COSC241 Assignment – Deal With It – Tobias Meyrick (1998580)

Report

1. Consider the count values resulting from the "pick up by rows" specification (those beginning with an L or R). What values do they take and why?

When picking up by rows (left top, left bottom, right top, right bottom), the resulting values for the count method are always 2 except for left top which takes only 1 transformation to get back to its original pile. This hold no matter how big the deck is or how many rows there are. Left top is just putting the cards down from left to right, then top to bottom so nothing really changes from the original configuration which is why it only has count of 1.

2. What is the maximum count value produced for any specification and any pile size of 20 or less? What pile size(s), row length(s) and specifications(s) produce it? Given a pile size, row length, and specification can you think of a way of computing it's count that doesn't rely on actually carrying out that many transformations?

The maximum count value for any given specification and pile size of 20 or less is 18. There are eight instances where this can happen. I show above the results in a screenshot of a test class I created to figure this out. I only show the results with the biggest count values of course.

```
Tobiass-MBP:PracticeCode tobiasmeyrick$ java Test
BL, pile size: 18, row length: 9, count: 18
BL, pile size: 18, row length: 6, count: 18
TR, pile size: 18, row length: 3, count: 18
TR, pile size: 18, row length: 2, count: 18
TL, pile size: 20, row length: 10, count: 18
BR, pile size: 20, row length: 5, count: 18
BR, pile size: 20, row length: 4, count: 18
TL, pile size: 20, row length: 2, count: 18
Tobiass-MBP:PracticeCode tobiasmeyrick$
```

3. There are 720 possible card piles consisting of the numbers 1 through 6 in some order. Call such a pile accessible if it can be reached from the original 123456 by some sequence of transformations. How many accessible piles are there? What about seven, eight or nine card piles? For how large a value of n do you think it might be feasible to compute the number of accessible piles (and why)?

I struggled firstly to come up with any sort of solution to this problem of finding accessible piles. At the end of it, I managed to come up with an algorithm, although based on my findings, I'm unsure of how correctly I've implemented it. I found through a recursive program that for card piles consisting of the numbers 1 through 6, there are 48 accessible piles. My hypothesis was that there would be a lot more accessible piles as it seems that when each transformation result can take on another 8 transformations and so forth, the exponential possible sequences should yield a massive amount of results. Because I only found that about 6.67% of the possible card piles are accessible, this either proved my hypothesis incorrect or has shown my program to be incorrect. I ran the program again, but changed the card pile size to seven, and updated the array size to a possible 5040 card piles as I knew that 7! Equals 5040. This time there was only 2 accessible piles, probably because the only factor of 7 is 1 and 7 which yield uninteresting results. Card piles of size 8 yield 24 accessible piles and a card pile of size 9 yields just 8 accessible piles. These last two results make me very suspicious of my program. If my program was correct, there would be no reason to think we couldn't really stretch the boat out and try massive values of n. This is where things got

interesting. I tried n=12, and I waited for 10 seconds before a huge amount of stack overflow errors appeared. This would tell me that 12 is too big of a pile. I tried a pile of size 10, and there are apparently 1920 accessible piles. This seems a lot more reasonable to me. Therefore n cannot be 12, and likely cannot be any non-prime number beyond 12. The reason for this is that prime numbers don't have any possible row lengths beyond 1 and n. At pile size 12, you can have row lengths 2, 3, 4 and 6, and there are 479,001,600 possible card piles. At this level of recursion, the stack becomes overloaded and you run into stack overflow issues. One way of avoiding this would be to provide a solution that is non-recursive. Still at a sufficiently large pile, this program would take a long time to run and require a lot of memory.