Applications of the Binary Search Tree

Hou Pong CHAN

Department of Computer Science and Engineering
The Chinese University of Hong Kong

Adapted from the slides of Junhao Gan
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 holds 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**.
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 $q$. 
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 $S$. 

![Binary Search Tree Diagram](image-url)
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 $u$. 

CSCI2100, CUHK  Applications of the Binary Search Tree
Example

Predecessor query for \( q = 42 \):

- Since \( u.\text{key} < q \), 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 $S$. 

```
Ex.

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 ← 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 $S$:

1. Initialize $p = \infty$.
2. Initialize $u \leftarrow$ the root of $T$.
3. Now $u.key = 40$ and $p = \infty$.
4. Since $u.key > q$, the successor of $q$ must be either $u$ or some node in the left subtree of $u$.
5. 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 $u ←$ 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 $S$. 
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.
**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:

```
balanced BST of ⌊n−1/2⌋ nodes
```

```
balanced BST of ⌈n−1/2⌉ nodes
```

is a balanced BST.
Construction of a Balanced BST

Assume that the $S$ of $n$ integers is stored in an array, the array is sorted. 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 $A$. 

CSCI2100, CUHK

Applications of the Binary Search Tree
**Construction of a Balanced BST**

- **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 $u$. 

\[
\text{balanced BST of } \lfloor \frac{n-1}{2} \rfloor \text{ nodes } \quad \text{balanced BST of } \lceil \frac{n-1}{2} \rceil \text{ nodes }
\]
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 $a \leq b$. 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

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

Define a concept first.

- 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.

```
          29  cnt = 12
         /    \
        12     47  cnt = 6
       /  \
      3    15  40  cnt = 2
     /  \
    11   18  41  cnt = 1
   /     /  \
  68   71   92 cnt = 1
```

Applications of the Binary Search Tree
Range Count Problem

For a range [2, 48], let $s$ be the successor of 2, $p$ the predecessor of 48 and $u$ the lowest common ancestor of $s$ and $p$.

Initialize a count $c = 1$ (since $u$ is within the range)

\[
\begin{array}{c}
\text{cnt} = 12 \\
\text{cnt} = 5 \\
\text{cnt} = 2 \\
\text{cnt} = 2 \\
\text{cnt} = 3 \\
\text{cnt} = 6 \\
\text{cnt} = 1 \\
\text{cnt} = 1 \\
\text{cnt} = 1 \\
\text{cnt} = 1 \\
\text{cnt} = 1 \\
\end{array}
\]
Range Count Problem

 Traverse the path from $u$’s left child to $s$. For every node $v$ being visited:

- $c += 1$
- $c +=$ the counter of $v$’s right child

$C$ is incremented by $1 + 2$. 
Range Count Problem

 Traverse the path from $u$'s left child to $s$. For every node $v$ being visited:

- $c += 1$
- $c +=$ the counter of $v$'s right child

$C$ is incremented by $1 + 1$. 

$CSCI2100$, $CUHK$  
Applications of the Binary Search Tree
Range Count Problem

Traverse the path from $u$’s right child to $p$. For every node $v$ being visited:

- $c += 1$
- $c +=$ the counter of $v$’s left child

$C$ is incremented by $1 + 2$. Finally, $c$ becomes 9.
**Range Count Problem**

We walked through two paths, at most $\log_2 n$ nodes in each path. For each node visited, we perform constant-time operations, which takes $O(1)$. Time complexity: $O(\log n)$
Range Count Problem

A simpler solution without using a binary search tree

- Use binary search algorithm to find the successor $s$ of $a$
- Use binary search algorithm to find the predecessor $p$ of $b$
- Let $l$ be the index of $s$, then let $u$ be the index $p$
- Return $u - l - 1$

The above algorithm uses two binary search, the time complexity is $O(\log n)$. 
Range Count Problem

Why don’t we just use the simpler solution? In practice, we may need to update (insert or delete) the elements in $S$.

Simpler Solution:
- Need to sort $S$ after each update.
- Cost for each update: $O(n \log n)$

Solution with BST:
- Need to insert or delete a node in the BST.
- Cost for each update: $O(\log n)$

That’s why we prefer the BST solution in practice.