Applications of the Binary Search Tree

Junhao Gan

ITEE
University of Queensland

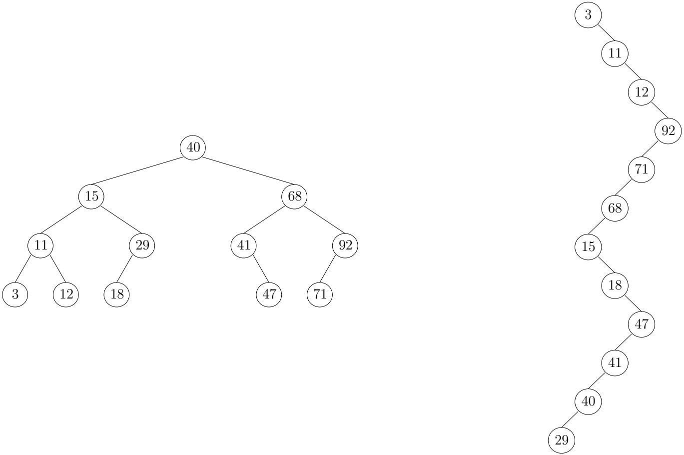
Recall

A binary search tree (BST) on a set S of n integers is a binary tree T satisfying all the following requirements:

- T has n nodes.
- Each node u in T stores a distinct integer in S, which is called the key of u.
- For every internal u, it hods that:
  - The key of u is larger than all the keys in the left subtree of u.
  - The key of u is smaller than all the keys in the right subtree of u.

Example

Two possible BSTs on $S = \{3, 11, 12, 15, 18, 29, 40, 41, 47, 68, 71, 92\}$:

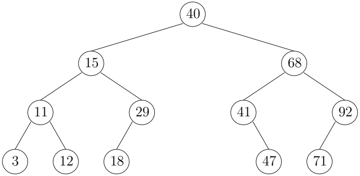


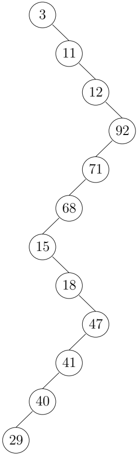
Recall

A binary tree T is balanced if the following holds on every internal node u of T:

- The height of the left subtree of u differs from that of the right subtree of u by at most 1.

Example





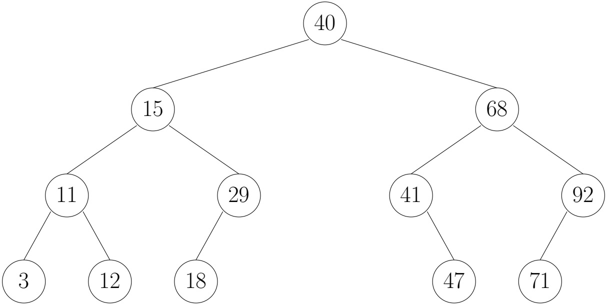
The BST on the left is balanced, while the one on the right is not.

Predecessor Query

Let S be a set of integers. A predecessor query for a given integer q is to find its predecessor in S, which is the largest integer in S that does not exceed q.

Example

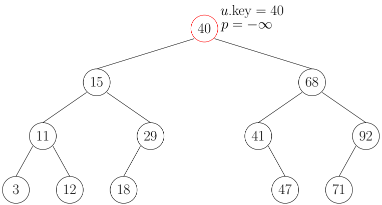
Suppose that $S = \{3, 11, 12, 15, 18, 29, 40, 41, 47, 68, 71, 92\}$ and we have a balanced BST T on S:



We want to find the predecessor of $q = 42$ in S.

Example

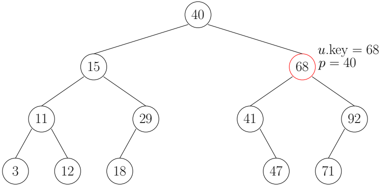
Predecessor query for $q = 42$:



- Initialize $p = -\infty$.
- Initialize $u \leftarrow$ the root of T.
- Now u.key $= 40$ and $p = -\infty$.
- Since u.key $< q$, the predecessor of q must be either u or some node in the right subtree of u.
- Set $p = 40$ and $u \leftarrow$ the right child of u.

Example

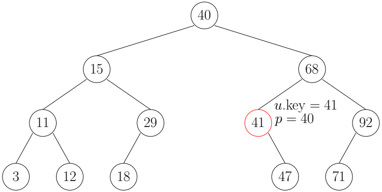
Predecessor query for $q = 42$:



- Since u.key $> q$, the predecessor of q must be either p or some node in the left subtree of u.
- Set $u \leftarrow$ the left child of u.

Example

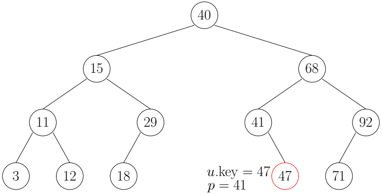
Predecessor query for $q = 42$:



- Since u.key $< q$, the predecessor of q must be either u or some node in the right subtree of u.
- Set $p = 41$ and $u \leftarrow$ the right child of u.

Example

Predecessor query for $q = 42$:



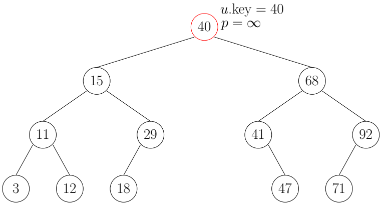
- Since u.key $> q$, the predecessor of q must be either p or some node in the left subtree of u.
- Set $u \leftarrow$ the left child of u.
- Since u is nil now, return $p = 41$ as the predecessor of q in S.

Successor Query

Let S be a set of integers. A successor query for a given integer q is to find its successor in S, which is the smallest integer in S that is no smaller than q.

Example

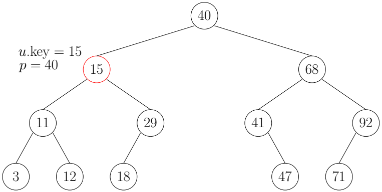
Successor query for $q = 17$ on S:



- Initialize $p = \infty$.
- Initialize $u \leftarrow$ the root of T.
- Now u.key $= 40$ and $p = \infty$.
- Since u.key $> q$, the successor of q must be either u or some node in the left subtree of u.
- Set $p = 40$ and $u \leftarrow$ the left child of u.

Example

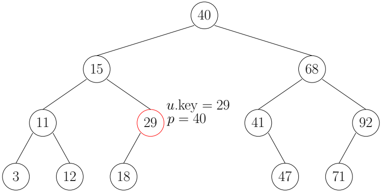
Successor query for $q = 17$ on S:



- Since u.key $< q$, the successor of q must be either p or some node in the right subtree of u.
- Set $u \leftarrow$ the right child of u.

Example

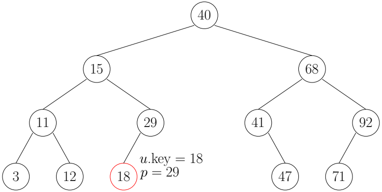
Successor query for $q = 17$ on S:



- Since u.key $> q$, the successor of q must be either u or some node in the left subtree of u.
- Set $p = 29$ and $u \leftarrow$ the left child of u.

Example

Successor query for $q = 17$ on S:



- Since u.key $> q$, the successor of q must be either u or some node in the left subtree of u.
- Set $p = 18$ and $u \leftarrow$ the left child of u.
- Since u is nil now, return $p = 18$ as the successor of q in S.

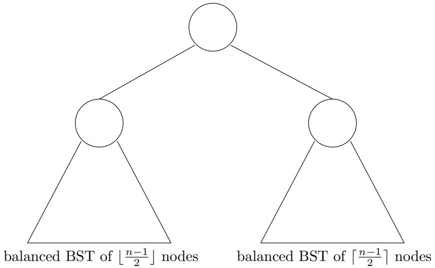
Construction of a Balanced BST

In the following, we will discuss how to construct a balanced BST T on a given sorted set S of n integers in $O(n)$ time.

Construction of a Balanced BST

- **Observation 1:** The subtree of any node in a balanced BST is also a balanced BST.

- **Observation 2:** A BST of n nodes constructed by the following form:



is a balanced BST.

Construction of a Balanced BST

Assume that the sorted set S of n integers is stored in an array with length n. A balanced BST on S can be constructed as follows:

- Base Case:
  - If $n = 0$, return nil.
  - If $n = 1$, create a node u with key $A[1]$ and return the pointer of u as the root of a balanced BST on A.
- Inductive Case:
  - Pick the median of A (i.e., $A[\lfloor \frac{n}{2} \rfloor]$) and create a node u for it.
  - Recursively construct a balanced BST on the portion of A positioned before the median, and set its root as the left child of u.
  - Recursively construct a balanced BST on the portion of A positioned after the median, and set its root as the right child of u.
  - Return the pointer of u.

Construction of a Balanced BST

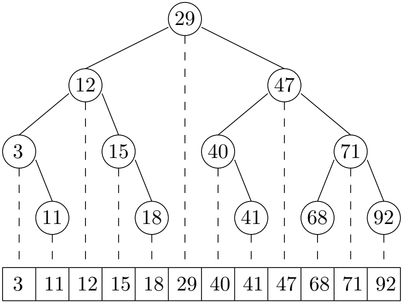
Let $f(n)$ be the maximum running time for constructing a balanced BST from an array of length n. Without loss of generality, suppose that n is a power of 2. We have:

$$f(1) = O(1)$$
$$f(n) = O(1) + 2 \cdot f(n/2)$$

Solving the recurrence gives $f(n) = O(n)$.

Example

Let us construct a balanced BST T on a sorted set $S = \{3, 11, 12, 15, 18, 29, 40, 41, 47, 68, 71, 92\}$ by the above algorithm. Suppose that S is stored in an array A of length 12.



Range Count Problem

Let S be a set of n integers. Given two integers a and b such that $a \leq b$, a range count query for the range $[a, b]$ is to find the number of integers in S which are in the range of $[a, b]$.

In the following, we will discuss how to augment a balanced BST on S to achieve:

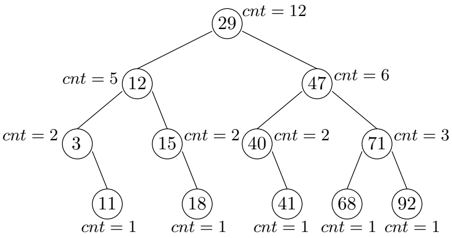
- $O(n)$ space consumption,
- $O(\log n)$ time for each query.

Range Count Problem

We augment a balanced BST T on S by storing one additional information in each node u that is:

- the number of nodes in the subtree of u.

For example,

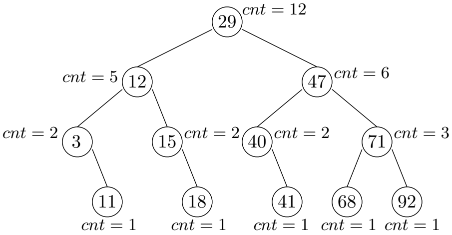


Range Count Problem

Before describing the query algorithm, introduce some concepts:

- Left-Hanging Node: Consider a path $P(u, v)$ from an ancestor u to a node v, if a node w is a left child node of some node on $P(u, v)$ and w is not on $P(u, v)$, then w is called a left-hanging node of $P(u, v)$.

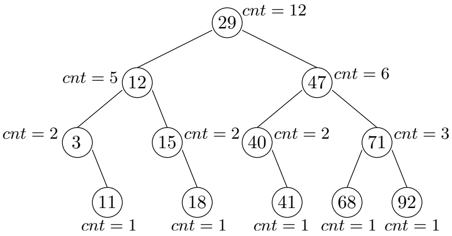
For example, consider a path $P(47, 68)$, the node with key 40 is a left-hanging node of $P(47, 68)$, while the node with key 68 is not.



Range Count Problem

- Right-Hanging Node: Consider a path $P(u, v)$ from an ancestor u to a node v, if a node w is a right child node of some node on $P(u, v)$ and w is not on $P(u, v)$, then w is called a right-hanging node of $P(u, v)$.

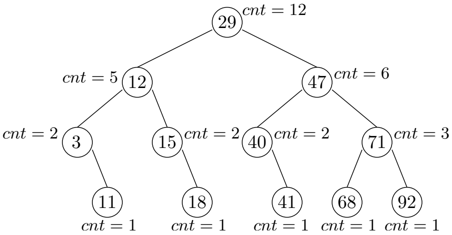
For example, consider a path $P(29, 3)$, the nodes with keys $11, 15, 47$ are right-hanging nodes of $P(29, 3)$.



Range Count Problem

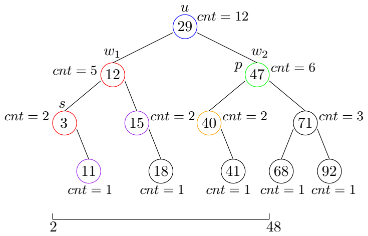
- Lowest Common Ancestor: Let t be the root. The lowest common ancestor of nodes v_1 and v_2 is the lowest node that is on both of the paths $P(t, v_1)$ and $P(t, v_2)$.

For example, the lowest common ancestor of node with key 3 and node with key 15 is the node with key 12.



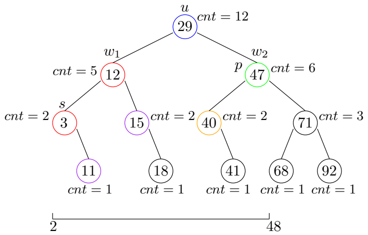
Range Count Problem

For a range $[a, b]$ (e.g. $[2, 48]$), let s be the successor of a, p the predecessor of b and u the lowest common ancestor of s and p. Let w_1 and w_2 be the left child and right child of u.



The purple nodes are the right-hanging nodes of $P(w_1, s)$ and the orange node is the left-hanging nodes of $P(w_2, p)$. Observe that all the nodes in the subtrees of these left- and right-handing nodes are in the range $[2, 48]$.

Range Count Problem



Therefore, the number c of nodes of T in the range $[2, 48]$ can be computed by:

- Initialize $c = 1$.
- Increase c by the number of nodes on $P(w_1, s)$ and $P(w_2, p)$ whose keys are in $[2, 48]$.
- For each right-hanging node v of $P(w_1, s)$, increase c by the counter of v.
- For each left-hanging node v of $P(w_2, p)$, increase c by the counter of v.

For example, $c = 1 + 2 + 1 + 1 + 2 + 2 = 9$.

Range Count Problem

The range count query algorithm for a given range $[a, b]$:

- Find the successor s of a and the predecessor p of b.
- Identify the lowest common ancestor u of s and p. Let w_1 and w_2 be the left and right child nodes of u.
- Initialize $c = 1$.
- Increase c by the number of nodes on $P(w_1, s)$ and $P(w_2, p)$ whose keys are in $[a, b]$.
- Walk along the path $P(w_1, s)$, for each right-hanging node v, increase c by the counter of v.
- Walk along the path $P(w_2, p)$, for each left-hanging node v, increase c by the counter of v.
- Return c.

The time complexity of the above query algorithm is $O(\log n)$.

Besides the range count problem, we can also augment a balanced BST on S to solve the following two interesting problems:

- Range Sum Problem: Given two integers a and b such that $a \leq b$, a range sum query for the range $[a, b]$ is to find the sum of the integers in S which are in the range of $[a, b]$.
- Range Max Problem: Given two integers a and b such that $a \leq b$, a range max query for the range $[a, b]$ is to find the max of the integers in S which are in the range of $[a, b]$.

Think: How?