Before the Break

- Dictionaries (Abstract Data Type)
- Binary search trees
- Balanced search trees
- AVL trees

Dynamic Data Structures

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

The Elephant in the Room

- “Why do we need to learn how an AVL tree works?”
  - Just use a STL, right?
- Learn how to think like a computer scientist.
- 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…

Today

- Two examples of augmenting balanced BSTs
  - Order Statistics
  - Orthogonal Range Searching

Augmented Reality

Basic methodology of Augmented Data Structures

- Choose underlying data structure
  - (tree, hash table, linked list, stack, etc.)
- Determine additional info needed.
- Modify data structure to maintain additional info when the structure changes.
  - (subject to insert/delete/etc.)
- Develop new operations.

Today

- Two examples of augmenting balanced BSTs
  - Order Statistics
  - Orthogonal Range Searching

Order Statistics

- Input:
  - A set of integers
- Output: `select(k)`
  - return the kth item in the set

Sort: $O(n \log n)$

QuickSelect: $O(n)$

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

`select(4)`

Order Statistics

- Solution 1:
  - Preprocess: sort --- $O(n \log n)$
  - Select: $O(1)$

- Solution 2:
  - Preprocess: nothing --- $O(1)$
  - QuickSelect: $O(n)$

- Trade-off: how many items to select?

Dynamic Order Statistics

- Implement a data structure that supports:
  - `insert(int key)`
  - `delete(int key)`
- and also:
  - `select(int k)`

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

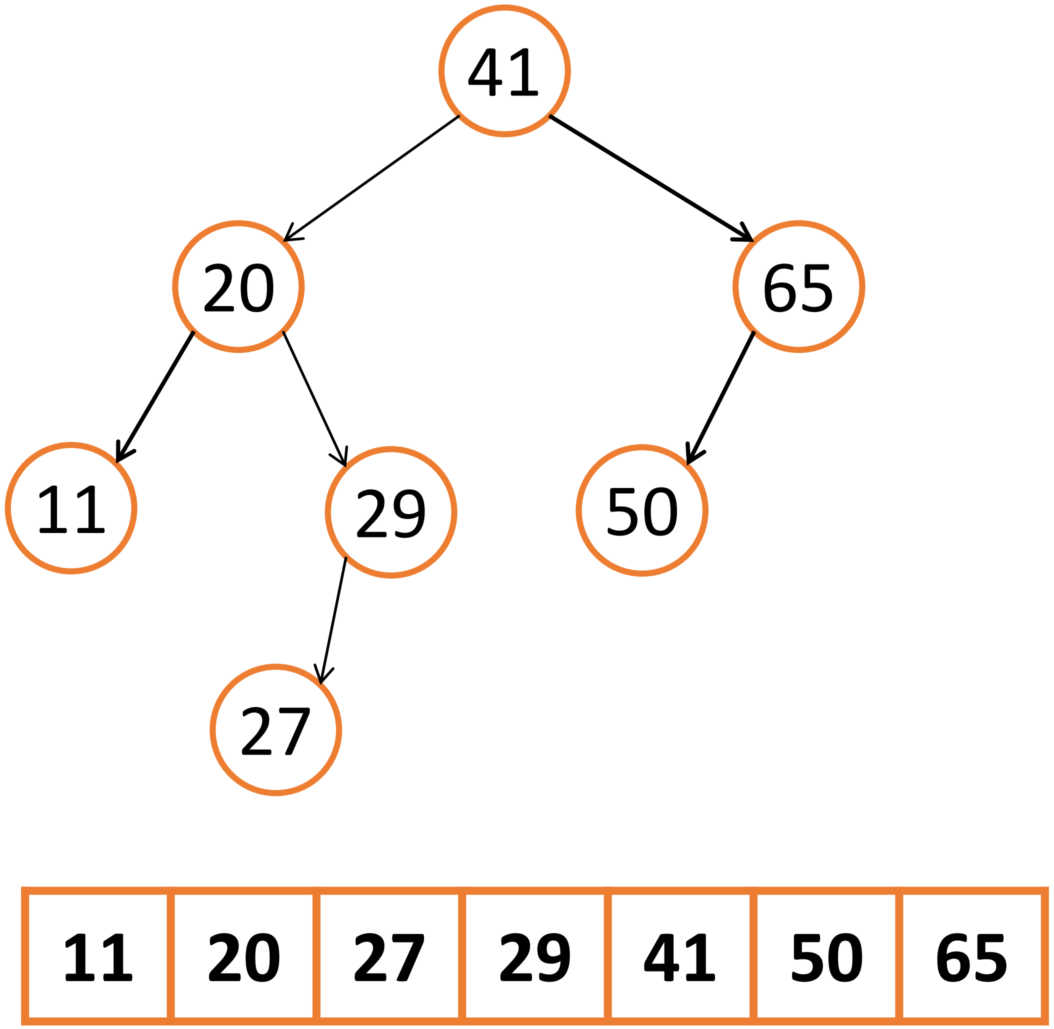
`select(4)`

Dynamic Order Statistics

- Solution 1:
  - Preprocess: sort --- $O(n \log n)$
  - Select: $O(1)$
  - Insert: $O(n)$
- Solution 2:
  - Preprocess: nothing --- $O(1)$
  - QuickSelect: $O(n)$
  - Insert: $O(1)$
- Trade-off: how many items to select/insert?

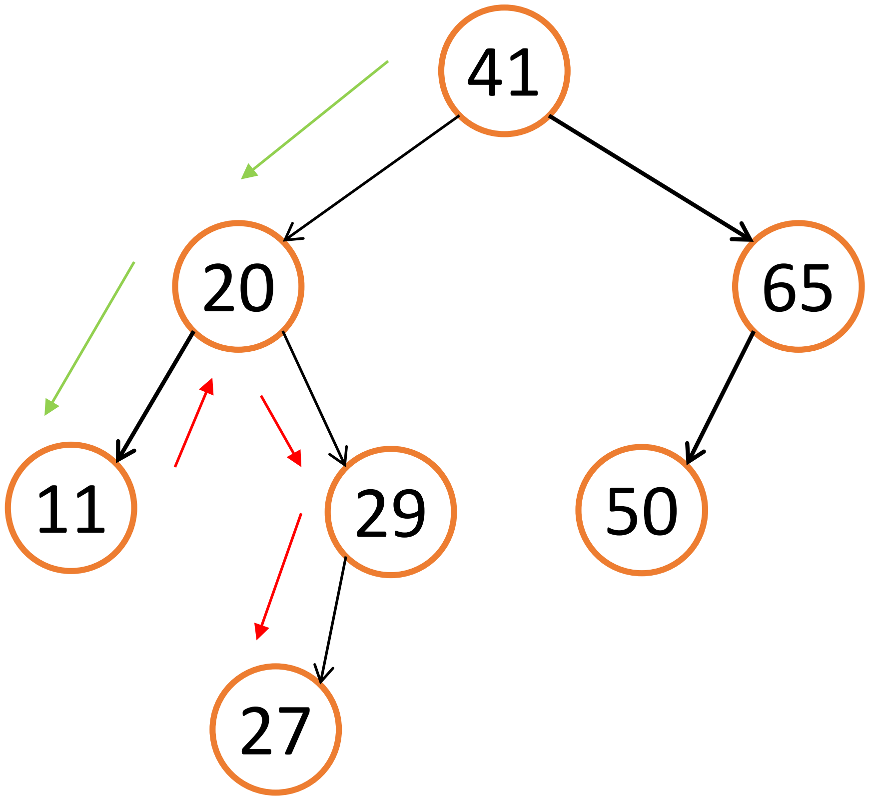
Dynamic Order Statistics

- Idea: use a (balanced) tree
- How to start?
- E.g. `select(3)`
  - From the root 41, go left or right?



Dynamic Order Statistics

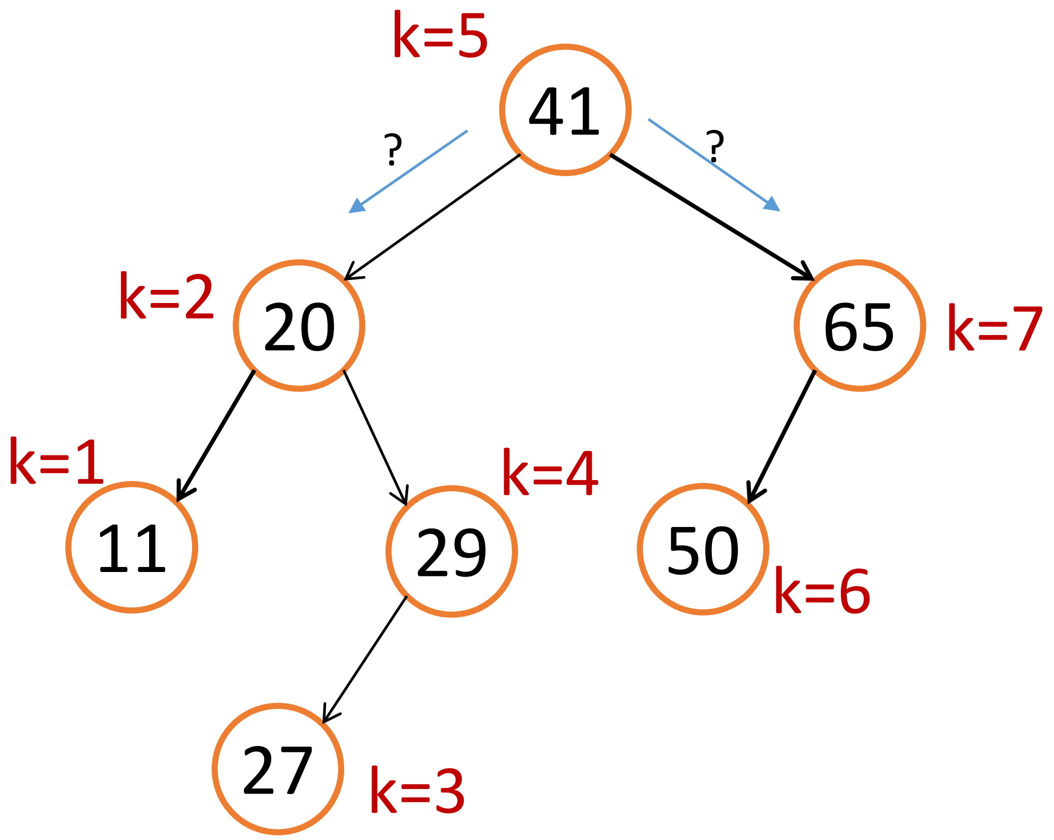
- Simple Solution
  - Search min
  - in-order traversal
- Time complexity: $O(\log n) + O(k)$
- What if I want the median?



11	**20**	**27**	**29**	**41**	**50**	**65**

Dynamic Order Statistics

- Idea: Augment!
- What extra information should we store?
- Store the rank in every node!
- Then how do we search?
  - e.g. if `select(3)`, from 41, go left or right?

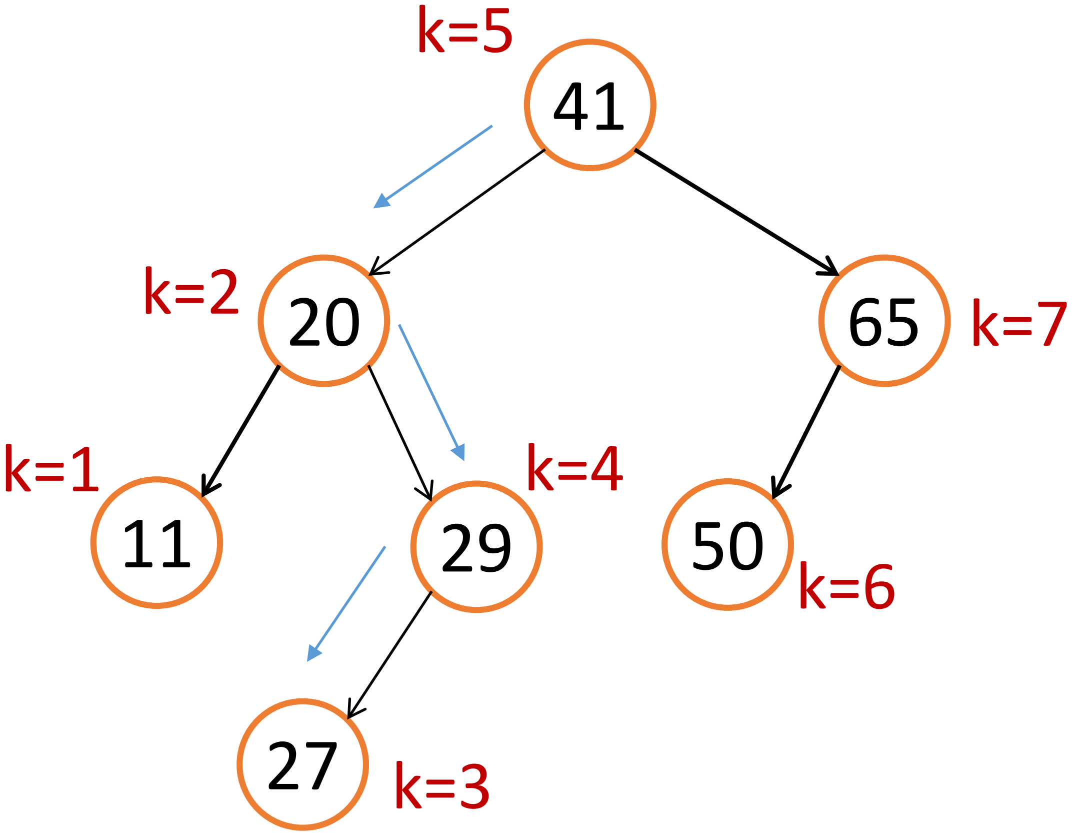


11	**20**	**27**	**29**	**41**	**50**	**65**

Dynamic Order Statistics

- `select(3)`
- If current rank > 3
  - Go left
  - else go right

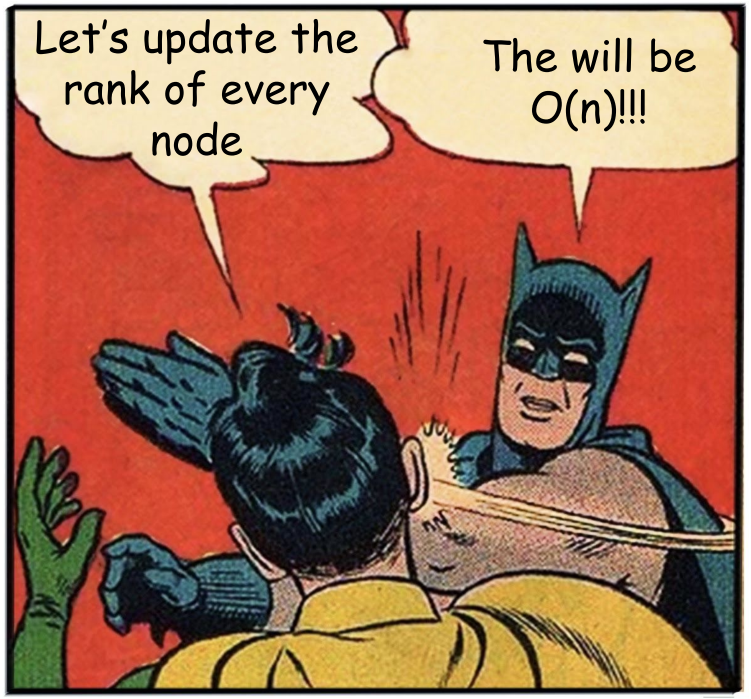
- What will be the problem?

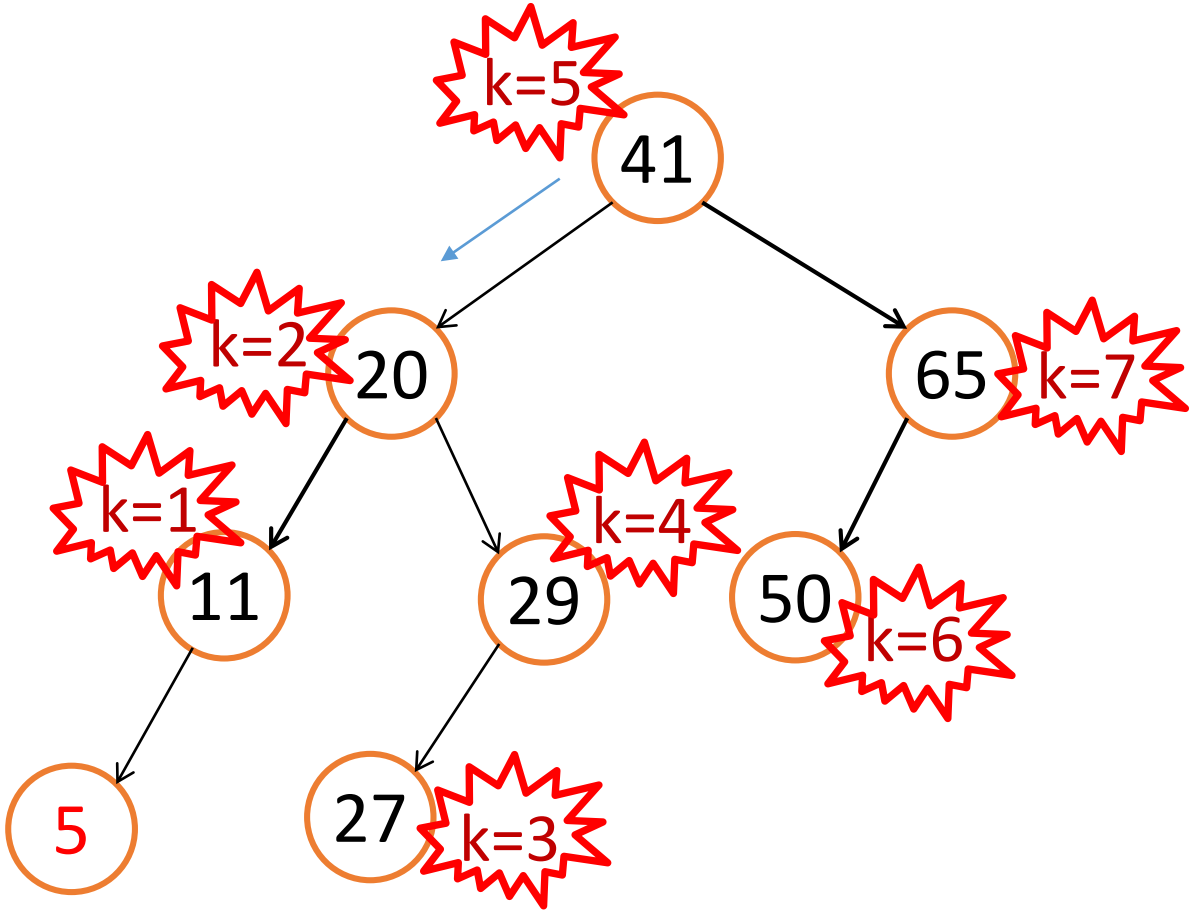


11	**20**	**27**	**29**	**41**	**50**	**65**

Dynamic Order Statistics

- `insert(5)`



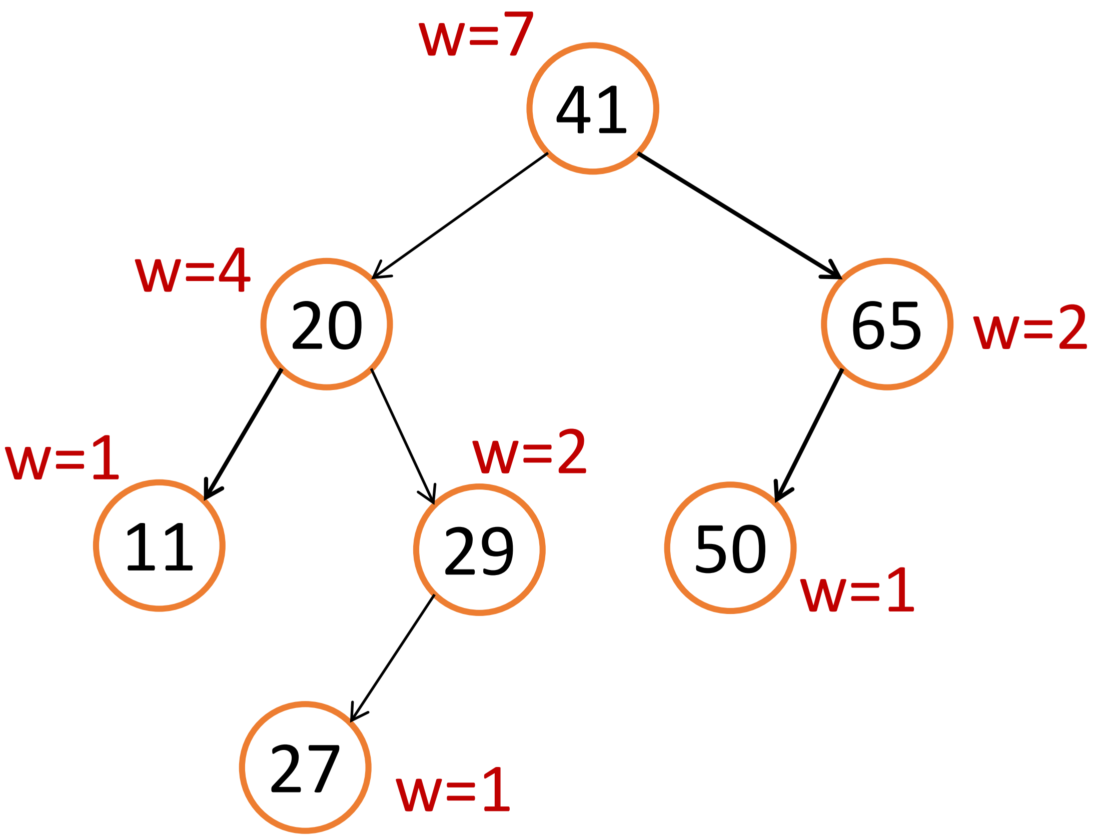


11	**20**	**27**	**29**	**41**	**50**	**65**

- The same will go for deletion

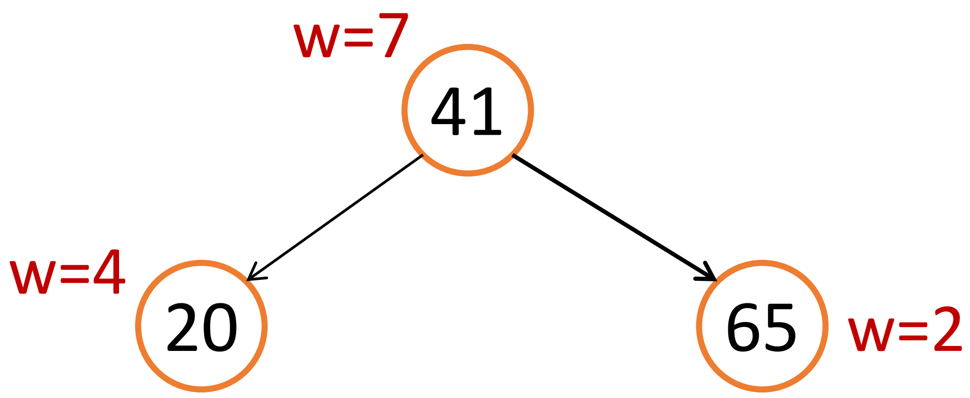
Dynamic Order Statistics

- 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`



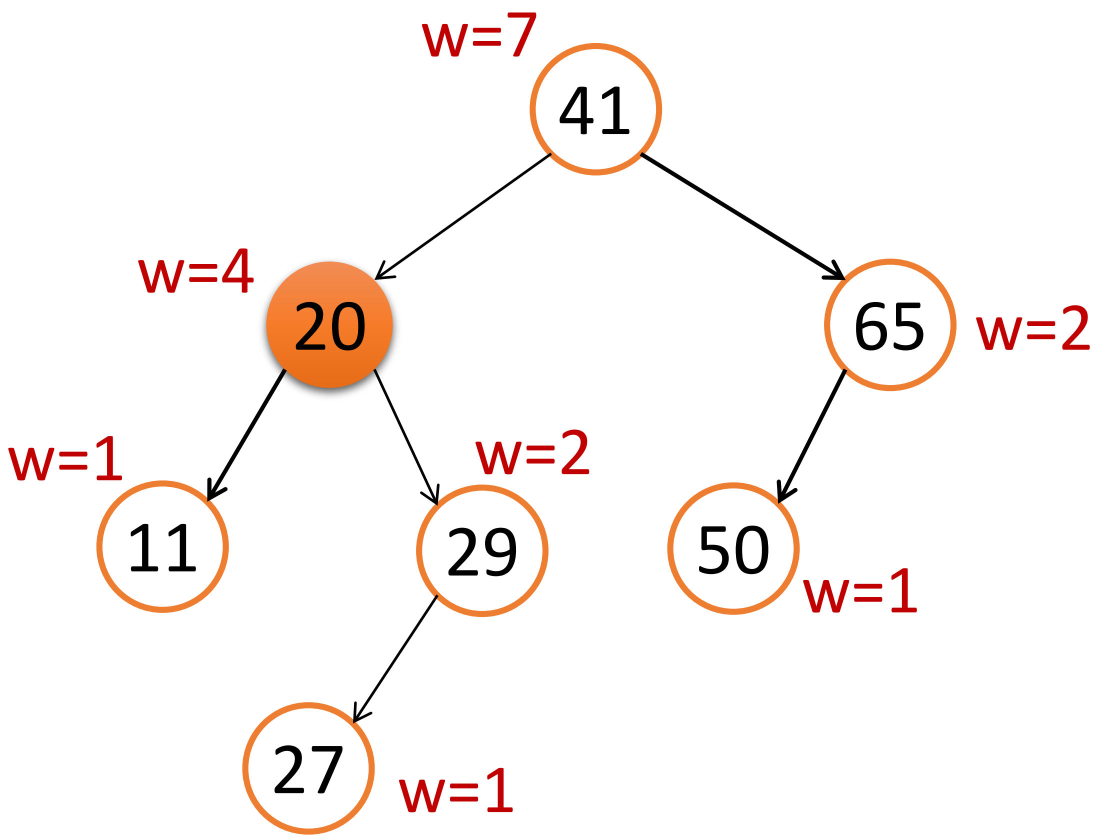
Dynamic Order Statistics

- `select(3)`
- Go left or right?
- We need the children to help
- Actually only your *left* child
  - How do you know the rank of the node by its left child?
- "rank in subtree" = `left.weight + 1`
- Left child has 4 nodes, the 3rd node must be in the left subtree



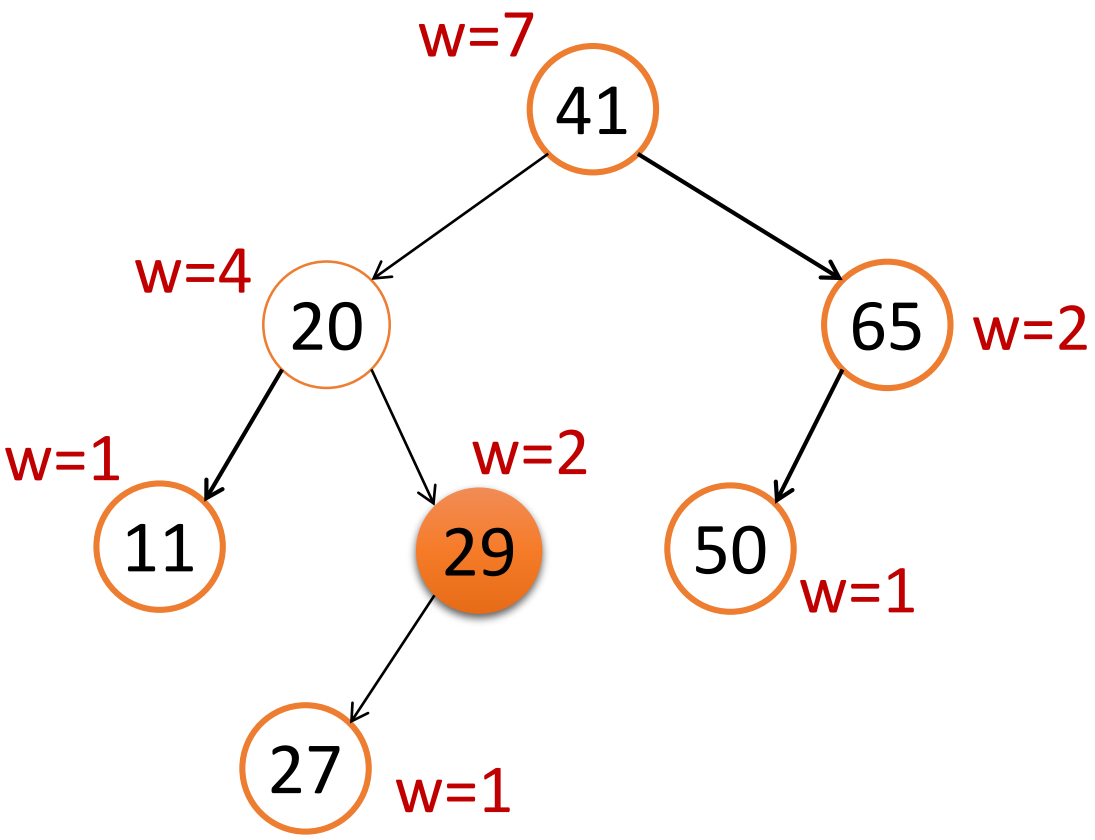
Dynamic Order Statistics

- `Select(3)`
- Now we are at the node of 20
- The left child of 20 has weight 1
- Then 20 must be rank 2
- The 3rd item must be in the right subtree of 20



Dynamic Order Statistics

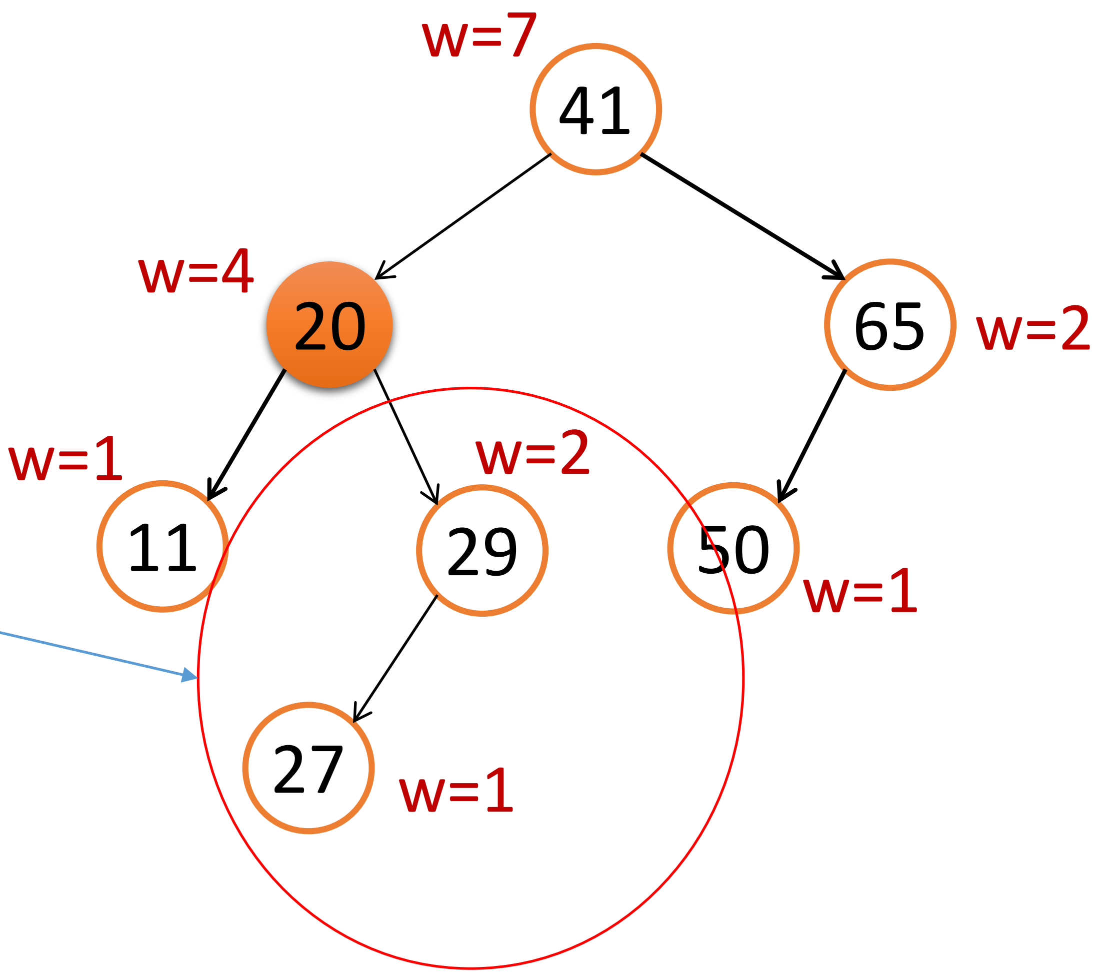
- `select(3)`
- Wait, can I do `select(3)` at 29?
- Rewind back to the previous step



Dynamic Order Statistics

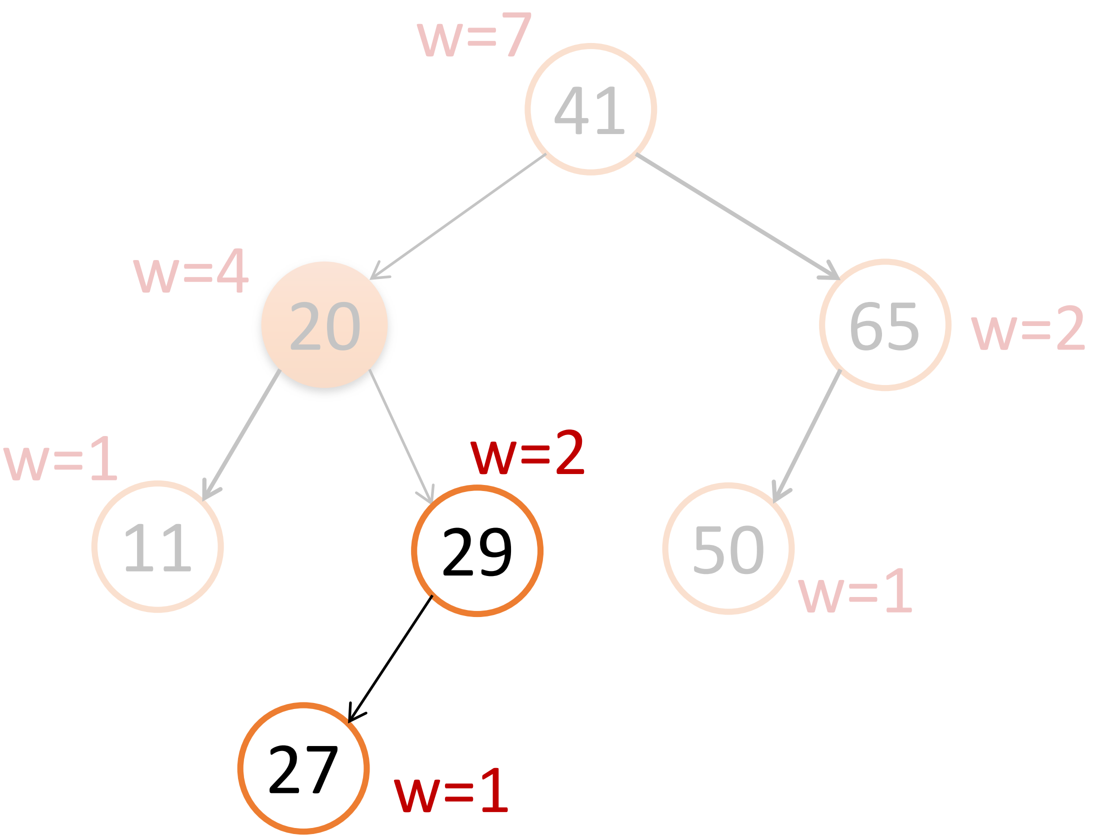
- `Select(3)`
- When we go to the right child of 20, we skip all the left subtree of 20 and the node 20 itself
- So we should search for

  rank 3 – 1 - 1 = rank 1
- in the right subtree of 20
- And it will become.....



Dynamic Order Statistics

- `Select(1)` at the node 29
  - Instead of `Select(3)`
- Then go left and reach 27

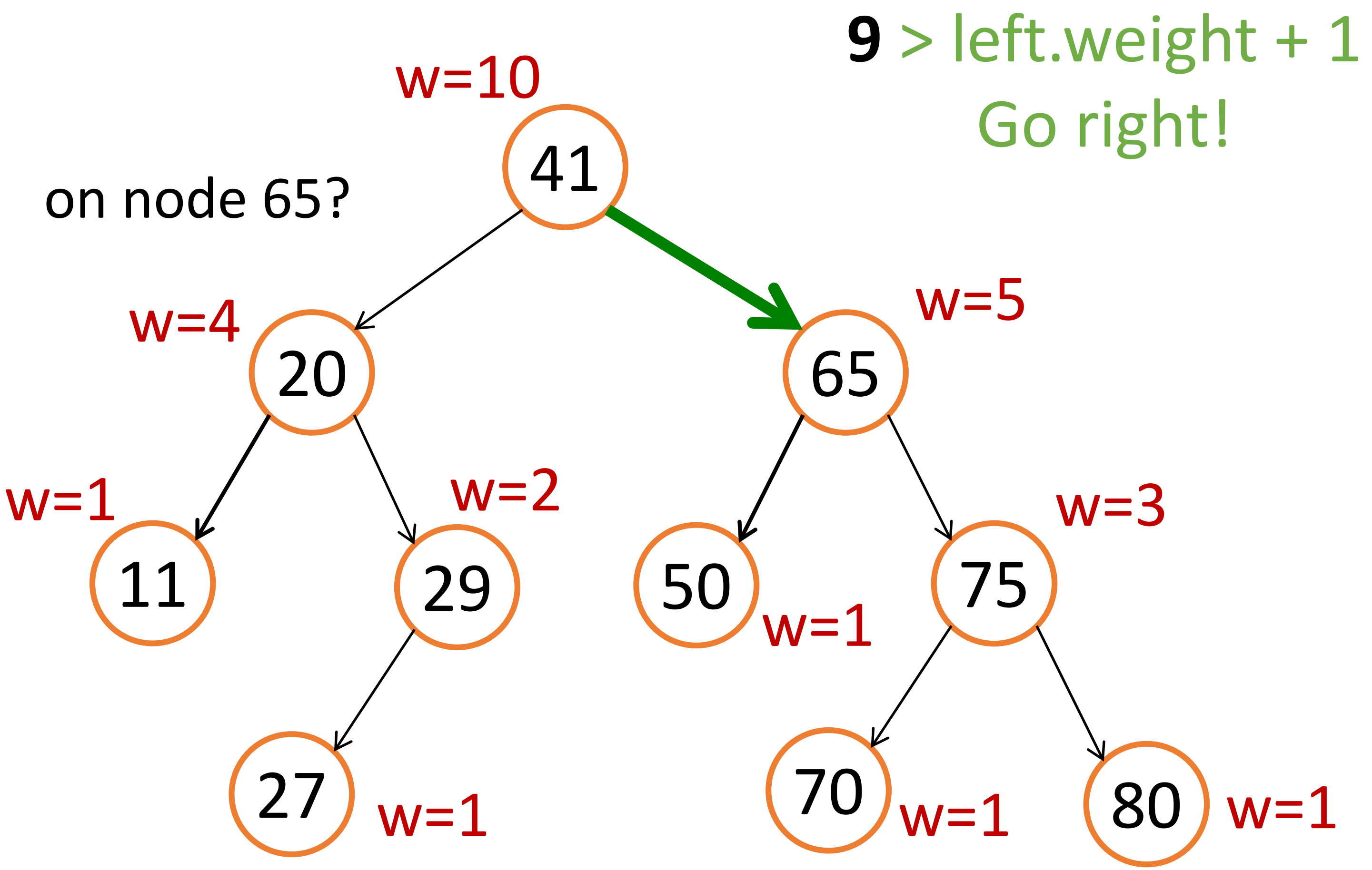


Dynamic Order Statistics

```
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(9)`
- Then?
  - Recursively call `select(9)` on node 65?
  - No

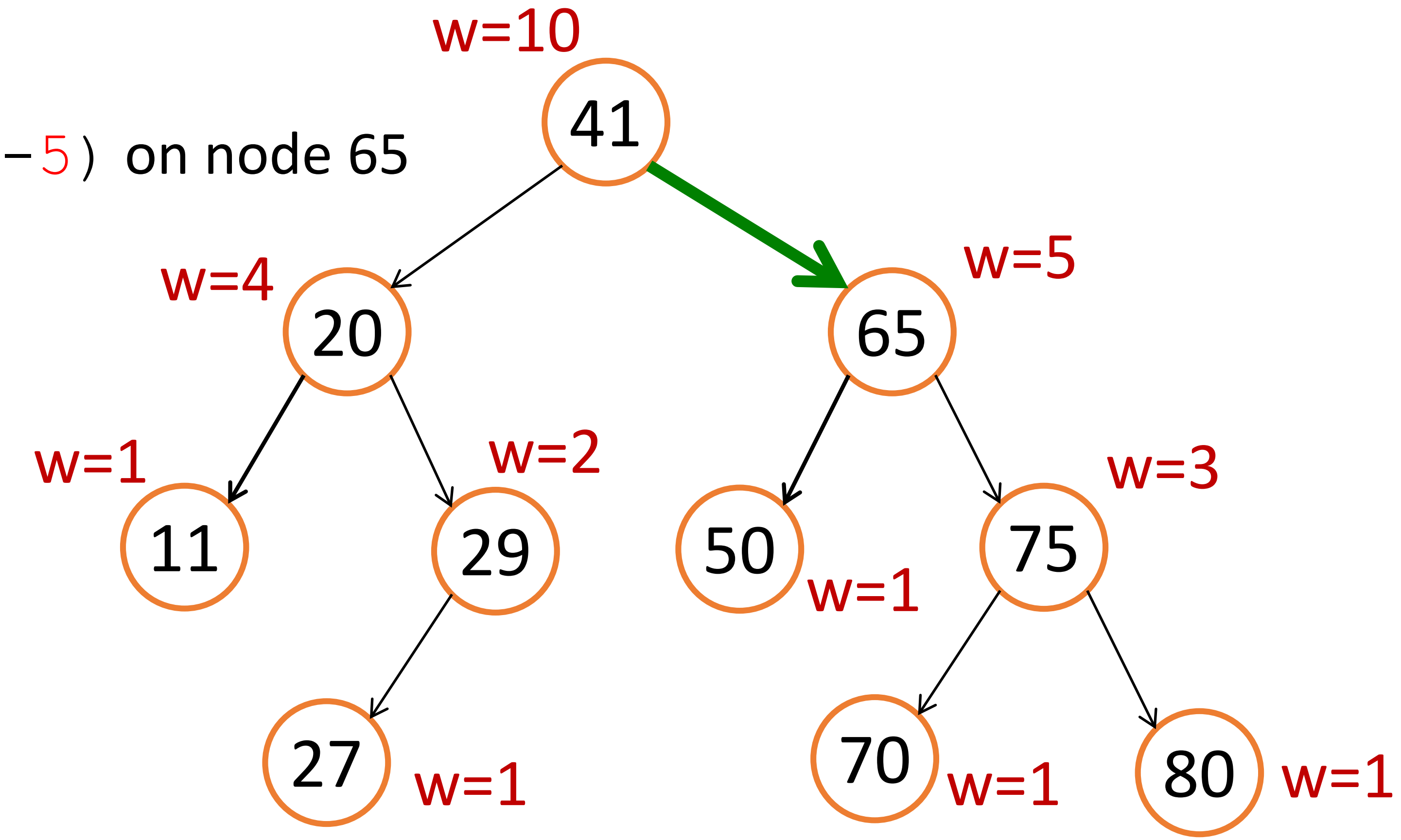


Dynamic Order Statistics

```
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);
```

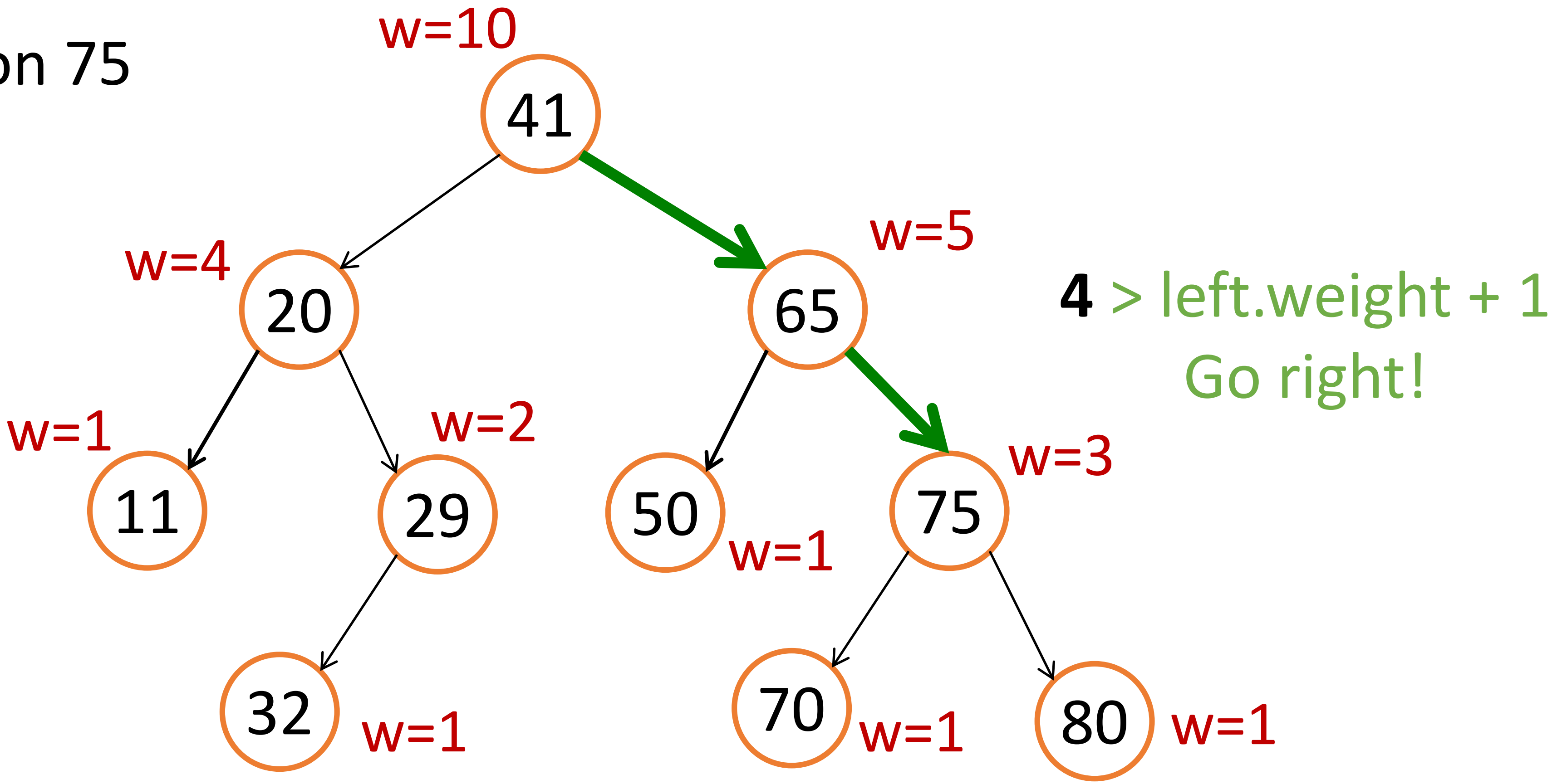
Dynamic Order Statistics

- `Select(9)`
- Then?
  - Recursively call `select(9-5)` on node 65



Dynamic Order Statistics

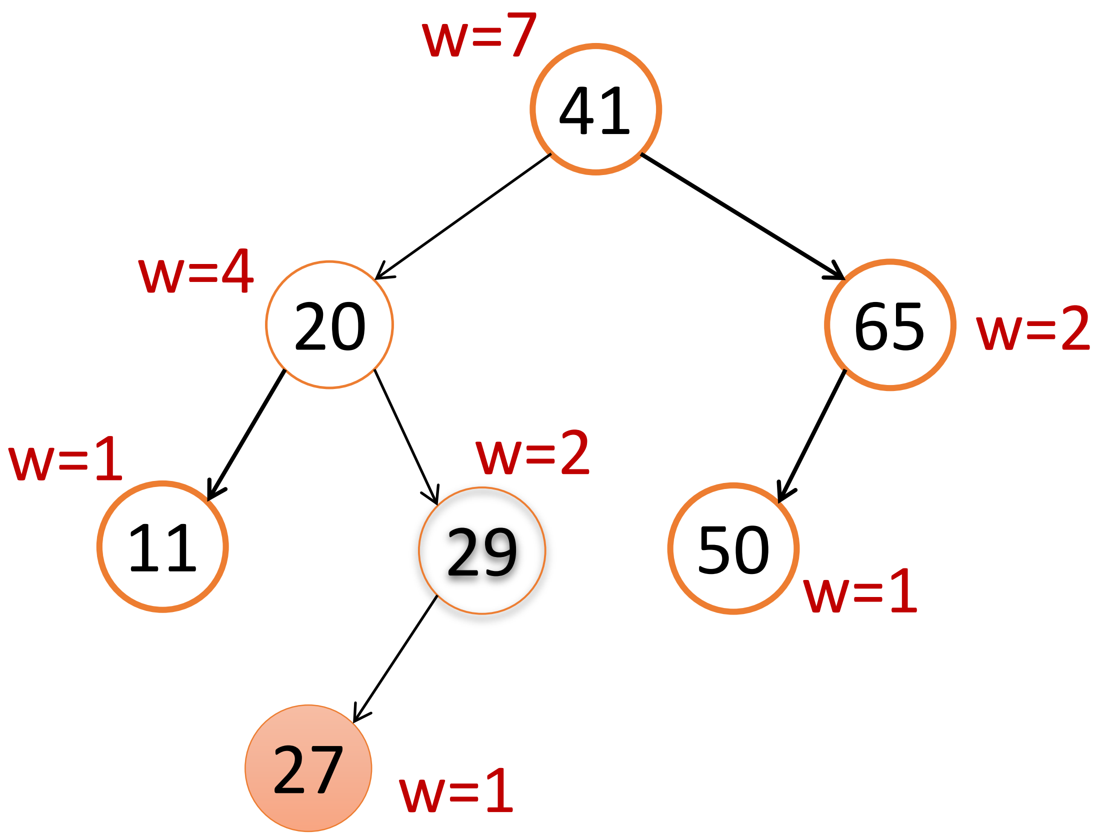
- `select(9-5 = 4)` **on 65**
- `select(4-2)` **on 75**



Dynamic Order Statistics

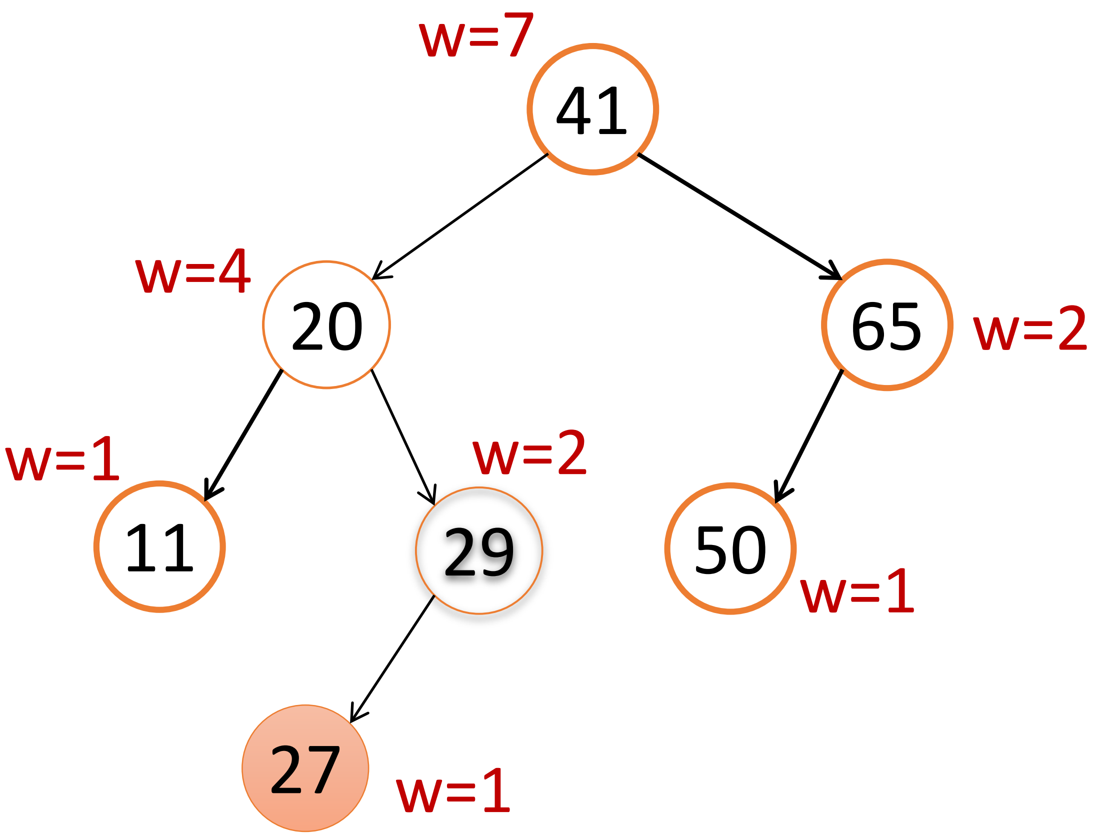
- `select(k)`
  - Find the item with rank k

- `rank(v)`
  - Find the rank of v
  - E.g. what is the rank of 27?



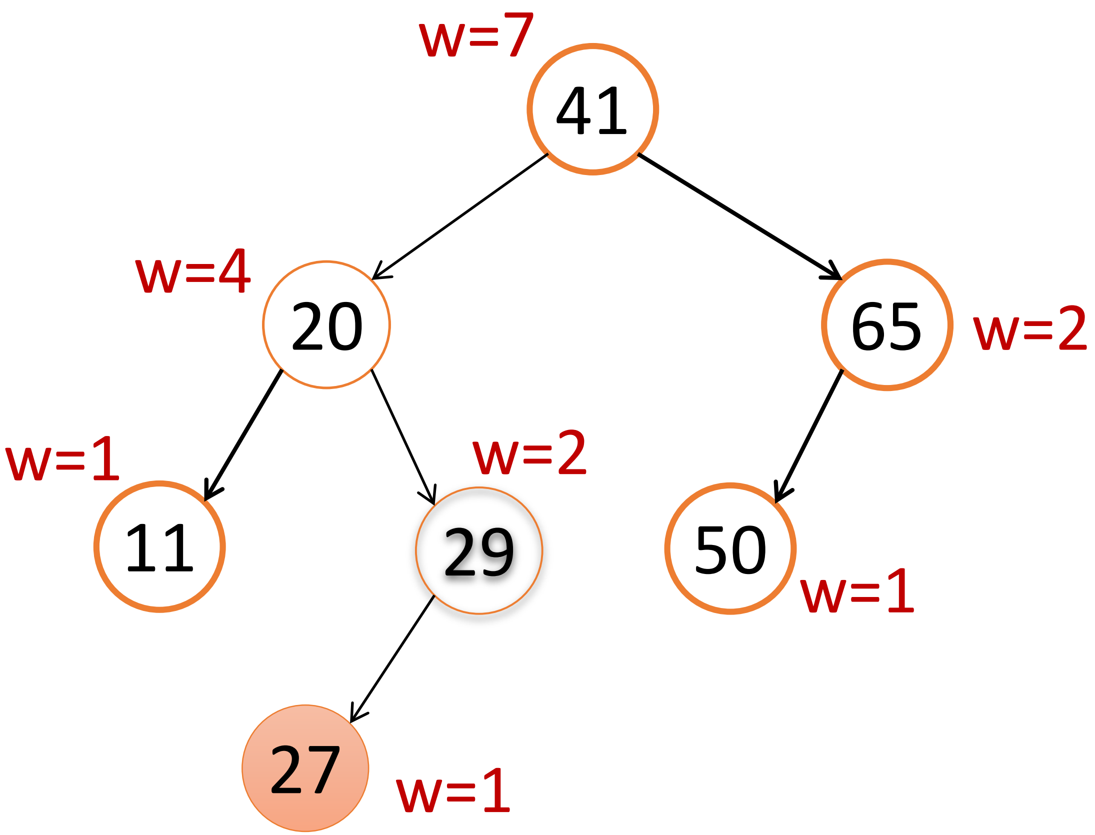
Dynamic Order Statistics

- `rank(v)`
  - Find the rank of v
  - E.g. what is the rank of 27?
- Find the rank of 27
  - Well we can `findMin` then in-order traversal to 27
  - Complexity?



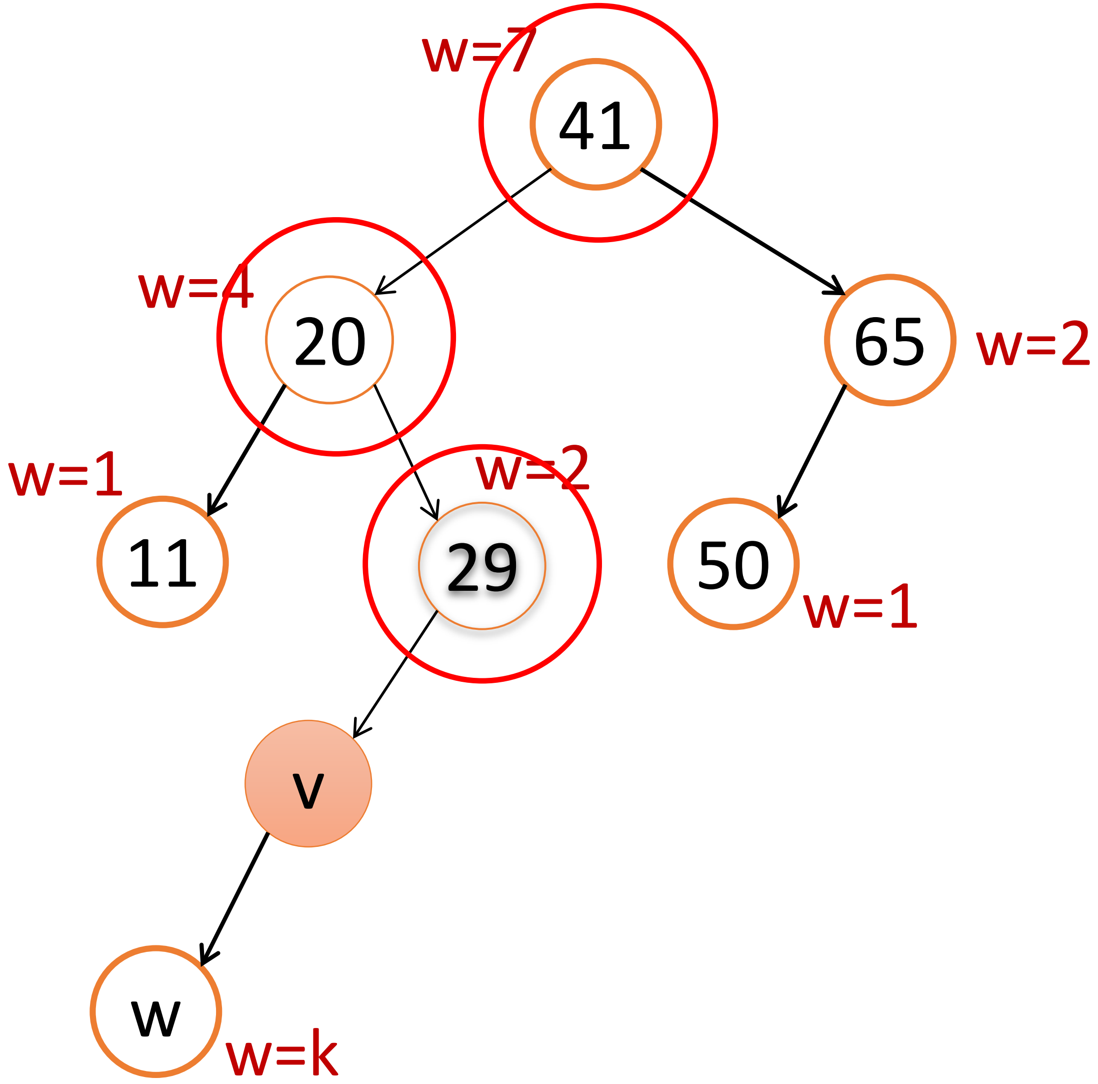
Dynamic Order Statistics

- `rank(v)`
- Idea: find the number of items that are smaller than v
- What are the nodes/subtrees that can tell you that?



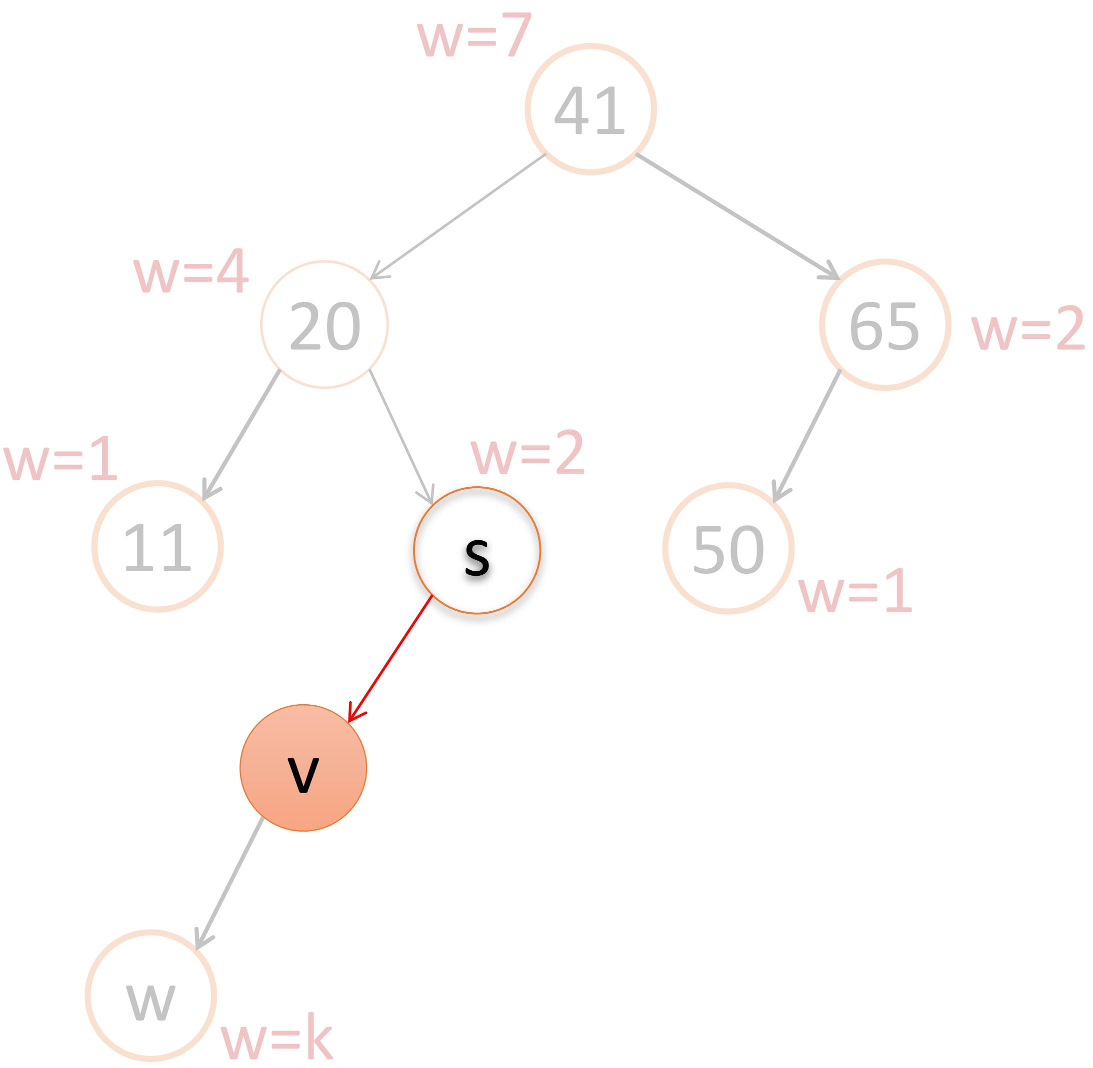
Dynamic Order Statistics

- rank(v)
  - Find all nodes that are smaller than v
- First, the node v may have a left child, and it will have k nodes that are smaller than v
- Second, when you trace all the ancestors of v, what are the nodes that you care?



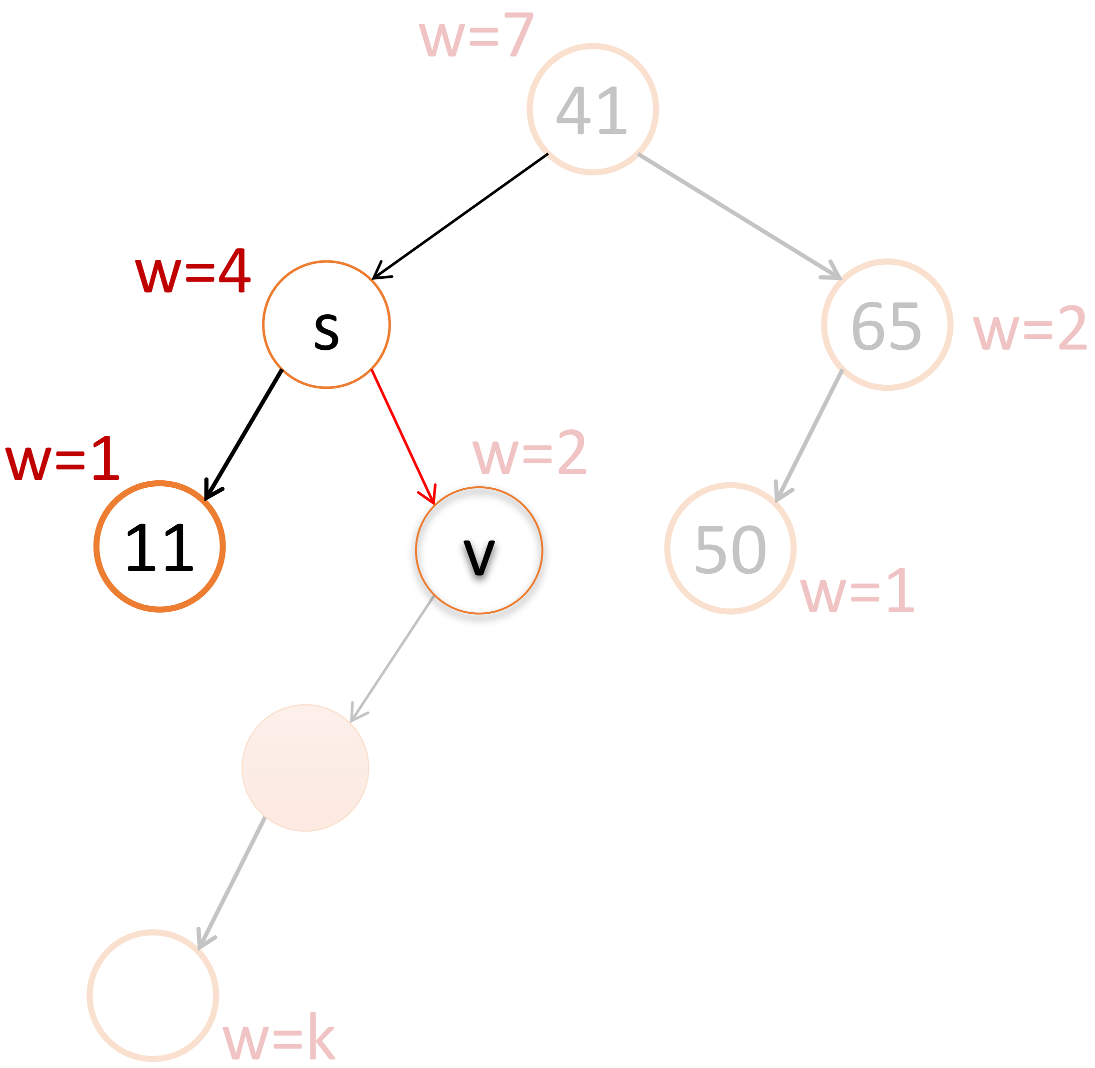
Dynamic Order Statistics

- `rank(v)`
  - Find all nodes that are smaller than v
- If we go up to a parent s that the current node is the left child
  - s and the right subtree of s will be all bigger than v
  - So we can ignore them



Dynamic Order Statistics

- `rank(v)`
  - Find all nodes that are smaller than v
- If we go up to a parent s that the current node is the right child
  - s and the left subtree of s will be all smaller than v
  - Number of nodes = the weight of the left child of s + 1

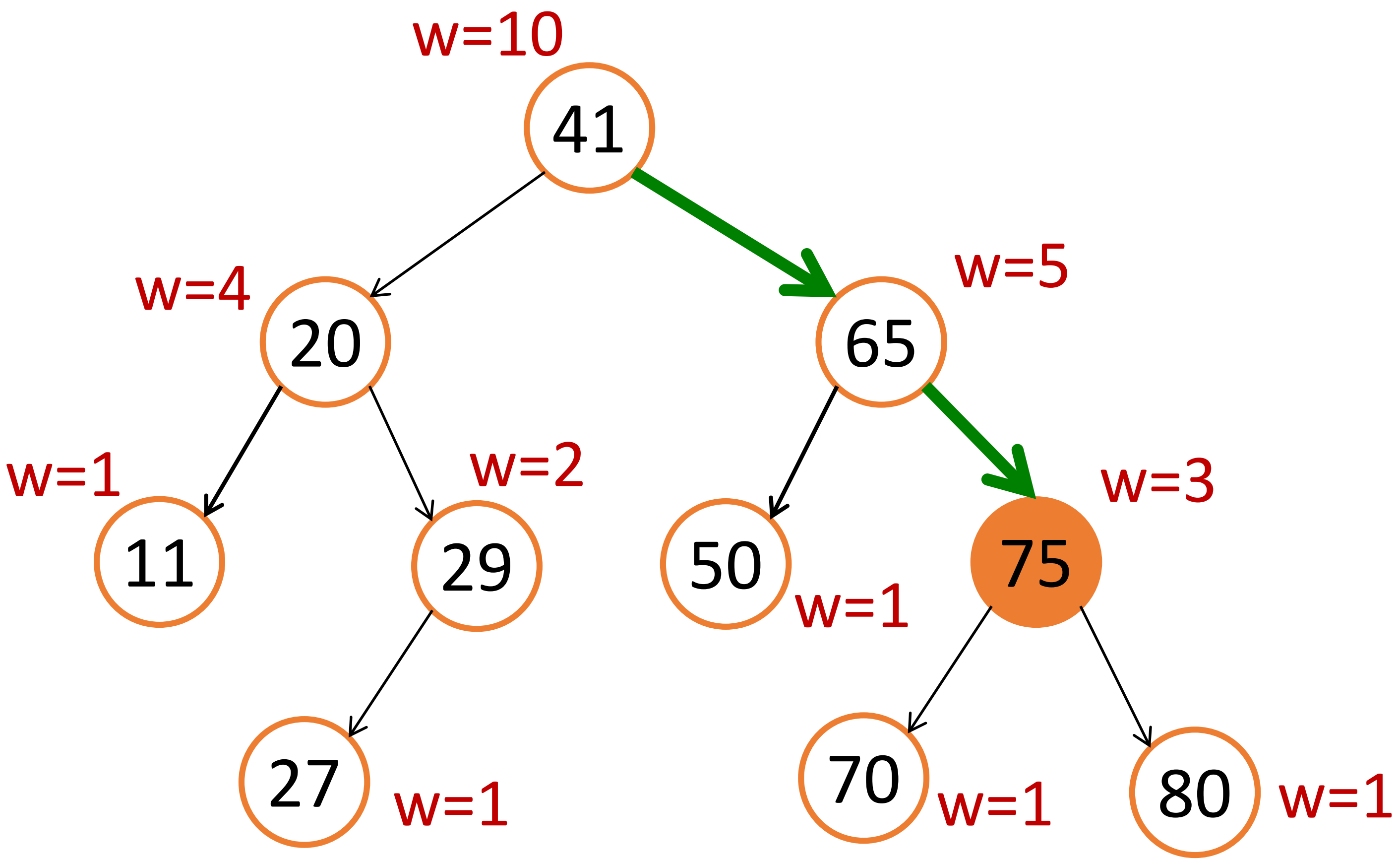


Rank(v)

```
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;
```

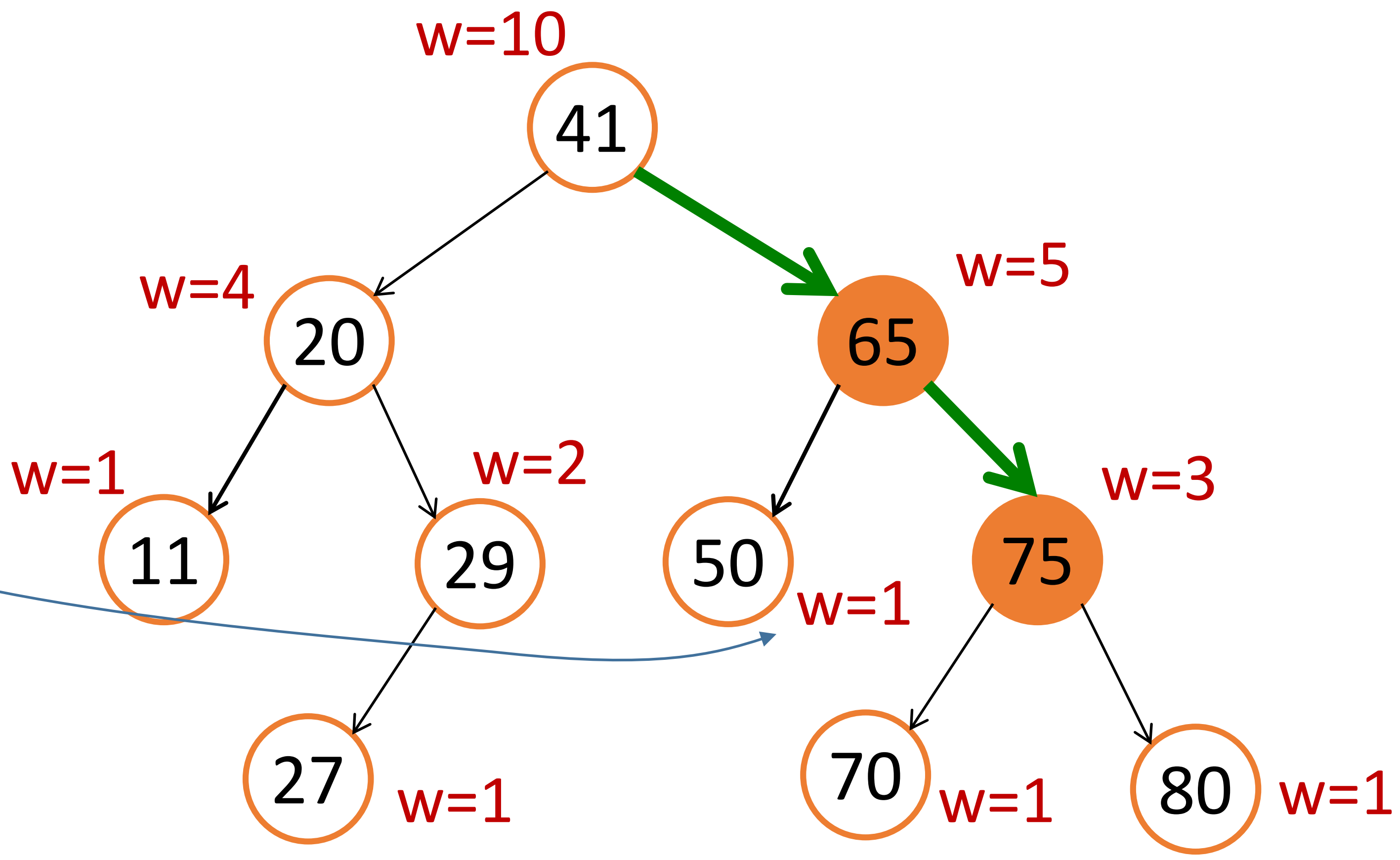
Rank Example

- `rank(75)`
- rank = 2



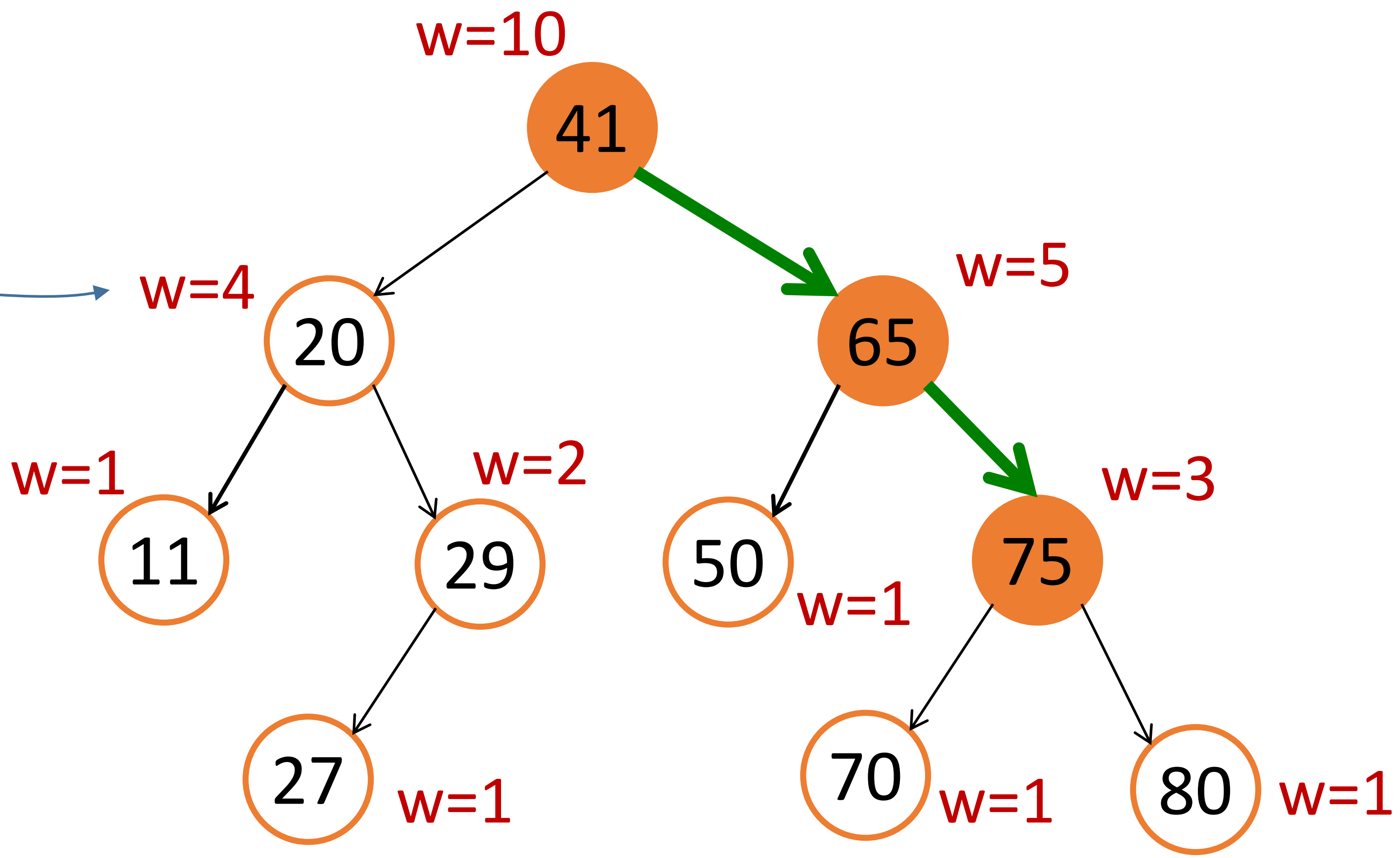
Rank Example

- `rank(75)`
- rank = rank + 1 + 1 = 4



Rank Example

- `rank(75)`
- rank = 4 + 4 + 1
- `rank(75)` = 9

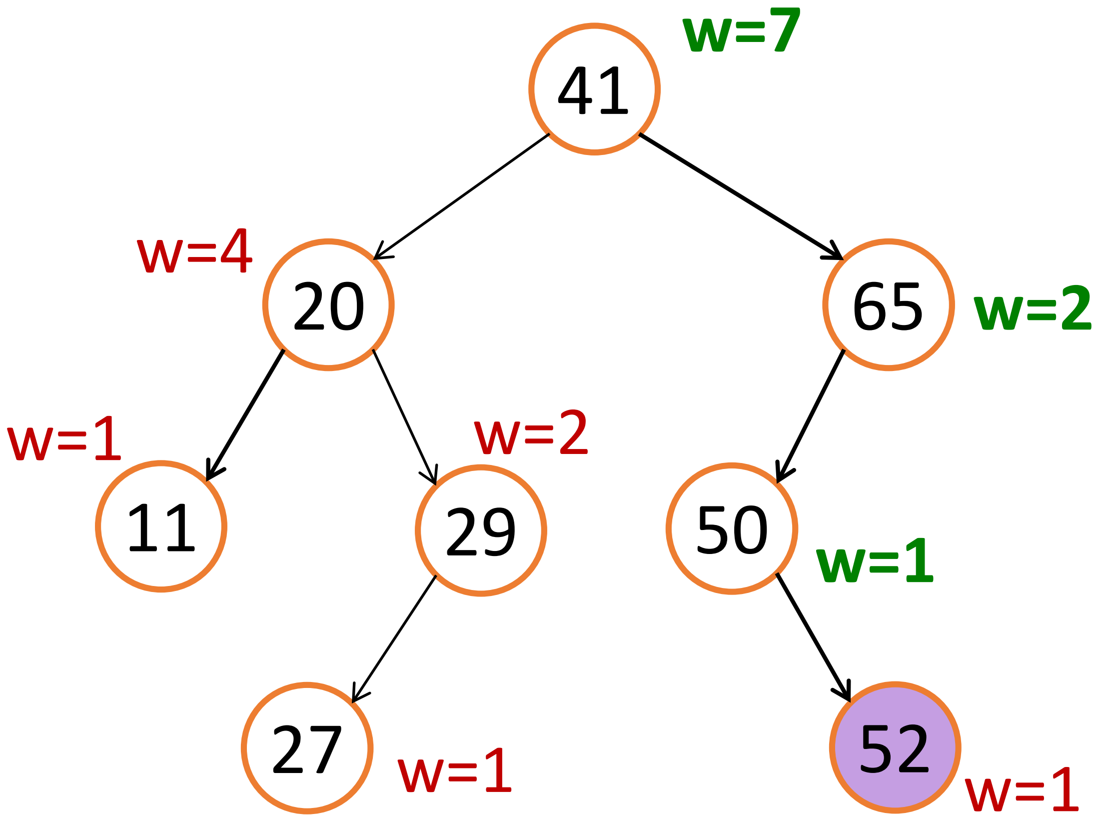


Basic methodology of Augmented Data Structures

- Choose underlying data structure
  - (tree, hash table, linked list, stack, etc.)
- Determine additional info needed.
- Modify data structure to maintain additional info when the structure changes.
  - (subject to insert/delete/etc.)
- Develop new operations.

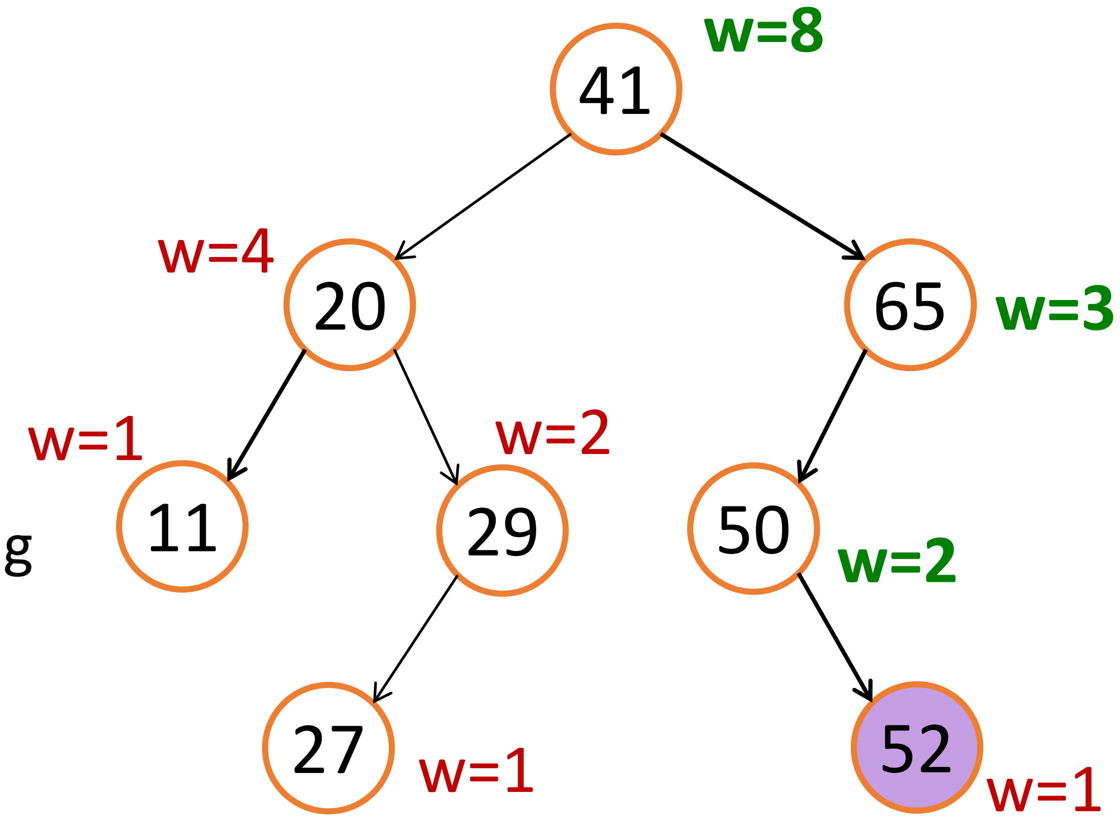
Maintain weight during insertions:

- E.g. `insert(52)`
- Some weights are not correct anymore
  - All the parents
- Update the weights of all the parents



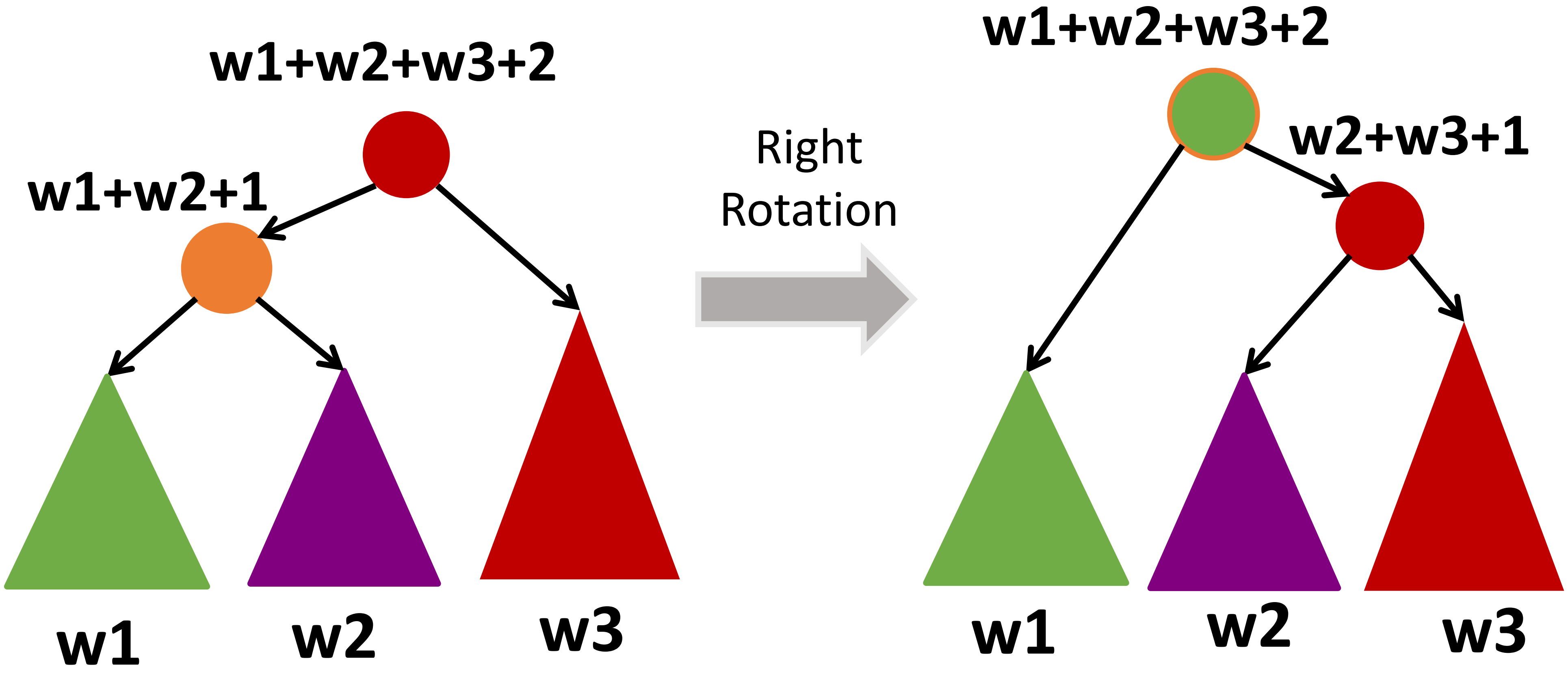
Maintain weight during insertions:

- E.g. `insert(52)`
- Now the weights are correct, but…
- Out of balance!
- We know how to do rotations to fix it, but…
  - How to maintain the weights during rotations?



Right Rotations

- For one rotation, how many nodes do we have to update?



Basic methodology of Augmented Data Structures

- Choose underlying data structure
  - (tree, hash table, linked list, stack, etc.)
- Determine additional info needed.
- Modify data structure to maintain additional info when the structure changes.
  - (subject to insert/delete/etc.)
- Develop new operations.

Today

- Two examples of augmenting balanced BSTs
  - Order Statistics
  - Orthogonal Range Searching

Range Search

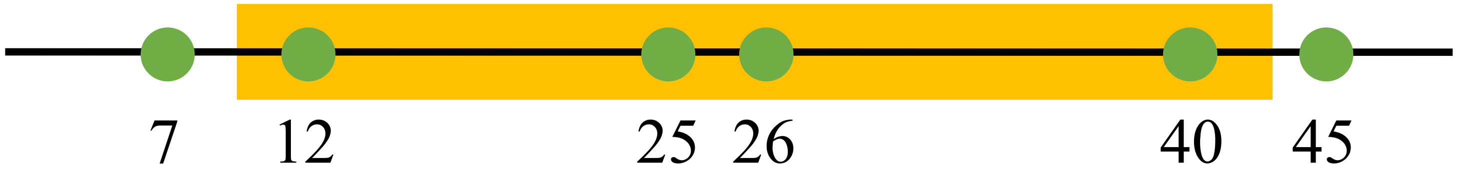
- Range List Query
  - Given two numbers a < b, **list out** all the elements x such that a <= x <= b
  - Easy!
  - $O(\log n + k)$
  - k is the number of output elements
- Range Count
  - Given two numbers a < b, **count** all the elements x such that a <= x <= b
  - Well, we can use the above
    - But… do we have to?

$$\text{rank}(b) - \text{rank}(a) + 1$$

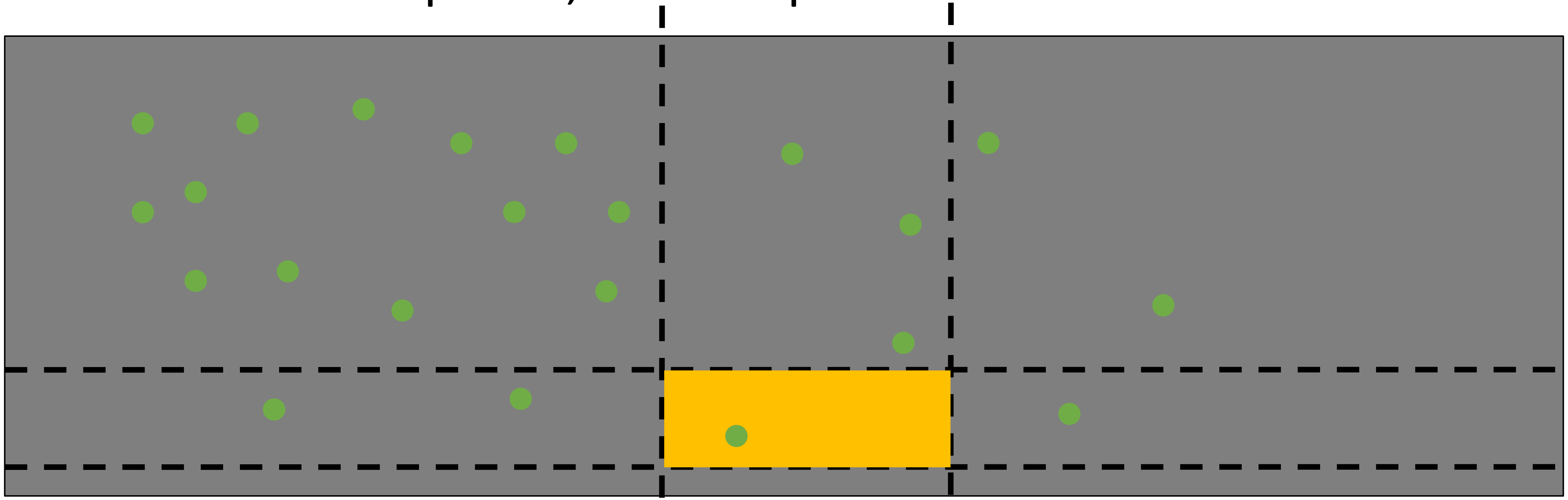
  - If a or b are not in BST, use their successor or predecessor respectively.

Range Search

- 1D: Given a range a,b, find all elements between a and b

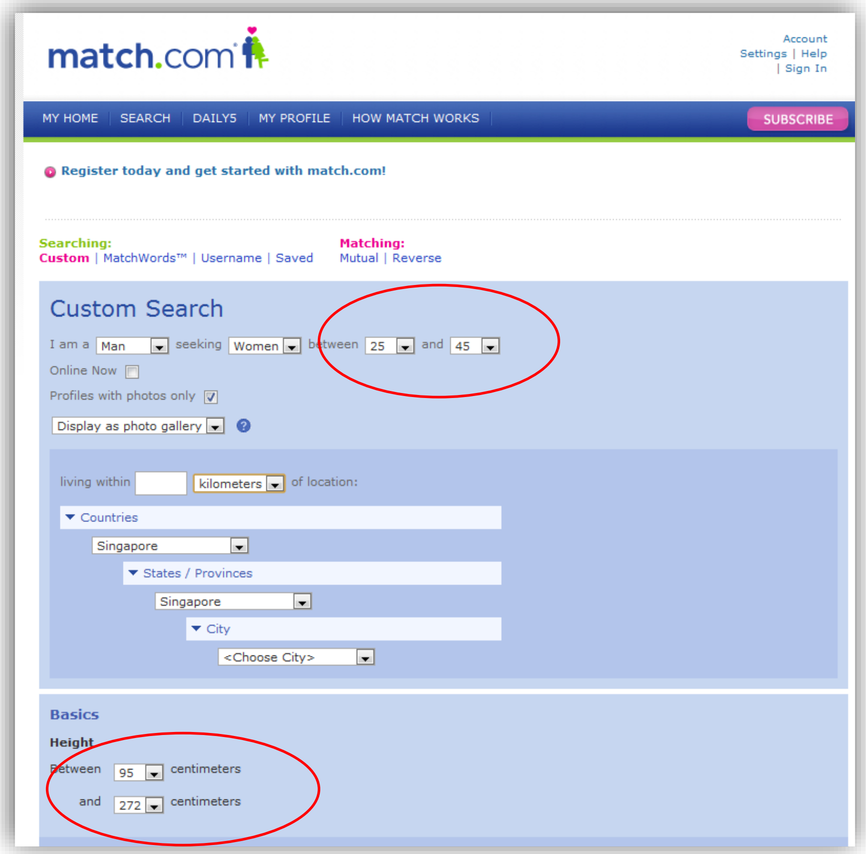


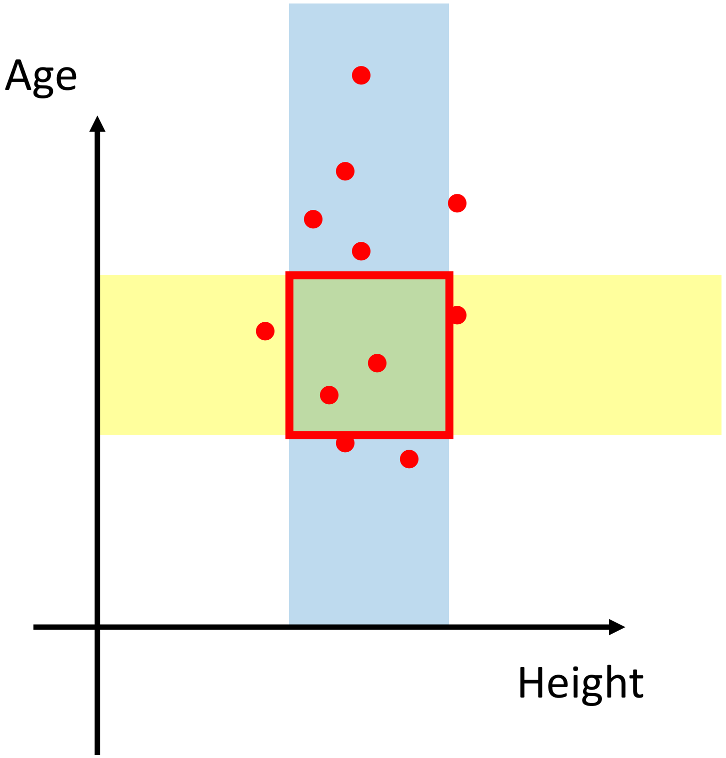
- 2D: Given some 2D points, find the points within a box



Multi-dimensional Range Query

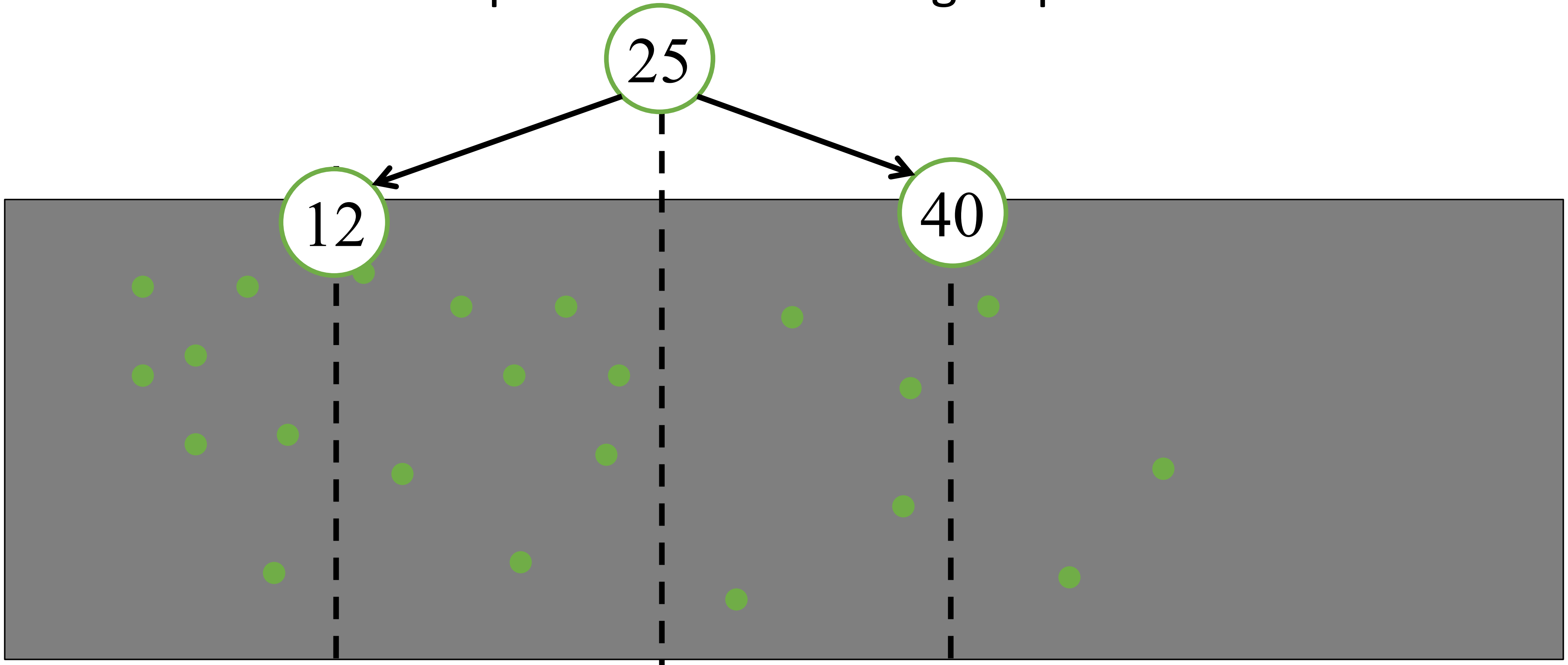
- Search for
  - Age between 25~30 AND
  - Height between 160~180cm





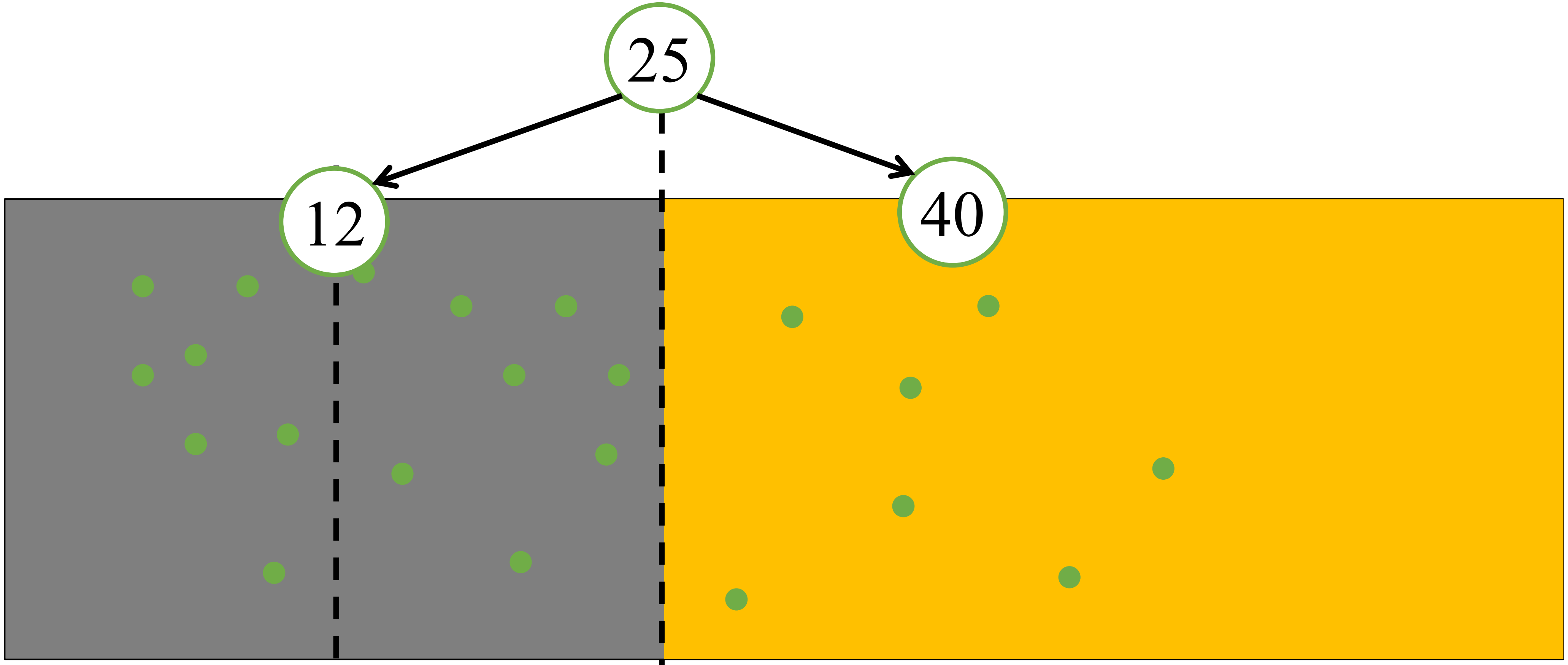
2D Range Search

- First, we build a BST according to the x-coordinates
- Each node divides the points into two subgroup



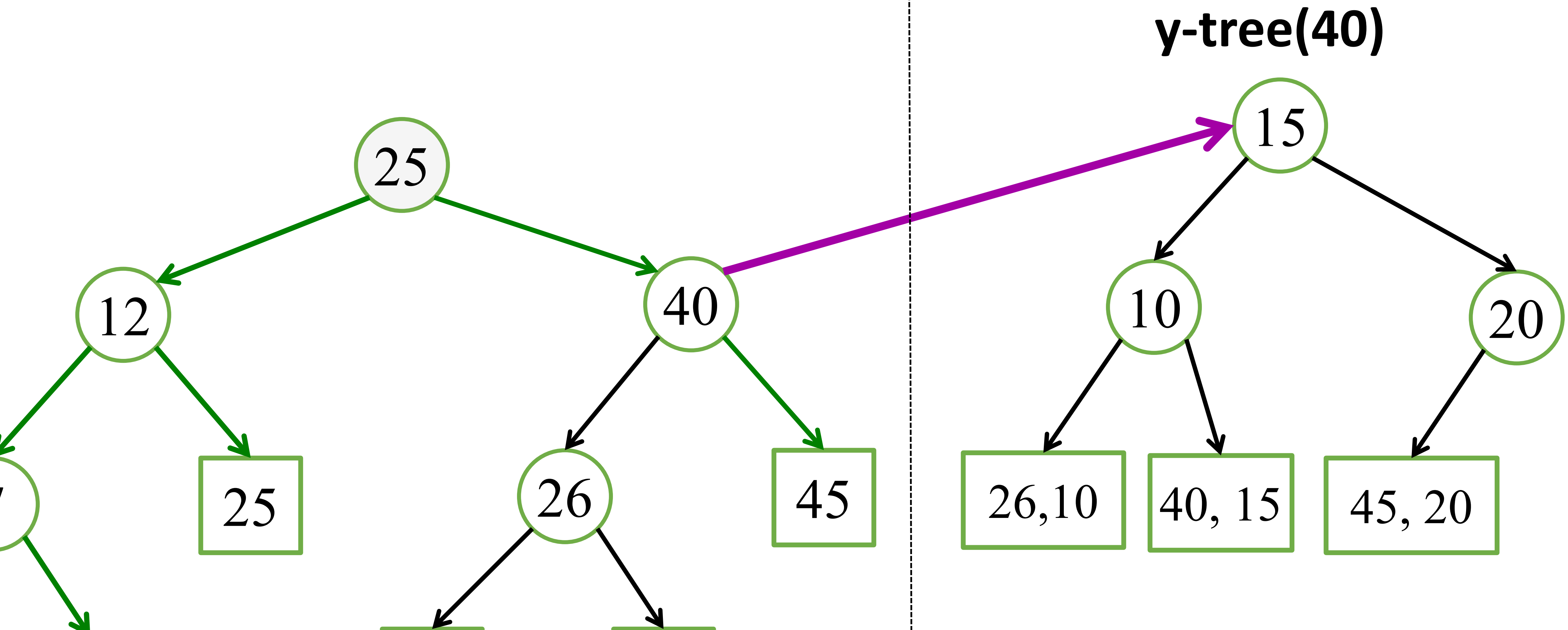
2D Range Search

- Node that the subtree of each node, includes a subset of the points, let's called this S



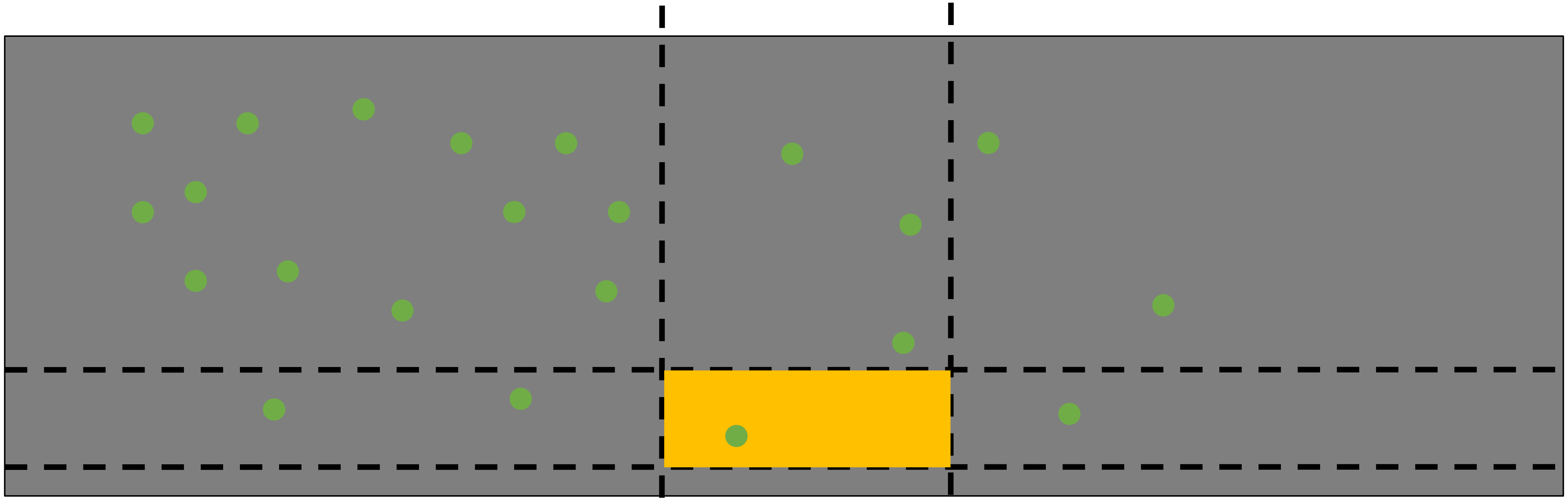
For each node

- Add a y-tree that is the BST of S by y-coodinates!



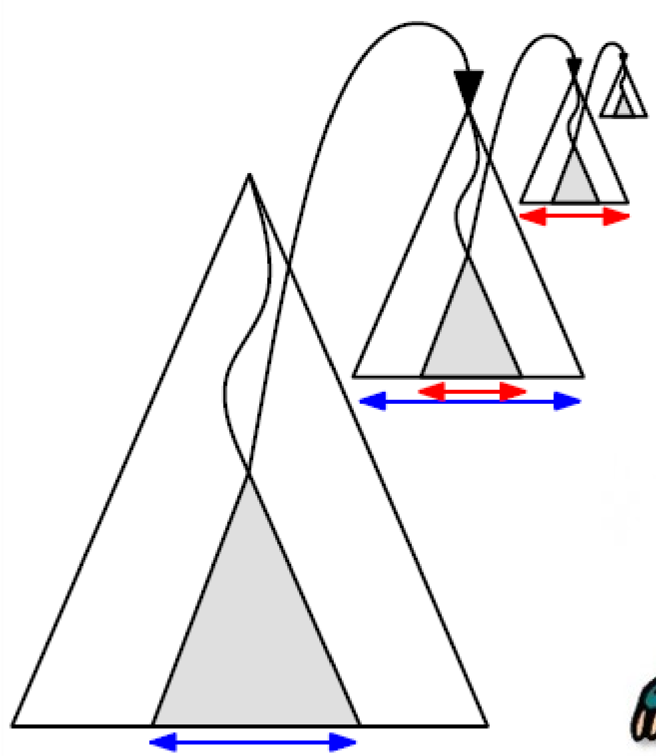
2D Range Search

- Build an x-tree using only x-coordinates.
- For every node in the x-tree, build a y-tree out of nodes in subtree using only y-coordinates



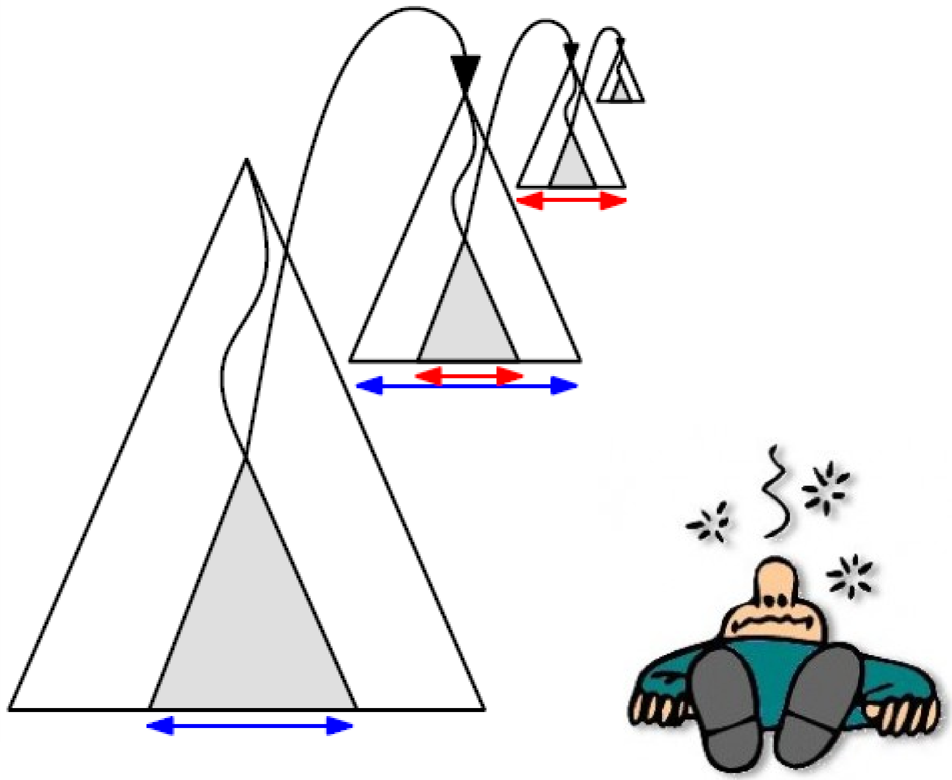
Query time: $O(\log^2 n + k)$

- Time:
  - $O(\log n)$ to find starting/ending node in x
  - And for each node in x, $O(\log n)$ y-tree-searches of cost $O(\log n)$
  - $O(k)$ enumerating output
- Space: $O(n \log n)$
  - Why?
- However,
  - Insertion, deletion, rotation maybe $O(n)$!



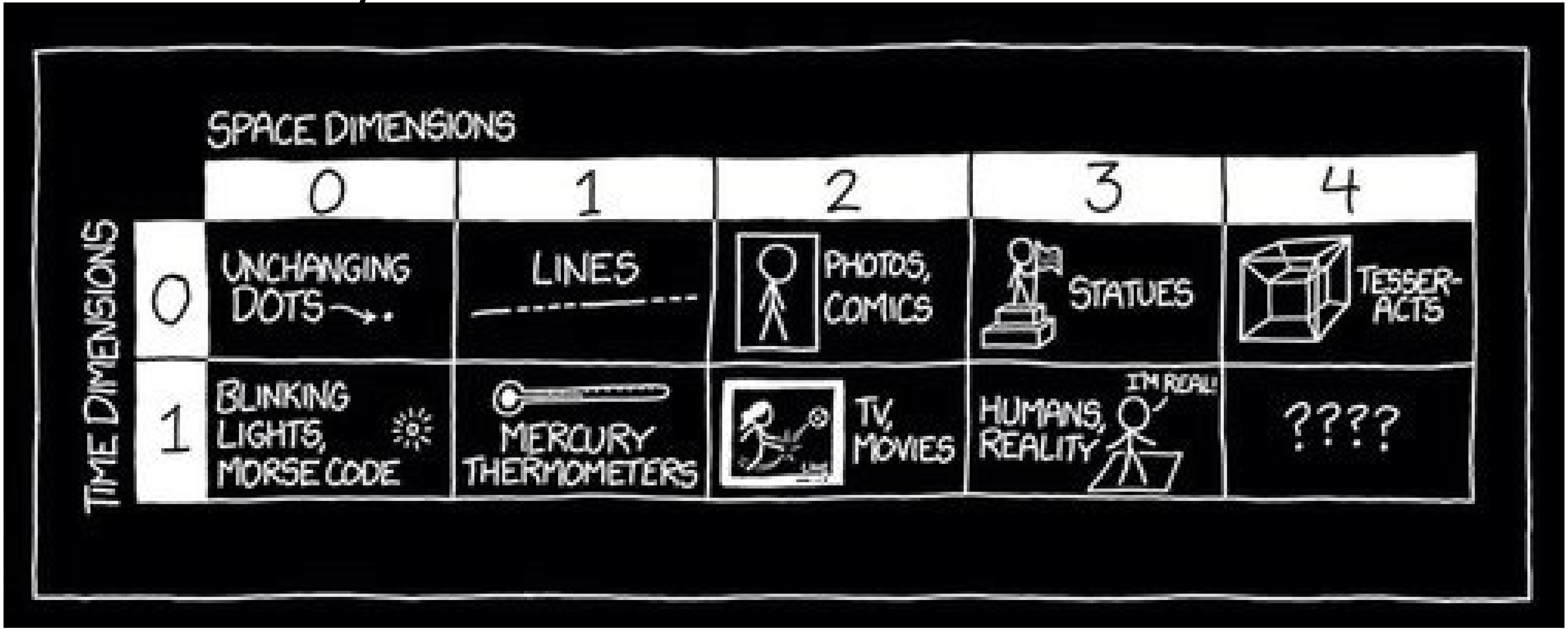
From 2D to N-dim

- What if you want high-dimensional range queries?
  - Query cost: $O(\log^d n + k)$
  - buildTree cost: $O(n \log^{d-1} n)$
  - Space: $O(n \log^{d-1} n)$



- Idea:
  - Store d–1 dimensional range-tree in each node of a 1D range-tree.
  - Construct the d–1-dimeionsal range-tree recursively.

The XKCD Guide to the Universe's Most Bizarre Physics



- https://www.wired.com/2014/11/xkcd-guide-to-dimensions/