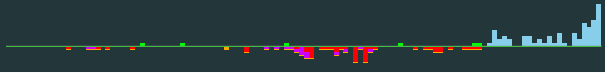
A: Alternating Branch Cutting

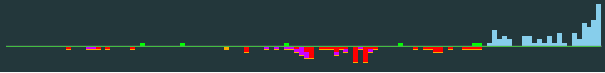
Problem author: Yanxu Chen



Problem: Given that each leaf of a ternary of height n is green with probability $0.5 + p$. After Alice and Emmy have pruned the tree, what is the probability that the final leaf is green?

A: Alternating Branch Cutting

Problem author: Yanxu Chen



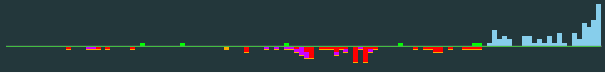
Problem: Given that each leaf of a ternary of height n is green with probability $0.5 + p$. After Alice and Emmy have pruned the tree, what is the probability that the final leaf is green?

Small case: For $n = 1$, we need a majority green, which has probability

$$(0.5 + p)^3 + 3(0.5 + p)^2(0.5 - p) = 0.5 + p(1.5 - 2p^2).$$

A: Alternating Branch Cutting

Problem author: Yanxu Chen



Problem: Given that each leaf of a ternary of height n is green with probability $0.5 + p$. After Alice and Emmy have pruned the tree, what is the probability that the final leaf is green?

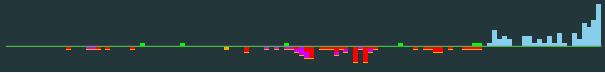
Small case: For $n = 1$, we need a majority green, which has probability

$$(0.5 + p)^3 + 3(0.5 + p)^2(0.5 - p) = 0.5 + p(1.5 - 2p^2).$$

Next case: For $n = 2$, each subtree of height 1 results in a green leaf with probability $0.5 + p_1$ for $p_1 = p(1.5 - 2p^2)$. We need a majority of these subtrees to result in a green leaf, which has probability $0.5 + p_2$ for $p_2 = p_1(1.5 - 2p_1^2)$.

A: Alternating Branch Cutting

Problem author: Yanxu Chen



Problem: Given that each leaf of a ternary of height n is green with probability $0.5 + p$. After Alice and Emmy have pruned the tree, what is the probability that the final leaf is green?

Small case: For $n = 1$, we need a majority green, which has probability

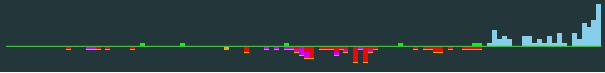
$$(0.5 + p)^3 + 3(0.5 + p)^2(0.5 - p) = 0.5 + p(1.5 - 2p^2).$$

Next case: For $n = 2$, each subtree of height 1 results in a green leaf with probability $0.5 + p_1$ for $p_1 = p(1.5 - 2p^2)$. We need a majority of these subtrees to result in a green leaf, which has probability $0.5 + p_2$ for $p_2 = p_1(1.5 - 2p_1^2)$.

General case: Define $p_0 = p$ and $p_{i+1} = p_i(1.5 - 2p_i^2)$. Then the answer is $0.5 + p_n$.

A: Alternating Branch Cutting

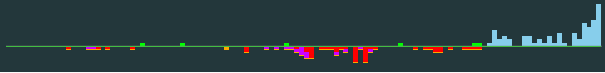
Problem author: Yanxu Chen



Pitfall: The input does not fit inside an extended precision floating-point value, due to the massive exponent. Need to manually keep track of the mantissa and the exponent and simply estimate $p_{i+1} = 1.5p_i$ when p_i is still very small.

A: Alternating Branch Cutting

Problem author: Yanxu Chen

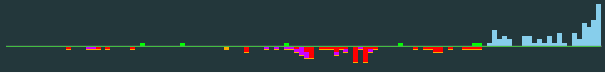


Pitfall: The input does not fit inside an extended precision floating-point value, due to the massive exponent. Need to manually keep track of the mantissa and the exponent and simply estimate $p_{i+1} = 1.5p_i$ when p_i is still very small.

Pitfall: Make sure negative values are handled correctly.

A: Alternating Branch Cutting

Problem author: Yanxu Chen



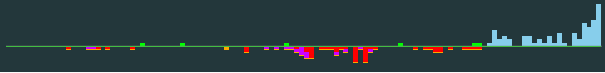
Pitfall: The input does not fit inside an extended precision floating-point value, due to the massive exponent. Need to manually keep track of the mantissa and the exponent and simply estimate $p_{i+1} = 1.5p_i$ when p_i is still very small.

Pitfall: Make sure negative values are handled correctly.

Pitfall: `decimal` in PyPy is slow. `p**3` is $30x$ slower than `p*p*p`. `decimal` in CPython is also 4 times faster...

A: Alternating Branch Cutting

Problem author: Yanxu Chen



Pitfall: The input does not fit inside an extended precision floating-point value, due to the massive exponent. Need to manually keep track of the mantissa and the exponent and simply estimate $p_{i+1} = 1.5p_i$ when p_i is still very small.

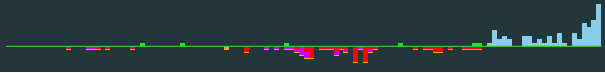
Pitfall: Make sure negative values are handled correctly.

Pitfall: `decimal` in PyPy is slow. `p**3` is $30x$ slower than `p*p*p`. `decimal` in CPython is also 4 times faster...

Running time: Linear $\mathcal{O}(n)$. Also possible in constant time $\mathcal{O}(1)$ by exploiting the loose precision requirements, only explicitly calculating p_i when $10^{-20} < |p_i| < 0.5 - 10^{-10}$.

A: Alternating Branch Cutting

Problem author: Yanxu Chen



Pitfall: The input does not fit inside an extended precision floating-point value, due to the massive exponent. Need to manually keep track of the mantissa and the exponent and simply estimate $p_{i+1} = 1.5p_i$ when p_i is still very small.

Pitfall: Make sure negative values are handled correctly.

Pitfall: `decimal` in PyPy is slow. `p**3` is $30x$ slower than `p*p*p`. `decimal` in CPython is also 4 times faster...

Running time: Linear $\mathcal{O}(n)$. Also possible in constant time $\mathcal{O}(1)$ by exploiting the loose precision requirements, only explicitly calculating p_i when $10^{-20} < |p_i| < 0.5 - 10^{-10}$.

Statistics: 139 submissions, 6 accepted, 72 unknown