CSC421 Assignment 3

Matt Triano

May 18, 2017

1 Problem 1) Coin Change

Given $n = 10$, and the coin denominations $d_1 = 1, d_2 = 5, d_3 = 8$, illustrate the coin change algorithm.

i	d_j S.T. $d_j \leq i$ and $C[i - d_j]$ is min	$C[i].value + 1$	$C[i].collection$
0	N/A	$C[0] = 0$	N/A
1	$d_1 = 1$	1	$[1]$
2	d_1	2	$[1, 1]$
3	d_1	3	$[1, 1, 1]$
4	d_1	4	$[1, 1, 1, 1]$
5	$d_2 = 5$	1	$[5]$
6	d_1	2	$[5, 1]$
7	d_1	3	$[5, 1, 1]$
8	$d_3 = 8$	1	$[8]$
9	d_1	2	$[8, 1]$
10	d_2	2	$[5, 5]$

2 Problem 2) Pascal's Triangle

2.1 2a) Recursive Definition

For row i and column j (where $i \geq j \geq 1$), $C[i, j]$ gives the value at point in Pascal's Triangle. These values can be recursively calculated from the relation below.

$$C[i, j] = \begin{cases} 1 & j = 1 \; OR \; i = j \\ C[i-1, j] + C[i-1, j-1] & j \neq 1 \; AND \; i \neq j \end{cases} \tag{1}$$

2.2 2B) Recursive Algorithm

From the algorithm provided below and the photo of an excel table, you see that there are repeated function calls, so there are overlapping computations.

```
def pascal_recursive(i,j):
        if (i - 1 == 0) or (i = j):
                return 1
    else:
                return pascal_recursive(i-1,j-1) + pascal_recursive(i-1,j)
```

	A	B	C	D
1	C(1,1) = 1			
2	C(2,1) = 1	C(2,2) = 1		
3	C(3,1) = 1	C(3,2) = C(2,2) + C(2,1) = 2	C(3,3) = 1	
4	C(4,1) = 1	C(4,2) = C(3,2) + C(3,1) = 3	C(4,3) = C(3,3) + C(3,2) = 3	C(4,4) = 1
5	C(5,1) = 1	C(5,2) = C(4,2) + C(4,1) = 4	C(5,3) = C(4,3) + C(4,2) = 6	C(5,4) = C(4,4) + C(4,3) = 4
6	C(6,1) = 1	C(6,2) = C(5,2) + C(5,1) = 5	C(6,3) = C(5,3) + C(5,2) = 10	C(6,4) = C(5,4) + C(5,3) = 10

Figure 1: Pascal's Triangle Calculations ('C' used as a dummy function name due to space constraints)

2.3 2C) Dynamic Programming

I implemented this algorithm using nested loops and storing results in an $i \times i$ array. The operations in the inner loop operate in constant time, and as the loops are nested, the algorithm operates in $O(n^2)$ time.

```python
def pascal_dynamic(i,j):
    triangle = [[0 for r in range(i)] for c in range(i)]
    for row in range(i):
        for col in range(row+1):
            if (col == 0) or (row == col):
                triangle[row][col] = 1
            else:
                triangle[row][col] = triangle[row-1][col] + triangle[row-1][col-1]
            if (row == i - 1) and (col == j - 1):
                return triangle
```

3 3) Longest Common Subsequence

For $X = \langle A, C, T, C, C, T, G, A, T \rangle$ and $Y = \langle T, C, A, G, G, A, C, T \rangle$, the longest common subsequence is given in the tables below.

00	0	0	1	2	3	4	5	6	7	8	9
0	0	0	A	C	T	C	C	T	G	A	T
0	0	0	0	0	0	0	0	0	0	0	0
1	T	0	$\uparrow 0$	$\uparrow 0$	$\nwarrow 1$	$\uparrow 1$	$\uparrow 1$	$\nwarrow 1$	$\uparrow 1$	$\uparrow 1$	$\nwarrow 1$
2	C	0	$\uparrow 0$	$\nwarrow 1$	$\uparrow 1$	$\nwarrow 2$	$\nwarrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$
3	A	0	$\nwarrow 1$	$\uparrow 1$	$\uparrow 1$	$\leftarrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\nwarrow 3$	$\uparrow 3$
4	G	0	$\leftarrow 1$	$\uparrow 1$	$\uparrow 1$	$\leftarrow 2$	$\uparrow 2$	$\uparrow 2$	$\nwarrow 3$	$\uparrow 3$	$\uparrow 3$
5	G	0	$\leftarrow 1$	$\uparrow 1$	$\uparrow 1$	$\leftarrow 2$	$\uparrow 2$	$\uparrow 2$	$\nwarrow 3$	$\uparrow 3$	$\uparrow 3$
6	A	0	$\nwarrow 1$	$\uparrow 1$	$\uparrow 1$	$\leftarrow 2$	$\uparrow 2$	$\uparrow 2$	$\leftarrow 3$	$\nwarrow 4$	$\uparrow 4$
7	C	0	$\leftarrow 1$	$\nwarrow 2$	$\uparrow 2$	$\nwarrow 2$	$\nwarrow 3$	$\uparrow 3$	$\uparrow 3$	$\leftarrow 4$	$\uparrow 4$
8	T	0	$\leftarrow 1$	$\leftarrow 2$	$\nwarrow 3$	$\uparrow 3$	$\uparrow 3$	$\nwarrow 4$	$\uparrow 4$	$\uparrow 4$	$\nwarrow 5$

00	0	0	1	2	3	4	5	6	7	8	9
0	0	0	A	C	T	C	C	T	G	A	T
0	0	0	0	0	0	0	*0*	0	0	0	0
1	T	0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ *1*	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 1
2	C	0	$\uparrow$ 0	$\nwarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\nwarrow$ 2	$\uparrow$ *2*	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 2
3	A	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 1	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ *2*	$\uparrow$ 2	$\nwarrow$ 3	$\uparrow$ 3
4	G	0	$\leftarrow$ 1	$\uparrow$ 1	$\uparrow$ 1	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ *2*	$\nwarrow$ 3	$\uparrow$ 3	$\uparrow$ 3
5	G	0	$\leftarrow$ 1	$\uparrow$ 1	$\uparrow$ 1	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ *2*	$\nwarrow$ 3	$\uparrow$ 3	$\uparrow$ 3
6	A	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 1	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ *2*	$\leftarrow$ *3*	$\nwarrow$ 4	$\uparrow$ 4
7	C	0	$\leftarrow$ 1	$\nwarrow$ 2	$\uparrow$ 2	$\nwarrow$ 2	$\nwarrow$ 3	$\uparrow$ 3	$\uparrow$ *3*	$\leftarrow$ *4*	$\uparrow$ 4
8	T	0	$\leftarrow$ 1	$\leftarrow$ 2	$\nwarrow$ 3	$\uparrow$ 3	$\uparrow$ 3	$\nwarrow$ 4	$\uparrow$ 4	$\uparrow$ 4	$\nwarrow$ *5*

TCGAT (I coded this algorithm) should be the answer, but I must have made an error while working it out by hand.

4 4) Longest Monotonically Increasing Subsequence

To achieve this, rather than pass two different sequences to the LCS (**L**ongest **C**ommon **S**equence) function, we'll pass in the sequence, X, and a sorted copy of that sequence, X_{sorted}. As X_{sorted} is already monotonically increasing (by virtue of being sorted), we see that any matching sequence found in X will also be monotonically increasing. We know from the textbook and from class that running LCS on sequences of lengths m and n has a running time of $\Theta(mn)$, so if both sequences are of length n, then the running time of the LCS function on this problem will be $\Theta(n^2)$. As the sort takes $O(n \lg(n))$ time and the copy takes $O(n)$ time, we see that this function is dominated by $\Theta(n^2)$, which is upper-bounded by $O(n^2)$.

5 5) Subset Sum Problem

By far, the hardest part about this problem was the notation.

5.1 5a) Computation for T[0,k]

It is trivial to compute whether 0 elements of T can sum to the value k. $T[0,0]$ is true, as 0 elements can sum to a total of 0, but any $k \geq 1$ if false, as 0 elements cannot sum to more than 0.

5.2 5b)

5.2.1 If T$[i, s']$ exists and does not include s_i

If $T[i-1, s' - S[i]] = True$, then...

I'm not quite clear on this. By my understanding, if the column for s_i doesn't have a 1 in it (ie s_i isn't in the subset), then it can't be represented by that subset.

5.2.2 If T$[i, s']$ exists and does include s_i

It the table has a 1 in the column for s_i, you can find the subset by taking the adding the value $S[i]$ to an empty list A. Then let $diff = s_i - S[i]$ and find the subset of values where $S[i_1] <= diff$ and select the entry where $diff - S[i_1]$ is a minimum. Append $S[i_1]$ to the list A. Continue until $diff == 0$ is true, and A is your subset.

	0	1	2	3	...	s-1	s
0					...		
1					...		
2					...		
3					...		
...	...	...	...	...	...		...
n-1					...		
n					...		

5.3 5c) Algorithm

I implemented the algorithm in the functions below.

```python
def subset_sum(list_a, target):
    n = len(list_a)
    table = [[0 for row in range(target)] for col in range(n)]
    for i in range(n):
        table[i][0] = 1
        for s in range(1, target):
            if list_a[i] <= target:
                if table[i-1][s] == 1:
                    table[i][s] = 1
                else:
                    table[i][s] = max(table[i-1][s], table[i-1][s-list_a[i]])
    return table

def val_in_subset_sum(table, val):
    for i in range(len(table)):
        if table[i][val-1] == 1:
            return "TRUE"
    return "FALSE"

def table_printer(result):
    i_len = len(result)
    s_len = len(result[0])
    header = []
    for s in range(s_len+1):
        buffer = '| '
        if s >= 9:
            buffer = '|'
        header.append(str(s) + buffer)
    print(''.join(header))
    for i in range(i_len):
        row = [str(i) + '| ']
        for s in range(s_len):
            row.append(str(result[i][s]) + '| ')
        print(''.join(row))

def main():
    S = [1, 2, 4, 10, 20, 25]
    val = 18
    table = subset_sum(S, val)
    table_printer(table)
    print(str(val) + ' in this subset? ' + str(val_in_subset_sum(table, val)))
```

```
if __name__ == "__main__":
    sys.exit(main())
```