CS 240 – Data Structures and Data Management

Module 8: Range-Searching in Dictionaries for Points

O. Veksler

Based on lecture notes by many previous cs240 instructors

David R. Cheriton School of Computer Science, University of Waterloo

Summer 2023

Outline

- Range-Searching in Dictionaries for Points
    - Range Trees
    - Conclusion

Outline

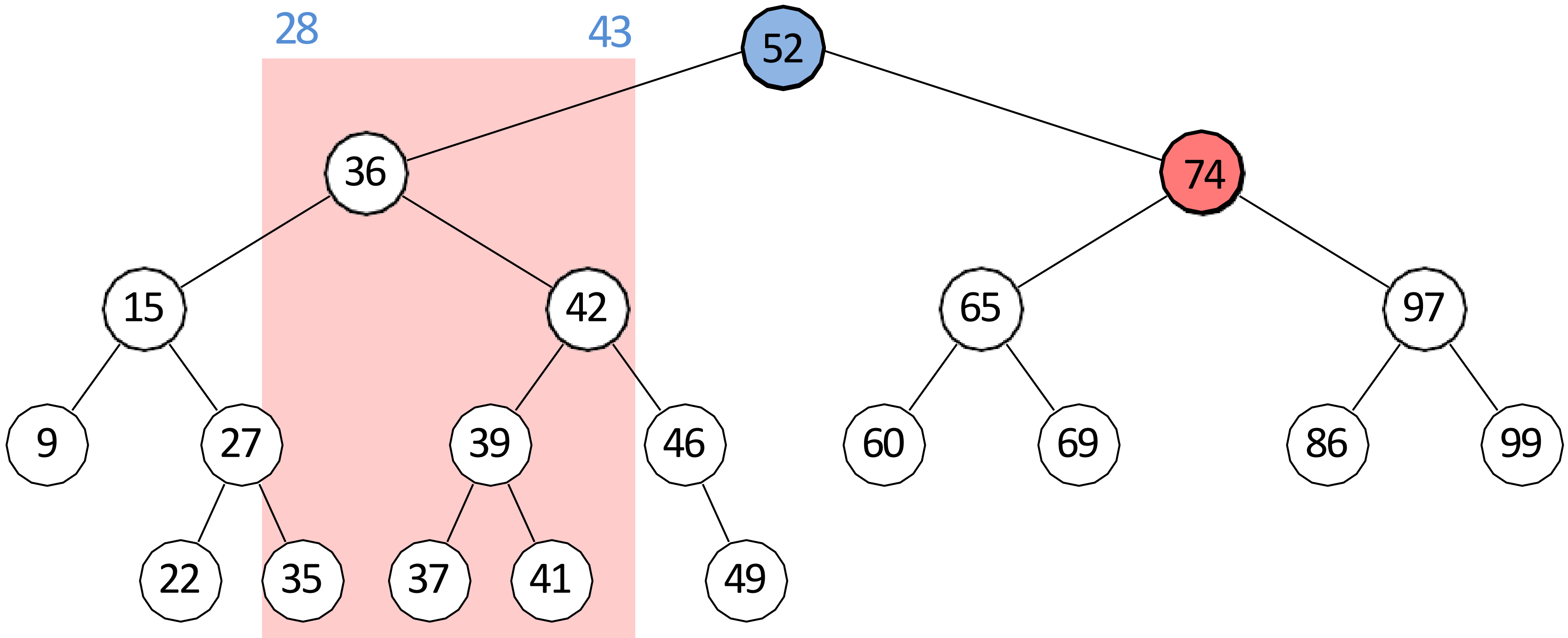
- Range-Searching in Dictionaries for Points
    - Range Trees
    - Conclusion

Towards Range Trees

- Quadtrees and kd-trees
  - intuitive and simple
  - but both may be slow for range searches
  - quadtrees are also potentially wasteful in space
- Consider BST/AVL trees
  - efficient for one-dimensional dictionaries, if balanced
    - range search is also efficient
  - can we use ideas from BST/AVL trees for multi dimensional dictionaries?
- First let us consider range search in BST

BST Range Search example

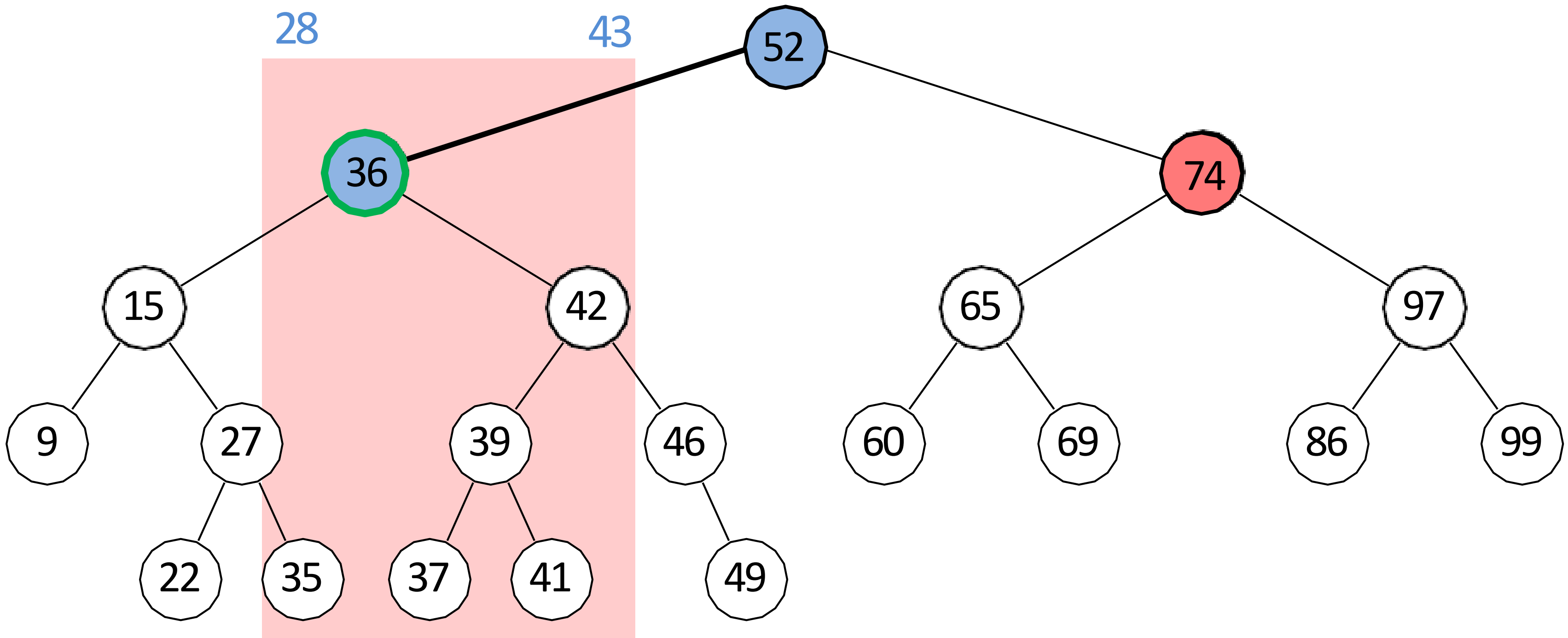
BST::RangeSearch-recursive$(T, 28, 43)$



- blue node: recurse either to the left, or to the right, or both (according to the key value)
  - boundary node, one or both subtrees may intersect range query
- red node: range search was not called on red node, but was called on its parent
  - outside node, subtree does not intersect range query
- green node : all the keys in the subtree are in the range
  - inside node, subtree completely inside range query

BST Range Search example

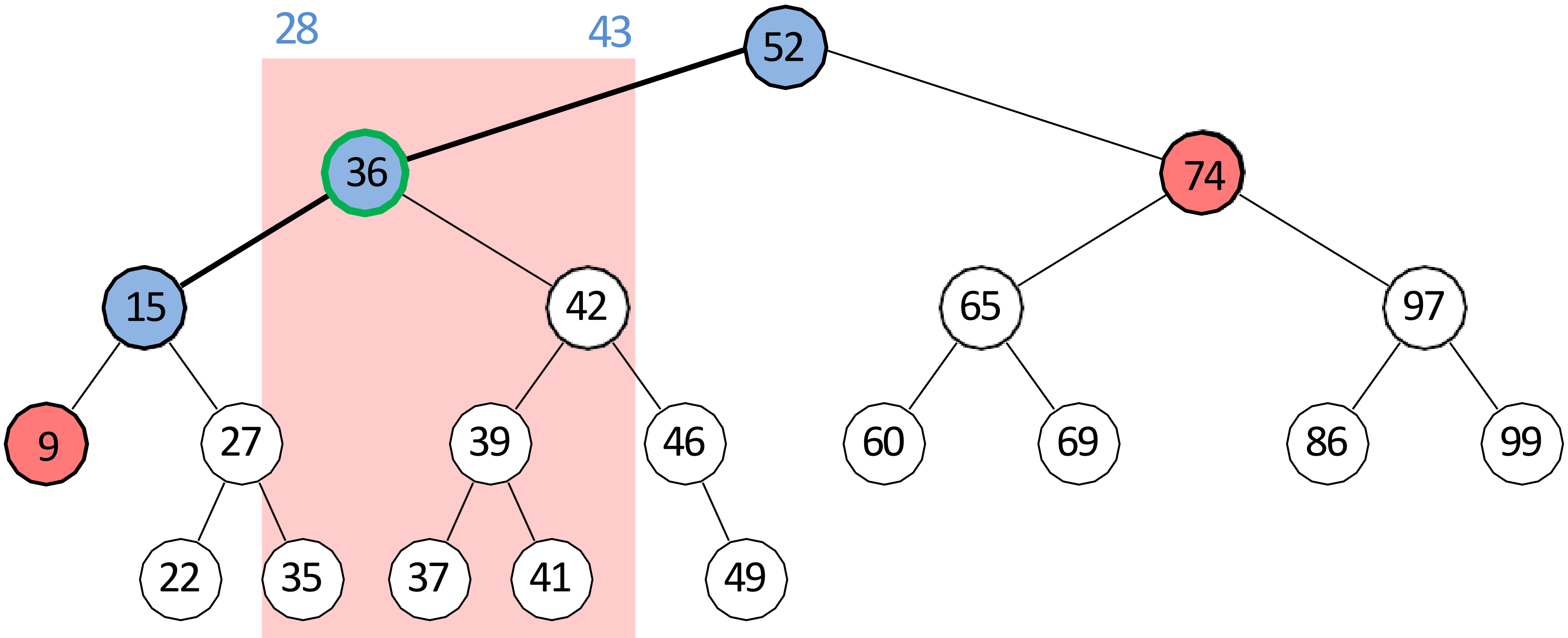
BST::RangeSearch-recursive$(T, 28, 43)$



- blue node: recurse either to the left, or to the right, or both (according to the key value)
    - boundary node, one or both subtrees may intersect range query
- red node: range search was not called on red node, but was called on its parent
    - outside node, subtree does not intersect range query
- green node : all the keys in the subtree are in the range
    - inside node, subtree completely inside range query

BST Range Search example

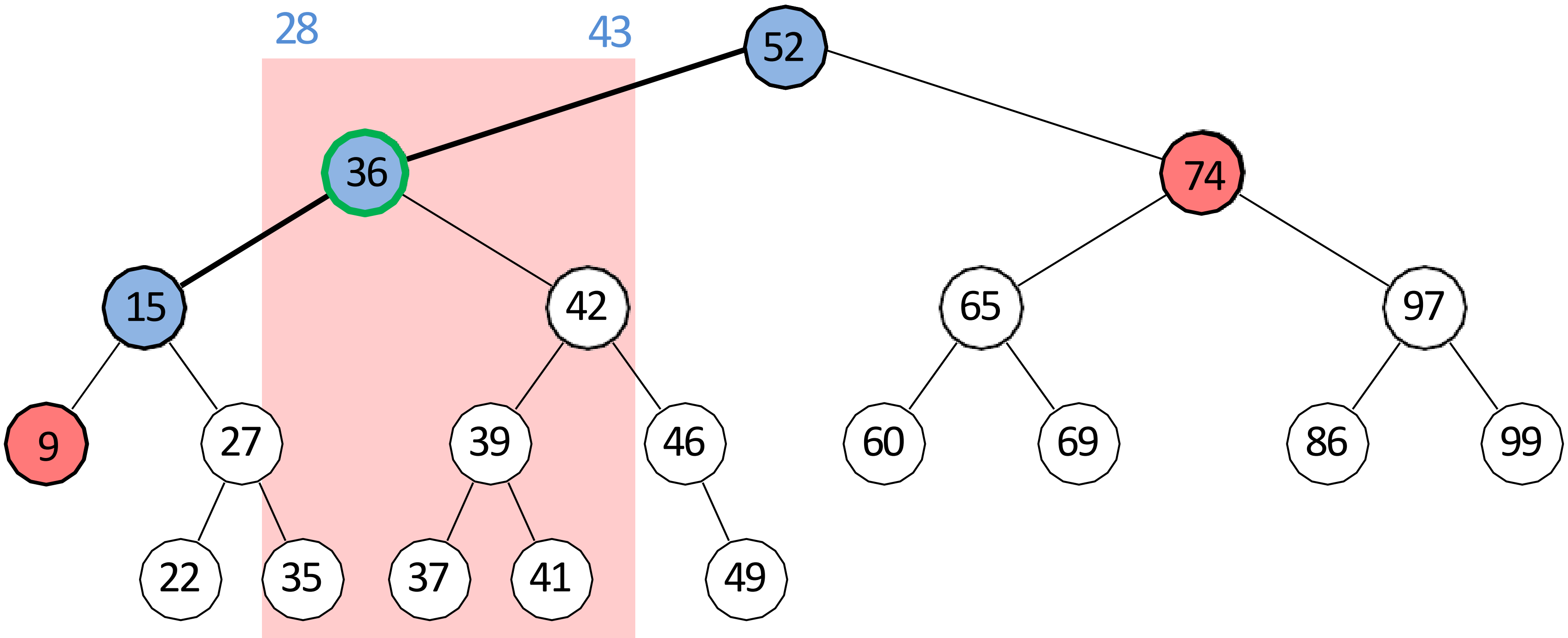
$\mathit{BST::RangeSearch\text{-}recursive}(T, 28, 43)$



- blue node: recurse either to the left, or to the right, or both (according to the key value)
  - boundary node, one or both subtrees may intersect range query
- red node: range search was not called on red node, but was called on its parent
  - outside node, subtree does not intersect range query
- green node : all the keys in the subtree are in the range
  - inside node, subtree completely inside range query

BST Range Search example

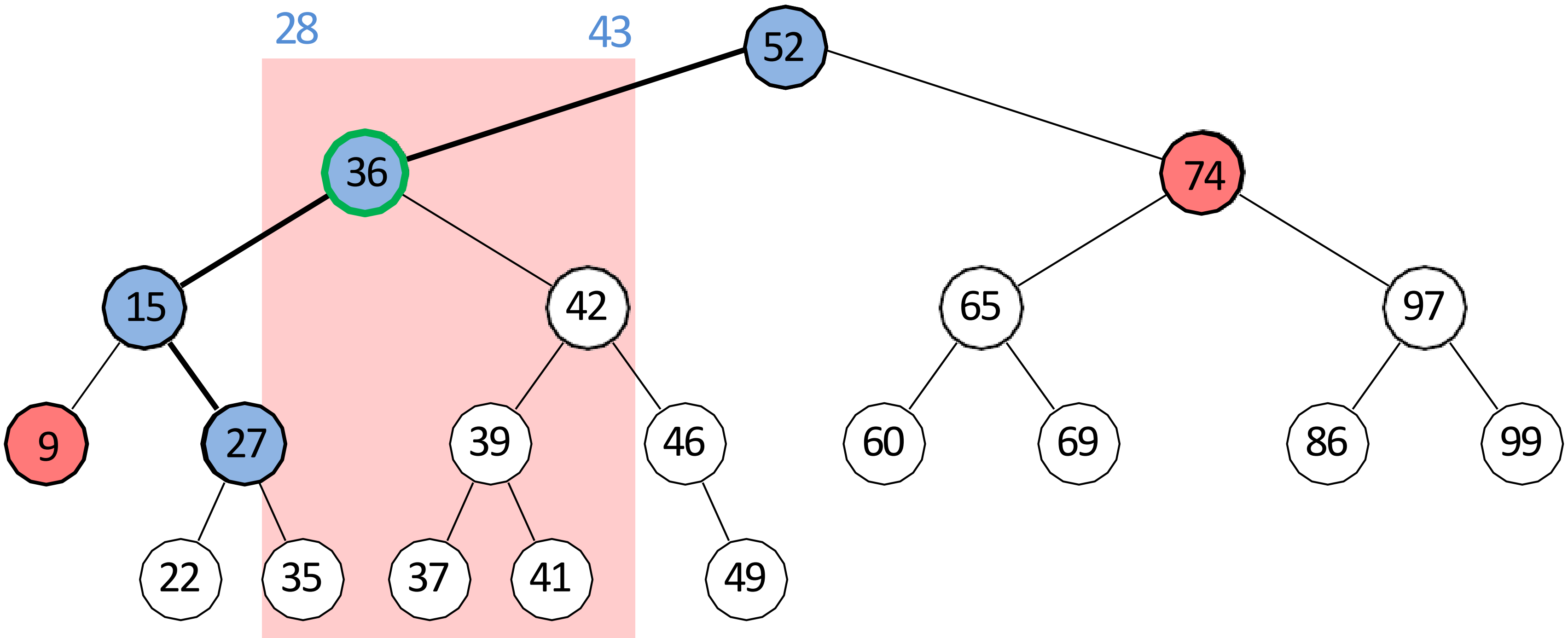
$\textit{BST::RangeSearch-recursive}(T, 28, 43)$



- blue node: recurse either to the left, or to the right, or both (according to the key value)
  - boundary node, one or both subtrees may intersect range query
- red node: range search was not called on red node, but was called on its parent
  - outside node, subtree does not intersect range query
- green node : all the keys in the subtree are in the range
  - inside node, subtree completely inside range query

BST Range Search example

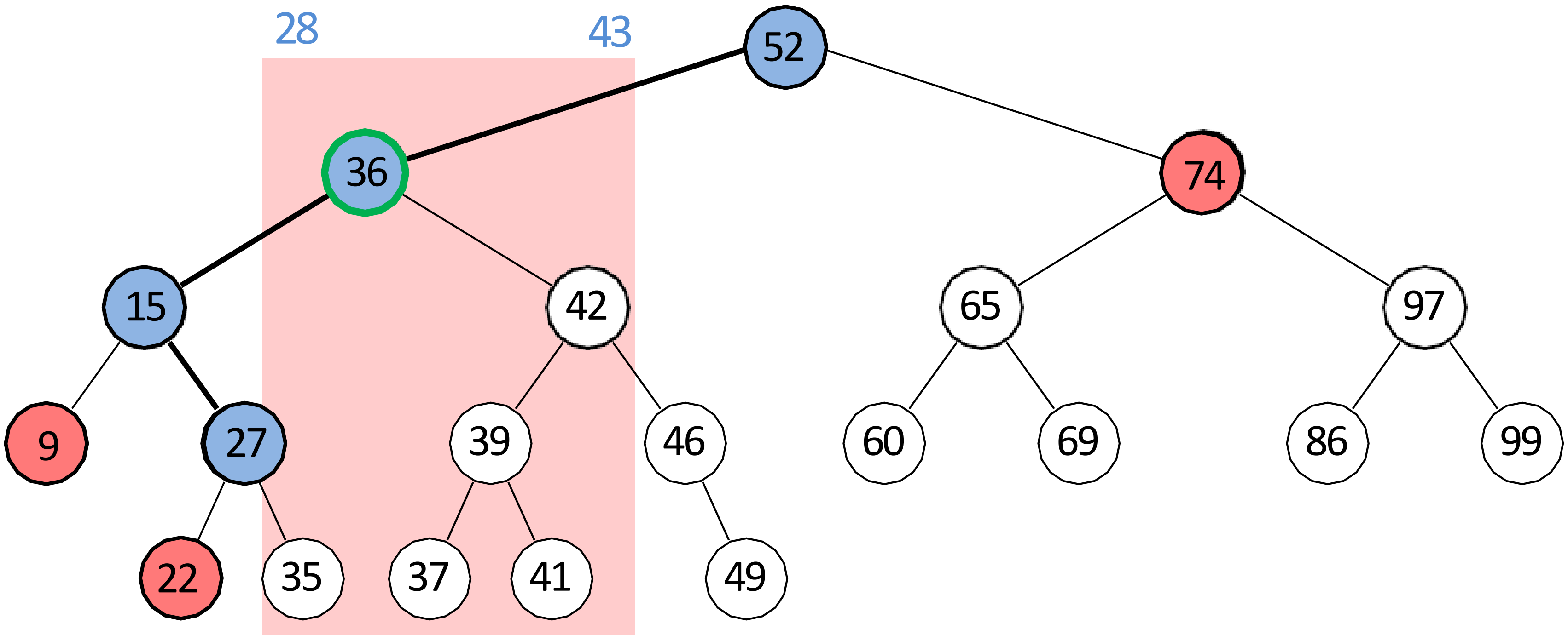
$\textit{BST::RangeSearch-recursive}(T, 28, 43)$



- blue node: recurse either to the left, or to the right, or both (according to the key value)
  - boundary node, one or both subtrees may intersect range query
- red node: range search was not called on red node, but was called on its parent
  - outside node, subtree does not intersect range query
- green node : all the keys in the subtree are in the range
  - inside node, subtree completely inside range query

BST Range Search example

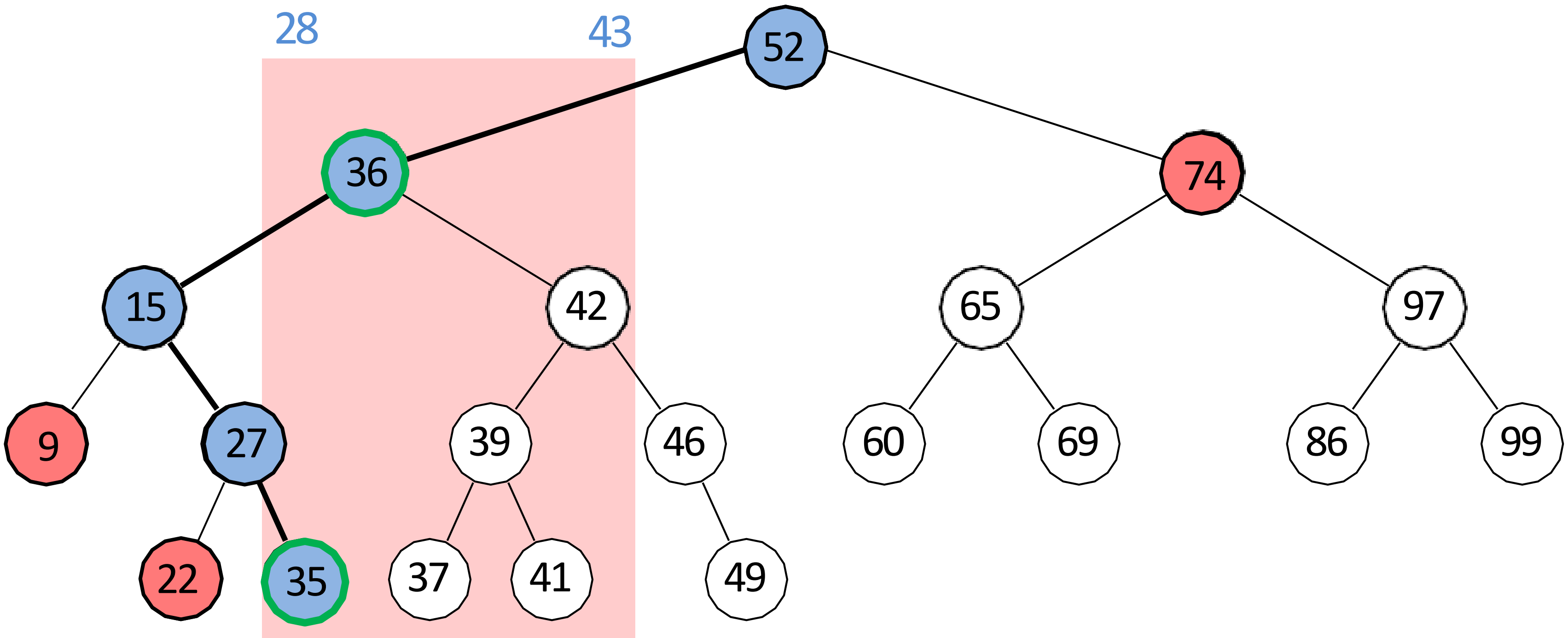
BST::RangeSearch-recursive$(T, 28, 43)$



- blue node: recurse either to the left, or to the right, or both (according to the key value)
  - boundary node, one or both subtrees may intersect range query
- red node: range search was not called on red node, but was called on its parent
  - outside node, subtree does not intersect range query
- green node : all the keys in the subtree are in the range
  - inside node, subtree completely inside range query

BST Range Search example

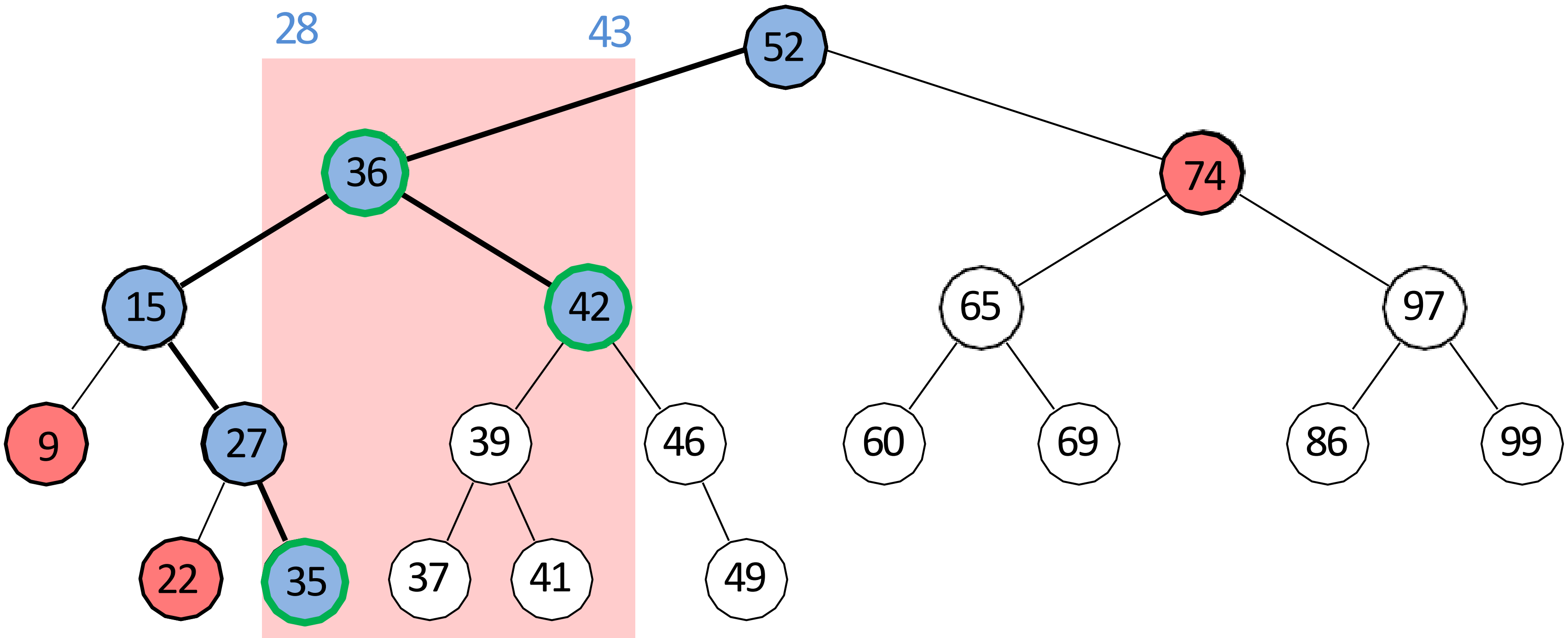
BST::RangeSearch-recursive$(T, 28, 43)$



- blue node: recurse either to the left, or to the right, or both (according to the key value)
  - boundary node, one or both subtrees may intersect range query
- red node: range search was not called on red node, but was called on its parent
  - outside node, subtree does not intersect range query
- green node : all the keys in the subtree are in the range
  - inside node, subtree completely inside range query

BST Range Search example

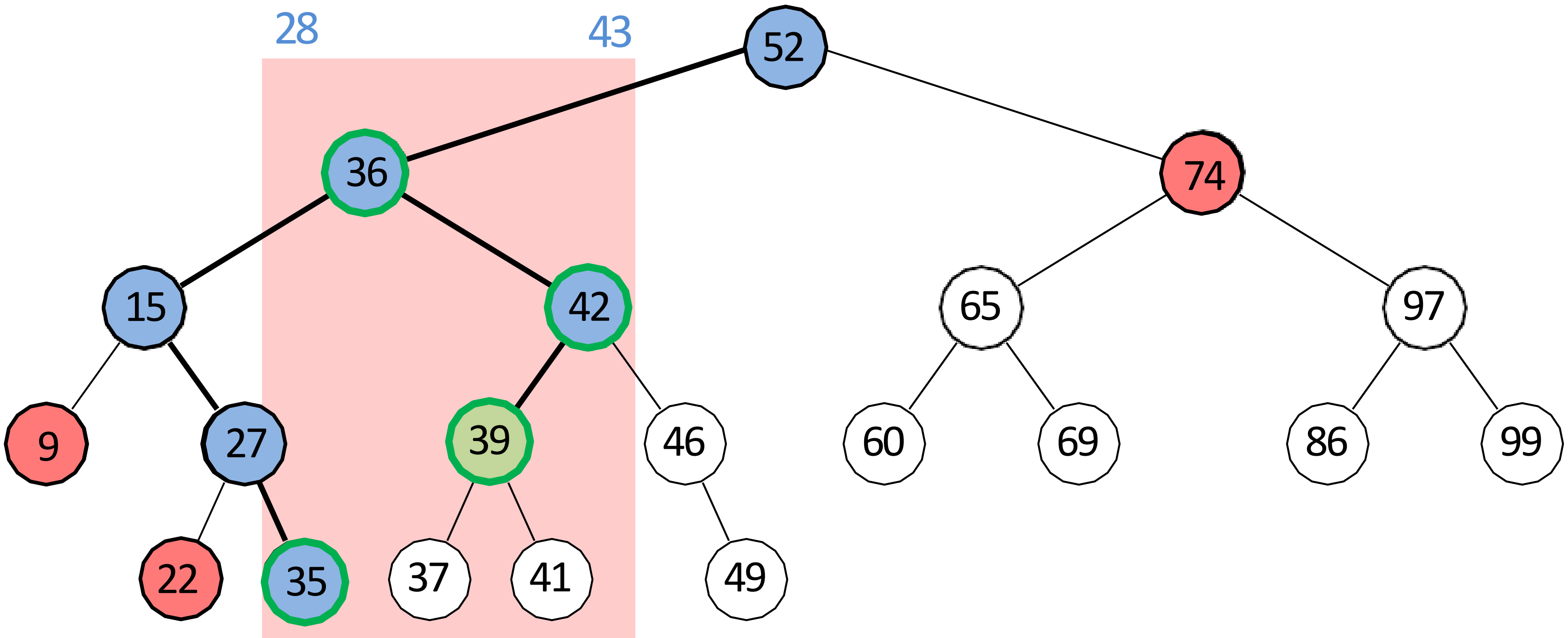
$BST::RangeSearch\text{-}recursive(T, 28, 43)$



- blue node: recurse either to the left, or to the right, or both (according to the key value)
  - boundary node, one or both subtrees may intersect range query
- red node: range search was not called on red node, but was called on its parent
  - outside node, subtree does not intersect range query
- green node : all the keys in the subtree are in the range
  - inside node, subtree completely inside range query

BST Range Search example

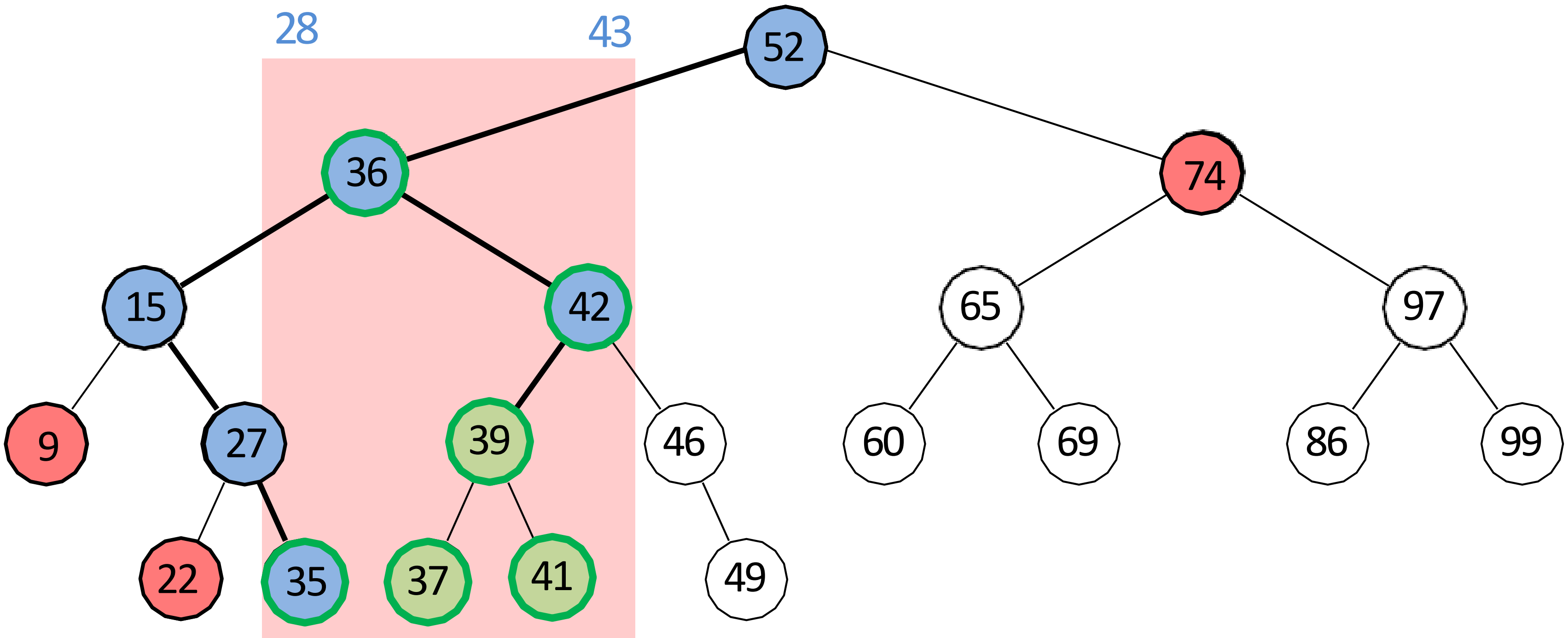
BST::RangeSearch-recursive$(T, 28, 43)$



- blue node: recurse either to the left, or to the right, or both (according to the key value)
  - boundary node, one or both subtrees may intersect range query
- red node: range search was not called on red node, but was called on its parent
  - outside node, subtree does not intersect range query
- green node : all the keys in the subtree are in the range
  - inside node, subtree completely inside range query

BST Range Search example

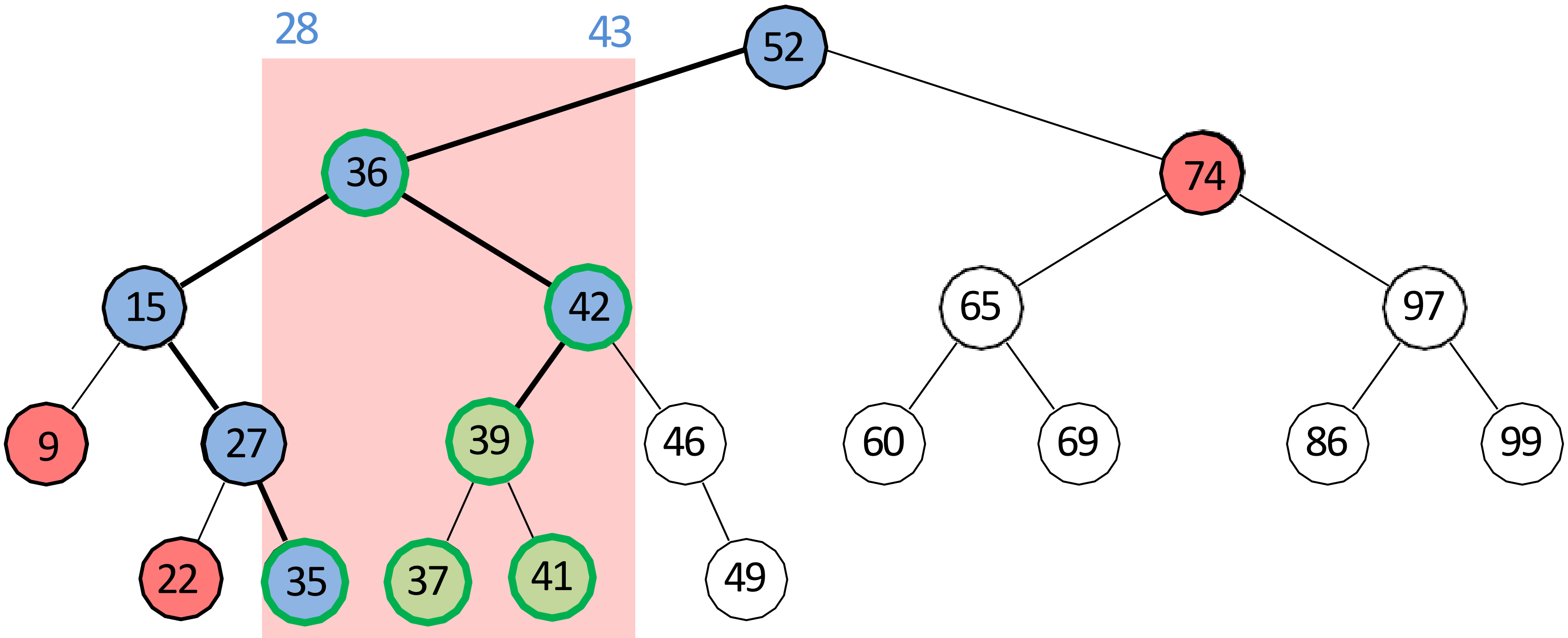
BST::RangeSearch-recursive$(T, 28, 43)$



- blue node: recurse either to the left, or to the right, or both (according to the key value)
  - boundary node, one or both subtrees may intersect range query
- red node: range search was not called on red node, but was called on its parent
  - outside node, subtree does not intersect range query
- green node : all the keys in the subtree are in the range
  - inside node, subtree completely inside range query

BST Range Search example

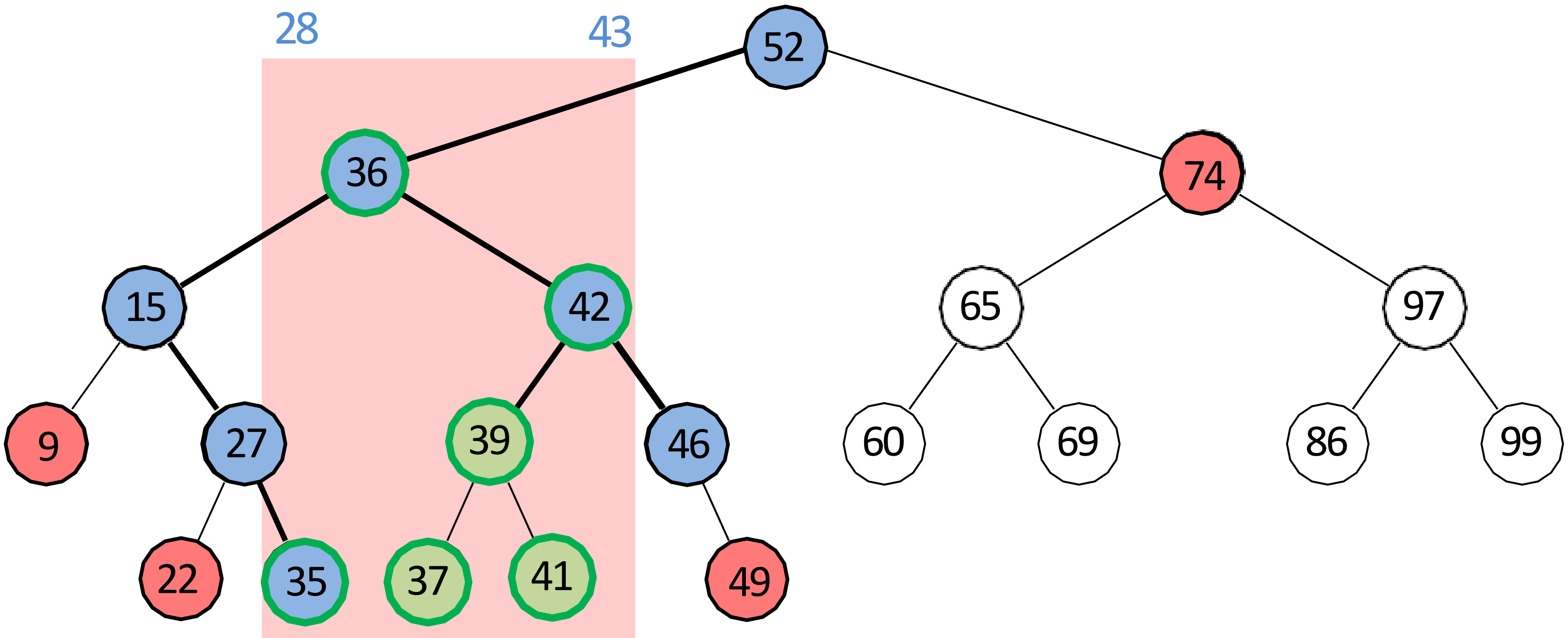
BST::RangeSearch-recursive$(T, 28, 43)$



- blue node: recurse either to the left, or to the right, or both (according to the key value)
  - boundary node, one or both subtrees may intersect range query
- red node: range search was not called on red node, but was called on its parent
  - outside node, subtree does not intersect range query
- green node : all the keys in the subtree are in the range
  - inside node, subtree completely inside range query

BST Range Search example

BST::RangeSearch-recursive$(T, 28, 43)$



- blue node: recurse either to the left, or to the right, or both (according to the key value)
    - boundary node, one or both subtrees may intersect range query
- red node: range search was not called on red node, but was called on its parent
    - outside node, subtree does not intersect range query
- green node : all the keys in the subtree are in the range
    - inside node, subtree completely inside range query

BST Range Search

BST::RangeSearch-recursive$(r \leftarrow root, k_1, k_2)$

r: root of a binary search tree, k_1, k_2: search keys

Returns keys in subtree at r that are in range $[k_1, k_2]$

if $r = NIL$ **then return**

if $k_1 \leq r.key \leq k_2$ **then**

$L \leftarrow$ *BST::RangeSearch-recursive*$(r.left, k_1, k_2)$

$R \leftarrow$ *BST::RangeSearch-recursive*$(l.right, k_1, k_2)$

return $L \cup \{r.key\} \cup R$

if $r.key < k_1$ **then**

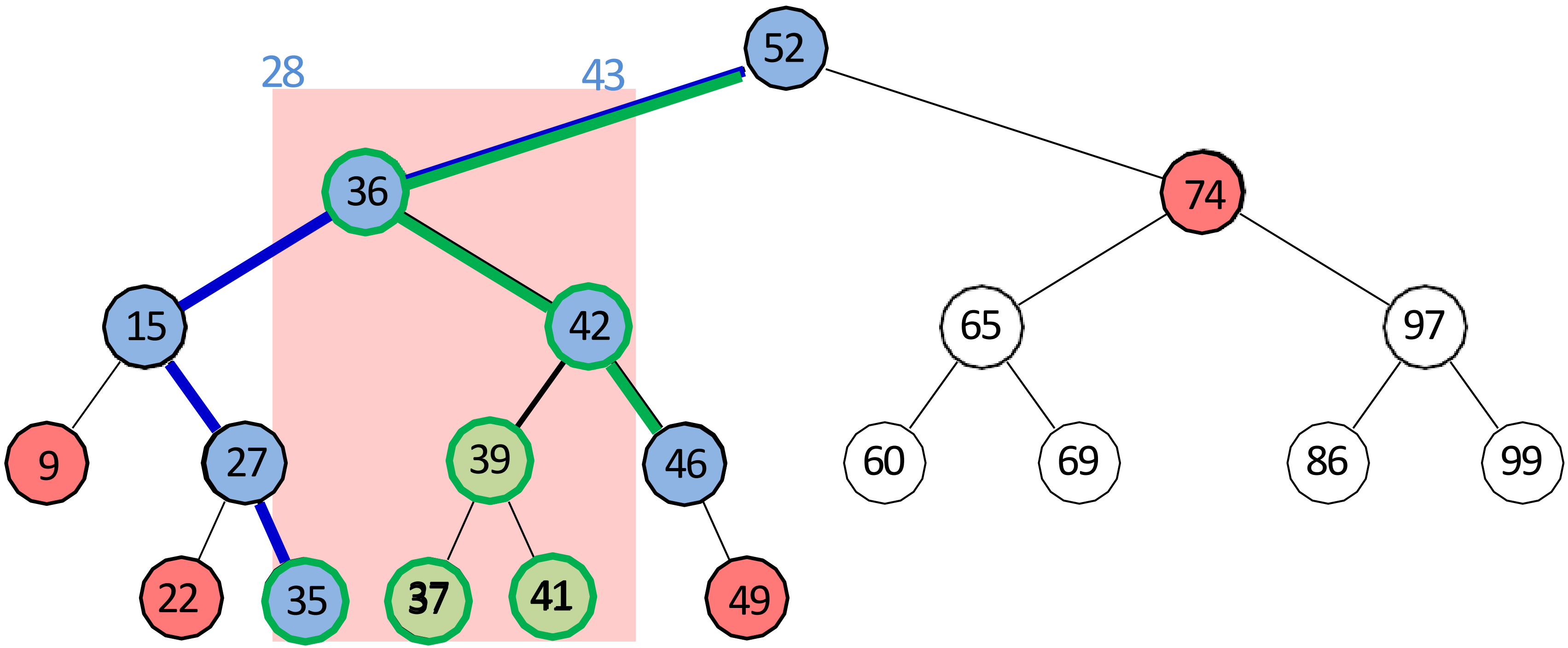
return *BST::RangeSearch-recusive*$(r.right, k_1, k_2)$

if $r.key > k_2$ **then**

return *BST-RangeSearch-recursive*$(r.left, k_1, k_2)$

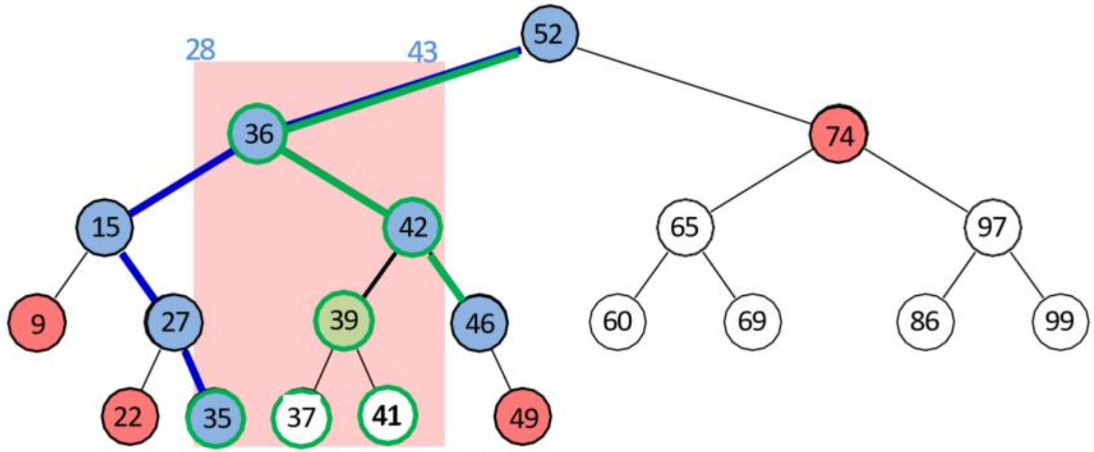
- Keys returned in sorted order

Modified BST Range Search



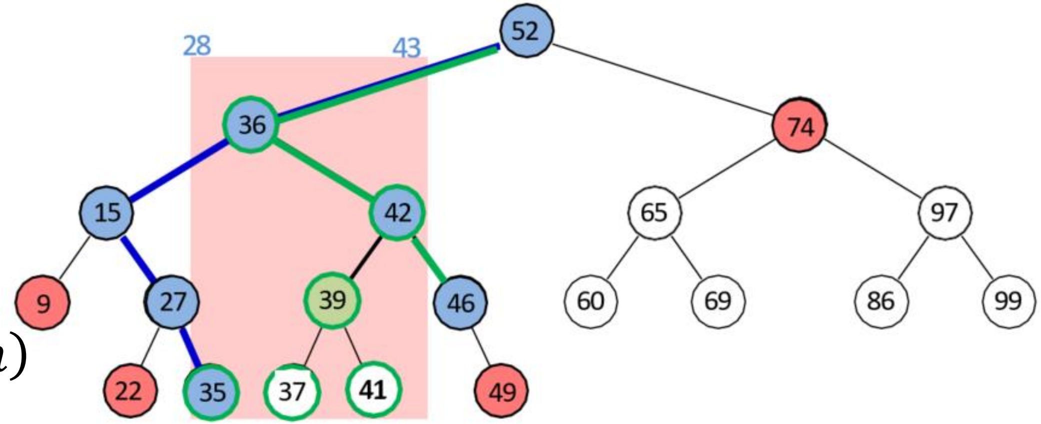
- Search for left boundary k_1 : this gives path ***P_1***
- Search for right boundary k_2 : this gives path ***P_2***
- Boundary (blue nodes) are exactly all the nodes on paths ***P_1*** and ***P_2***
- Nodes are partitioned into three groups: boundary, outside, inside

Modified BST Range Search



- Boundary nodes: nodes in P_1 and P_2
    - check if boundary nodes are in the search range
- Outside nodes: nodes that are left of P_1 or right of P_2
    - outside nodes are not in the search range
    - range search is never called on an outside node
- Inside nodes: nodes that are right of P_1 and left of P_2
    - **we will stop the search at the topmost inside node**
    - all descendants of such node are in the range, just report them without search
    - this is not more efficient for BST range search, but will be efficient when we move to 2D search in ***range trees***

Modified BST Range Search Analysis



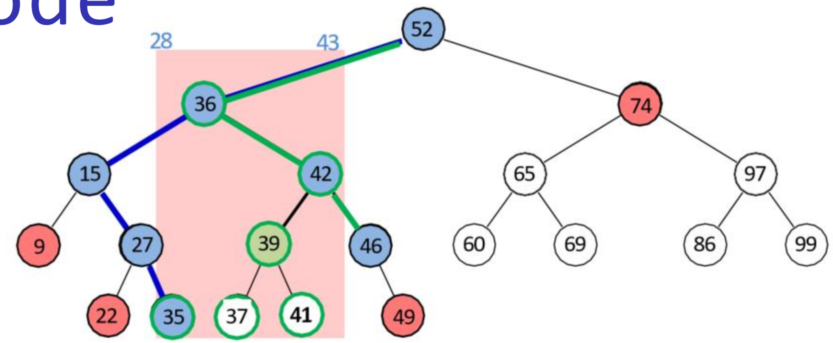
- Assume balanced BST
- Running time consists of
  1. search for path ***P*₁**
     - $O(\log n)$
  2. search for path ***P*₂** is $O(\log n)$
     - $O(\log n)$
  3. check if boundary nodes in the range
     - $O(1)$ at each boundary node, there are $O(\log n)$ of them, $O(\log n)$ total time
  4. spend $O(1)$ at each topmost inside node
     - since each topmost inside node is a child of boundary node, there are at most $O(\log n)$ topmost inside nodes, so total time $O(\log n)$
  5. report descendants in subtrees of all topmost inside nodes
     - topmost nodes are disjoint, so #descendants for inside topmost nodes is at most s, output size

$$\sum_{\substack{\text{topmost inside}\\ \text{node } v}} \#\text{descendants of } v \le s$$

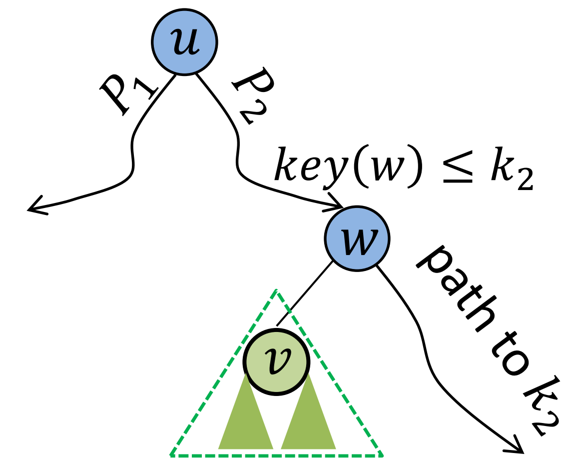
- Total time $O(s + \log n)$

How to Find Top Inside Node

- v is a top inside node if
  - v is not is in P_1 or P_2
  - parent of v is in P_1 or P_2 (but not both)
  - if parent is in P_1, then v is right child
  - if parent is in P_2, then v is left child



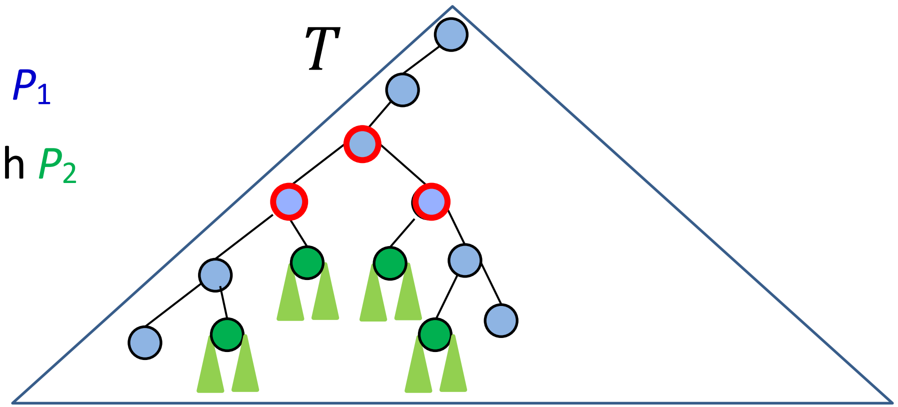
$$k_1 \le key(u) < k_2$$



$$k_1 \le key(u) < \text{ everything} < key(w) \le k_2$$

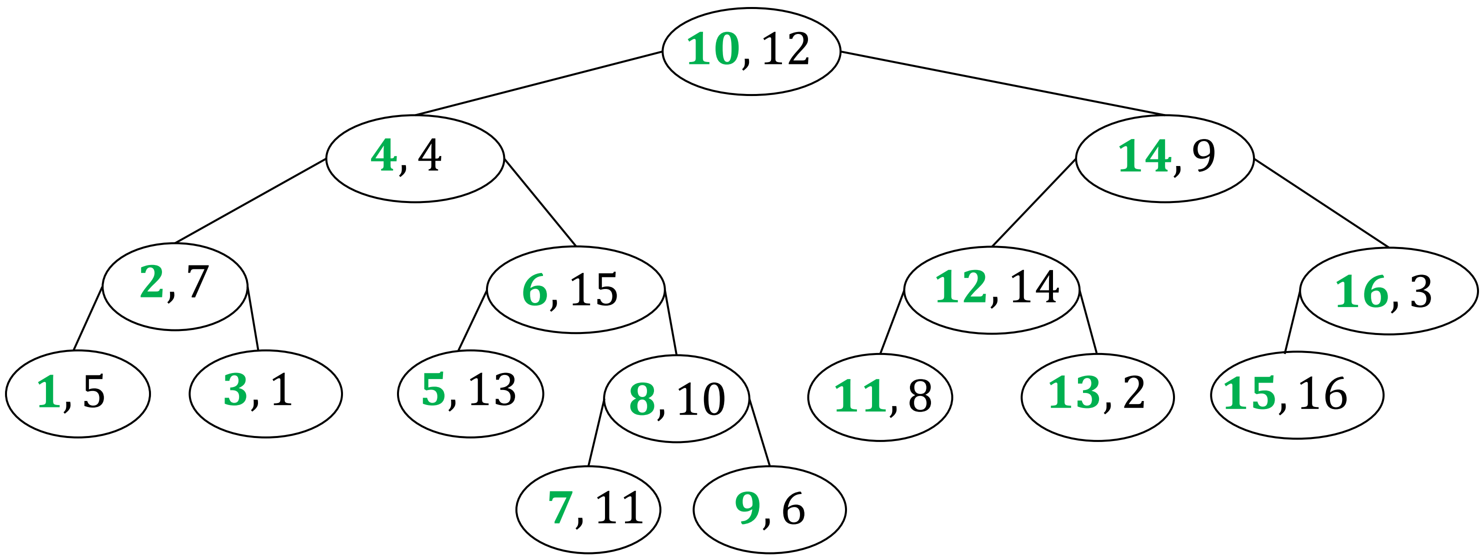
- Thus for each top inside node can report all descendants, no need for search
  - BST range search does not become not faster overall, but top inside nodes are important for $2d$ range search efficiency
  - also important if need to just count the number of points in the search range

Modified BST Range Search Summary



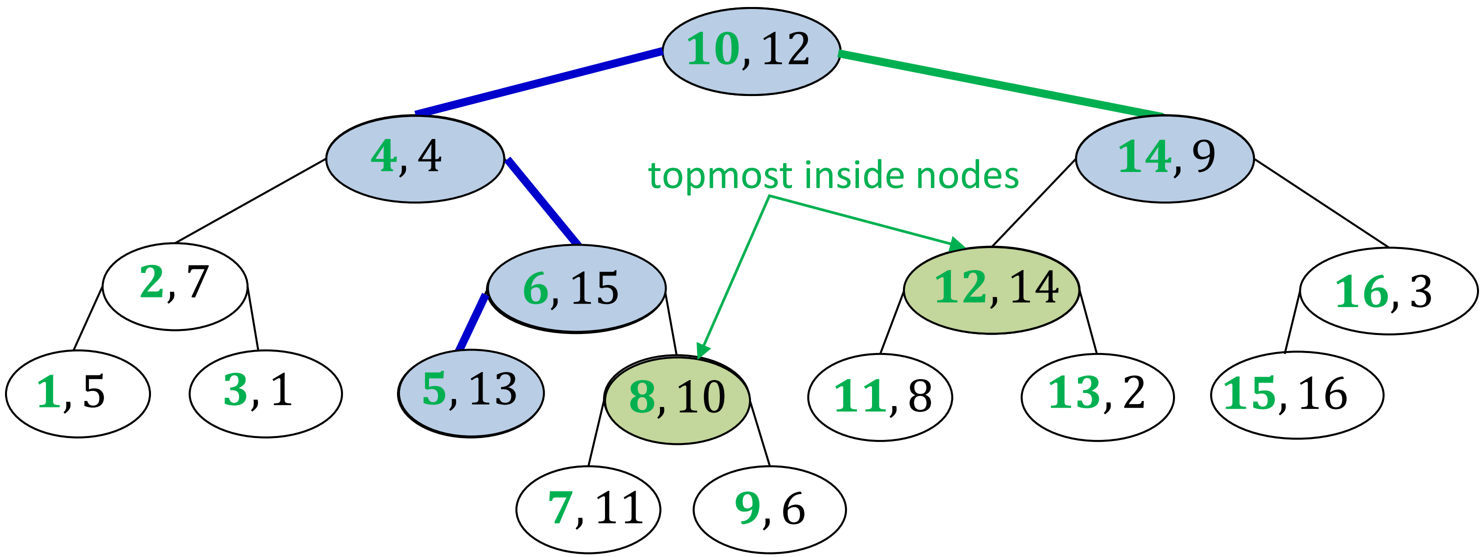
- Search for k_1: this gives left boundary path P_1
- Search for k_2: this gives right boundary path P_2
- Find all topmost inside nodes
  - not in P_1 or P_2
  - left children of nodes in P_2
  - right children of nodes in P_1
- Inside node (which is not a topmost inside) is in a subtree of some topmost inside node
- Set of inside nodes = union disjoint subtrees rooted at topmost inside nodes
- To output nodes in the search range
  - test each node in P_1 , P_2 and report if in range
  - go over all topmost inside nodes and report all nodes in their subtree

2D Range Tree Motivation



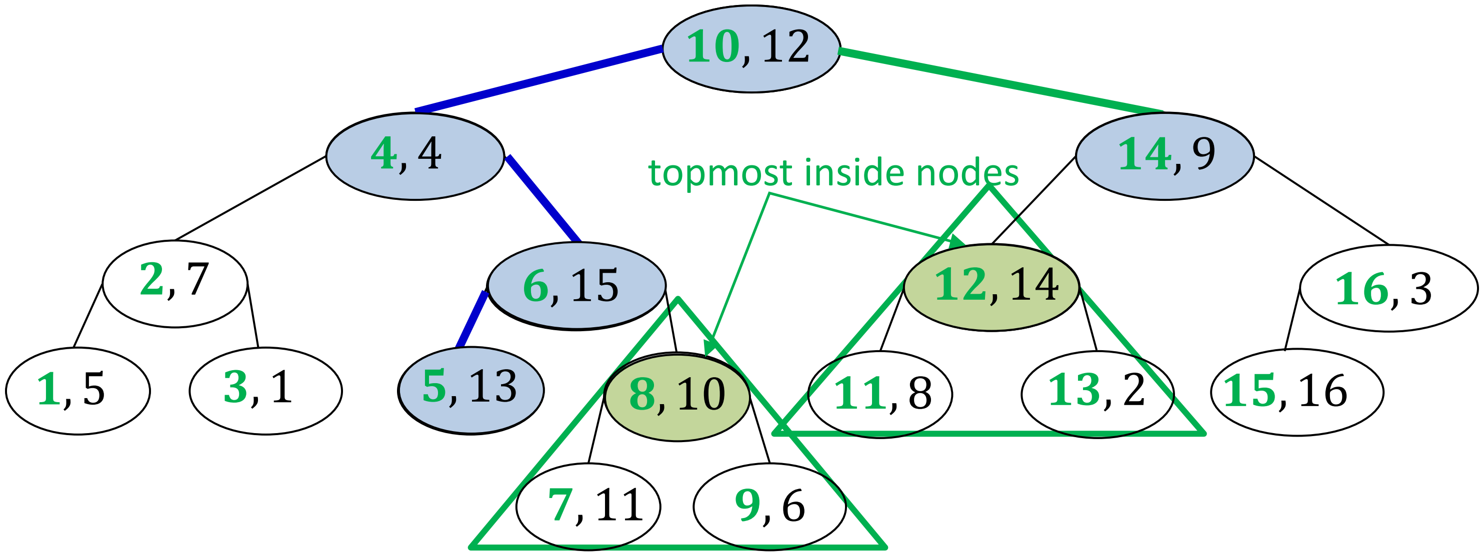
- Have a set of 2D points
  - $S = \{(1,5), (2,7), (3,1), (4,4), (5,13), (6,15)(7,11), (8,10), (9,6), (10,12), (11,8), (12,14), (13,2), (14,9), (15,16), (16,3)\}$
- Example of 2D range search
- *BST-RangeSearch*$(T, 5, 14, 5, 9)$
  - find all points with $5 \leq x \leq 14$ and $5 \leq y \leq 9$
- Construct BST with x-coordinate key
  - recall that points are in general positon, so all x-keys are distinct
    - for any (x_1, y_1) and (x_2, y_2) in our set of points, $x_1 \neq x_2$
  - can search efficiently based only on x-coordinate

2D Range Tree Motivation



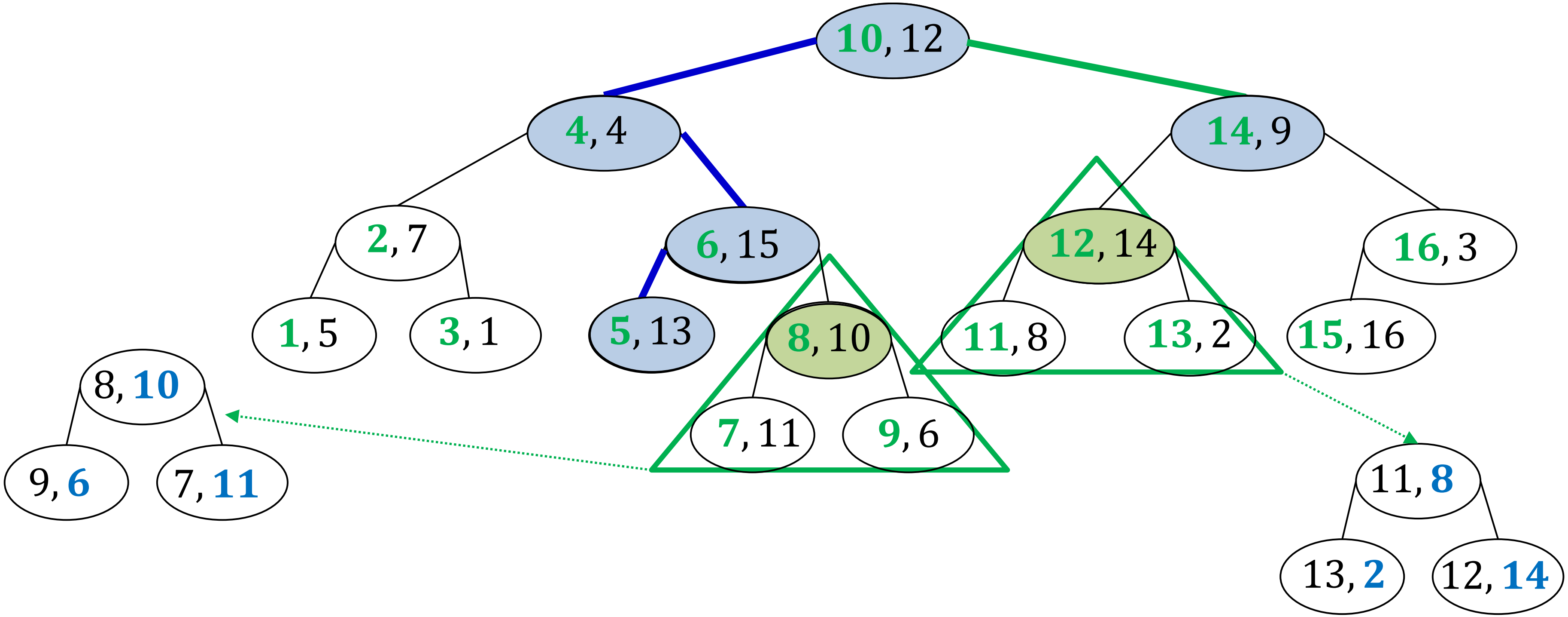
- Consider $2D$ range search *BST-RangeSearch*$(T, 5, 14, 5, 9)$
- First perform *BST-RangeSearch*$(T, 5, 14)$
  - let A be the set of nodes *BST-RangeSearch*$(T, 5, 14)$ returns
    - $A = \{(10{,}12), (6{,}15), (5{,}13), (14{,}9), (8{,}10), (7{,}11), (9{,}6), (12{,}14), (11{,}8), (13{,}2)\}$
  - let B be the set of nodes *BST-RangeSearch*$(T, 5, 14, 5, 9)$ should return
    - $B \subseteq A$
  - Need to go over all nodes in A and check if their y-coordinate is in valid range, $O(|A|)$
    - could be very inefficient
    - for example, $|A|$ can be, say $\Theta(n)$ and $|B|$ could be $O(1)$
      - $O(n)$, as bad as exhaustive search and worse than kd-trees search, $O(|B| + \sqrt{n})$

2D Range Tree Motivation



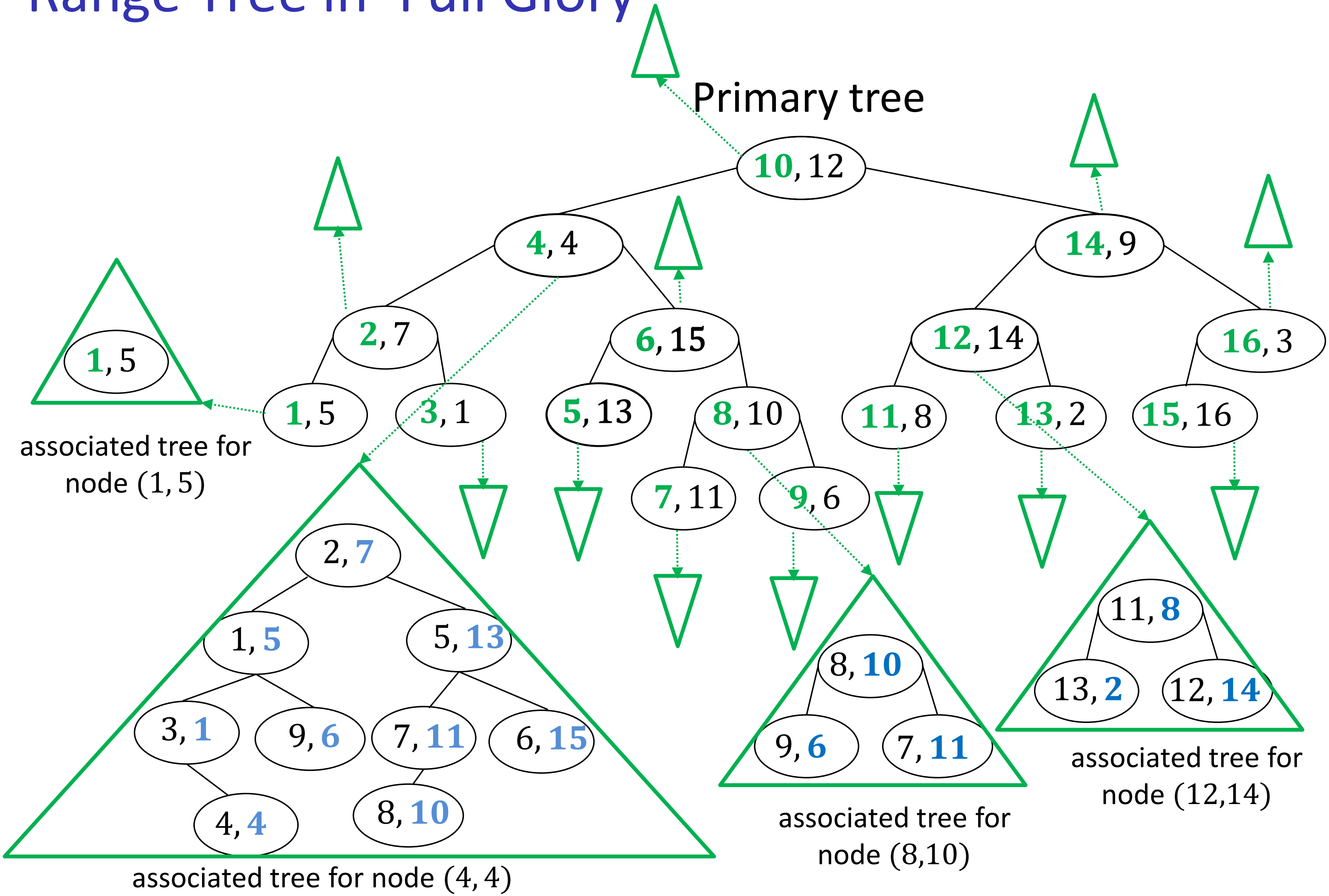
- Consider $2D$ range search *BST-RangeSearch*$(T, 5, 14, 5, 9)$
- First perform only **partial** *BST-RangeSearch*$(T, 5, 14)$
  - find boundary and topmost inside nodes, takes $O(\log n)$ time
- Next
  - for boundary nodes, check if **both** x and y coordinates are in the range, takes $O(\log n)$ time as there are $O(\log n)$ boundary nodes
  - inside nodes are stored in $O(\log n)$ subtrees, with a topmost inside node as a root of each subtree
    - if we could search these subtrees, time would be very efficient
    - however these subtrees do not support efficient search by y coordinate

2D Range Tree Motivation

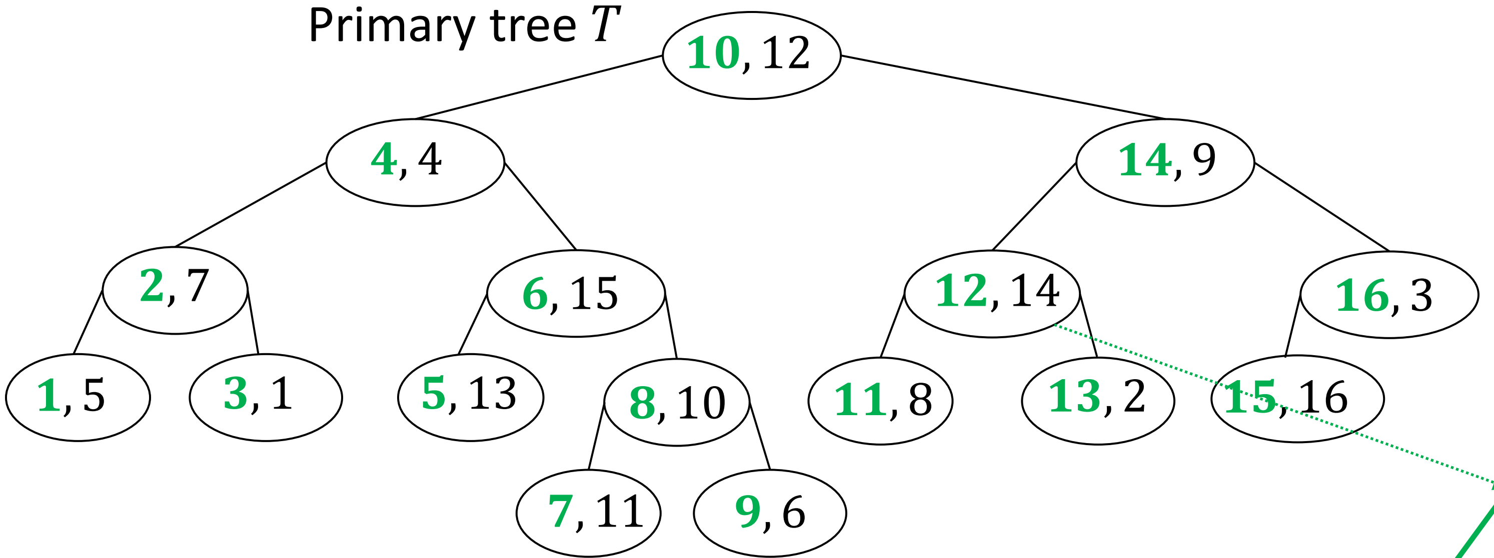


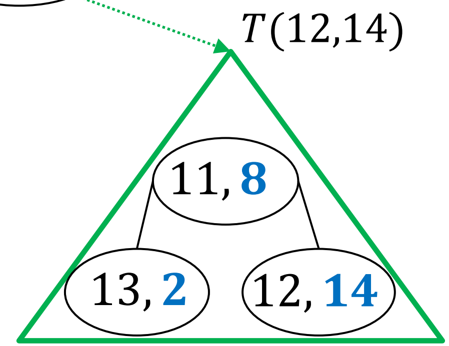
- Need to search subtrees by y-coordinate, but they are x-coordinate based
- Brute-force solution
  - create an associate balanced BST tree for each node v
  - stores the same items as the main (primary) subtree rooted at node v
  - but key is y-coordinate

Range Tree in 'Full Glory'



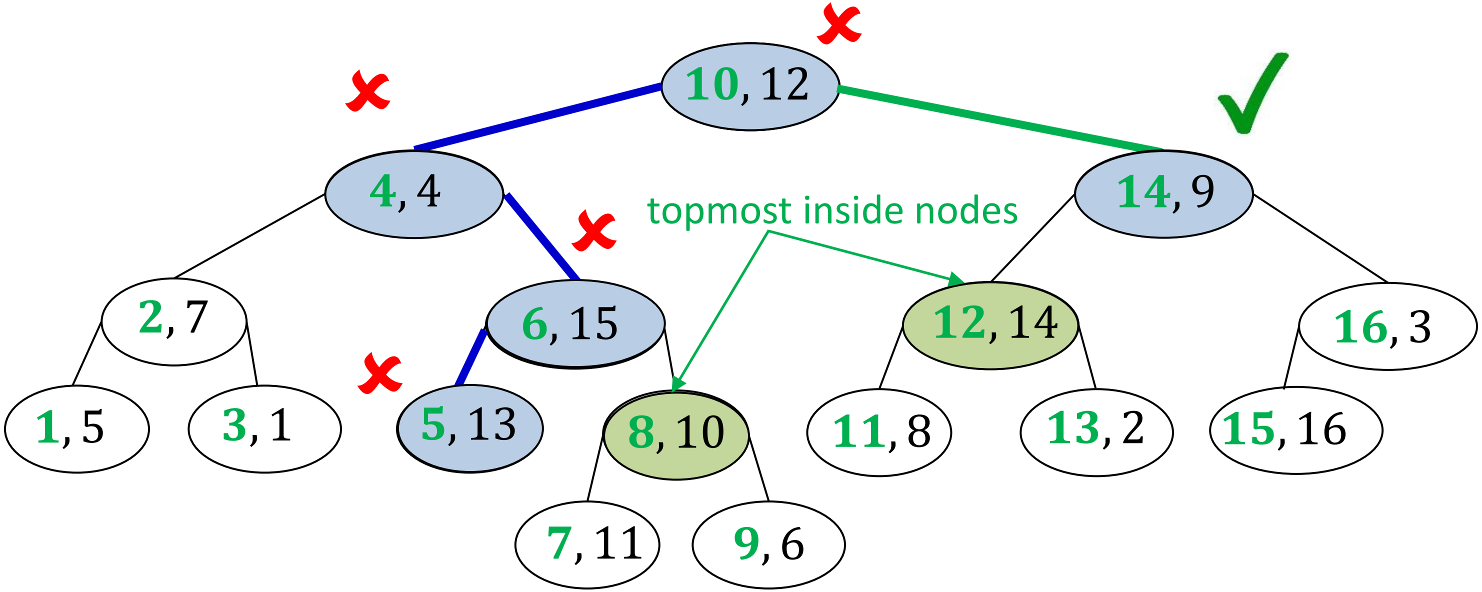
2-dimensional Range Trees Full Definition





- Points $S = \{(x_0, y_0), (x_1, y_1), \dots, (x_{n-1}, y_{n-1})\}$
- Range tree is a tree of trees (a *multi-level* data structure)
  - Primary structure
    - balanced BST T storing S and uses x-coordinates as keys
    - assume T is balanced, so height is $O(\log n)$
  - Each node v of T stores an associated tree $T(v)$, which is a balanced BST
    - let $S(v)$ be all descendants of v in T, including v
    - $T(v)$ stores $S(v)$ in BST, using y-coordinates as key
      - note that v is not necessarily the root of $T(v)$

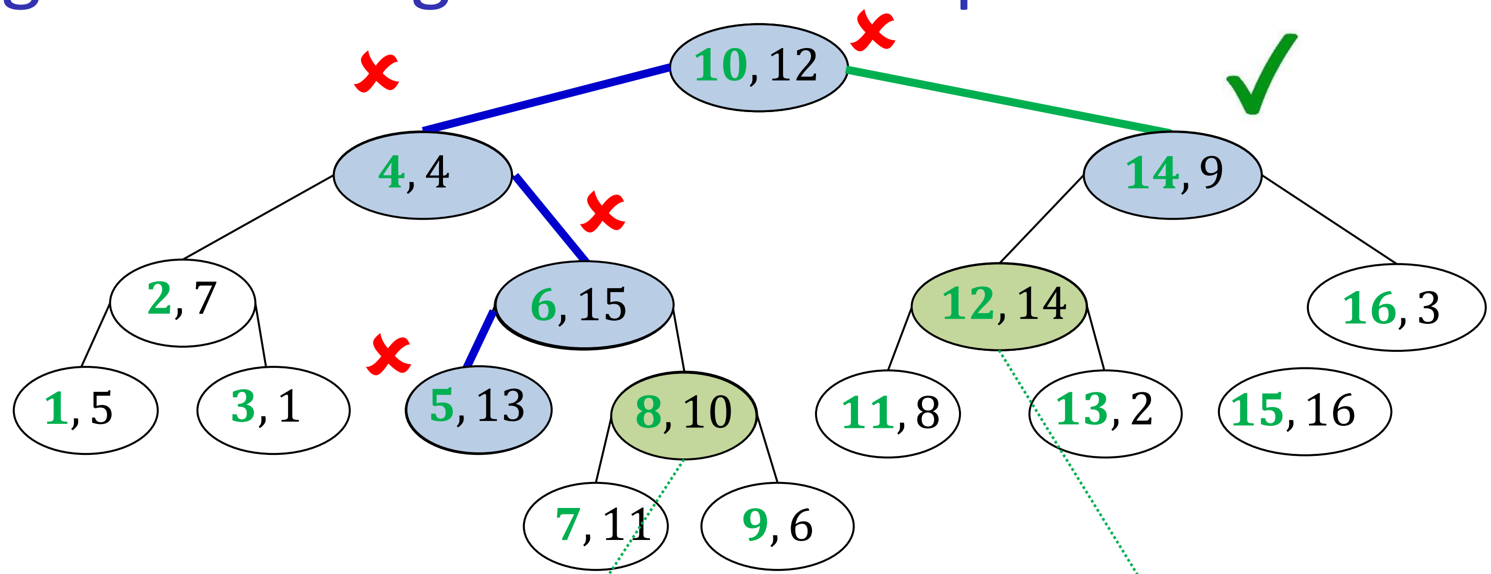
Range search in 2D Range Tree Overview



- *RangeTree::RangeSearch*(T, x_1, x_2, y_1, y_2)
  - *RangeTree::RangeSearch*$(T, 5, 14, 5, 9)$

1. Perform *modified BST-RangeSearch*$(T, 5, 14)$
   - find boundary and topmost inside nodes, but do not go through the inside subtrees
   - modified version takes $O(\log n)$ time
     - does not visit all the nodes in valid range for *BST-RangeSearch*$(T, 5, 14)$
2. Check if boundary nodes have valid x-coordinate **and** valid y-coordinate
3. For every topmost inside node v, search in associated tree *BST::RangeSearch*$(T(v), 5, 9)$

Range Tree Range Search Example Finished

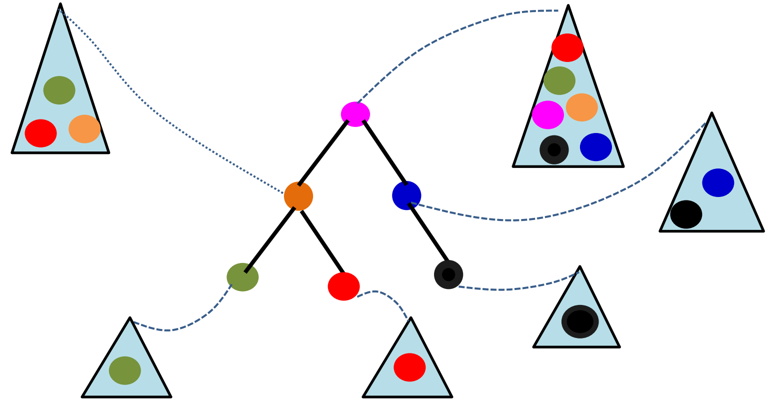


- *RangeTree::RangeSearch*$(T, 5, 14, 5, 9)$
- For every topmost inside node v, search in associated tree *BST-RangeSearch*$(T(v), 5, 9)$

BST-rangeSearch$(T(8{,}10), 5{,}9)$

BST-RangeSearch$(T(12{,}14), 5{,}9)$

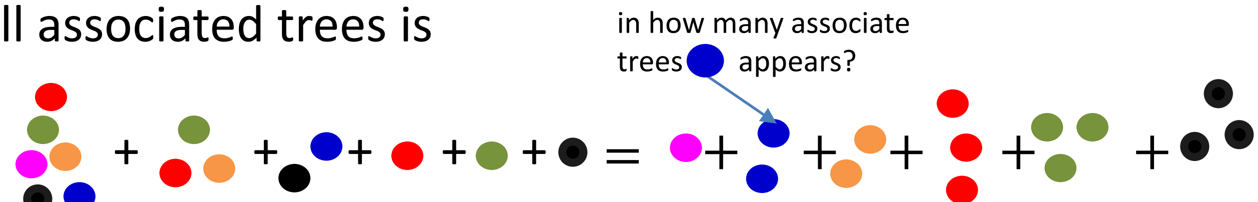
Range Tree Space Analysis



- Primary tree T uses $O(n)$ space
- For each v, associated tree $T(v)$ uses $O(|T(v)|)$ space

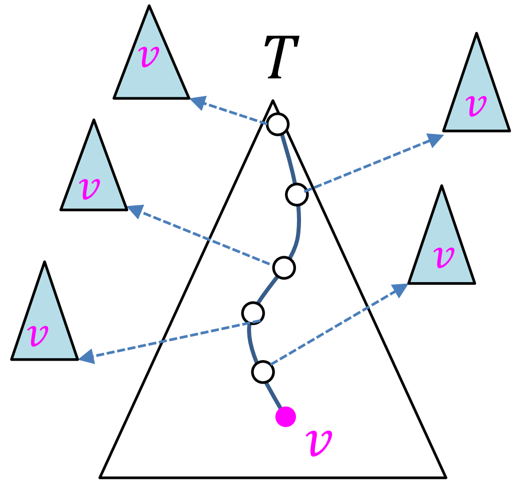
- Space for all associated trees is

$$\sum_{v \in T} |T(v)| =$$



$$= \sum_{v \in T} \underbrace{\text{\#of ancestors of } v}_{\le c \log n}$$

$$\le \sum_{v \in T} c \log n = cn \log n$$



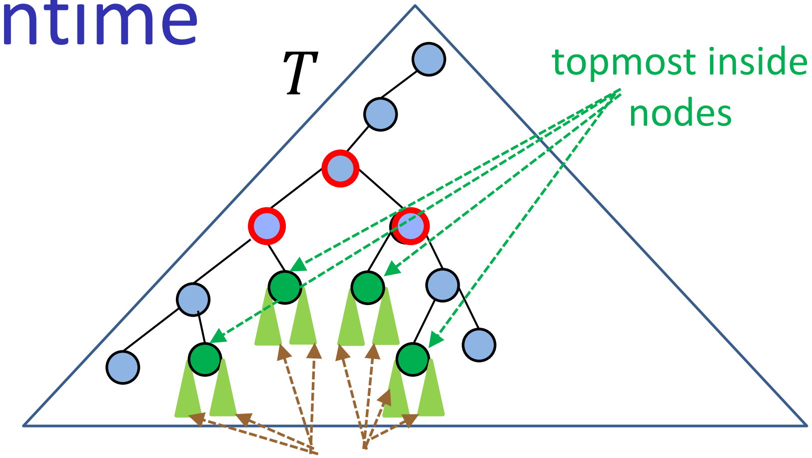
#of ancestors of v

- Space is $O(n \log n)$
  - in the worst case, have $n/2$ leaves at the last level, and space needed is $\Theta(n \log n)$

Range Trees: Dictionary Operations

- Search(x, y)
  - search by x coordinate in the primary tree T
- Insert(x, y)
  - first, insert point by x-coordinate into the primary tree T
  - then walk up to root and insert point by y-coordinate in *all* $T(v)$ of nodes v on path to root
- Delete
  - analogous to insertion
- **Problem**
  - want binary search trees to be balanced
  - if we use AVL-trees, it makes insert/delete very slow
    - rotation at v changes $S(v)$ and hence requires re-build of $T(v)$
  - instead of rotations, can allow certain imbalance, rebuild entire subtree if violated
    - no details

Range Trees: Range Search Runtime



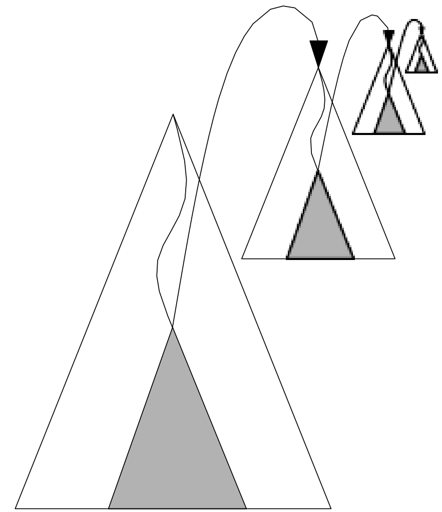
- Find boundary nodes in the primary tree and check if keys are in the range
  - $O(\log n)$
- Find topmost inside nodes in primary tree
  - $O(\log n)$
- For each topmost inside node v, perform range search for y-range in associate tree
  - $O(\log n)$ topmost inside nodes
  - let s_v be #items returned for the subtree of topmost node v
  - running time for one search is $O(\log n + s_v)$

$$\sum_{\substack{\text{topmost inside} \\ \text{node } v}} c(\log n + s_v) = \underbrace{\sum_{\substack{\text{topmost inside} \\ \text{node } v}} c \log n}_{O(\log^2 n)} + \underbrace{\sum_{\substack{\text{topmost inside} \\ \text{node } v}} c s_v}_{\leq cs}$$

- Time for range search in range tree: $O(s + \log^2 n)$
  - can make this even more efficient, but this is beyond the scope of the course

Range Trees: Higher Dimensions

- Range trees can be generalized to d -dimensional space
  - space $O(n\,(\log n)^{d-1})$
  - construction time $O(n\,(\log n)^{d})$
  - range search time $O(s + (\log n)^{d})$
- Note: d is considered to be a constant
- Space-time tradeoff compared to kd trees



Outline

- Range-Searching in Dictionaries for Points
    - Range Search
    - Multi-Dimensional Data
    - Quadtrees
    - kd-Trees
    - Range Trees
    - Conclusion

Range Search Data Structures Summary

- Quadtrees
  - simple, easy to implement insert/delete (i.e. dynamic set of points)
  - work well only if points evenly distributed
  - wastes space for higher dimensions
  - convention: points on split lines belong to the right/top side
- kd-trees
  - linear space
  - range search is $O(s + \sqrt{n})$
  - inserts/deletes destroy balance and range search time
    - fix with occasional rebuilt
  - convention: points on split lines belong to the right/top side
- Range trees
  - fastest range search $O(s + \log^2 n)$
  - wastes some space
  - insert and delete destroy balance, but can fix this with occasional rebuilt