CSCI 373

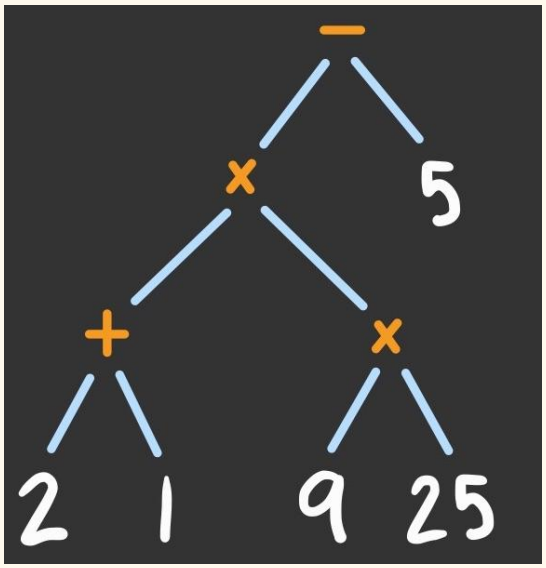
8 Out of 10 Cats Does Search

One

8 Out of Cats Does Countdown is a British comedy/game show (based on a longer-running, more conventional game show called Countdown) that features two games: one word-based (the "letters game") and one arithmetic-based (the "numbers game"). This question focuses on the numbers game, an example of which can be watched at:
https://www.youtube.com/watch?v=xkBUKoZ3qC4

The game provides players with a target number, e.g. 670, and six smaller numbers (e.g. 25, 9, 5, 4, 1, 2). All the numbers are positive integers. The objective is to compute the target number using arithmetic operations on the smaller numbers. For instance: 670 = ((2+1)*(9*25))-5. Further rules:

- You don't need to use all the numbers.
- You may only use each number at most once.
- The only permitted operations are addition, subtraction, multiplication, and division.



Another way to express these rules: the objective is to create a binary operator tree where the leaves are a subset of smaller numbers, such that the value of the operator tree is equal to the target number.

$Q =$

$\Sigma =$

$\Delta =$

$q_0 =$

$F =$

Define a state machine that formalizes this problem.

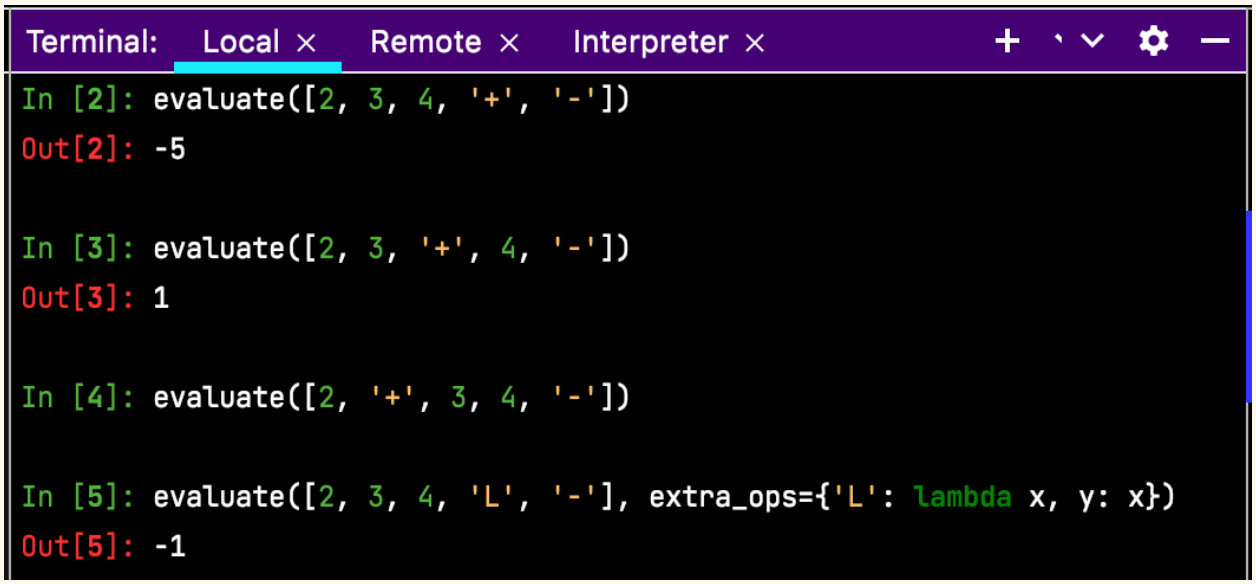
What is the branching factor?
What is the solution depth?
What is the maximum depth?

Given the characteristics of the state machine, what is the best search strategy: BFS, DFS, or IDS? Justify your answer.

Tips:

- This is an exercise in formalization and analysis, not optimization. You are not being graded on the efficiency of your state machine, but rather the precision with which you express it. Mathematical notation is preferred to English statements like "Q is a set of binary trees with operators labeling their internal nodes," but some English is fine (and probably necessary), as long as it is precise.
- For simplicity, you may assume that the six smaller numbers are all distinct.
- You may use the following notation without defining it:
    - For a finite set X, let S(X) be the set of permutations of X
    - For a finite set X and a positive integer k, let C(X, k) be the set of samples of k elements from X (without repetition).
    - For a finite set X and a positive integer k, let `mset`(X, k) be the set of samples of k elements from X (with repetition).
- In your formalization, you may assume access to a function called **`evaluate`** that computes the value of a postfix expression (see

https://en.wikipedia.org/wiki/Reverse_Polish_notation), or returns None if the expression is malformed. Besides the postfix expression itself, the function takes an additional optional argument called extra_ops, which allows you to define additional binary operators (for instance, below we define a binary operator called “L” that simply returns its left argument).



let $\Omega = \{+, -, *, \div\}$

let N be the six smaller numbers.

let t be the target number

$$Q = \{[]\} \cup \bigcup_{k=1}^{6} \{s \mid \Omega_{k-1} \in \text{mset}(\Omega, k-1), N_k \in C(N, k), s \in S(N_k \cup \Omega_{k-1})\}$$

$$\Sigma = N \cup \Omega$$

$$\Delta = \{(q, \sigma, q + [\sigma]) \mid q \in Q, \sigma \in \Sigma\}$$

$$q_0 = []$$

$$F = \{q \in Q \mid \text{evaluate}(q) = t\}$$

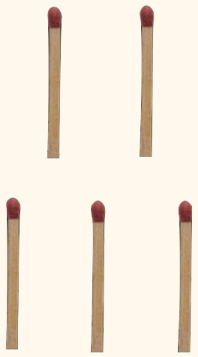
there are 10 possible actions (4 operators and 6 numbers) so the branching factor is 10.

each action appends exactly one symbol to a sequence, starting with the empty sequence. since the longest sequences have length 11, the maximum depth is 11

the solution depth depends on the choice of numbers

since the solution depth might be considerably smaller than the maximum depth, we should use bfs or ids to take advantage of their $O(b^d)$ runtime. but since b^d might be 10^{11}, bfs might run out of memory — thus ids is probably the best choice

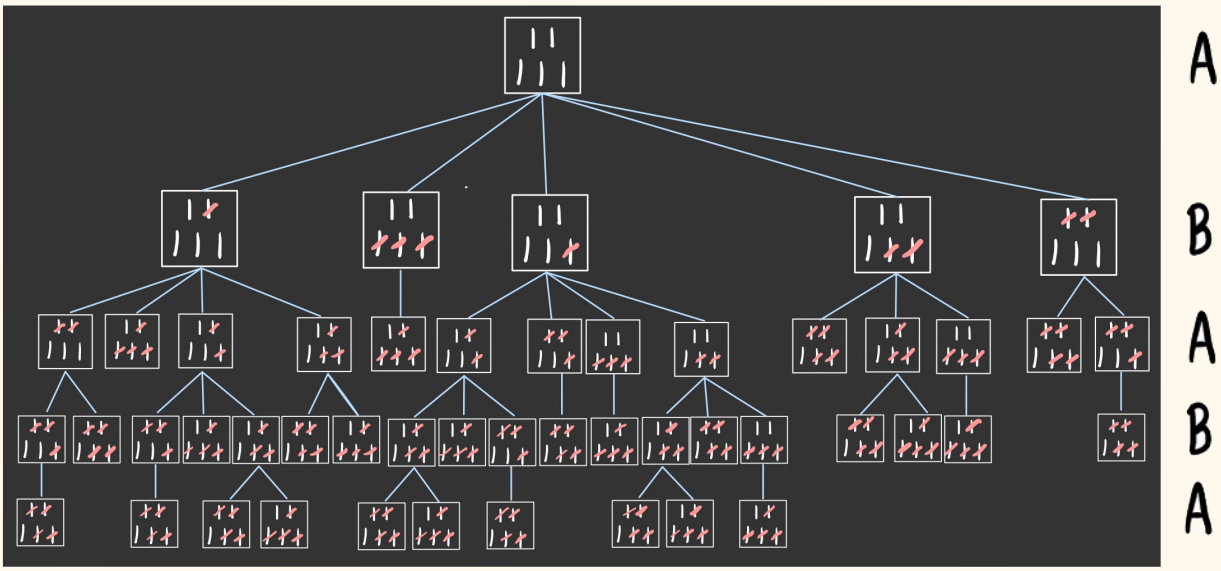
Two



In the game of Nim, two players take turns removing one or more matchsticks from a single row (assume that these matchsticks are always removed right to left). The player who removes the last matchstick loses the game.

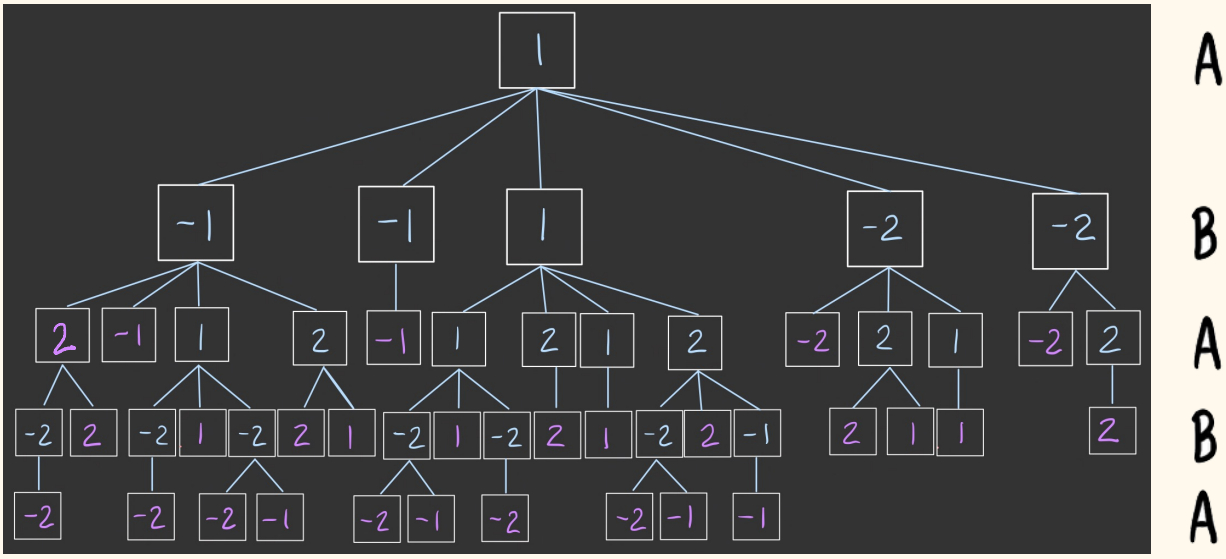
Suppose we start the game with the setup on the right, where row 1 contains two matchsticks, and row 2 contains three matchsticks. We will adopt the following scoring system: if you force your opponent to remove the last matchstick from row 1, you get 1 point, but if you force your opponent to remove the last matchstick from row 2, you get 2 points.

Complete the minimax search tree by filling in the missing states:

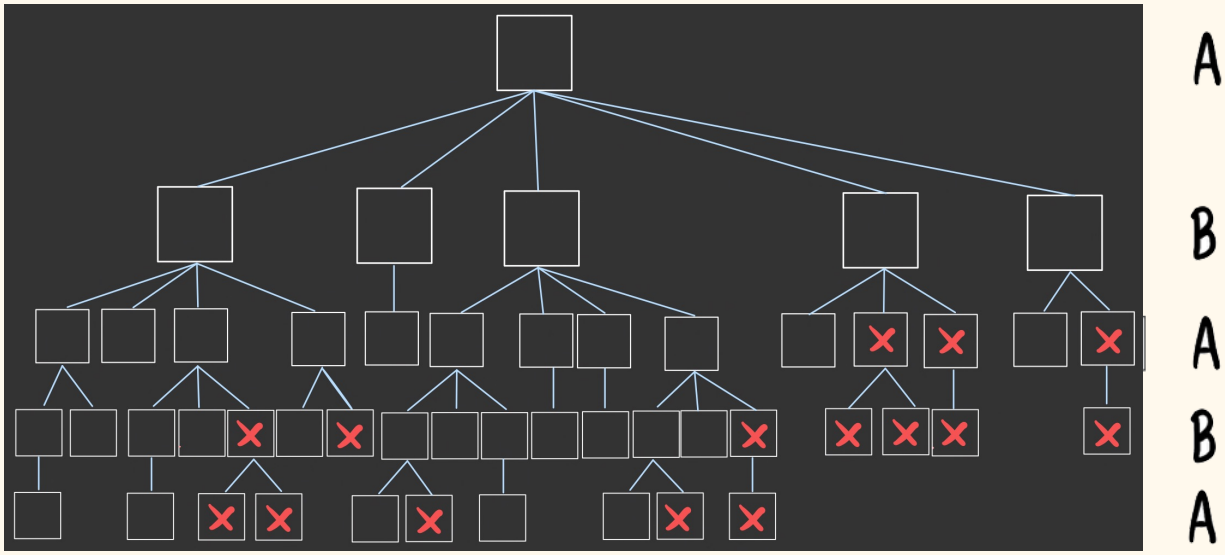


Note that you do not have to fill in the current player for each node. Each level of the tree is already labeled with the current player. Moreover, assume that a player never takes all the remaining matchsticks unless they're forced to, thus each state contains at least one unremoved matchstick.

Next, fill in the minimax value of each node:



Now draw an X on the search nodes that are not visited by minimax search if you use alpha-beta pruning. Assume that children are visited in a left-to-right order.



How might we make this search more efficient from a runtime perspective, using one of the techniques previously introduced in the context of single-agent search? Hint: the technique I'm thinking of would not work as well if each state included information about which player removed which match.

there are a lot of duplicate states in the search tree. rather than recomputing the minimax value of a previously encountered state, we can store (i.e. memoize) the minimax values of previously encountered states in a hash table, and simply look up the value when we re-encounter a state