VE281

Data Structures and Algorithms

Trie, M-way Search Trees, and 2-3 Trees

Announcement

- Programming Project Two will be announced by tonight.

  - Due in 15 days by 11:59 pm on Nov. 30th , 2012.

  - It is related to the materials taught in this and the next lecture, i.e., **2-3 tree**.
    - All the slides have been put online.

  - Not easy. Start early!

Review

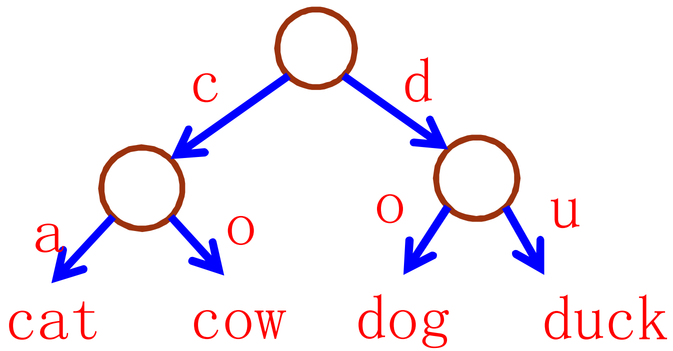
- Priority Queue
  - **`getMin`**, **`enqueue`**, **`dequeueMin`**
  - Implemented as a binary heap
- Min Heap and Its Operations
  - Properties
  - **`enqueue`**: Percolate up; Complexity: $O(\log n)$
  - **`dequeueMin`**: Percolate down; Complexity: $O(\log n)$
- Initializing a Min Heap
  - Complexity $O(n)$

Outline

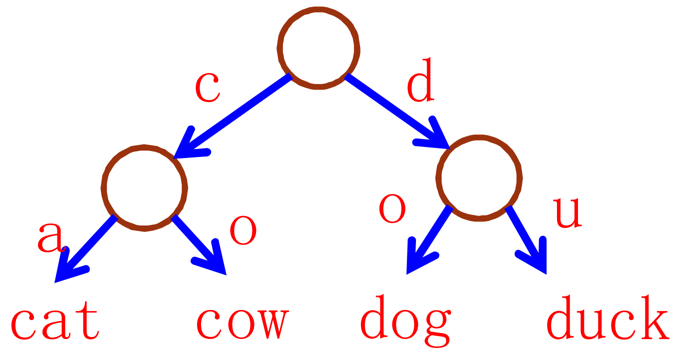
- Trie
- M-way Search Tree
- 2-3 Tree: Basics
- 2-3 Tree: Insertion

Trie

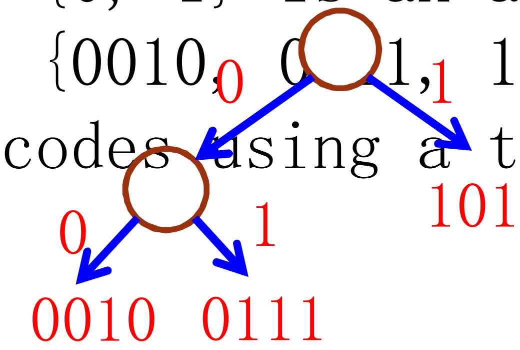
- A trie is a tree that uses parts of the key, as opposed to the whole key, to perform search.
- A trie stores data records only in **leaf** nodes. Internal nodes serve as placeholders to direct the search process.



Trie

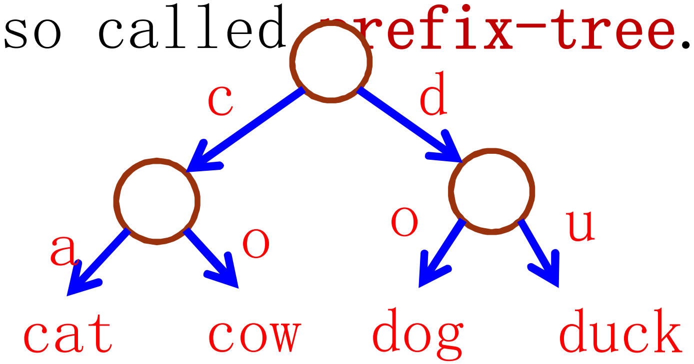


- Trie usually is used to store a set of strings from an alphabet.
  - The alphabet is in the general sense, not necessarily the English alphabet.
- For example, {0, 1} is an alphabet for binary codes {0010, 0111, 101}. We can store these three codes using a trie.



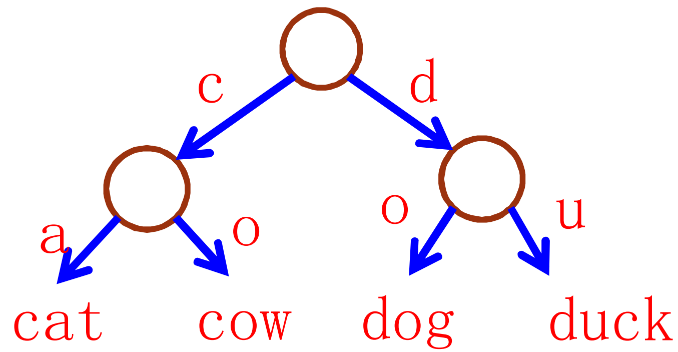
Trie

- Each edge of the trie is labeled with symbols from the alphabet.
  - The labels can be stored either at the children nodes or at the parent node.
- Labels of edges on the path from the root to any leaf in the trie forms a **prefix** of a string in that leaf.
  - Trie is also called **prefix-tree**.



Trie

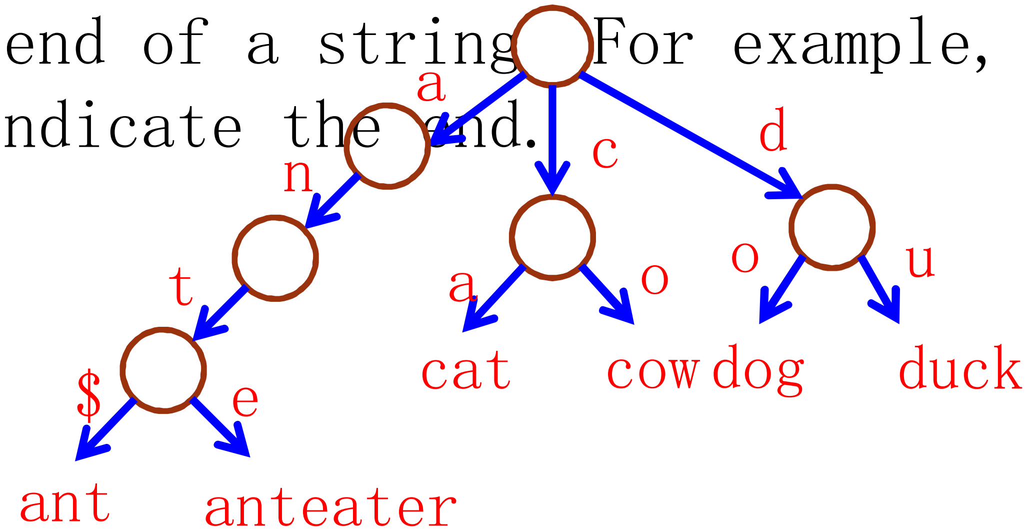
- The most significant symbol in a string determines the branch direction at the root.
- Each internal node is a "**branch**" point.
- As long as there is only one key in a branch, we do not need any further internal node below that branch; we can put the word directly as the leaf of that branch.



Trie

Implementation Issue

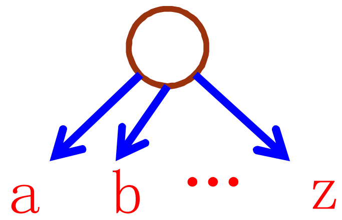
- Sometimes, a string in the set is exactly a **prefix** of another string.
  - For example, "ant" is a prefix of "anteater".
  - How can we make "ant" as a leaf in the trie?
- We add a symbol to the alphabet to indicate the end of a string. For example, use "$" to indicate the end.



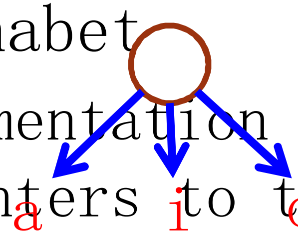
Trie

Implementation Issue

- We can keep an array of pointers in a node, which corresponds to **all** possible symbols in the alphabet.



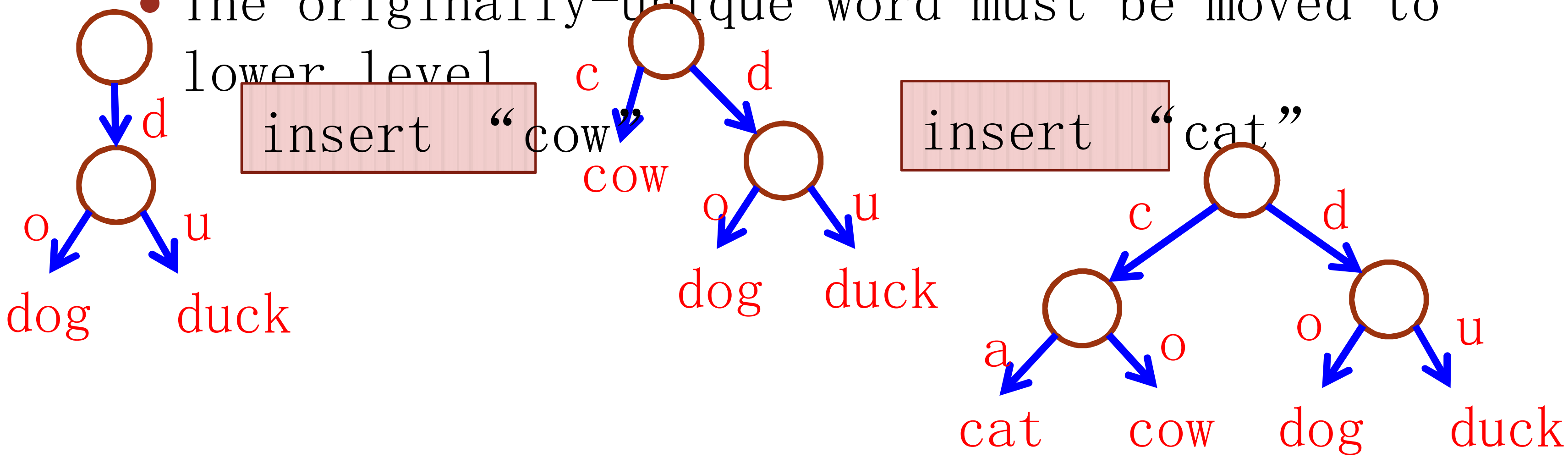
- However, most internal nodes have branches to only a small fraction of the possible symbols in the alphabet.
  - An alternate implementation is to store a linked list of pointers to the child nodes.



Trie

Insertion

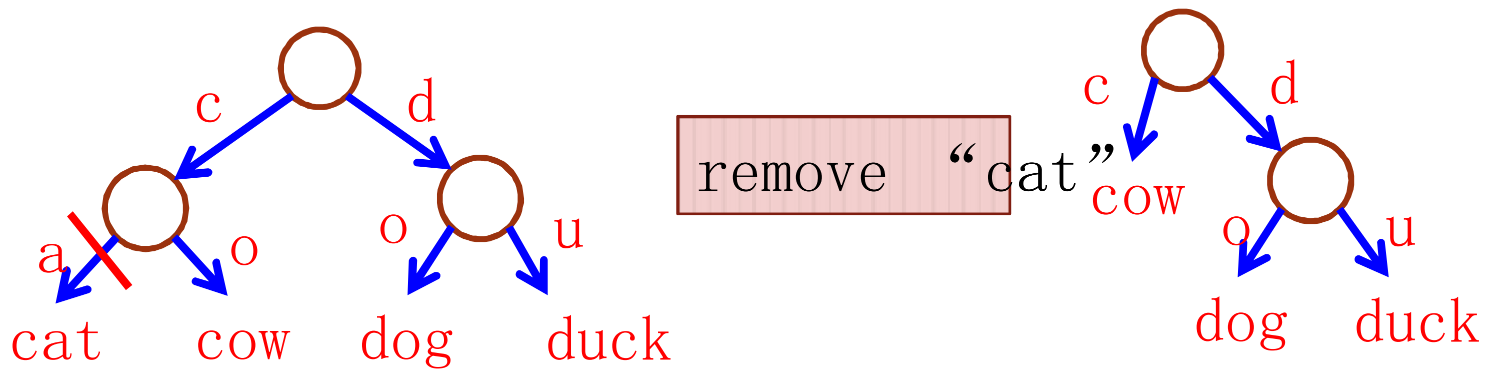
- Follow the search path, starting from the root.
- If a new branch is needed, add it.
- When the search leads to a leaf, a conflict occurs. We need to branch.
  - The originally-unique word must be moved to lower level.



Trie

Removal

- The key to be removed is always at the leaf.
- After deleting the key, if the parent of that key now has only one child *C*, remove the parent node and move key *C* one level up.
  - If key *C* is the only child of its new parent, repeat the above procedure again.



Time Complexity of Trie

- In the worst case, inserting or finding a key that consists of k symbols is $O(k)$.
  - This does not depend on the number of keys N.
  - Comparison: stroring 32 integers in the range [0, 127] using a trie versus using a BST. What are heights in the worst case?

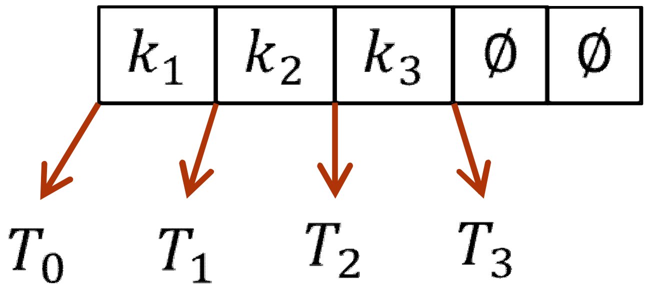
- Sometimes we can access records even faster.
  - A key is stored at the depth which is enough to distinguish it with others.
  - For example, in dictionary, we can find the word "qwerty" with just "qw".

Outline

- Trie
- M-way Search Tree
- 2-3 Tree: Basics
- 2-3 Tree: Insertion

M-Way Search Trees

- M-way search tree is a generalization of binary search tree.
- Every node in the M-way search tree contains $n-1$ keys and n subtrees, where $2 \le n \le M$.
  - Example: an internal node of a 6-way search tree

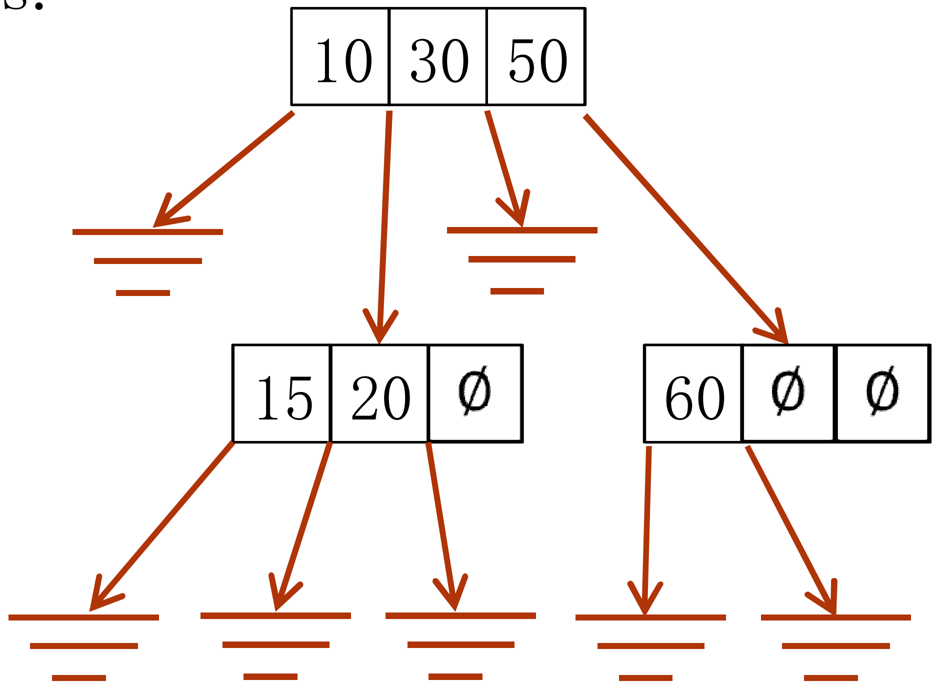


- Suppose the $n-1$ keys are $k_1, k_2, \ldots, k_{n-1}$ and the n subtrees are $T_0, T_1, \ldots, T_{n-1}$.
  - All the keys in subtree T_{i-1} are smaller than k_i.
  - All the keys in subtree T_i are larger than k_i.

M-Way Search Trees

Example

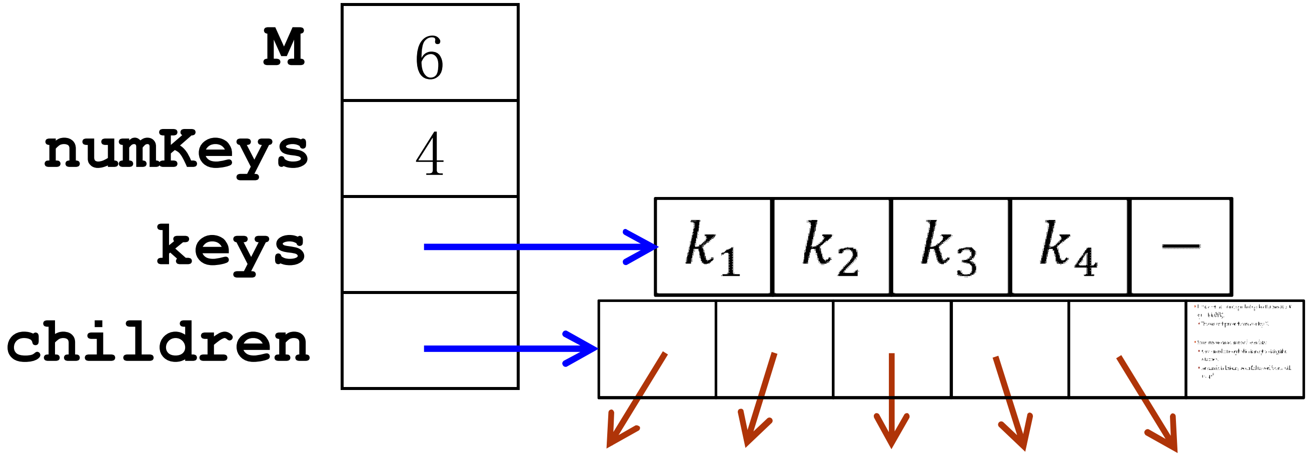
- A 4-way search tree
  - Each node has at most 3 keys and 4 subtrees.
  - However, each node does not need to have 3 keys.



M-Way Search Trees

Representation

```
struct mnode {
  int M;
  int numKeys;
  Key keys[M-1];
  mnode *children[M];
}
```



M-Way Search Trees

Search

- Similar to search on BST with more than one comparison per node.

- Complexity analysis: Consider an M-way search tree with K keys and N nodes.
  - N satisfies $\frac{K}{M-1} \leq N \leq K$.
  - The average height is $\Theta(\log_M N) = \Theta(\log_M K)$.
  - If all nodes have $M-1$ keys, with linear search on each node, the time complexity is $\Theta(M \log_M K)$.
  - With binary search on each node, it takes $\Theta(\log_2 M \log_M K)$ time.

Balanced *M*-Way Search Trees

- M-way search tree is not guaranteed to be "balanced."
  - Its height in the worst case is $\Theta(N)=\Theta(K)$, where N is the number of nodes and K is the number of keys.

- Recall that AVL tree is a balanced binary search tree. We also have "**balanced**" M-way search trees.
  - They need extra operations to maintain balance.

- We will study a balanced 3-way tree: **2-3 tree**.

Outline

- Trie
- M-way Search Tree
- 2-3 Tree: Basics
- 2-3 Tree: Insertion

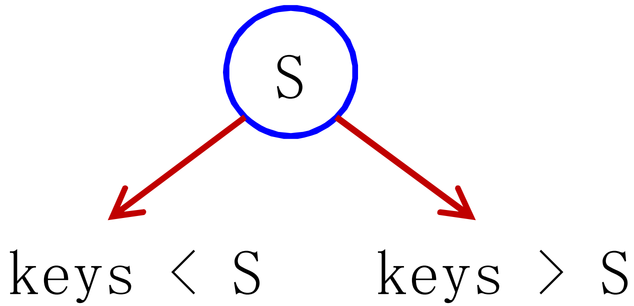
Properties of 2-3 Trees

- It is a 3-way search tree, i.e., each node contains 1 key or 2 keys.
  - A node with 1 key has 2 subtrees. It is called a 2-node.
  - A node with 2 keys has 3 subtrees. It is called a 3-node.

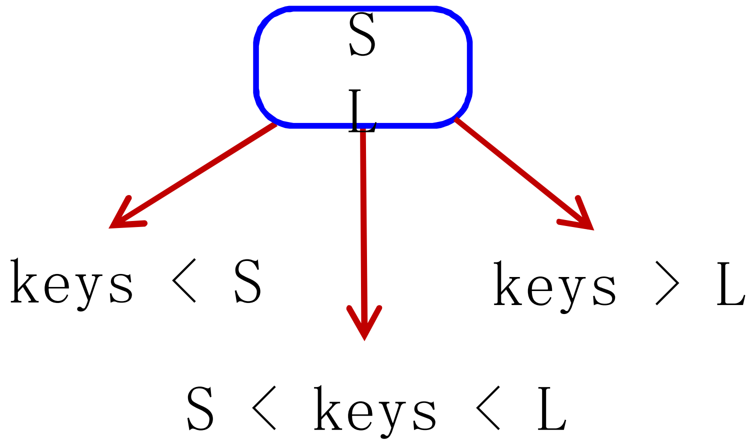
- All leaf nodes (i.e., nodes whose subtrees are all empty) are at the same level.
- The two subtrees of any internal 2-node are non-empty.
- The three subtrees of any internal 3-node are non-empty.

2-Nodes and 3-Nodes

2-Node

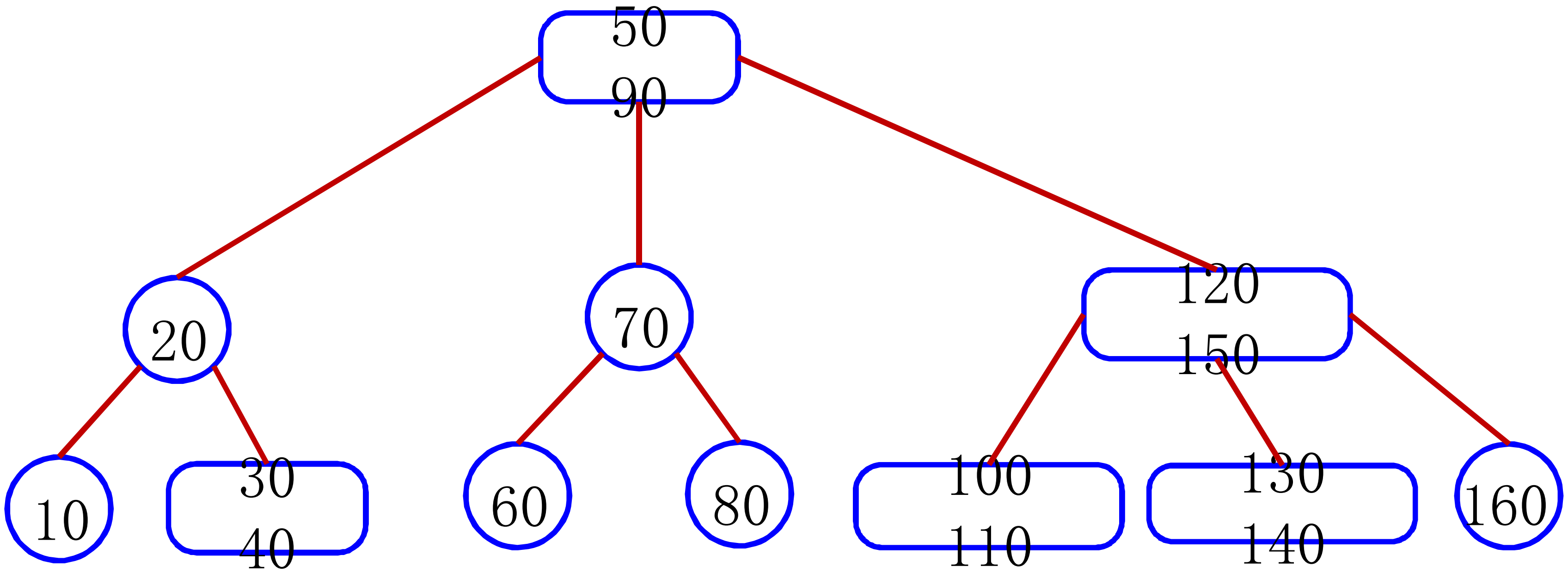


3-Node



2-3 Trees

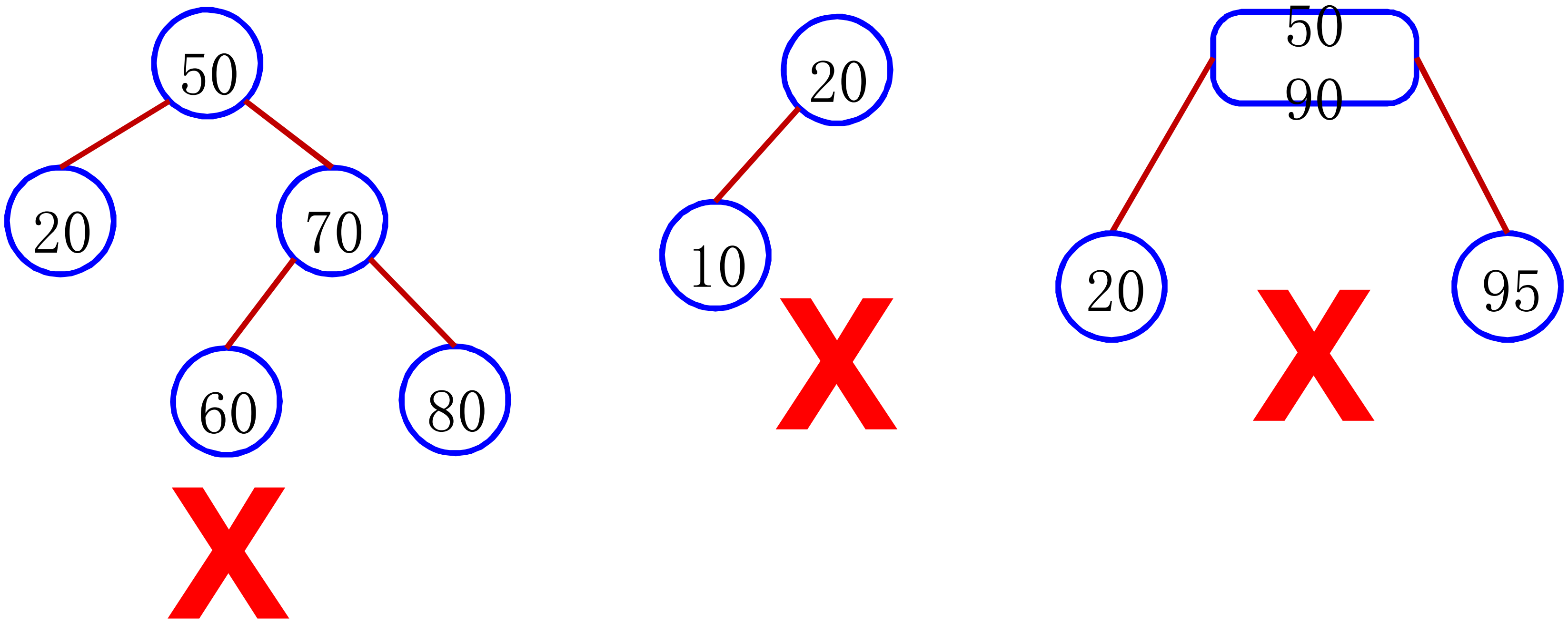
Example



2-3 Trees

Example

- Are they 2-3 trees?



Number of Keys versus Height

- Consider a 2-3 tree of height h.
- When does the tree contain minimum number of keys?
  - All the nodes are 2-nodes.
  - The number of nodes is $\sum_{i=0}^{h} 2^i = 2^{h+1} - 1$.
  - The number of keys is $2^{h+1} - 1$.
- When does the tree contain maximum number of keys?
  - All the nodes are 3-nodes.
  - The number of nodes is $\sum_{i=0}^{h} 3^i = (3^{h+1} - 1)/2$.
  - The number of keys is $3^{h+1} - 1$.
- The height of a 2-3 tree with N keys is $\Theta(\log N)$.

Representation of 2-3 Tree Node

```
struct Node {
  Key lkey, rkey;
  Node *left, *center, *right;
};
```

- Representing both a 2-node and a 3-node.
- For a 2-node, set **rkey = emptyKey** and **right = NULL**

2-3 Trees

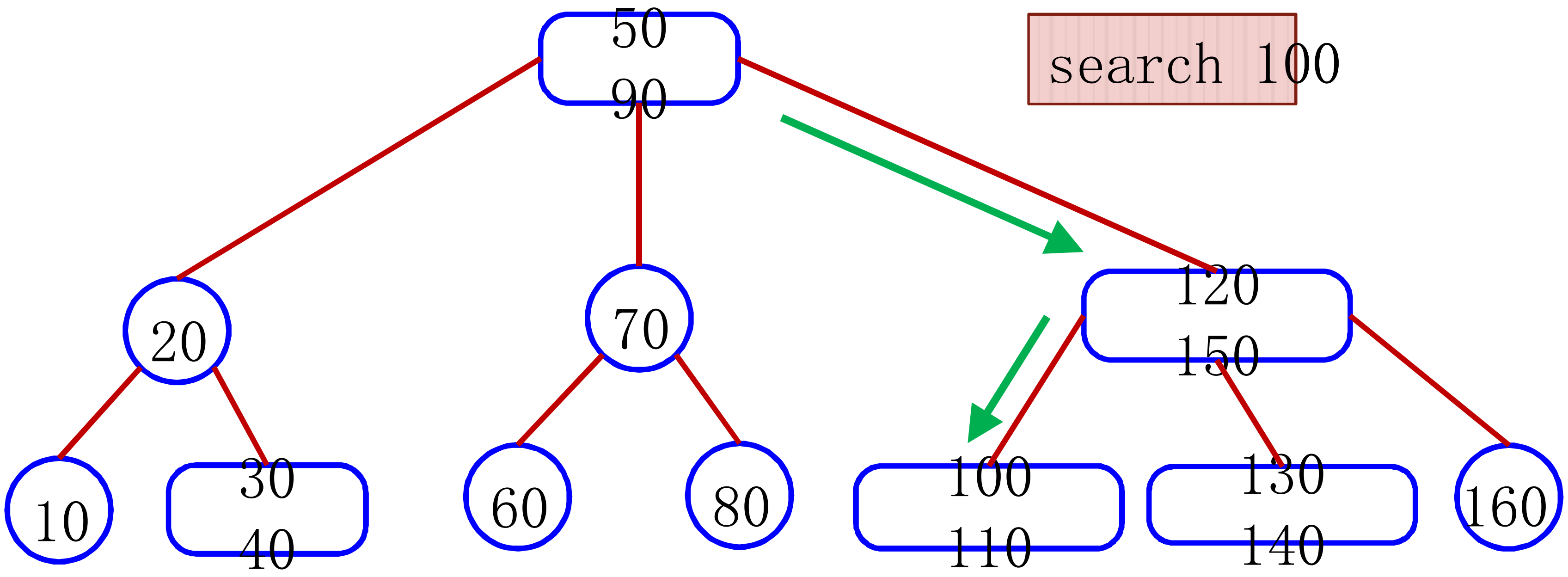
In-Order Traversal

- Visit left subtree
- Visit left key
- Visit center subtree
- If a 3-node
  - Visit right key
  - Visit right subtree

```
void inOrder(Node *node) {
  if(!node) return;
  inOrder(node->left);
  visit(node->lkey);
  inOrder(node->center);
  if(isThreeNode(node)) {
    visit(node->rkey);
    inOrder(node->right);
  }
}
```

2-3 Trees

Search



2-3 Trees

Search

```
Node *search(Node *cur, Key sKey)
// Illustration for the 3-nodes. Need
// to modify this for the 2-nodes.
// EFFECTS: return the node contains sKey
{
  if(!cur) return NULL;
  if(sKey==cur->lkey || sKey==cur->rkey)
    return cur;
  if(sKey < cur->lkey)
    return search(cur->left, sKey);
  if(sKey > cur->rkey)
    return search(cur->right, sKey);
  return search(cur->center, sKey);
}
```

Outline

- Trie
- M-way Search Tree
- 2-3 Tree: Basics
- 2-3 Tree: Insertion

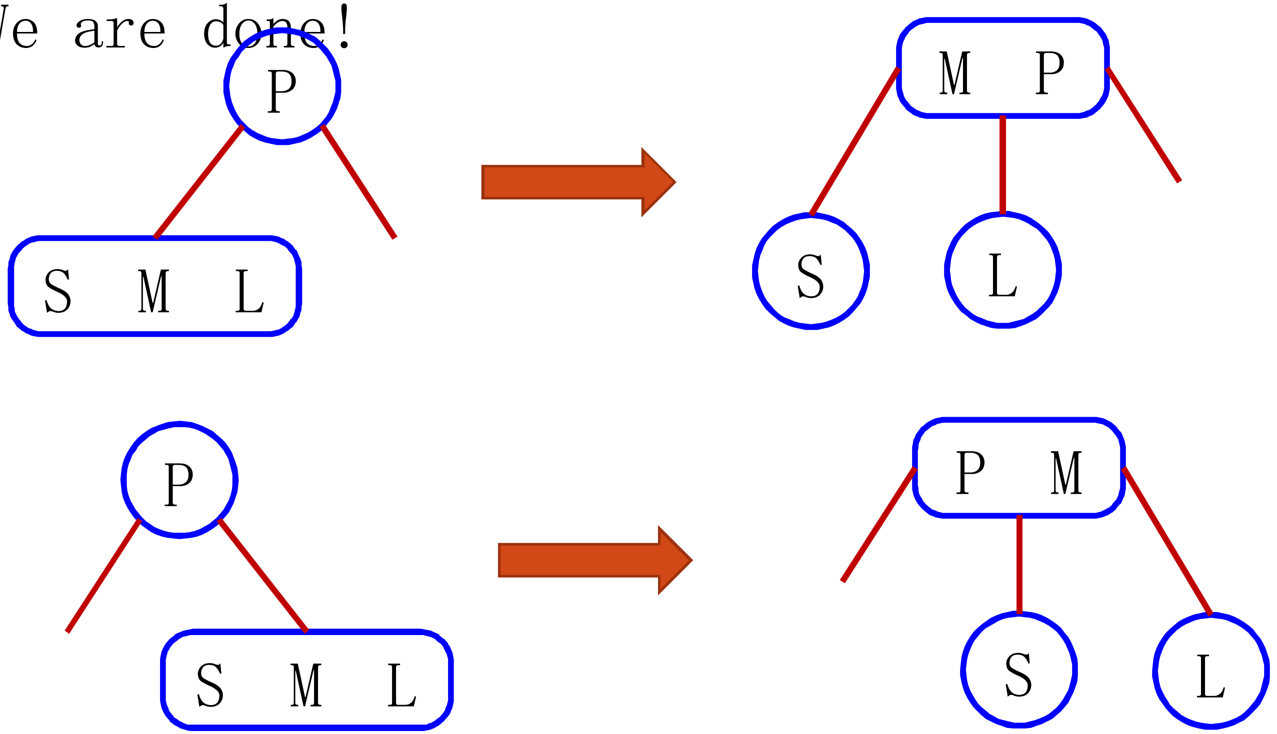
2-3 Trees

Insertion

- Search with the key until you reach a leaf
  - If the leaf is a 2-node, put the key in that node. The leaf now becomes a 3-node.
  - If the leaf is a 3-node, we need to **split** the leaf and move the middle key to the parent.

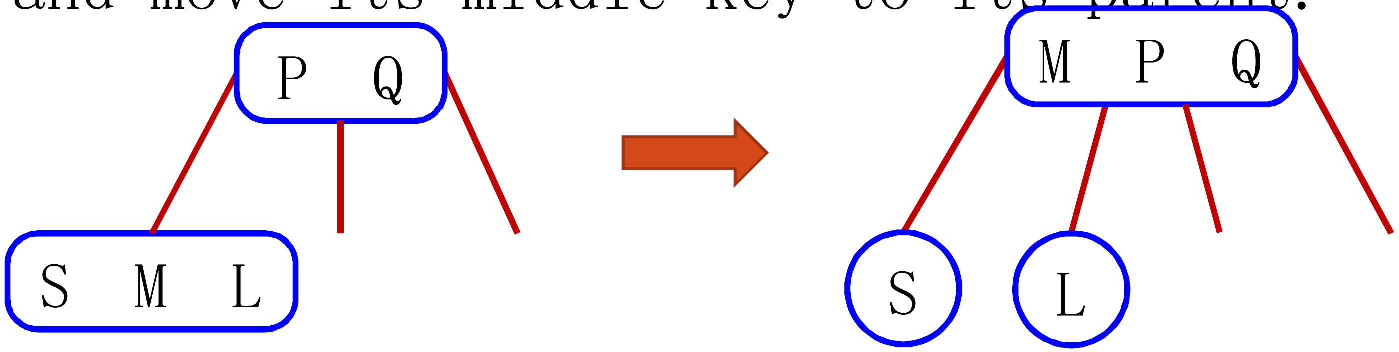
Splitting a Leaf Node

- If the parent is a 2-node, it becomes a 3-node.
  - We are done!

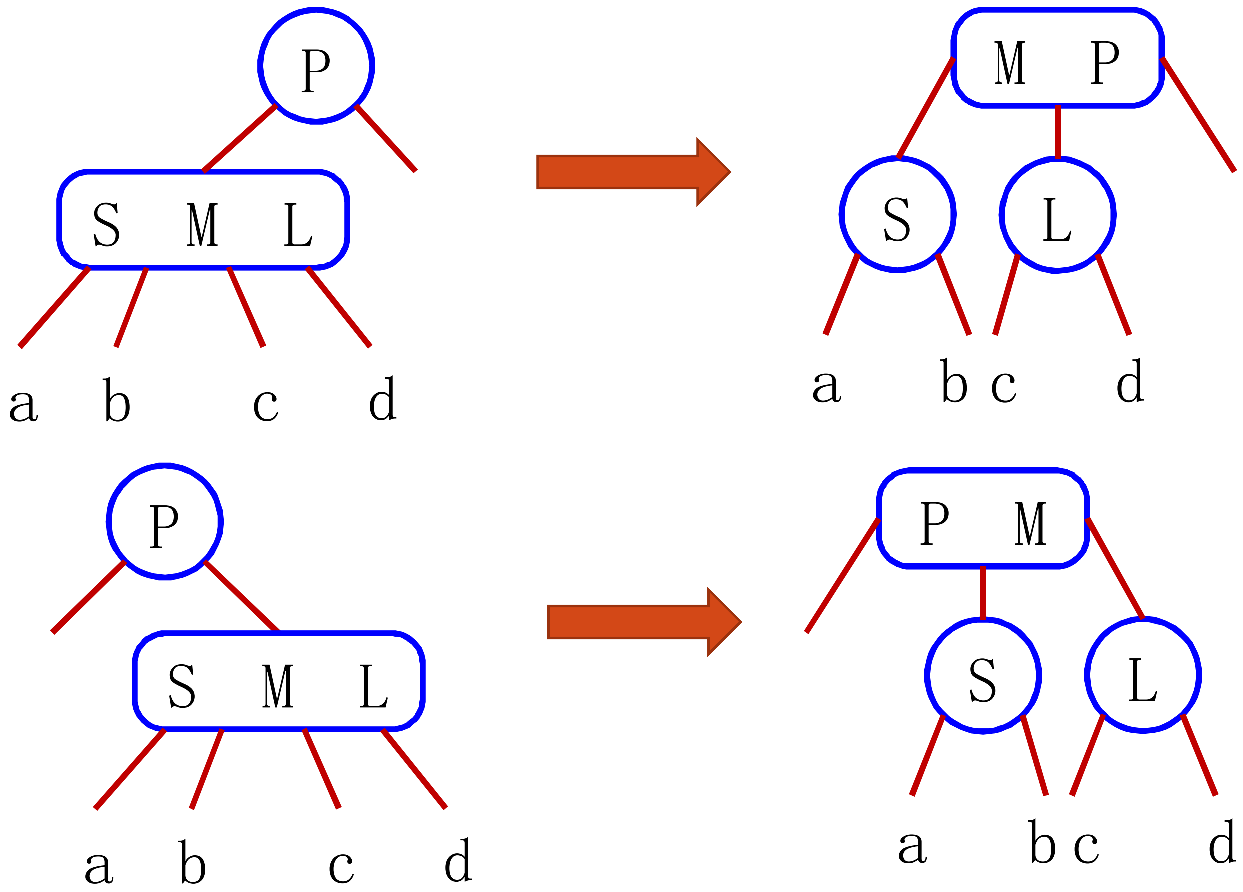


Splitting a Leaf Node

- If the parent is a 3-node, it now contains 3 keys.
  - It violates the 2-3 tree property!
- We need to further split an **internal** node and move its middle key to its parent.



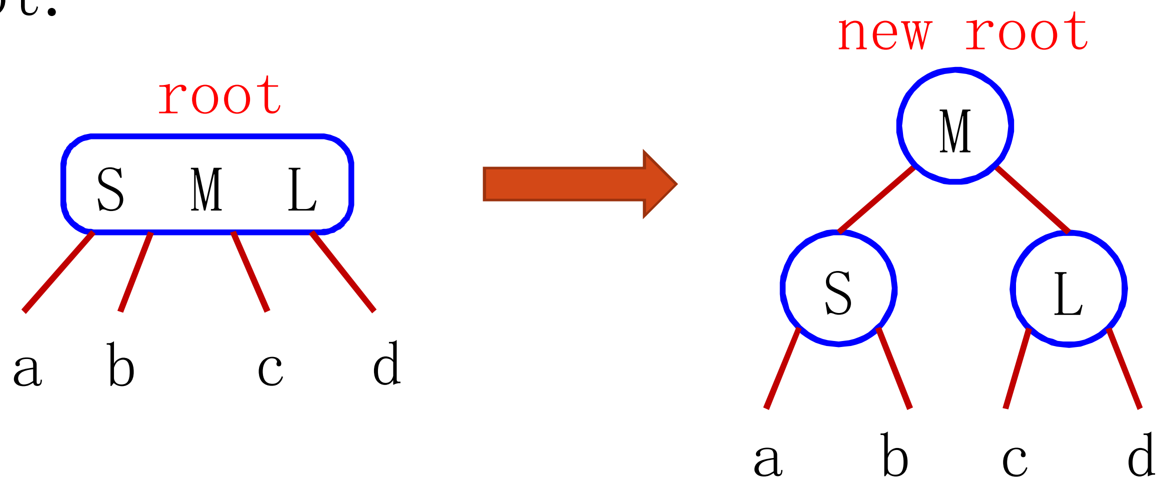
Splitting an Internal Node



Note: the order on keys is prese

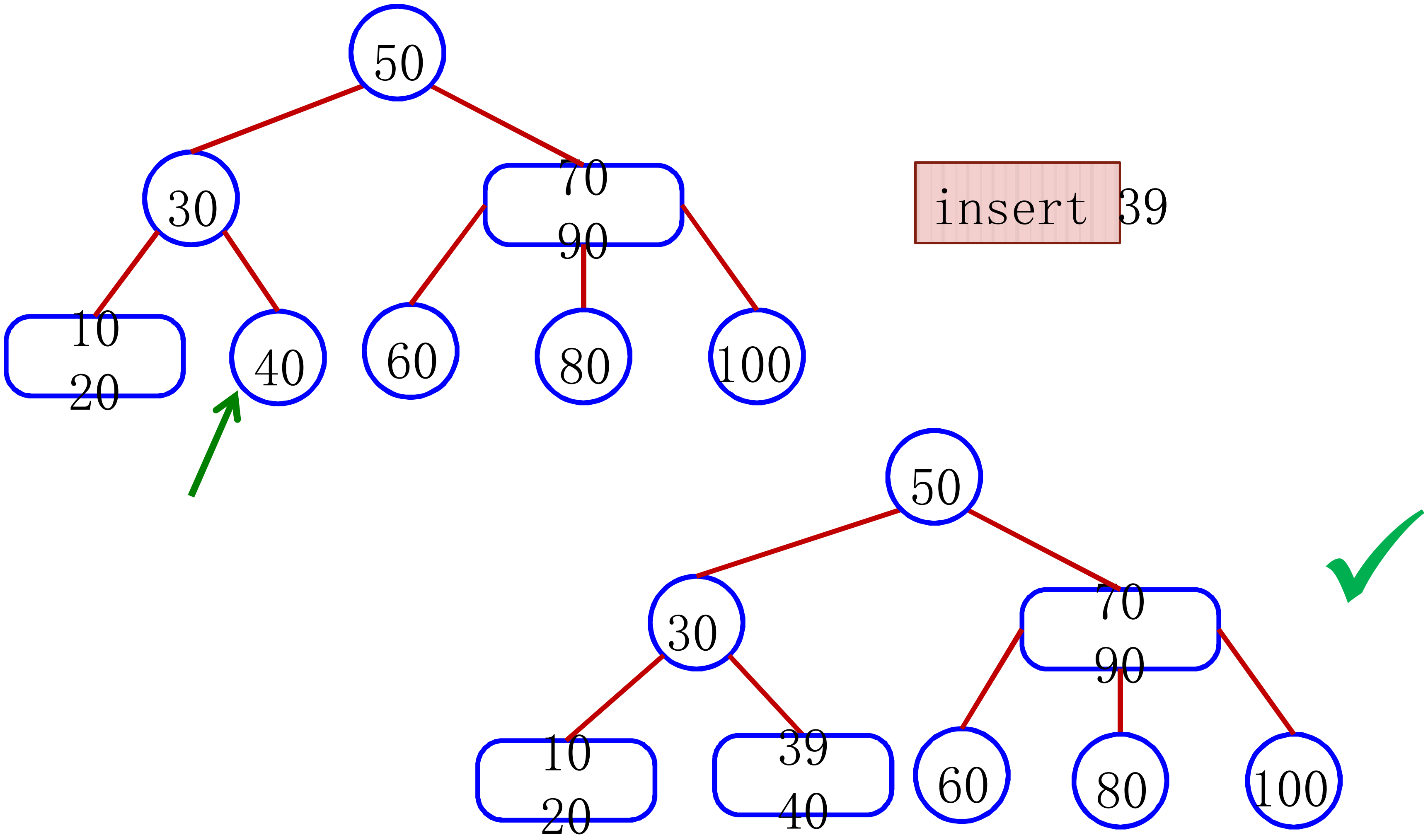
Splitting a Root

- We may **repeat** splitting an internal node and moving its middle key to its parent.
- In the extreme case, we may split the root and move its middle key up, creating a new root.



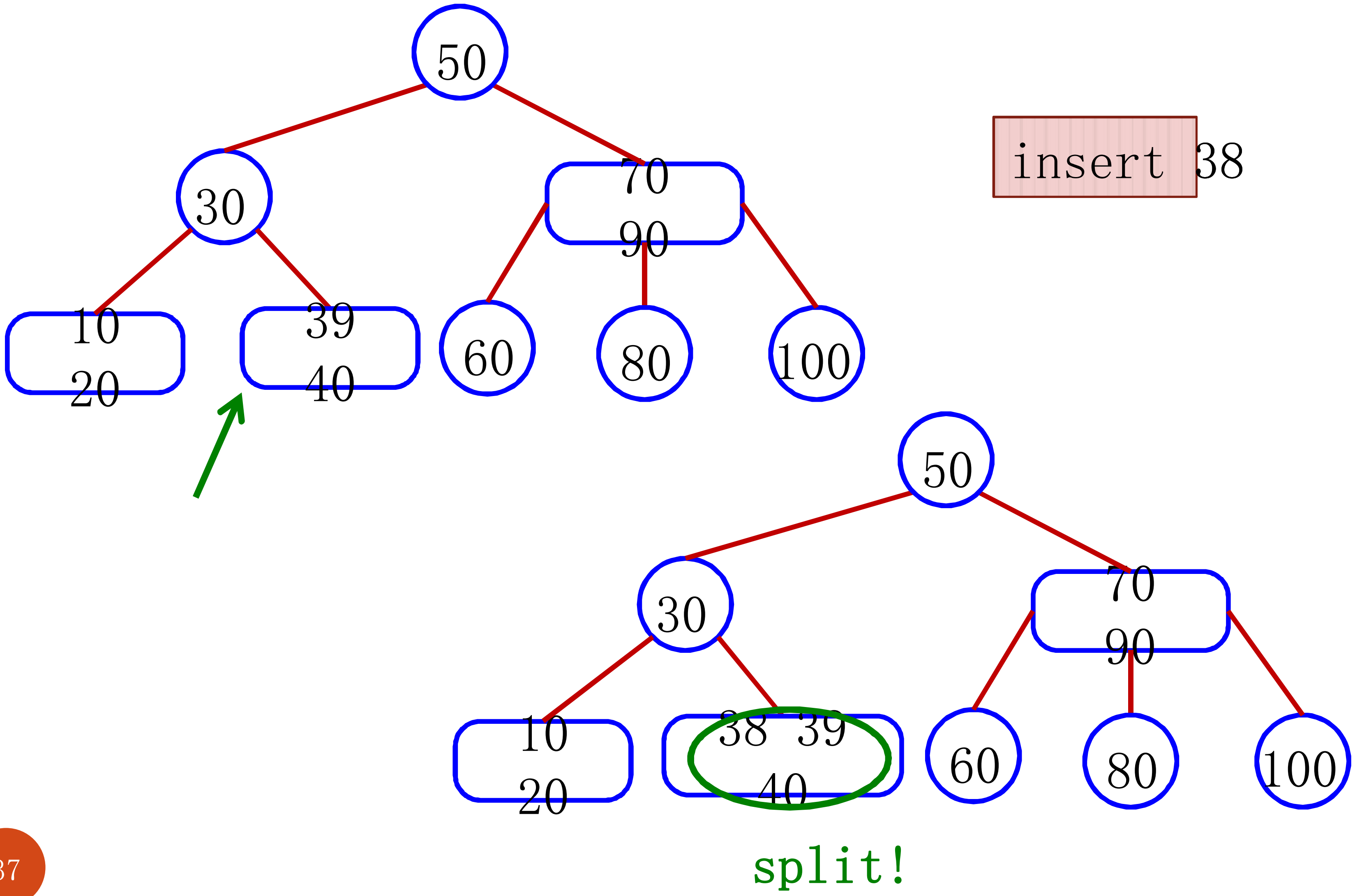
2-3 Trees Insertion

Example



2-3 Trees Insertion

Example



2-3 Trees Insertion

Example