Wanted to use constraint satisfaction. Turns out I need the best solution, not just any solution. Could have used more logic to backtrack maybe…

Went with a DFS, only 9 layers deep. But that is still a huge state space 9 X 100 per position.

Tried to optimize, sorted lists and pruned after max value found. Turned out not to help.

With 3 players at each position all 9 positions selected:

Timing with sorted list pruning:

; cpu time (non-gc) 3.656250 sec user, 0.000000 sec system

; cpu time (gc) 1.031250 sec user, 0.000000 sec system

; cpu time (total) 4.687500 sec user, 0.000000 sec system

; real time 4.688000 sec (99.99%)

; space allocation:

; 16,191,327 cons cells, 210,185,344 other bytes, 0 static bytes

; Page Faults: major: 0 (gc: 0), minor: 0 (gc: 0)

Timing without sorted list pruing:

; cpu time (non-gc) 2.250000 sec user, 0.062500 sec system

; cpu time (gc) 0.656250 sec user, 0.000000 sec system

; cpu time (total) 2.906250 sec user, 0.062500 sec system

; real time 2.968000 sec (100.0%)

; space allocation:

; 10,242,685 cons cells, 129,305,088 other bytes, 0 static bytes

; Page Faults: major: 0 (gc: 0), minor: 0 (gc: 0)

Removal of 2 check functions: With pruning:

; cpu time (non-gc) 5.078125 sec user, 0.031250 sec system

; cpu time (gc) 1.156250 sec user, 0.000000 sec system

; cpu time (total) 6.234375 sec user, 0.031250 sec system

; real time 6.277000 sec (99.82%)

; space allocation:

; 20,923,701 cons cells, 274,048,936 other bytes, 0 static bytes

; Page Faults: major: 0 (gc: 0), minor: 0 (gc: 0)

Without pruning returns:

; cpu time (non-gc) 3.500000 sec user, 0.031250 sec system

; cpu time (gc) 1.093750 sec user, 0.000000 sec system

; cpu time (total) 4.593750 sec user, 0.031250 sec system

; real time 4.623000 sec (100.0%)

; space allocation:

; 15,076,292 cons cells, 195,363,272 other bytes, 0 static bytes

; Page Faults: major: 0 (gc: 0), minor: 0 (gc: 0)

FAST LOAD:

; cpu time (non-gc) 0.015625 sec user, 0.015625 sec system

; cpu time (gc) 0.015625 sec user, 0.000000 sec system

; cpu time (total) 0.031250 sec user, 0.015625 sec system

; real time 0.055000 sec (85.23%)

; space allocation:

; 8,217 cons cells, 1,101,160 other bytes, 0 static bytes

; Page Faults: major: 0 (gc: 0), minor: 0 (gc: 0)

10 “random” players each position, all 9 positions filled:

; cpu time (non-gc) 784.312500 sec (00:13:04.312500) user, 0.406250 sec system

; cpu time (gc) 355.140625 sec (00:05:55.140625) user, 0.234375 sec system

; cpu time (total) 1139.453125 sec (00:18:59.453125) user, 0.640625 sec system

; real time 1152.828000 sec (00:19:12.828000) ( 98.9%)

; space allocation:

; 867,244,020 cons cells, 2,853,146,528 other bytes, 0 static bytes

; Page Faults: major: 0 (gc: 0), minor: 0 (gc: 0)

Given 9 numbers that add to 55000 then create lists of all the players at each position that fill them.

Search through those lists

Originally had 6 function calls that loop through each lineup in each section. That adds up! (upto 9 cycles X 6 calls X MILLIONS of recursions). Use let statement to only calculate 3 and then compare 4 times.