ZHAOKAI XU (A14738474)

JIAYI ZHANG(A92062390)

Report: DP vs Recursion

Overall, the naïve solution is extremely time-inefficient compared to the two other implementations, while both the 1st and the 2nd solutions using recursion and memorized recursion have stack-space constraints. The last implementation is the most efficient one, which takes O(N*F) to build the DP table and produce the results, while the naïve recursion takes O(N^F) by breaking the problem into N^F sub-problems and solve each with O(1). The only improvement of memorized recursion is by adding a map to cache the already calculated results, but still costly for each function call.

For (N=1000 and F = 20), it is already taking the naïve algorithm a very long time (can't even be timed) to finish while it takes very short time for the other two implementations. For (N = 1000000 and F = 100), both of the two recursion methods give a seg fault, while the DP solution (3rd) is still efficient.

```
→  W18-PA3 git:(master) ✗ cat testcases/usb100.txt 1
1000000
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 5
9 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86
87 88 89 90 91 92 93 94 95 96 97 98 99 100
cat: 1: No such file or directory
→  W18-PA3 git:(master) ✗ build/TestUSB testcases/usb100.txt 1
[1]    42517 segmentation fault  build/TestUSB testcases/usb100.txt 1
→  W18-PA3 git:(master) ✗ build/TestUSB testcases/usb100.txt 2
[1]    42559 segmentation fault  build/TestUSB testcases/usb100.txt 2
→  W18-PA3 git:(master) ✗ build/TestUSB testcases/usb100.txt 3
Result of find_files_dp: 10000
printf: 4.56386
```

Part1: naïve solution analysis

The rough estimation of the number of subproblem using recursion is O(N^F) where N represents the size of USB and F represents the number of is files in the files list. For each function call, we run the iteration the number of files times to make a recursive call by passing in the USB size to be the the origin USB size deducting the current file size.

For example, given 5 to be the USB size and 3, 2, 1 to be the size of files in a list. F(5) recursively calls F(2), F(3) and F(4) in the iteration, then F(2), F(3) returns 1 because files with size 2 and 3 are found in the list, and then F(4) calls F(1), F(2) and F(3) recursively in another iteration. From the above example, we can see that the naïve solution recursively computes overlapped sub-problems. Since it does not cache the results of sub-problems, some sub-problems will be calculated multiply times.

Part2: memorized comparison

Memorization improves the naïve solution by recording any results of calculations when encountering a new subproblem, so that no subproblem is calculated more than once. Once encountered a repeating subproblem, its value, which is stored in the "memorize" map, is immediately returned. For a test case n = 10000 and F = 100 (1-100), the naïve solution was unable to finish in a reasonable amount of time while memorization + recursion took only 0.942 sec (by adding a timer in testUSB) to finish the task.

```
→  W18-PA3 git:(master) ✗ vim testcases/usb100.txt
→  W18-PA3 git:(master) ✗ build/TestUSB testcases/usb100.txt 3
Result of find_files_dp: 100
printf: 0.059903
→  W18-PA3 git:(master) ✗ build/TestUSB testcases/usb100.txt 2
Result of find_files_memoized: 100
printf: 0.942764
```

Part3: DP comparison

Comparing the runtime between memorized recursion and dynamic programming, dynamic programing takes a bottom-up iterative approach starting from the smallest possible USB size, while the memorized recursion still makes many recursive function calls in the runtime stack, which is costly. For the performance comparison, we used the test case of (F=20000 & n = 100 (1-100)). It took the recursive implementation 1.69 sec to complete while it took the iteration implementation only 0.079 sec. The bottom-up approach not only avoids abusing the stack, but also improves the time/space efficiency for solving this problem.

```
→  W18-PA3 git:(master) ✗ cat testcases/usb100.txt
20000
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 2
 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56
9 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84
87 88 89 90 91 92 93 94 95 96 97 98 99 100
→  W18-PA3 git:(master) ✗ build/TestUSB testcases/usb100.txt 2
Result of find_files_memoized: 200
printf: 1.69085
→  W18-PA3 git:(master) ✗ build/TestUSB testcases/usb100.txt 3
Result of find_files_dp: 200
printf: 0.07911
```