The 24 Game

Mark Cabanero

October 16, 2016

Program Design

Description

The 24 game requires players to try and use the basic mathematical operators $(+, -, \times, \div)$ and four cards to try and get a result of 24. To achieve this programatically, `cards.java`:

1. generates all 1,820 $\binom{4+13-1}{4}$ unique combinations for the 24 game.
2. generates all 64 (4^3) unique combinations for the operators.
3. generates all 24 $(4!)$ unique permutation per card combination.
4. generates all 5 possible postfix strings per operator combination and card set permutation.

The last three alone requires 7,680 calculations to see if just one unique card combination can equal 24, resulting in $\sim$14 million calculations if run to completion. This doesn't include the amount of calculations for creating the unique permutations per card combinations. The program does this with a series of functions to prevent unnecessary bloat per method.

Trade Offs

This program unfortunately runs too slowly to be considered efficient. The amount of calculations alone cross over to approximately 14 million, but the inefficiency lies within the unique permutation per card combination. Considering that the permutations are calculated through recursion and depth-first search, one permutation would run in the time to traverse the height of the tree. Since the height of the tree is four (four items until the four items are swapped), this will run 24 times for the amount of various

permutations there are. There are other ways to run this faster, but this is a straightforward implementation in order to get all the permutations per combination.

Also, storing all of these combinations in linked lists make sense: the insertion is fairly straight forward and in constant time, and the items only need to be accessed in sequential order. In order to make iteration easy, an Iterator was called to iterate over all the unique combinations.

For any string creation (whenever the postfix notation steps were printed, or postfix notation expression building), StringBuilder was used in order to cut down on time in building the string. Even though the complexity and the size of the string does not benefit immensely, it leads to overall faster code.

Extensions

- Check that no negative values enter in the stack, and prematurely cutting the evaluation short. Typically, the 24 game does not deal with negative numbers, and this would stay more truthful to the game, yet seeing a solution where two negative numbers multiplied together to get 24 is always interesting.

- Check that no fractional values enter in the stack, and prematurely cutting the evaluation short. Just as above, fractions are rarely used in the 24 game, yet there are some interesting cases where the fractions ended up equaling 24. There is a potential for possible deviations from 24 due to decimal representation in computer systems, but on the whole, it shouldn't impact the final result too greatly.

- Give an option for calculating and printing out fractional/negative solutions. Considering the unique solutions that come about, only printing out straightforward integer solutions would be a lot easier for verification, but not printing those out allows for more unique solutions to come about. Also, this would require many overloaded/optional operators with logic to perform the correct calculation. In the short-term, this is not necessary for the project, but allows for possible expansion.

Test Cases

Considering that these test cases were generated from brute force of all the permutations per operator combinations per postfix combinations, there was

a lot of unprinted tests and checks that went on. To test those that came out, three random evaluated statements were checked to see if they are accurate: one with only positive numbers, one that deals with negative numbers, and one that deals with fractions.

- 5 10 + 13 - 12 *

$$\begin{aligned} 5.0 + 10.0 &= 15.0 \\ 15.0 - 13.0 &= 2.0 \\ 2.0 * 12.0 &= 24.0 \end{aligned}$$

- 8 3 13 4 - / /

$$\begin{aligned} 13.0 - 4.0 &= 9.0 \\ 3.0/9.0 &= 0.3333333333333333 \\ 8.0/0.3333333333333333 &= 24.0 \end{aligned}$$

- 2 4 13 13 + - -

$$\begin{aligned} 13.0 + 13.0 &= 26.0 \\ 4.0 - 26.0 &= -22.0 \\ 2.0 - -22.0 &= 24.0 \end{aligned}$$

All of these test cases end up coming up true, especially the weird fractional one. $\frac{3}{9} = \frac{1}{3}$, and $\frac{8}{\frac{1}{3}} = 8 \times 3 = 24$.

Questions

1. To evaluate postfix notation nonrecursively, numbers are pushed onto the stack until an operator is encountered. Then, the top two numbers are taken off the stack. Order is important for subtraction and division; the first number popped actually is the number subtracting/dividing. Once the last operator is evaluated, then the last value on the stack will be the result.