Algorithms for Highest Safe Rung (= Product Stress Testing)

(see e.g., Kleinberg and Tardos)

Ruiyang Xu and Karl Lieberherr January 21, 2018

For algorithms courses.

Problem: Given q questions and k rungs (= products) find the largest number n of rungs (=stress levels) that can be correctly handled (i.e., the highest safe rung (= highest safe stress level) is correctly determined). The output is a binary search tree (BST) with n leaves 0..n-1, depth q and every root to leaf path has at most k left branches. Each node has outdegree 0 or 2 and is labeled with a number 1..n-1. For internal nodes the label says at which rung an experiment is performed, for leaves it gives the highest safe rung.

There are two kinds of algorithmic techniques that can be used: Dynamic Programming or a Greedy Saturation algorithm. The second one is simpler in that it does not need the machinery of Dynamic Programming to avoid an exponential running time. The second algorithm is linear in the size of the BST.

Algorithms

1. Use recurrence relation: MR(q,k)=MR(q-1,k-1)+MR(q-1,k) with proper initialization resulting in a modified Pascal's Triangle where the right diagonal consists of the powers of 2.
    a. Construct the BST and label the nodes using a bottom-up algorithm labeling the leaves 0..n-1.
2. Construct the BST using a greedy algorithm that starts at the root. We use a seperate labeling with pairs (q,k). The root is labeled with (q,k). If a node is labeled (q,k), the left successor is labeled (q-1,k-1) and the right successor (q-1,k). A node is expandable if (q>0) and (k>0). Expand the nodes until no leaf is expandable (Greedy saturation).
    a. Apply 1a. to label the tree.
    b. Ruiyang provides the following algorithm to compute MR(q,k) using the Greedy Saturation algorithm:

```
def FindTreeN(q, k):
    key = 0
    leaves = {}
    leaves[key] = (0,0)
    n = 1
    while len(leaves)>0:
      target = leaves.keys()[random.randint(0,len(leaves)-1)]
      extend = leaves[target]
```

```
if extend[0]<q and extend[1]<k:
    key += 1
    leaves[key] = (extend[0]+1, extend[1])
    key += 1
    leaves[key] = (extend[0]+1, extend[1]+1)
    n += 1
    del leaves[target]
else:
    del leaves[target]
```