The expected height of a binary search tree

Elias Nodland

11.04.2020

Introduction

A kattis programming problem posed the question: *What is the expected height of a binary search tree consisting of N nodes?*

The problem can be found at: https://open.kattis.com/problems/wiseguy

1. The tree is to be constructed from a list of N integers in the range of $[1, N]$, each integer occurring only once. This list is random, meaning each of the $N!$ permutations are equally probable.
2. Each element is always passed to the top node. If the element is larger, it is passed down to the right, smaller to the left. This is done until every element is in the tree.
3. The height of the tree is equal to the depth of the deepest node. The expected height for any N is equal to the sum of the height of all the $N!$ trees divided by $N!$.

Theory

To clarify the structure of the tree an illustration for $N = 3$ is included below. The list permutation is paired with its corresponding tree.

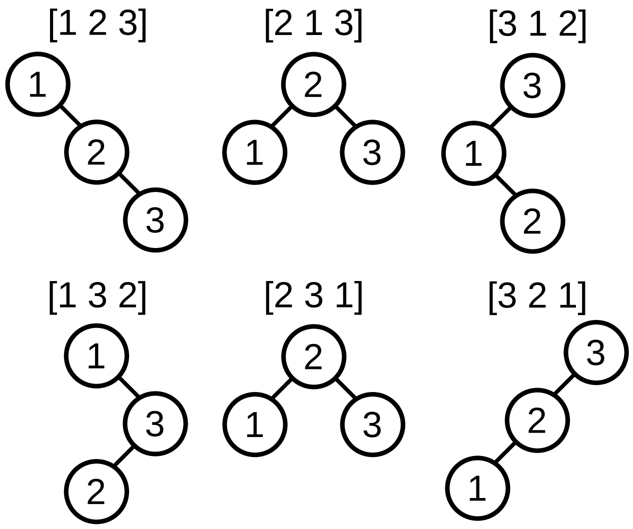


Figure 1: All permutations for N = 3

Notice that some permutations will generate the same tree.

Given the formula from the introduction the expected height of a binary search tree with 3 nodes is then:

$$\frac{1+1+2+2+2+2}{3!} = \frac{5}{3} \approx 1.6667$$

For trees of less than $N = 10$ nodes it is possible to use brute force to find a solution. This approach is infeasible for any large N as it has a complexity of $O(N!)$.

Solution

For any tree of N nodes it can take on N general forms.

For a tree of 4 nodes there are therefore 4 forms it can take on. These are illustrated below.

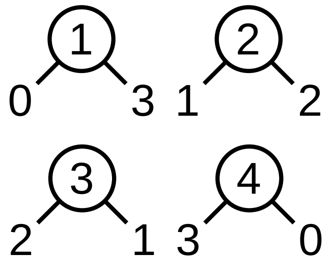


Figure 2: All forms for $N = 4$

The top node can be 1, 2, 3, or 4. Depending on what the top node is, the rest of the nodes will land on some side. This is indicated by the numbers that don't have a circle around them. There are exactly $(N-1)!$ trees of each form.

To calculate the probability that a tree of 4 nodes has some height h *or less* is the same problem as calculating the probability that the largest height of any of the two subtrees is $h-1$ *or less*.

The equation looks like this:

$$P(H \leq h) = P(L \leq h-1) \cdot P(R \leq h-1)$$

Where H is the height of the tree, L is the height of the left subtree and R the height of the right subtree. $P(H \leq h)$ is the probability of the height H being less than or equal to h.

The probability is summed accross every form and divided by N, the number of forms to arrive at the final probability for every height.

Implementation

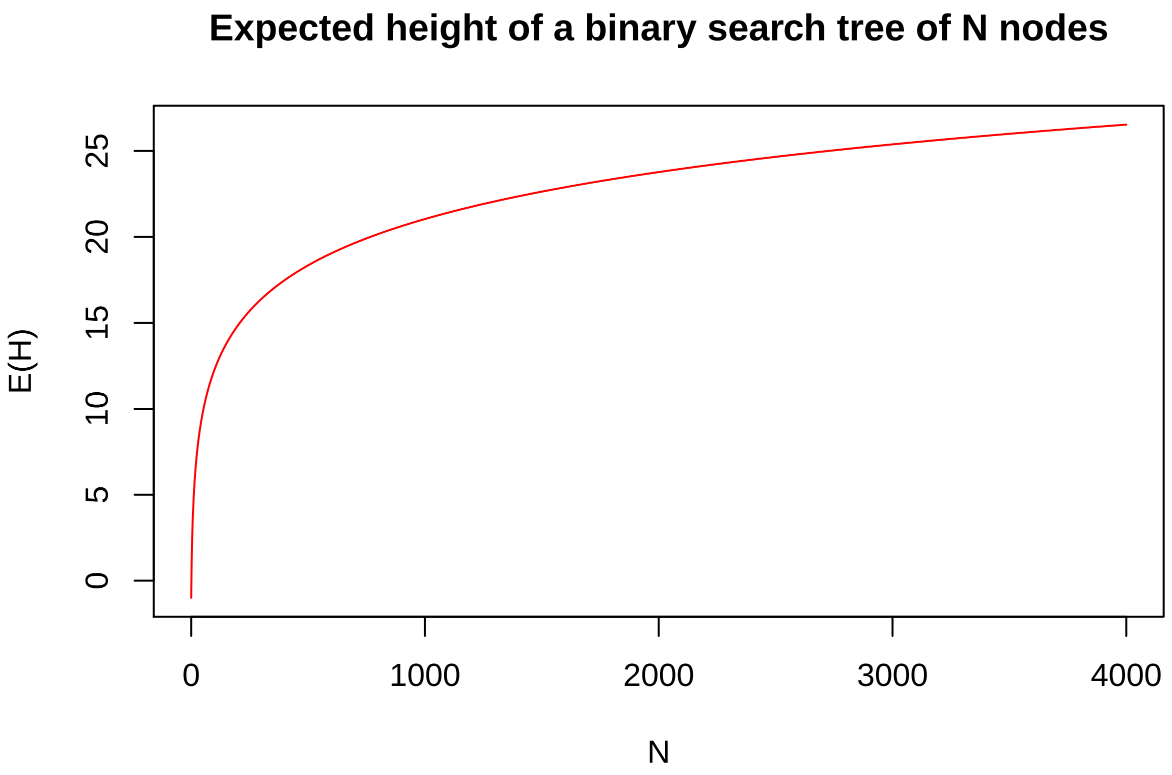
The implementation of the algorithm is done in python and relies on scipy and numpy.

```python
import numpy as np

# This function shifts an array a to the right by 1 filling with 0's (The first
# element becomes 0) ex: [1 2 3] → [0 1 2]
def shift(a):
    for i in range(len(a) - 1, 0, -1):
        a[i] = a[i - 1]
    a[0] = 0

N = 15 + 1

# Prepare matrices
P = np.zeros((N, N))

# Initialize with known probabilities
P[:, 0] = np.ones((1, N))
P[:, 1] = np.ones((1, N))

for n in range(2, N):
    # For each N
    for l in range(0, n):
        r = n - l - 1
        # Multiply left and right probabilites and add to result
        P[:, n] += np.multiply(P[:, l], P[:, r])
    P[:, n] = np.divide(P[:, n], n)
    # Shift result to the right. This is done because the probability of some
    # height h is a function of the probability of h - 1 Shifting to the right
    # has the desired effect
    shift(P[:, n])

# The result is currently cumulative meaning that P[n][h] = P(H <= h) for some n
# This loop reverses that by observing that P(H = h) = P(H <= h) - P(H <= h-1)
for k in range(0, N):
    for i in range(N - 1, 0, -1):
        P[i, k] -= P[i - 1, k]

# For every n, go through all probabilities and multiply with corresponding
# height and sum up to get expected height. E(H) = sum(h * P(H = h))
E = np.zeros(N)
for n in range(1, N):
    s = 0
    for h in range(0, N, 1):
        s += P[h, n] * h;
    E[n] = s

E[0] = -1.0 # Height of tree with 0 nodes is -1. Included for completeness
```

Table 1: Expected heights for trees with N nodes

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
-1	0	1	1.67	2.33	2.8	3.27	3.67	4.02	4.34	4.64	4.92	5.17	5.41	5.63	5.84

The algorithm has a complexity of $O(n^3)$ and space complexity of $O(n^2)$. The python version completes in just under 1 second for $N = 500$. The algorithm was also written in `C++` to generate the graph below for the first 4000 N.



This took just under a minute on battery power on an i7-1065G7. Doubling the N, the algorithm should in theory take 8 times longer to compute so if you have 3 days and $\approx$ 32GB of RAM you can compute the expected height of a binary tree with 60000 nodes.

Summary

The algorithm has room for optimization, but it is a big improvement over the naive $N!$ algorithm. It is likely that there exists some way to drastically speed this up and there definitely exists some way to predict values with high accuracy, but the speed and efficiency of this algorithm is well within the bounds of the original problem.