Data Structures and Algorithms (DSA)

Lecture 10_2: Binary Search Trees

(Average Case Analysis)

Overview

- Binary Search Trees
  - Searching
    - Best Case / Worst Case Analysis
    - Average Case Analysis

Runtime Analysis

Analyzing the runtime, we may have different perspectives:

- Worst case analysis (done so far, default case)
- Best case analysis
- Average case analysis

Notation:

i: problem instance

T(i): runtime on i

I_n: set of instances of size n.

Worst Case Analysis

$$T_w(n) = \max\{T(i) : i \in I_n\}$$

Example:

Find for binary search trees: $T_w(n) = \Theta(n)$

- Tree may degenerate to a list
- Tree has height n-1
- We want to find the element of height n-1

Best Case Analysis

$$T_b(n) = \min\{T(i) : i \in I_n\}$$

Example:

Find for binary search trees: $T_b(n) = O(1)$

- The element is at the root of the tree

Analysis for height of a binary search tree:

- Worst case: $\Theta(n)$
- Best case: $\Theta(\log n)$

Average Case Analysis

$$T_a(n) = \frac{1}{|I_n|} \sum_{i \in I_n} T(i)$$

Average the runtime over all possible inputs

Average time for find (degenerated tree)

Assume: T is generated to a list and consists of elements 1, 2, …, n.

Input for find: Element $i \in \{1, \ldots, n\}$

Average time to find an element in T chosen uniformly at random is

$$\frac{1}{n} \cdot (1 + 2 + \ldots + n) = (n+1)/2$$

Average time for find (balanced tree)

Assume:

- T is perfectly balanced.
- $n = 2^k - 1$ elements.

Observation:

- There are 2^i elements at depth i, $0 \leq i \leq k-1$.
- Time to find element at depth i is i+1.

Time to find an element in T chosen uniformly at random is

$$\frac{1}{n} \cdot \sum_{i=0}^{k-1} (i+1) \cdot 2^i$$

$$= \frac{1}{n} \cdot (k \cdot 2^{k-1} + (k-1) \cdot 2^{k-2} + \ldots + 1 \cdot 2^0)$$

$$
\begin{aligned}
= \frac{1}{n} \cdot (\quad & 2^{k-1} + 2^{k-2} + \ldots + 2^{0} \\
+ \quad & 2^{k-1} + 2^{k-2} + \ldots + 2^{1} \\
+ \quad & 2^{k-1} + 2^{k-2} + \ldots + 2^{2} \\
& \ldots \\
+ \quad & 2^{k-1} + 2^{k-2} \\
+ \quad & 2^{k-1})
\end{aligned}
$$

Use geometric series:

$$\sum_{i=0}^{k} 2^i = 1 + 2 + 4 + \ldots 2^k = 2^{k+1} - 1$$

We get:

$$= \frac{1}{n} \cdot ((2^k - 2^0) + (2^k - 2^1) + (2^k - 2^2) + \ldots (2^k - 2^{k-1}))$$

$$= \frac{1}{n} \cdot (k \cdot 2^k - (2^k - 1))$$

$$= \frac{1}{n} \cdot (\log(n+1) \cdot (n+1) - n)$$

$$= (1 + \frac{1}{n}) \log(n+1) - 1$$

Theorem

Theorem: The average time to find an element in a perfectly balanced binary tree with $n = 2^k-1$ elements is $(1 + \frac{1}{n}) \log(n+1) - 1$

Average Case for random insertion

- Assume that the items to be inserted are in random order.

- We may be lucky and the tree has small depth (does not degenerate to a list)

Question:

- What is the average time to find an element in such a tree?

Permutations of n elements

Assume that we have a set of n elements

Consider all permutations of these elements

There are n! permutations.

Example: Set {1, 2, 3}

Permutations:

(1,2,3), (1,3,2), (2,1,3), (2,3,1), (3,1,2), (3,2,1)

Analysis

In our analysis:

- we average over the different permutations for building the binary search tree.
- all queries for the elements.

Formally, we consider “double expected value” with respect to:

- the order of elements inserted
- the element we query

Cost of a search tree

c(v): number of nodes on the path from the root to v.

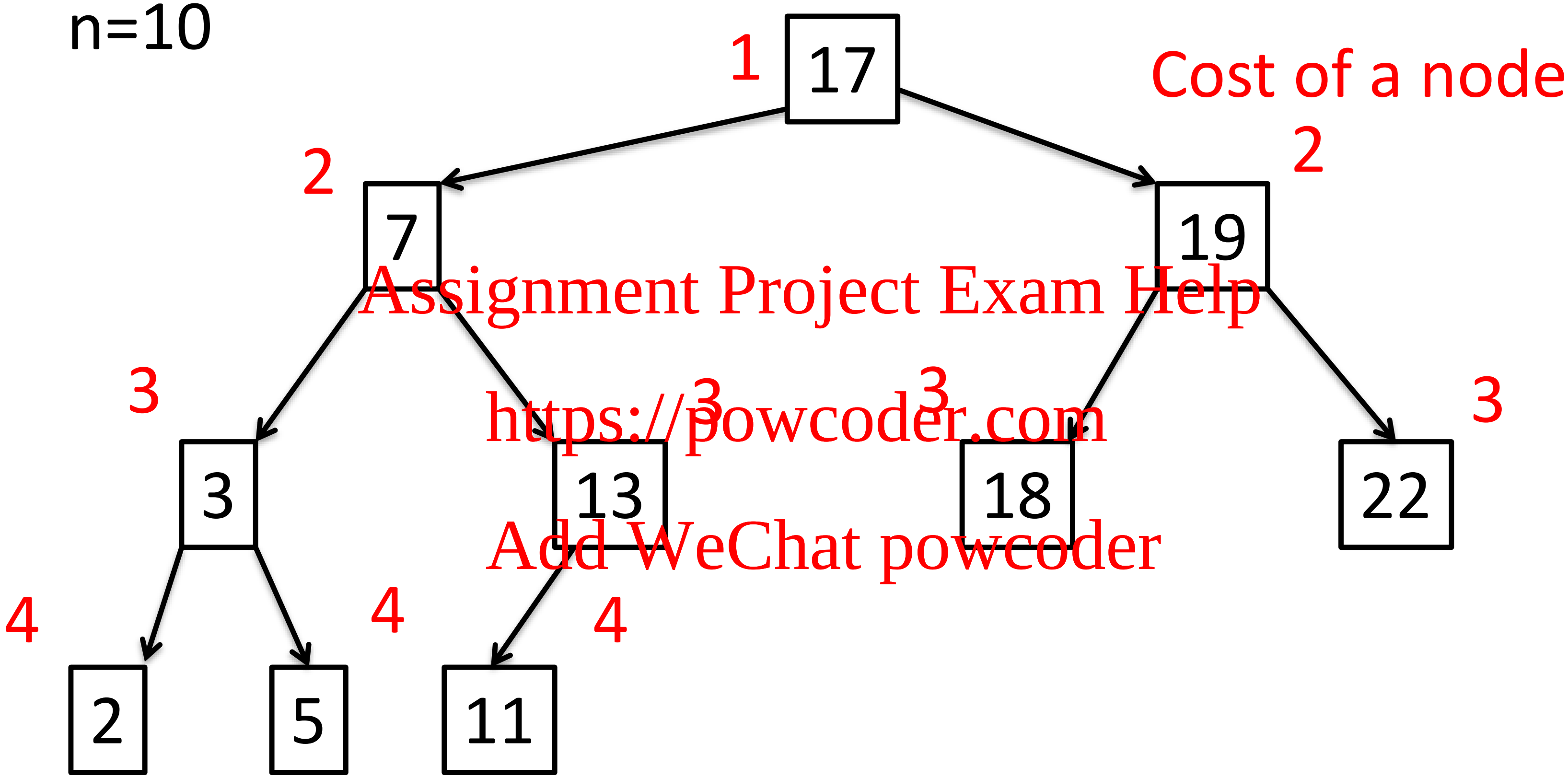
Cost of a tree T:

$$C(T) = \sum_{v \in T} c(v)$$

Average search cost of a tree T: $C(T)/n$

Cost of a tree

n=10



Cost of the tree C(T) = 1+2+2+3+3+3+3+4+4+4=29

Average search time for T: C(T) / n = 29 / 10 = 2.9

Average costs of a tree

Let E(n) be the average cost of tree with n elements.

Recursion:

$$E(0) = 0$$

$$E(1) = 1$$

$$E(n) = n + \frac{1}{n} \sum_{i=1}^{n} (E(i-1) + E(n-i))$$

Recursive Formula

i-1 elements go into the left subtree

n-i elements go into the right subtree

$$E(n) = n + \frac{1}{n}\sum_{i=1}^{n}(E(i-1) + E(n-i))$$

Each element i is with equal probability the root

Root lies on every path to a node

Solve Recursion

- Recursive Formula seems to be complicated.
- Is it worth the effort?

Reasons for doing that:

- Result is interesting
- Math tricks can often be used
- Similar analysis gives average case results for the Quicksort algorithm.

Solving Recursion

$$E(n) = n + \frac{1}{n}\sum_{i=1}^{n}(E(i-1) + E(n-i))$$

contains E(0), E(1), …, E(n-1).

First step:

- Get a recursive formula for E(n) that only depends on E(n-1).

Consider $n \cdot E(n) - (n-1)E(n-1)$

This implies that E(n-2), …, E(1) get the same factor and cancel out, i. e.

$$n \cdot E(n) = n^2 + \sum_{i=1}^{n} (E(i-1) + E(n-i))$$

$$= n^2 + 2 \cdot (E(1) + E(2) + \ldots E(n-1))$$

$$(n-1)\cdot E(n-1) = (n-1)^2 + \sum_{i=2}^{n}(E(i-1)+E(n-i))$$

$$= (n-1)^2 + 2\cdot(E(1)+E(2)+\ldots E(n-2))$$

$$n\cdot E(n) - (n-1)E(n-1)$$

$$= n^2 - (n-1)^2 + 2\cdot E(n-1)$$

$$= 2n - 1 + 2\cdot E(n-1)$$

$$n \cdot E(n) - (n+1) \cdot E(n-1) = 2n - 1$$

Divide by n(n+1)

$$\frac{1}{n+1} \cdot E(n) - \frac{1}{n} \cdot E(n-1) = \frac{2n-1}{n(n+1)}$$

Consider:

$$Z(n) = \frac{1}{n+1} \cdot E(n)$$

$$
\begin{aligned}
Z(n) &= Z(n-1) + \frac{2n-1}{n(n+1)} \\
&= Z(n-2) + \frac{2(n-1)-1}{(n-1)n} + \frac{2n-1}{n(n+1)} \\
&= Z(0) + \sum_{i=1}^{n} \frac{2i-1}{i(i+1)}
\end{aligned}
$$

Use: $$\frac{1}{i(i+1)} = \frac{1}{i} - \frac{1}{i+1}$$

Then we get:

$$Z(n) = 2\sum_{i=1}^{n}\frac{i}{i} - 2\sum_{i=1}^{n}\frac{i}{i+1}$$

$$- \sum_{i=1}^{n}\frac{1}{i} + \sum_{i=1}^{n}\frac{1}{i+1}$$

$$= 2n - 2n + 2\sum_{i=1}^{n} \frac{1}{i+1} - 1 + \frac{1}{n+1}$$

$$= 2\sum_{i=1}^{n} \frac{1}{i} - 2 + \frac{2}{n+1} - 1 + \frac{1}{n+1}$$

$$= 2 \cdot H(n) - 3 + \frac{3}{n+1}$$

Harmonic sum $H(n) = \sum_{i=1}^{n} \frac{1}{i}$

Remember: $Z(n) = \frac{1}{n+1} \cdot E(n)$

$$E(n) = (n+1) \cdot Z(n)$$

$$= 2(n+1) \cdot H(n) - 3(n+1) + 3$$

Average Cost for Find

Average cost for find after random insertion:

$$E(n)/n = 2 \cdot \frac{n+1}{n} \cdot H(n) - 3 \cdot \frac{n+1}{n} + \frac{3}{n}$$

Using: $\ln(n+1) \leq H(n) \leq \ln n + 1$

we get

$$\begin{aligned} E(n)/n &= 2 \cdot \ln n - O(1) = (2 \ln 2) \cdot \log n - O(1) \\ &\approx 1.386 \cdot \log n \end{aligned}$$

Theorem

Theorem: The insertion of n randomly chosen elements leads to a Binary Search Tree whose expected time for a successful find operation is

$$(2\ln 2)\cdot \log n + O(1) \approx 1.386 \cdot \log n$$

Runtimes for Binary Search Tree

Find, insert, remove:

Worst case: $\Theta(n)$

Best case: $\Theta(\log n)$

Average case: $\Theta(\log n)$

Aim: Time O(log n) in the worst case