Results

Part 1: Divide & Conquer

Questions

1. Suppose the player starts with 5 tokens and wants to play 1 round? What is the expected number of tokens they will have after 1 round? How about 10 tokens and 1 round? Is there a commonality to the expected number of tokens won/lost after playing 1 round, regardless of the starting number of tokens?
2. Suppose the player starts with 5 tokens and wants to play 5 rounds? What is the expected number of tokens they will have after 5 rounds? How about 10 tokens and 5 rounds? Is there a commonality to the expected number of tokens won/lost after playing 5 rounds, regardless of the starting number of tokens?
3. Based on your experiments, does the starting number of tokens have any impact on the amount of work (assuming the number of rounds is the same)? Can you explain why that would be the case?
4. Is there a practical limit to how many rounds your expectedTopDown method can handle? If so, what is that limit?

Answers

1. For five tokens and one round, 4.8888888888889. For ten tokens, one round, 9.8888888888889. Regardless of the starting number of tokens, every round the use players, they lose ~0.111111111111 tokens.
2. For five tokens and five rounds, 4.4444444444443. For ten tokens and five rounds, 9.4444444444443. Regardless of the starting number of tokens, they both end with ~0.555555555555 removed from their original balance.
3. No, there is no difference in timing since tokens are not the limiting factor; the number of rounds is.

Tokens	Rounds	Results	Time
50	5	49.44444444444444	2 ms
500,000	5	499999.4444444442	1 ms
5,000,000	5	4.999999444441E7	1 ms

4. Twenty rounds seems to be a practical limit, as it takes more than six minutes. This is clearly exponential.

Tokens	Rounds	Time
50	17	21320 ms
50	18	49987 ms
50	19	348302 ms

Part 2: Bottom-Up

Questions

1. Does this new method return the same answers as the previous one? How would you go about testing this?
2. Is this version noticeably faster? If so, how long can the games be before the delay using expectedBottomUp is significant?
3. If you start with a small number of tokens (e.g., 10) and play a significantly larger number of rounds (e.g., 100), how many tokens are you expected to win/lose? If you play the same number of rounds but with a comparably large number of starting tokens (e.g., tokens = rounds), are the expected wins/losses the same or different? Can you provide an explanation?'

Answers

1. Yes, this method does. I ran the same tests as the initial method test.

Tokens	Rounds	Results
5	1	4.8888888888888889
10	1	9.8888888888888889
5	5	4.4444444444444443
10	5	9.4444444444444443

2. Yes, this version is much faster.

Tokens	Rounds	Time
50	19	1 ms
50	2,000	58 ms
50	5,000	167 ms
50	10,000	764 ms
50	15,000	1927 ms
50	20,000	`java heap space` error

The issue ends up being in heap space, not the time it takes to complete.

3. In the case of 100 rounds and 10 tokens, you lose about 11.111, which places you into the negatives (-1.11111111). In the case of 100 rounds and 100 tokens, you lose 11.1111111, which is the same as the first scenario. Again, the amount of tokens lost is $\text{rounds} \times 0.1111$.

Part 3: Caching

Questions

1. Does this new method return the same answers as the previous two? How would you go about testing this?
2. How does the method compare with the other two in terms of speed? Can it handle longer games than `expectedTopDown`? How does it compare with `expectedBottomUp`?

Answers

1. Yes, this method does. I ran the same tests as the initial method test.

Tokens	Rounds	Results
5	1	4.8888888888888889
10	1	9.8888888888888889
5	5	4.4444444444444443
10	5	9.4444444444444443

2. It can handle much longer games than `expectedTopDown`, but it cannot handle as long of games at `expectedBottomUp`, nor does it complete the ones it can handle in thet same amount of time.

Tokens	Rounds	Time
50	19	1ms
50	2,000	879 ms
50	5,000	6,014 ms
50	10,000	`java.lang.StackOverflowError`