CSC510 Homework 3

Chris Camano: ccamano@sfsu.edu

November 4, 2022

Question 1

Describe a dynamic programming algorithm for determining whether you can buy exactly n cupcakes using boxes of 4,6 , and 9 cupcakes. Your answer should be boolean-valued function of the following form:4

CUPCAKEDP(n) :
// your code here

What's the runtime of your algorithm?

Recall the previous recursive cupcake algorithm ;

Algorithm 1 CUPCAKE (n)

```
1: if n=0 then
       return True
2: end if
3: if n < 0 then
       return False
4: else
       return  CUPCAKE(n-9) or CUPCAKE(n-6) or CUPCAKE(n-4)
```

A dynamic programming version of this algorithm would be:

The runtime of this algorithm is =O(n)

Algorithm 2 DPCUPCAKE (n)

```
Init C=[1,...,max(n,9)]
C[1]=False
C[2]=False
C[3]=False
C[4]=True
C[5]=False
C[6]=True
C[7]=False
C[8]=True
C[9]=True
for ( doi=10,...,n)
    C[i]=C[i-4] or C[i-6] or C[i-9]
end forreturn C[n] =0
```

Question 2

Actually, there's a faster way of doing the previous problem, assuming arithmetic operations can be done in constant time. Give a constant-time algorithm for the above problem, and prove the correctness of your algorithm. (Hint: try coding up your algorithm in the previous part. Do you see a pattern?)

Allow us to deploy some light number theory here: Given an integer of desired cupcakes we can begin by computing the value modulo 10. If the remainder is either 0,4,6,or 9 then we know that we can form an expression as follows:

$$k(4+6)+r \quad k \in \mathbb{Z}, r \in 0,4,6,9$$

Using this method we also get a free solution for when the remainder is 8 since 8=4(2)
Great so we have covered almost half of the possible remainders with this expression:
For values 1,2,3,5,7 we will need to deploy some more customized reasoning. Do there exist linear combinations of our three possible box sizes whose remainder mod 10 is equal to the numbers listed above?: Here is proof of the first occurence that we see the digit listed:

$$\begin{aligned}
9(1)+6(2) &= 21 \mod 10 = 1 \\
6(2) &= 12 \mod 10 = 2 \\
9(1)+4(1) &= 13 \mod 10 = 3 \\
9(1)+6(1) &= 15 \mod 10 = 5 \\
9+4(1) &= 17 \mod 10 = 7
\end{aligned}$$

Thus we have demonstrated that there exist some linear combination of our box sizes that form the full residue system of $\mathbb{Z}_n$ for certain size conditions. In code:

Algorithm 3 CUPCAKE (n)

```
if n mod 10=0 or 4 or 8 or 6 or 9 then
    return True
else if n mod 10=1 and n>20 then
    return True
else if n≥10 then
    return True
else
    return False
end if
```

Question 3

The SFSU Robotics Club used your algorithm from the previous homework and found that it was too slow for large mazes. They've asked you to come up with a more efficient algorithm for the same problem, i.e., counting the number of solutions for a robot to traverse a maze using only down and right steps:

Describe a dynamic programming algorithm for finding the number of ways for the robot to navigate from the top-left cell (coordinate $(1,1)$) to the bottom-right cell (coordinate (m,n), where $1 \leq i \leq m$ and $1 \leq j \leq n$) of the maze. Your answer should be an integer-valued function of the following form:

MaZEROBOTDP$(W[1,\ldots,m][1,\ldots,n])$:
/ / your code here

What's the runtime of your algorithm? (Hint: this should resemble LCS)

Previous algorithm:

Algorithm 4 MAZEROBOT $(W[1,\ldots,m][1,\ldots,n],i,j)$

```
if W[1][1]=TRUE then
    return 0
end if
if W[i][j]=TRUE then
    return 0
end if
if i < 1 or j < 1 then
    return 0
end if
if i=1 and j=1 then
    return 1
else
    return MAZEROBOT(W,i-1,j)+MAZEROBOT(W,i,j-1)
```

Algorithm 5 MAZEROBOTDP $(W[1,\ldots,m][1,\ldots,n],i,j)$

```
1:  Init S[0,..,m][0,...,n]
2:  for i=0,...,n do
3:      S[0][i]=0
4:
5:  end for
6:  for i=0,...,m do
7:      S[i][0]=0
8:
9:  end for
10: for i=1,...,m do
11:     for j=1,...,n do
12:         if W[i][j]=0
13:             S[i][j]=S[i][j-1]+S[i-1][j]
14:         else
15:             S[i][j]=0
16:         end if
17:     end for
18: end for
        return S[m][n]  then =0
```

Above is the improved algorithm :
Small bug with =0 being printed at the end of my algorithms please ignore.

The runtime of this algorithm like LCS is O(mn)

Question 4

Run the dynamic programming algorithm, including reconstruction, to find a longest common subsequence of the strings STREET and TESTER. Your final answer should be a string, not just the length.

I have implemented the algorithm given by the pseudocode in the lecture and determined that the longest common subsequence is STE.An equivilant common subsequence is STR.

Weirdly enough depending on the order of the arguments in my function I got different answers: passing in(STREET,TESTER) Returned TE passing (TESTER,STREET) returned STE.
This pattern also continued for the BAUHAUS ABACUS example and was able to be replicated with a different recursive algorithm I found online that also computes the longest common subsequence. Here is the matrix I generated during DP for TESTER STREET

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 2 & 2 & 2 \\ 0 & 1 & 1 & 1 & 2 & 2 & 2 \\ 0 & 1 & 2 & 2 & 2 & 2 & 3 \\ 0 & 1 & 2 & 2 & 3 & 3 & 3 \\ 0 & 1 & 2 & 3 & 3 & 3 & 3 \end{bmatrix}$$