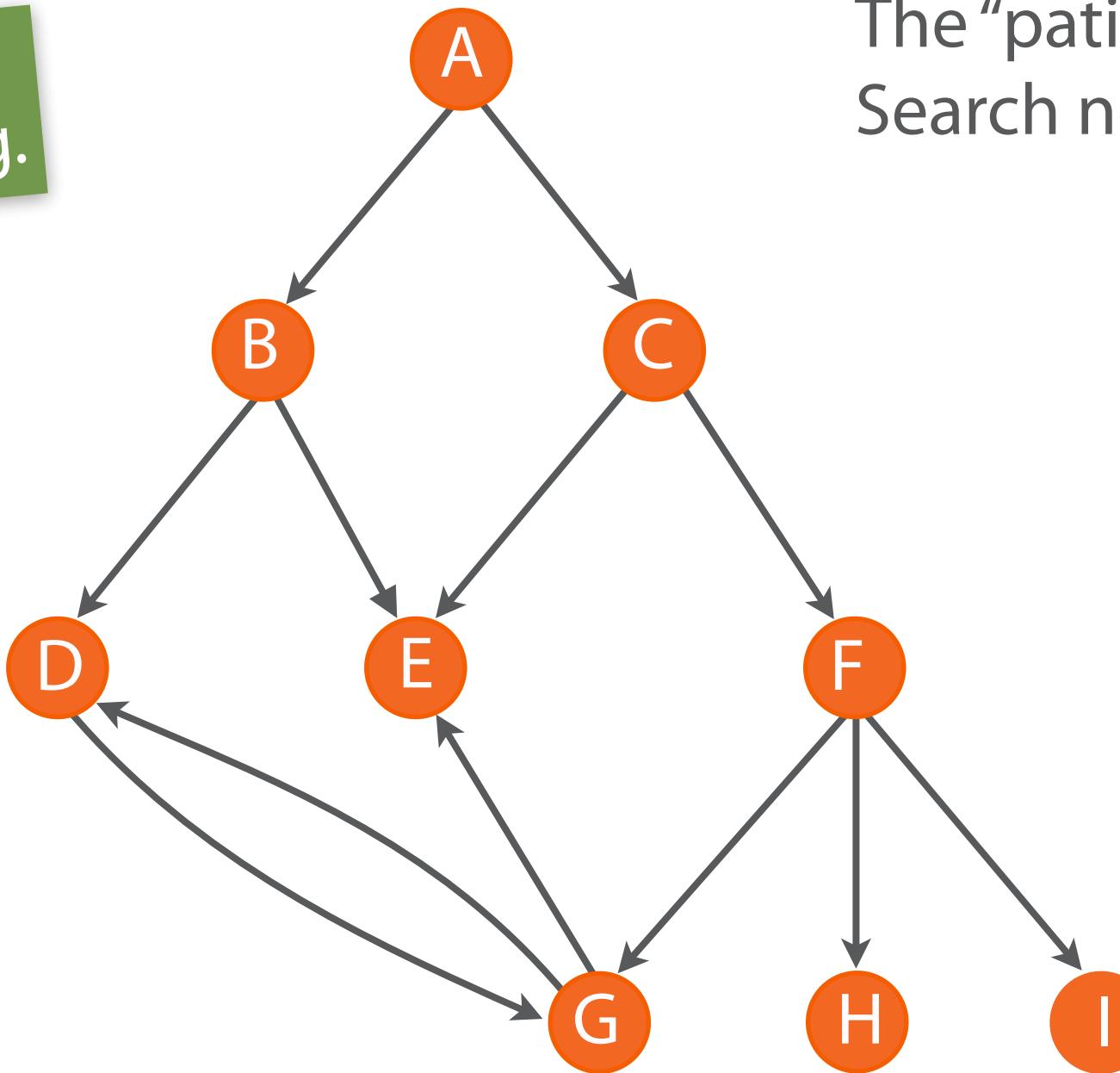
The “patient” strategy:  
Search nearest nodes first.



# Breadth First Search (BFS)

Explore all short-distance options before proceeding.

The “patient” strategy:  
Search nearest nodes first.

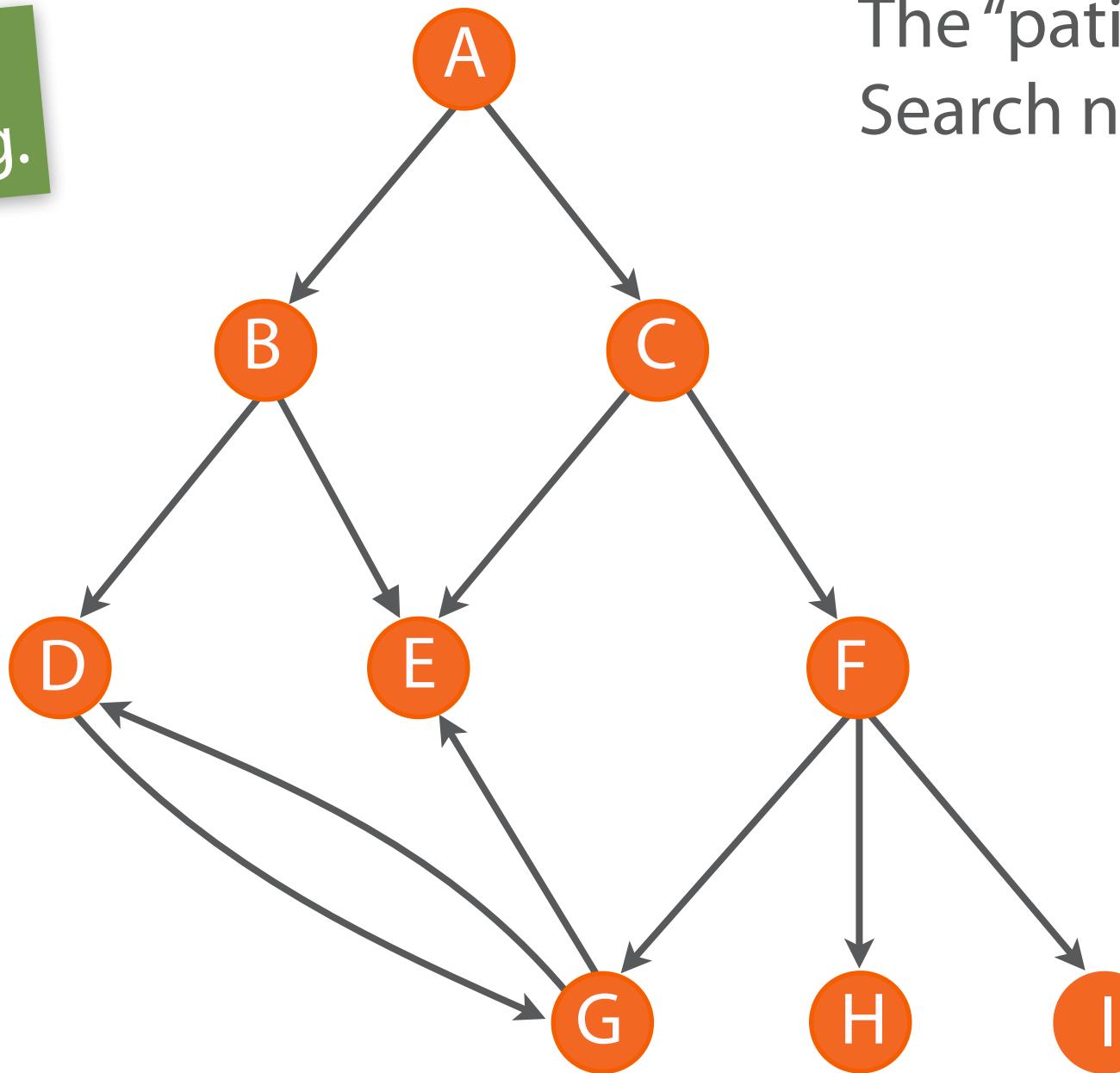


# Breadth First Search (BFS)

Explore all short-distance options before proceeding.

Find distance to starting point node.  
(Routing)

The “patient” strategy:  
Search nearest nodes first.

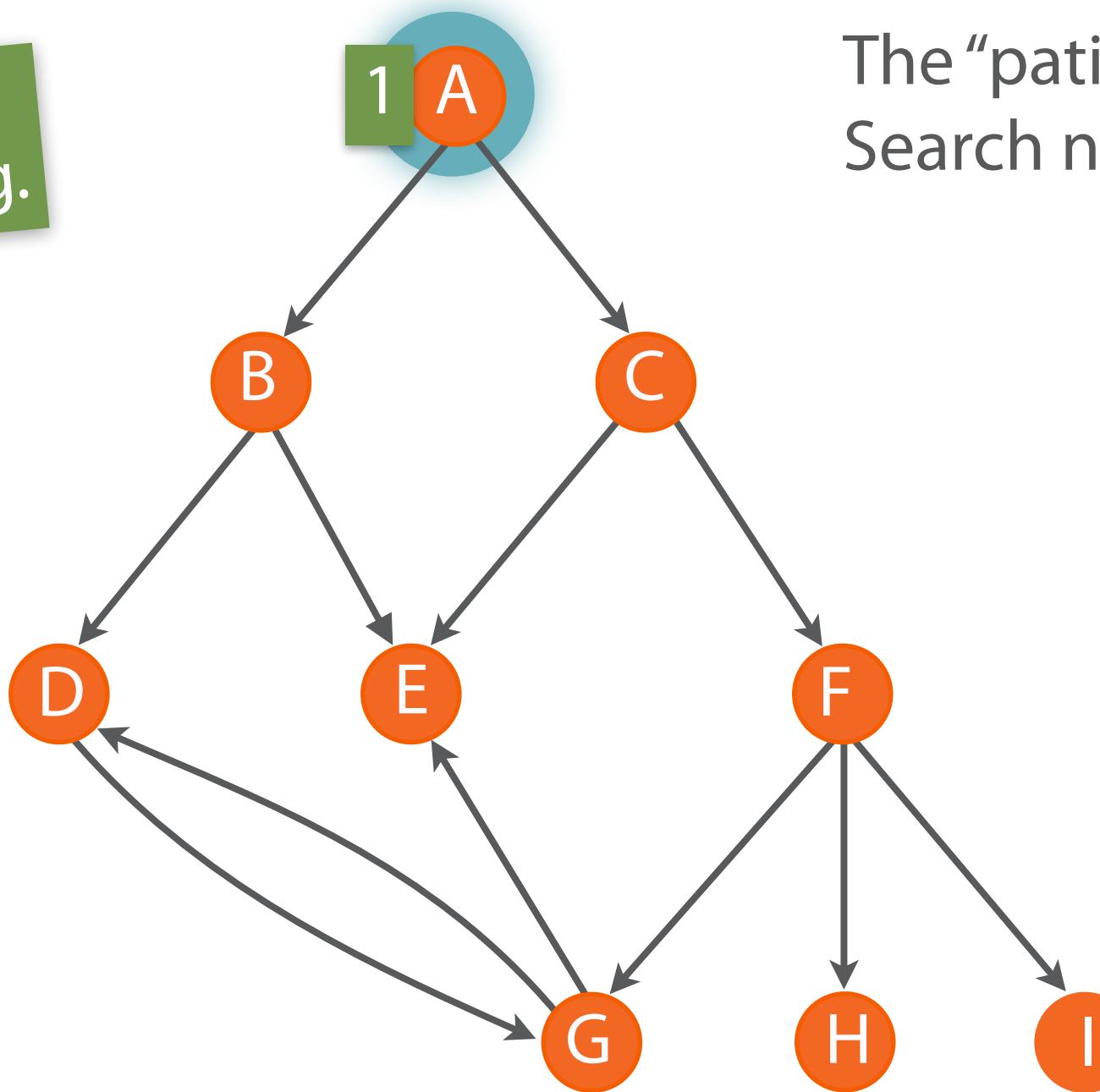


# Breadth First Search (BFS)

Explore all short-distance options before proceeding.

Find distance to starting point node.  
(Routing)

The “patient” strategy:  
Search nearest nodes first.



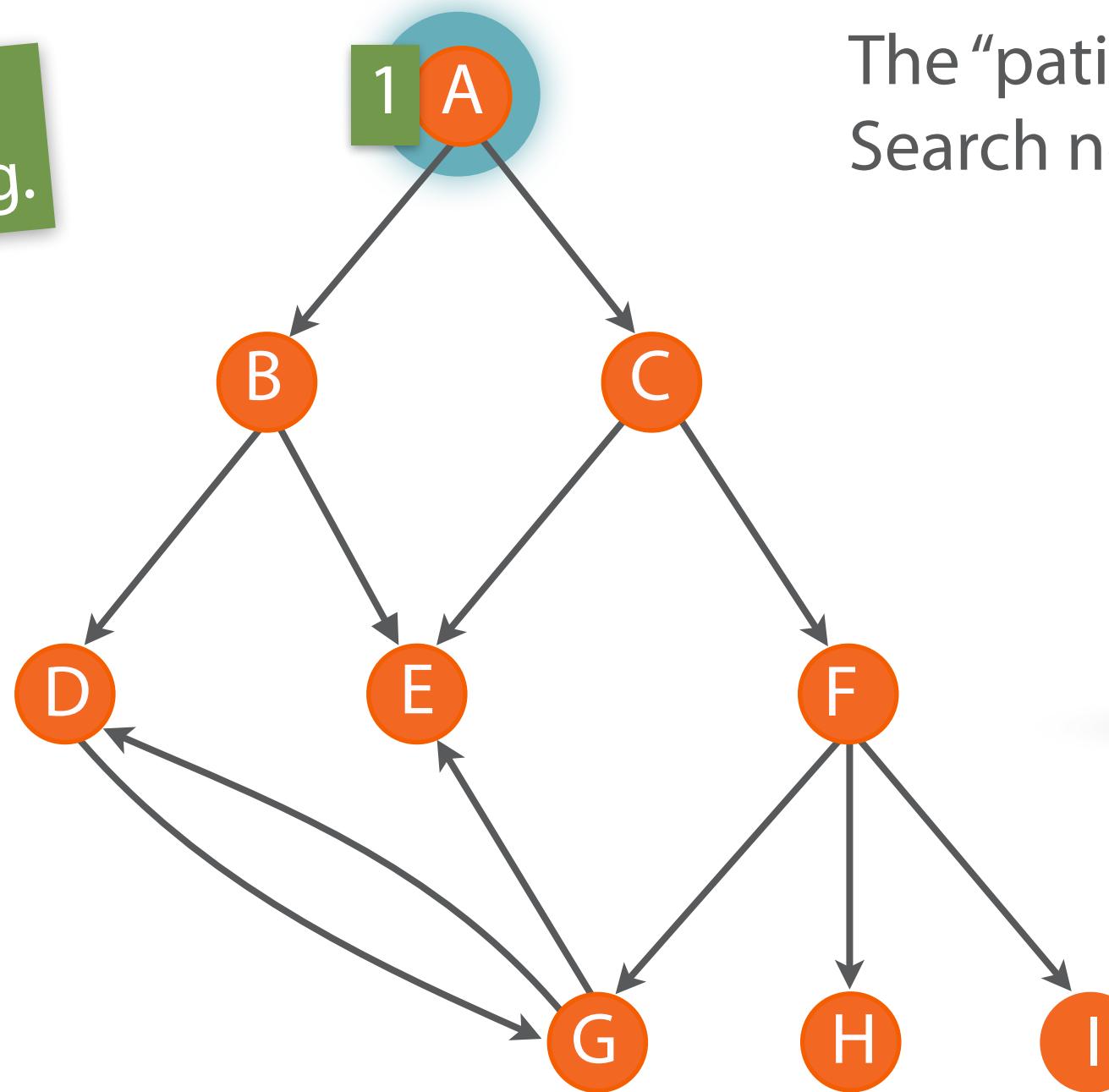
# Breadth First Search (BFS)

Explore all short-distance options before proceeding.

Find distance to starting point node.  
(Routing)

The “patient” strategy:  
Search nearest nodes first.

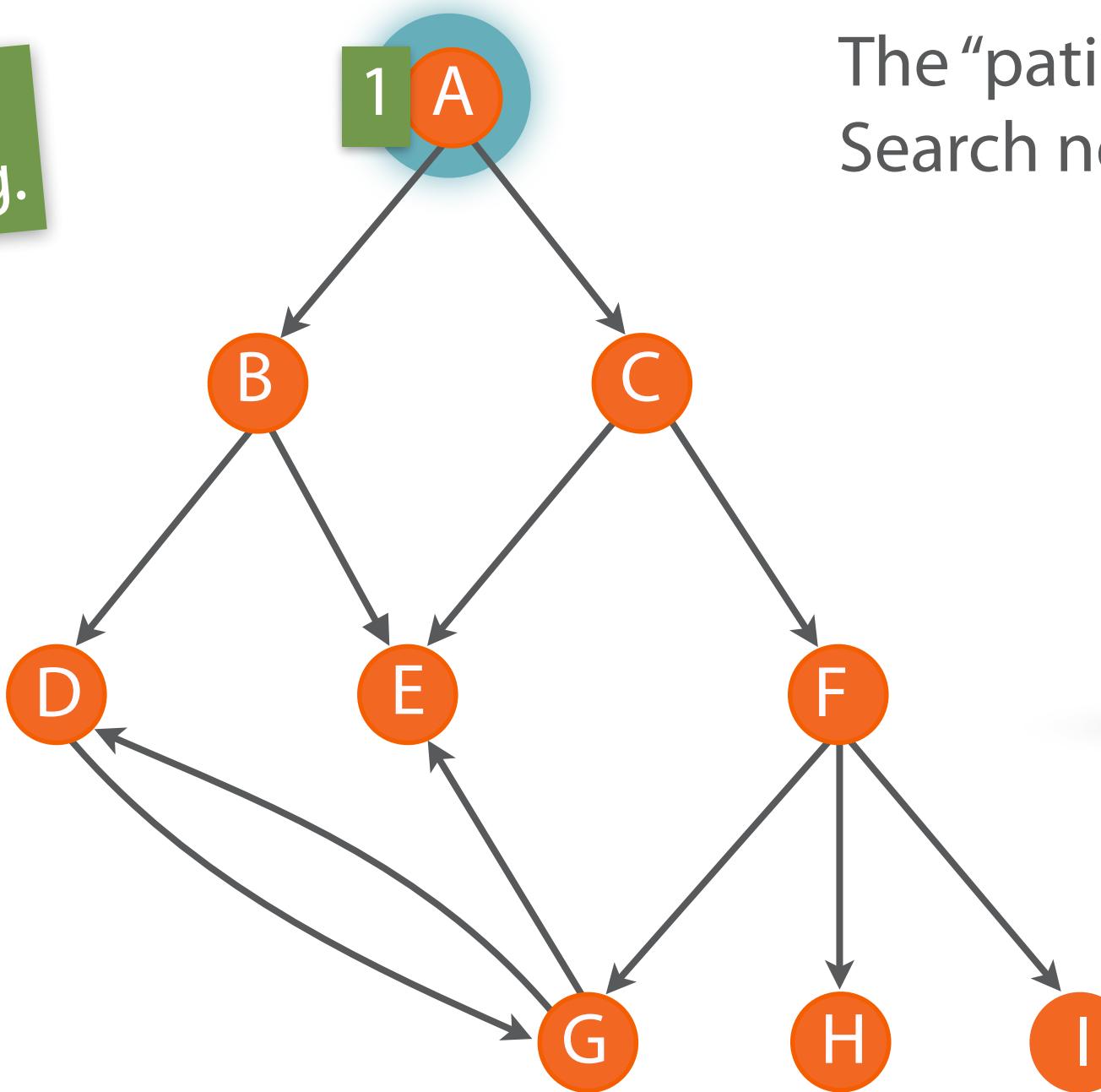
Queue



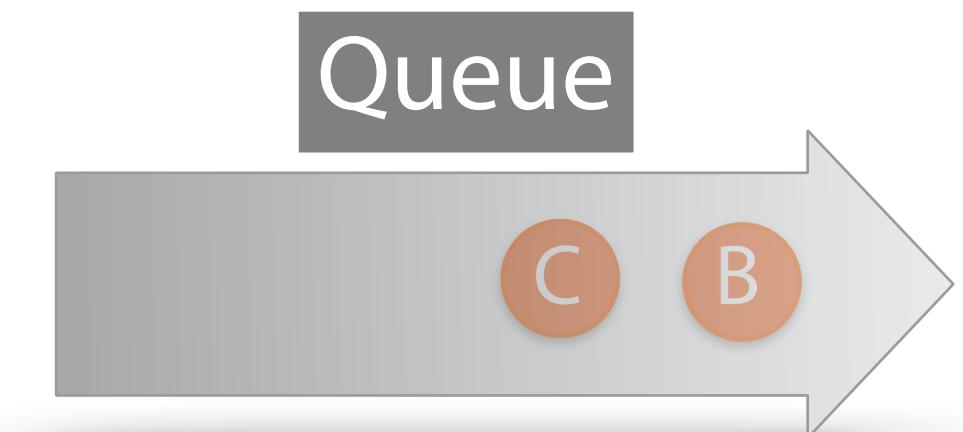
# Breadth First Search (BFS)

Explore all short-distance options before proceeding.

Find distance to starting point node.  
(Routing)



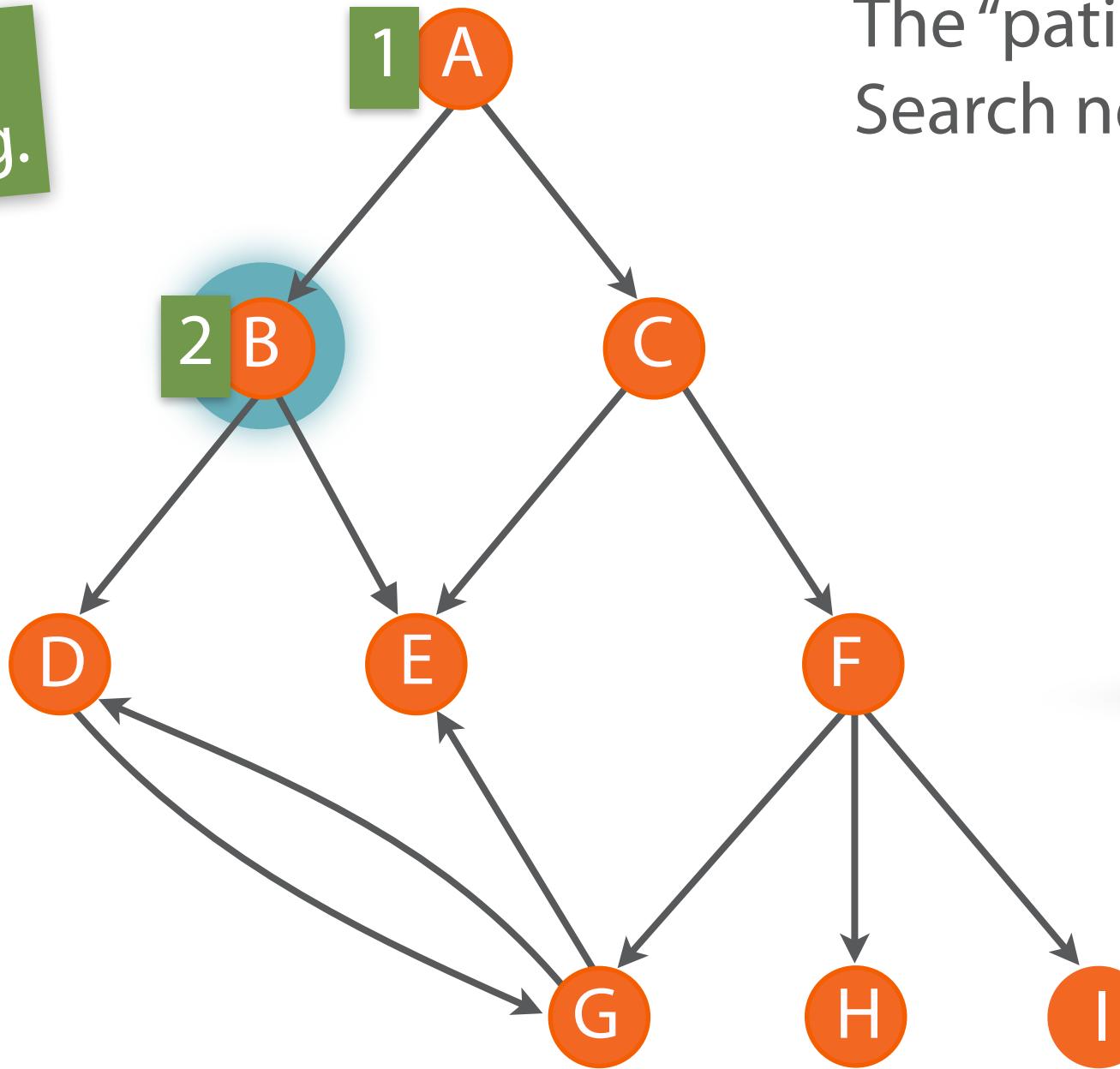
The “patient” strategy:  
Search nearest nodes first.



# Breadth First Search (BFS)

Explore all short-distance options before proceeding.

Find distance to starting point node.  
(Routing)



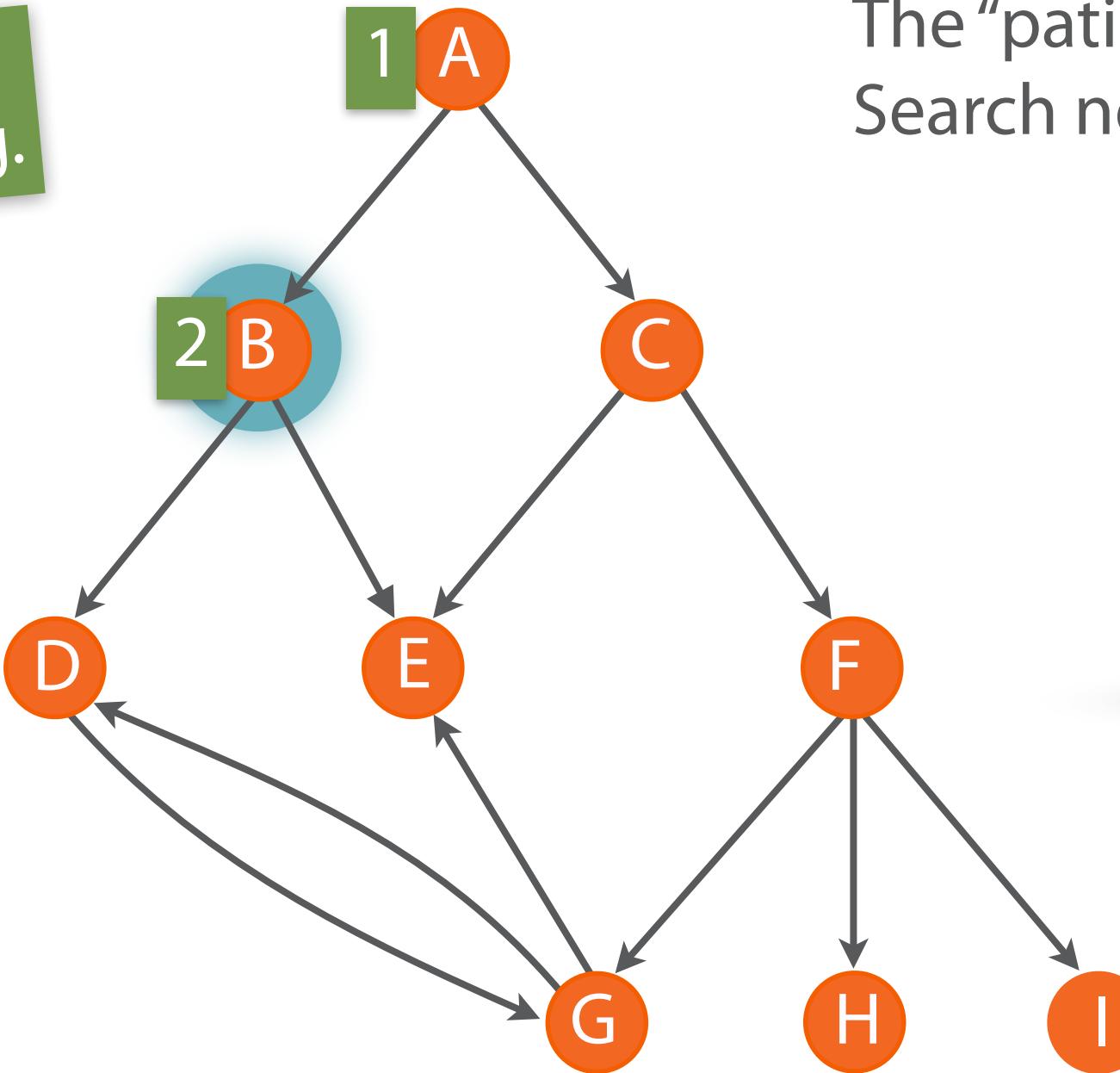
The “patient” strategy:  
Search nearest nodes first.



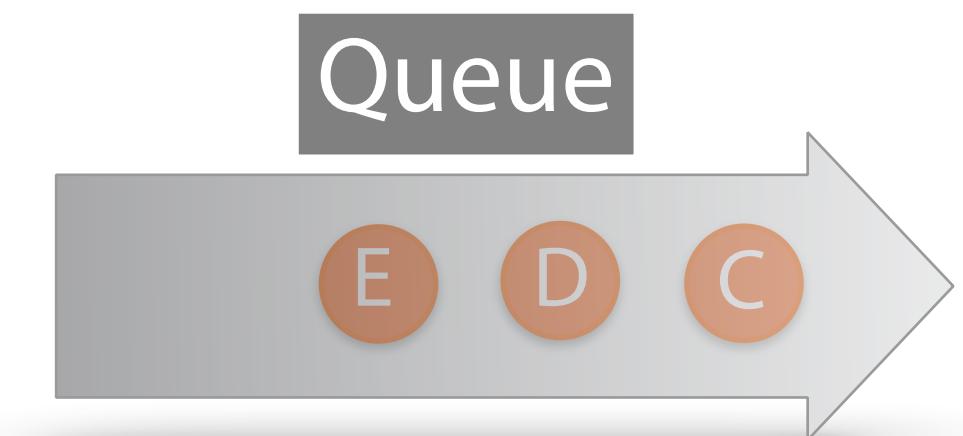
# Breadth First Search (BFS)

Explore all short-distance options before proceeding.

Find distance to starting point node.  
(Routing)



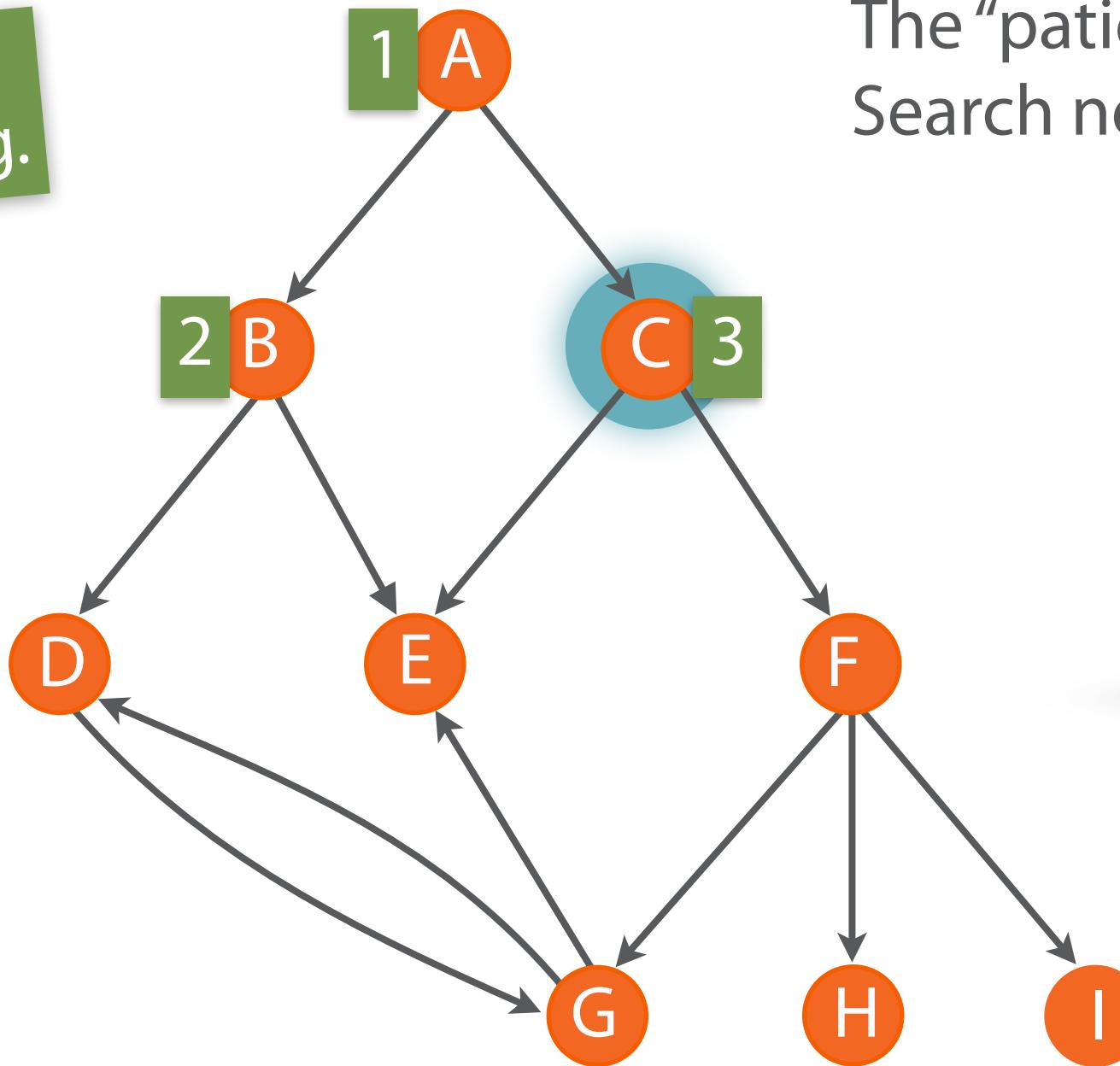
The “patient” strategy:  
Search nearest nodes first.



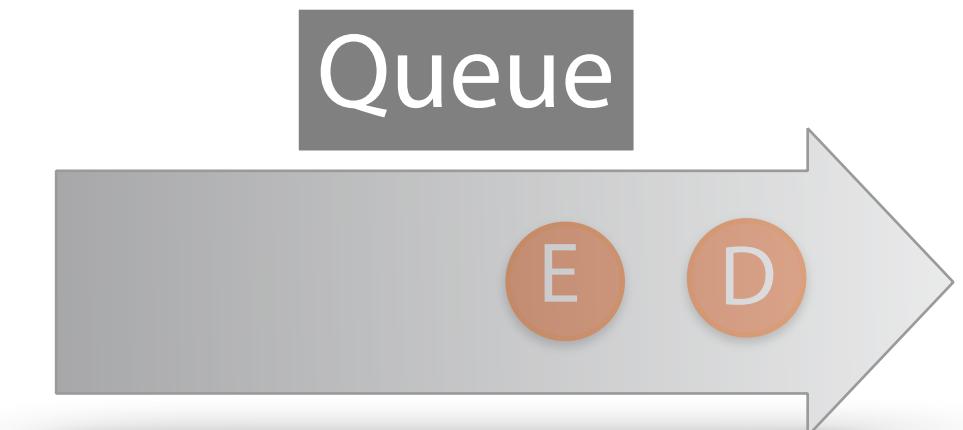
# Breadth First Search (BFS)

Explore all short-distance options before proceeding.

Find distance to starting point node.  
(Routing)



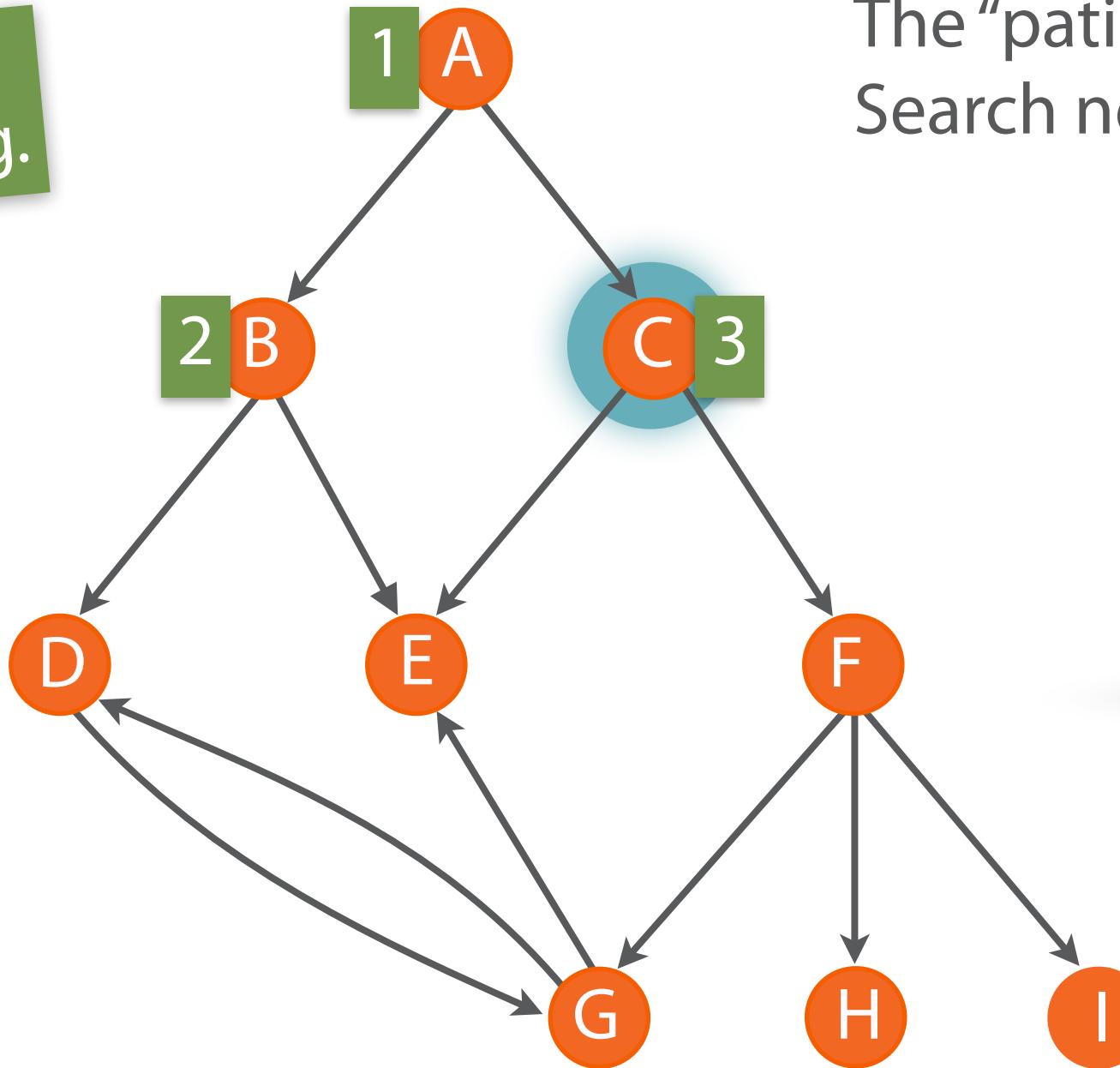
The “patient” strategy:  
Search nearest nodes first.



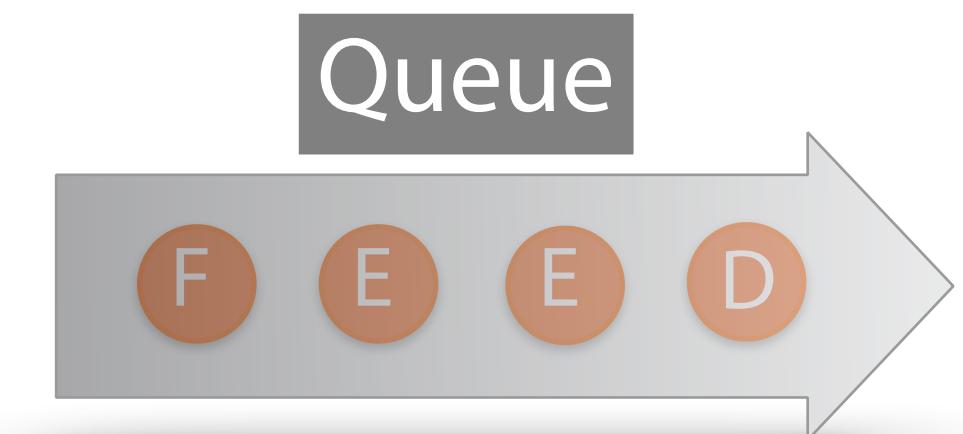
# Breadth First Search (BFS)

Explore all short-distance options before proceeding.

Find distance to starting point node.  
(Routing)



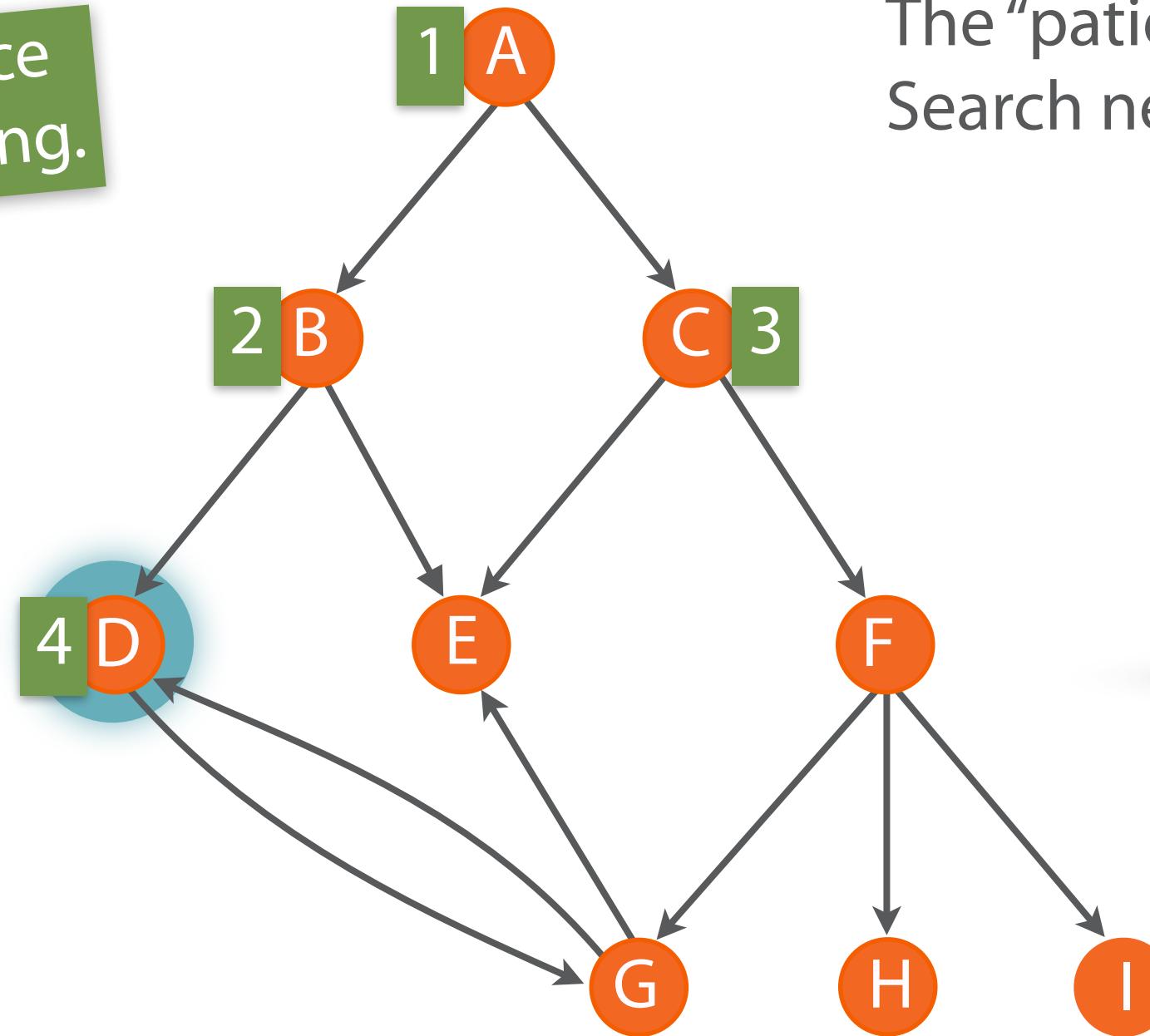
The “patient” strategy:  
Search nearest nodes first.



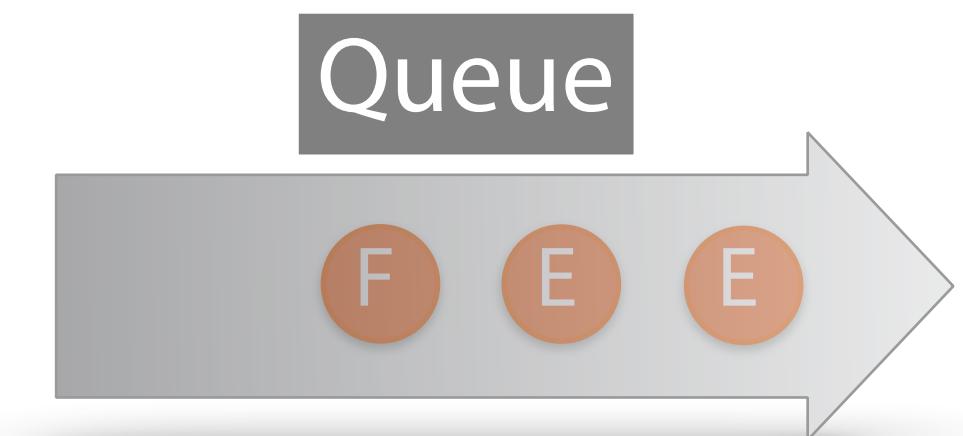
# Breadth First Search (BFS)

Explore all short-distance options before proceeding.

Find distance to starting point node.  
(Routing)



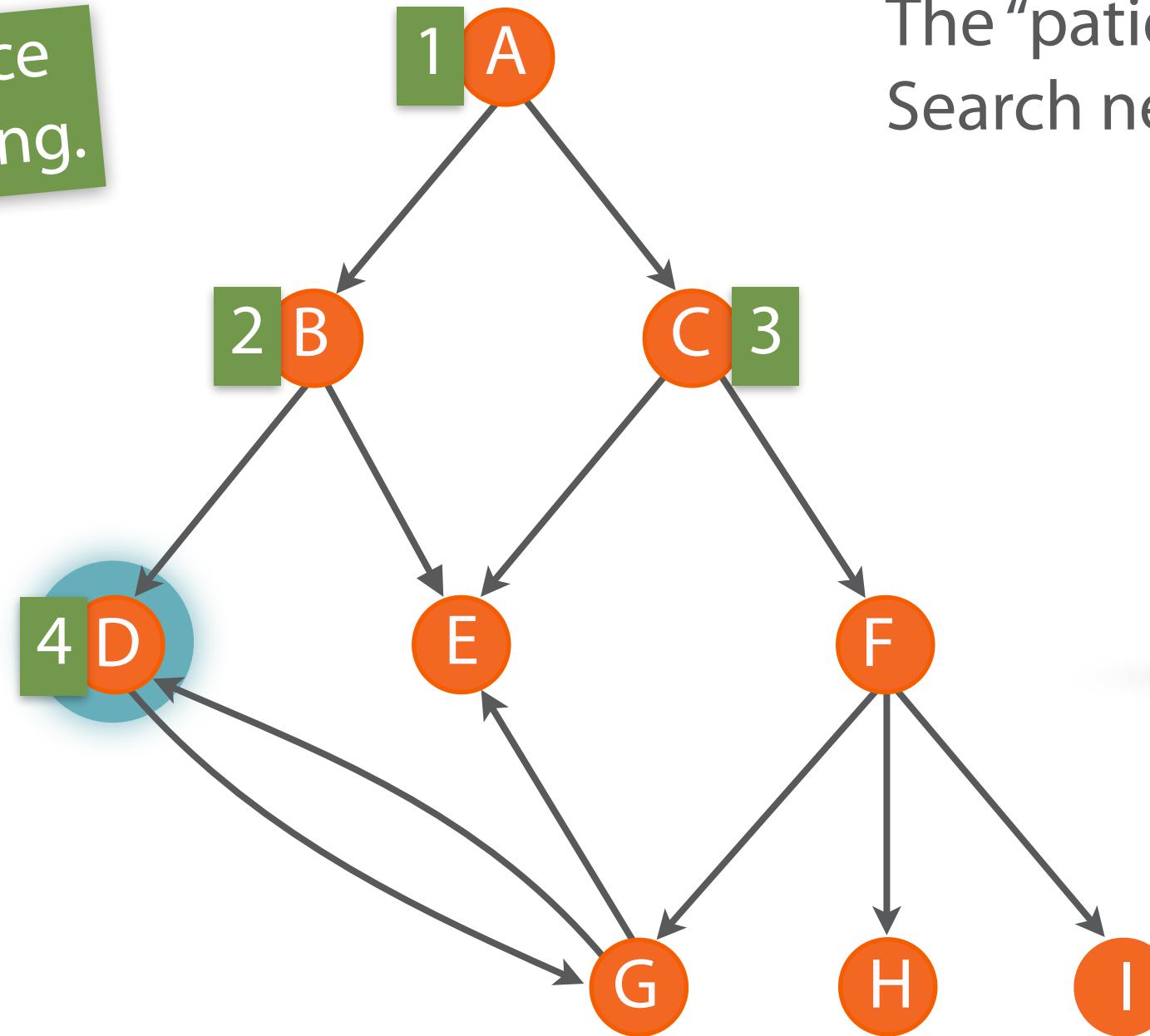
The “patient” strategy:  
Search nearest nodes first.



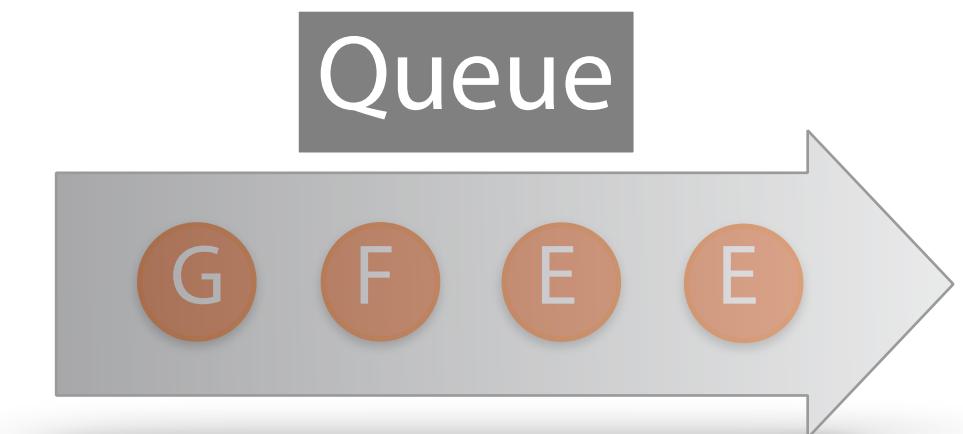
# Breadth First Search (BFS)

Explore all short-distance options before proceeding.

Find distance to starting point node.  
(Routing)



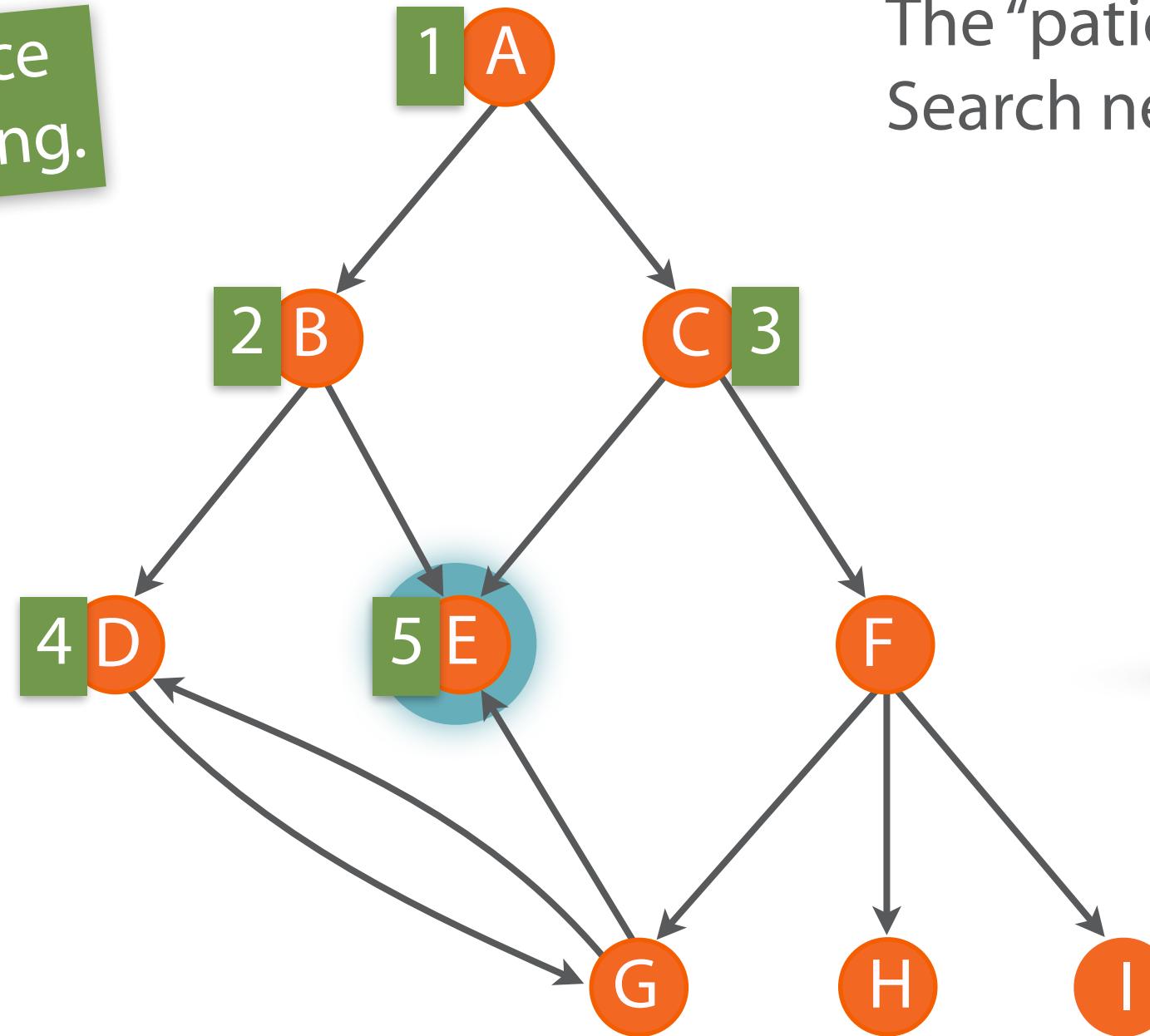
The “patient” strategy:  
Search nearest nodes first.



# Breadth First Search (BFS)

Explore all short-distance options before proceeding.

Find distance to starting point node.  
(Routing)



The “patient” strategy:  
Search nearest nodes first.

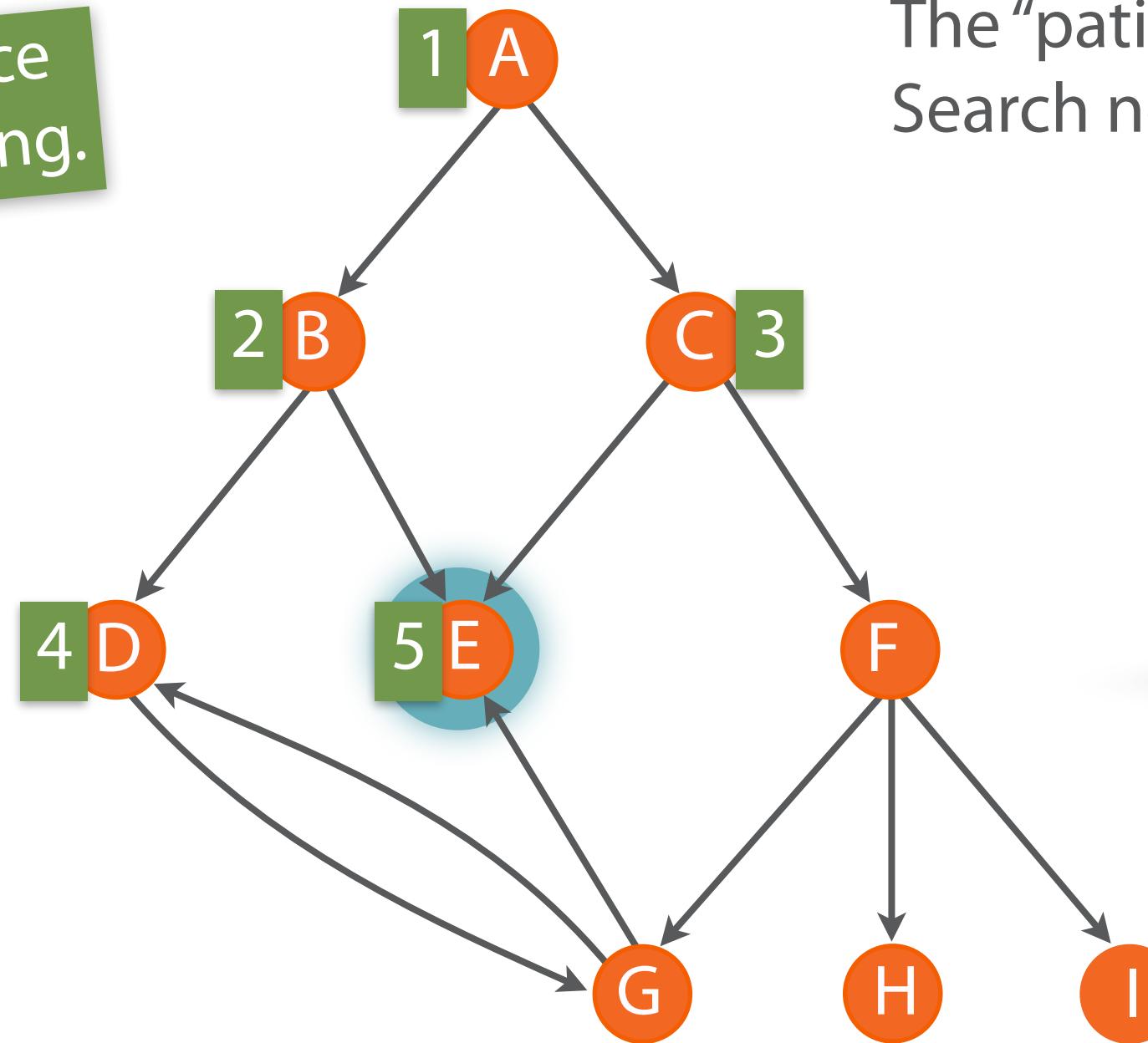


# Breadth First Search (BFS)

Explore all short-distance options before proceeding.

Find distance to starting point node.  
(Routing)

The “patient” strategy:  
Search nearest nodes first.

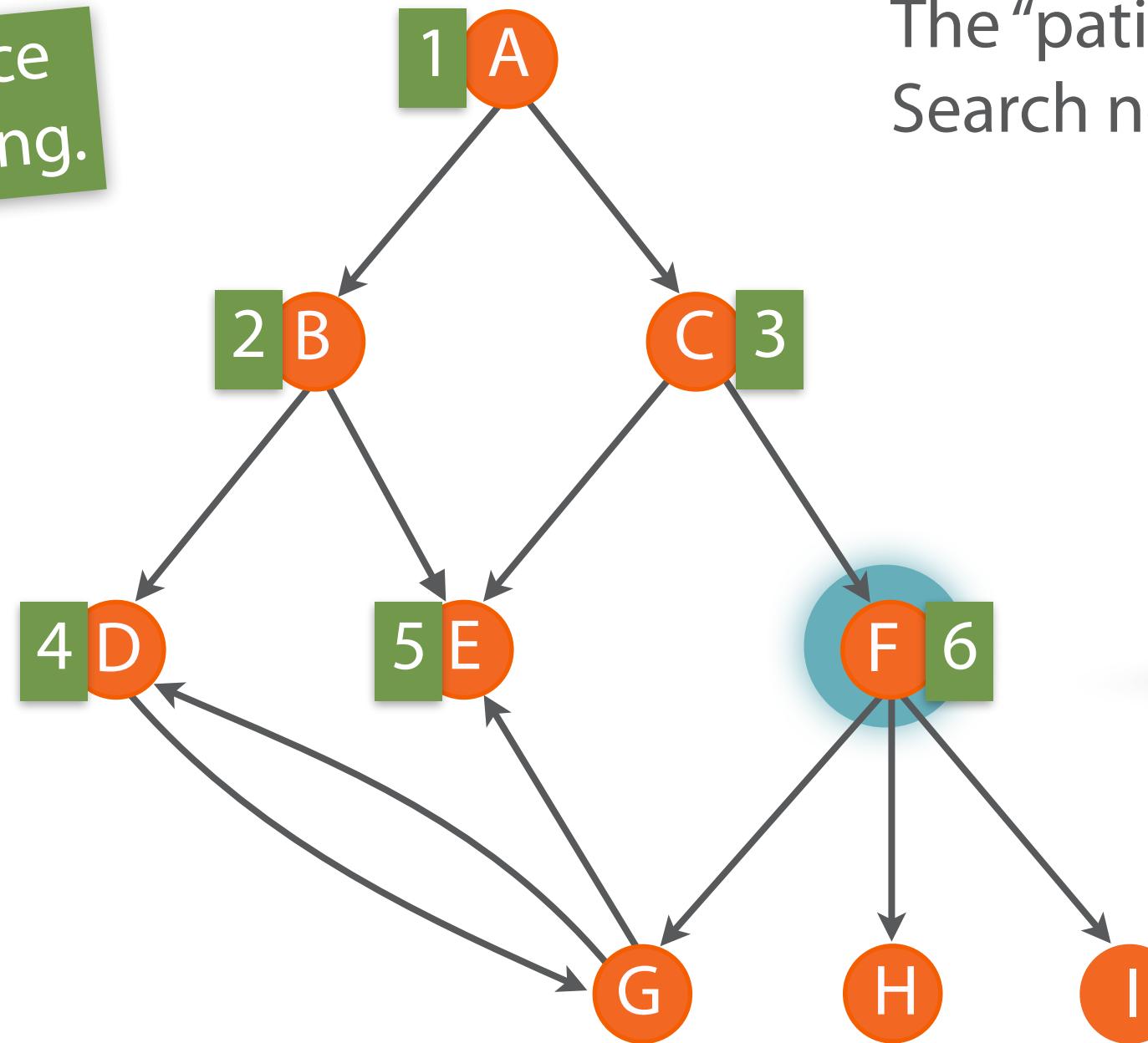


# Breadth First Search (BFS)

Explore all short-distance options before proceeding.

Find distance to starting point node.  
(Routing)

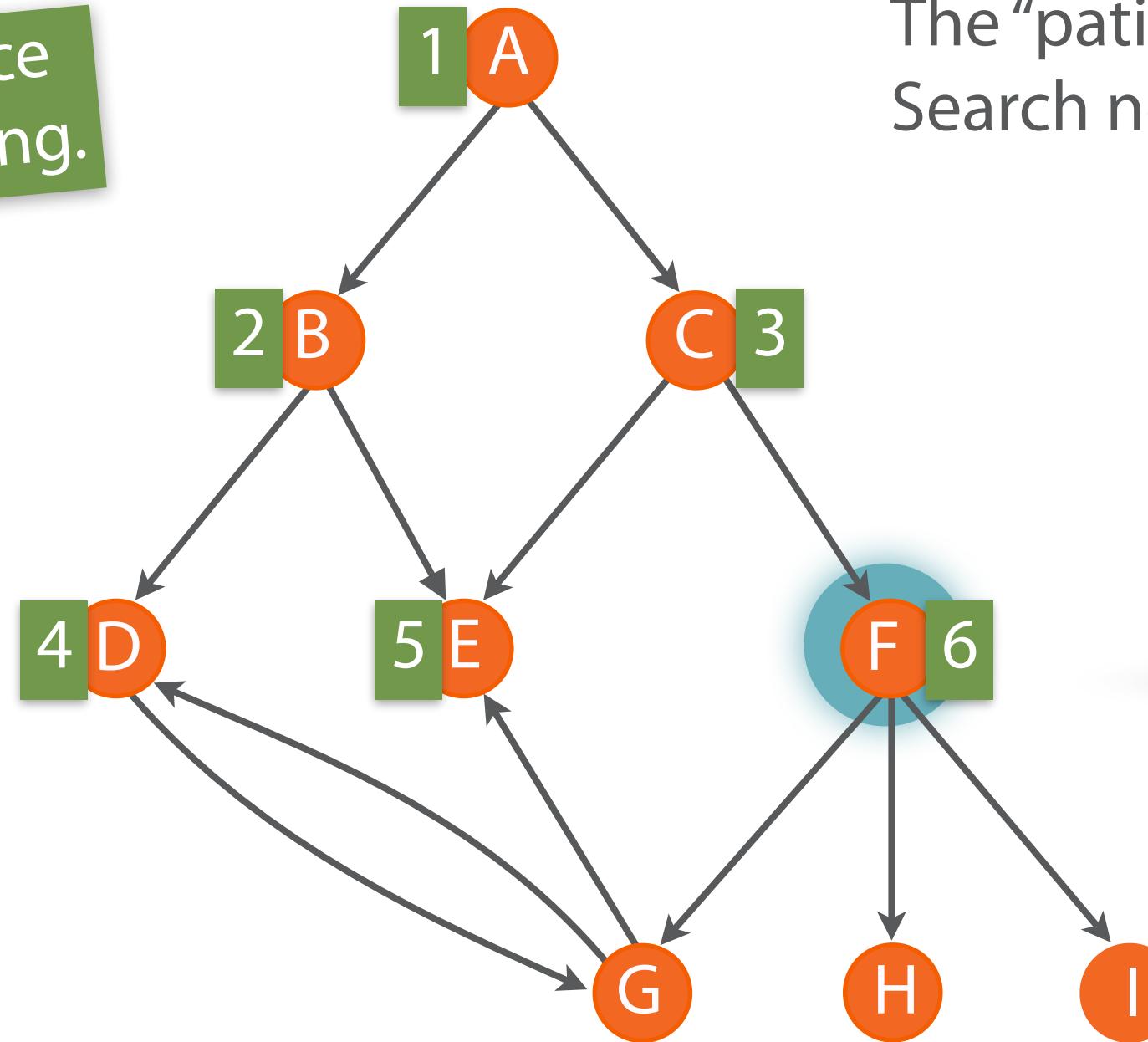
The “patient” strategy:  
Search nearest nodes first.



# Breadth First Search (BFS)

Explore all short-distance options before proceeding.

Find distance to starting point node.  
(Routing)



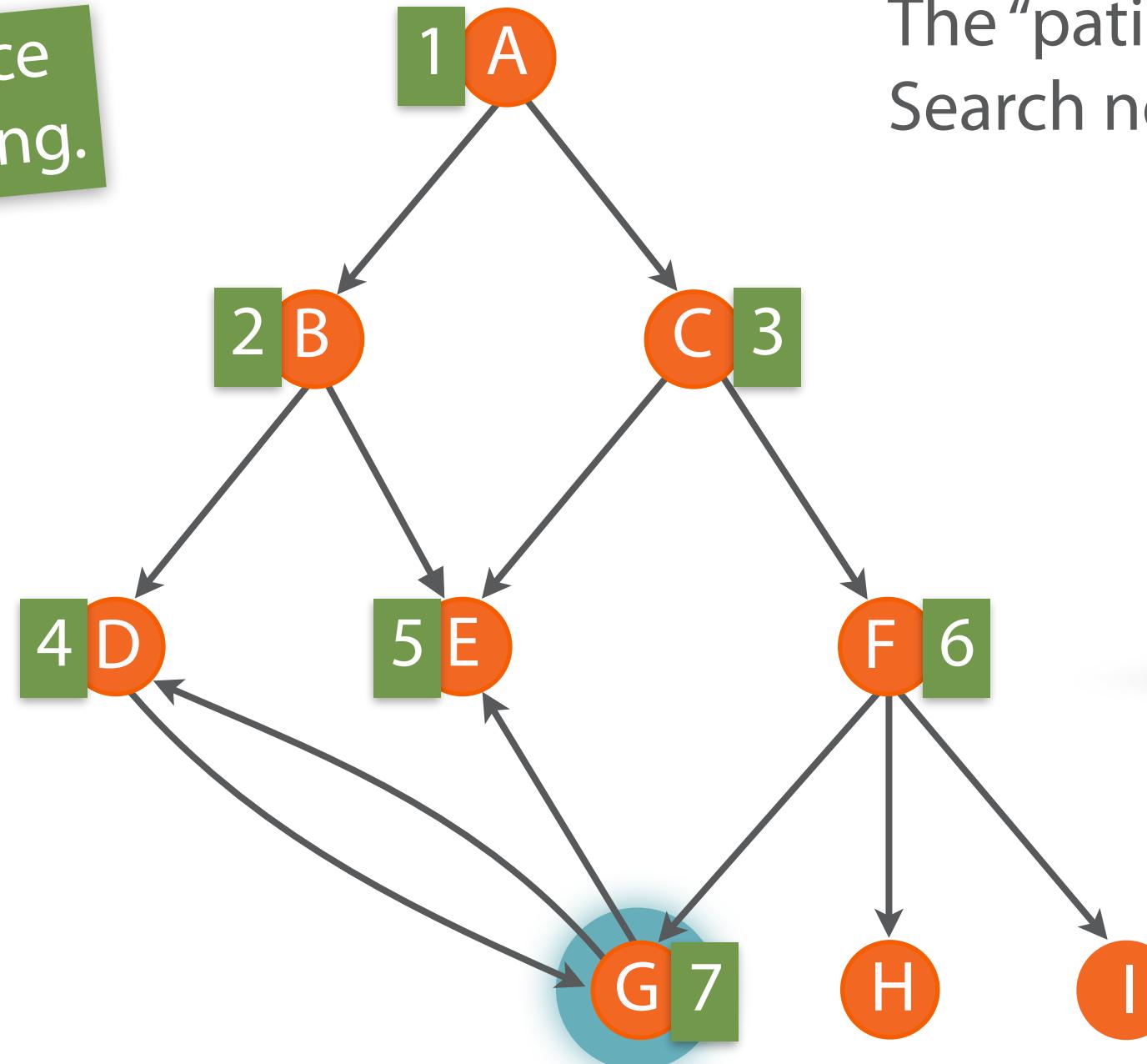
The “patient” strategy:  
Search nearest nodes first.



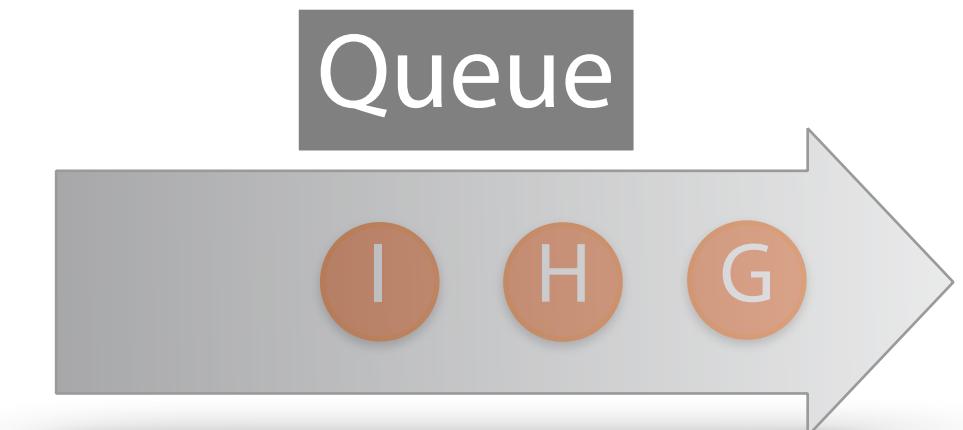
# Breadth First Search (BFS)

Explore all short-distance options before proceeding.

Find distance to starting point node.  
(Routing)



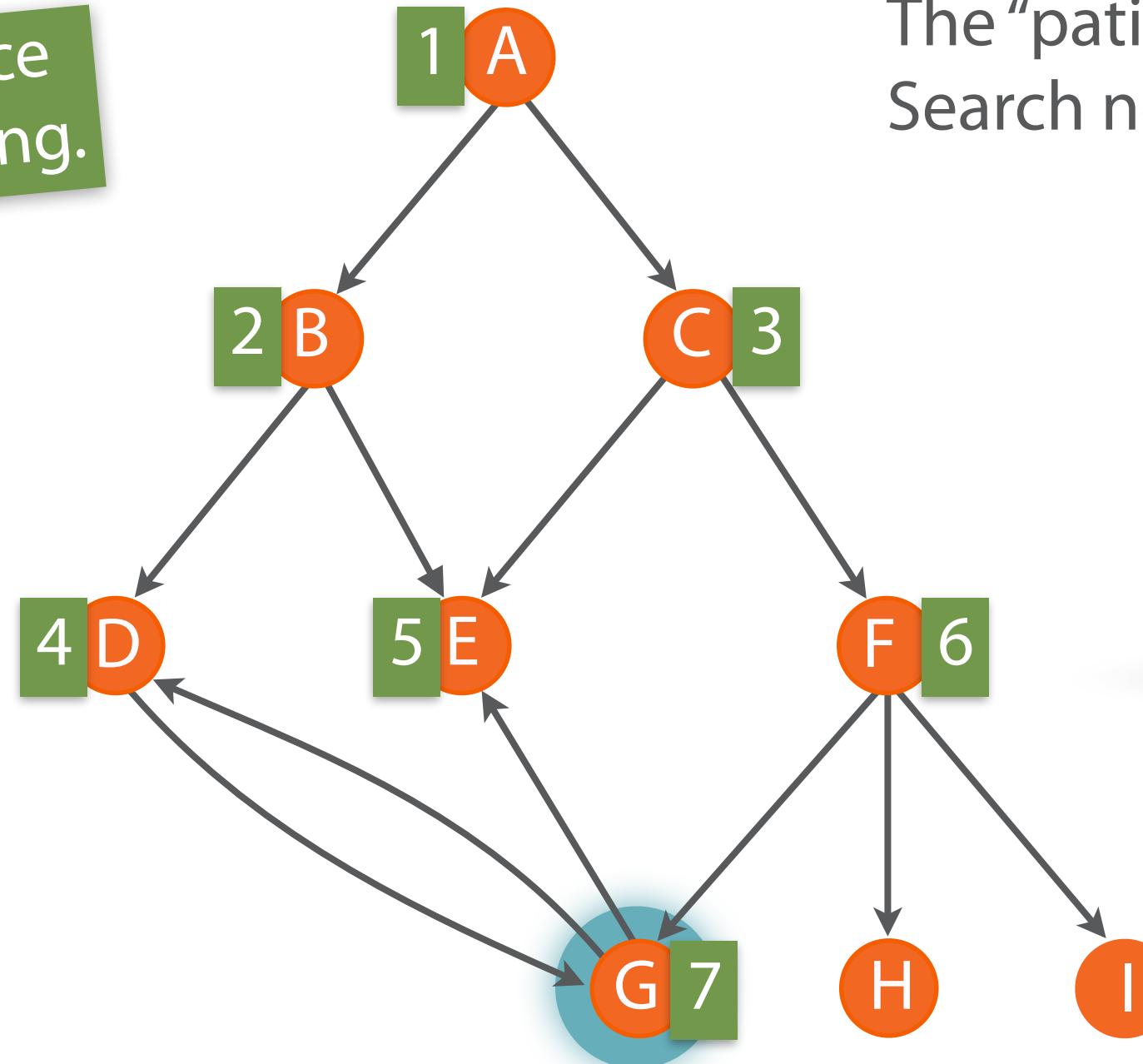
The “patient” strategy:  
Search nearest nodes first.



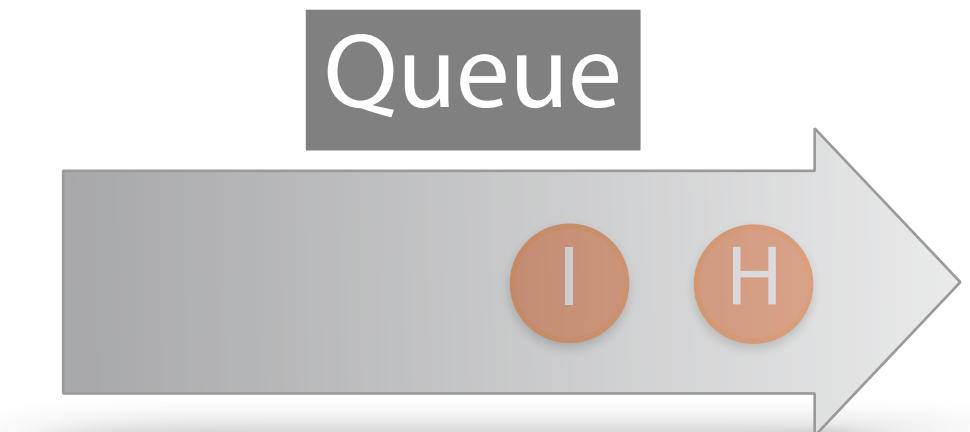
# Breadth First Search (BFS)

Explore all short-distance options before proceeding.

Find distance to starting point node.  
(Routing)



The “patient” strategy:  
Search nearest nodes first.

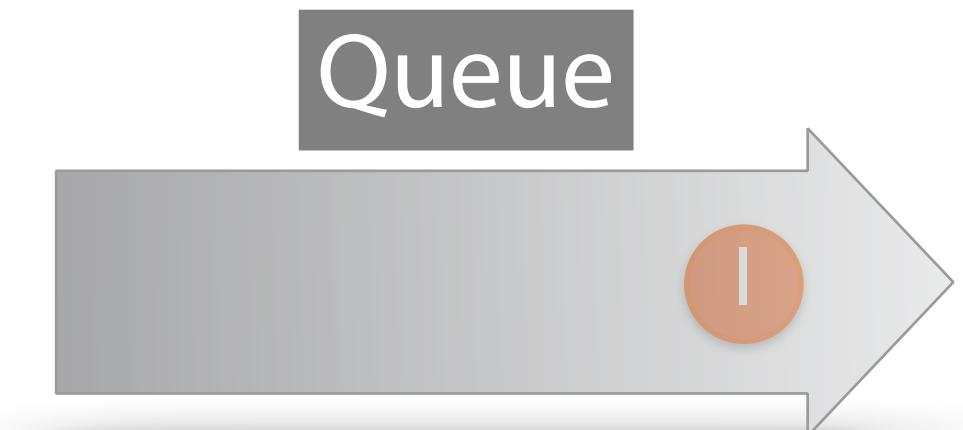
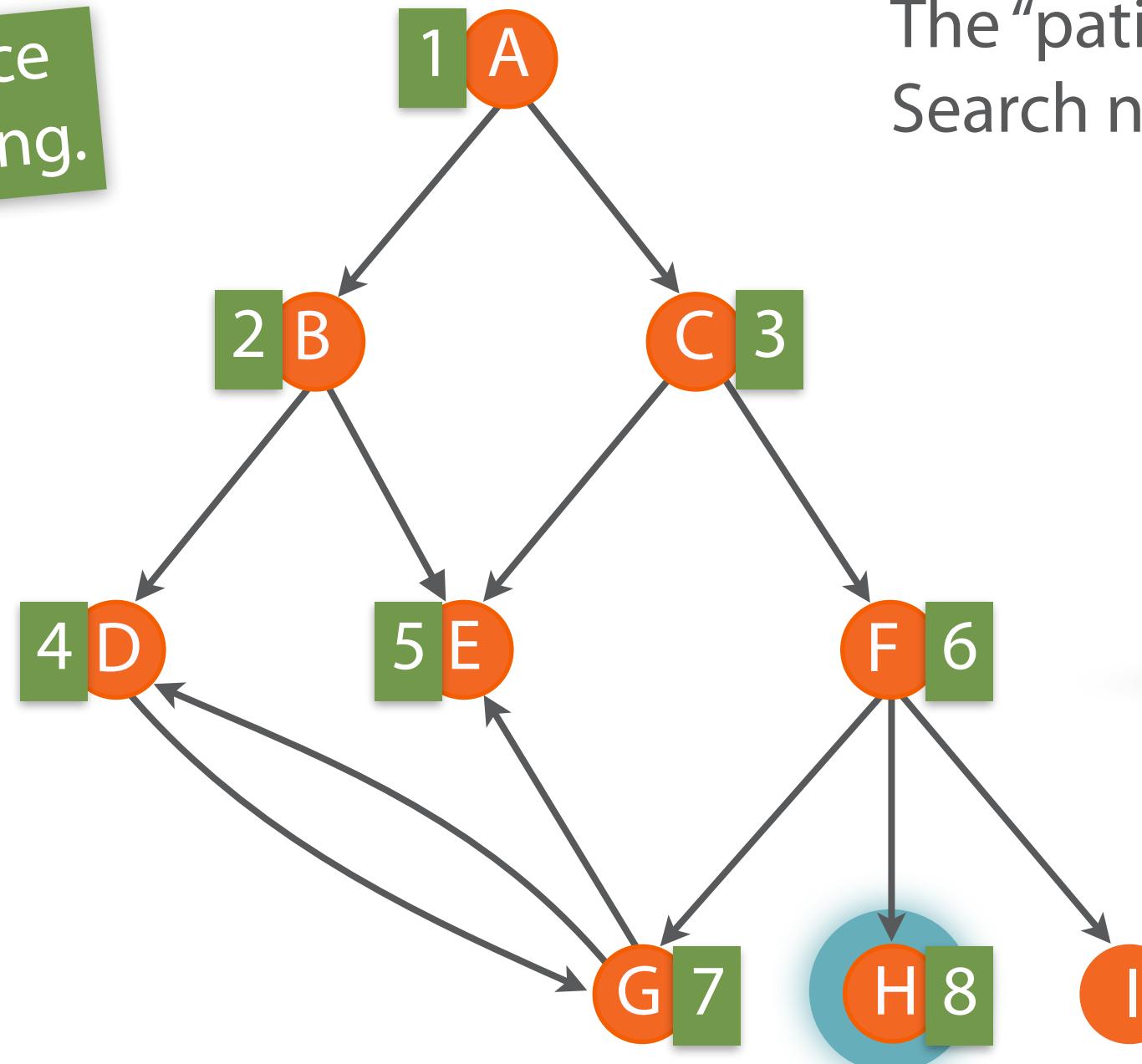


# Breadth First Search (BFS)

Explore all short-distance options before proceeding.

Find distance to starting point node.  
(Routing)

The “patient” strategy:  
Search nearest nodes first.



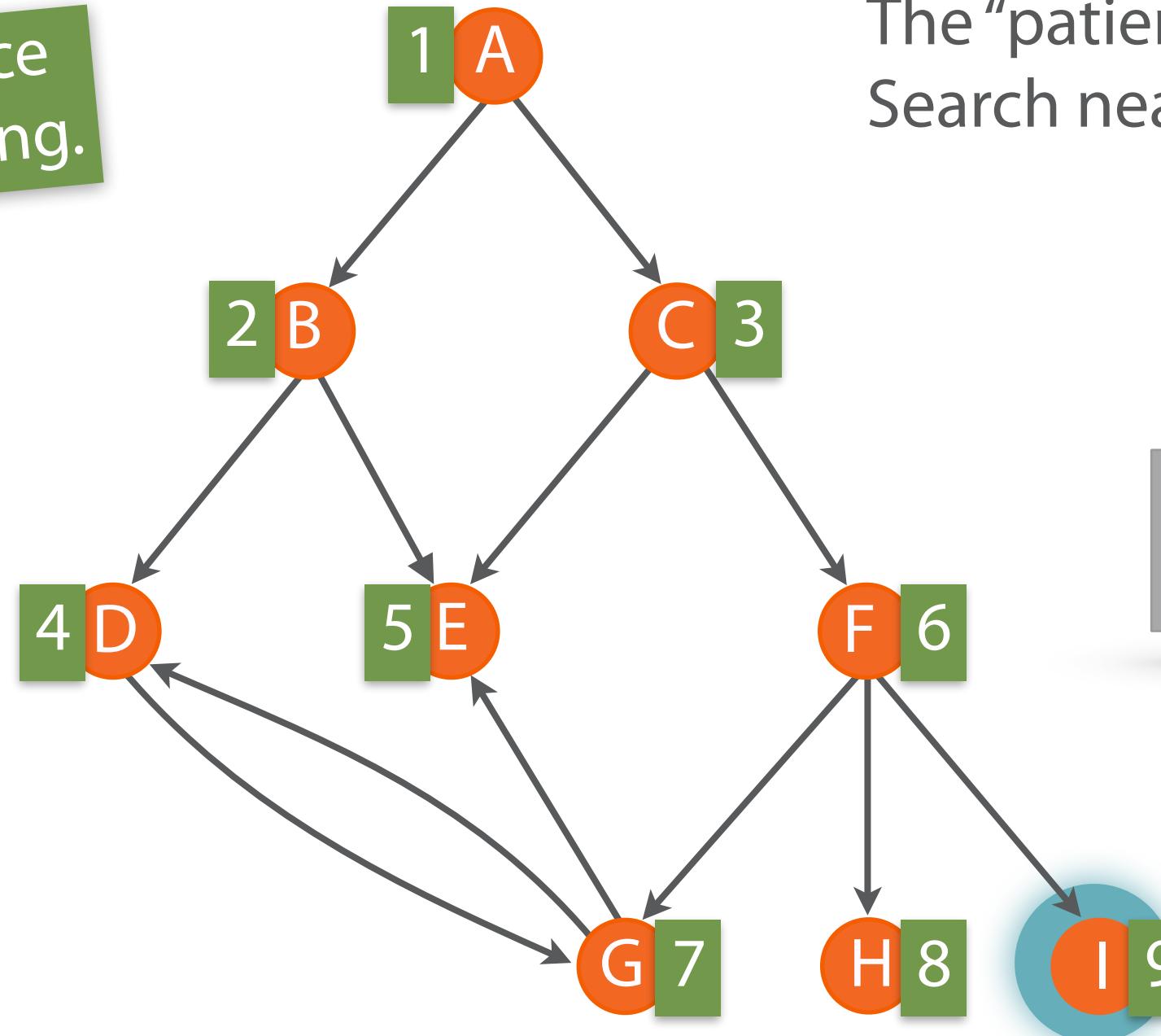
# Breadth First Search (BFS)

Explore all short-distance options before proceeding.

Find distance to starting point node.  
(Routing)

The “patient” strategy:  
Search nearest nodes first.

Queue



# Algorithms

Graph traversal

Brute force  
Greedy algorithms

Divide  
and  
conquer

Dynamic  
programming

Branch  
and  
bound

# Algorithms

Divide  
and  
conquer

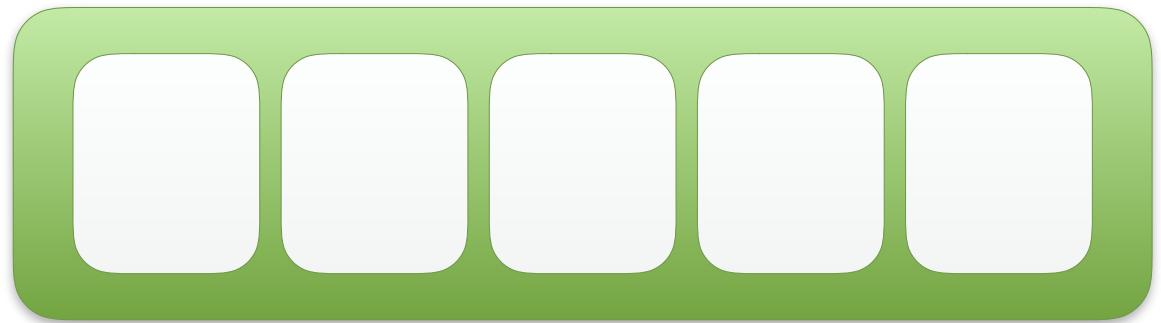
Graph travers

Dynamic  
programming

Brute force  
Greedy algorithms

Branch  
and  
bound

# Brute force



# Brute force



# Brute force



# Brute force



# Brute force

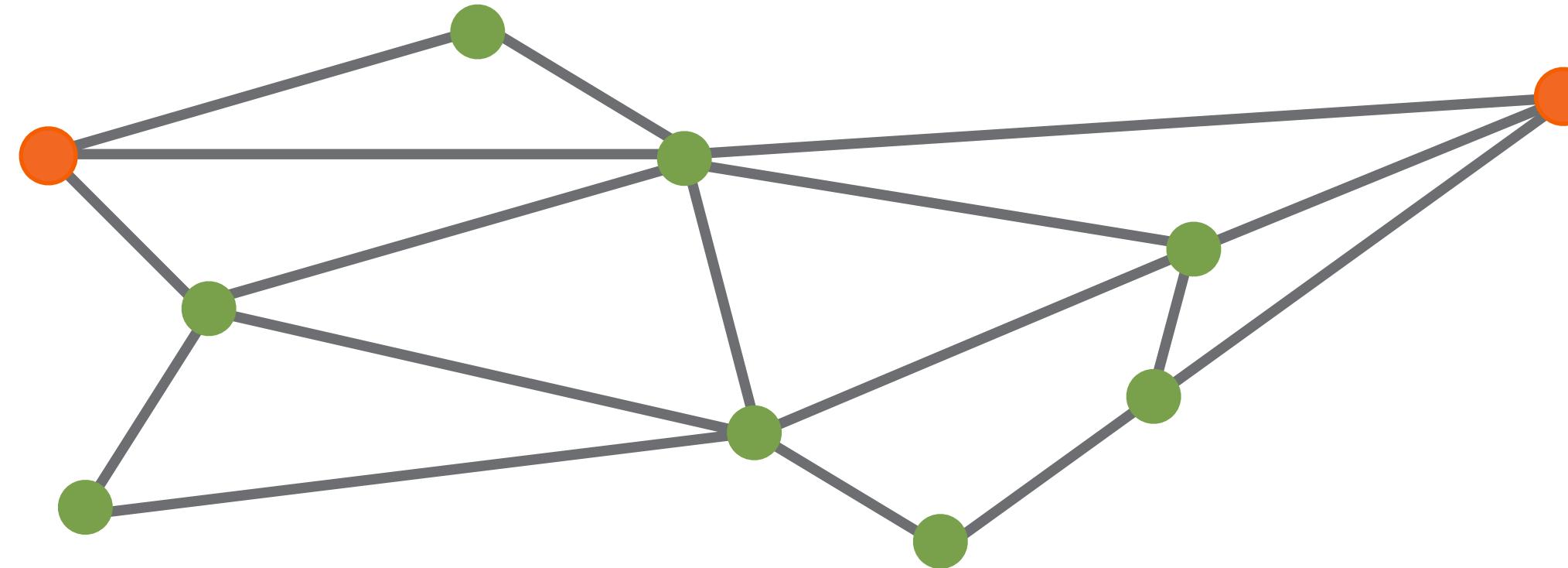


$10 \cdot 10 \cdot 10 \cdot 10 \cdot 10 = 100,000$  combinations

# Brute force

0 0 0 0 2

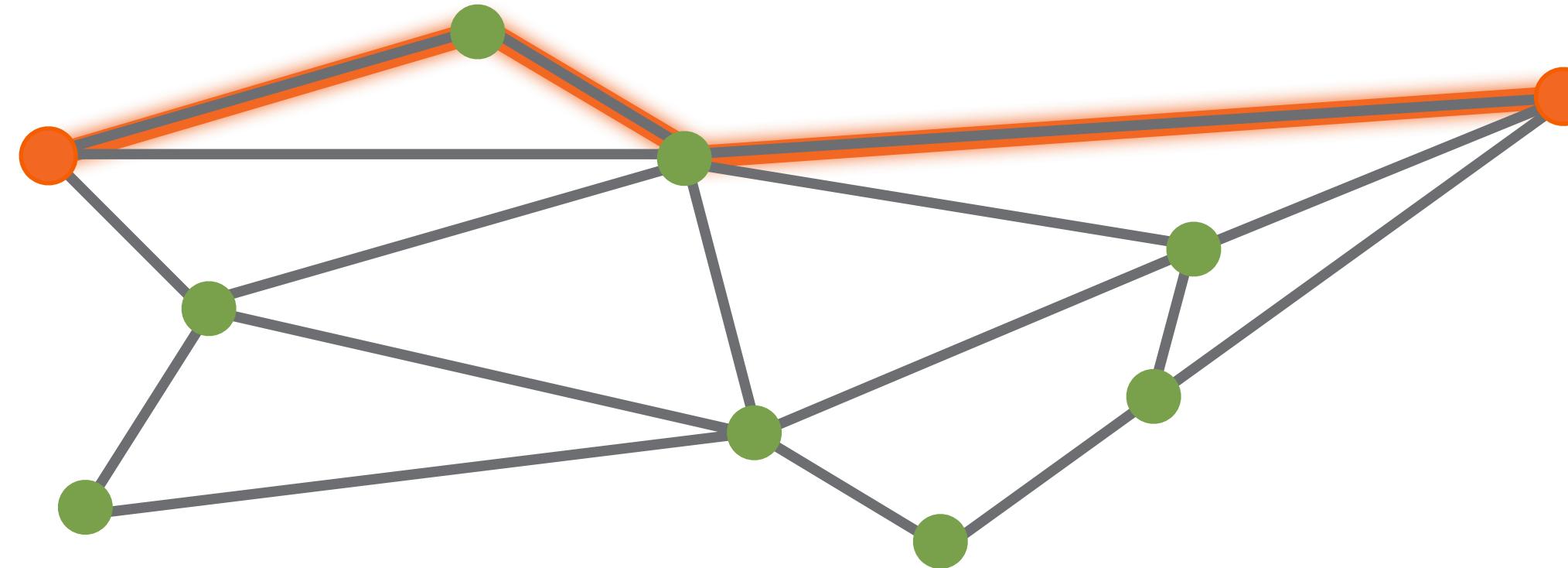
$10 \cdot 10 \cdot 10 \cdot 10 \cdot 10 = 100,000$  combinations



# Brute force

0 0 0 0 2

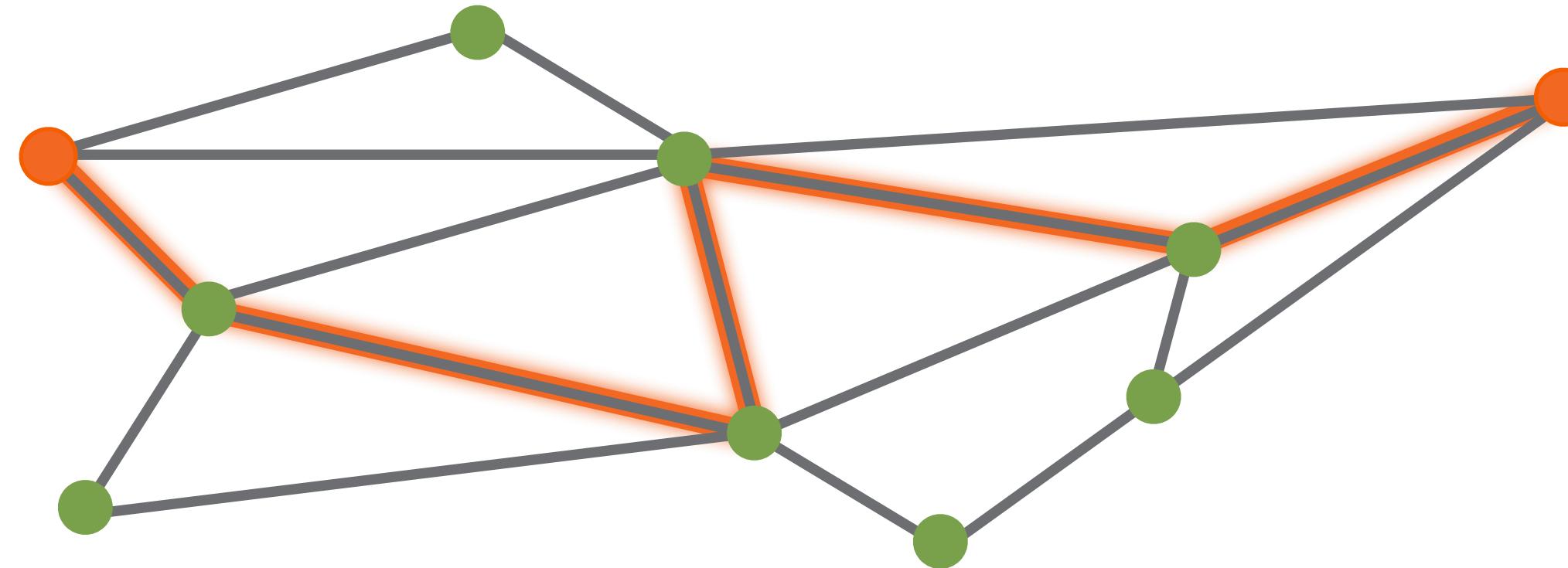
$$10 \cdot 10 \cdot 10 \cdot 10 \cdot 10 = 100,000 \text{ combinations}$$



# Brute force



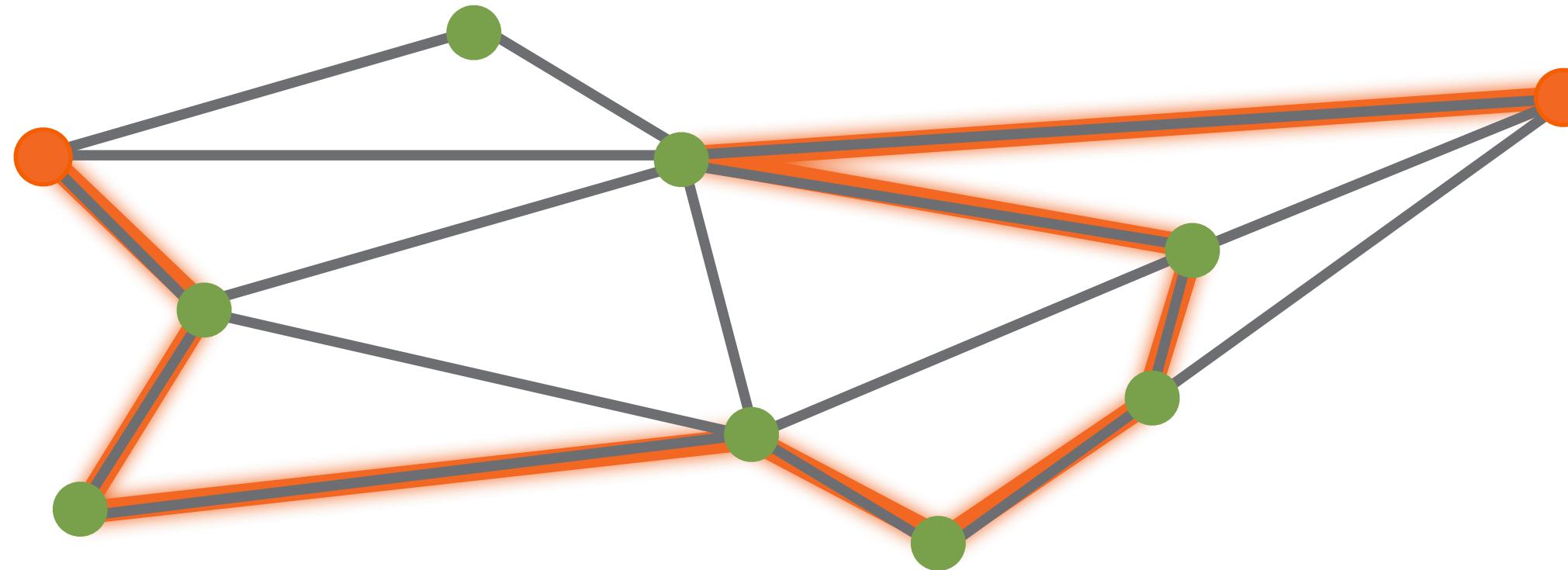
$10 \cdot 10 \cdot 10 \cdot 10 \cdot 10 = 100,000$  combinations



# Brute force

0 0 0 0 2

$$10 \cdot 10 \cdot 10 \cdot 10 \cdot 10 = 100,000 \text{ combinations}$$

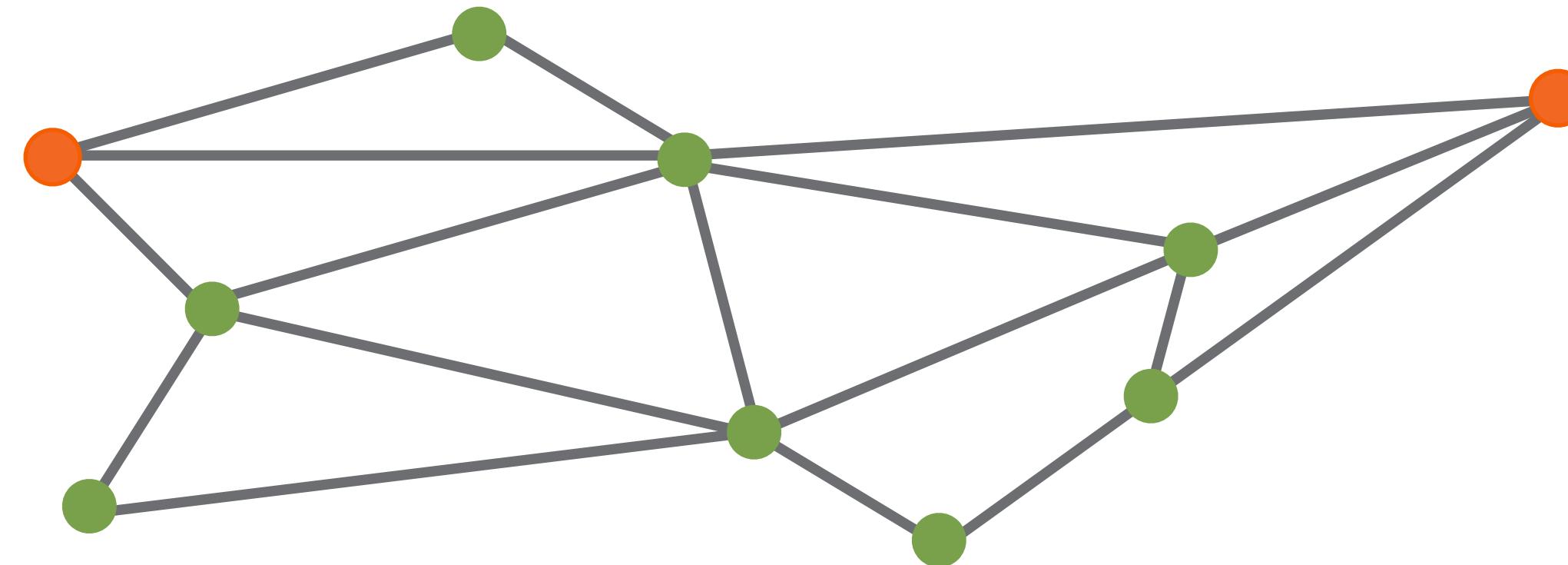


# Brute force

Generate all possible  
solution candidates

0 0 0 0 2

$$10 \cdot 10 \cdot 10 \cdot 10 \cdot 10 = 100,000 \text{ combinations}$$



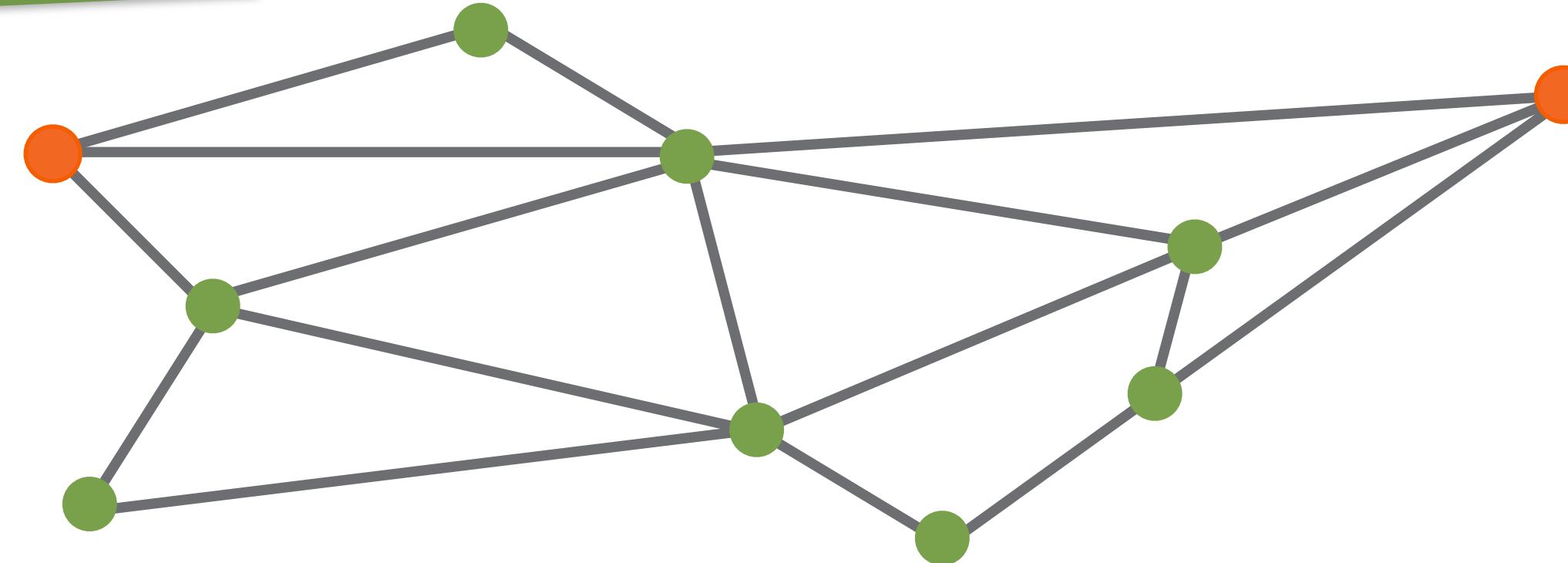
# Brute force

Generate all possible  
solution candidates

0 0 0 0 2

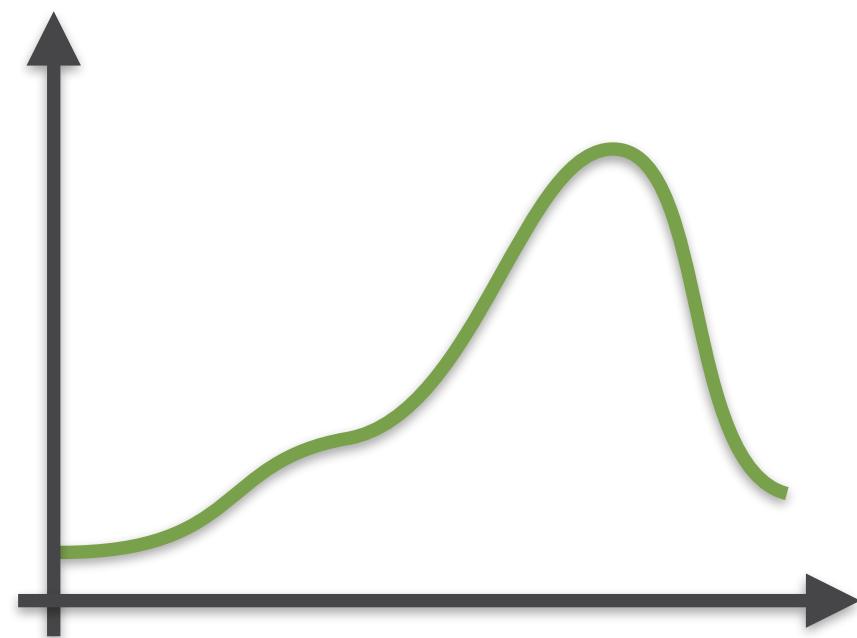
Check each candidate

$$10 \cdot 10 \cdot 10 \cdot 10 \cdot 10 = 100,000 \text{ combinations}$$

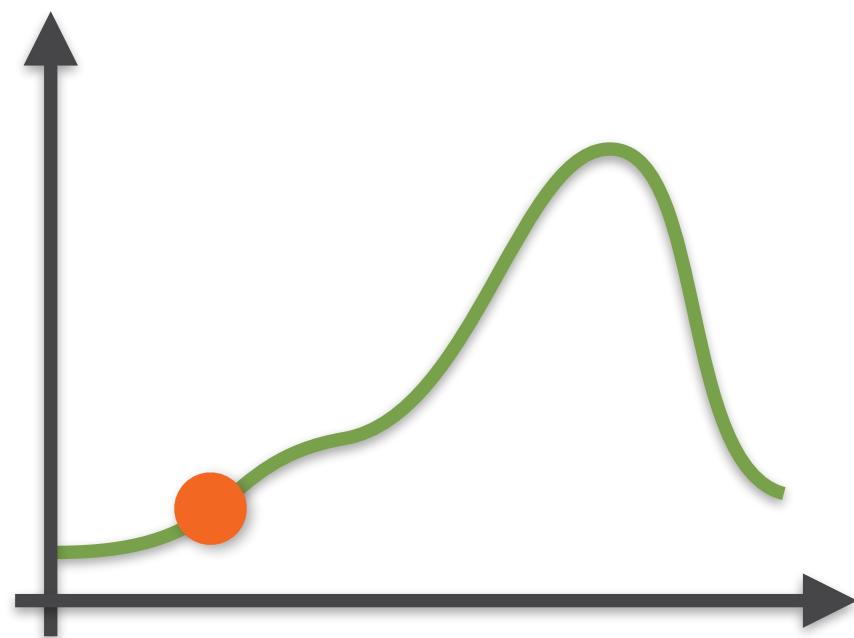


# Greedy Algorithms

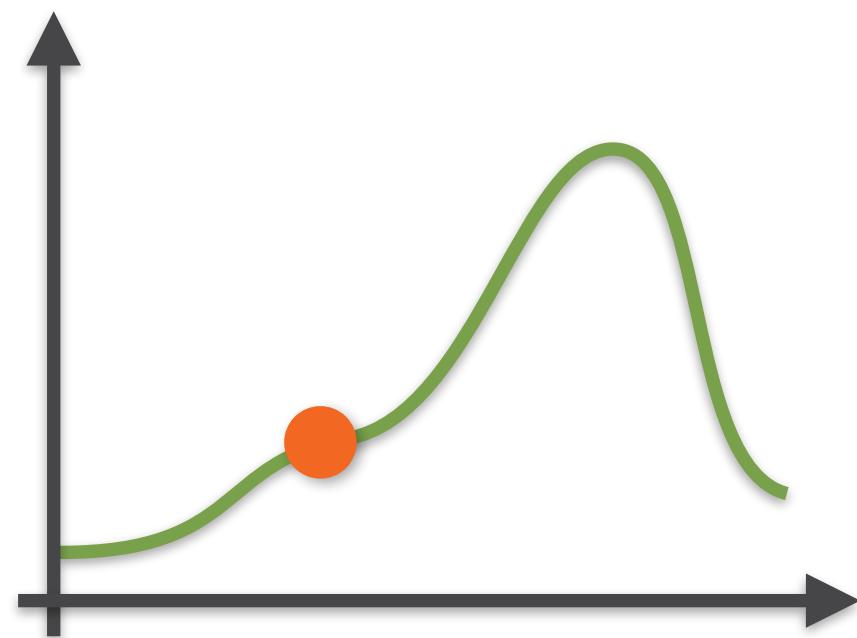
# Greedy Algorithms



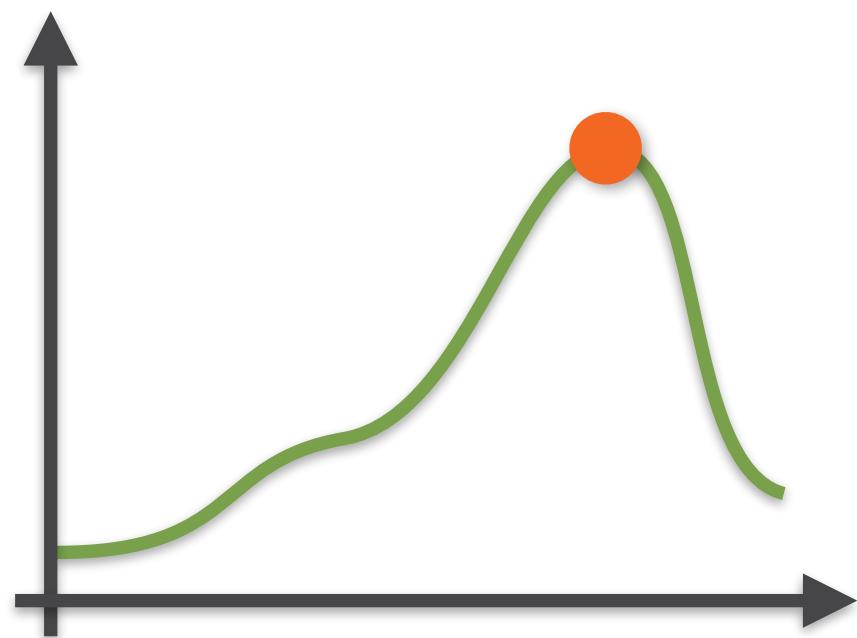
# Greedy Algorithms



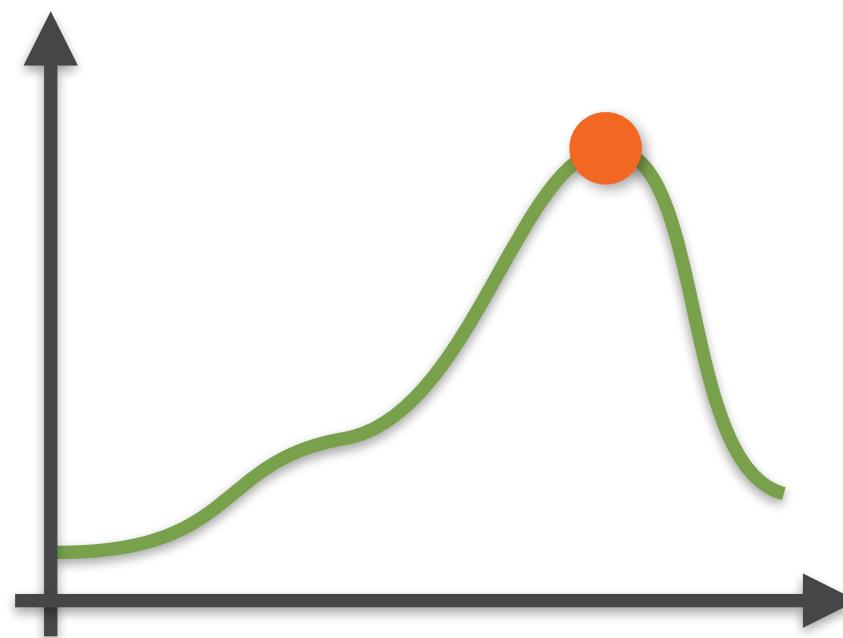
# Greedy Algorithms



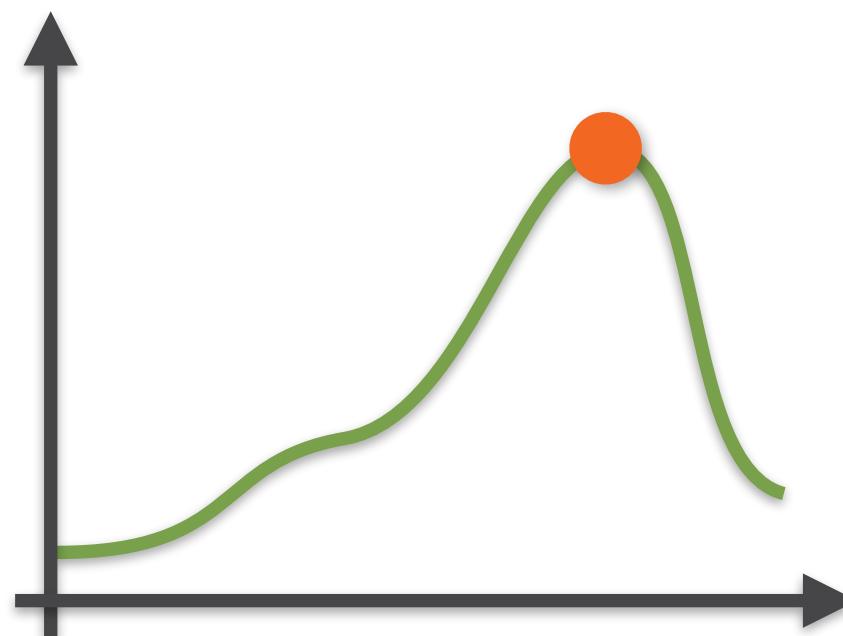
# Greedy Algorithms



# Greedy Algorithms

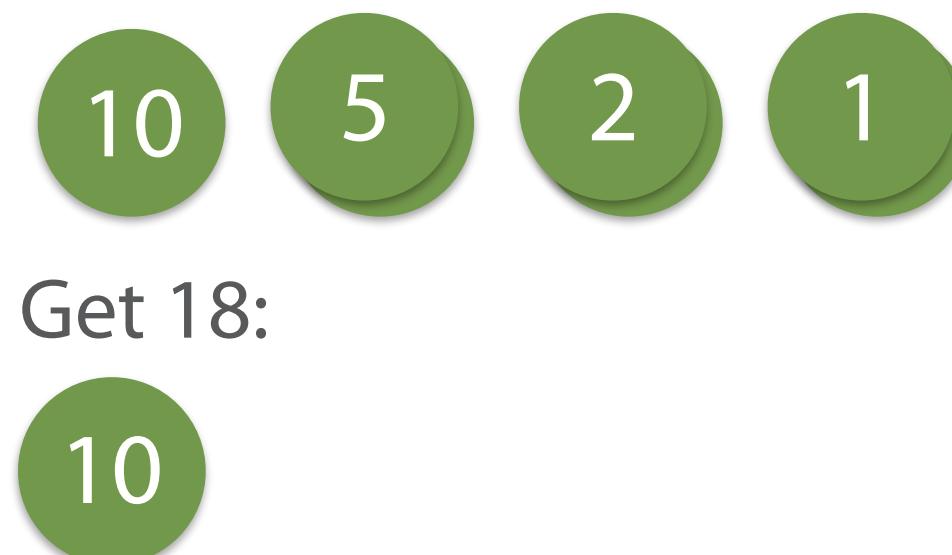
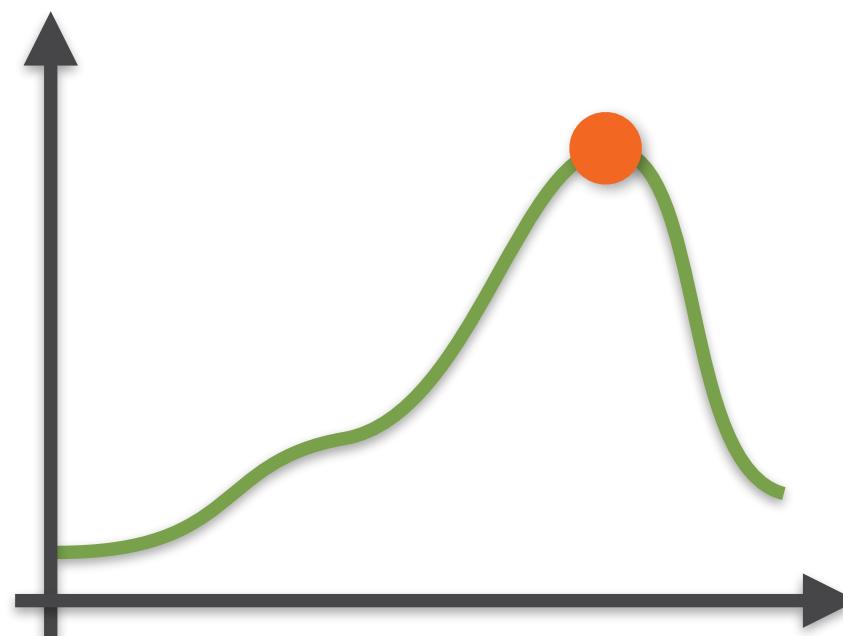


# Greedy Algorithms

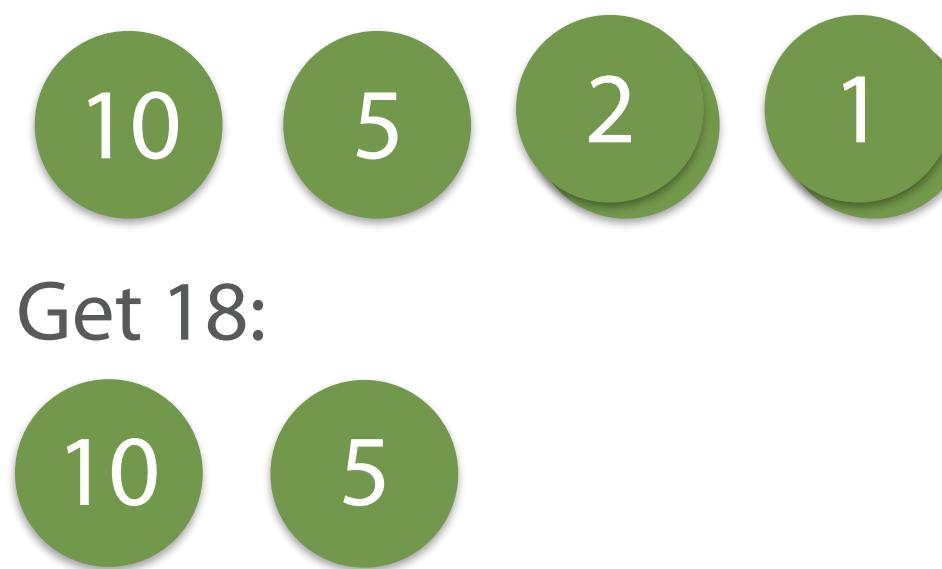
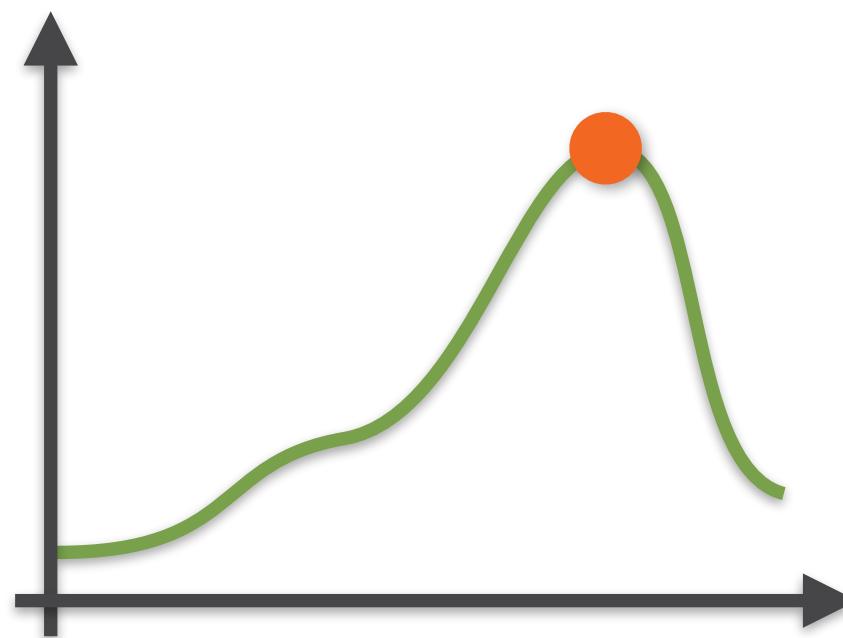


Get 18:

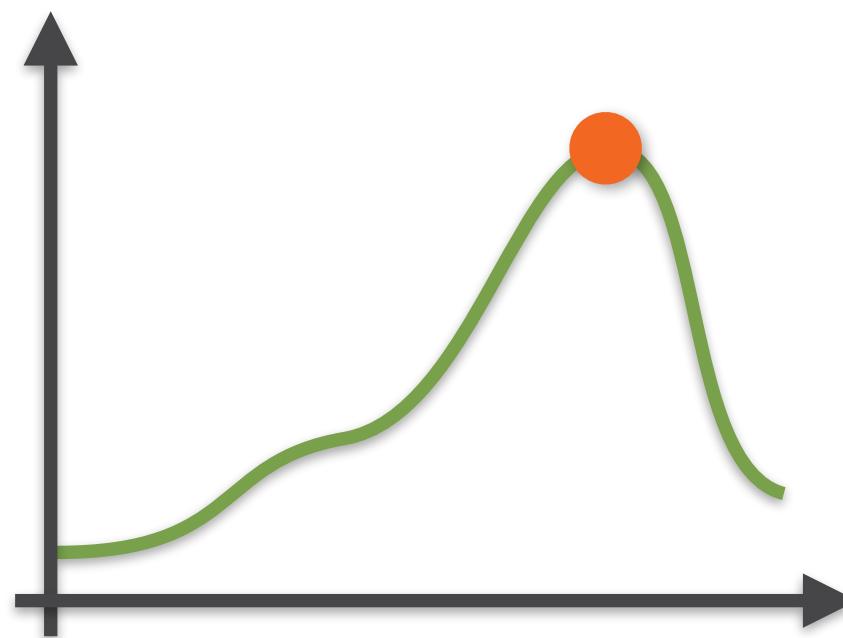
# Greedy Algorithms



# Greedy Algorithms



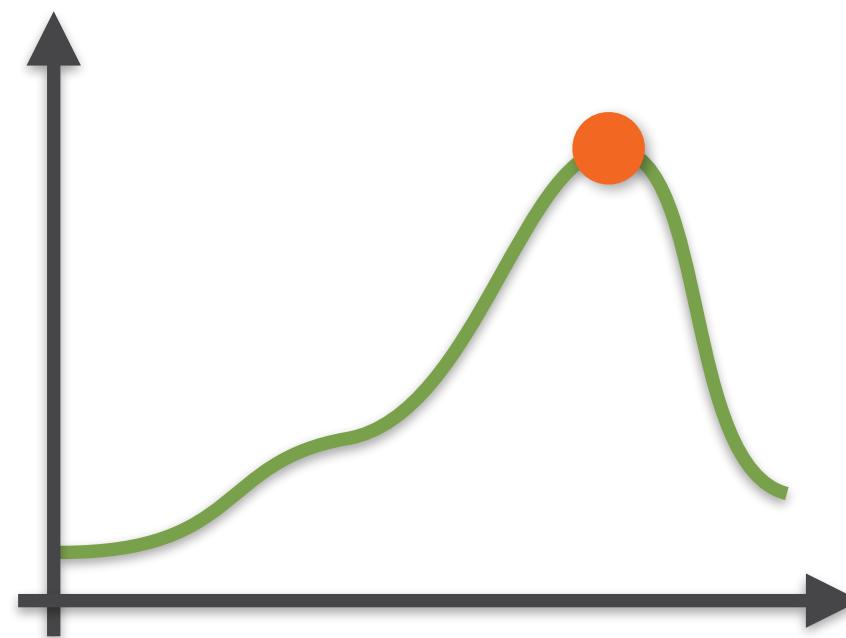
# Greedy Algorithms



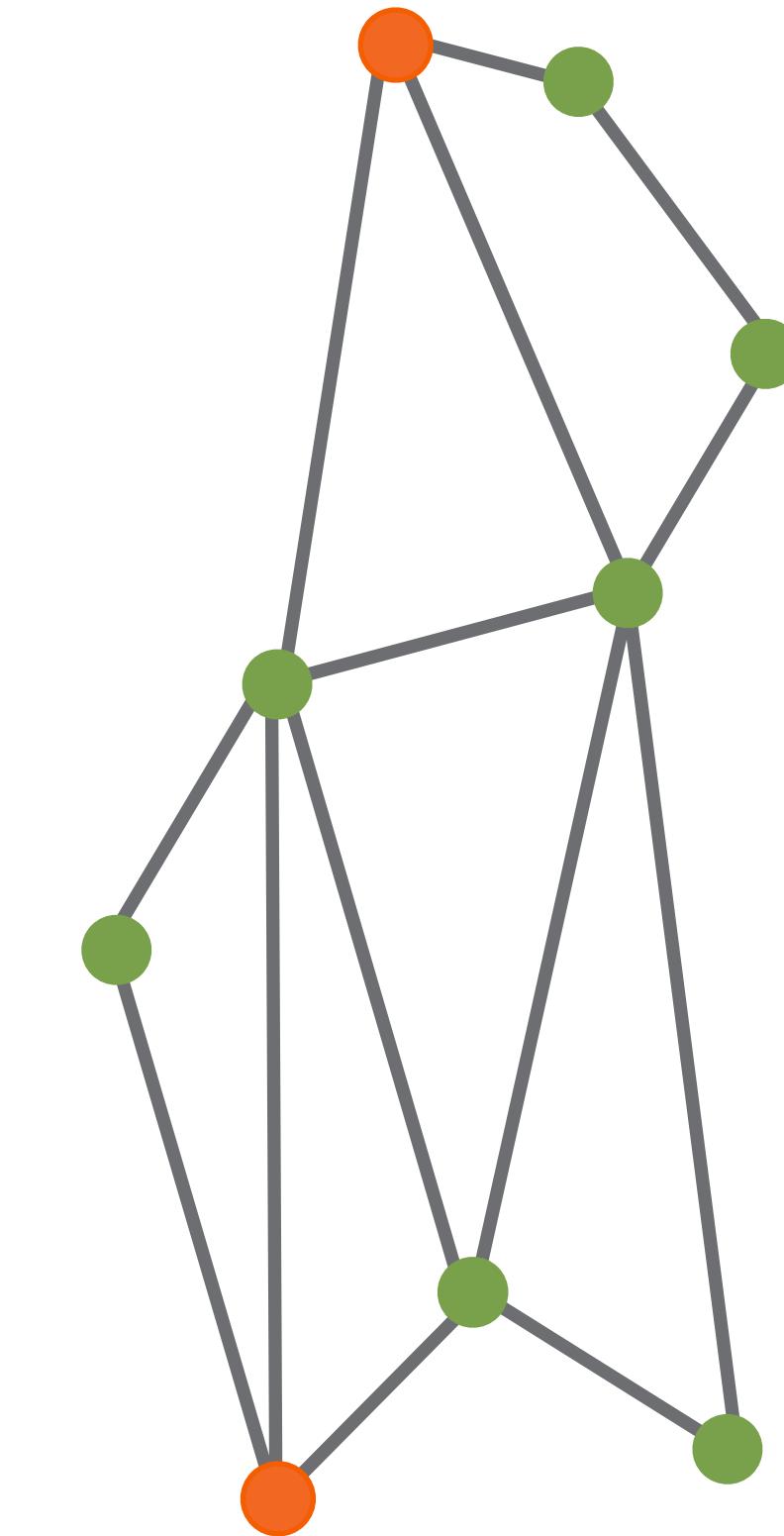
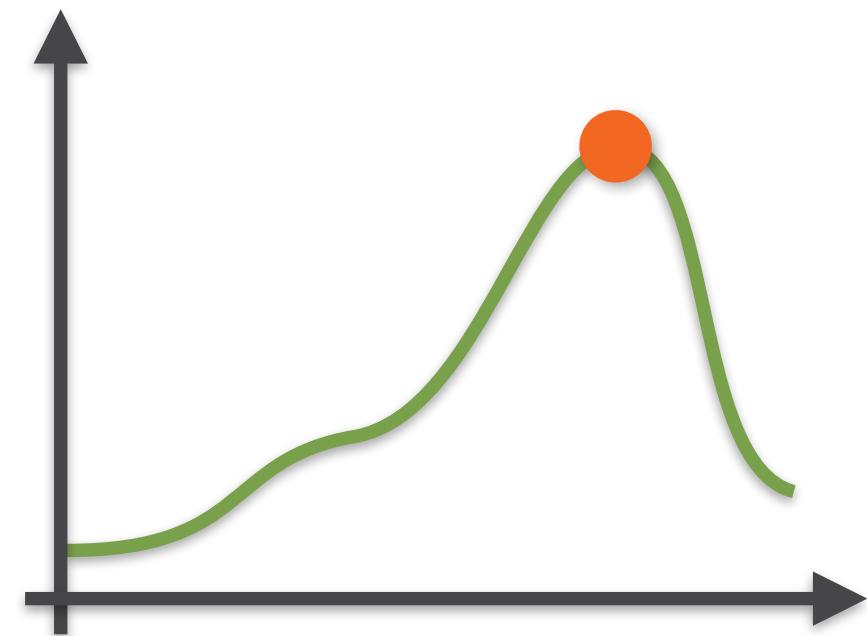
Get 18:



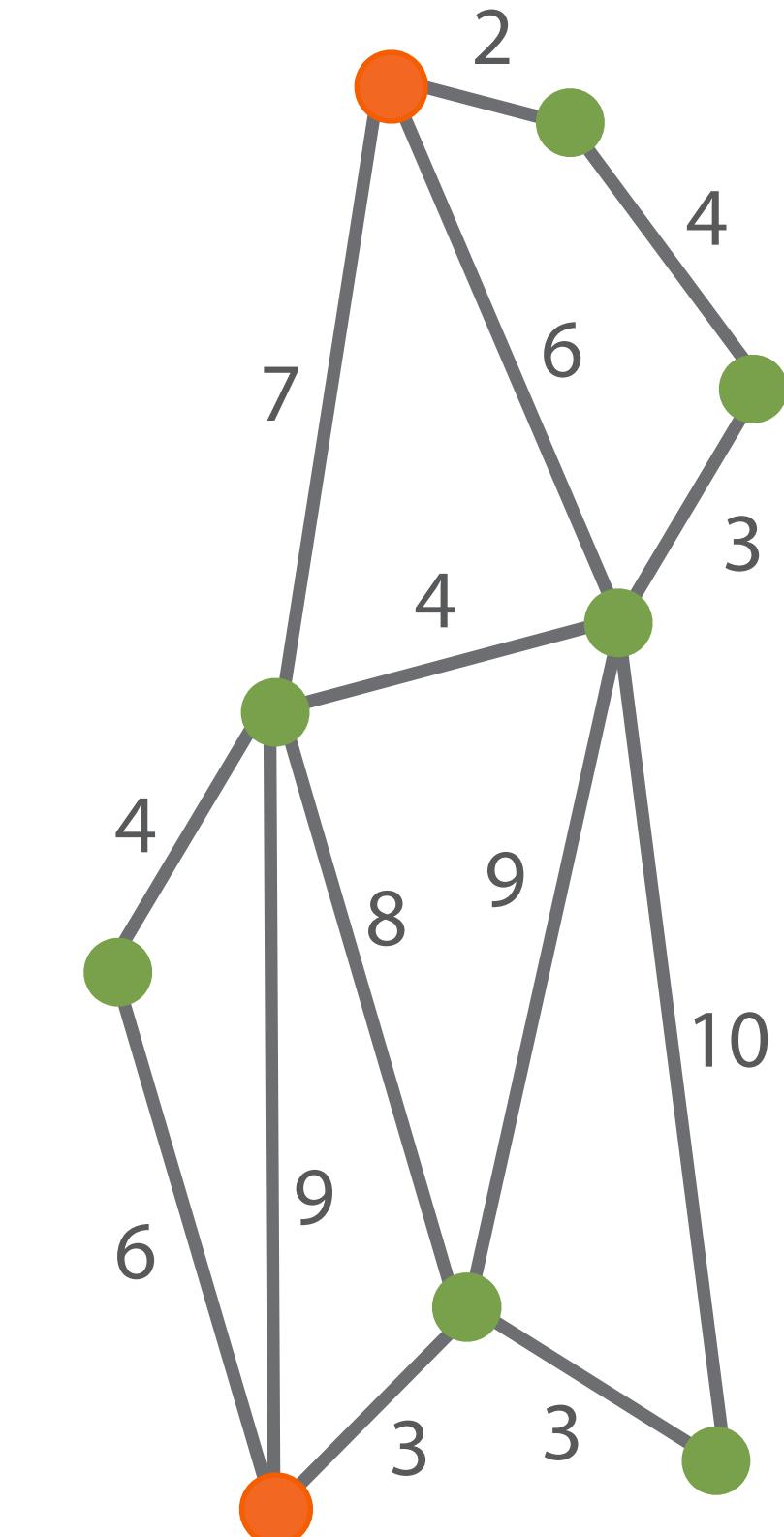
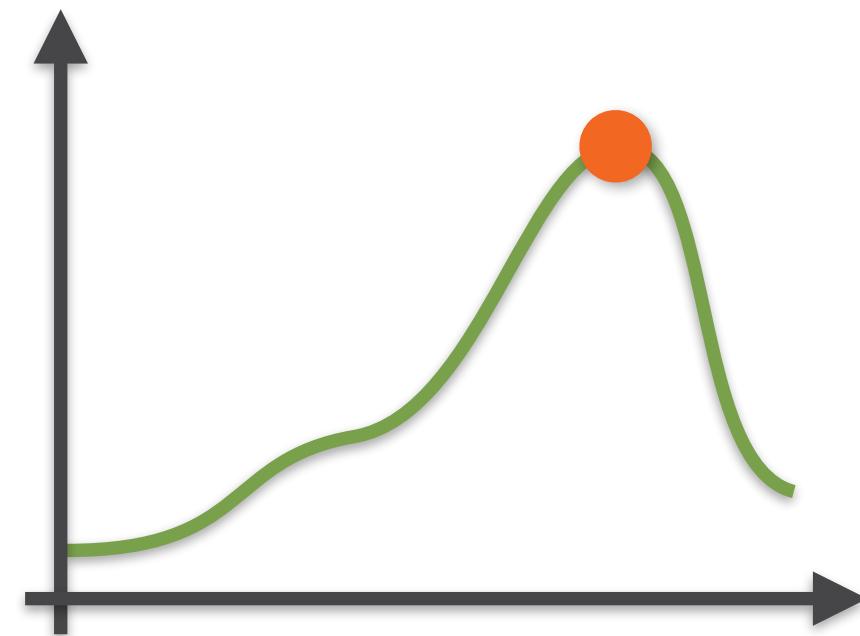
# Greedy Algorithms



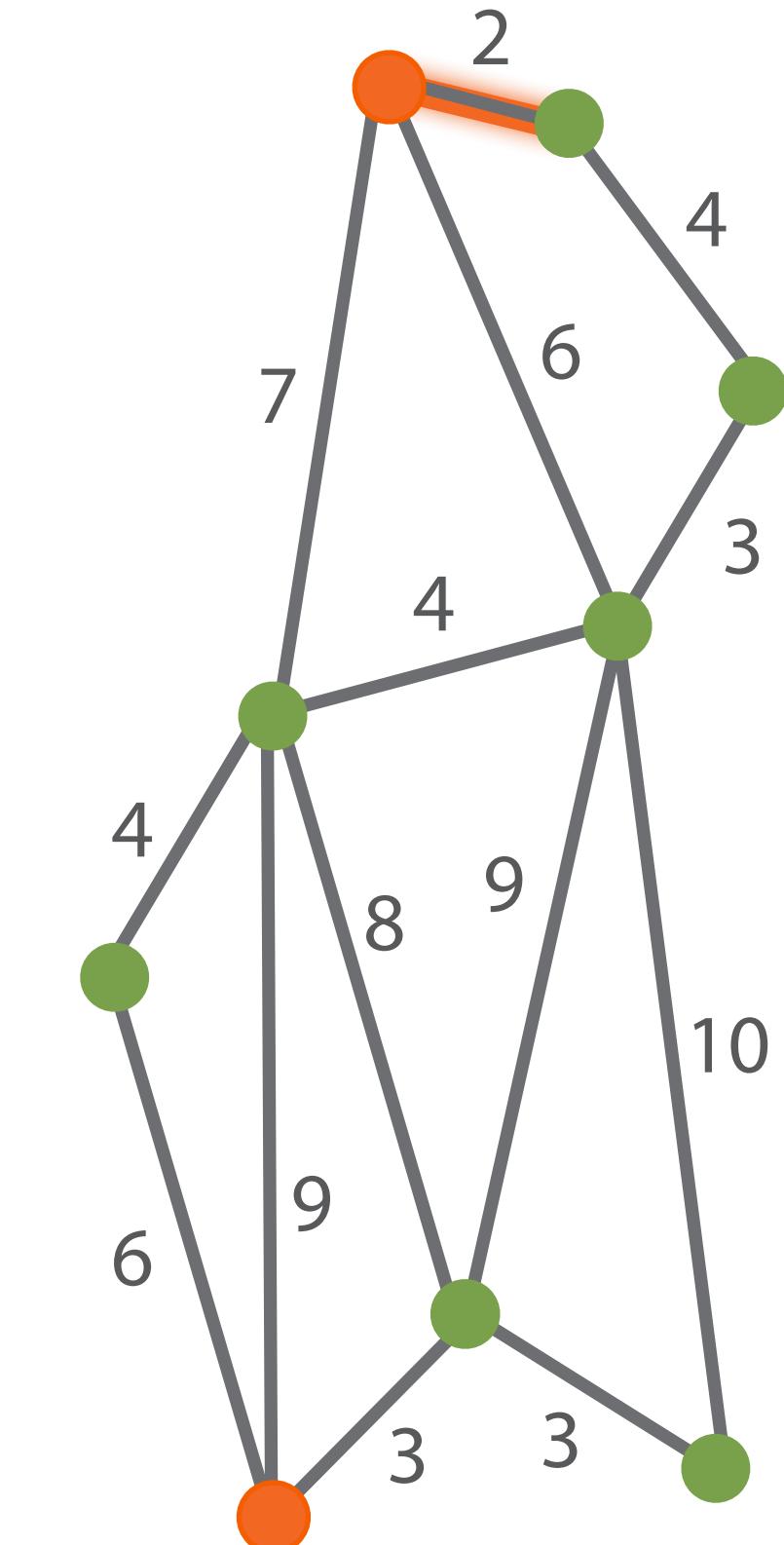
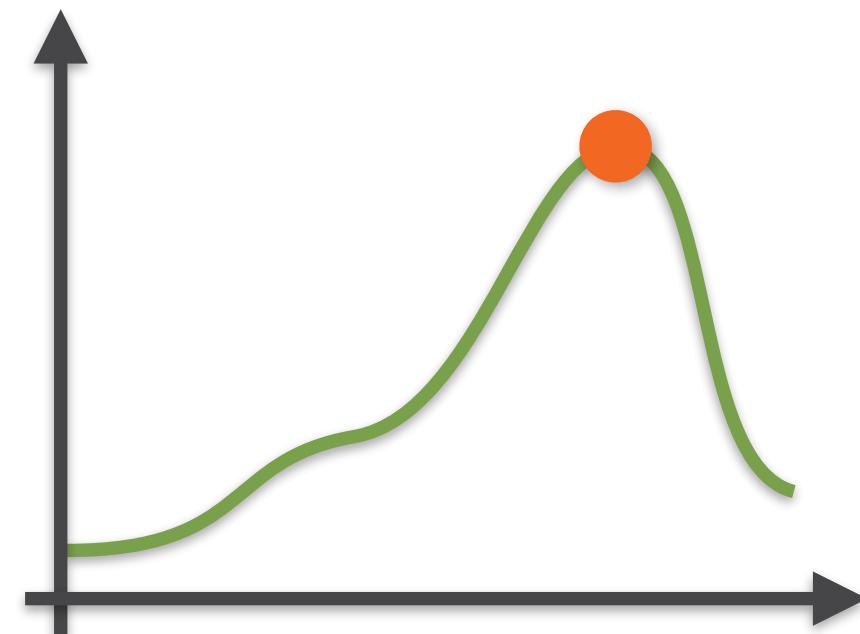
# Greedy Algorithms



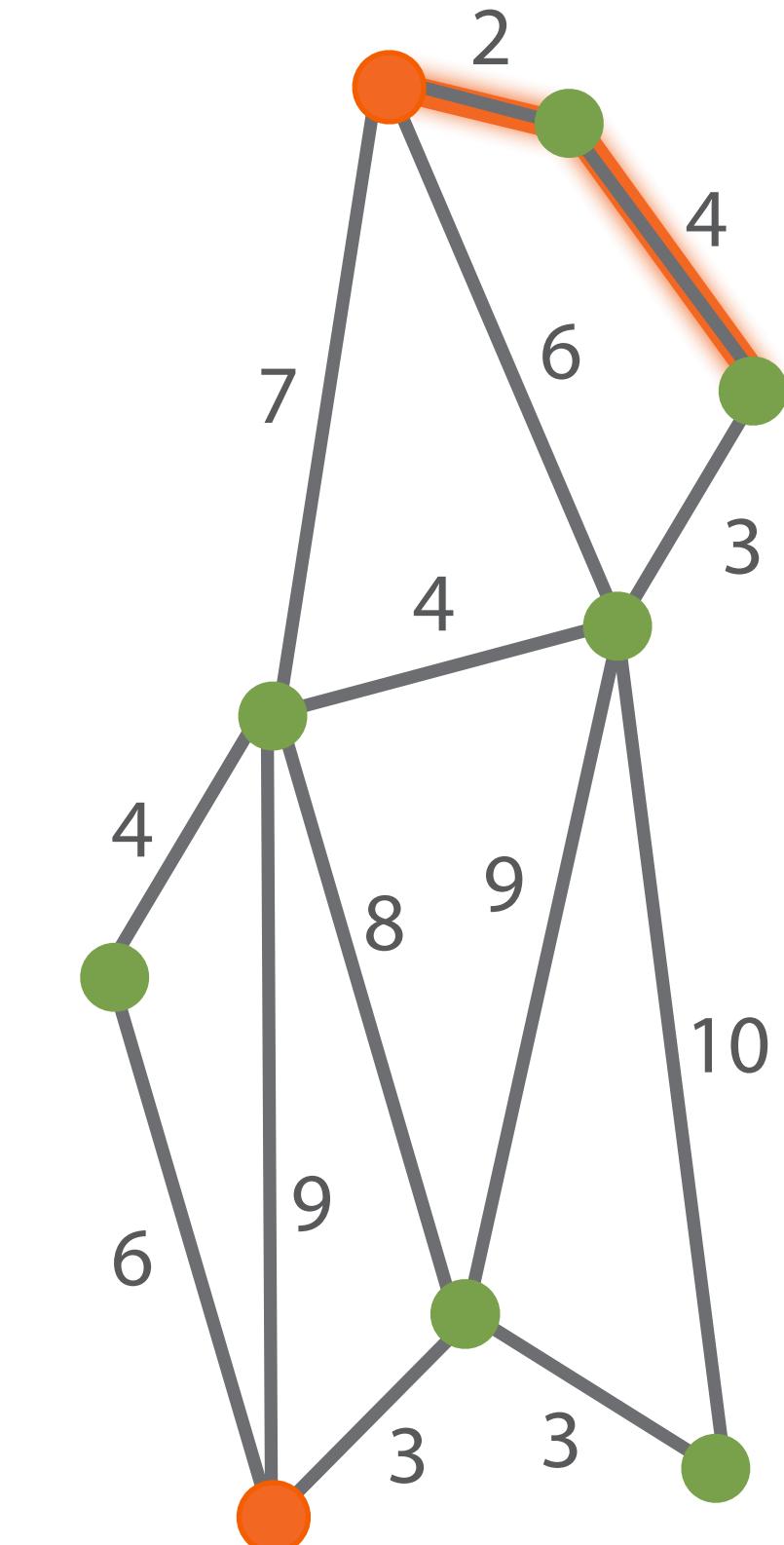
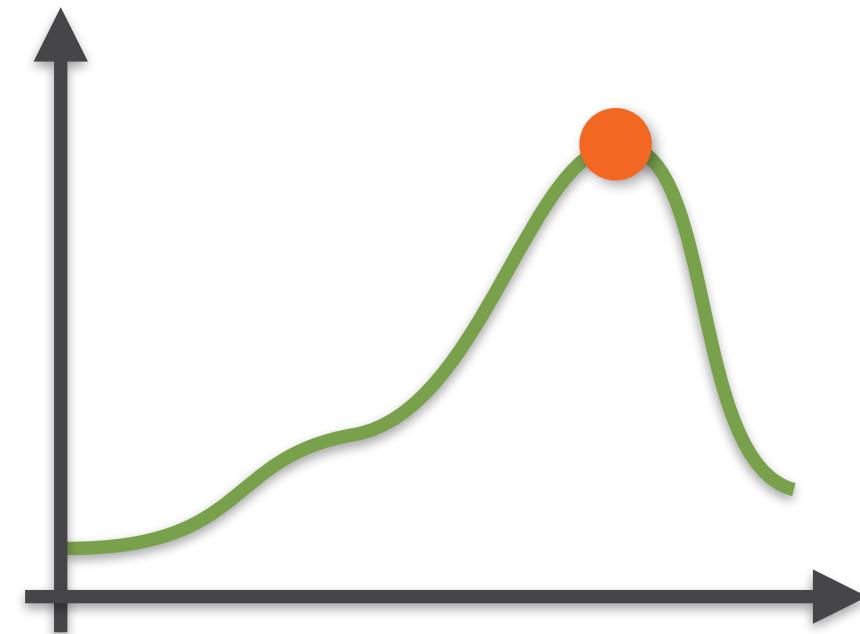
# Greedy Algorithms



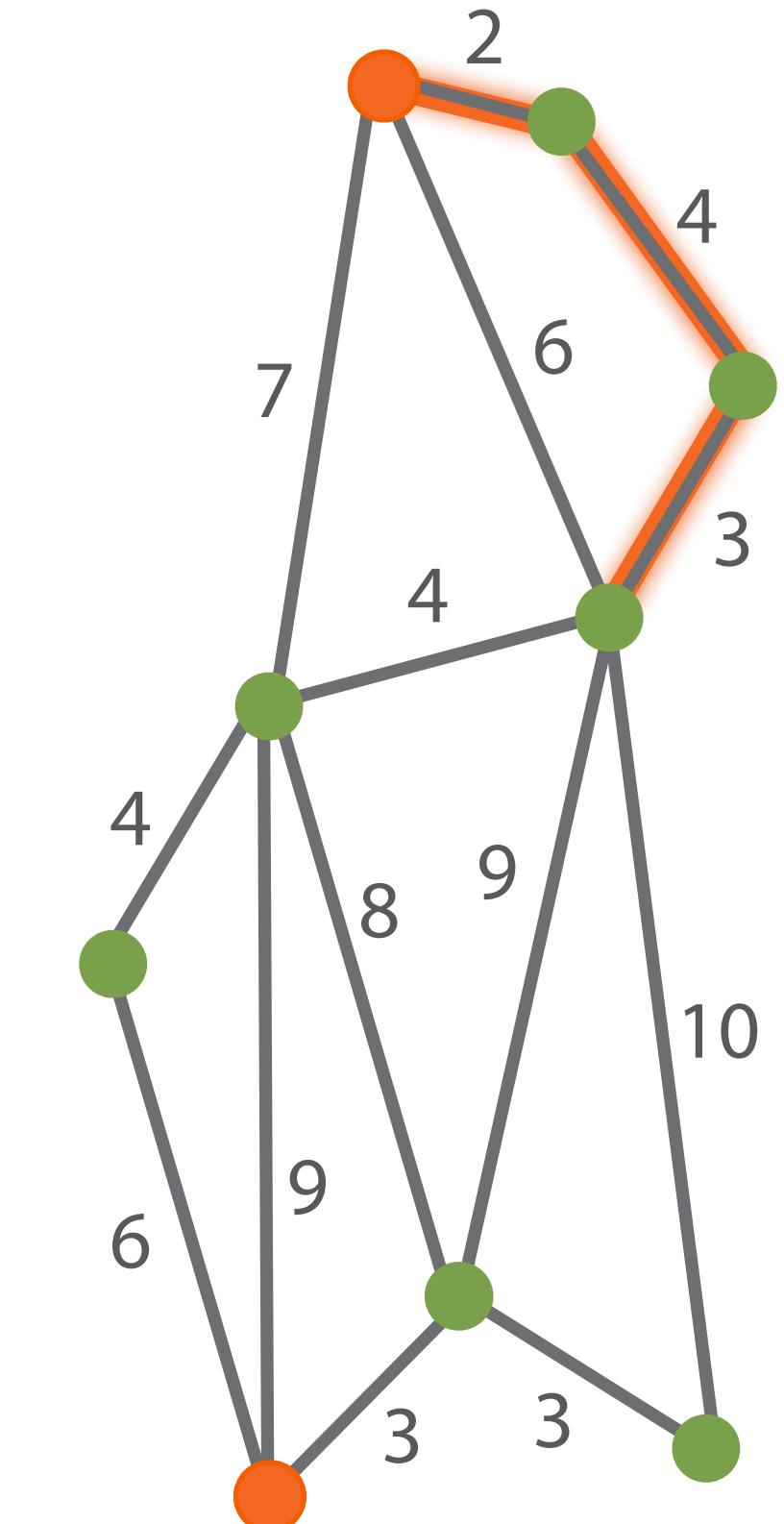
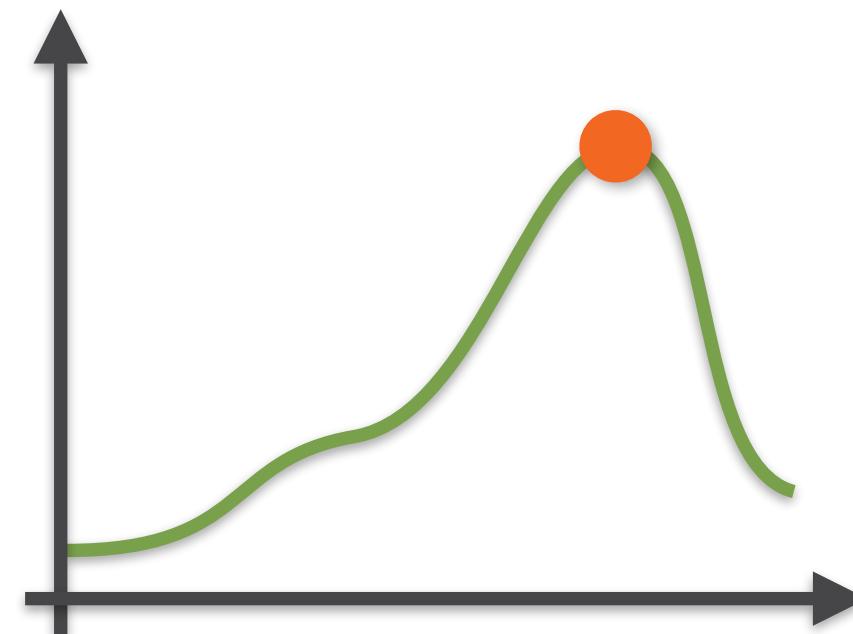
# Greedy Algorithms



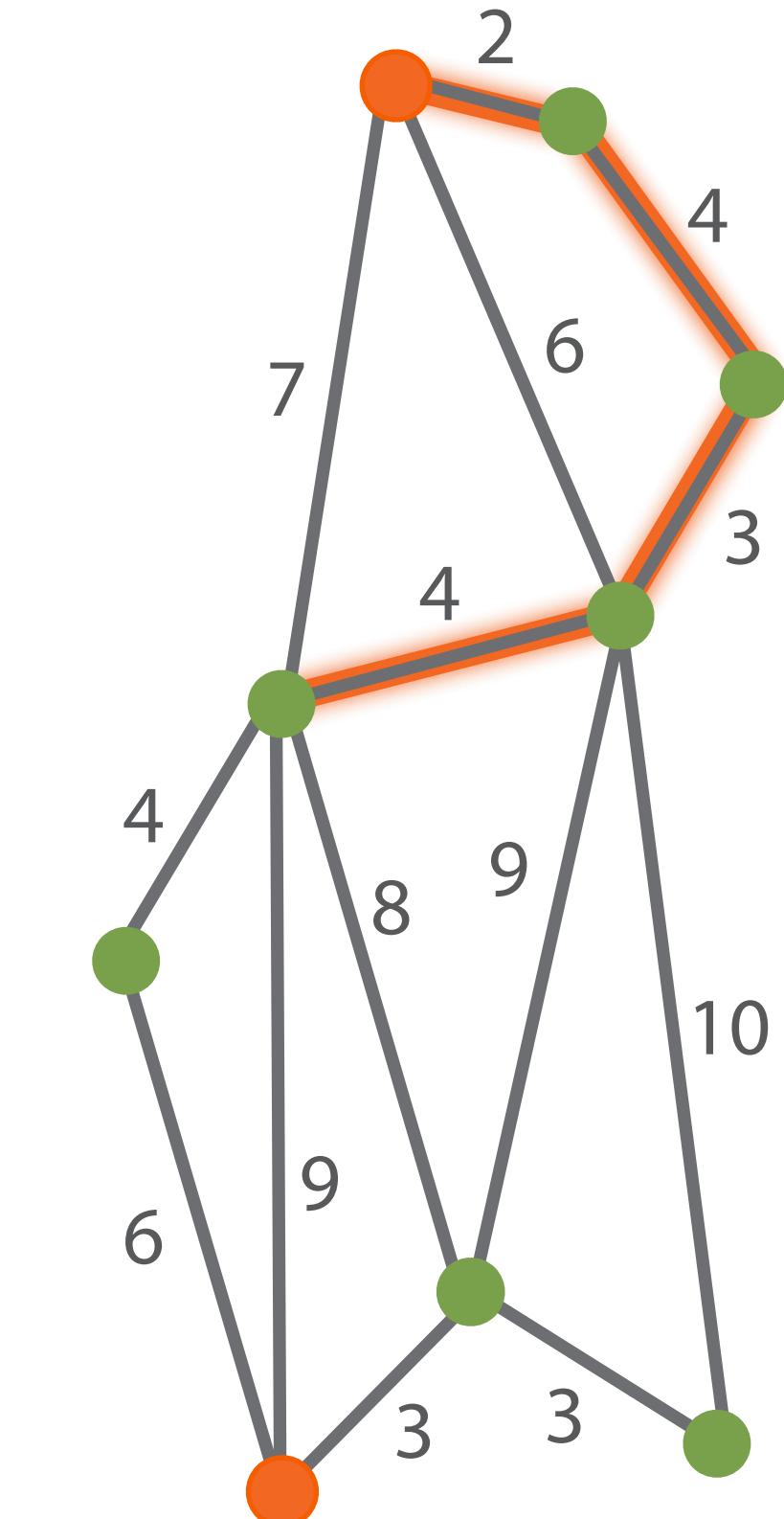
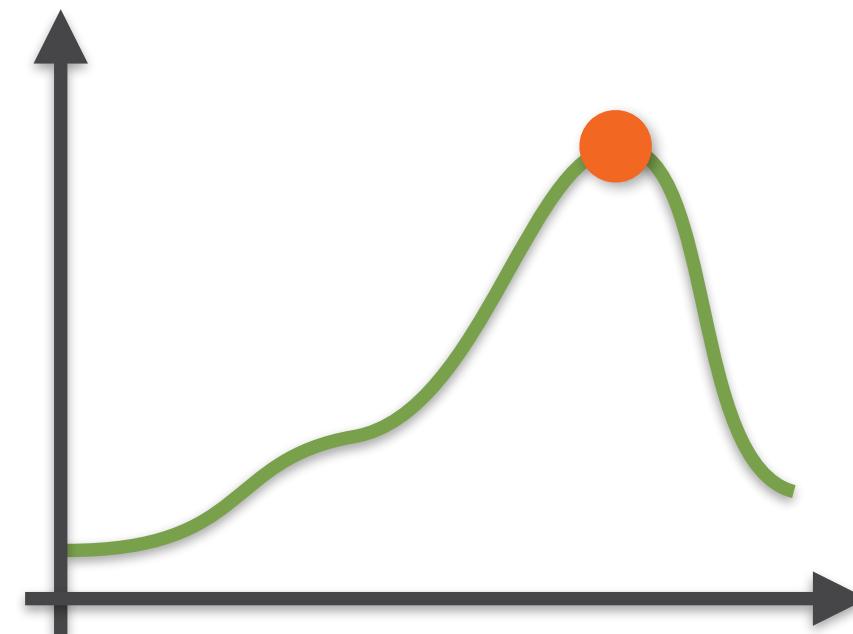
# Greedy Algorithms



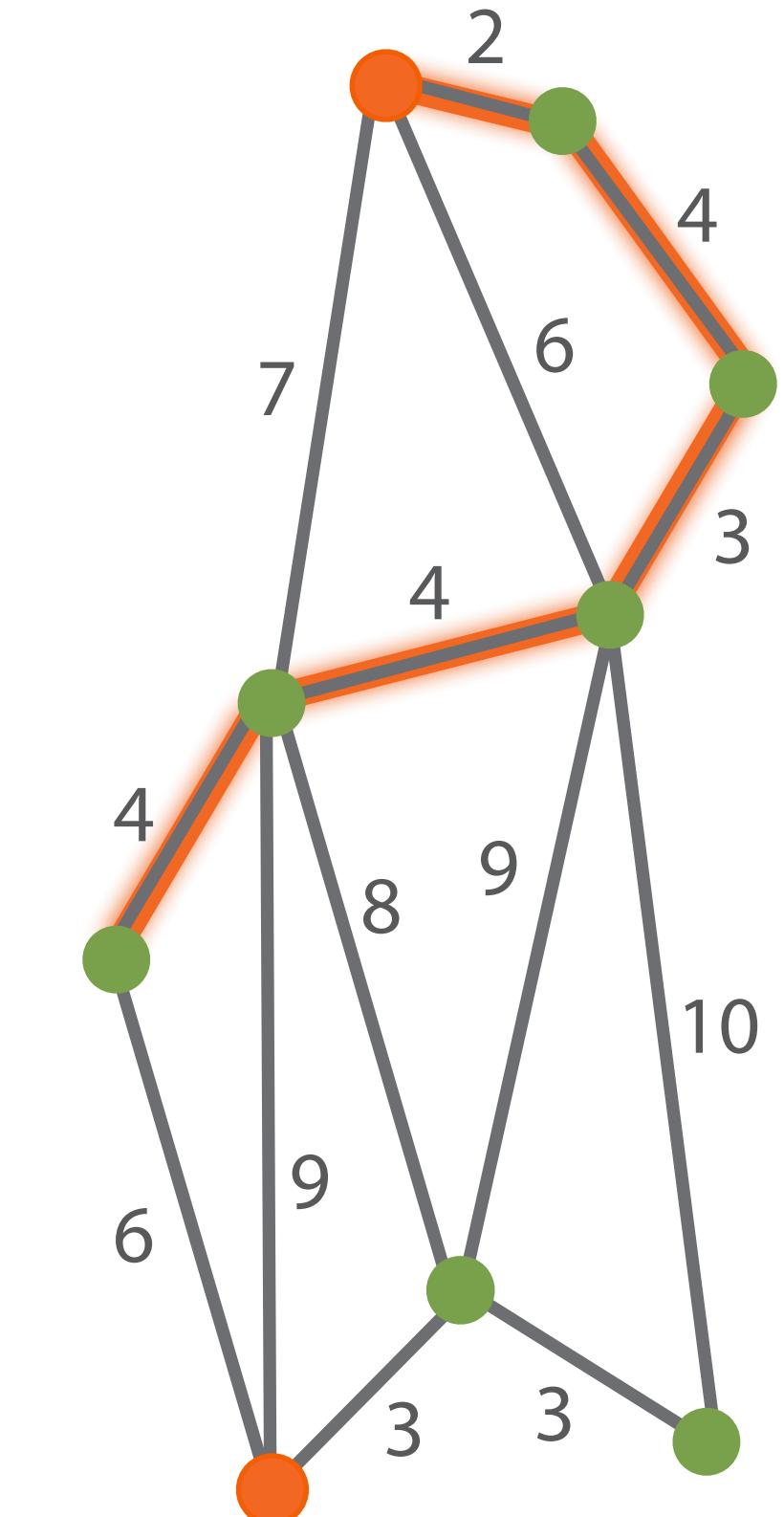
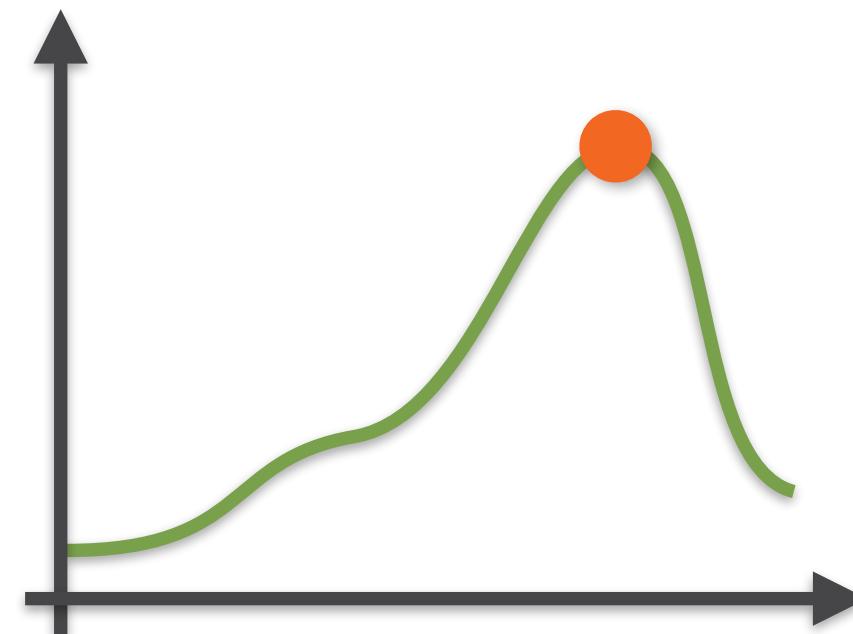
# Greedy Algorithms



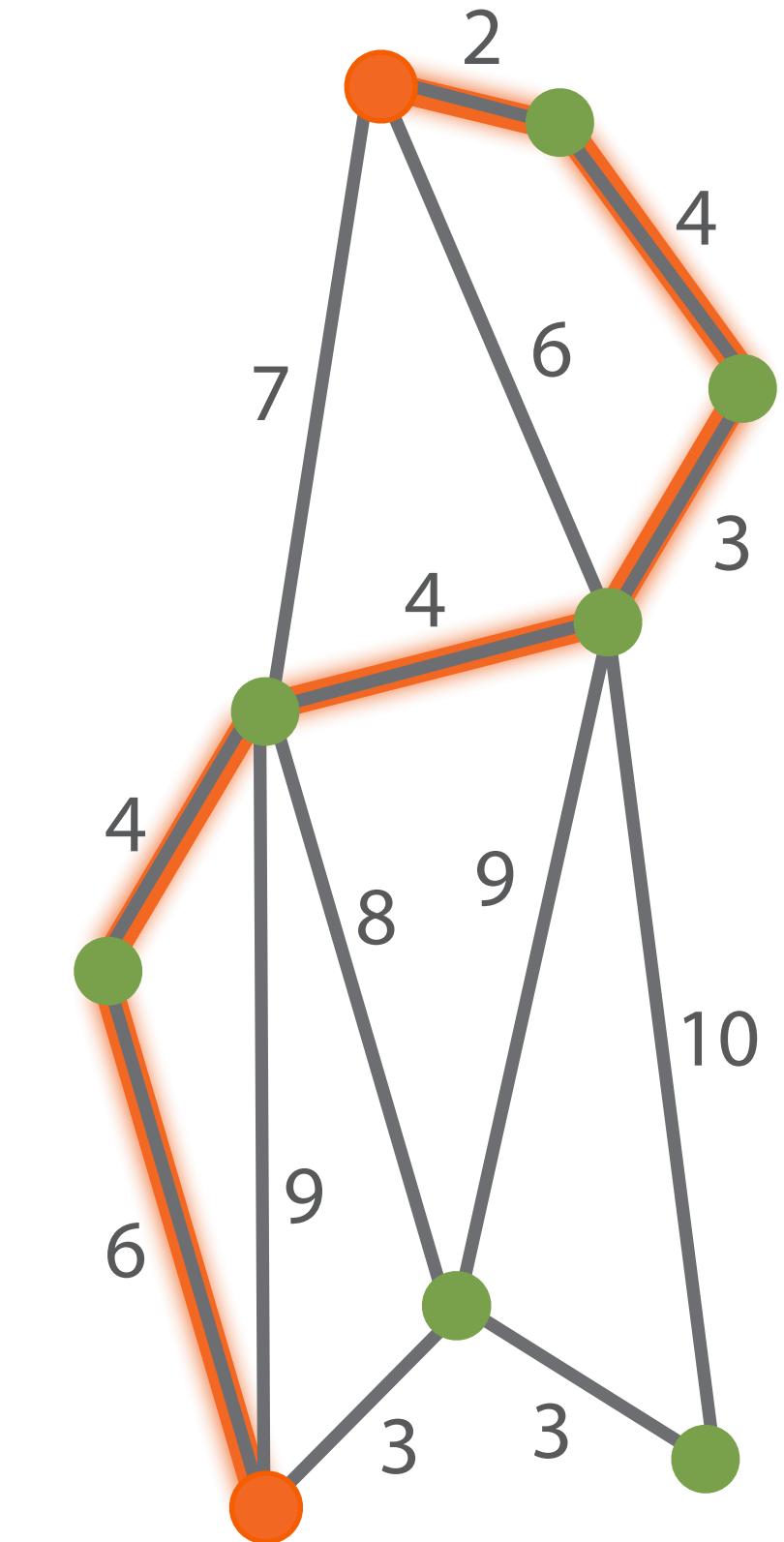
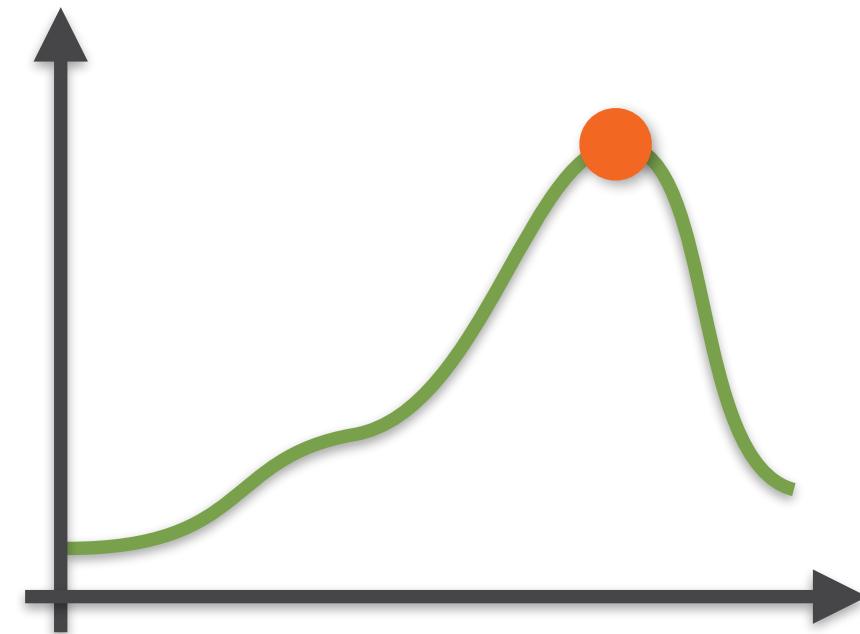
# Greedy Algorithms



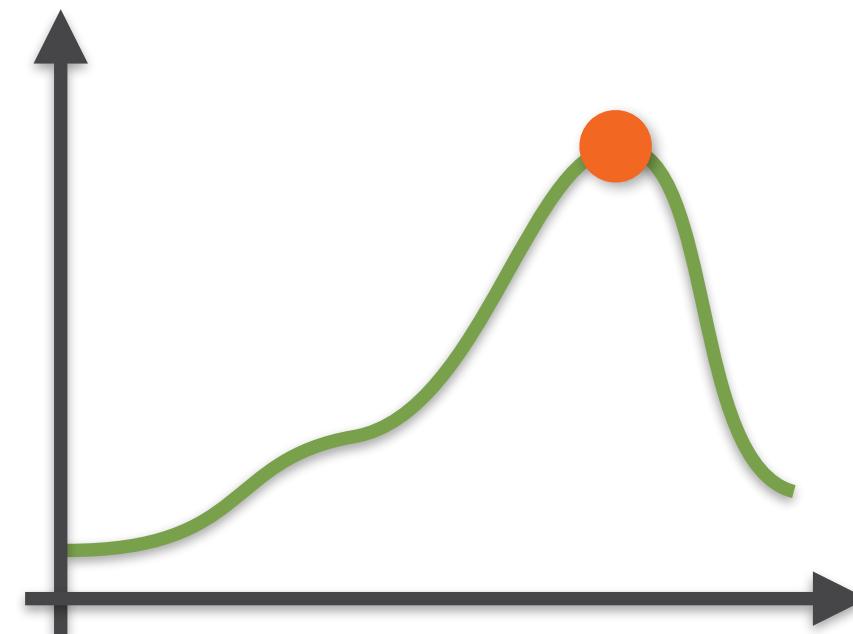
# Greedy Algorithms



# Greedy Algorithms



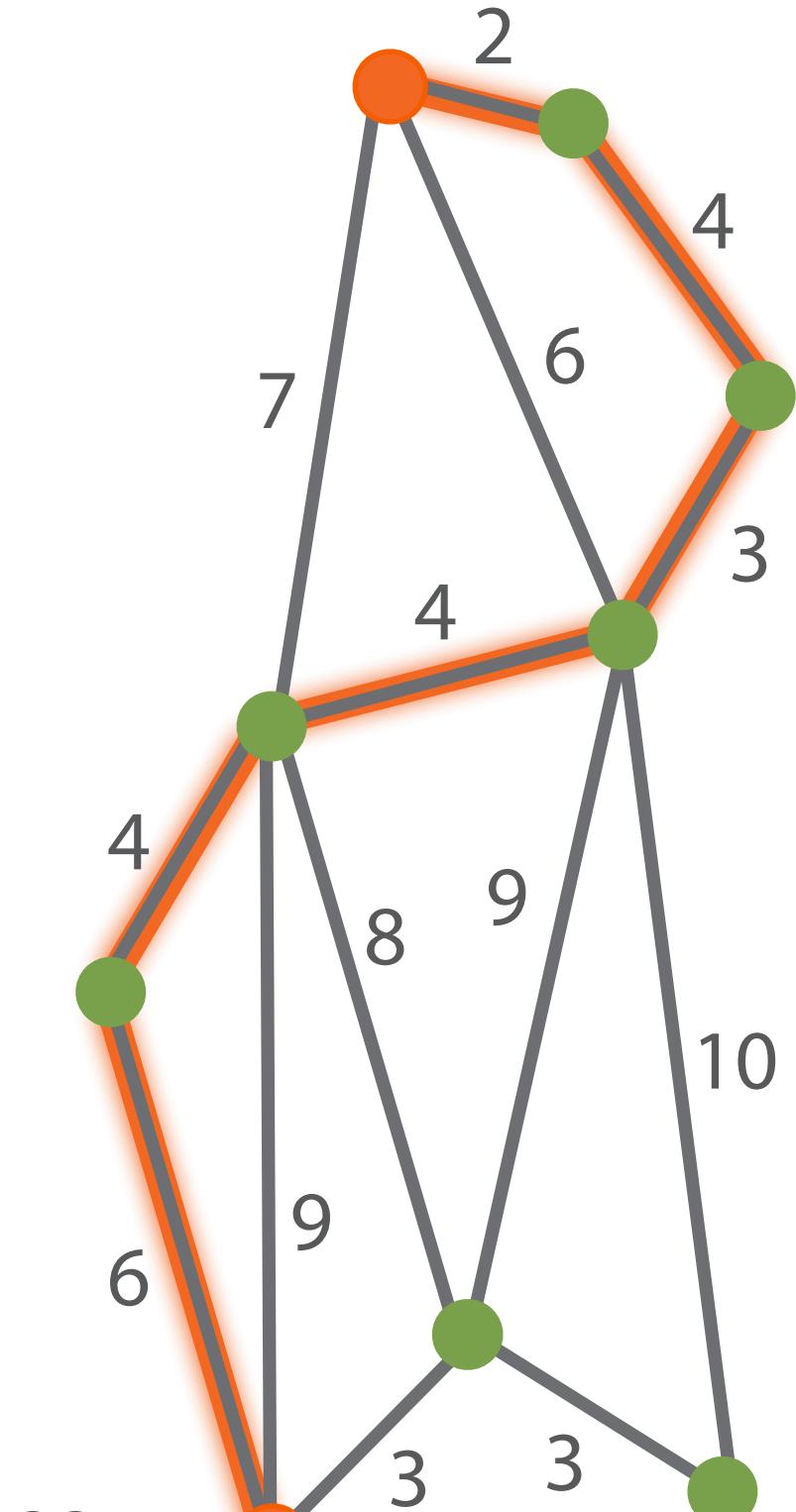
# Greedy Algorithms



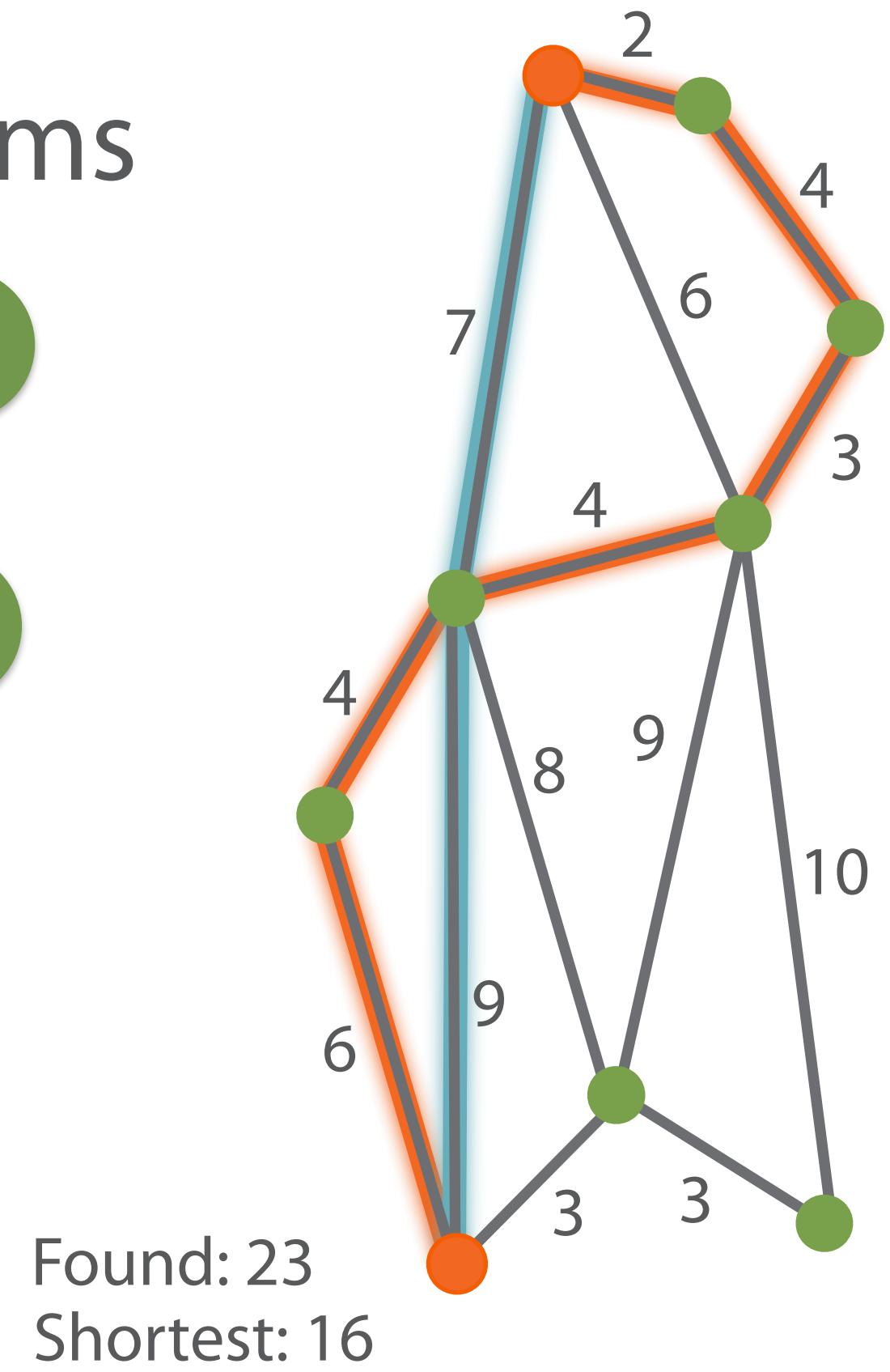
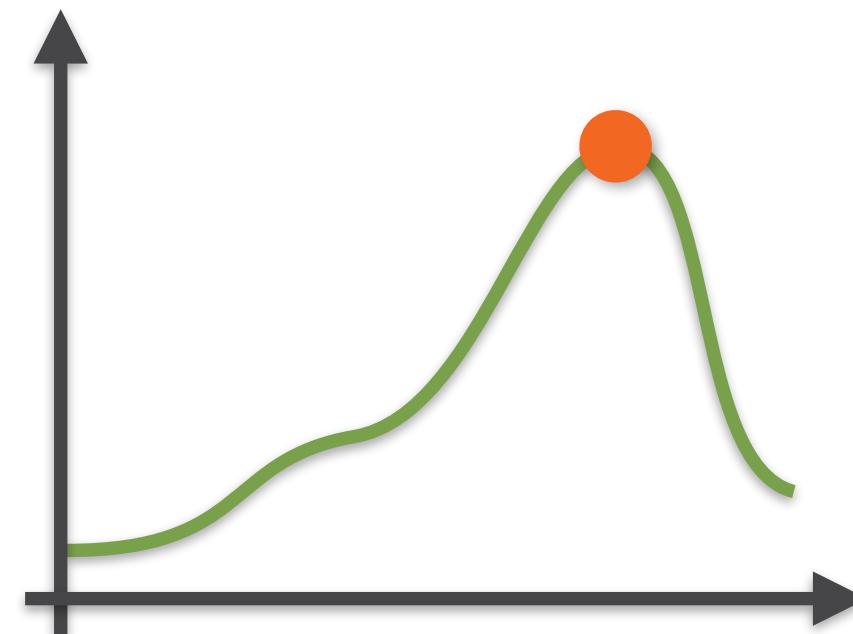
Get 18:



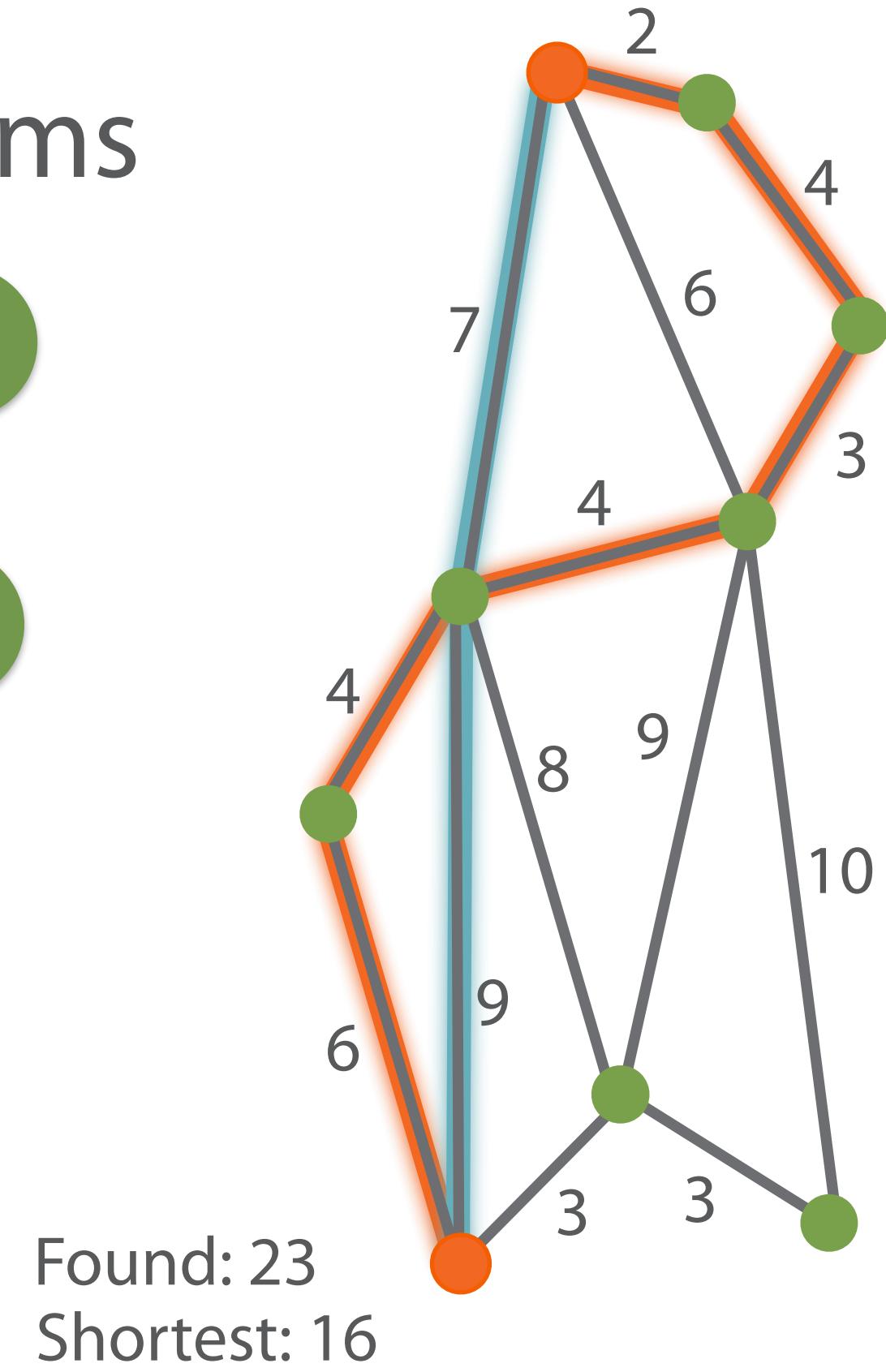
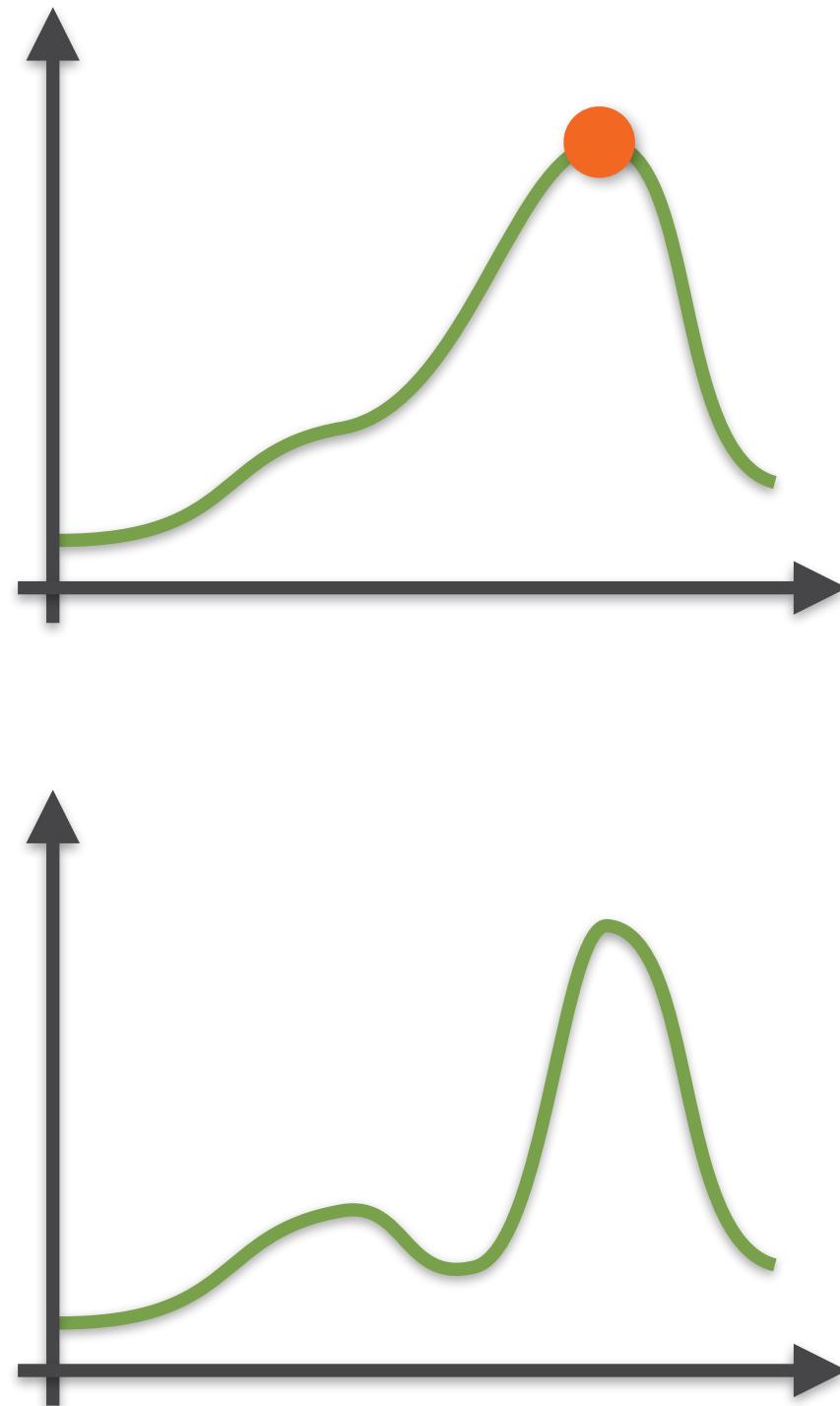
Found: 23



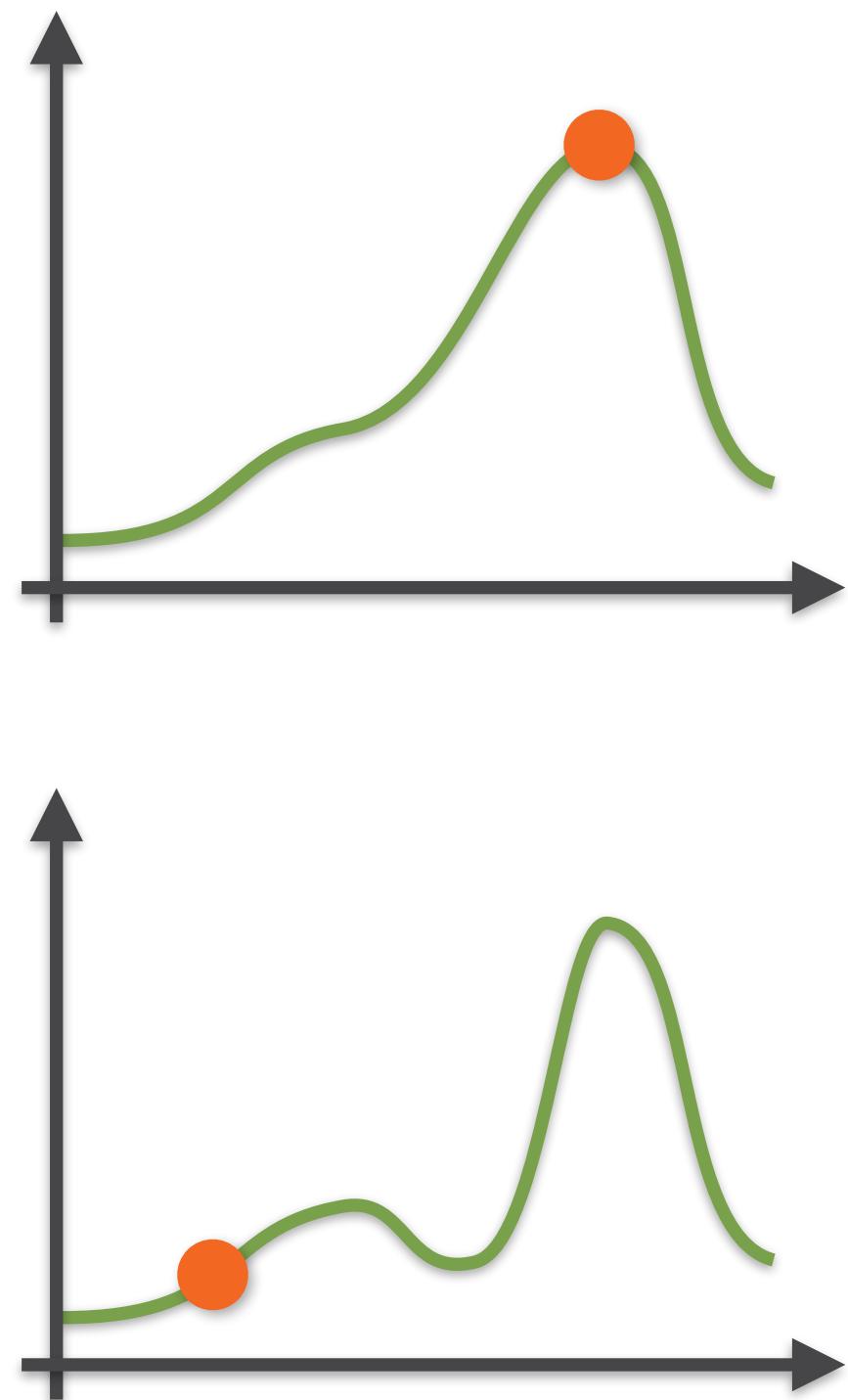
# Greedy Algorithms



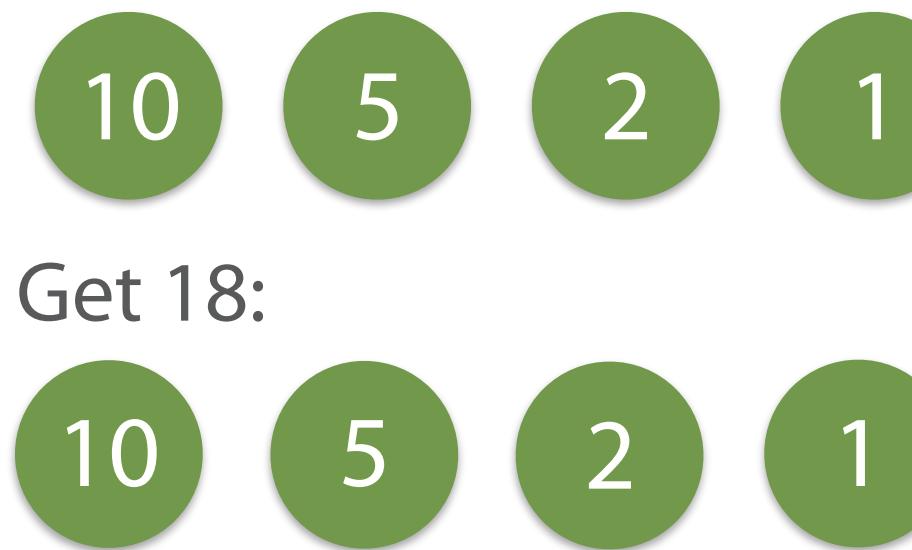
# Greedy Algorithms



# Greedy Algorithms

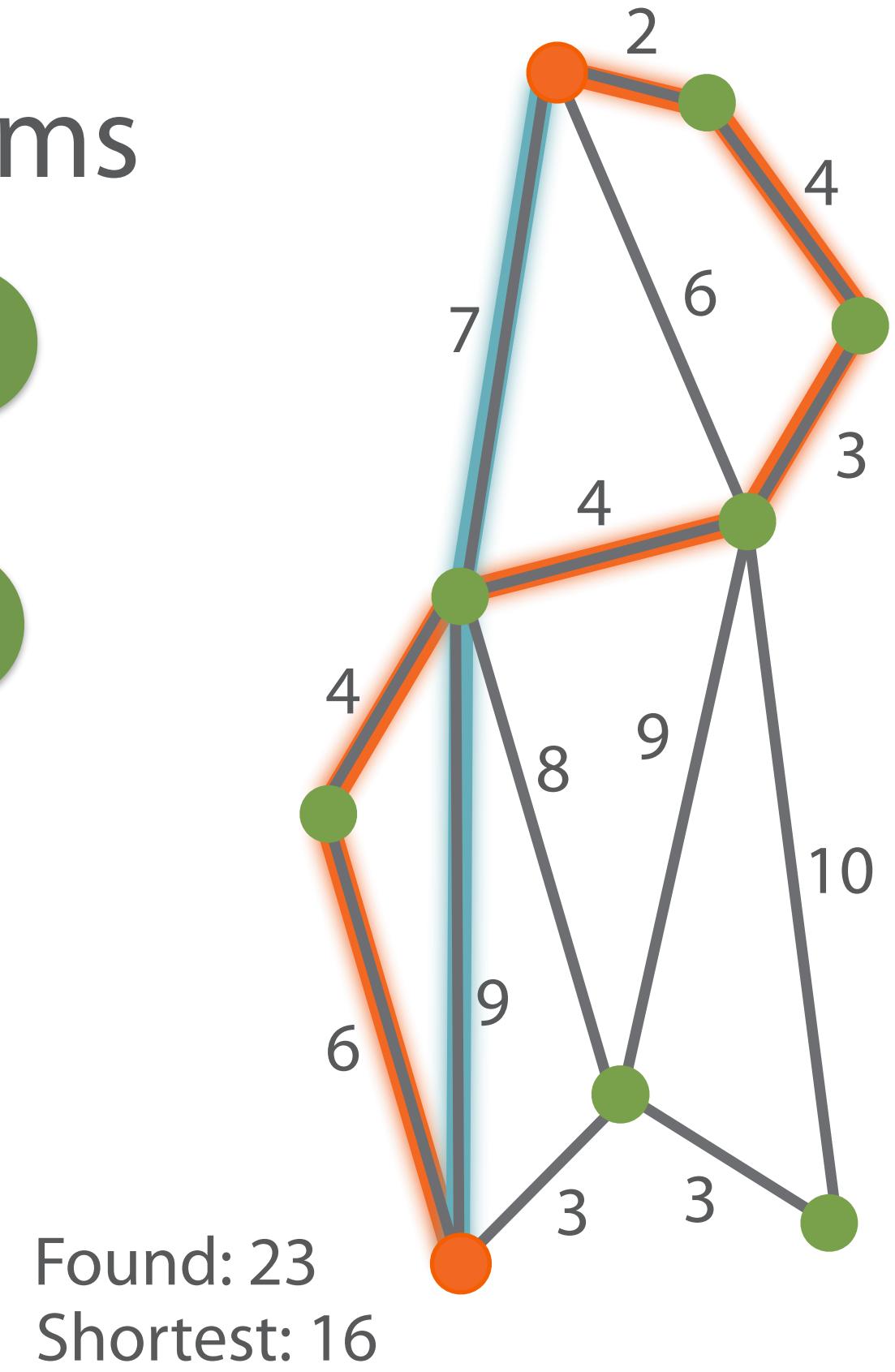
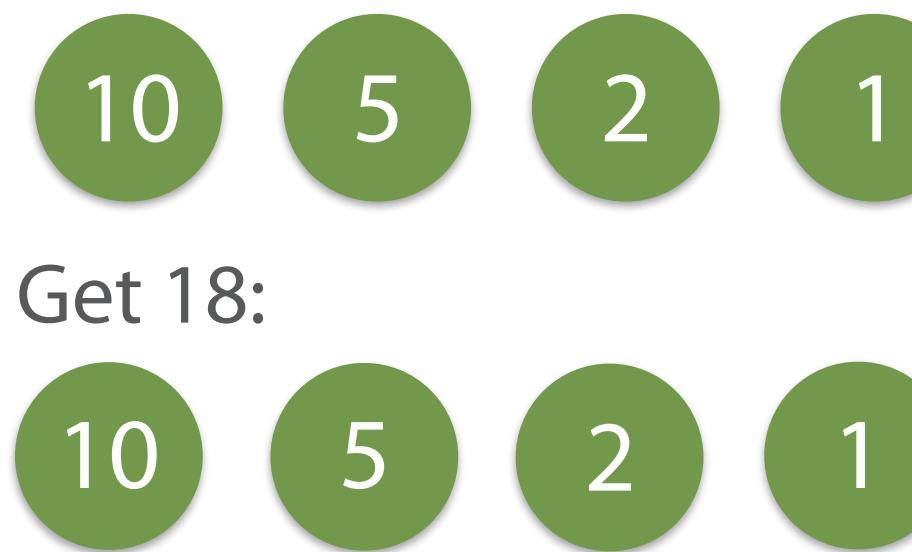
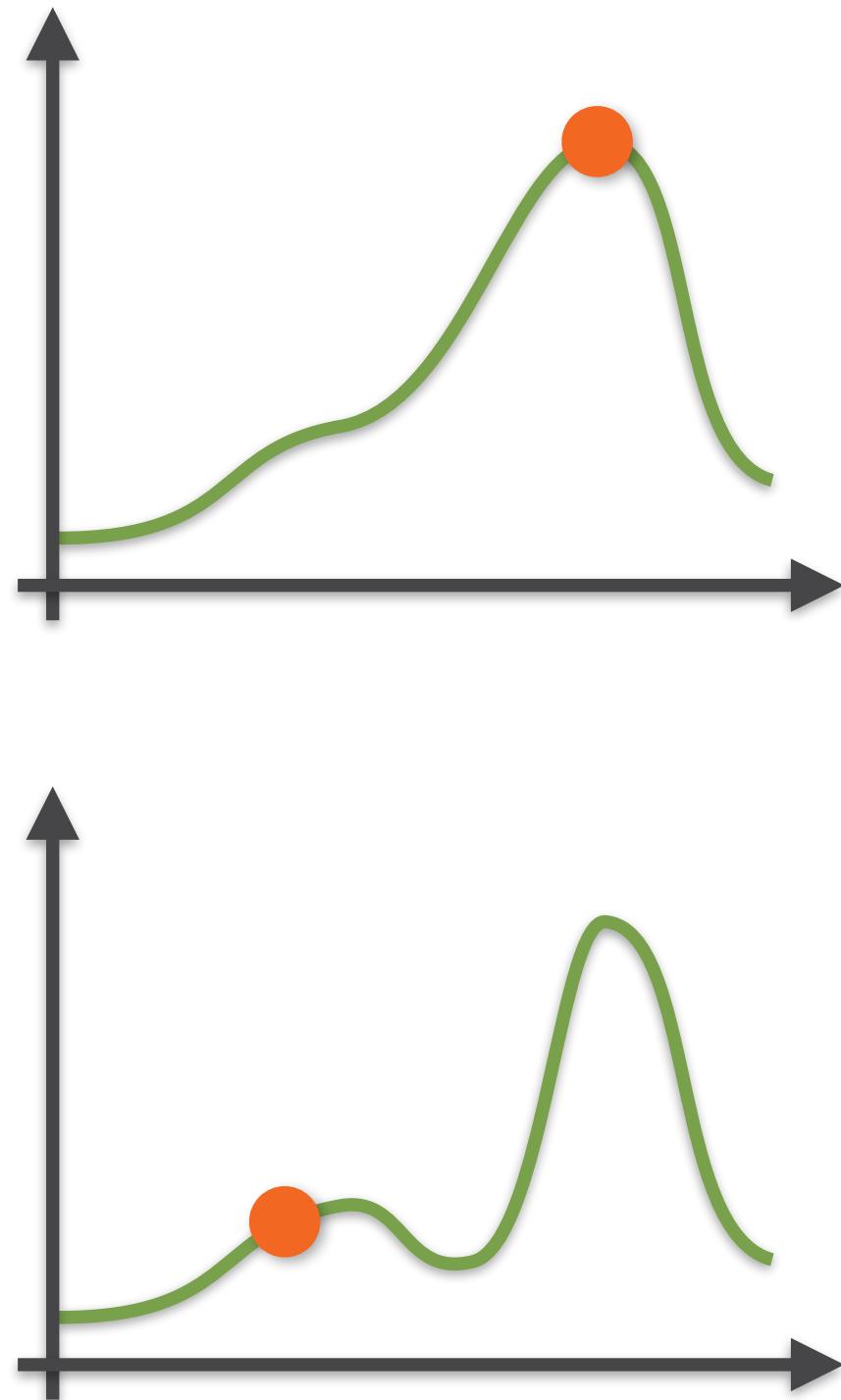


# Get 18:

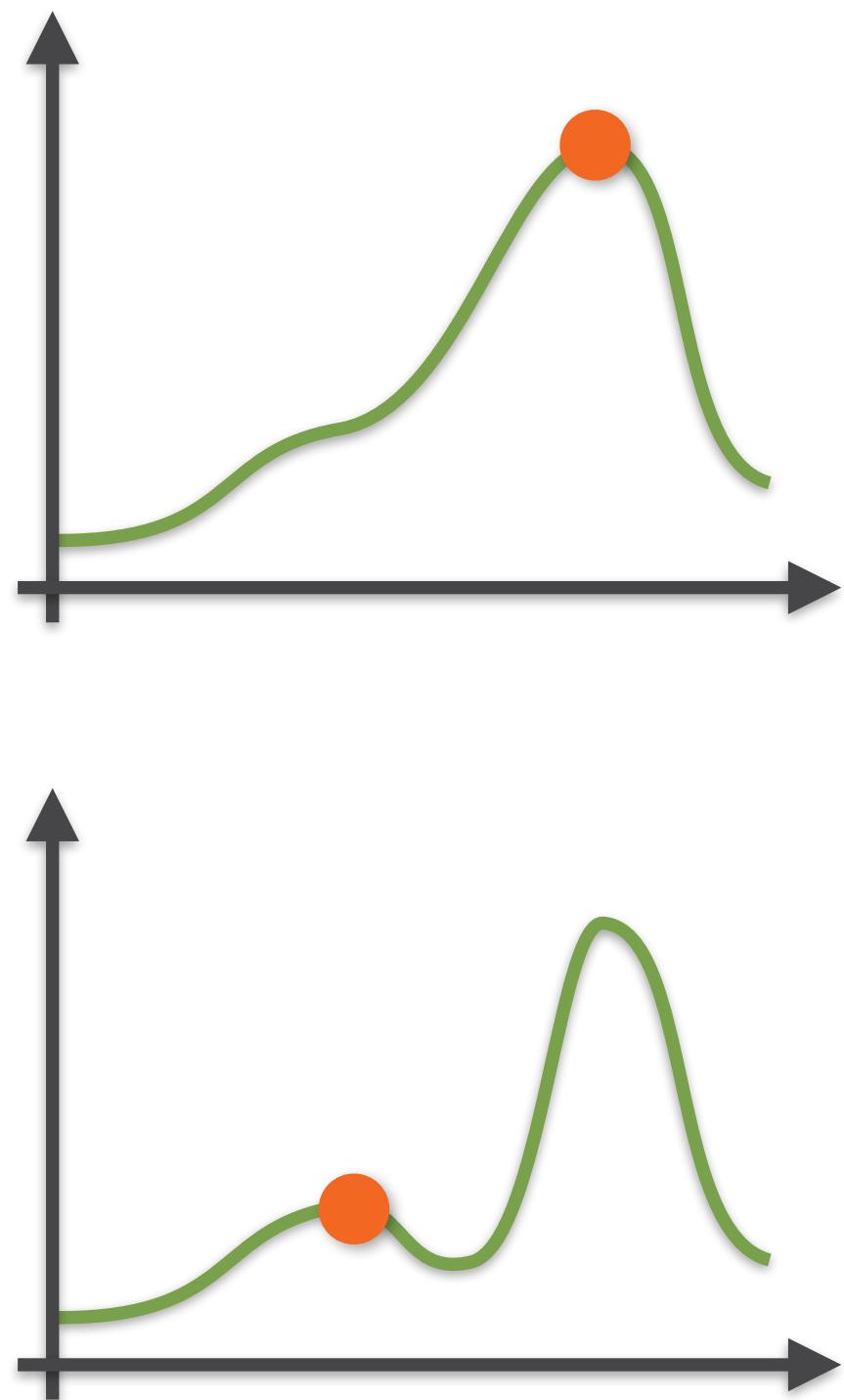


Found: 23  
Shortest: 16

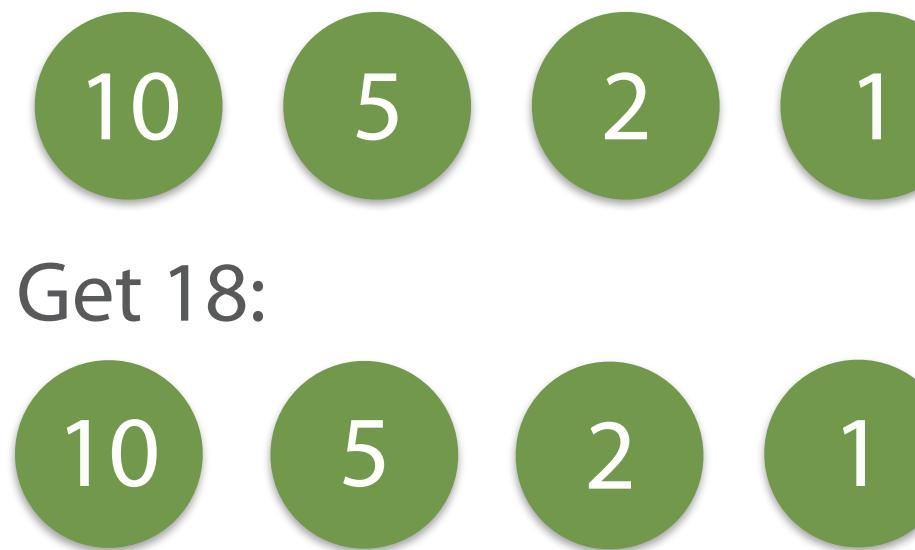
# Greedy Algorithms



# Greedy Algorithms

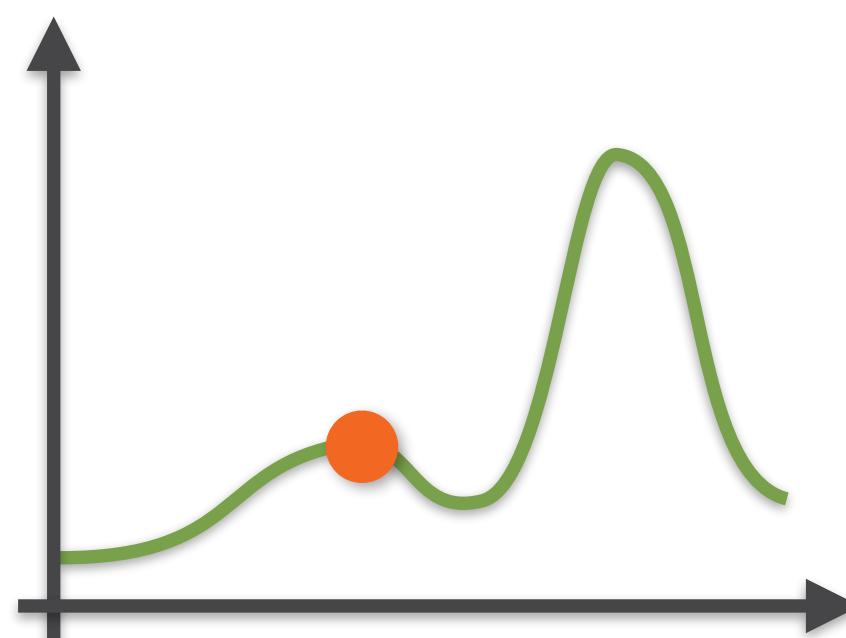
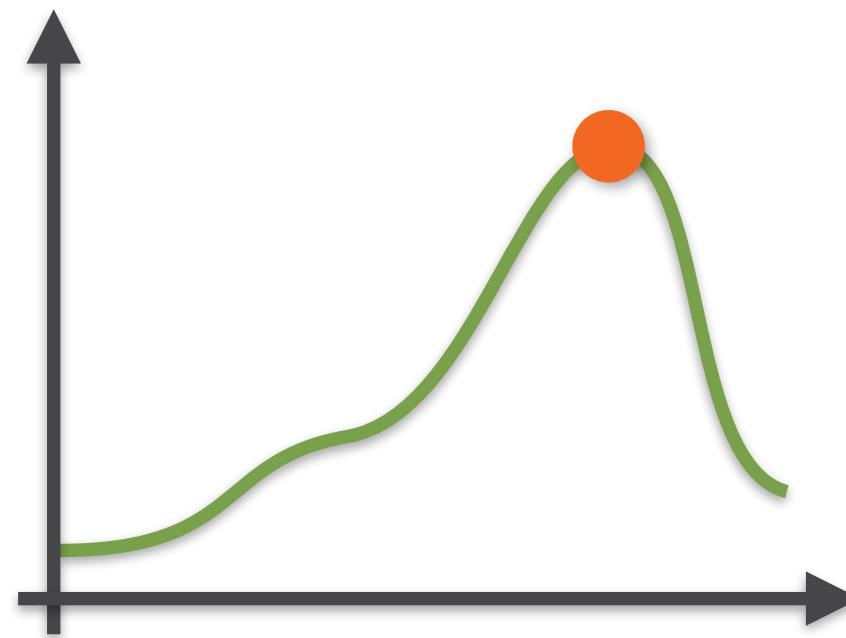


# Get 18:



Found: 23  
Shortest: 16

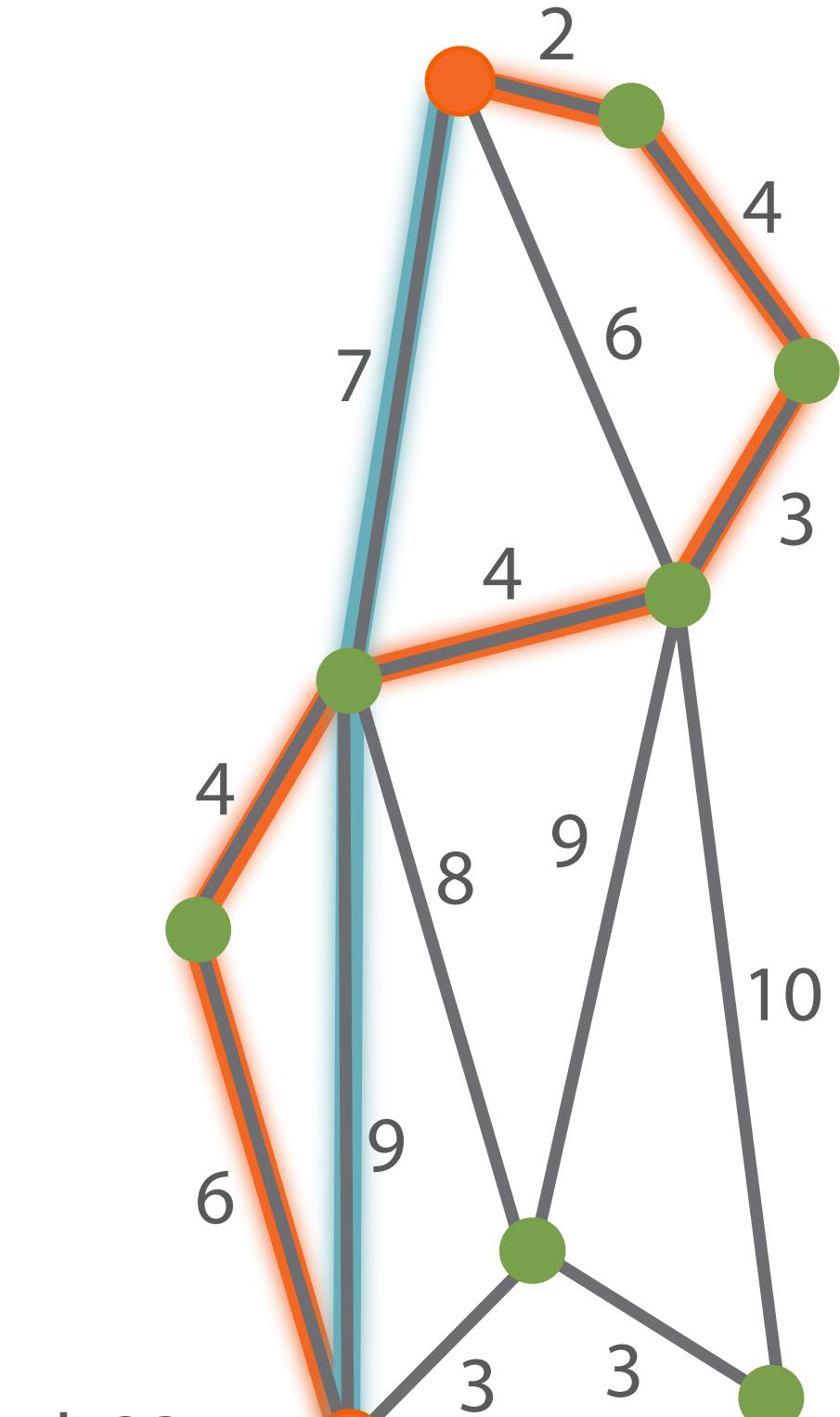
# Greedy Algorithms



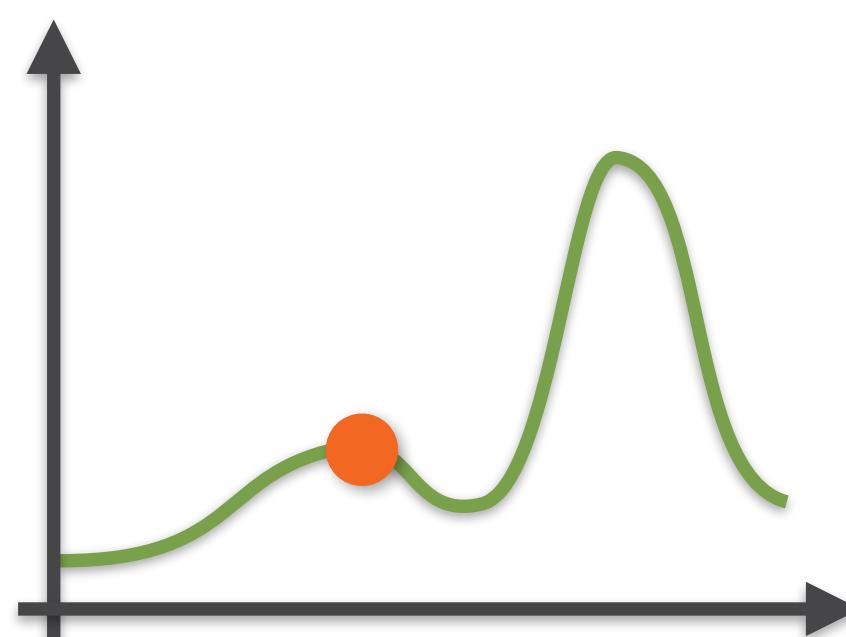
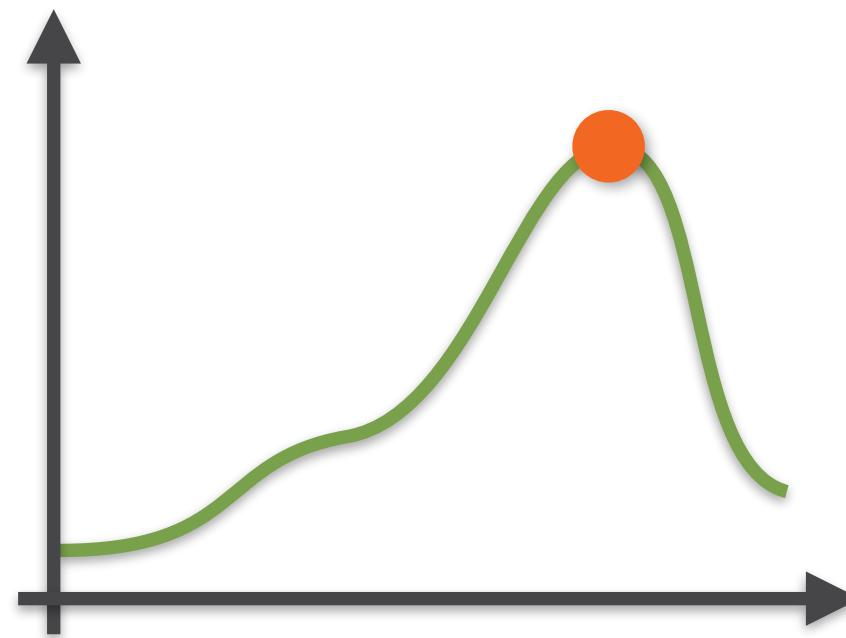
Get 18:



Found: 23  
Shortest: 16



# Greedy Algorithms

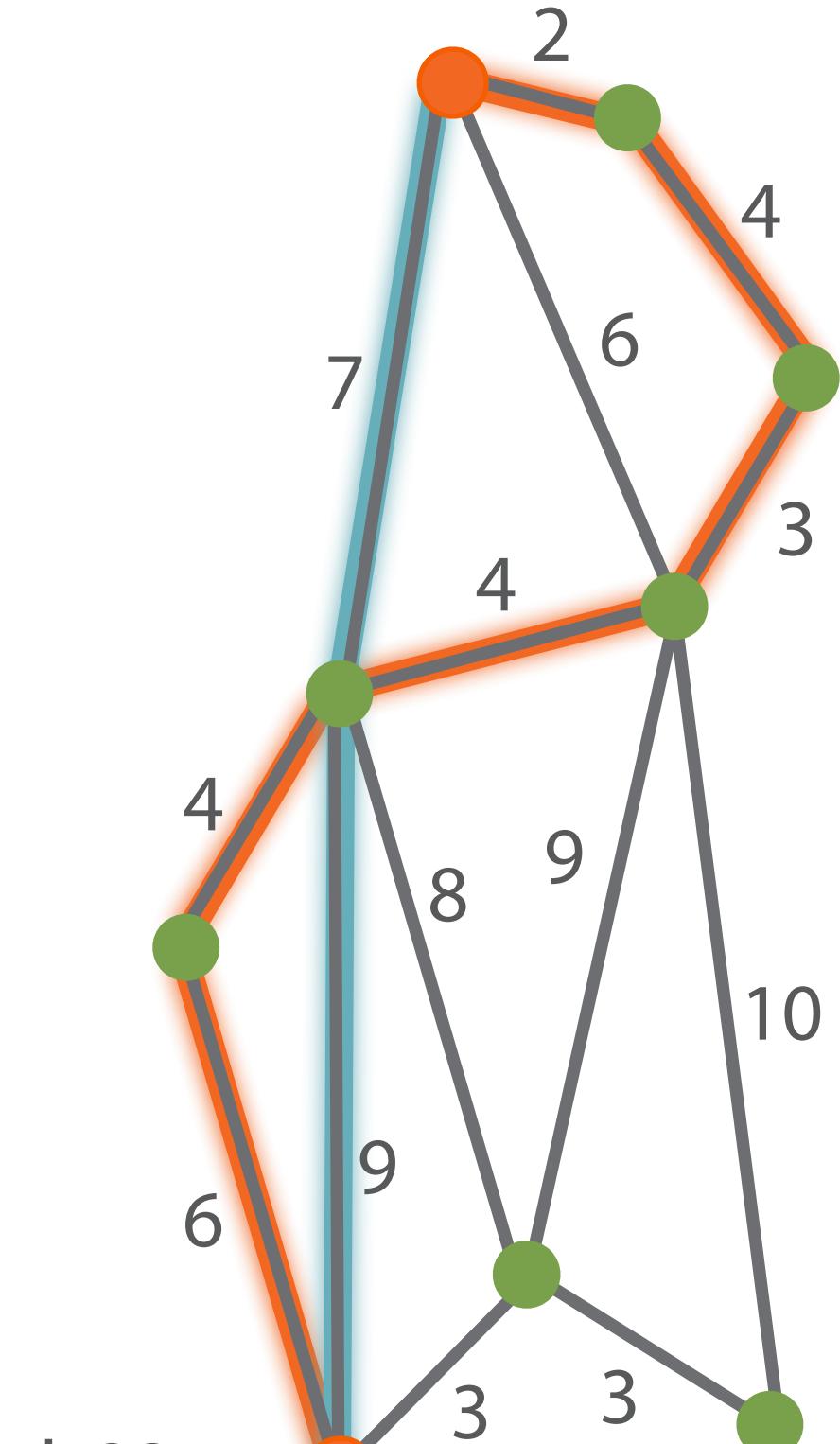


Get 18:

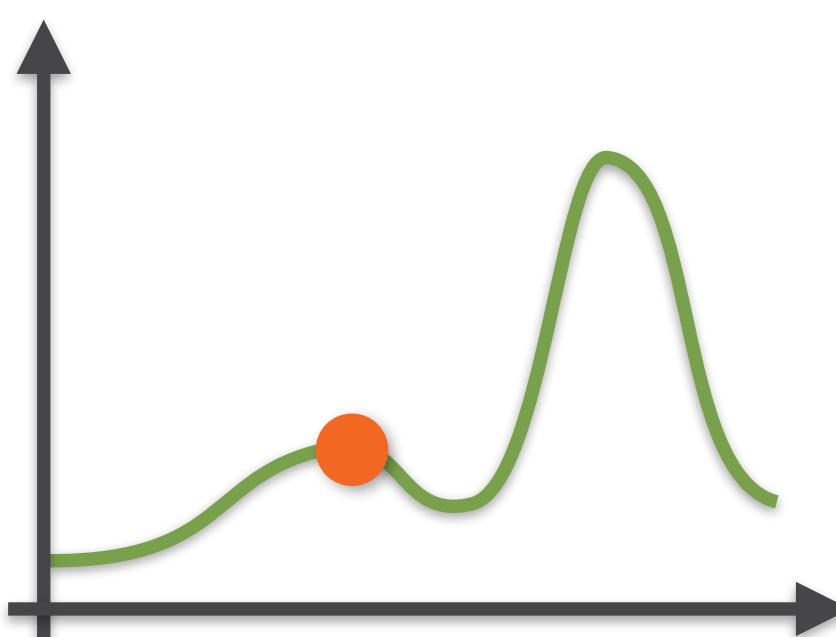
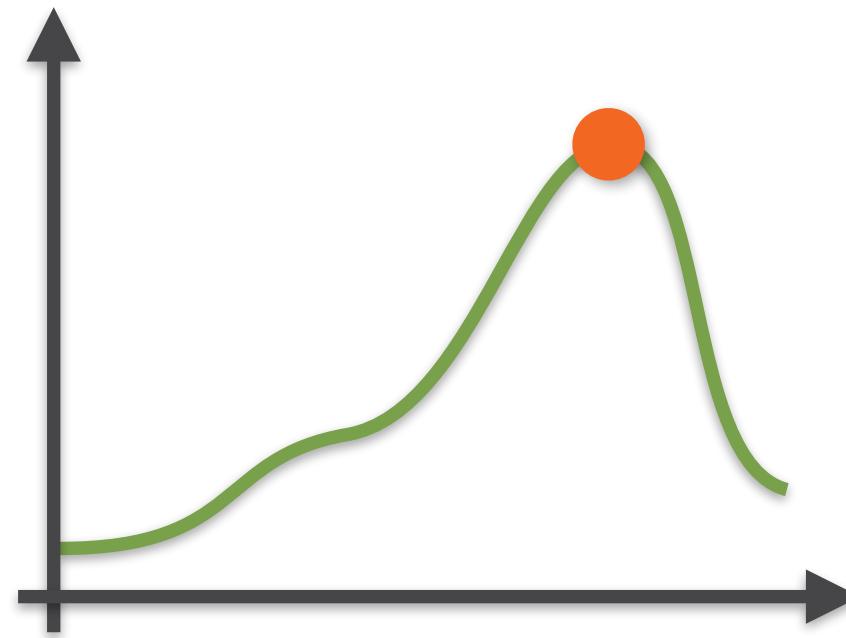


Get 18:

Found: 23  
Shortest: 16



# Greedy Algorithms



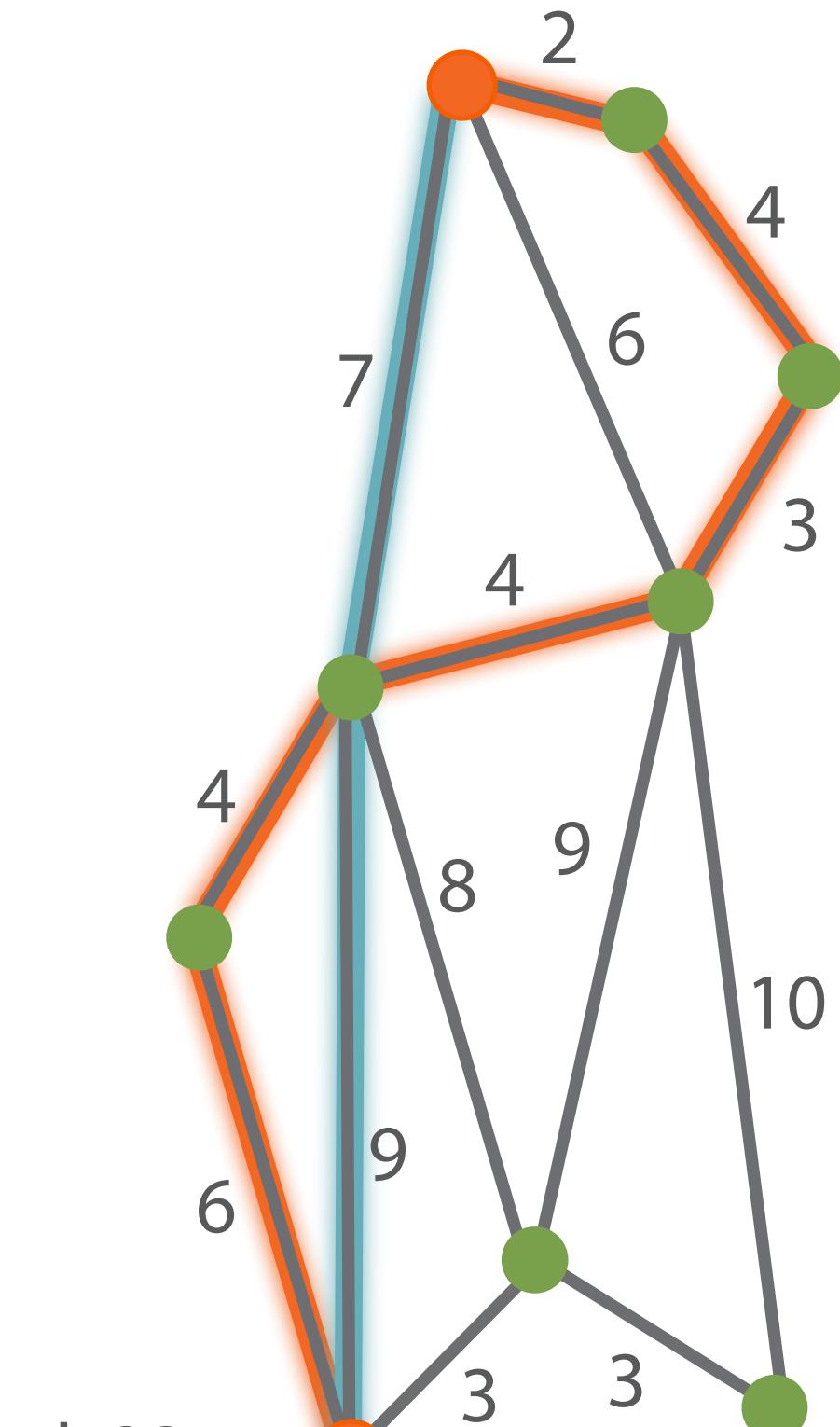
Get 18:



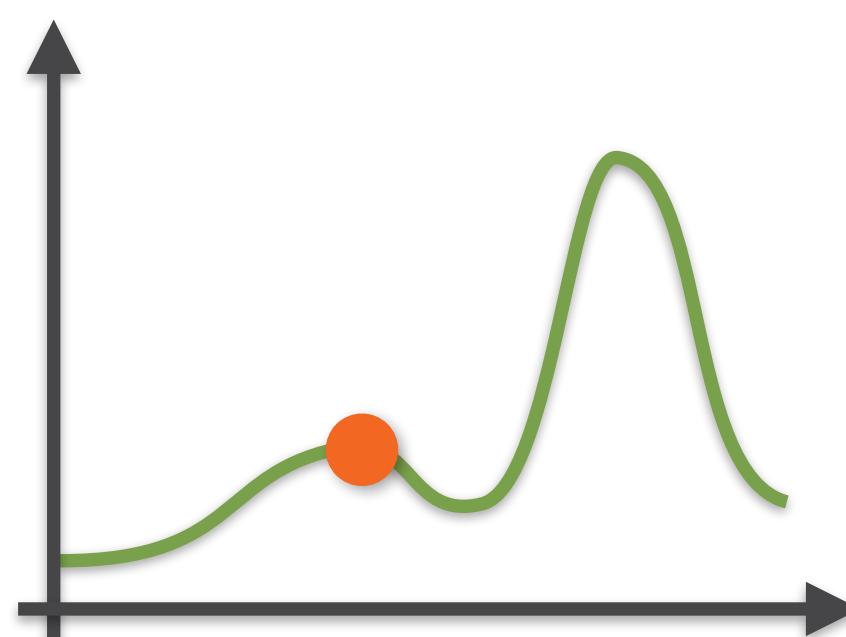
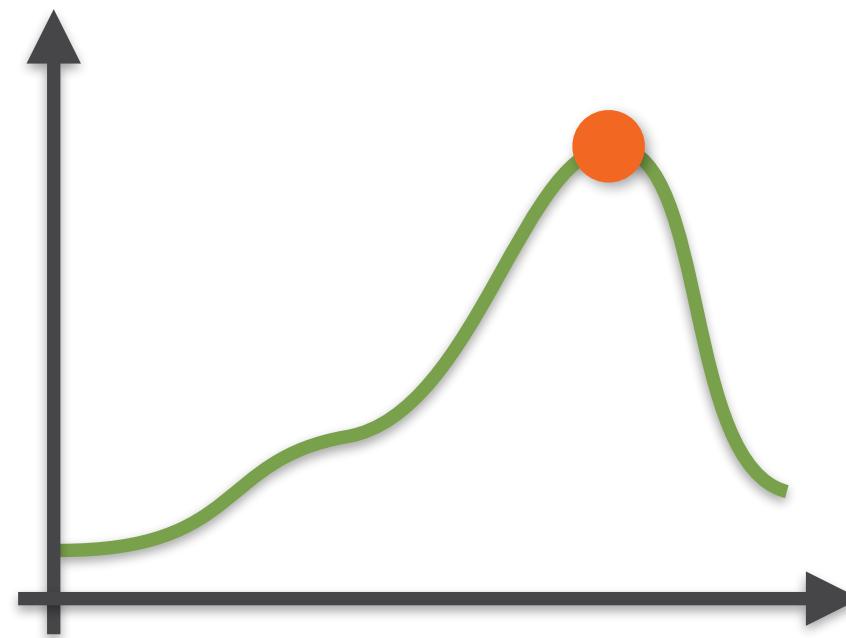
Get 18:



Found: 23  
Shortest: 16



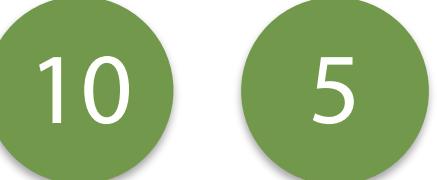
# Greedy Algorithms



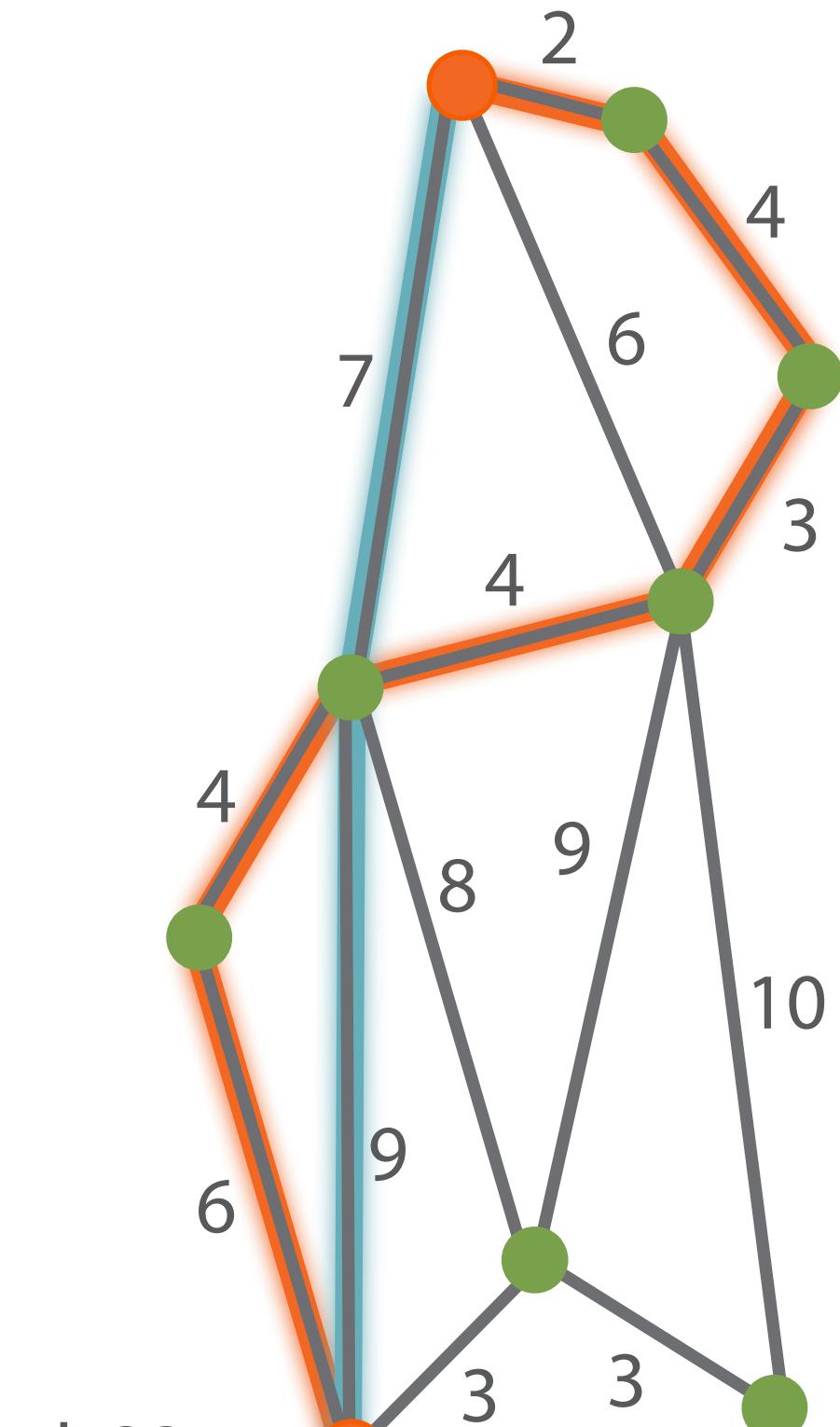
Get 18:



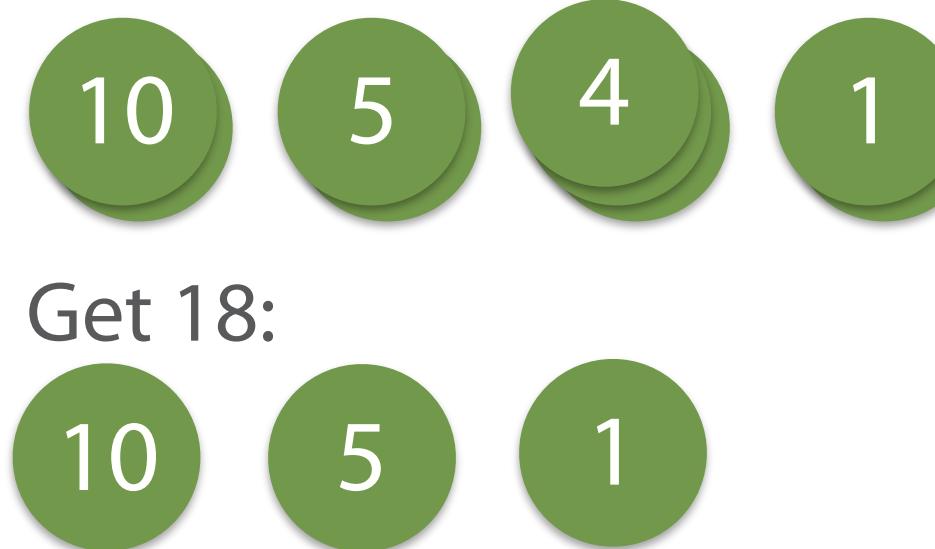
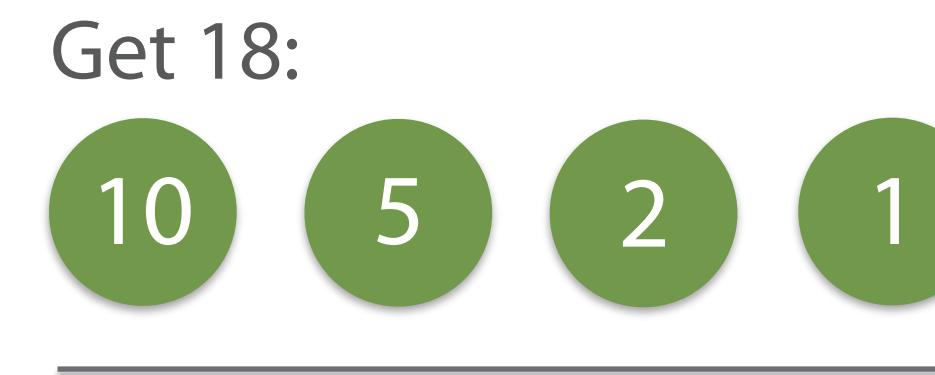
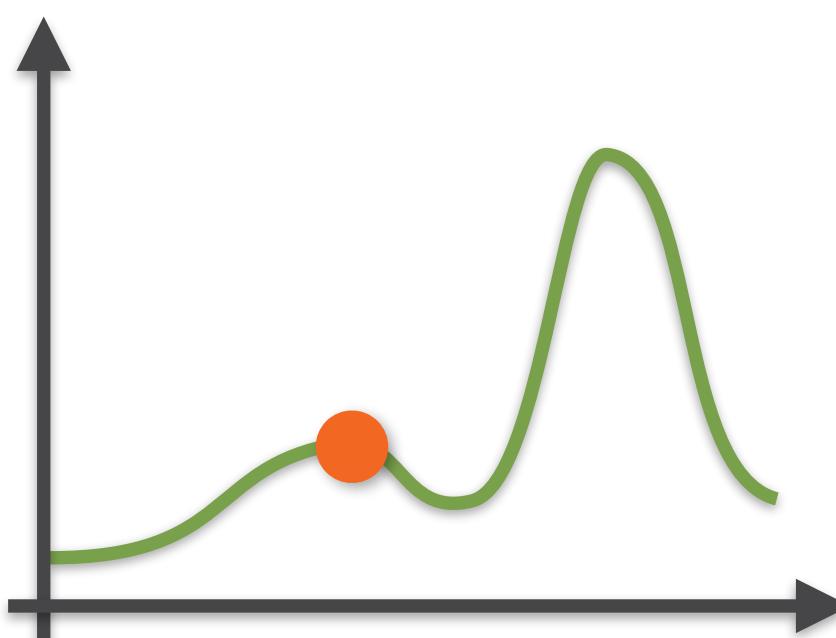
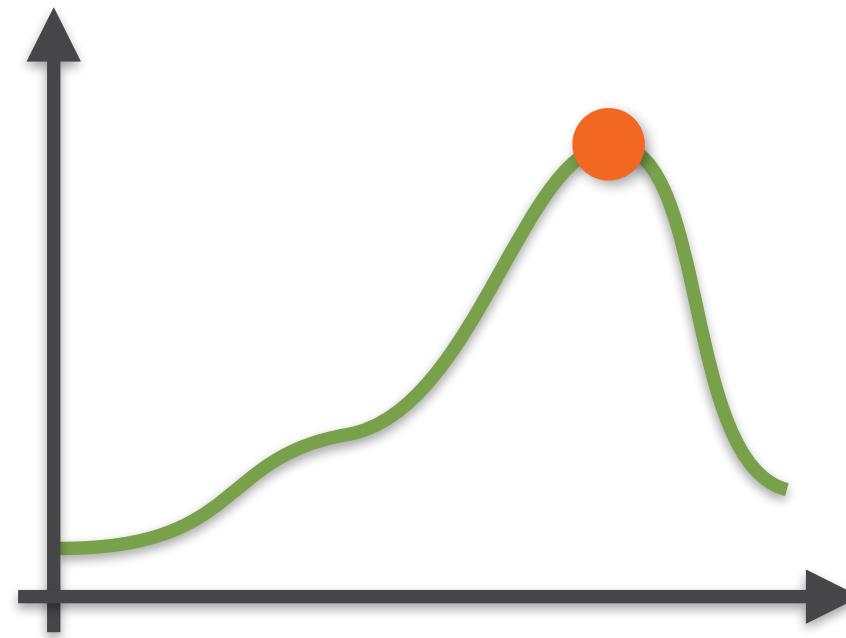
Get 18:



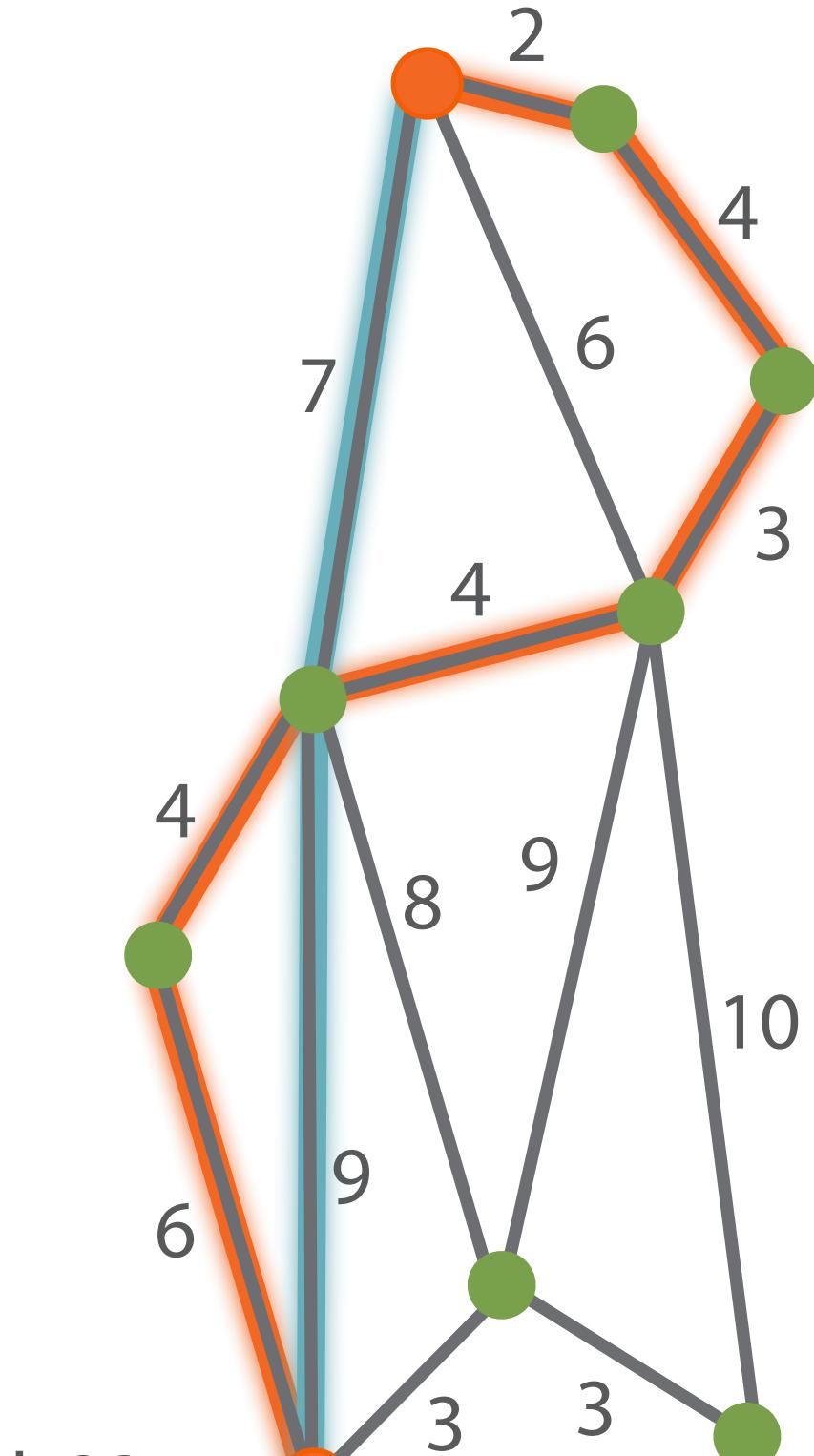
Found: 23  
Shortest: 16



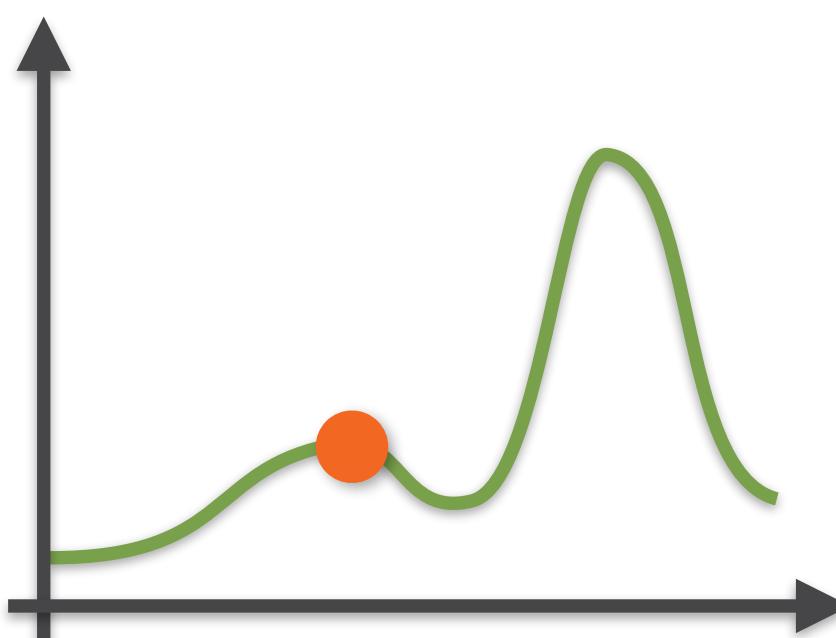
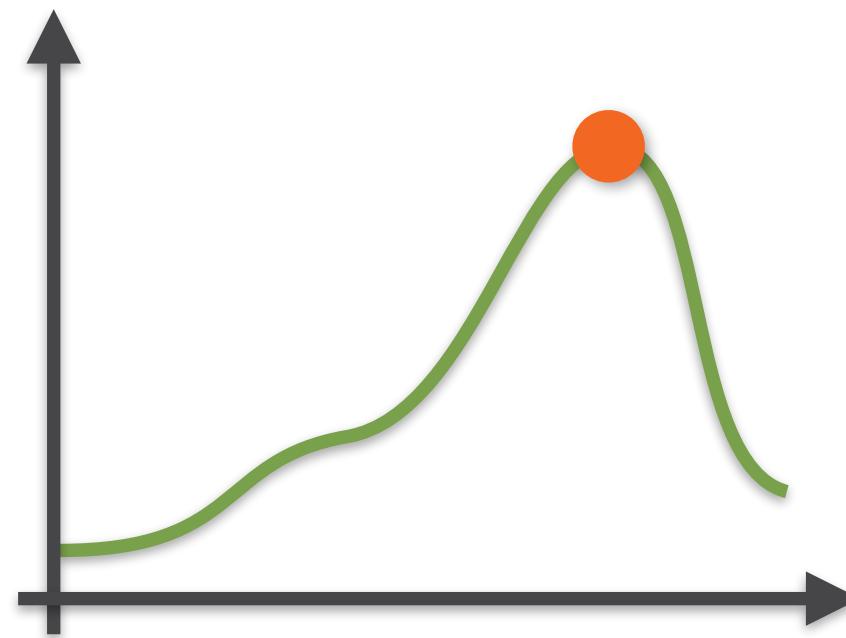
# Greedy Algorithms



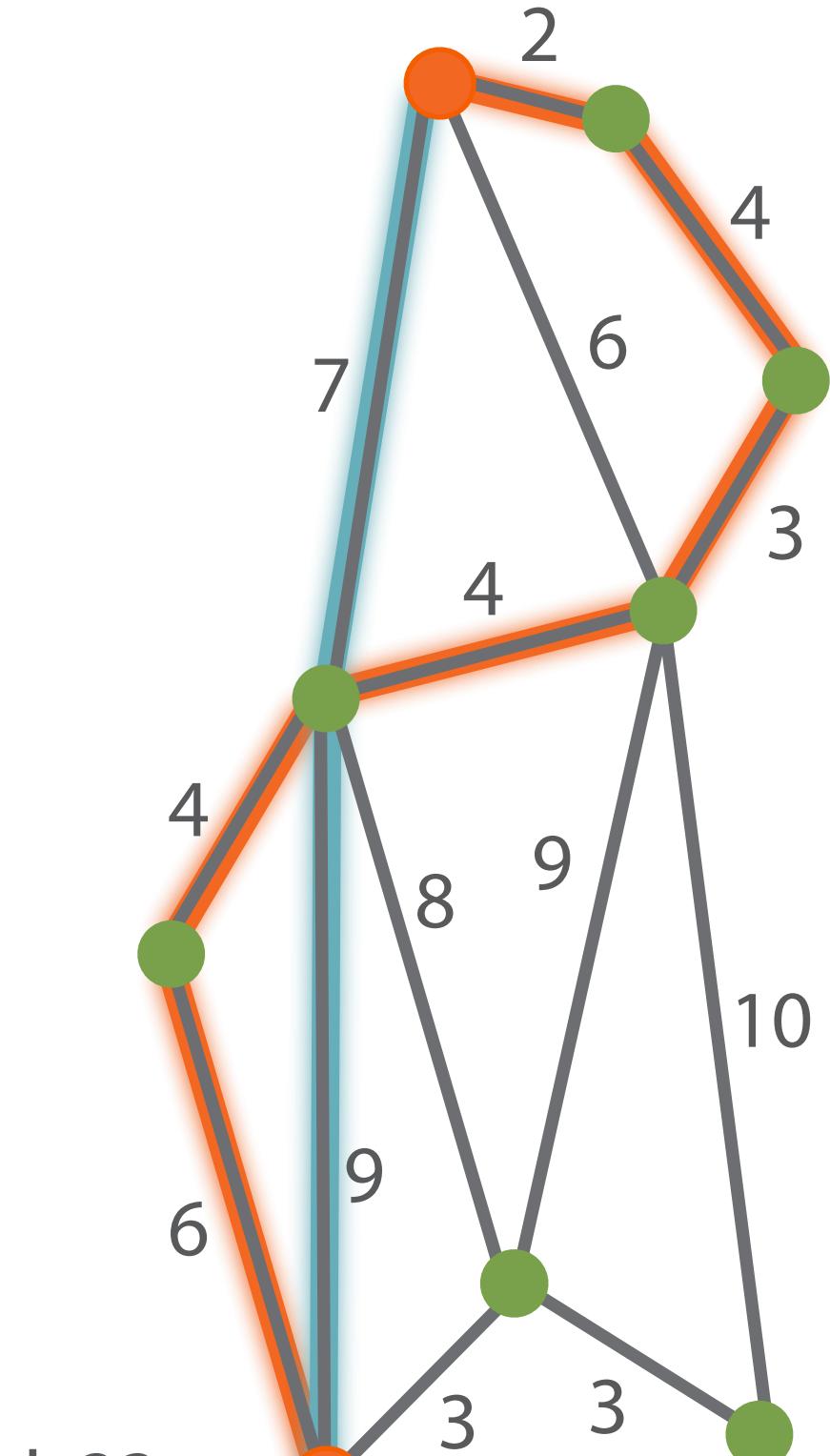
Found: 23  
Shortest: 16



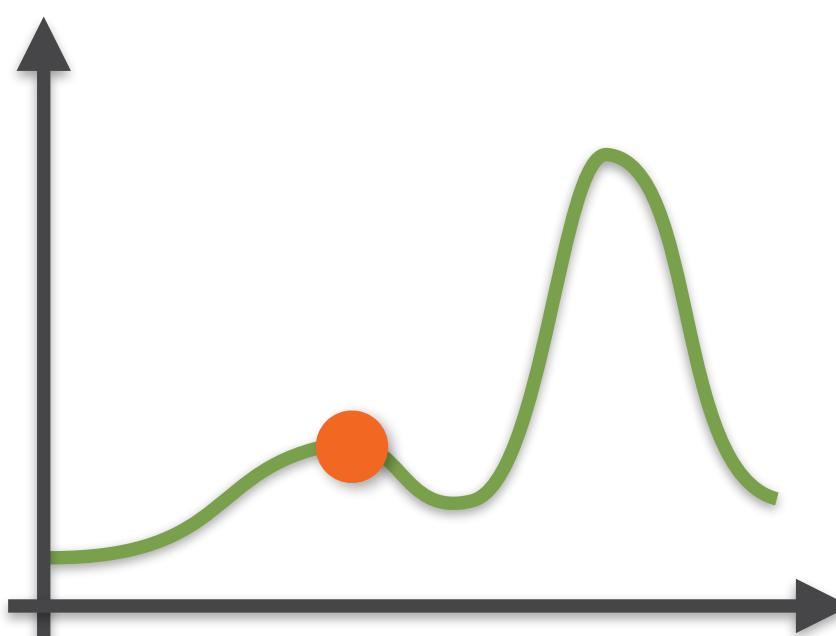
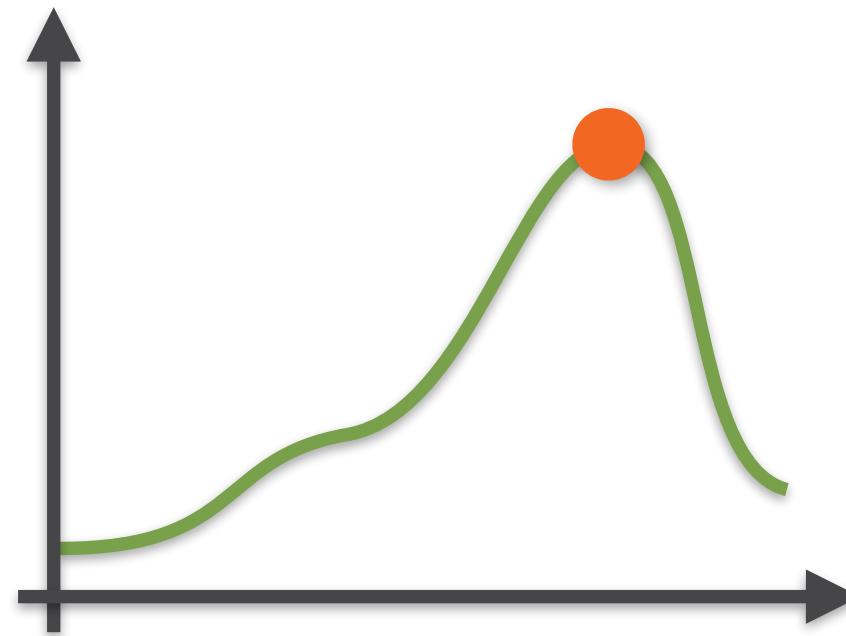
# Greedy Algorithms



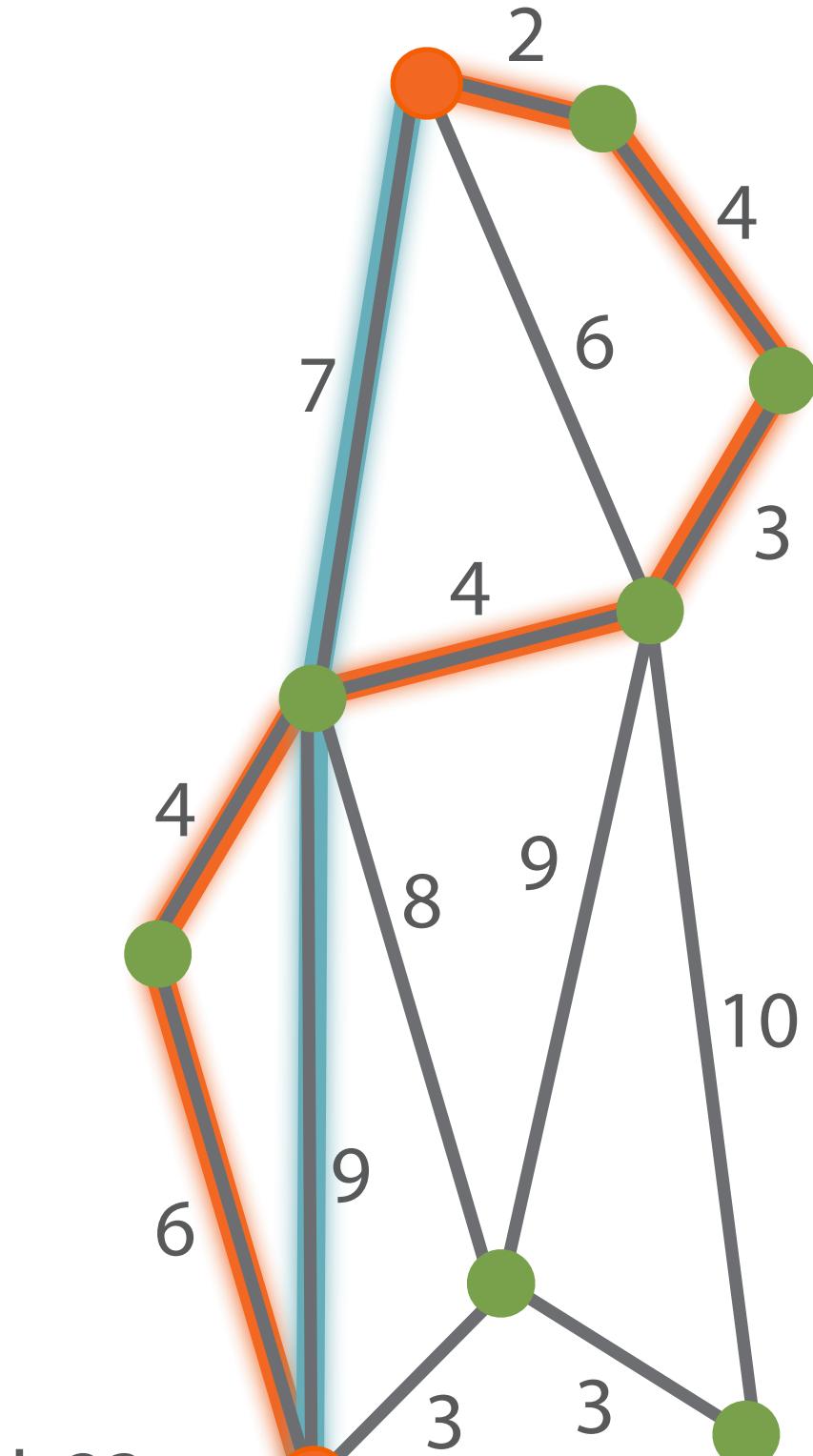
Found: 23  
Shortest: 16



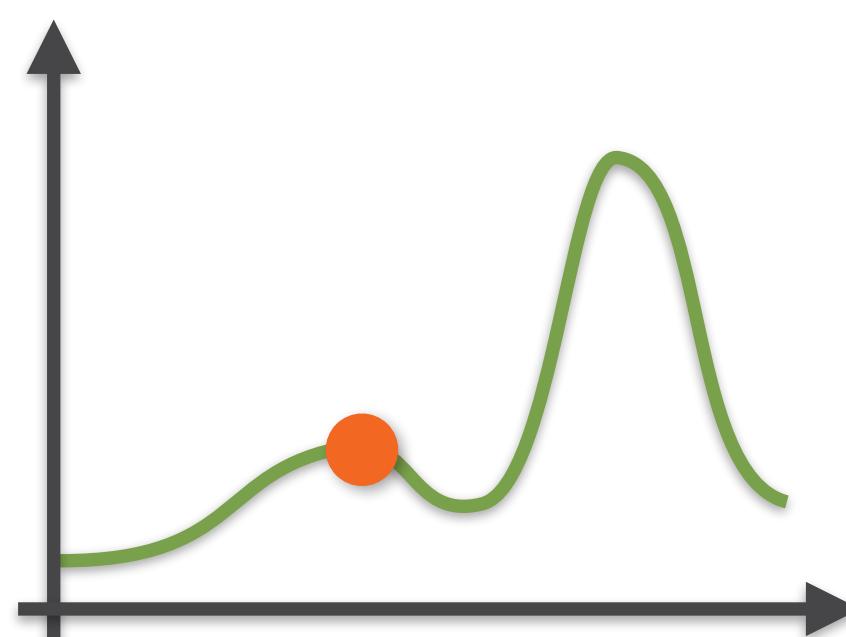
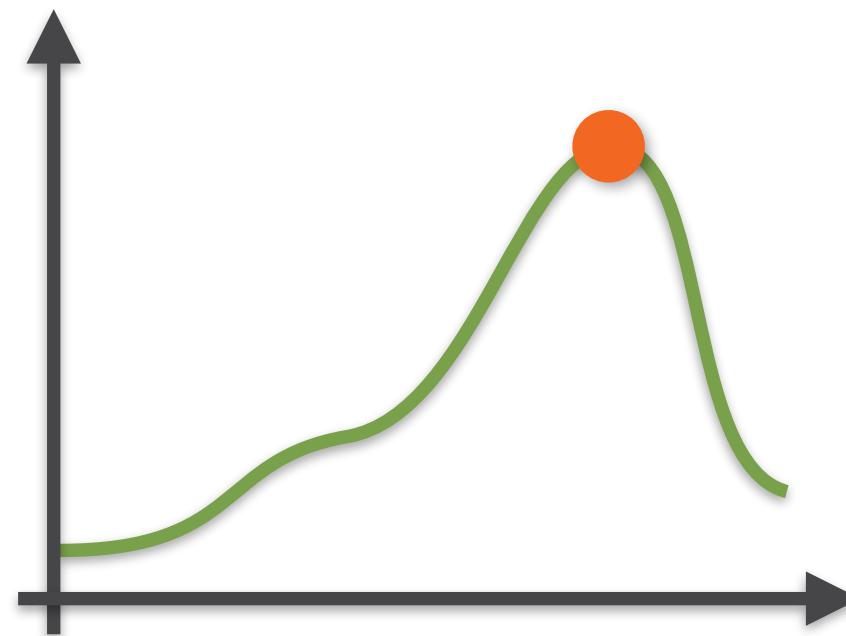
# Greedy Algorithms



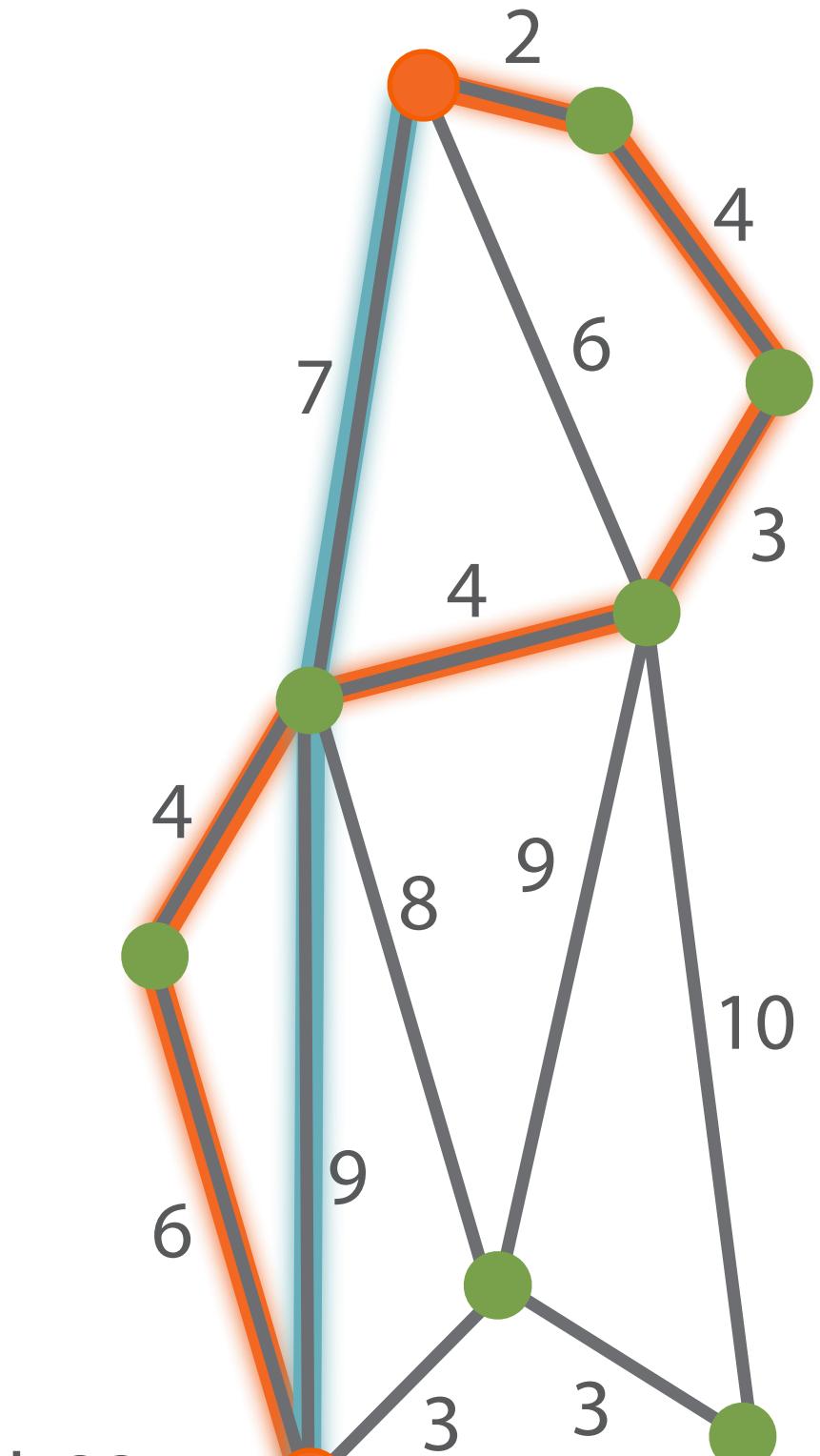
Found: 23  
Shortest: 16



# Greedy Algorithms



Found: 23  
Shortest: 16



# Algorithms

Graph traversal

Brute force  
Greedy algorithms

Divide  
and  
conquer

Dynamic  
programming

Branch  
and  
bound

# Algorithms

Divide  
and  
conquer

Graph traversal

Dynamic  
gramming

Brute force  
Greedy algorithms

Branch  
and  
bound

# Main Principle

# Main Principle



# Main Principle



Divide into  
sub-problems



# Main Principle



Divide into  
sub-problems



Solve sub-problems



# Main Principle



Divide into  
sub-problems



Solve sub-problems



Deduce the  
“large” solution  
(and conquer)



# Apply Recursively

# Apply Recursively

?

# Apply Recursively

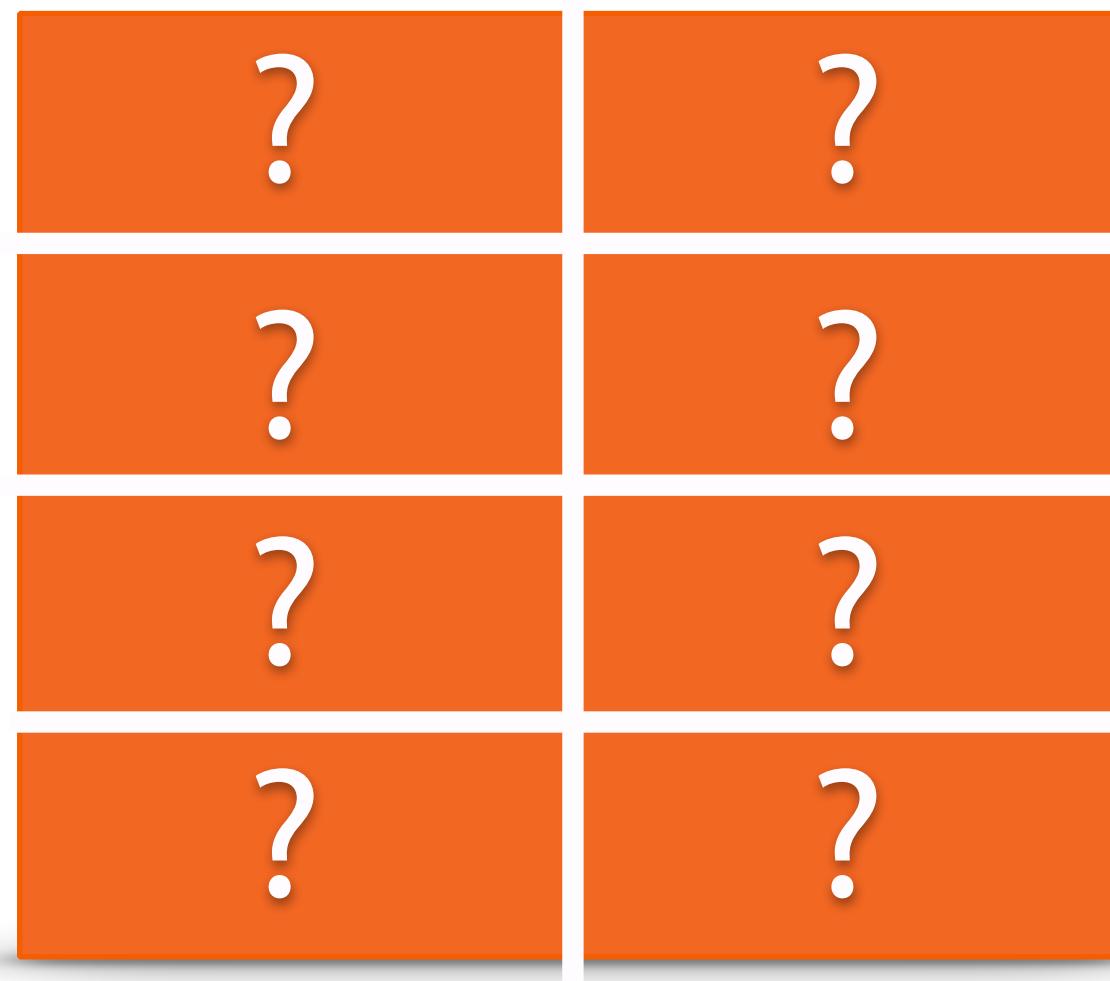
?

?

# Apply Recursively



# Apply Recursively



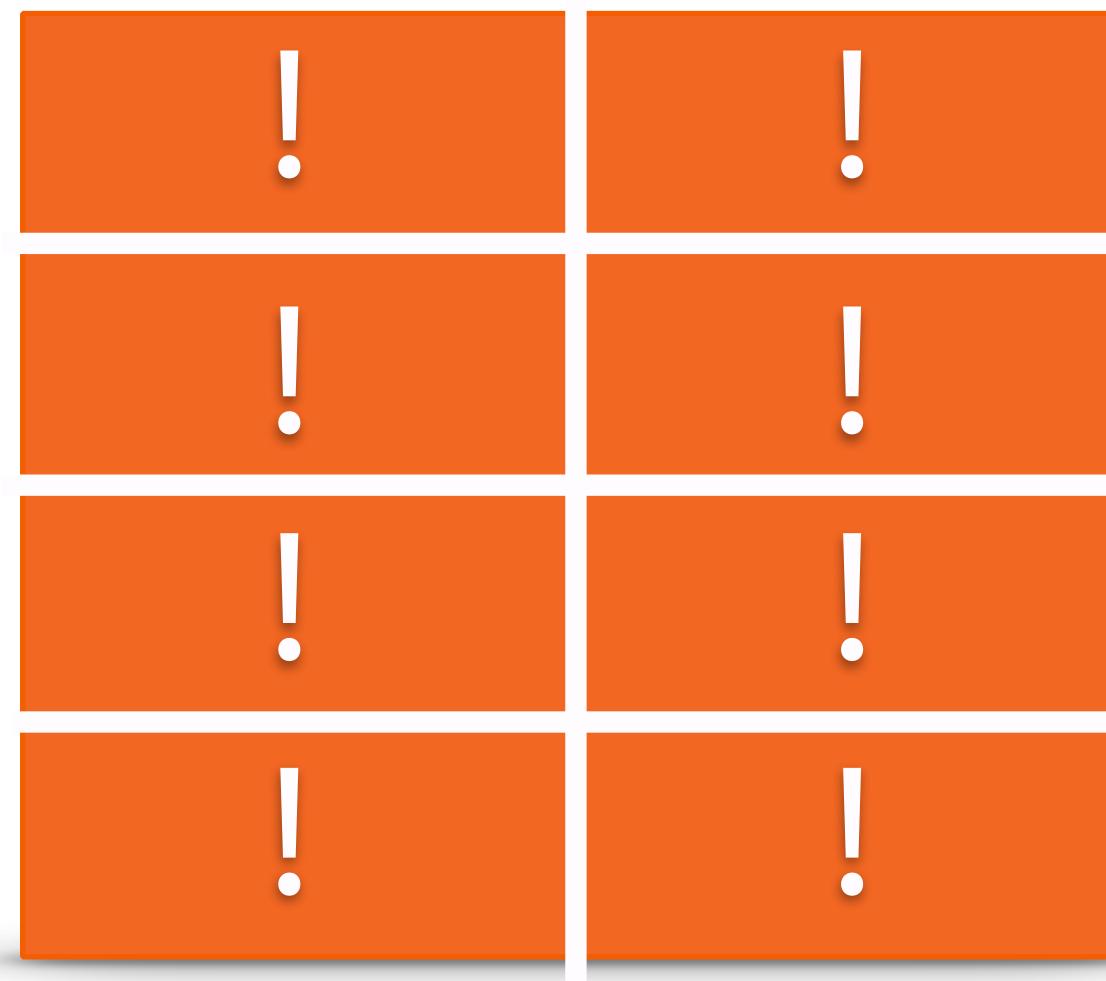
# Apply Recursively

?	?	?	?
?	?	?	?
?	?	?	?
?	?	?	?

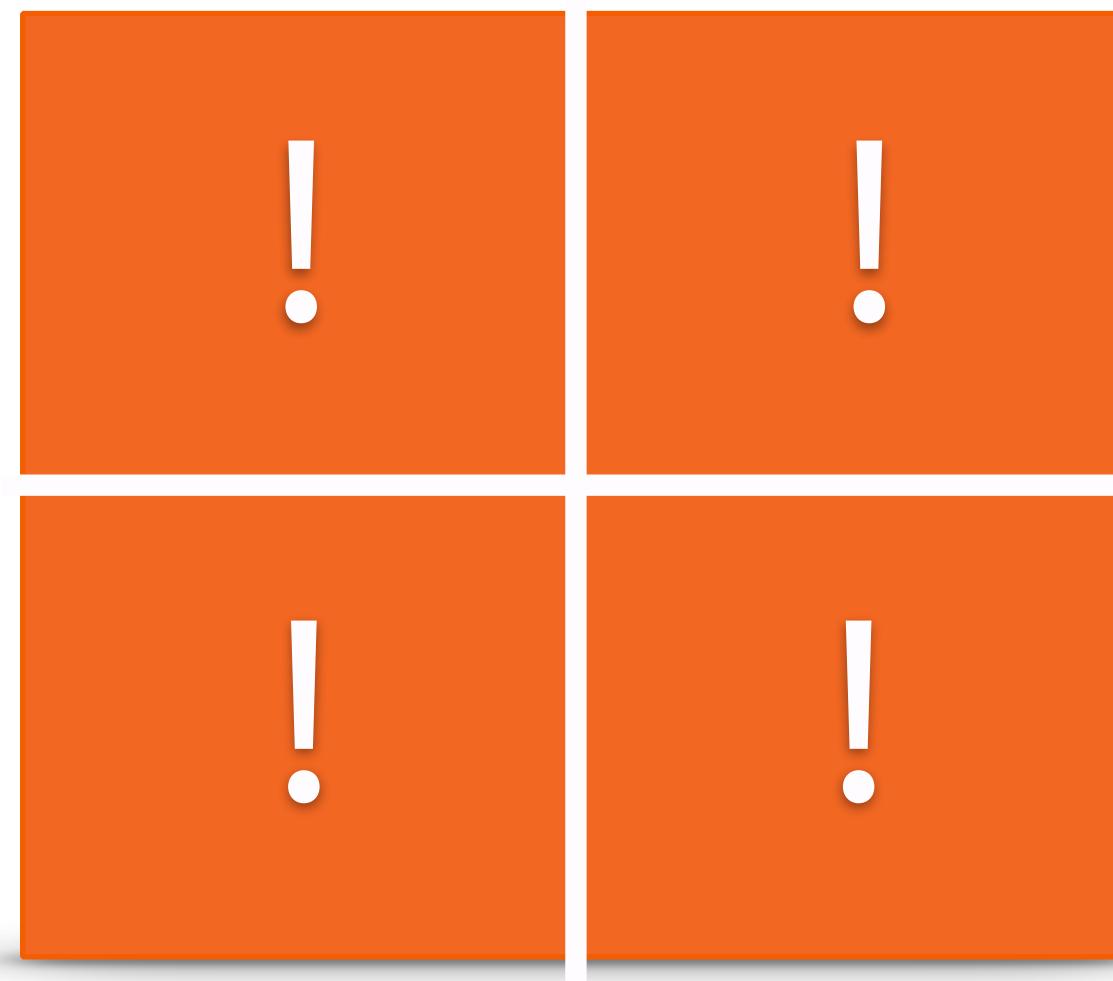
# Apply Recursively



# Apply Recursively



# Apply Recursively



# Apply Recursively



!



!

# Apply Recursively



!

# Apply Recursively



How to...

# Apply Recursively



How to...

divide into sub-problems?

# Apply Recursively



How to...

divide into sub-problems?

solve micro-problem?

# Apply Recursively



How to...

divide into sub-problems?

solve micro-problem?

assemble parent solution?

# Sorting – Quicksort

How to...

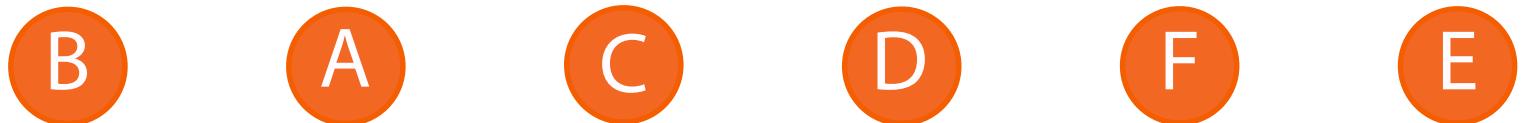
divide into sub-problems?

solve micro-problem?

assemble parent solution?

# Sorting – Quicksort

How to...



divide into sub-problems?

solve micro-problem?

assemble parent solution?

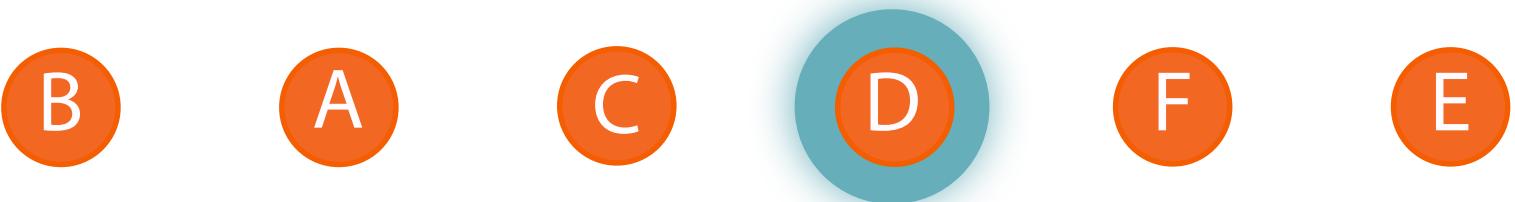
# Sorting – Quicksort

How to...

divide into sub-problems?

solve micro-problem?

assemble parent solution?

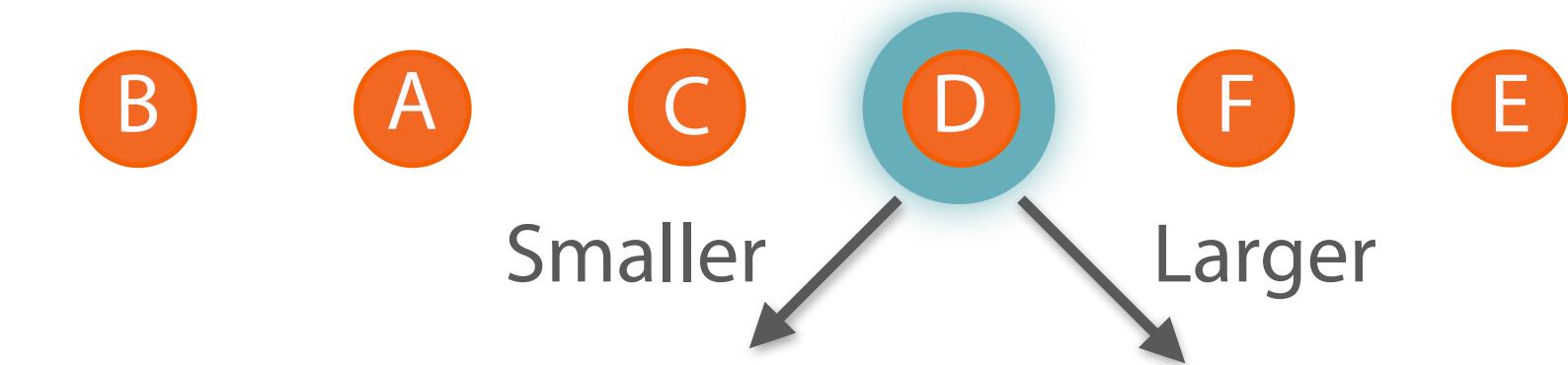


# Sorting – Quicksort

How to...

divide into sub-problems?

solve micro-problem?



assemble parent solution?

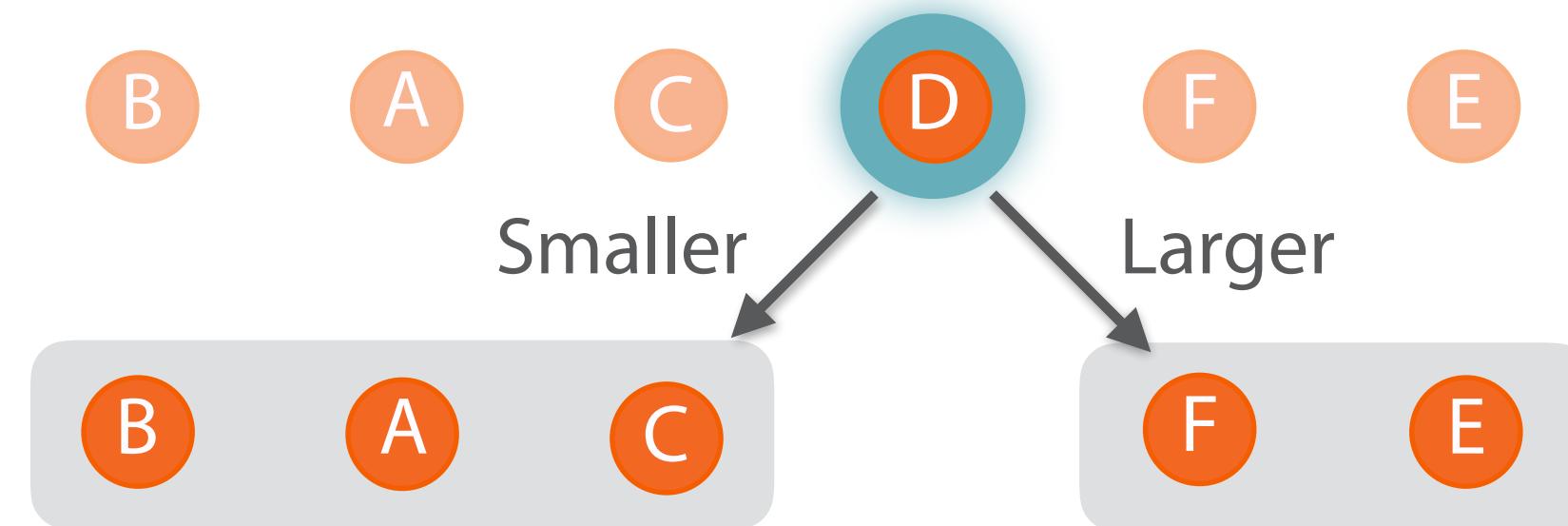
# Sorting – Quicksort

How to...

divide into sub-problems?

solve micro-problem?

assemble parent solution?



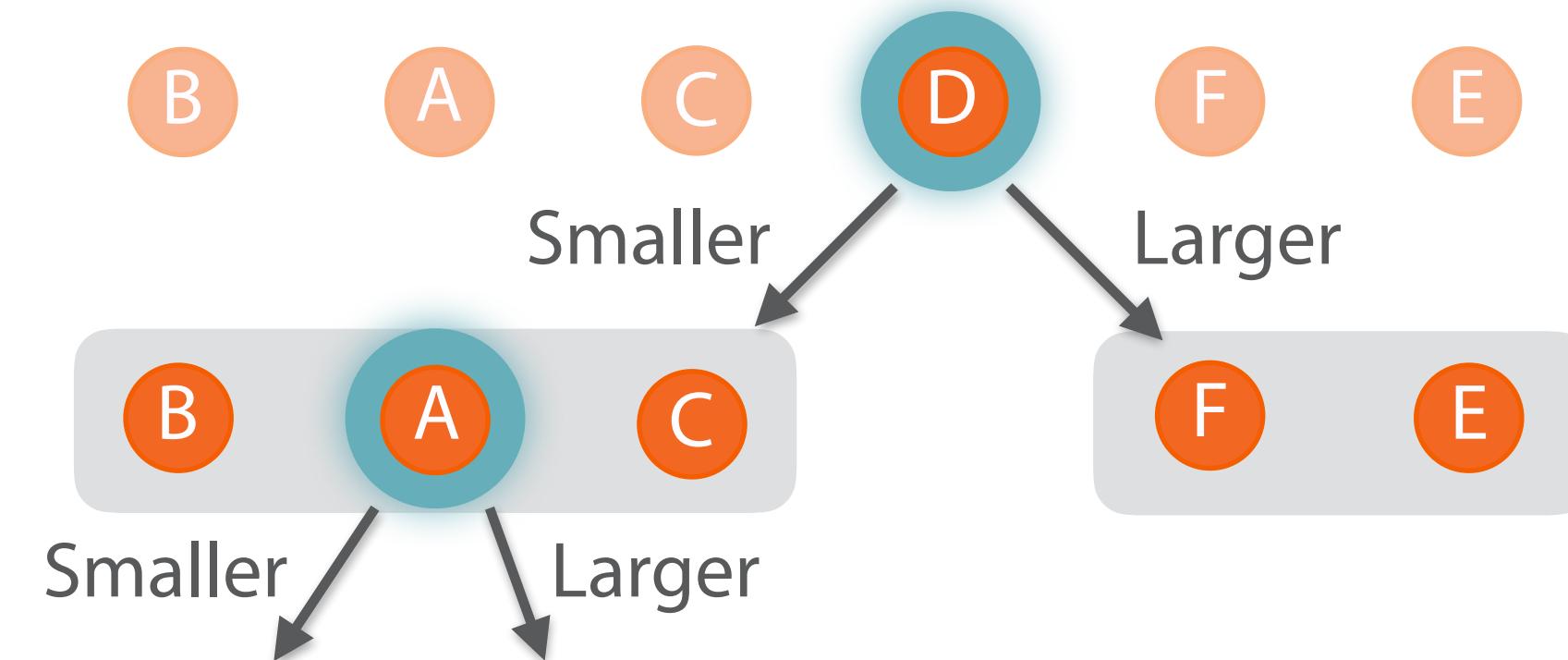
# Sorting – Quicksort

How to...

divide into sub-problems?

solve micro-problem?

assemble parent solution?



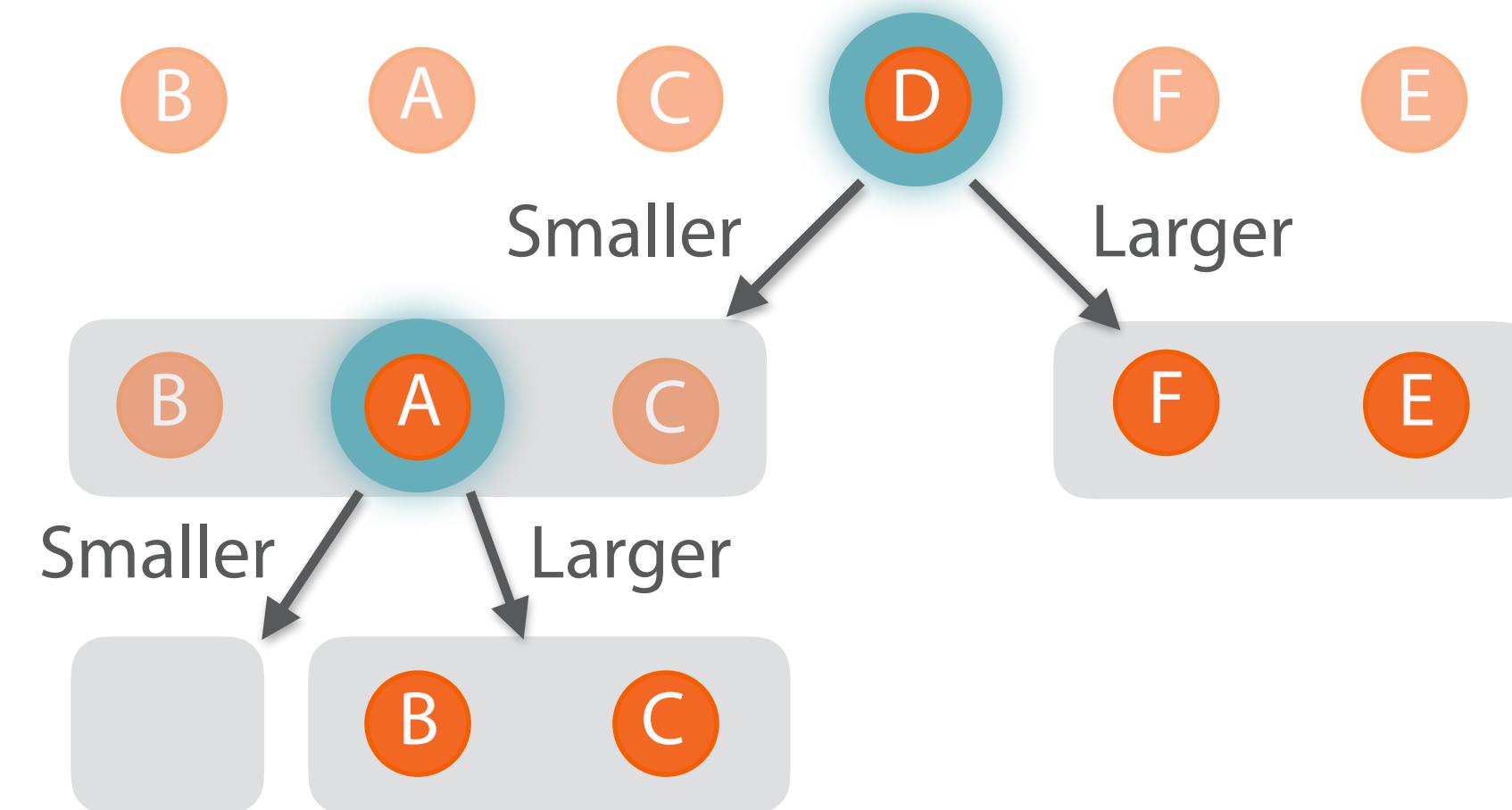
# Sorting – Quicksort

How to...

divide into sub-problems?

solve micro-problem?

assemble parent solution?



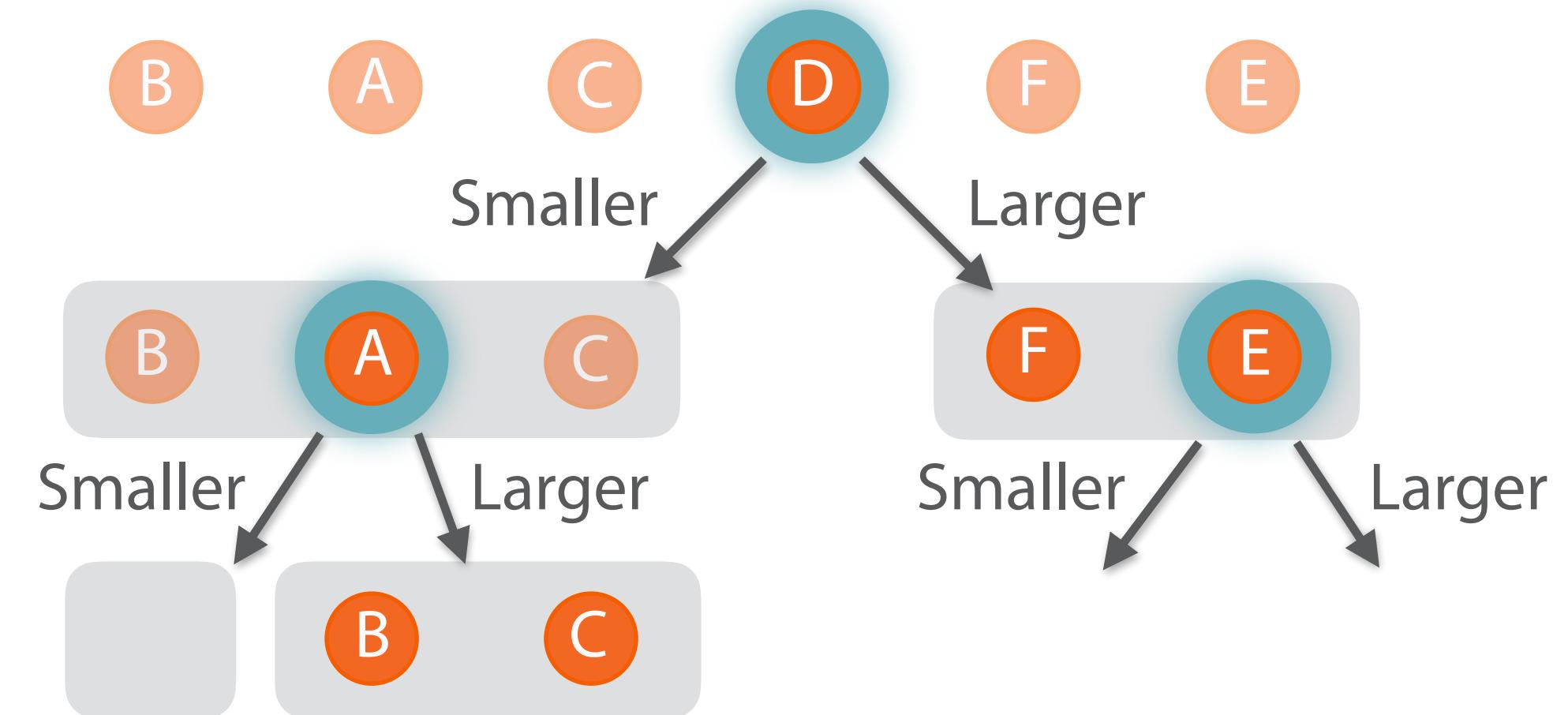
# Sorting – Quicksort

How to...

divide into sub-problems?

solve micro-problem?

assemble parent solution?



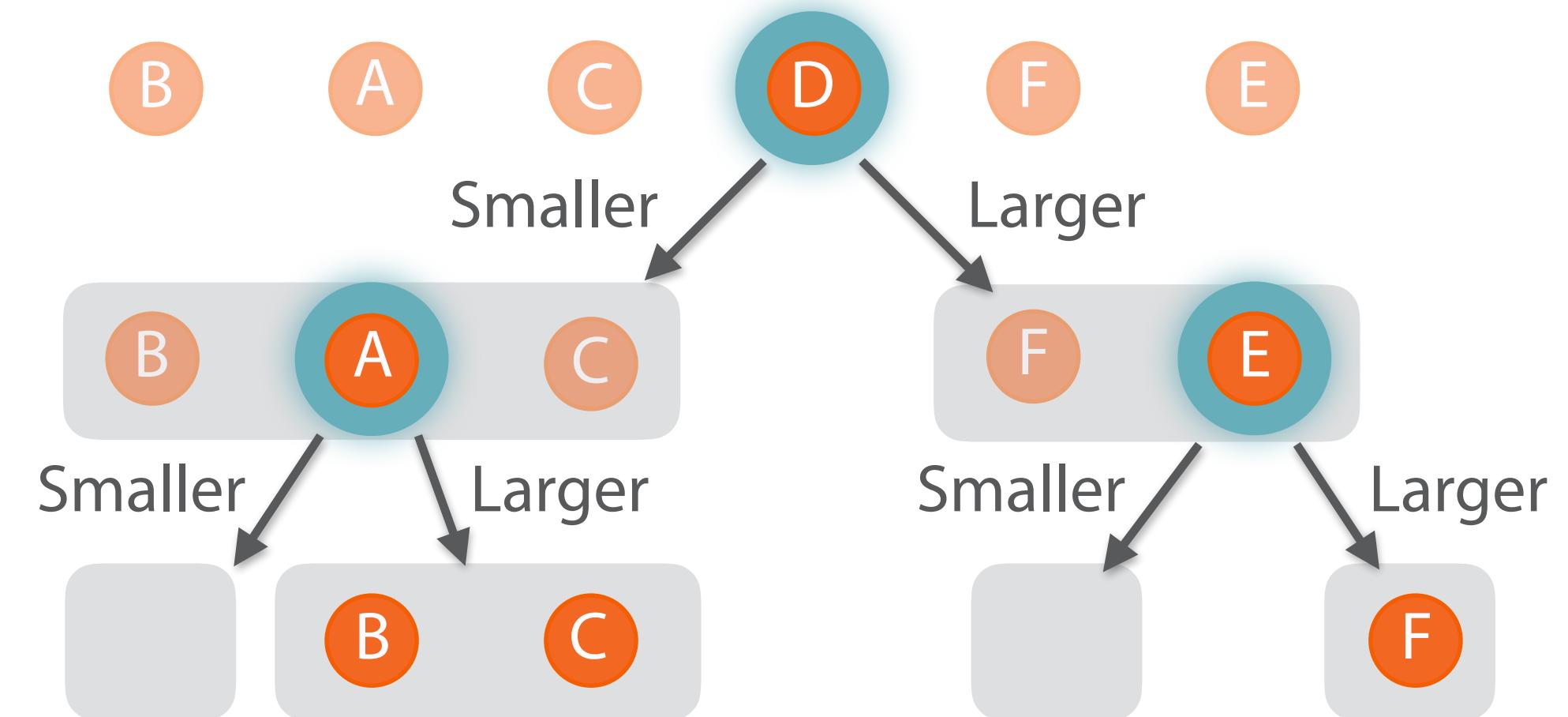
# Sorting – Quicksort

How to...

divide into sub-problems?

solve micro-problem?

assemble parent solution?



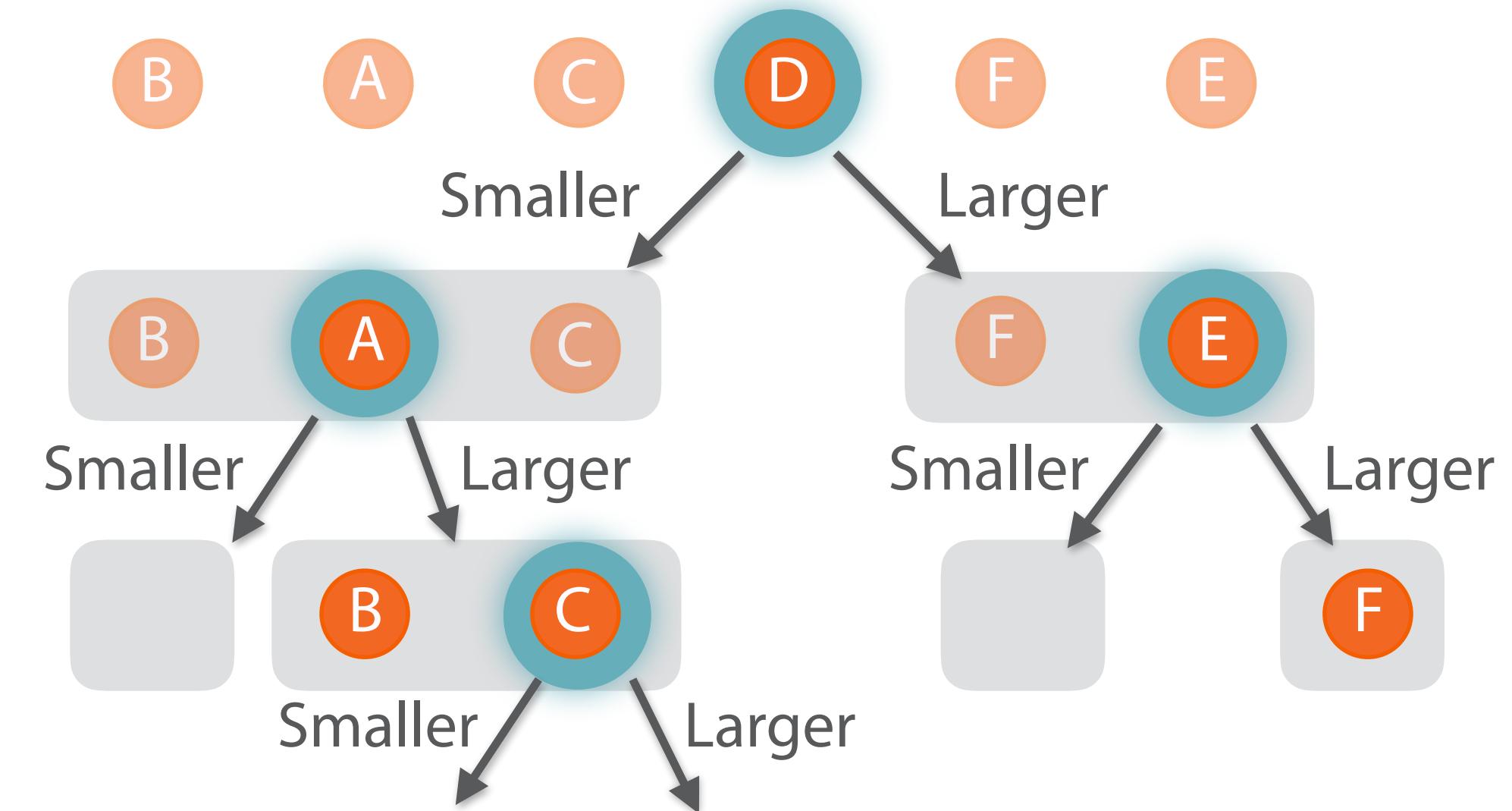
# Sorting – Quicksort

How to...

divide into sub-problems?

solve micro-problem?

assemble parent solution?



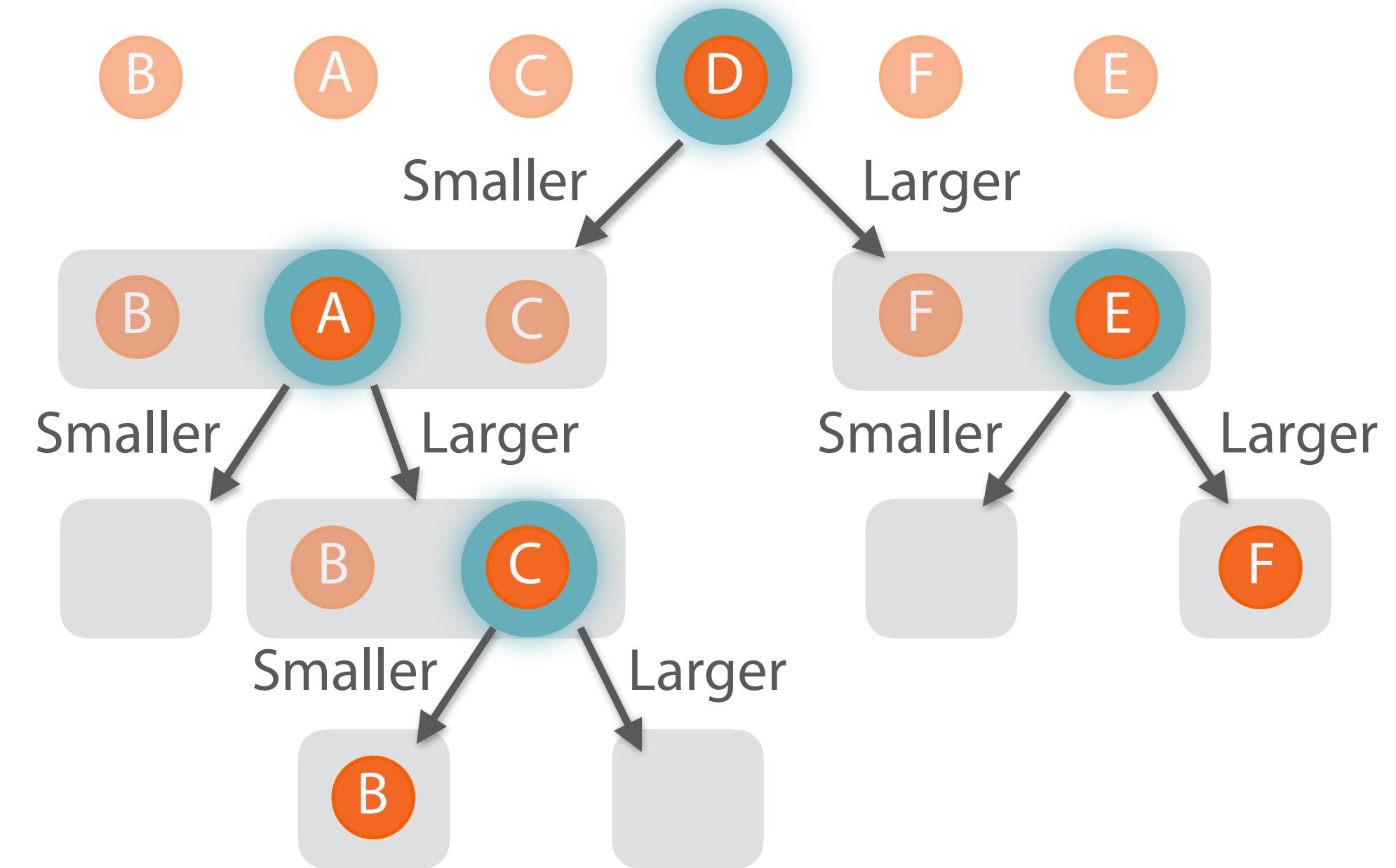
# Sorting – Quicksort

How to...

divide into sub-problems?

solve micro-problem?

assemble parent solution?



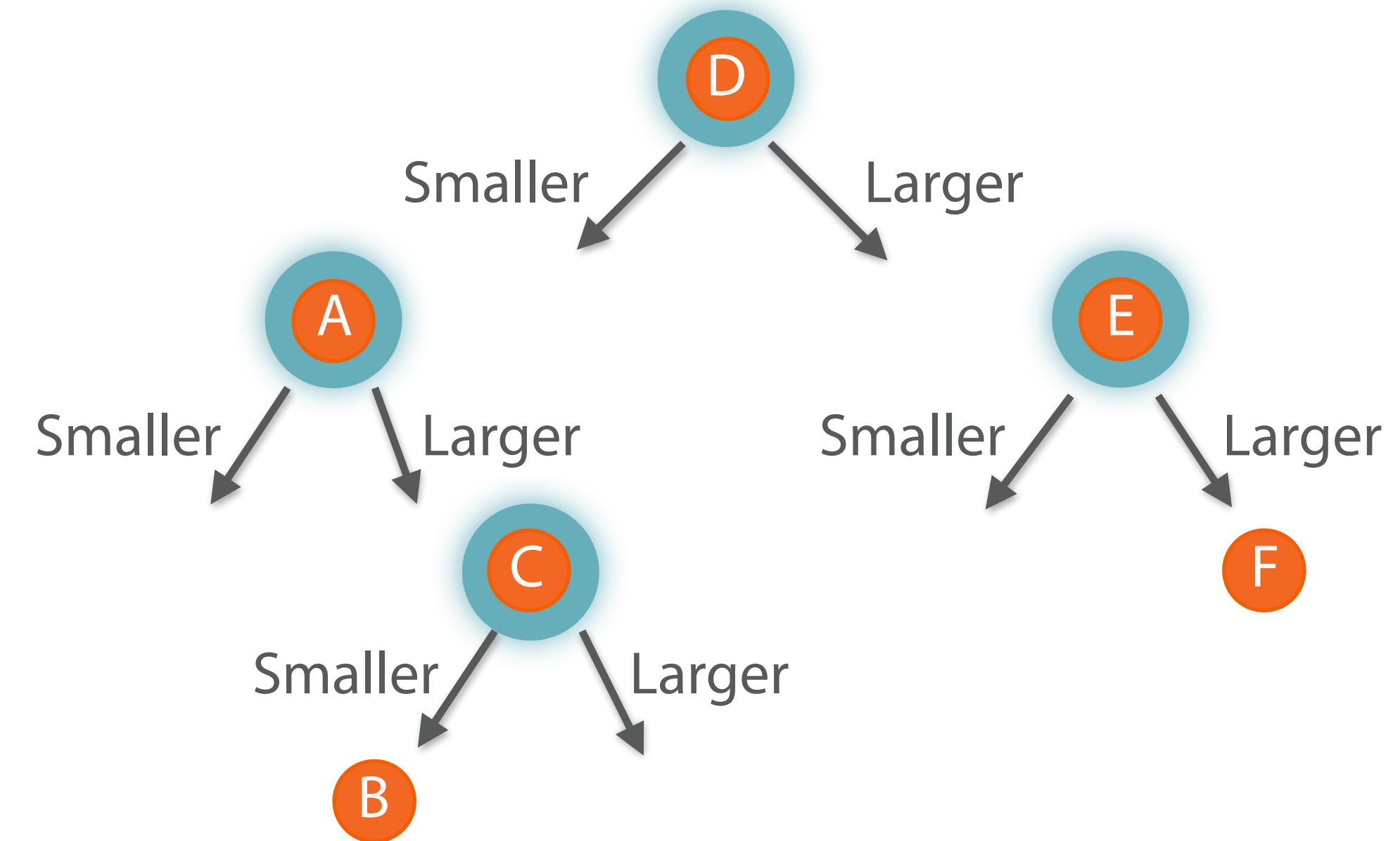
# Sorting – Quicksort

How to...

divide into sub-problems?

solve micro-problem?

assemble parent solution?



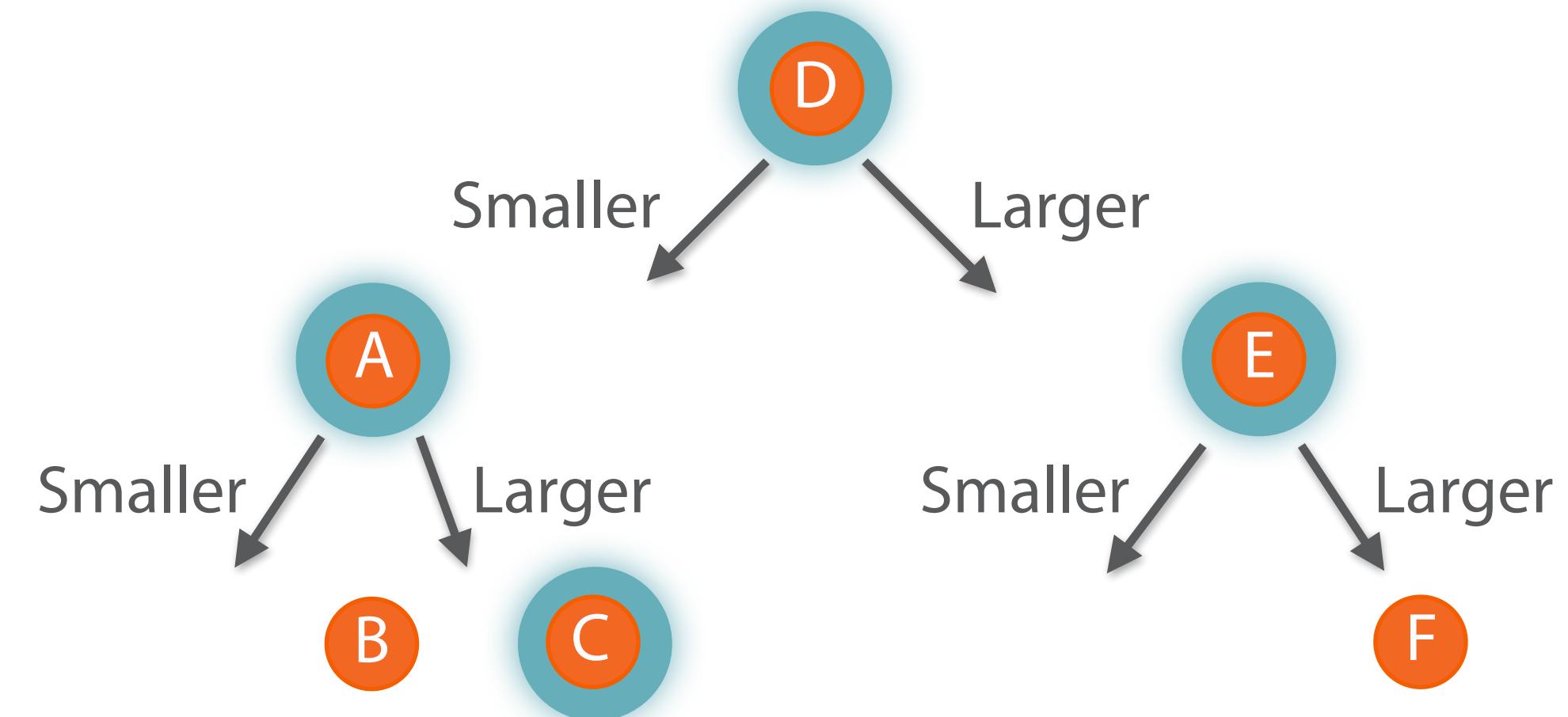
# Sorting – Quicksort

How to...

divide into sub-problems?

solve micro-problem?

assemble parent solution?

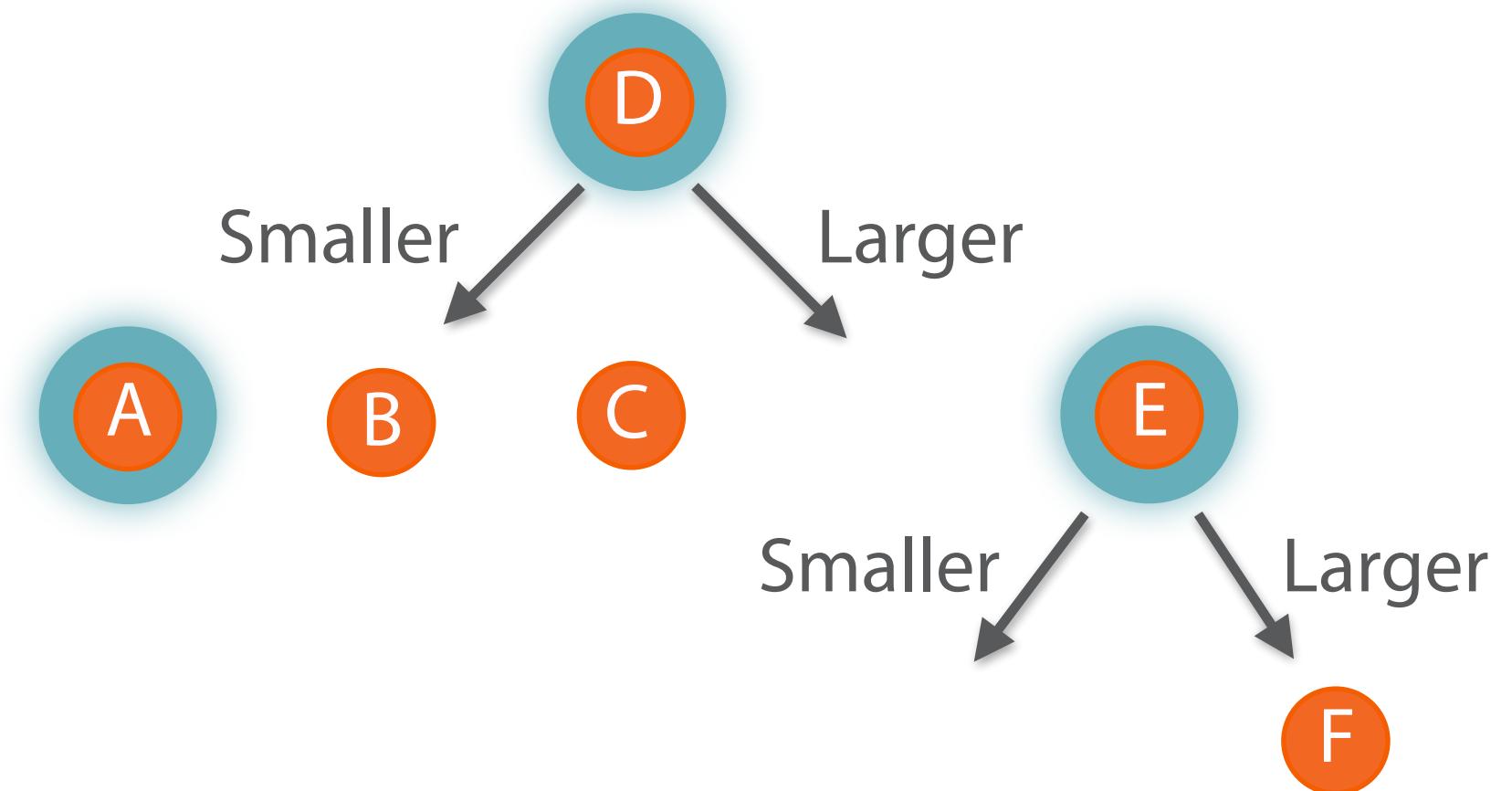


# Sorting – Quicksort

How to...

divide into sub-problems?

solve micro-problem?



assemble parent solution?

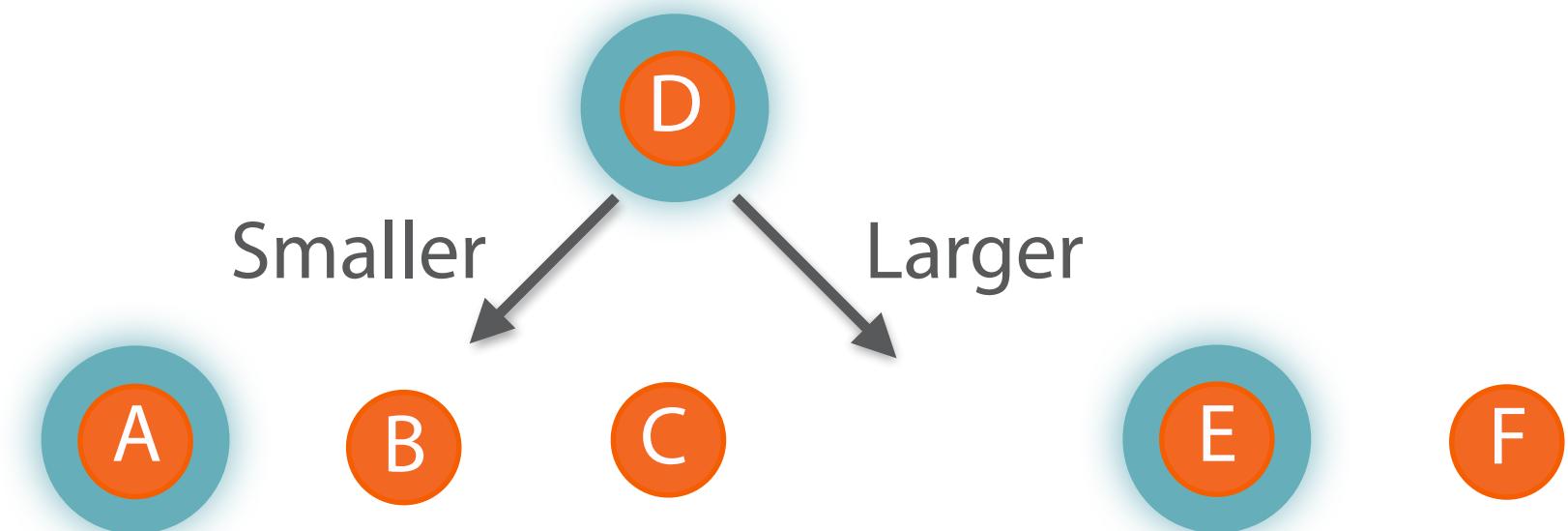
# Sorting – Quicksort

How to...

divide into sub-problems?

solve micro-problem?

assemble parent solution?

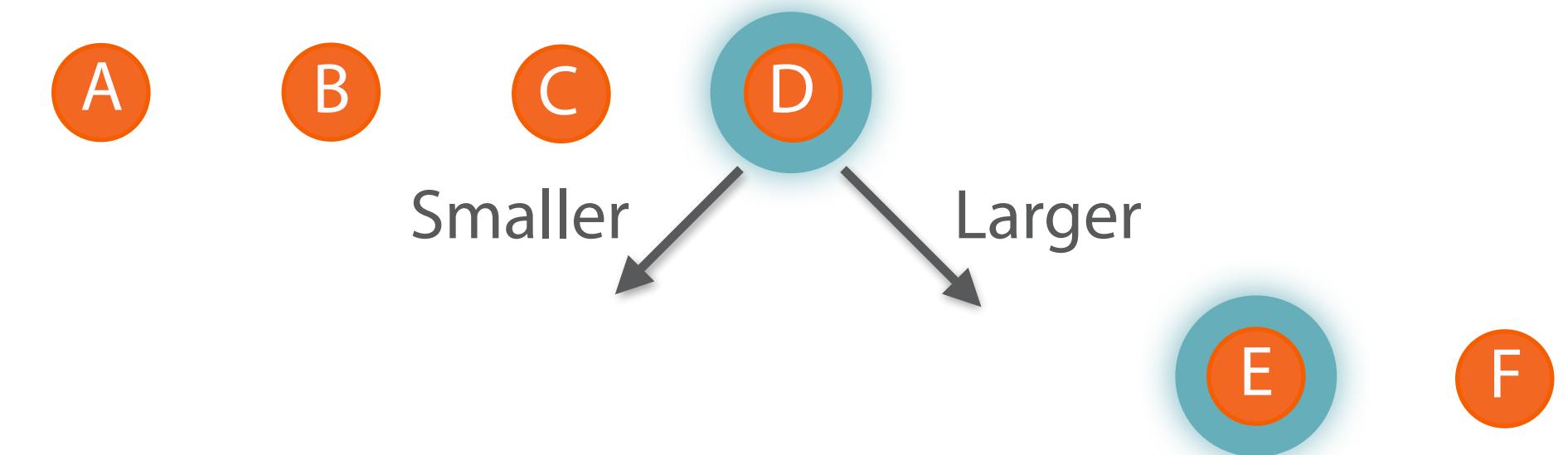


# Sorting – Quicksort

How to...

divide into sub-problems?

solve micro-problem?



assemble parent solution?

# Sorting – Quicksort

How to...



divide into sub-problems?

solve micro-problem?

assemble parent solution?

# Sorting – Quicksort

How to...



divide into sub-problems?

Pick an element, and split into a  
“smaller than” and a “larger than” array.

solve micro-problem?

assemble parent solution?

# Sorting – Quicksort

How to...



divide into sub-problems?

Pick an element, and split into a  
“smaller than” and a “larger than” array.

solve micro-problem?

Just return the *single* element.

assemble parent solution?

# Sorting – Quicksort

How to...



divide into sub-problems?

Pick an element, and split into a  
“smaller than” and a “larger than” array.

solve micro-problem?

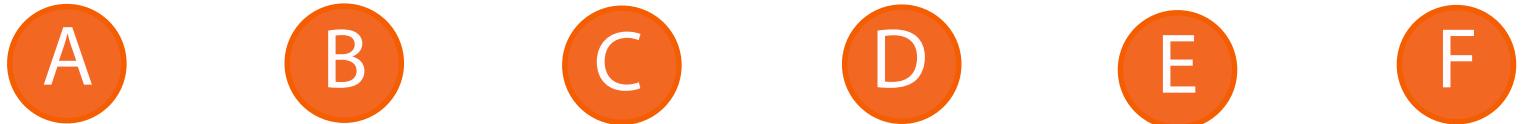
Just return the *single* element.

assemble parent solution?

Merge picked element with the “smaller than”  
and “larger than” sub-solutions.

# Sorting – Quicksort

How to...



divide into sub-problems?

Pick an element, and split into a  
“smaller than” and a “larger than” array.

solve micro-problem?

Just return the *single* element.

assemble parent solution?

Merge picked element with the “smaller than”  
and “larger than” sub-solutions.

# Sorting – Quicksort

How to...



$N$  elements in the list

divide into sub-problems?

Pick an element, and split into a “smaller than” and a “larger than” array.

**Worst case**

solve micro-problem?

Just return the *single* element.

**Average case**

assemble parent solution?

Merge picked element with the “smaller than” and “larger than” sub-solutions.

# Sorting – Quicksort

How to...



divide into sub-problems?

Pick an element, and split into a “smaller than” and a “larger than” array.

solve micro-problem?

Just return the *single* element.

**Worst case**

$N$  levels

assemble parent solution?

Merge picked element with the “smaller than” and “larger than” sub-solutions.

**Average case**

# Sorting – Quicksort

How to...



$N$  elements in the list

divide into sub-problems?

Pick an element, and split into a “smaller than” and a “larger than” array.

**Worst case**

$N$  levels

at most  $N$  elements each

solve micro-problem?

Just return the *single* element.

**Average case**

assemble parent solution?

Merge picked element with the “smaller than” and “larger than” sub-solutions.

# Sorting – Quicksort

How to...



$N$  elements in the list

divide into sub-problems?

Pick an element, and split into a “smaller than” and a “larger than” array.

**Worst case**

$N$  levels

at most  $N$  elements each

$O(N^2)$

solve micro-problem?

Just return the *single* element.

**Average case**

assemble parent solution?

Merge picked element with the “smaller than” and “larger than” sub-solutions.

# Sorting – Quicksort

How to...



divide into sub-problems?

Pick an element, and split into a “smaller than” and a “larger than” array.

solve micro-problem?

Just return the *single* element.

assemble parent solution?

Merge picked element with the “smaller than” and “larger than” sub-solutions.

$N$  elements in the list

**Worst case**

$N$  levels

at most  $N$  elements each

$O(N^2)$

**Average case**

$\log_2 N$  levels

# Sorting – Quicksort

How to...



$N$  elements in the list

divide into sub-problems?

Pick an element, and split into a “smaller than” and a “larger than” array.

**Worst case**

$N$  levels

at most  $N$  elements each

$O(N^2)$

solve micro-problem?

Just return the *single* element.

**Average case**

$\log_2 N$  levels

at most  $N$  elements each

assemble parent solution?

Merge picked element with the “smaller than” and “larger than” sub-solutions.

# Sorting – Quicksort

How to...



$N$  elements in the list

divide into sub-problems?

Pick an element, and split into a “smaller than” and a “larger than” array.

**Worst case**

$N$  levels

at most  $N$  elements each

$O(N^2)$

solve micro-problem?

Just return the *single* element.

**Average case**

$\log_2 N$  levels

at most  $N$  elements each

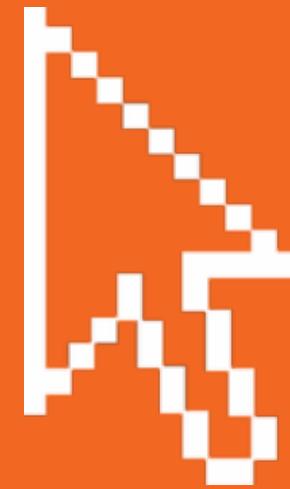
$O(N \cdot \log_2 N)$

assemble parent solution?

Merge picked element with the “smaller than” and “larger than” sub-solutions.

# Quicksort

A code walk-through



# Other Examples

# Other Examples

Other sorting algorithms  
- most notably *merge sort*

# Other Examples

Other sorting algorithms  
- most notably *merge sort*

1, 4, 7, 10

2, 3, 8, 9

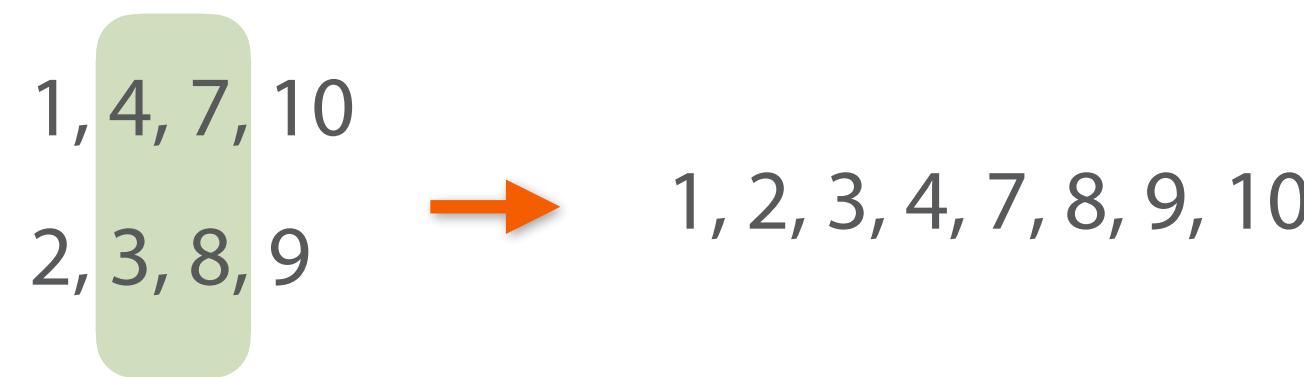
# Other Examples

Other sorting algorithms  
- most notably *merge sort*

1, 4, 7, 10  
2, 3, 8, 9       1, 2, 3, 4, 7, 8, 9, 10

# Other Examples

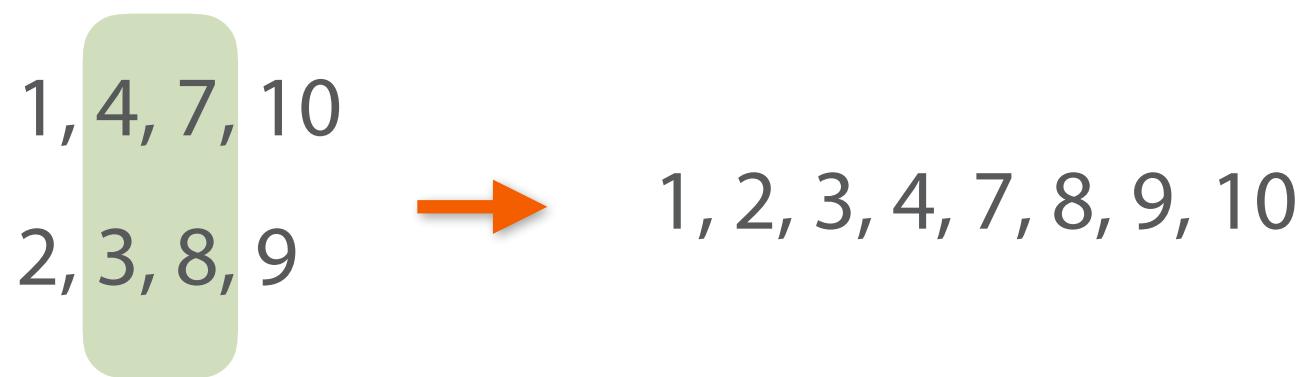
Other sorting algorithms  
- most notably *merge sort*



# Other Examples

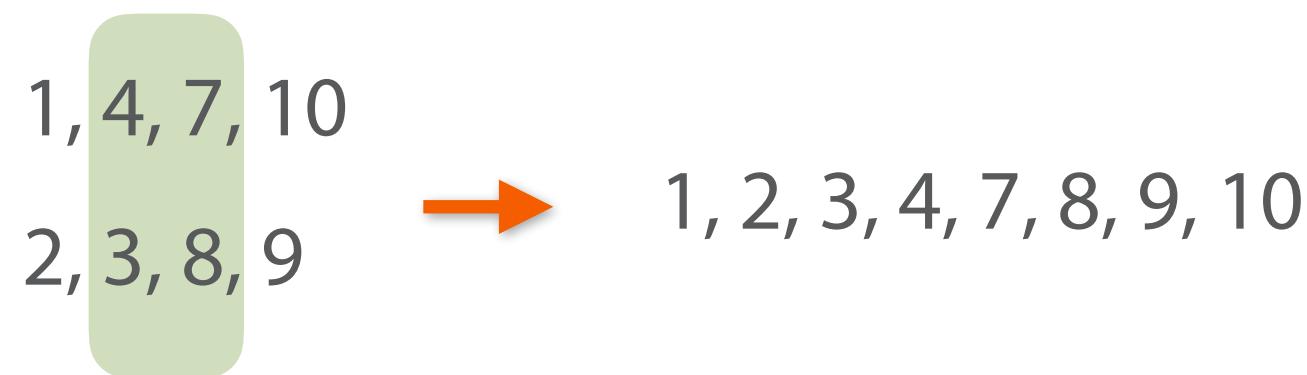
Other sorting algorithms  
- most notably *merge sort*

Binary search



# Other Examples

Other sorting algorithms  
- most notably *merge sort*

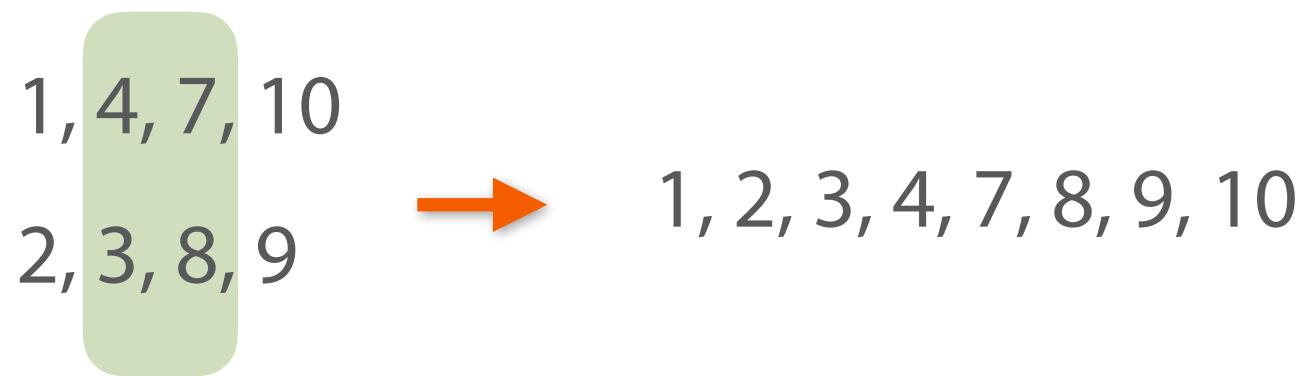


Binary search

1, 12, 32, 84, 127, 828, 1092, 1107, 2398

# Other Examples

Other sorting algorithms  
- most notably *merge sort*

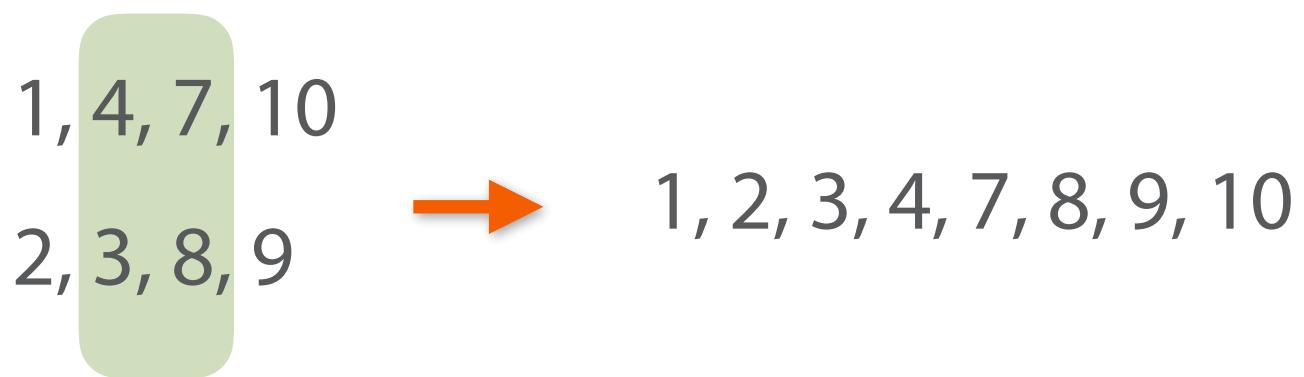


Binary search

1, 12, 32, 84, 127, 828, 1092, 1107, 2398

# Other Examples

Other sorting algorithms  
- most notably *merge sort*



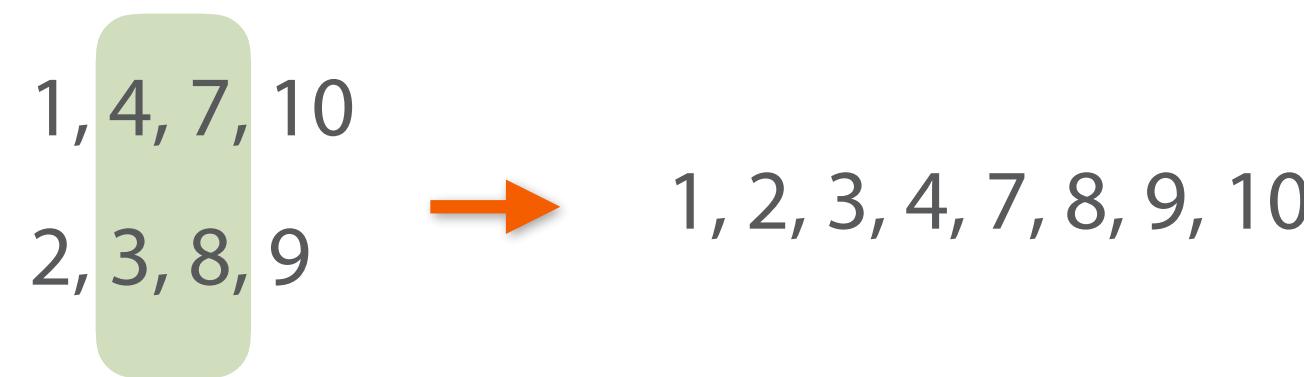
Binary search

1, 12, 32, 84, 127, 828, 1092, 1107, 2398

Multiplication of large numbers and matrices

# Other Examples

Other sorting algorithms  
- most notably *merge sort*



Binary search

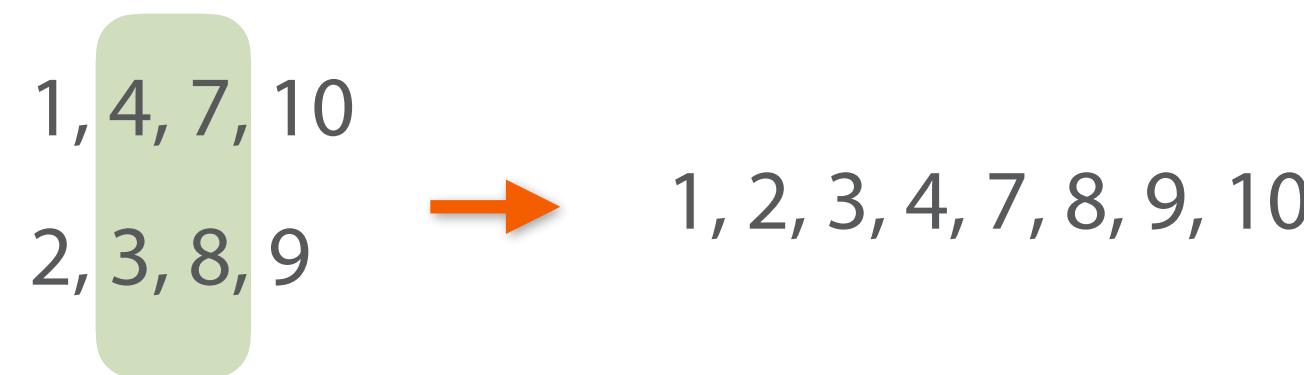
1, 12, 32, 84, 127, 828, 1092, 1107, 2398

Multiplication of large numbers and matrices

8728198751778243 · 7831378239428347

# Other Examples

Other sorting algorithms  
- most notably *merge sort*



Binary search

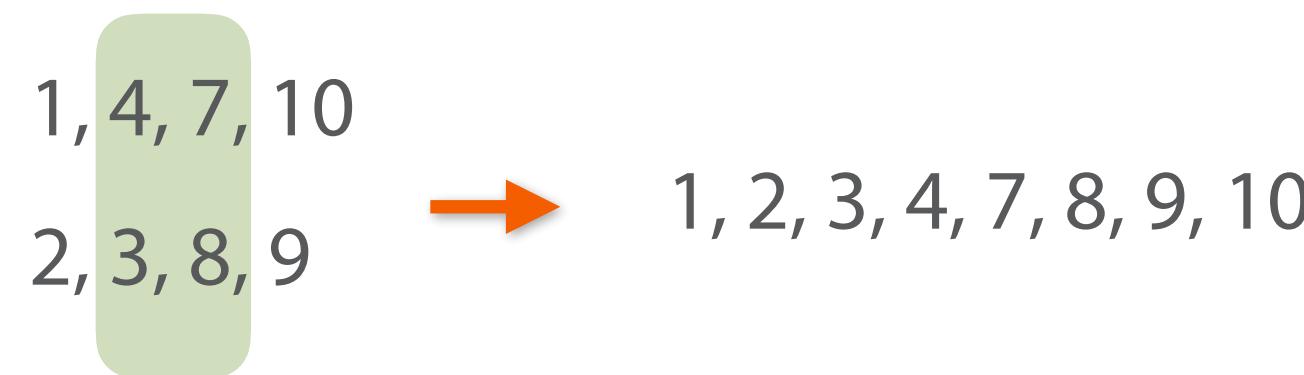
1, 12, 32, 84, 127, 828, 1092, 1107, 2398

Multiplication of large numbers and matrices

8728198751778243 · 7831378239428347

# Other Examples

Other sorting algorithms  
- most notably *merge sort*



Binary search

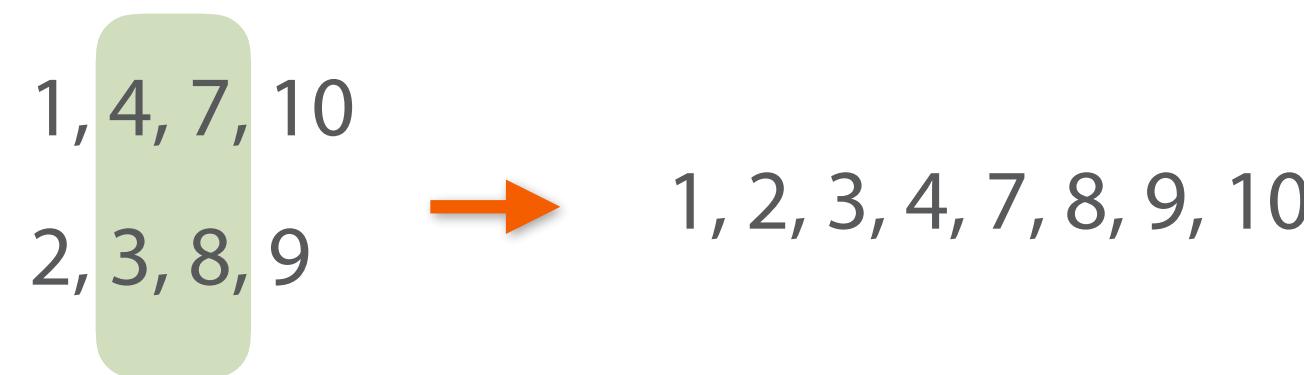
1, 12, 32, 84, 127, 828, 1092, 1107, 2398

Multiplication of large numbers and matrices

8728198751778243 · 7831378239428347

# Other Examples

Other sorting algorithms  
- most notably *merge sort*



Binary search

1, 12, 32, 84, 127, 828, 1092, 1107, 2398

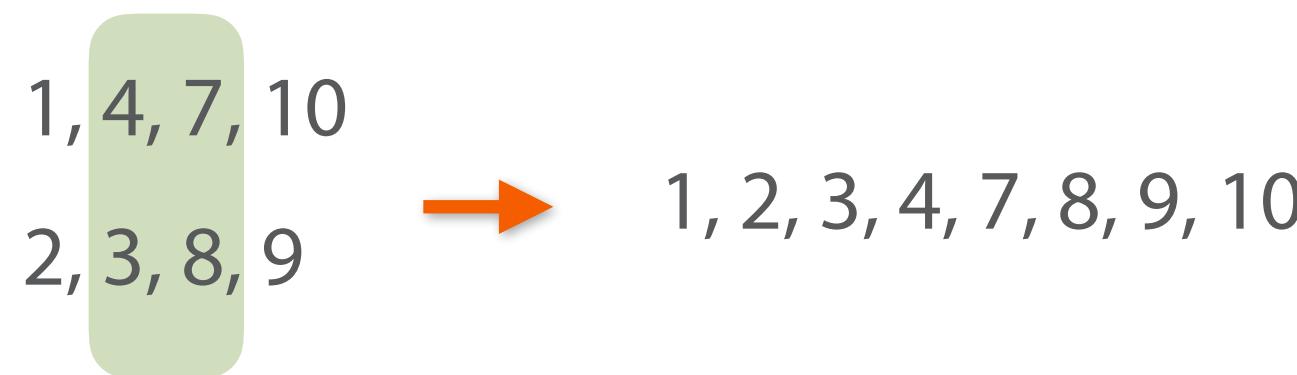
Multiplication of large numbers and matrices

8728198751778243 · 7831378239428347

Find the two closest points in a plane

# Other Examples

Other sorting algorithms  
- most notably *merge sort*



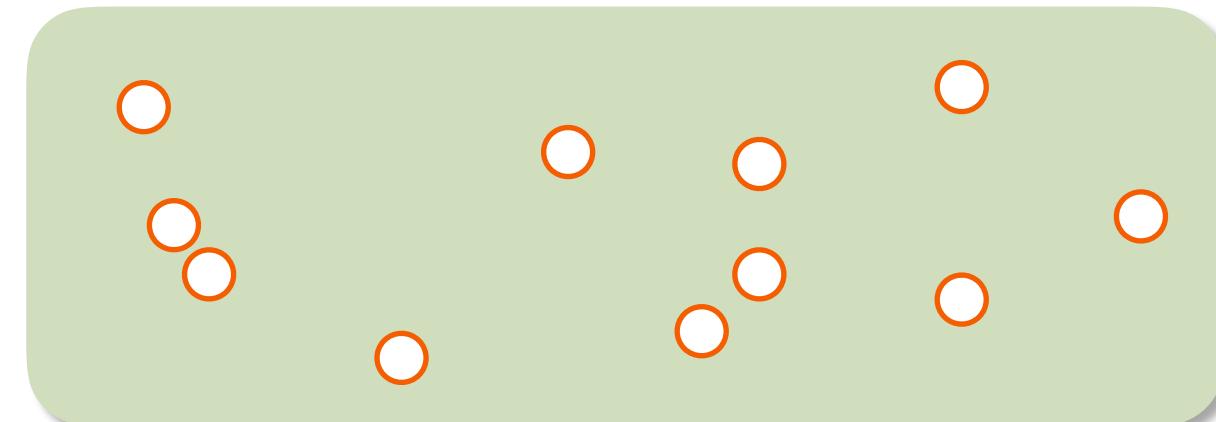
Binary search

1, 12, 32, 84, 127, 828, 1092, 1107, 2398

Multiplication of large numbers and matrices

8728198751778243 · 7831378239428347

Find the two closest points in a plane



# Other Examples

Other sorting algorithms  
- most notably *merge sort*



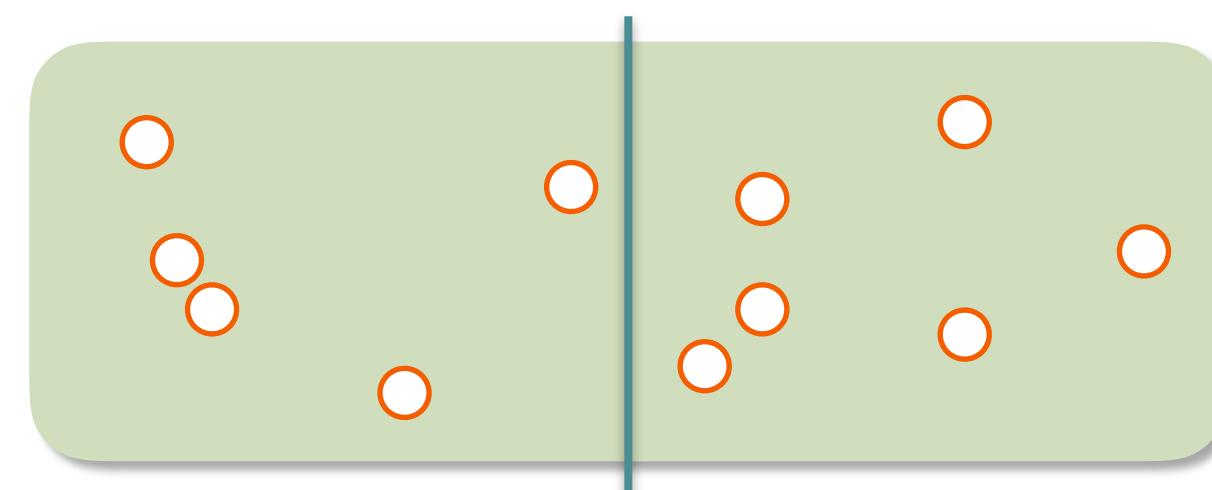
Binary search

1, 12, 32, 84, 127, 828, 1092, 1107, 2398

Multiplication of large numbers and matrices

8728198751778243 · 7831378239428347

Find the two closest points in a plane



# Algorithms

Graph traversal

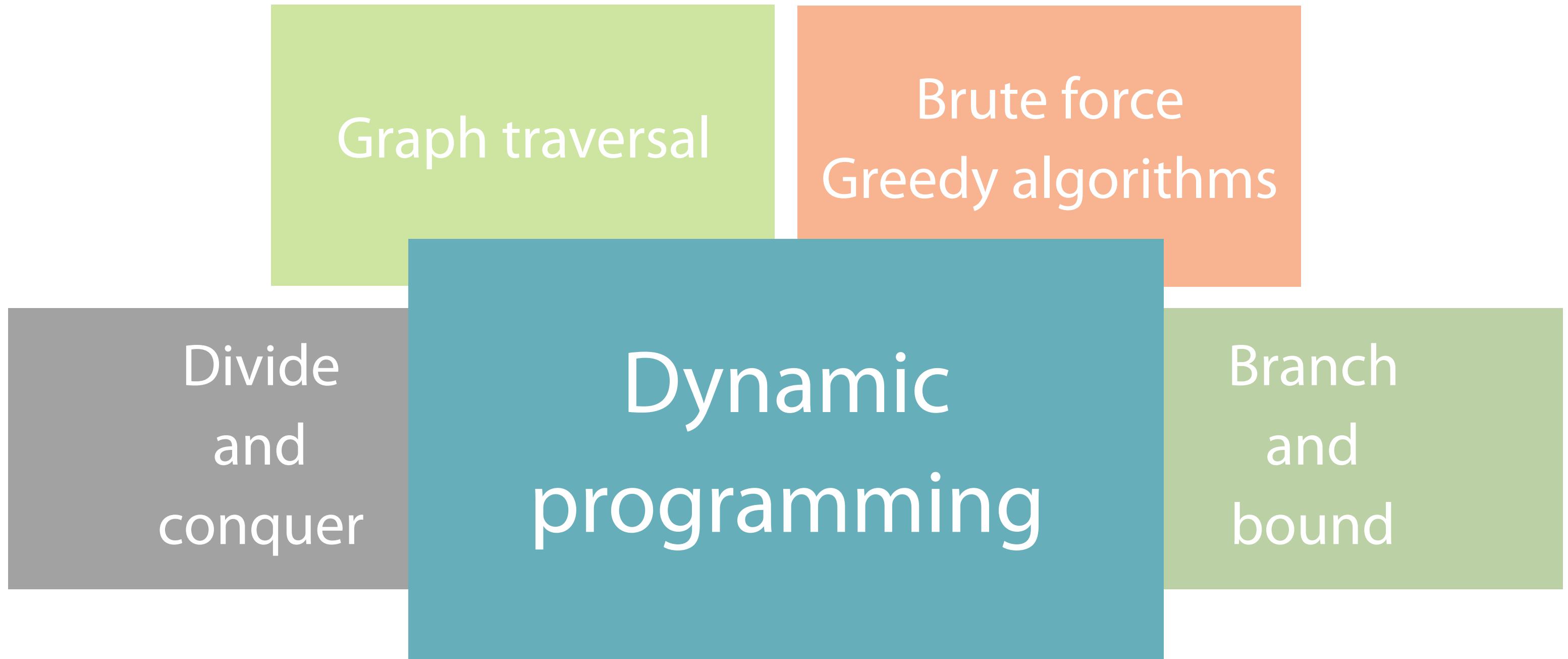
Brute force  
Greedy algorithms

Divide  
and  
conquer

Dynamic  
programming

Branch  
and  
bound

# Algorithms



# Main Principle

# Main Principle



# Main Principle



Divide into  
sub-problems



Solve sub-problems



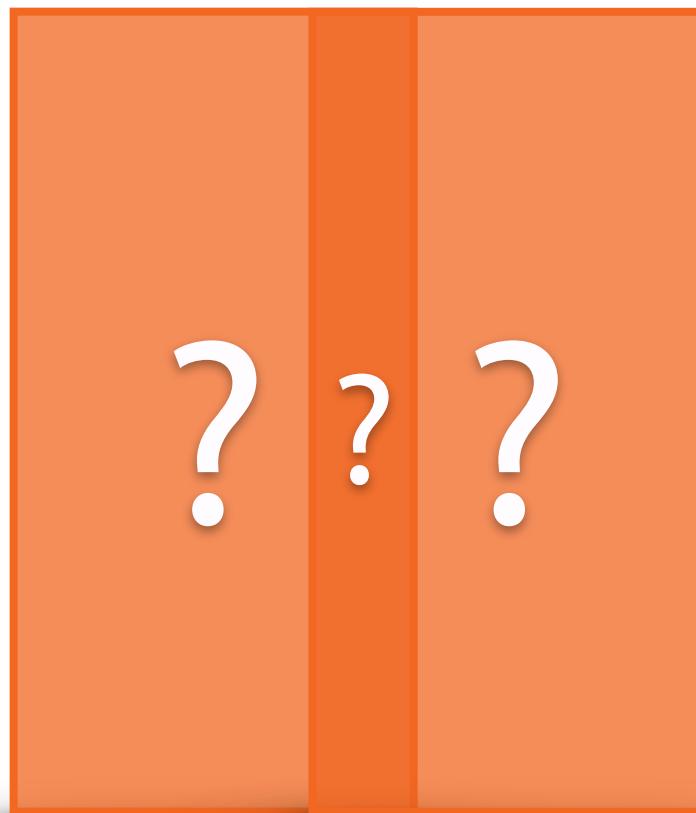
# Main Principle



Divide into  
sub-problems



Solve sub-problems



# Main Principle



Divide into  
sub-problems



Solve sub-problems



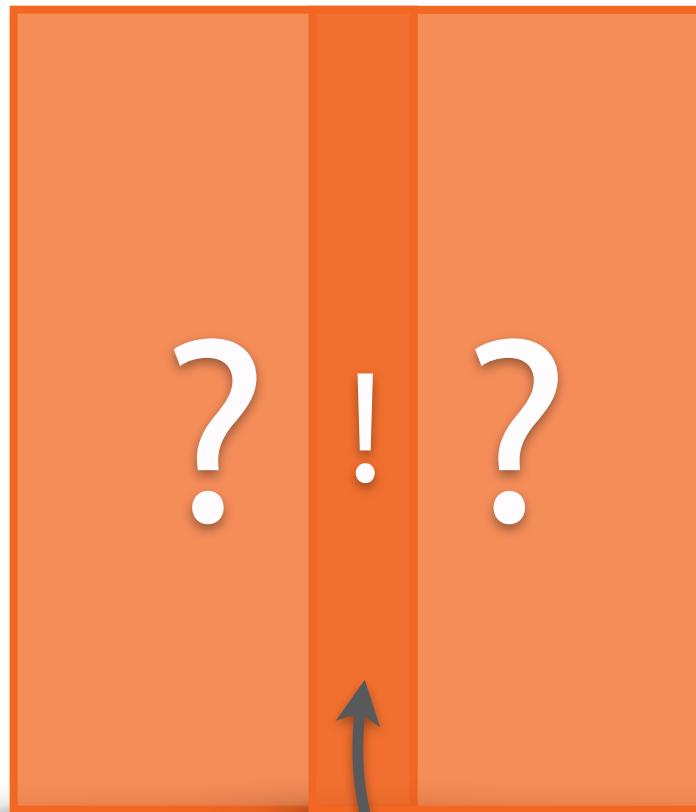
# Main Principle



Divide into  
sub-problems



Solve sub-problems



Solve once and cache result

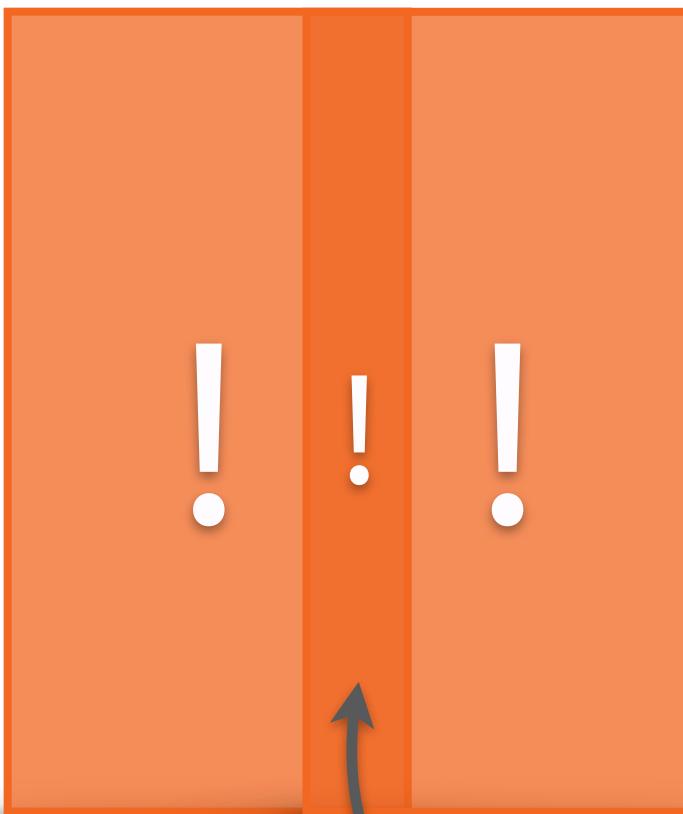
# Main Principle



Divide into  
sub-problems



Solve sub-problems



Solve once and cache result

# Main Principle

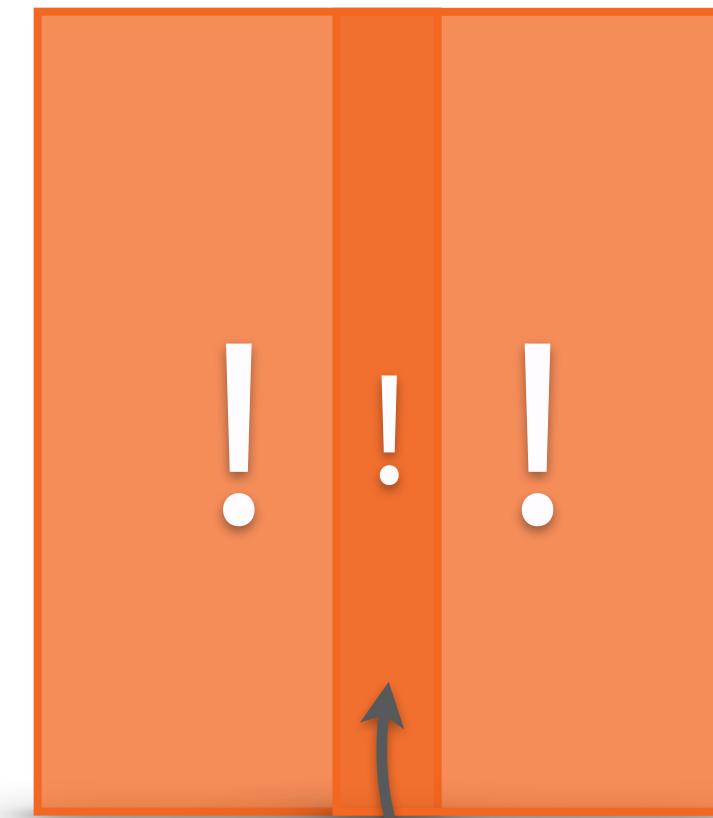


Divide into  
sub-problems



Deduce the  
“large” solution  
from sub-results

Solve sub-problems



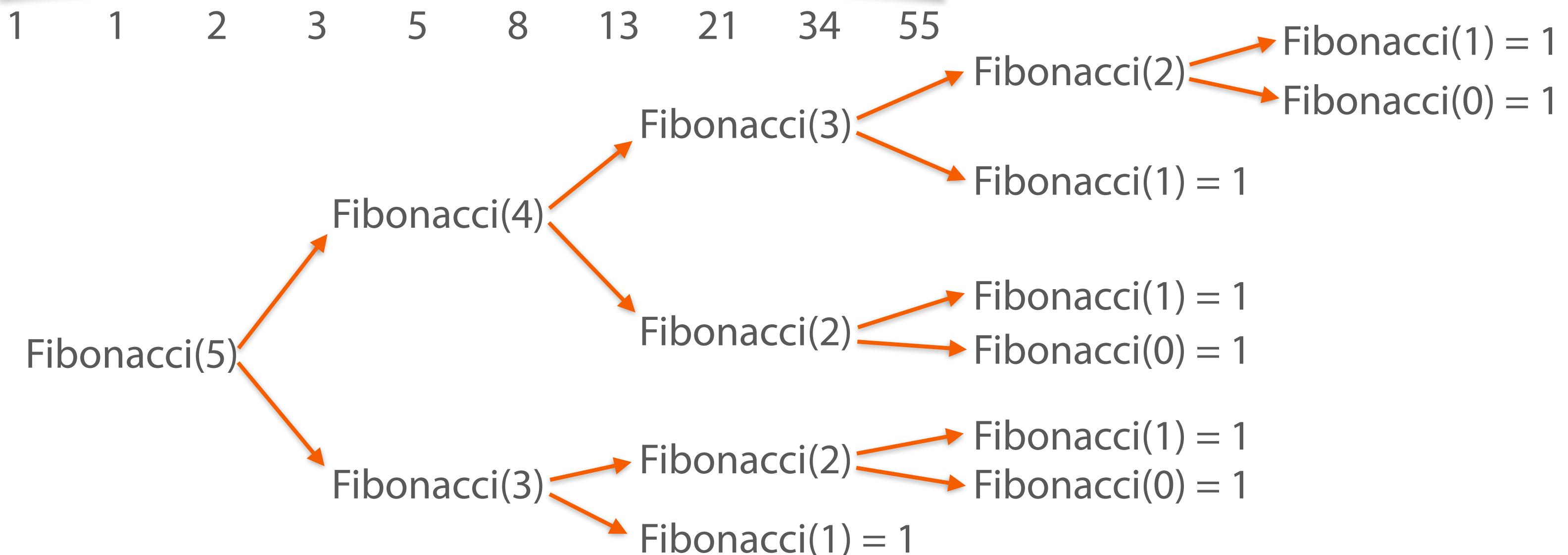
Solve once and cache result

```
int Fibonacci(int n)
{
    if (n <= 1) // throw error if n < 0
        return 1;
    return Fibonacci(n - 1) + Fibonacci(n - 2);
}
```

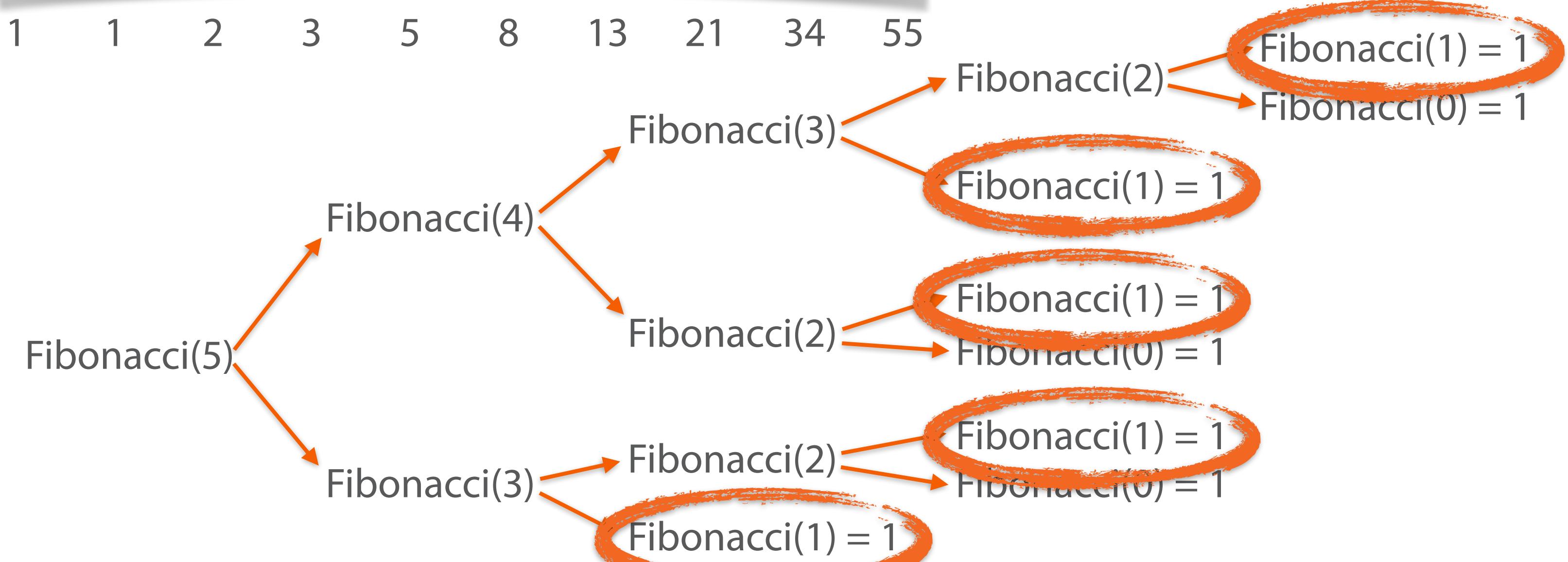
```
int Fibonacci(int n)
{
    if (n <= 1) // throw error if n < 0
        return 1;
    return Fibonacci(n - 1) + Fibonacci(n - 2);
}
```

1      1      2      3      5      8      13      21      34      55

```
int Fibonacci(int n)
{
    if (n <= 1) // throw error if n < 0
        return 1;
    return Fibonacci(n - 1) + Fibonacci(n - 2);
}
```

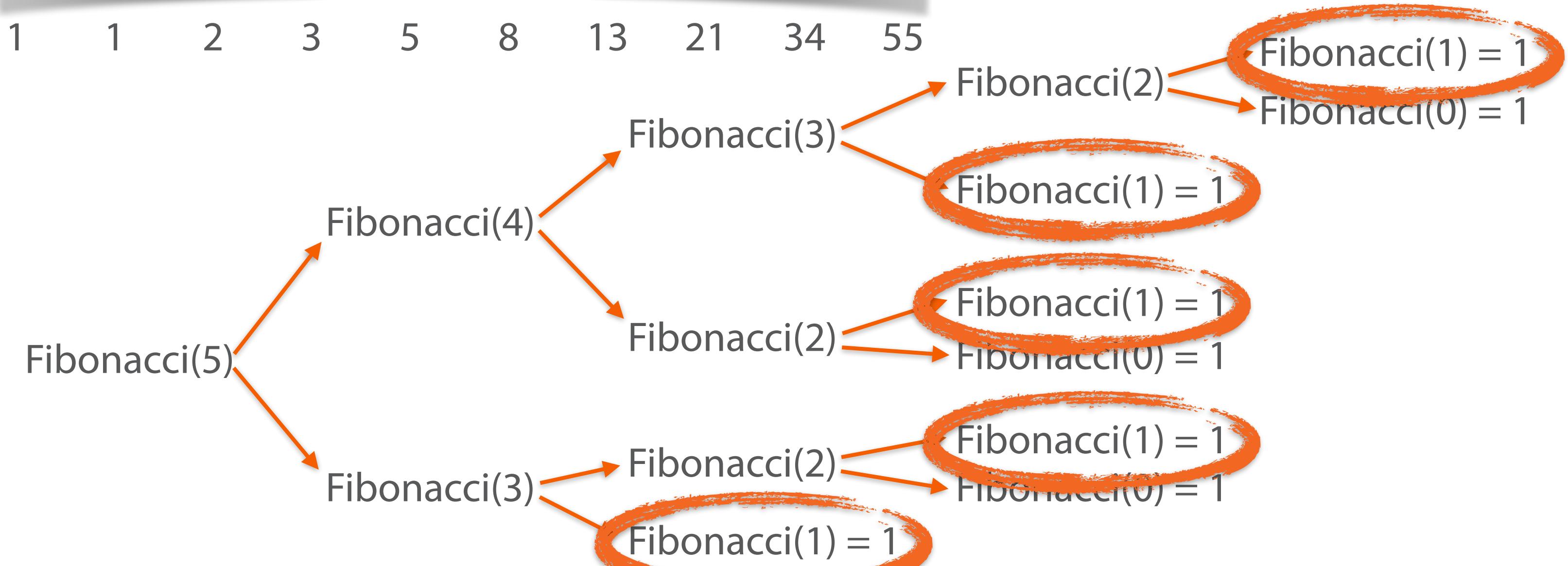


```
int Fibonacci(int n)
{
    if (n <= 1) // throw error if n < 0
        return 1;
    return Fibonacci(n - 1) + Fibonacci(n - 2);
}
```



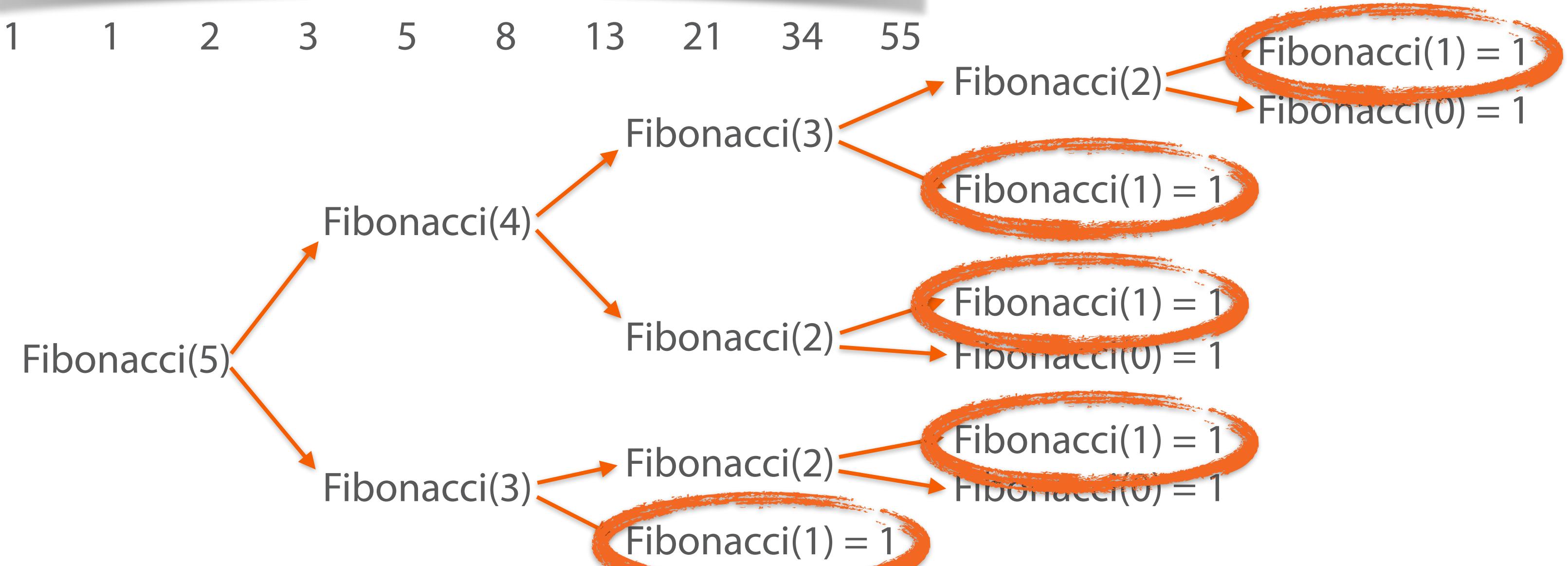
```
int Fibonacci(int n)
{
    if (n <= 1) // throw error if n < 0
        return 1;
    return Fibonacci(n - 1) + Fibonacci(n - 2);
}
```

Fibonacci(30)...



```
int Fibonacci(int n)
{
    if (n <= 1) // throw error if n < 0
        return 1;
    return Fibonacci(n - 1) + Fibonacci(n - 2);
}
```

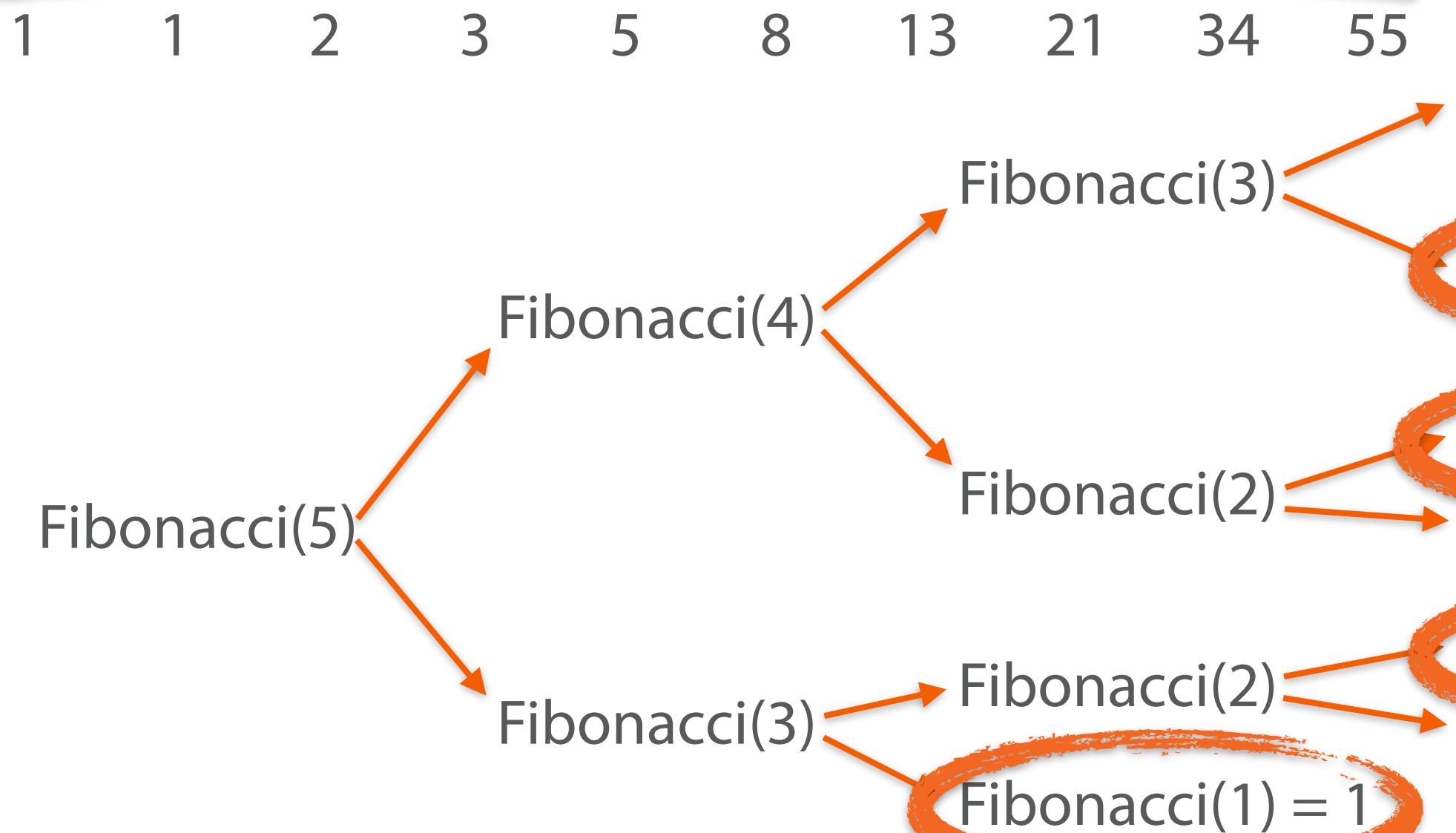
Fibonacci(30)...  
Fibonacci(0): 514,229  
Fibonacci(1): 832,040  
Fibonacci(2): 514,229  
Fibonacci(3): 317,811



```

int Fibonacci(int n)
{
    if (n <= 1) // throw error if n < 0
        return 1;
    return Fibonacci(n - 1) + Fibonacci(n - 2);
}

```



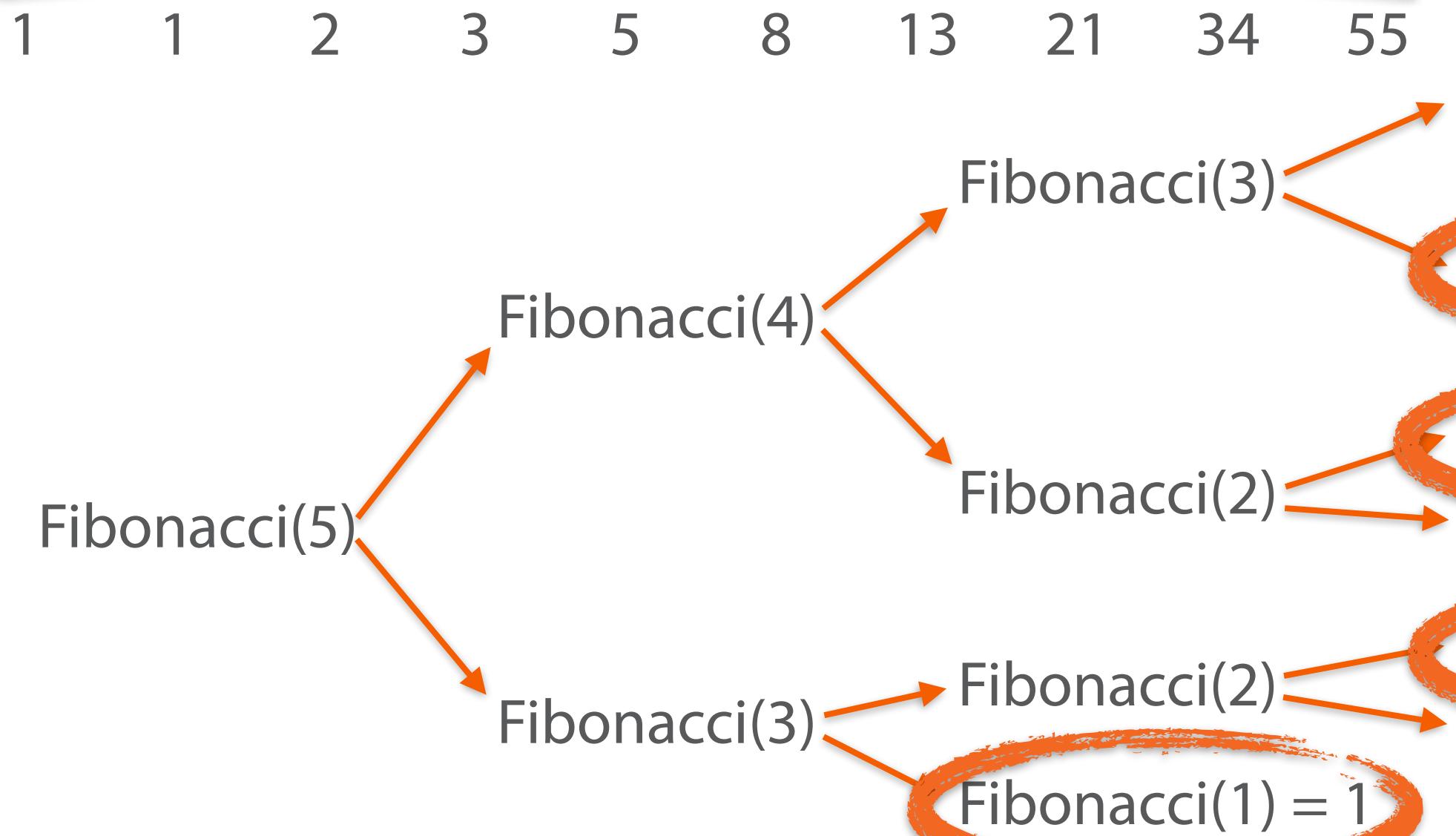
$n$	$Fibonacci(n)$
0	4181
1	6765
2	4181
3	2584
4	1597
5	987
6	610
7	377
8	233
9	144

$n$	$Fibonacci(n)$
5	5
10	89
11	55
12	34
13	21
14	13
15	8
16	5
17	3
18	2
19	1
20	1

```

int Fibonacci(int n)
{
    if (n <= 1) // throw error if n < 0
        return 1;
    return Fibonacci(n - 1) + Fibonacci(n - 2);
}

```



n	count
0	4181
1	6765
2	4181
3	2584
4	1597
5	987
6	610
7	377
8	233
9	144

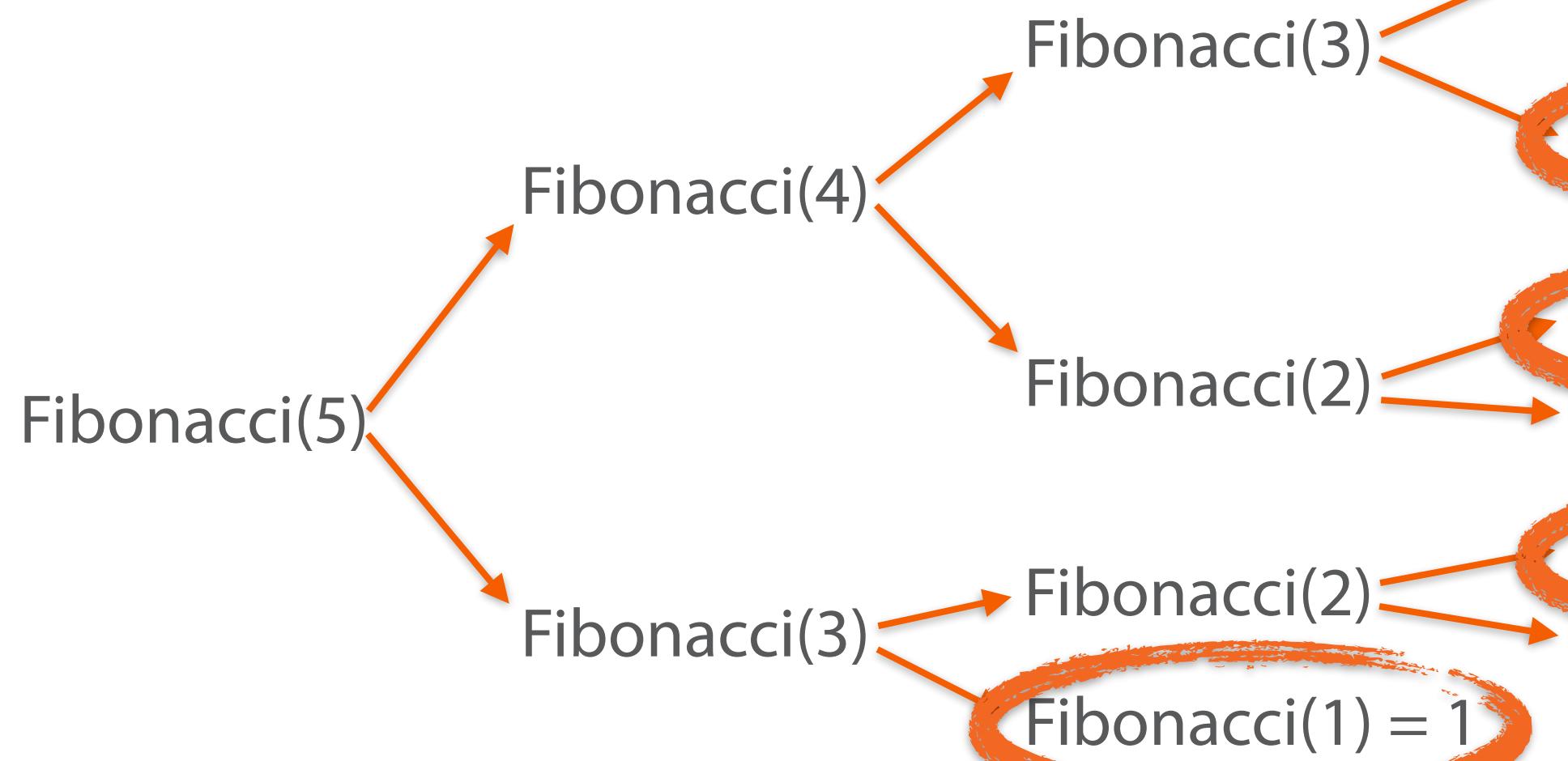
n	count
10	55
11	34
12	21
13	13
14	8
15	5
16	3
17	2
18	1
19	1
20	1

```

int Fibonacci(int n)
{
    if (n <= 1) // throw error if n < 0
        return 1;
    return Fibonacci(n - 1) + Fibonacci(n - 2);
}

```

1 1 2 3 5 8 13 21 34 55



<i>n</i>	<i>count</i>
0	4181
1	6765
2	4181
3	2584
4	1597
5	987
6	610
7	377
8	233
9	144
10	89
11	55
12	34
13	21
14	13
15	8
16	5
17	3
18	2
19	1
20	1

<i>n</i>	<i>count</i>
0	4181
1	6765
2	4181
3	2584
4	1597
5	987
6	610
7	377
8	233
9	144
10	89
11	55
12	34
13	21
14	13
15	8
16	5
17	3
18	2
19	1
20	1

```

int Fibonacci(int n)
{
    if (n <= 1) // throw error if n < 0
        return 1;
    return Fibonacci(n - 1) + Fibonacci(n - 2);
}

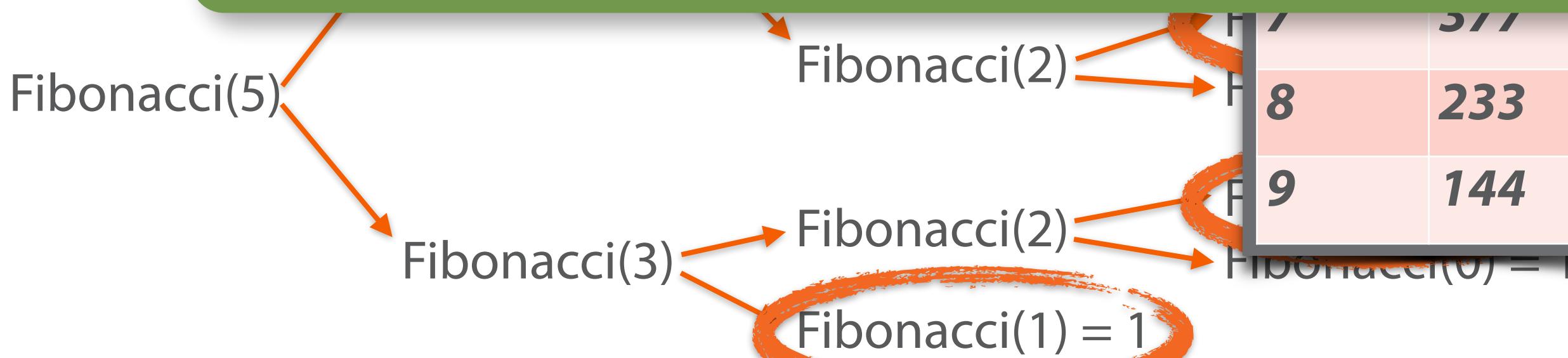
```

Fibonacci(80)	
n	count
0	4181
1	6765
2	10946
3	17711
4	28657
5	46368
6	75025
7	121393
8	196418
9	317811
10	514229
11	832041
12	1346269
13	2178309
14	3524578
15	5698845
16	9227465
17	15000000
18	24226895
19	39226895
20	63453790

# Challenge

Try to run  $\text{Fibonacci}(8)$  and  $\text{Fibonacci}(80)$

Do you ever finish with the latter?



n	count
0	577
1	233
2	144
3	89
4	55
5	34
6	21
7	13
8	8
9	5
10	3
11	2
12	1
13	1
14	1
15	1
16	1
17	1
18	1
19	1
20	1

# Fixing the recursive Fibonacci

...by using dynamic programming



# Strategy Considerations

# Strategy Considerations



Top-down:

# Strategy Considerations



Top-down:  
Call top-level function, and build cache and solution along the way.

# Strategy Considerations



Top-down:  
Call top-level function, and build cache and solution along the way.



Bottom-up:

# Strategy Considerations



Top-down:  
Call top-level function, and build cache and solution along the way.



Bottom-up:  
Construct cache first. Deduce solution next.

# Strategy Considerations



Top-down:  
Call top-level function, and build cache and solution along the way.



Bottom-up:  
Construct cache first. Deduce solution next.

Pick most valuable VMs  
for a server within its  
capacity

# Strategy Considerations



Top-down:  
Call top-level function, and build cache and solution along the way.



Bottom-up:  
Construct cache first. Deduce solution next.

Pick most valuable VMs  
for a server within its  
capacity

Pick most valuable batch  
jobs to run within time  
limit

# Strategy Considerations



Top-down:  
Call top-level function, and build cache and solution along the way.



Bottom-up:  
Construct cache first. Deduce solution next.

Pick most valuable VMs  
for a server within its  
capacity

Pick most valuable batch  
jobs to run within time  
limit

Pick investments with  
highest potential within  
budget

# The 0/1 Knapsack Problem

# The 0/1 Knapsack Problem

Knapsack capacity: 4

# The 0/1 Knapsack Problem

Knapsack capacity: 4



# The 0/1 Knapsack Problem

Knapsack capacity: 4



Weight: 2

Value: 3



Weight: 2

Value: 1



Weight:1

Value: 3

# The 0/1 Knapsack Problem

Knapsack capacity: 4



Weight: 2    Weight: 2    Weight: 1

Value: 3    Value: 1    Value: 3

$2 \cdot 2 \cdot 2 = 2^3$  combinations

# The 0/1 Knapsack Problem

Knapsack capacity: 4



Weight: 2    Weight: 2    Weight: 1

Value: 3    Value: 1    Value: 3

$2 \cdot 2 \cdot 2 = 2^3$  combinations

For  $N$  items:  $2^N$  combinations

# The 0/1 Knapsack Problem

Knapsack capacity: 4



Weight: 2

Value: 3



Weight: 2

Value: 1



Weight:1

Value: 3

# The 0/1 Knapsack Problem

Knapsack capacity: 4



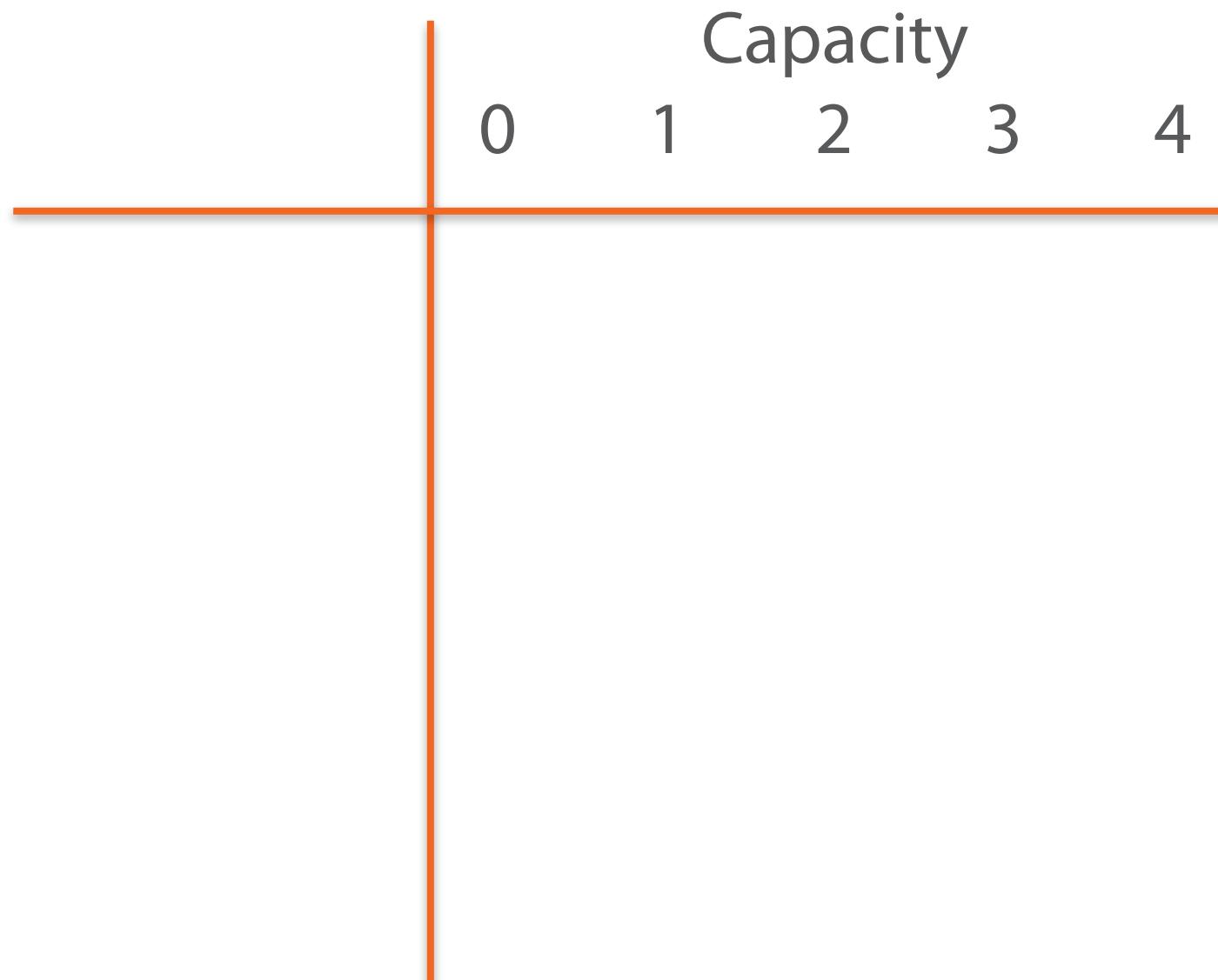
Weight: 2  
Value: 3



Weight: 2  
Value: 1



Weight: 1  
Value: 3



# The 0/1 Knapsack Problem

Knapsack capacity: 4



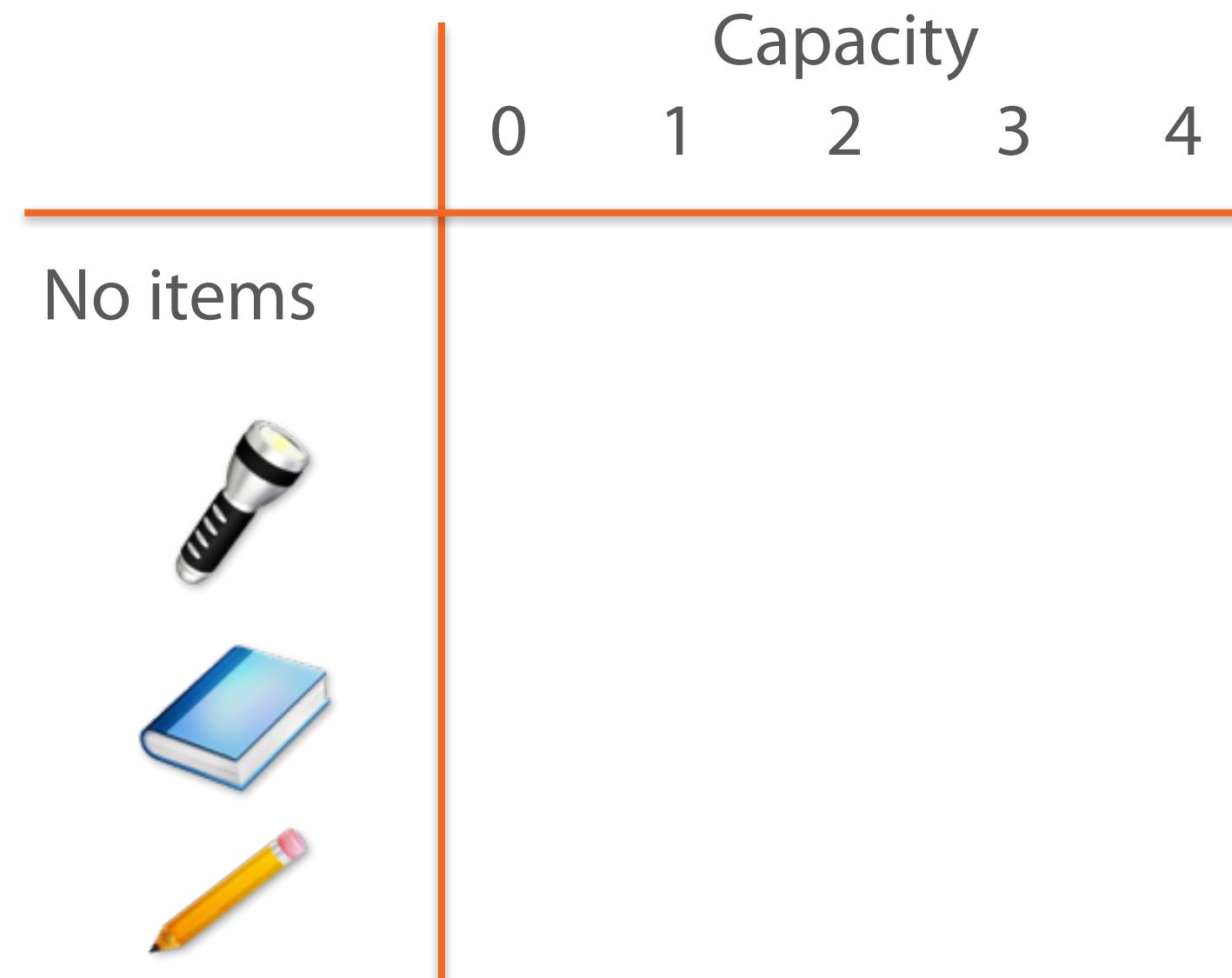
Weight: 2  
Value: 3



Weight: 2  
Value: 1



Weight: 1  
Value: 3



# The 0/1 Knapsack Problem

Knapsack capacity: 4

		
Weight: 2	Weight: 2	Weight: 1
Value: 3	Value: 1	Value: 3

	Capacity					Maximum value possible
	0	1	2	3	4	
No items	?	?	?	?	?	
	?	?	?	?	?	
	?	?	?	?	?	
	?	?	?	?	?	

# The 0/1 Knapsack Problem

Knapsack capacity: 4



Weight: 2    Weight: 2    Weight: 1  
Value: 3    Value: 1    Value: 3

		Capacity					Maximum value possible
		0	1	2	3	4	
No items	$i_0$	?	?	?	?	?	?
	$i_1$	?	?	?	?	?	
	$i_2$	?	?	?	?	?	?
	$i_3$	?	?	?	?	?	?

# The 0/1 Knapsack Problem

Knapsack capacity: 4



Weight: 2    Weight: 2    Weight: 1  
Value: 3    Value: 1    Value: 3

		Capacity					Maximum value possible
		0	1	2	3	4	
No items	$i_0$	?	?	?	?	?	?
	$i_1$	?	?	?	?	?	
	$i_2$	?	?	?	?	?	?
	$i_3$	?	?	?	?	?	?
		$M[items + 1][capacity + 1]$					

# The 0/1 Knapsack Problem

Knapsack capacity: 4



Weight: 2    Weight: 2    Weight: 1  
Value: 3    Value: 1    Value: 3

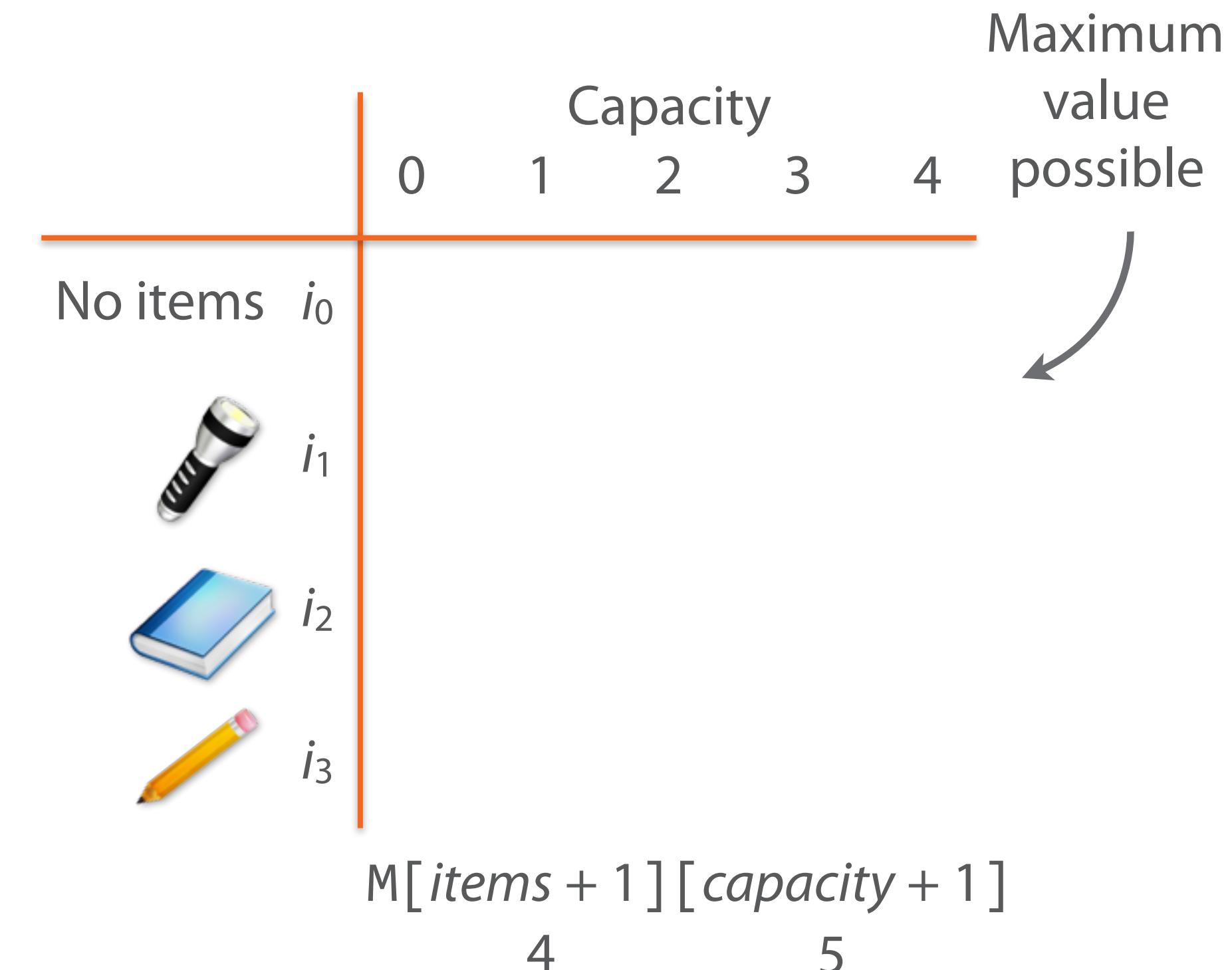
		Capacity					Maximum value possible
		0	1	2	3	4	
No items	$i_0$	?	?	?	?	?	?
	$i_1$	?	?	?	?	?	
	$i_2$	?	?	?	?	?	?
	$i_3$	?	?	?	?	?	?
		$M[items + 1][capacity + 1]$					4
							5

# The 0/1 Knapsack Problem

Knapsack capacity: 4



Weight: 2    Weight: 2    Weight: 1  
Value: 3    Value: 1    Value: 3

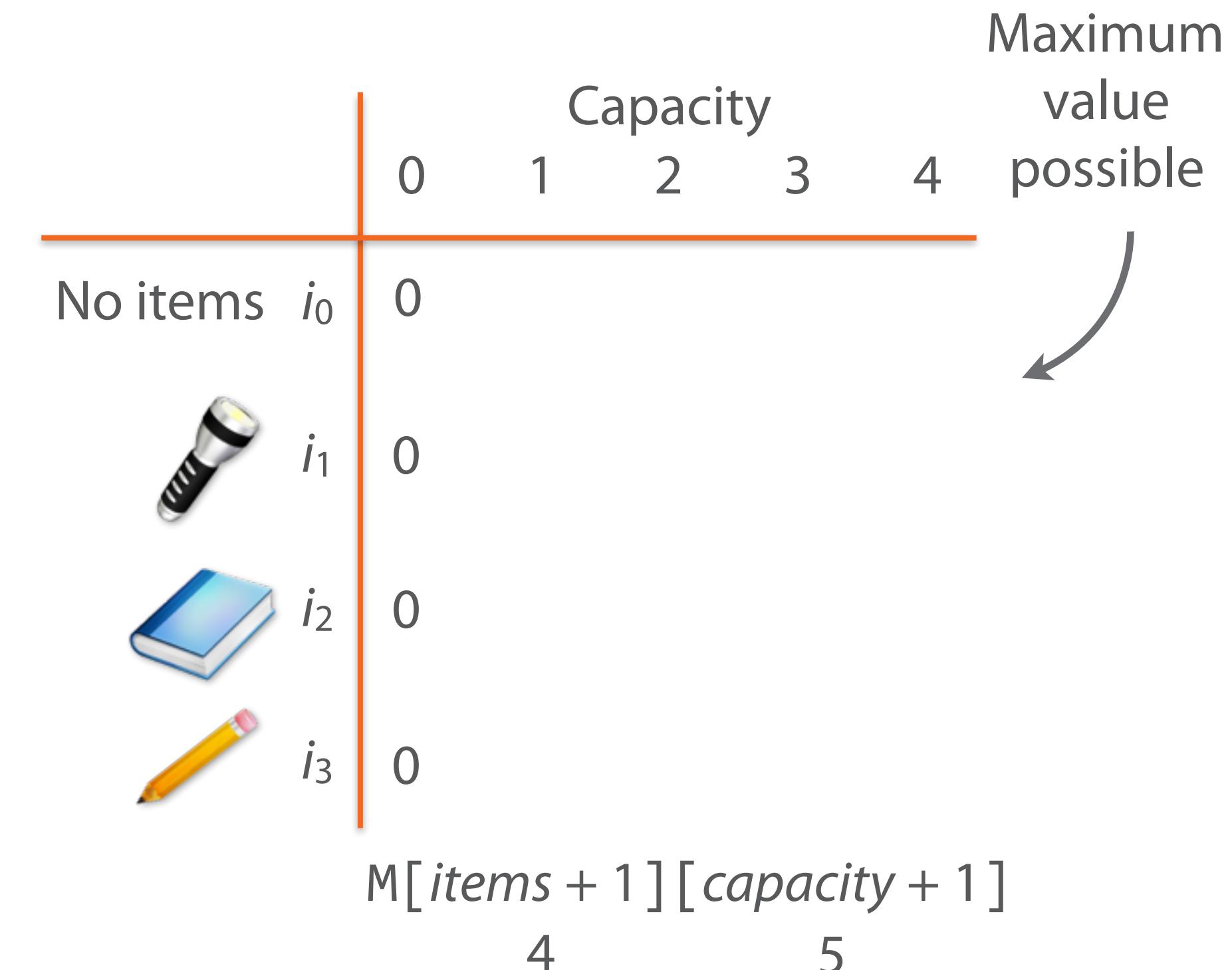


# The 0/1 Knapsack Problem

Knapsack capacity: 4



Weight: 2    Weight: 2    Weight: 1  
Value: 3    Value: 1    Value: 3

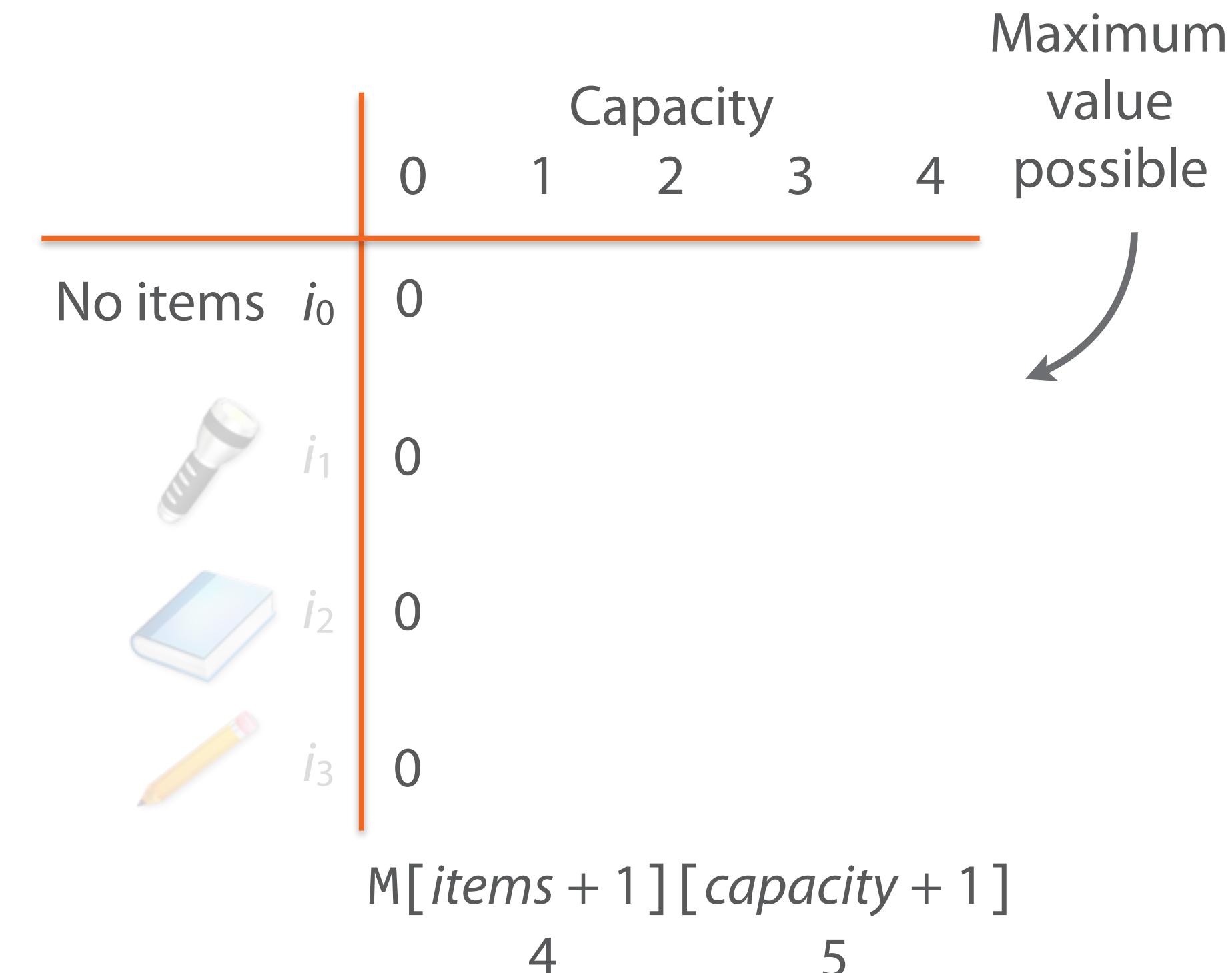


# The 0/1 Knapsack Problem

Knapsack capacity: 4



Weight: 2    Weight: 2    Weight: 1  
Value: 3    Value: 1    Value: 3

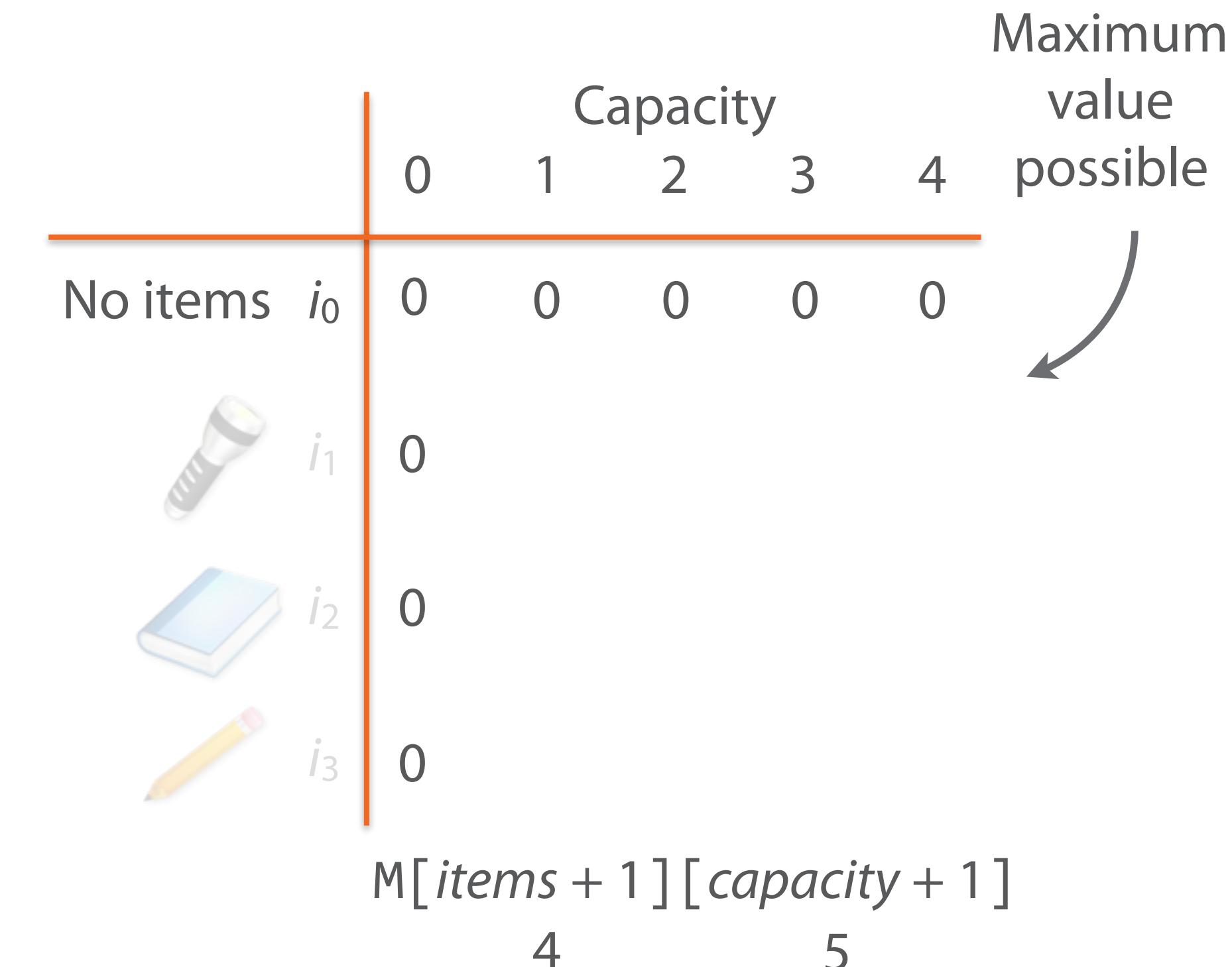


# The 0/1 Knapsack Problem

Knapsack capacity: 4



Weight: 2    Weight: 2    Weight: 1  
Value: 3    Value: 1    Value: 3

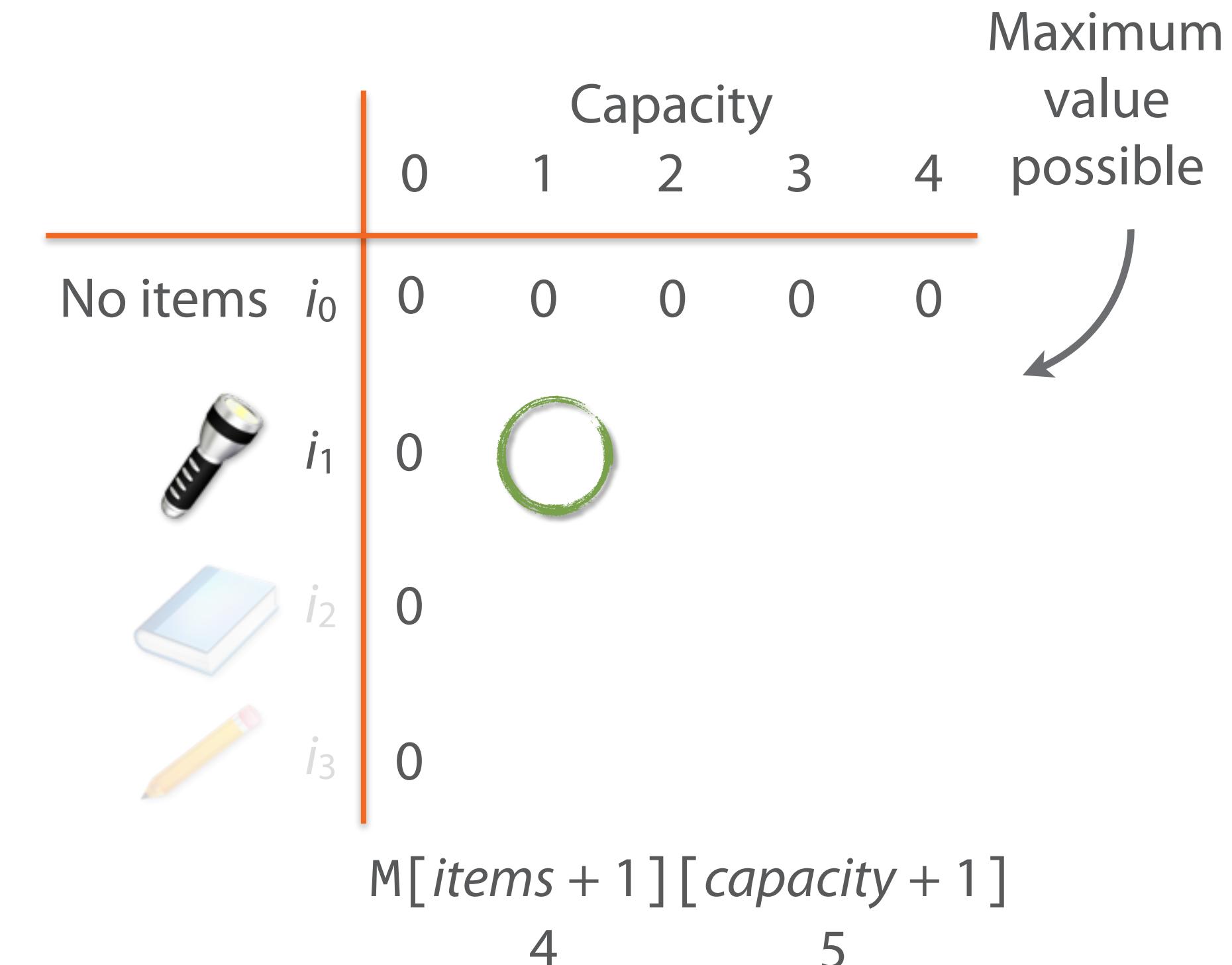


# The 0/1 Knapsack Problem

Knapsack capacity: 4



Weight: 2    Weight: 2    Weight: 1  
Value: 3    Value: 1    Value: 3



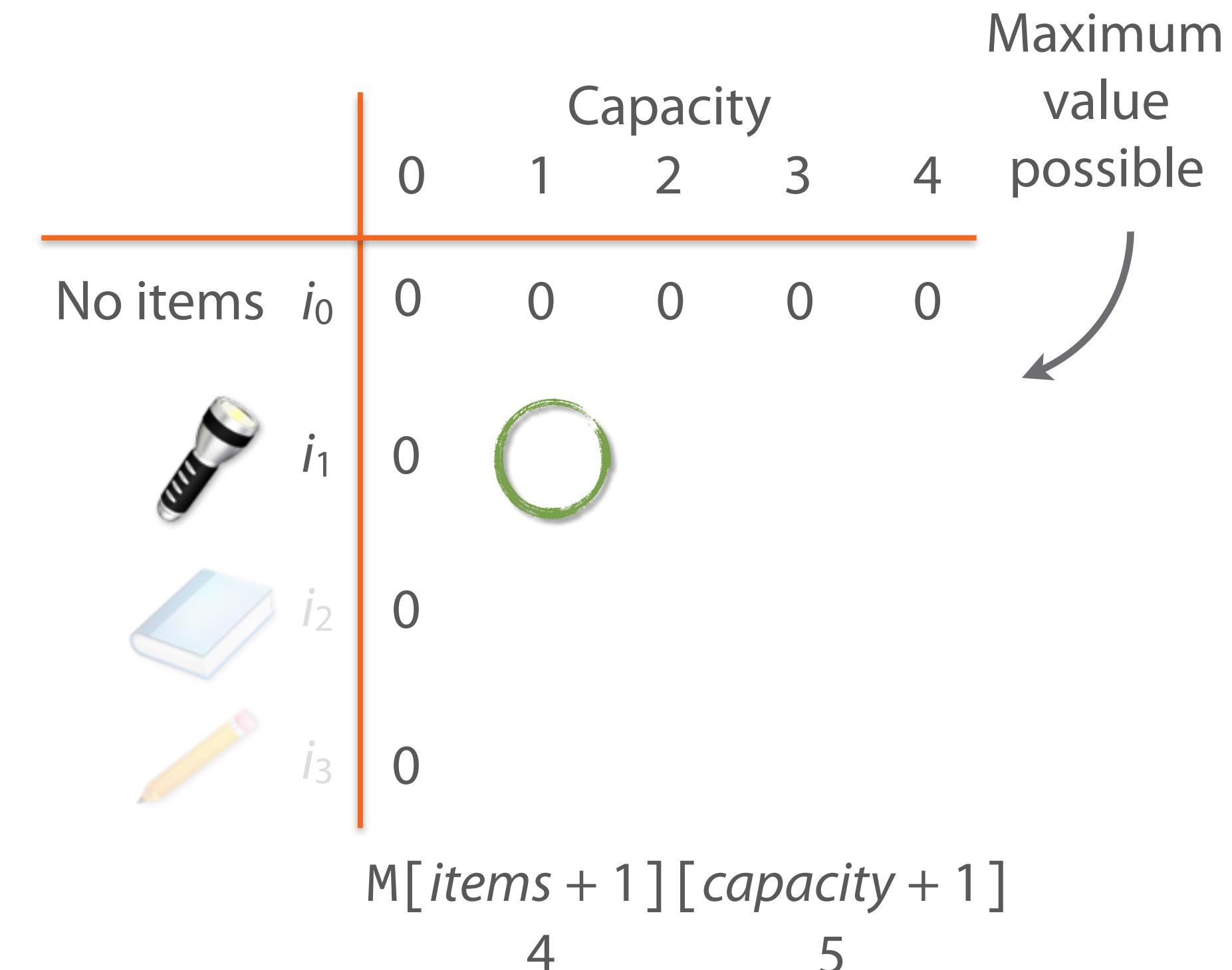
# The 0/1 Knapsack Problem

Knapsack capacity: 4



Weight: 2    Weight: 2    Weight: 1  
Value: 3    Value: 1    Value: 3

$M[i][capacity] =$



# The 0/1 Knapsack Problem

Knapsack capacity: 4

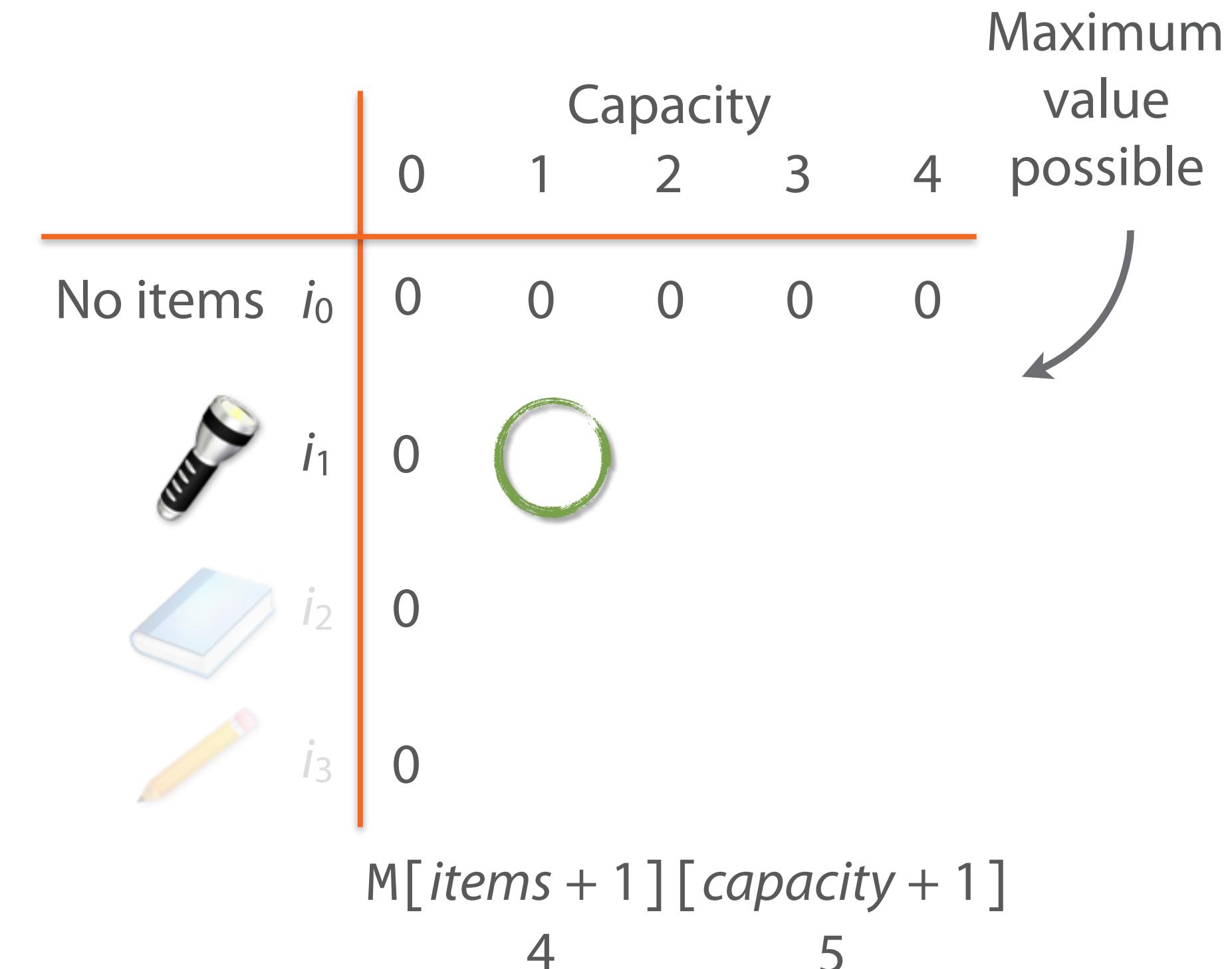


Weight: 2    Weight: 2    Weight: 1  
Value: 3    Value: 1    Value: 3

$M[i][capacity] =$

if excluded:

if included and small enough:



# The 0/1 Knapsack Problem

Knapsack capacity: 4



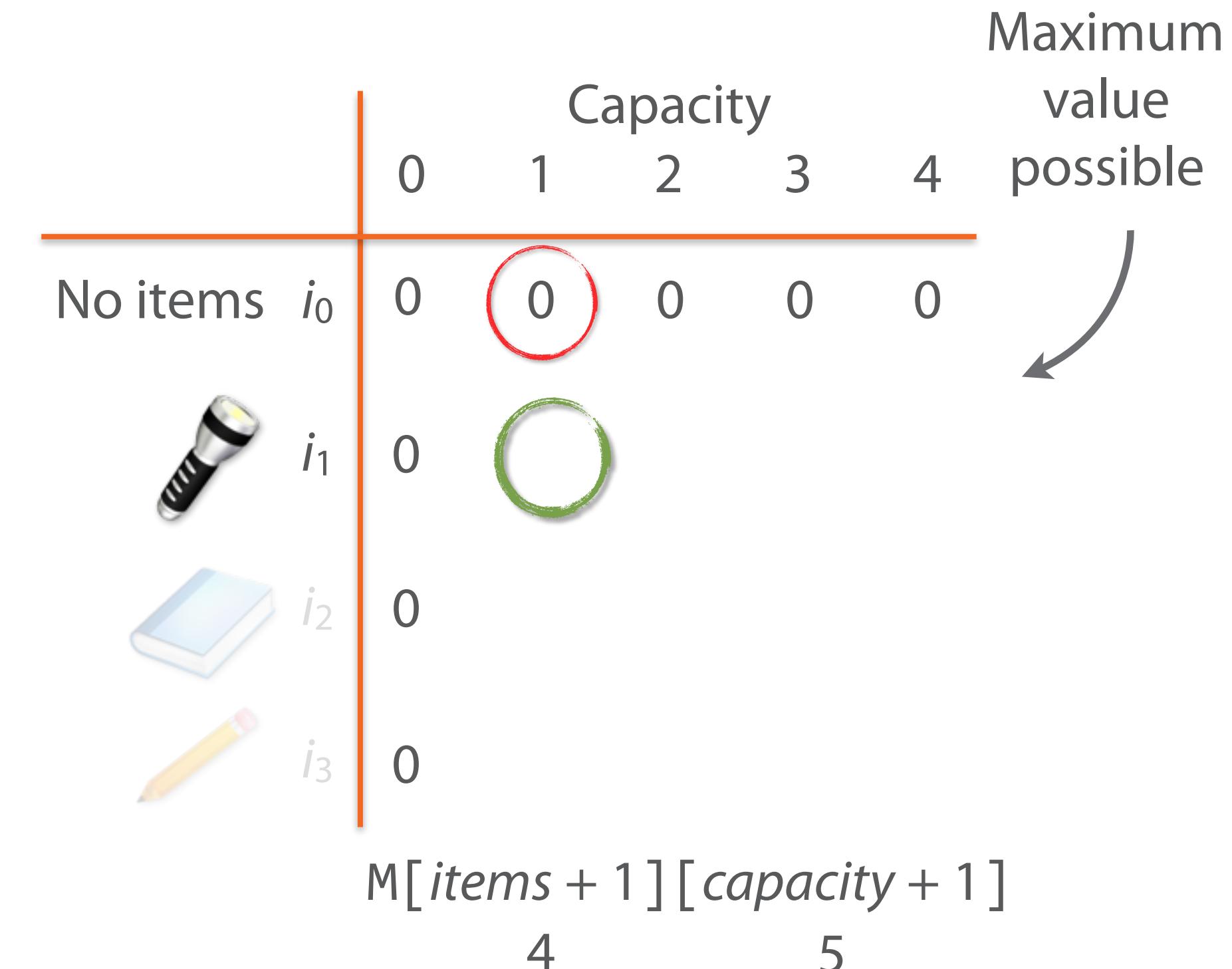
Weight: 2    Weight: 2    Weight: 1  
Value: 3    Value: 1    Value: 3

$M[i][capacity] =$

if excluded:

$M[i - 1][capacity]$

if included and small enough:



# The 0/1 Knapsack Problem

Knapsack capacity: 4



Weight: 2      Weight: 2      Weight: 1  
Value: 3      Value: 1      Value: 3

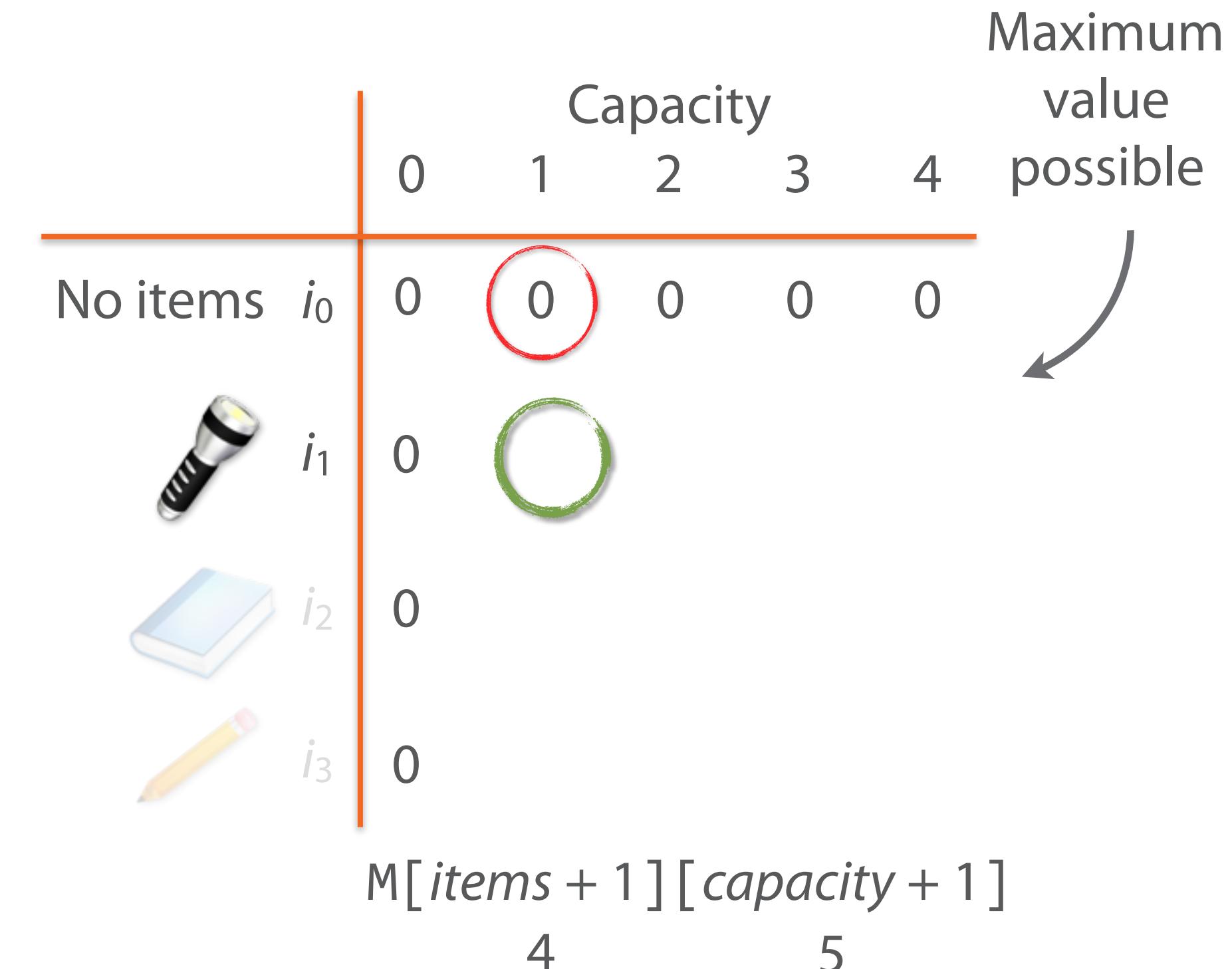
$M[i][capacity] =$

if excluded:

$M[i - 1][capacity]$

if included and small enough:

$value(i)$



# The 0/1 Knapsack Problem

Knapsack capacity: 4



Weight: 2      Weight: 2      Weight: 1  
Value: 3      Value: 1      Value: 3

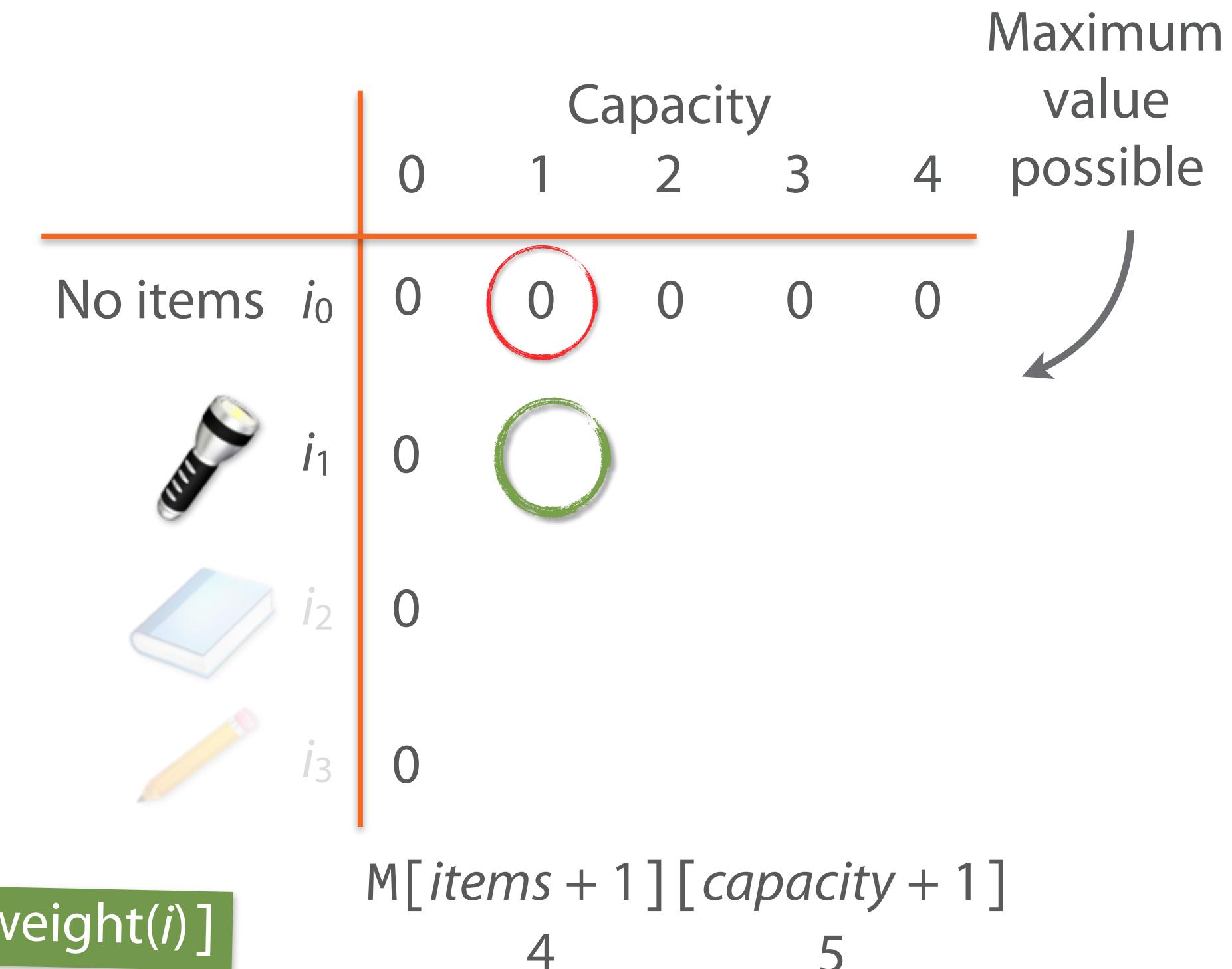
$M[i][capacity] =$

if excluded:

$M[i - 1][capacity]$

if included and small enough:

$value(i) + M[i - 1][capacity - weight(i)]$



# The 0/1 Knapsack Problem

Knapsack capacity: 4		
Weight: 2	Weight: 2	Weight: 1
Value: 3	Value: 1	Value: 3

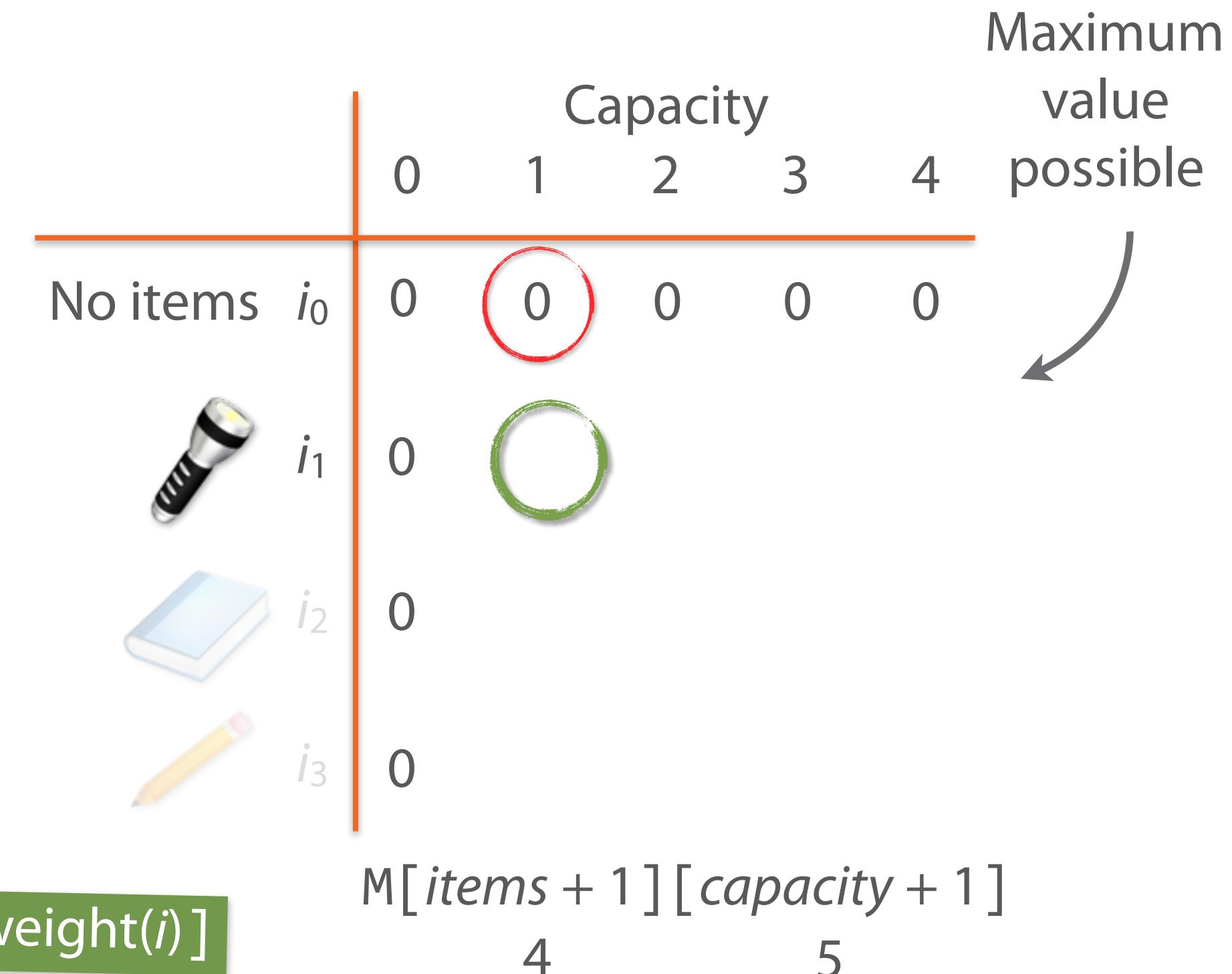
$$M[i][capacity] = \max(\text{excluded}, \text{included})$$

if excluded:

$$M[i-1][capacity]$$

if included and small enough:

$$\text{value}(i) + M[i-1][capacity - weight(i)]$$



# The 0/1 Knapsack Problem

Knapsack capacity: 4		
		
Weight: 2 Value: 3	Weight: 2 Value: 1	Weight: 1 Value: 3

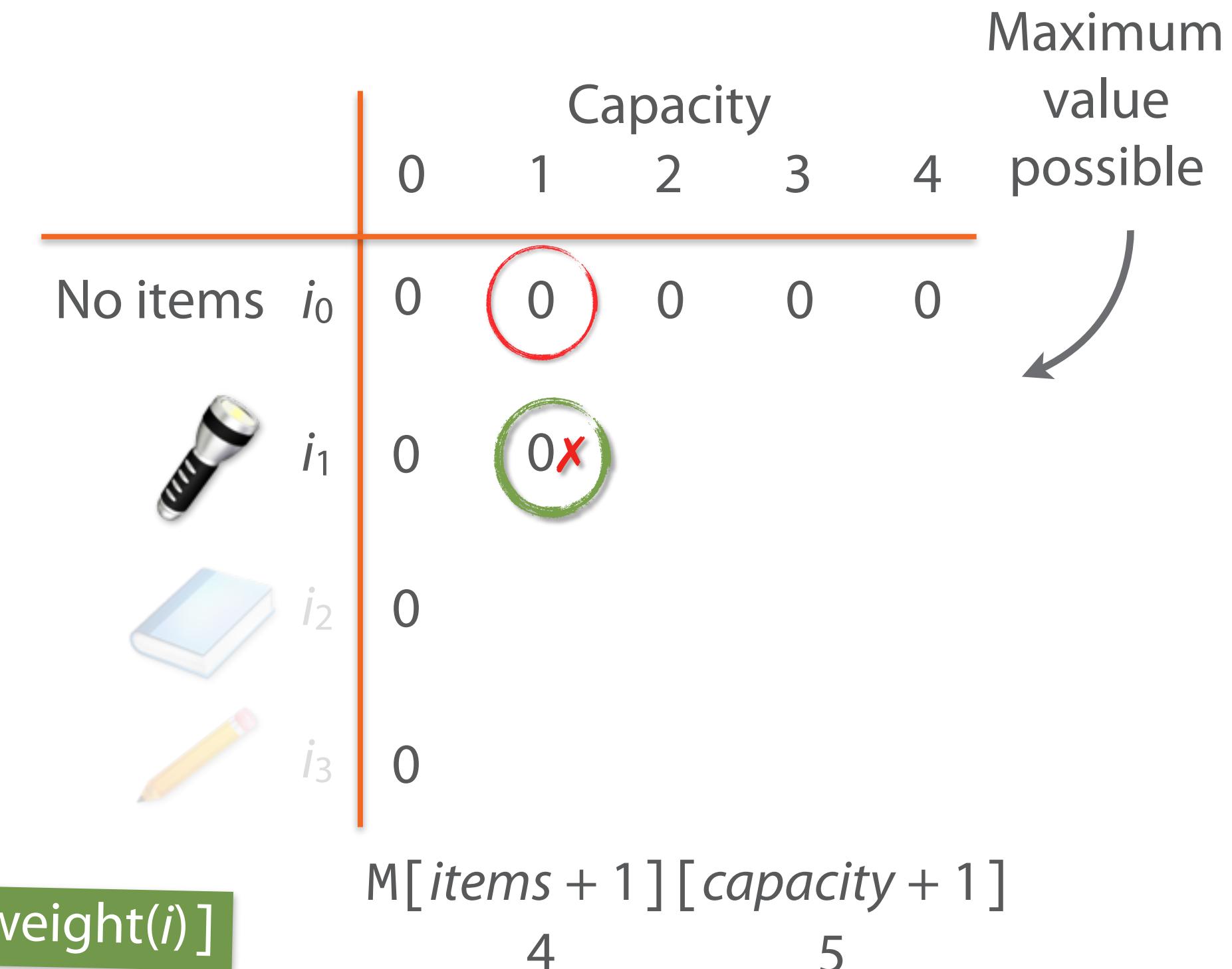
$$M[i][capacity] = \max(\text{excluded}, \text{included})$$

if excluded:

$$M[i-1][capacity]$$

if included and small enough:

$$\text{value}(i) + M[i-1][capacity - weight(i)]$$



# The 0/1 Knapsack Problem

Knapsack capacity: 4		
Weight: 2	Weight: 2	Weight: 1
Value: 3	Value: 1	Value: 3

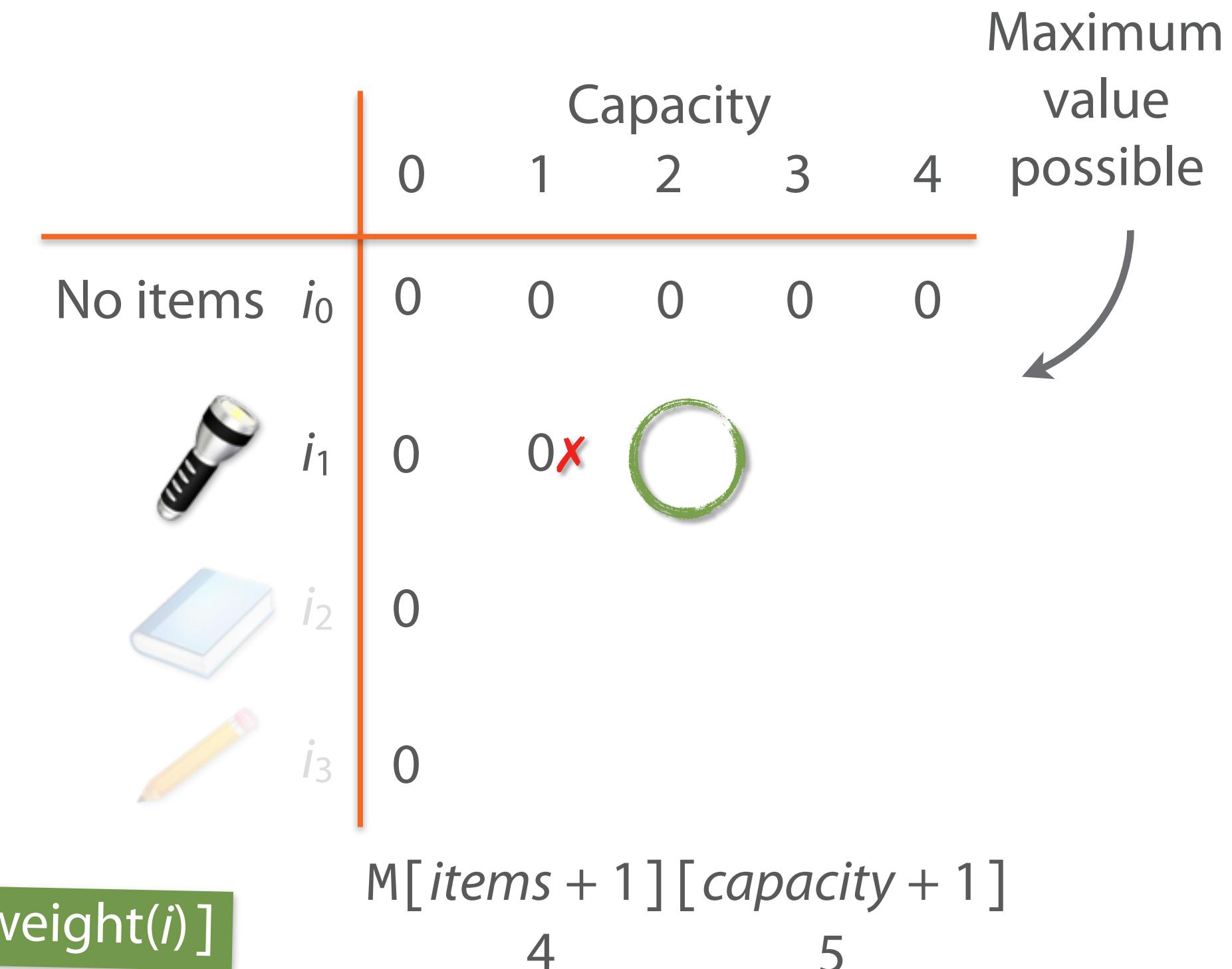
$$M[i][capacity] = \max(\text{value}(i) + M[i-1][capacity - weight(i)], M[i-1][capacity])$$

if excluded:

$$M[i-1][capacity]$$

if included and small enough:

$$\text{value}(i) + M[i-1][capacity - weight(i)]$$



# The 0/1 Knapsack Problem

Knapsack capacity: 4		
		
Weight: 2 Value: 3	Weight: 2 Value: 1	Weight: 1 Value: 3

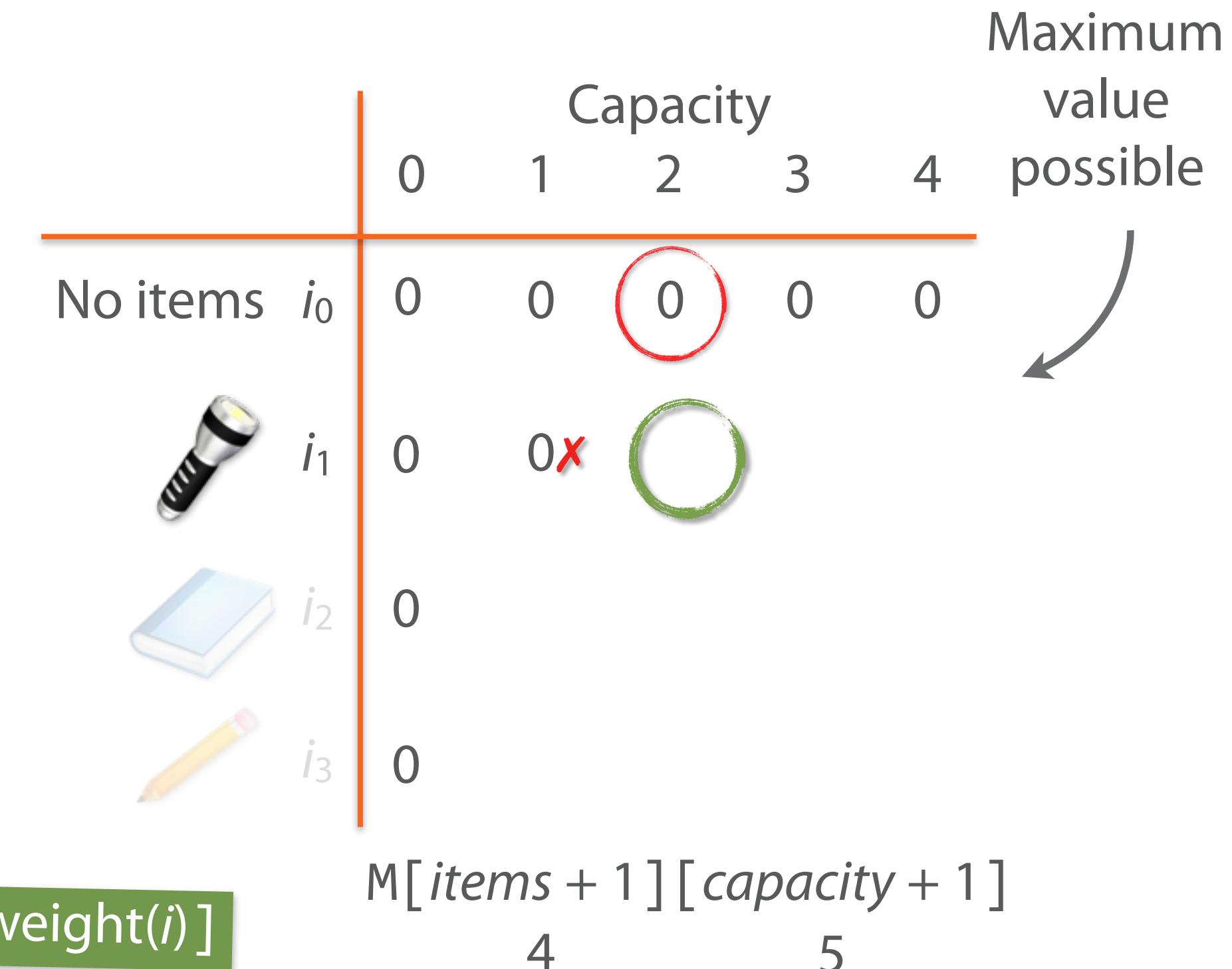
$$M[i][capacity] = \max(\text{excluded}, \text{included})$$

if excluded:

$$M[i-1][capacity]$$

if included and small enough:

$$\text{value}(i) + M[i-1][capacity - weight(i)]$$



# The 0/1 Knapsack Problem

Knapsack capacity: 4



Weight: 2      Weight: 2      Weight: 1  
Value: 3      Value: 1      Value: 3

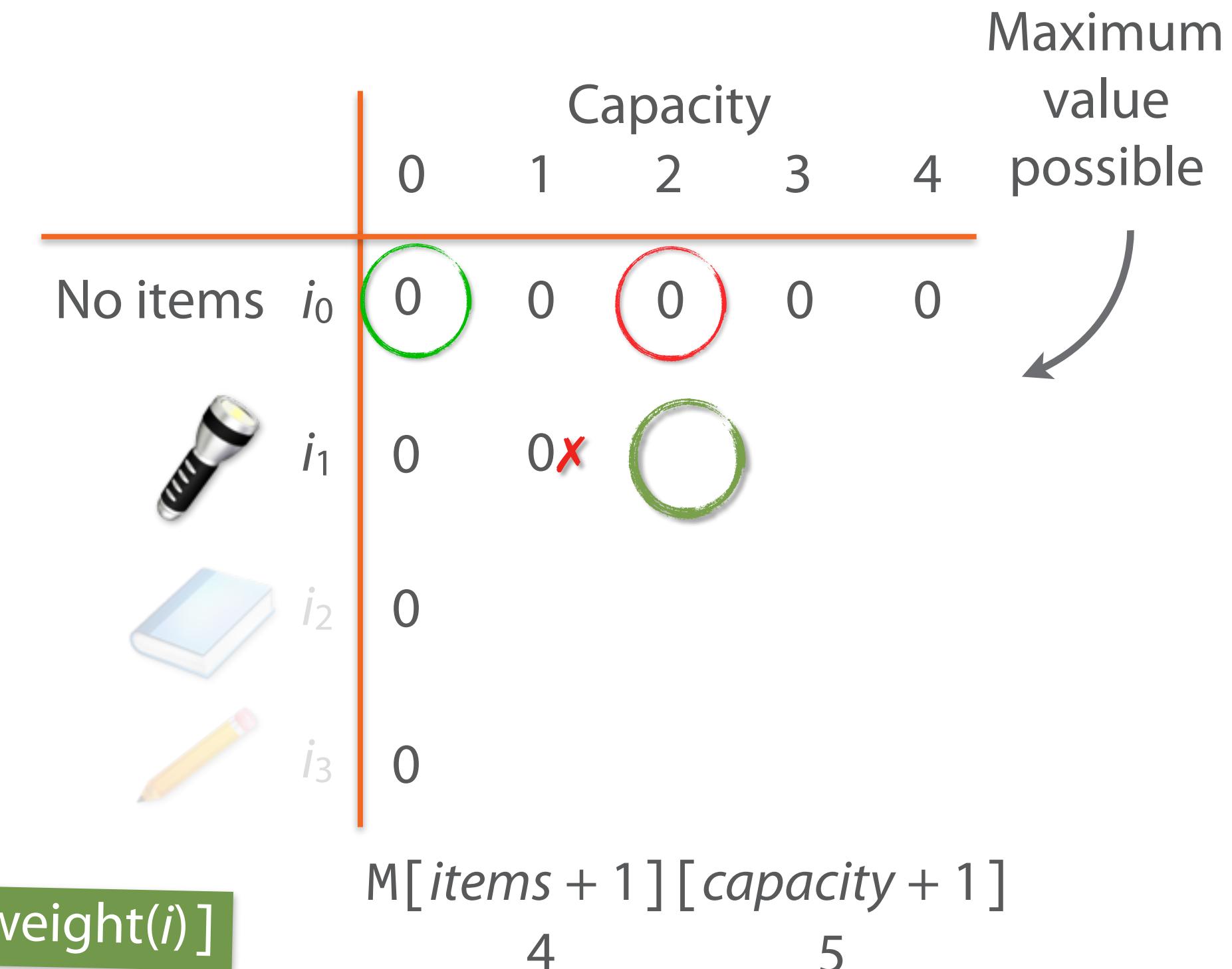
$$M[i][capacity] = \max(\text{excluded}, \text{included})$$

if excluded:

$$M[i-1][capacity]$$

if included and small enough:

$$\text{value}(i) + M[i-1][capacity - weight(i)]$$



# The 0/1 Knapsack Problem

Knapsack capacity: 4



Weight: 2      Weight: 2      Weight: 1  
Value: 3      Value: 1      Value: 3

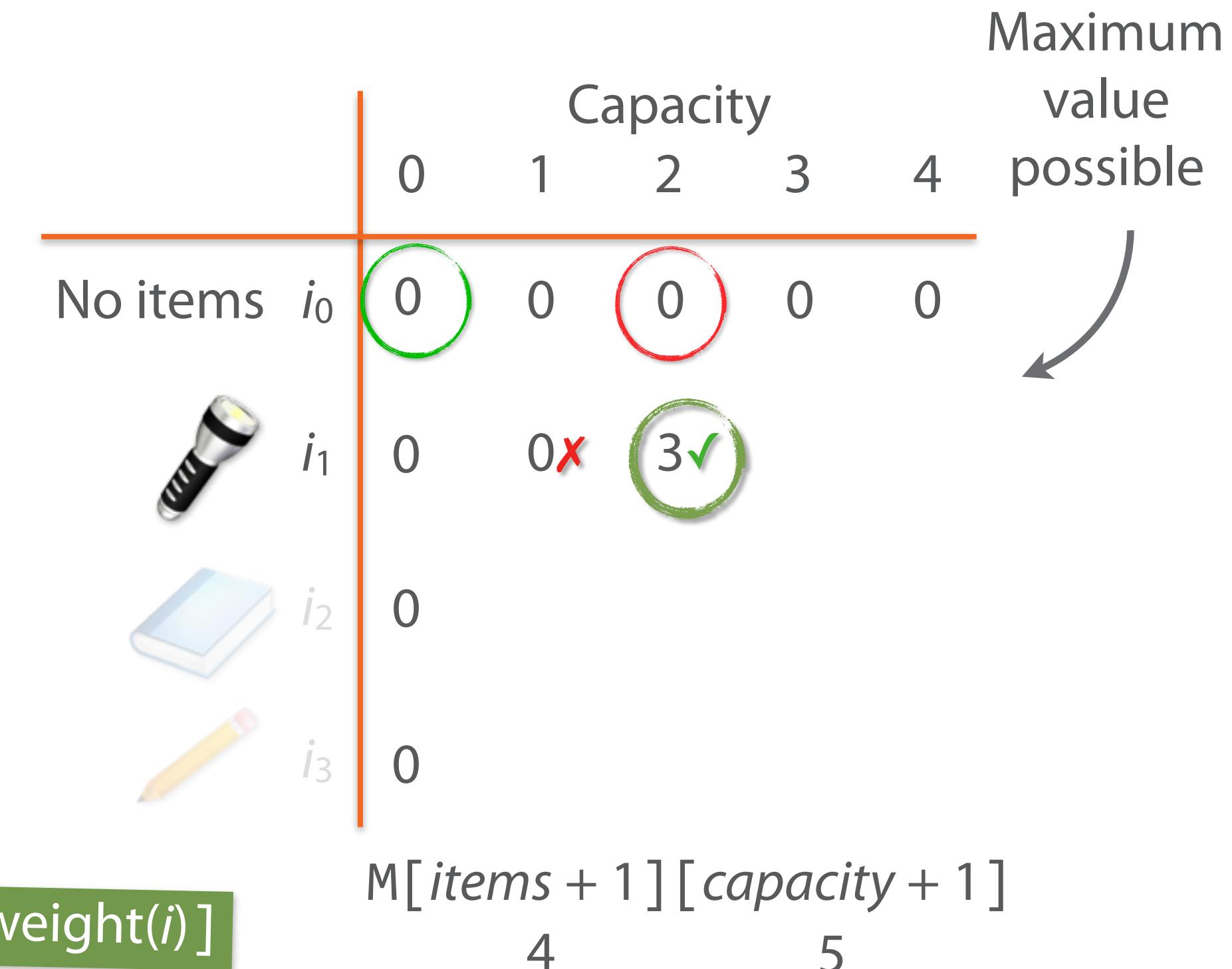
$$M[i][capacity] = \max(\text{excluded}, \text{included})$$

if excluded:

$$M[i-1][capacity]$$

if included and small enough:

$$\text{value}(i) + M[i-1][capacity - weight(i)]$$



# The 0/1 Knapsack Problem

Knapsack capacity: 4		
		
Weight: 2	Weight: 2	Weight: 1
Value: 3	Value: 1	Value: 3

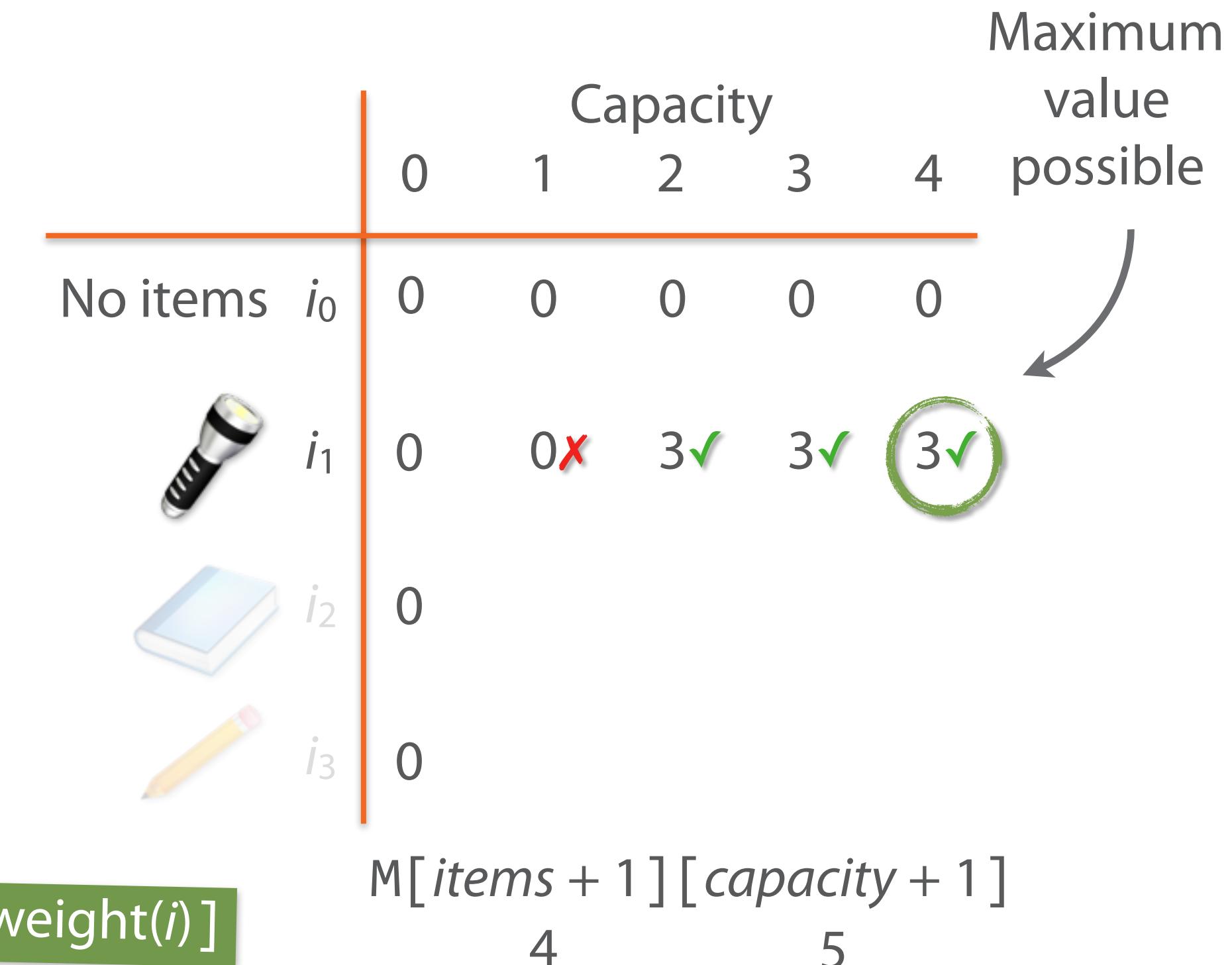
$$M[i][capacity] = \max(M[i-1][capacity], value(i) + M[i-1][capacity - weight(i)])$$

if excluded:

$$M[i-1][capacity]$$

if included and small enough:

$$value(i) + M[i-1][capacity - weight(i)]$$



# The 0/1 Knapsack Problem

Knapsack capacity: 4		
		
Weight: 2	Weight: 2	Weight: 1

Value: 3      Value: 1      Value: 3

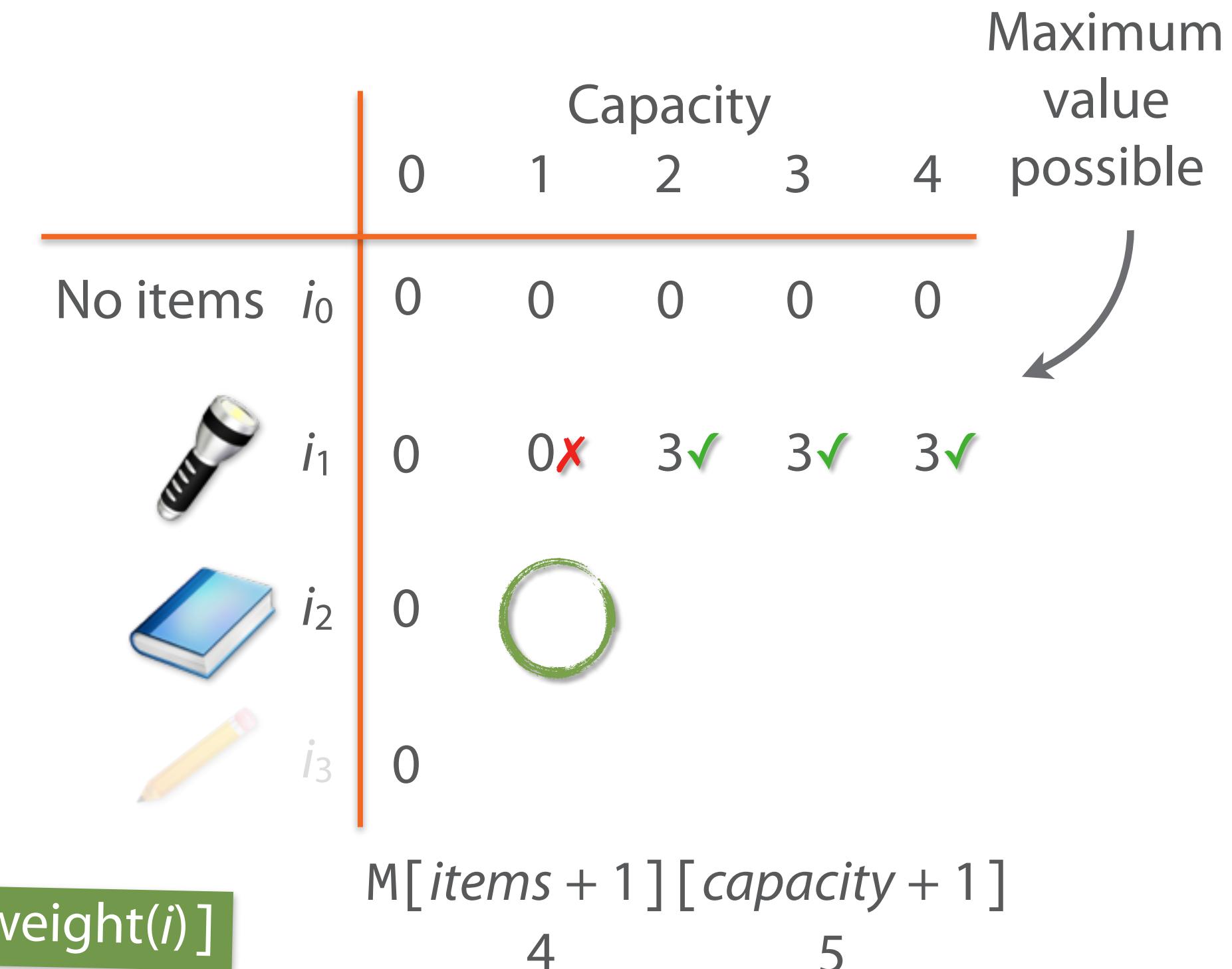
$$M[i][capacity] = \max(\text{value}(i), M[i-1][capacity])$$

if excluded:

$$M[i-1][capacity]$$

if included and small enough:

$$\text{value}(i) + M[i-1][capacity - weight(i)]$$



# The 0/1 Knapsack Problem

Knapsack capacity: 4		
		
Weight: 2	Weight: 2	Weight: 1
Value: 3	Value: 1	Value: 3

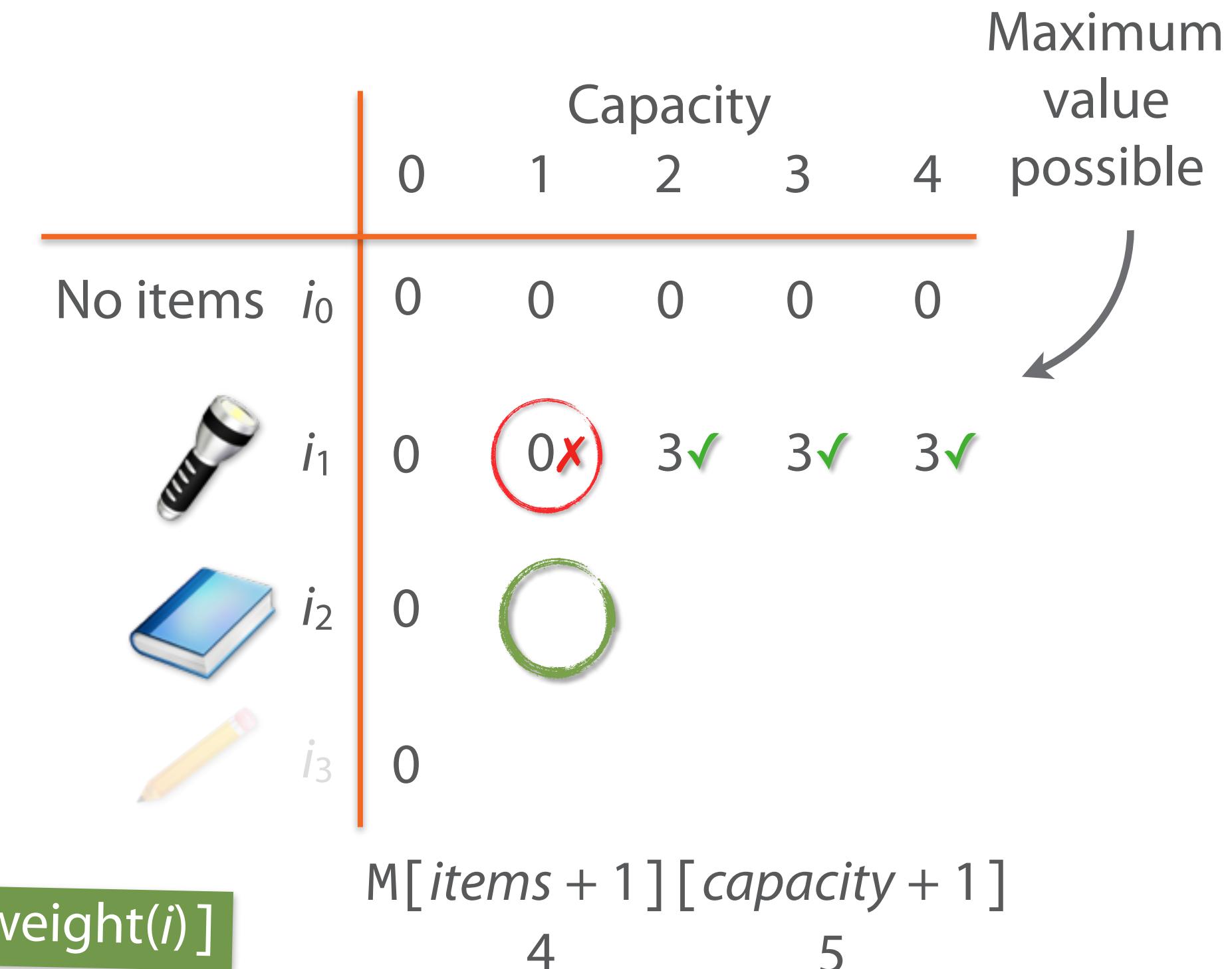
$$M[i][capacity] = \max(M[i-1][capacity], value(i) + M[i-1][capacity - weight(i)])$$

if excluded:

$$M[i-1][capacity]$$

if included and small enough:

$$value(i) + M[i-1][capacity - weight(i)]$$



# The 0/1 Knapsack Problem

Knapsack capacity: 4		
		
Weight: 2	Weight: 2	Weight: 1

Value: 3      Value: 1      Value: 3

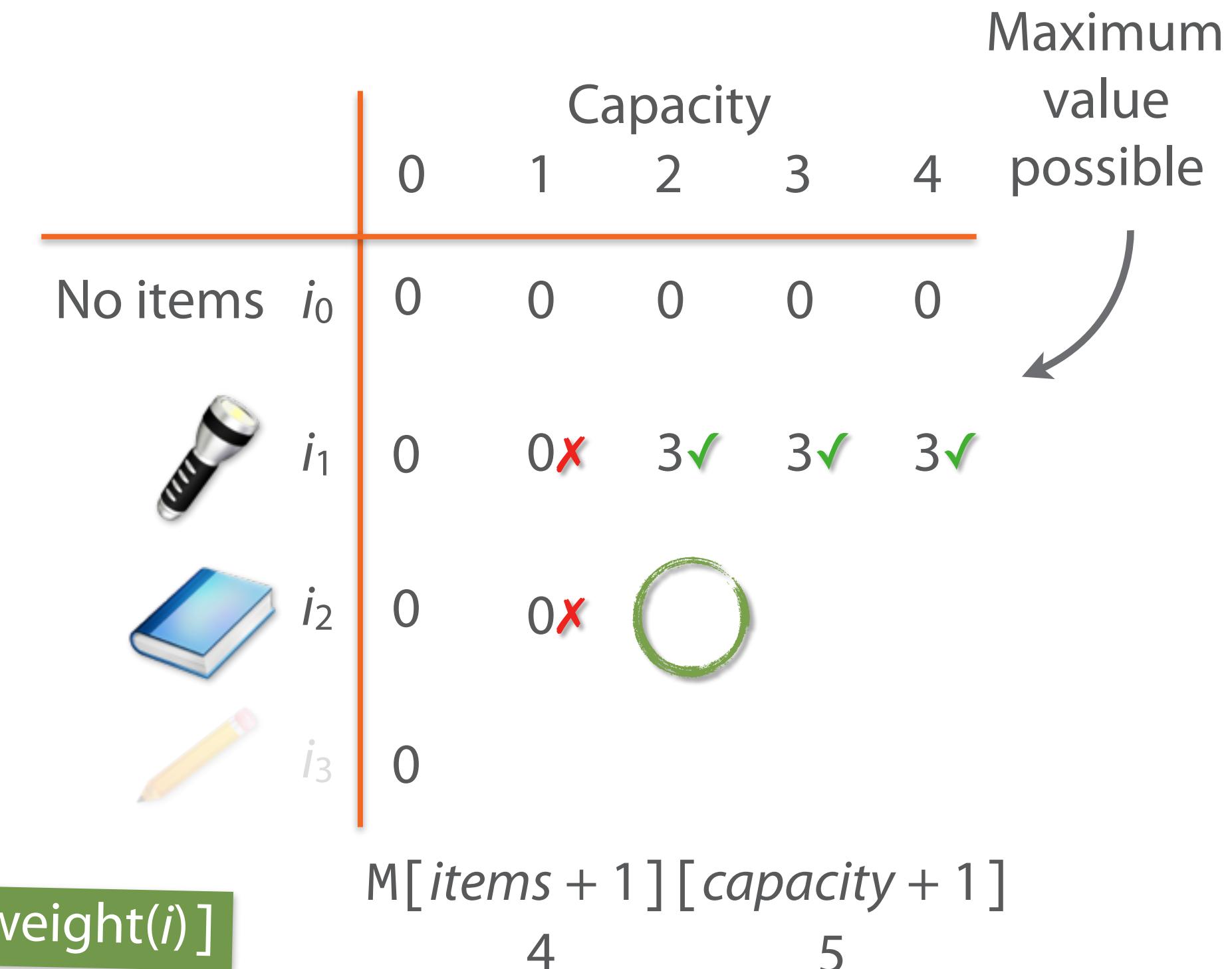
$$M[i][capacity] = \max(M[i-1][capacity], M[i-1][capacity - weight(i)] + value(i))$$

if excluded:

$$M[i-1][capacity]$$

if included and small enough:

$$value(i) + M[i-1][capacity - weight(i)]$$



# The 0/1 Knapsack Problem

Knapsack capacity: 4		
		
Weight: 2	Weight: 2	Weight: 1
Value: 3	Value: 1	Value: 3

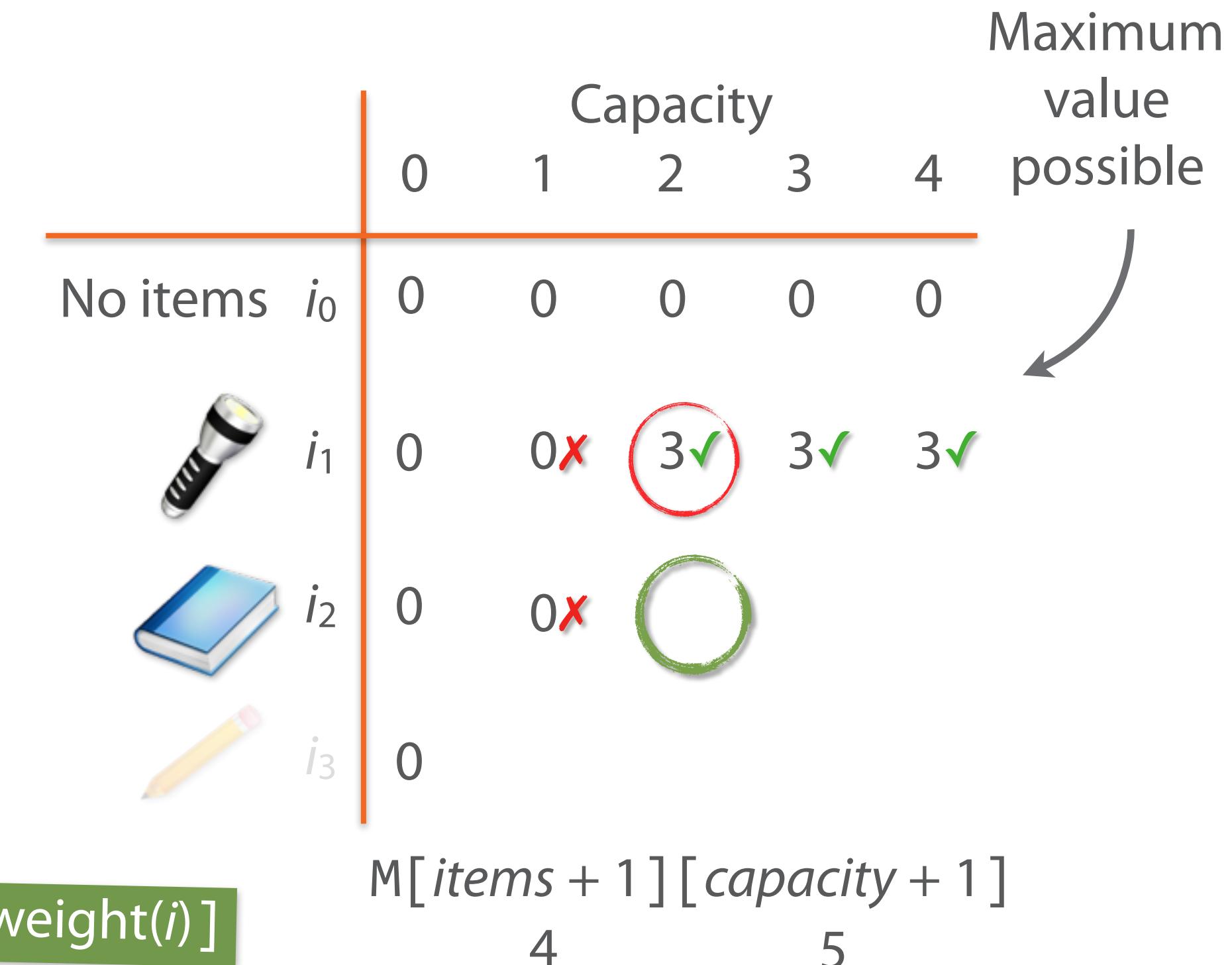
$$M[i][capacity] = \max(M[i-1][capacity], value(i) + M[i-1][capacity - weight(i)])$$

if excluded:

$$M[i-1][capacity]$$

if included and small enough:

$$value(i) + M[i-1][capacity - weight(i)]$$



# The 0/1 Knapsack Problem

Knapsack capacity: 4		
Weight: 2	Weight: 2	Weight: 1
Value: 3	Value: 1	Value: 3

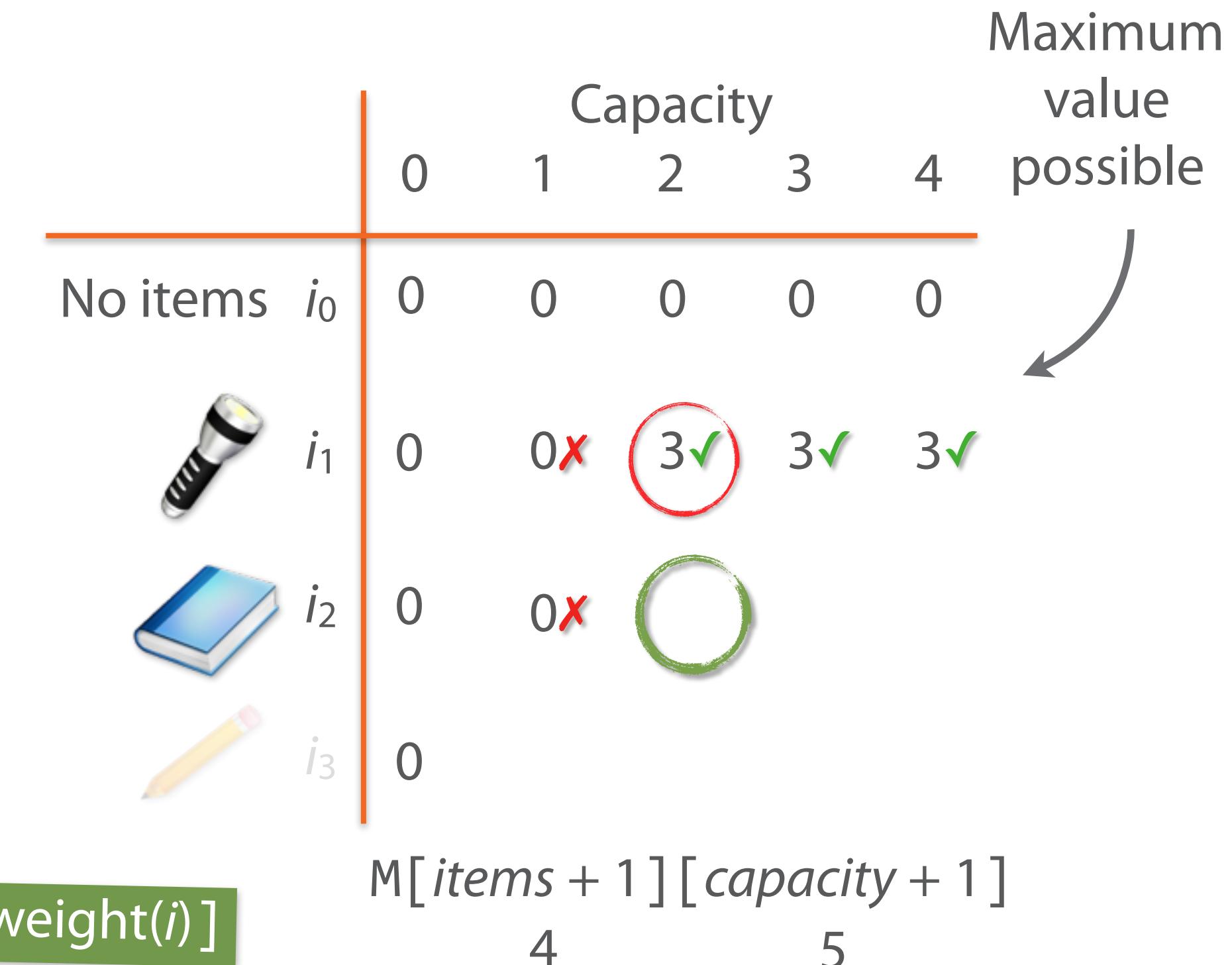
$$M[i][capacity] = \max(M[i-1][capacity], value(i) + M[i-1][capacity - weight(i)])$$

if excluded:

$$M[i-1][capacity]$$

if included and small enough:

$$value(i) + M[i-1][capacity - weight(i)]$$



# The 0/1 Knapsack Problem

Knapsack capacity: 4		
		
Weight: 2	Weight: 2	Weight: 1
Value: 3	Value: 1	Value: 3

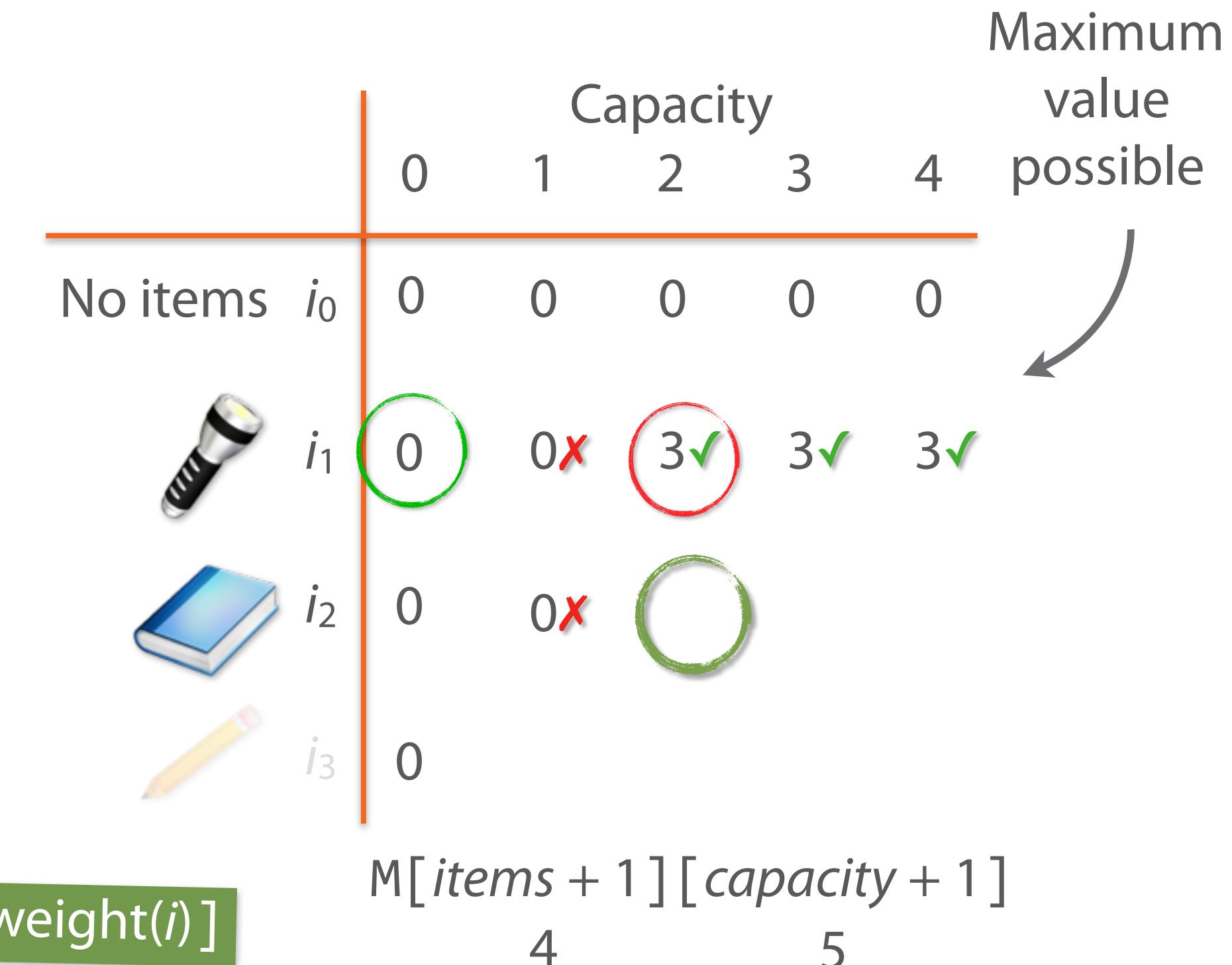
$$M[i][capacity] = \max(M[i-1][capacity], value(i) + M[i-1][capacity - weight(i)])$$

if excluded:

$$M[i-1][capacity]$$

if included and small enough:

$$value(i) + M[i-1][capacity - weight(i)]$$



# The 0/1 Knapsack Problem

Knapsack capacity: 4		
Weight: 2	Weight: 2	Weight: 1
Value: 3	Value: 1	Value: 3

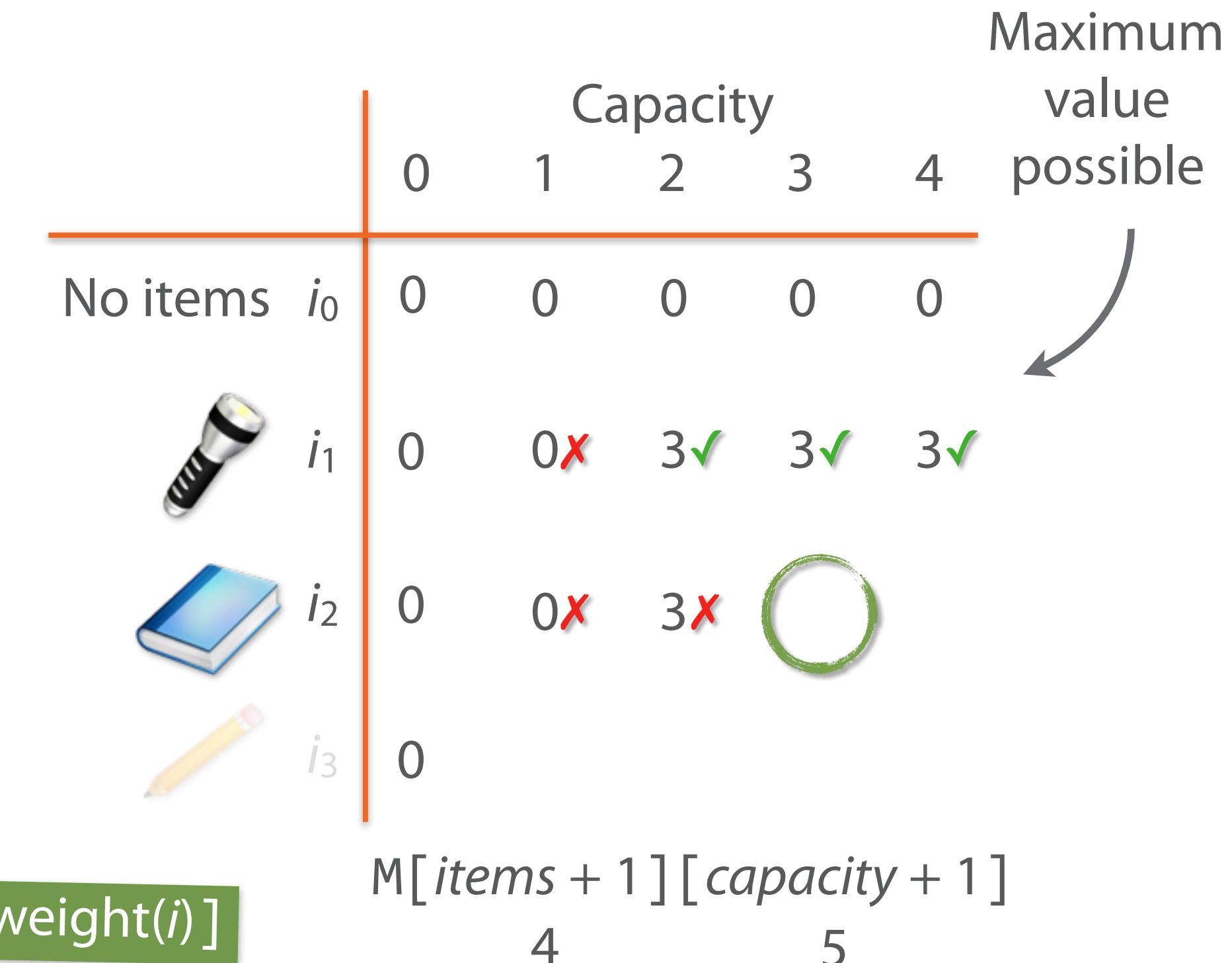
$$M[i][capacity] = \max(\text{value}(i) + M[i-1][capacity - weight(i)], M[i-1][capacity])$$

if excluded:

$$M[i-1][capacity]$$

if included and small enough:

$$\text{value}(i) + M[i-1][capacity - weight(i)]$$



# The 0/1 Knapsack Problem

Knapsack capacity: 4

Weight: 2    Weight: 2    Weight: 1  
 Value: 3    Value: 1    Value: 3

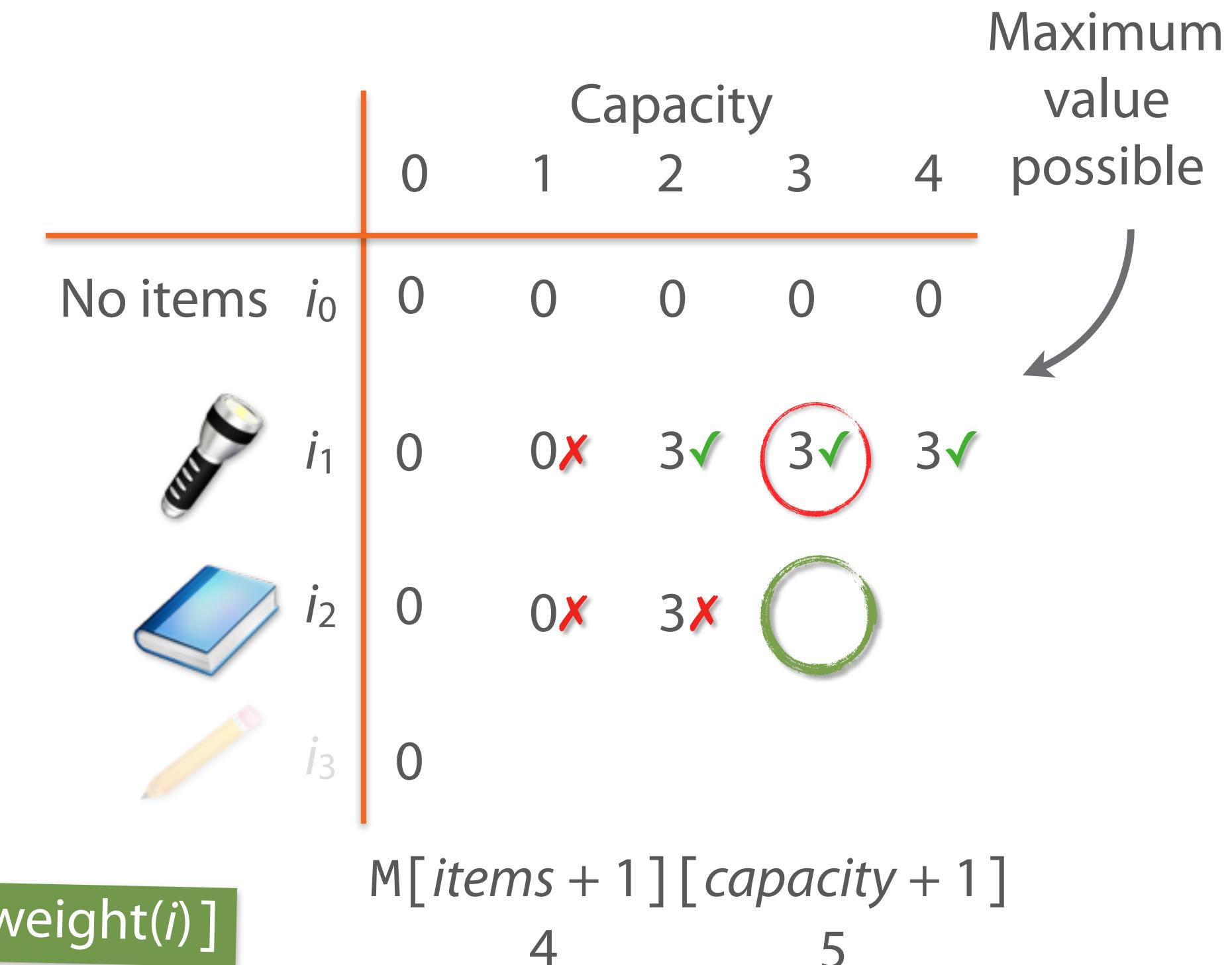
$$M[i][capacity] = \max(M[i-1][capacity], value(i) + M[i-1][capacity - weight(i)])$$

if excluded:

$$M[i-1][capacity]$$

if included and small enough:

$$value(i) + M[i-1][capacity - weight(i)]$$



# The 0/1 Knapsack Problem

Knapsack capacity: 4



Weight: 2    Weight: 2    Weight: 1  
 Value: 3    Value: 1    Value: 3

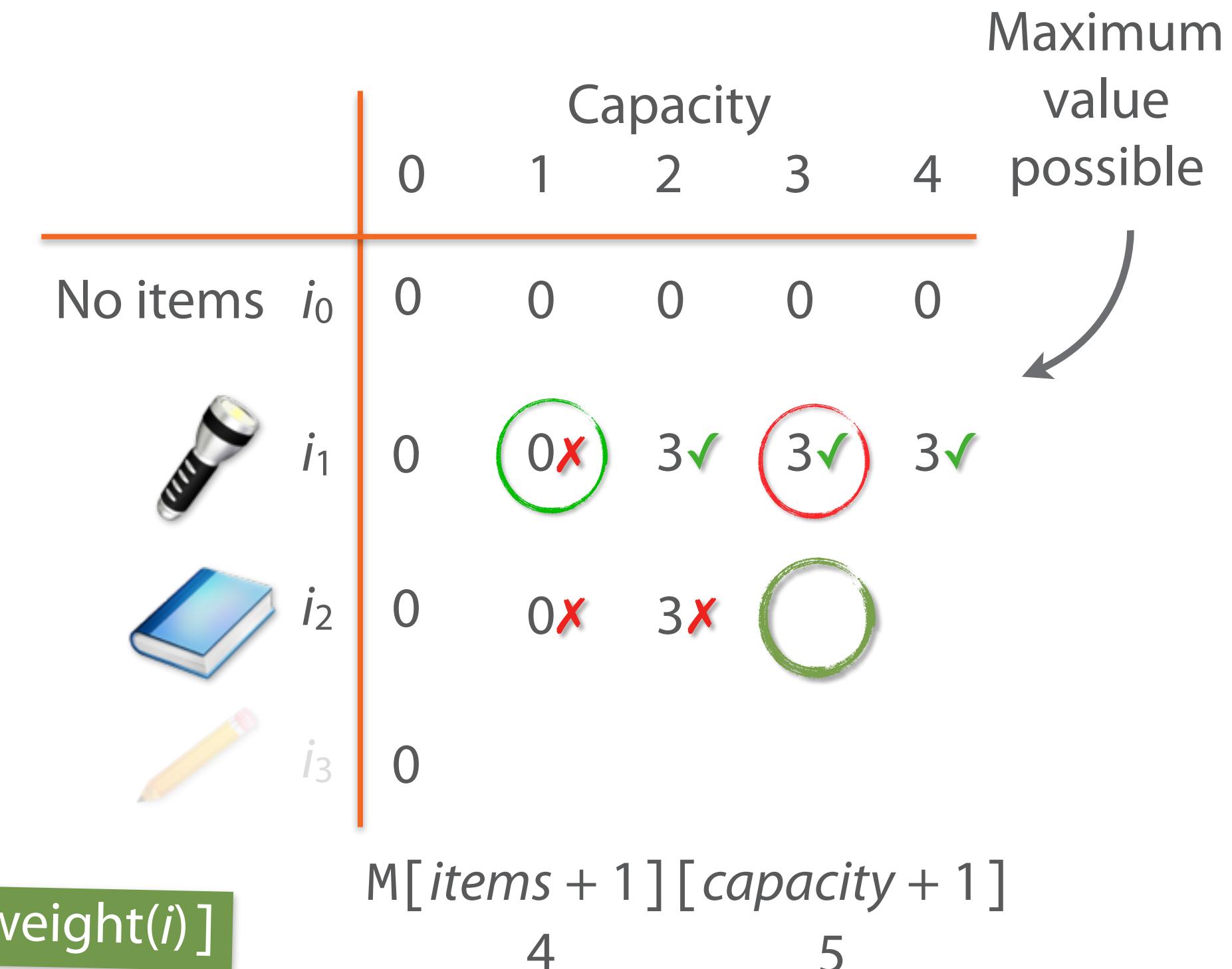
$$M[i][capacity] = \max(M[i-1][capacity], value(i) + M[i-1][capacity - weight(i)])$$

if excluded:

$$M[i-1][capacity]$$

if included and small enough:

$$value(i) + M[i-1][capacity - weight(i)]$$



# The 0/1 Knapsack Problem

Knapsack capacity: 4		
Weight: 2	Weight: 2	Weight: 1
Value: 3	Value: 1	Value: 3

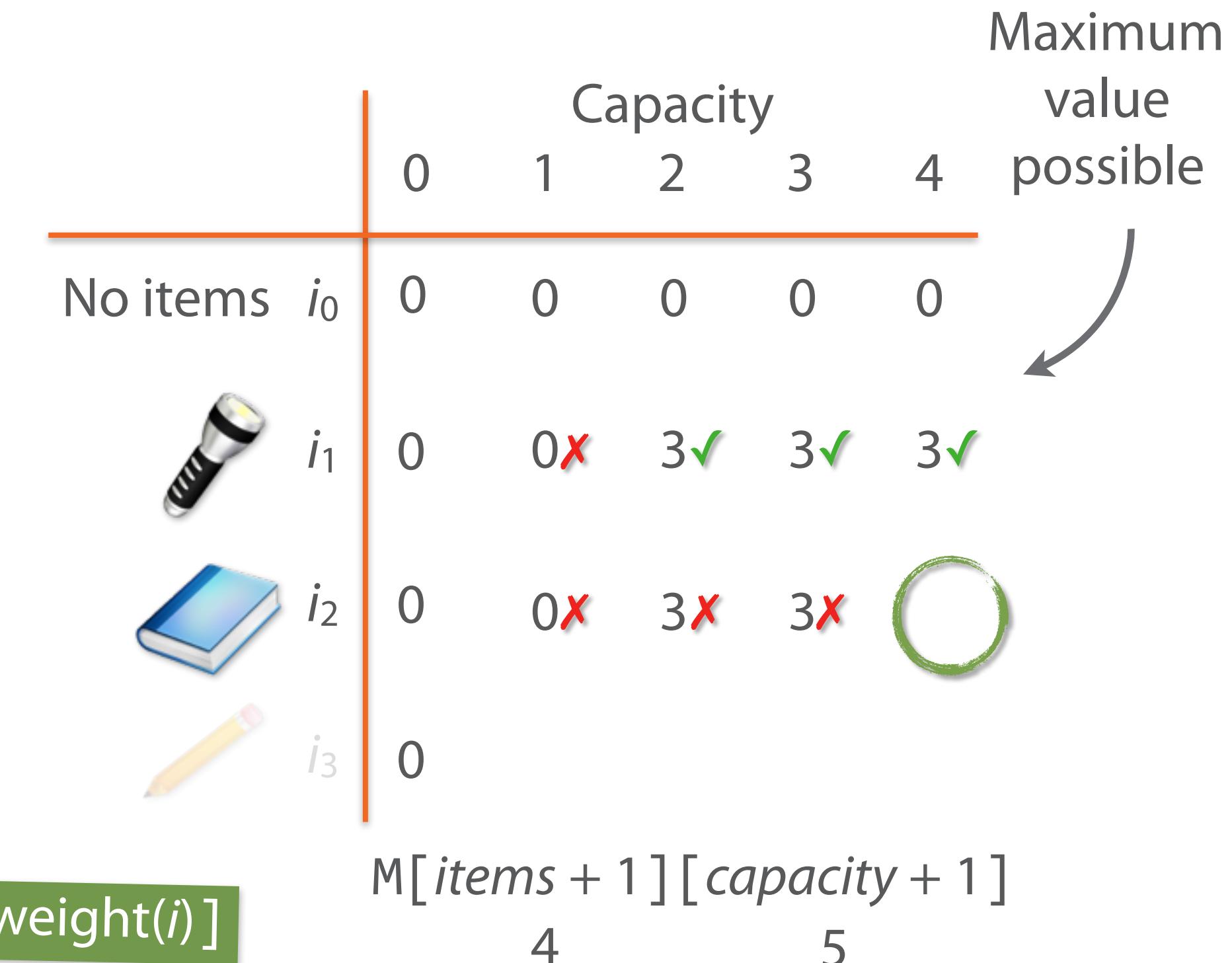
$$M[i][capacity] = \max(M[i-1][capacity], value(i) + M[i-1][capacity - weight(i)])$$

if excluded:

$$M[i-1][capacity]$$

if included and small enough:

$$value(i) + M[i-1][capacity - weight(i)]$$



# The 0/1 Knapsack Problem

Knapsack capacity: 4



Weight: 2    Weight: 2    Weight: 1  
 Value: 3    Value: 1    Value: 3

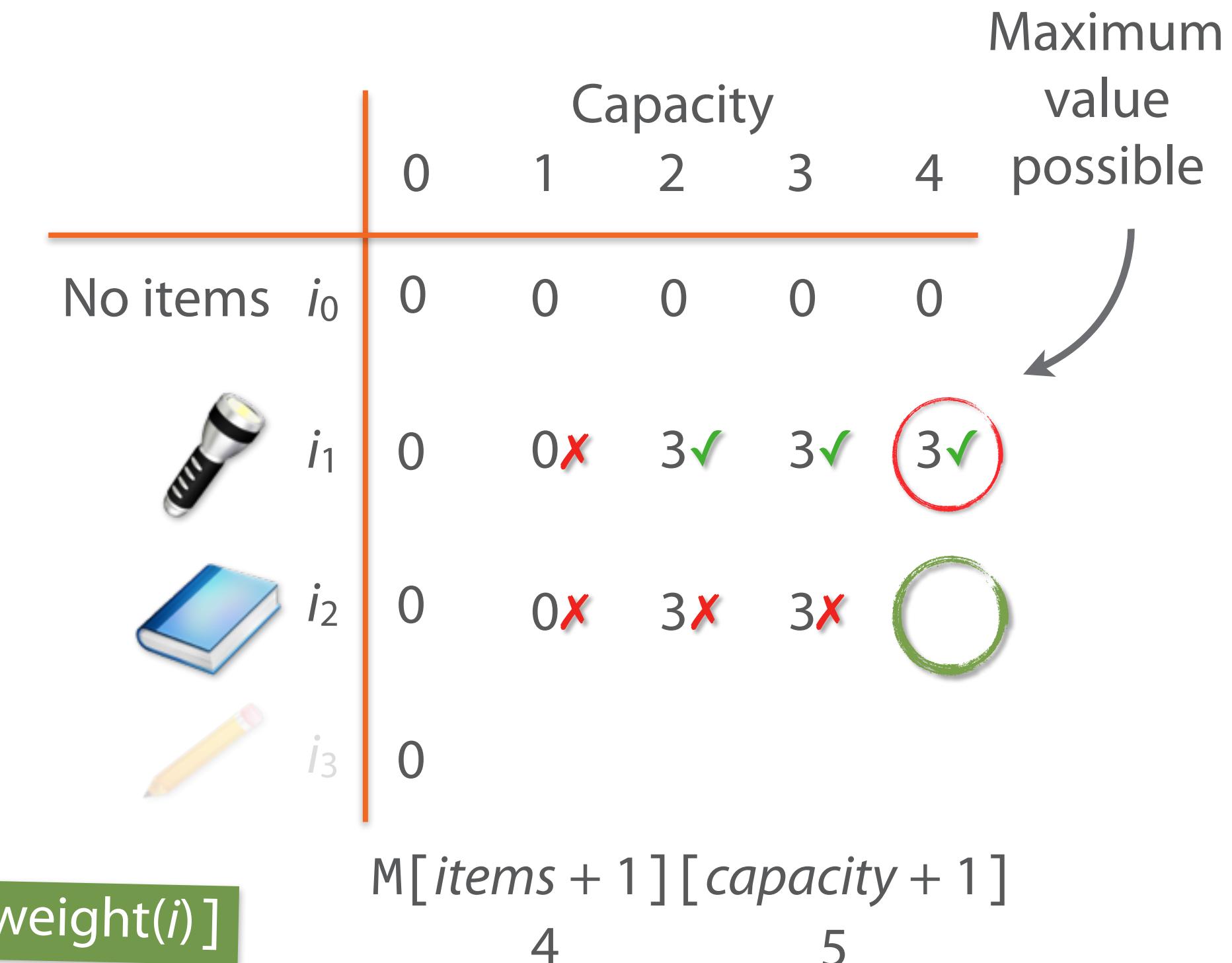
$$M[i][capacity] = \max(M[i-1][capacity], value(i) + M[i-1][capacity - weight(i)])$$

if excluded:

$$M[i-1][capacity]$$

if included and small enough:

$$value(i) + M[i-1][capacity - weight(i)]$$



# The 0/1 Knapsack Problem

Knapsack capacity: 4



Weight: 2    Weight: 2    Weight: 1  
 Value: 3    Value: 1    Value: 3

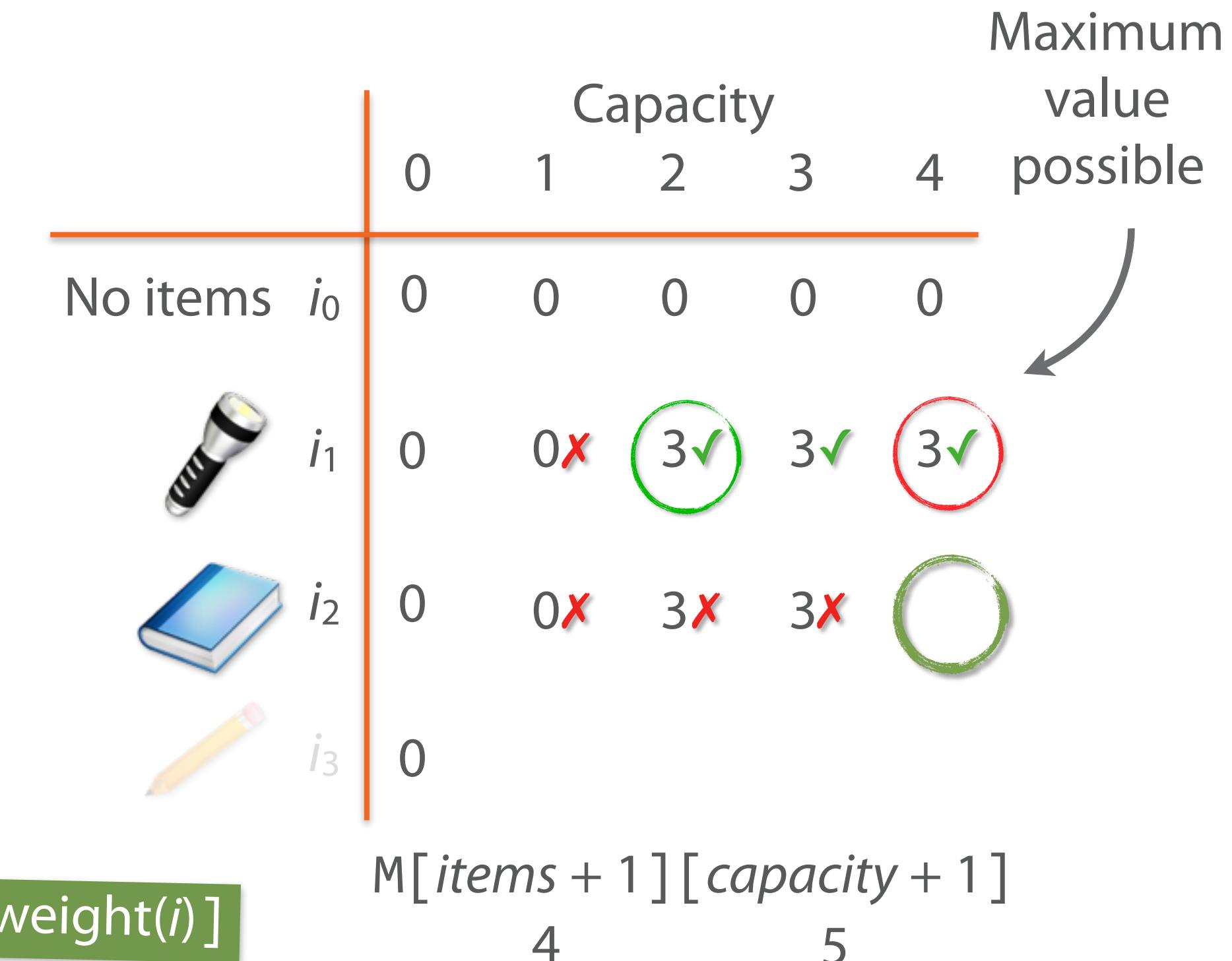
$$M[i][capacity] = \max(M[i-1][capacity], value(i) + M[i-1][capacity - weight(i)])$$

if excluded:

$$M[i-1][capacity]$$

if included and small enough:

$$value(i) + M[i-1][capacity - weight(i)]$$



# The 0/1 Knapsack Problem

Knapsack capacity: 4		
Weight: 2	Weight: 2	Weight: 1
Value: 3	Value: 1	Value: 3

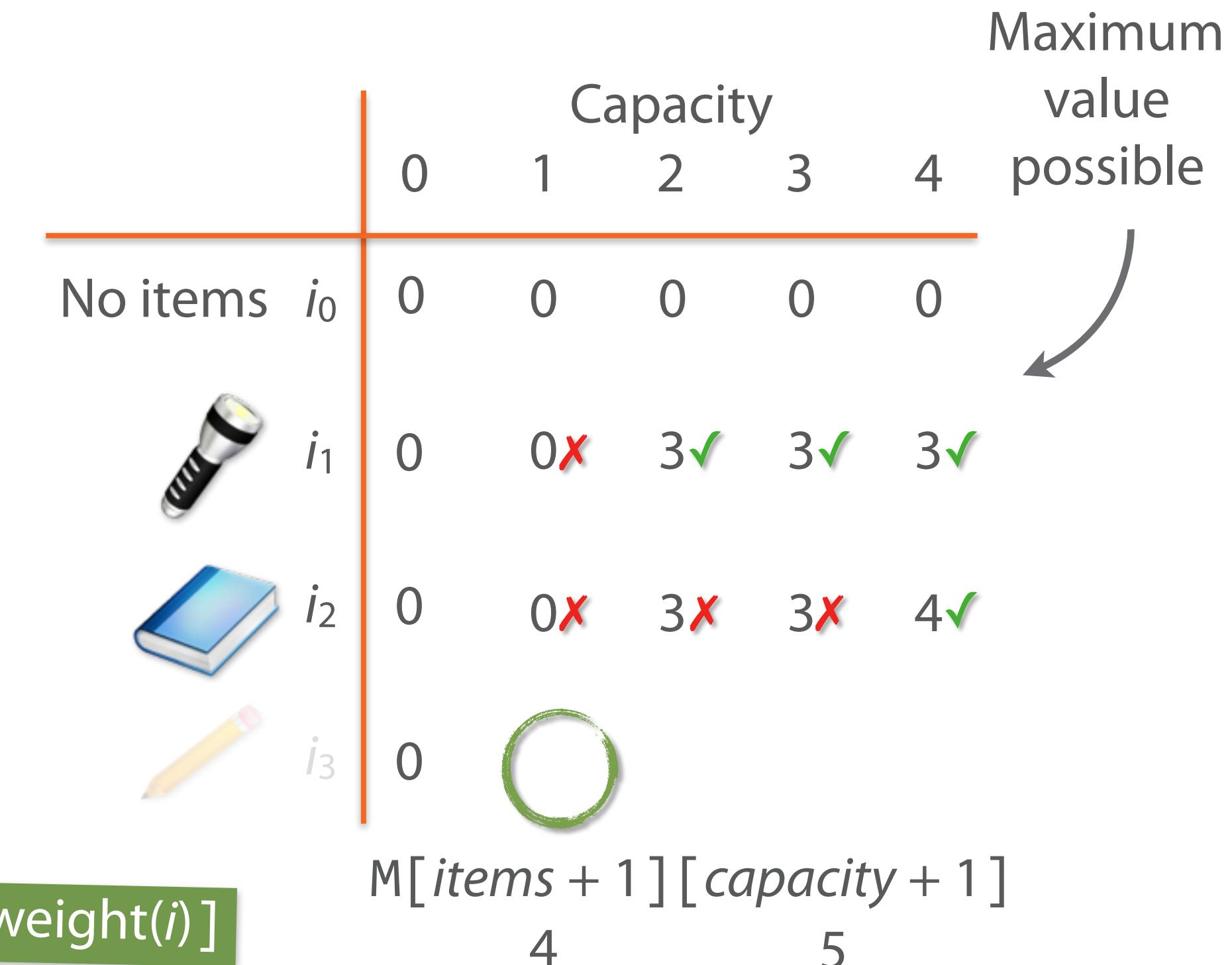
$$M[i][capacity] = \max(M[i-1][capacity], M[i-1][capacity - weight(i)] + value(i))$$

if excluded:

$$M[i-1][capacity]$$

if included and small enough:

$$value(i) + M[i-1][capacity - weight(i)]$$



# The 0/1 Knapsack Problem

Knapsack capacity: 4		
		
Weight: 2	Weight: 2	Weight: 1
Value: 3	Value: 1	Value: 3

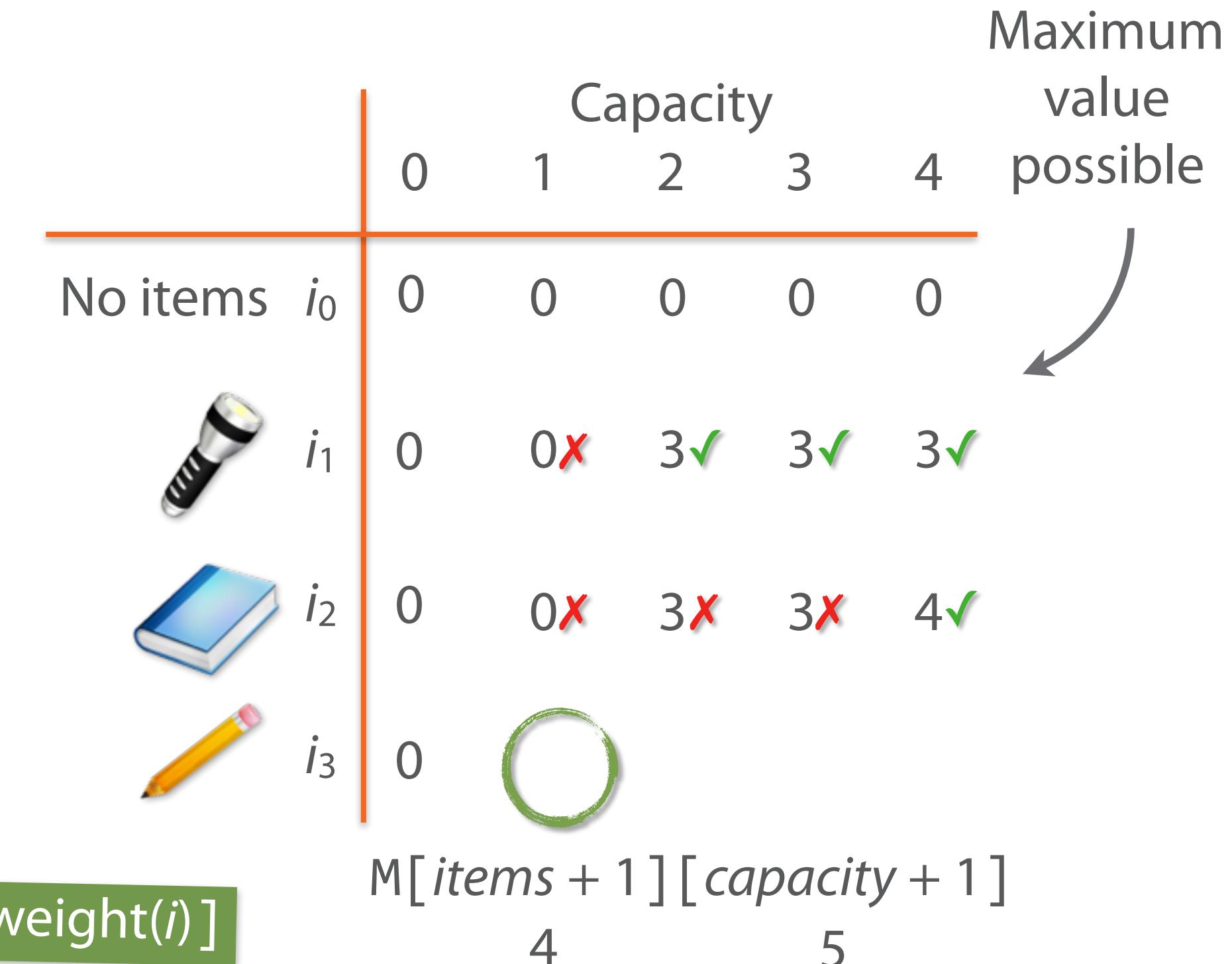
$$M[i][capacity] = \max(M[i-1][capacity], value(i) + M[i-1][capacity - weight(i)])$$

if excluded:

$$M[i-1][capacity]$$

if included and small enough:

$$value(i) + M[i-1][capacity - weight(i)]$$



# The 0/1 Knapsack Problem

Knapsack capacity: 4		
		
Weight: 2	Weight: 2	Weight: 1
Value: 3	Value: 1	Value: 3

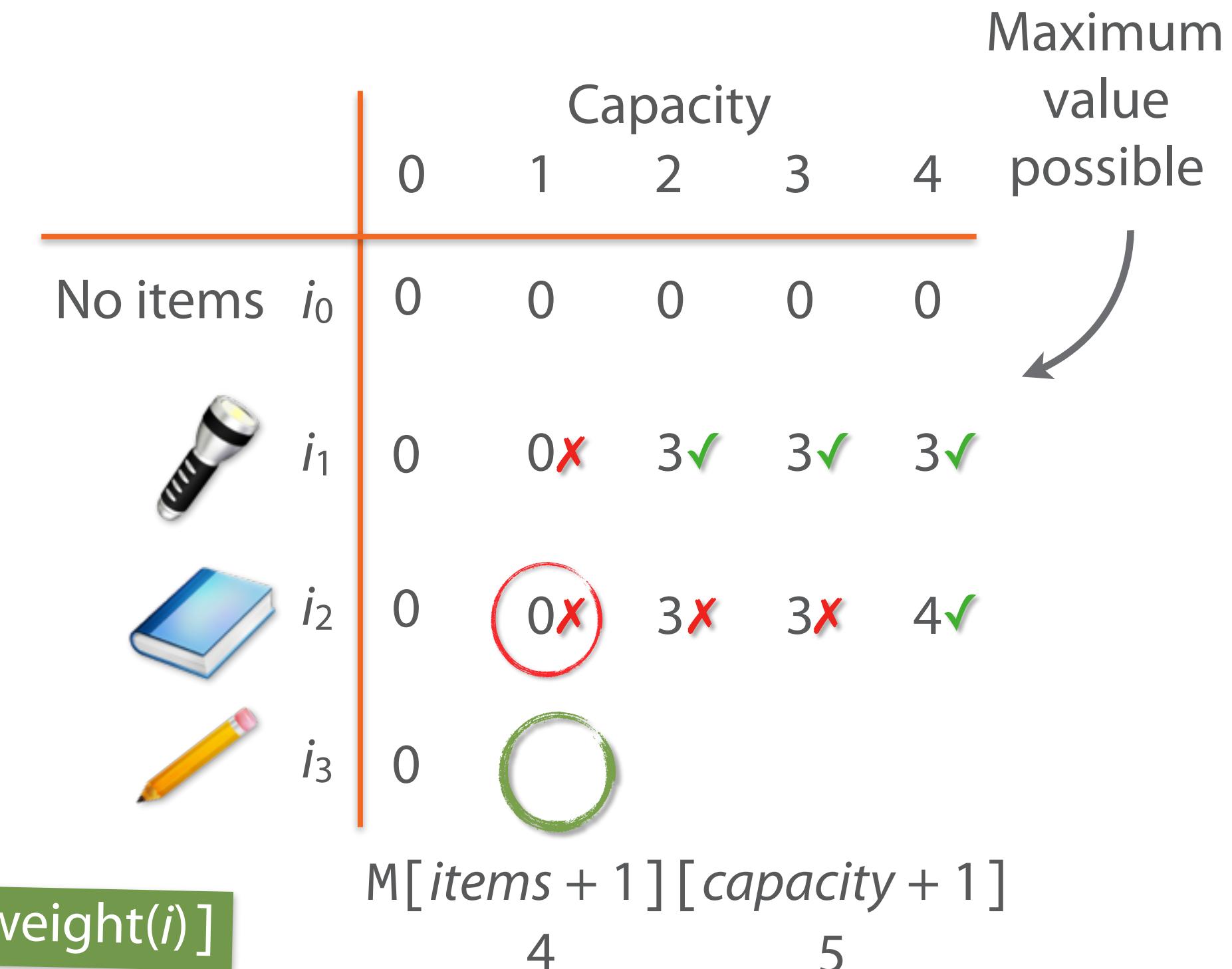
$$M[i][capacity] = \max(M[i-1][capacity], value(i) + M[i-1][capacity - weight(i)])$$

if excluded:

$$M[i-1][capacity]$$

if included and small enough:

$$value(i) + M[i-1][capacity - weight(i)]$$



# The 0/1 Knapsack Problem

Knapsack capacity: 4



Weight: 2    Weight: 2    Weight: 1  
 Value: 3    Value: 1    Value: 3

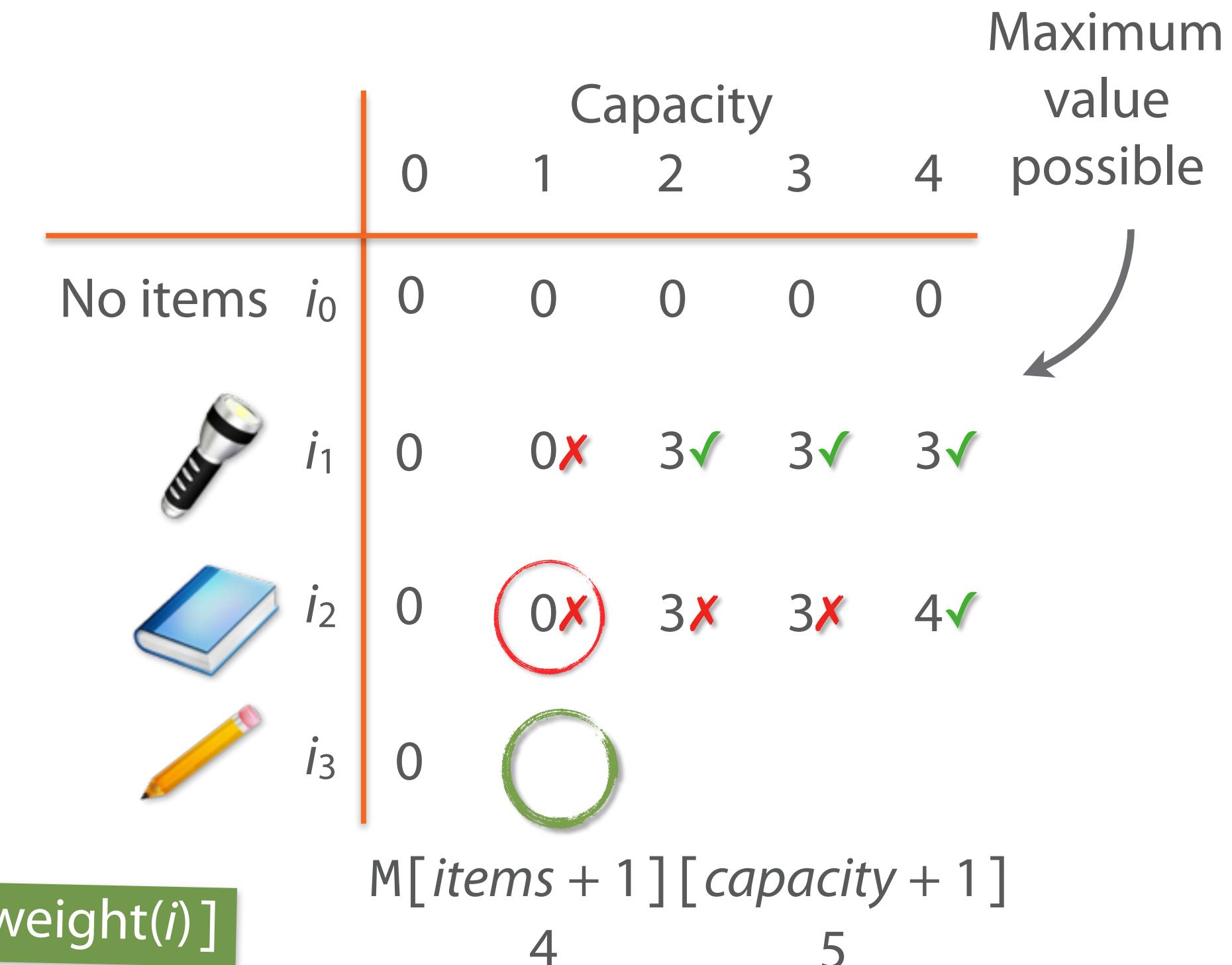
$$M[i][capacity] = \max(M[i-1][capacity], value(i) + M[i-1][capacity - weight(i)])$$

if excluded:

$$M[i-1][capacity]$$

if included and small enough:

$$value(i) + M[i-1][capacity - weight(i)]$$



# The 0/1 Knapsack Problem

Knapsack capacity: 4



Weight: 2    Weight: 2    Weight: 1  
 Value: 3    Value: 1    Value: 3

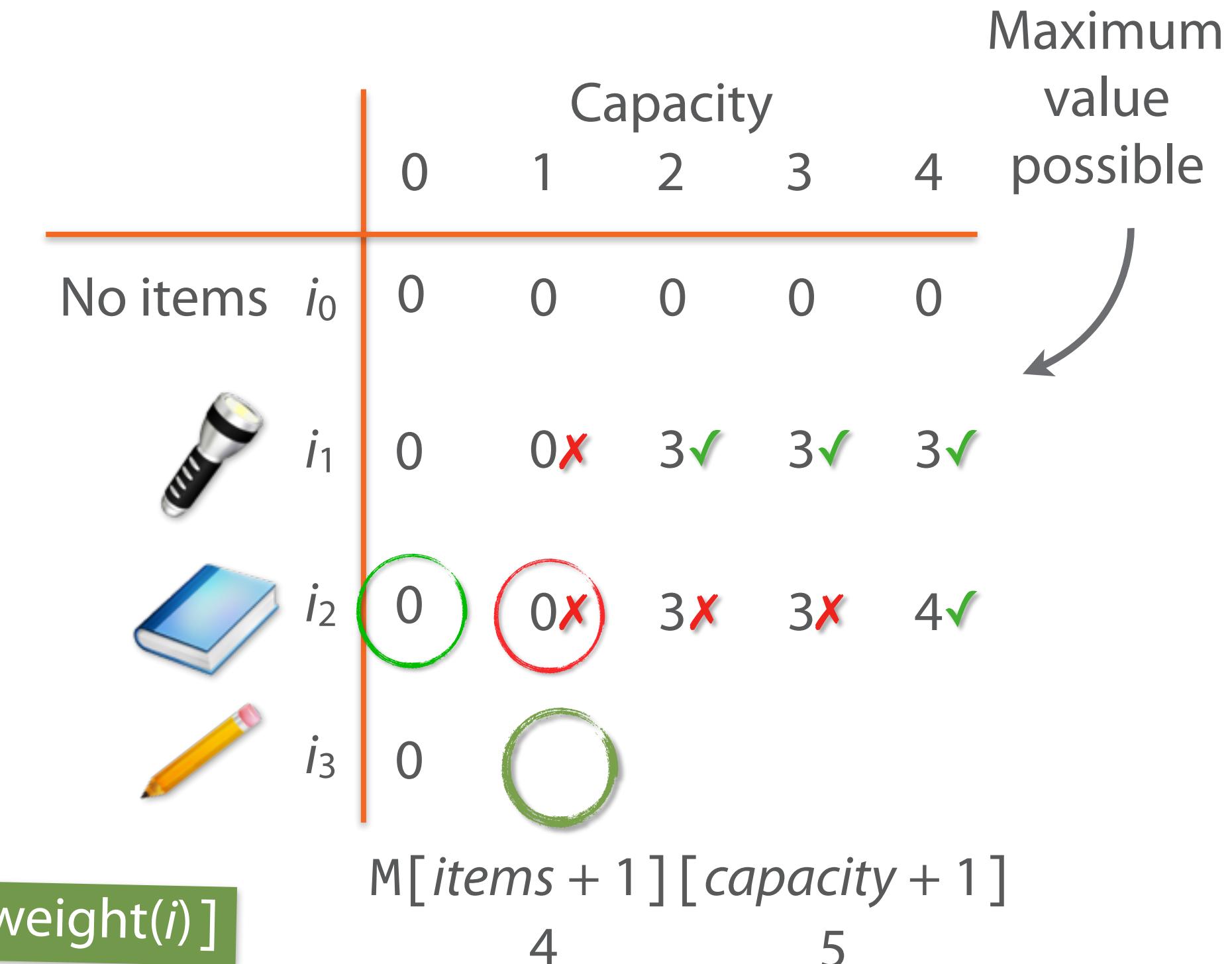
$$M[i][capacity] = \max(M[i-1][capacity], value(i) + M[i-1][capacity - weight(i)])$$

if excluded:

$$M[i-1][capacity]$$

if included and small enough:

$$value(i) + M[i-1][capacity - weight(i)]$$



# The 0/1 Knapsack Problem

Knapsack capacity: 4

Weight: 2    Weight: 2    Weight: 1  
 Value: 3    Value: 1    Value: 3

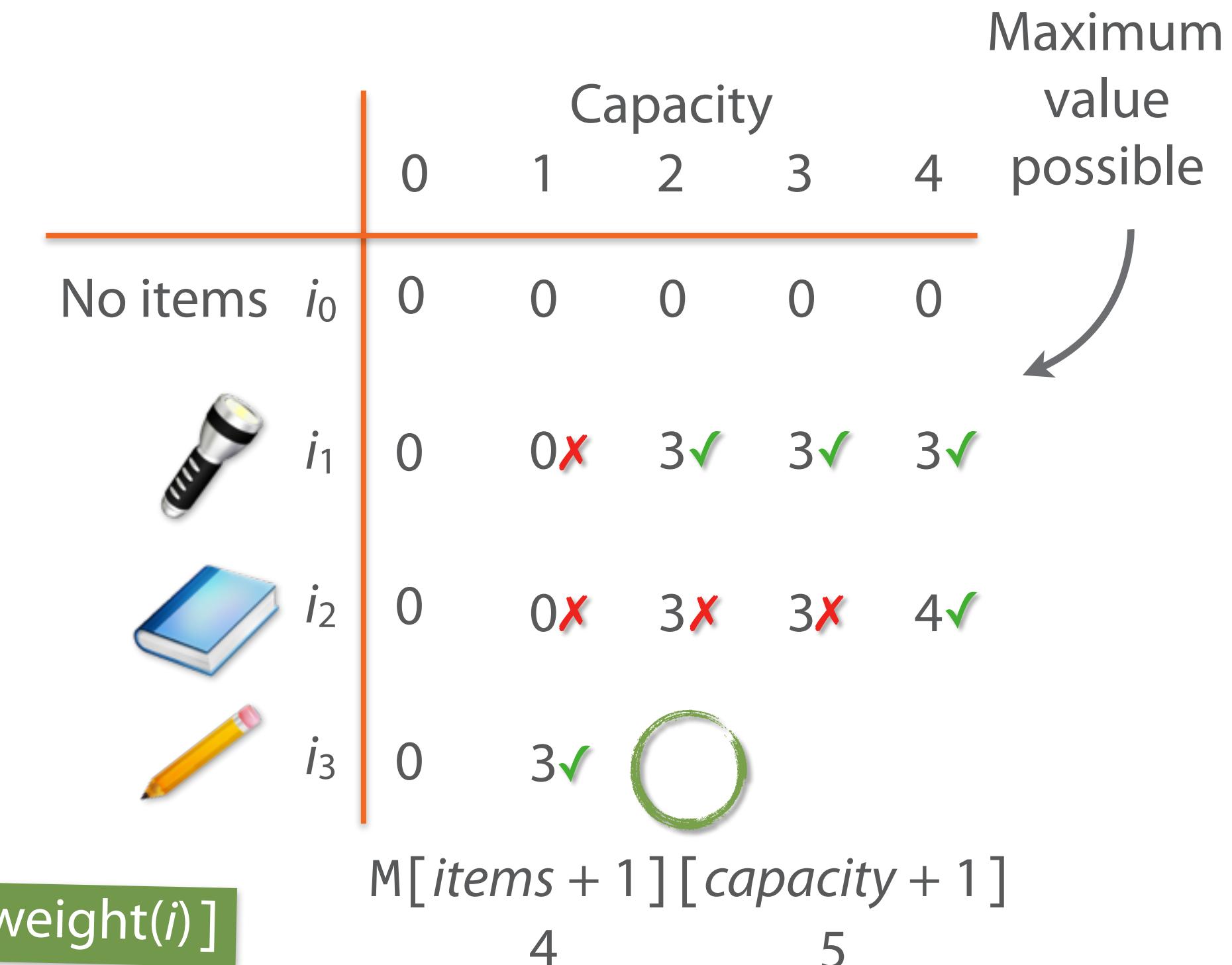
$$M[i][capacity] = \max(M[i-1][capacity], M[i-1][capacity - weight(i)] + value(i))$$

if excluded:

$$M[i-1][capacity]$$

if included and small enough:

$$value(i) + M[i-1][capacity - weight(i)]$$



# The 0/1 Knapsack Problem

Knapsack capacity: 4



Weight: 2    Weight: 2    Weight: 1  
 Value: 3    Value: 1    Value: 3

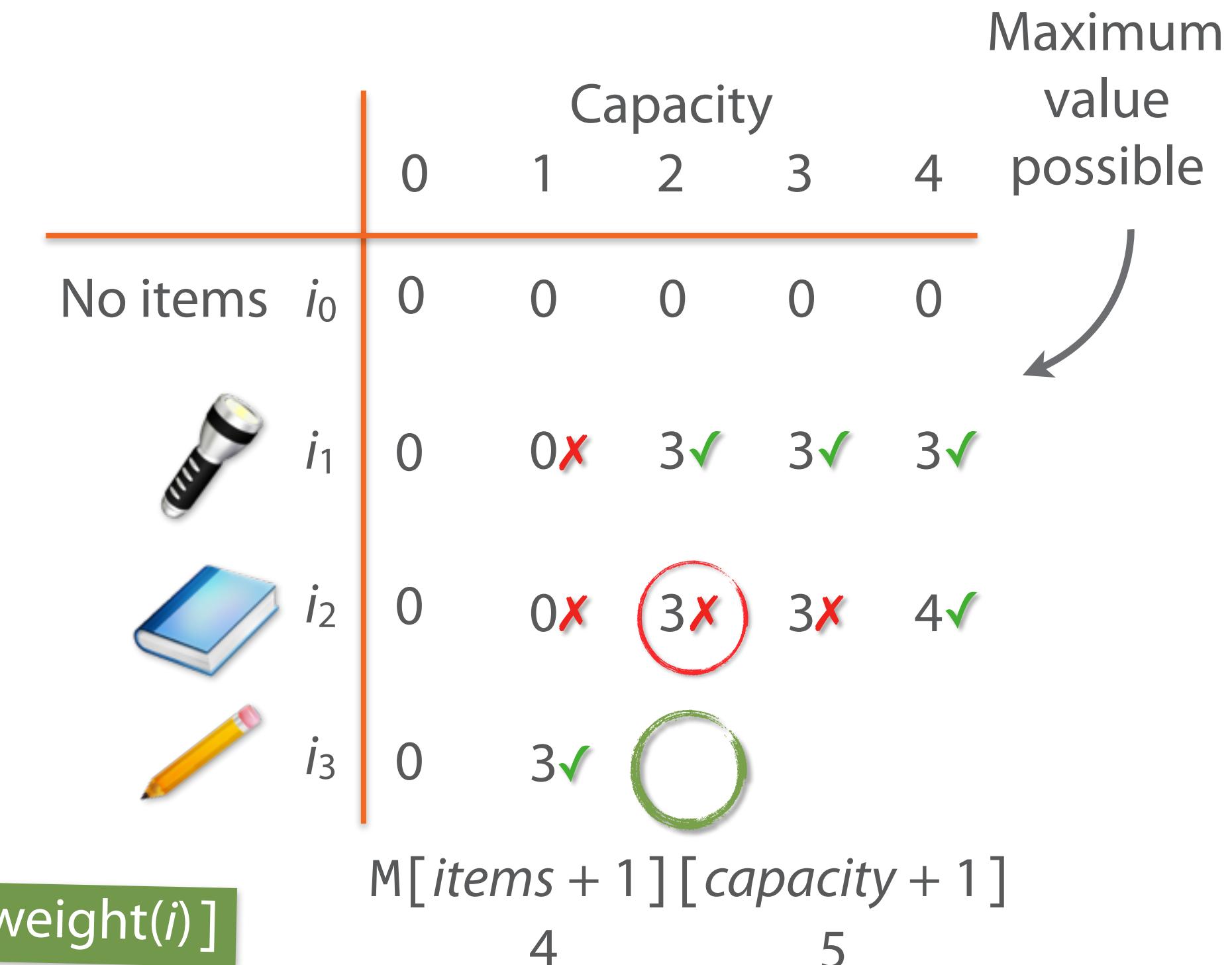
$$M[i][capacity] = \max(M[i-1][capacity], value(i) + M[i-1][capacity - weight(i)])$$

if excluded:

$$M[i-1][capacity]$$

if included and small enough:

$$value(i) + M[i-1][capacity - weight(i)]$$



# The 0/1 Knapsack Problem

Knapsack capacity: 4



Weight: 2    Weight: 2    Weight: 1  
 Value: 3    Value: 1    Value: 3

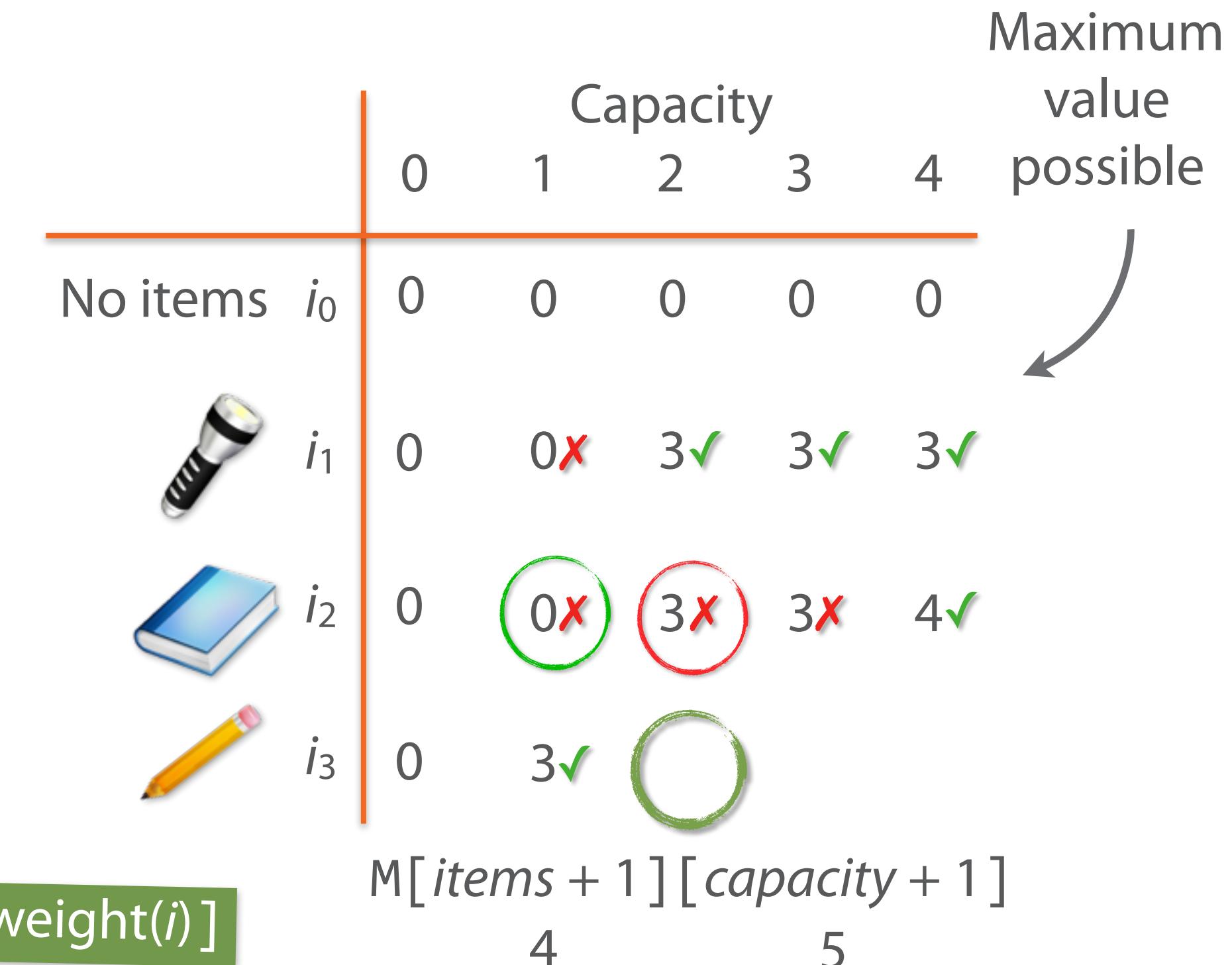
$$M[i][capacity] = \max(M[i-1][capacity], value(i) + M[i-1][capacity - weight(i)])$$

if excluded:

$$M[i-1][capacity]$$

if included and small enough:

$$value(i) + M[i-1][capacity - weight(i)]$$



# The 0/1 Knapsack Problem

Knapsack capacity: 4



Weight: 2    Weight: 2    Weight: 1  
 Value: 3    Value: 1    Value: 3

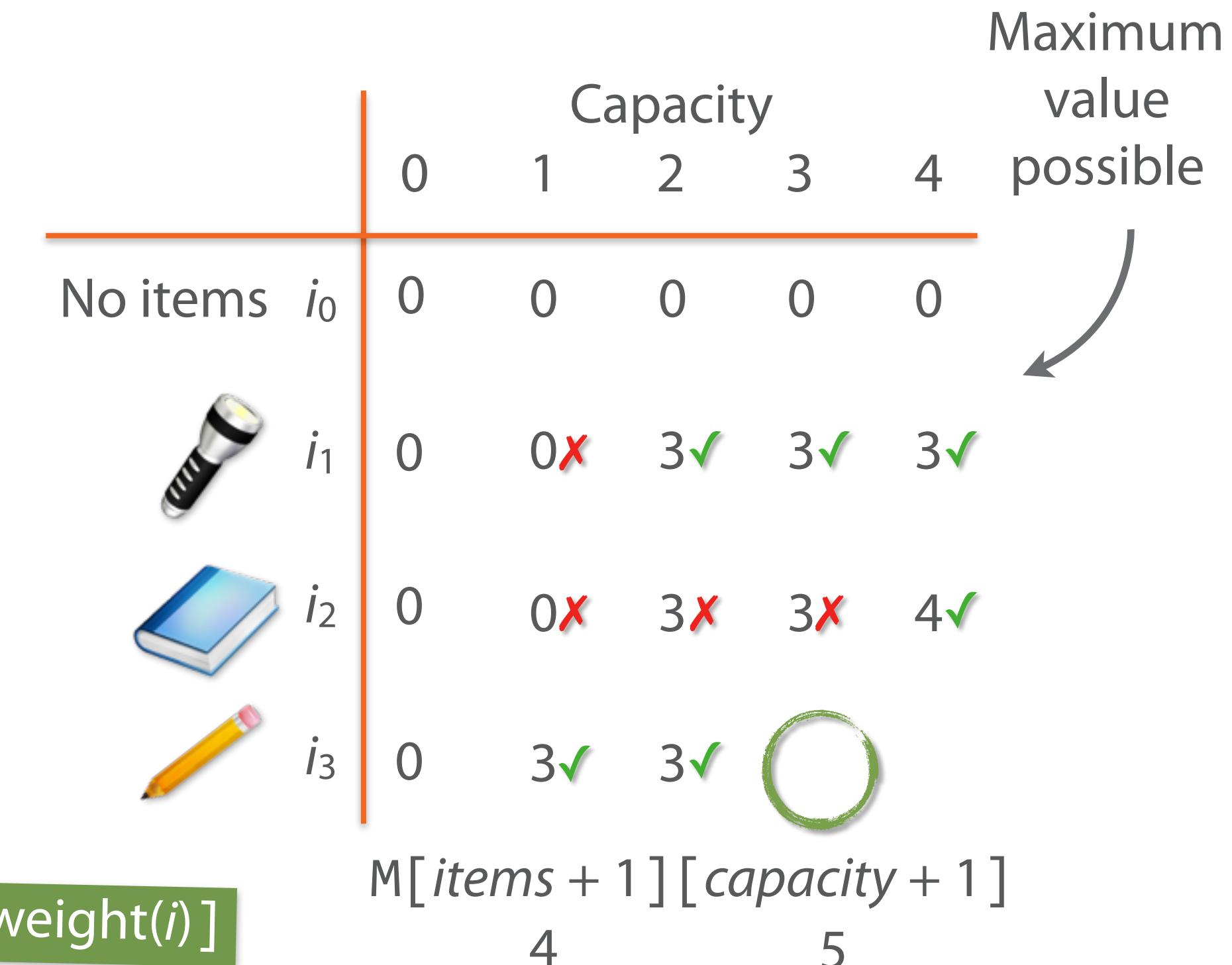
$$M[i][capacity] = \max(M[i-1][capacity], value(i) + M[i-1][capacity - weight(i)])$$

if excluded:

$$M[i-1][capacity]$$

if included and small enough:

$$value(i) + M[i-1][capacity - weight(i)]$$



# The 0/1 Knapsack Problem

Knapsack capacity: 4



Weight: 2    Weight: 2    Weight: 1  
 Value: 3    Value: 1    Value: 3

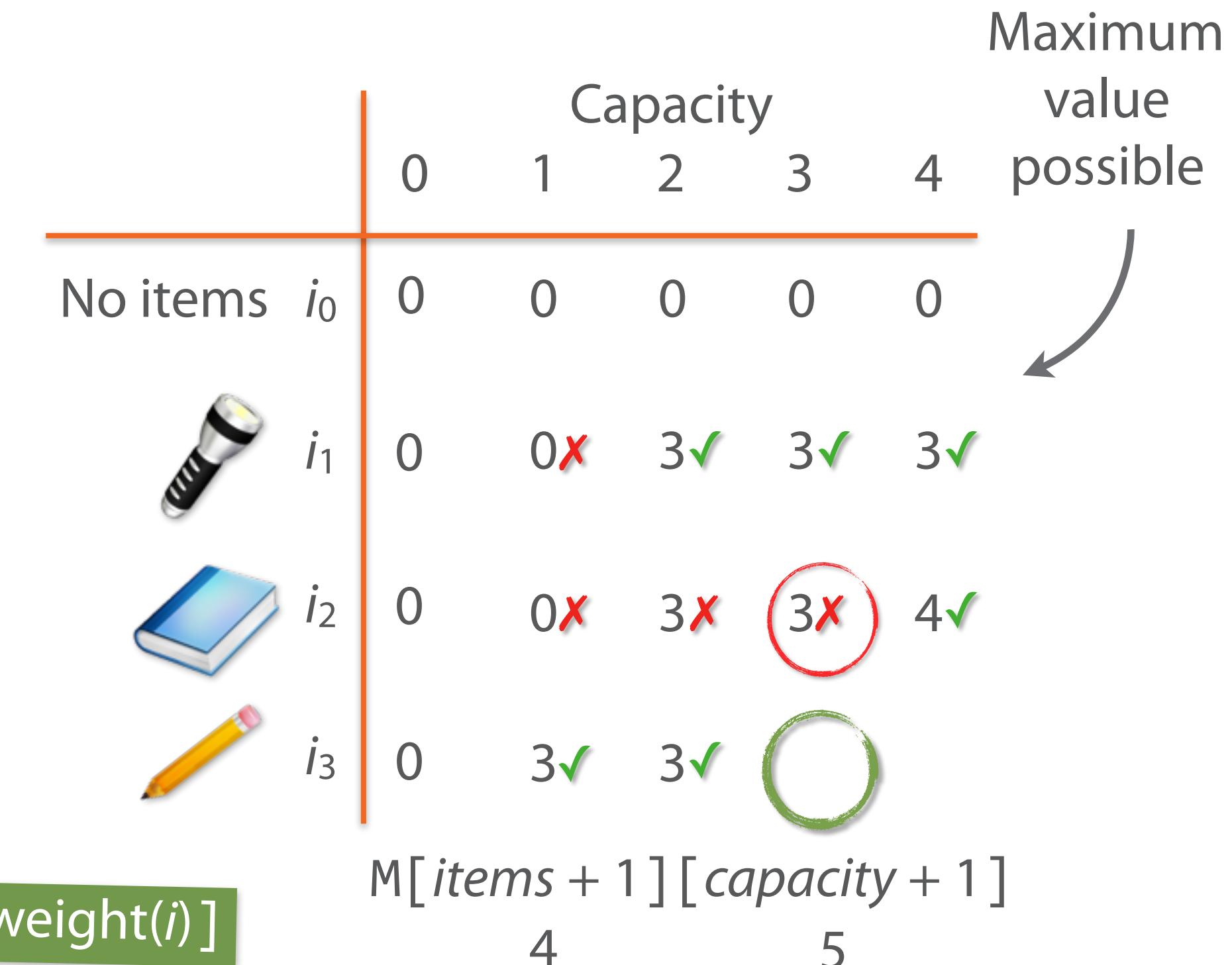
$$M[i][capacity] = \max(\text{excluded}, \text{included})$$

if excluded:

$$M[i-1][capacity]$$

if included and small enough:

$$\text{value}(i) + M[i-1][capacity - weight(i)]$$



# The 0/1 Knapsack Problem

Knapsack capacity: 4



Weight: 2    Weight: 2    Weight: 1  
 Value: 3    Value: 1    Value: 3

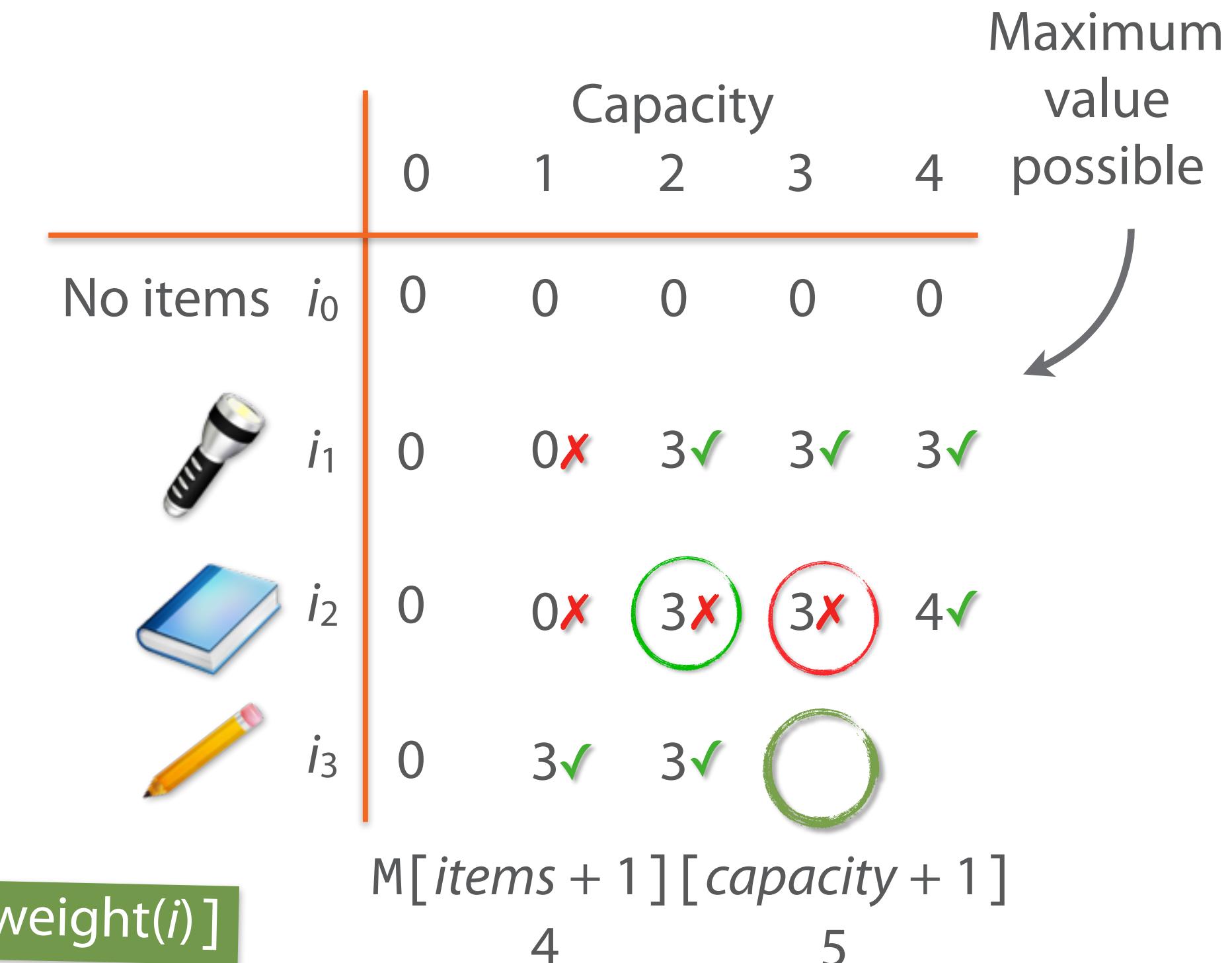
$$M[i][capacity] = \max(M[i-1][capacity], value(i) + M[i-1][capacity - weight(i)])$$

if excluded:

$$M[i-1][capacity]$$

if included and small enough:

$$value(i) + M[i-1][capacity - weight(i)]$$



# The 0/1 Knapsack Problem

Knapsack capacity: 4



Weight: 2    Weight: 2    Weight: 1  
 Value: 3    Value: 1    Value: 3

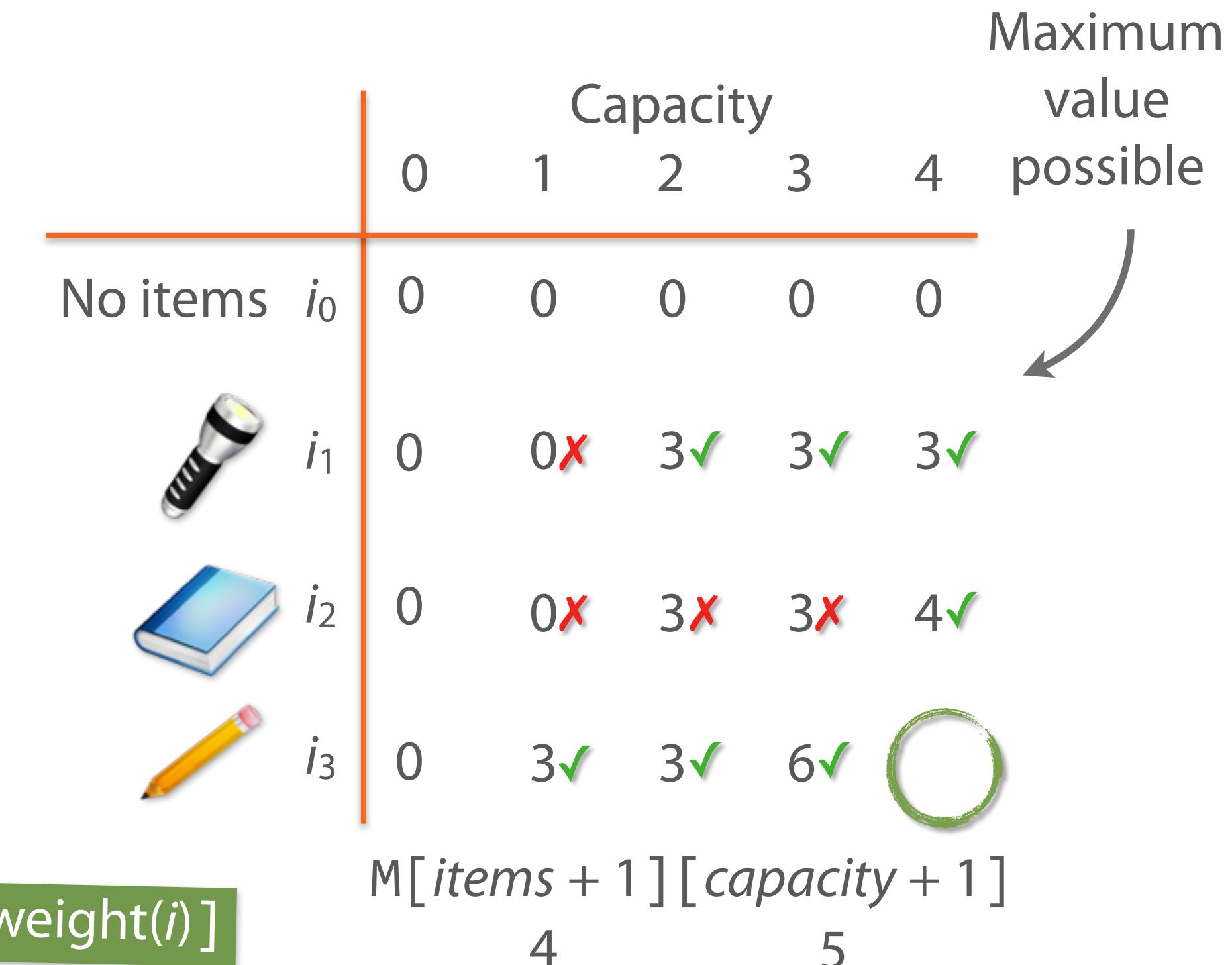
$$M[i][capacity] = \max(M[i-1][capacity], value(i) + M[i-1][capacity - weight(i)])$$

if excluded:

$$M[i-1][capacity]$$

if included and small enough:

$$value(i) + M[i-1][capacity - weight(i)]$$



# The 0/1 Knapsack Problem

Knapsack capacity: 4



Weight: 2    Weight: 2    Weight: 1  
 Value: 3    Value: 1    Value: 3

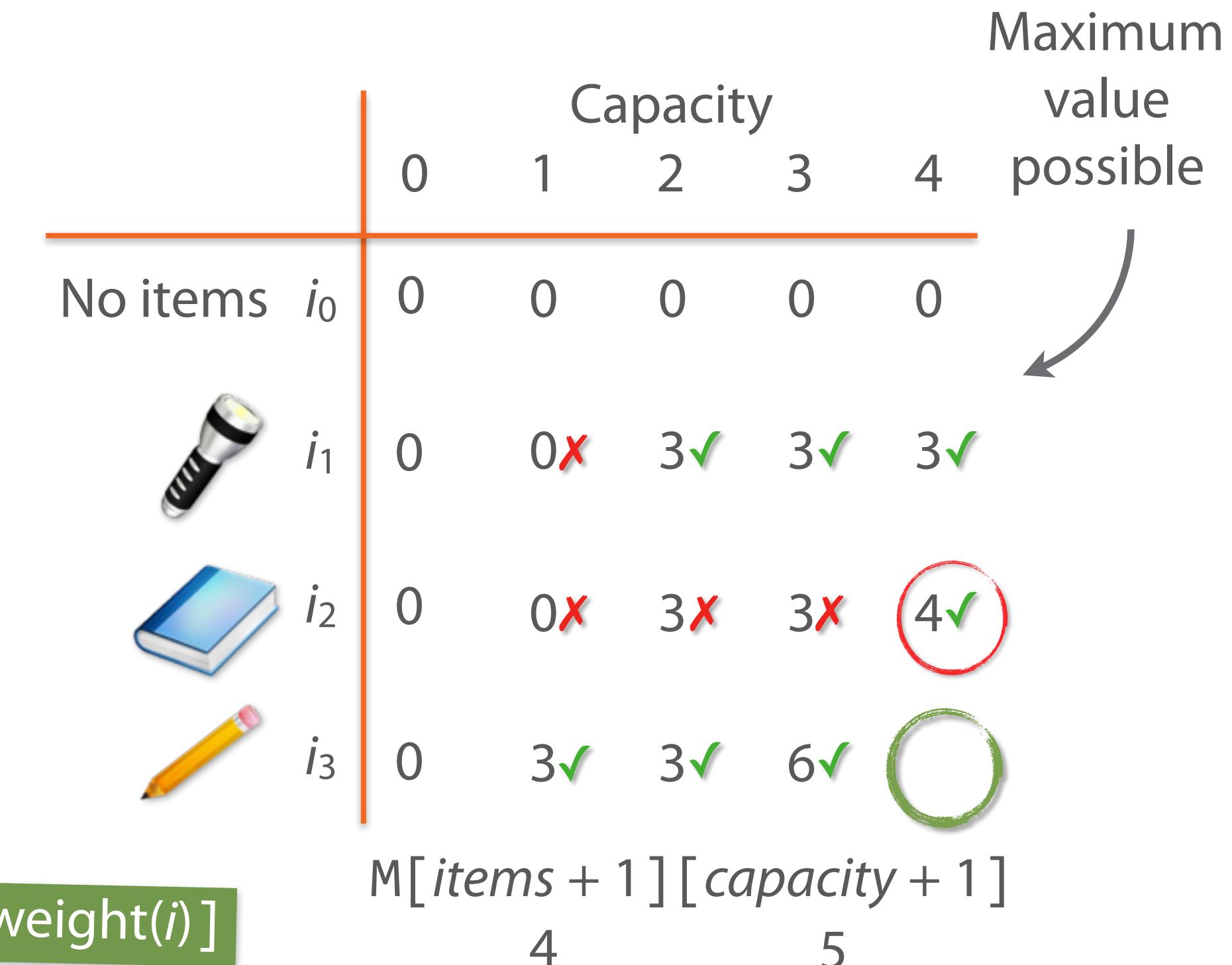
$$M[i][capacity] = \max(M[i-1][capacity], value(i) + M[i-1][capacity - weight(i)])$$

if excluded:

$$M[i-1][capacity]$$

if included and small enough:

$$value(i) + M[i-1][capacity - weight(i)]$$



# The 0/1 Knapsack Problem

Knapsack capacity: 4



Weight: 2    Weight: 2    Weight: 1  
 Value: 3    Value: 1    Value: 3

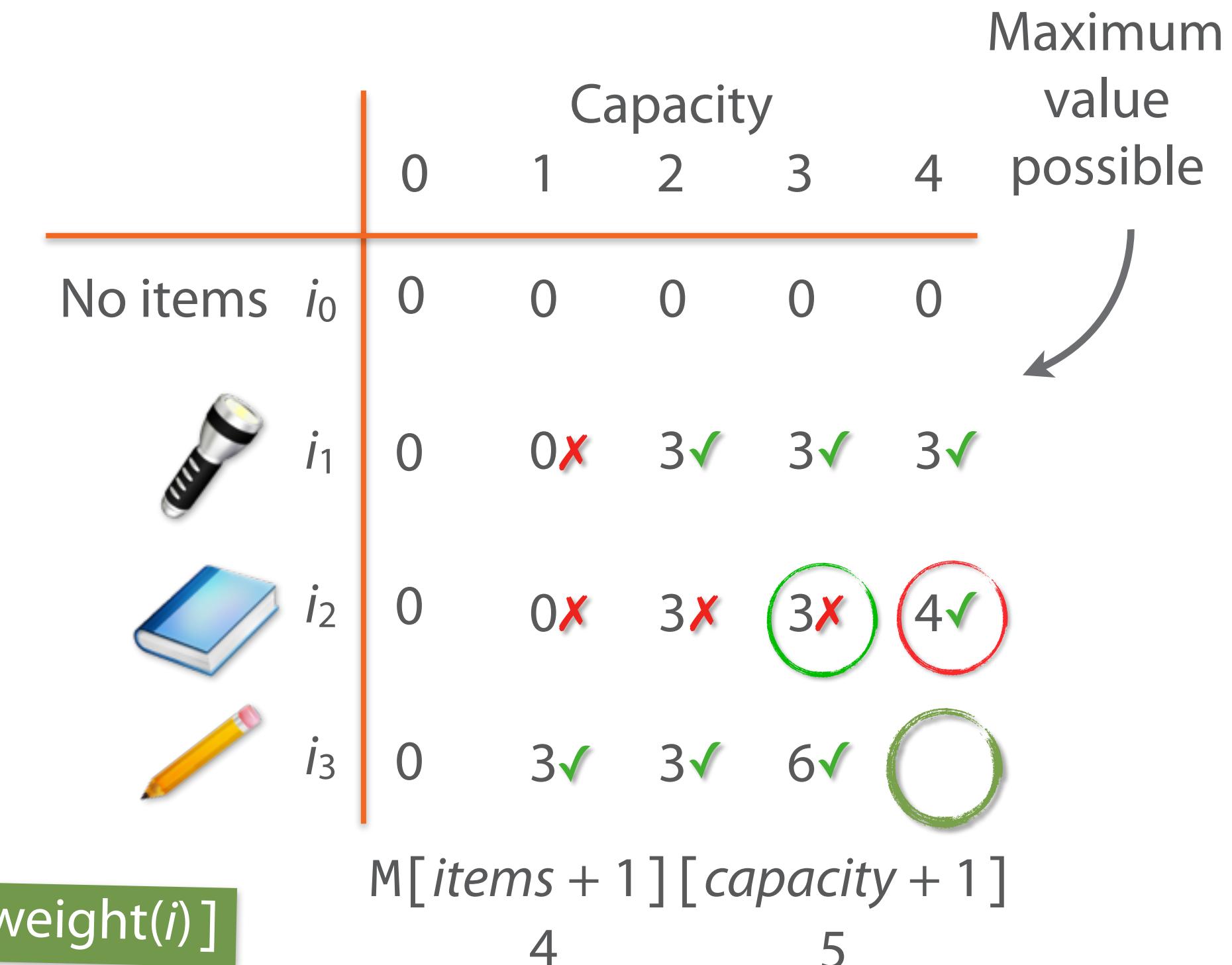
$$M[i][capacity] = \max(M[i-1][capacity], value(i) + M[i-1][capacity - weight(i)])$$

if excluded:

$$M[i-1][capacity]$$

if included and small enough:

$$value(i) + M[i-1][capacity - weight(i)]$$



# The 0/1 Knapsack Problem

Knapsack capacity: 4		
		
Weight: 2	Weight: 2	Weight: 1
Value: 3	Value: 1	Value: 3

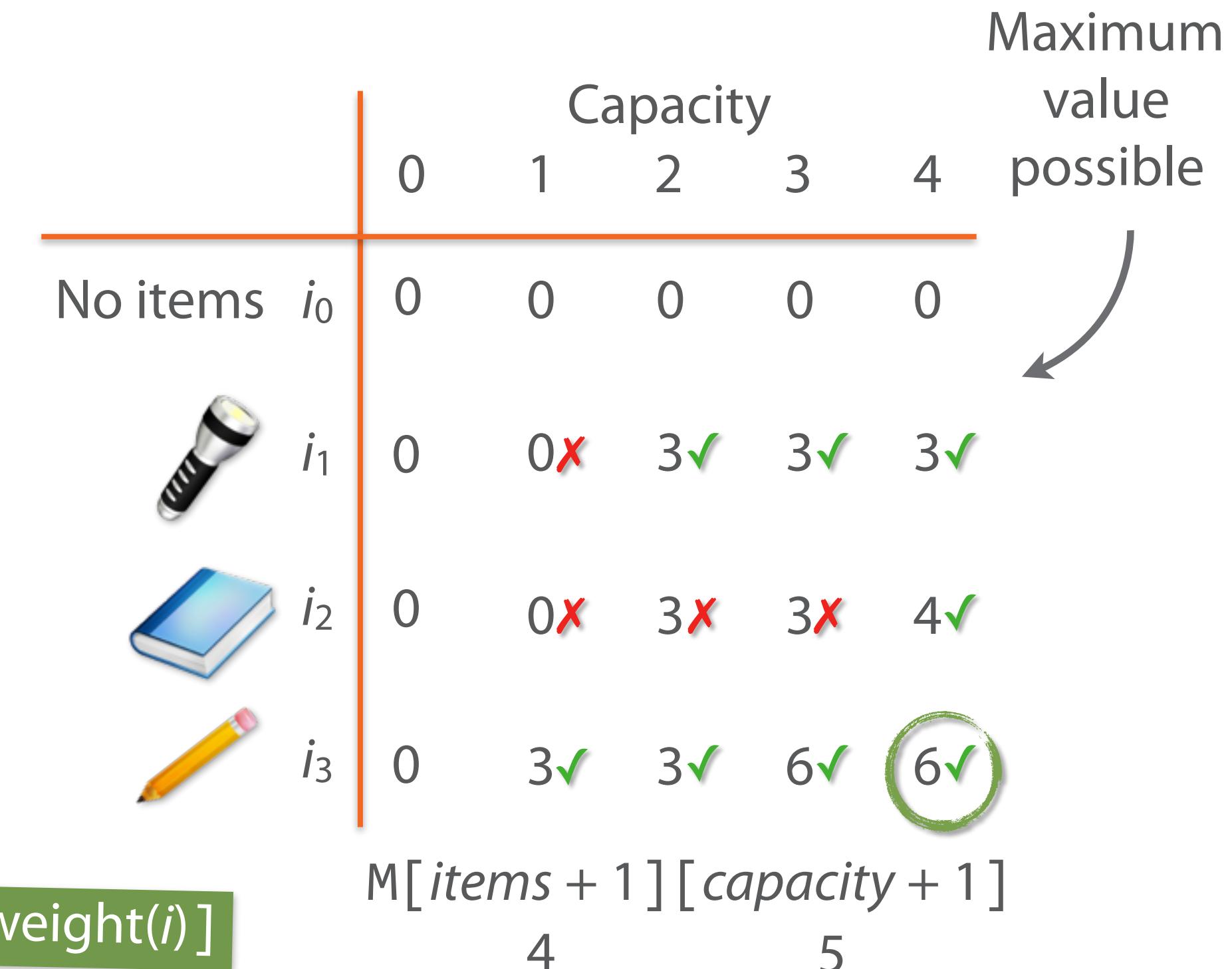
$$M[i][capacity] = \max(M[i-1][capacity], value(i) + M[i-1][capacity - weight(i)])$$

if excluded:

$$M[i-1][capacity]$$

if included and small enough:

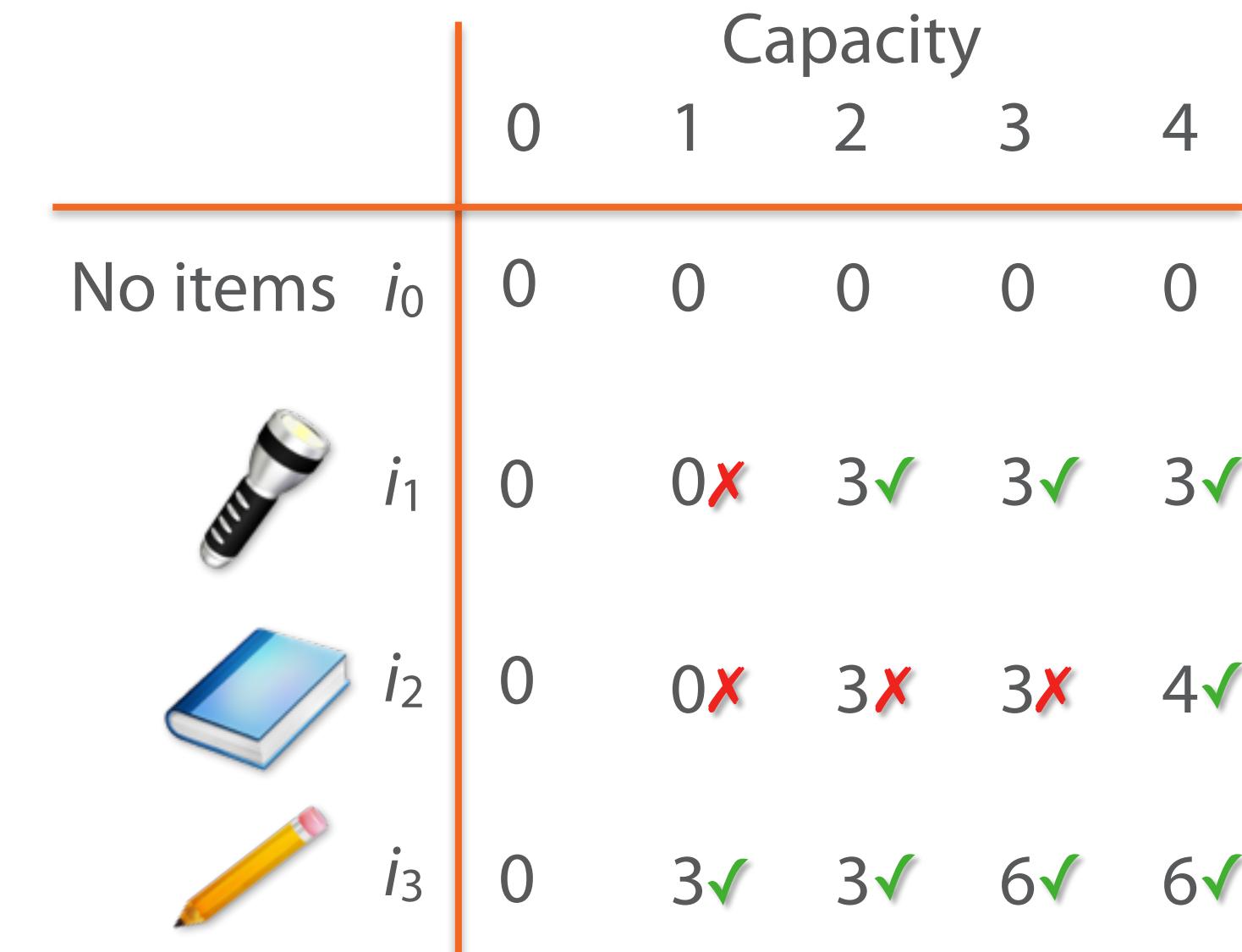
$$value(i) + M[i-1][capacity - weight(i)]$$



# The 0/1 Knapsack Problem

Knapsack capacity: 4

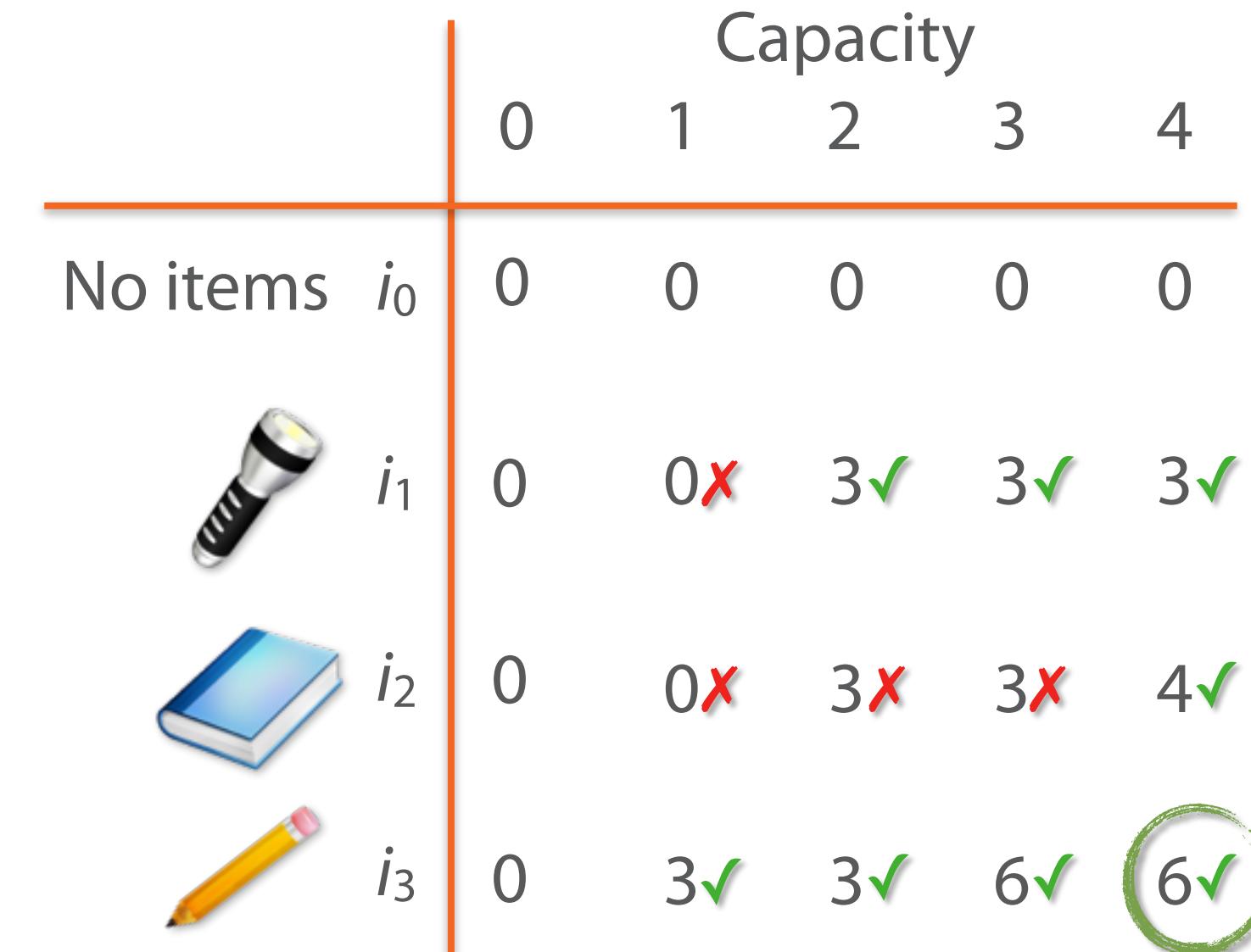
		
Weight: 2	Weight: 2	Weight: 1
Value: 3	Value: 1	Value: 3



# The 0/1 Knapsack Problem

Knapsack capacity: 4

		
Weight: 2	Weight: 2	Weight: 1
Value: 3	Value: 1	Value: 3



# The 0/1 Knapsack Problem

Knapsack capacity: 4



Weight: 2  
Value: 3

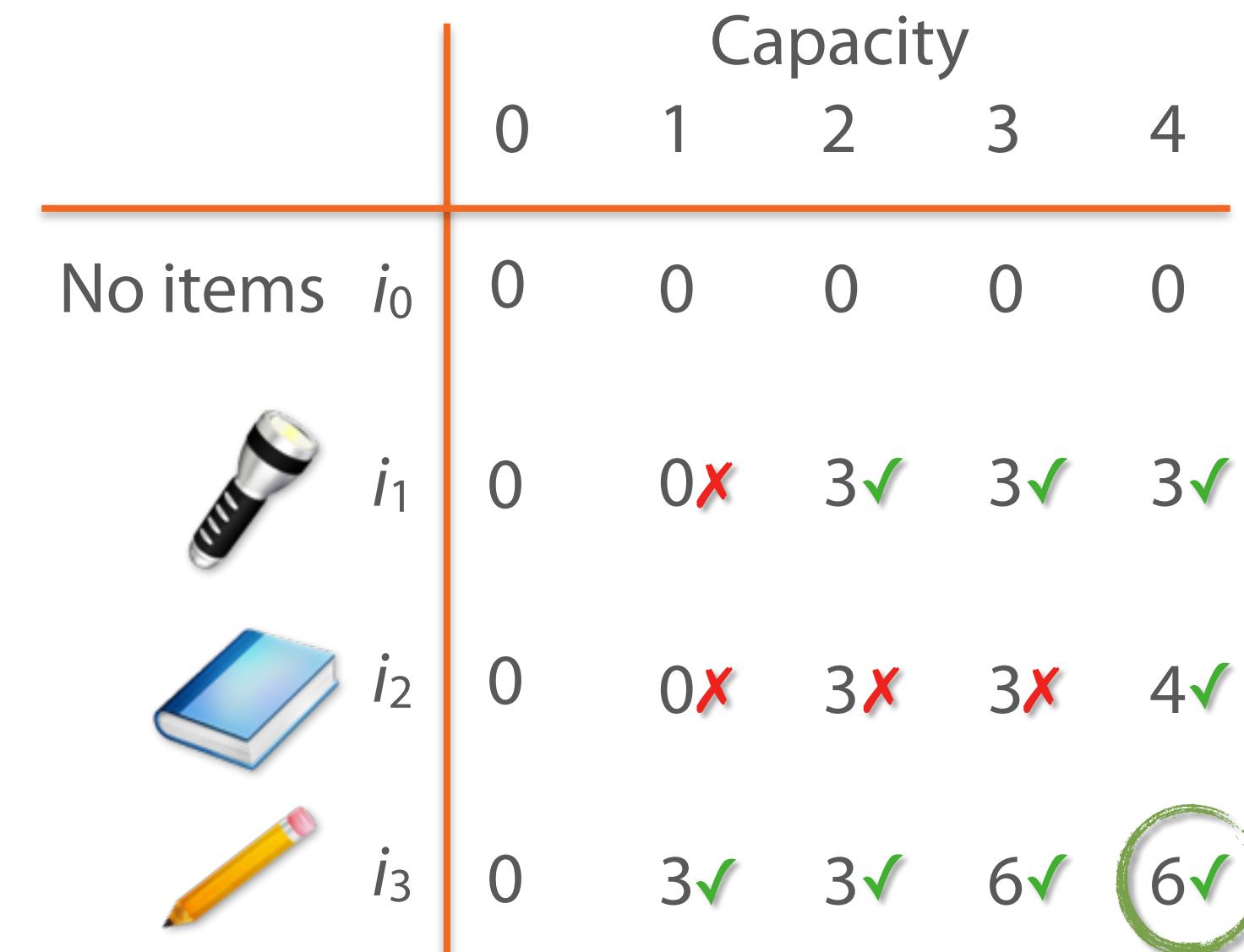


Weight: 2  
Value: 1



Weight: 1  
Value: 3

Maximum value: 6



# The 0/1 Knapsack Problem

Knapsack capacity: 4



Weight: 2  
Value: 3

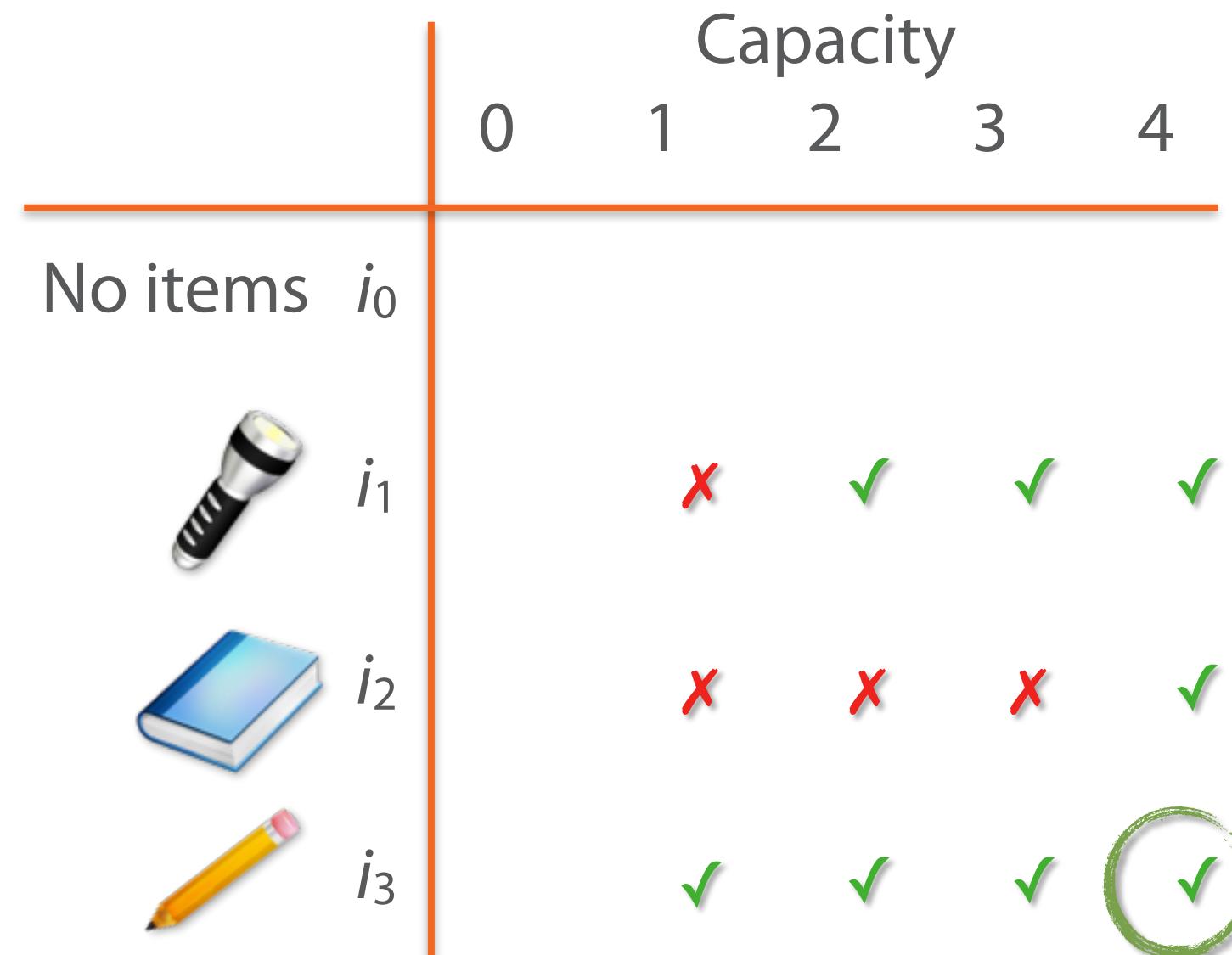


Weight: 2  
Value: 1



Weight: 1  
Value: 3

Maximum value: 6



# The 0/1 Knapsack Problem

Knapsack capacity: 4



Weight: 2  
Value: 3

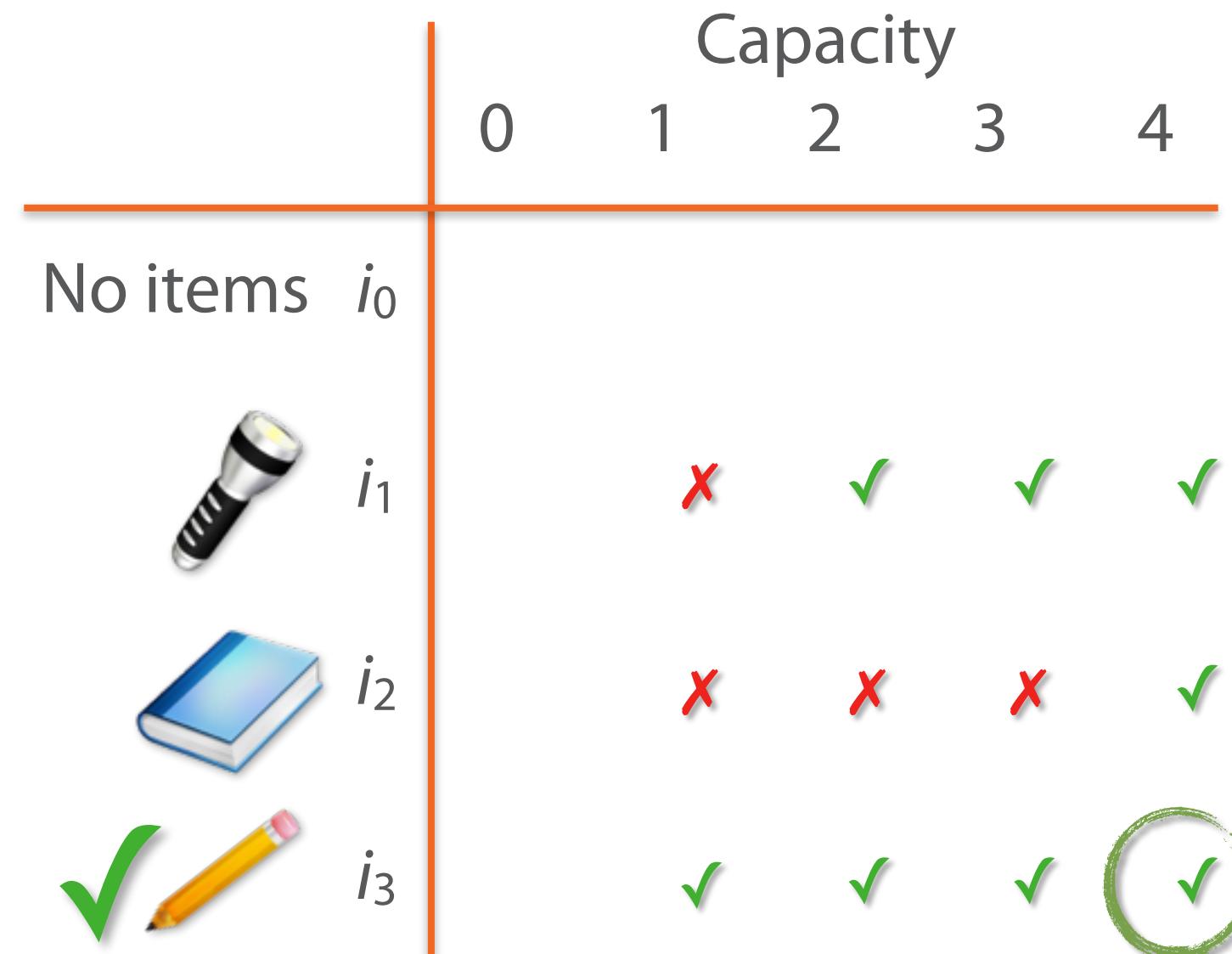


Weight: 2  
Value: 1



Weight: 1  
Value: 3

Maximum value: 6



# The 0/1 Knapsack Problem

Knapsack capacity: 4



Weight: 2  
Value: 3

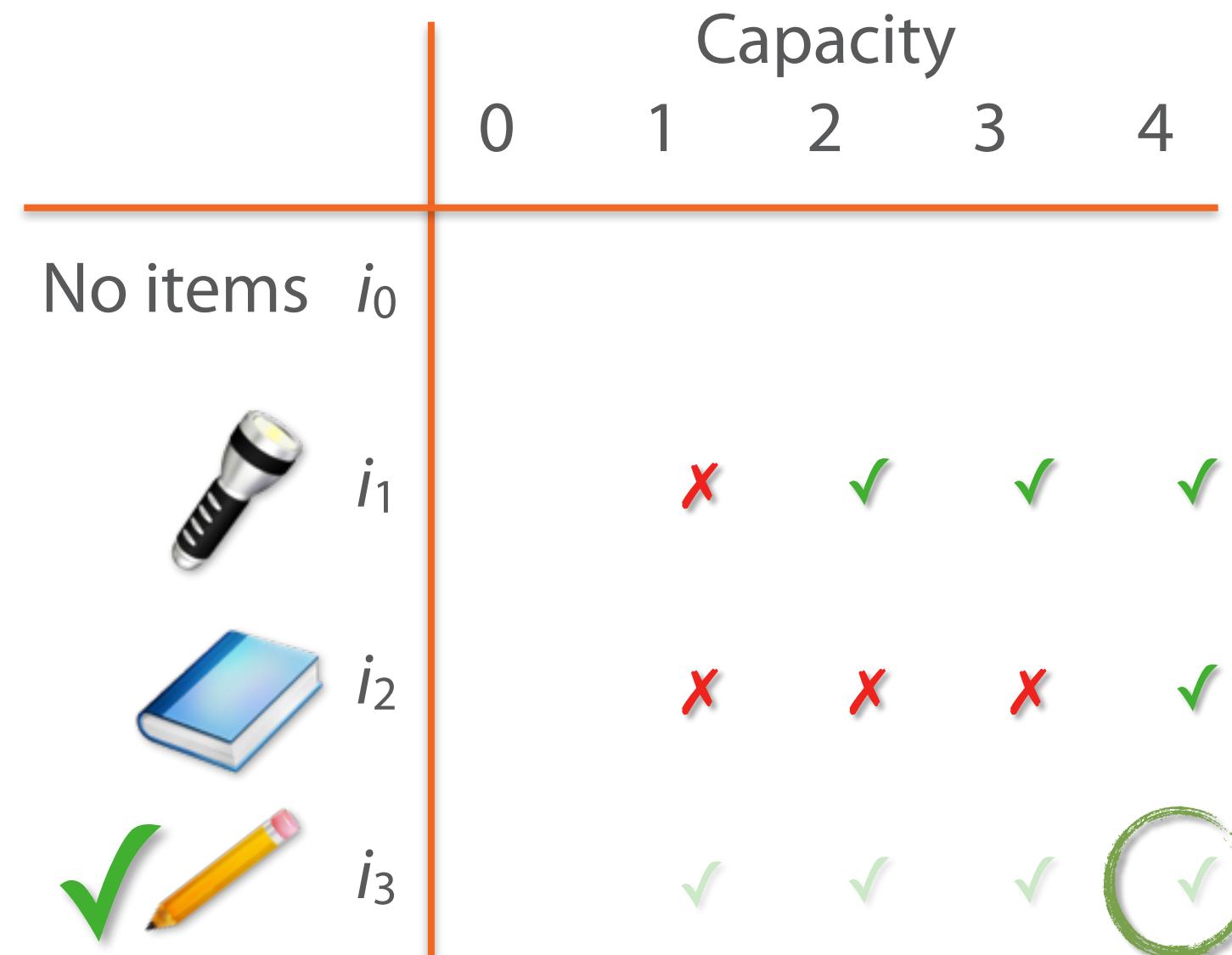


Weight: 2  
Value: 1



Weight: 1  
Value: 3

Maximum value: 6



# The 0/1 Knapsack Problem

Knapsack capacity: 4



Weight: 2  
Value: 3

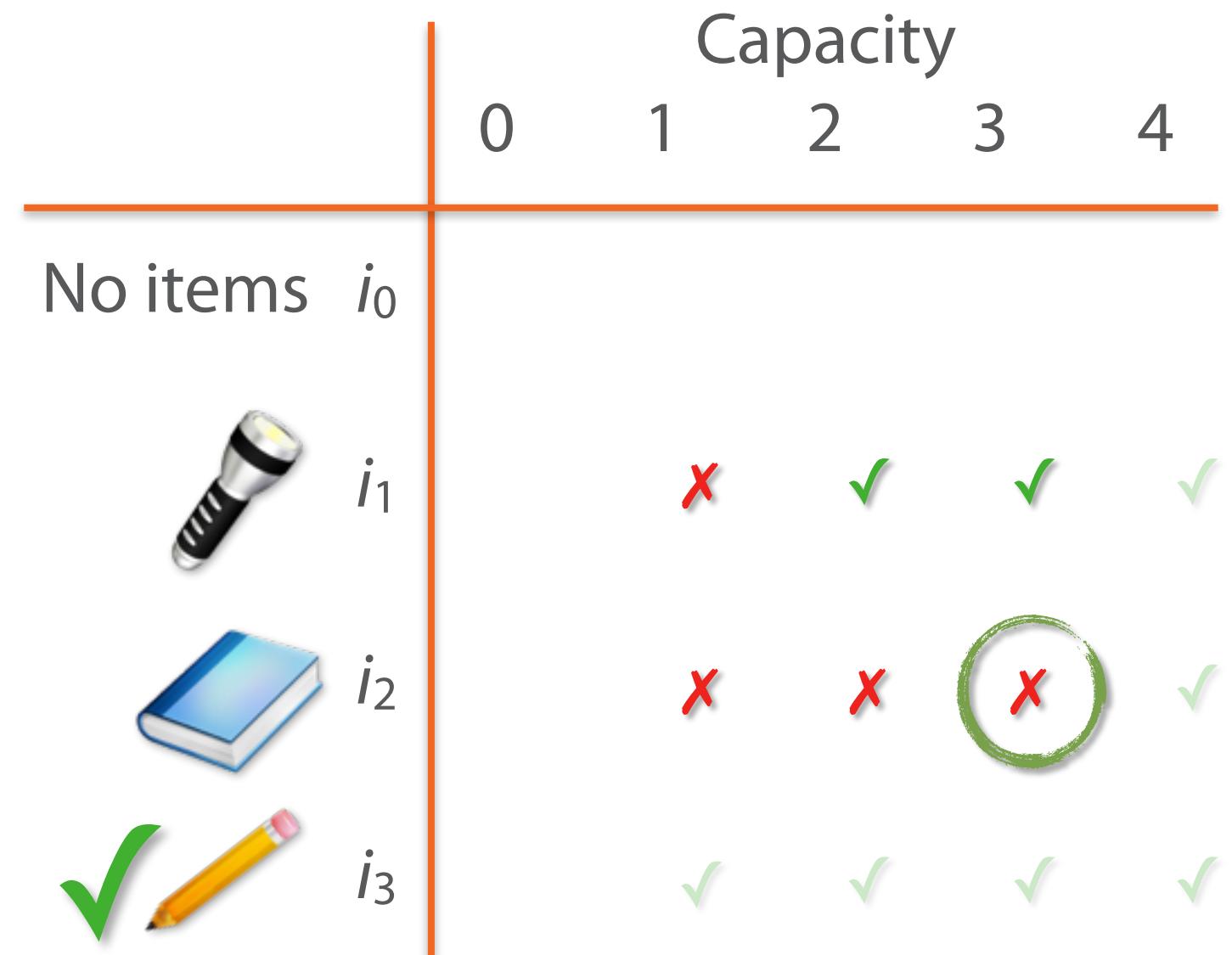


Weight: 2  
Value: 1



Weight: 1  
Value: 3

Maximum value: 6



# The 0/1 Knapsack Problem

Knapsack capacity: 4



Weight: 2  
Value: 3

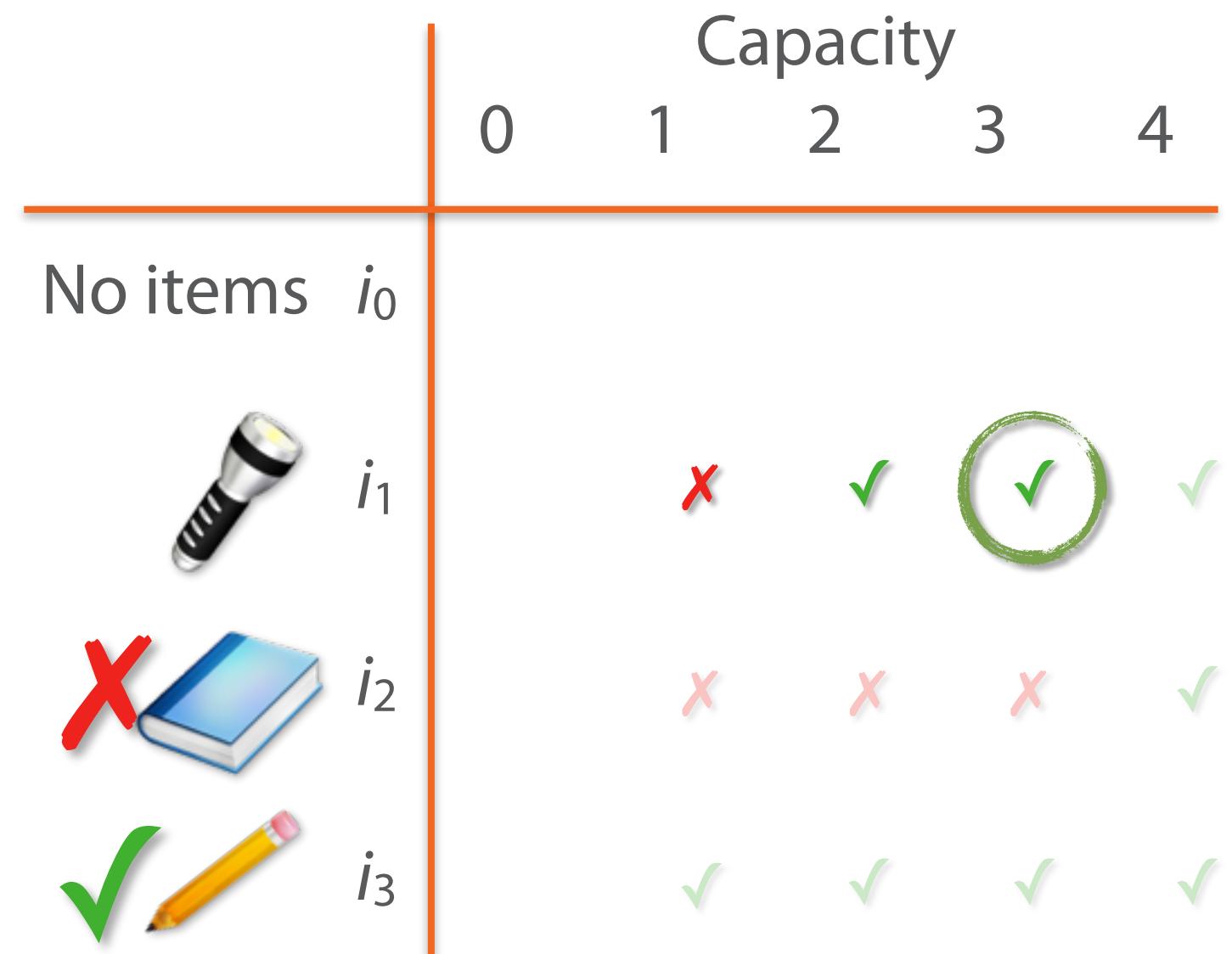


Weight: 2  
Value: 1



Weight: 1  
Value: 3

Maximum value: 6



# The 0/1 Knapsack Problem

Knapsack capacity: 4



Weight: 2  
Value: 3

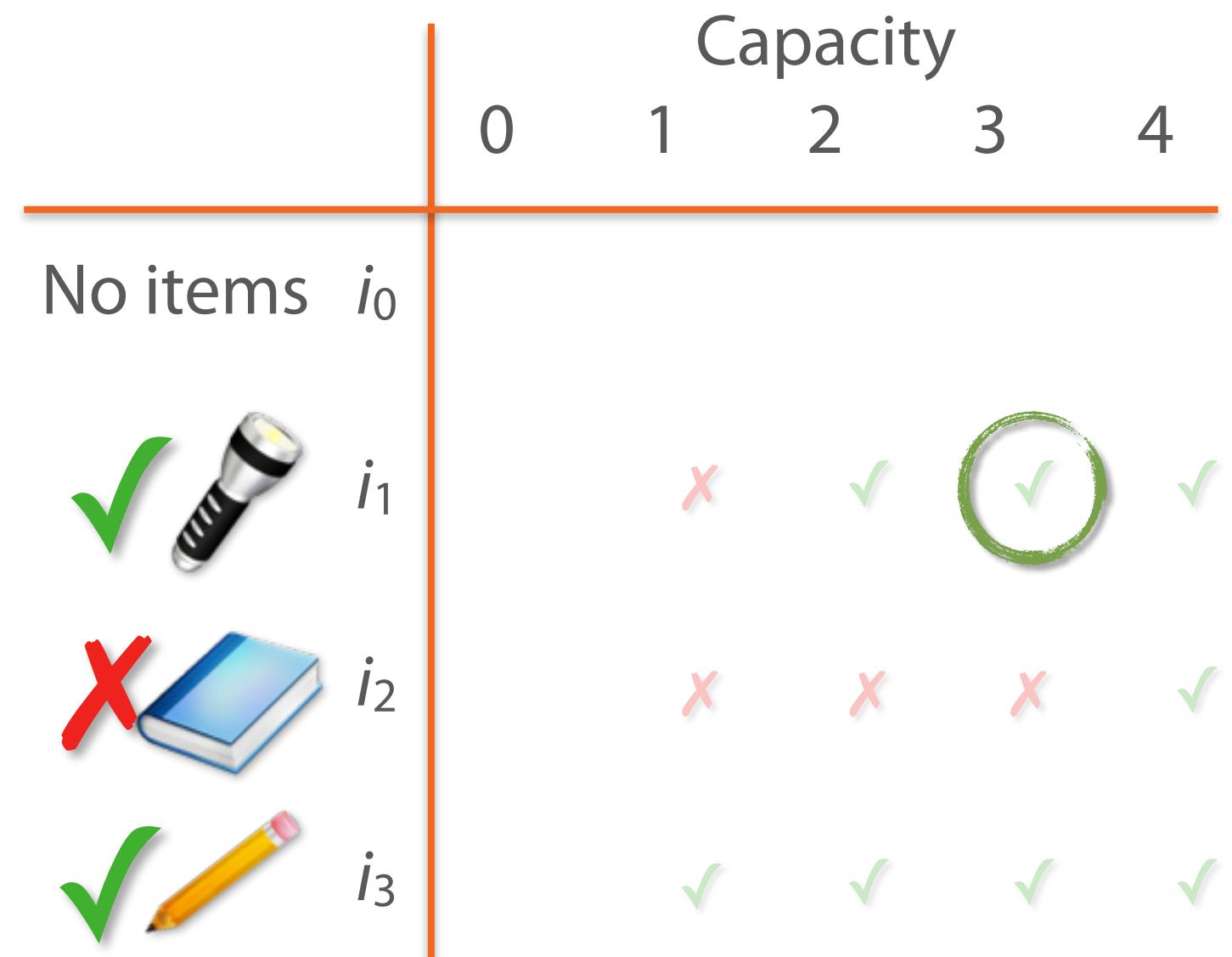


Weight: 2  
Value: 1



Weight: 1  
Value: 3

Maximum value: 6



# The 0/1 Knapsack Problem

Knapsack capacity: 4



Weight: 2  
Value: 3



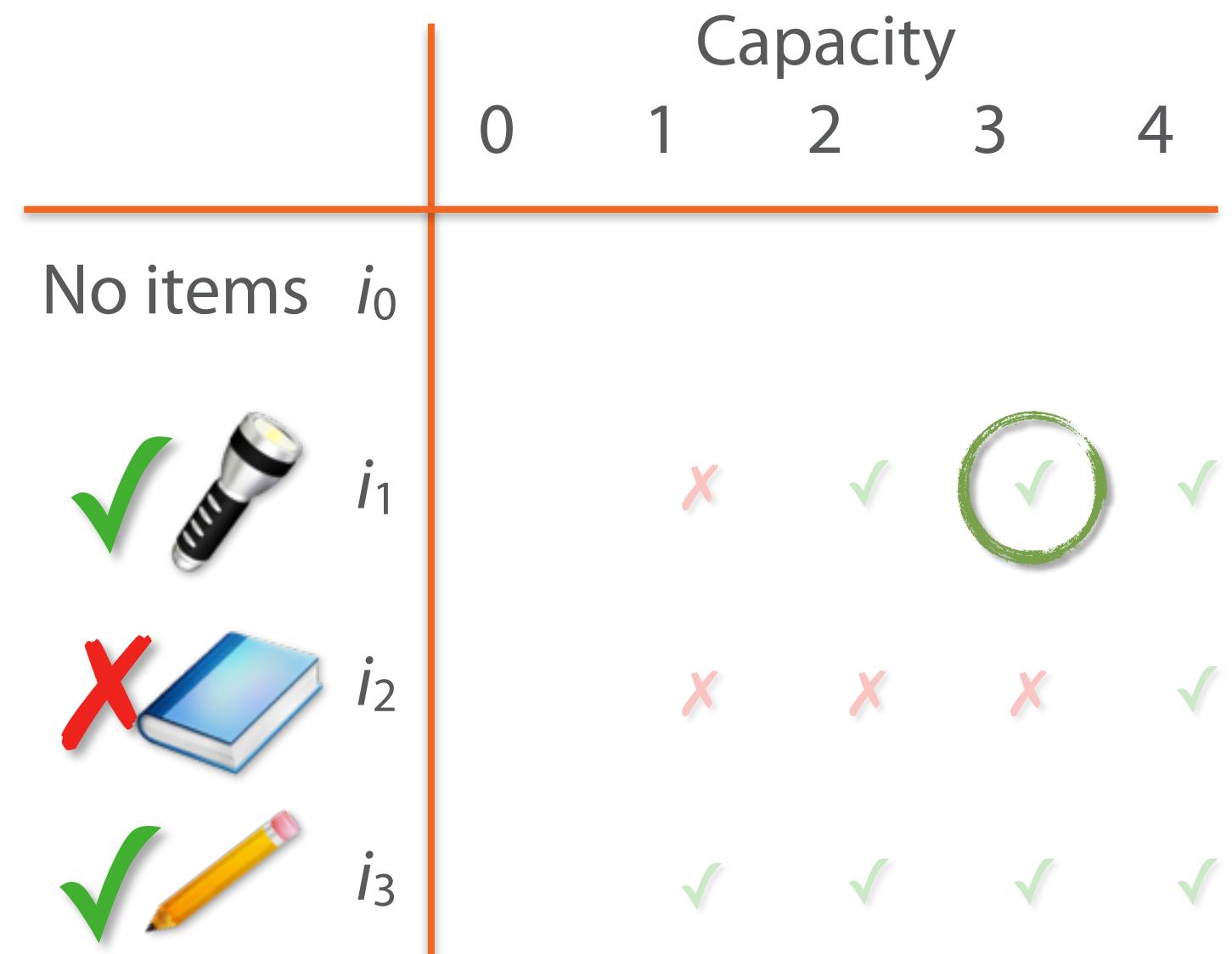
Weight: 2  
Value: 1



Weight: 1  
Value: 3

Maximum value: 6

For  $N$  items and capacity of  $C$ :



# The 0/1 Knapsack Problem

Knapsack capacity: 4



Weight: 2  
Value: 3



Weight: 2  
Value: 1

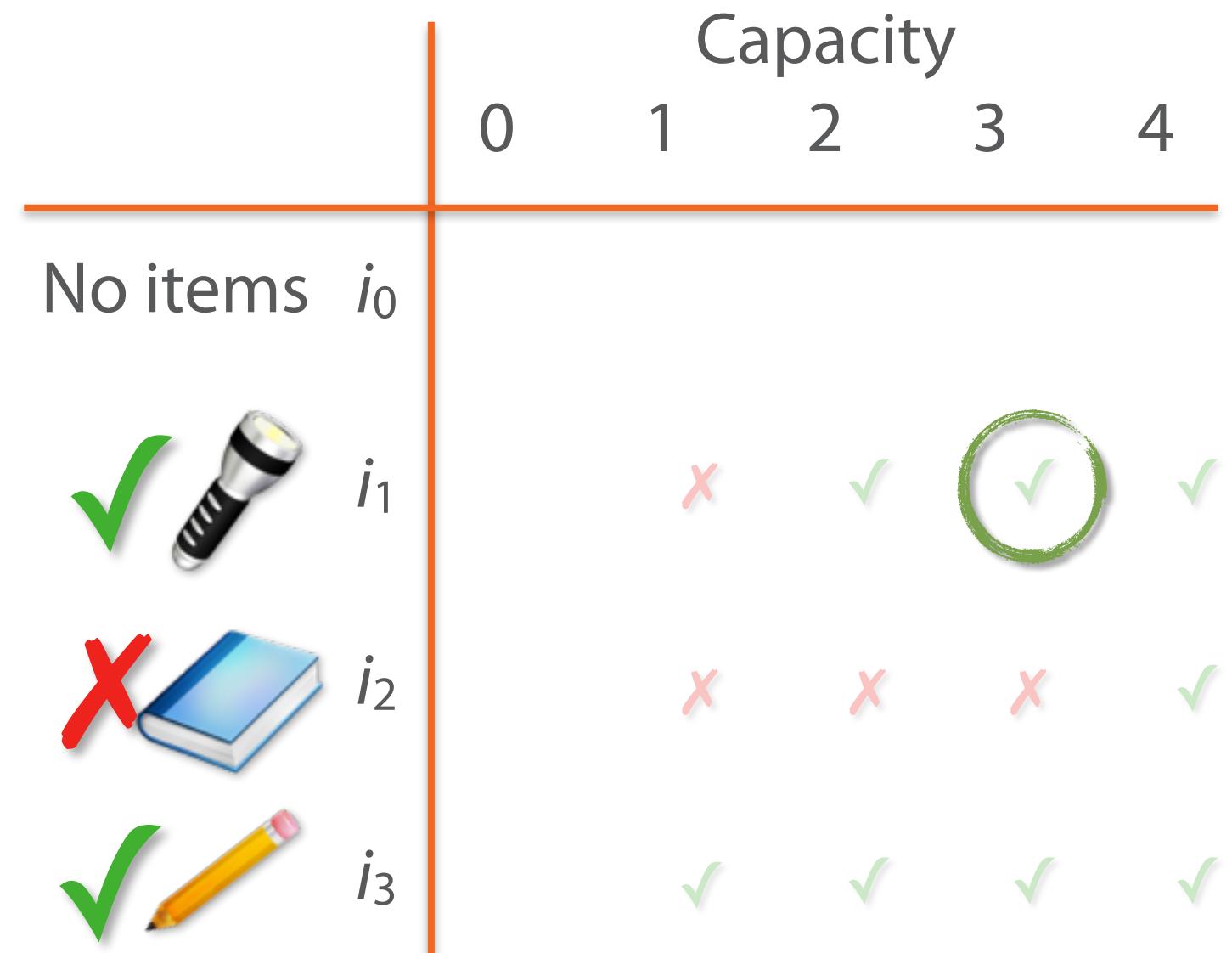


Weight: 1  
Value: 3

Maximum value: 6

For  $N$  items and capacity of  $C$ :

Brute force:  $\Theta(2^N)$



# The 0/1 Knapsack Problem

Knapsack capacity: 4



Weight: 2  
Value: 3



Weight: 2  
Value: 1



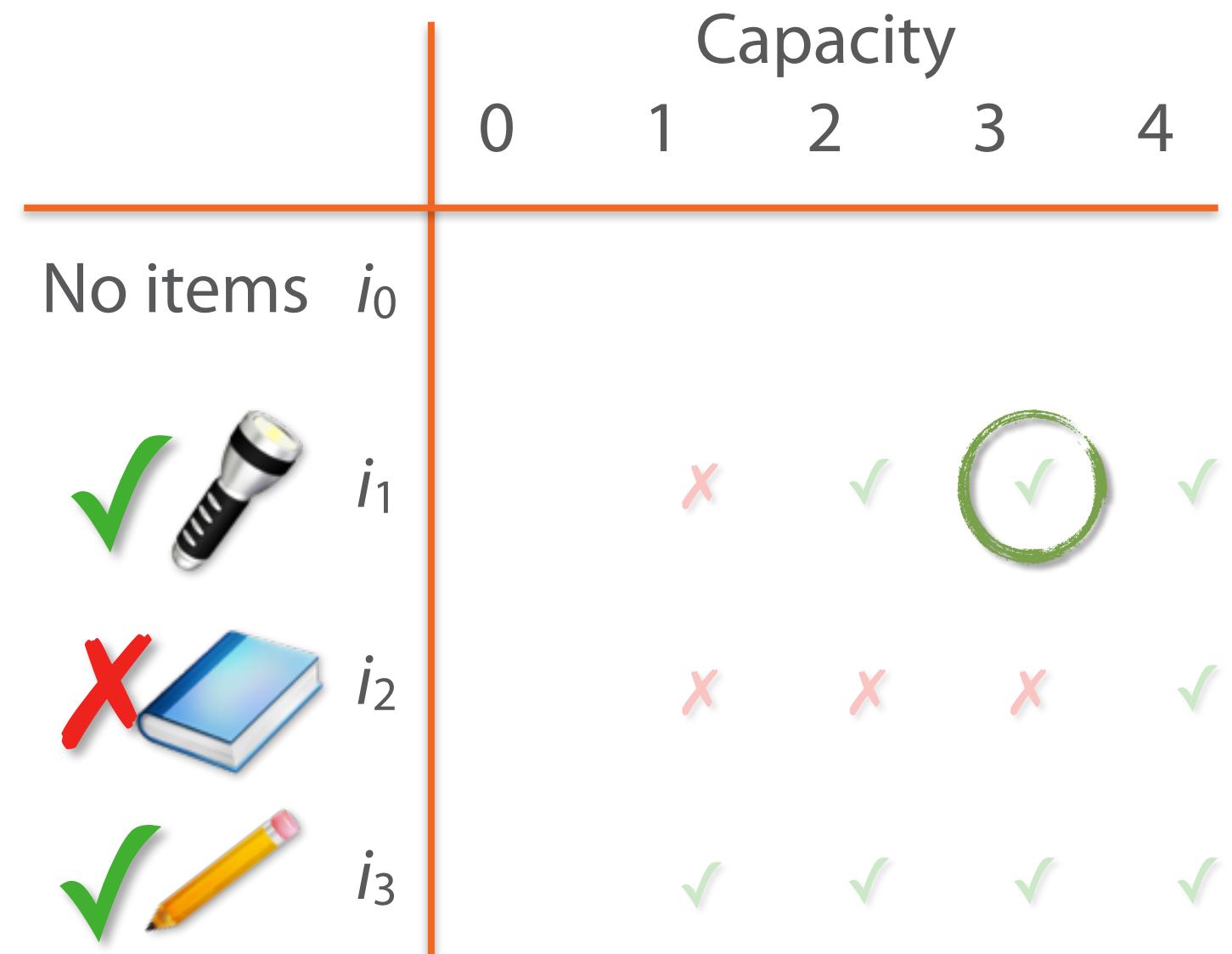
Weight: 1  
Value: 3

Maximum value: 6

For  $N$  items and capacity of  $C$ :

Brute force:  $\Theta(2^N)$

Dynamic programming:  $\Theta(N \cdot C)$



# Other Examples

# Other Examples

## Text justification

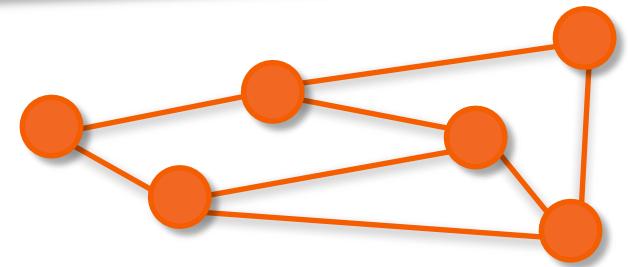
Lorem ipsum dolor sit amet,  
consectetur adipiscing elit, sed do  
eiusmod tempor incididunt ut  
labore et dolore magna aliqua.

# Other Examples

## Text justification

Lorem ipsum dolor sit amet,  
consectetur adipiscing elit, sed do  
eiusmod tempor incididunt ut  
labore et dolore magna aliqua.

## Routing / shortest paths



# Other Examples

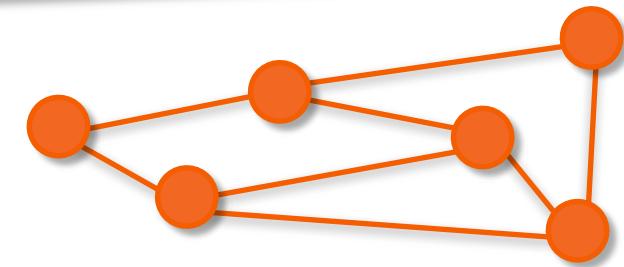
## Text justification

Lorem ipsum dolor sit amet,  
consectetur adipiscing elit, sed do  
eiusmod tempor incididunt ut  
labore et dolore magna aliqua.

diff foo.txt bar.txt  
(Longest common subsequence)

```
@@ -47,6 +47,7 @@
<Compile Include="Node.cs" />
<Compile Include="Program.cs" />
<Compile Include="Properties\AssemblyInfo.cs" />
+ <Compile Include="DivideAndConquer.cs" />
</ItemGroup>
<ItemGroup>
```

## Routing / shortest paths



# Other Examples

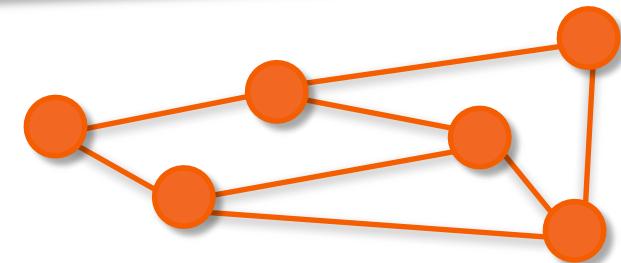
## Text justification

Lorem ipsum dolor sit amet,  
consectetur adipiscing elit, sed do  
eiusmod tempor incididunt ut  
labore et dolore magna aliqua.

diff foo.txt bar.txt  
(Longest common subsequence)

```
@@ -47,6 +47,7 @@
<Compile Include="Node.cs" />
<Compile Include="Program.cs" />
<Compile Include="Properties\AssemblyInfo.cs" />
+ <Compile Include="DivideAndConquer.cs" />
</ItemGroup>
<ItemGroup>
```

## Routing / shortest paths



## Sequence alignment

# Other Examples

## Text justification

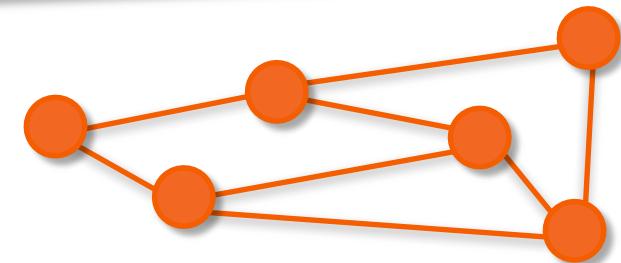
Lorem ipsum dolor sit amet,  
consectetur adipiscing elit, sed do  
eiusmod tempor incididunt ut  
labore et dolore magna aliqua.

diff foo.txt bar.txt  
(Longest common subsequence)

```
@@ -47,6 +47,7 @@
<Compile Include="Node.cs" />
<Compile Include="Program.cs" />
<Compile Include="Properties\AssemblyInfo.cs" />
+ <Compile Include="DivideAndConquer.cs" />
</ItemGroup>
<ItemGroup>
```

Various games  
(Tetris, Super Mario, Blackjack, etc.)

## Routing / shortest paths



Sequence alignment

# Other Examples

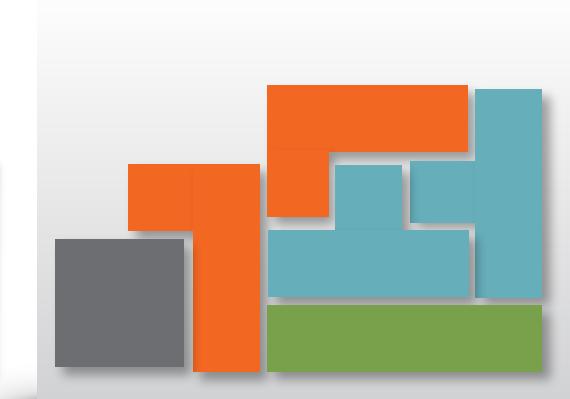
## Text justification

Lorem ipsum dolor sit amet,  
consectetur adipiscing elit, sed do  
eiusmod tempor incididunt ut  
labore et dolore magna aliqua.

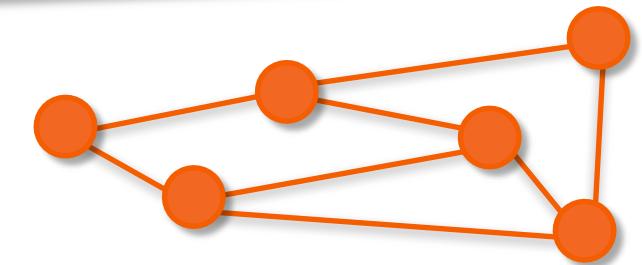
## diff foo.txt bar.txt (Longest common subsequence)

```
@@ -47,6 +47,7 @@
<Compile Include="Node.cs" />
<Compile Include="Program.cs" />
<Compile Include="Properties\AssemblyInfo.cs" />
+ <Compile Include="DivideAndConquer.cs" />
</ItemGroup>
<ItemGroup>
```

## Various games (Tetris, Super Mario, Blackjack, etc.)



## Routing / shortest paths



## Sequence alignment

# Algorithms

Graph traversal

Brute force  
Greedy algorithms

Divide  
and  
conquer

Dynamic  
programming

Branch  
and  
bound

# Algorithms

Graph traversal

Brute force  
Greedy algorithms

Divide  
and  
conquer

Dynam  
programm

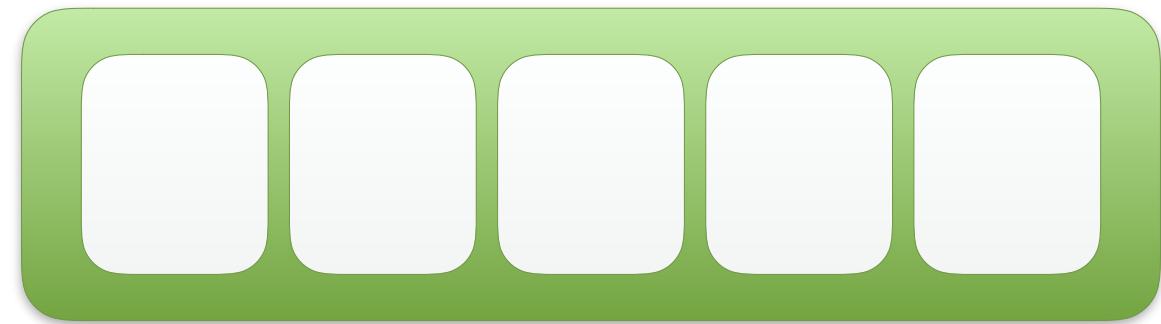
Branch  
and  
bound

# Main Principle

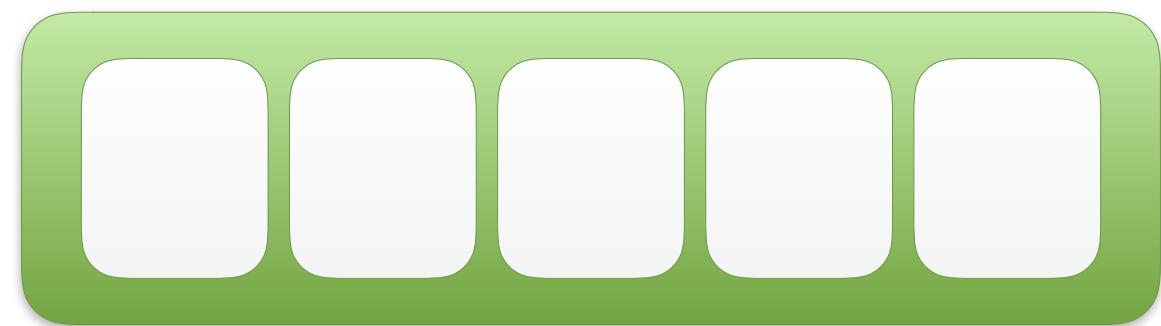
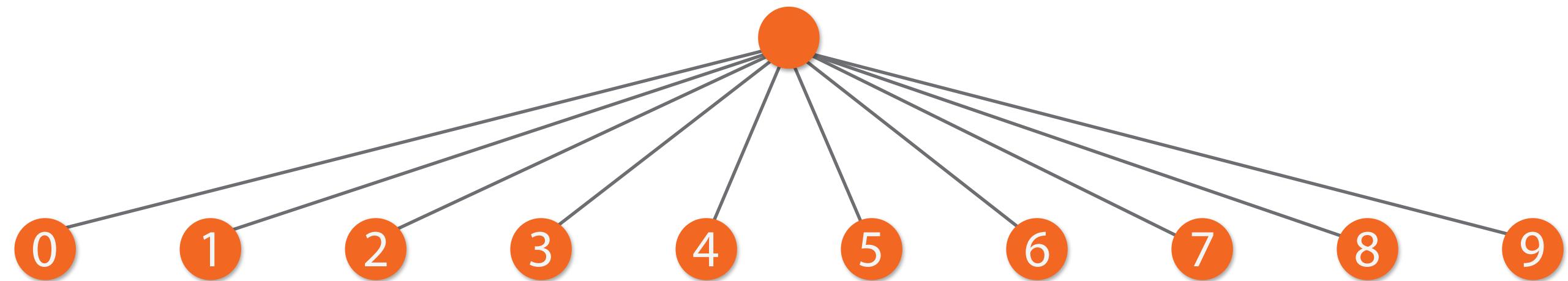
# Main Principle



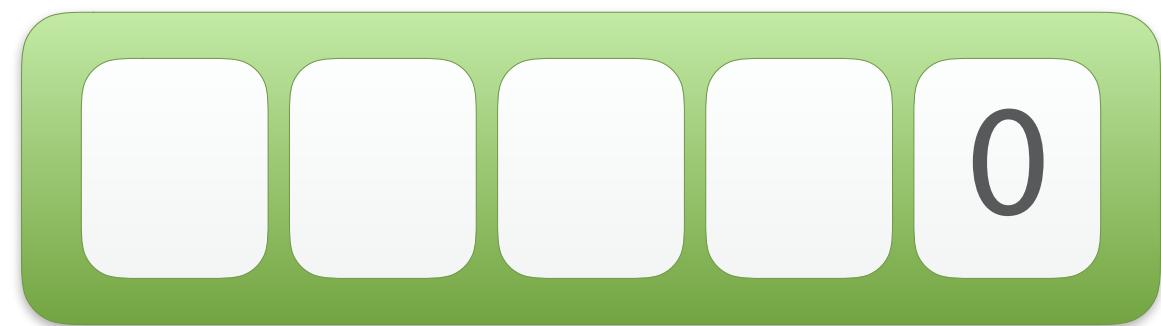
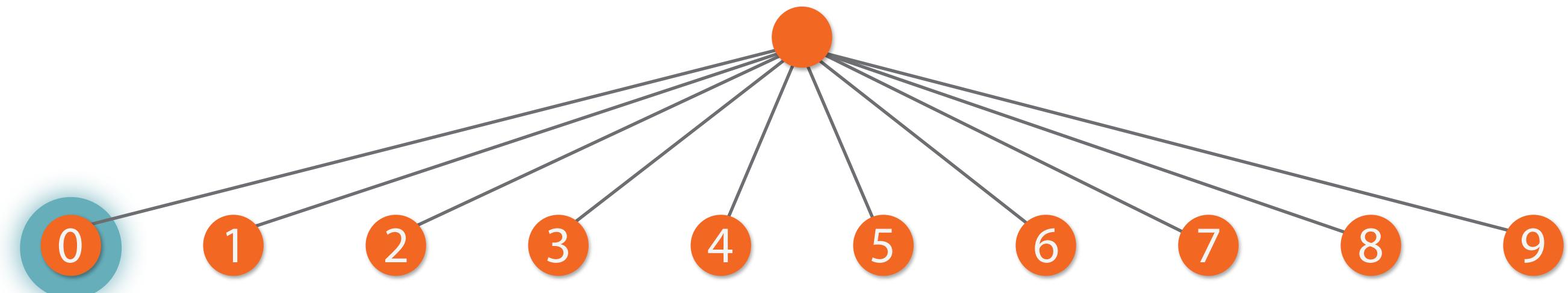
# Main Principle



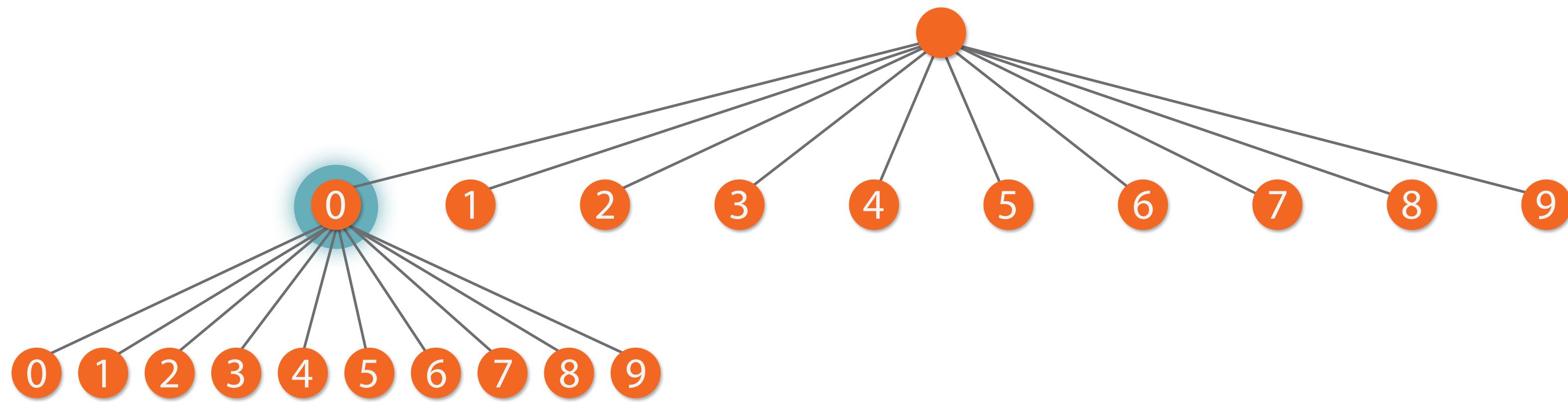
# Main Principle



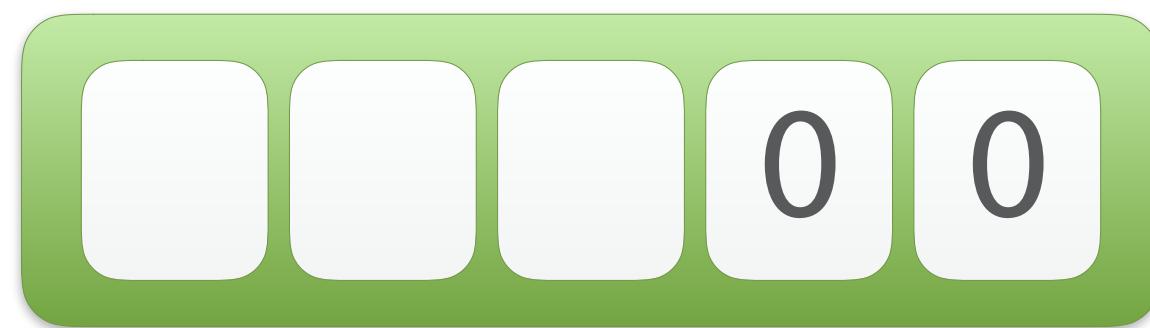
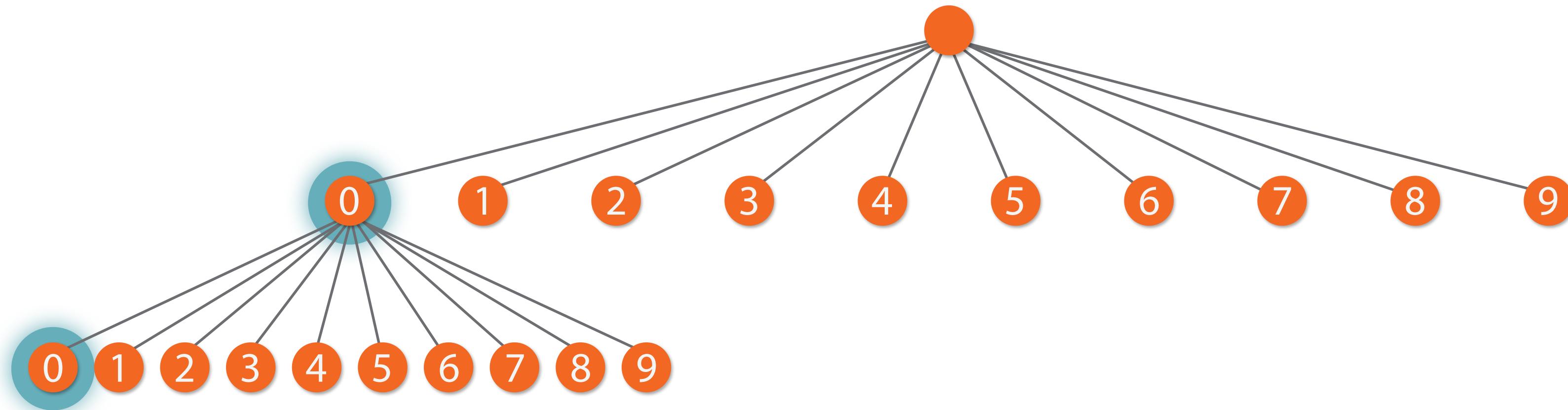
# Main Principle



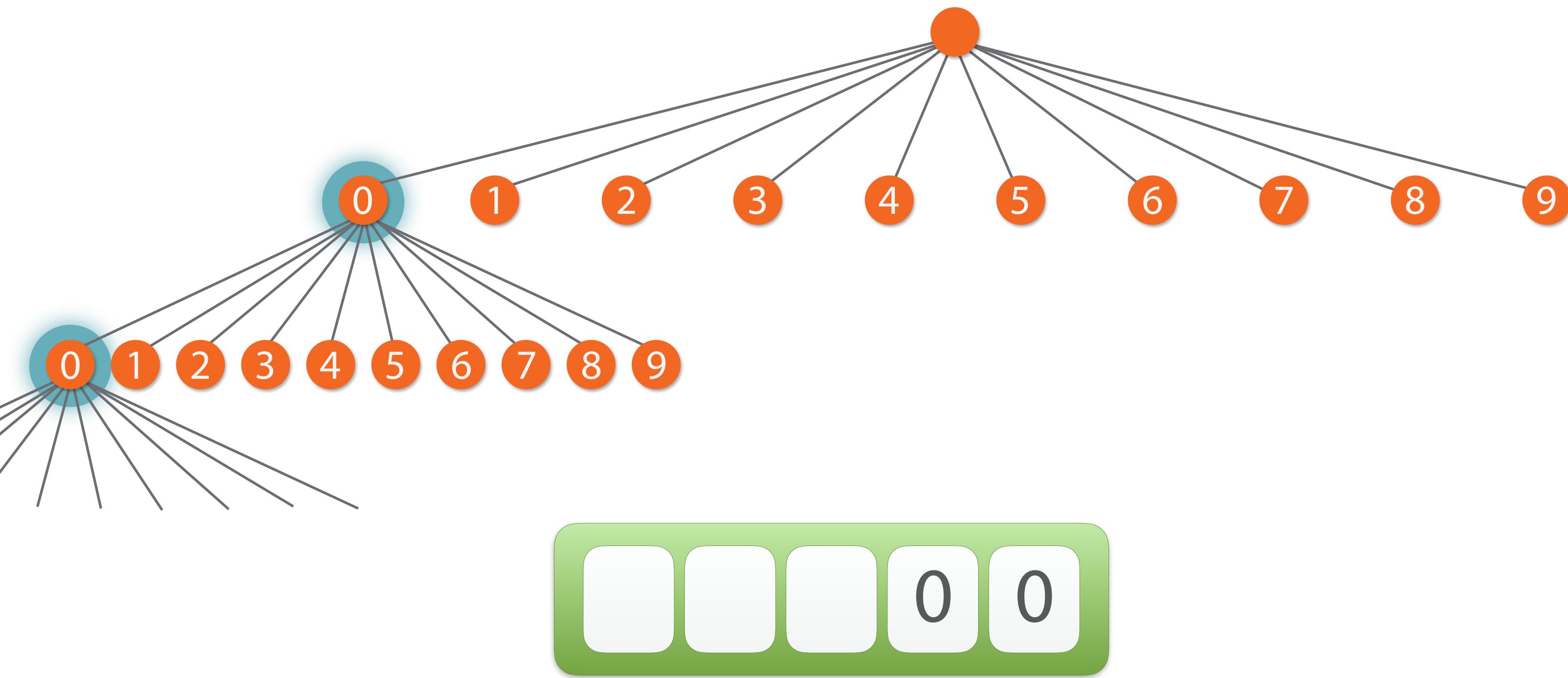
# Main Principle



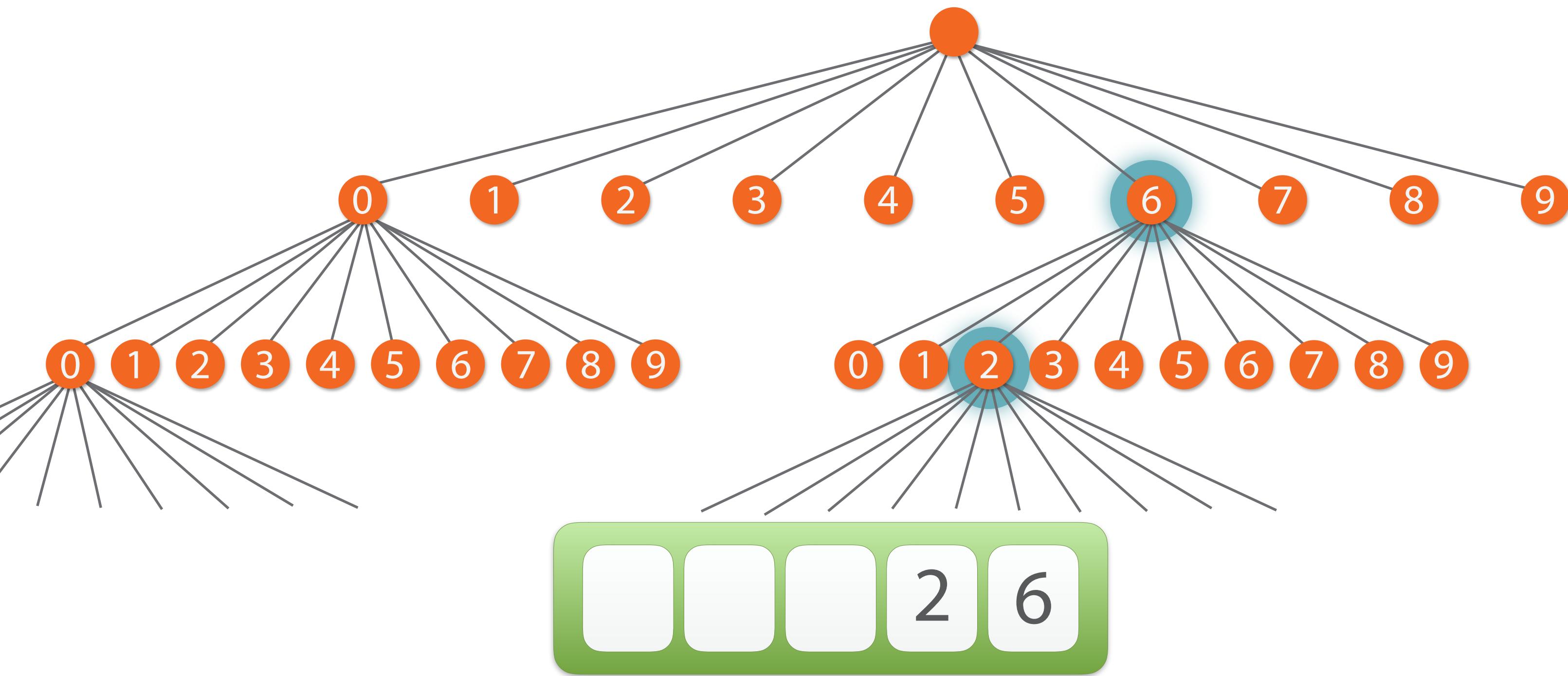
# Main Principle



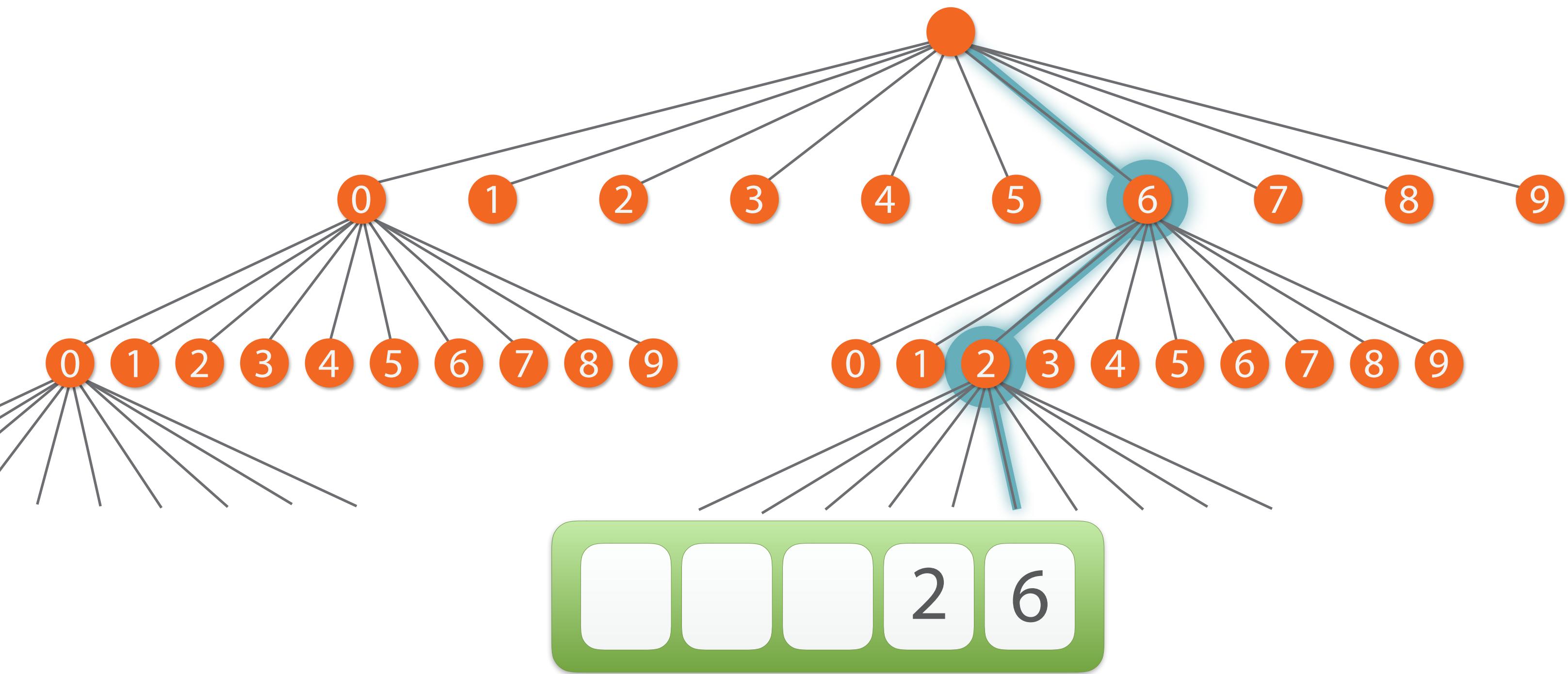
# Main Principle



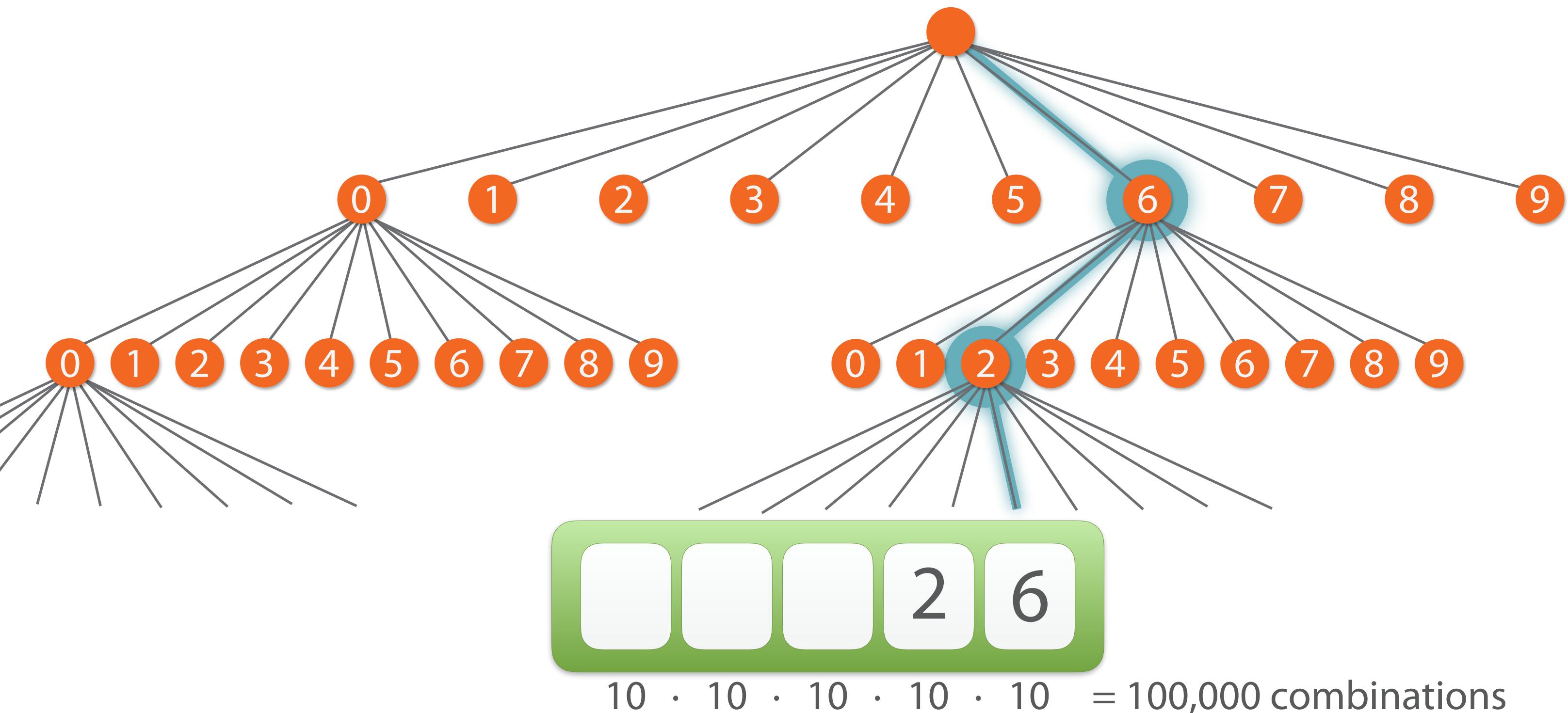
# Main Principle



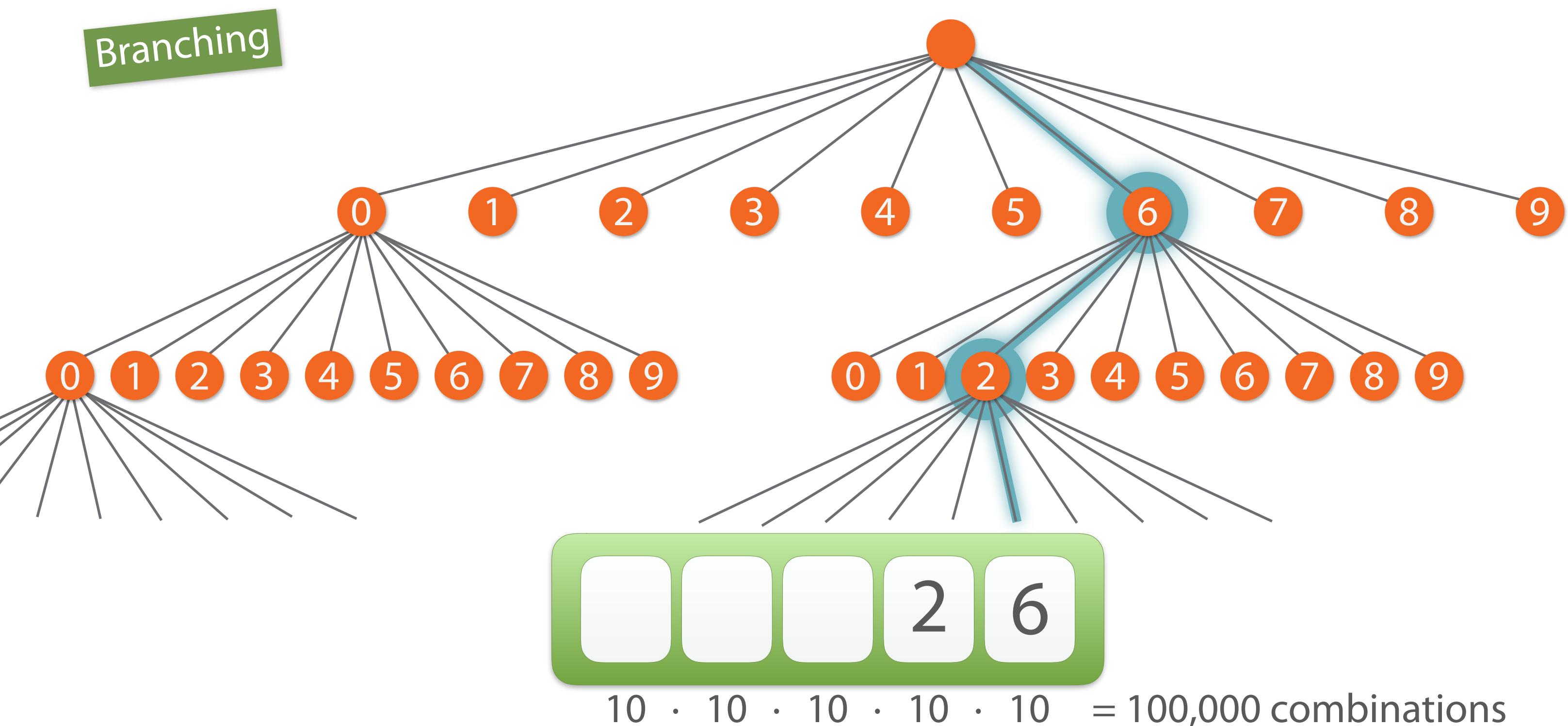
# Main Principle



# Main Principle

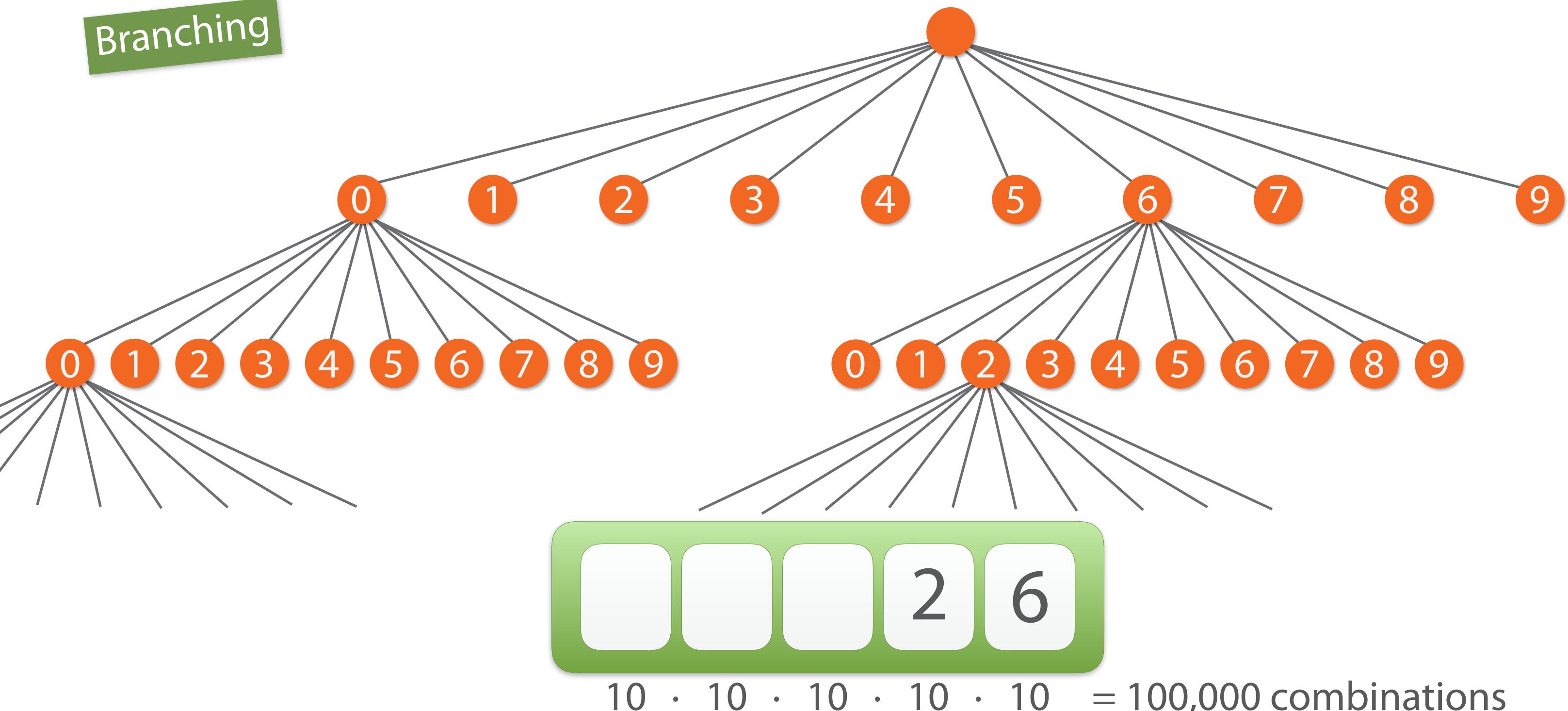


# Main Principle



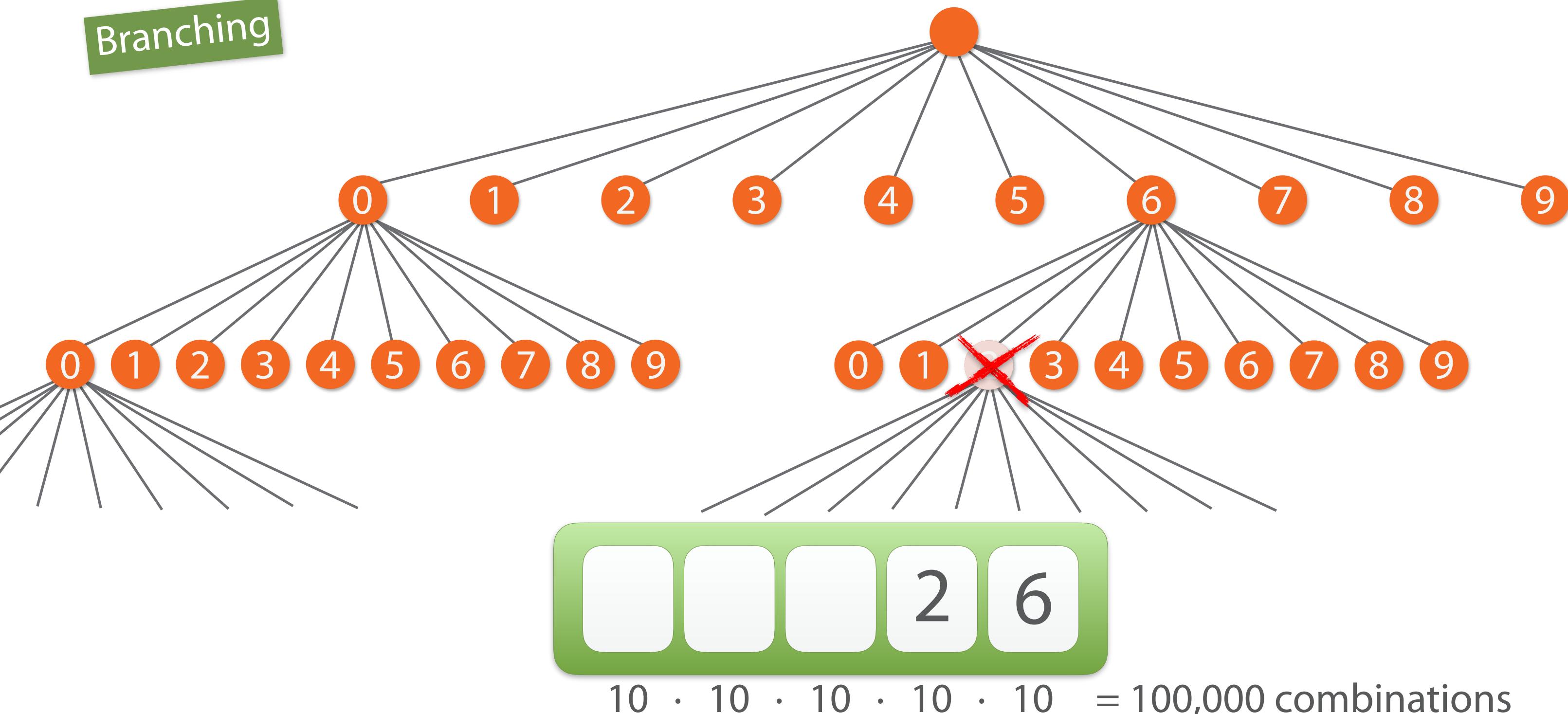
# Main Principle

# Branching

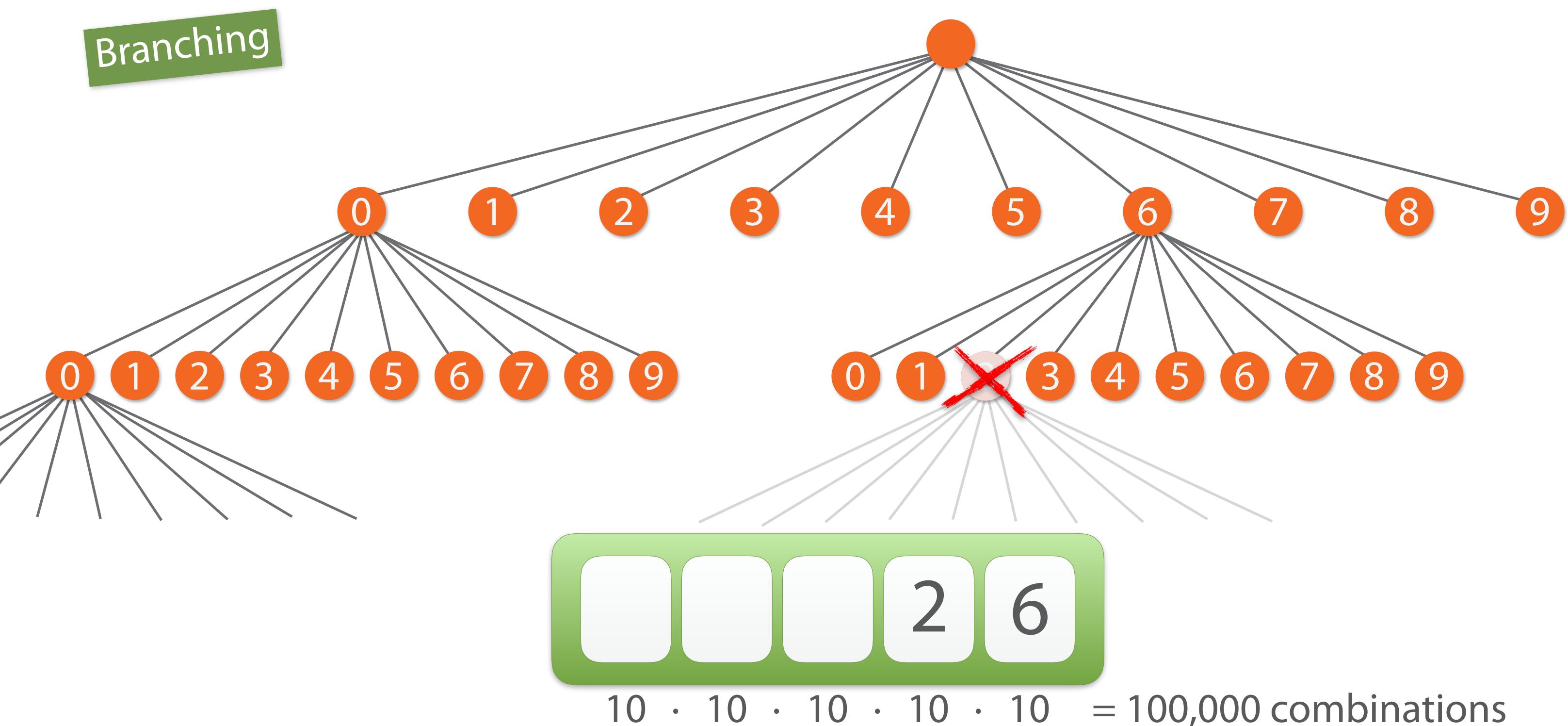


# Main Principle

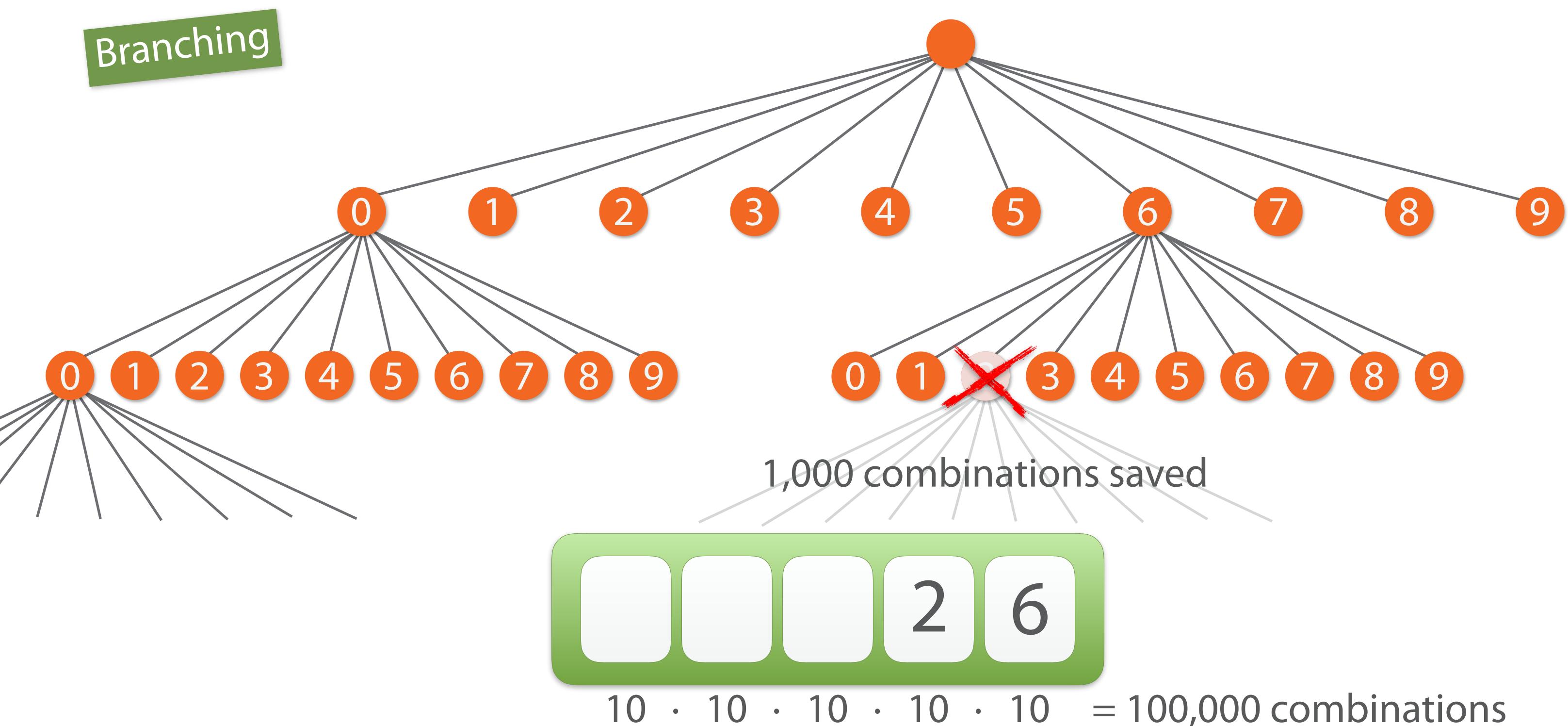
Branching



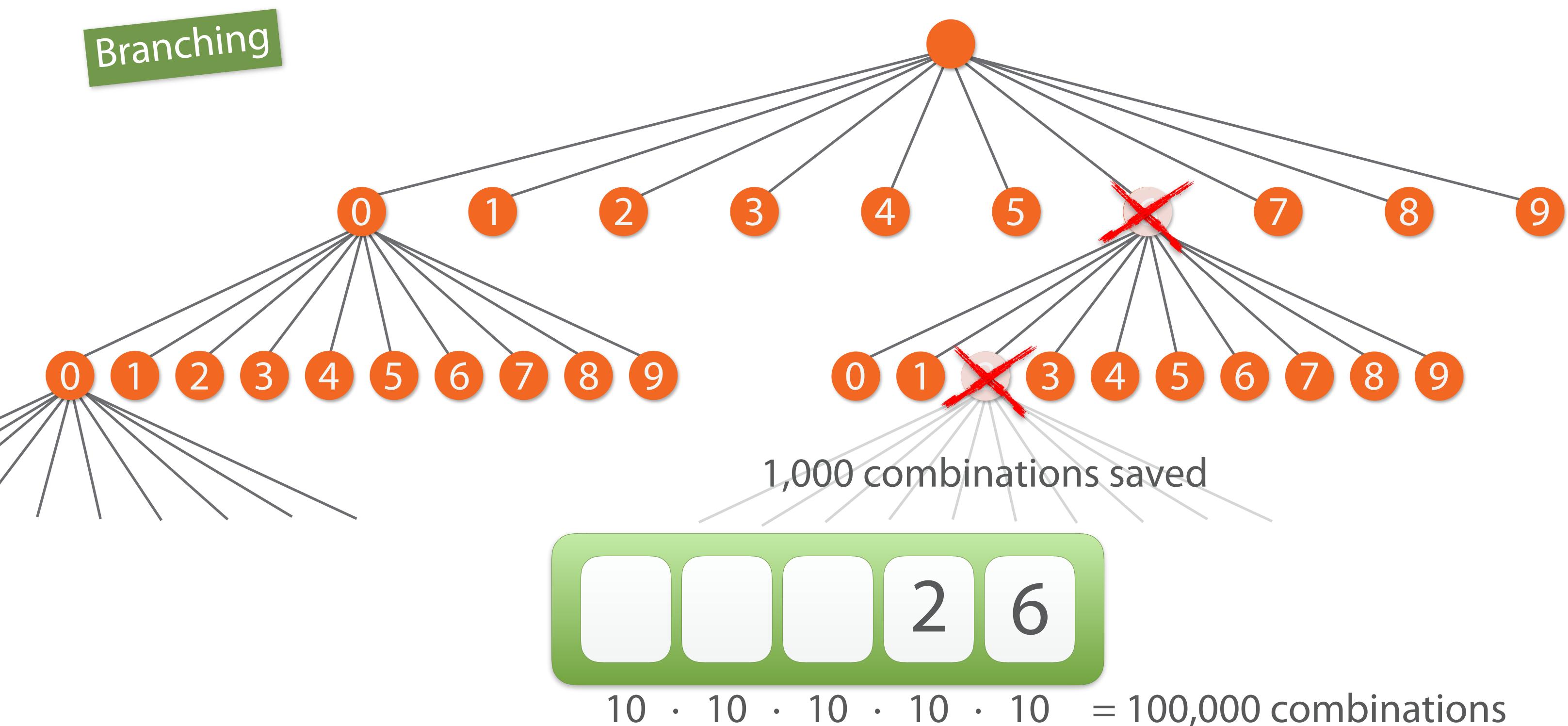
# Main Principle



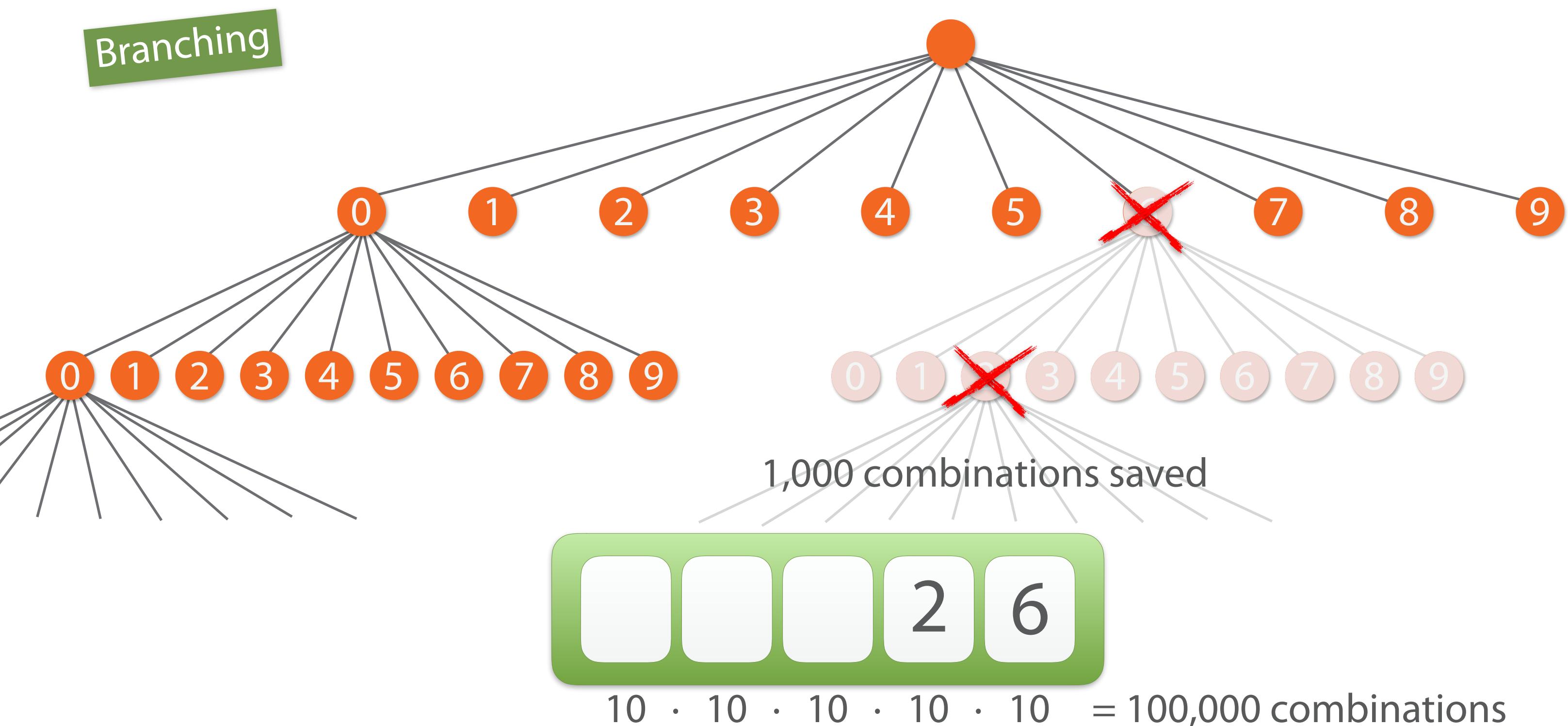
# Main Principle



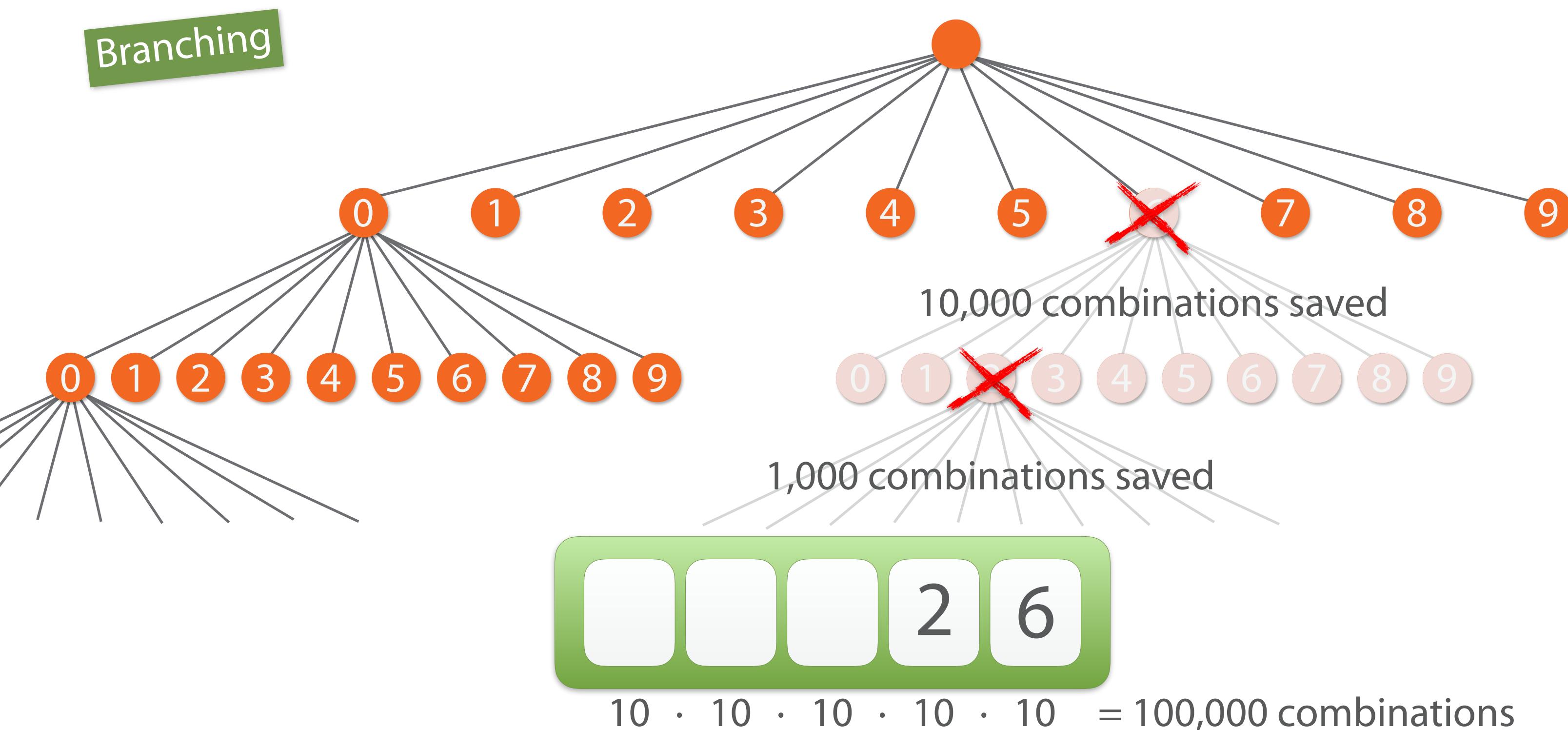
# Main Principle



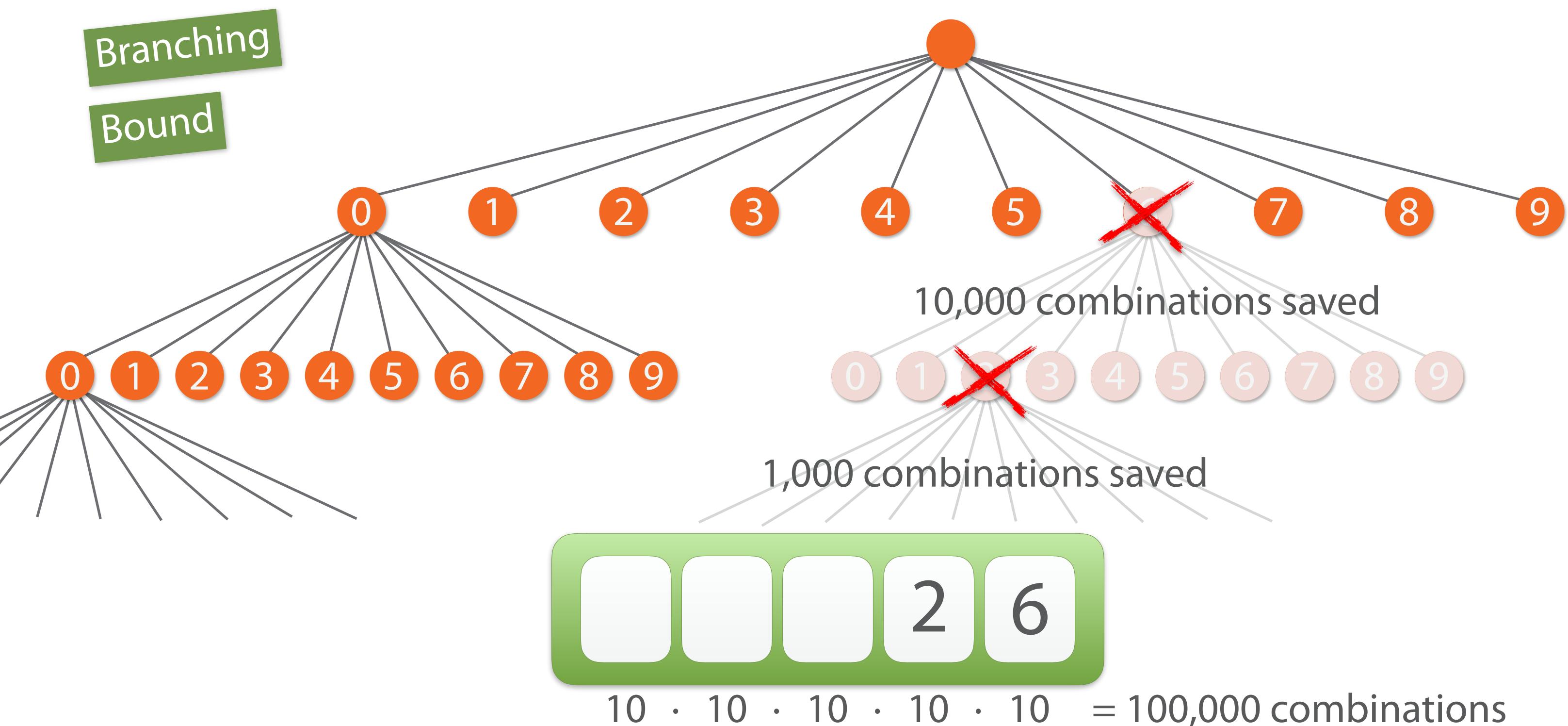
# Main Principle



# Main Principle



# Main Principle



# The Assignment Problem

# The Assignment Problem



# The Assignment Problem



# The Assignment Problem

- 1 
- 2 
- 3 
- 4 

- A 
- B 
- C 
- D 

# The Assignment Problem

	1	2	3	4
A		3	5	9
B		9	3	3
C		1	4	2
D		5	3	7
				2

# The Assignment Problem

	1	2	3	4
A		3	5	9
B		9	3	3
C		1	4	2
D		5	3	7



# The Assignment Problem

Best so far:

	1 	2 	3 	4 
A 	3	5	9	2
B 	9	3	3	4
C 	1	4	2	6
D 	5	3	7	2



# The Assignment Problem

Best so far:

	1	2	3	4
A	3	5	9	2
B	9	3	3	4
C	1	4	2	6
D	5	3	7	2



# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
A	3	5	9	2
B	9	3	3	4
C	1	4	2	6
D	5	3	7	2



# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
A	3	5	9	2
B	9	3	3	4
C	1	4	2	6
D	5	3	7	2



# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
A	3	5	9	2
B	9	3	3	4
C	1	4	2	6
D	5	3	7	2

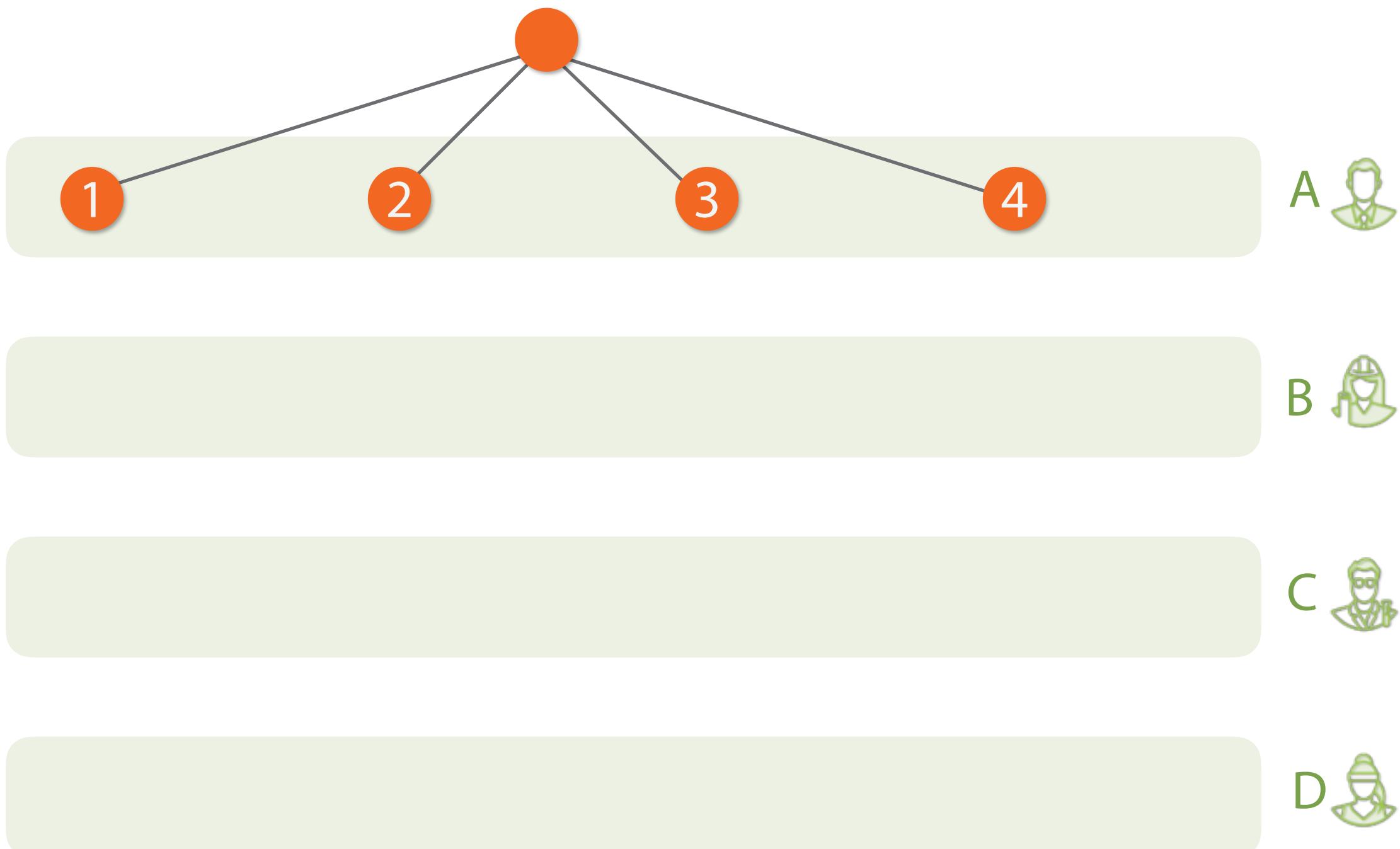


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	2
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2

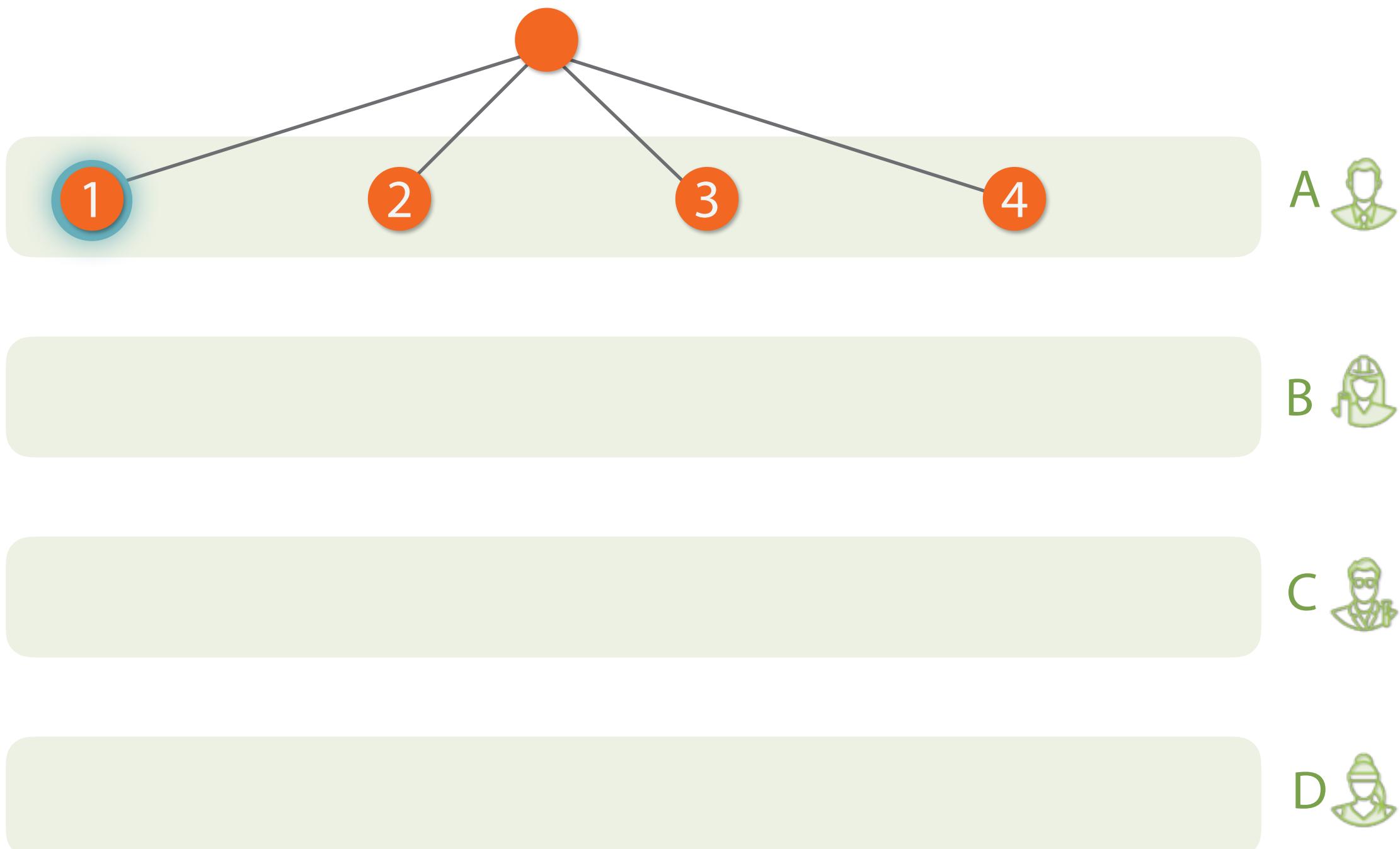


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	5	9	2	
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2

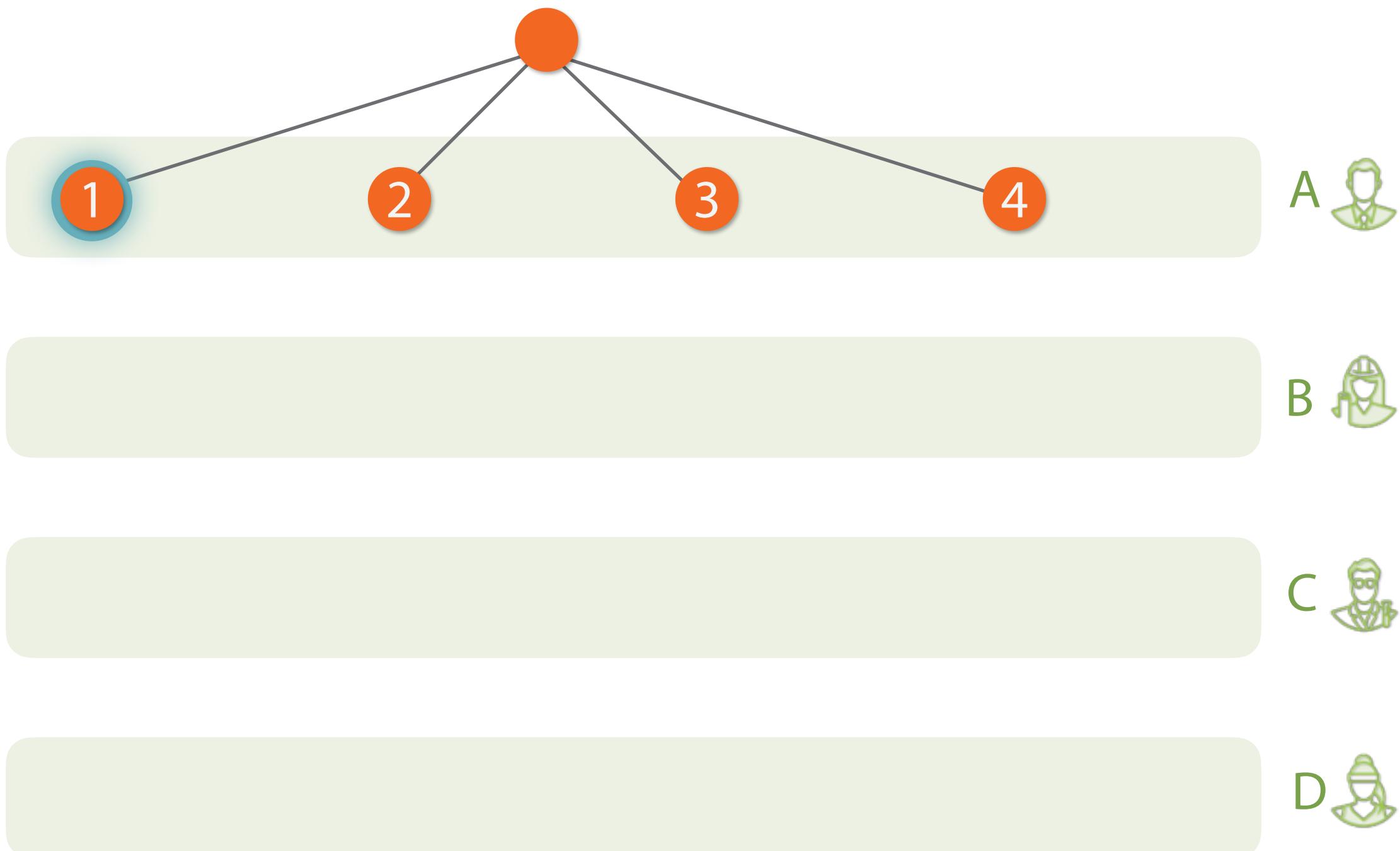


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
A	3	5	9	2
B	9	3	3	4
C	1	4	2	6
D	5	3	7	2

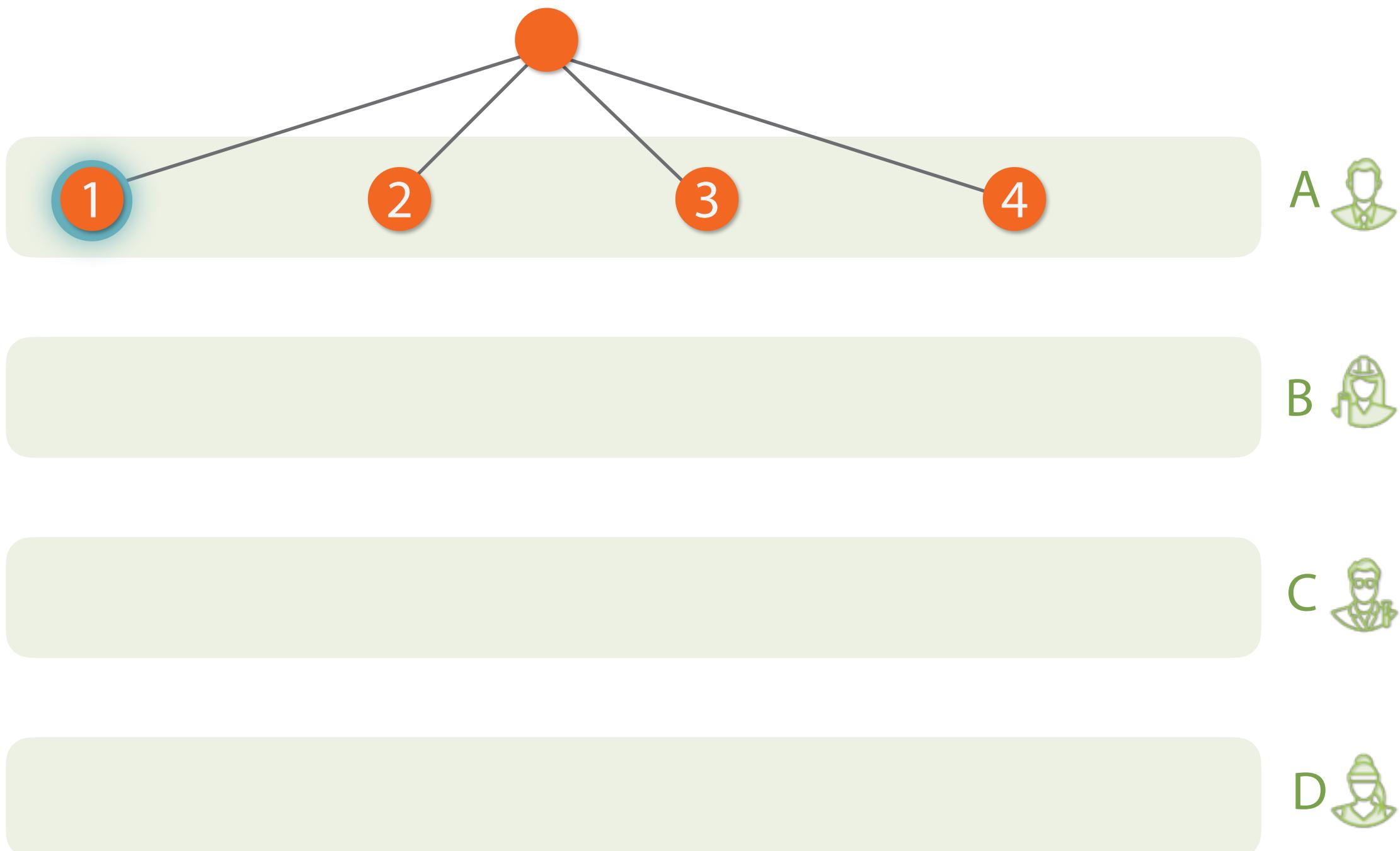


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	2
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2

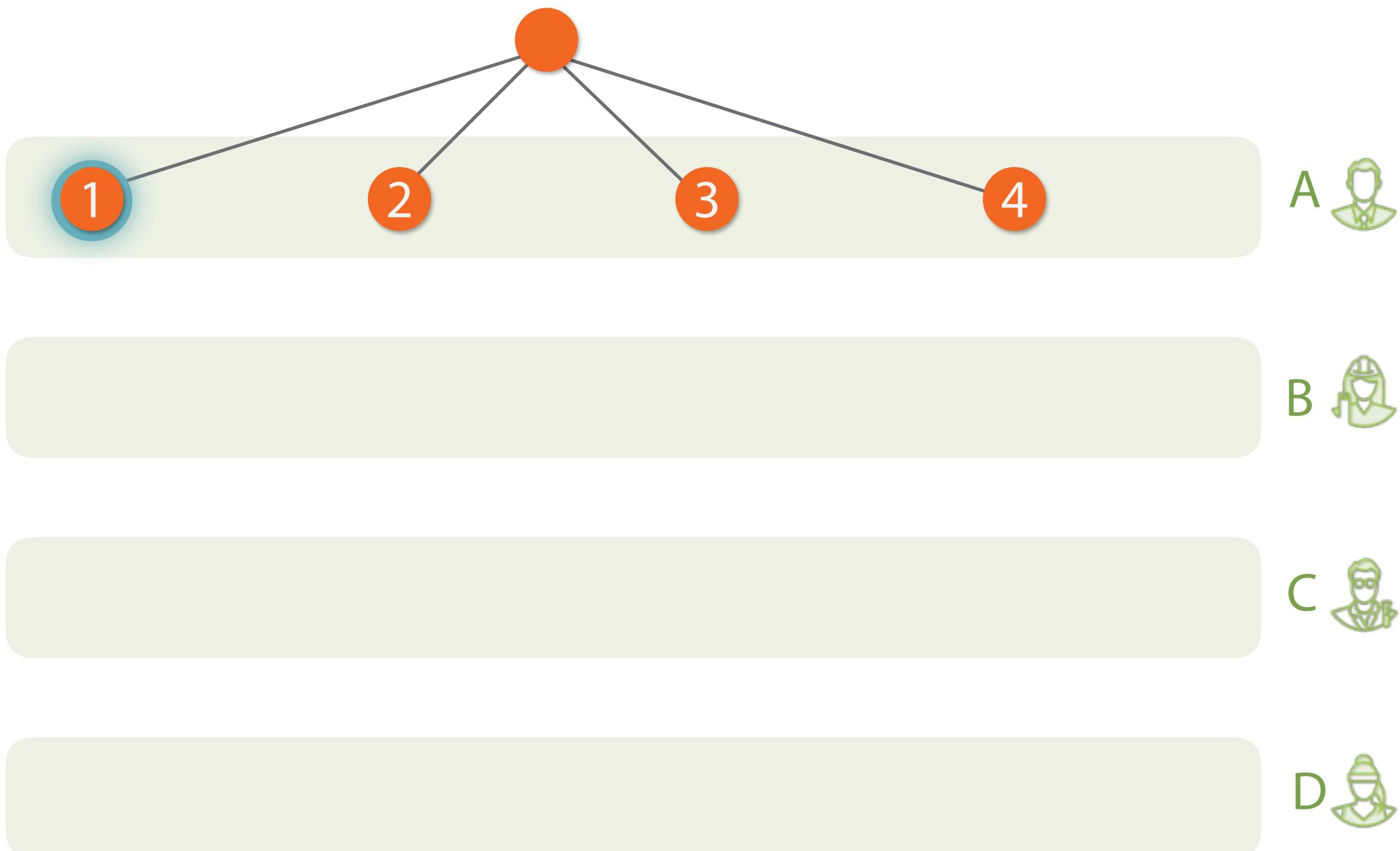


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	2
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2

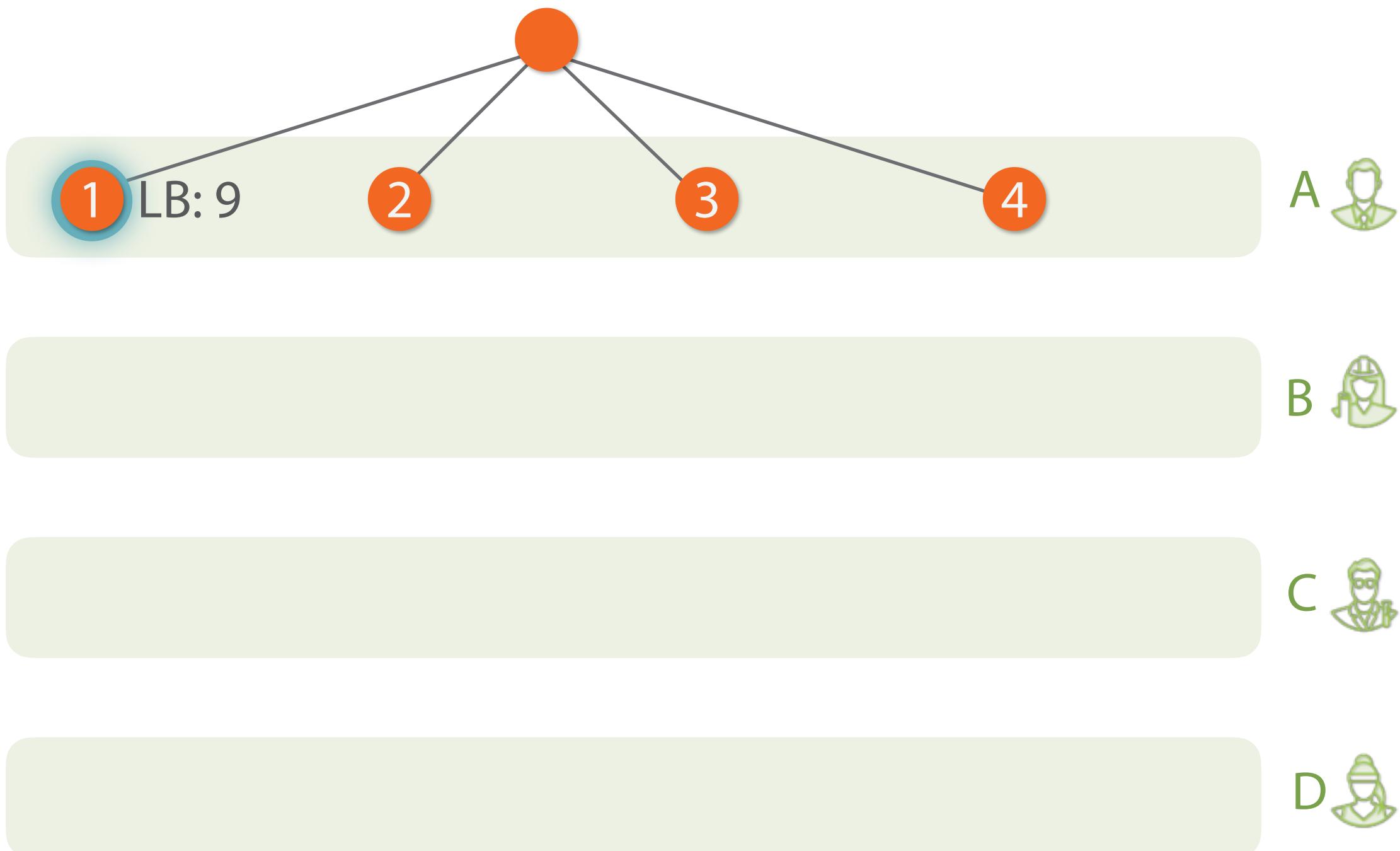


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	2
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2

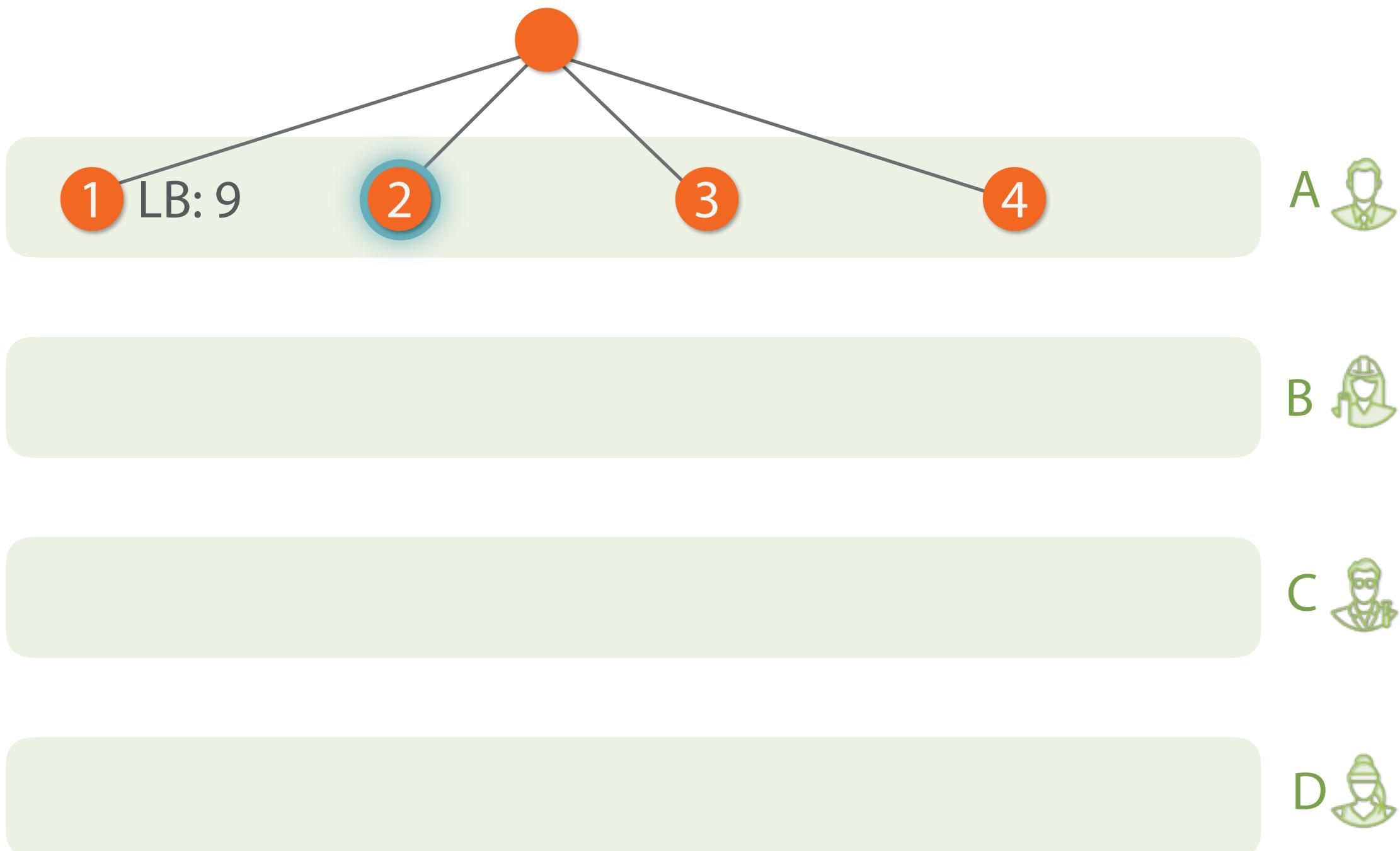


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	2
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2

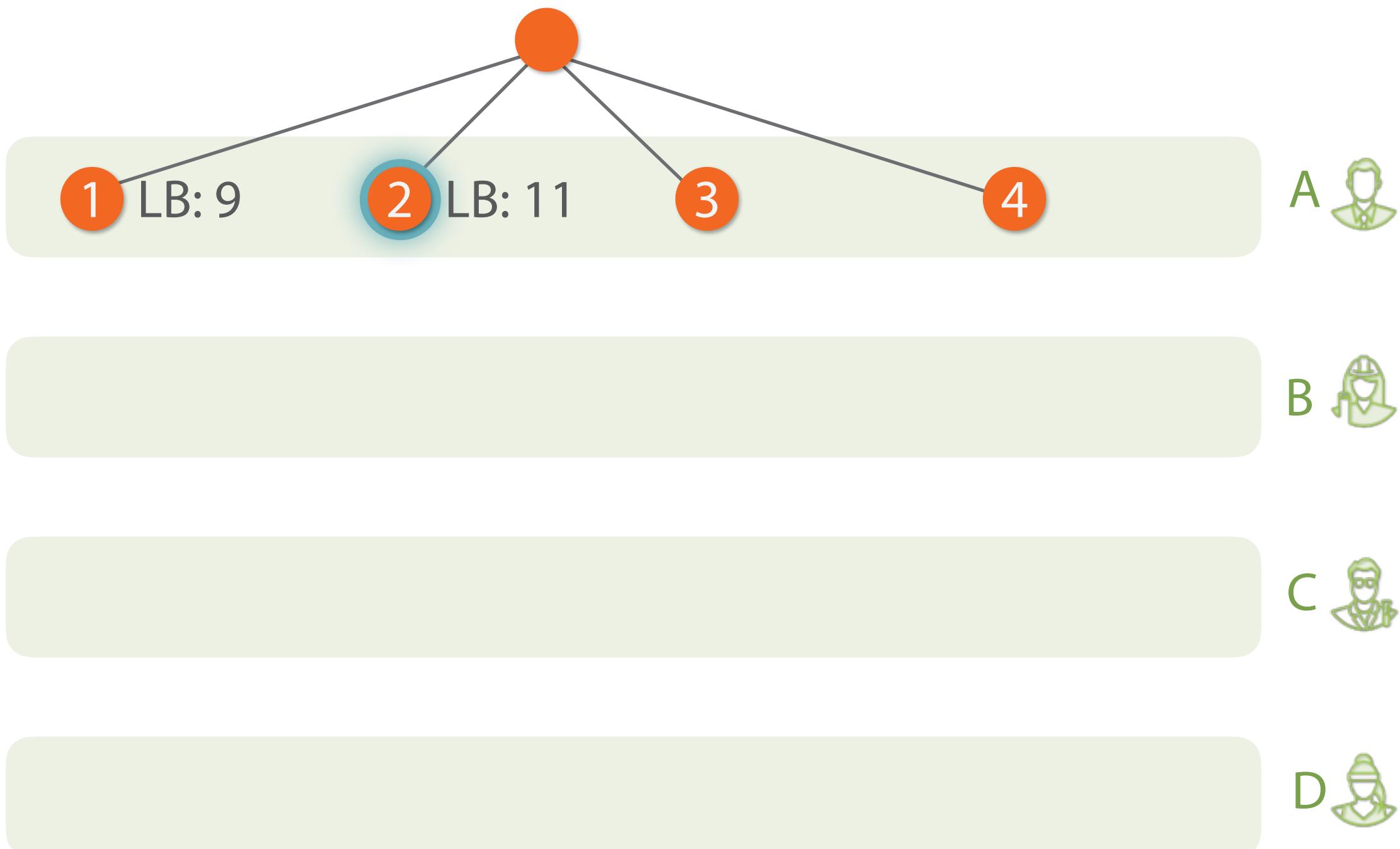


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	2
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2

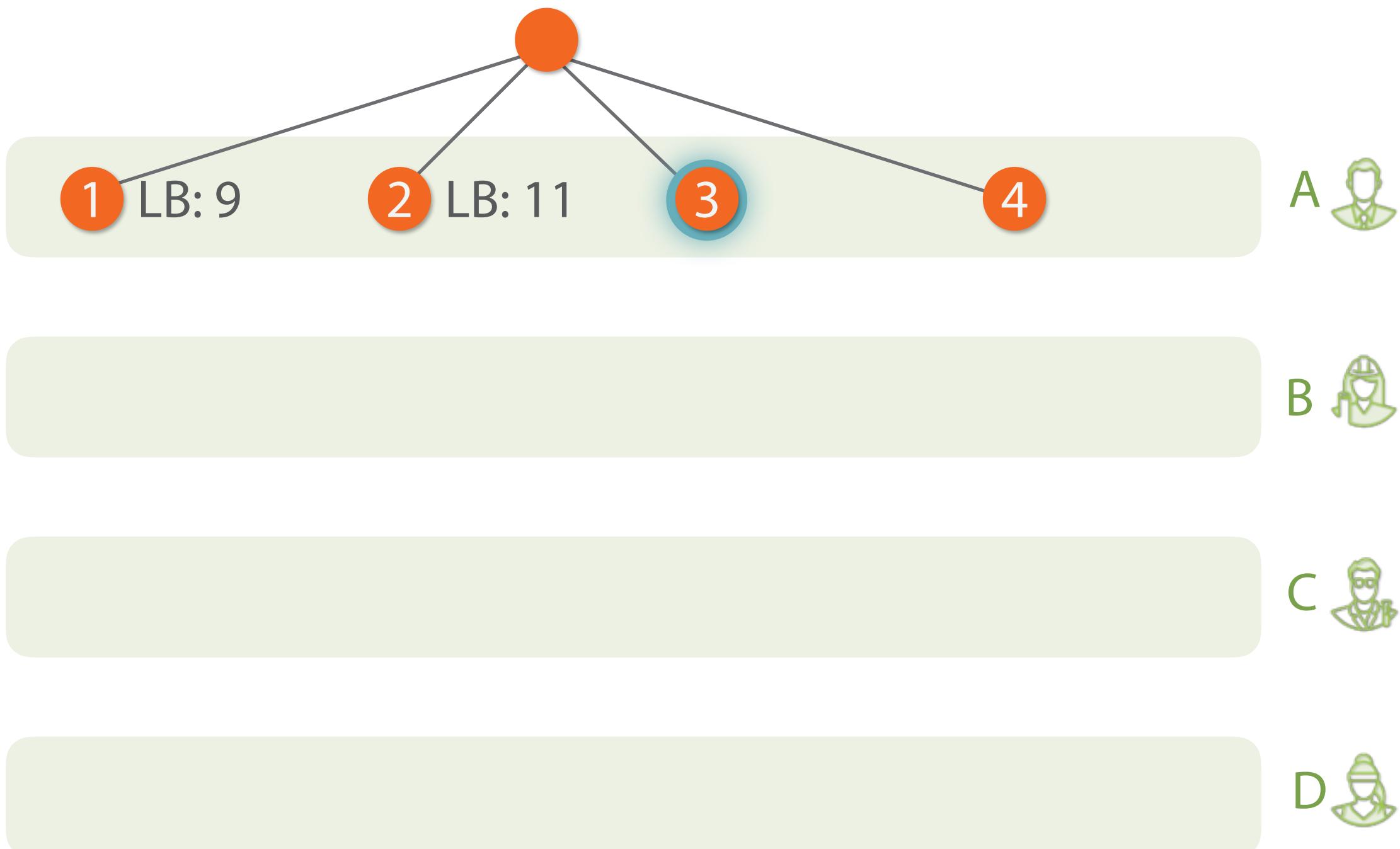


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	2
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2

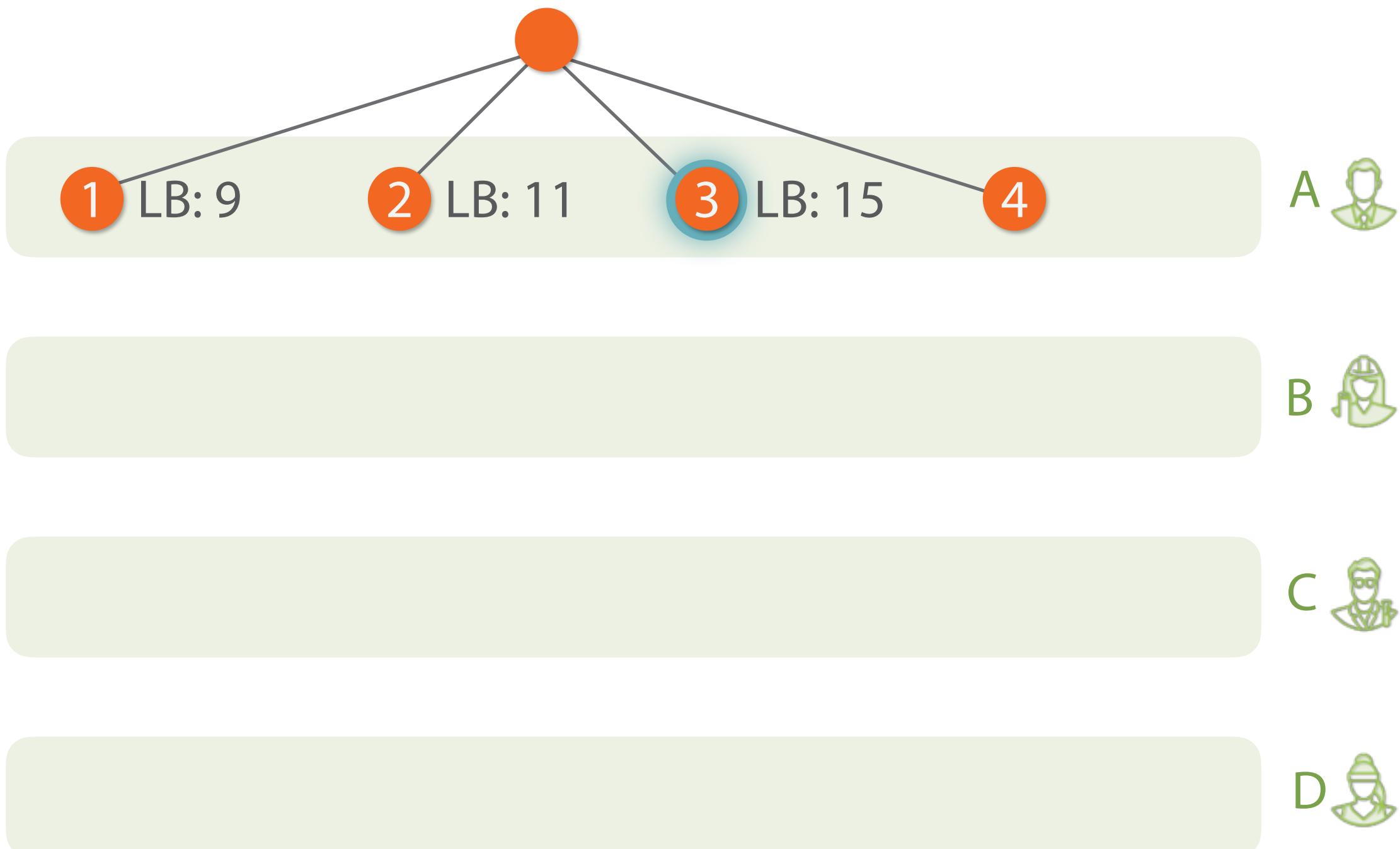


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	2
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2

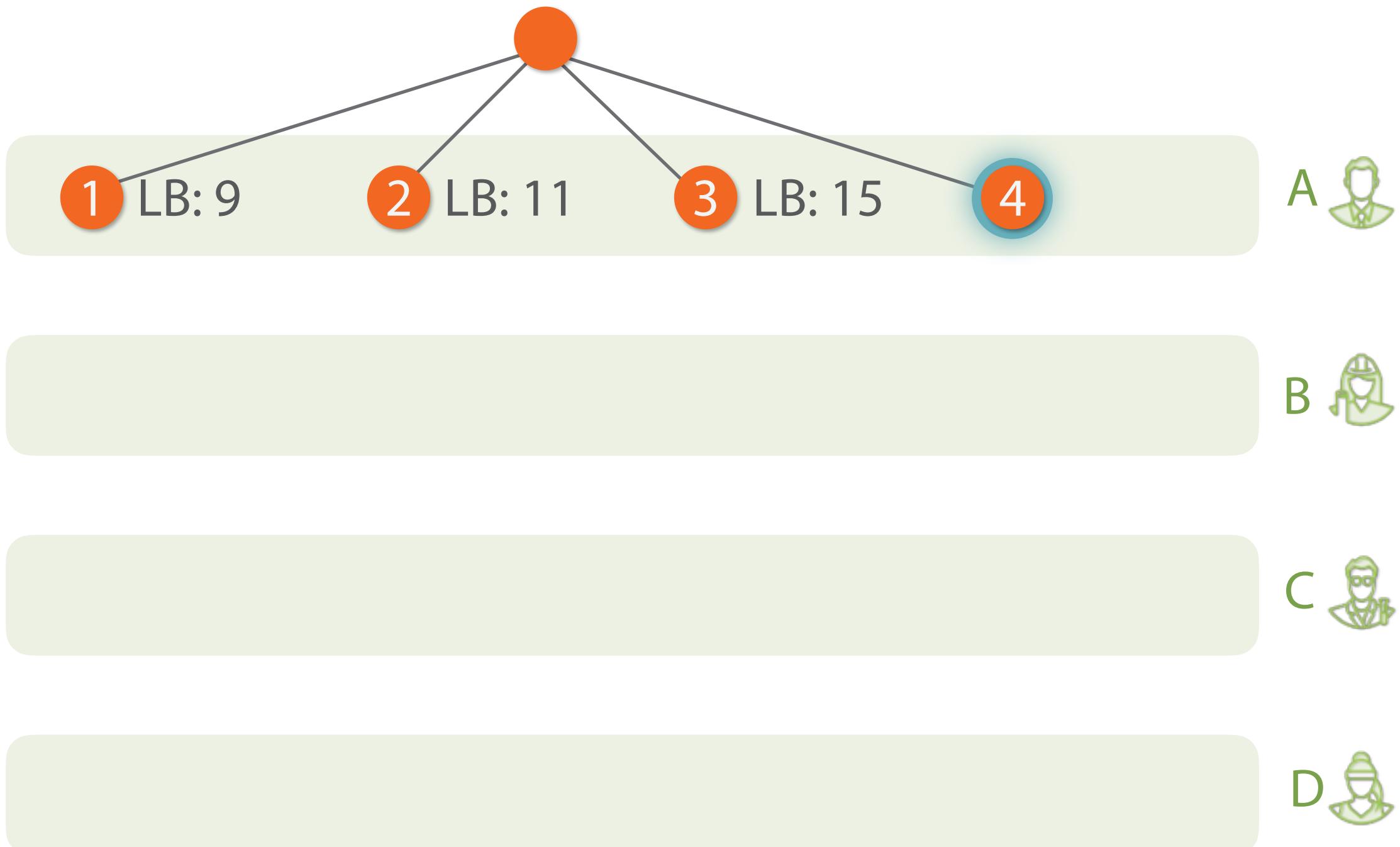


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	4
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2

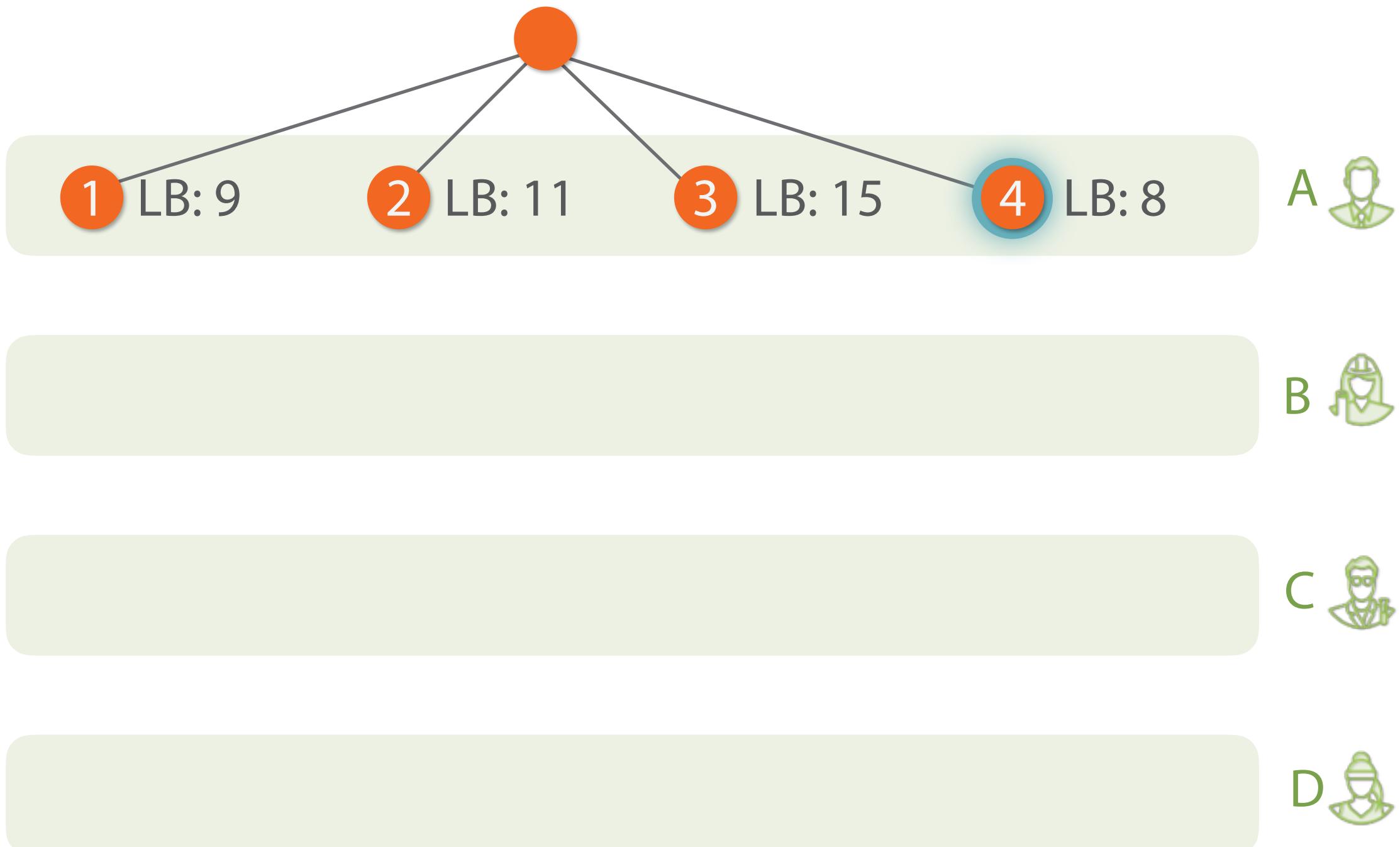


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	4
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2

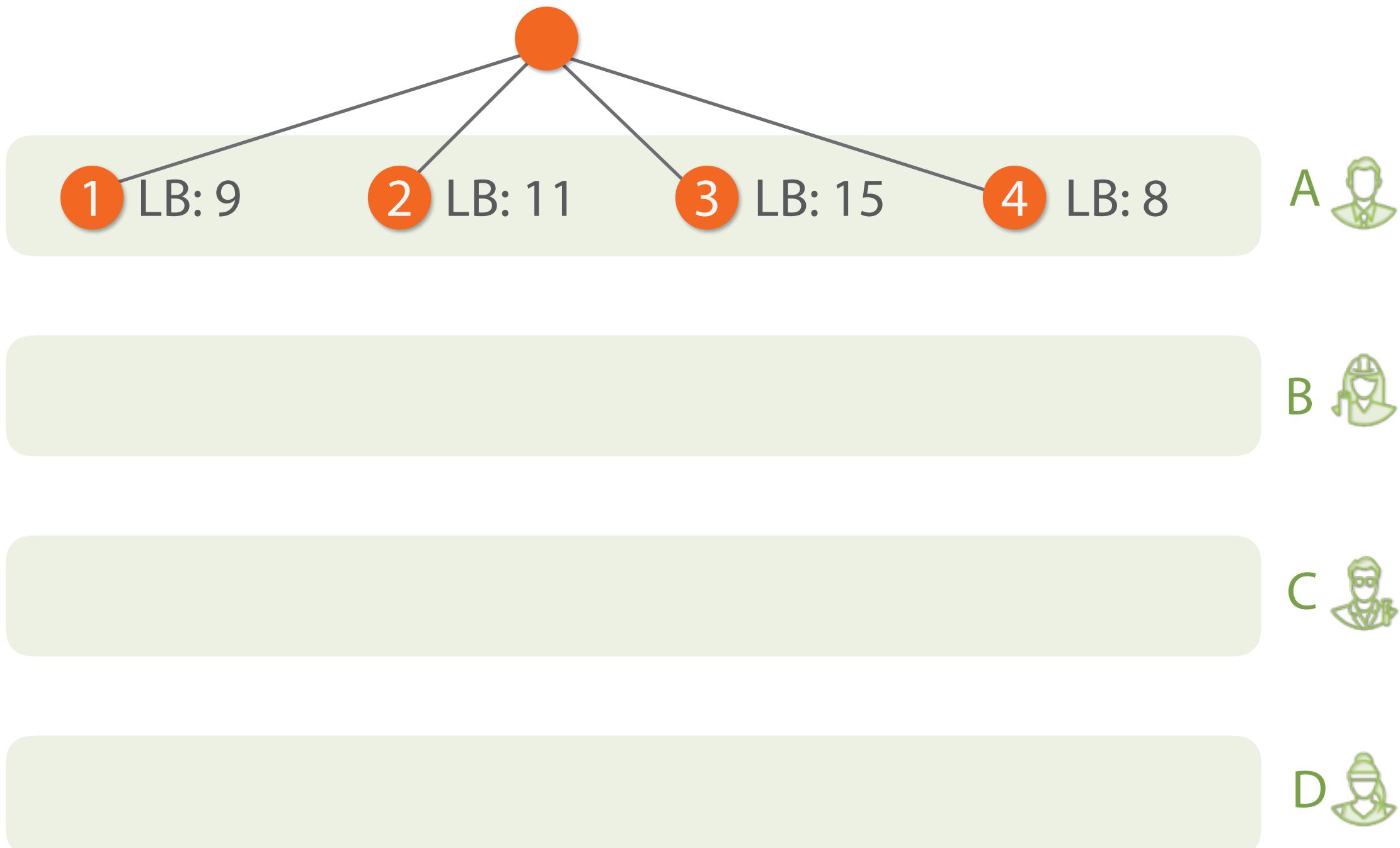


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	2
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2

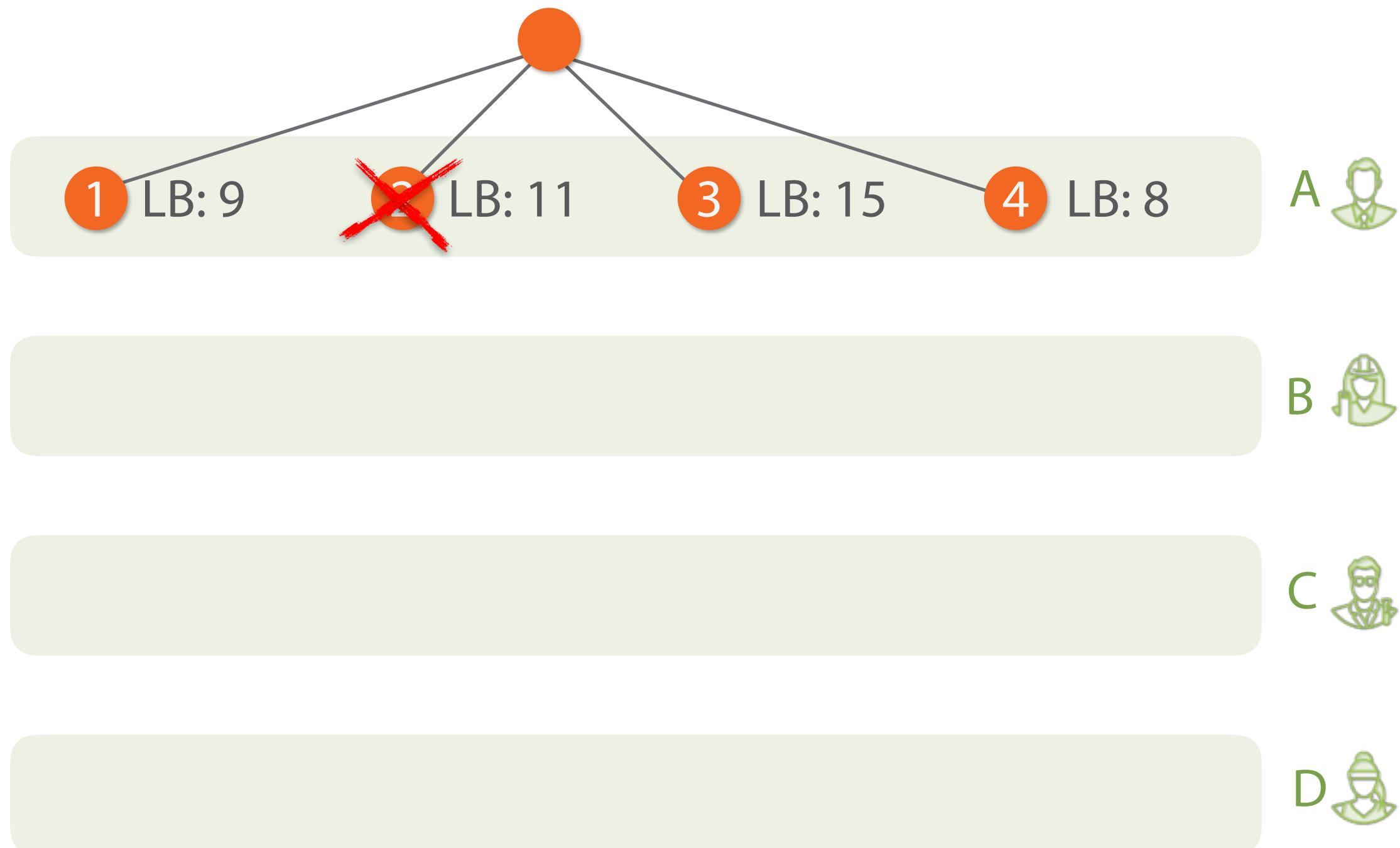


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	2
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2

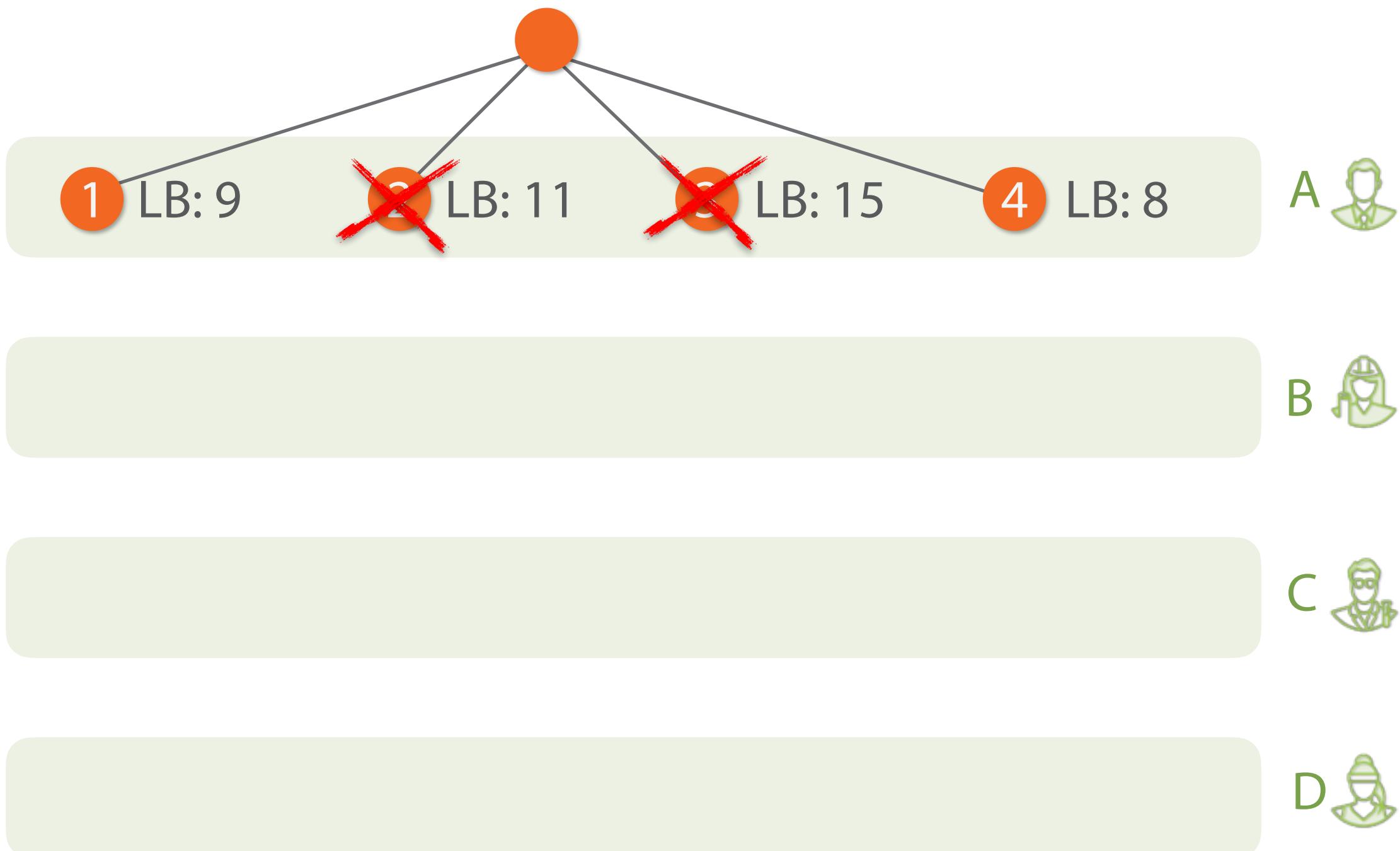


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	2
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2



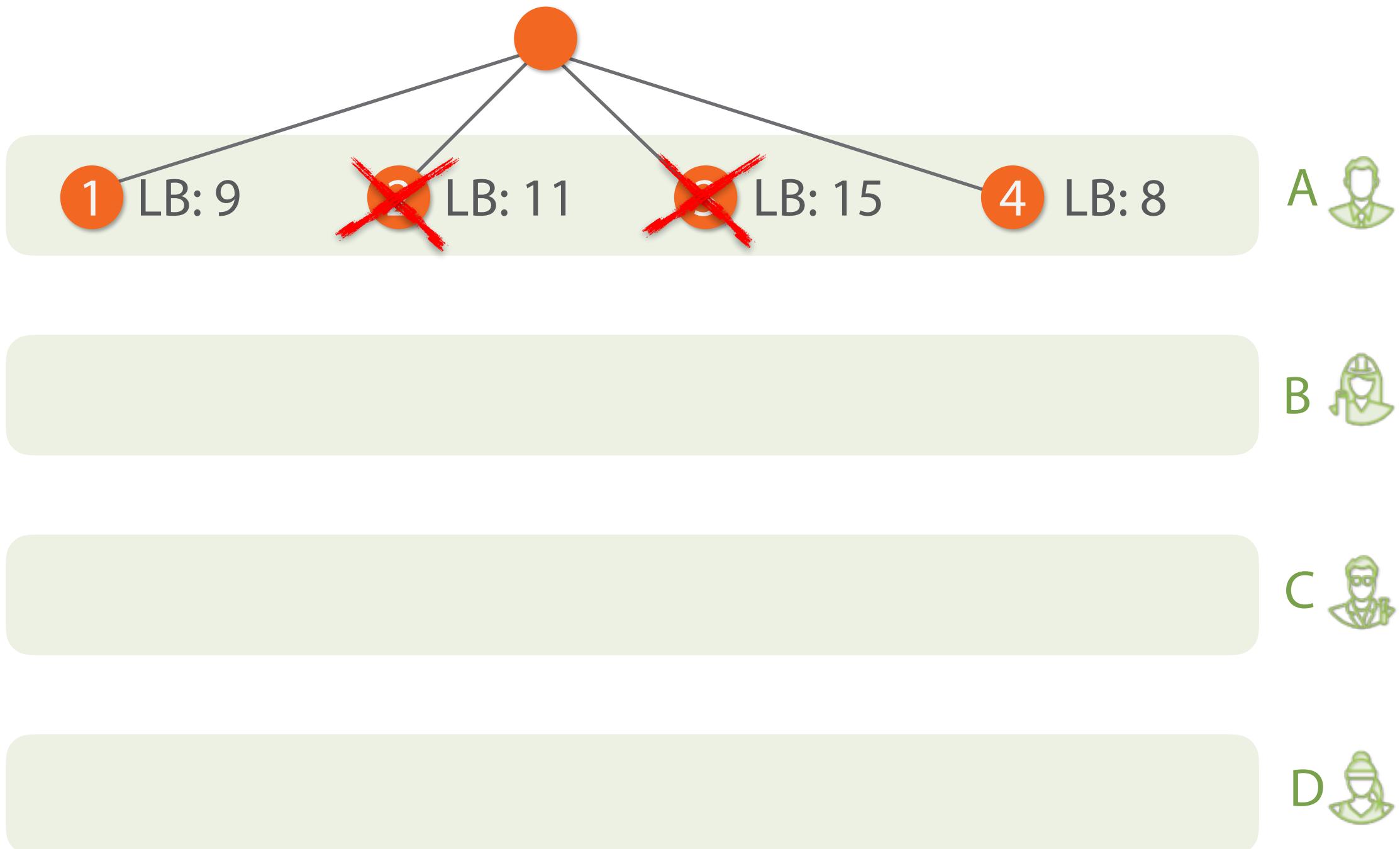
# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	2
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2

(1, ?, ?, ?)  
LB: 9

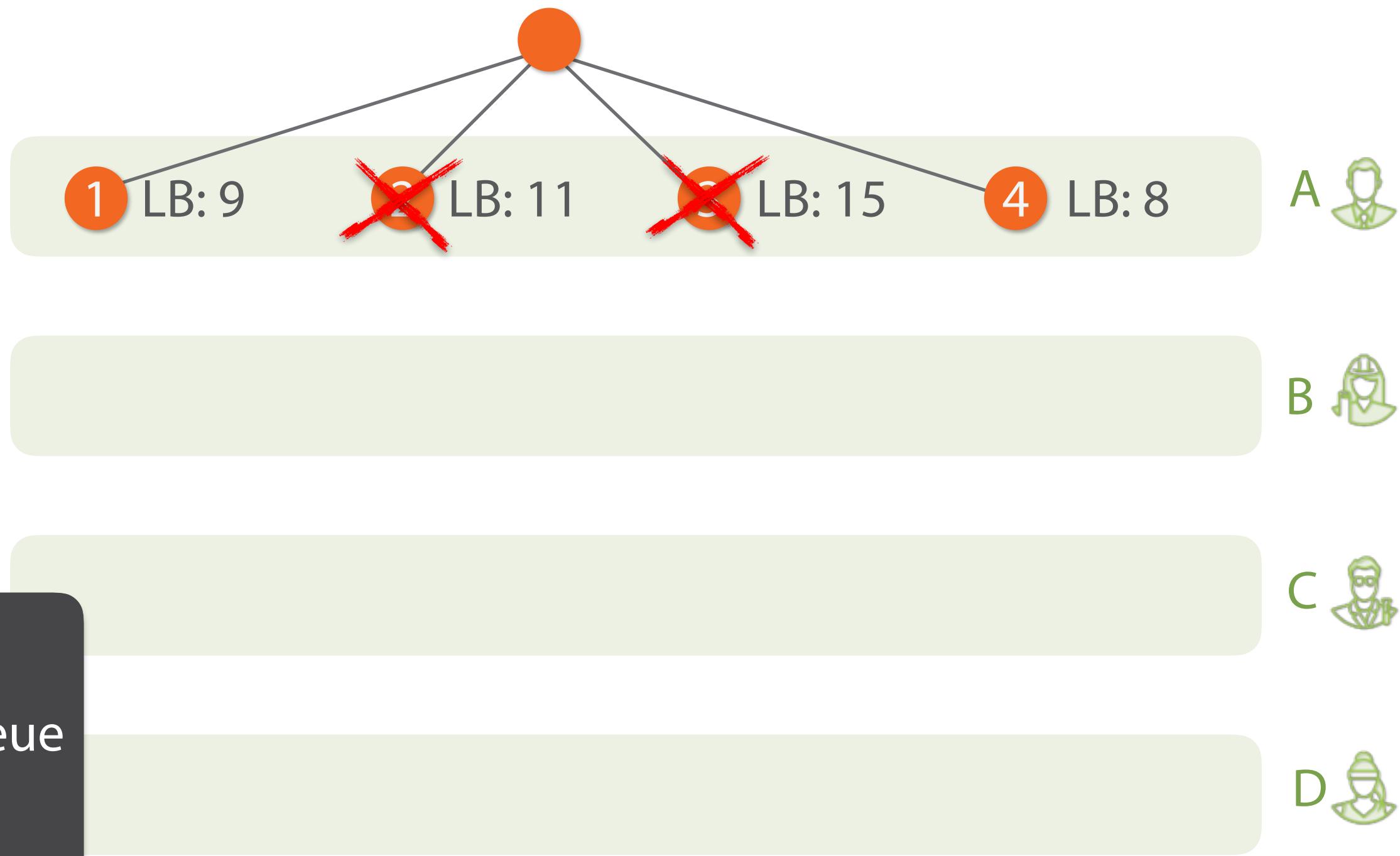


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	2
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2

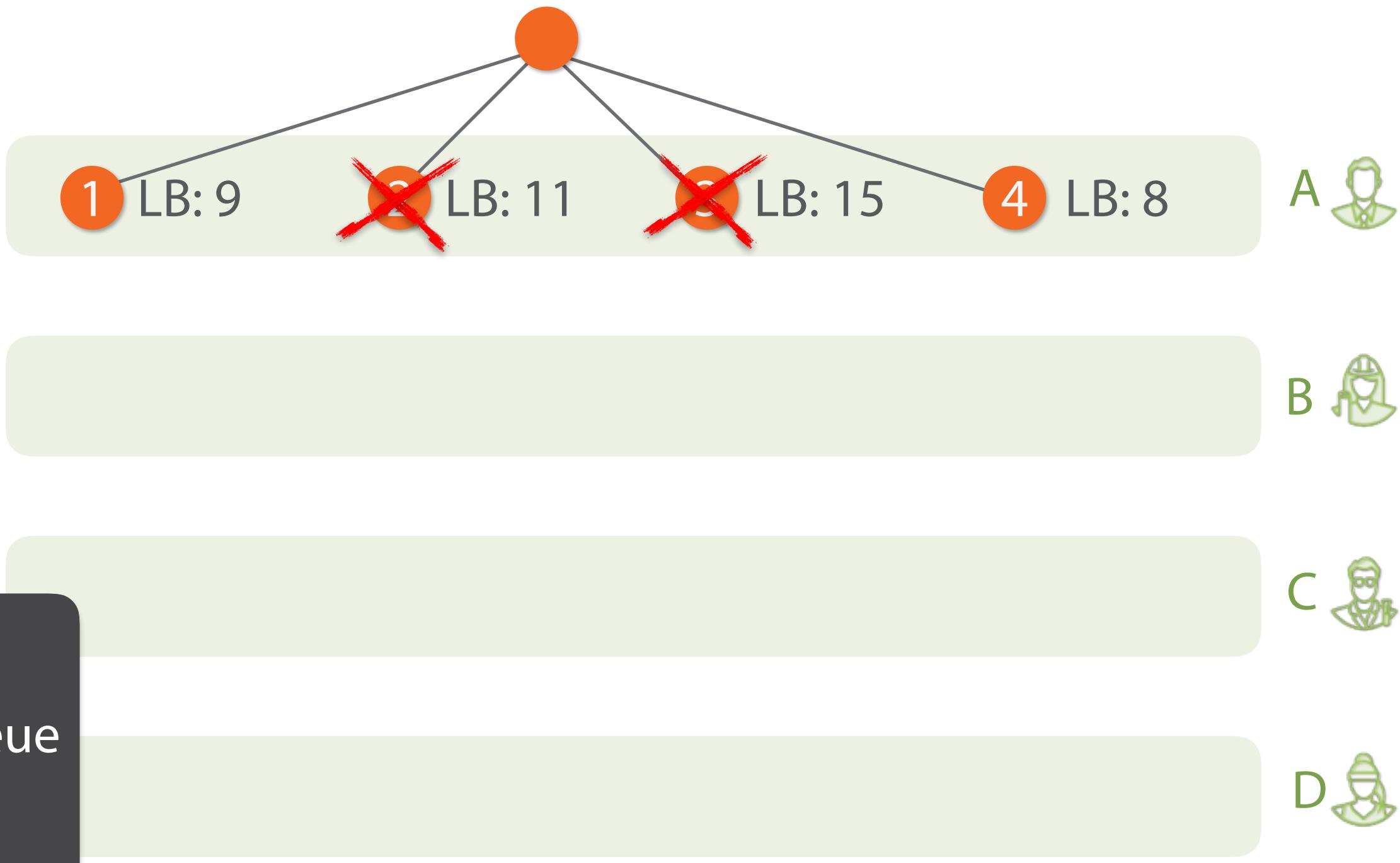


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	2
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2

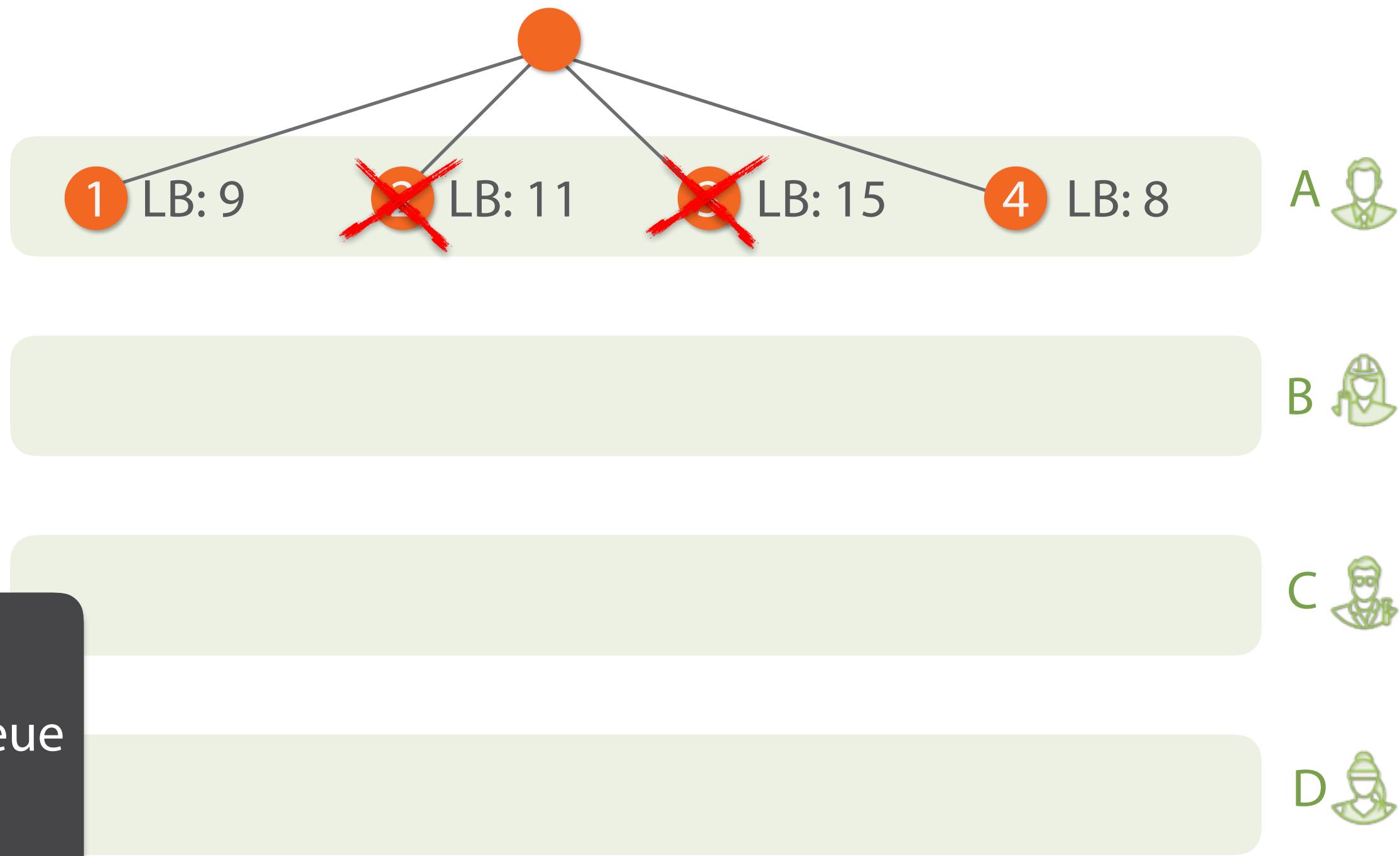


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	2
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2

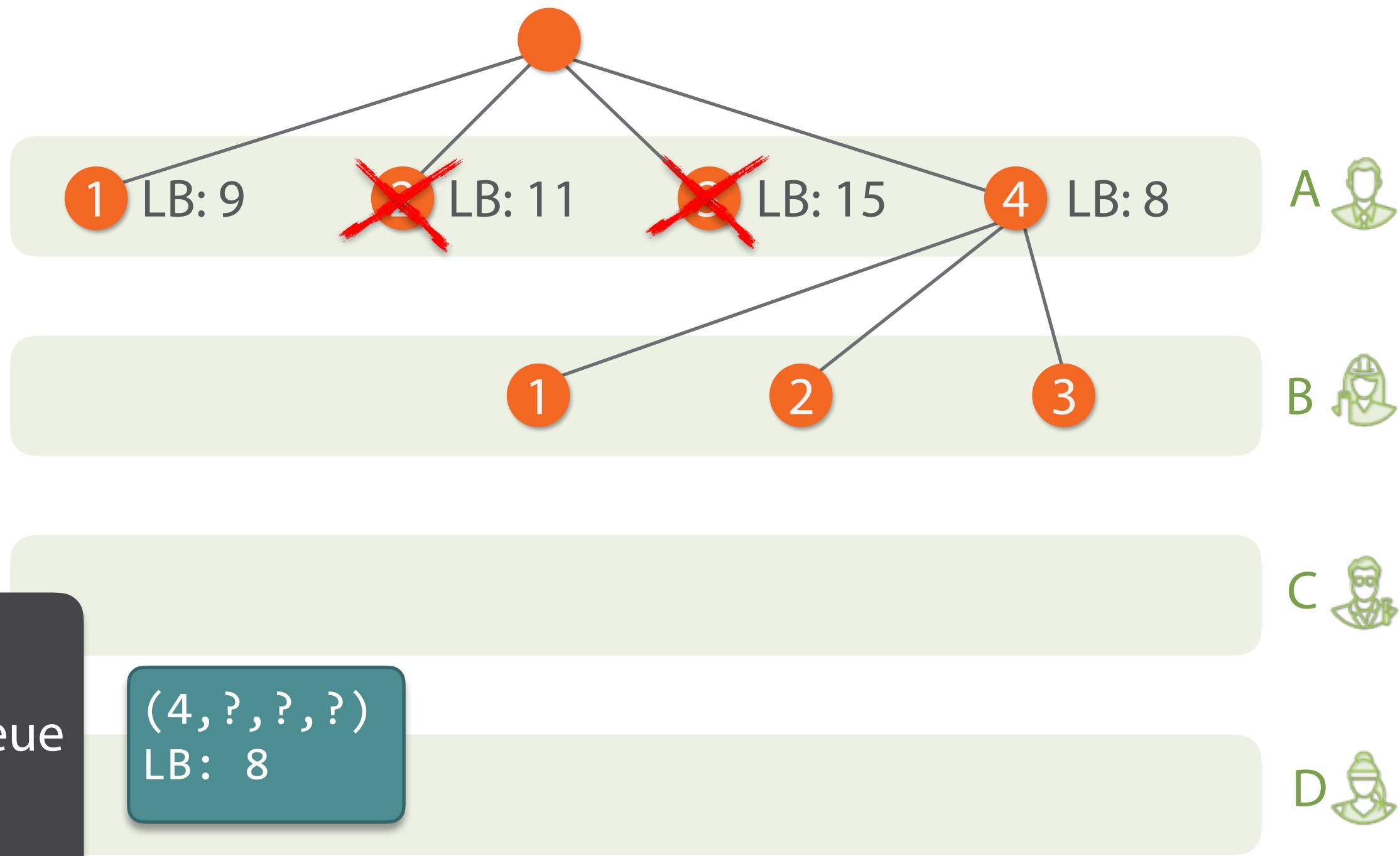


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	2
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2

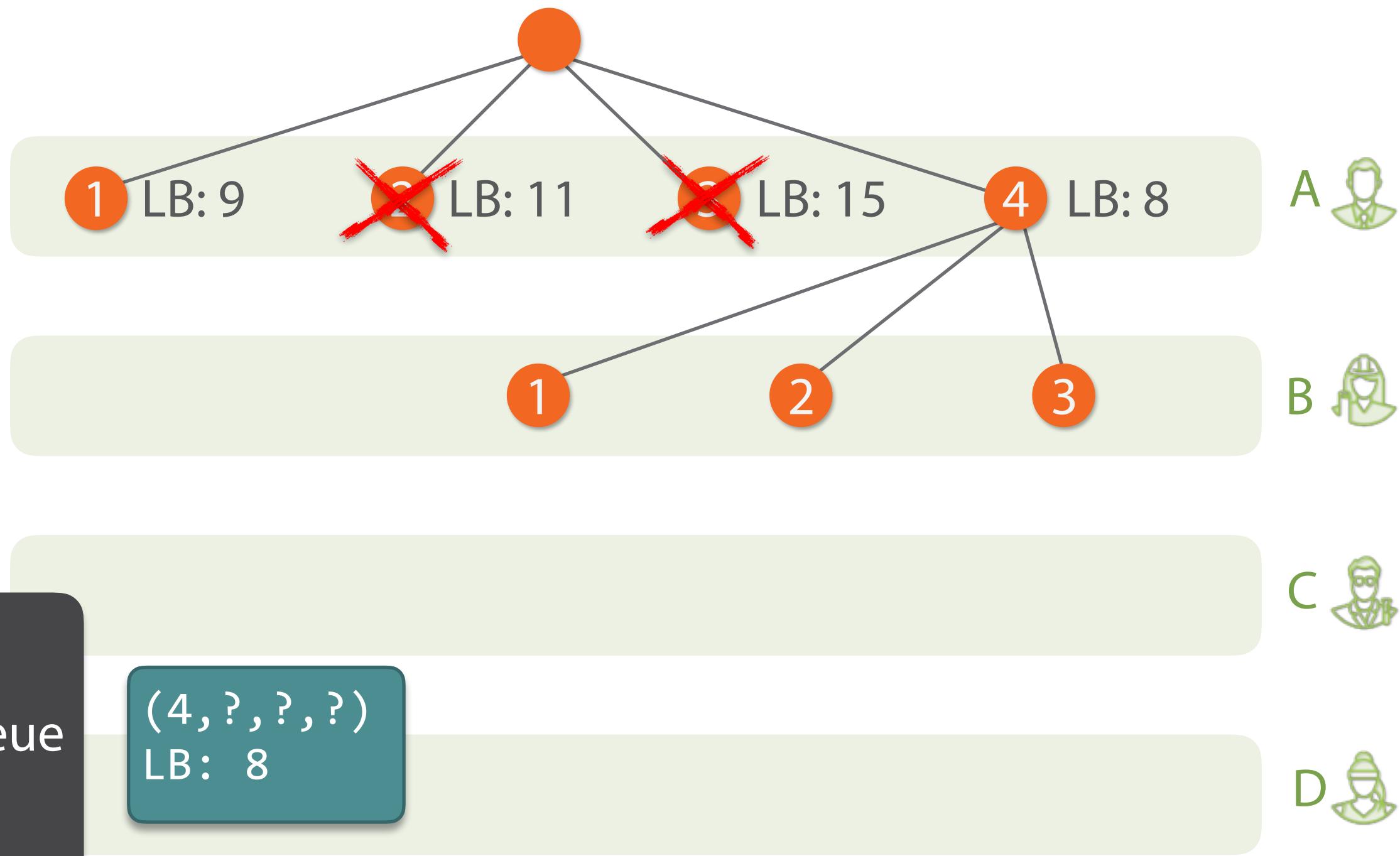


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	4
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2

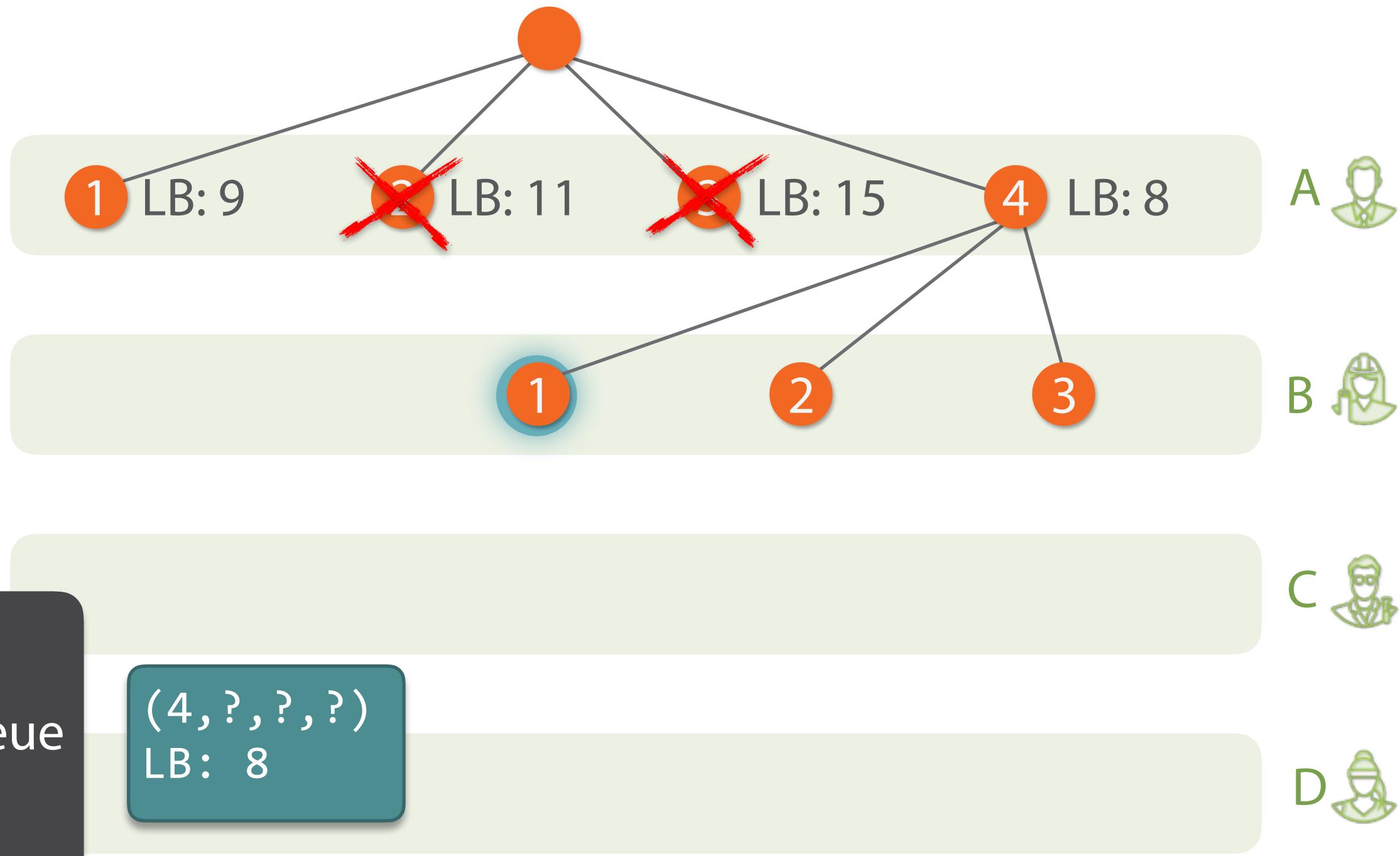


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	4
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2

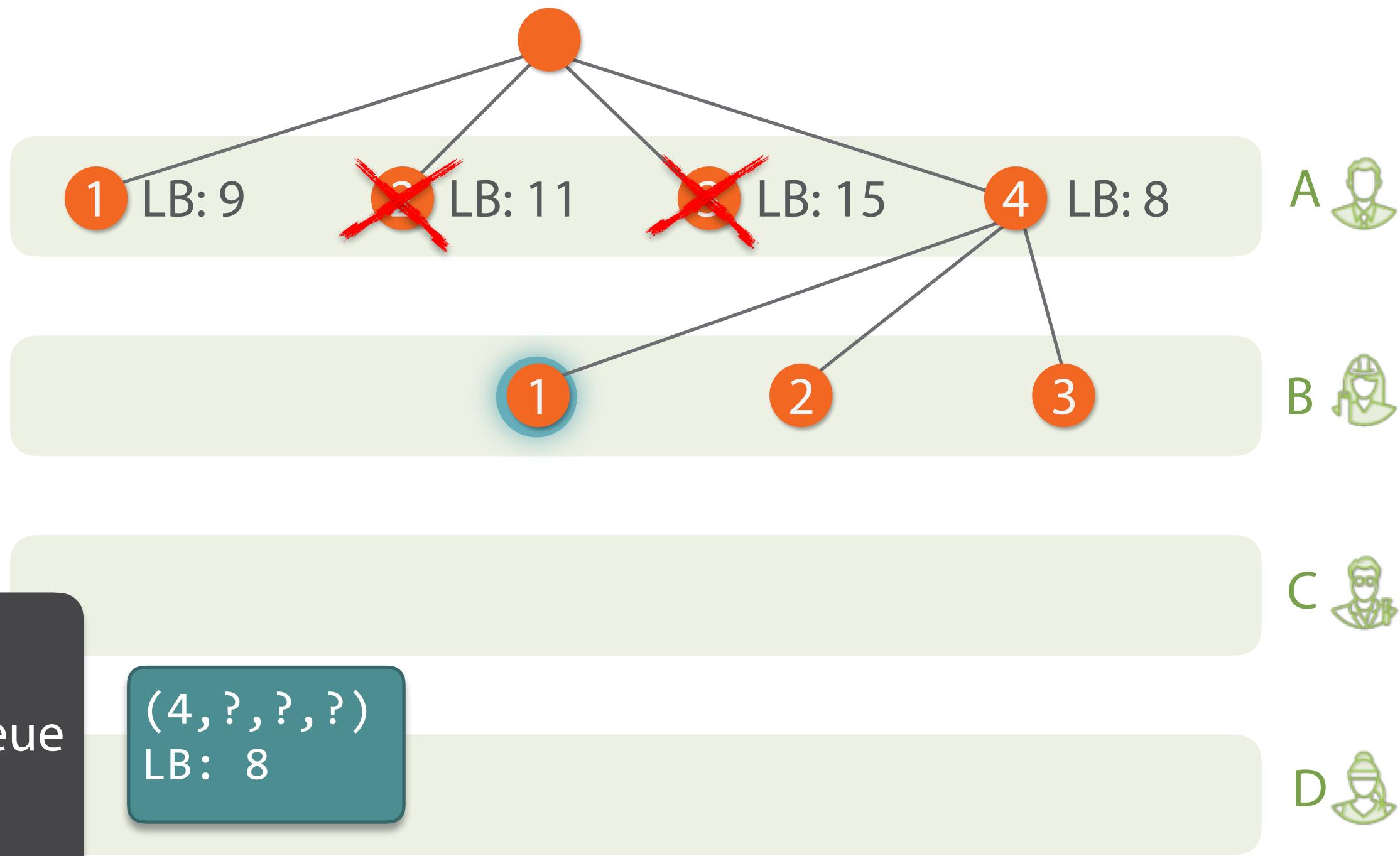


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	4
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2

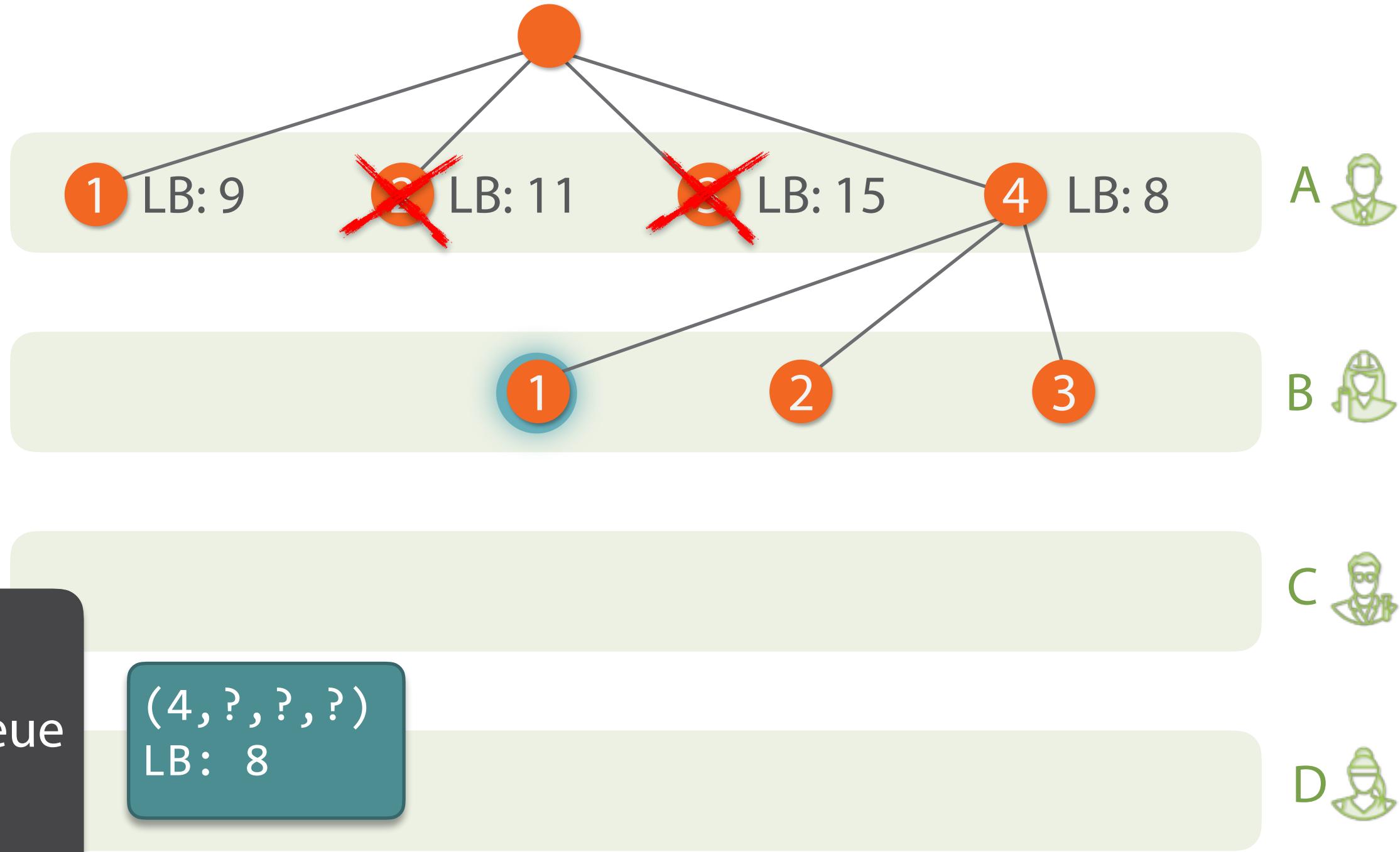


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	4
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2

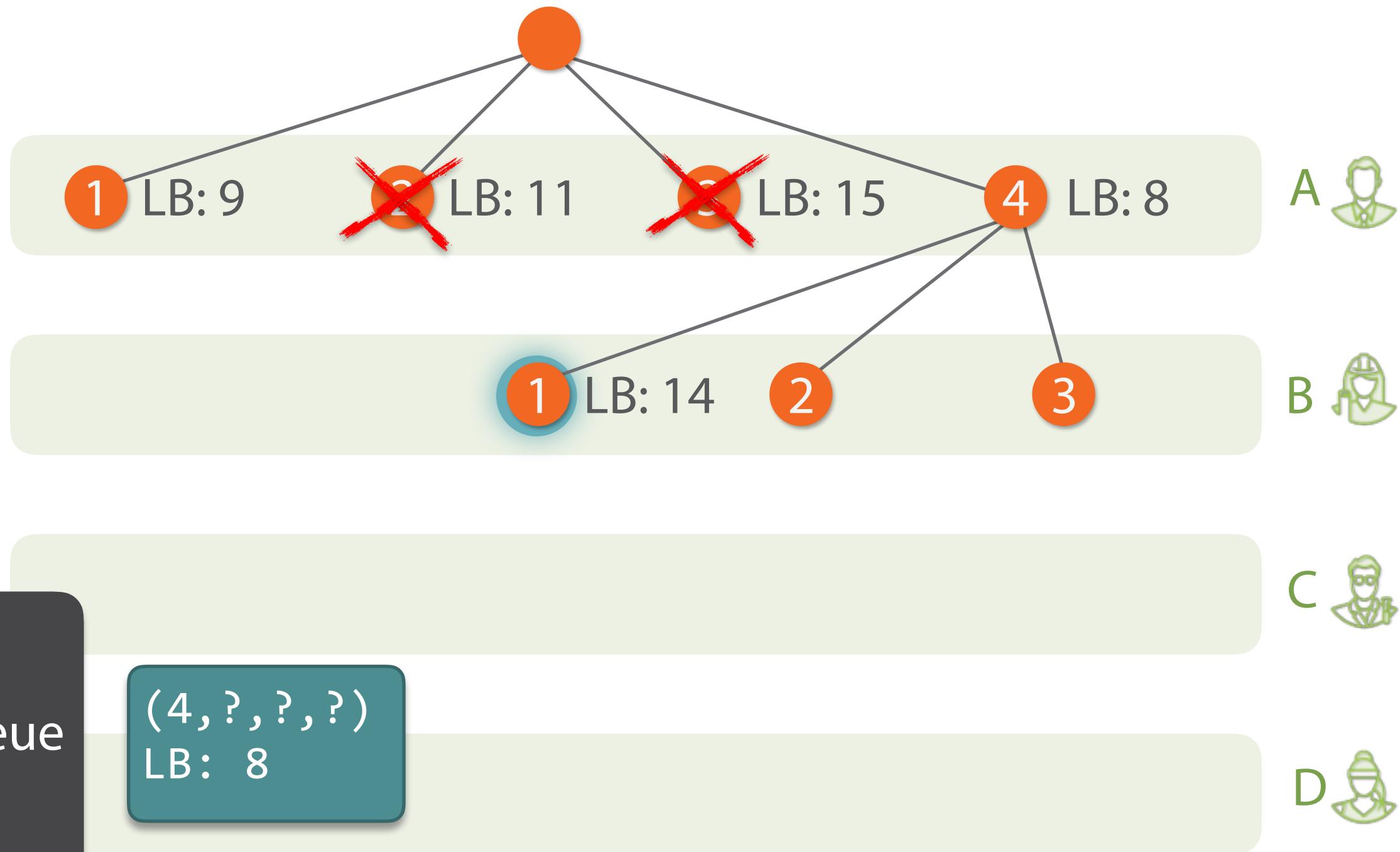


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	4
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2

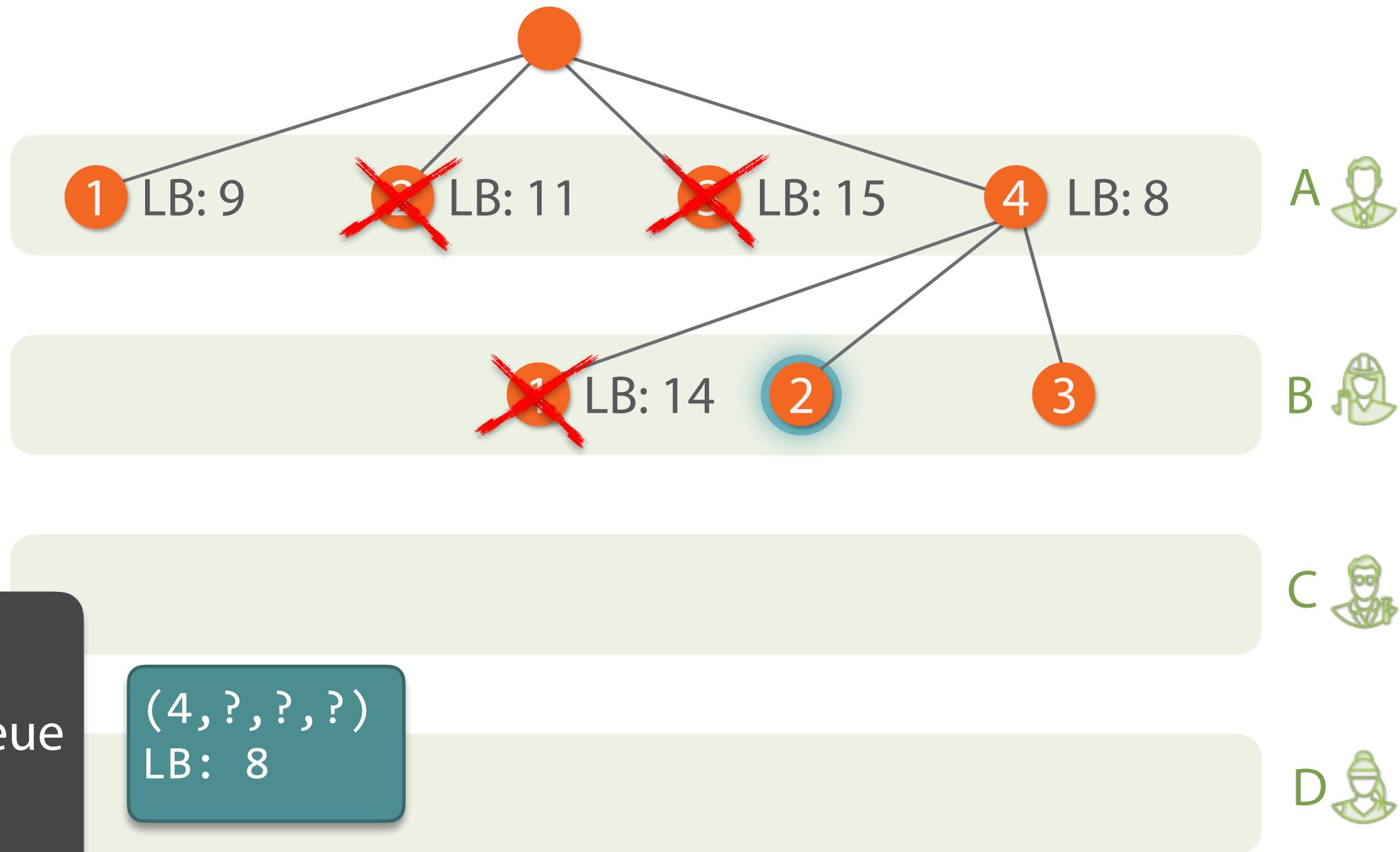


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	4
2	9	3	3	4
3	4	2	6	
4	5	3	7	2

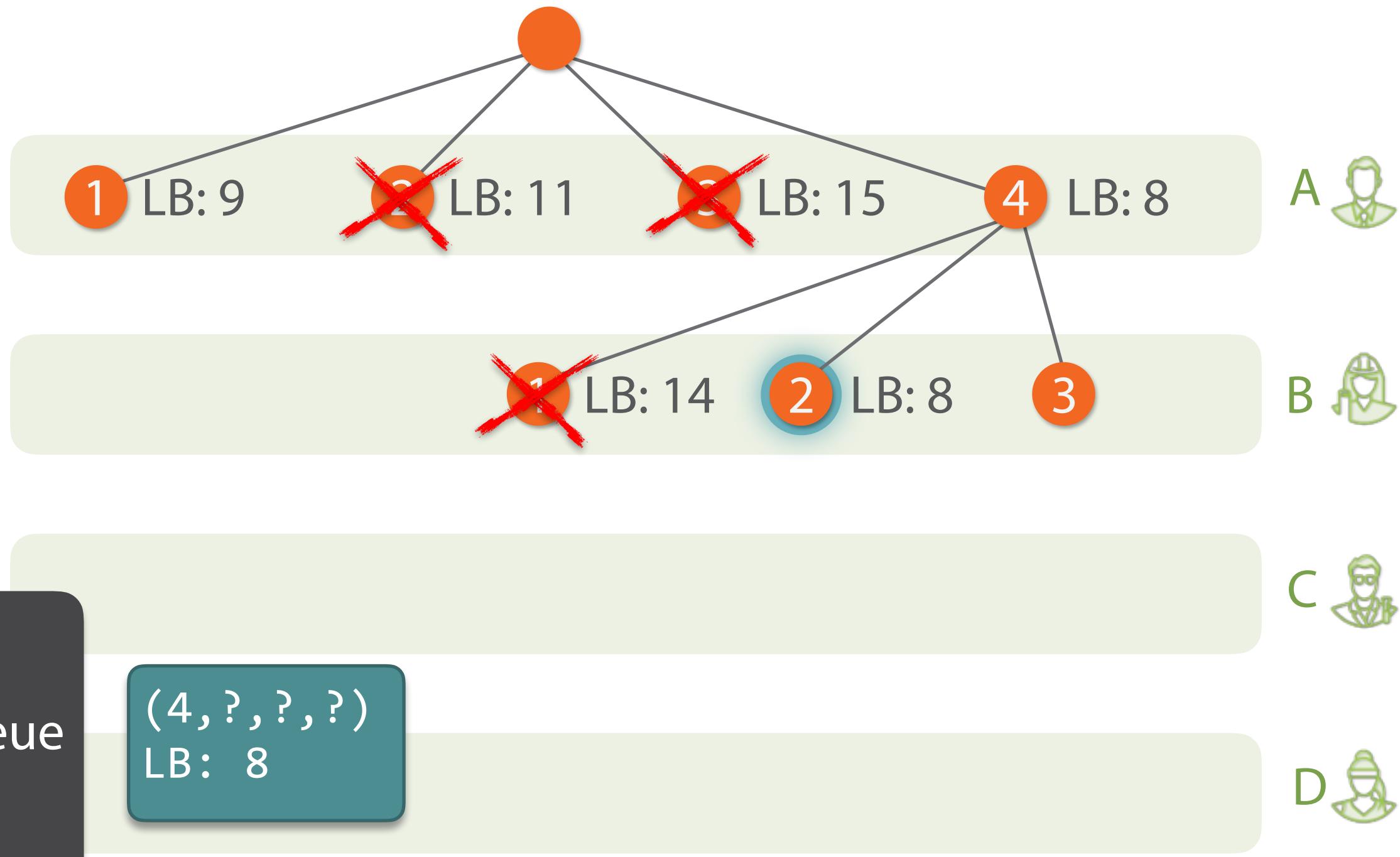


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	4
2	9	3	3	4
3	4	2	6	
4	5	3	7	2

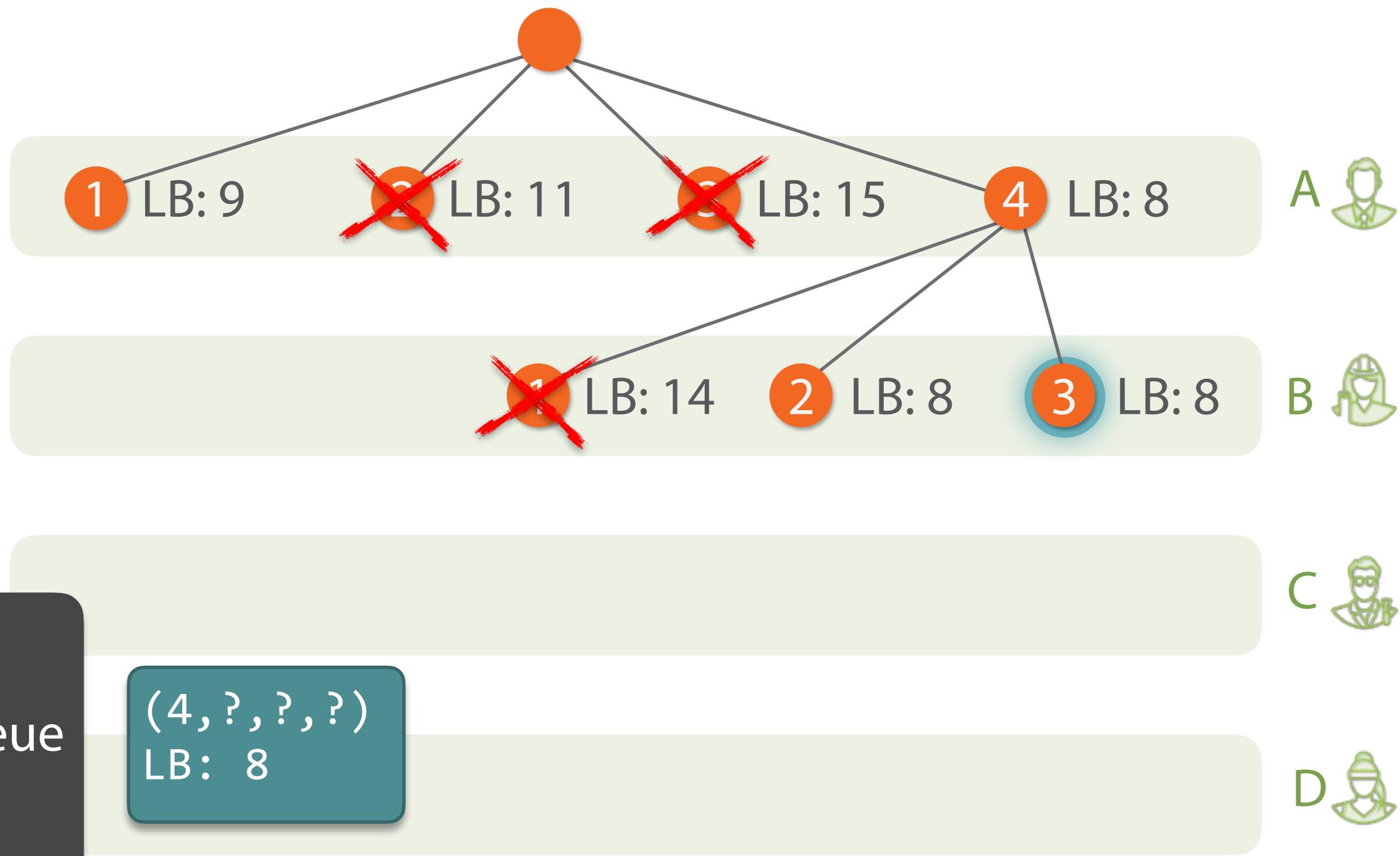


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	2
2	9	3	3	4
3	4	2	6	
4	5	3	7	2

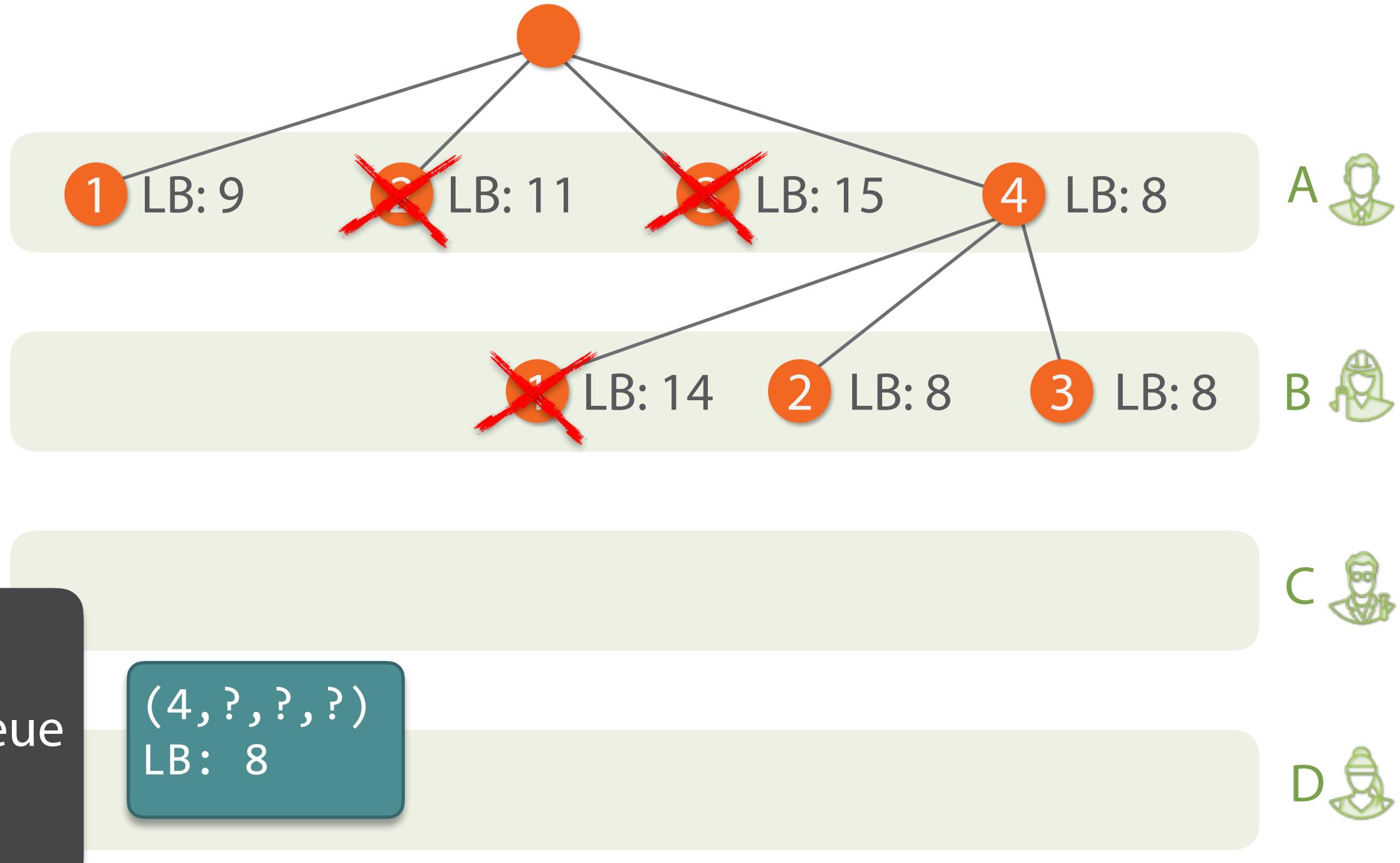


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	2
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2

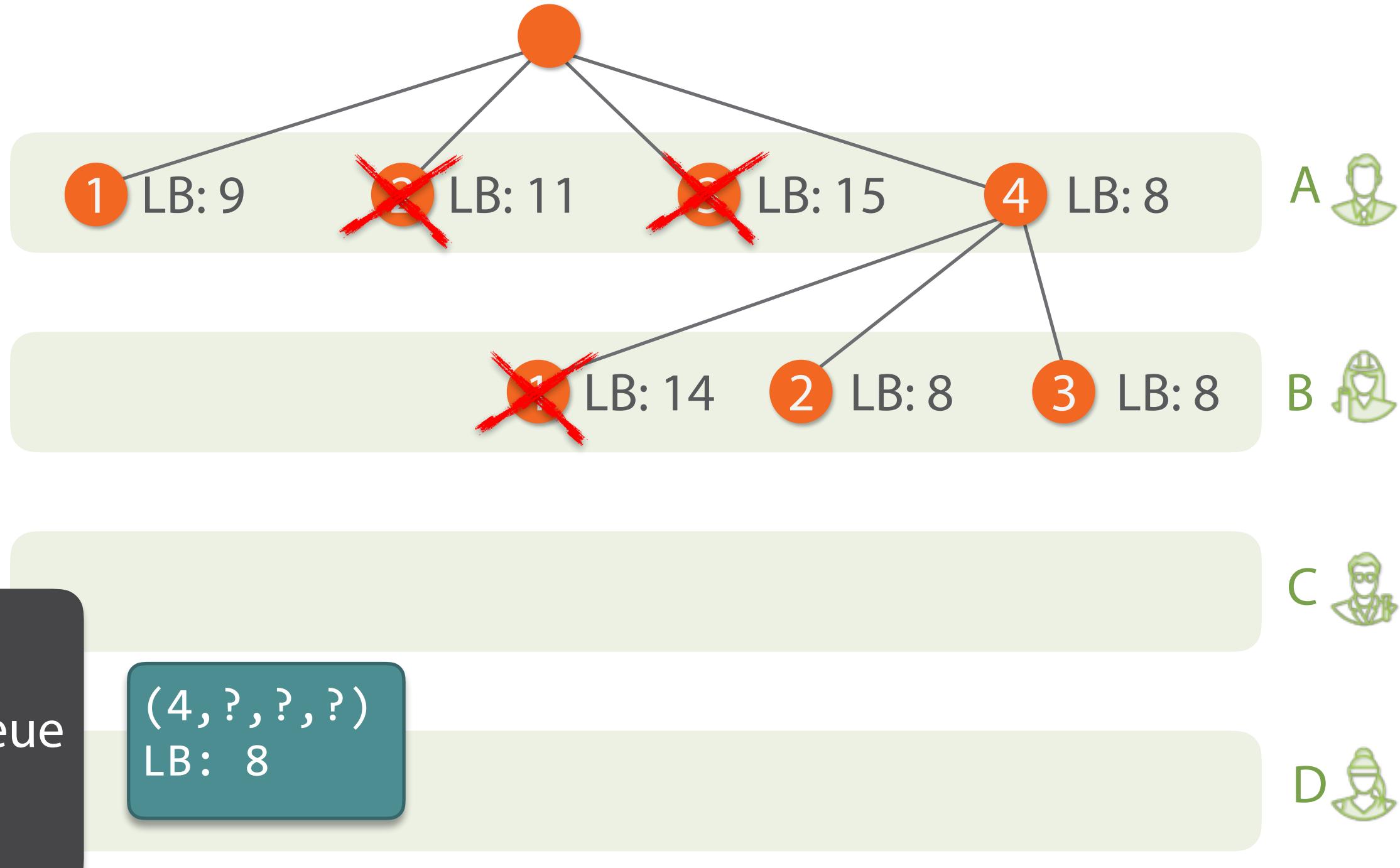


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	2
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2

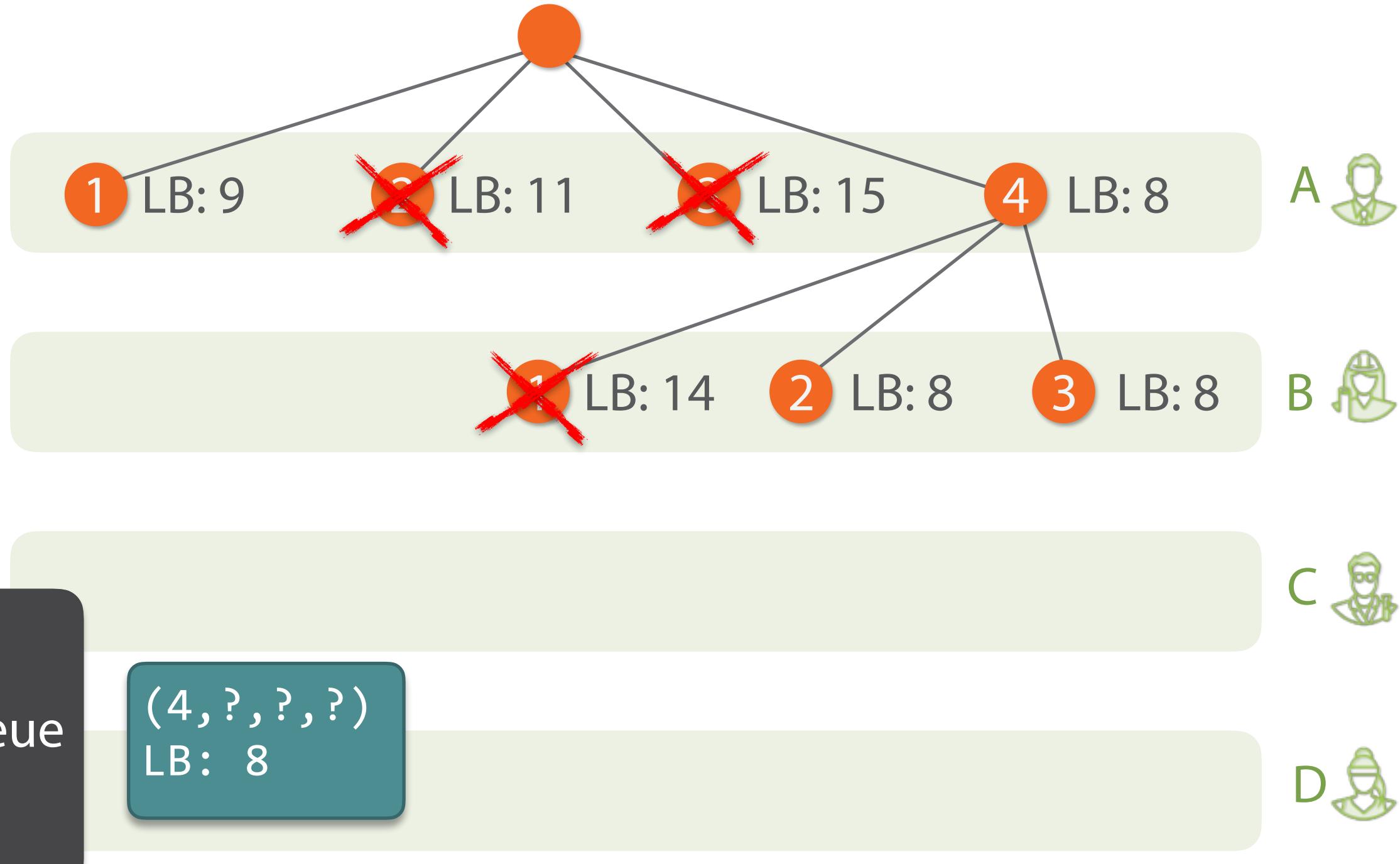


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	2
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2

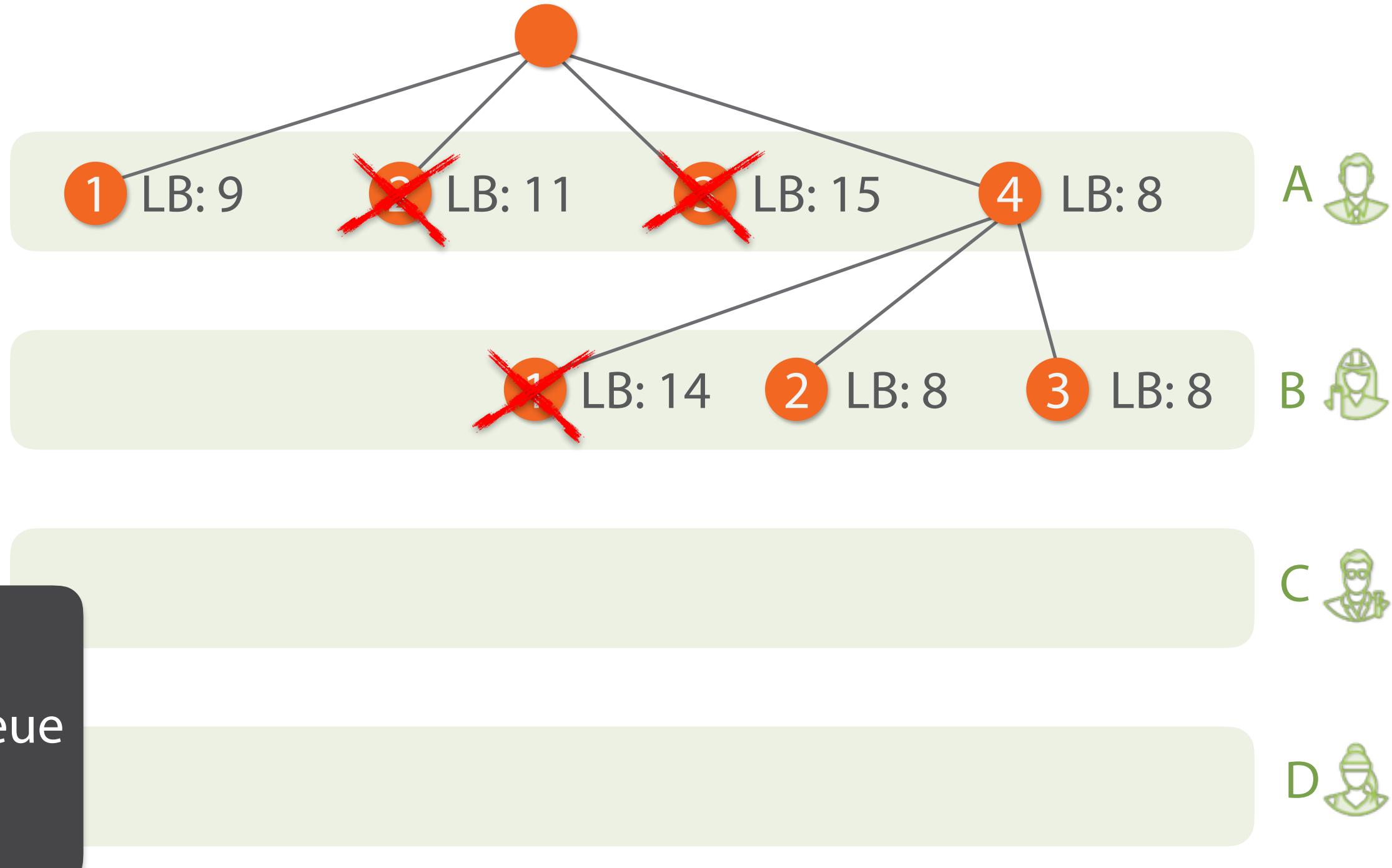


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	2
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2

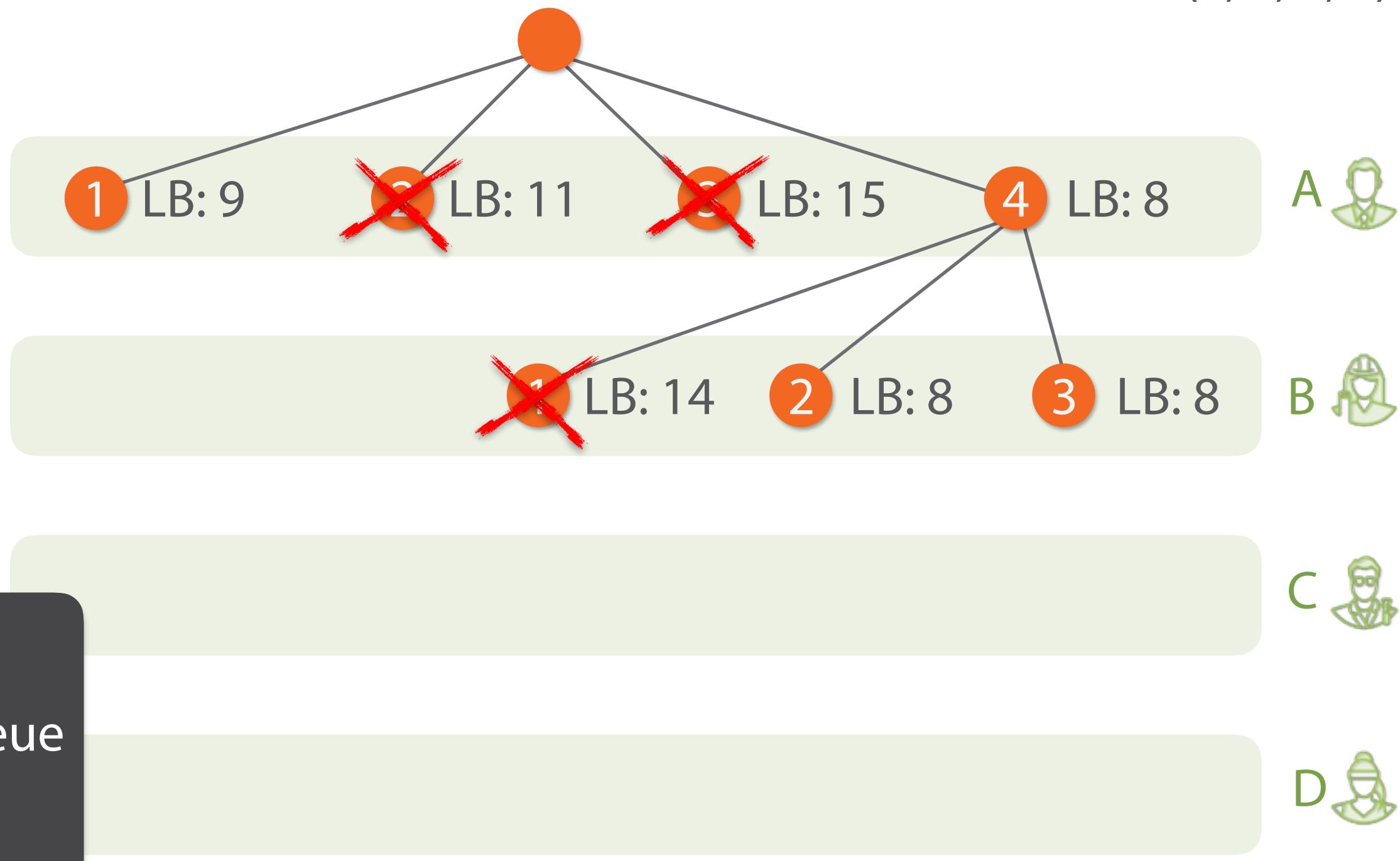


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	2
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2

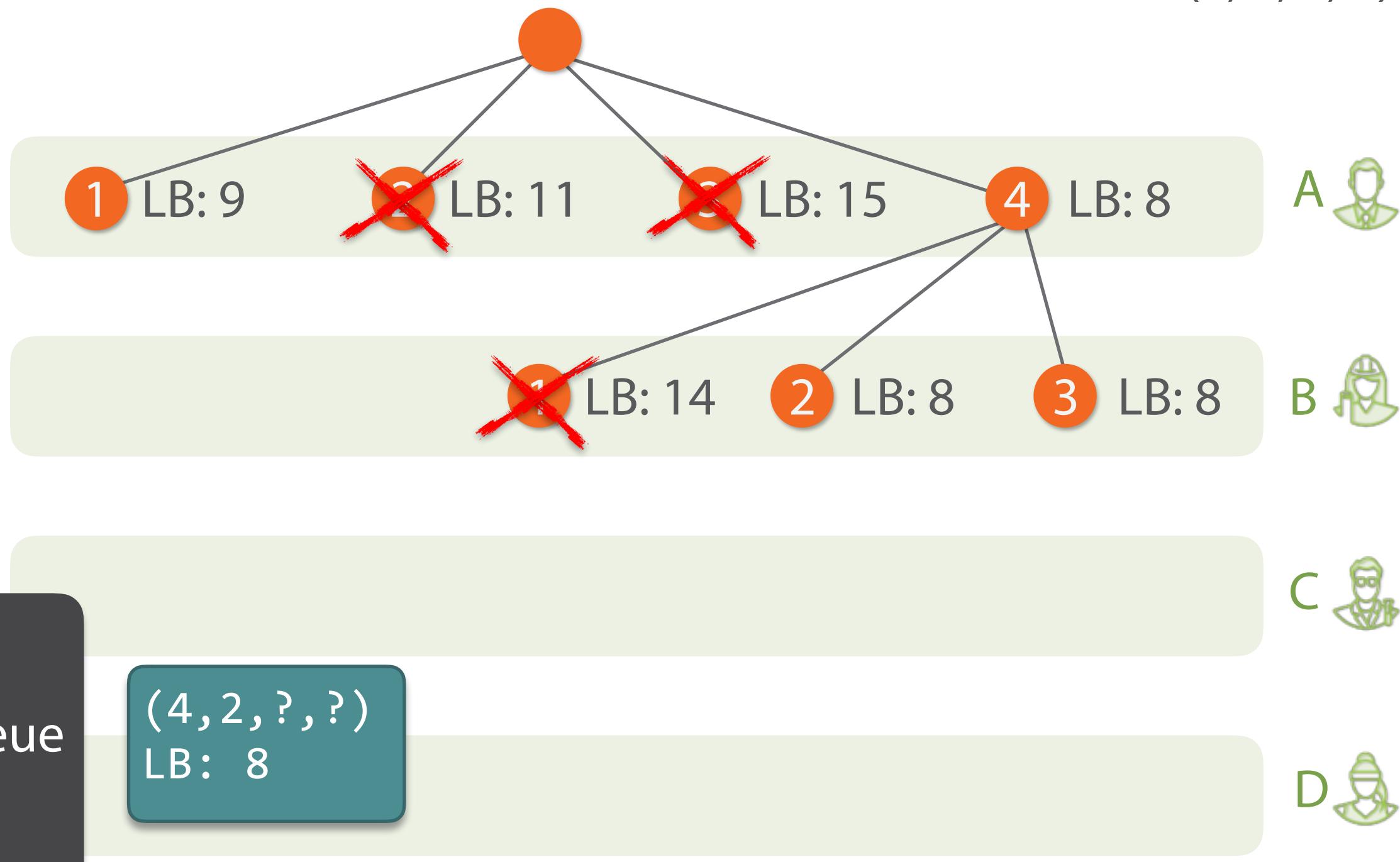


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	2
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2

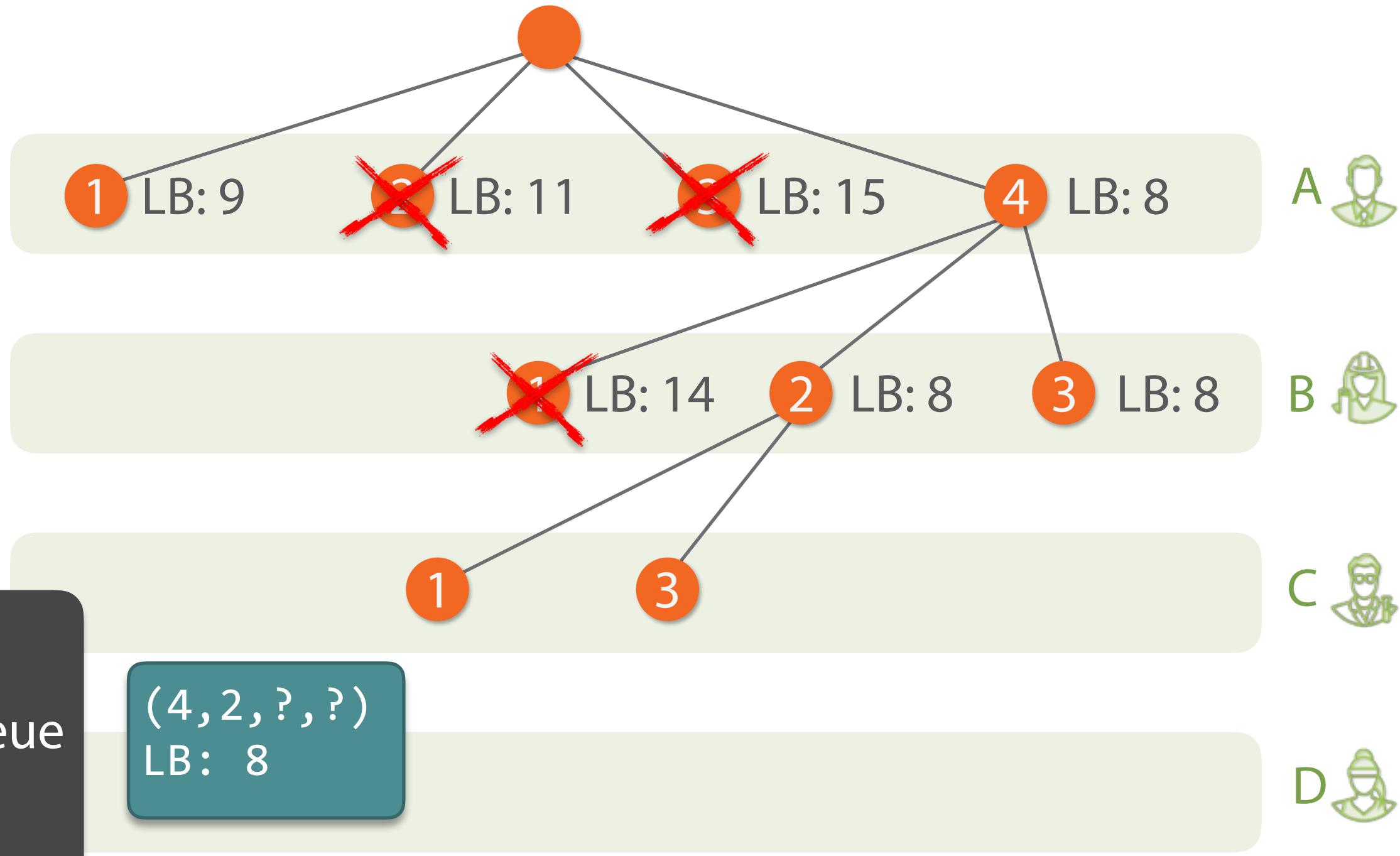


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

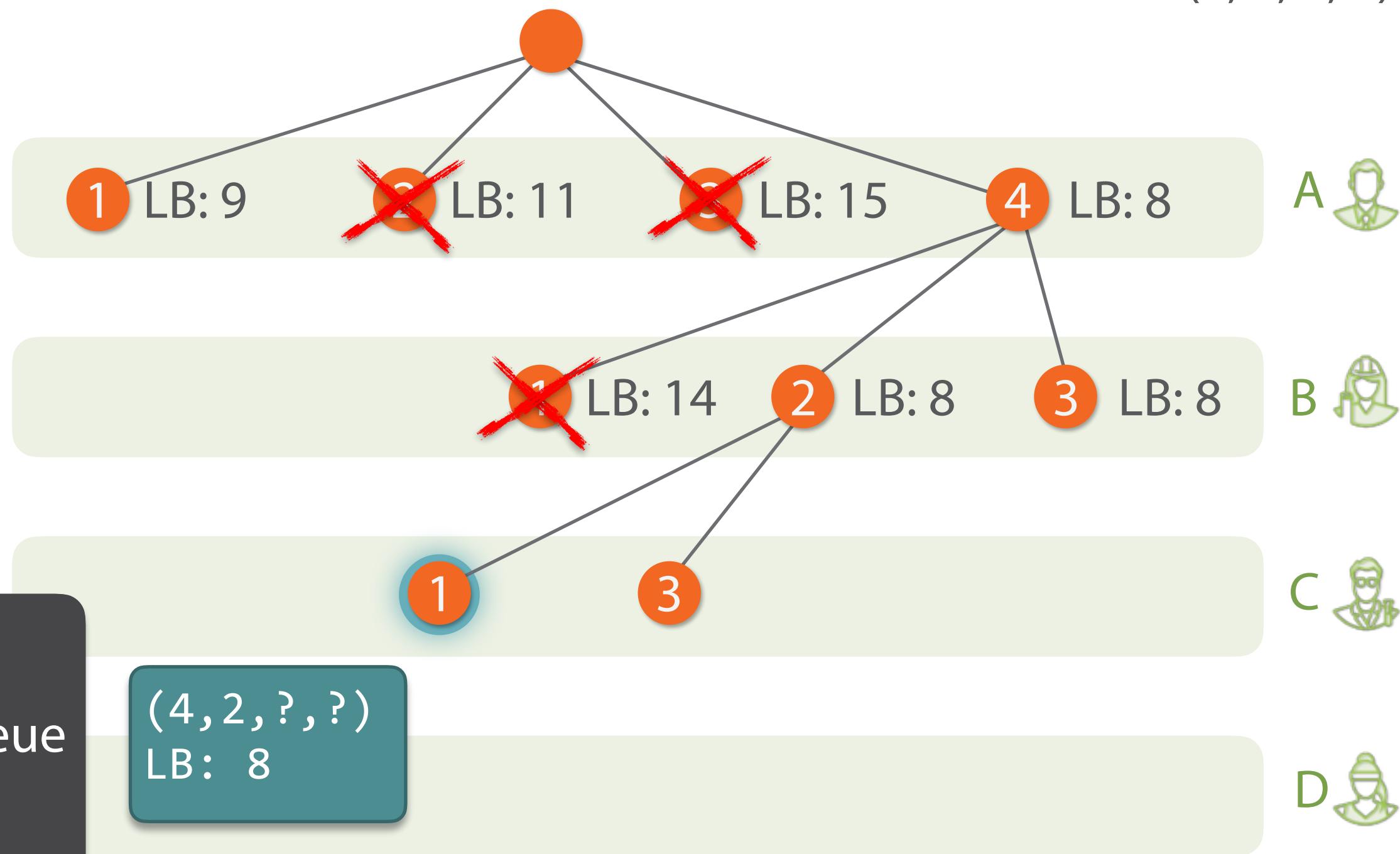
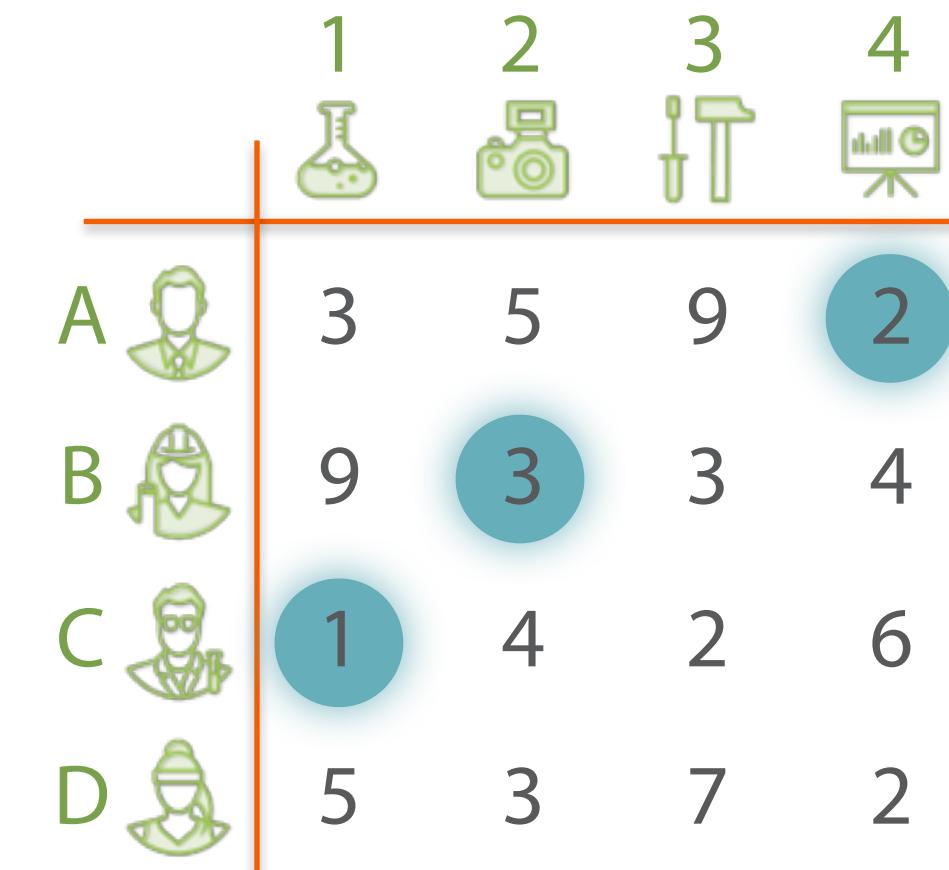
	1	2	3	4
1	3	5	9	4
2	9	3	7	2
3	1	4	2	6
4	5	3	7	2



# The Assignment Problem

# Best so far: 10

(1, 2, 3, 4)

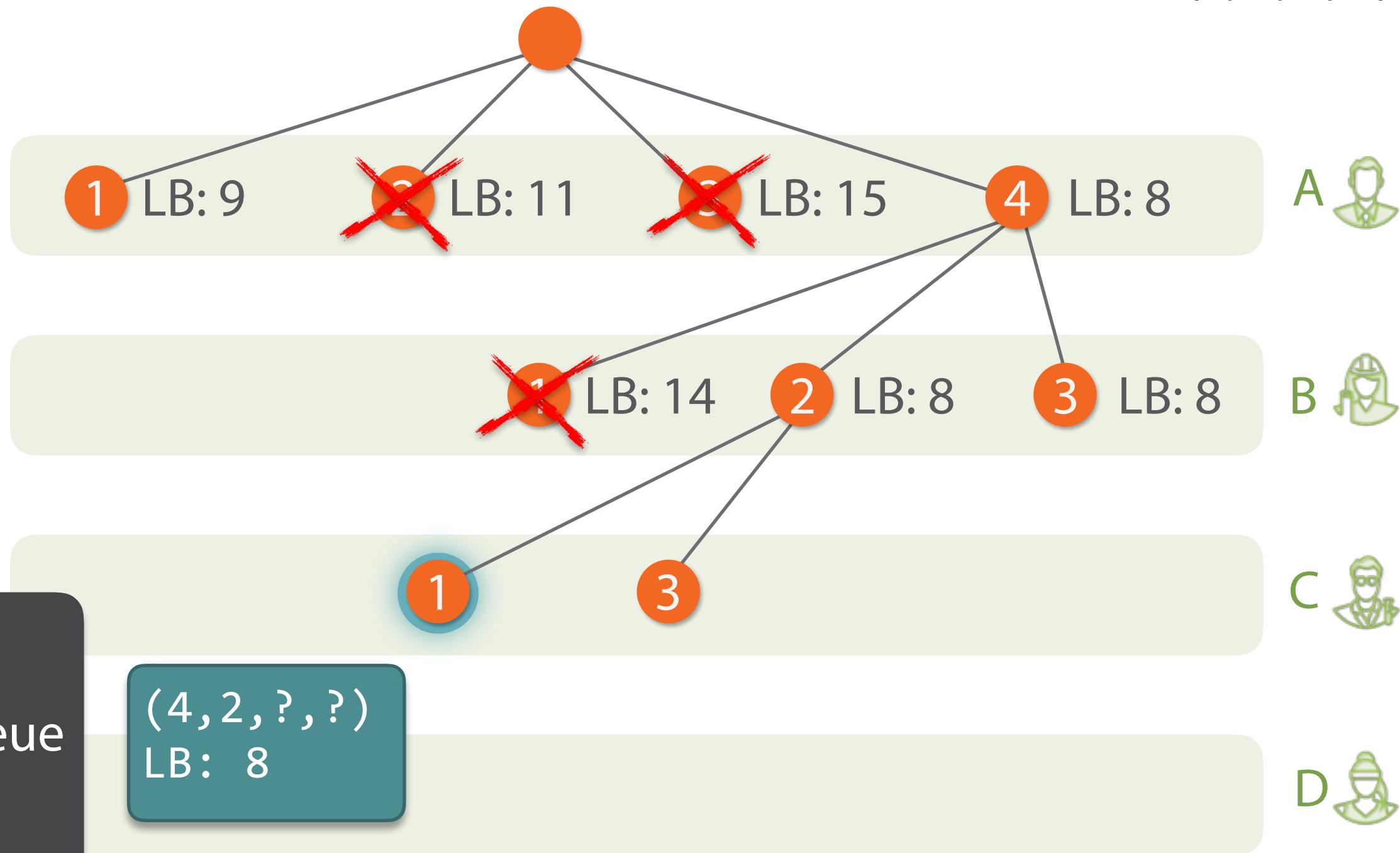


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	4
2	9	3	3	4
3	4	2	6	
4	5	3	7	2

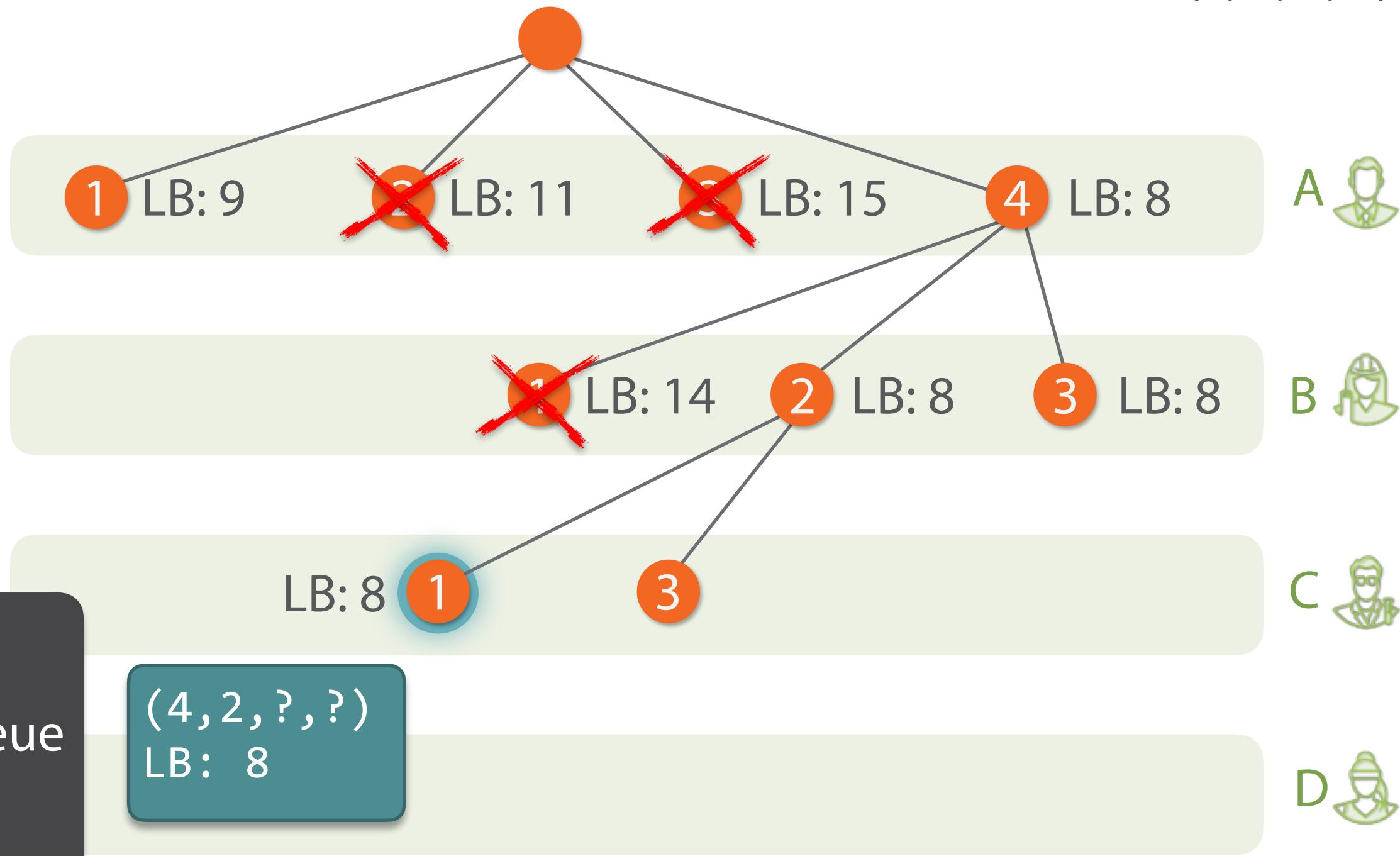


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	4
2	9	3	3	4
3	4	2	6	
4	5	3	7	2

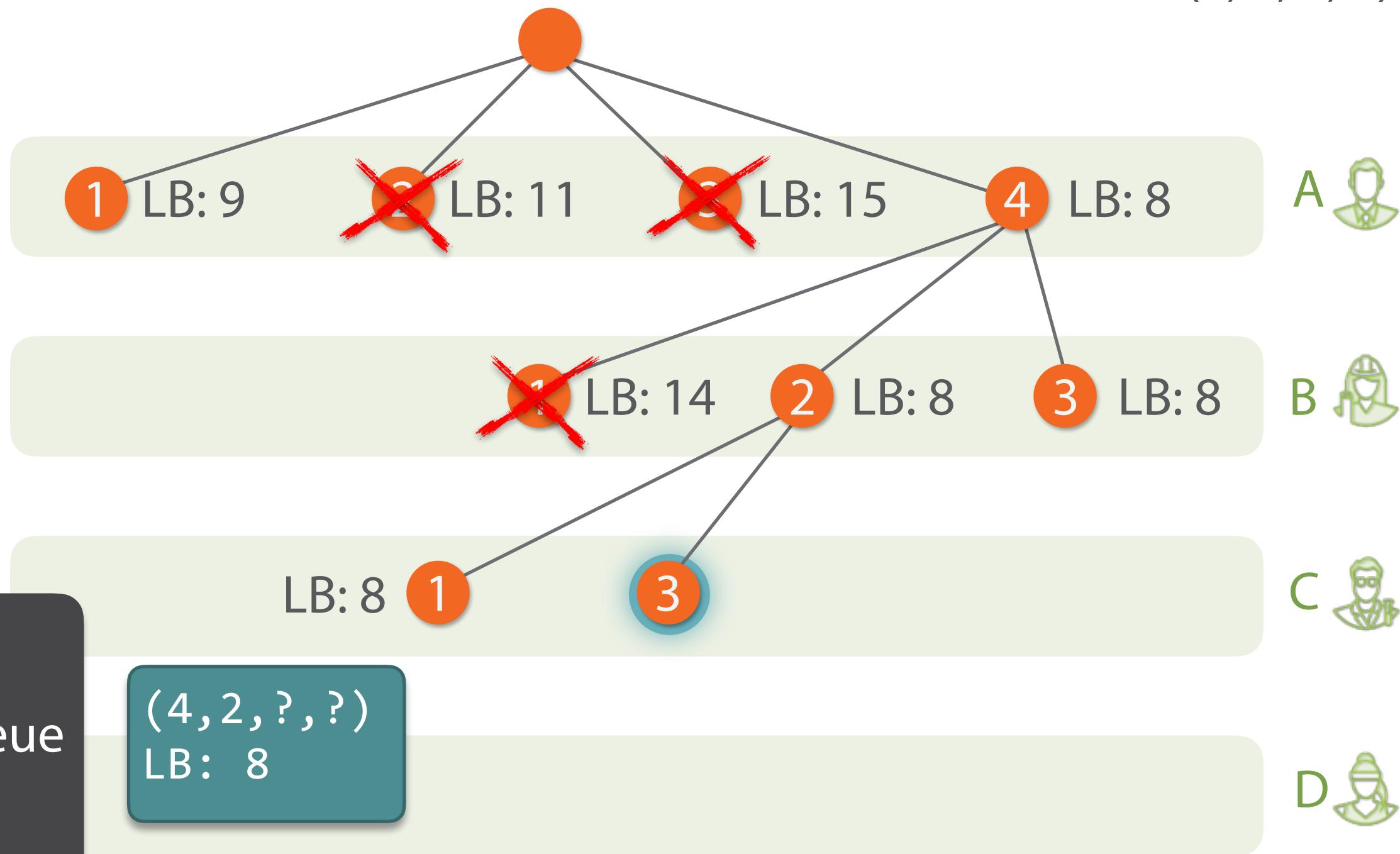


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	4
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2

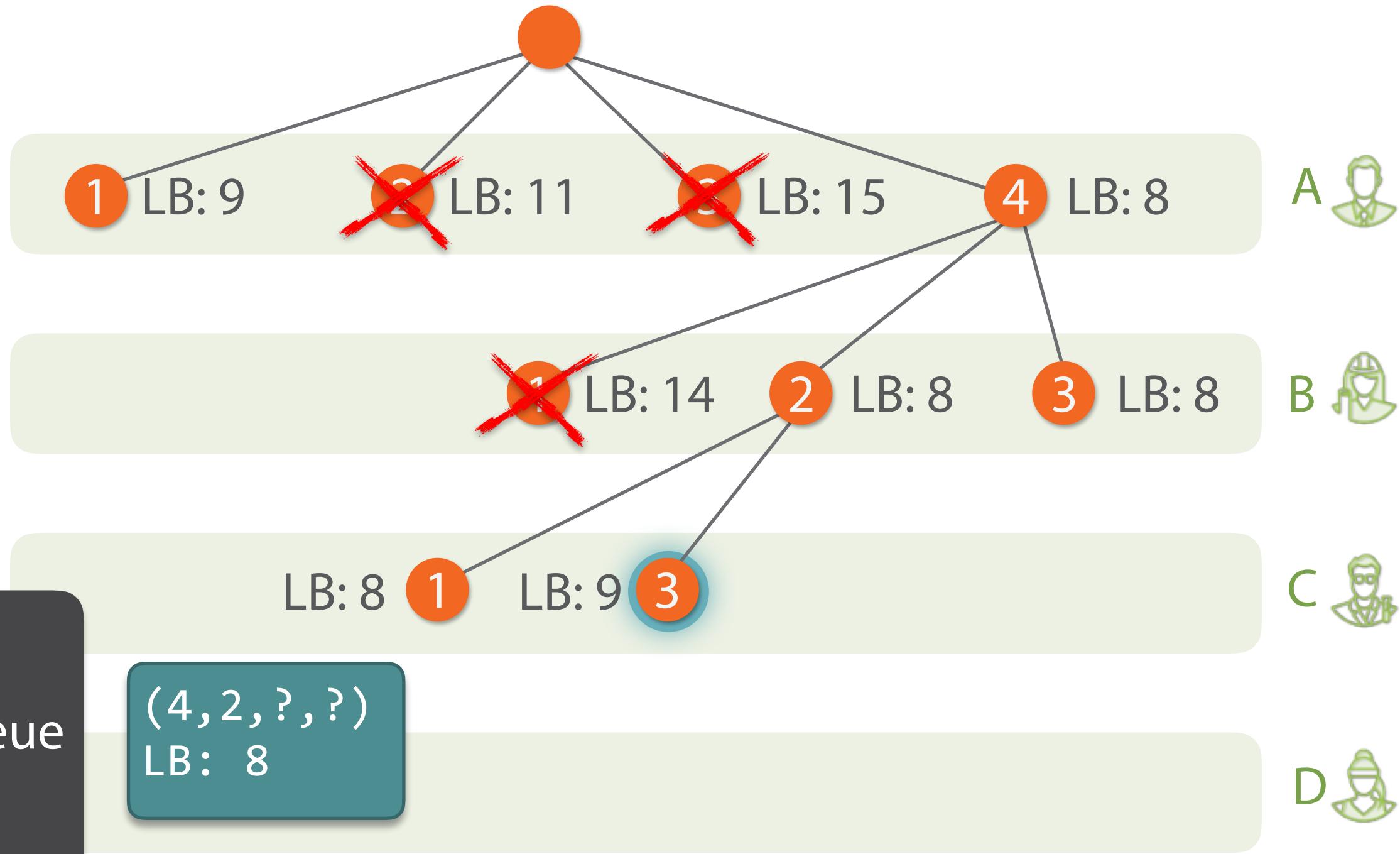


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
A	3	5	9	2
B	9	3	4	
C	1	4	2	6
D	5	3	7	2

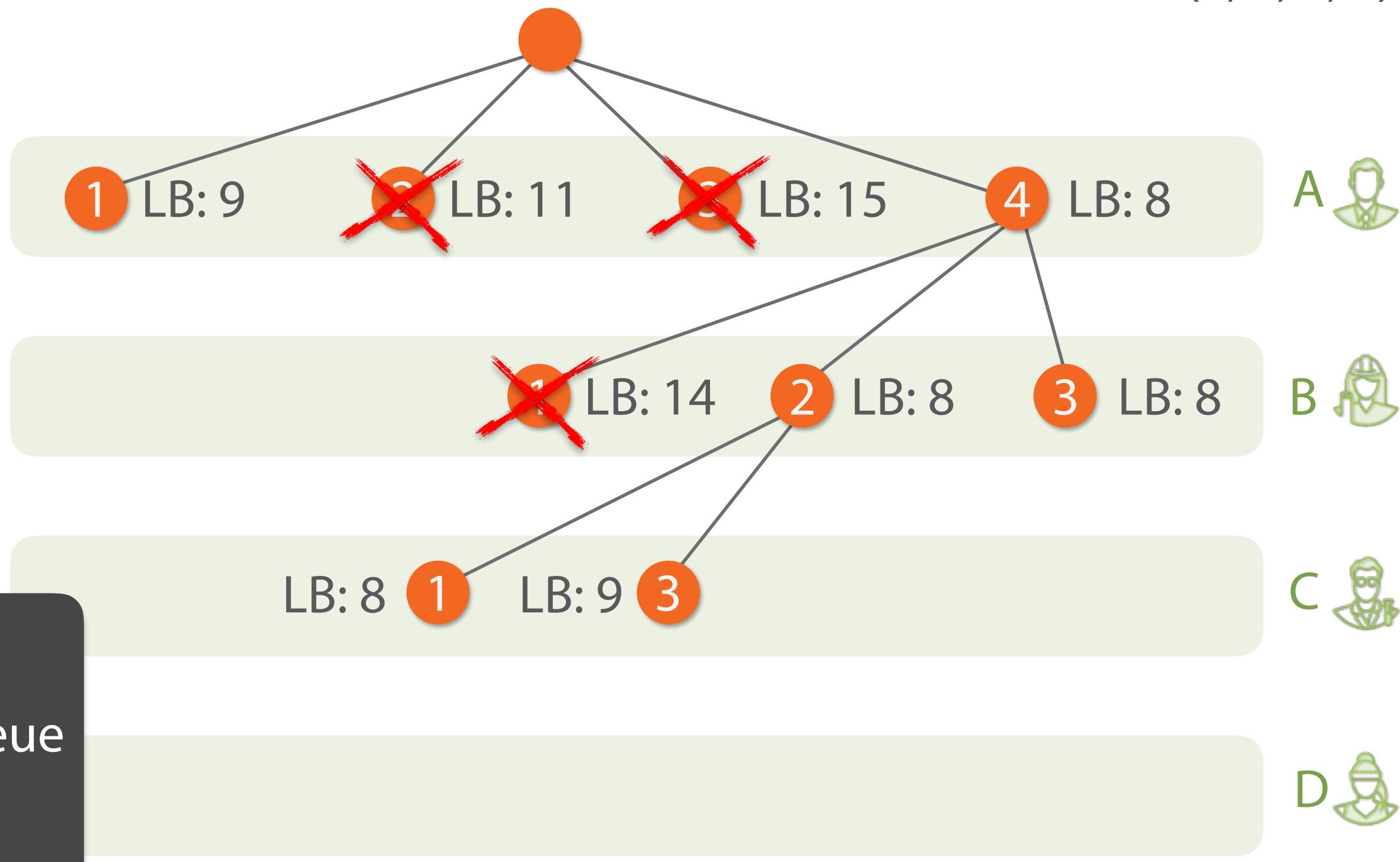


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	2
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2

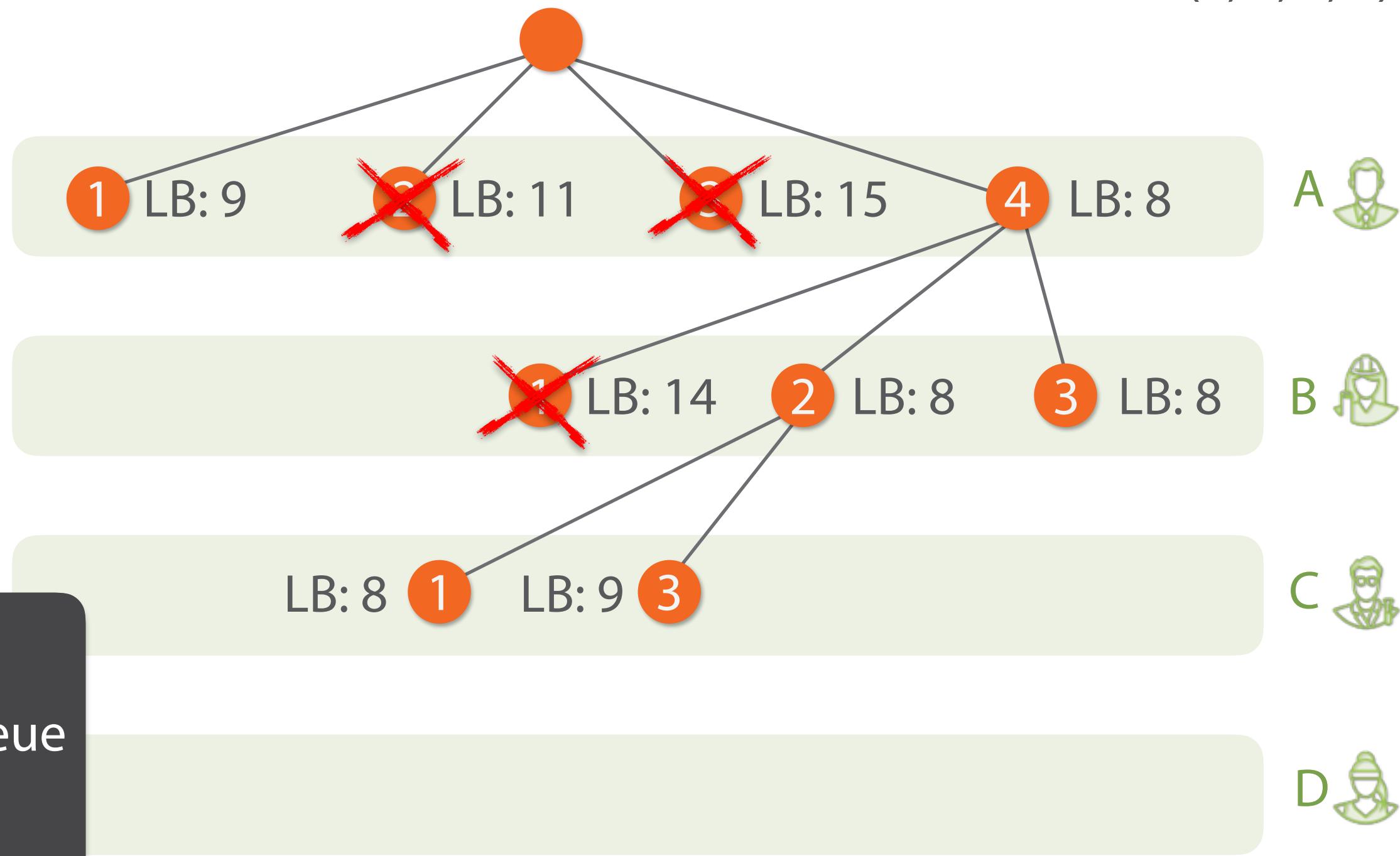


# The Assignment Problem

# Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
A	3	5	9	2
B	9	3	3	4
C	1	4	2	6
D	5	3	7	2

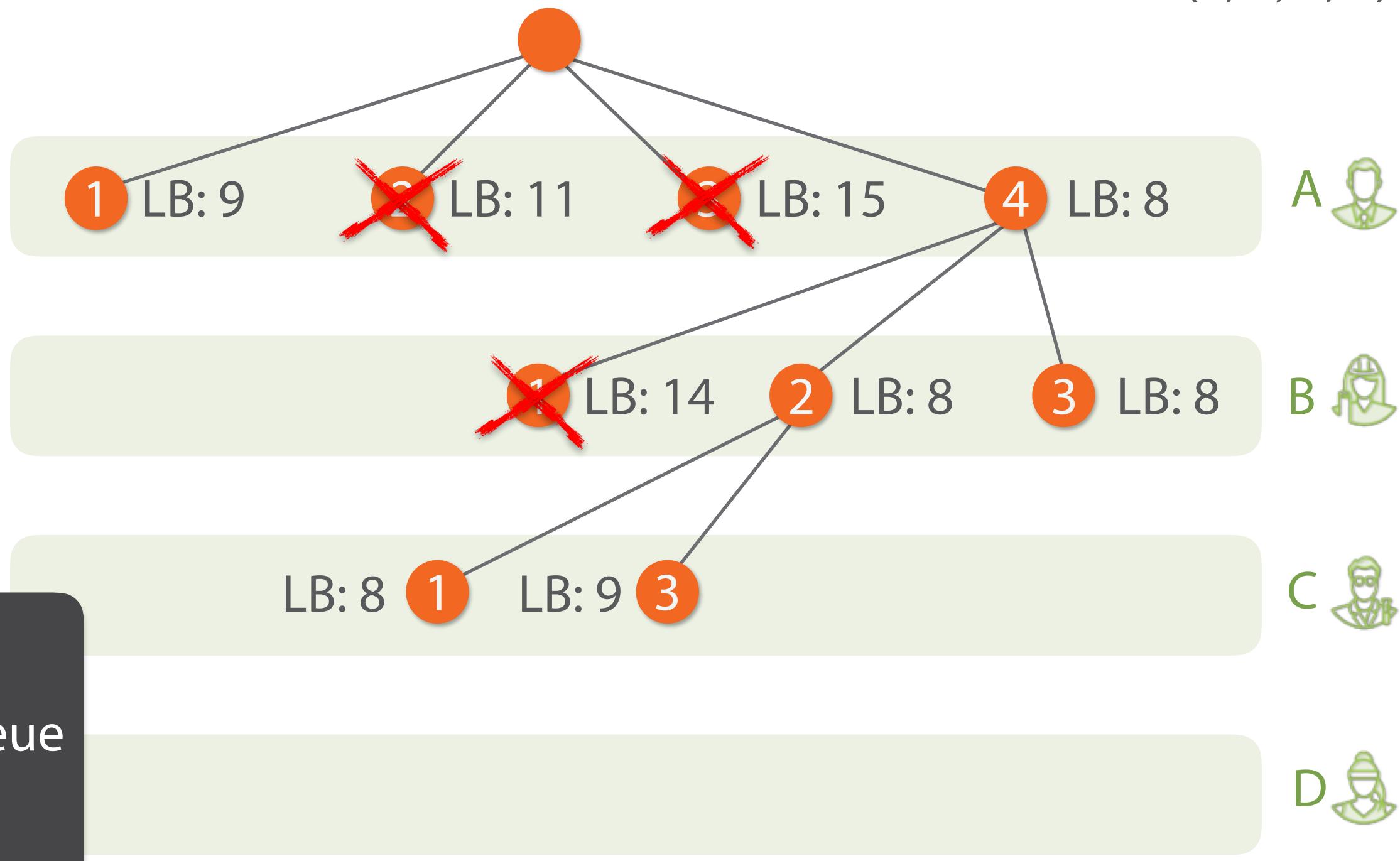


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

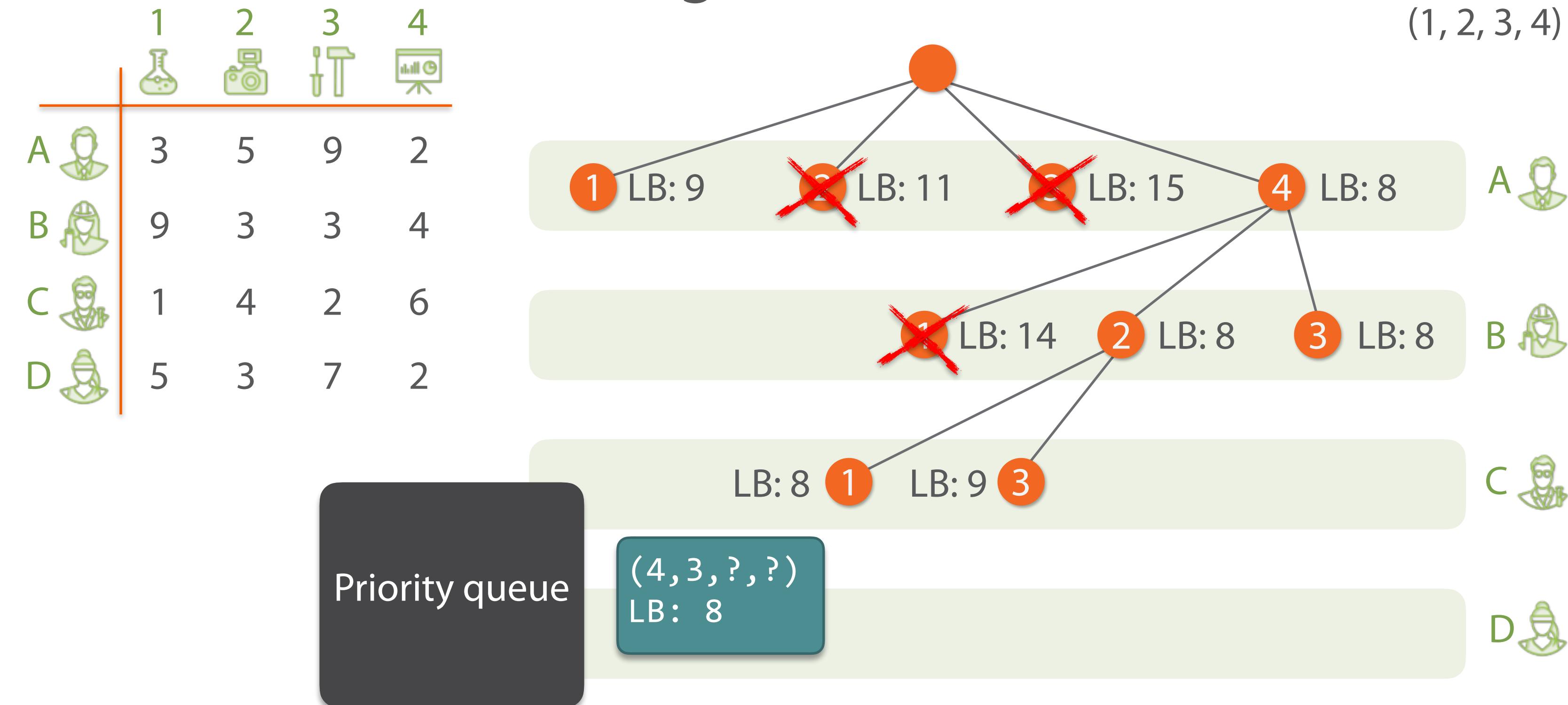
	1	2	3	4
1	3	5	9	2
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2



# The Assignment Problem

# Best so far: 10

(1, 2, 3, 4)

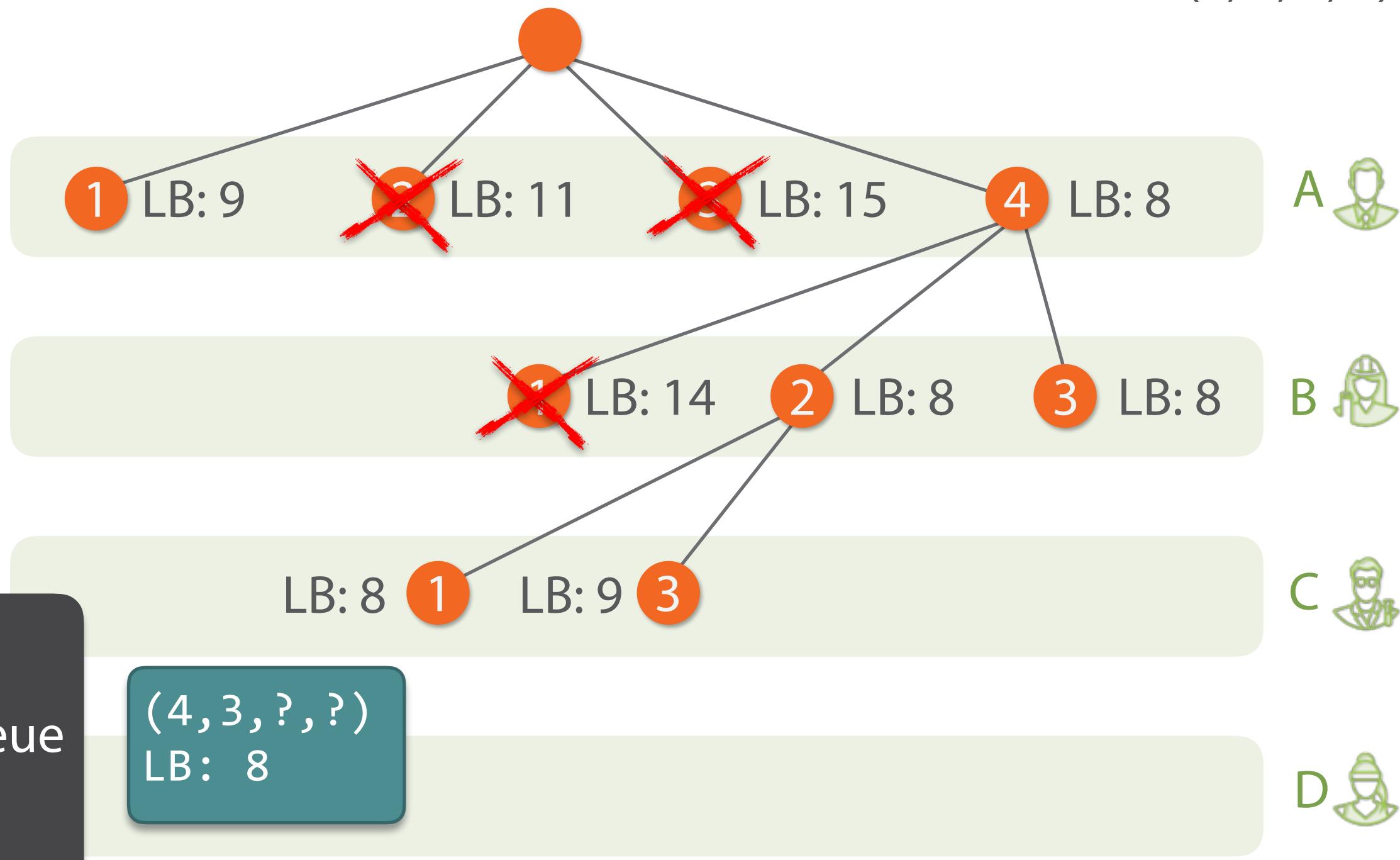


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	2
2	9	3	7	4
3	1	4	2	6
4	5	3	7	2

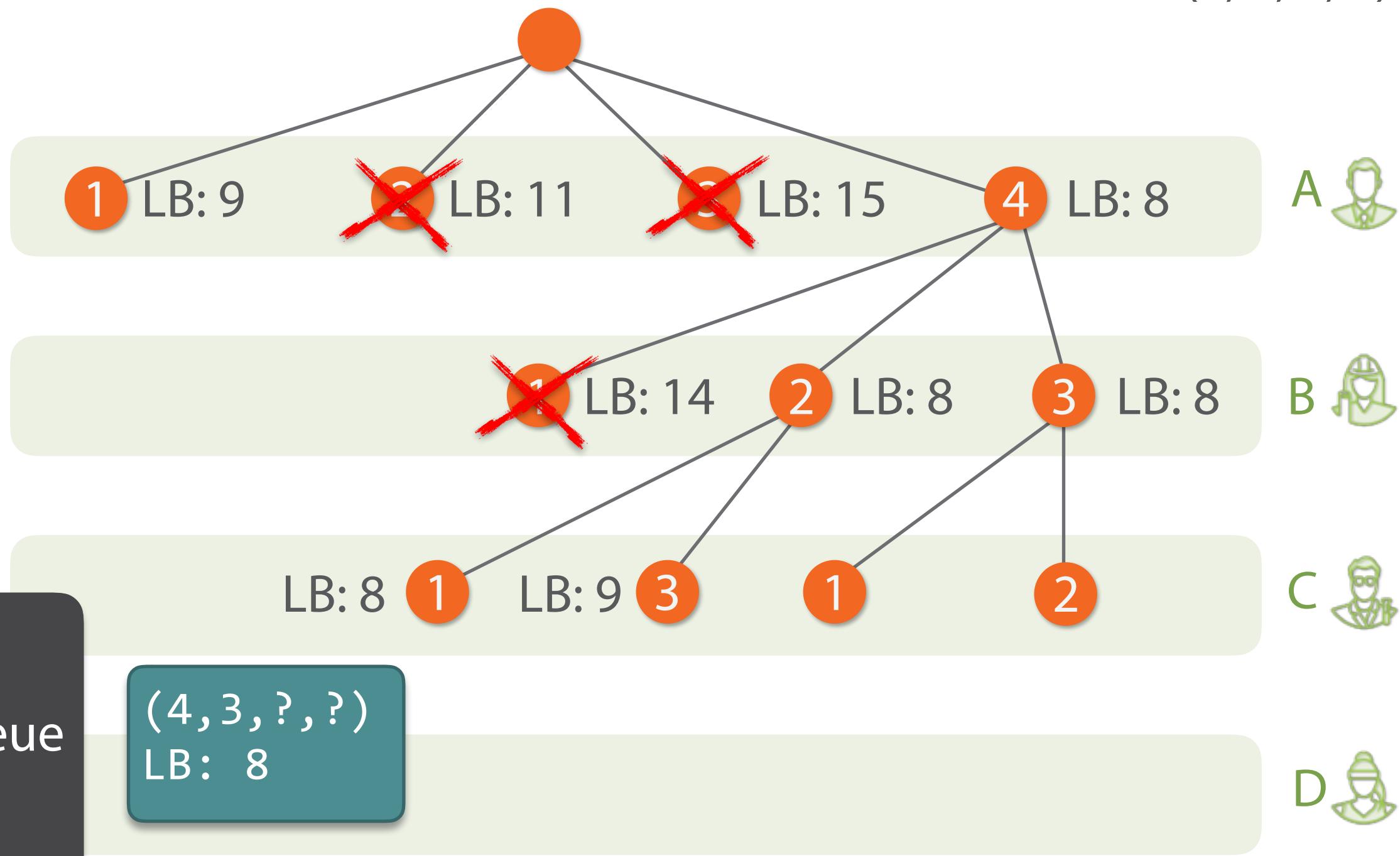


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	2
2	9	3	7	4
3	1	4	2	6
4	5	3	7	2

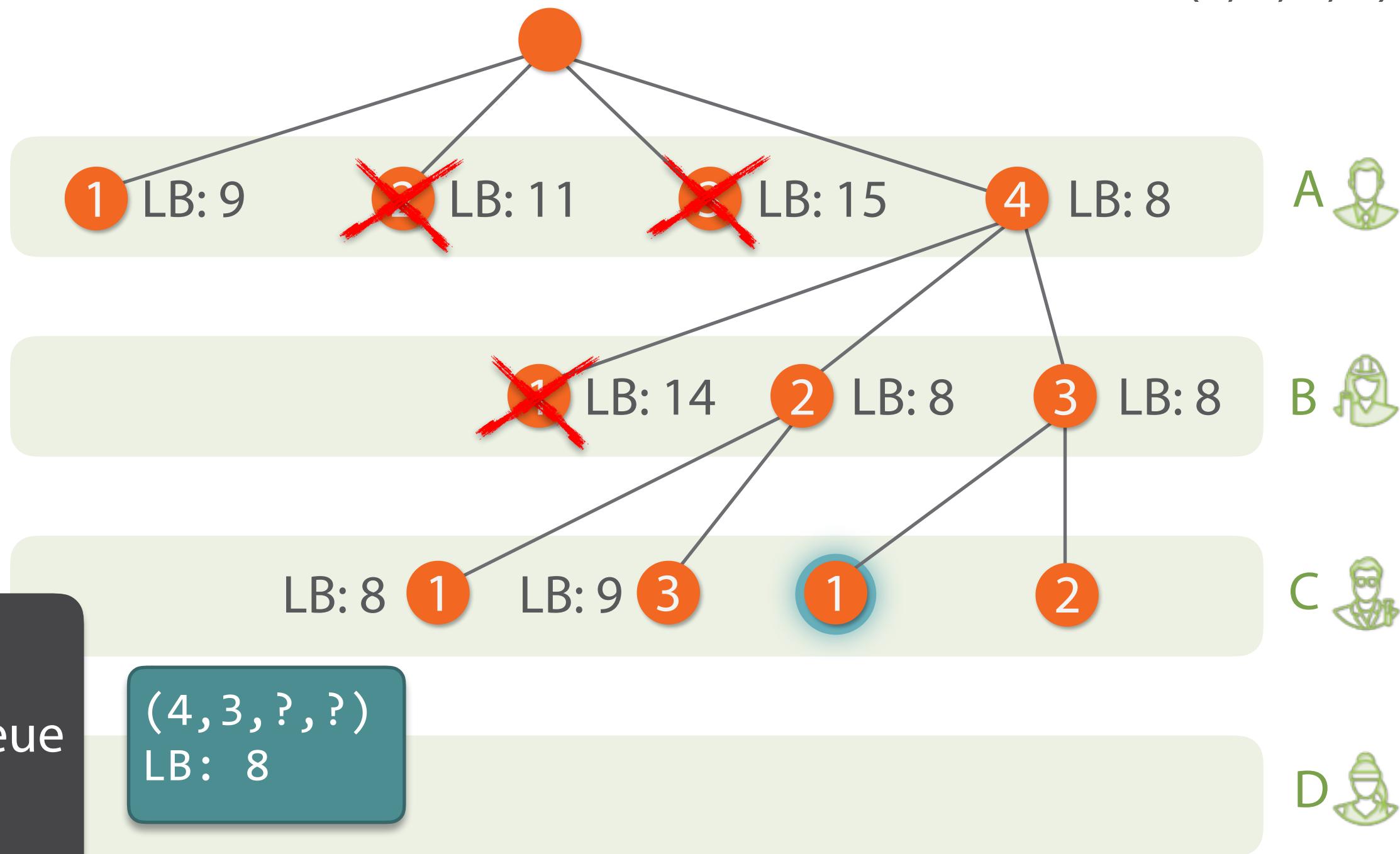


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	2
2	9	3	4	3
3	4	2	6	1
4	5	3	7	2

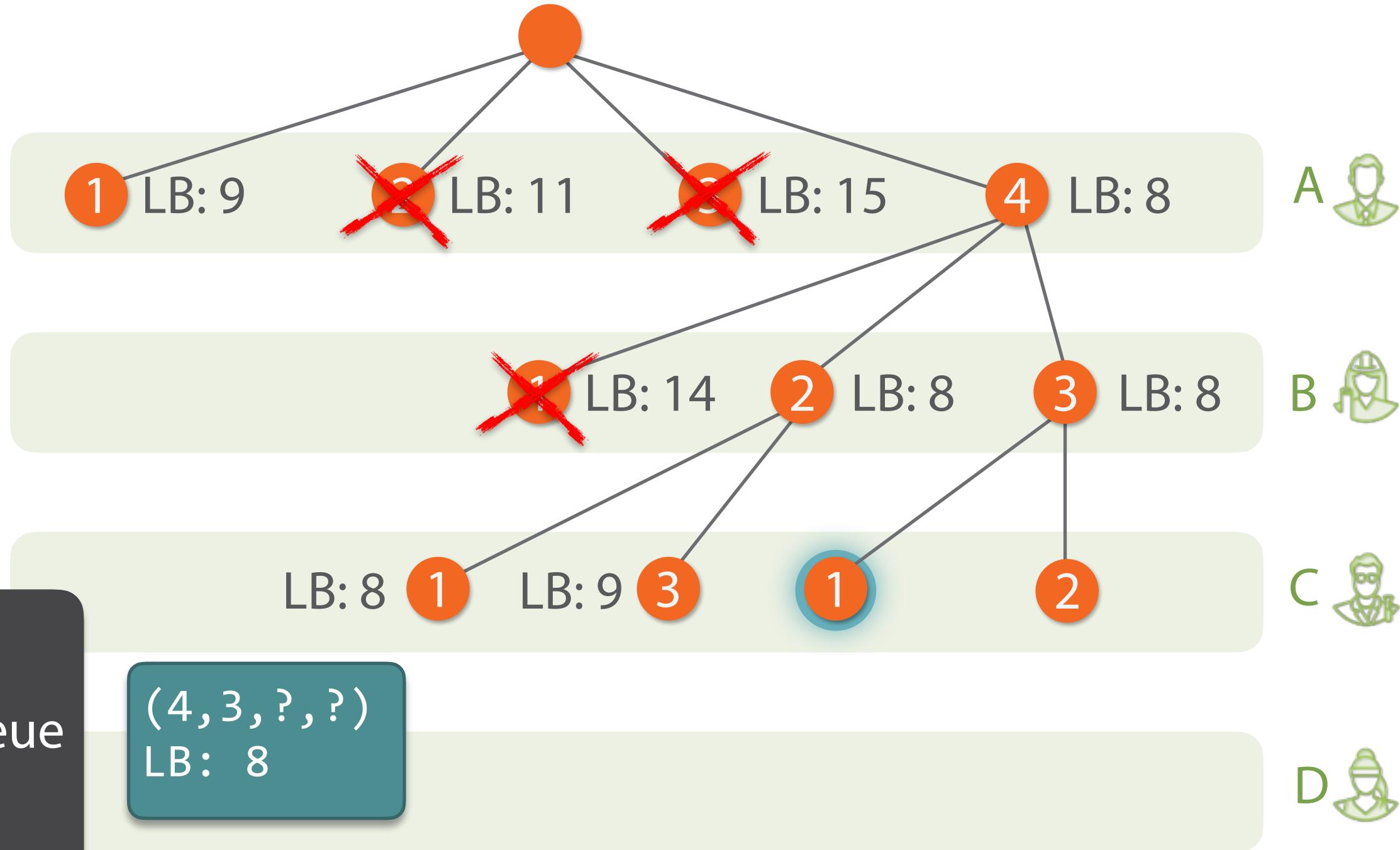


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	2
2	9	3	4	3
3	4	2	6	1
4	5	3	7	2

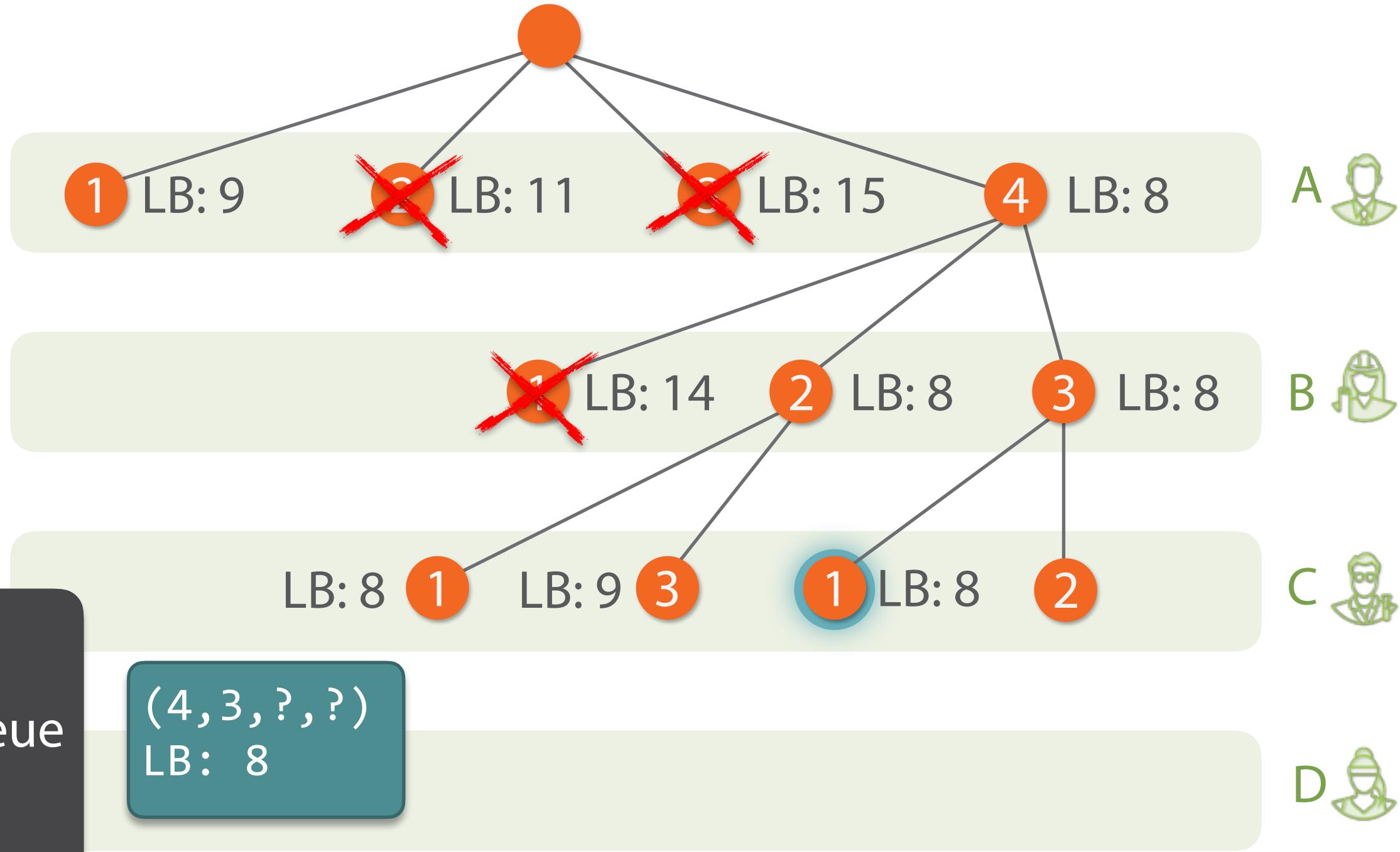


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	2
2	9	3	4	3
3	4	2	6	1
4	5	3	7	2

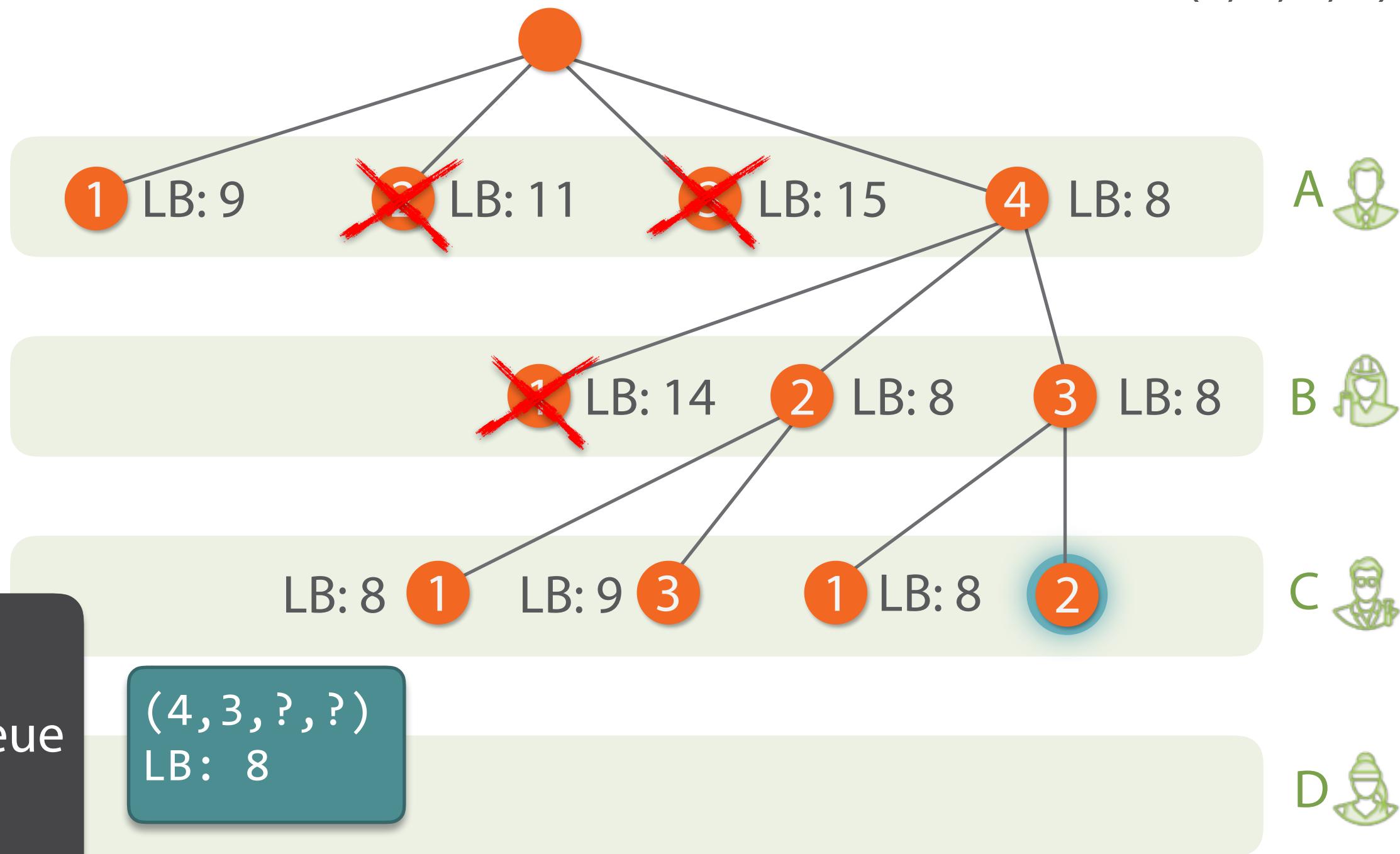


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	2
2	9	3	4	3
3	1	2	6	4
4	5	3	7	2

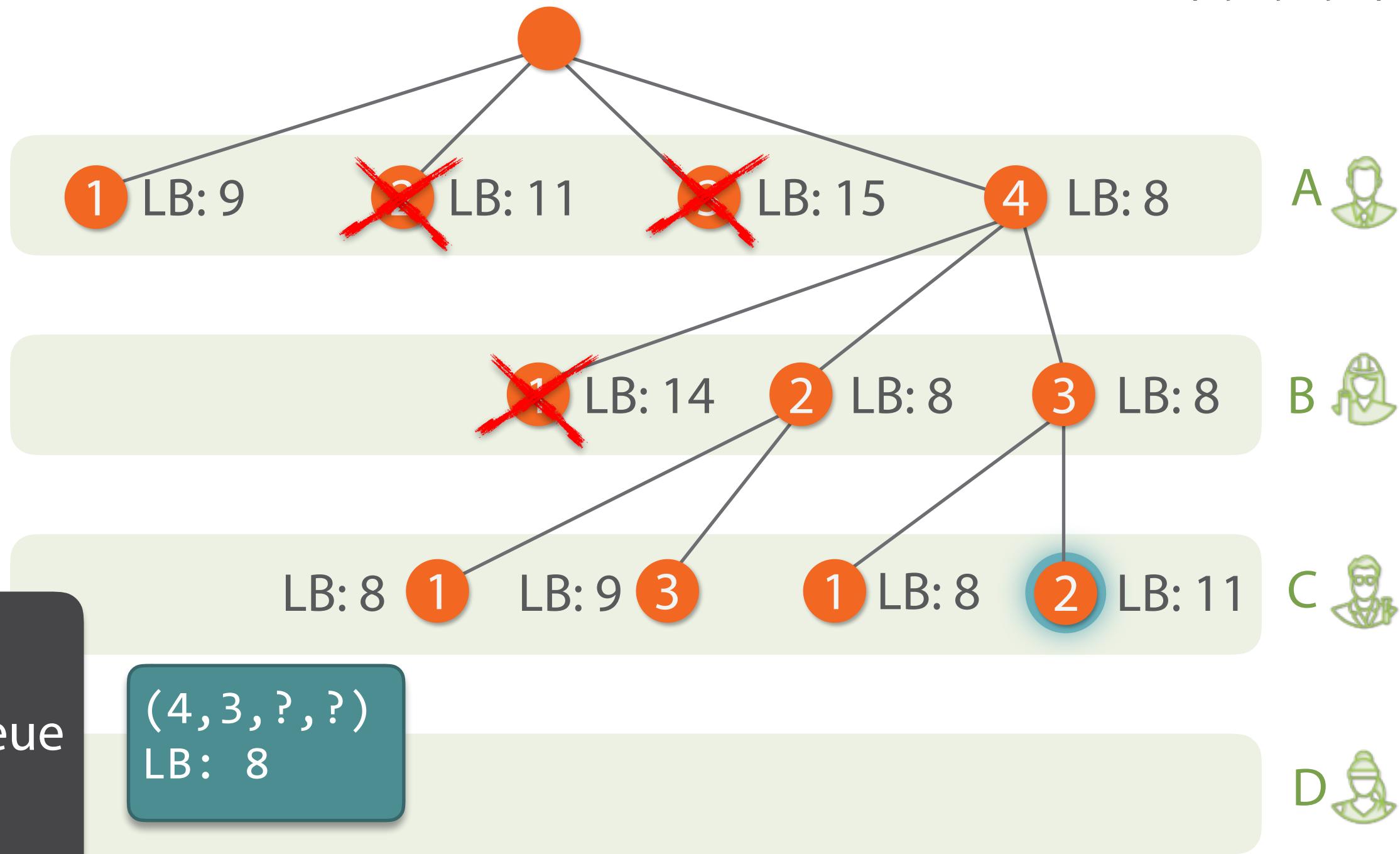


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	2
2	9	3	4	
3	1	4	2	6
4	5	3	7	2

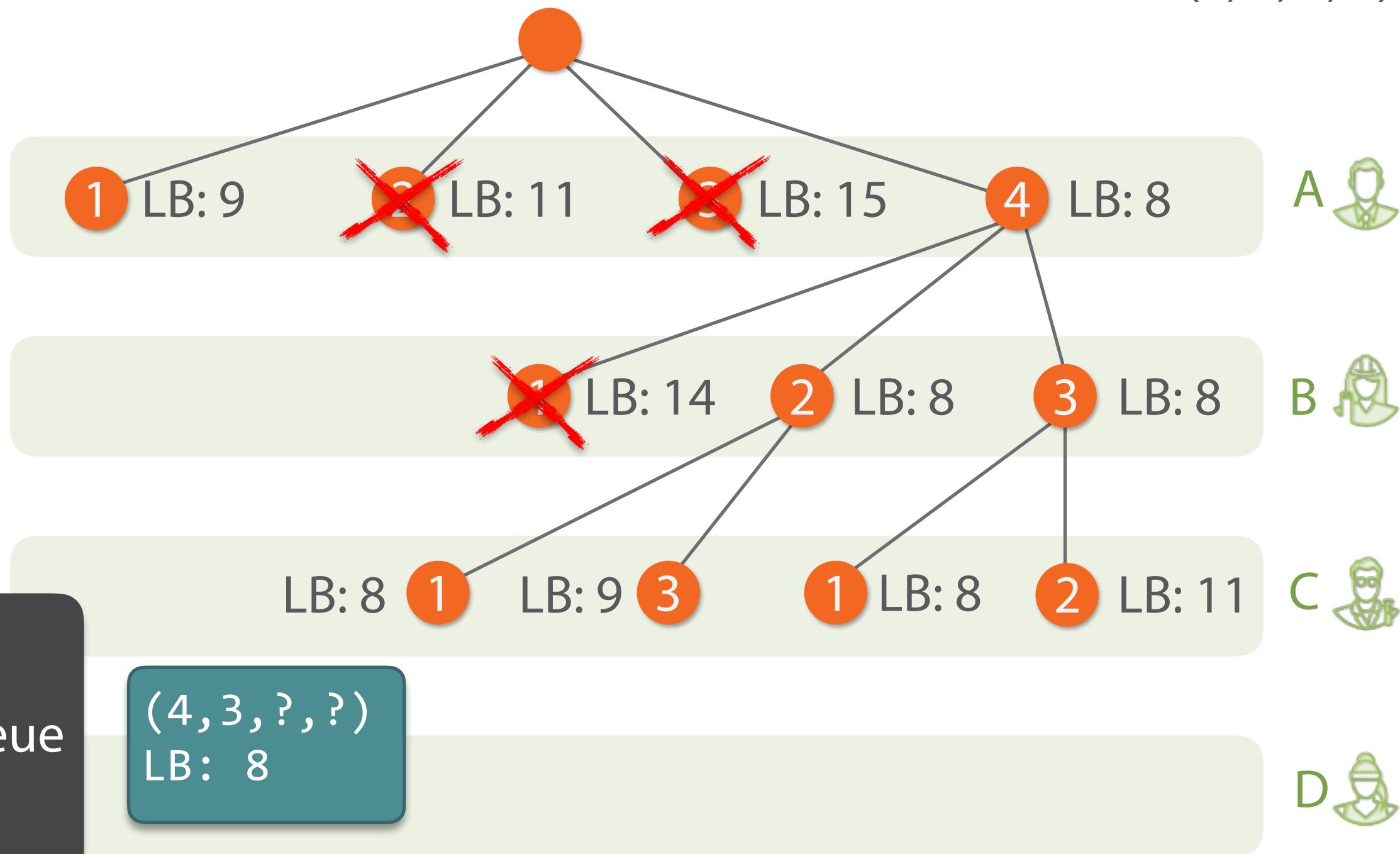


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	2
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2

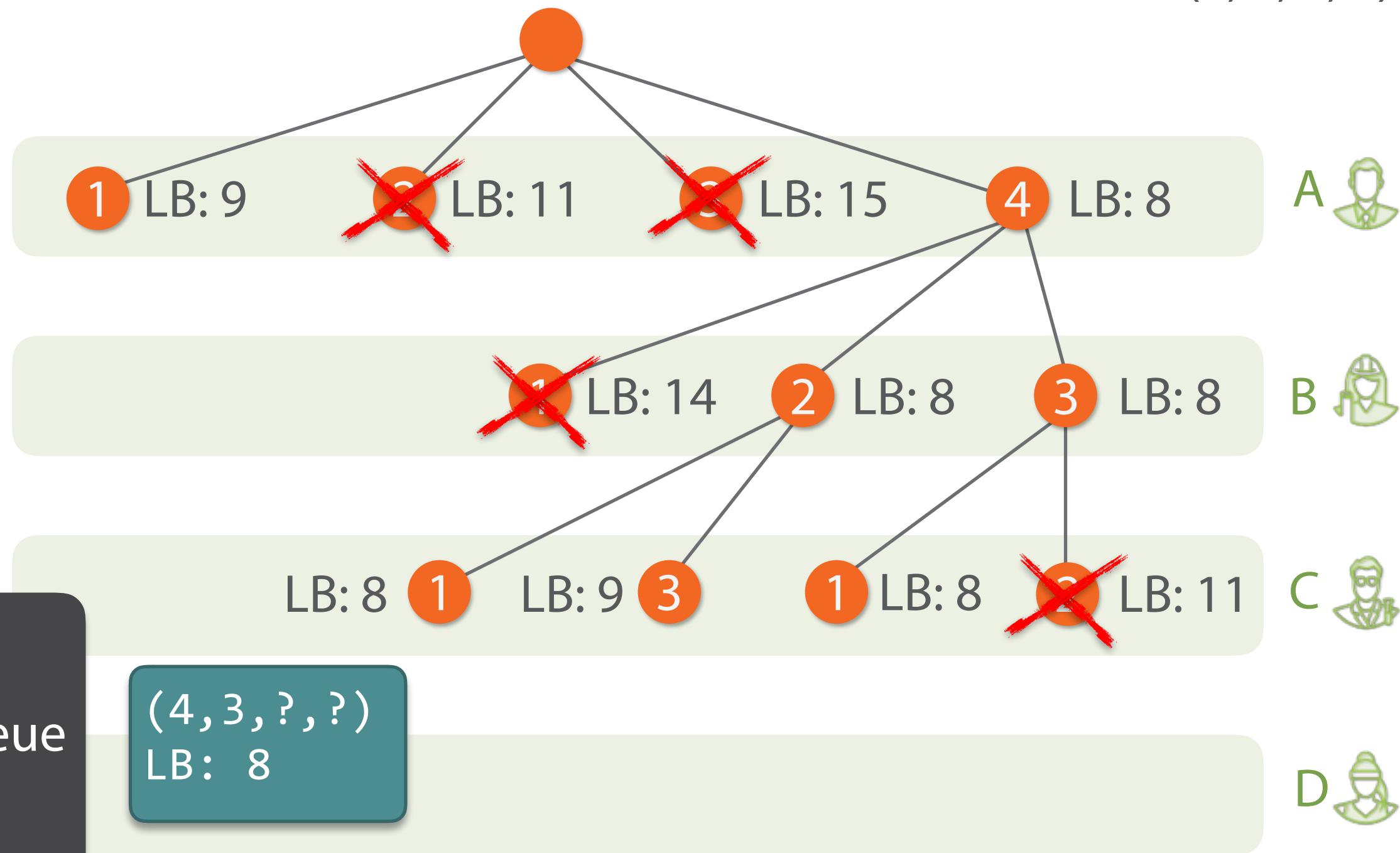


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

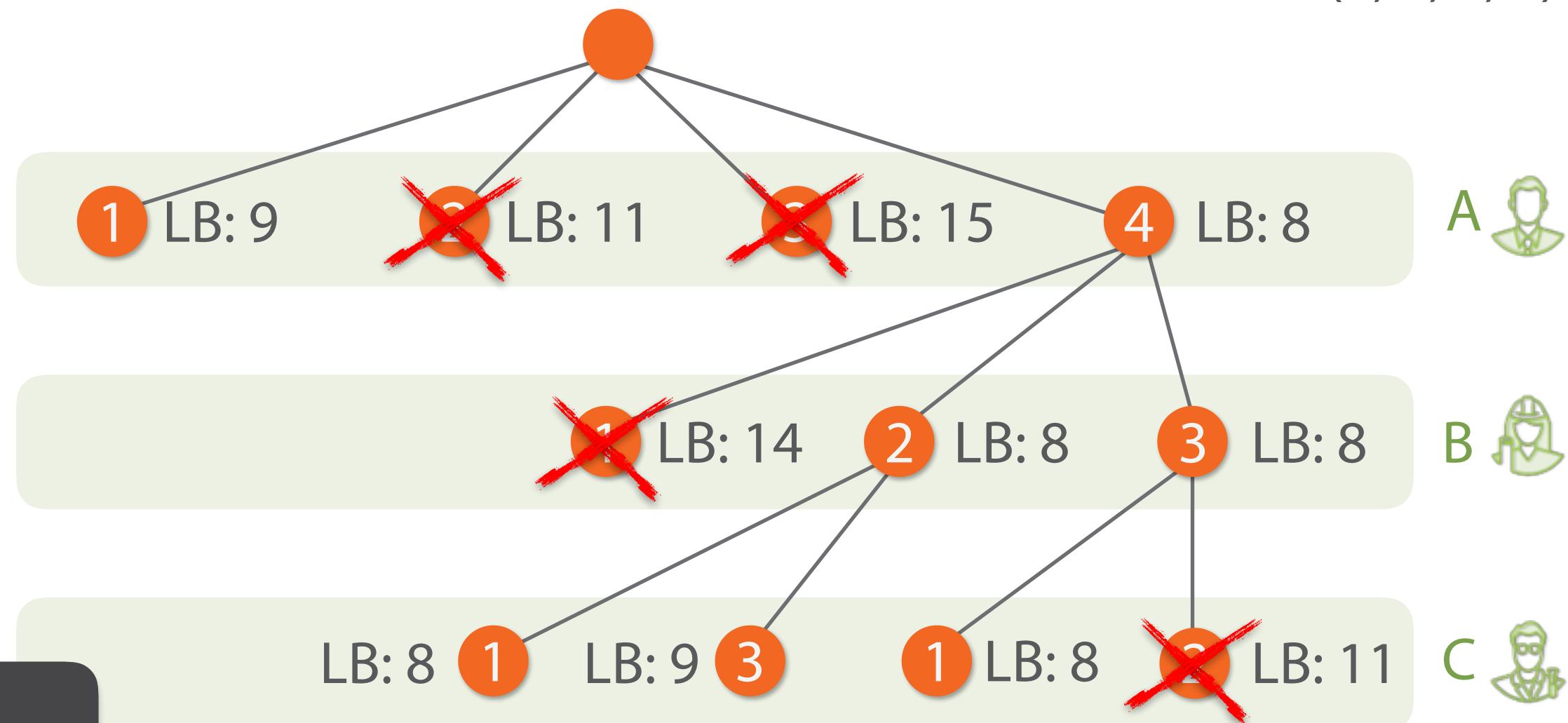
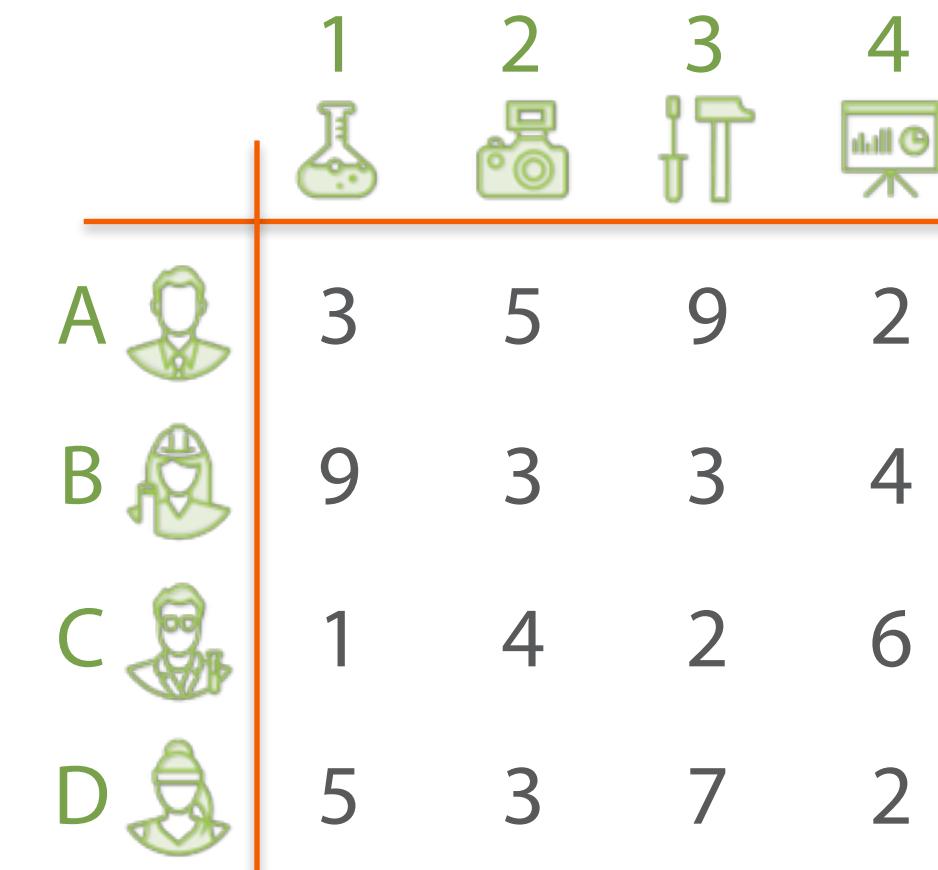
	1	2	3	4
1	3	5	9	2
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2



# The Assignment Problem

# Best so far: 10

(1, 2, 3, 4)



$$(4, 3, 1, ?)$$

LB: 8

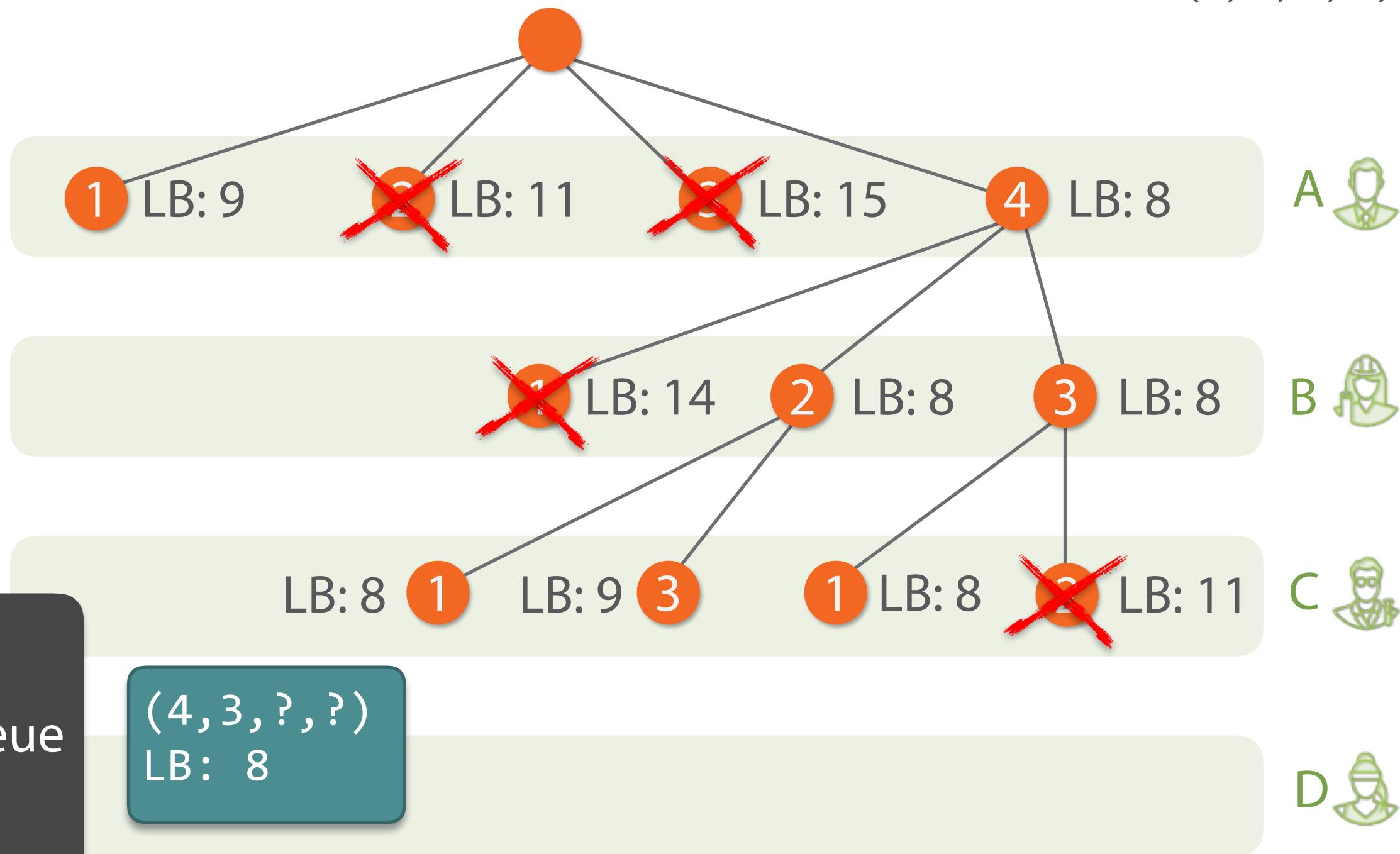
# Priority queue

# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	2
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2

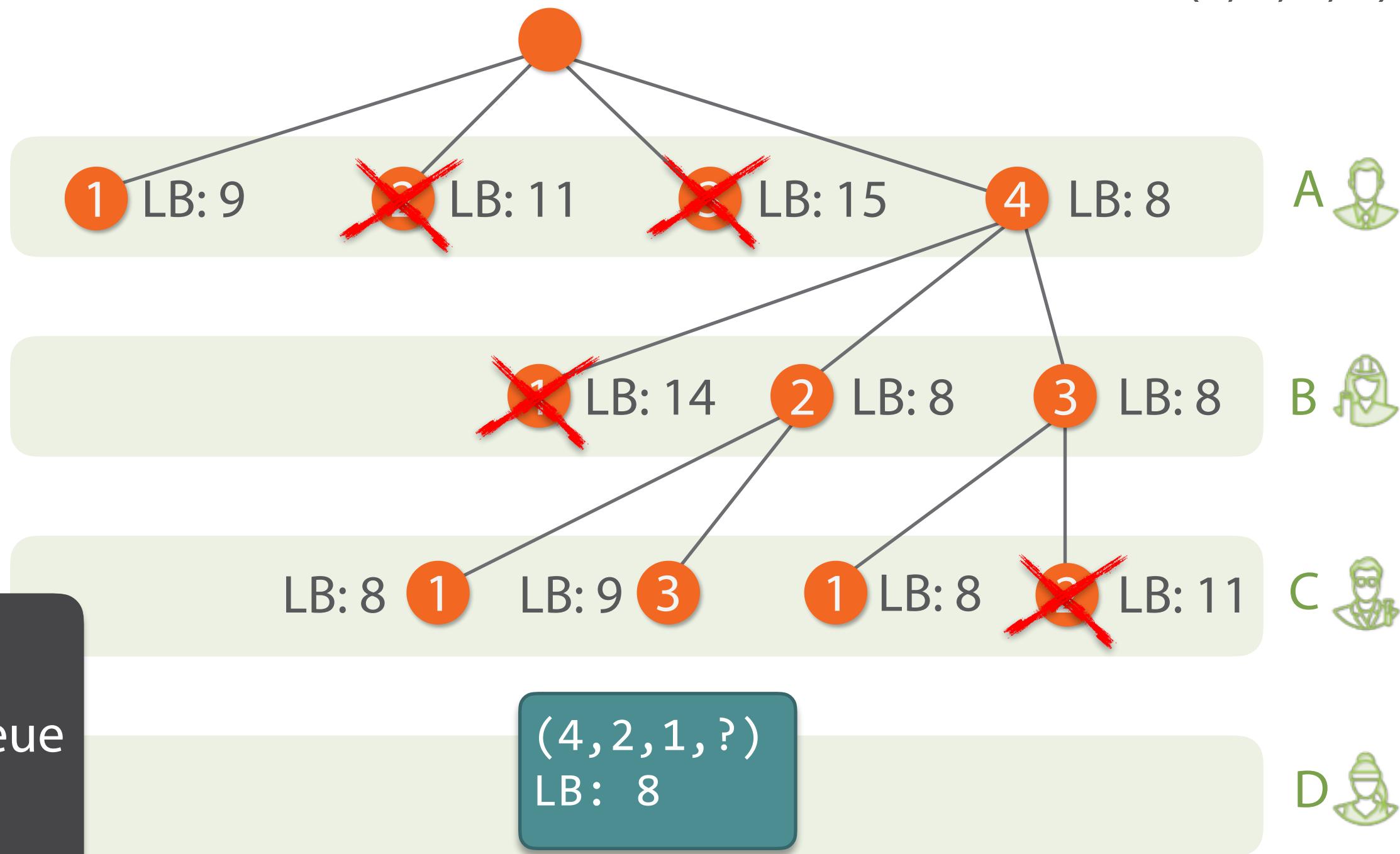


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	2
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2

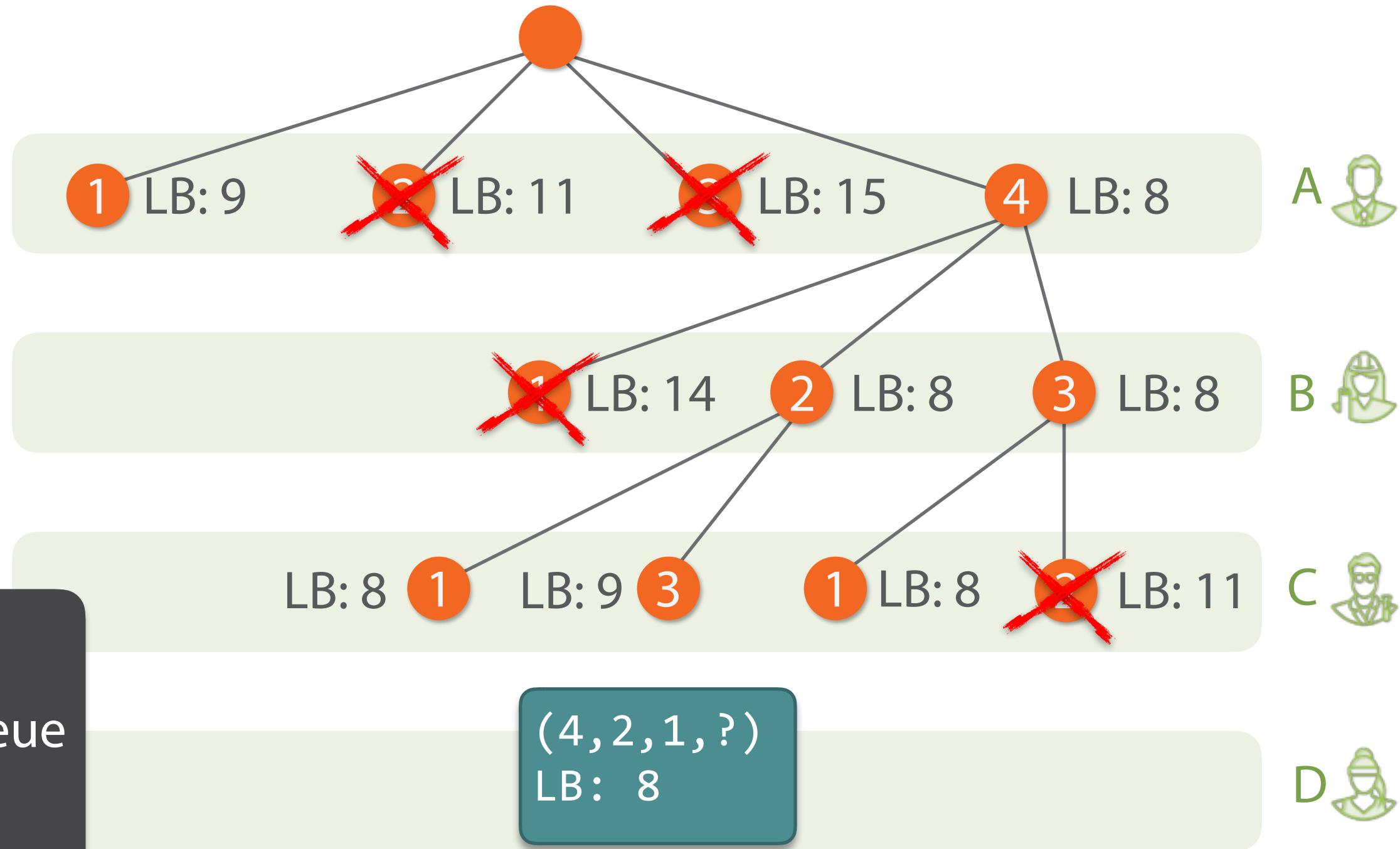


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	4
2	9	3	3	4
3	4	2	6	
4	5	3	7	2

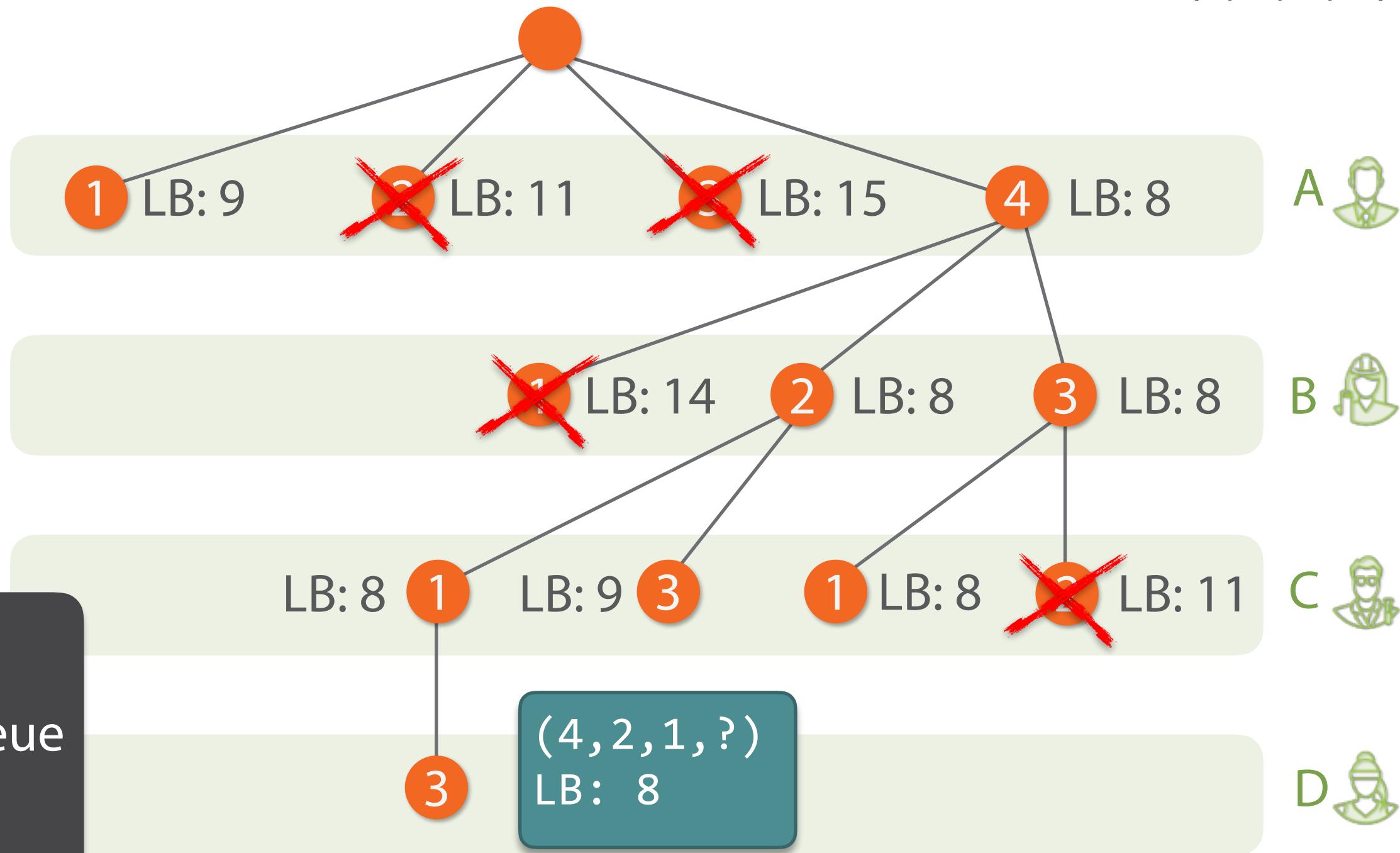


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	4
2	9	3	3	4
3	4	2	6	
4	5	3	2	2

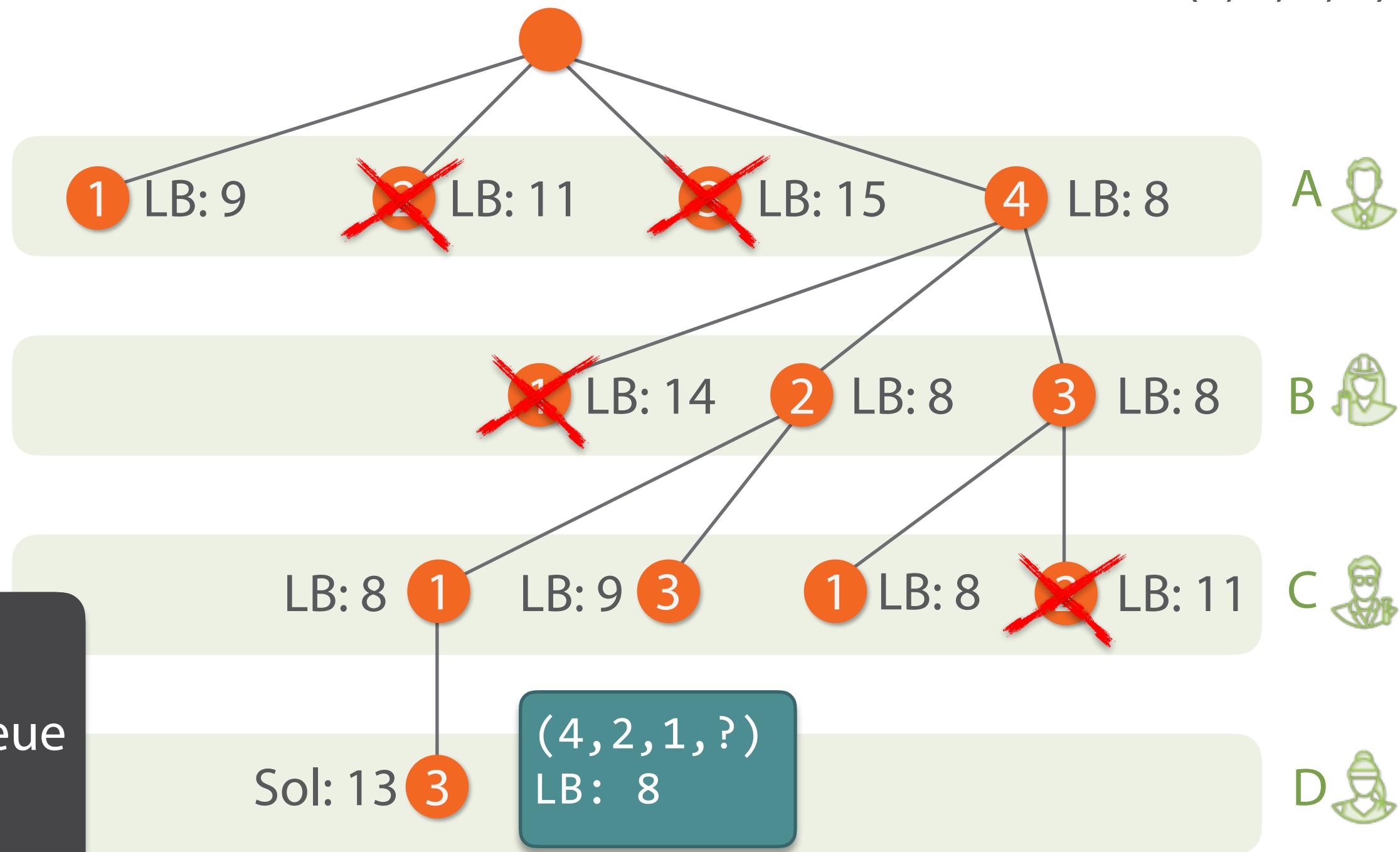
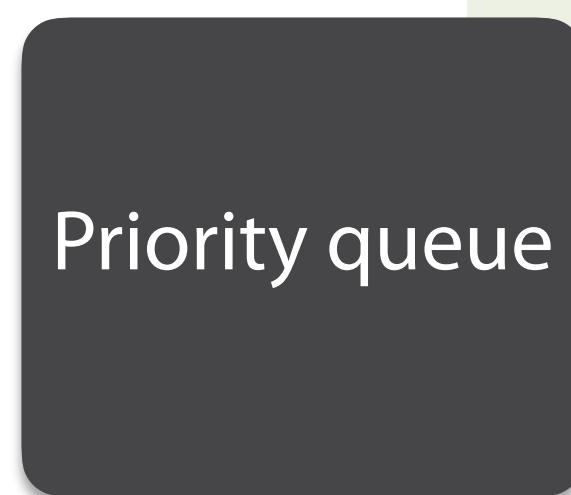


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	4
2	9	3	3	4
3	4	2	6	
4	5	3	2	2

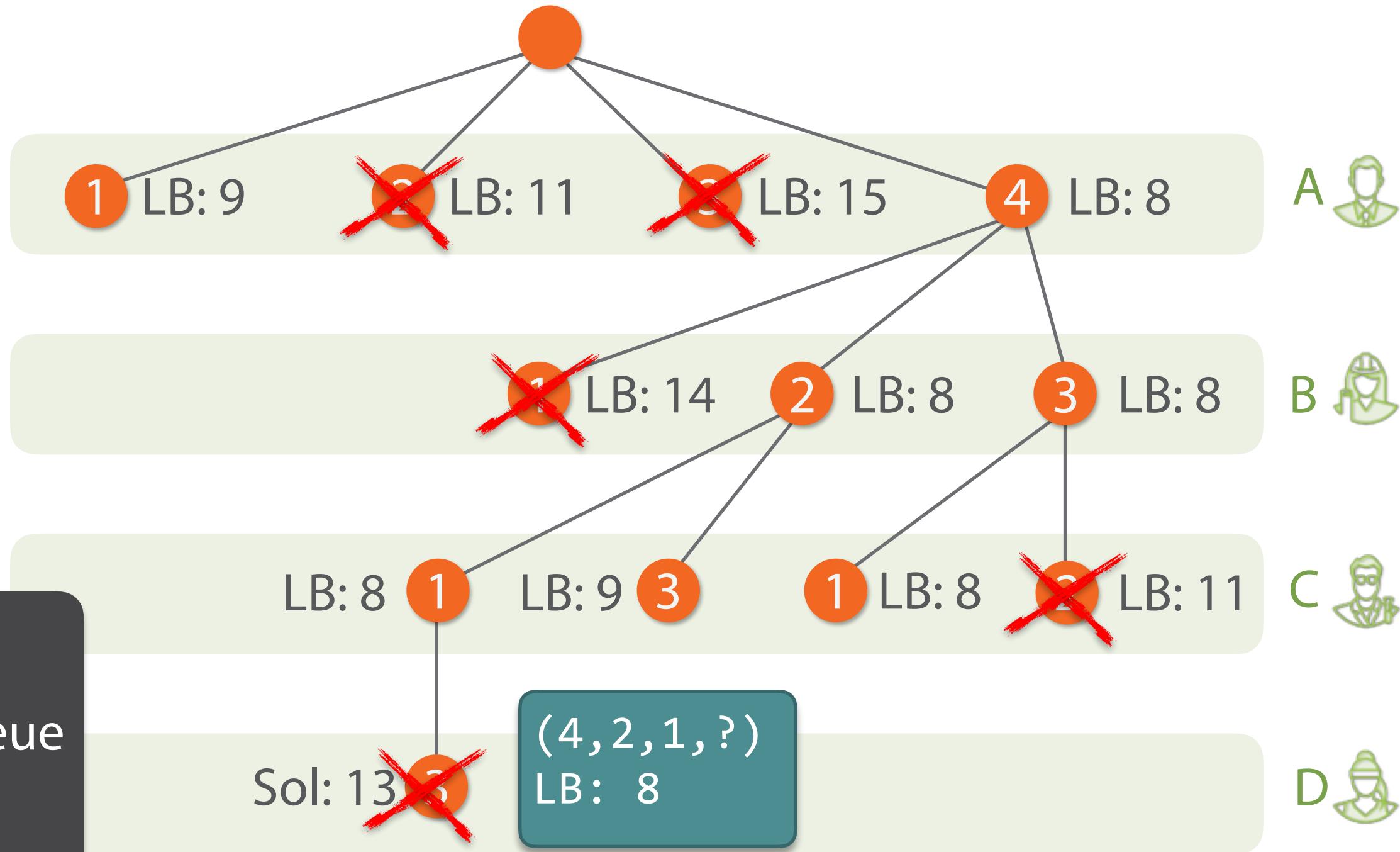
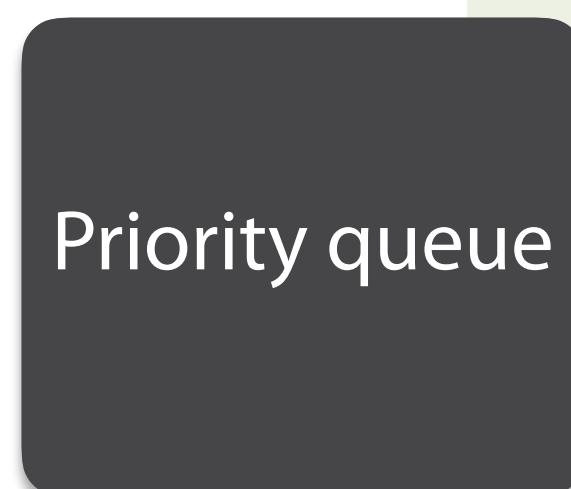


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	4
2	9	3	3	4
3	4	2	6	
4	5	3	2	2

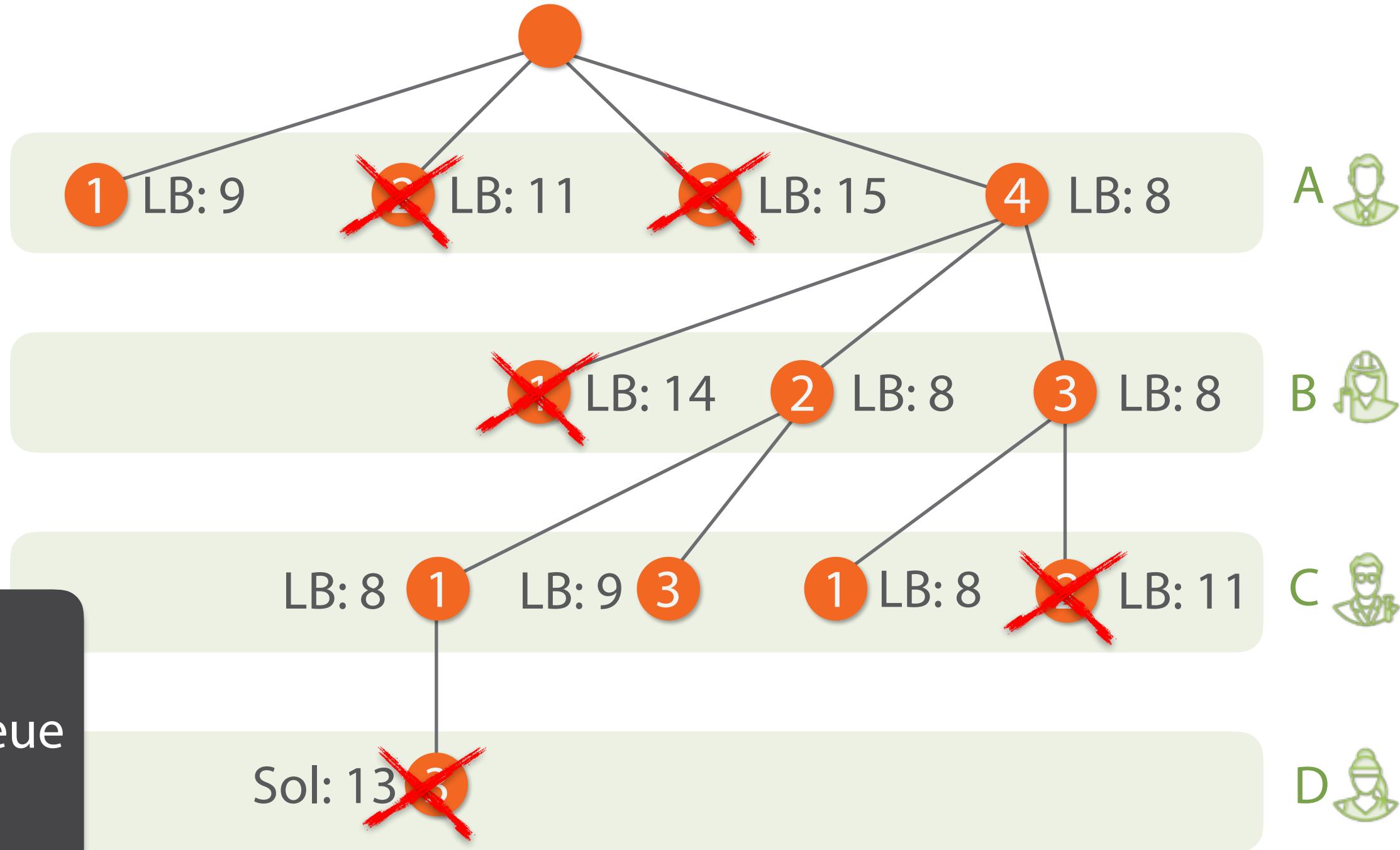


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	2
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2

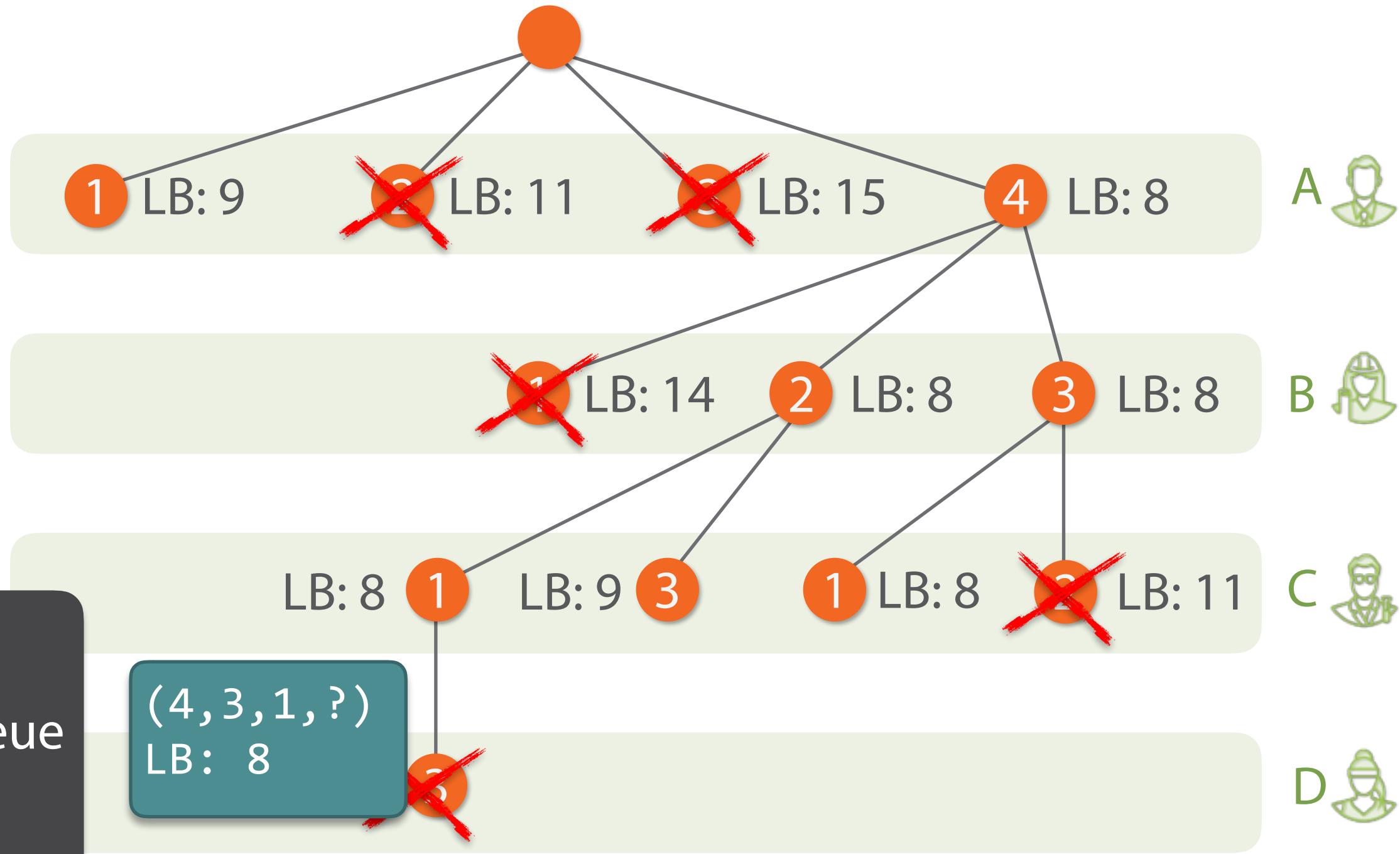


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	2
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2

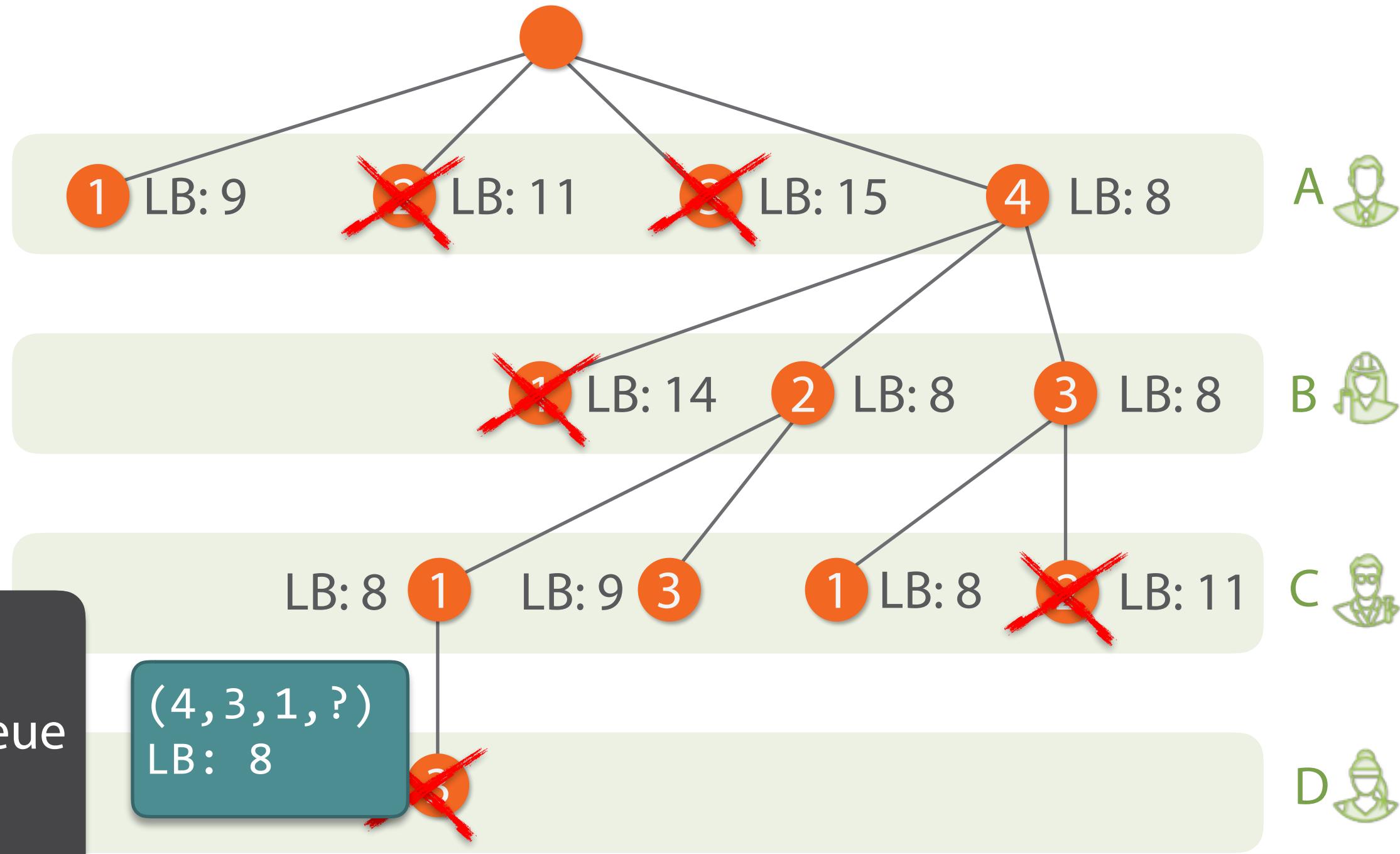


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

	1	2	3	4
1	3	5	9	2
2	9	3	4	1
3	4	2	6	
4	5	3	7	2

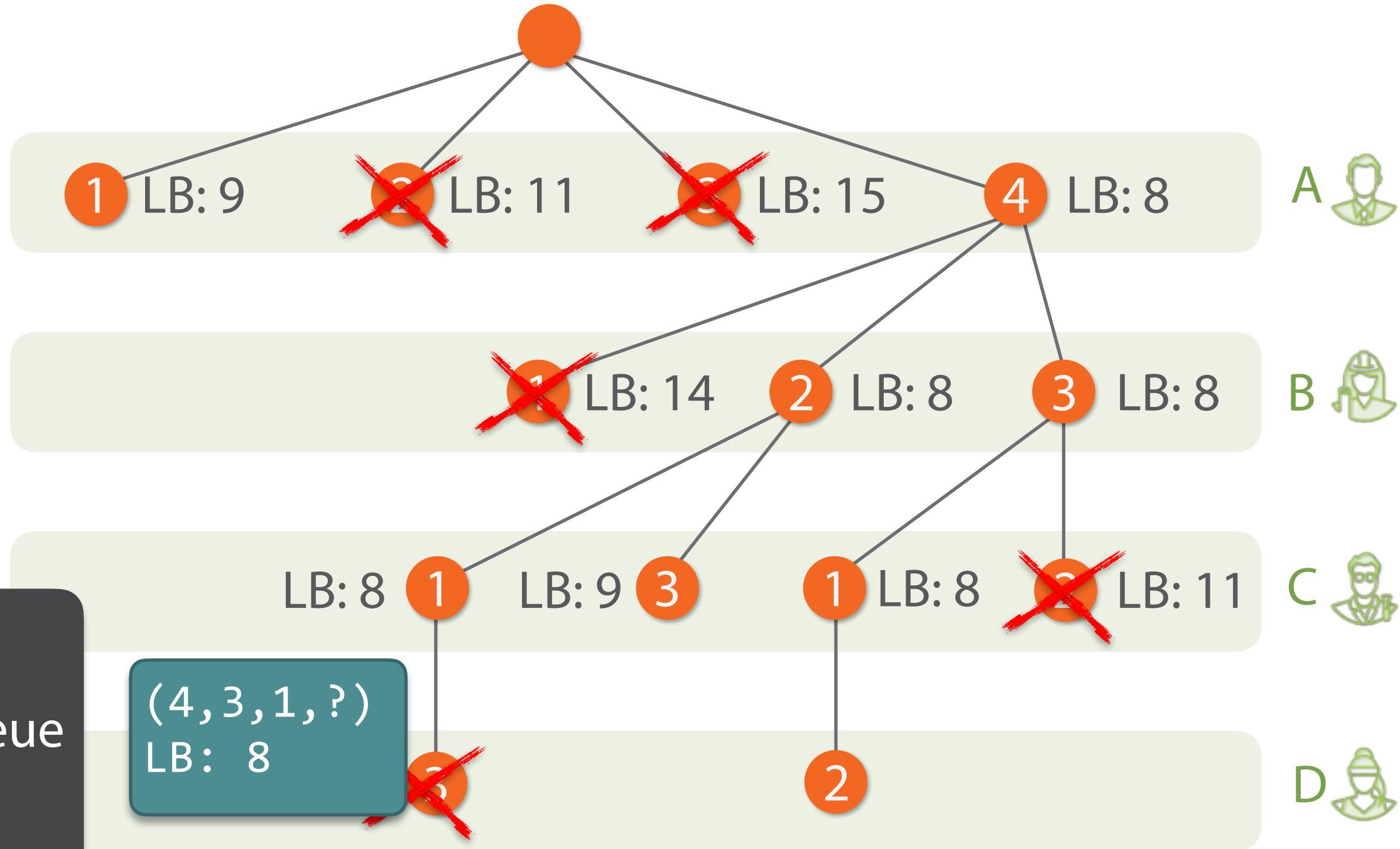


# The Assignment Problem

Best so far: 10

(1, 2, 3, 4)

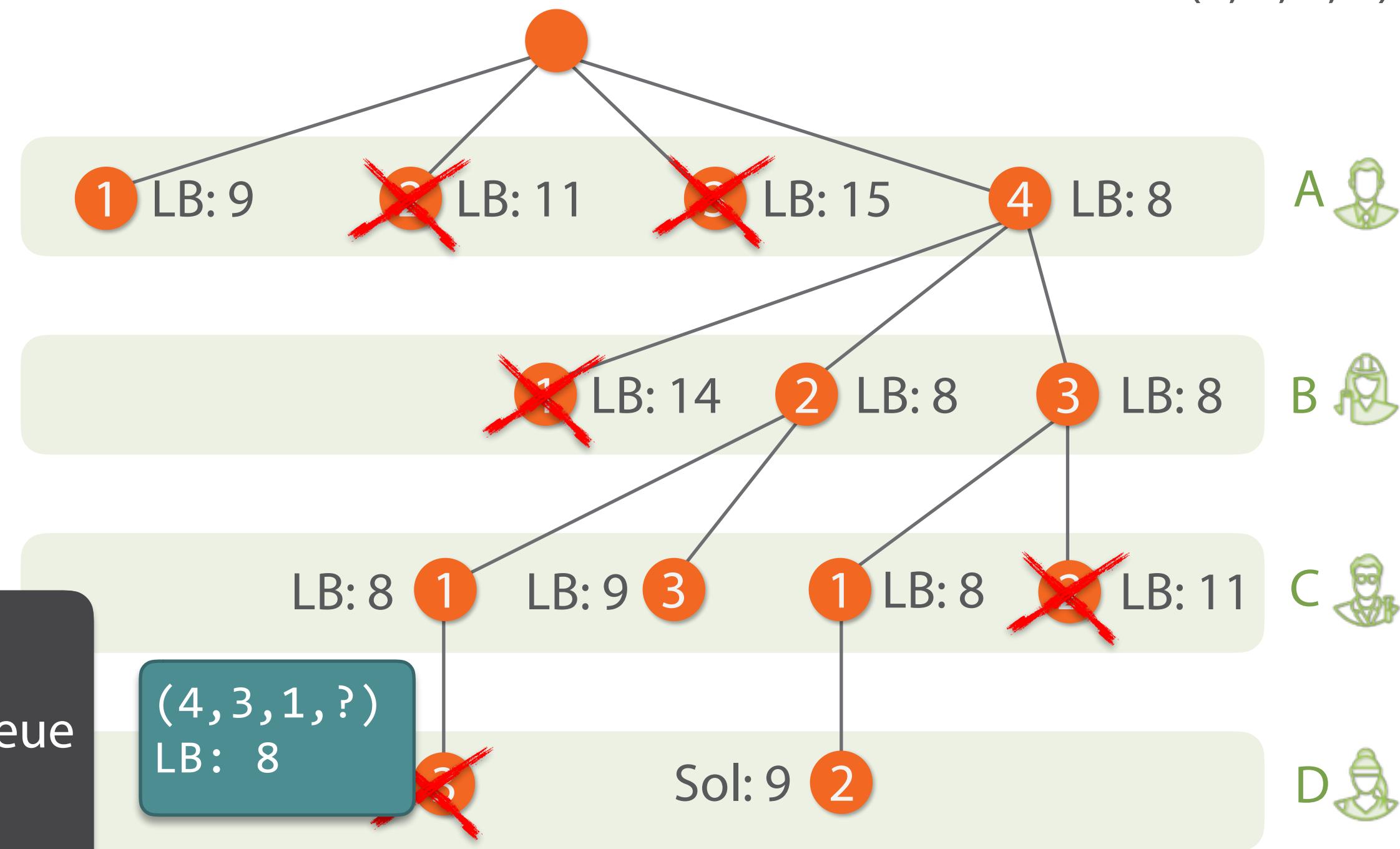
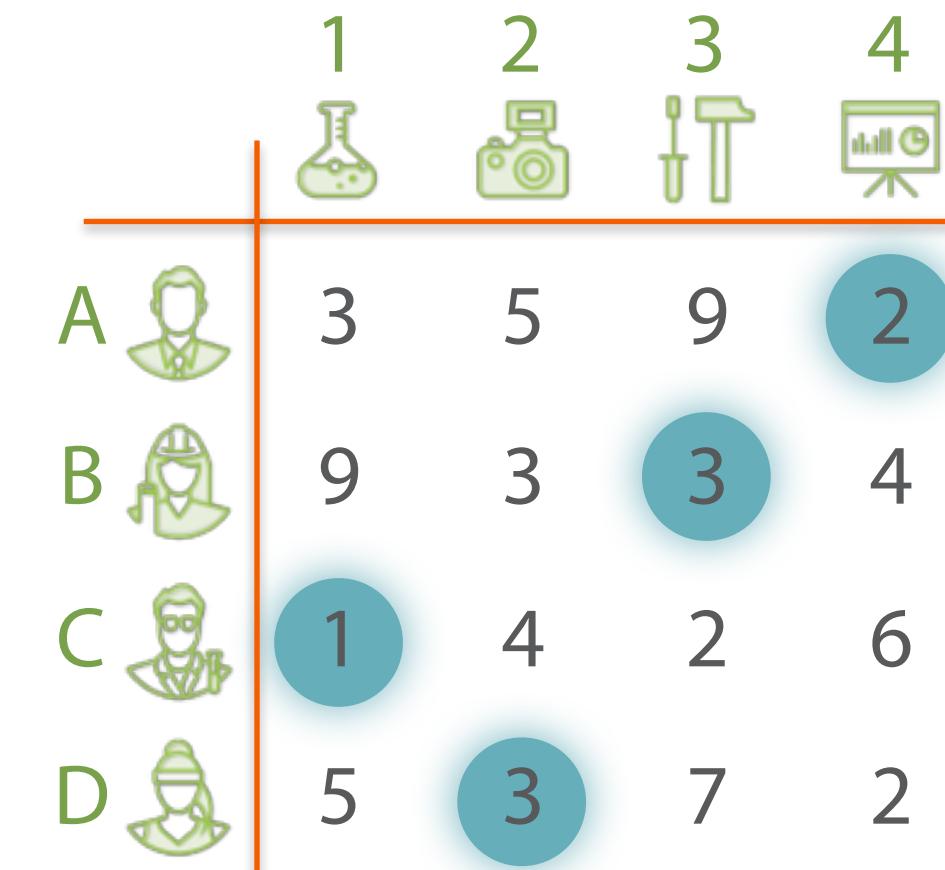
	1	2	3	4
1	3	5	9	2
2	9	3	1	4
3	4	2	6	
4	5	7	2	



# The Assignment Problem

# Best so far: 10

(1, 2, 3, 4)

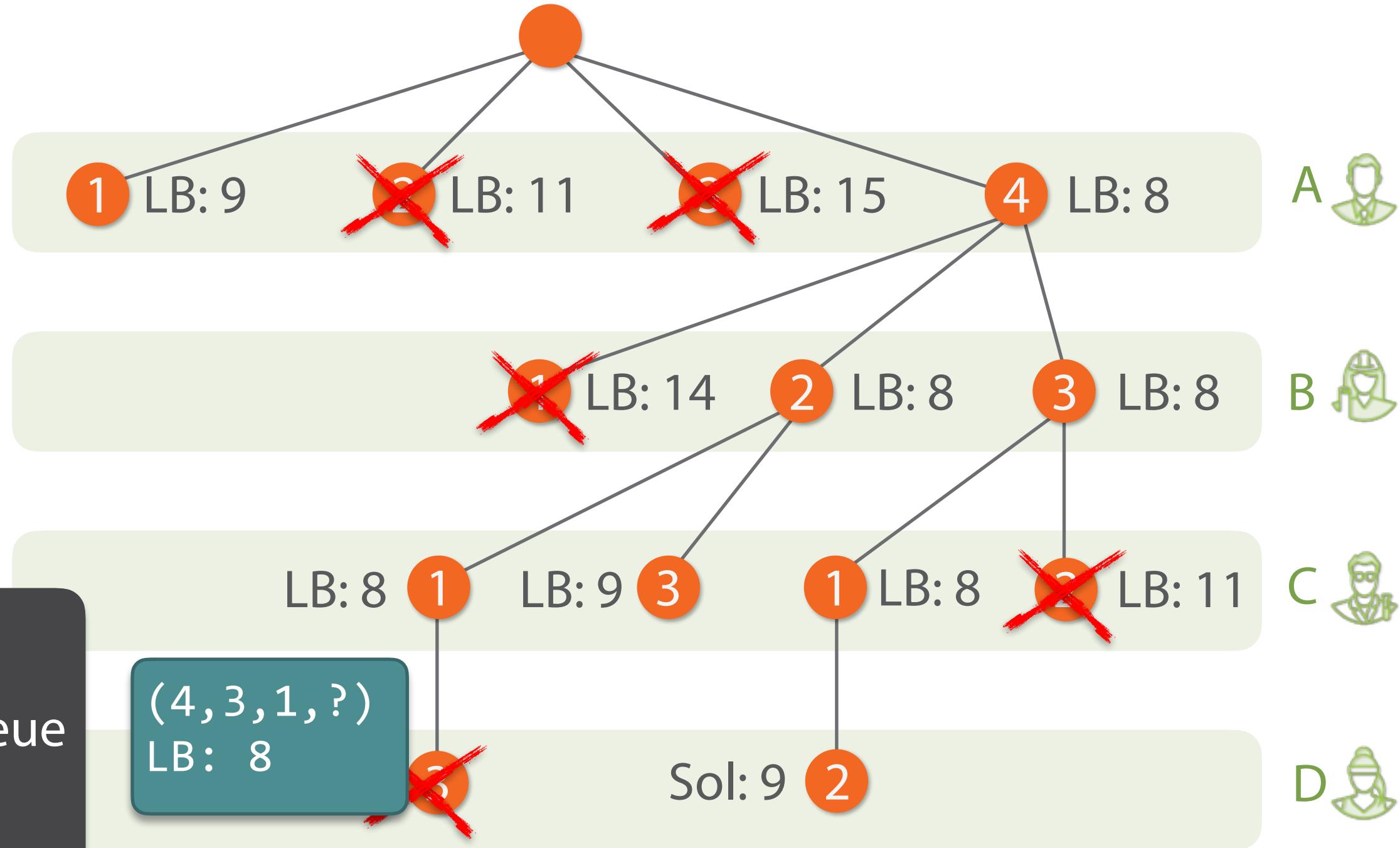


# The Assignment Problem

Best so far: 9

(4, 3, 1, 2)

	1	2	3	4
1	3	5	9	2
2	9	3	4	4
3	1	4	2	6
4	5	3	7	2

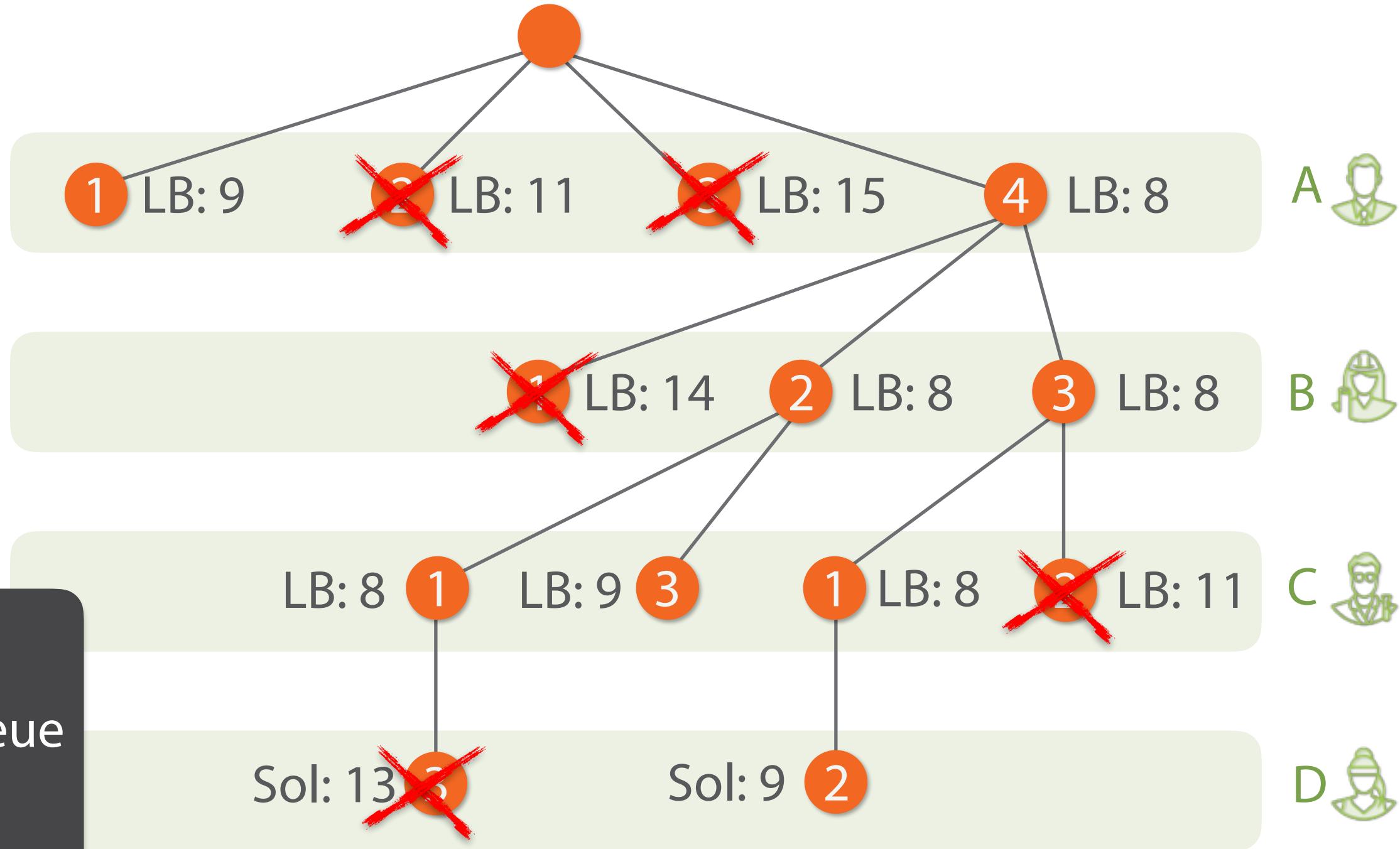


# The Assignment Problem

Best so far: 9

(4, 3, 1, 2)

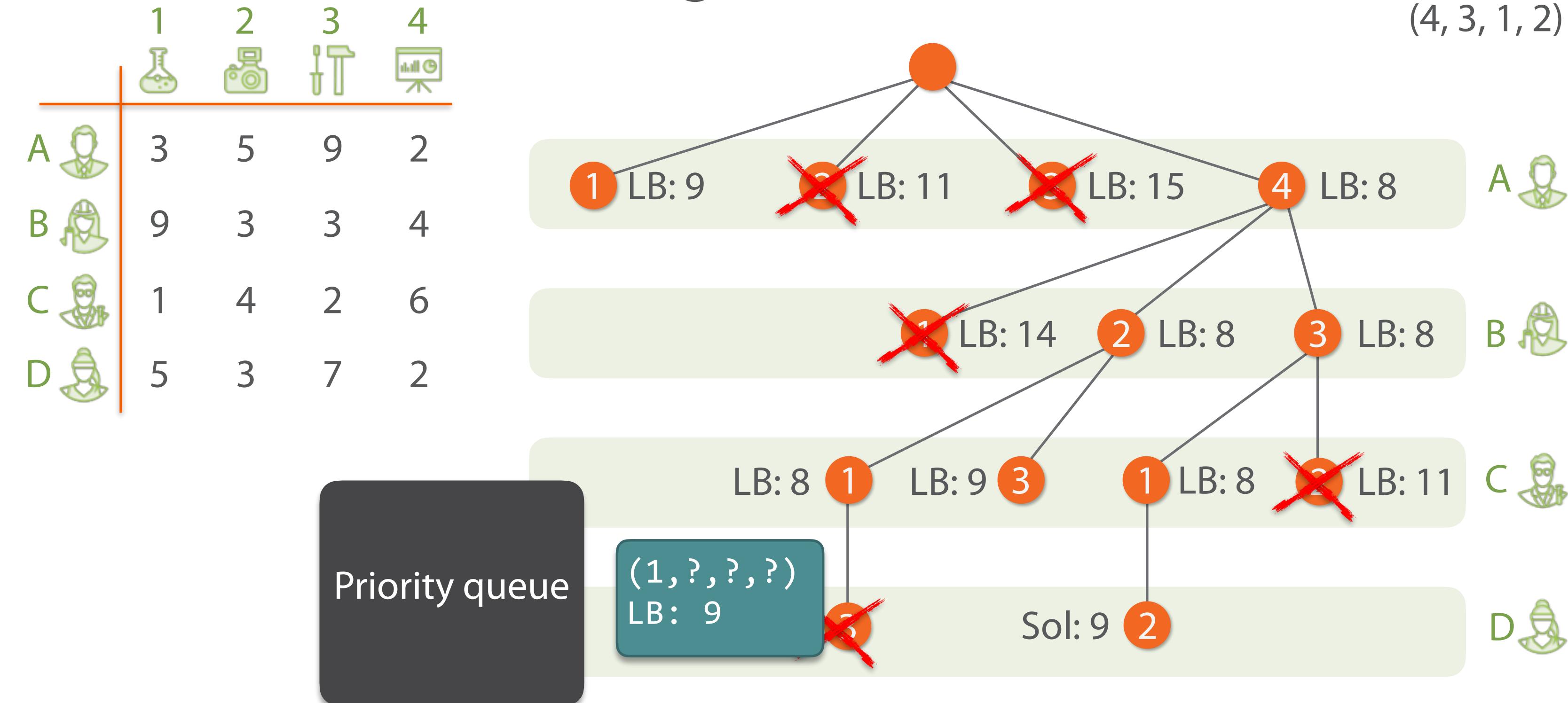
	1	2	3	4
1	3	5	9	2
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2



# The Assignment Problem

# Best so far: 9

(4, 3, 1, 2)



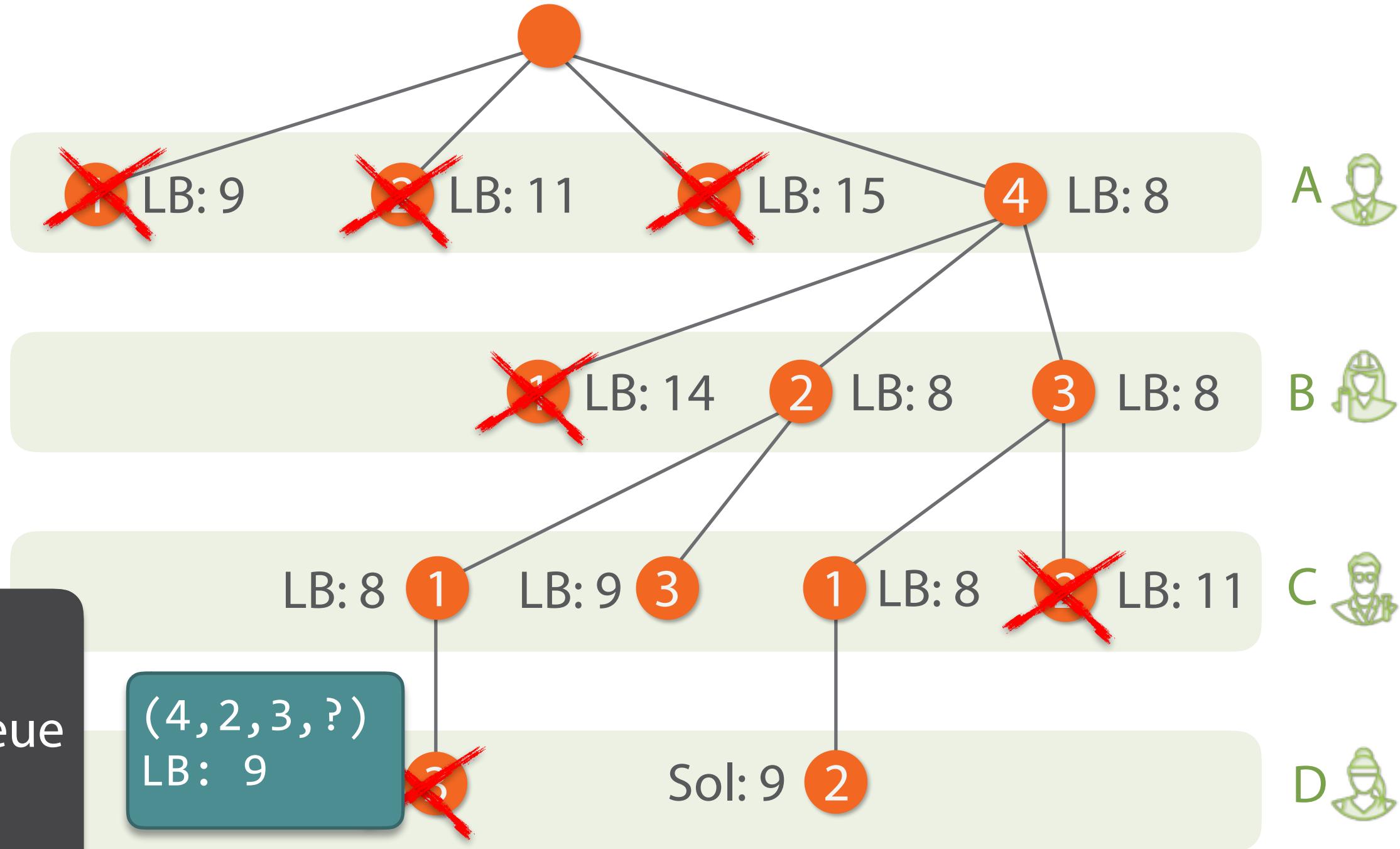


# The Assignment Problem

Best so far: 9

(4, 3, 1, 2)

	1	2	3	4
1	3	5	9	2
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2

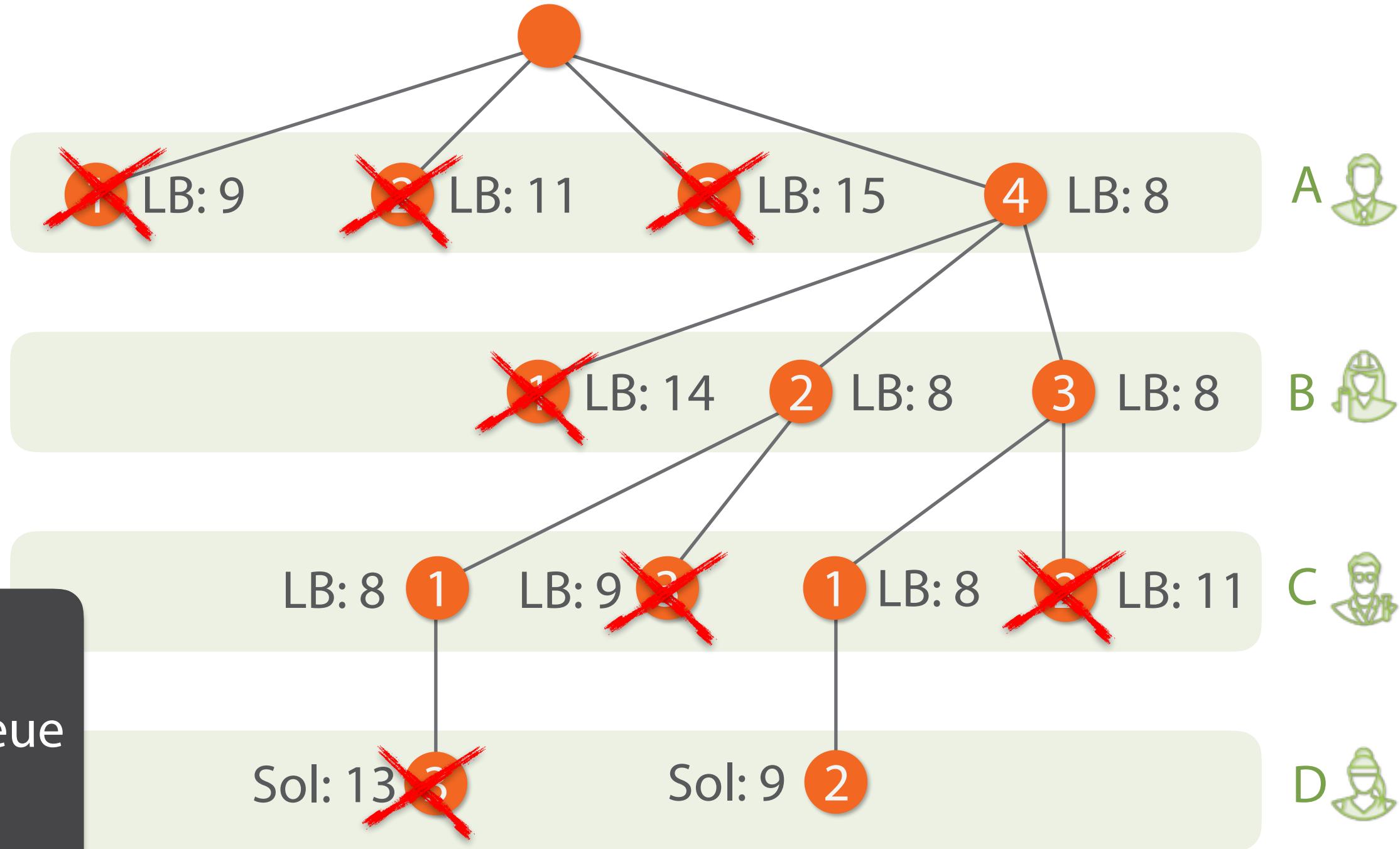


# The Assignment Problem

Best so far: 9

(4, 3, 1, 2)

	1	2	3	4
1	3	5	9	2
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2

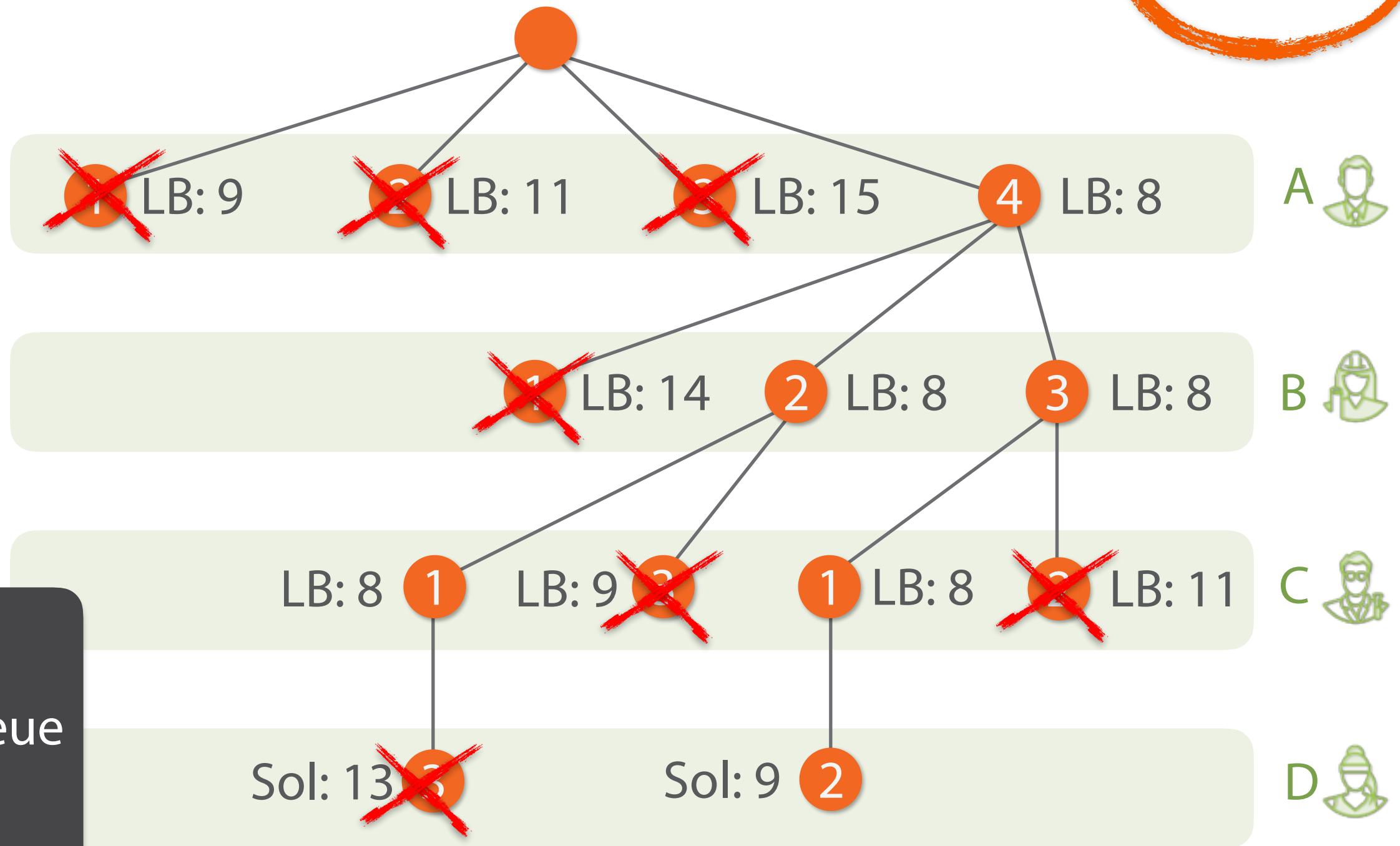


# The Assignment Problem

Best so far: 9

(4, 3, 1, 2)

	1	2	3	4
1	3	5	9	2
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2

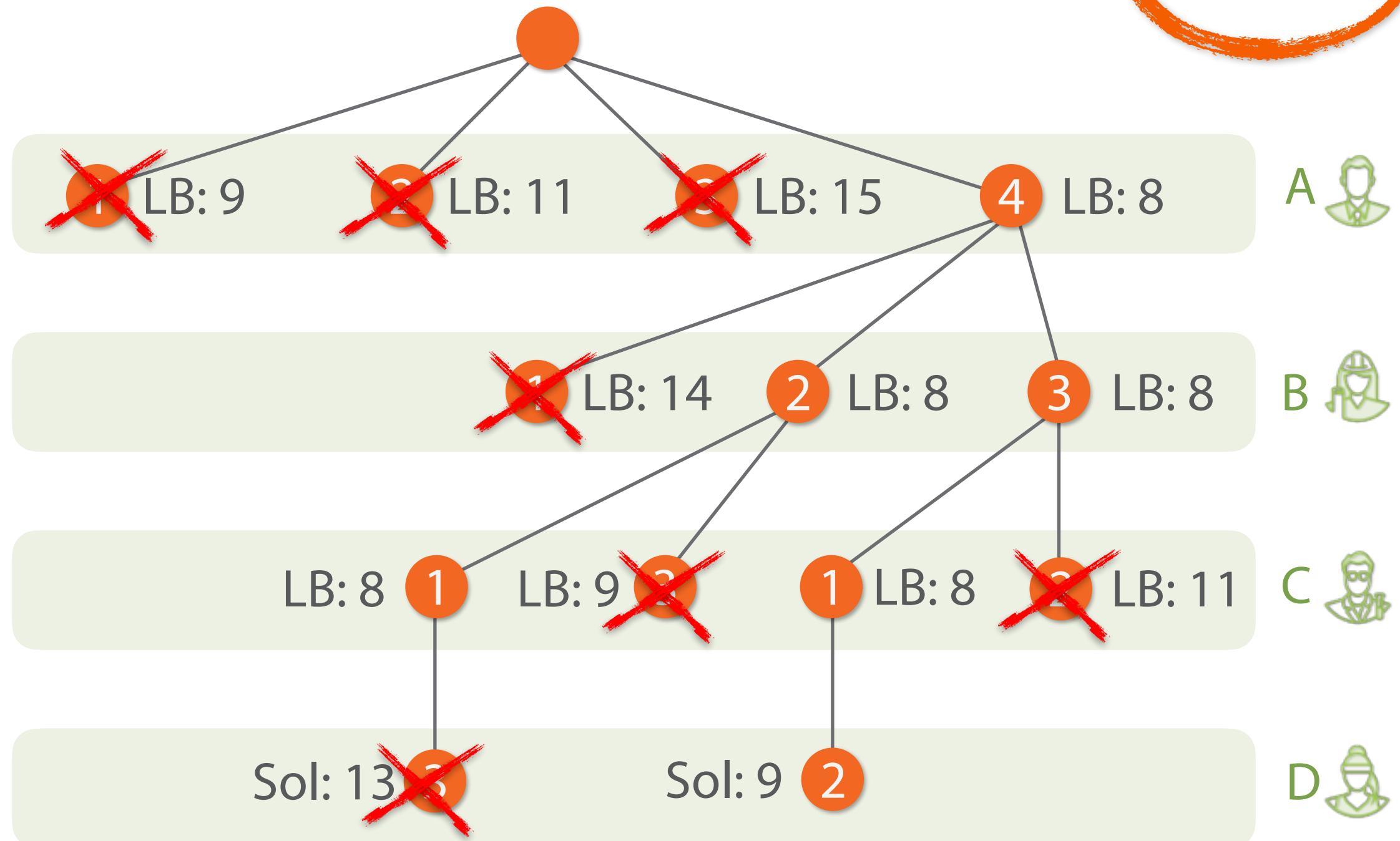


# The Assignment Problem

Best so far: 9

(4, 3, 1, 2)

	1	2	3	4
1	3	5	9	2
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2



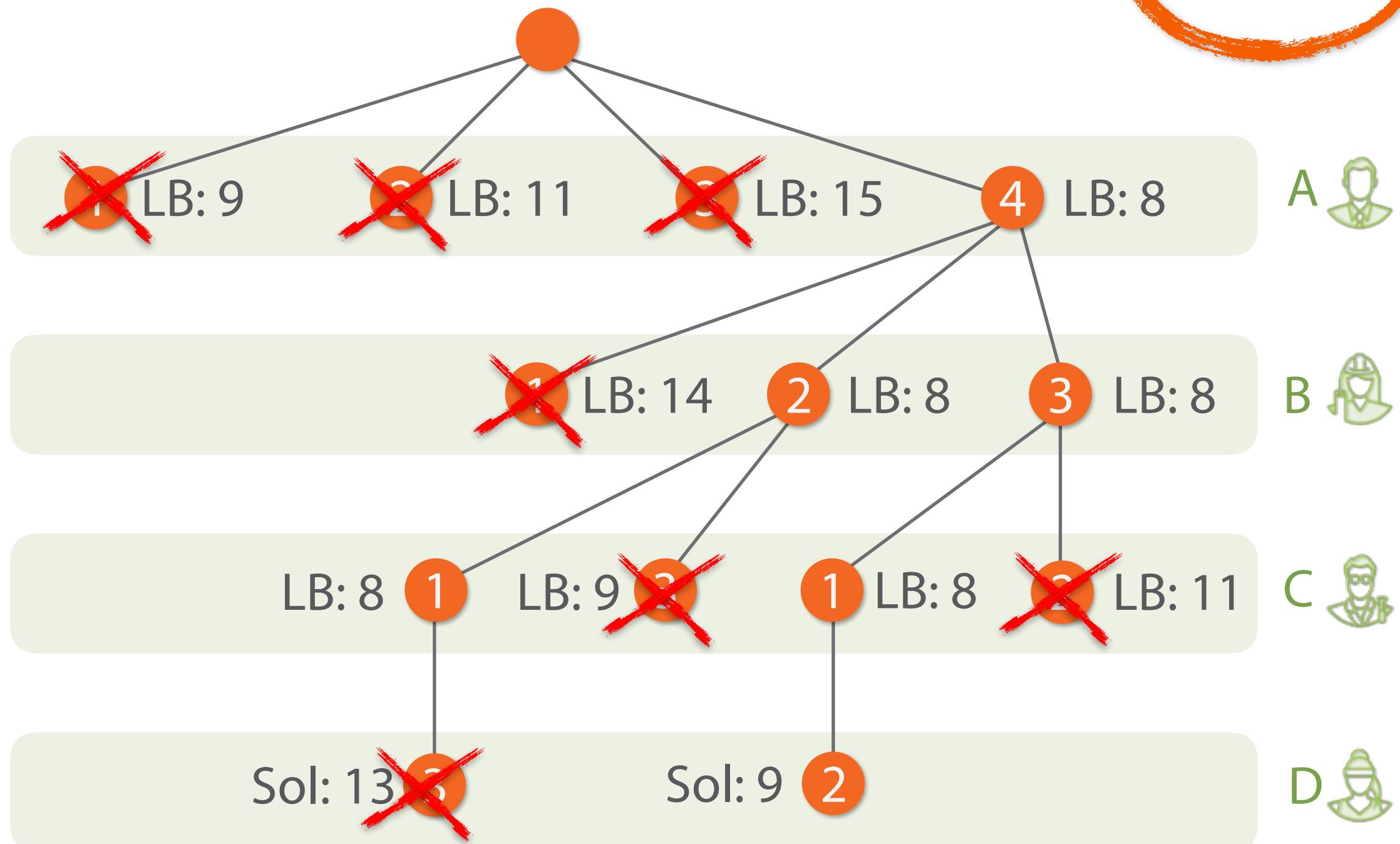
# The Assignment Problem

Best so far: 9

(4, 3, 1, 2)

	1	2	3	4
1	3	5	9	2
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2

For  $N$  agents (and tasks):



# The Assignment Problem

Best so far: 9

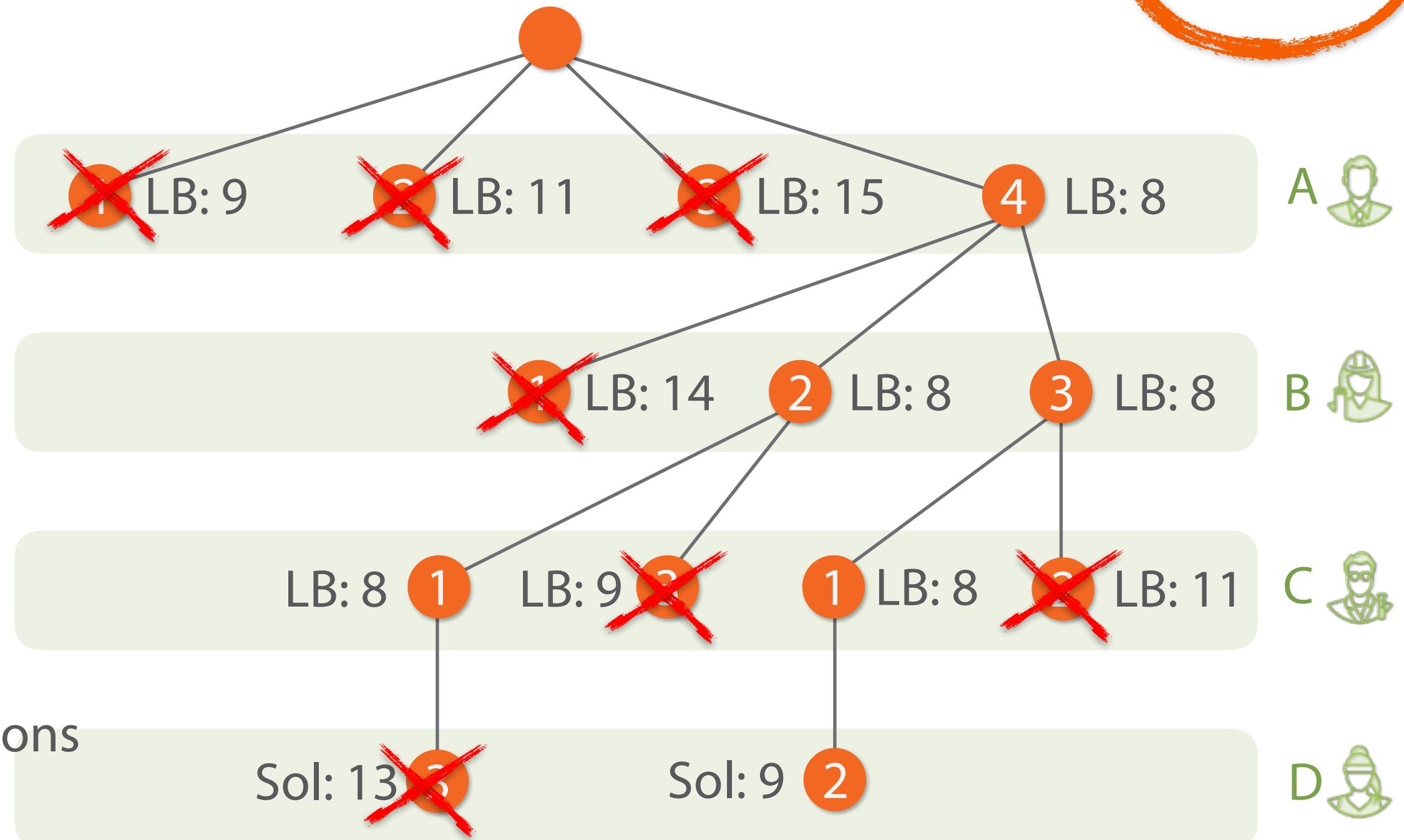
(4, 3, 1, 2)

	1	2	3	4
1	3	5	9	2
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2

For  $N$  agents (and tasks):

Brute force:

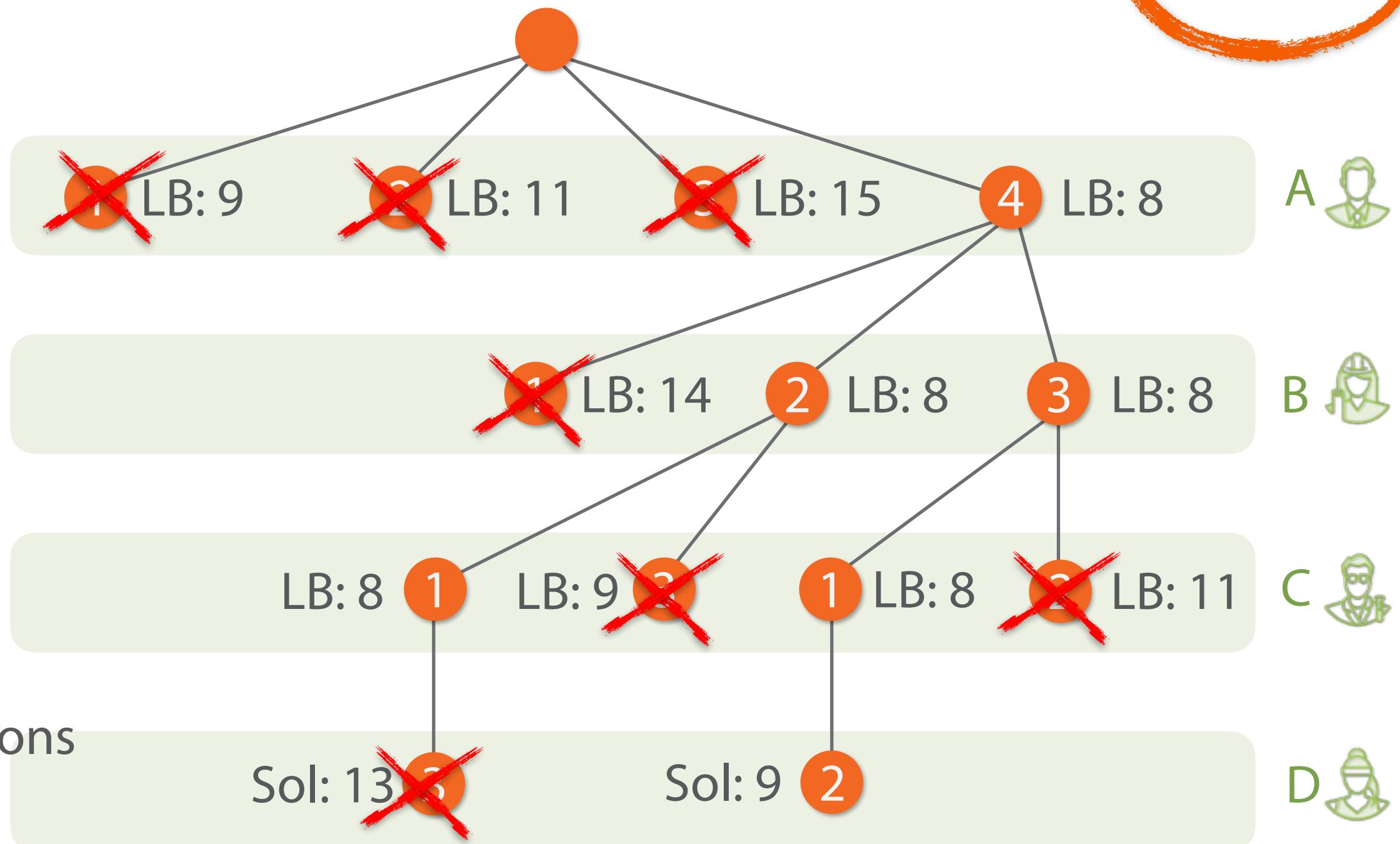
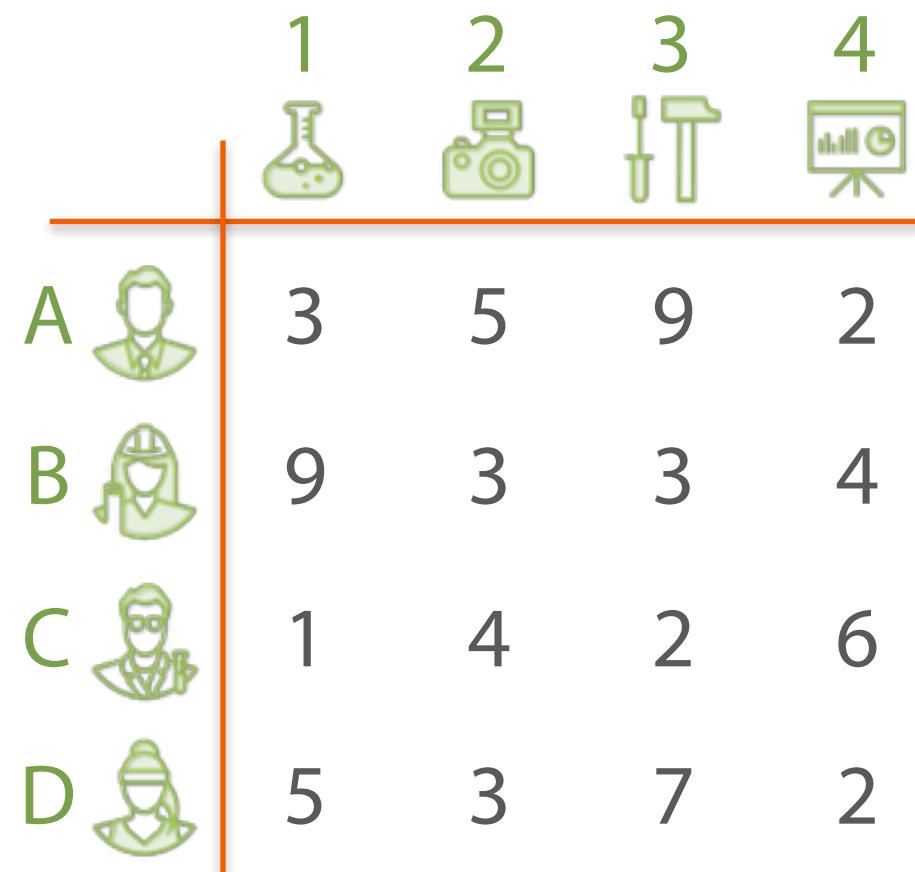
$N \cdot (N - 1) \cdots 2 \cdot 1$  combinations



# The Assignment Problem

# Best so far: 9

(4, 3, 1, 2)



# The Assignment Problem

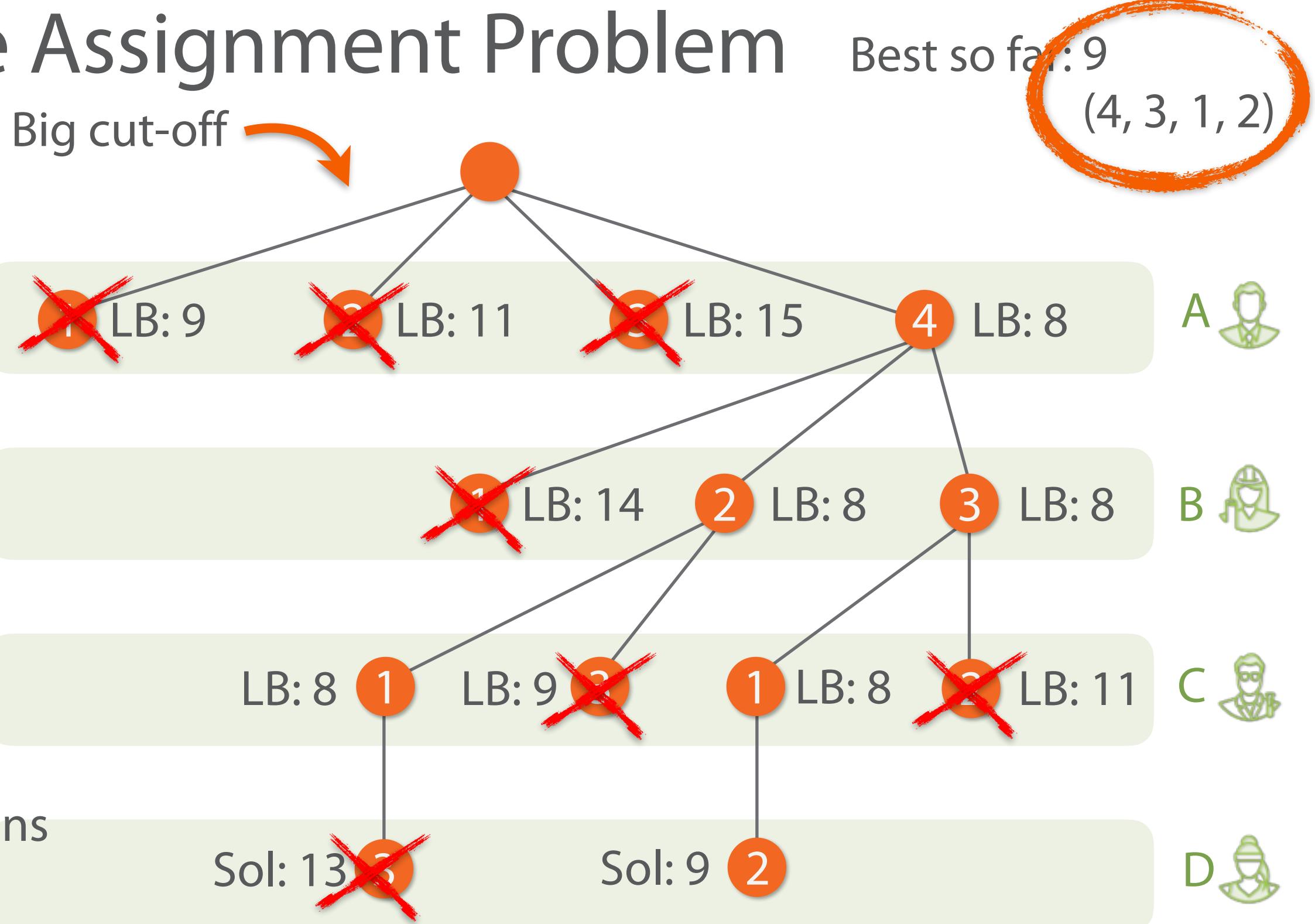
	1	2	3	4
1	3	5	9	2
2	9	3	3	4
3	1	4	2	6
4	5	3	7	2

For  $N$  agents (and tasks):

Brute force:

$N \cdot (N - 1) \cdots 2 \cdot 1$  combinations

$\Theta(N!)$



# Other Examples

# Other Examples

Generally: selecting *values* to  
pre-defined *placeholders*

# Other Examples

Generally: selecting *values* to  
pre-defined *placeholders*

Combination lock (with constraints)

# Other Examples

Generally: selecting *values* to  
pre-defined *placeholders*

Combination lock (with constraints)

Assignment problem

# Other Examples

Generally: selecting *values* to  
pre-defined *placeholders*

Combination lock (with constraints)

Assignment problem

0/1 knapsack problem

# Other Examples

Generally: selecting *values* to  
pre-defined *placeholders*

Combination lock (with constraints)

Assignment problem

0/1 knapsack problem

Choosing “good” subsets

# Other Examples

Generally: selecting *values* to  
pre-defined *placeholders*

Choosing “good” subsets

Choosing relevant features for model  
construction in machine learning

Combination lock (with constraints)

Assignment problem

0/1 knapsack problem

# Other Examples

Generally: selecting *values* to  
pre-defined *placeholders*

Choosing “good” subsets

Combination lock (with constraints)

Assignment problem

0/1 knapsack problem

Choosing relevant features for model  
construction in machine learning

Finding shortest paths in a graph

# Other Examples

Generally: selecting *values* to  
pre-defined *placeholders*

Choosing “good” subsets

Deciding configurations

Choosing relevant features for model  
construction in machine learning

Finding shortest paths in a graph

Combination lock (with constraints)

Assignment problem

0/1 knapsack problem

# Other Examples

Generally: selecting *values* to  
pre-defined *placeholders*

Combination lock (with constraints)

Assignment problem

0/1 knapsack problem

Choosing “good” subsets

Choosing relevant features for model  
construction in machine learning

Finding shortest paths in a graph

Deciding configurations

Arranging packages in a container

# Other Examples

Generally: selecting *values* to pre-defined *placeholders*

Choosing “good” subsets

Deciding configurations

“Good enough” solutions

Combination lock (with constraints)

Assignment problem

0/1 knapsack problem

Choosing relevant features for model construction in machine learning

Finding shortest paths in a graph

Arranging packages in a container

# Other Examples

Generally: selecting *values* to pre-defined *placeholders*

Combination lock (with constraints)

Assignment problem

0/1 knapsack problem

Choosing “good” subsets

Choosing relevant features for model construction in machine learning

Finding shortest paths in a graph

Deciding configurations

Arranging packages in a container

“Good enough” solutions

Calculating both lower and upper bounds

# Lessons Learned

Graph traversal

Brute force  
Greedy algorithms

Divide  
and  
conquer

Dynamic  
programming

Branch  
and  
bound

# Lessons Learned

Graph traversal

Brute force  
Greedy algorithms

Divide  
and  
conquer

Dynamic  
programming

Branch  
and  
bound

# Lessons Learned

Graphs and trees

Graph traversal

Brute force  
Greedy algorithms

Divide  
and  
conquer

Dynamic  
programming

Branch  
and  
bound

# Lessons Learned

Graphs and trees

Depth first search

Graph traversal

Divide  
and  
conquer

Dynamic  
programming

Brute force  
Greedy algorithms

Branch  
and  
bound

# Lessons Learned

Graphs and trees

Depth first search

Preorder / postorder

Graph traversal

Divide  
and  
conquer

Dynamic  
programming

Brute force  
Greedy algorithms

Branch  
and  
bound

# Lessons Learned

Graphs and trees

Breadth first search

Depth first search

Graph traversal

Preorder / postorder

Divide  
and  
conquer

Brute force  
Greedy algorithms

Dynamic  
programming

Branch  
and  
bound

# Lessons Learned

Graphs and trees

Breadth first search

Depth first search

Graph traversal

Preorder / postorder

Divide  
and  
conquer

Dynamic  
programming

Brute force  
Greedy algorithms

Branch  
and  
bound

# Lessons Learned

Graphs and trees

Breadth first search

Depth first search

Graph traversal

Preorder / postorder

Divide  
and  
conquer

Dynamic  
programming

Brute force: the “naive” approach

Brute force  
Greedy algorithms

Branch  
and  
bound

# Lessons Learned

Graphs and trees

Breadth first search

Depth first search

Graph traversal

Preorder / postorder

Divide  
and  
conquer

Dynamic  
programming

Brute force: the “naive” approach

Brute force  
Greedy algorithms

Greedy: always  
go with what is  
“best” now

Branch  
and  
bound

# Lessons Learned

Graphs and trees

Breadth first search

Depth first search

Graph traversal

Preorder / postorder

Divide  
and  
conquer

Dynamic  
programming

Brute force: the “naive” approach

Brute force  
Greedy algorithms

Greedy: always  
go with what is  
“best” now

Branch  
and  
bound

# Lessons Learned

Graphs and trees

Breadth first search

Depth first search

Graph traversal

Preorder / postorder

Divide into  
sub-  
problems

Divide  
and  
conquer

Brute force: the “naive” approach

Brute force  
Greedy algorithms

Greedy: always  
go with what is  
“best” now

Dynamic  
programming

Branch  
and  
bound

# Lessons Learned

Graphs and trees

Breadth first search

Depth first search

Graph traversal

Preorder / postorder

Divide into  
sub-  
problems

Divide  
and  
conquer

Combine sub-solutions

Brute force: the “naive” approach

Brute force

Greedy algorithms

Greedy: always  
go with what is  
“best” now

Dynamic  
programming

Branch  
and  
bound

# Lessons Learned

Graphs and trees

Breadth first search

Depth first search

Graph traversal

Preorder / postorder

Divide into  
sub-  
problems

Divide  
and  
conquer

Combine sub-solutions

Brute force: the “naive” approach

Brute force

Greedy algorithms

Greedy: always  
go with what is  
“best” now

Dynamic  
programming

Branch  
and  
bound

# Lessons Learned

Graphs and trees

Breadth first search

Depth first search

Graph traversal

Preorder / postorder

Divide into  
sub-  
problems

Divide  
and  
conquer

Combine sub-solutions

Brute force: the “naive” approach

Brute force

Greedy algorithms

Greedy: always  
go with what is  
“best” now

Cache overlapping sub-results

Dynamic  
programming

Branch  
and  
bound

# Lessons Learned

Graphs and trees

Breadth first search

Depth first search

Graph traversal

Preorder / postorder

Divide into  
sub-  
problems

Divide  
and  
conquer

Combine sub-solutions

Brute force: the “naive” approach

Brute force

Greedy algorithms

Greedy: always  
go with what is  
“best” now

Cache overlapping sub-results

Dynamic  
programming

Top-down

Branch  
and  
bound

# Lessons Learned

Graphs and trees

Breadth first search

Depth first search

Graph traversal

Preorder / postorder

Divide into  
sub-  
problems

Divide  
and  
conquer

Combine sub-solutions

Brute force: the “naive” approach

Brute force

Greedy algorithms

Greedy: always  
go with what is  
“best” now

Cache overlapping sub-results

Dynamic  
programming

Top-down

Bottom-up

Branch  
and  
bound

# Lessons Learned

Graphs and trees

Breadth first search

Depth first search

Graph traversal

Preorder / postorder

Divide into  
sub-  
problems

Divide  
and  
conquer

Combine sub-solutions

Brute force: the “naive” approach

Brute force

Greedy algorithms

Greedy: always  
go with what is  
“best” now

Cache overlapping sub-results

Dynamic  
programming

Top-down

Bottom-up

Branch  
and  
bound

# Lessons Learned

Graphs and trees

Breadth first search

Depth first search

Graph traversal

Preorder / postorder

Divide into  
sub-  
problems

Divide  
and  
conquer

Combine sub-solutions

Brute force: the “naive” approach

Brute force

Greedy algorithms

Greedy: always  
go with what is  
“best” now

Cache overlapping sub-results

Dynamic  
programming

Top-down

Bottom-up

Branch  
and  
bound

Branch out on each “choice”

# Lessons Learned

Graphs and trees

Breadth first search

Depth first search

Graph traversal

Preorder / postorder

Divide into  
sub-  
problems

Divide  
and  
conquer

Combine sub-solutions

Brute force: the “naive” approach

Brute force

Greedy algorithms

Greedy: always  
go with what is  
“best” now

Cache overlapping sub-results

Dynamic  
programming

Top-down

Bottom-up

Branch  
and  
bound

Bound  
sub-  
solutions  
and skip  
sub-trees

Branch out on each “choice”

# Lessons Learned

Graphs and trees

Breadth first search

Depth first search

Graph traversal

Preorder / postorder

Divide into  
sub-  
problems

Divide  
and  
conquer

Combine sub-solutions

Brute force: the “naive” approach

Brute force

Greedy algorithms

Greedy: always  
go with what is  
“best” now

Cache overlapping sub-results

Dynamic  
programming

Top-down

Bottom-up

Branch  
and  
bound

Bound  
sub-  
solutions  
and skip  
sub-trees

Branch out on each “choice”