CS2040S

Data Structures and Algorithms

Trees -

Augmentations, Other Trees, Problem Solving with Trees

Midterm

March 3rd at MPSH2A/B (Week 7 Monday)
6:30PM to 9:00PM

1 Page 2-sided A4 cheat sheet

Topics: Up until this lecture

Format change this year! no longer 100% MCQ
Instead: A few short answer questions

CS2040S

Data Structures and Algorithms

Trees -

Augmentations, Other Trees, Problem Solving with Trees

Where were we?

Trees

- Terminology
- Traversals
- Operations

Balanced Trees

- Height-balanced binary search trees
- AVL trees
- Rotations

Todays Plan

Data structure design

- More Augmentation on Balanced Trees

Tries

- How to handle text?

Problem Solving Using Trees

- Thinking with Trees

Dynamic Data Structures

1. Maintain a set of items

2. Modify the set of items

3. Answer queries.

Dynamic Data Structures

1. Maintain a set of items

2. Modify the set of items

3. Answer queries.

B-trees are at the heart of *every* database!

Big picture idea:

Trees are a good way to store, summarize, and search dynamic data.

Dynamic Data Structures

- Operations that create a data structure
  - build (preprocess)
- Operations that modify the structure
  - insert
  - delete
- Query operations
  - search, select, etc.

So far, we have learned trees.

So far, we have learned trees.

There’s two aspects about them that go in different directions:

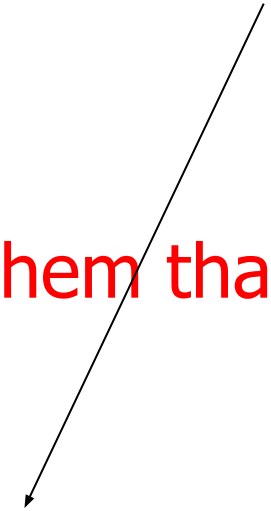
So far, we have learned trees.

There's two aspects about them that go in different directions:

1. Using trees to solve bigger problems
2. Creating new kinds of trees.
   - Via augmentations
   - Completely new kinds

So far, we have learned trees.

This is probably a little apparent



There's two aspects about them that go in different directions:

1. Using trees to solve bigger problems
2. Creating new kinds of trees.
   - Via augmentations
   - Completely new kinds

"Why do we need to learn how an AVL tree works?"

Just use a Java TreeMap, right?

"Why do we need to learn how an AVL tree works?"

1. Learn how to think like a computer scientist.

"Why do we need to learn how an AVL tree works?"

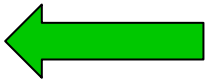
1. Learn how to think like a computer scientist.
2. Learn to modify existing data structures to solve new problems.

Augmented Data Structures

Many problems require storing additional data in a standard data structure.

Augment more frequently than invent...

Todays Plan

Data structure design 

- More Augmentation on Balanced Trees

Tries

- How to handle text?

Problem Solving Using Trees

- Thinking with Trees

Augmenting data structures

Basic methodology:

1. Choose underlying data structure

   (tree, hash table, linked list, stack, etc.)

Augmenting data structures

Basic methodology:

1. Choose underlying data structure

   (tree, hash table, linked list, stack, etc.)

2. Determine additional info needed.

Augmenting data structures

Basic methodology:

1. Choose underlying data structure

   (tree, hash table, linked list, stack, etc.)

2. Determine additional info needed.

3. Modify data structure to *maintain* additional info when the structure changes.

   (subject to insert/delete/etc.)

Augmenting data structures

Basic methodology:

1. Choose underlying data structure

   (tree, hash table, linked list, stack, etc.)

2. Determine additional info needed.

3. Modify data structure to *maintain* additional info when the structure changes.

   (subject to insert/delete/etc.)

4. Develop new operations.

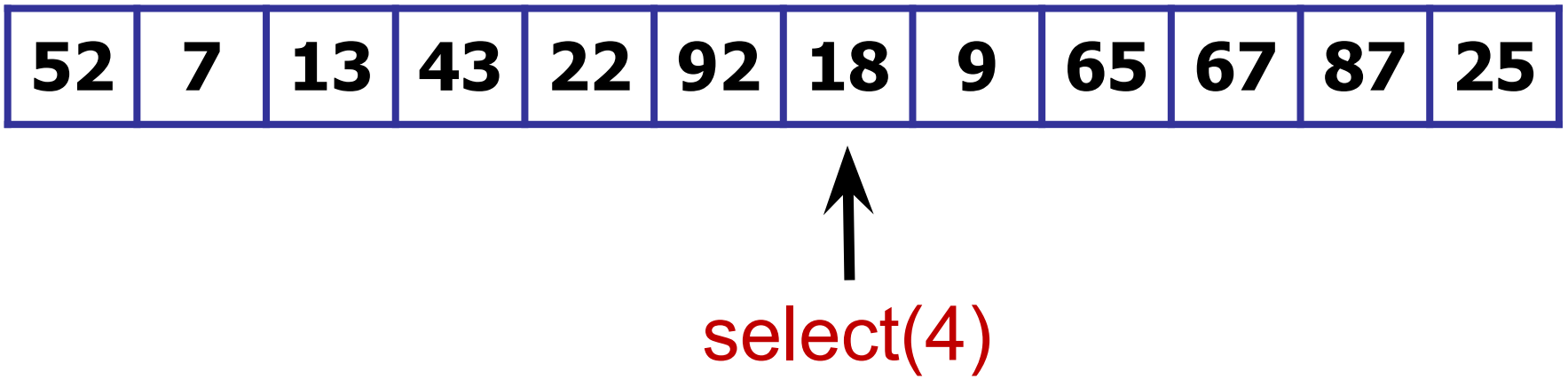
Order Statistics

Input

A set of integers.

Output: select(k)

The k^{th} item in the set.



select(2) returns:

52	7	13	43	22	92	18	9	65	67	87	25

1. 52
2. 9
3. 13
4. 43
5. 25

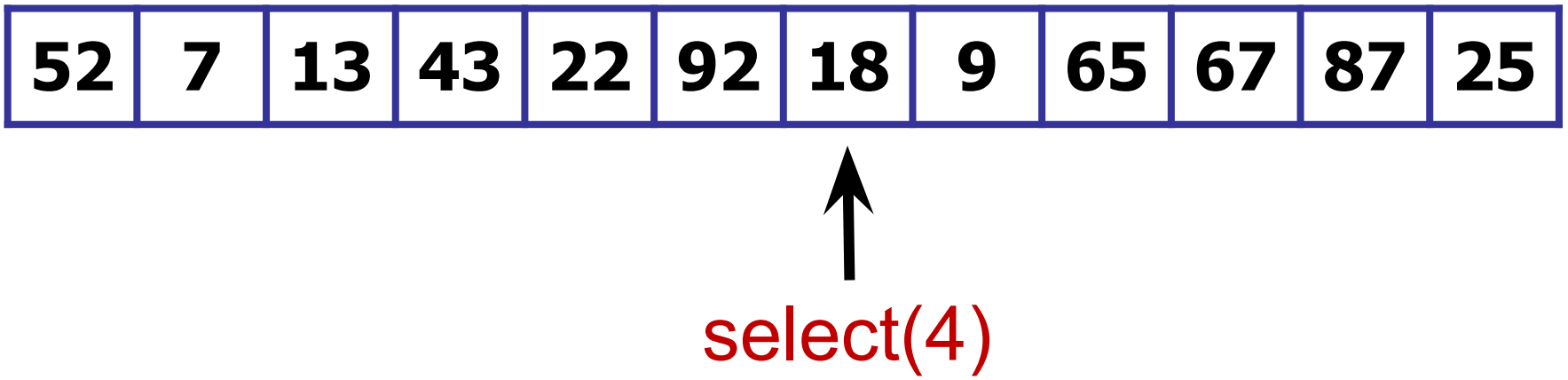
Order Statistics

Input

A set of integers.

Output: select(k)

The k^{th} item in the set.



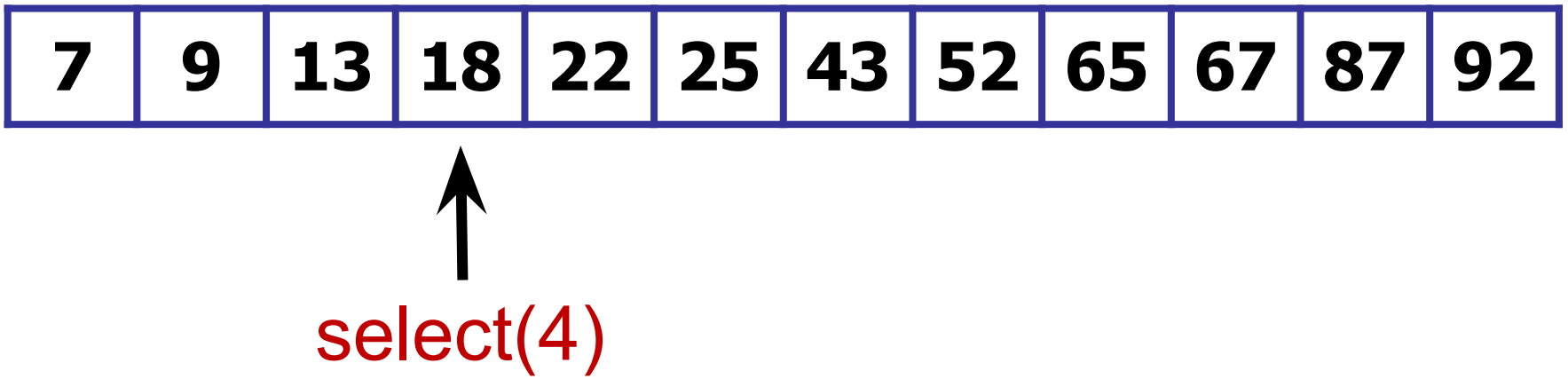
Order Statistics

Input

A set of integers.

Output: select(k)

The k^{th} item in the set.



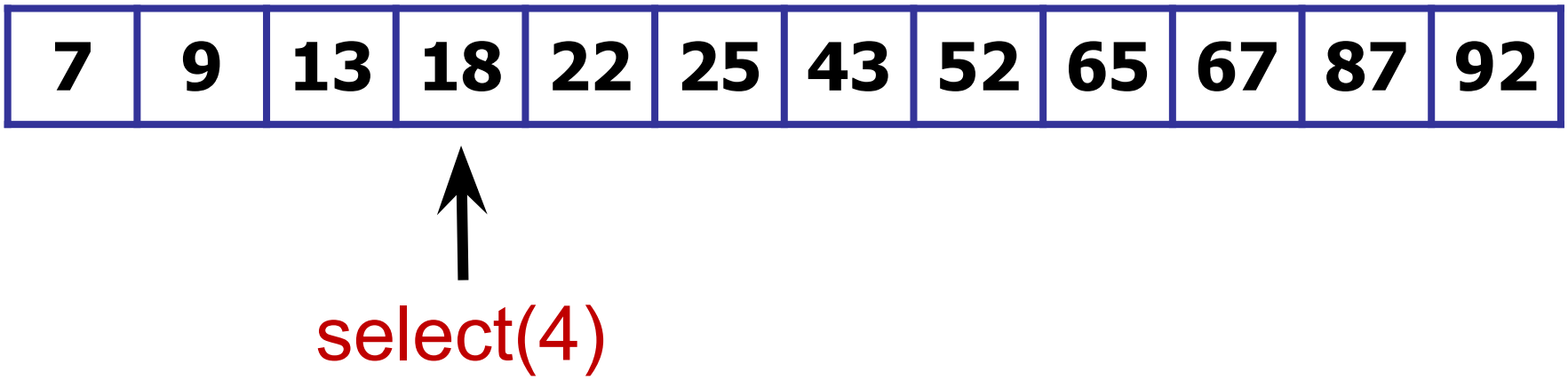
Order Statistics

Input

A set of integers.

Output: select(k) ⟶ Sort: O(n log n)

The k^{th} item in the set.



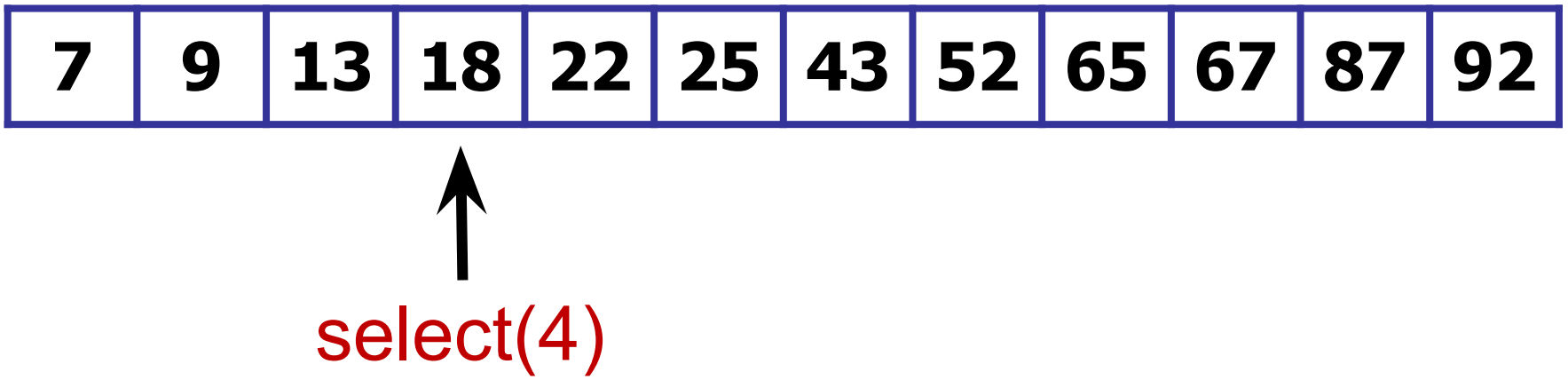
Order Statistics

Input

A set of integers.

Output: select(k) ⟶ QuickSelect: O(n)

The k^{th} item in the set.



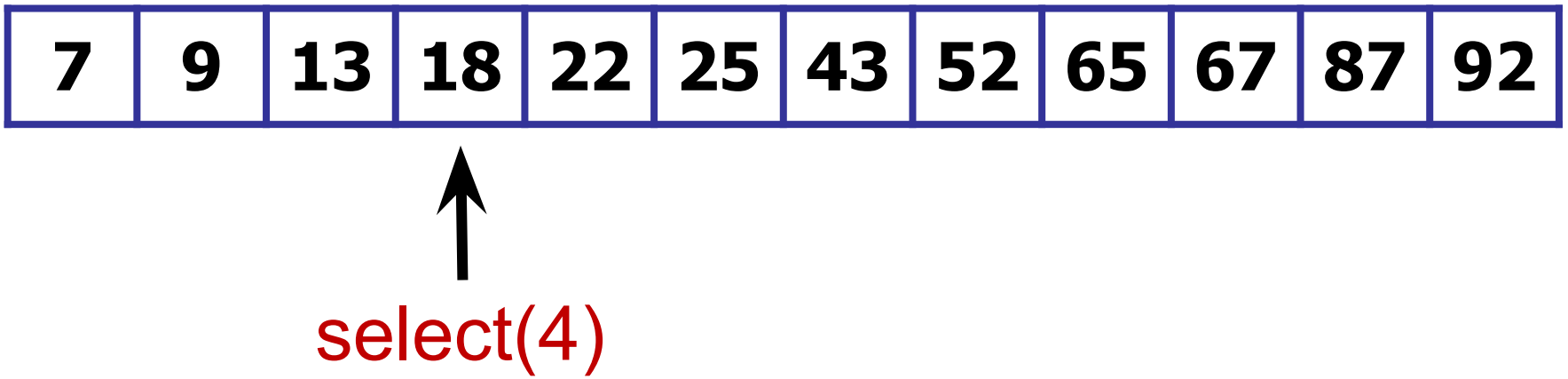
Order Statistics

Solution 1:

Sort: O(n log n)

Solution 2:

QuickSelect: O(n)



Order Statistics

Solution 1:

Preprocess: sort --- O(n log n)

Select: O(1)

Solution 2:

Preprocess: nothing --- O(1)

QuickSelect: O(n)

Order Statistics

Solution 1:

Preprocess: sort --- O(n log n)

Select: O(1)

Solution 2:

Preprocess: nothing --- O(1)

QuickSelect: O(n)

Question: What if we didn't have the entire array in advance?

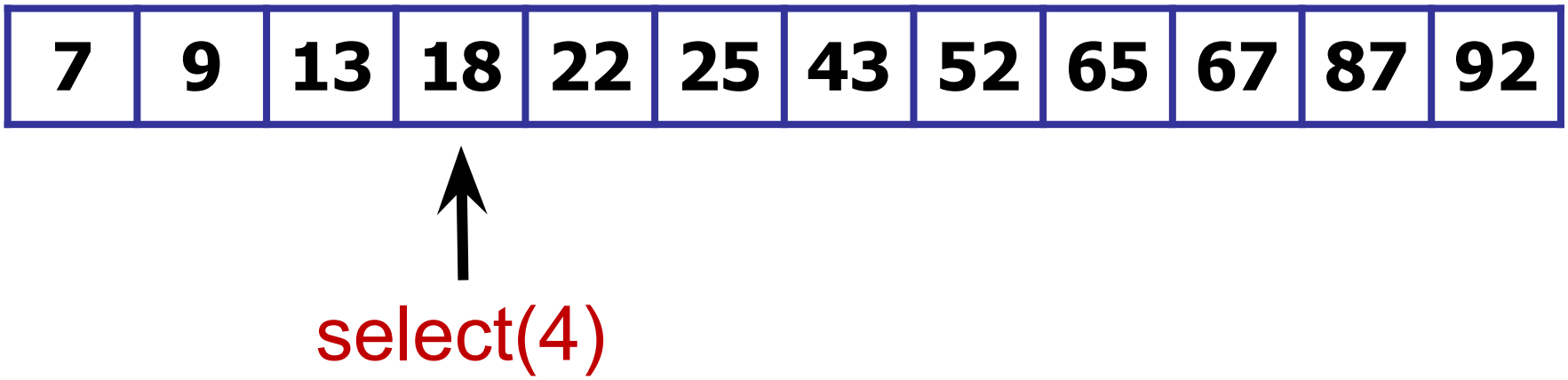
Dynamic Order Statistics

Implement a data structure that supports:

- insert(int key)
- delete(int key)

and also:

- select(int k)



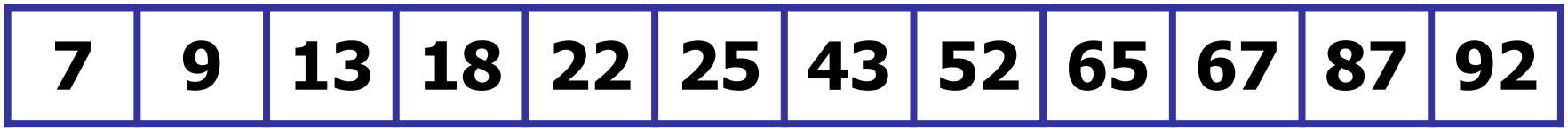
Dynamic Order Statistics

Solution 1:

Basic structure: sorted array A.

insert(int item): add item to sorted array A.

select(int k): return A[k]



Dynamic Order Statistics

Solution 2:

Basic structure: unsorted array A.

insert(int item): add item to end of array A.

select(int k): run QuickSelect(k)

7	9	13	18	22	25	43	52	65	67	87	92

When is it more efficient to maintain a sorted array (Solution 1)?

A. Always

B. When there are more inserts than selects.

C. When there are more selects than inserts.

D. Never

E. I'm confused.

Dynamic Order Statistics

	Insert	Select
Solution 1: Sorted Array	O(n)	O(1)
Solution 2: Unsorted Array	O(1)	O(n)

7	9	13	18	22	25	43	52	65	67	87	92

Dynamic Order Statistics

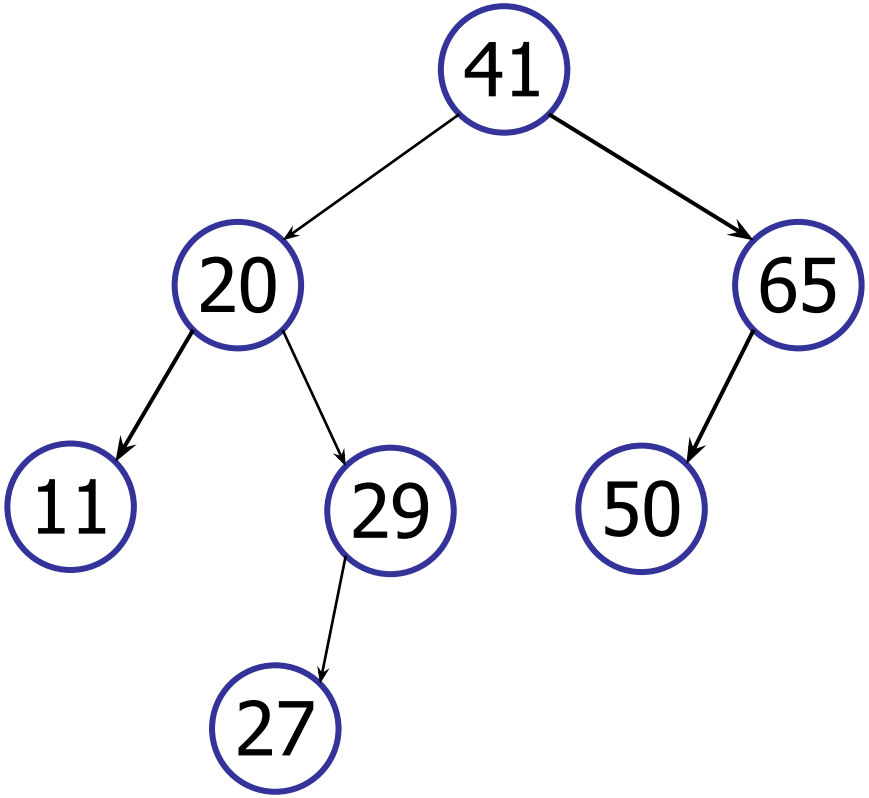
	Insert	Select
Solution 1: Sorted Array	O(n)	O(1)
Solution 2: Unsorted Array	O(1)	O(n)

expected running time if using randomised Quickselect

7	9	13	18	22	25	43	52	65	67	87	92

Dynamic Order Statistics

Today: use a (balanced) tree

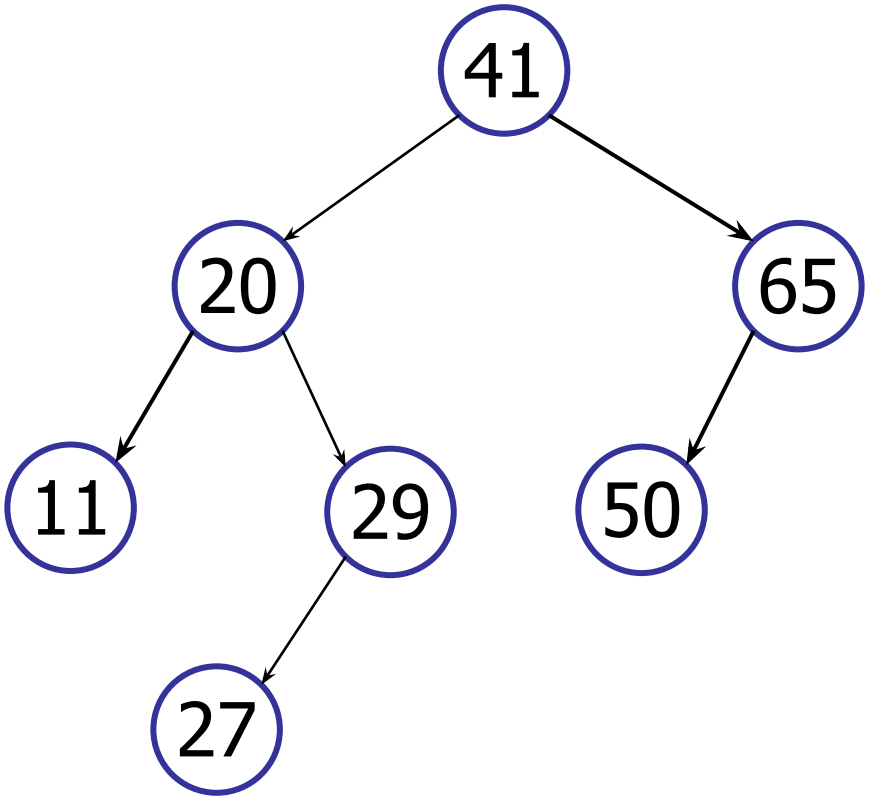


11	20	27	29	41	50	65

Dynamic Order Statistics

Simple solution: traversal

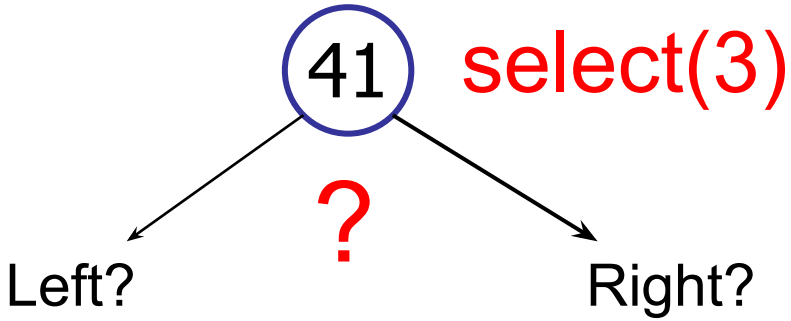
select(k): O(k)
in-order traversal



11	20	27	29	41	50	65

Dynamic Order Statistics

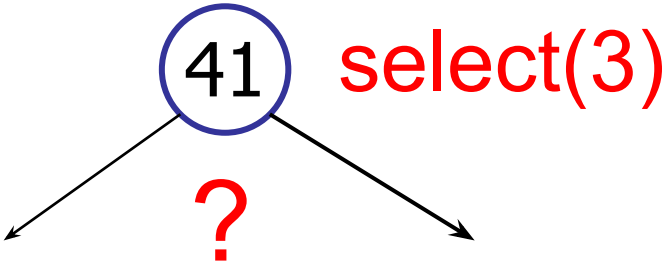
How to find the right item?



Dynamic Order Statistics

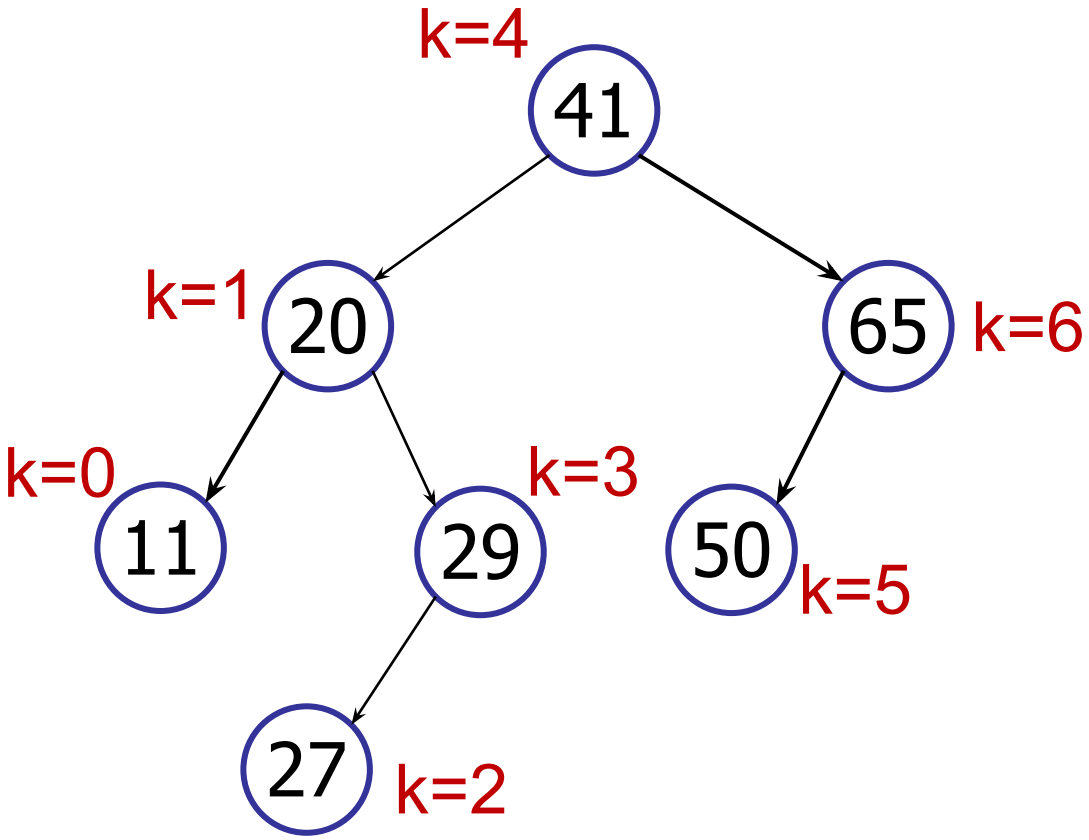
Augment!

What extra information would help?



Dynamic Order Statistics

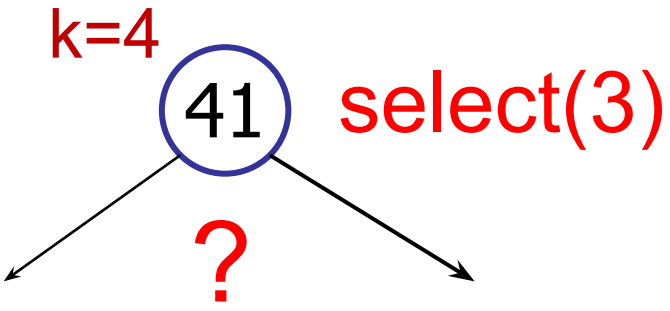
Idea: store rank in every node



11	20	27	29	41	50	65

Dynamic Order Statistics

Idea: store rank in every node

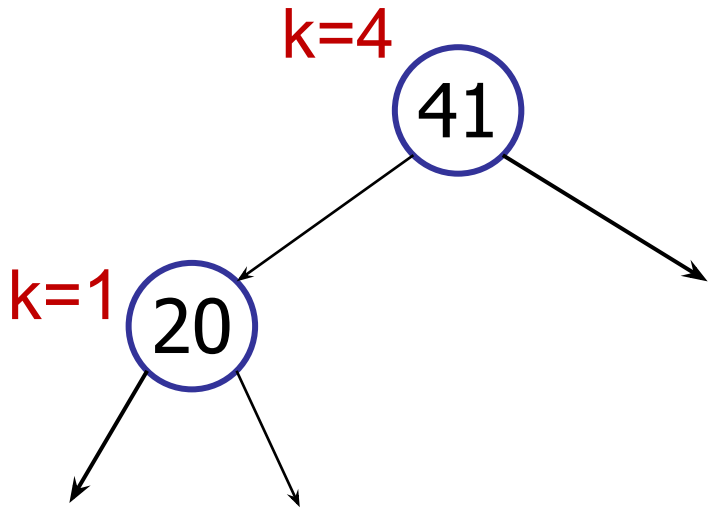


11	20	27	29	41	50	65

Dynamic Order Statistics

Idea: store rank in every node

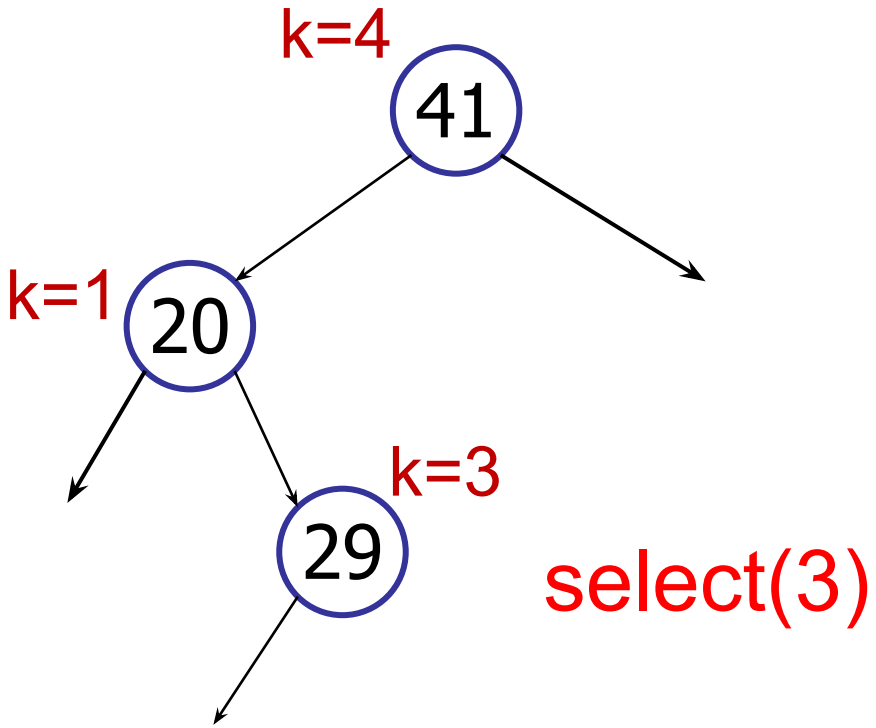
select(3)



11	20	27	29	41	50	65

Dynamic Order Statistics

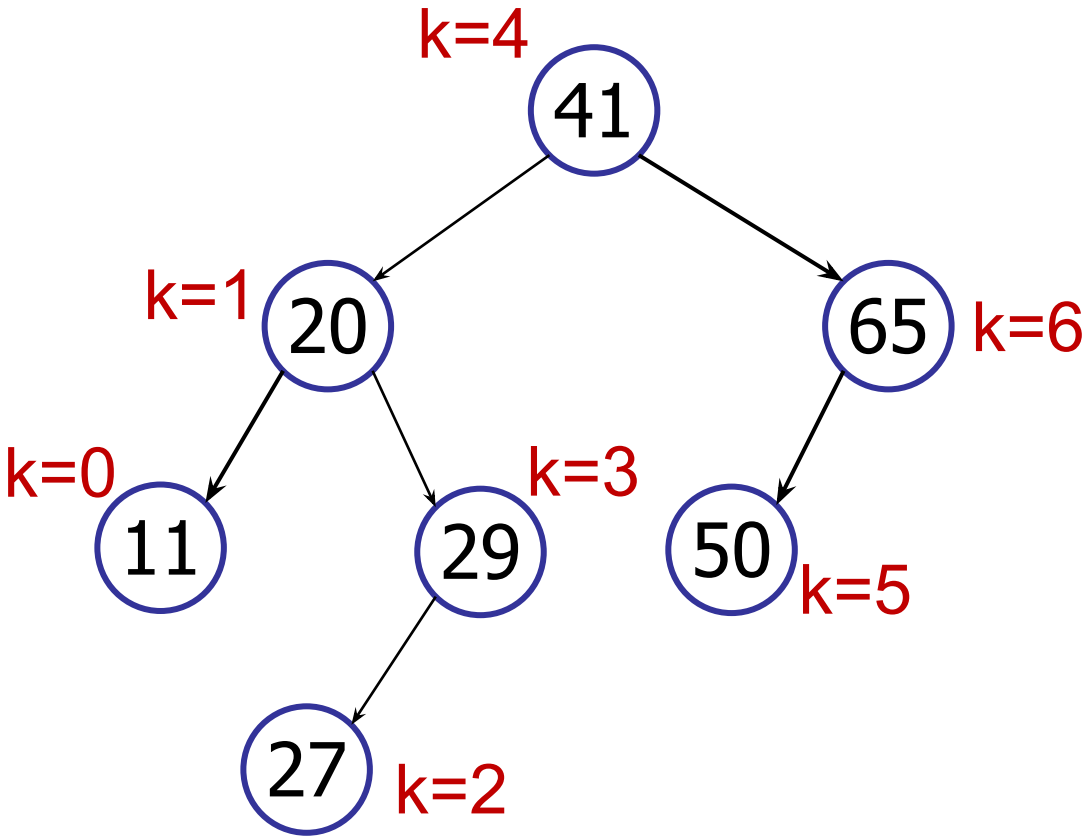
Idea: store rank in every node



11	20	27	29	41	50	65

Dynamic Order Statistics

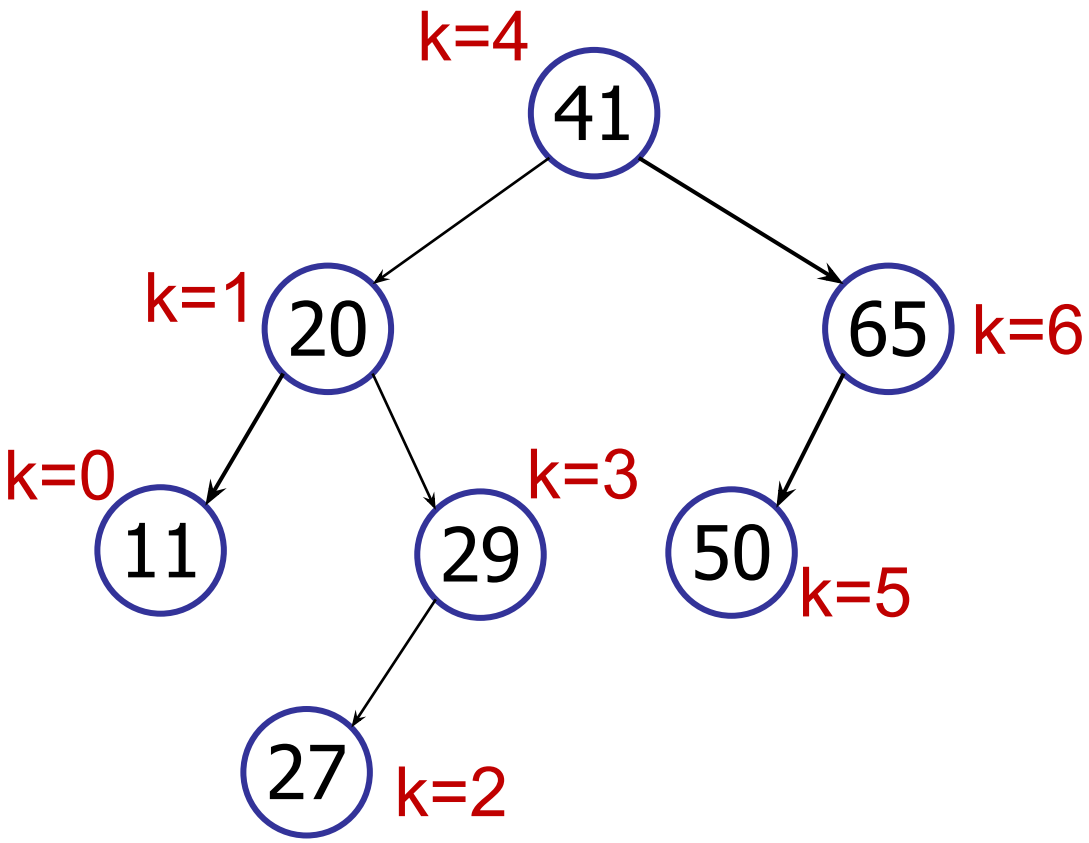
Idea: store rank in every node



11	20	27	29	41	50	65

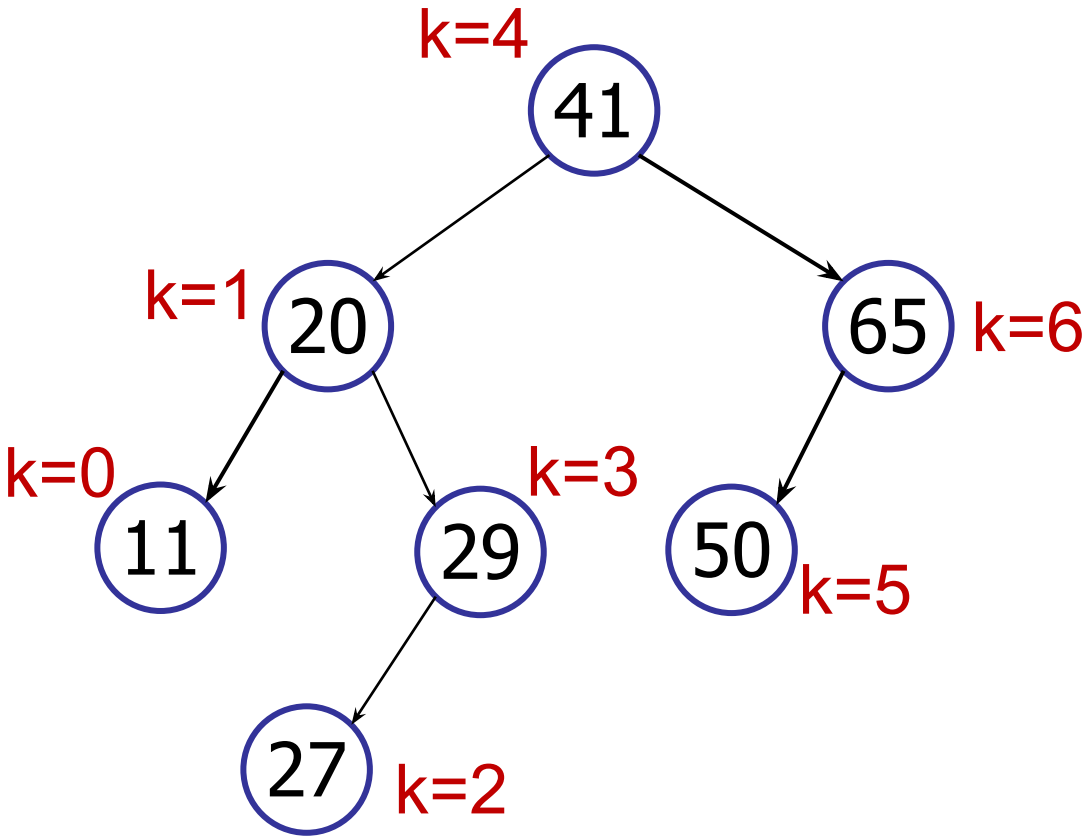
Dynamic Order Statistics

Question: What goes wrong if you store ranks on every node??



Dynamic Order Statistics

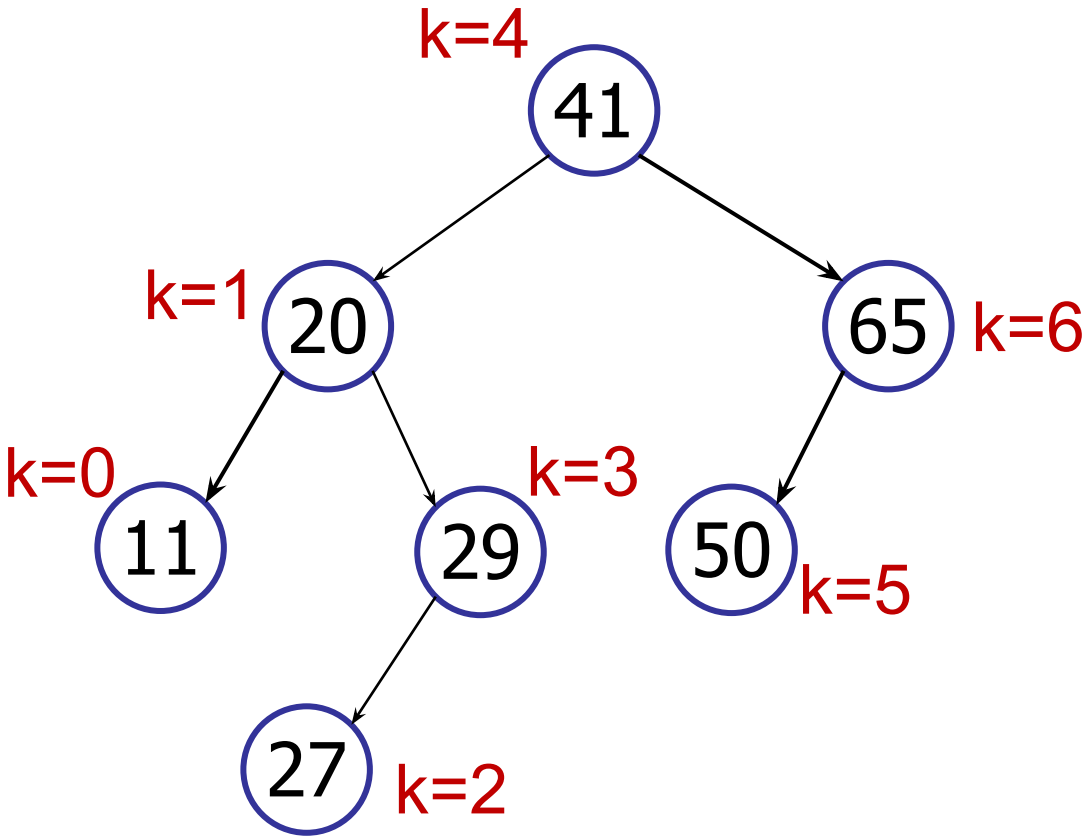
Idea: store rank in every node



Problem: insert(5)

Dynamic Order Statistics

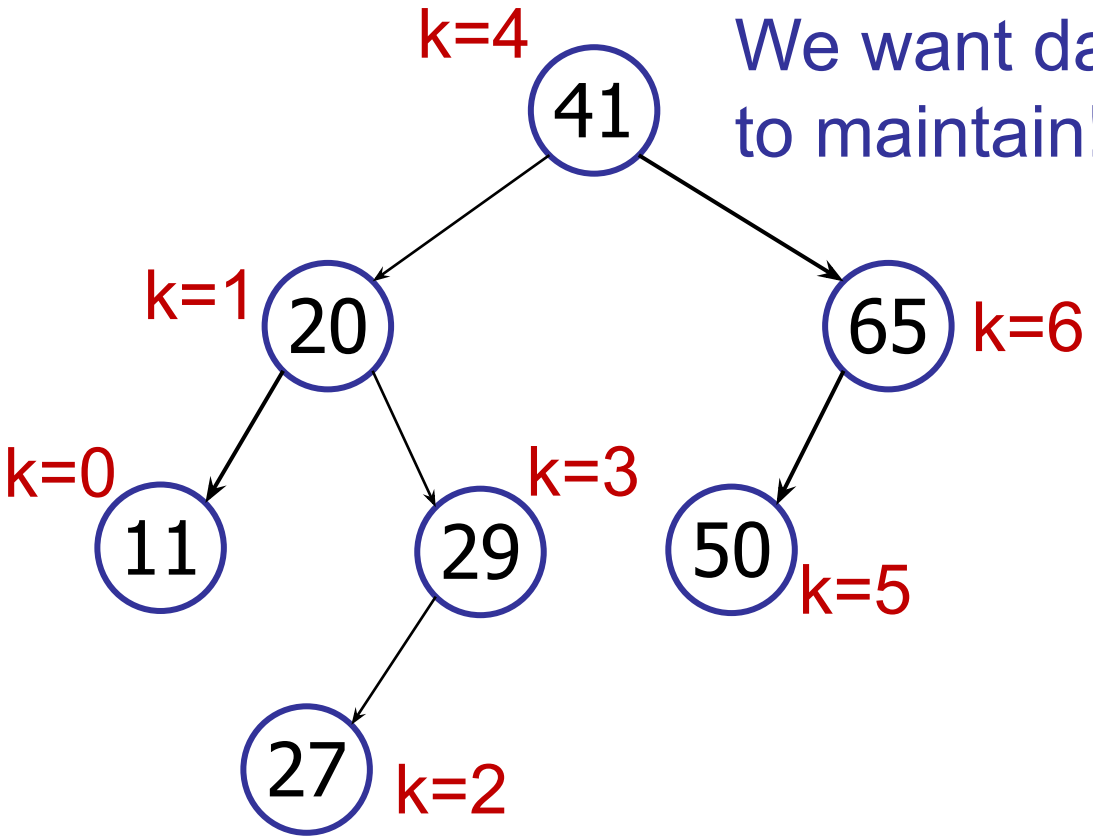
Idea: store rank in every node



Problem: insert(5) requires updating *all* the ranks!

Dynamic Order Statistics

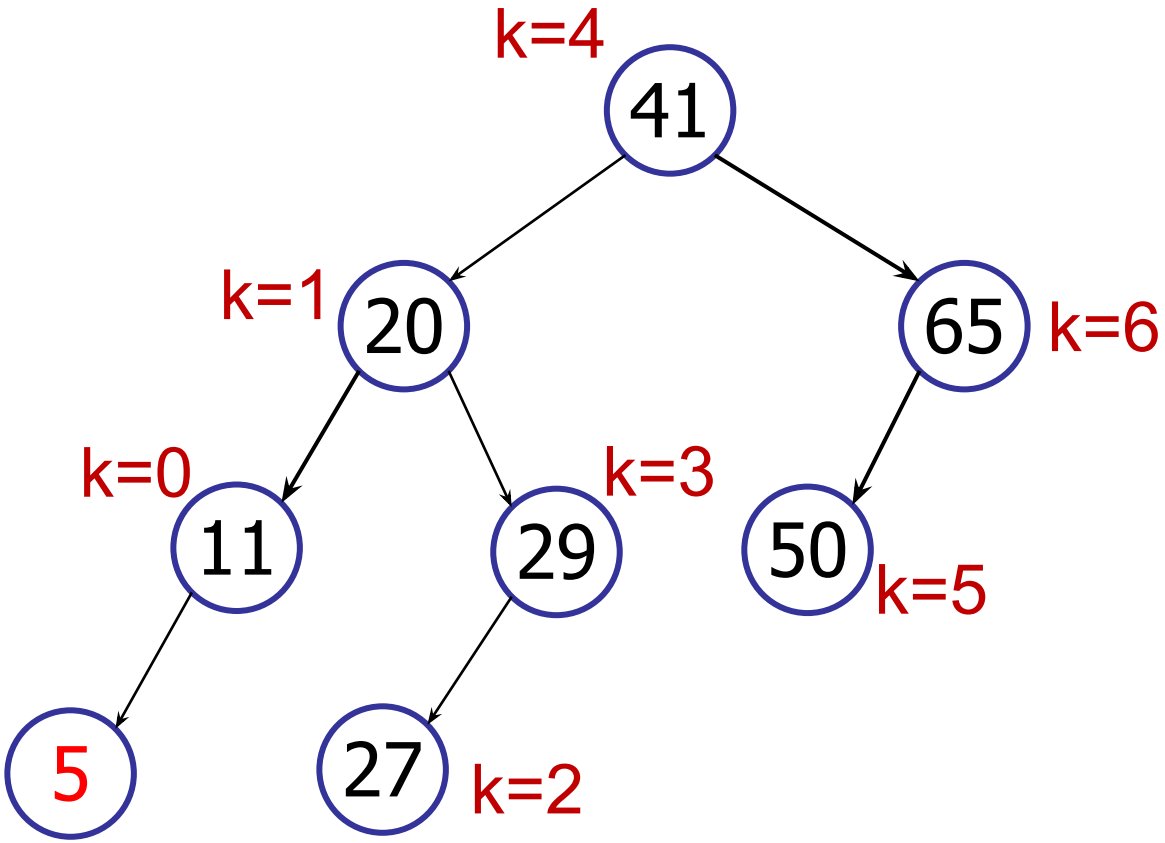
Idea: store rank in every node



Problem: insert(5) requires updating *all* the ranks!

Dynamic Order Statistics

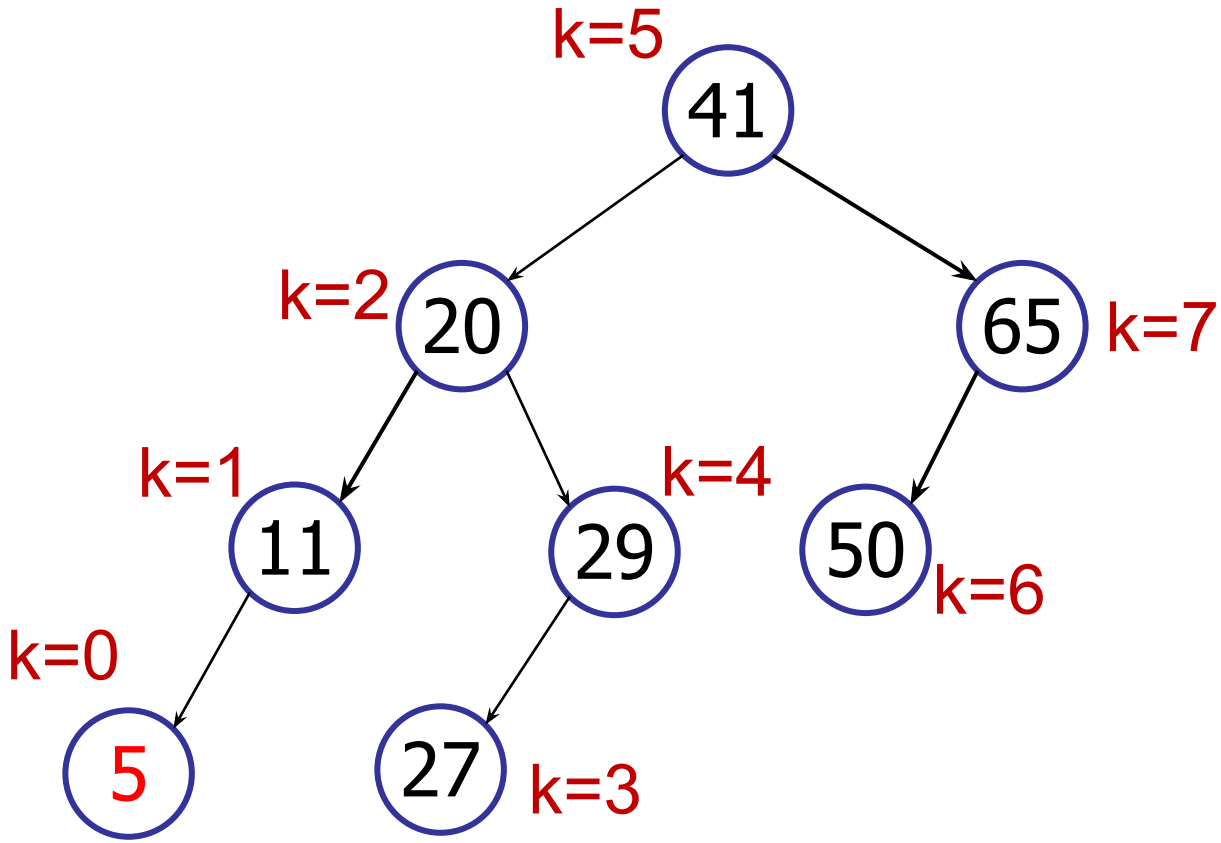
Idea: store rank in every node



5	11	20	27	29	41	50	65

Dynamic Order Statistics

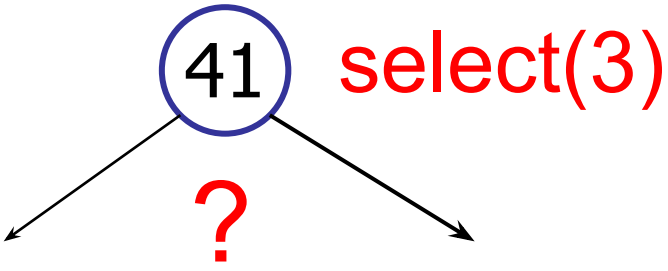
Conclusion: too expensive to store rank in every node!



5	11	20	27	29	41	50	65

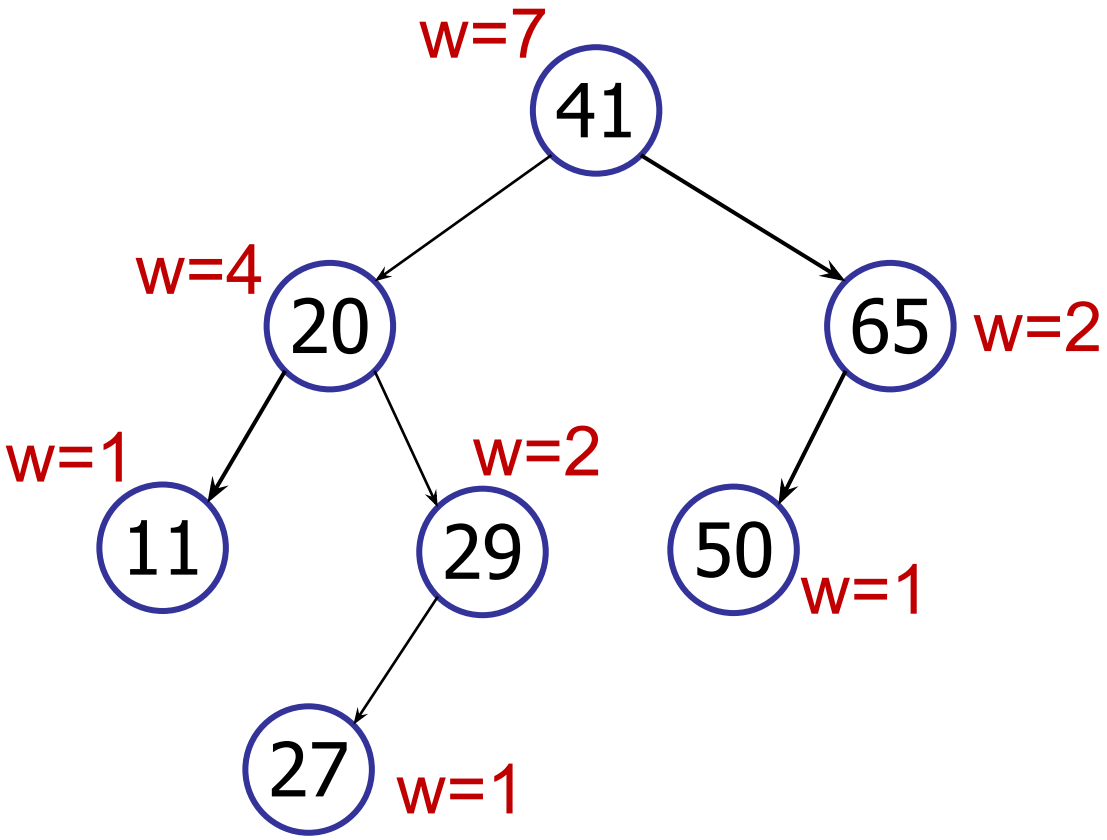
Dynamic Order Statistics

What should we store in each node?



Dynamic Order Statistics

Idea: store *size* of sub-tree in every node



Dynamic Order Statistics

Idea: store size of sub-tree in every node

The weight of a node is the size of the tree rooted at that node.

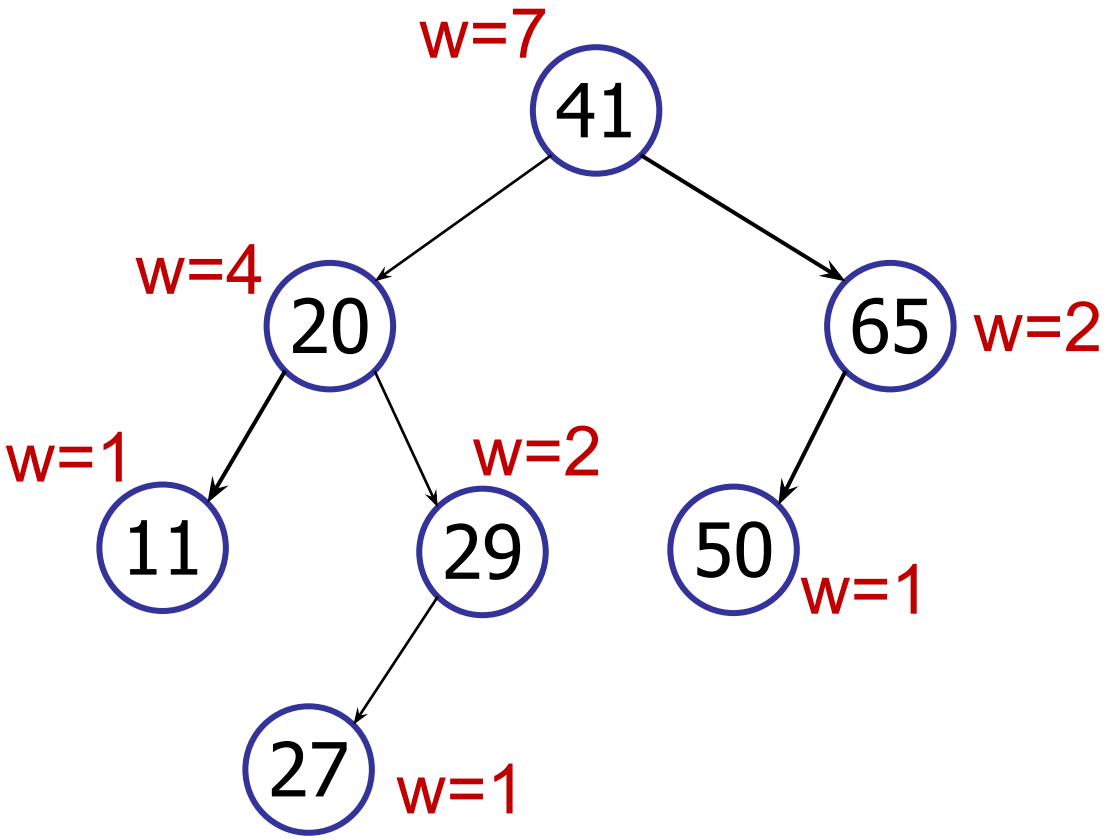
Define weight:

w(leaf) = 1

w(v) = w(v.left) + w(v.right) + 1

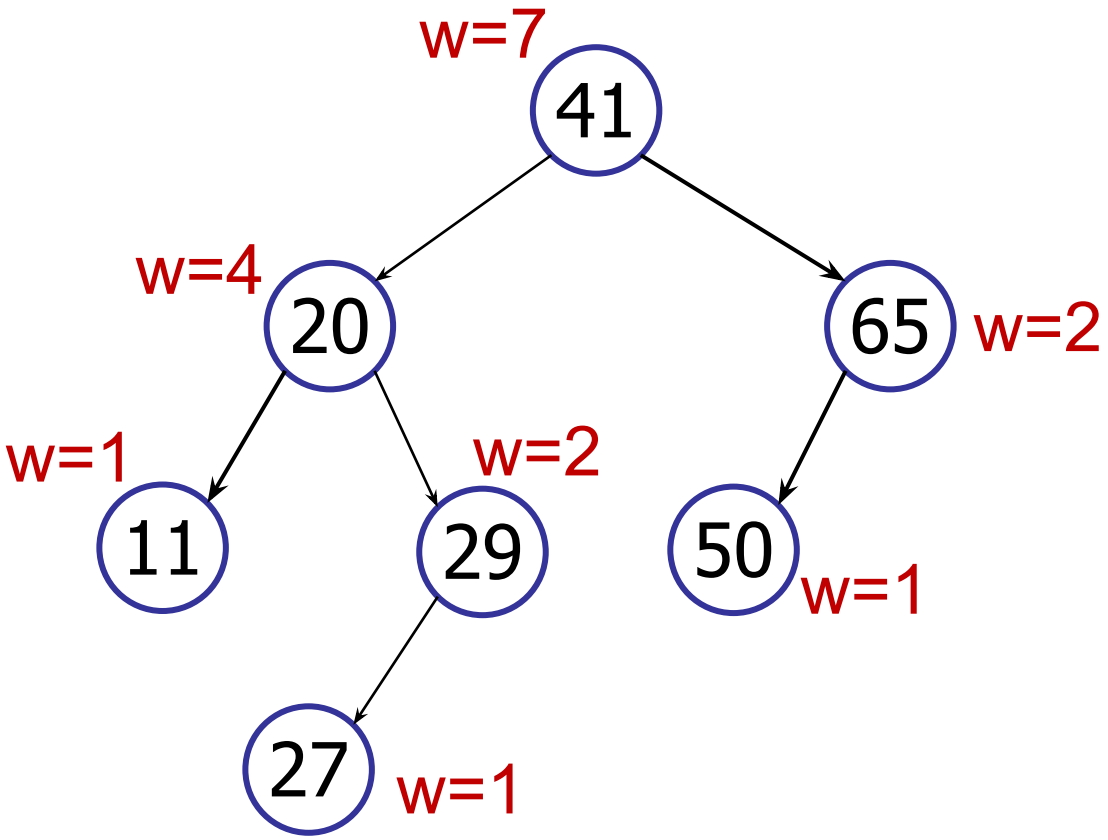
Dynamic Order Statistics

Idea: store *size* of sub-tree in every node



Dynamic Order Statistics

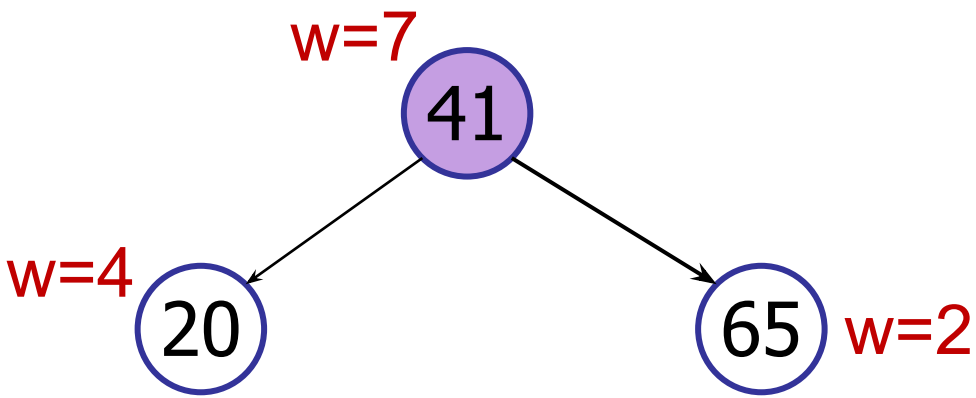
Idea: store *size* of sub-tree in every node



We can use this size to tell how many keys to our left!

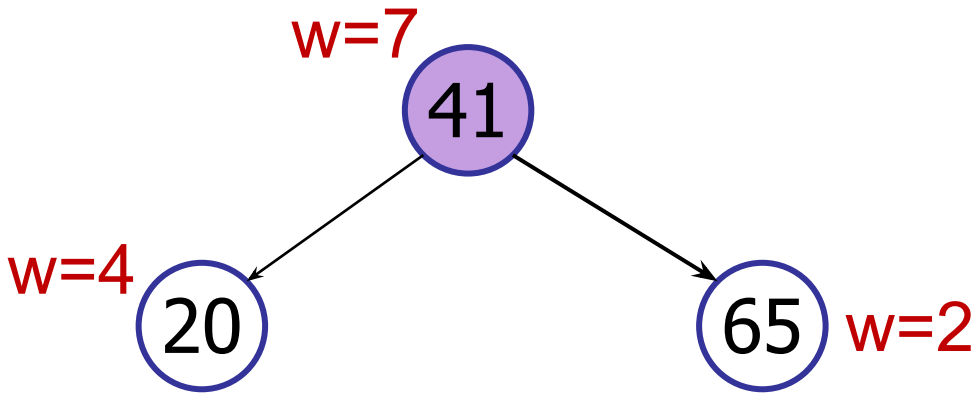
Dynamic Order Statistics

Example: select(3)



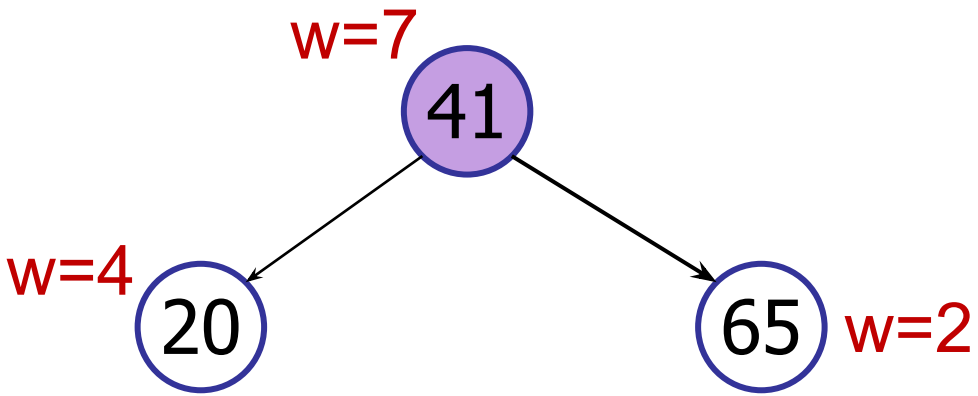
What is the rank of 41?

1. 1
2. 3
3. 5 ✓
4. 7
5. 9
6. Can't tell.



Dynamic Order Statistics

Example: select(3)

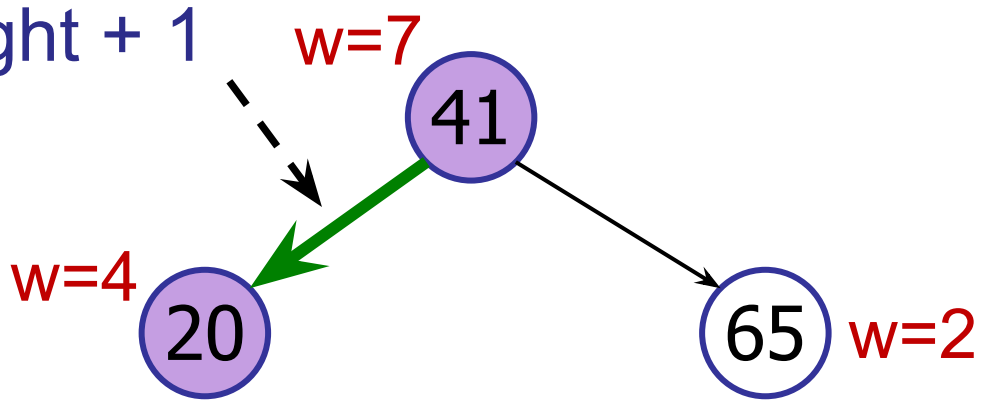


"rank in subtree" = left.weight + 1

Dynamic Order Statistics

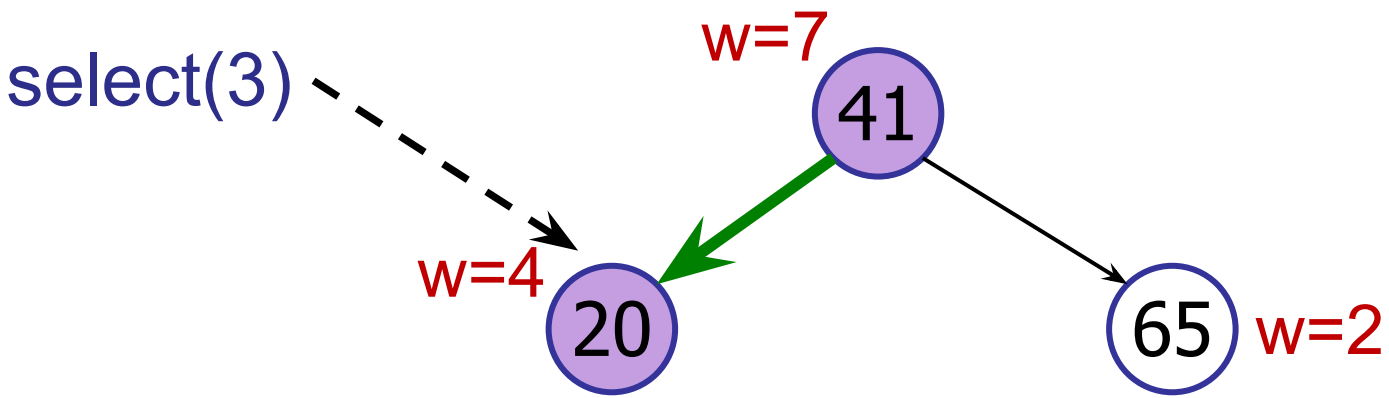
Example: select(3)

3 < left.weight + 1
Go left!



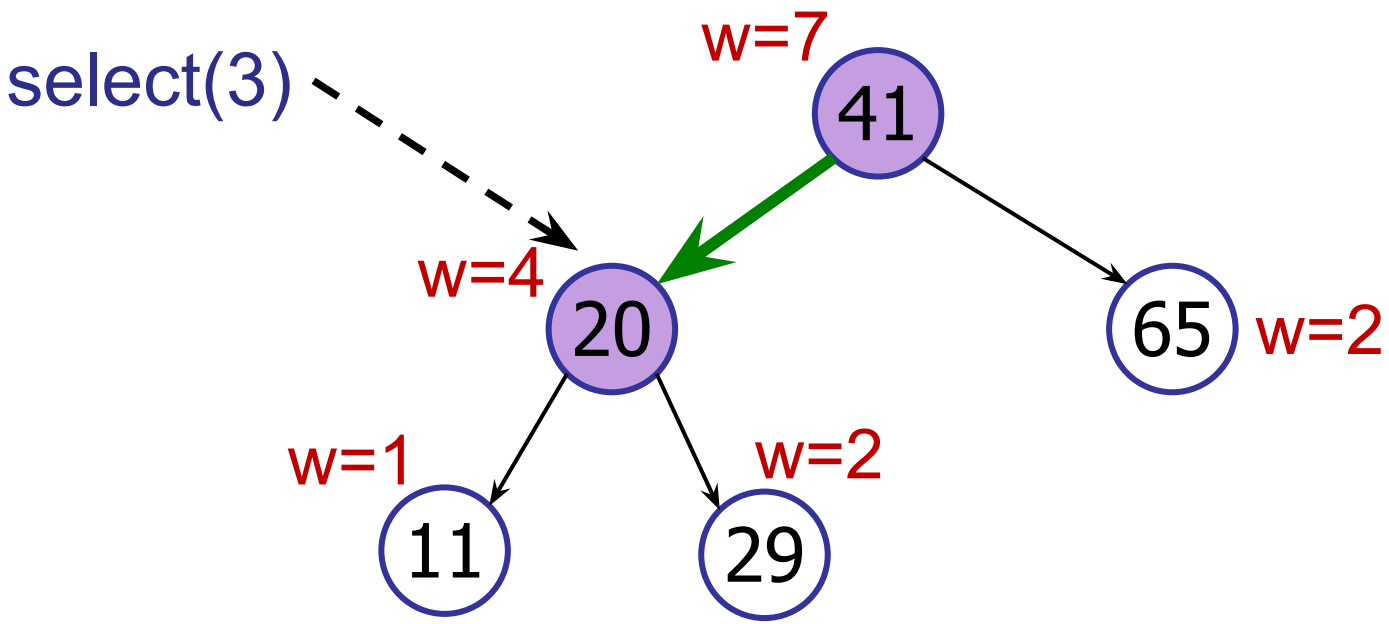
Dynamic Order Statistics

Example: select(3)



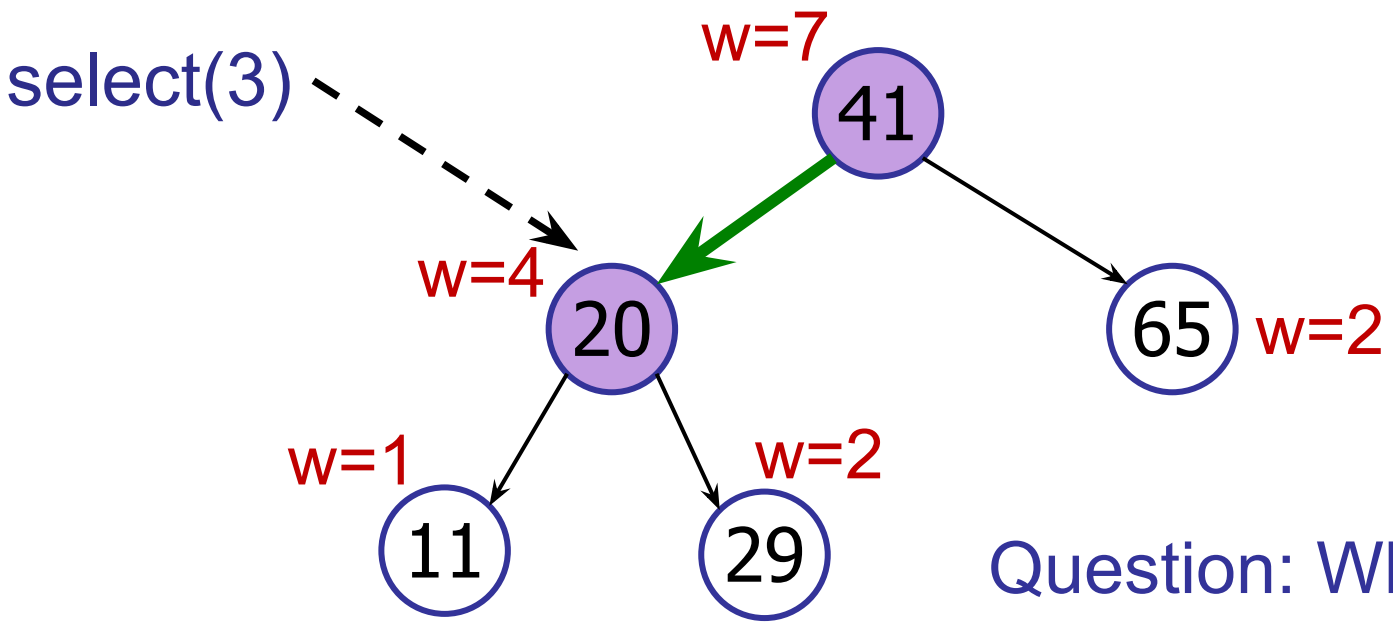
Dynamic Order Statistics

Example: select(3)



Dynamic Order Statistics

Example: select(3)



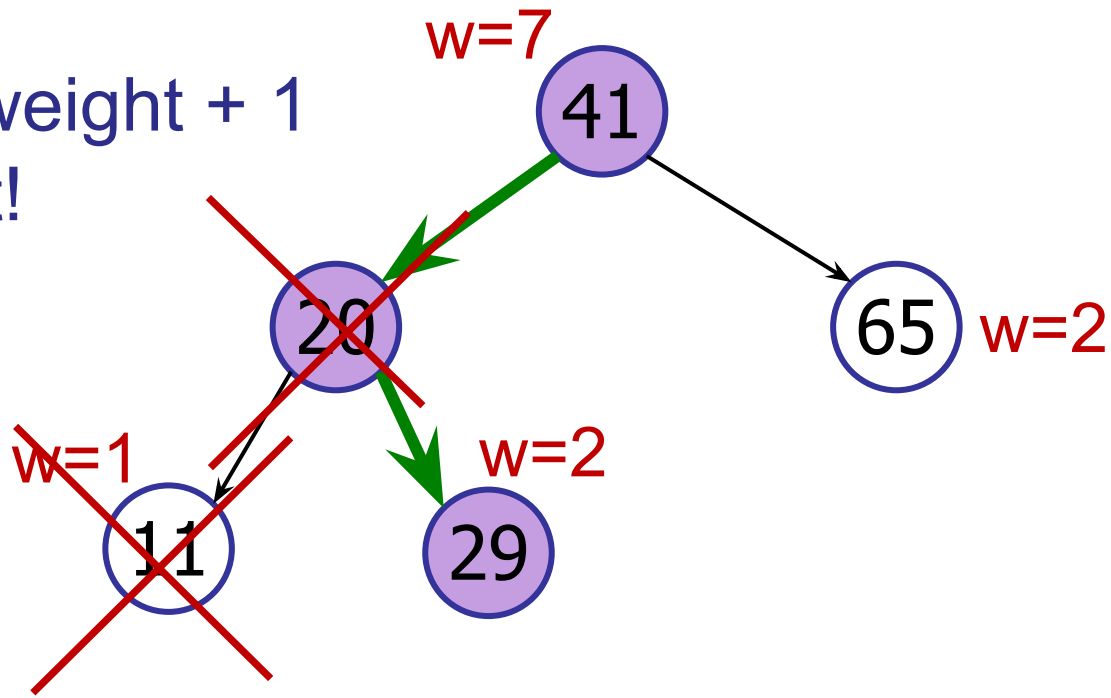
Question: Which side do we recurse on?

Which rank should we recurse on?

Dynamic Order Statistics

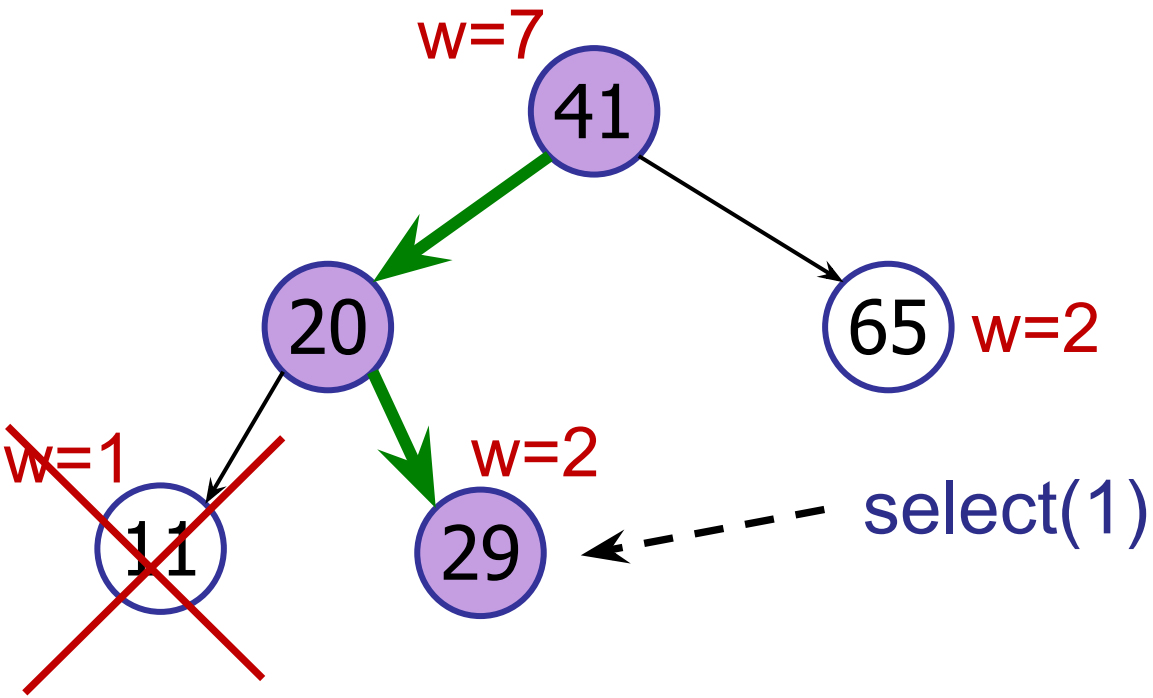
Example: select(3)

3 > left.weight + 1
Go right!



Dynamic Order Statistics

Example: select(3)



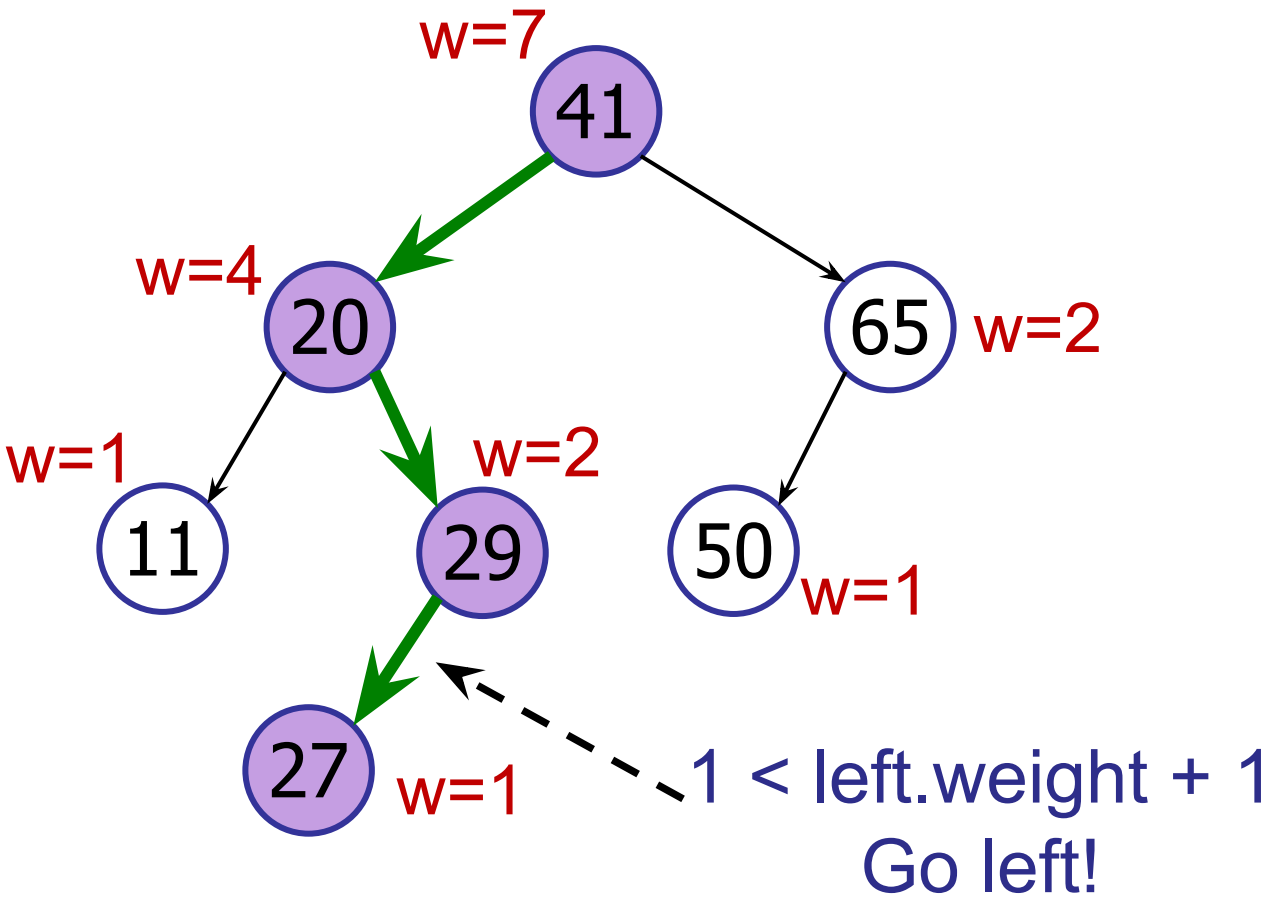
Item to select:

3 – (left.weight + 1) =

3 – (1 + 1) = 1

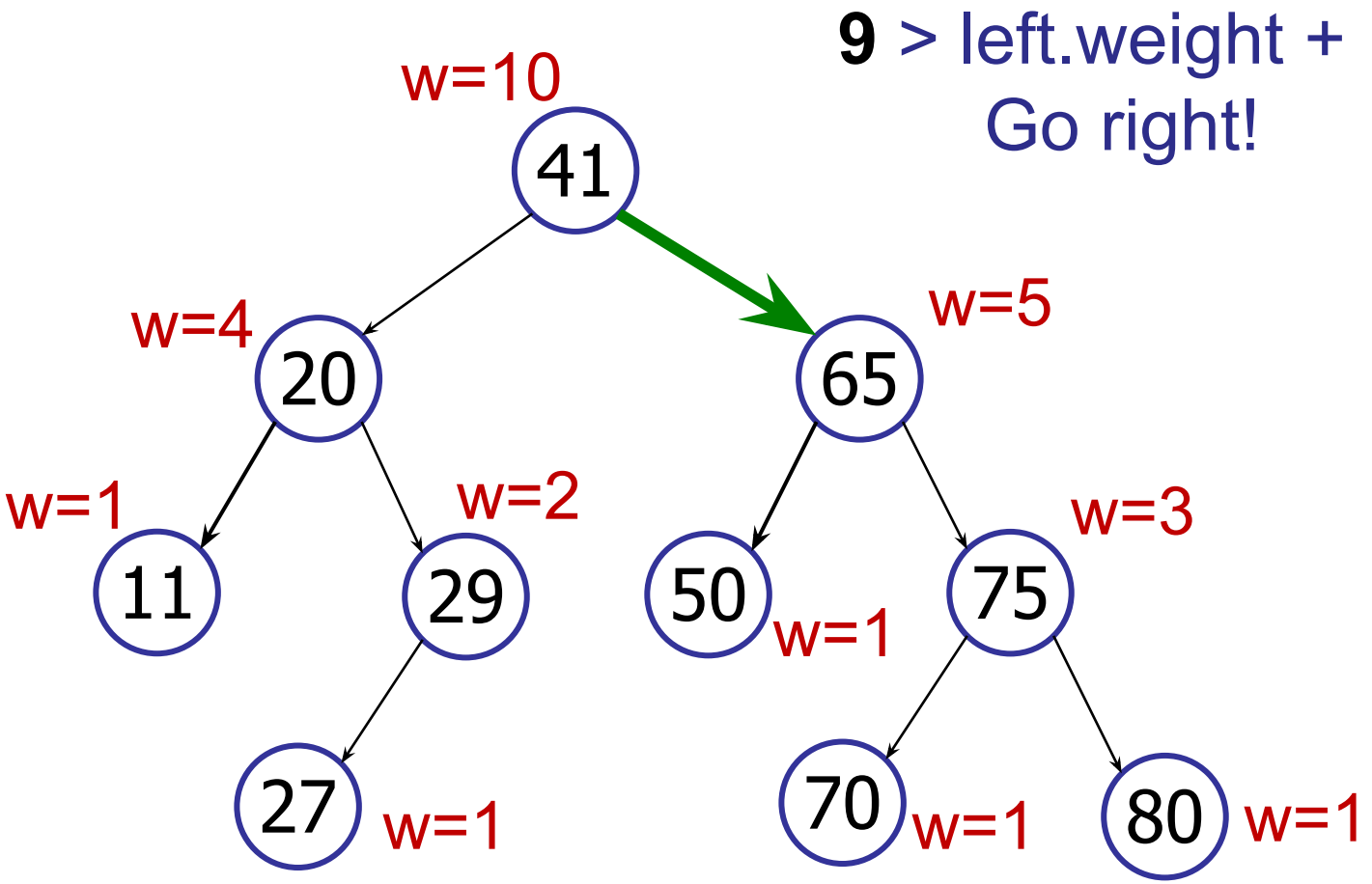
Dynamic Order Statistics

Example: select(3)

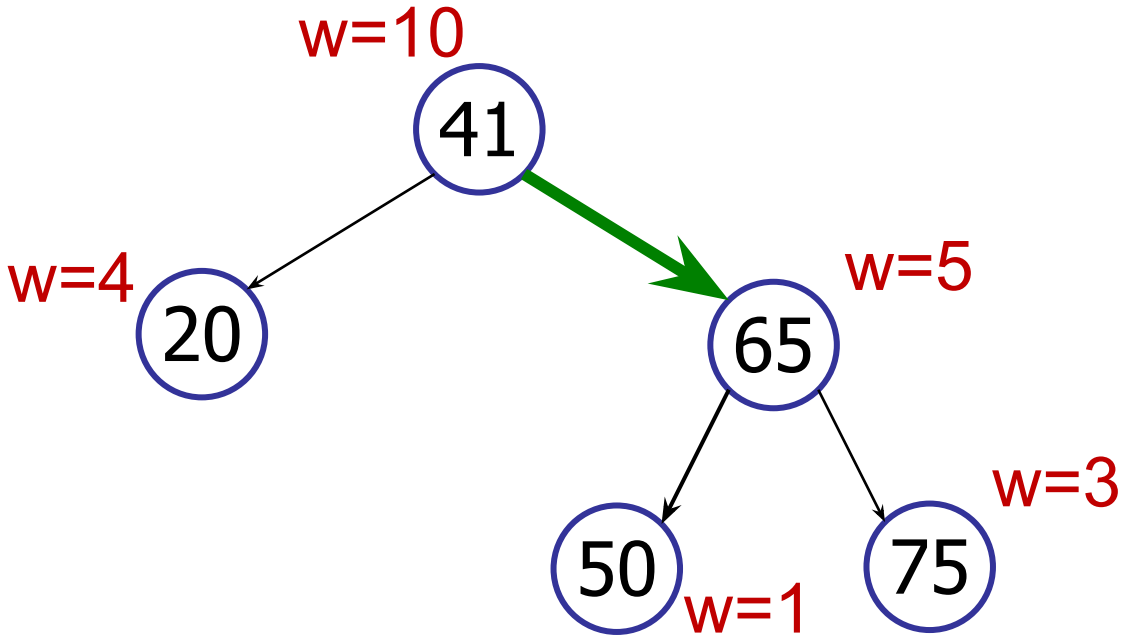


Dynamic Order Statistics

Example: select(9)



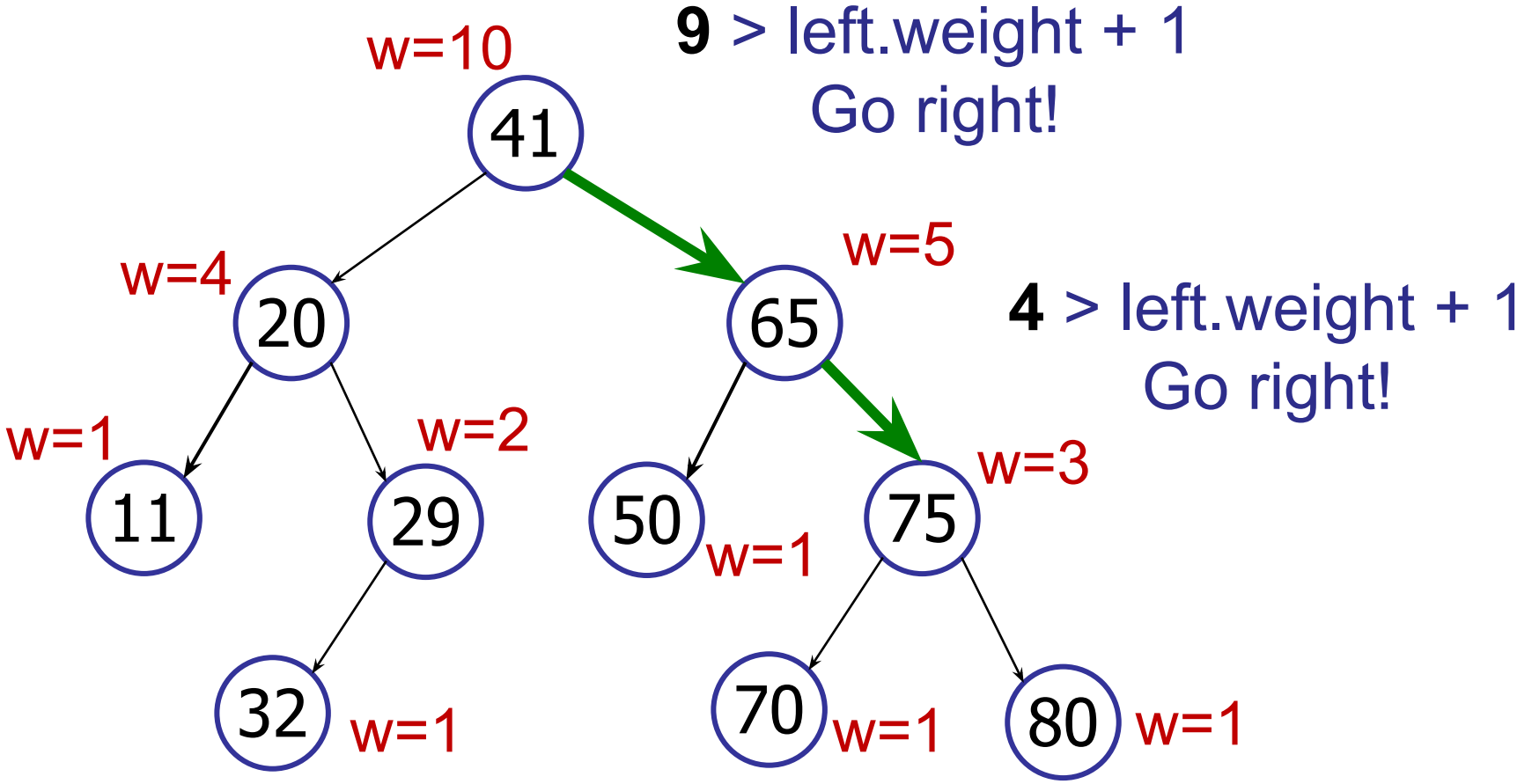
select(9)



1. Go left at 65
2. Go right at 65
3. Stop at 65
4. I'm confused

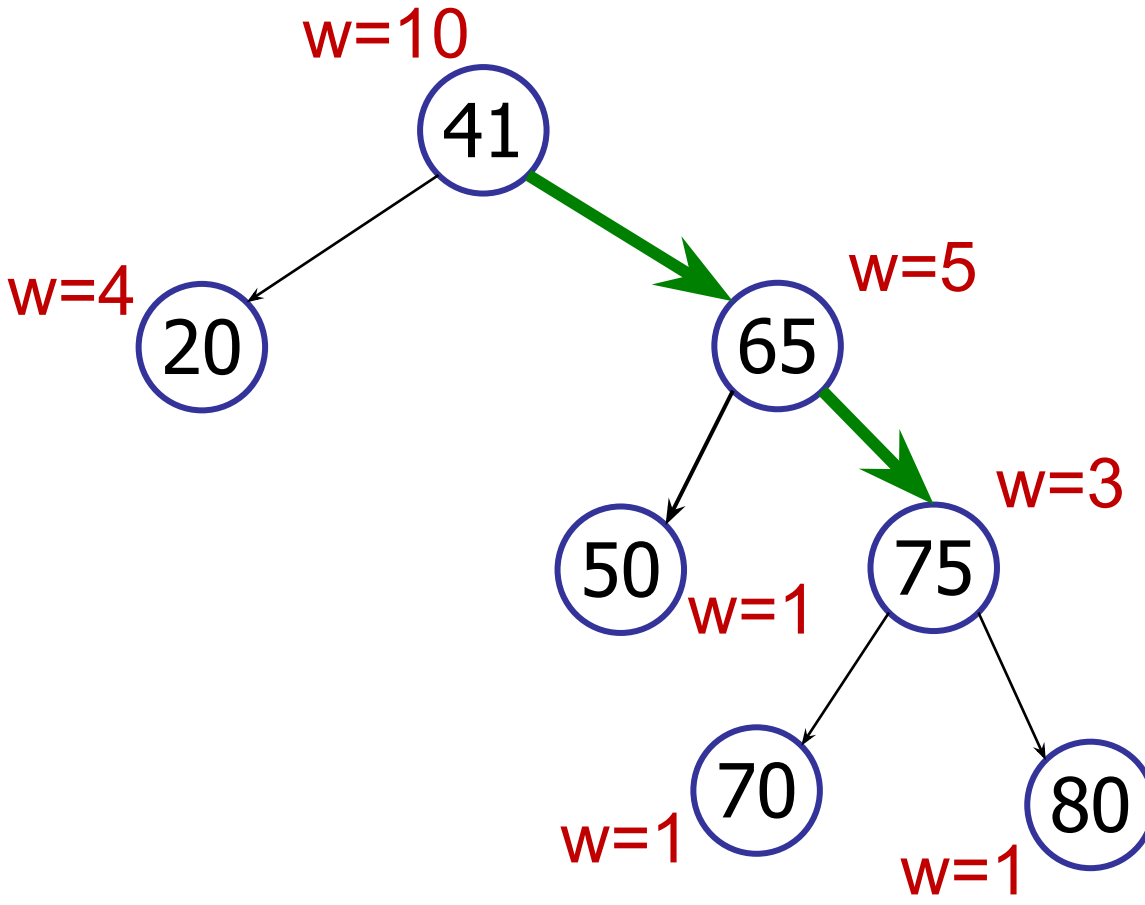
Dynamic Order Statistics

select(9)



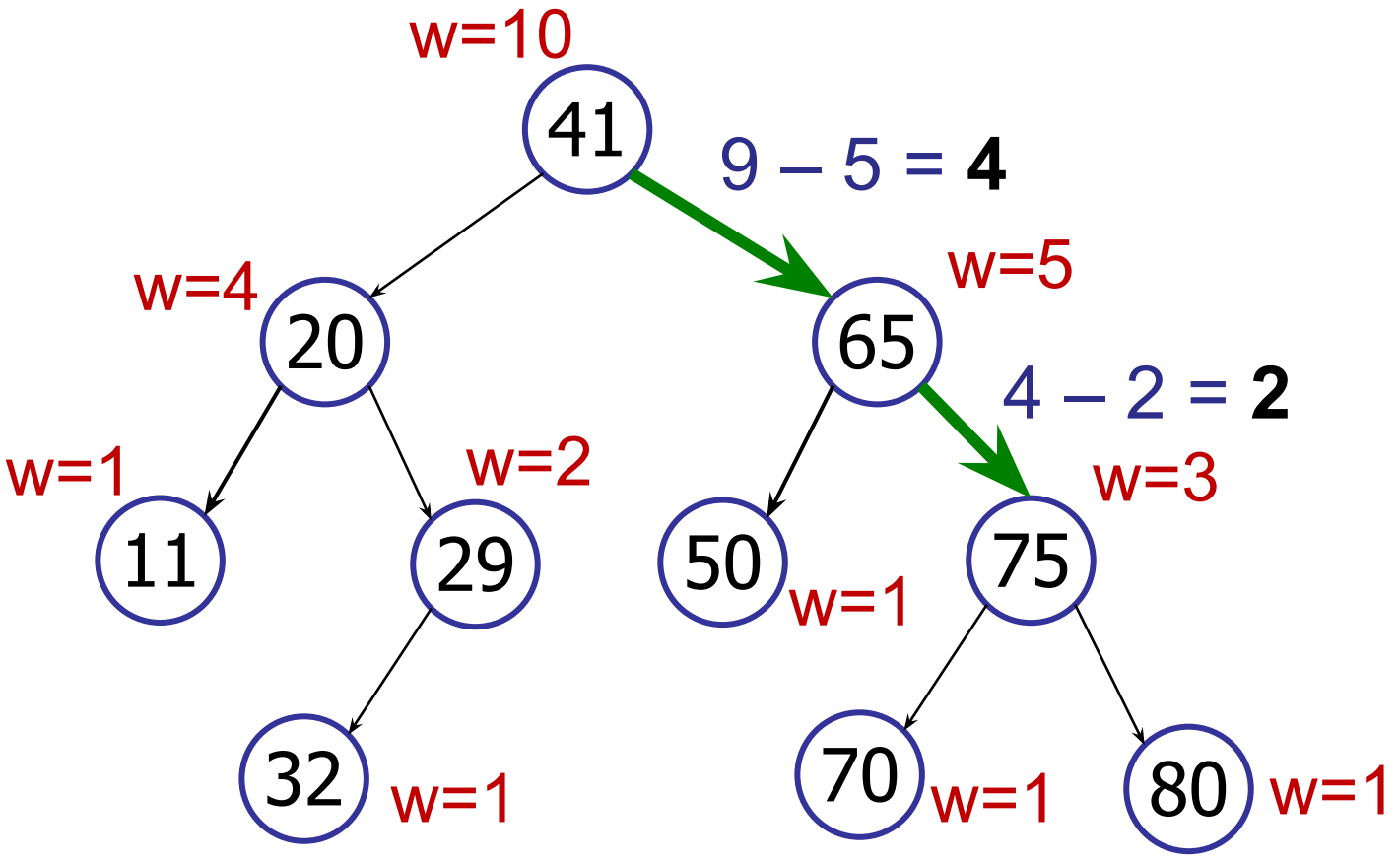
select(9)

1. Go left at 75
2. Go right at 75
3. Stop at 75 ✓
4. I'm confused



Dynamic Order Statistics

select(9)



Dynamic Order Statistics

select(k)

```
rank = m_left.weight + 1;
if (k == rank) then
    return v;
else if (k < rank) then
    return m_left.select(k);
else if (k > rank) then
    return m_right.select(k-rank);
```

Dynamic Order Statistics

select(k)

```
rank = m_left.weight + 1;
if (k == rank) then
    return v;
else if (k < rank) then
    return m_left.select(k);
else if (k > rank) then
    return m_right.select(k-rank);
```

Found the item we're looking for

Dynamic Order Statistics

select(k)

```
rank = m_left.weight + 1;
if (k == rank) then
    return v;
else if (k < rank) then
    return m_left.select(k);
else if (k > rank) then
    return m_right.select(k-rank);
```

Rank we're looking for belongs in left sub-tree

Dynamic Order Statistics

select(k)

```
rank = m_left.weight + 1;
if (k == rank) then
    return v;
else if (k < rank) then
    return m_left.select(k);
else if (k > rank) then
    return m_right.select(k-rank);
```

Rank we're looking for belongs in left sub-tree

Dynamic Order Statistics

select(k) : finds the node with rank k

Example: find the 10th tallest student in the class.

Dynamic Order Statistics

select(k) : finds the node with rank k

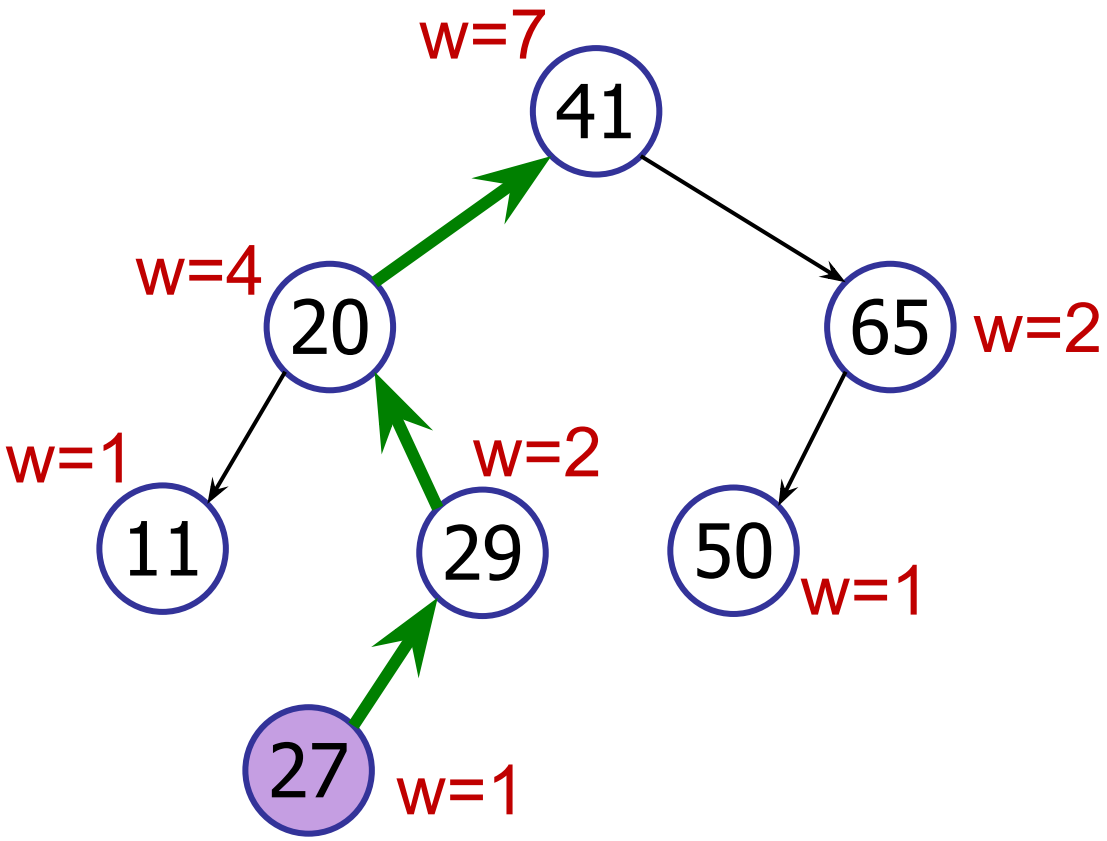
Example: find the 10th tallest student in the class.

rank(v) : computes the rank of a node v

Example: determine the percentile of Johnny's height. Is Johnny in the 10^{th} percentile or the 90^{th} percentile?

Dynamic Order Statistics

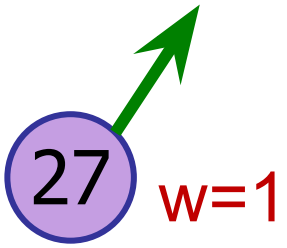
Example: rank(27)



rank = 1

Dynamic Order Statistics

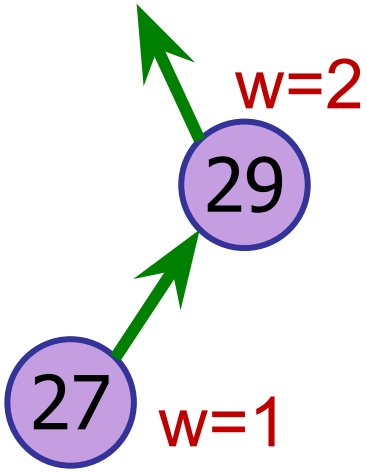
Example: rank(27)



rank = 1

Dynamic Order Statistics

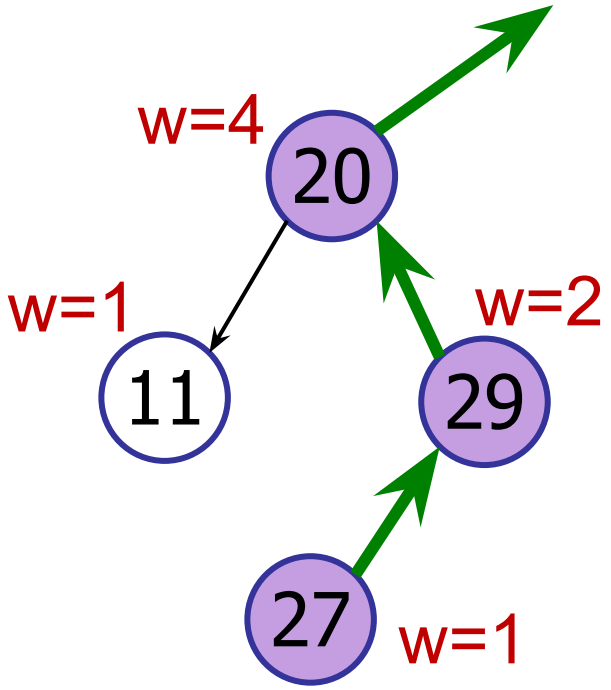
Example: rank(27)



rank = 1

Dynamic Order Statistics

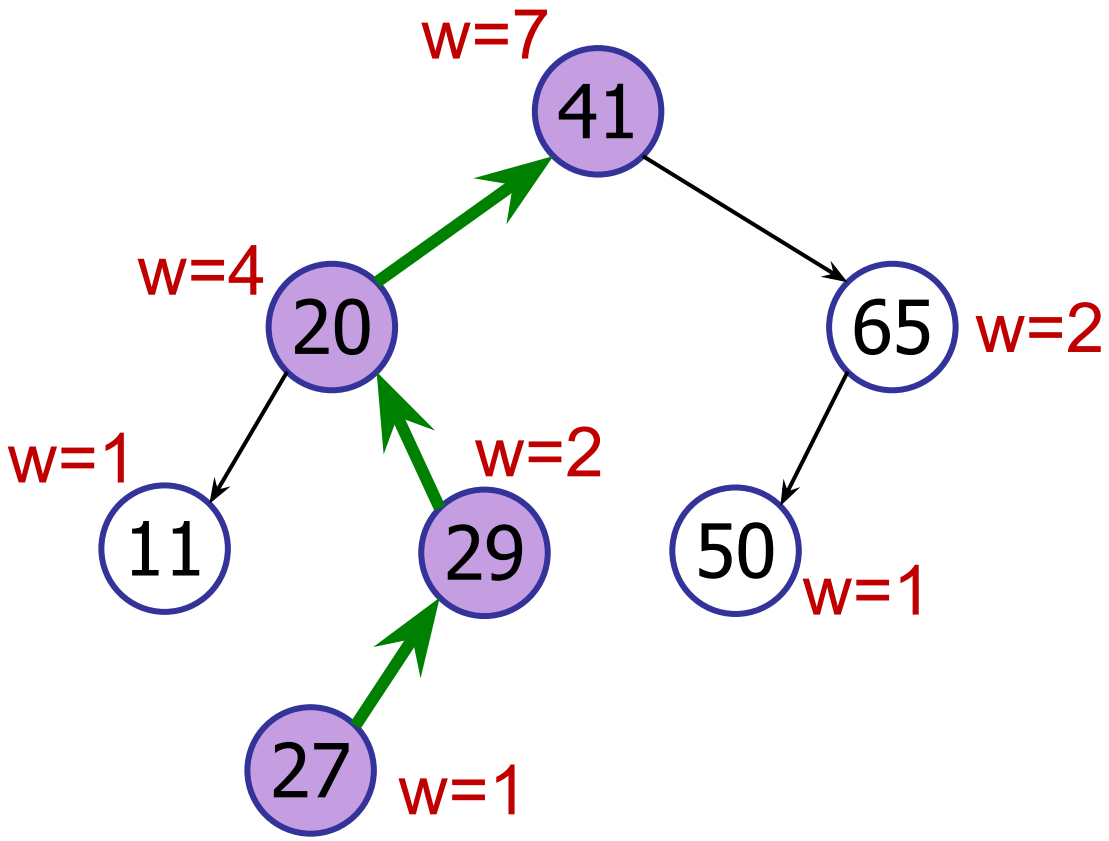
Example: rank(27)



rank = 1 + 2

Dynamic Order Statistics

Example: rank(27)



rank = 1 + 2 = 3

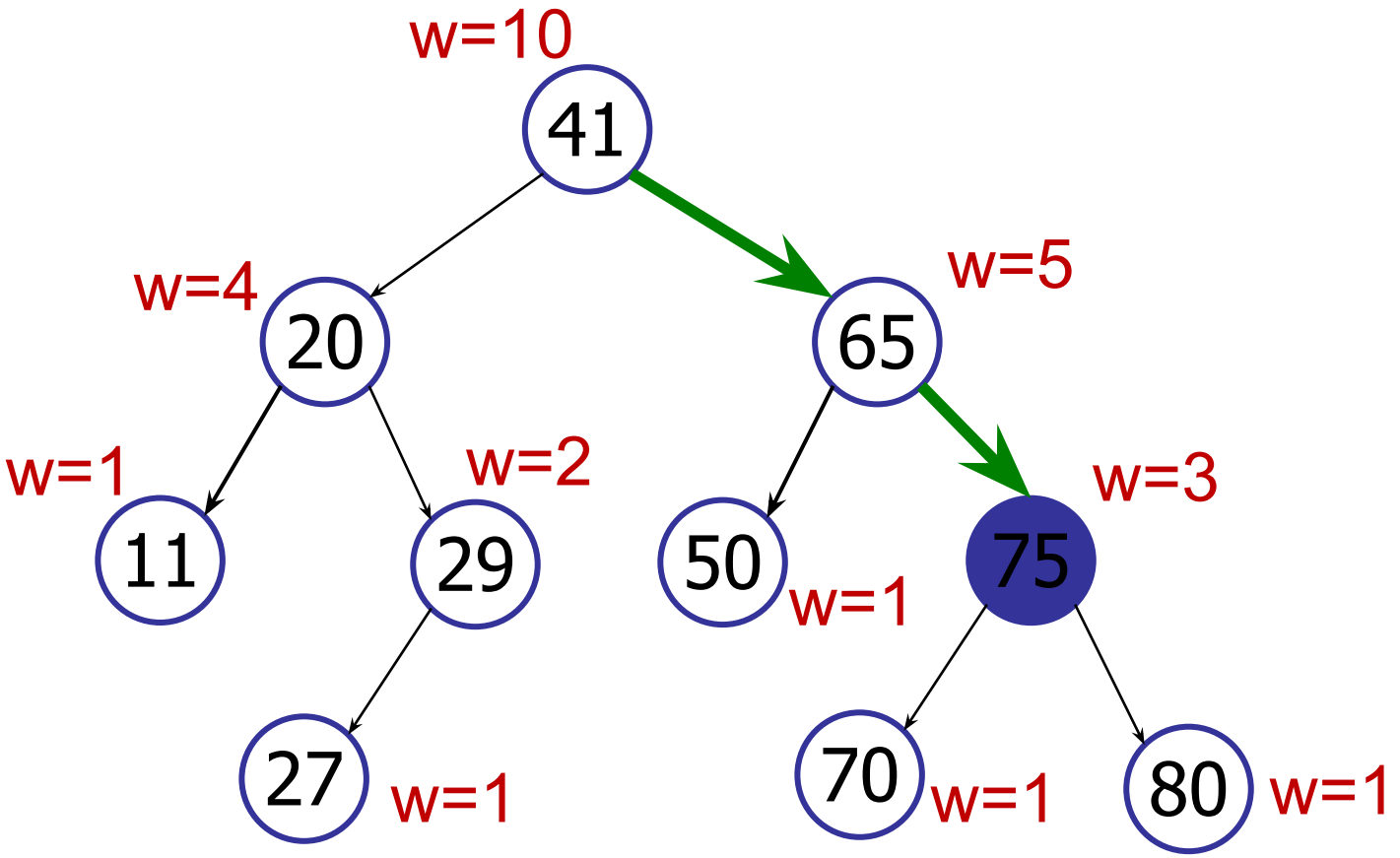
Dynamic Order Statistics

Rank(v) : computes the rank of a node v

```
rank(node)
        rank = node.left.weight + 1;
        while (node != null) do
            if node is left child then
                do nothing
            else if node is right child then
                rank += node.parent.left.weight + 1;
            node = node.parent;
        return rank;
```

Dynamic Order Statistics

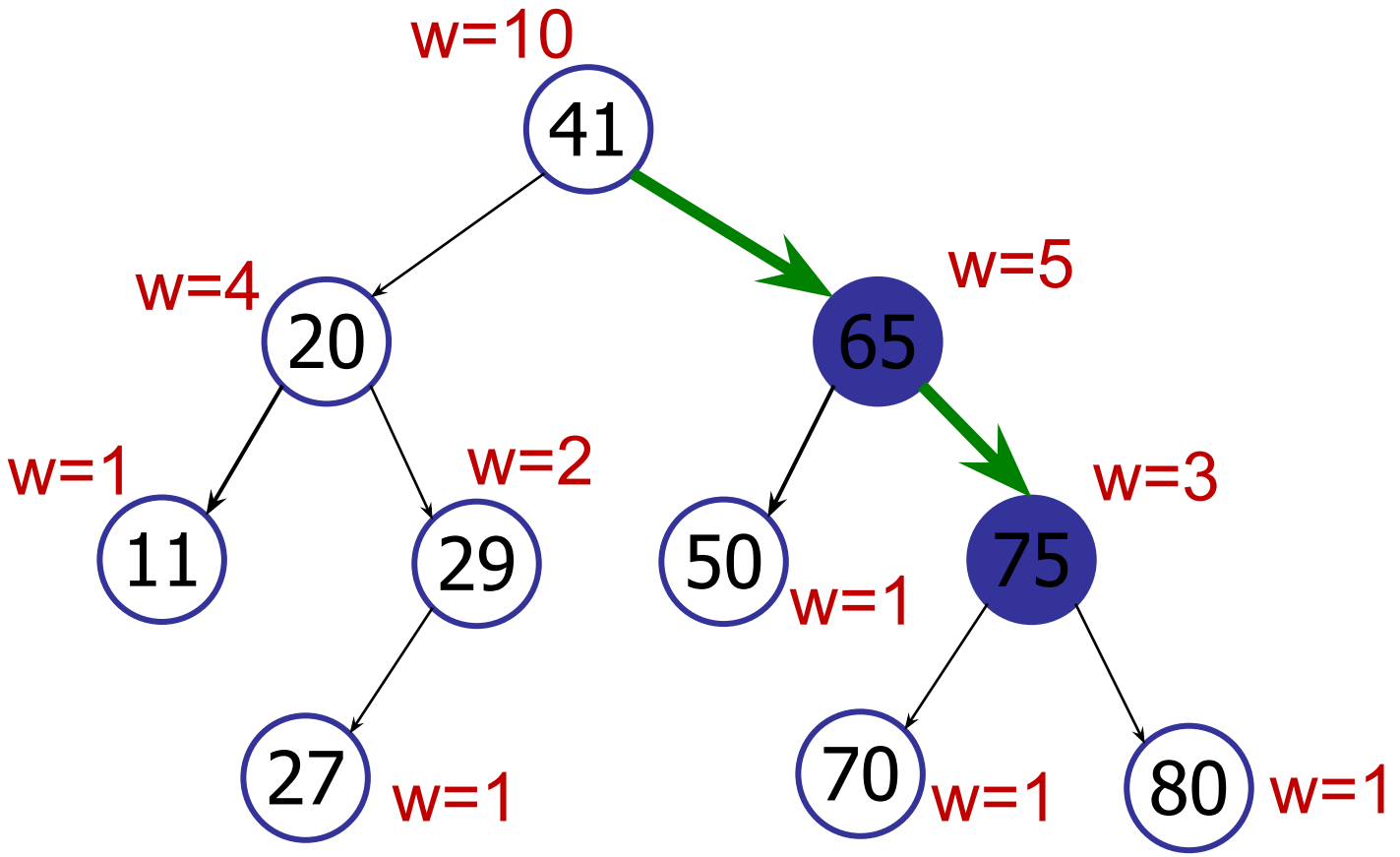
rank(75)



rank = 2

Dynamic Order Statistics

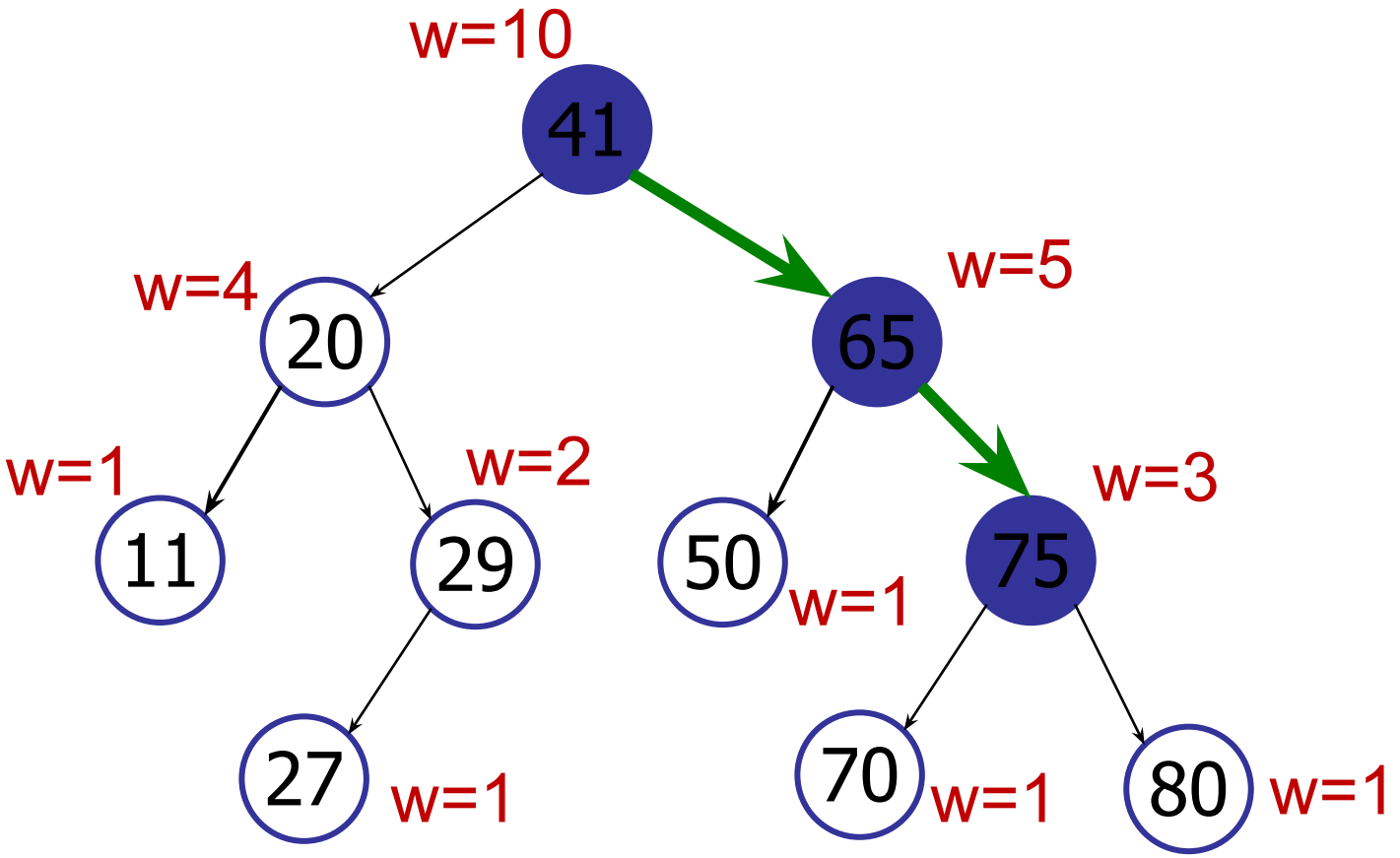
rank(75)



rank = 2 + 2

Dynamic Order Statistics

rank(75)



rank = 2 + 2 + 5 = 9

Dynamic Order Statistics

Rank(v) : computes the rank of a node v

```
rank(node)
        rank = node.left.weight + 1;
        while (node != null) do
            if node is left child then
                do nothing
            else if node is right child then
                rank += node.parent.left.weight + 1;
            node = node.parent;
        return rank;
```

Augmenting data structures

Basic methodology:

1. Choose underlying data structure:

   AVL tree

2. Determine additional info needed:

   Weight of each node

3. Maintained info as data structure is modified.

   Update weights as needed

4. Develop new operations using the new info.

   Select and Rank

Augmenting data structures

Basic methodology:

1. Choose underlying data structure:

   AVL tree

2. Determine additional info needed:

   Weight of each node

3. Maintained info as data structure is modified.

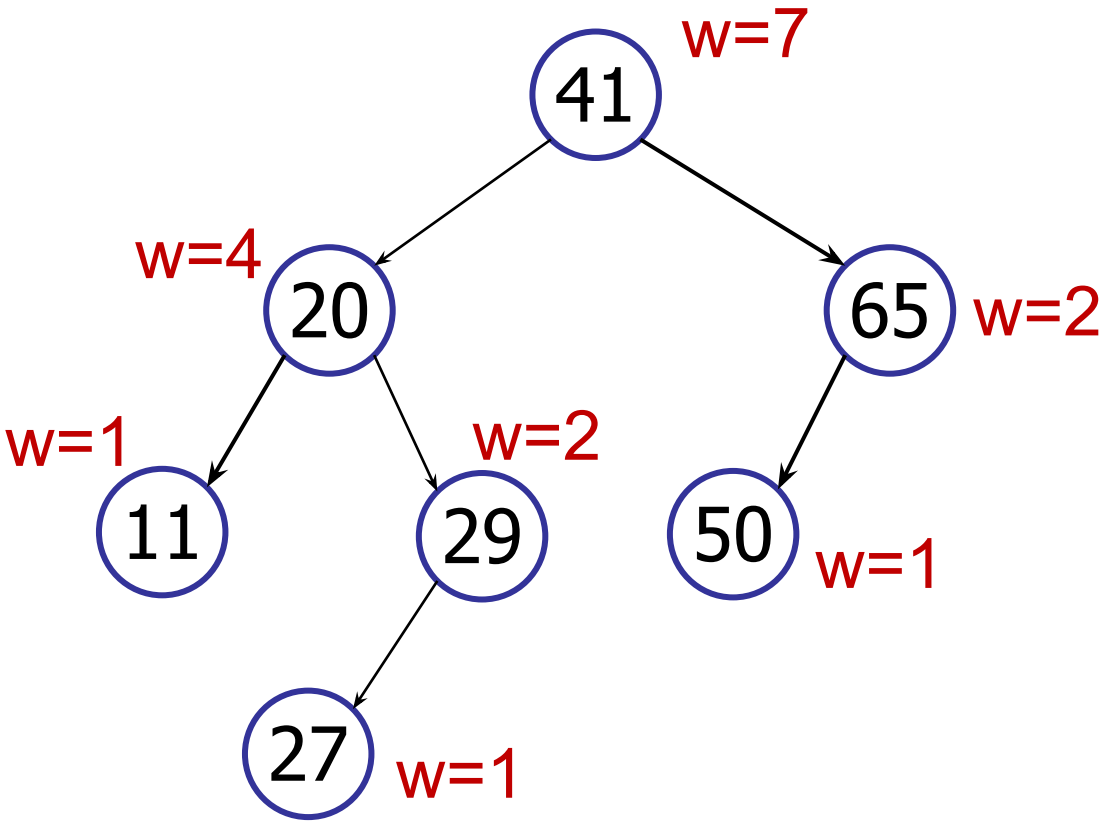
   Update weights as needed

4. Develop new operations using the new info.

   Select and Rank

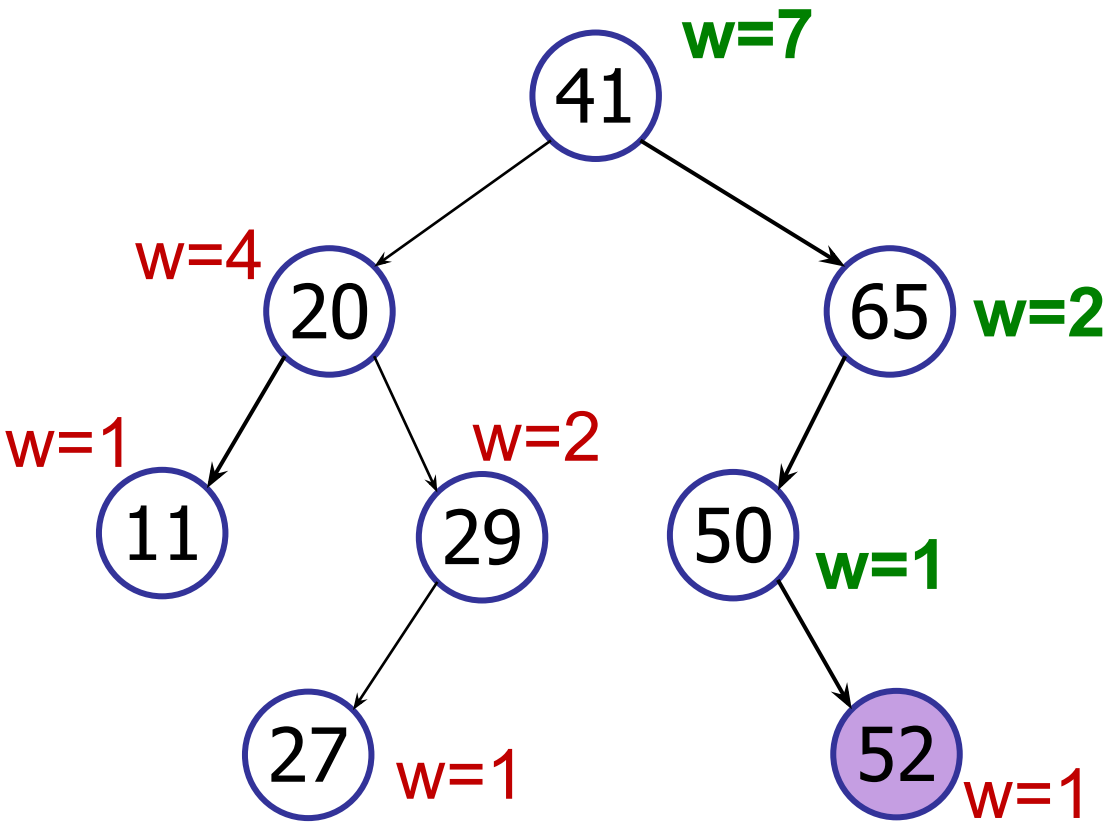
Augmented Trees

Maintain weight during insertions:



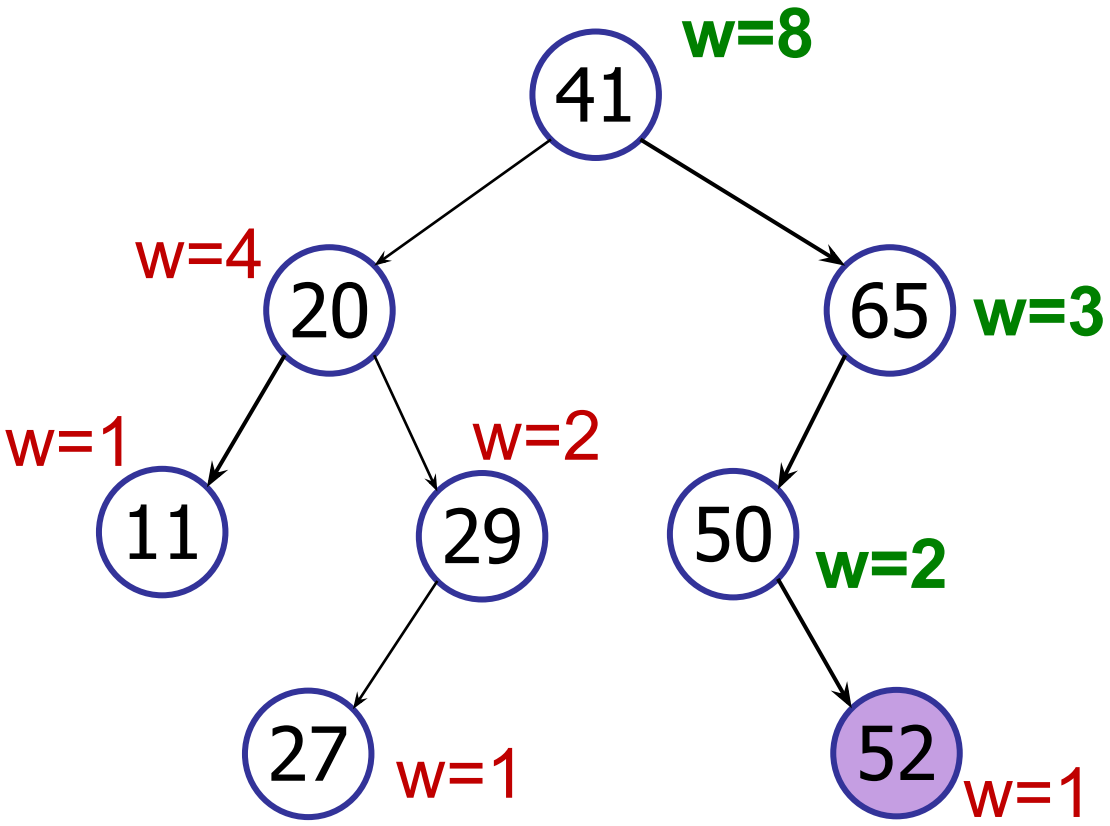
Augmented Trees

Maintain weight during insertions:



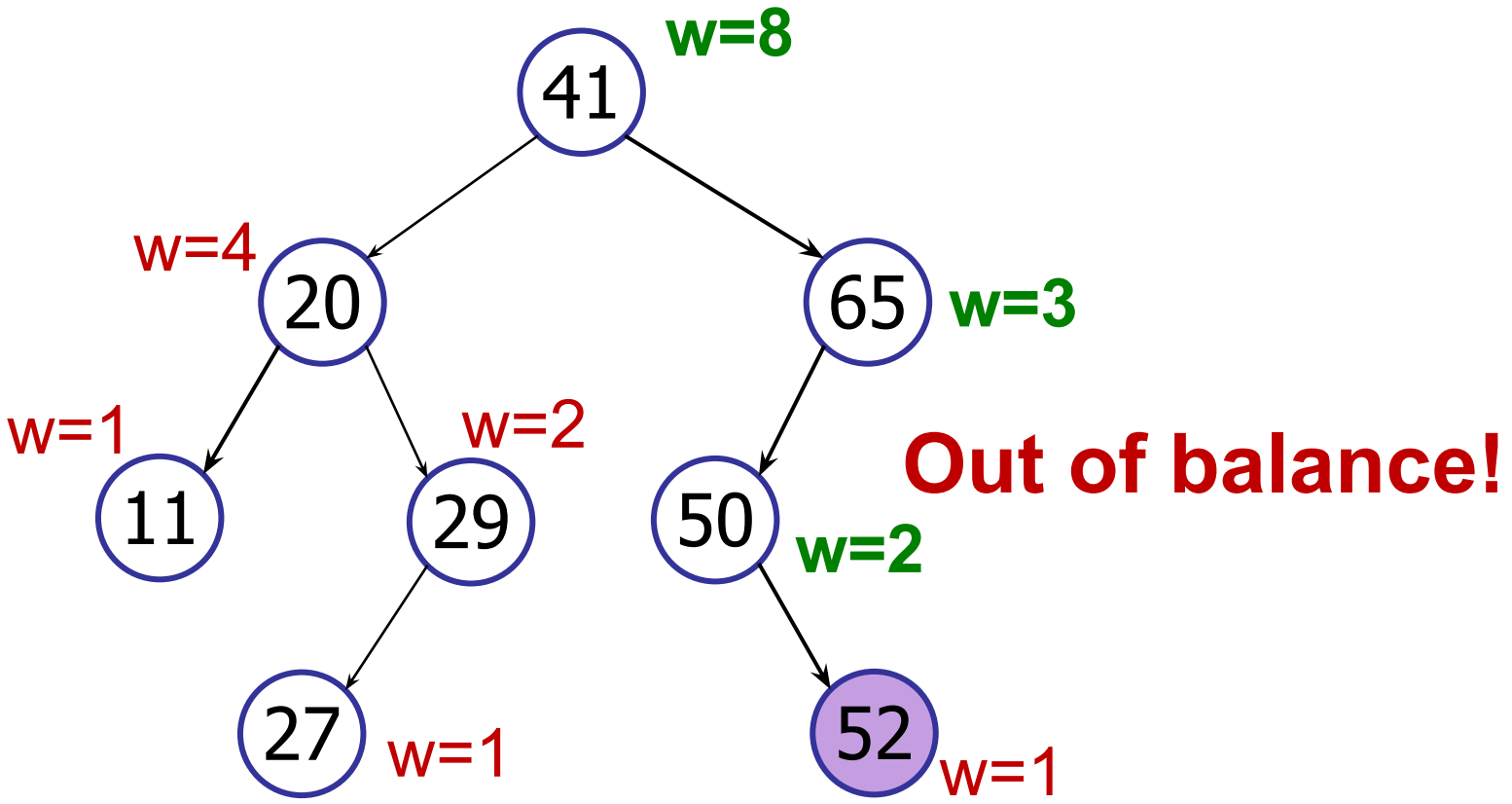
Augmented Trees

Maintain weight during insertions:



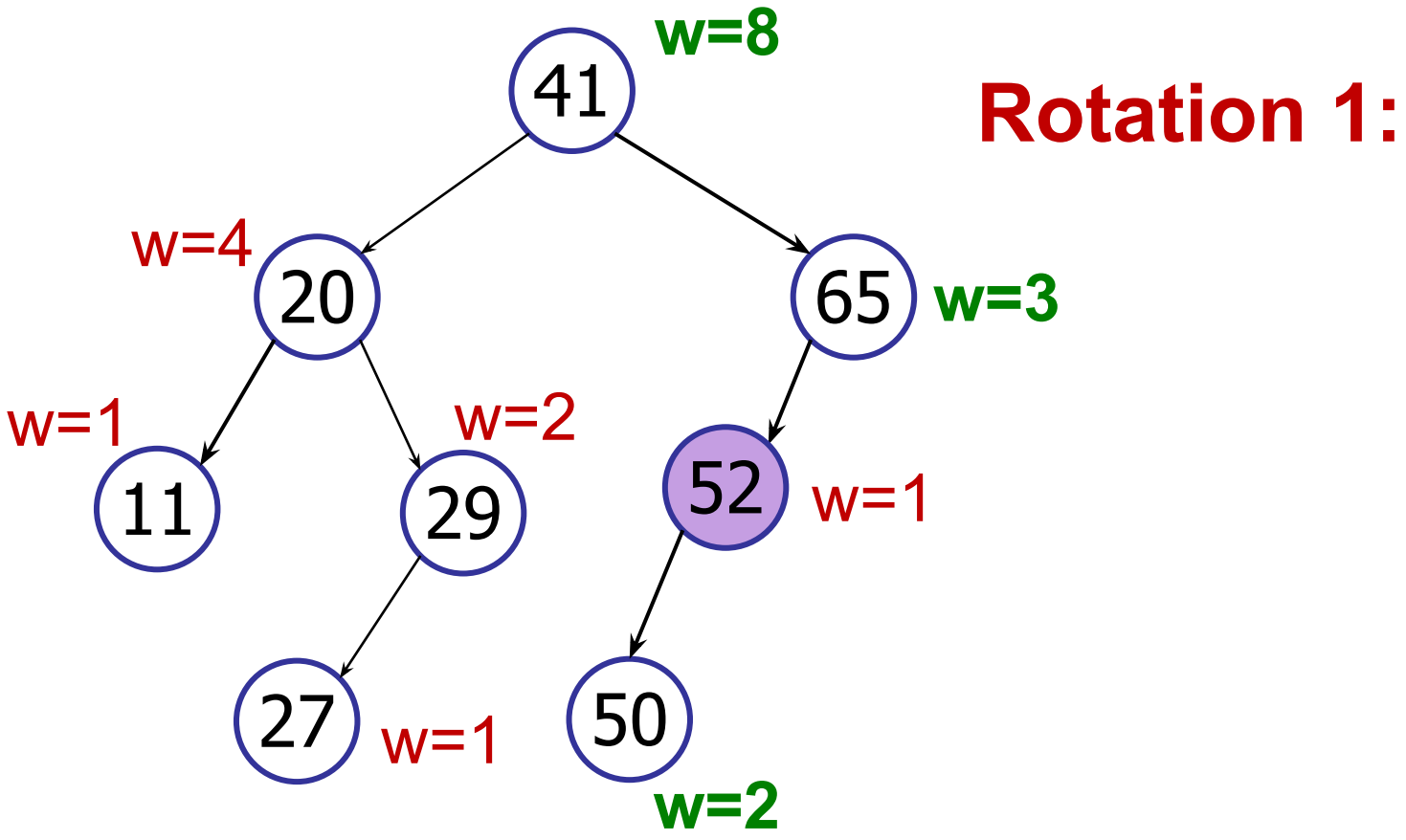
Augmented Trees

Maintain weight during insertions:



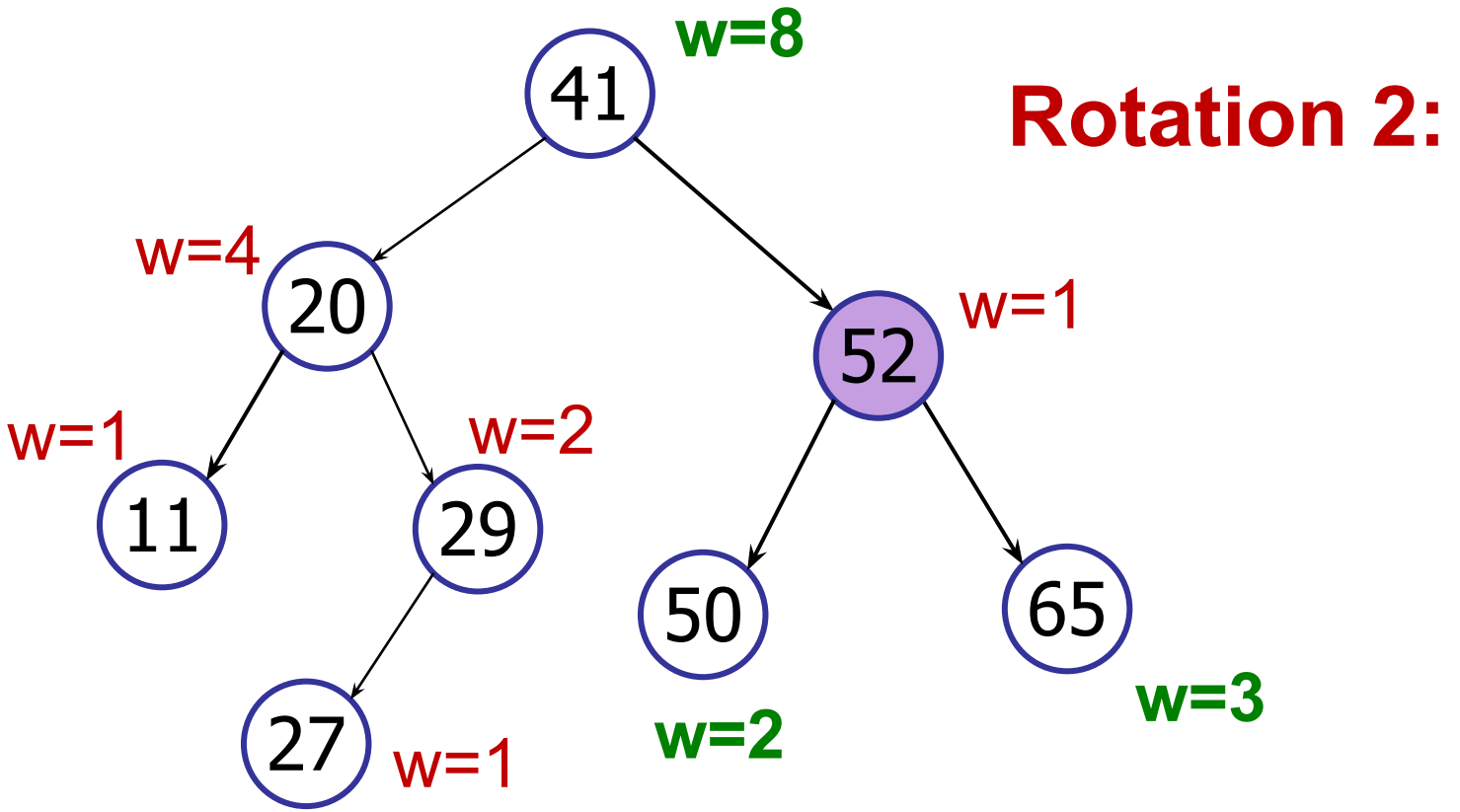
Augmented Trees

Maintain weight during insertions:



Augmented Trees

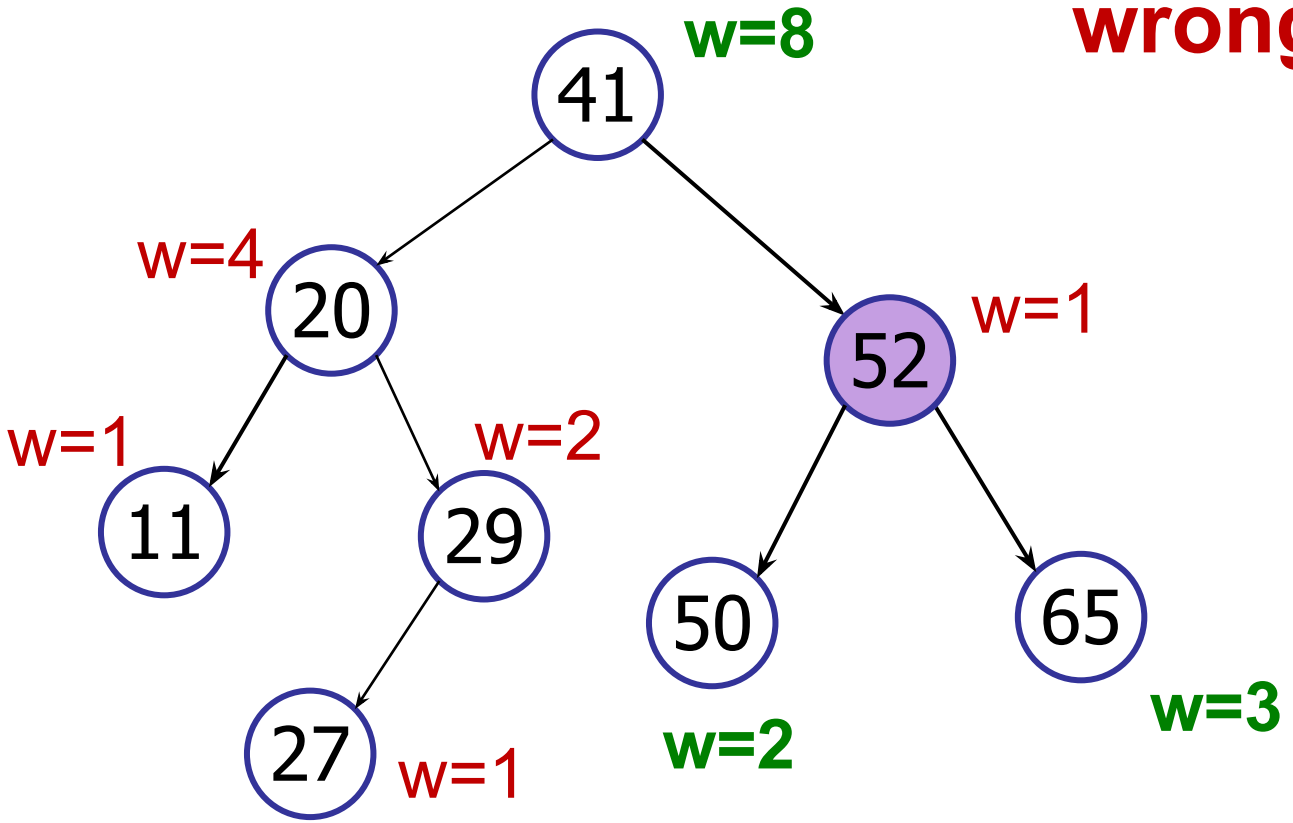
Maintain weight during insertions:



Augmented Trees

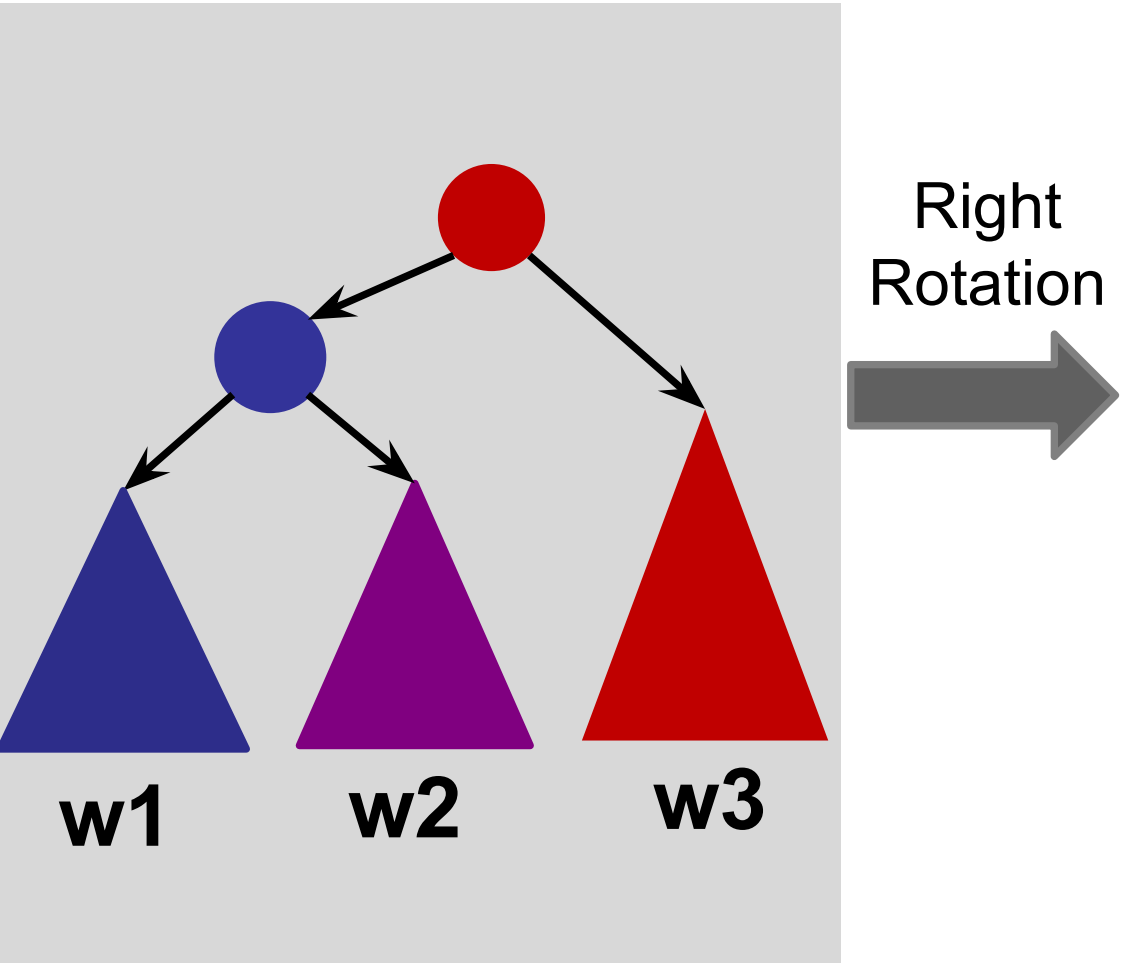
How to update weights on rotation?

Weights all wrong!



Augmented Trees

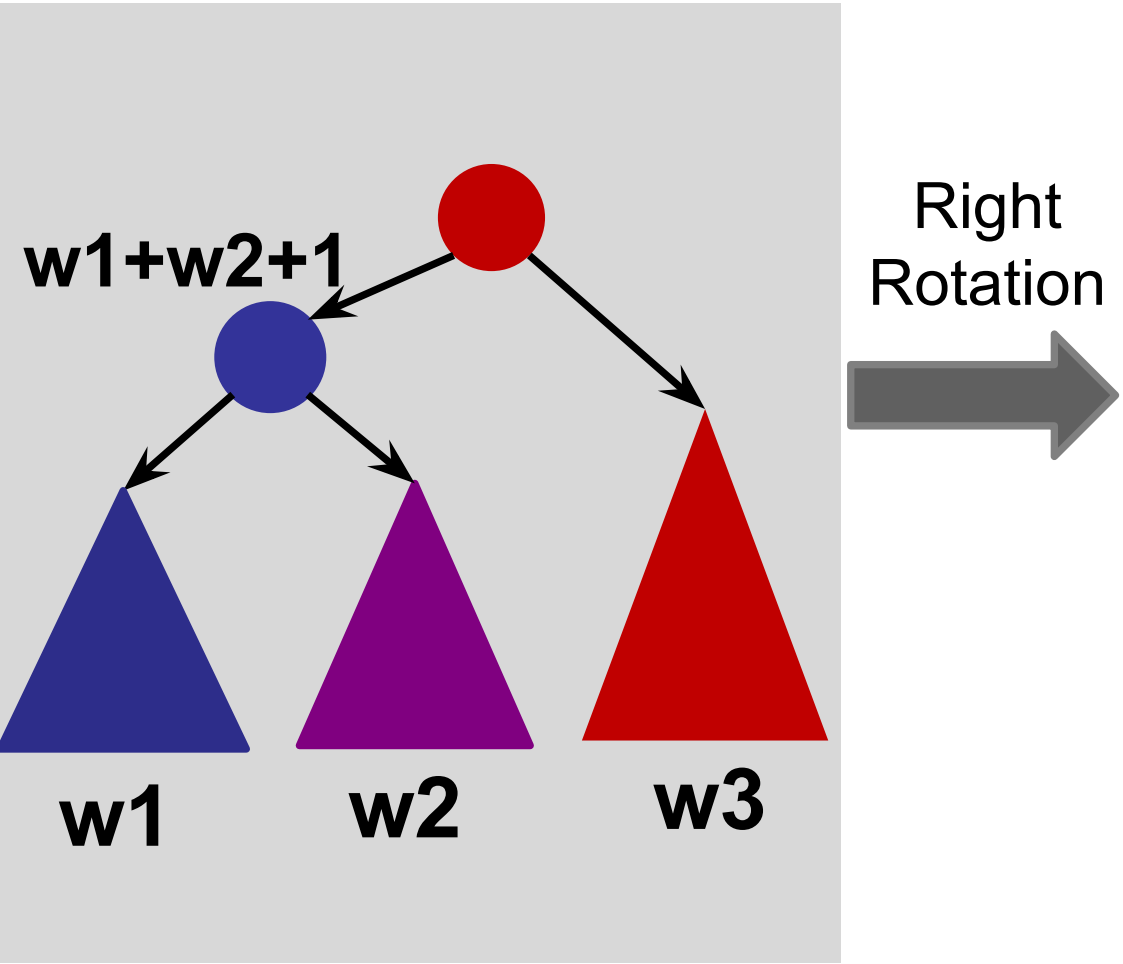
Maintain weight during rotations:



Right
Rotation

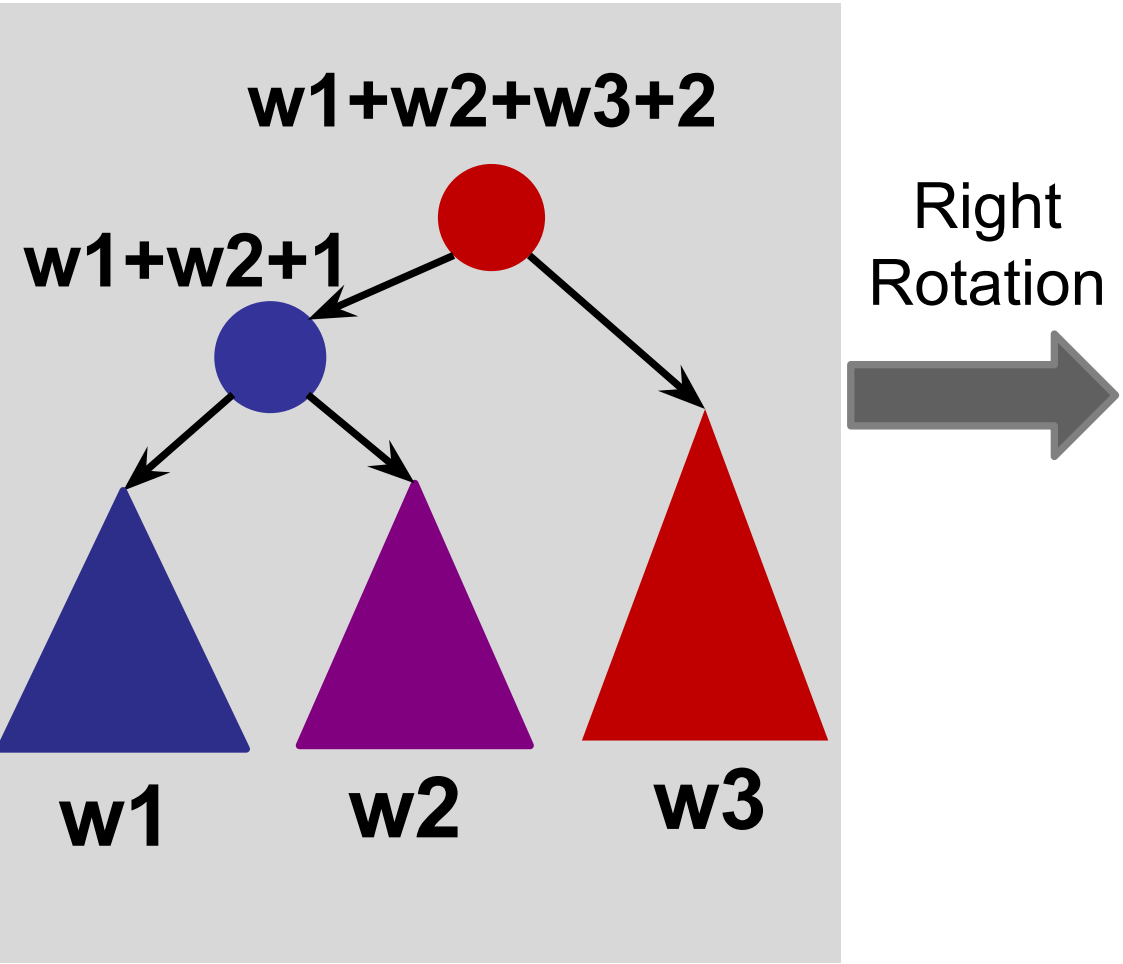
Augmented Trees

Maintain weight during rotations:



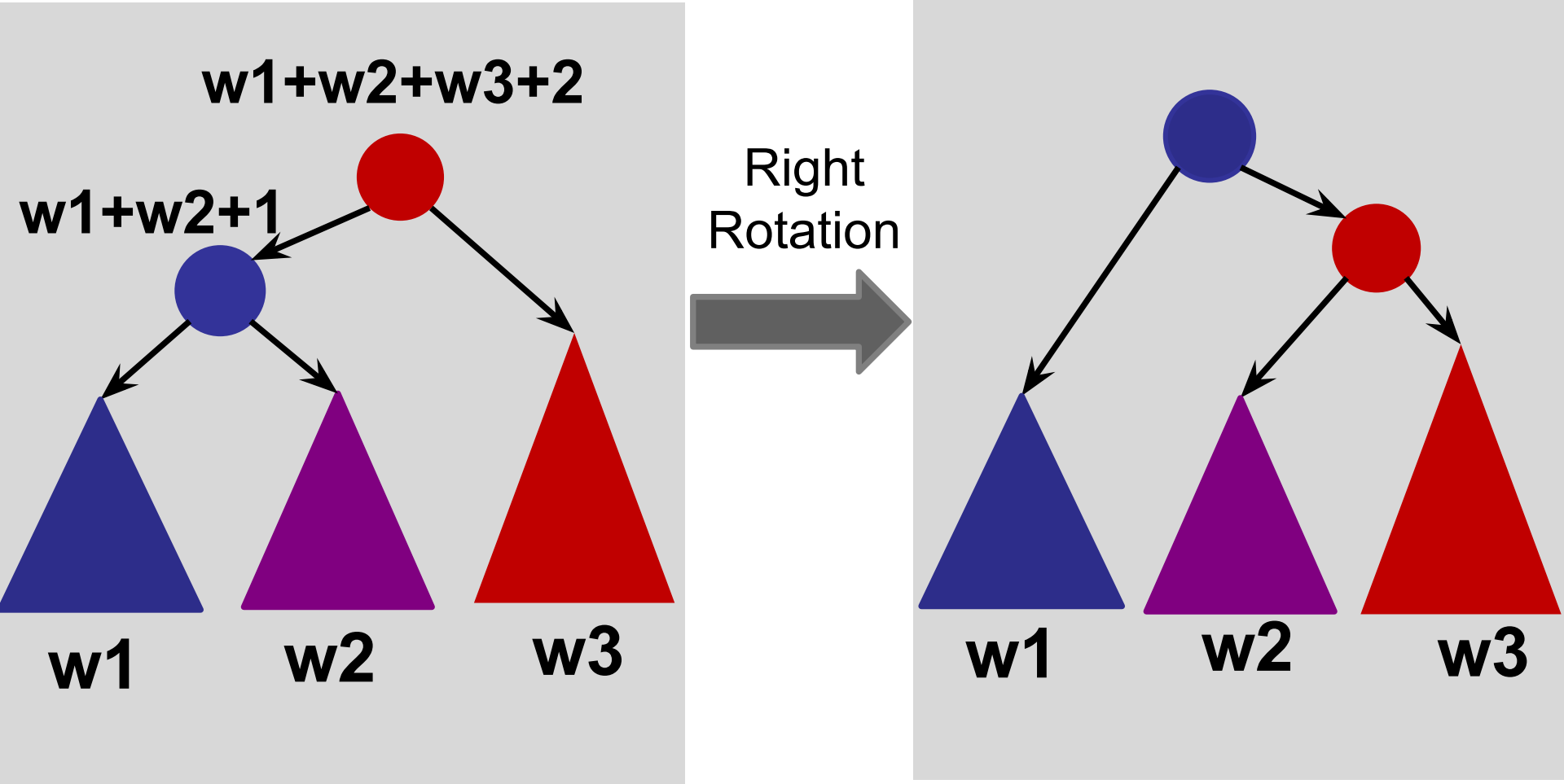
Augmented Trees

Maintain weight during rotations:



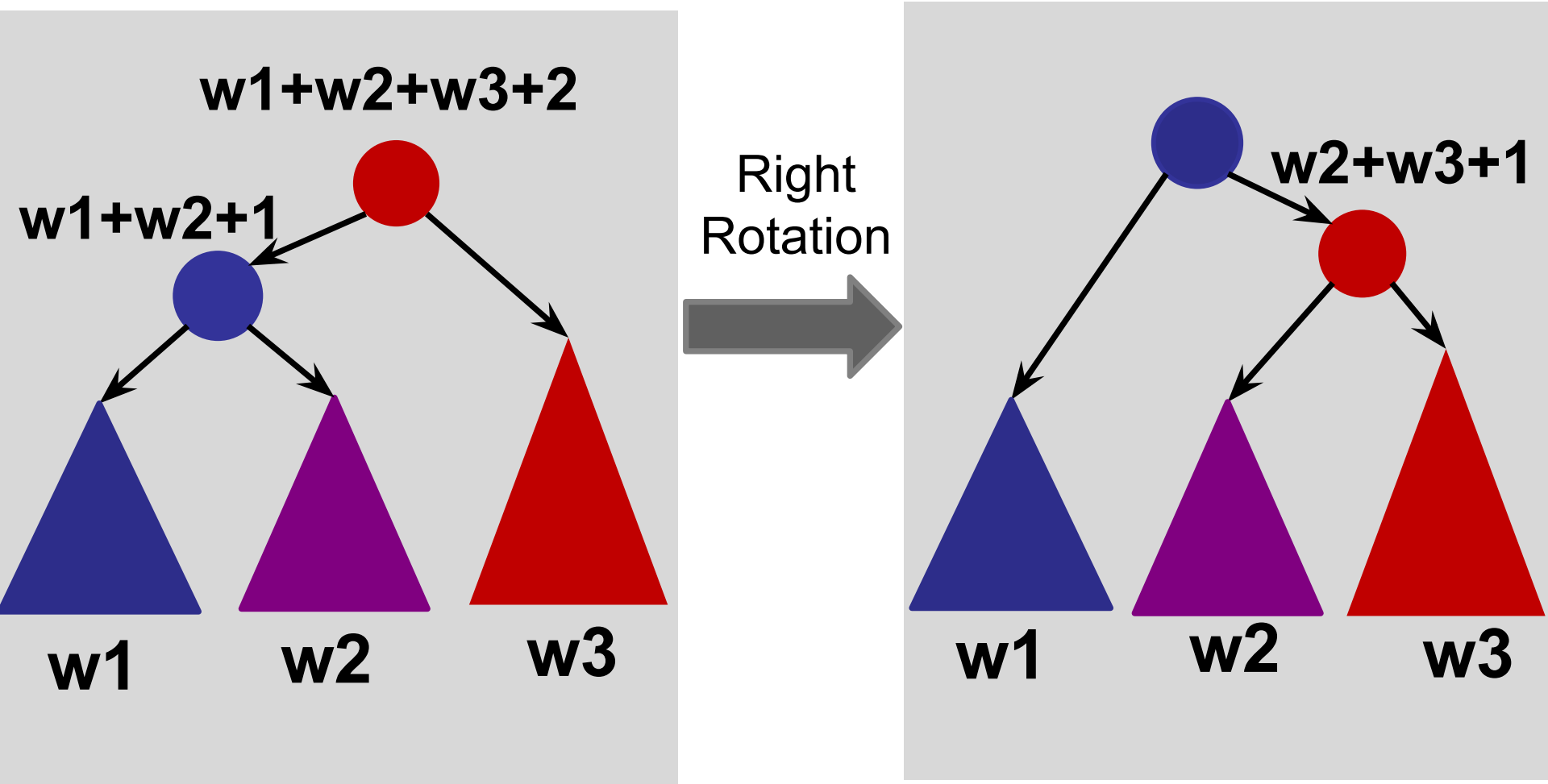
Augmented Trees

Maintain weight during rotations:



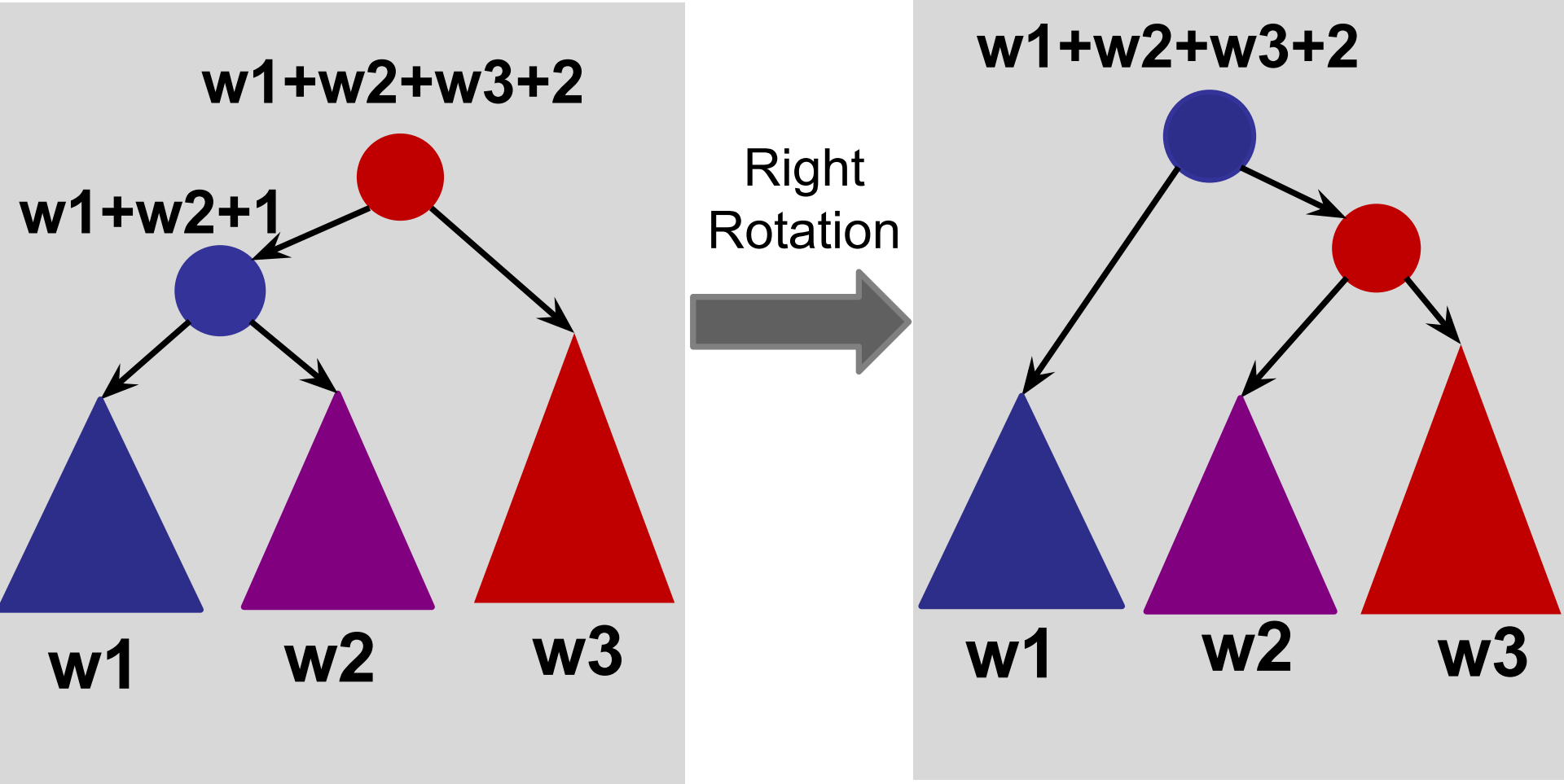
Augmented Trees

Maintain weight during rotations:



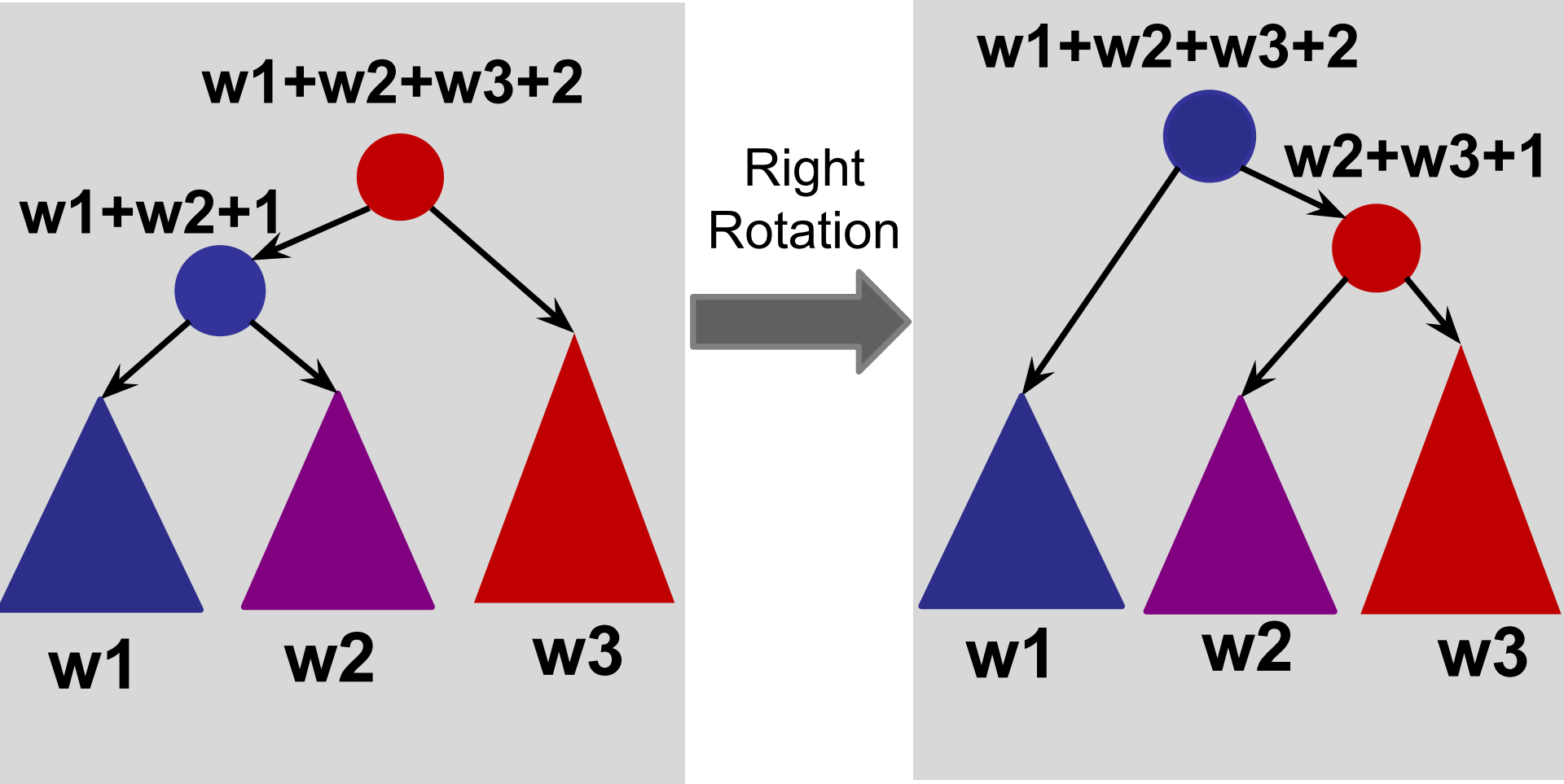
Augmented Trees

Maintain weight during rotations:



Augmented Trees

Maintain weight during rotations:



How long does it take to update the weights during a rotation?

1. O(1)
2. O(log n)
3. O(n)
4. $O(n^2)$
5. What is a rotation?

Augmenting data structures

Basic methodology:

1. Choose underlying data structure

   (tree, hash table, linked list, stack, etc.)

2. Determine additional info needed.

3. Verify that the additional info can be maintained as the data structure is modified.

   (subject to insert/delete/etc.)

4. Develop new operations using the new info.

Other Augmentations

What about duplicates? How can we augment the trees to handle that?

Maybe add a "count" to multiple insertions of the same value. How does that affect the other operations?

Other Augmentations

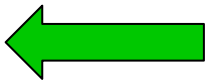
Finding max/min takes O(log n) time.

Can we augment this to take O(1) time?

Todays Plan

Data structure design

- More Augmentation on Balanced Trees

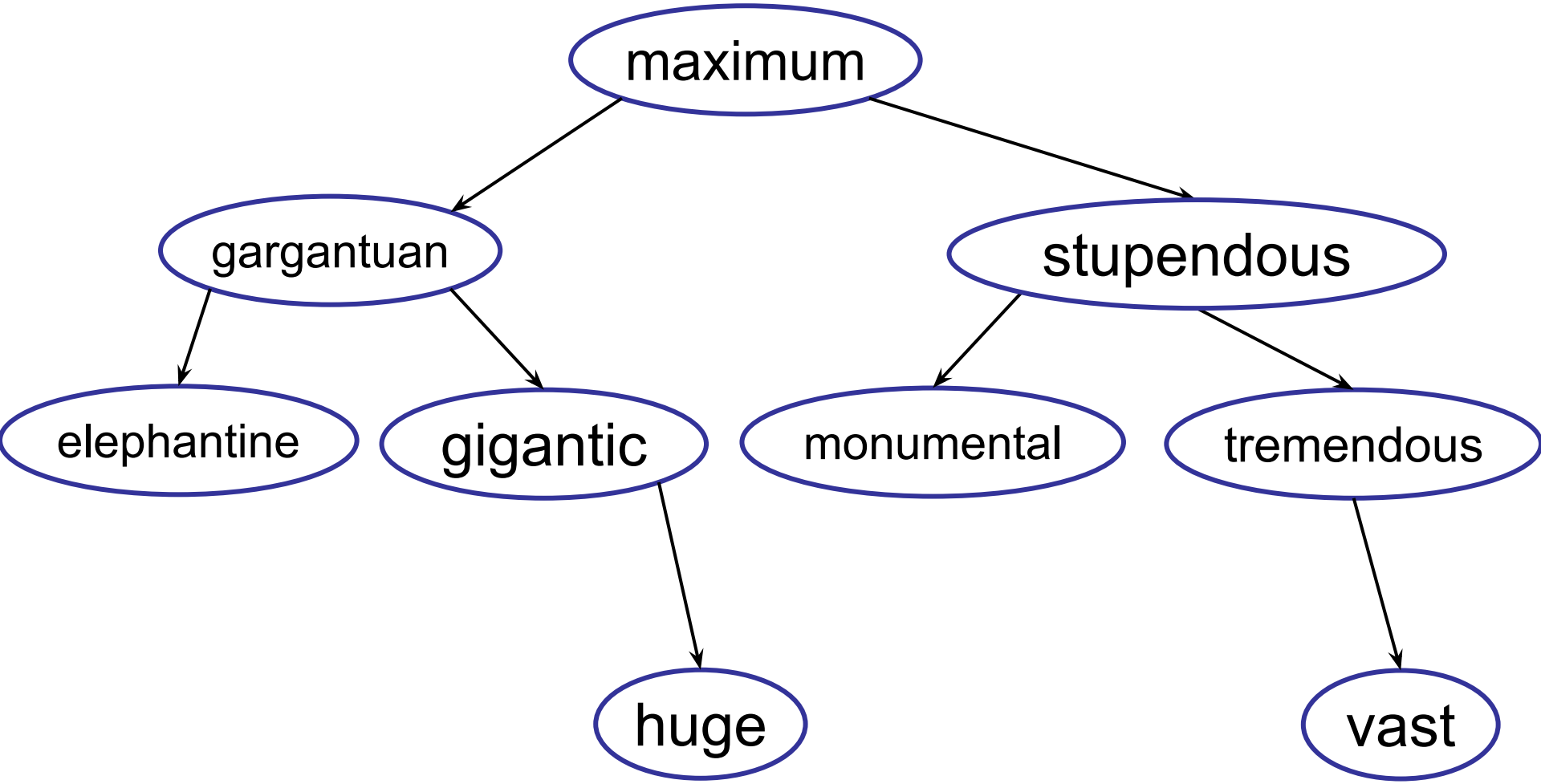
Tries 

- How to handle text?

Problem Solving Using Trees

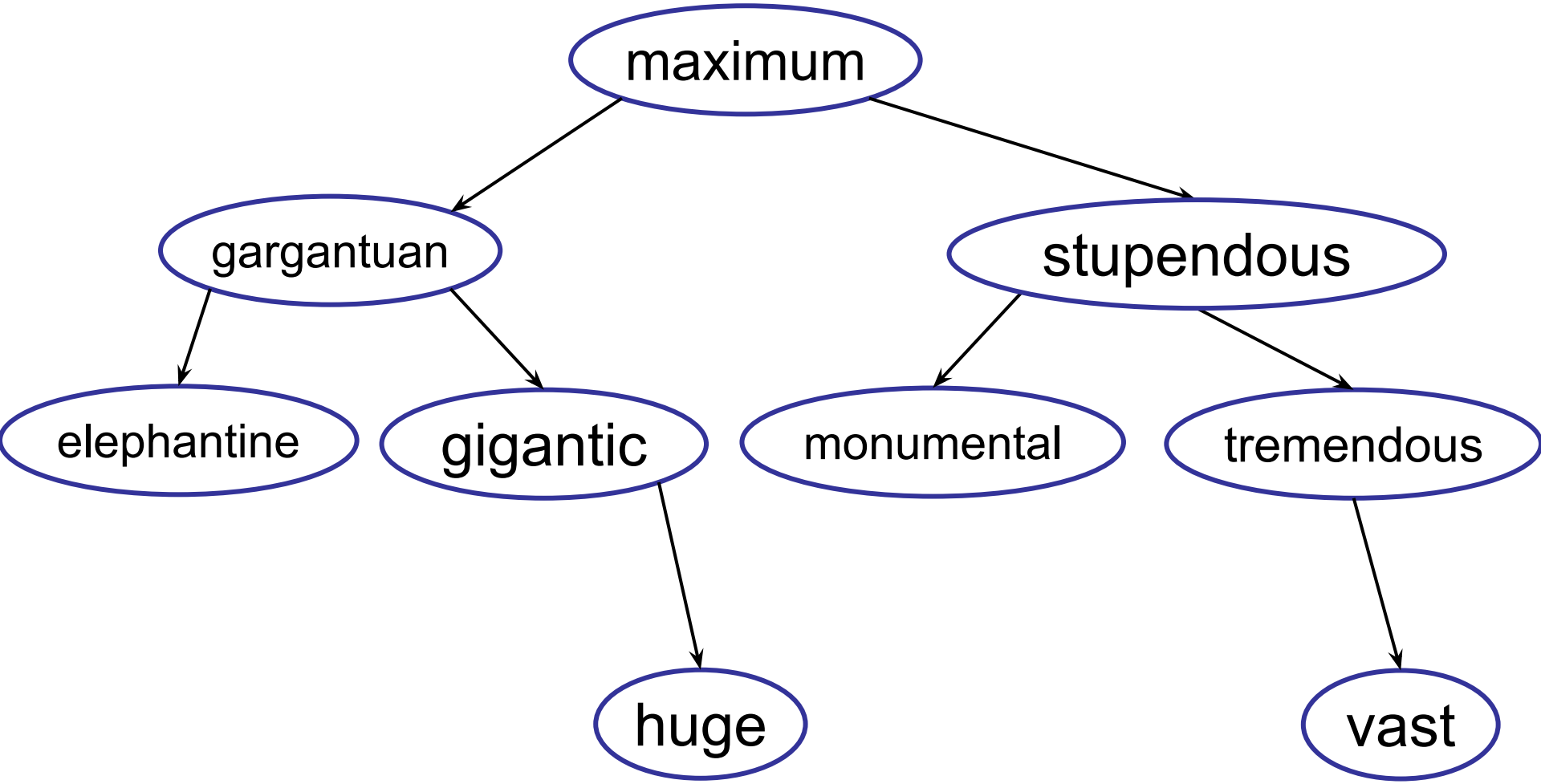
- Thinking with Trees

What about text strings?



Implement a searchable dictionary!

What about text strings?



What about storing them in a balanced BST?

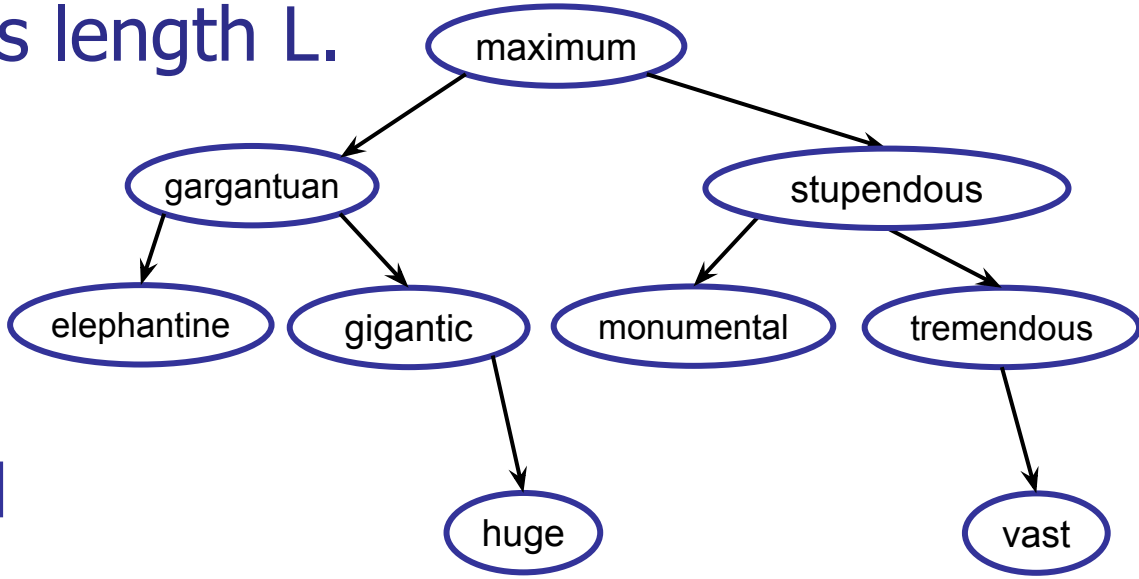
What about text strings?

Cost of comparing two strings:

- Cost[A ?= B] = min(A.length, B.length)
- Compare strings letter by letter

Cost of tree operation:

- Assume string has length L.
- Cost: O(hL)

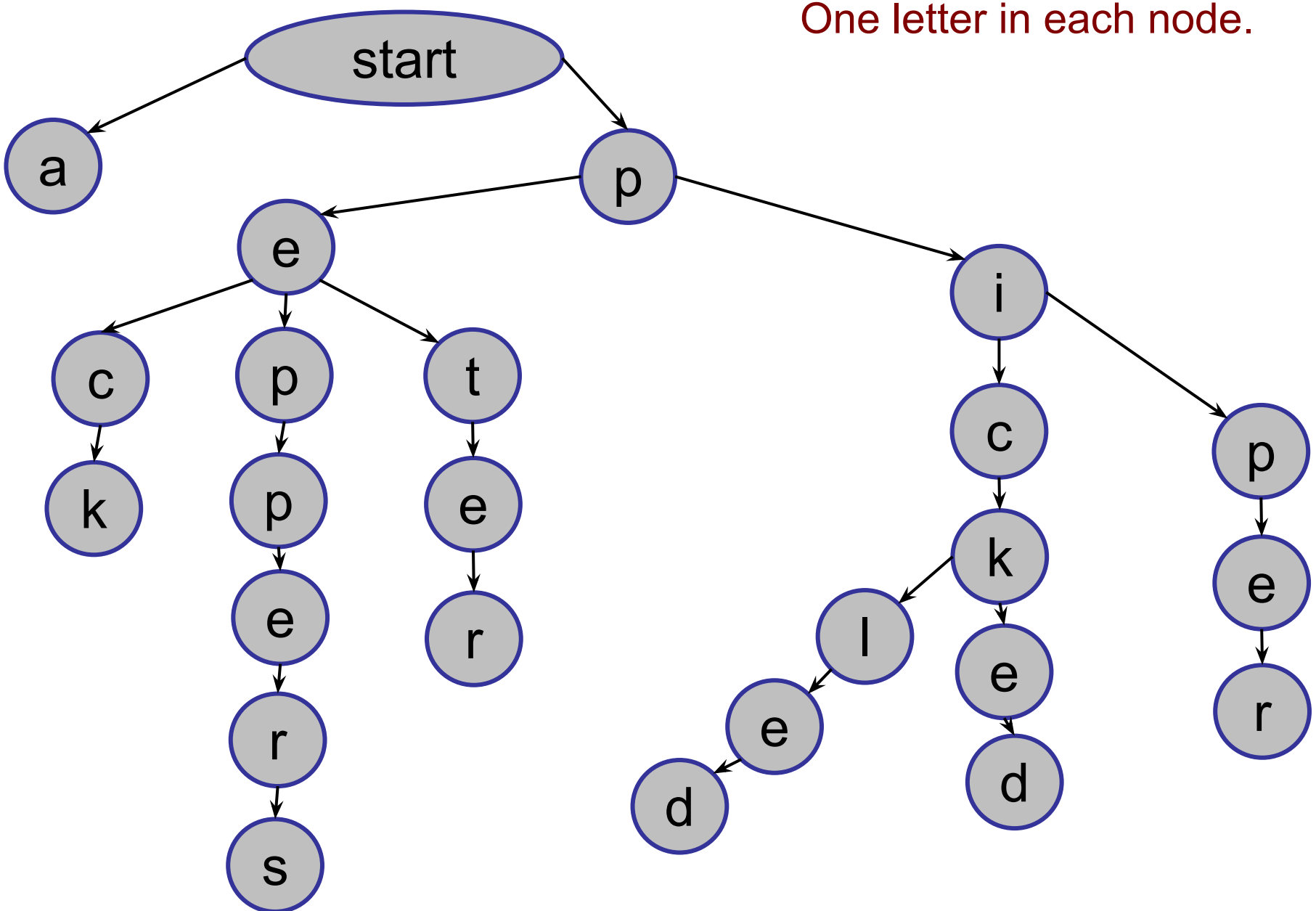


[In the worst case.]

[Optimizations are possible.]

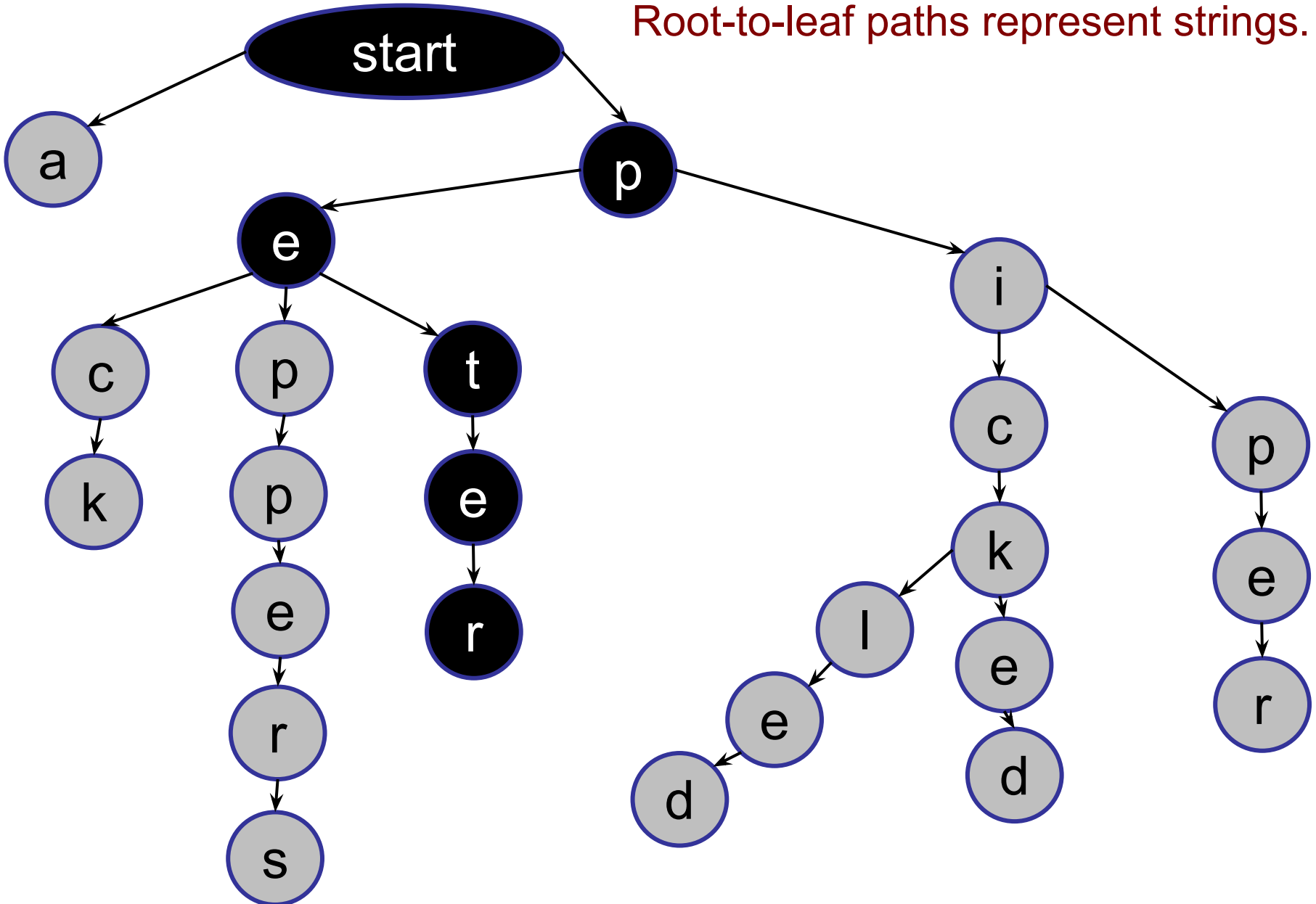
Trie [prounounced: try]

One letter in each node.



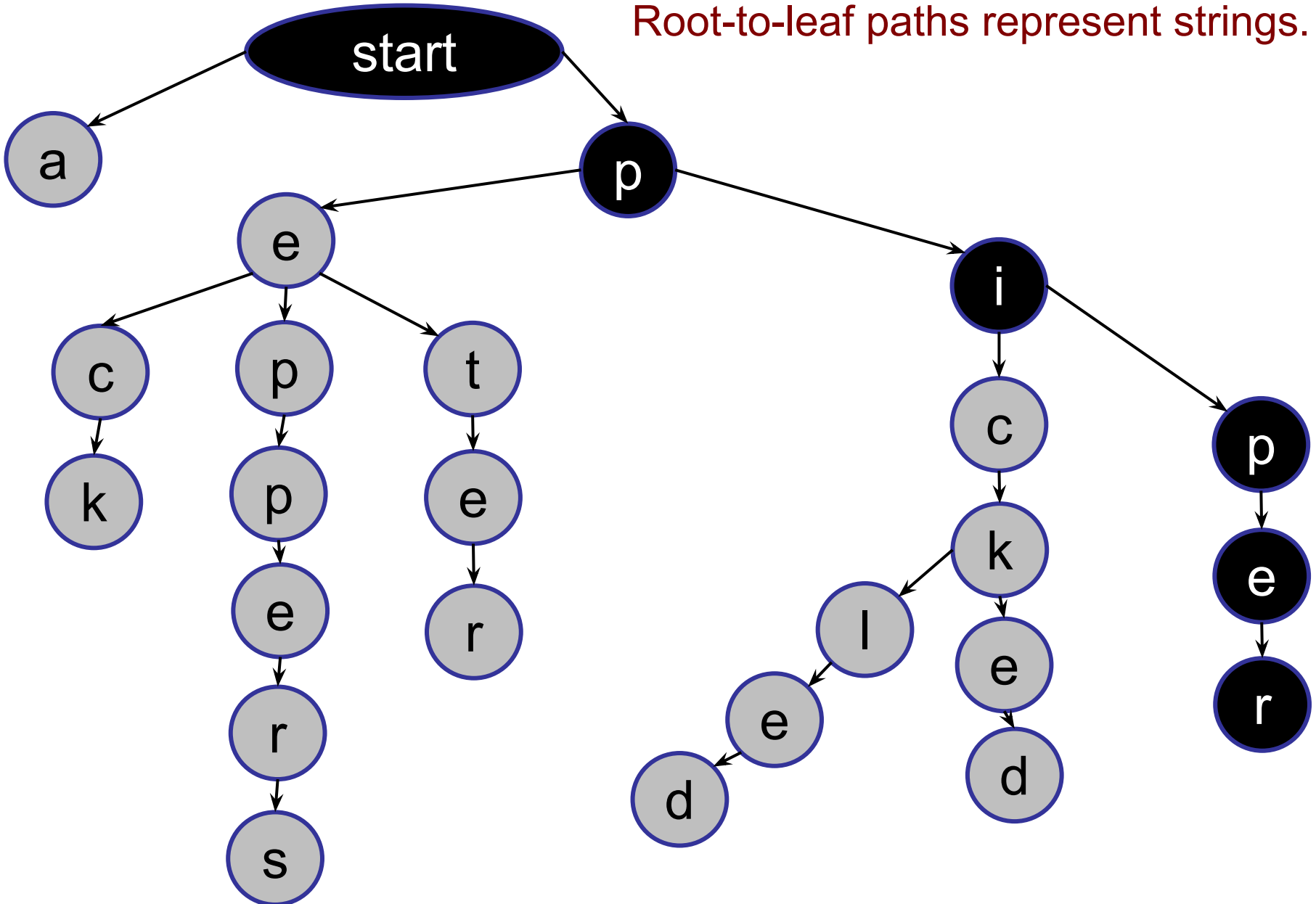
Trie [prounounced: try]

Root-to-leaf paths represent strings.



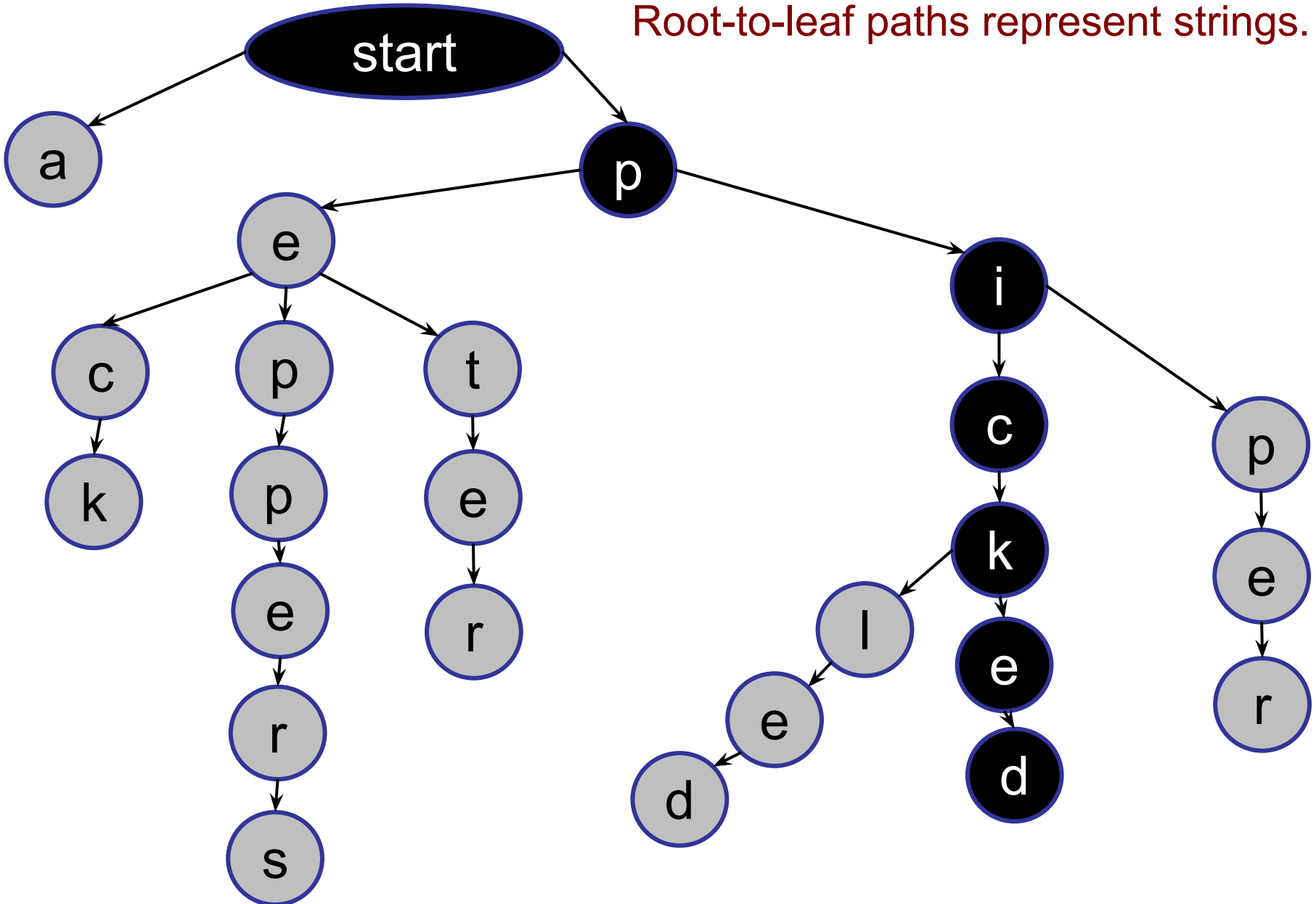
Trie [prounounced: try]

Root-to-leaf paths represent strings.



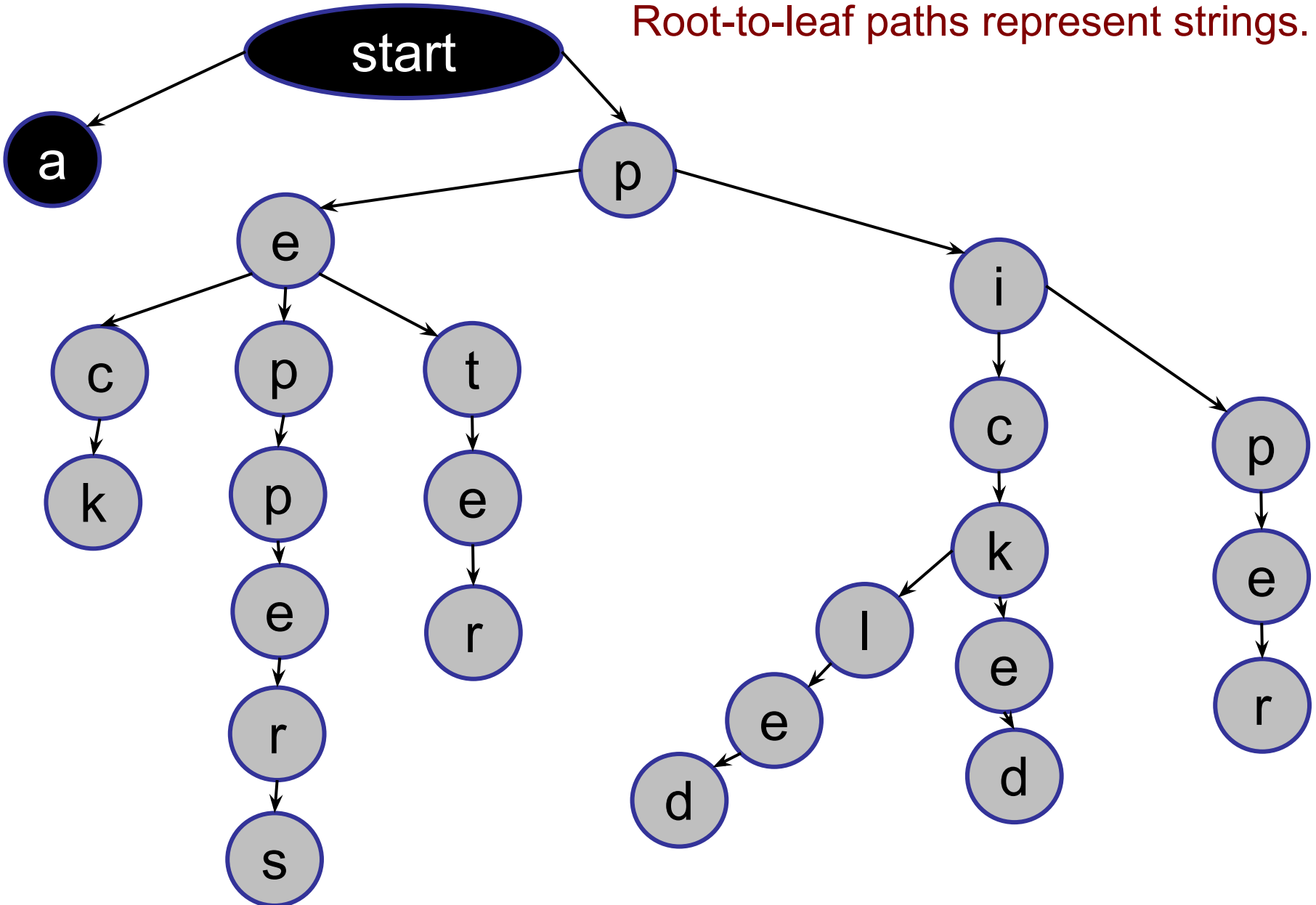
Trie [prounounced: try]

Root-to-leaf paths represent strings.



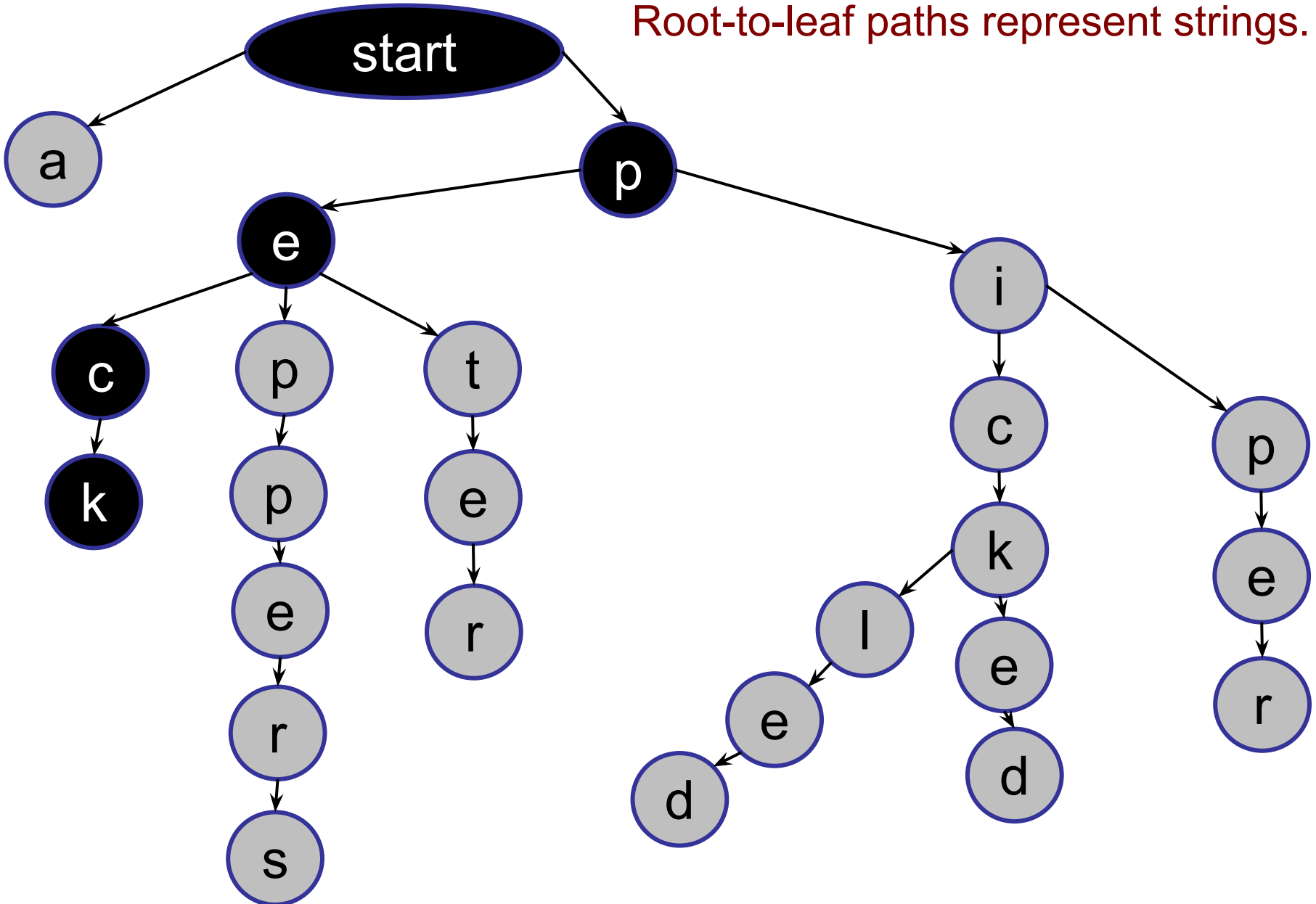
Trie [prounounced: try]

Root-to-leaf paths represent strings.



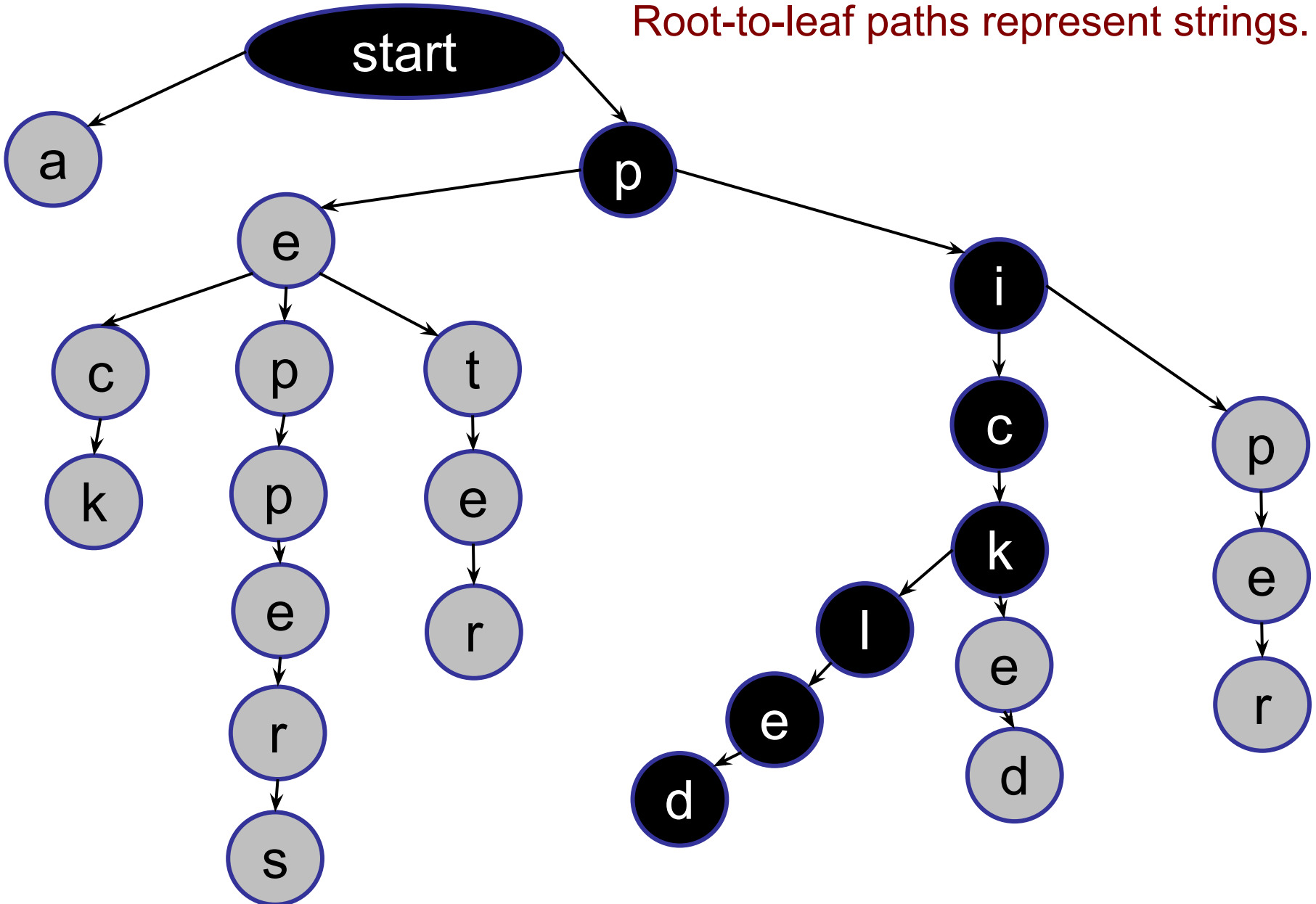
Trie [prounounced: try]

Root-to-leaf paths represent strings.



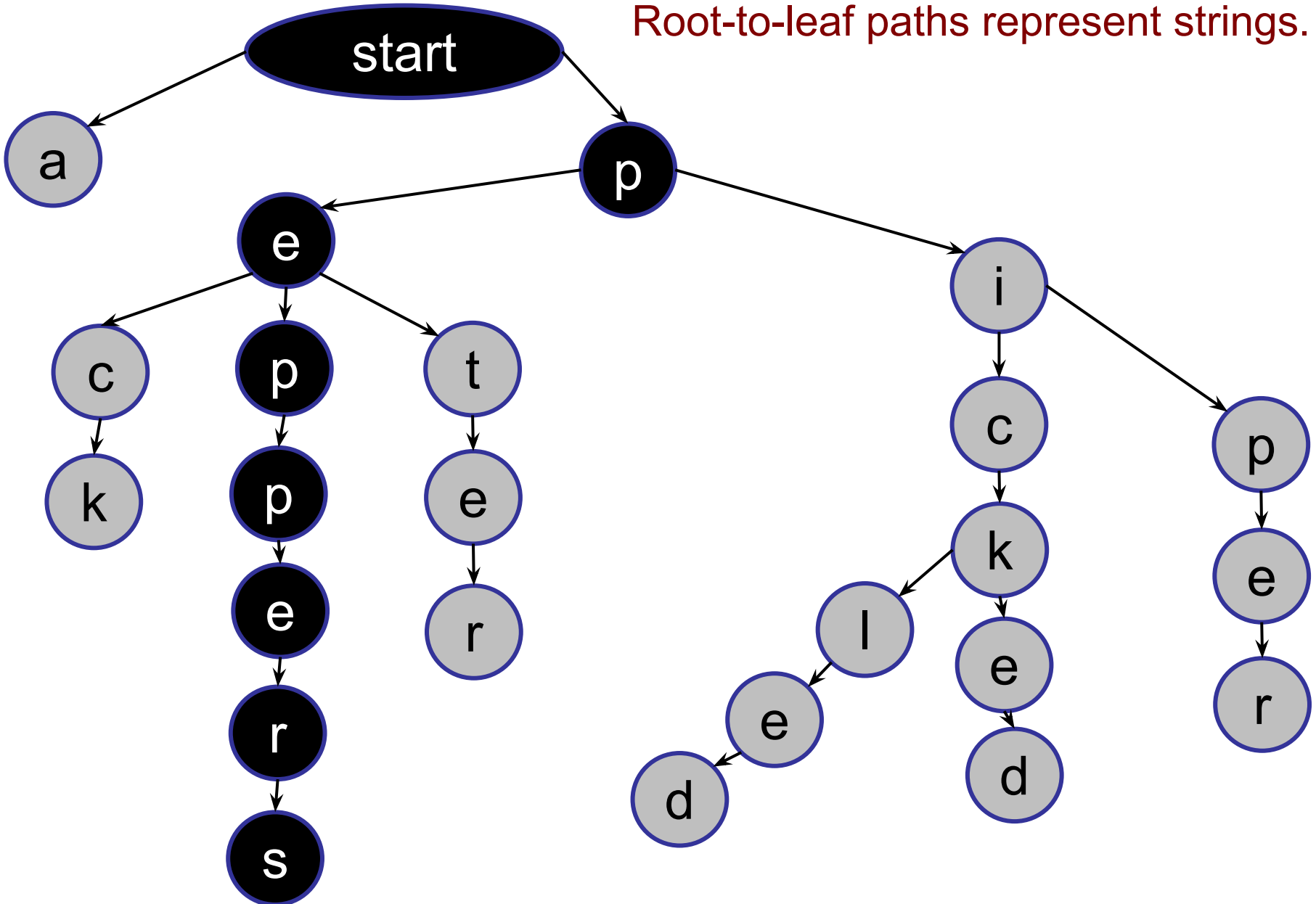
Trie [prounounced: try]

Root-to-leaf paths represent strings.

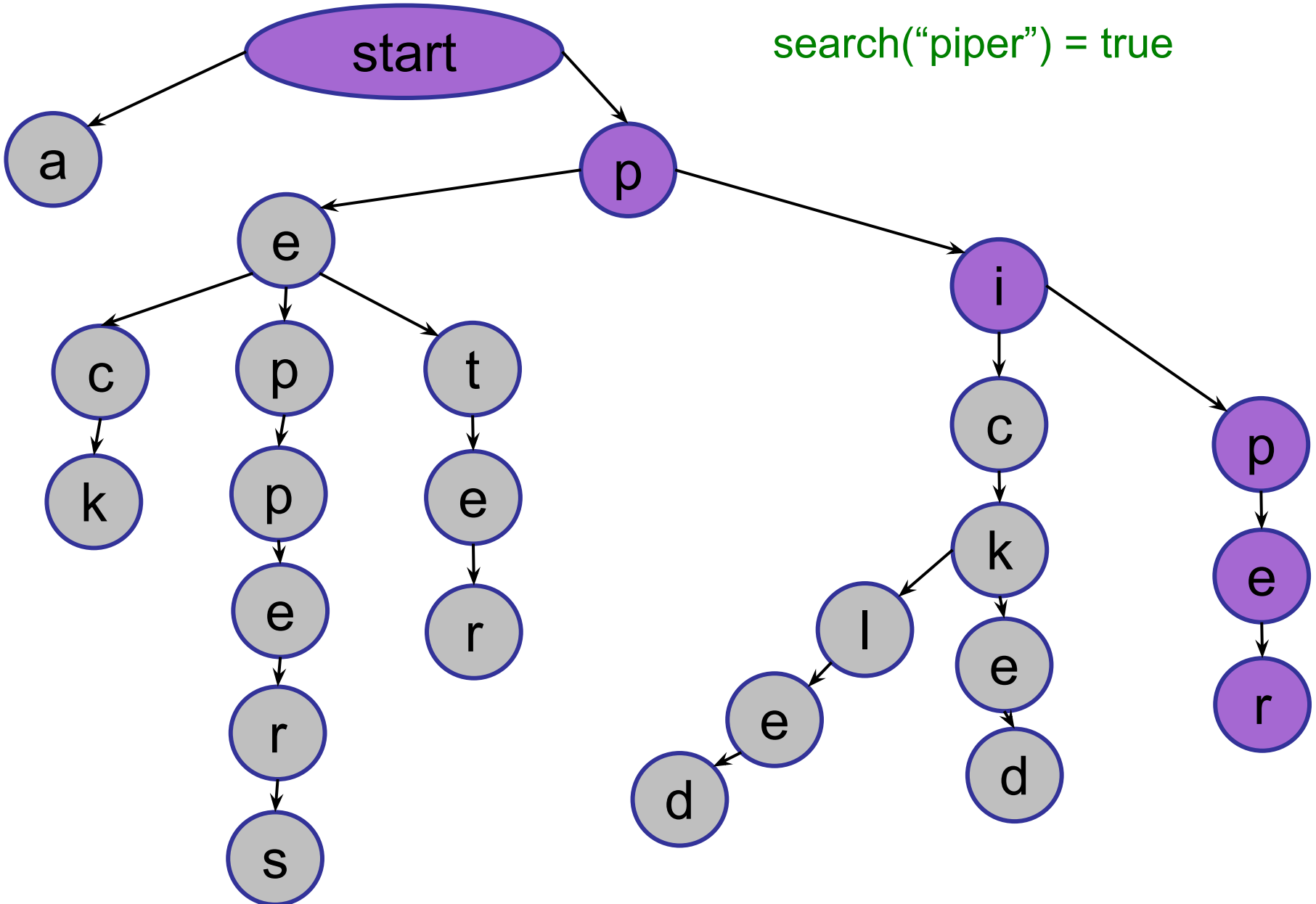


Trie [prounounced: try]

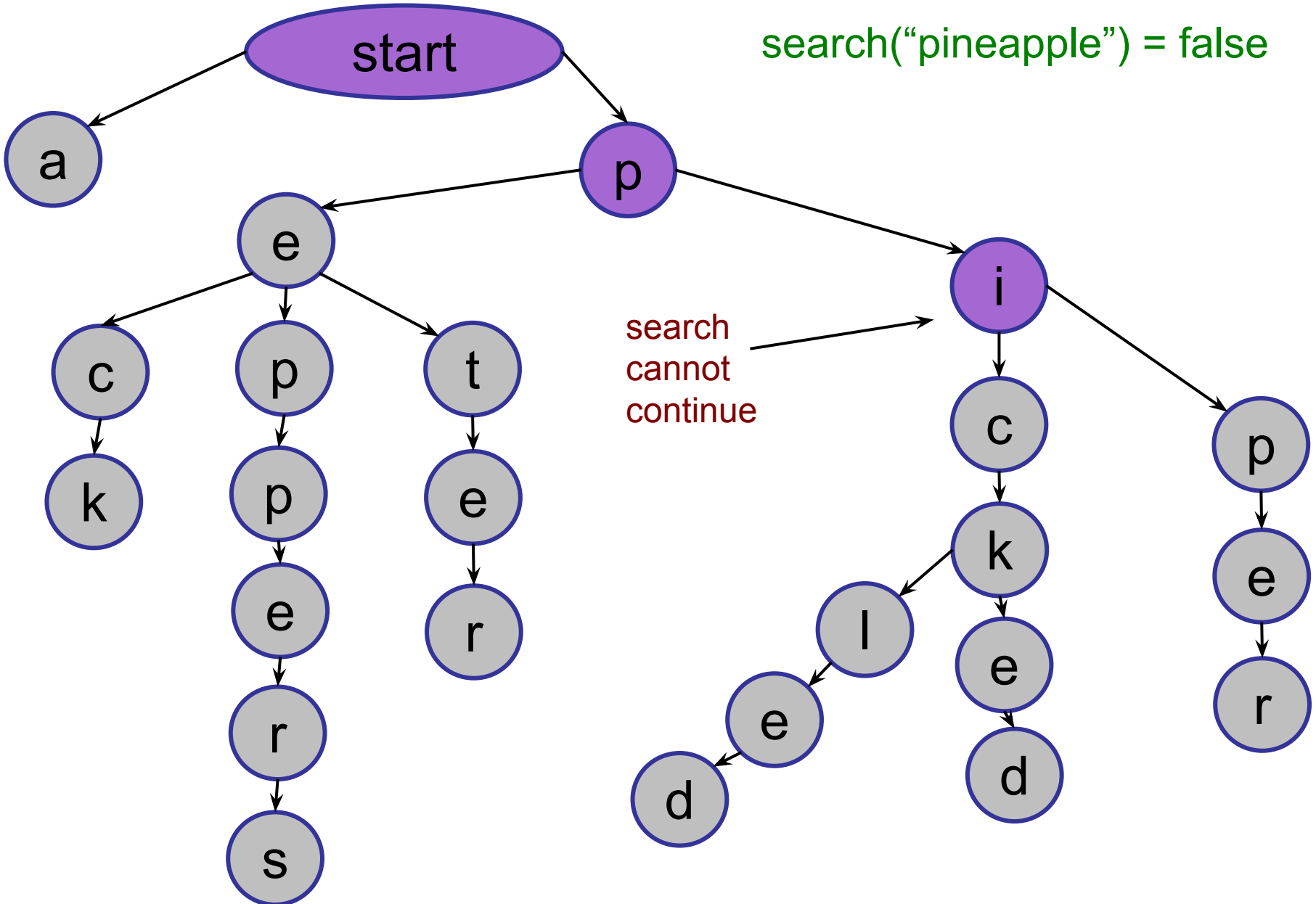
Root-to-leaf paths represent strings.



Searching a Trie

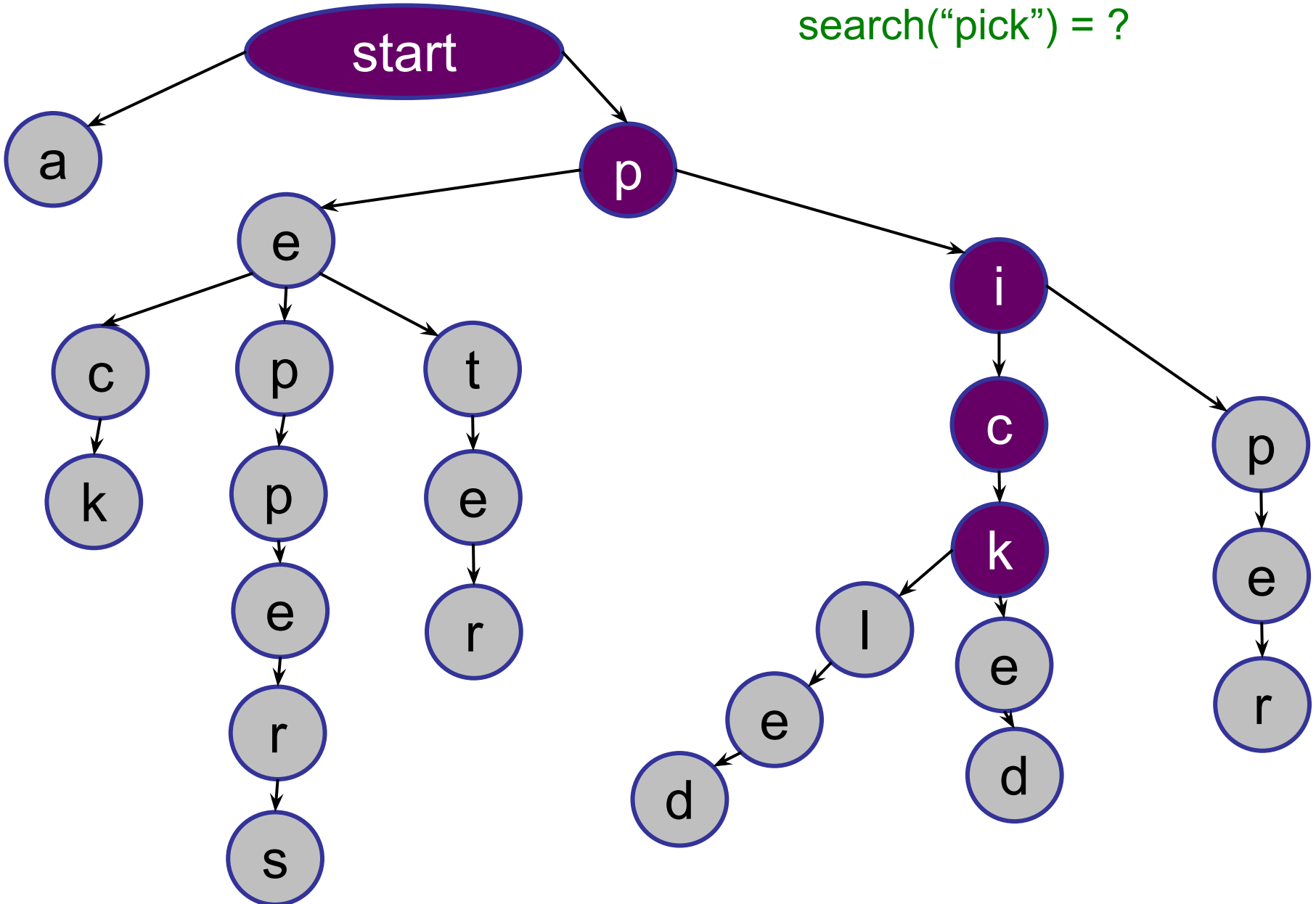


Searching a Trie

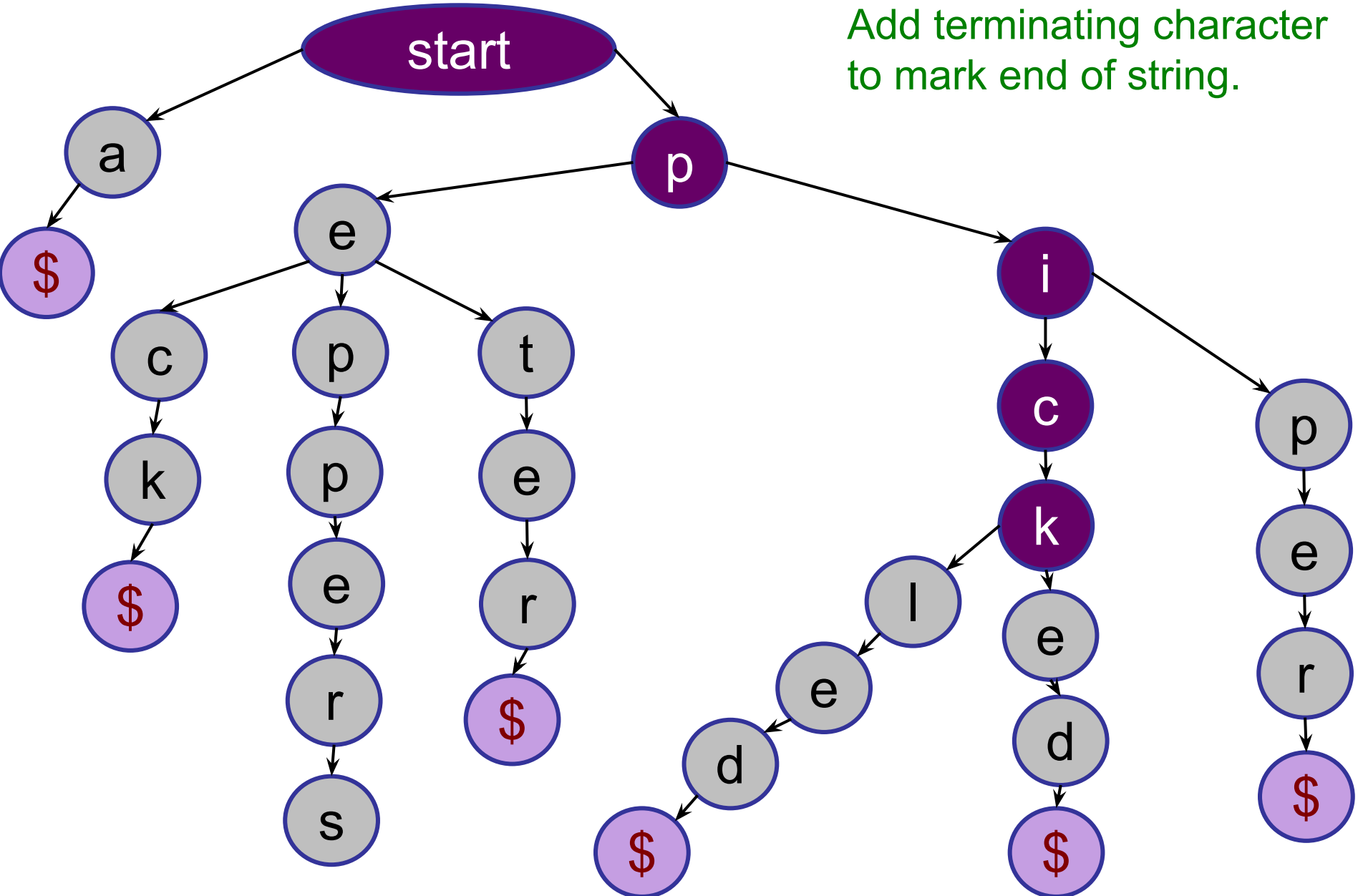


search(“pineapple”) = false

Trie Details

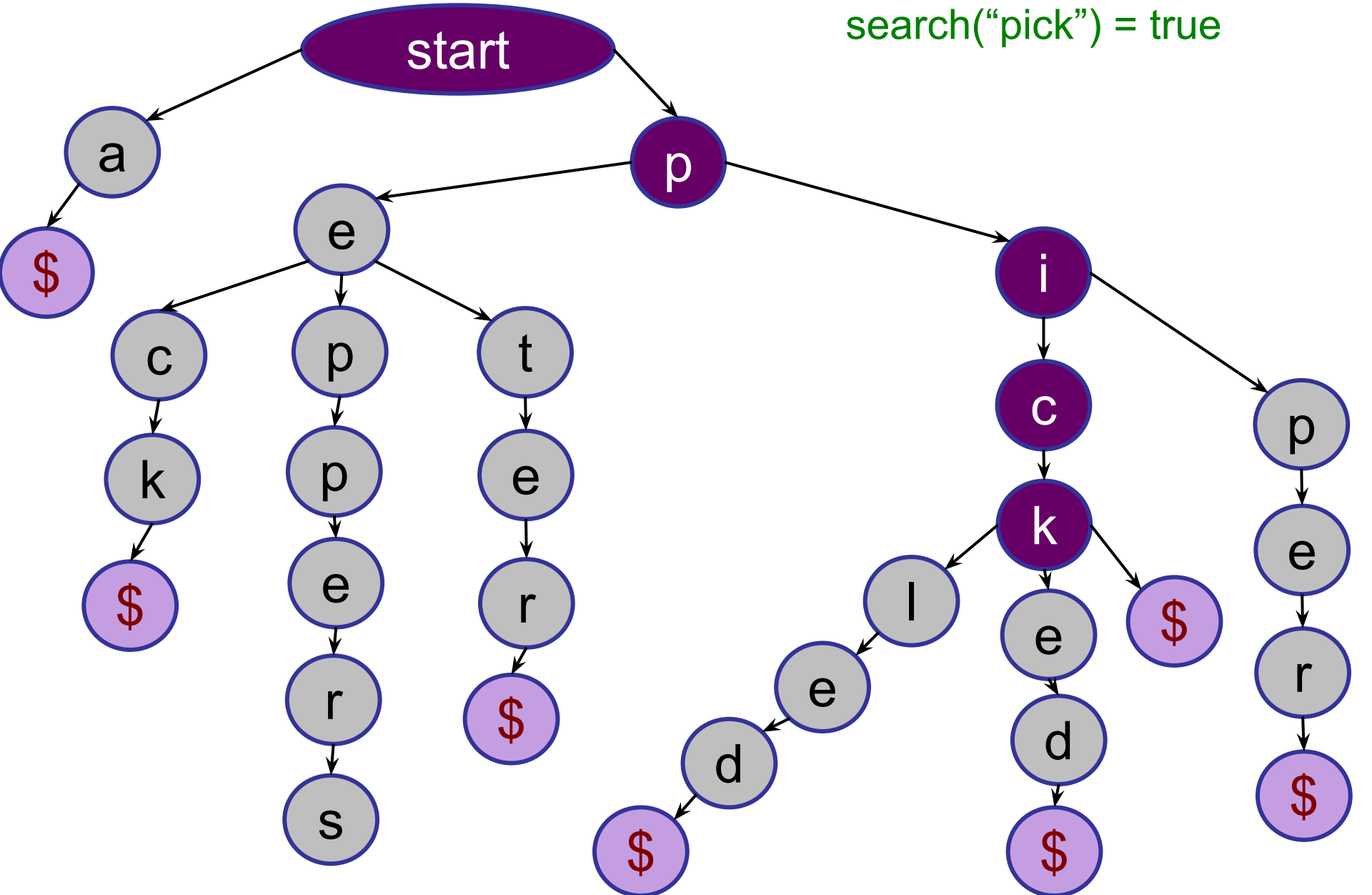


Trie Details



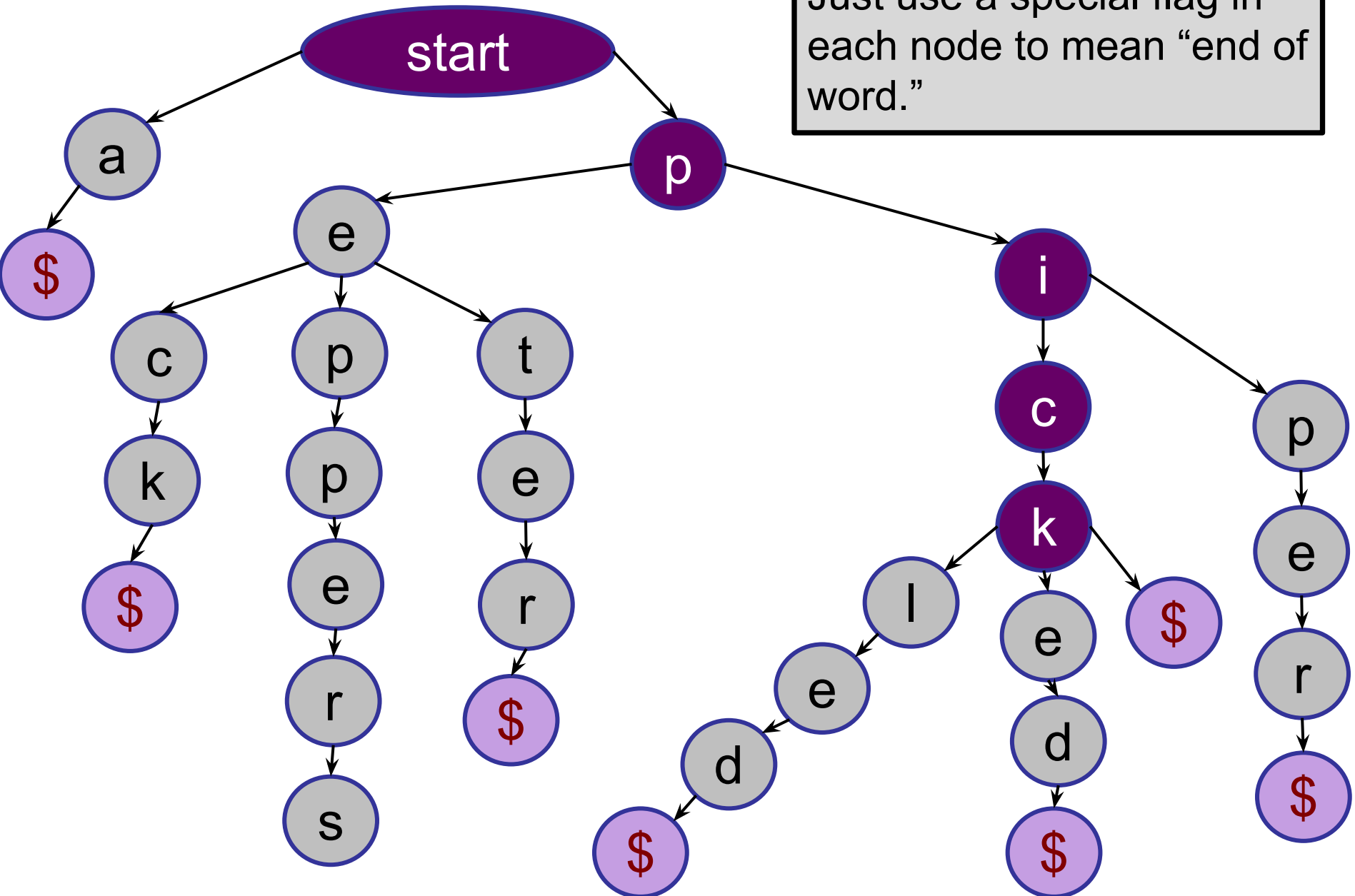
Add terminating character to mark end of string.

Trie Details



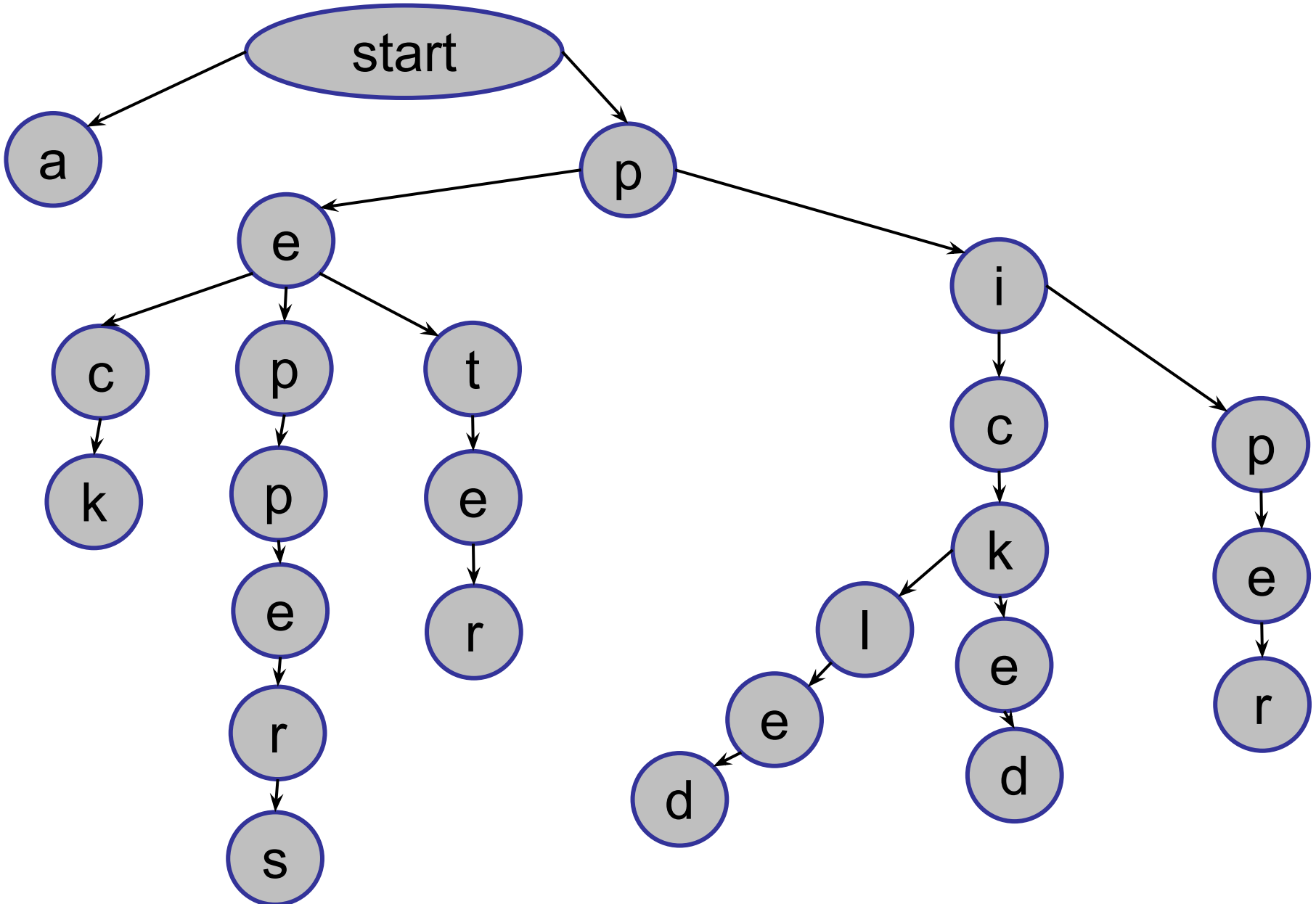
Trie Details

Or:
Just use a special flag in each node to mean “end of word.”



Trie

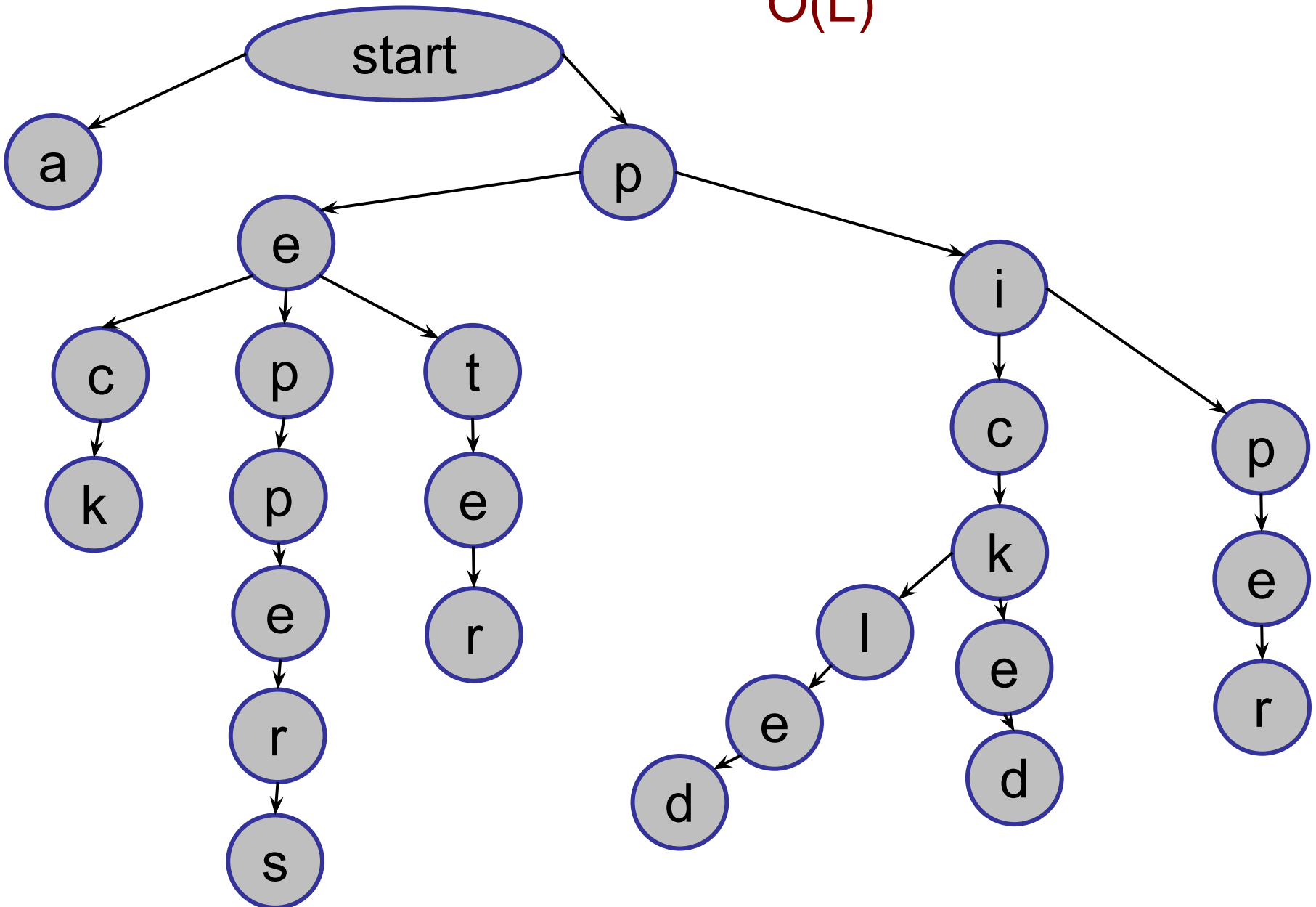
Cost to search for a string of length L?



Trie

Cost to search for a string of length L?

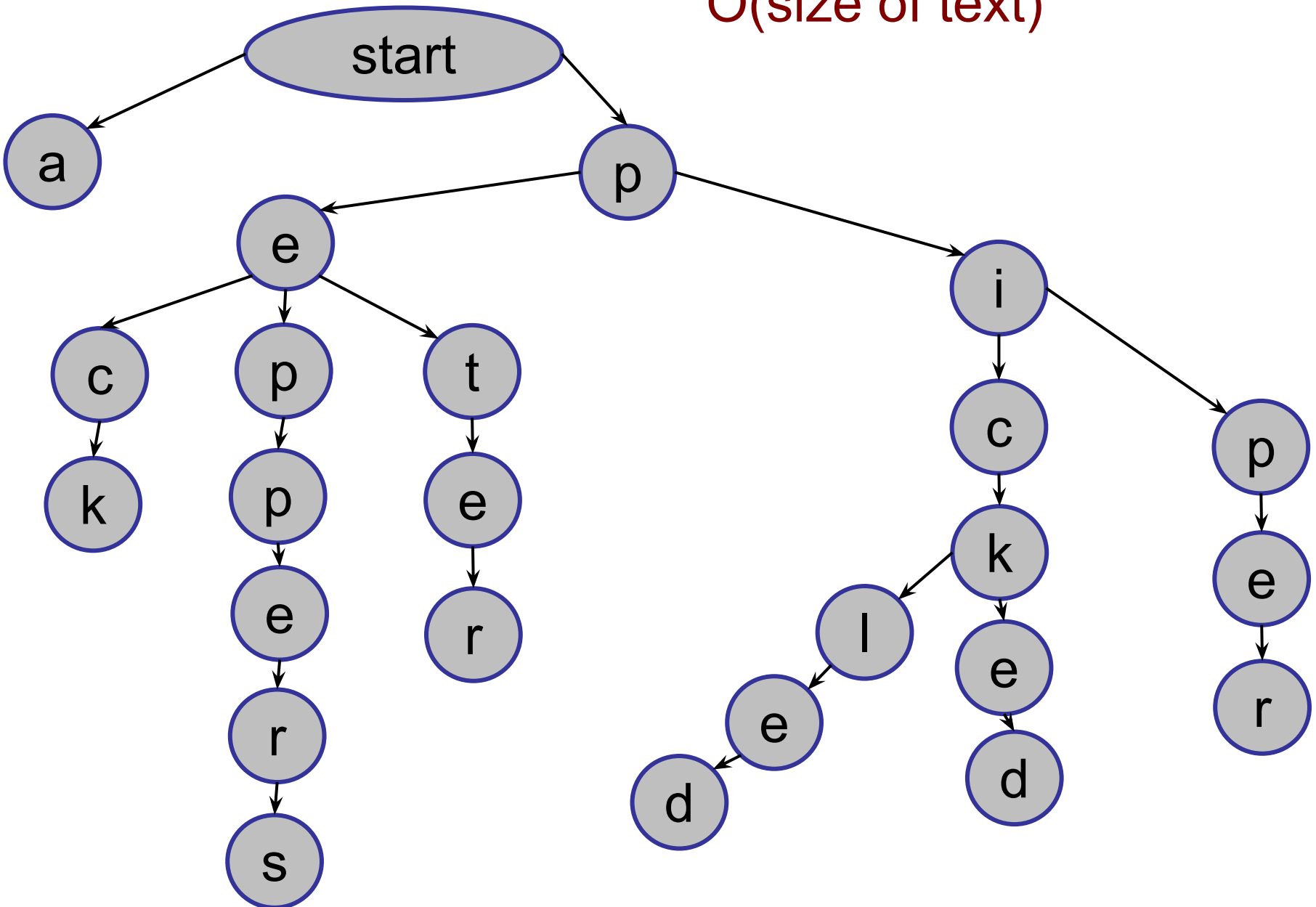
O(L)



Trie

Space for storing a try?

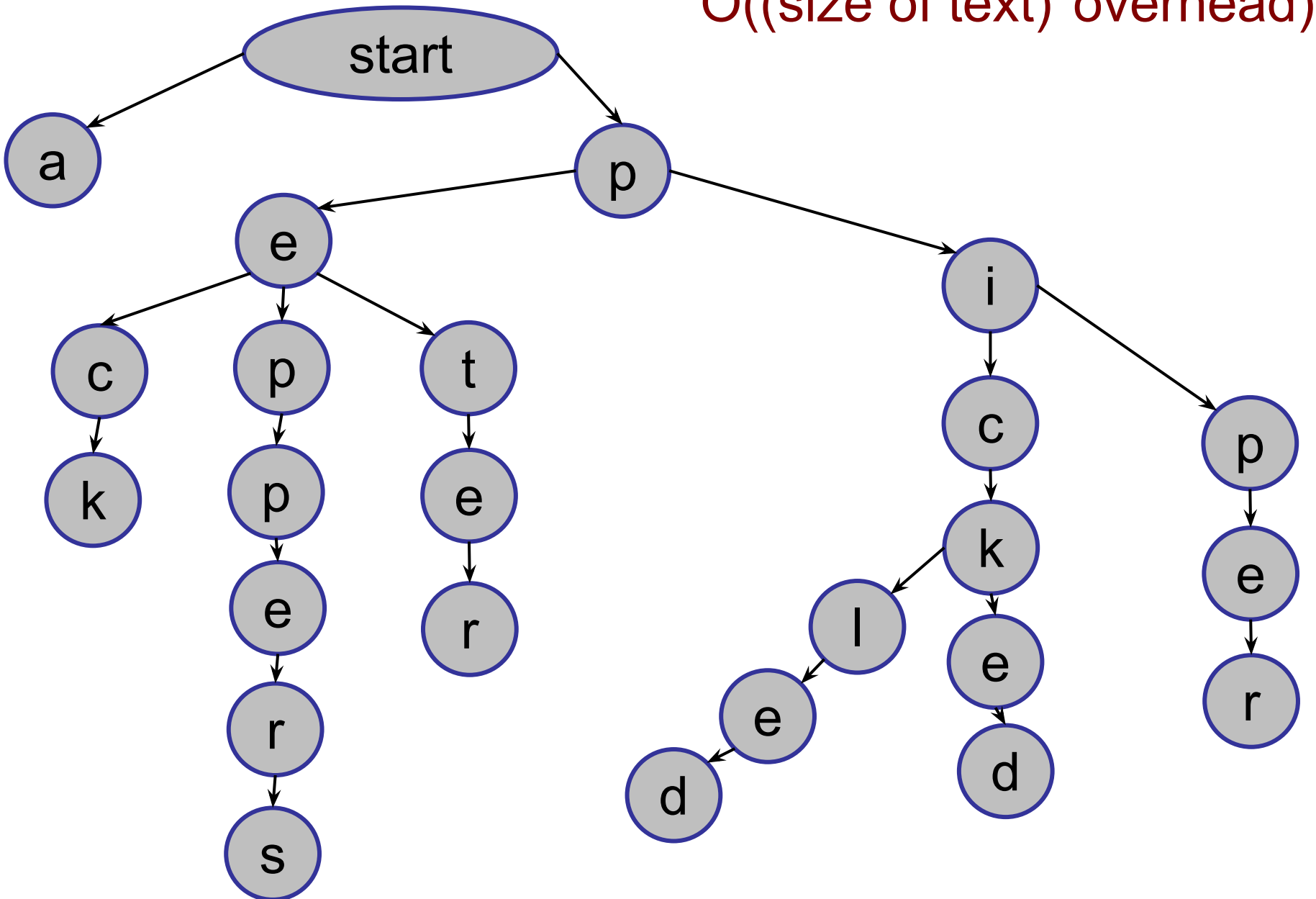
O(size of text)



Trie

Space for storing a try?

O((size of text)*overhead)



Trie Tradeoffs

Time:

- Trie tends to be faster: O(L) vs. O(Lh).
- Does not depend on number of strings.

Even faster if string is not in trie!

Trie Tradeoffs

Time:

- Trie tends to be faster: O(L).
- Does not depend on size of total text.
- Does not depend on number of strings.

Space:

- Trie tends to use more space.
- BST and Trie use O(text size) space.
- But Trie has more nodes and more overhead.

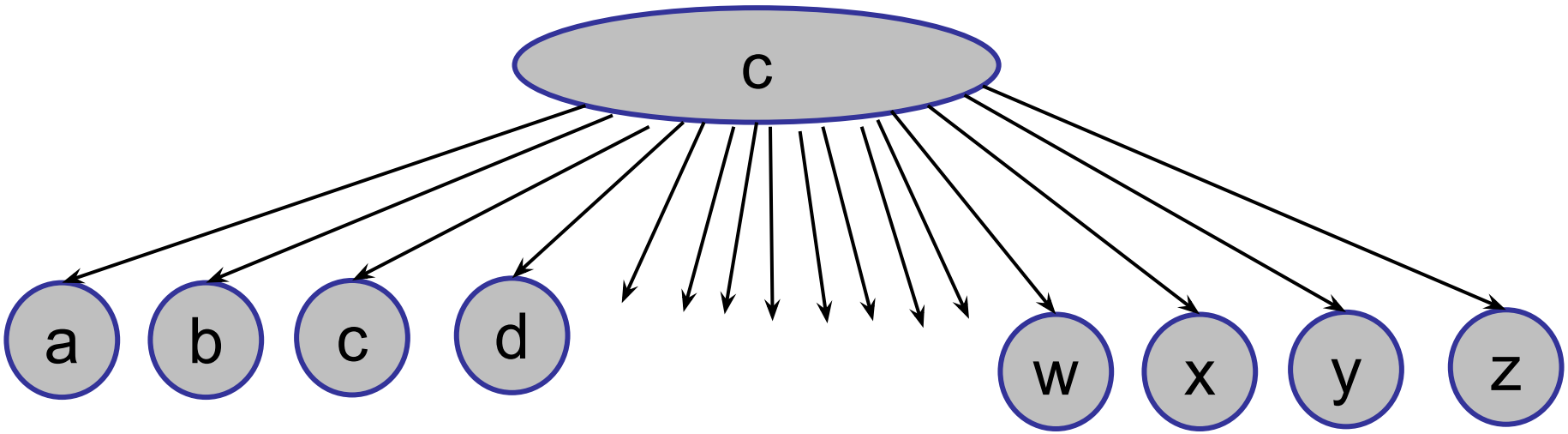
Trie Space

Trie node:

- Has many children.
- For strings: fixed degree.
- Ascii character set: 256

wasted space?

```
TrieNode children[] = new TrieNode[256];
```



Trie Applications

String dictionaries

- Searching
- Sorting / enumerating strings

Partial string operations:

- Prefix queries: find all the strings that start with pi.
- Long prefix: what is the longest prefix of "pickling" in the trie?
- Wildcards: find a string of the form "pi??le" in the trie.

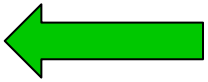
Todays Plan

Data structure design

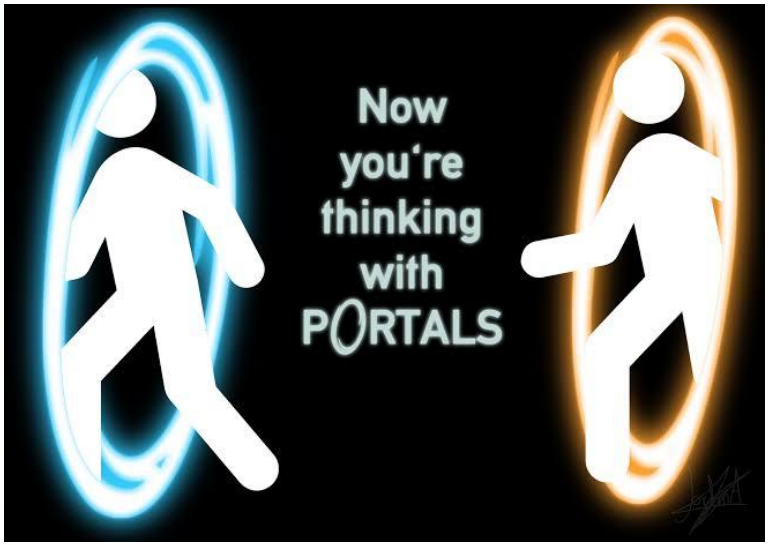
- More Augmentation on Balanced Trees

Tries

- How to handle text?

Problem Solving Using Trees 

- Thinking with Trees


Now
you're
thinking
with
PORTALS


Now
you're
thinking
with
trees

Solving Problems Using Trees

Few things to consider:

1. What are the keys for the tree?

   Determined by how we want to order the values

2. What values do the keys map to?

   Determined by what we actually want to store

3. Which operations did we need?

   Usually related to typical tree operations shown so far

Solving Problems Using Trees

Few things to consider:

What are the keys for the tree?

Determined by how we want to order the values

Sometimes this is a little non-trivial. Will see an example later in lecture.

Solving Problems Using Trees

Few things to consider:

1. What are the keys for the tree?

   Determined by how we want to order the values

2. What values do the keys map to?

   Determined by what we actually want to store

3. Which operations did we need?

   Usually related to typical tree operations shown so far

Solving Problems Using Trees

Few things to consider:

Which operations did we need?

Usually related to typical tree operations shown so far

Typically, if it is a problem where you want:

1. max/min operations
2. rank/select operations
3. successor/predecessor operations

Solving Problems Using Trees

Few things to consider:

Which operations did we need?

Usually related to typical tree operations shown so far

Typically, if it is a problem where you want:

1. max/min operations
2. rank/select operations
3. successor/predecessor operations

Then a tree is helpful!

Solving Problems Using Trees

Few things to consider:

Wh

max/min/rank/select/successor/predecessor are operations where keys need to be orderable.

A tree helps maintain this ordering!

so far

Typically, if it is a problem where you want:

1. max/min operations
2. rank/select operations
3. successor/predecessor operations

Then a tree is helpful!

Counting Inversions

52	7	13	43	22	92	18	9	65	67	87	25

Inversion: A pair (i, j) where:

1. i < j
2. arr[i] > arr[j]

Counting Inversions

52	7	13	43	22	92	18	9	65	67	87	25

E.g. arr[1] > arr[2]
that's one inversion

arr[2] < arr[3]
no inversion there

Counting Inversions

52	7	13	43	22	92	18	9	65	67	87	25

Goal: Count the number of inversions.

Counting Inversions

52	7	13	43	22	92	18	9	65	67	87	25

Goal: Count the number of inversions.

Simple algorithm: Just run insertion sort and count the number of swaps

Counting Inversions

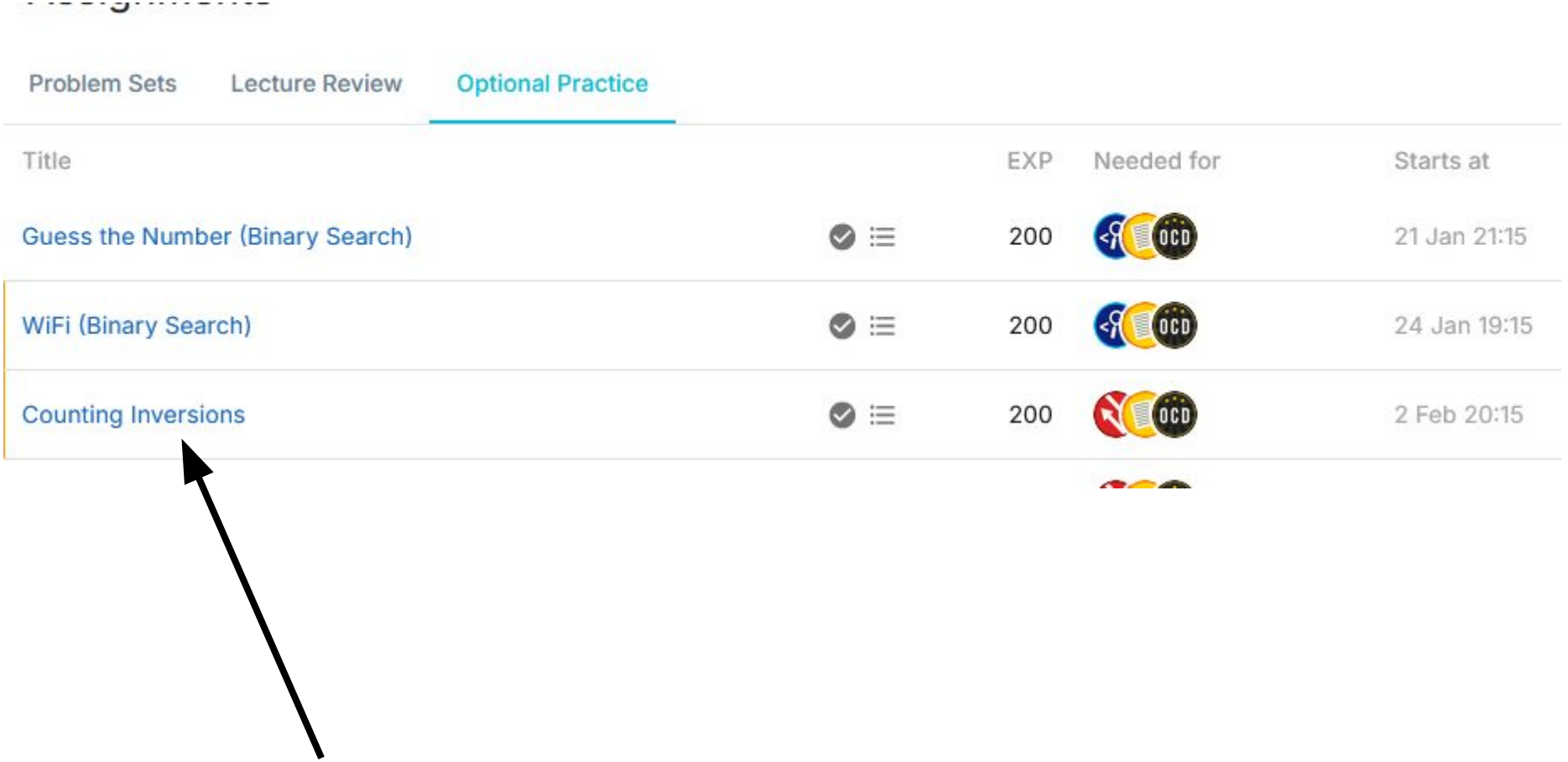
52	7	13	43	22	92	18	9	65	67	87	25

Goal: Count the number of inversions.

Simple algorithm: Just run insertion sort and count the number of swaps

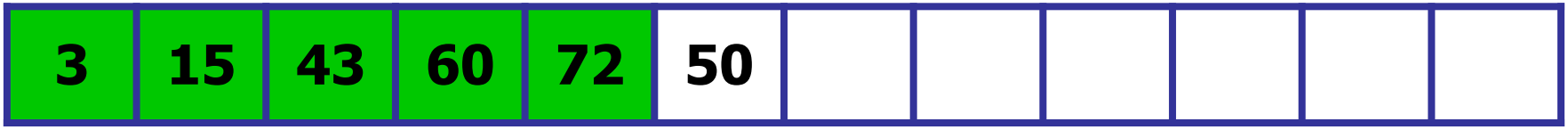
$O(n^2)$ running time

Counting Inversions



intended solution sketch: use mergesort-style recursion

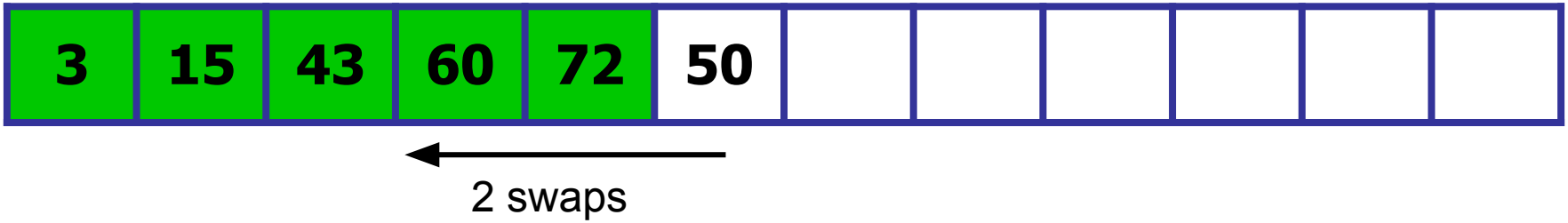
Counting Inversions



Let's re-examine the insertion sort idea.

Recall: To insert the ith element, we find the leftmost index for which it needs to be placed

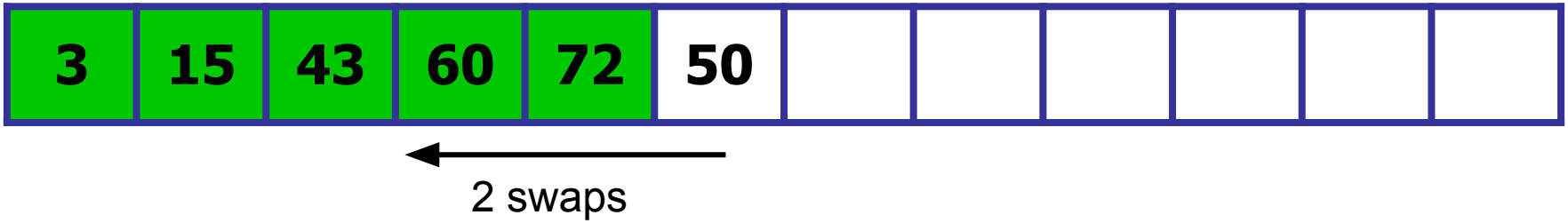
Counting Inversions



Let's re-examine the insertion sort idea.

If an item has to move back 2 places, then that means there are 2 items behind that are out of order with it.

Counting Inversions

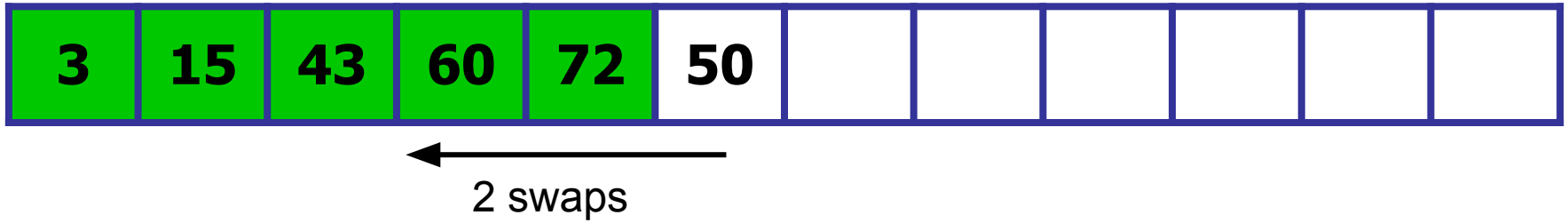


Let's re-examine the insertion sort idea.

If an item has to move back 2 places, then that means there are 2 items behind that are out of order with it.

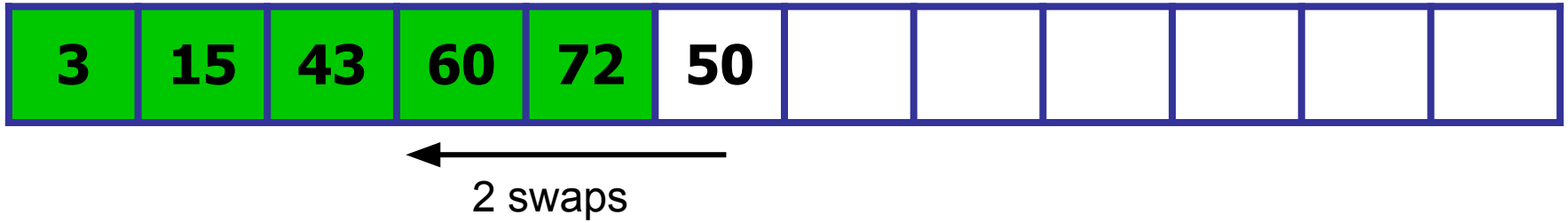
i.e. 2 inversions

Counting Inversions



Alternative view: We want to find “how far back” the i^{th} item has to go.

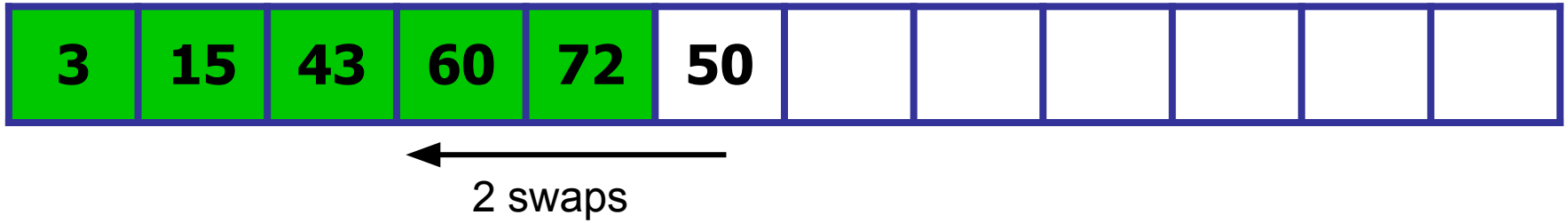
Counting Inversions



Alternative view: We want to find "how far back" the i^{th} item has to go.

If we stored the first 5 items in a tree, which operation helps us here?

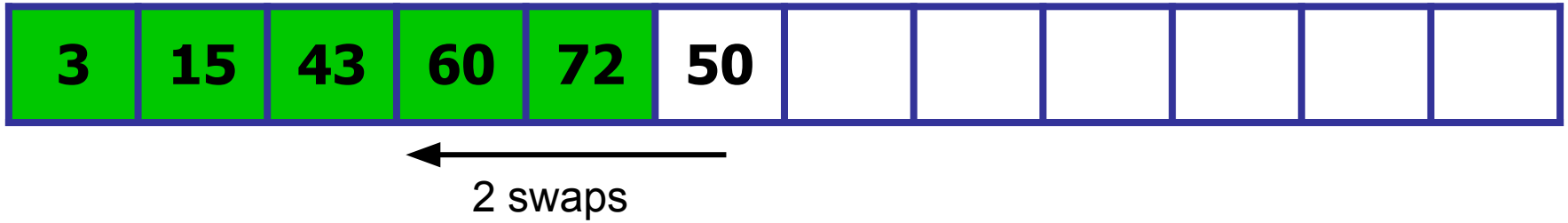
Counting Inversions



There are 5 items, and it ranks 4th
(if we inserted 50 into the tree)

6 - 4 = 2

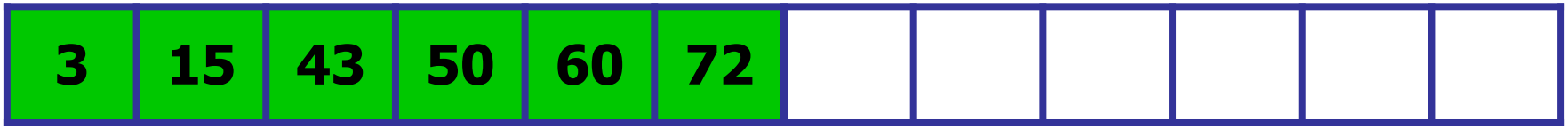
Counting Inversions



There are 5 items, and it ranks 4th
(if we inserted 50 into the tree)

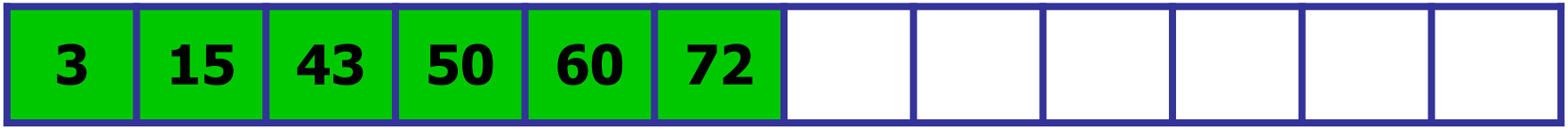
after inserting arr[i] into tree
6 - 4 = 2
i - rank(arr[i])

Counting Inversions



After we insert it, it becomes part of the "sorted" prefix. then we rinse and repeat

Counting Inversions



Pseudo-code:

```
AVL tree   // balanced tree that stores integers
inversion_count = 0

for i from 1 to n
    insert arr[i] into the tree
    inversion_count += (i - tree.rank(arr[i]))

return inversion_count
```

Counting Inversions

Few things to consider:

1. What are the keys for the tree?

   Array elements

2. What values do the keys map to?

   In this case, the keys suffice.

3. Which operations did we need?

   Rank (Ordered statistics)

Another Problem:

We are at a game where people are free to join anytime. (with a unique id) with an initial amount of points.

Another Problem:

We are at a game where people are free to join anytime. (with a unique id) with an initial amount of points.

Once in a while, everyone gets 50 points for just having participated. (only the ones who have joined so far)

Another Problem:

We are at a game where people are free to join anytime. (with a unique id) with an initial amount of points.

Once in a while, everyone gets 50 points for just having participated. (only the ones who have joined so far)

Report the id with the maximum points

Another Problem:

We are at a game where people are free to join anytime. (with a unique id) with an initial amount of points.

insert(id, points)

Once in a while, everyone gets 50 points for just having participated. (only the ones who have joined so far)

reward_all()

Report the id with the maximum points

get_max_id()

Another Problem:

We are at a game where people are free to join anytime. (with a unique id) with an initial amount of points.

insert(id, points)

What is the key?
What is the value?

Once in a while, everyone gets 50 points for just having participated. (only the ones who have joined so far)

reward_all()

Report the id with the maximum points

get_max_id()

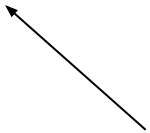
Another Problem:

We are at a game where people are free to join anytime. (with a unique id) with an initial amount of points.

Once in a while, everyone gets 50 points for just having participated. (only the ones who have joined so far)

Report the id with the maximum points

insert(id, points)



What if we used points as the key, and id as the value?

reward_all()

get_max_id()

Another Problem:

We are at a game where people are free to join anytime. (with a unique id) with an initial amount of points.

Once in a while, everyone gets 50 points for just having participated. (only the ones who have joined so far)

Report the id with the maximum points

insert(id, points)

What if we used points as the key, and id as the value?

reward_all()

Then how do we implement this?

get_max_id()

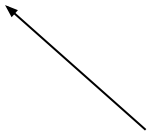
Another Problem:

We are at a game where people are free to join anytime. (with a unique id) with an initial amount of points.

Once in a while, everyone gets 50 points for just having participated. (only the ones who have joined so far)

Report the id with the maximum points

insert(id, points)



What if we used points as the key, and id as the value?

reward_all()

Simple idea: Just rebuild the entire tree.

O(n log n) time.

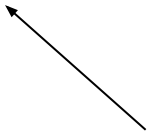
Another Problem:

We are at a game where people are free to join anytime. (with a unique id) with an initial amount of points.

Once in a while, everyone gets 50 points for just having participated. (only the ones who have joined so far)

Report the id with the maximum points

insert(id, points)



What if we used points as the key, and id as the value?

reward_all()

Simple idea: Just rebuild the entire tree.

Smarter way:

O(n) time.

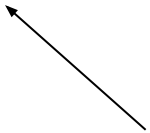
Another Problem:

We are at a game where people are free to join anytime. (with a unique id) with an initial amount of points.

Once in a while, everyone gets 50 points for just having participated. (only the ones who have joined so far)

Report the id with the maximum points

insert(id, points)



What if we were smarter about what key we should insert?

Another Problem:

We are at a game where people are free to join anytime. (with a unique id) with an initial amount of points.

Once in a while, everyone gets 50 points for just having participated. (only the ones who have joined so far)

Report the id with the maximum points

Idea:

1. Store as an extra variable bonus_points, the total number of points disbursed to everyone.

Another Problem:

We are at a game where people are free to join anytime. (with a unique id) with an initial amount of points.

Once in a while, everyone gets 50 points for just having participated. (only the ones who have joined so far)

Report the id with the maximum points

Idea:

1. Store as an extra variable bonus_points, the total number of points disbursed to everyone.

2. Whenever you insert a new person, don't insert points as key, instead:

   points - bonus_points

Another Problem:

Intuition:

The new person who got inserted did not get the bonus_points that was handed out so far. Whereas everyone else in the tree already has.

So the key should be:

points - bonus_points

Idea:

1. Store as an extra variable bonus_points, the total number of points disbursed to everyone.

2. Whenever you insert a new person, don't insert points as key, instead:

   points - bonus_points

Another Problem:

Intuition:

The new person who got inserted did not get the bonus_points that was handed out so far. Whereas everyone else in the tree already has.

So the key should be:

points - bonus_points

Invariant:

Every key in the tree is such that:

key_value + bonus_points

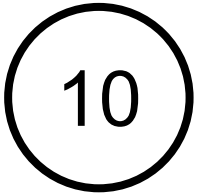
is the actual points that the player has.

Another Problem:

E.g.

bonus_points = 0

Person 1 comes in with 10 points.

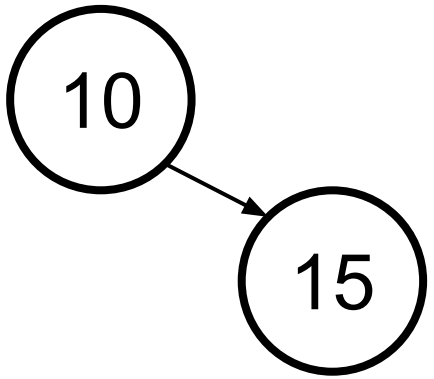


Another Problem:

E.g.

bonus_points = 0

Person 1 comes in with 10 points.



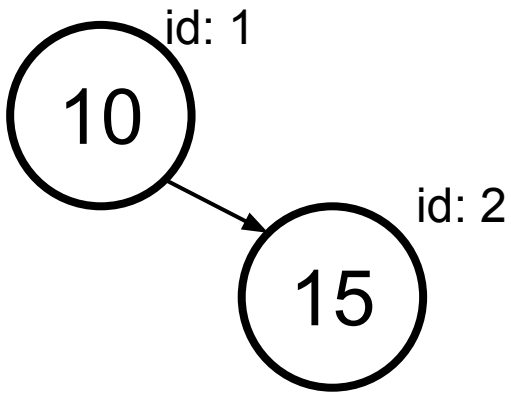
Person 2 comes in with 15 points

Another Problem:

E.g.

bonus_points = 50

Person 1 comes in with 10 points.



Person 2 comes in with 15 points

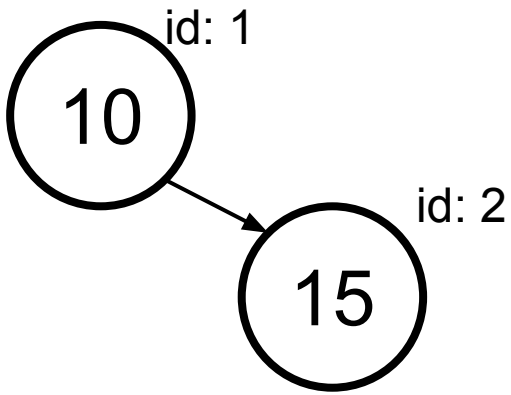
50 Bonus points given!

Another Problem:

E.g.

bonus_points = 50

Person 1 comes in with 10 points.



Person 2 comes in with 15 points

50 Bonus points given!

Person 3 comes in with 63 points

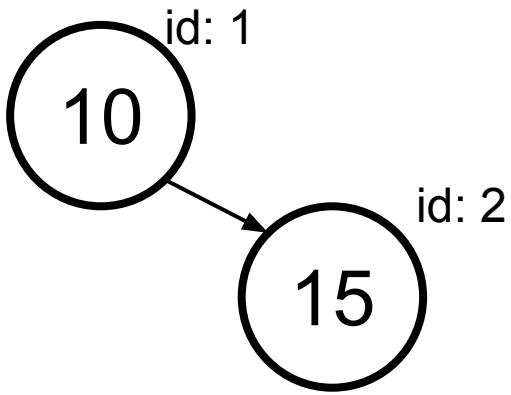
What should we insert?

Another Problem:

E.g.

bonus_points = 50

Person 1 comes in with 10 points.



Person 2 comes in with 15 points

50 Bonus points given!

Person 3 comes in with 63 points

Insert 63 - 50!

Another Problem:

E.g.

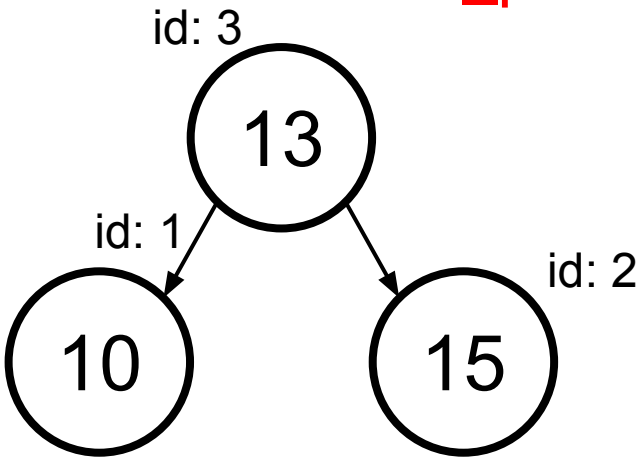
Person 1 comes in with 10 points.

Person 2 comes in with 15 points

50 Bonus points given!

Person 3 comes in with 63 points

bonus_points = 50



Insert 63 - 50!

Another Problem:

E.g.

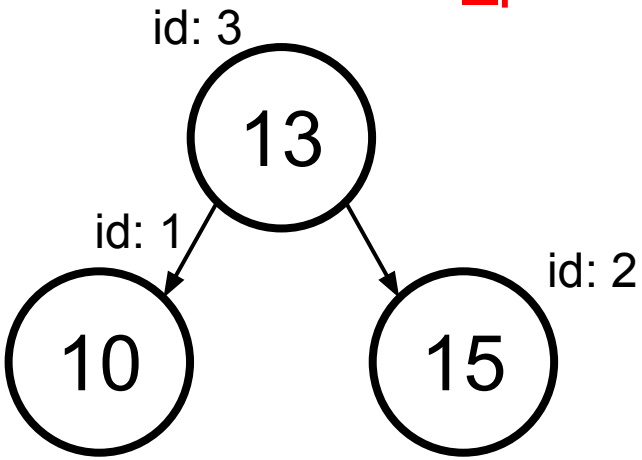
Person 1 comes in with 10 points.

Person 2 comes in with 15 points

50 Bonus points given!

Person 3 comes in with 63 points

bonus_points = 50



Notice how everyone's points + bonus_points is correct.

Another Problem:

E.g.

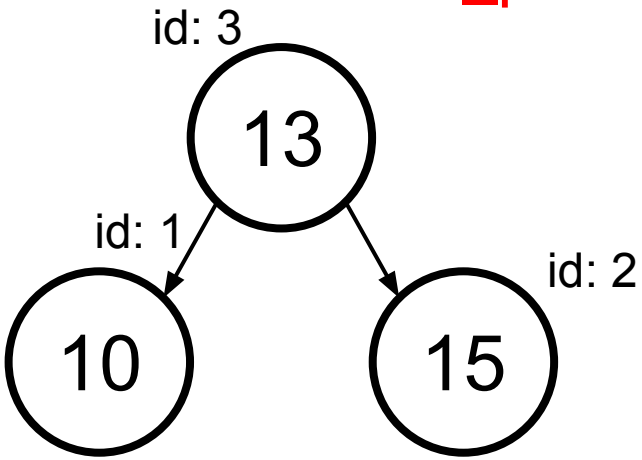
Person 1 comes in with 10 points.

Person 2 comes in with 15 points

50 Bonus points given!

Person 3 comes in with 63 points

bonus_points = 50



To report maximum, we just need the value of the maximum key

E.g. right now it's player 2

Another Problem:

E.g.

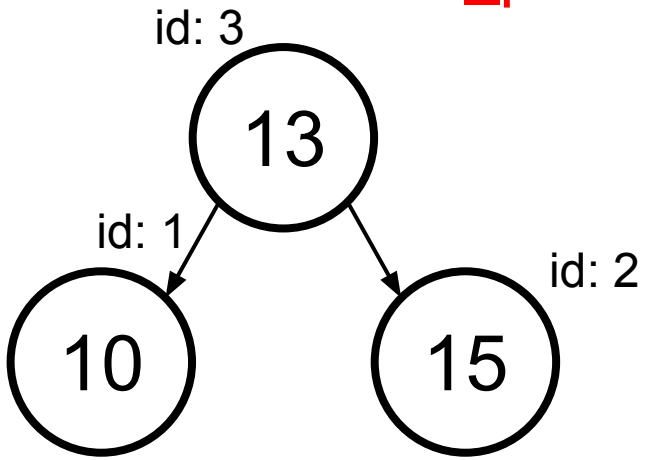
Person 1 comes in with 10 points.

Person 2 comes in with 15 points

50 Bonus points given!

Person 3 comes in with 63 points

bonus_points = 50



If we wanted, we could even report the players in increasing order of points.

Another Problem:

E.g.

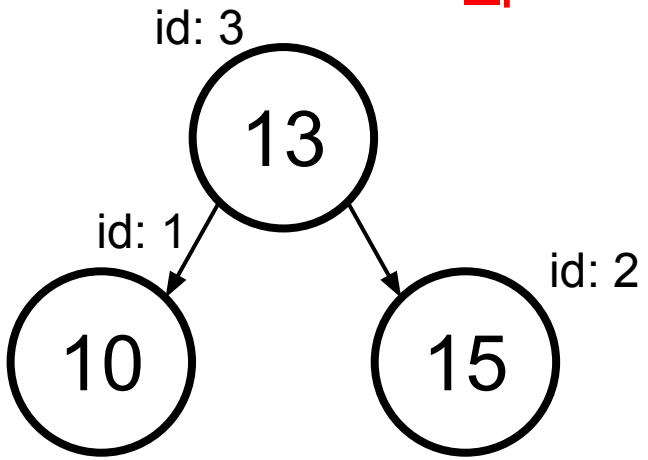
Person 1 comes in with 10 points.

Person 2 comes in with 15 points

50 Bonus points given!

Person 3 comes in with 63 points

bonus_points = 50



If we wanted, we could even report the players in increasing order of points.

Another Problem:

We are at a game where people are free to join anytime. (with a unique id) with an initial amount of points.

insert(id, points)

O(log n)

Once in a while, everyone gets 50 points for just having participated. (only the ones who have joined so far)

reward_all()

O(1)

Report the id with the maximum points

get_max_id()

O(log n)

Another Problem:

Story:

There are n people who live in a neighbourhood from left to right. There are 2 bbq parties that need to be thrown.

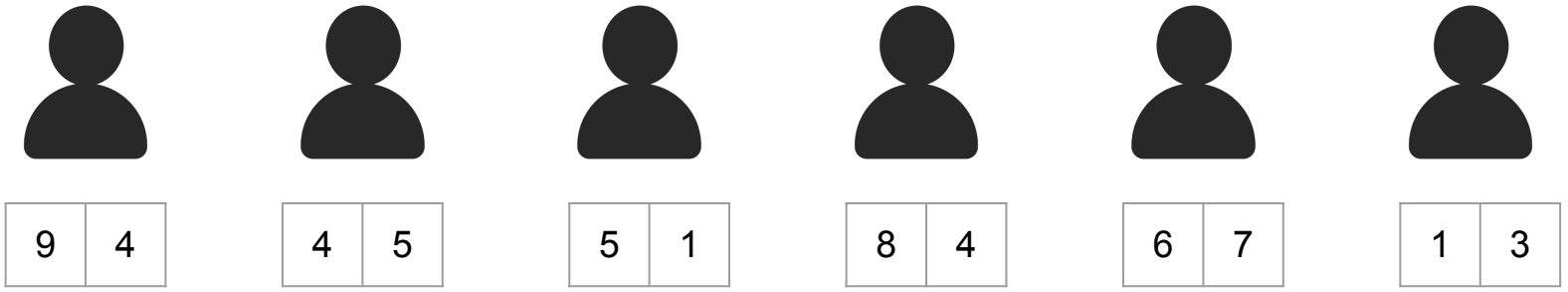
Another Problem:

Story:

There are n people who live in a neighbourhood from left to right. There are 2 bbq parties that need to be thrown.

The left party wants L people, and the right part wants R people.

Another Problem:

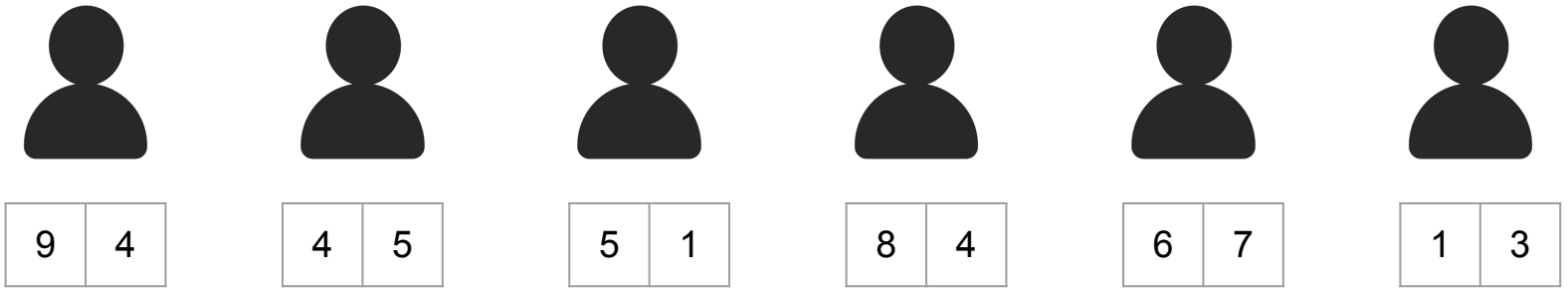


Story:

The left party wants L people, and the right part wants R people.

The ith person offers (a_i, b_i) amount of food — a_i if they join the left party, and b_i if they join the right party.

Another Problem:

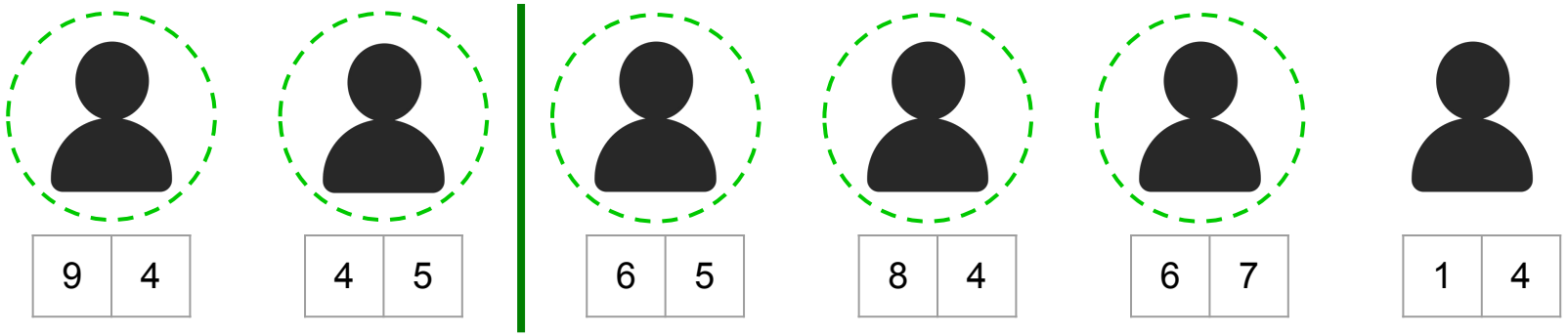


Story:

The ith person offers (a_i, b_i) amount of food — a_i if they join the left party, and b_i if they join the right party.

Restriction: Must draw a line and only take people left of the line for the left party, everyone to the right for the right party

Another Problem:



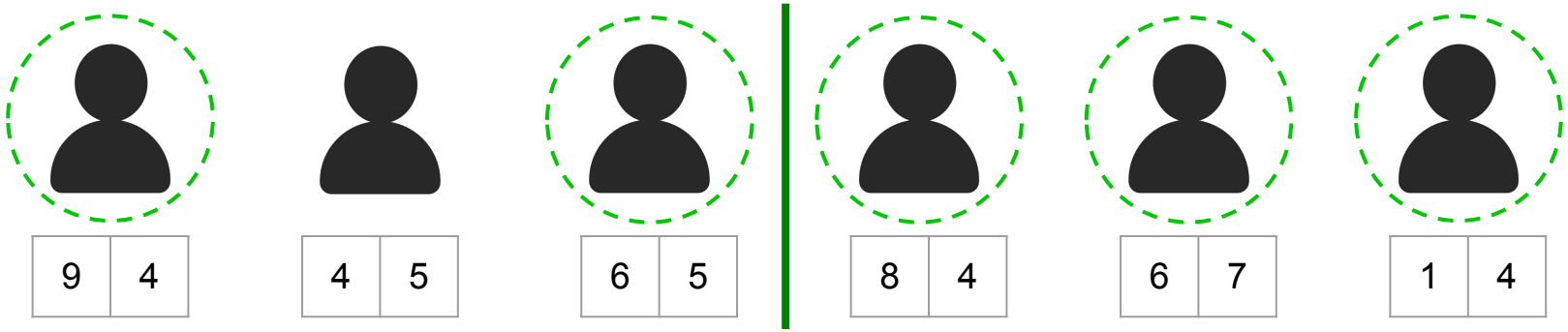
left food: 9 + 4 = 13

right food: 5 + 4 + 7 = 16

E.g. if L = 2, and R = 3, there are 2 possible lines to draw.

total food: 13 + 16 = 29

Another Problem:



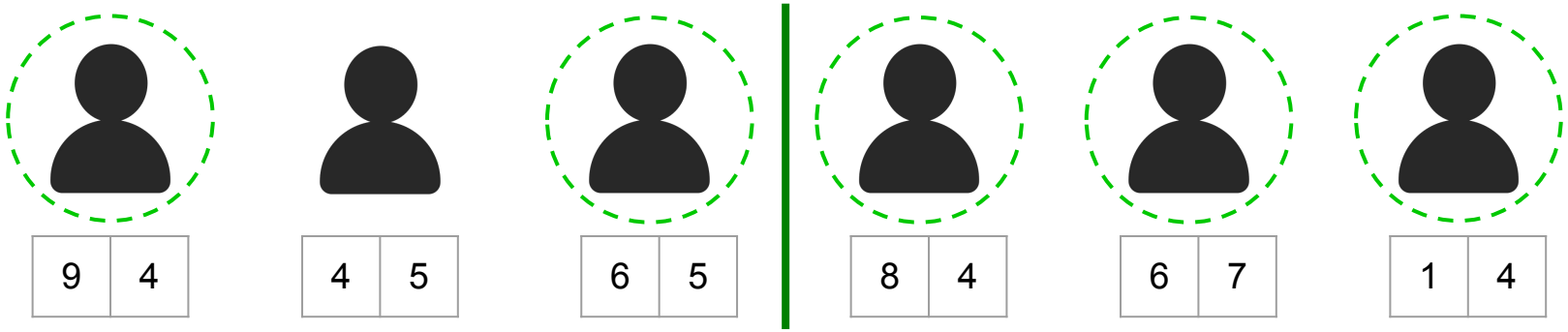
left food: 9 + 6 = 15

right food: 4 + 7 + 4 = 15

E.g. if L = 2, and R = 3, there are 2 possible lines to draw.

total food: 15 + 15 = 30

Another Problem:



left food: 9 + 6 = 15

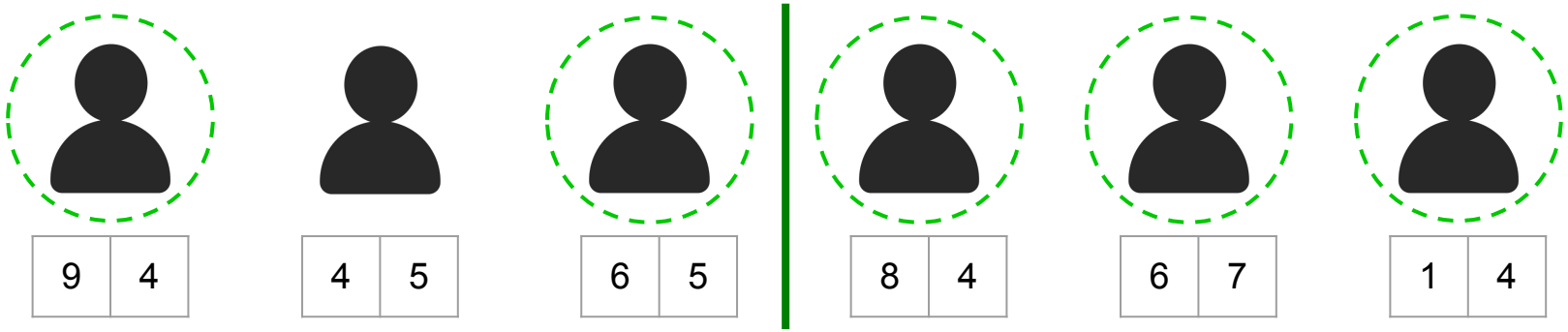
right food: 4 + 7 + 4 = 15

E.g. if L = 2, and R = 3, there are 2 possible lines to draw.

total food: 15 + 15 = 30

max possible was to draw the line in the middle for 30 food

Another Problem:



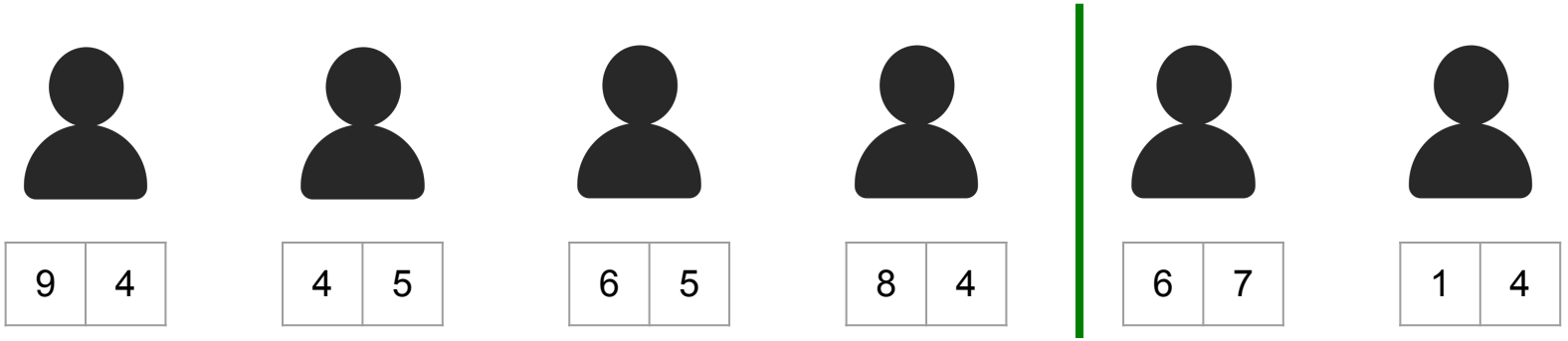
left food: 9 + 6 = 15

right food: 4 + 7 + 4 = 15

In general given n pairs for food, and L and R:

Find the maximum obtainable food.

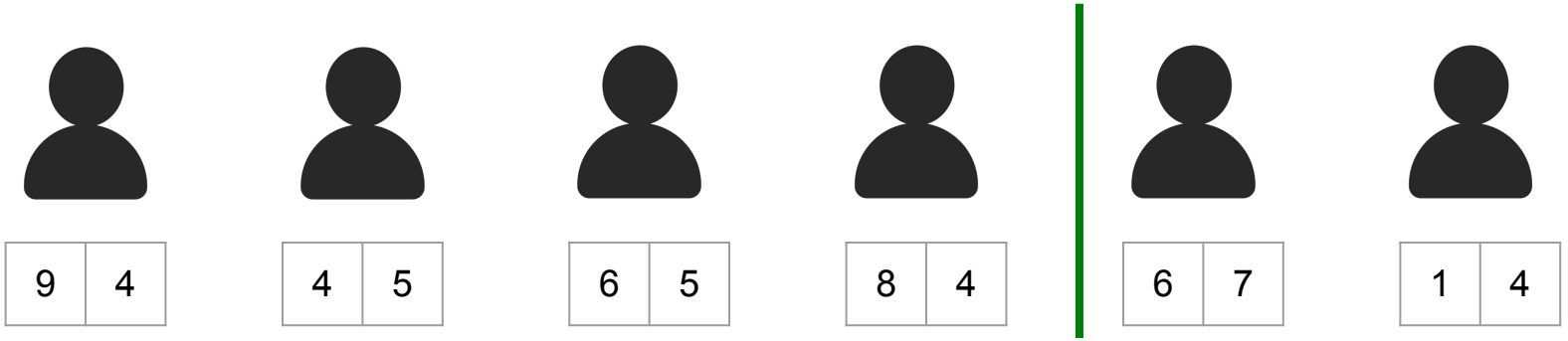
Another Problem:



E.g. n = 6, R = 2, L = 2

Idea 1: If we know R, we know that furthest to the right we can draw the line is (n - R)

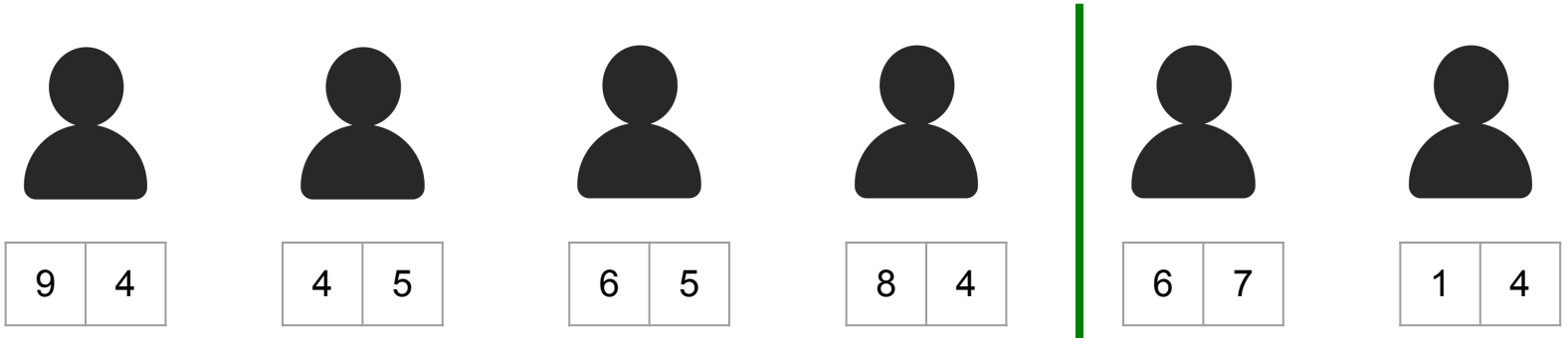
Another Problem:



E.g. n = 6, R = 2, L = 2

At this point with this fixed line, what is the max obtainable food?

Another Problem:

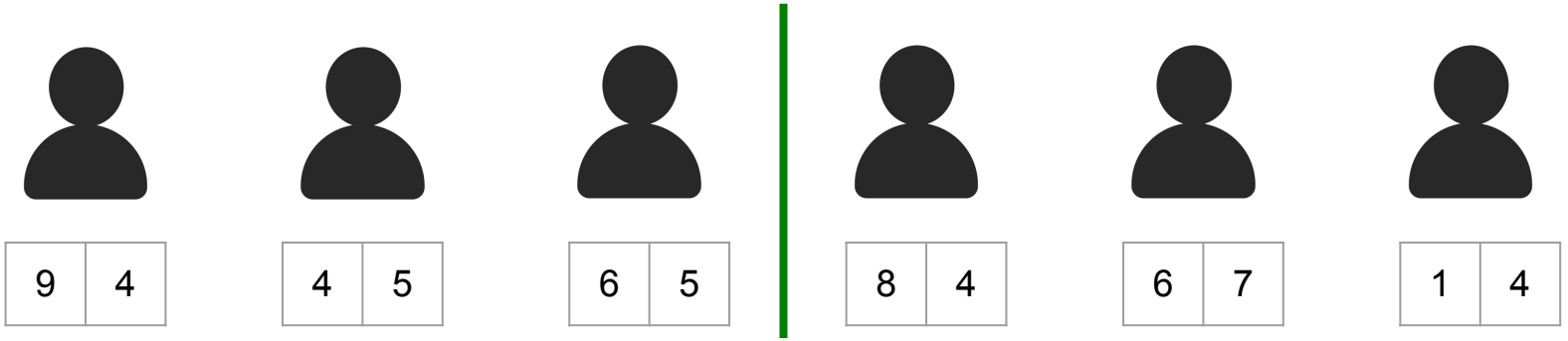


E.g. n = 6, R = 2, L = 2

At this point with this fixed line, what is the max obtainable food?

We want the largest 2 values (among the 2 values) to the right of the line, and the 2 largest values to the left of the line (among the 4 values)

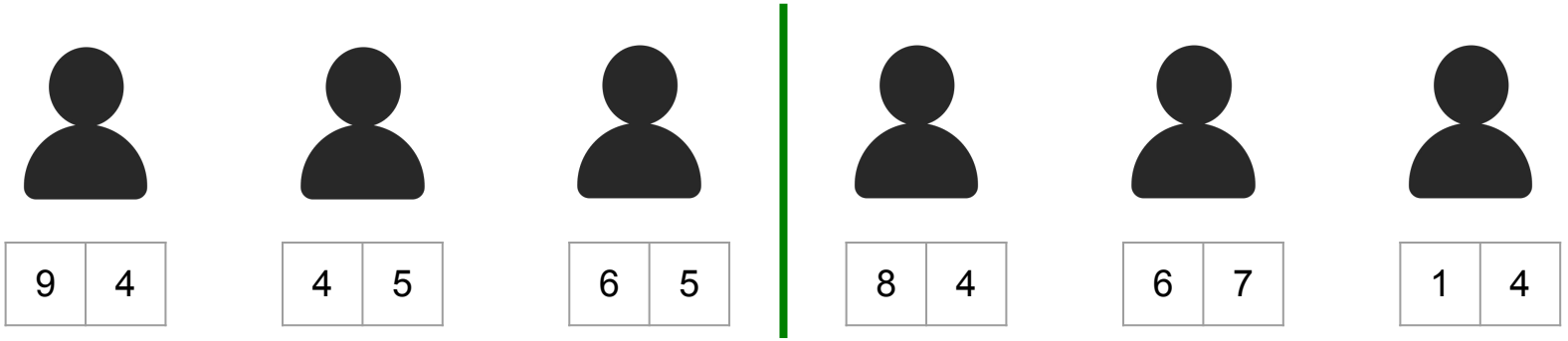
Another Problem:



E.g. n = 6, R = 2, L = 2

Next: Let's say we moved the line left, what is the max obtainable food now?

Another Problem:

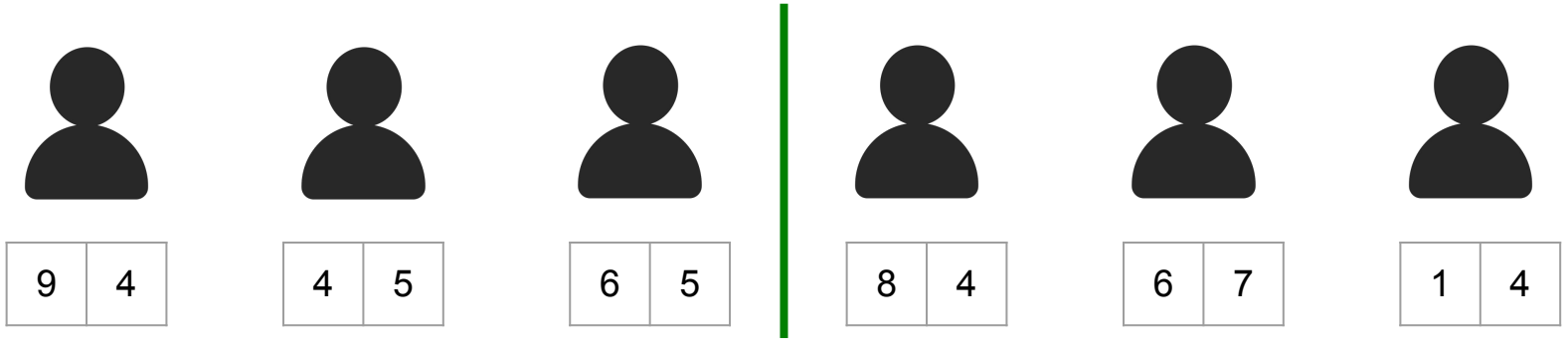


E.g. n = 6, R = 2, L = 2

Next: Let's say we moved the line left, what is the max obtainable food now?

We want the largest 2 values (among the 3 values) to the right of the line, and the 2 largest values to the left of the line (among the 3 values)

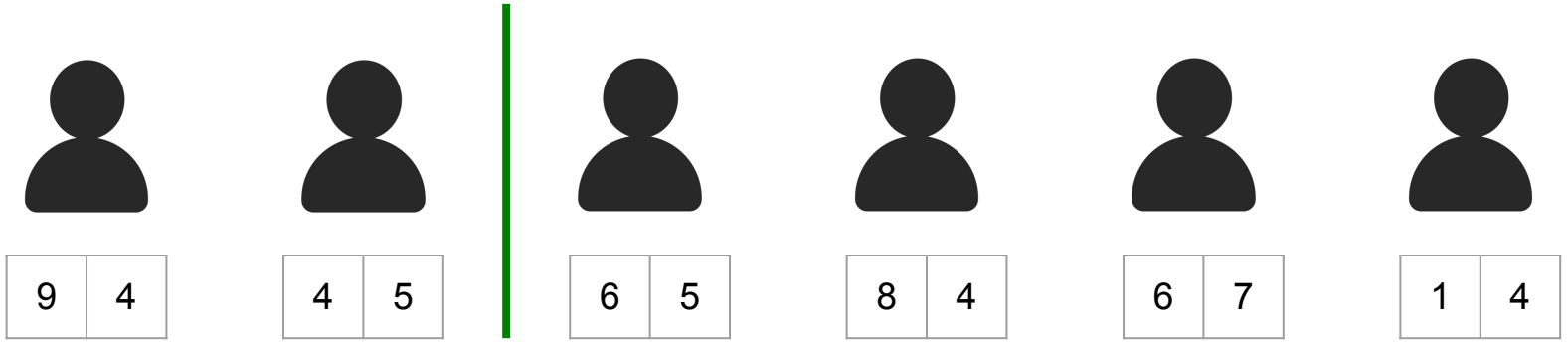
Another Problem:



E.g. n = 6, R = 2, L = 2

Next: When do we stop moving the line?

Another Problem:

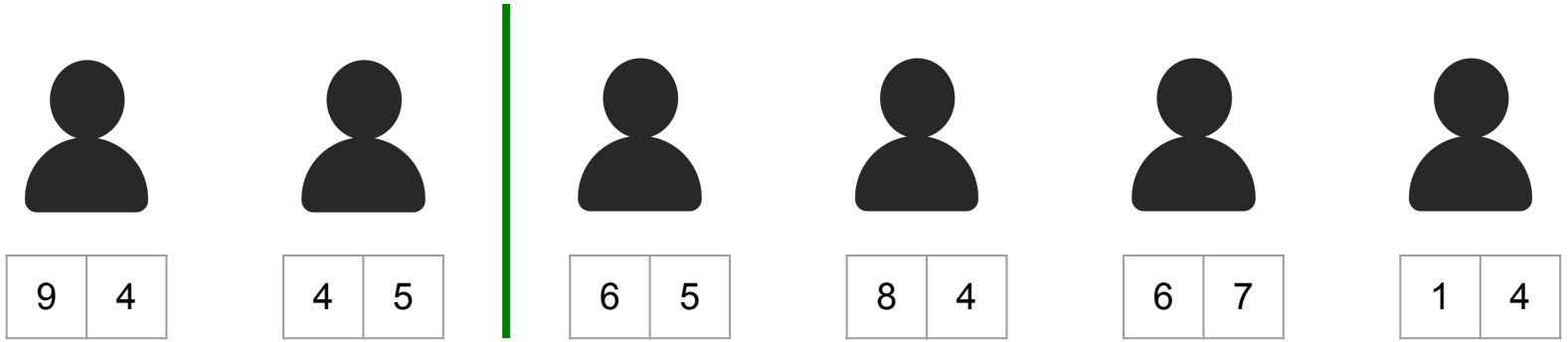


E.g. n = 6, R = 2, L = 2

Next: When do we stop moving the line?

When the line only has L = 2 people to the left of it.

Another Problem:



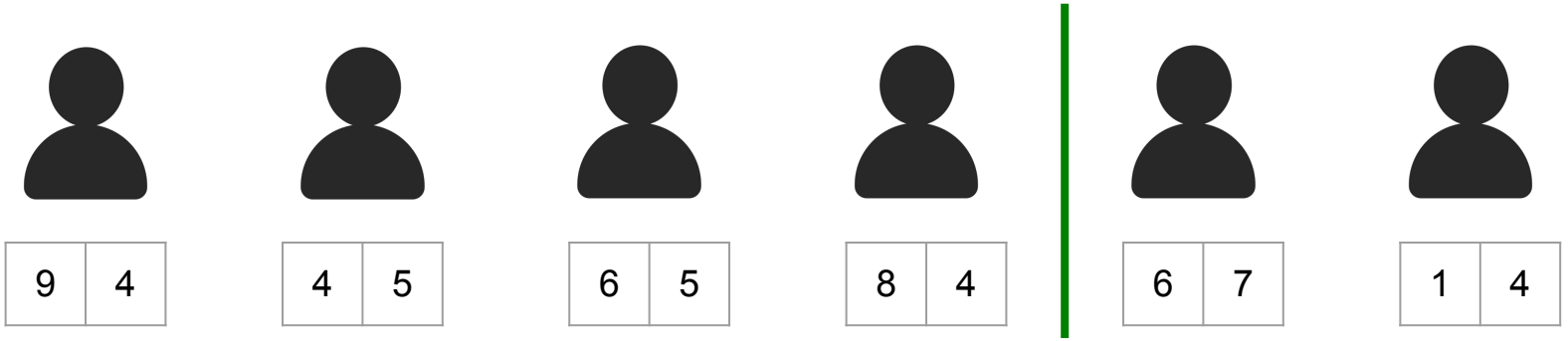
E.g. n = 6, R = 2, L = 2

Next: When do we stop moving the line?

When the line only has L = 2 people to the left of it.

Then for the last time we try to take the largest 2 values to the left of the line, and largest 2 to the right.

Another Problem:



E.g. n = 6, R = 2, L = 2

max_food = 0

set the line so that R people are to the right of the line

while the line has >= L people to the left of the line

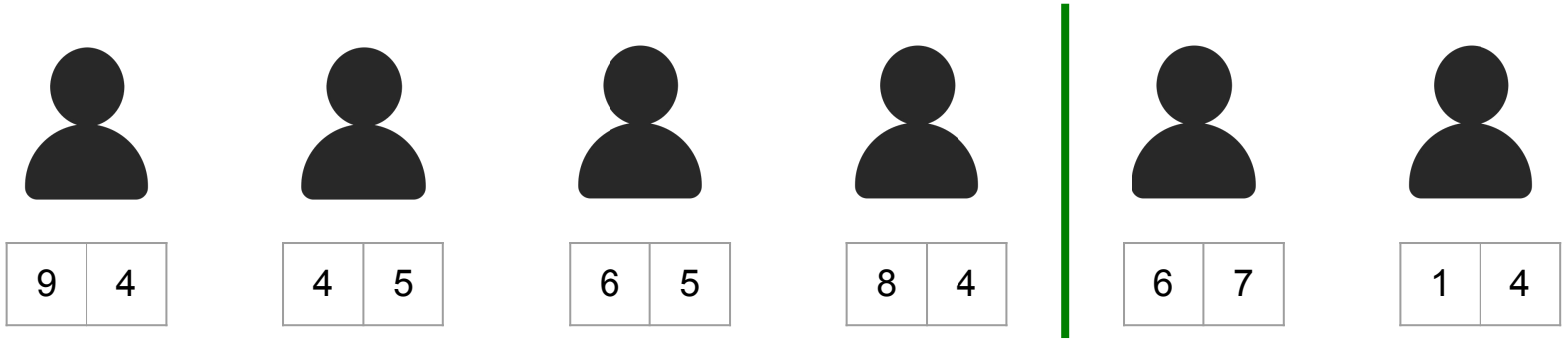
    find the R largest b values on the right of the line, sum them up

    find the L largest a values on the left of the line, sum them up

    max_food = max(max_food, sum of both values)

return max

Another Problem:



What's the time complexity?

max_food = 0

set the line so that R people are to the right of the line

while the line has >= L people to the left of the line

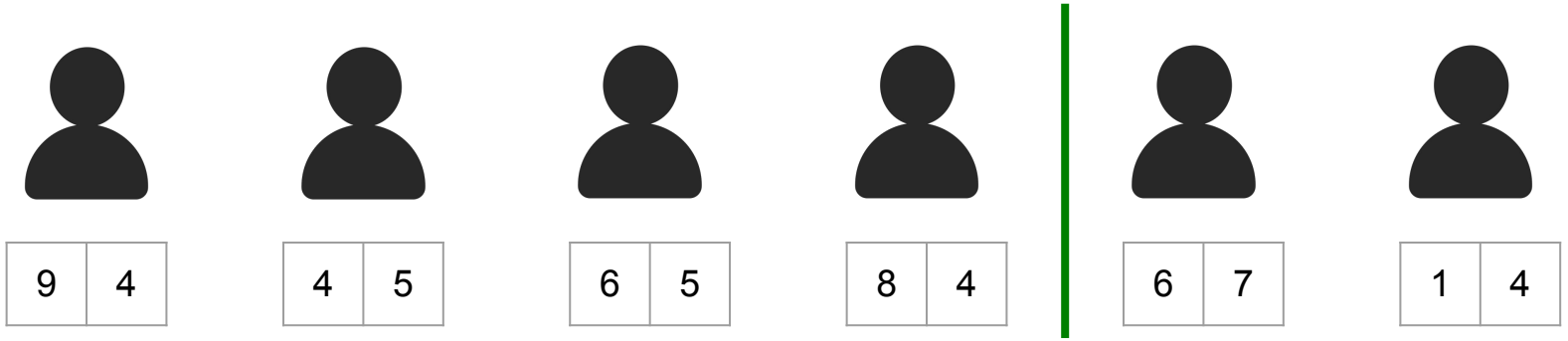
    find the R largest b values on the right of the line, sum them up

    find the L largest a values on the left of the line, sum them up

    max_food = max(max_food, sum of both values)

return max

Another Problem:



What's the time complexity?

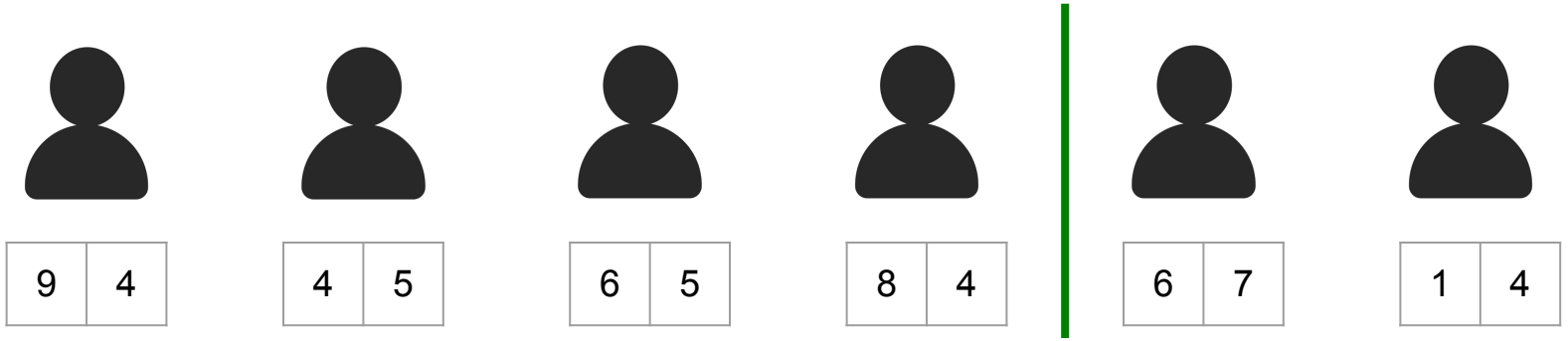
O(n) iterations

O(R) cost

O(L) cost

```
max_food = 0
set the line so that R people are to the right of the line
while the line has >= L people to the left of the line
    find the R largest b values on the right of the line, sum them up
    find the L largest a values on the left of the line, sum them up
    max_food = max(max_food, sum of both values)
return max
```

Another Problem:



If R = O(n), L = O(n), then time complexity = $O(n^2)$

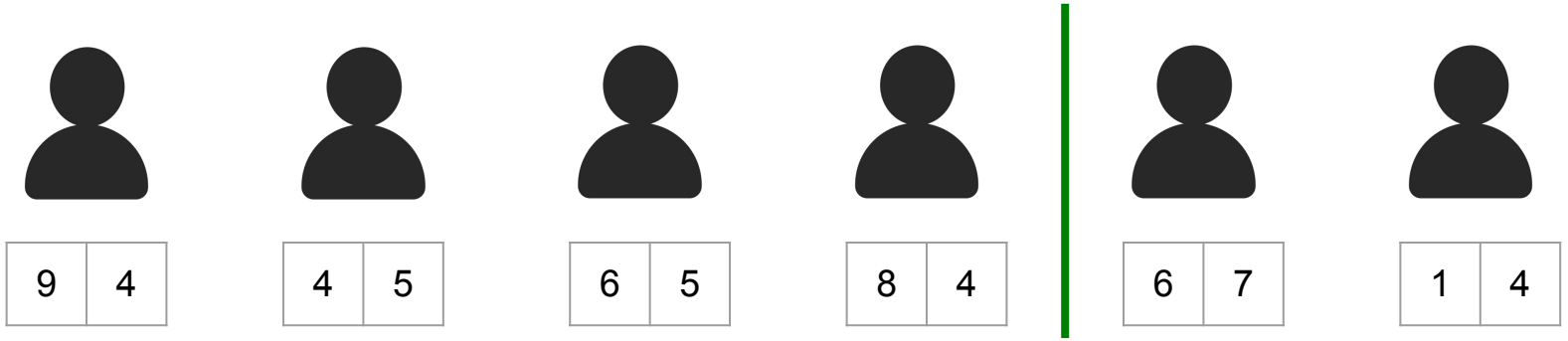
O(n) iterations

O(R) cost

O(L) cost

```
max_food = 0
set the line so that R people are to the right of the line
while the line has >= L people to the left of the line
    find the R largest b values on the right of the line, sum them up
    find the L largest a values on the left of the line, sum them up
    max_food = max(max_food, sum of both values)
return max
```

Another Problem:



$O(n^2)$ Can we do better?

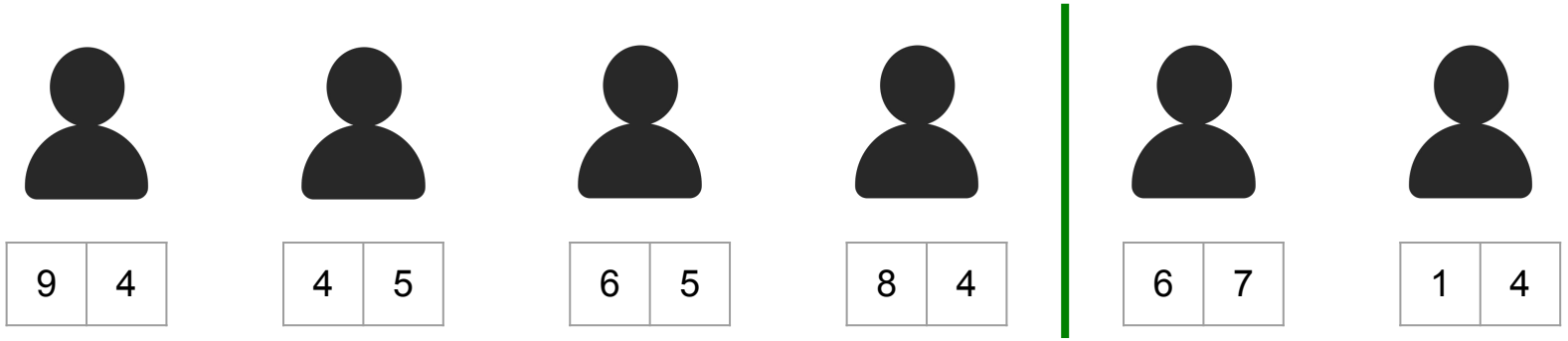
O(n) iterations

O(R) cost

O(L) cost

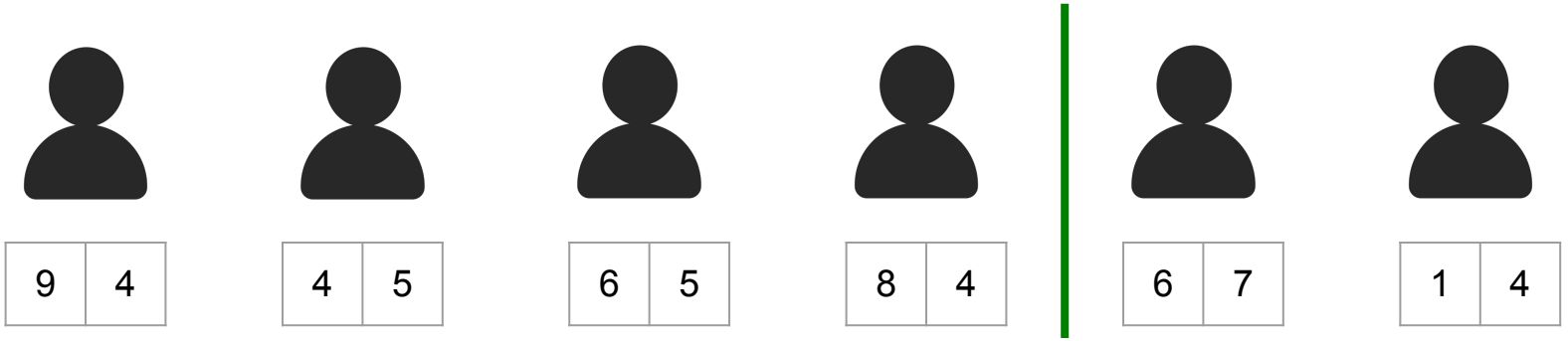
```
max_food = 0
set the line so that R people are to the right of the line
while the line has >= L people to the left of the line
    find the R largest b values on the right of the line, sum them up
    find the L largest a values on the left of the line, sum them up
    max_food = max(max_food, sum of both values)
return max
```

Another Problem:



Broad Strokes: Maintain 2 trees. This way we can use rank/select to select the largest L/R values.

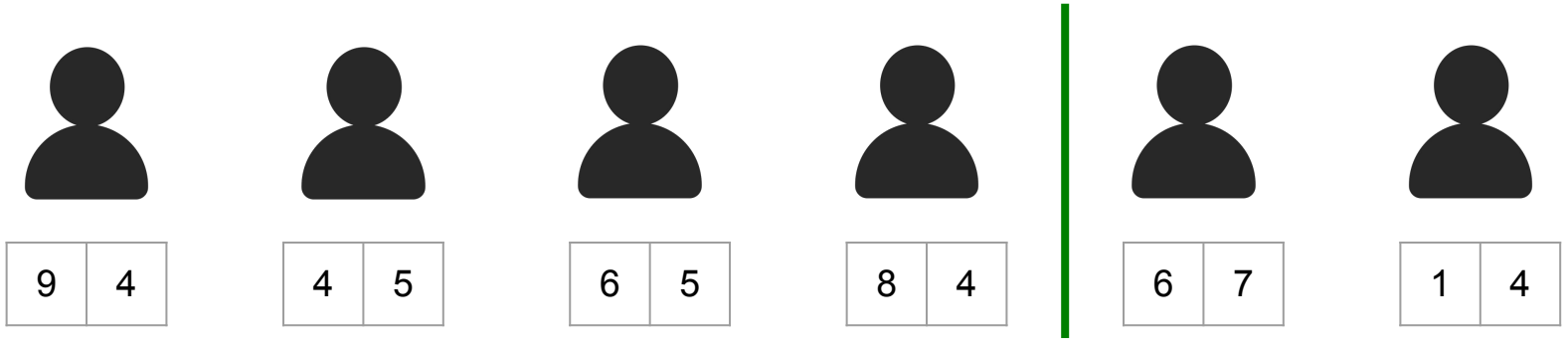
Another Problem:



Broad Strokes: Maintain 2 trees. This way we can use rank/select to select the largest L/R values.

If we do this right, it is O(n log n).

Another Problem:

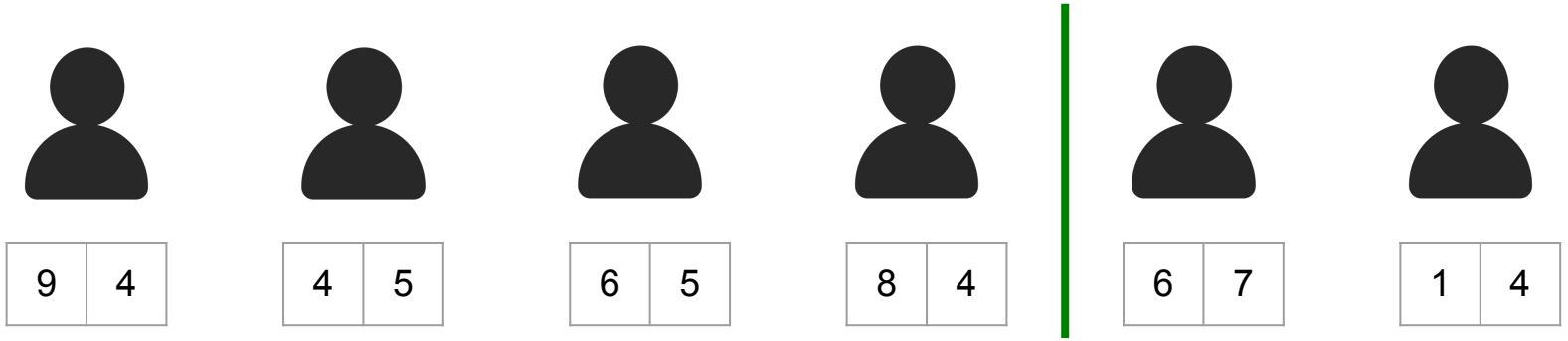


Broad Strokes: Maintain 2 trees. This way we can use rank/select to select the largest L/R values.

If we do this right, it is O(n log n).

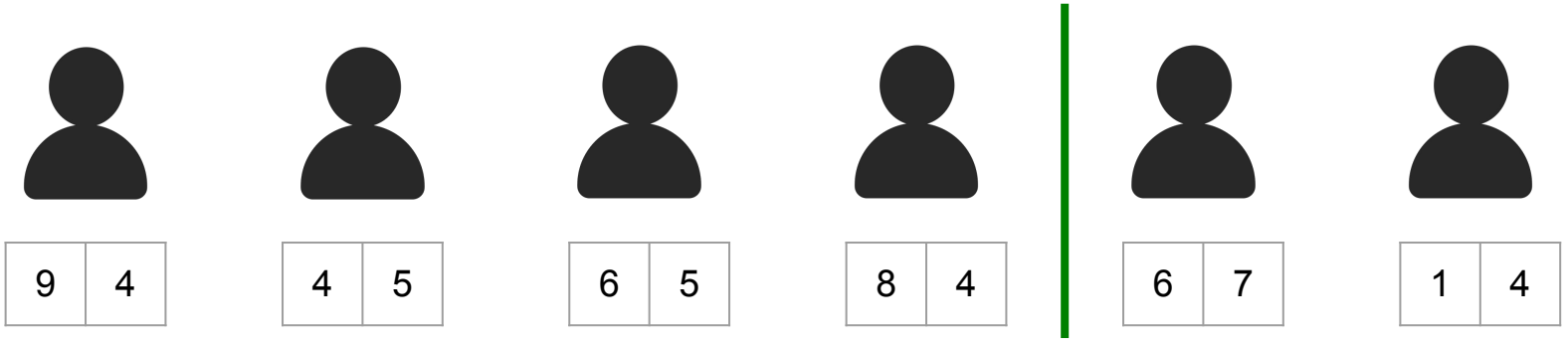
Warning! Details are important! How EXACTLY we wish to use the tree matters!

Another Problem:



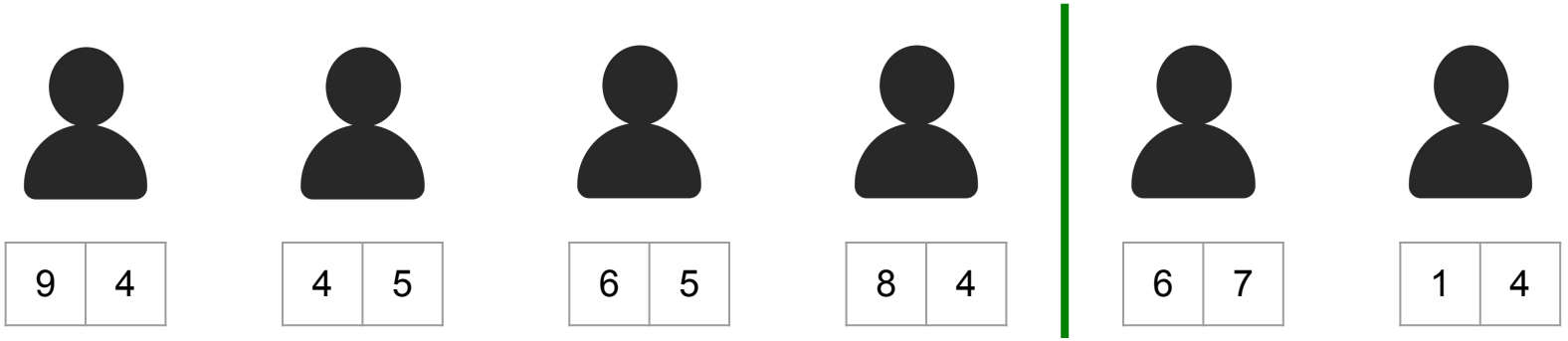
E.g. if we simply use the tree to enumerate the largest L/R values, is it actually O(log n) time?

Another Problem:



E.g. if we simply use the tree to enumerate the largest L/R values, is it actually O(log n) time? No!

Another Problem:



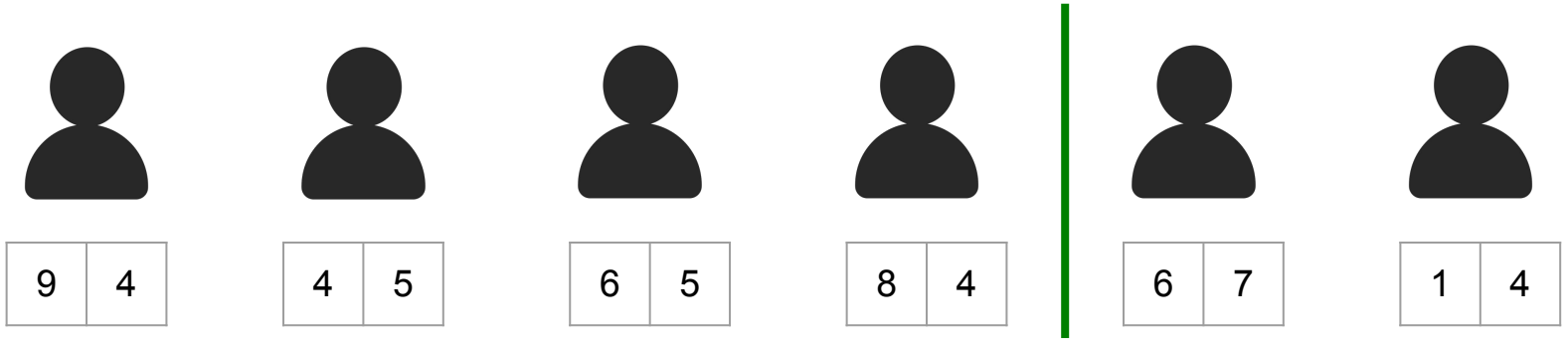
Let's see what we want in from the tree. Pay attention to:

1. What the keys are.
2. What the values are.
3. Which operations we want to use.

Another Problem:

L = 2

R = 2

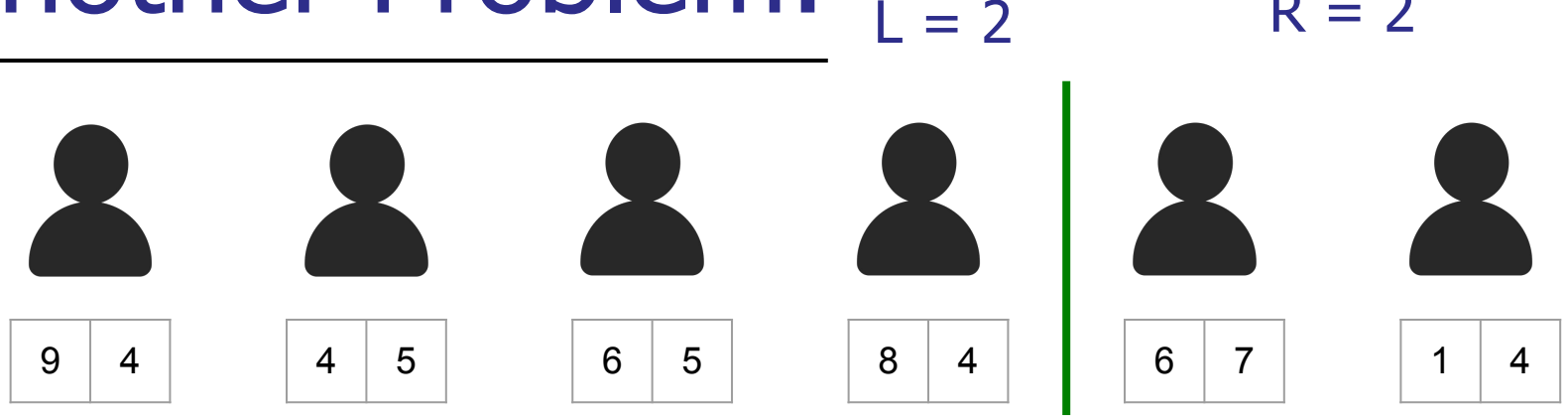


Left tree: [4, 6, 8, 9]

Right tree: [4, 7]

Pay the one time cost of finding L maximum values, and sum it up, call it left_sum.

Another Problem:



Left tree: [4, 6, 8, 9]

Right tree: [4, 7]

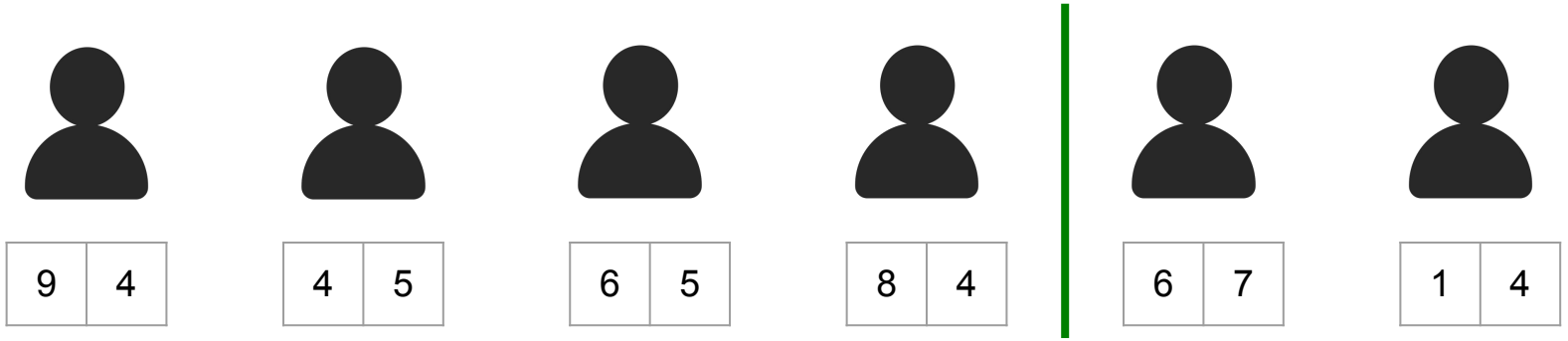
Pay the one time cost of finding L maximum values, and sum it up, call it left_sum.

Pay the one time cost of finding R maximum values, and sum it up, call it right_sum.

Another Problem:

L = 2

R = 2



Left tree: [4, 6, 8, 9]

left_sum = 17

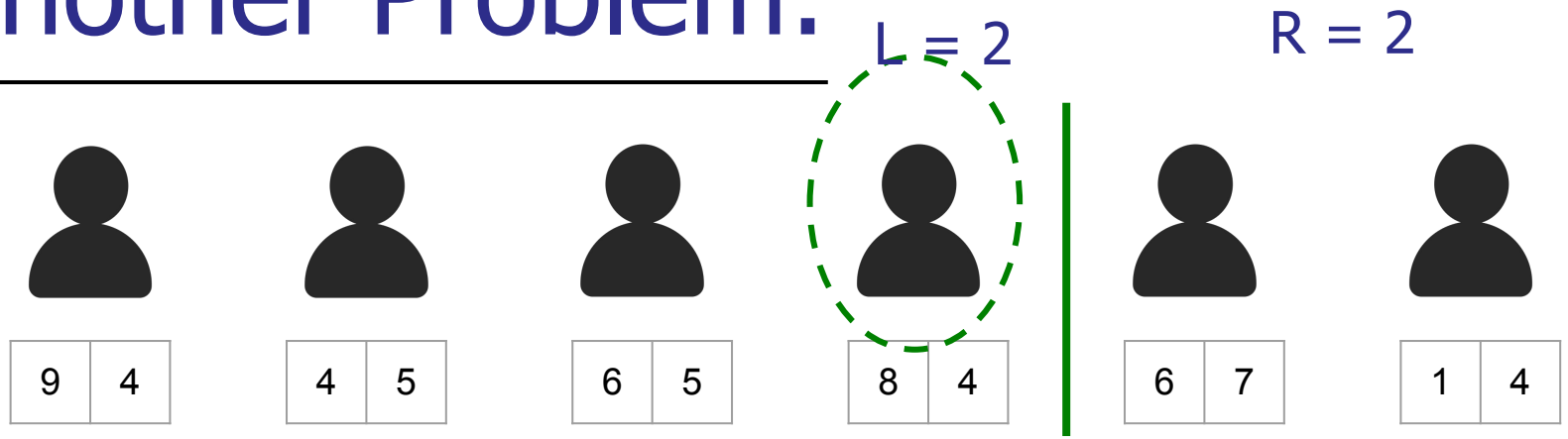
Right tree: [4, 7]

right_sum = 11

Pay the one time cost of finding L maximum values, and sum it up, call it left_sum.

Pay the one time cost of finding R maximum values, and sum it up, call it right_sum.

Another Problem:



Left tree: [4, 6, 8, 9]

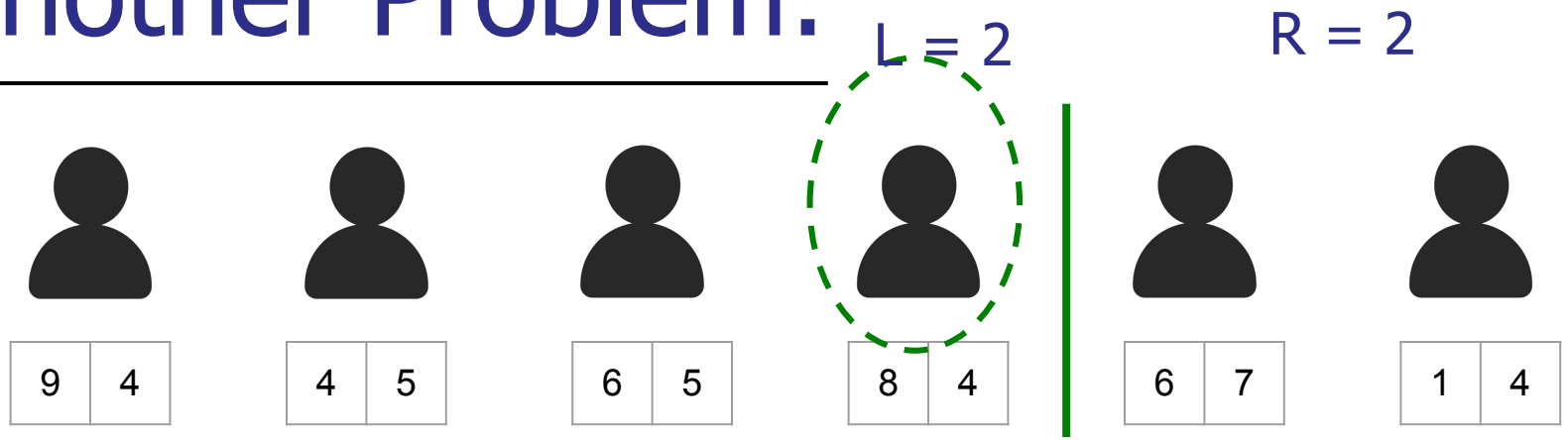
left_sum = 17

Right tree: [4, 7]

right_sum = 11

To move to the line to the left, remove the corresponding person from the left tree, and insert it into the right tree

Another Problem:



Left tree: [4, 6, 8, 9]

left_sum = 17

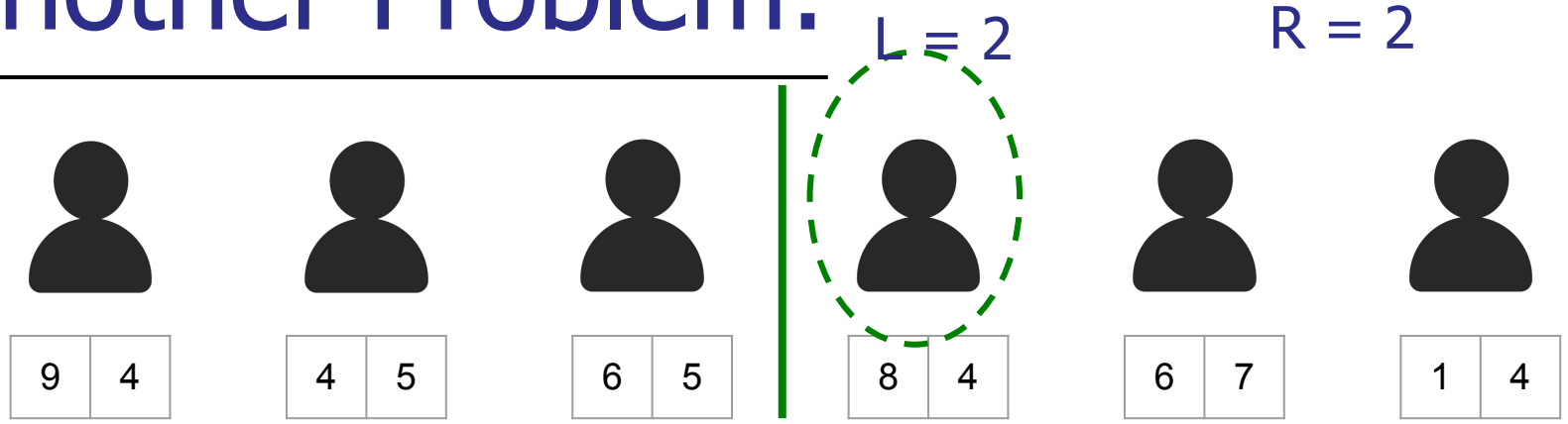
Right tree: [4, 7]

right_sum = 11

To move to the line to the left, remove the corresponding person from the left tree, and insert it into the right tree

How should we update left_sum and right_sum?

Another Problem:



Left tree: [4, 6, 8, 9]

left_sum = 17

Right tree: [4, 7]

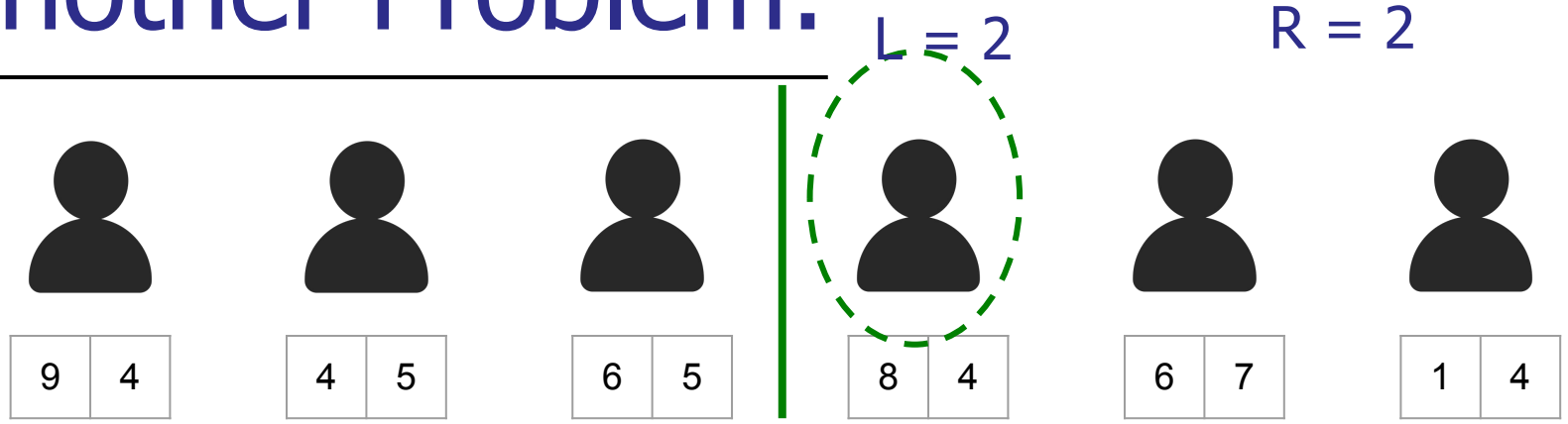
right_sum = 11

To move to the line to the left, remove the corresponding person from the left tree, and insert it into the right tree

How should we update left_sum and right_sum?

e.g. Remove 8 from the left tree, insert 4 in the right tree.

Another Problem:



Left tree: [4, 6, 9]

left_sum = 17

Right tree: [4, 4, 7]

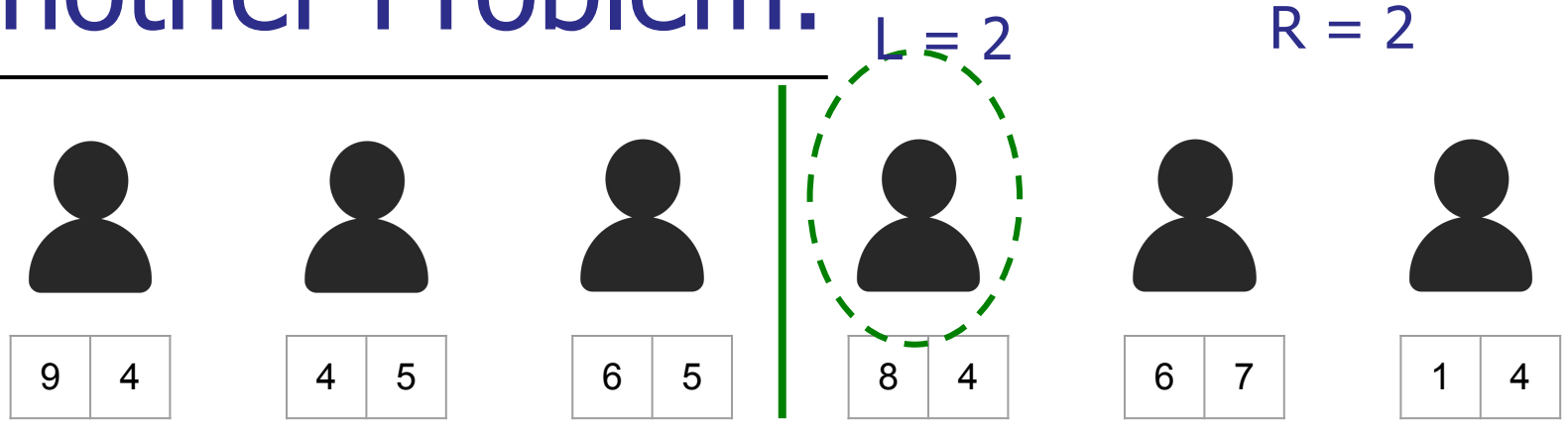
right_sum = 11

To move to the line to the left, remove the corresponding person from the left tree, and insert it into the right tree

How should we update left_sum and right_sum?

e.g. Remove 8 from the left tree, insert 4 in the right tree.

Another Problem:



Left tree: [4, 6, 9]

left_sum = 17

Right tree: [4, 4, 7]

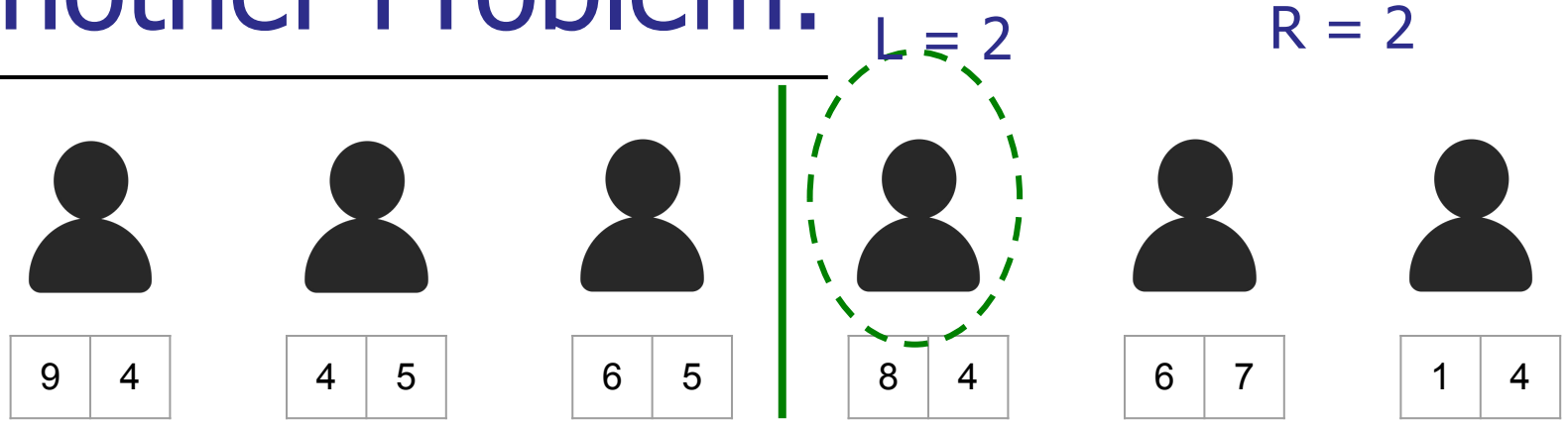
right_sum = 11

To move to the line to the left, remove the corresponding person from the left tree, and insert it into the right tree

How should we update left_sum and right_sum?

e.g. Remove 8 from the left tree, insert 4 in the right tree.

Another Problem:



Left tree: [4, 6, 9]

left_sum = 17

Right tree: [4, 4, 7]

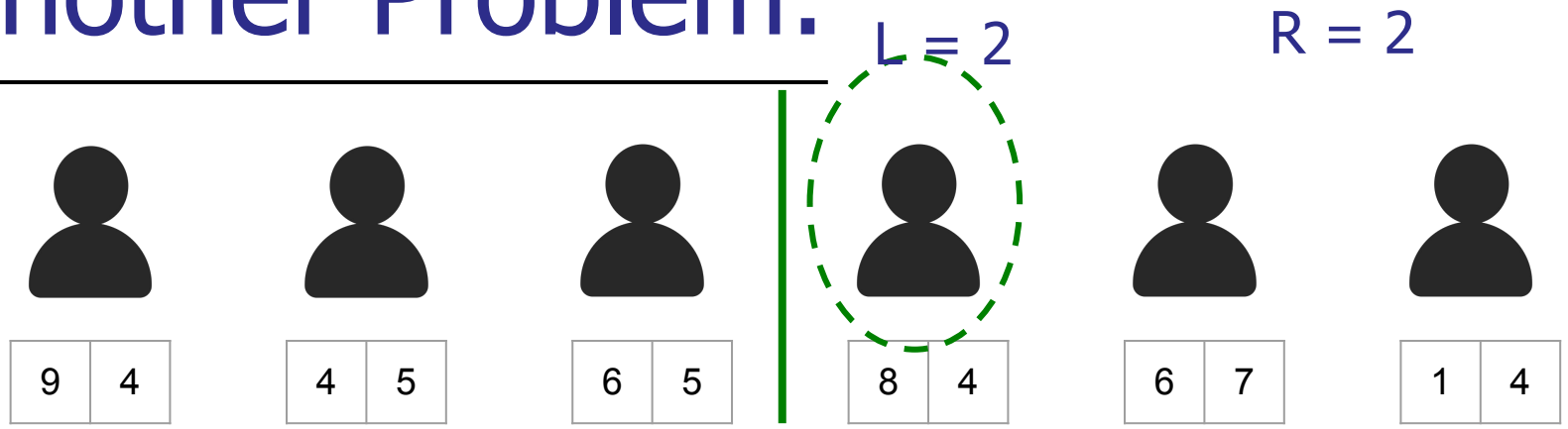
right_sum = 11

How should we update left_sum and right_sum?

e.g. Remove 8 from the left tree, insert 4 in the right tree.

Need to check if the left_sum and right_sum is affected.

Another Problem:



Left tree: [4, 6, 9]

left_sum = 17

Right tree: [4, 4, 7]

right_sum = 11

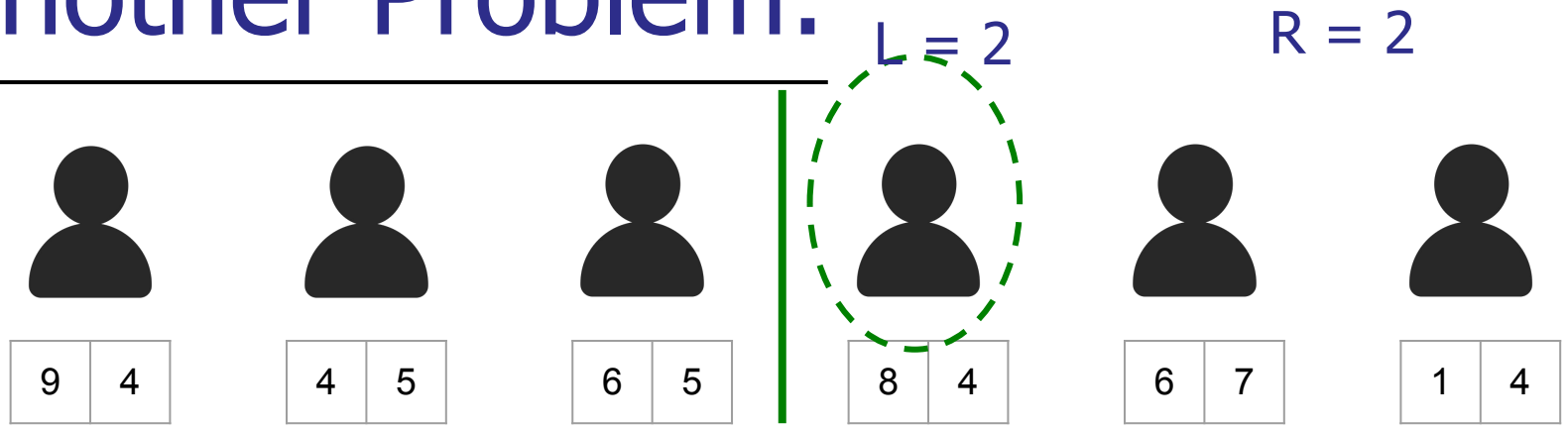
Use the select operation! E.g. look at the left tree, the previous 2nd largest value was 8.

After removal, we know that the second largest value has changed.

To update: add new second largest value, subtract value to be removed.

new value = 17 + 6 - 8 = 15

Another Problem:



Left tree: [4, 6, 9]

left_sum = 15

Right tree: [4, 4, 7]

right_sum = 11

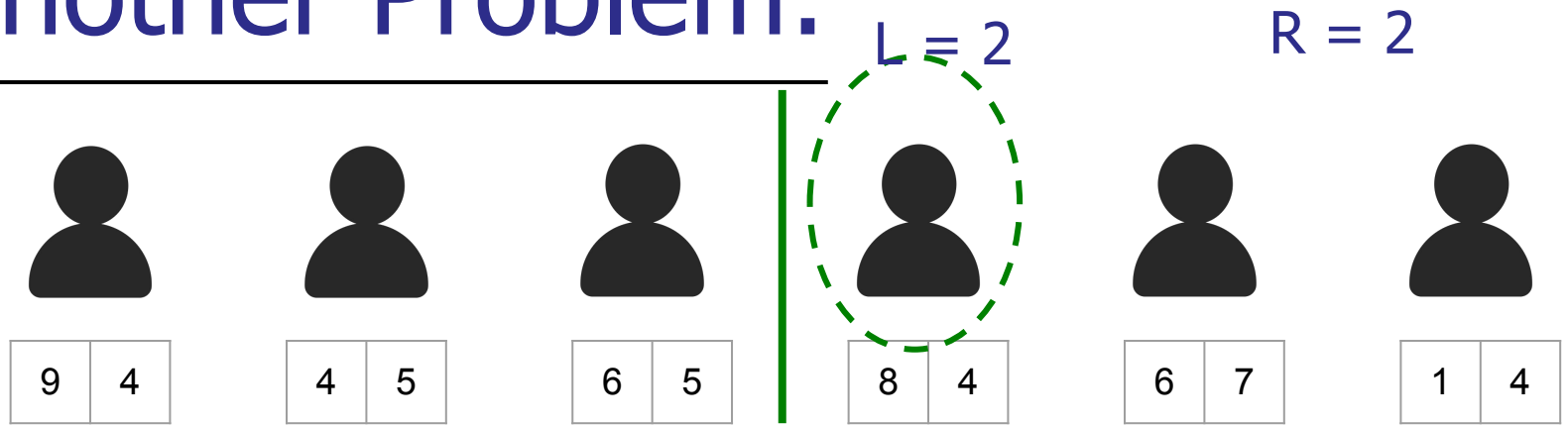
Use the select operation! E.g. look at the left tree, the previous 2nd largest value was 8.

After removal, we know that the second largest value has changed.

To update: add new second largest value, subtract value to be removed.

new value = 17 + 6 - 8 = 15

Another Problem:



Left tree: [4, 6, 9]

left_sum = 15

Right tree: [4, 4, 7]

right_sum = 11

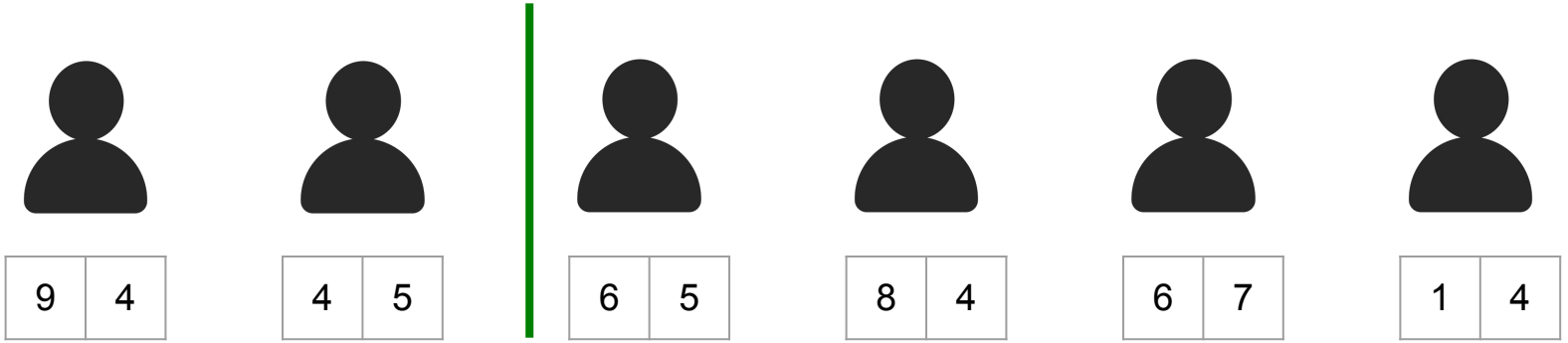
Use the select operation! What about the right tree?

If the 2nd largest value has increased after insertion, we update right sum. Right tree went from [4, 7] to [4, 4, 7]. No change!

Another Problem:

L = 2

R = 2



Left tree: [4, 6, 9]

left_sum = 15

Right tree: [4, 4, 7]

right_sum = 11

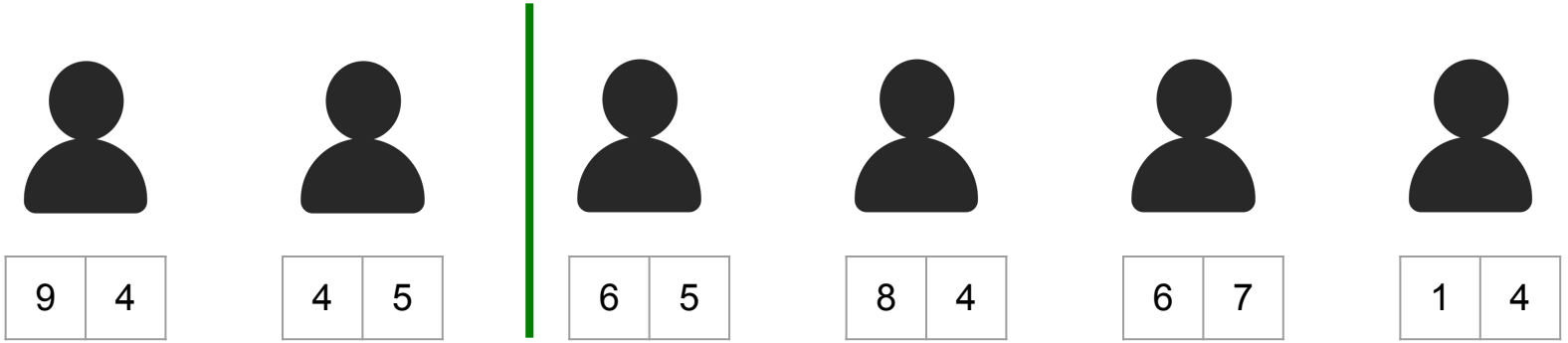
This time: Want to remove 6 from left tree, insert 5 into right tree.

We know this means left_sum will change because the 2nd largest value changes from 6 to 4.

Another Problem:

L = 2

R = 2



Left tree: [4, 6, 9]

left_sum = 15

Right tree: [4, 4, 7]

right_sum = 11

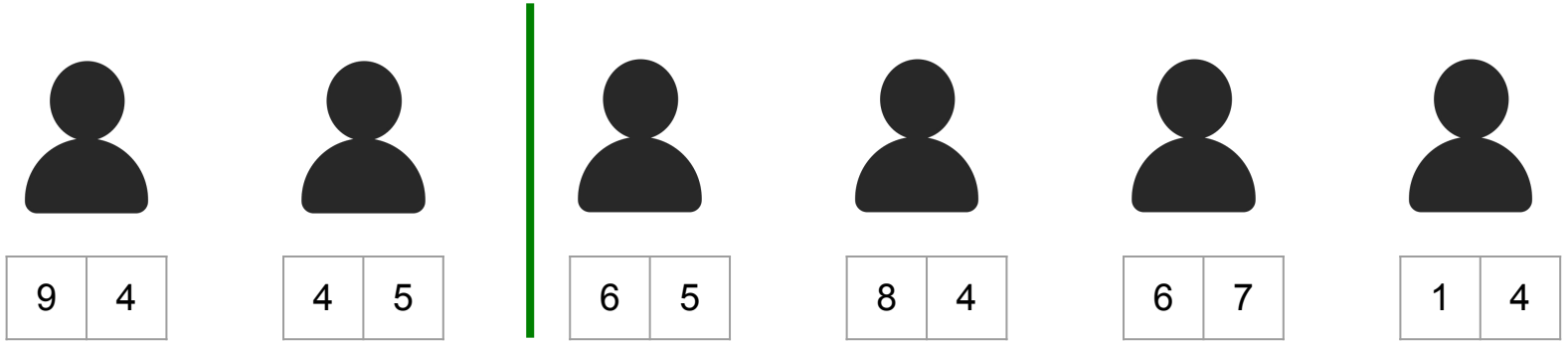
This time: Want to remove 6 from left tree, insert 5 into right tree.

We know this means left_sum will change because the 2nd largest value changes from 6 to 4.

We also know right_sum will change from 4 to 5 after insertion.

Another Problem:

L = 2 R = 2



Left tree: [4, 9]

left_sum = 15

Right tree: [4, 4, 5, 7]

right_sum = 11

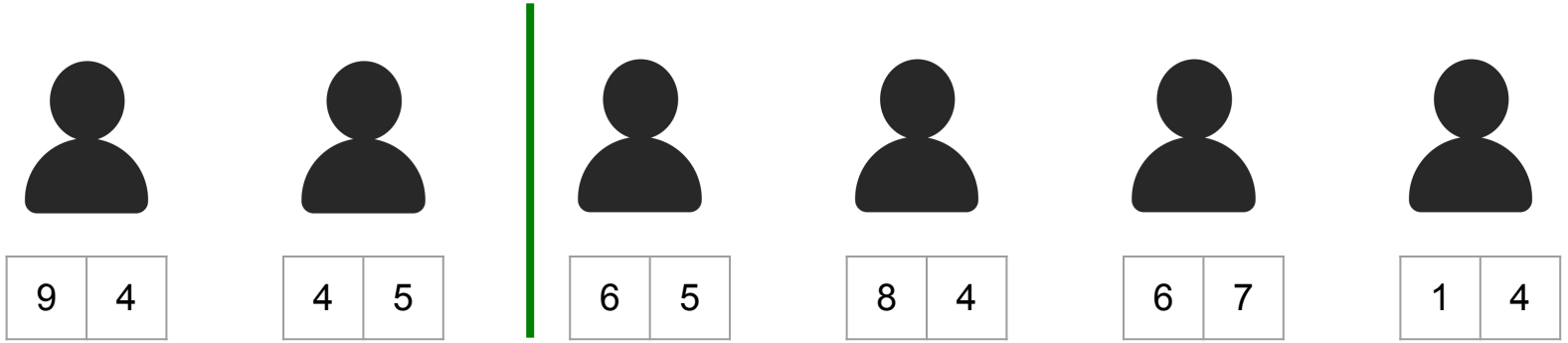
This time: Want to remove 6 from left tree, insert 5 into right tree.

We know this means left_sum will change because the 2nd largest value changes from 6 to 4.

We also know right_sum will change from 4 to 5 after insertion.

Another Problem:

L = 2 R = 2



Left tree: [4, 9]

left_sum = 15

Right tree: [4, 4, 5, 7]

right_sum = 11

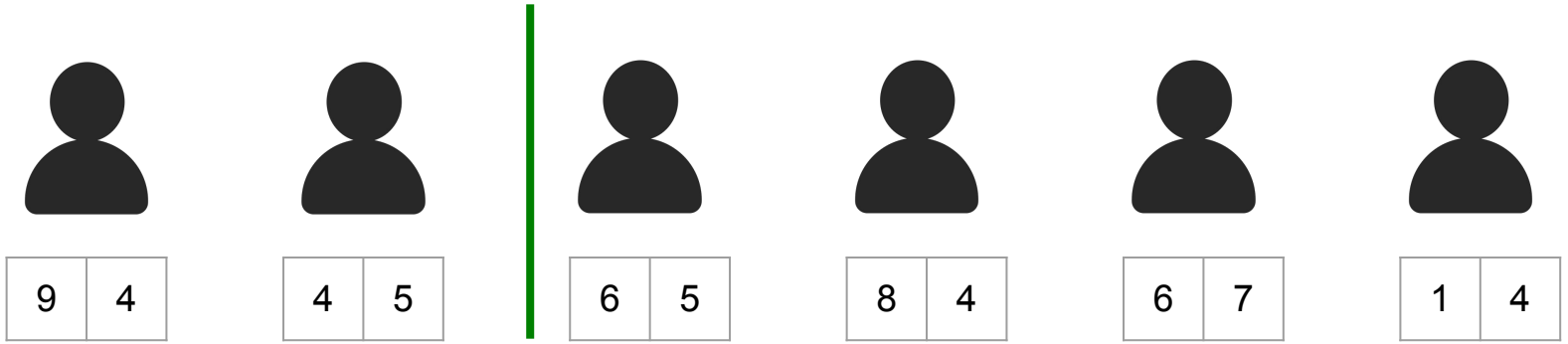
We know this means left_sum will change because the 2nd largest value changes from 6 to 4. We also know right_sum will change from 4 to 5 after insertion.

New left_sum = 15 - 6 + 4 = 13

New right_sum = 11 - 4 + 5 = 12

Another Problem:

L = 2 R = 2



Left tree: [4, 9]

left_sum = 13

Right tree: [4, 4, 5, 7]

right_sum = 12

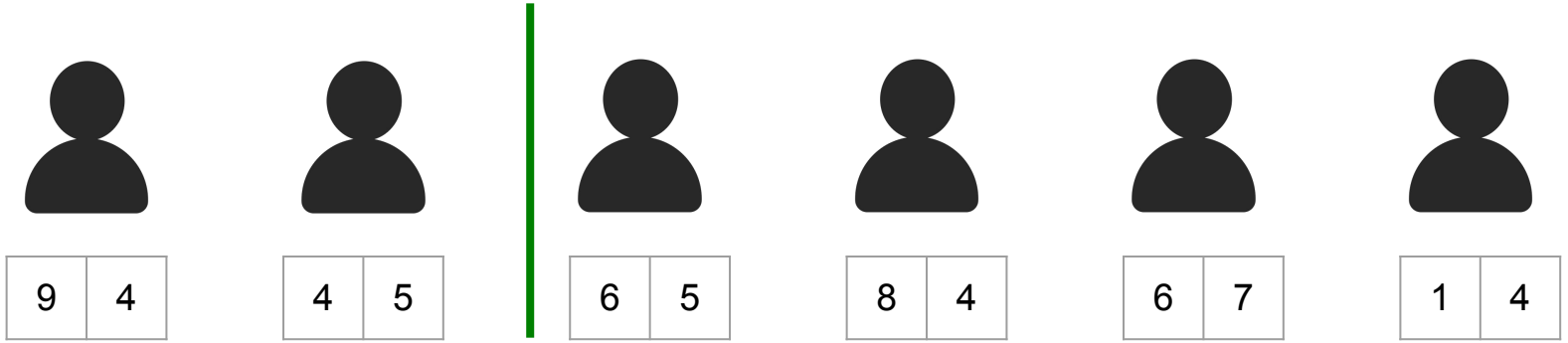
We know this means left_sum will change because the 2nd largest value changes from 6 to 4. We also know right_sum will change from 4 to 5 after insertion.

New left_sum = 15 - 6 + 4 = 13

New right_sum = 11 - 4 + 5 = 12

Another Problem:

L = 2 R = 2



Left tree: [4, 9]

left_sum = 13

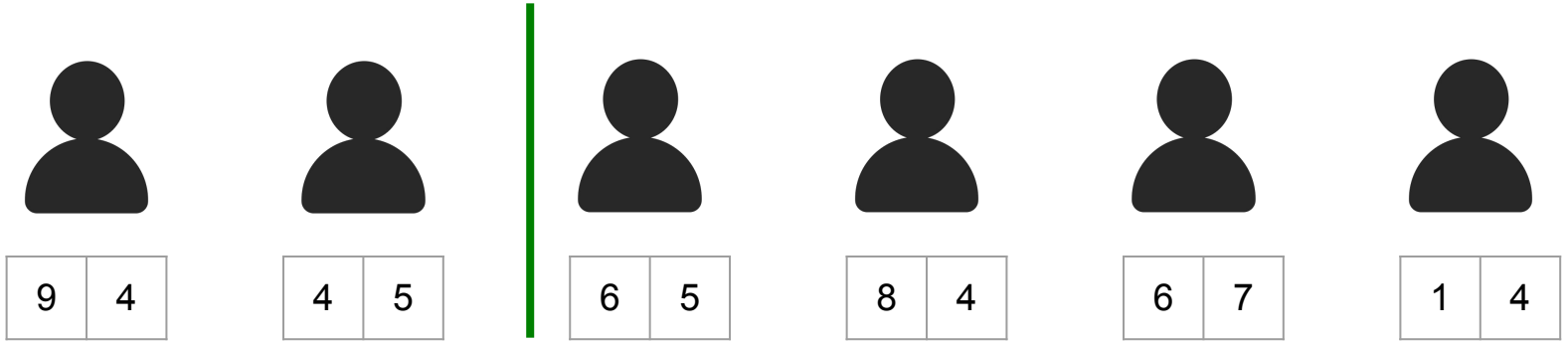
Right tree: [4, 4, 5, 7]

right_sum = 12

Final piece: If we tracked the maximum possible values of 13 + 12, then we are done!

Another Problem:

L = 2 R = 2



Left tree: [4, 9]

left_sum = 13

Right tree: [4, 4, 5, 7]

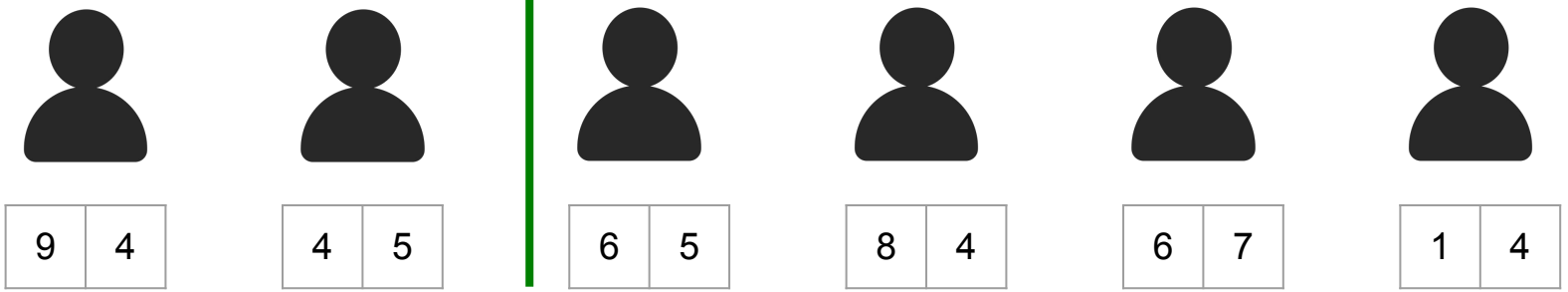
right_sum = 12

Final piece: If we tracked the maximum possible values of 13 + 12, then we are done!

Total time complexity: O(n log n)

Another Problem:

L = 2 R = 2



Left tree: [4, 9]

left_sum = 13

Right tree: [4, 4, 5, 7]

right_sum = 12

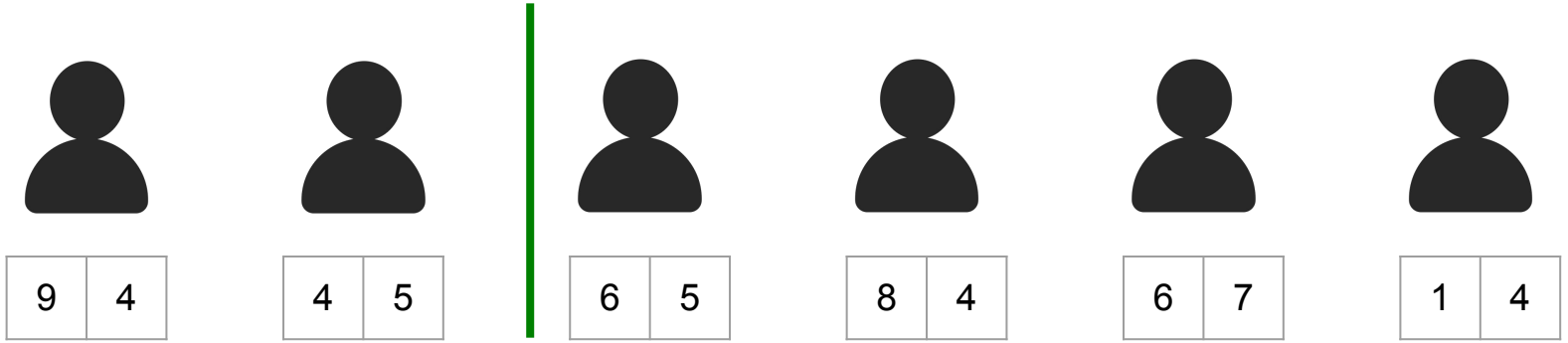
Final piece: If we tracked the maximum possible values of 13 + 12, then we are done!

Total time complexity: O(n log n)

Spend some time thinking about why that is.

Another Problem:

L = 2 R = 2



Left tree: [4, 9]

left_sum = 13

Right tree: [4, 4, 5, 7]

right_sum = 12

Challenge: Think about the details and write the code/pseudocode in detail.

E.g.

1. What are the exact value to select at each iteration?
2. What were the keys?
3. Did we need to map to specific values?

Another Problem:

Clarification 1:

We know we have to update left_sum if the value we want to remove is one of the L largest ones from the left tree.

What operation should we use?

If the value we want to remove is indeed one of the L largest ones, how do we update left_sum?

Another Problem:

Clarification: 1

We know we have to update left_sum if the value we want to remove is one of the L largest ones from the left tree.

What operation should we use?

rank!

If the value we want to remove is indeed one of the L largest ones, how do we update left_sum?

select the L^{th} largest value! Why?

Another Problem:

Clarification: 1

E.g. we want the 3 largest values from some left tree:

Before removal: [1, 3, 5, 7, 8]

Say we had to remove value 7 from the left tree.

Another Problem:

Clarification: 1

E.g. we want the 3 largest values from some left tree:

Before removal: [1, 3, 5, 7, 8]

Say we had to remove value 7 from the left tree.

After removal: [1, 3, 5, 8]

Notice that the only change is that the new second largest value is 3. The sum changes from 5 + 7 + 8 to 3 + 5 + 8

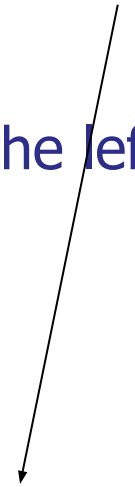
Another Problem:

Clarification: 1

E.g. we want the 3 largest values from some left tree:

Before removal: [1, 3, 5, 7, 8]

subtract value to be removed
add L^{th} largest value



Say we had to remove value 7 from the left tree.

After removal: [1, 3, 5, 8]

Notice that the only change is that the new second largest value is 3. The sum changes from 5 + 7 + 8 to 3 + 5 + 8

Another Problem:

Clarification 2:

Why is it O(n log n) time?

Another Problem:

Clarification 2:

Why is it O(n log n) time?

We have O(n) possible places to draw the line, each time we move the line to the left, it takes O(log n) time to update both trees and sums.

Todays Plan

Data structure design

- More Augmentation on Balanced Trees

Tries

- How to handle text?

Problem Solving Using Trees

- Thinking with Trees

Wednesday:

Hashing!