Lecture #24: Randomly Constructed Binary Search Trees
Key Concepts

Note: This material is ***for interest only*** — it will not be included in any assignment or test in this course.

Binary Search Trees

Binary search trees are data structures that are used to represent *finite sets* whose elements are chosen from a *universe* with a "total order": The set $\mathbb{Z}$ of integers is one example of a universe that satisfies this property.

- Since each element of the set (represented by a binary search tree) is stored at a node, the ***size*** of a binary search tree is the same as the size of the finite set that it represents.
- Since every binary search tree is a binary tree, the ***depth*** of a tree is always at least logarithmic in its size, and at most linear in its size.
- Algorithms for searches, insertions and deletions in a binary search tree all use a number of steps that is at most linear in the ***depth*** of the binary search tree, in the worst case — so binary search trees where the depth is only logarithmic in the size of the tree are preferred.
- The ***shape*** of a binary search tree does not depend on the set of values stored in it — just the size of this set. Thus, if $n \geq 0$, and we wish to consider binary search trees with size n, it is sufficient to consider binary search trees that store the set

  $$\{1, 2, \ldots, n\}$$

  of positive integers whose values are at most n.

Sample Space and Probability Distribution

- Let n be a non-negative integer. Any binary search tree with size n, that stores the set $\{1, 2, \dots, n\}$, can be created by starting with an empty tree and storing the integers $1, 2, \dots, n$ *in some order*.
- The ***sample space***, Ω_n, used to model this is the set of all ***permutations***
$$\rho : \{1, 2, \dots, n\} \to \{1, 2, \dots, n\}$$
of the first n positive integers. Such a permutation, ρ, can also be shown as
$$(\alpha_1, \alpha_2, \dots, \alpha_n)$$
— where $\rho(i) = \alpha_i$ for $1 \leq i \leq n$ — and the size of Ω_n is $n!$.
- The ***uniform probability distribution***, mapping each element of Σ_n to $\frac{1}{n!!}$, is used. That is, it is assumed that *each* of the $n!$ relative orderings of the integers $1, 2, \dots, n$ is used to create a binary search tree with size n — using each relative ordering of the keys with the same probability.

Random Variables

Two random variables are considered,

- For $\alpha \in \Sigma_n$, $d(\alpha)$ is the *depth* of the binary search tree obtained by inserting the integers $1, 2, \dots, n$ into an initially empty binary search tree, using the ordering of these integers given by α.
- The *exponential depth* is the function $xd : \Sigma_n \to \mathbb{N}$ such that, for $\alpha \in \Sigma_n$,
$$xd(\alpha) = 2^{d(\alpha)}.$$
That is, $xd(\alpha)$ is the product of $d(\alpha)$ copies of "2" (or "2 raised to the *power* $d(\alpha)$"), for $d(\alpha)$ as above.

Results

By showing that both Ω_n and the probability distribution P can be described in different ways (without really changing the set, or the probability distribution being defined) it can be shown that
$$\mathsf{E}[xd] \leq n^3.$$

This can be used to establish each of the following:

- $\mathsf{E}[d] \leq 3 \log_2 n$.
- For every positive integer k — using the probability distribution that has been introduced — the probability that the depth of a "randomly generated" binary search tree is more than $k + 3 \log_2 n$ is at most 2^{-k}.

Note: As always, the results obtained, using this kind of analysis, are only relevant if the ***assumptions*** made, for the analysis, are correct.

With that noted, binary search trees have now been used for a considerable amount of time — and experience using them suggests that they *do* support efficient computations, as these results suggest.