knopp_daniel_Midterm2

December 1, 2023

1 Midterm 2

Value: 100 points

Release Date: Nov 30th 12:30 pm

Due Date: Dec 1st 12:30 pm (Canvas) Late submission until Dec 1st, 1:30 pm (Canvas). **Submitting a late exam will result in a 10 point penalty. Please make sure to turn in your exam on time.**

You are not allowed to ask for any help from anyone while performing the task individually. Please complete the activity on your own.

Provide Jupyter Notebook with your solution. If possible add a PDF file of the final file as a backup.

1.1 Q1 (20 points) - Top K Frequent Elements

Given a non-empty array of integers, find the top k frequent elements.

To find the top k frequent elements in a non-empty array of integers, you can use a max heap. The max heap allows you to efficiently keep track of the frequency of each element and extract the top k frequent elements. This makes the process of finding the most frequent elements in an array of integers much easier and faster.

- Part A. Please explain your approach to the problem in 3-4 paragraphs and assess the complexity of your solution using Big O Notation. You may use an external library to implement this solution, but you must review and understand the complexity of the built-in algorithms to provide a descriptive argument. Please indicate which libraries you are including in your solution and why.
- Part B. As part of the task, you need to write and test your code. To test your algorithm, you need to use at least two arrays. One array, named `nums`, should be fixed and defined with values [1, 1, 1, 2, 2, 3]. The other array, named `nums_random`, should contain 100 elements, with each element being a random number between 1 and 5 (uniform distribution).

Part A Here

The methodology for this problem was quite simple. I used the property of a max heap to get the top k frequent values. Since a max heap always has the maximum value on the top, if I want the top k values I just need to pop the top node off k times (each time I pop of the node, the tree is reorganized by swapping the top node with the bottom node, then removing the bottom node, and finally sorting the new top node accordingly to maintain the max heap structure). Before creating my heap, I first used a dictionary to count the occurances of each number in the list and store the frequency of occurance as the values to each unique number key. Then, it was simpler for me to convert the dictionary into a tuple because the heapify() method would automatically sort tuples in a list based on the first element (which I made sure was the frequency).

I chose to use the heapq package instead of building my own heap for simplicity. I understand how these methods work and thus didn't find it necessary to recreate them since it was not required. As I mentioned earlier, once the heap was built I then just needed to sequentially pop off the top k elements and store their numbers (not frequencies) into a results list to return to the user.

The time complexity for this function is O(n + n + n + k*log(n)) -> O(n + k*log(n)) in the average case or O(n + n + n*log(n) + k*log(n)) -> O((n+k)*log(n)) in the worst case where n is the number of elements in the list and k is the number of top elements we are concerned with. Each of these are mentioned in comments in the code below in their respective orders. The most significant terms come from the heapify() step and the popping of the top k values from the heap. For the heapify() method, on average the elements will be organized into a max heap using n steps becuase in the best case it only takes the depth of the tree but for the worst case (when the array is in the opposite order it needs to be) it can take n*log(n) becuase you must swap a node every time you add a new one. Since the worst case scenario is quite rare, the average case is commonly what is used. For the popping code, each time a node is removed it must first be swapped with the end-node (node at the bottom of the heap), then removed, and then the new top node (which used to be the bottom one) must be swapped down the tree until the none of its children are greater than it - this results in a time complexity of O(log(n)).

The space complexity for this function is O(2u + 2u + k) -> O(u + k) where u is the number of unique elements in the input list and k is the number of top elemnts we care about. This is because we store the frequencies of each unique value in the input list first in a dictionary and then also in the tuple which gets sorted to become the heap. Then the results array stores the top k frequent elements. If we wanted to be more efficient, we could compute the frequencies directly with a list of tuples instead of as a dictionary first.

[ ]:
```
# Part B

# Import heapq library (will be used to sort a list such that it matches the␣
↪structure of a max heap)
import heapq

# Function to find the top k frequent elements
def topKFrequent(nums, k):

    # First, we must get the frequency of occurance of each value in the list -␣
↪we can do this by using a dictionary
    freq_dict = {}

    # Loop through all elements in the list and add them to the dictionary with␣
↪a frequency of 1, if the value already exists, increment the frequency value
    # Time Complexity: O(n) since we must loop through all elements in the list
    for num in nums:
        if num in freq_dict.keys():
```

```
            freq_dict[num] += 1
        else:
            freq_dict[num] = 1

    # To implement a heap, it is easiest to convert the dictionary into a list␣
↪of tuples
    # Time Complexity: O(n) in worst case with no duplicates since we must loop␣
↪through all elements in the dictionary
    freq_list = [(-freq, num) for num, freq in freq_dict.items()] # Note that␣
↪frequency is stored as a negative because the default behavior of heapq.
↪heapify() is to sort in ascending order (min heap)

    # Now, we can use the heapify() method of the heapq library to sort the␣
↪list into a max heap
    # Time Complexity: O(n) in average case or O(nlogn) in worst case
    heapq.heapify(freq_list) # Note that heapify will sort the 'nodes' based on␣
↪the first element of the tuple (in this case, the frequency)

    # Finally, we can loop k times and pop the top node from the max heap and␣
↪append it to a result list
    # Time Complexity: O(k) since we only need to loop k times
    results = []
    for _ in range(k):
        # Time Complexity: O(logn) since we must pop the top node from the heap␣
↪and on average the time complexity to propogate the new root node to the␣
↪bottom of the heap is O(logn)
        results.append(heapq.heappop(freq_list)[1]) # Note since each element␣
↪in the list is a tuple, we only need to grab the second element (index 1) to␣
↪get the value corresponding to the kth max frequency

    # Return the results list
    return results
```

[ ]:

```
# Import modules
import random

# Example usage
nums = [1, 1, 1, 2, 2, 3]
k = 2
print(f'top {k} frequent values in {nums} are: {topKFrequent(nums, k)}')  #␣
↪Output should be [1, 2]

#define an array with randoms numnbers
nums_random = [random.randint(1, 5) for i in range(100)]
```

```
print(f'top {k} frequent values in (sorted for visualization)␣
↪{sorted(nums_random)} are: {topKFrequent(nums_random, k)}')
```

```
top 2 frequent values in [1, 1, 1, 2, 2, 3] are: [1, 2]
top 2 frequent values in (sorted for visualization) [1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4,
4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
5, 5, 5, 5, 5, 5, 5, 5, 5, 5] are: [5, 1]
```

2 Q2 (20 points) - Number Of Ways To Make Change

Finding the number of ways to make change can be a tricky problem. If you're given an array of positive integers representing different coin denominations and a target amount of money, how can you determine the number of ways to make change for that amount using those denominations? This function requires you to solve that problem by returning the total number of ways to make change for the given target amount, assuming you have an unlimited number of coins available for each denomination.

Hints:

- To tackle the problem of making change for different amounts, it is recommended to create an array that lists the number of ways to make change for each amount between 0 and n, including 0. It's important to note that there is only one way to make change for 0, which is to not use any coins.
- To solve the problem, start by building up the array mentioned in Hint #1 one coin denomination at a time. Begin by finding the number of ways to make change for all amounts between 0 and n using only one denomination. Then, move on to finding the number of ways to make change using two denominations, and so on, until you have used all of the available denominations. This approach will help you arrive at the solution easily and efficiently.

Sample Input

```
n=6
denoms = [1, 5]
```

Sample Output

```
2 // 1x1 + 1x5 and 6x1
```

- Part 1. Explain your approach, including your thinking process, data structure used and the complexity of your solution in terms of time and space using Big O notation.
- Part 2. Make sure to code and thoroughly test your solution. Additionally, provide some samples to help test your code effectively. You can use US Dollars to test your code like us_denoms=[1, 5, 10, 20, 50, 100]

Part 1 Here

It took me a very long time to figure out how to approach this problem. I chose to use dynamic programming as it seemed like the most efficient method. This problem is very complex and hard to visualize, so I created a visualization table in my output to help explain what is going on in the algorithm. To tackle the problem, as with many dynamic programming solutions, I tried to break up the problem into smaller pieces. First, I wanted to start with only the smallest denomination and figure out how many ways I could produce each number from 0 to n using just that one coin. Of course, this is trivial and I can only ever make change for each number using my 1 available coin. Building on this base, though, I can then add the ability to use a second coin - the 5 cent piece (for example). The way this algorithms works is essentially nothing changes while I'm allowed to use the new coin until I can actually use it (this is what the if statement is for in my code below). Once I can use my new coin, I can still arrive at the specific amount using only 1 cent coins as I figured out earlier, but now I can also use a 5 cent piece. The total number of unique combinations of coins is then simply the number of unique combos from the previous row of the table plus the number of unique combinations of my current row in the table when I subtract the coin I am using from the current value. This can be visualized by each element in the [1, 5] row being the sum of the row above it plus and value 5 elements to the left (assuming you are passed the value of 5 and are able to use the new coin). This same procedure is repeated for each new coin that is added - with the difference being that you are summing the previos row with the value in the current row x elements to the left (where x is the value of the coin - for example [1, 5, 10, 25, 50, 100]).

To keep track of the unique combinations, I had to use a very complex data structure. The easiest way I could think of was to use a dictionary of lists of tuples. Let's break it down from the outside in. At the base level, I have a dictionary whos keys are the various possible amounts from 0 to n. The values of this dictionary are lists that contain all the possible unique coins that could be used to make change for the specific amount. Each element of this list is a tuple where each element of the tuple is the number of each coin that is used in the unique combination. The indices inside the tuple match the corresponding indices for each coin in the 'denoms' list that specifies the coins we can use. During the algorithm, each time I use a new coin I need to add the new coin to all possible combinations for the previous value (the element in the table x elements to the left). I do this by creating a helper function to sum tuples element-wise and each time I add a coin, I add a corresponding tuple that maps to just that individual coin (for example, adding a 10 cent coin would be the tuple: (0, 0, 1, 0, 0, 0) given that my denoms list is [1, 5, 10, 25, 50, 100]). At the end of the function, I can then convert each unique combination tuple into an easy-to-read string using another helper function I made. Finaly, I output the total number of combinations for the requested amount and all the explicit unique combinations possible.

The time complexity of this algorithms is O(n*d) where n is the target amount and d is the number of unique coin denominations. This is due to the nested for loop inside the algorithm; for each denomination, we compute all possible combinations for each incremental amount as the sum of 2 previously known (memorized) values in the ways array. The space complexity of this algorithm is the amount of space required to keep track of the total number of unique ways to arrive at the number (size of the ways array = target amount, n) and the total amount of space required to store the combos dictionary. If I was not also printing out each of the unique combinations, the space complexity would simply be O(n) where n is the target amount. Since I am storing all unique combinations, the space complexity of the combos dictionary is O(n c_n* * d) where n is the total amount, c_n is the number of unique combinations for each amount, and d is the number of unique coin denominations. As you can see, including a printout of all the unique combinations is very space consuming compared to just counting the number of possible ways and becomes the dominant factor when considering space constraints.

[ ]:
```python
# Create a helper function to convert a tuple of number of coins of each denomination into a string
def coin_tuple_to_string(coin_tuple, denoms):

    # Initialize a string for the output
    out = ''

    # Loop over all elements of the tuple
    for i, num_coin in enumerate(coin_tuple):

        # Add to string using format (number) x (denom) + ...
        out += f'{num_coin}x{denoms[i]}c + ' if num_coin > 0 else ''

    return out[:-3] # Remove the last ' + ' from the string
```

[ ]:
```python
# Create a helper function to sum two tuples together element-wise
def add_tuples(tuple1, tuple2):
    return tuple(map(sum, zip(tuple1, tuple2)))
```

[ ]:
```python
# Function to print the unique ways to make change for a given value
def numberOfWaysToMakeChange(n, denoms):

    # Initialize a list of number of ways to make up each amount from 0 to n
    ways = [0 for _ in range(n+1)]

    # For zero, can only make change one way (zero coins, base case)
    ways[0] = 1

    # Initialize a dictionary to store the list of coin combination tuples for each amount (each tuple in the list represents a unique combination of coins)
    combos = {0: [tuple(0 for _ in denoms)]}

    # Print the header to a visualization table to explain what's going on
    print('Below is a table showing the number of unique ways to make change for each amount from 0 to n (columns) using increasingly more coins (rows):')
    print('NOTE: this table is best displayed in VS Code Jupyter Nodetook, or you can copy/paste into some text editor that won\'t wrap the text onto multiple lines')
    print()
    header = f'{"Allowed Coins":^24}' + '| ' + ' | '.join([f'{str(i):^3}' for i in range(n+1)])
    print(header)
    print('=' * len(header))

    # Loop over each coin (track the index of the coin for use later when adding coing usage to the combination tuples)
    for i, coin in enumerate(denoms):
```

```
        # Loop over all amounts from 1 to n
        for amount in range(1, n+1):

            # If the coin is less than or equal to the current amount, we can␣
↪use it to make change
            if coin <= amount:

                # Add the number of ways to make change for the remaining␣
↪amount after we use the current coin (value in row above plus value in row x␣
↪elements to the left (if possible) in the visualization table, where x is␣
↪the value of the coin)
                ways[amount] += ways[amount - coin]

                # Add the coin to each combination tuple for the remaining␣
↪amount after we use the current coin and append the new combinations to the␣
↪list for the current amount
                combos[amount] = combos.get(amount, []) + [add_tuples(combo,␣
↪tuple(1 if idx == i else 0 for idx in range(len(denoms)))) for combo in␣
↪combos[amount - coin]]

        # Print the current row of the table
        print('[' + f'{", ".join([str(coin) for coin in denoms[:i+1]]):<21}' +␣
↪'] | ' + ' | '.join([f'{str(i):^3}' for i in ways]))

    # Add some white space between the table and the results
    print()

    # Convert the combo dictionary tuples into nicely formatted strings using␣
↪the helper function above
    combos = {key: [coin_tuple_to_string(combo, denoms) for combo in value] for␣
↪key, value in combos.items()}

    # Return a string with the number of ways to make change for n cents and␣
↪the unique combinations explicitly listed
    return '\n'.join([f'Number of unique coin combinations to make {n} cents␣
↪using coins {denoms}: {ways[n]} (see unique combinations below):\n'] +␣
↪[f'Combination {i+1:<3}: {combo}' for i, combo in enumerate(combos[n])])
```

[ ]:

```
# Initialize the list of denominations for US currency [cents]
denoms = [1, 5, 10, 25, 50, 100]
amount = 100

# Get the results
results = numberOfWaysToMakeChange(amount, denoms)
```

```
# Note to self, add formatter to HTML output <pre> element for no wrap and side
↪scrolling:  style="white-space: pre; overflow-x: scroll;"
```

```
Below is a table showing the number of unique ways to make change for each
amount from 0 to n (columns) using increasingly more coins (rows):
NOTE: this table is best displayed in VS Code Jupyter Nodetook, or you can
copy/paste into some text editor that won't wrap the text onto multiple lines

     Allowed Coins       |  0  |  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  | 59  | 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
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
======================================================================
[1                   ] |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |
1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1
|  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |
1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1
|  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |
1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1
|  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |
1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1
[1, 5                ] |  1  |  1  |  1  |  1  |  1  |  2  |  2  |  2  |  2  |
2  |  3  |  3  |  3  |  3  |  3  |  4  |  4  |  4  |  4  |  4  |  5  |  5  |  5
|  5  |  5  |  6  |  6  |  6  |  6  |  6  |  7  |  7  |  7  |  7  |  7  |  8  |
8  |  8  |  8  |  8  |  9  |  9  |  9  |  9  |  9  | 10  | 10  | 10  | 10  | 10
| 11  | 11  | 11  | 11  | 11  | 12  | 12  | 12  | 12  | 12  | 13  | 13  | 13  |
13  | 13  | 14  | 14  | 14  | 14  | 14  | 15  | 15  | 15  | 15  | 15  | 16  | 16
| 16  | 16  | 16  | 17  | 17  | 17  | 17  | 17  | 18  | 18  | 18  | 18  | 18  |
19  | 19  | 19  | 19  | 19  | 20  | 20  | 20  | 20  | 20  | 21
[1, 5, 10            ] |  1  |  1  |  1  |  1  |  1  |  2  |  2  |  2  |  2  |
2  |  4  |  4  |  4  |  4  |  4  |  6  |  6  |  6  |  6  |  6  |  9  |  9  |  9
|  9  |  9  | 12  | 12  | 12  | 12  | 12  | 16  | 16  | 16  | 16  | 16  | 20  |
20  | 20  | 20  | 20  | 25  | 25  | 25  | 25  | 25  | 30  | 30  | 30  | 30  | 30
| 36  | 36  | 36  | 36  | 36  | 42  | 42  | 42  | 42  | 42  | 49  | 49  | 49  |
49  | 49  | 56  | 56  | 56  | 56  | 56  | 64  | 64  | 64  | 64  | 64  | 72  | 72
| 72  | 72  | 72  | 81  | 81  | 81  | 81  | 81  | 90  | 90  | 90  | 90  | 90  |
100 | 100 | 100 | 100 | 100 | 110 | 110 | 110 | 110 | 110 | 121
```

```
[1, 5, 10, 25         ] |  1  |  1  |  1  |  1  |  1  |  2  |  2  |  2  |  2  |
2  |  4  |  4  |  4  |  4  |  4  |  6  |  6  |  6  |  6  |  6  |  9  |  9  |  9
|  9  |  9  | 13  | 13  | 13  | 13  | 13  | 18  | 18  | 18  | 18  | 18  | 24  |
24  | 24  | 24  | 24  | 31  | 31  | 31  | 31  | 31  | 39  | 39  | 39  | 39  | 39
| 49  | 49  | 49  | 49  | 49  | 60  | 60  | 60  | 60  | 60  | 73  | 73  | 73  |
73  | 73  | 87  | 87  | 87  | 87  | 87  | 103 | 103 | 103 | 103 | 103 | 121 |
121 | 121 | 121 | 121 | 141 | 141 | 141 | 141 | 141 | 163 | 163 | 163 | 163 |
163 | 187 | 187 | 187 | 187 | 187 | 213 | 213 | 213 | 213 | 213 | 242
[1, 5, 10, 25, 50     ] |  1  |  1  |  1  |  1  |  1  |  2  |  2  |  2  |  2  |
2  |  4  |  4  |  4  |  4  |  4  |  6  |  6  |  6  |  6  |  6  |  9  |  9  |  9
|  9  |  9  | 13  | 13  | 13  | 13  | 13  | 18  | 18  | 18  | 18  | 18  | 24  |
24  | 24  | 24  | 24  | 31  | 31  | 31  | 31  | 31  | 39  | 39  | 39  | 39  | 39
| 50  | 50  | 50  | 50  | 50  | 62  | 62  | 62  | 62  | 62  | 77  | 77  | 77  |
77  | 77  | 93  | 93  | 93  | 93  | 93  | 112 | 112 | 112 | 112 | 112 | 134 |
134 | 134 | 134 | 134 | 159 | 159 | 159 | 159 | 159 | 187 | 187 | 187 | 187 |
187 | 218 | 218 | 218 | 218 | 218 | 252 | 252 | 252 | 252 | 252 | 292
[1, 5, 10, 25, 50, 100] |  1  |  1  |  1  |  1  |  1  |  2  |  2  |  2  |  2  |
2  |  4  |  4  |  4  |  4  |  4  |  6  |  6  |  6  |  6  |  6  |  9  |  9  |  9
|  9  |  9  | 13  | 13  | 13  | 13  | 13  | 18  | 18  | 18  | 18  | 18  | 24  |
24  | 24  | 24  | 24  | 31  | 31  | 31  | 31  | 31  | 39  | 39  | 39  | 39  | 39
| 50  | 50  | 50  | 50  | 50  | 62  | 62  | 62  | 62  | 62  | 77  | 77  | 77  |
77  | 77  | 93  | 93  | 93  | 93  | 93  | 112 | 112 | 112 | 112 | 112 | 134 |
134 | 134 | 134 | 134 | 159 | 159 | 159 | 159 | 159 | 187 | 187 | 187 | 187 |
187 | 218 | 218 | 218 | 218 | 218 | 252 | 252 | 252 | 252 | 252 | 293
```

[ ]:
```python
# Print the output seperately (so that can view the above table better in HTML␣
↪document)
print(results)
```

```
Number of unique coin combinations to make 100 cents using coins [1, 5, 10, 25,
50, 100]: 293 (see unique combinations below):

Combination 1  : 100x1c
Combination 2  : 95x1c + 1x5c
Combination 3  : 90x1c + 2x5c
Combination 4  : 85x1c + 3x5c
Combination 5  : 80x1c + 4x5c
Combination 6  : 75x1c + 5x5c
Combination 7  : 70x1c + 6x5c
Combination 8  : 65x1c + 7x5c
Combination 9  : 60x1c + 8x5c
Combination 10 : 55x1c + 9x5c
Combination 11 : 50x1c + 10x5c
Combination 12 : 45x1c + 11x5c
Combination 13 : 40x1c + 12x5c
Combination 14 : 35x1c + 13x5c
Combination 15 : 30x1c + 14x5c
```

```
Combination 16 : 25x1c + 15x5c
Combination 17 : 20x1c + 16x5c
Combination 18 : 15x1c + 17x5c
Combination 19 : 10x1c + 18x5c
Combination 20 : 5x1c + 19x5c
Combination 21 : 20x5c
Combination 22 : 90x1c + 1x10c
Combination 23 : 85x1c + 1x5c + 1x10c
Combination 24 : 80x1c + 2x5c + 1x10c
Combination 25 : 75x1c + 3x5c + 1x10c
Combination 26 : 70x1c + 4x5c + 1x10c
Combination 27 : 65x1c + 5x5c + 1x10c
Combination 28 : 60x1c + 6x5c + 1x10c
Combination 29 : 55x1c + 7x5c + 1x10c
Combination 30 : 50x1c + 8x5c + 1x10c
Combination 31 : 45x1c + 9x5c + 1x10c
Combination 32 : 40x1c + 10x5c + 1x10c
Combination 33 : 35x1c + 11x5c + 1x10c
Combination 34 : 30x1c + 12x5c + 1x10c
Combination 35 : 25x1c + 13x5c + 1x10c
Combination 36 : 20x1c + 14x5c + 1x10c
Combination 37 : 15x1c + 15x5c + 1x10c
Combination 38 : 10x1c + 16x5c + 1x10c
Combination 39 : 5x1c + 17x5c + 1x10c
Combination 40 : 18x5c + 1x10c
Combination 41 : 80x1c + 2x10c
Combination 42 : 75x1c + 1x5c + 2x10c
Combination 43 : 70x1c + 2x5c + 2x10c
Combination 44 : 65x1c + 3x5c + 2x10c
Combination 45 : 60x1c + 4x5c + 2x10c
Combination 46 : 55x1c + 5x5c + 2x10c
Combination 47 : 50x1c + 6x5c + 2x10c
Combination 48 : 45x1c + 7x5c + 2x10c
Combination 49 : 40x1c + 8x5c + 2x10c
Combination 50 : 35x1c + 9x5c + 2x10c
Combination 51 : 30x1c + 10x5c + 2x10c
Combination 52 : 25x1c + 11x5c + 2x10c
Combination 53 : 20x1c + 12x5c + 2x10c
Combination 54 : 15x1c + 13x5c + 2x10c
Combination 55 : 10x1c + 14x5c + 2x10c
Combination 56 : 5x1c + 15x5c + 2x10c
Combination 57 : 16x5c + 2x10c
Combination 58 : 70x1c + 3x10c
Combination 59 : 65x1c + 1x5c + 3x10c
Combination 60 : 60x1c + 2x5c + 3x10c
Combination 61 : 55x1c + 3x5c + 3x10c
Combination 62 : 50x1c + 4x5c + 3x10c
Combination 63 : 45x1c + 5x5c + 3x10c
```

```
Combination 64 : 40x1c + 6x5c + 3x10c
Combination 65 : 35x1c + 7x5c + 3x10c
Combination 66 : 30x1c + 8x5c + 3x10c
Combination 67 : 25x1c + 9x5c + 3x10c
Combination 68 : 20x1c + 10x5c + 3x10c
Combination 69 : 15x1c + 11x5c + 3x10c
Combination 70 : 10x1c + 12x5c + 3x10c
Combination 71 : 5x1c + 13x5c + 3x10c
Combination 72 : 14x5c + 3x10c
Combination 73 : 60x1c + 4x10c
Combination 74 : 55x1c + 1x5c + 4x10c
Combination 75 : 50x1c + 2x5c + 4x10c
Combination 76 : 45x1c + 3x5c + 4x10c
Combination 77 : 40x1c + 4x5c + 4x10c
Combination 78 : 35x1c + 5x5c + 4x10c
Combination 79 : 30x1c + 6x5c + 4x10c
Combination 80 : 25x1c + 7x5c + 4x10c
Combination 81 : 20x1c + 8x5c + 4x10c
Combination 82 : 15x1c + 9x5c + 4x10c
Combination 83 : 10x1c + 10x5c + 4x10c
Combination 84 : 5x1c + 11x5c + 4x10c
Combination 85 : 12x5c + 4x10c
Combination 86 : 50x1c + 5x10c
Combination 87 : 45x1c + 1x5c + 5x10c
Combination 88 : 40x1c + 2x5c + 5x10c
Combination 89 : 35x1c + 3x5c + 5x10c
Combination 90 : 30x1c + 4x5c + 5x10c
Combination 91 : 25x1c + 5x5c + 5x10c
Combination 92 : 20x1c + 6x5c + 5x10c
Combination 93 : 15x1c + 7x5c + 5x10c
Combination 94 : 10x1c + 8x5c + 5x10c
Combination 95 : 5x1c + 9x5c + 5x10c
Combination 96 : 10x5c + 5x10c
Combination 97 : 40x1c + 6x10c
Combination 98 : 35x1c + 1x5c + 6x10c
Combination 99 : 30x1c + 2x5c + 6x10c
Combination 100: 25x1c + 3x5c + 6x10c
Combination 101: 20x1c + 4x5c + 6x10c
Combination 102: 15x1c + 5x5c + 6x10c
Combination 103: 10x1c + 6x5c + 6x10c
Combination 104: 5x1c + 7x5c + 6x10c
Combination 105: 8x5c + 6x10c
Combination 106: 30x1c + 7x10c
Combination 107: 25x1c + 1x5c + 7x10c
Combination 108: 20x1c + 2x5c + 7x10c
Combination 109: 15x1c + 3x5c + 7x10c
Combination 110: 10x1c + 4x5c + 7x10c
Combination 111: 5x1c + 5x5c + 7x10c
```

Combination 112: 6x5c + 7x10c
Combination 113: 20x1c + 8x10c
Combination 114: 15x1c + 1x5c + 8x10c
Combination 115: 10x1c + 2x5c + 8x10c
Combination 116: 5x1c + 3x5c + 8x10c
Combination 117: 4x5c + 8x10c
Combination 118: 10x1c + 9x10c
Combination 119: 5x1c + 1x5c + 9x10c
Combination 120: 2x5c + 9x10c
Combination 121: 10x10c
Combination 122: 75x1c + 1x25c
Combination 123: 70x1c + 1x5c + 1x25c
Combination 124: 65x1c + 2x5c + 1x25c
Combination 125: 60x1c + 3x5c + 1x25c
Combination 126: 55x1c + 4x5c + 1x25c
Combination 127: 50x1c + 5x5c + 1x25c
Combination 128: 45x1c + 6x5c + 1x25c
Combination 129: 40x1c + 7x5c + 1x25c
Combination 130: 35x1c + 8x5c + 1x25c
Combination 131: 30x1c + 9x5c + 1x25c
Combination 132: 25x1c + 10x5c + 1x25c
Combination 133: 20x1c + 11x5c + 1x25c
Combination 134: 15x1c + 12x5c + 1x25c
Combination 135: 10x1c + 13x5c + 1x25c
Combination 136: 5x1c + 14x5c + 1x25c
Combination 137: 15x5c + 1x25c
Combination 138: 65x1c + 1x10c + 1x25c
Combination 139: 60x1c + 1x5c + 1x10c + 1x25c
Combination 140: 55x1c + 2x5c + 1x10c + 1x25c
Combination 141: 50x1c + 3x5c + 1x10c + 1x25c
Combination 142: 45x1c + 4x5c + 1x10c + 1x25c
Combination 143: 40x1c + 5x5c + 1x10c + 1x25c
Combination 144: 35x1c + 6x5c + 1x10c + 1x25c
Combination 145: 30x1c + 7x5c + 1x10c + 1x25c
Combination 146: 25x1c + 8x5c + 1x10c + 1x25c
Combination 147: 20x1c + 9x5c + 1x10c + 1x25c
Combination 148: 15x1c + 10x5c + 1x10c + 1x25c
Combination 149: 10x1c + 11x5c + 1x10c + 1x25c
Combination 150: 5x1c + 12x5c + 1x10c + 1x25c
Combination 151: 13x5c + 1x10c + 1x25c
Combination 152: 55x1c + 2x10c + 1x25c
Combination 153: 50x1c + 1x5c + 2x10c + 1x25c
Combination 154: 45x1c + 2x5c + 2x10c + 1x25c
Combination 155: 40x1c + 3x5c + 2x10c + 1x25c
Combination 156: 35x1c + 4x5c + 2x10c + 1x25c
Combination 157: 30x1c + 5x5c + 2x10c + 1x25c
Combination 158: 25x1c + 6x5c + 2x10c + 1x25c
Combination 159: 20x1c + 7x5c + 2x10c + 1x25c

Combination 160: 15x1c + 8x5c + 2x10c + 1x25c
Combination 161: 10x1c + 9x5c + 2x10c + 1x25c
Combination 162: 5x1c + 10x5c + 2x10c + 1x25c
Combination 163: 11x5c + 2x10c + 1x25c
Combination 164: 45x1c + 3x10c + 1x25c
Combination 165: 40x1c + 1x5c + 3x10c + 1x25c
Combination 166: 35x1c + 2x5c + 3x10c + 1x25c
Combination 167: 30x1c + 3x5c + 3x10c + 1x25c
Combination 168: 25x1c + 4x5c + 3x10c + 1x25c
Combination 169: 20x1c + 5x5c + 3x10c + 1x25c
Combination 170: 15x1c + 6x5c + 3x10c + 1x25c
Combination 171: 10x1c + 7x5c + 3x10c + 1x25c
Combination 172: 5x1c + 8x5c + 3x10c + 1x25c
Combination 173: 9x5c + 3x10c + 1x25c
Combination 174: 35x1c + 4x10c + 1x25c
Combination 175: 30x1c + 1x5c + 4x10c + 1x25c
Combination 176: 25x1c + 2x5c + 4x10c + 1x25c
Combination 177: 20x1c + 3x5c + 4x10c + 1x25c
Combination 178: 15x1c + 4x5c + 4x10c + 1x25c
Combination 179: 10x1c + 5x5c + 4x10c + 1x25c
Combination 180: 5x1c + 6x5c + 4x10c + 1x25c
Combination 181: 7x5c + 4x10c + 1x25c
Combination 182: 25x1c + 5x10c + 1x25c
Combination 183: 20x1c + 1x5c + 5x10c + 1x25c
Combination 184: 15x1c + 2x5c + 5x10c + 1x25c
Combination 185: 10x1c + 3x5c + 5x10c + 1x25c
Combination 186: 5x1c + 4x5c + 5x10c + 1x25c
Combination 187: 5x5c + 5x10c + 1x25c
Combination 188: 15x1c + 6x10c + 1x25c
Combination 189: 10x1c + 1x5c + 6x10c + 1x25c
Combination 190: 5x1c + 2x5c + 6x10c + 1x25c
Combination 191: 3x5c + 6x10c + 1x25c
Combination 192: 5x1c + 7x10c + 1x25c
Combination 193: 1x5c + 7x10c + 1x25c
Combination 194: 50x1c + 2x25c
Combination 195: 45x1c + 1x5c + 2x25c
Combination 196: 40x1c + 2x5c + 2x25c
Combination 197: 35x1c + 3x5c + 2x25c
Combination 198: 30x1c + 4x5c + 2x25c
Combination 199: 25x1c + 5x5c + 2x25c
Combination 200: 20x1c + 6x5c + 2x25c
Combination 201: 15x1c + 7x5c + 2x25c
Combination 202: 10x1c + 8x5c + 2x25c
Combination 203: 5x1c + 9x5c + 2x25c
Combination 204: 10x5c + 2x25c
Combination 205: 40x1c + 1x10c + 2x25c
Combination 206: 35x1c + 1x5c + 1x10c + 2x25c
Combination 207: 30x1c + 2x5c + 1x10c + 2x25c

```
Combination 208: 25x1c + 3x5c + 1x10c + 2x25c
Combination 209: 20x1c + 4x5c + 1x10c + 2x25c
Combination 210: 15x1c + 5x5c + 1x10c + 2x25c
Combination 211: 10x1c + 6x5c + 1x10c + 2x25c
Combination 212: 5x1c + 7x5c + 1x10c + 2x25c
Combination 213: 8x5c + 1x10c + 2x25c
Combination 214: 30x1c + 2x10c + 2x25c
Combination 215: 25x1c + 1x5c + 2x10c + 2x25c
Combination 216: 20x1c + 2x5c + 2x10c + 2x25c
Combination 217: 15x1c + 3x5c + 2x10c + 2x25c
Combination 218: 10x1c + 4x5c + 2x10c + 2x25c
Combination 219: 5x1c + 5x5c + 2x10c + 2x25c
Combination 220: 6x5c + 2x10c + 2x25c
Combination 221: 20x1c + 3x10c + 2x25c
Combination 222: 15x1c + 1x5c + 3x10c + 2x25c
Combination 223: 10x1c + 2x5c + 3x10c + 2x25c
Combination 224: 5x1c + 3x5c + 3x10c + 2x25c
Combination 225: 4x5c + 3x10c + 2x25c
Combination 226: 10x1c + 4x10c + 2x25c
Combination 227: 5x1c + 1x5c + 4x10c + 2x25c
Combination 228: 2x5c + 4x10c + 2x25c
Combination 229: 5x10c + 2x25c
Combination 230: 25x1c + 3x25c
Combination 231: 20x1c + 1x5c + 3x25c
Combination 232: 15x1c + 2x5c + 3x25c
Combination 233: 10x1c + 3x5c + 3x25c
Combination 234: 5x1c + 4x5c + 3x25c
Combination 235: 5x5c + 3x25c
Combination 236: 15x1c + 1x10c + 3x25c
Combination 237: 10x1c + 1x5c + 1x10c + 3x25c
Combination 238: 5x1c + 2x5c + 1x10c + 3x25c
Combination 239: 3x5c + 1x10c + 3x25c
Combination 240: 5x1c + 2x10c + 3x25c
Combination 241: 1x5c + 2x10c + 3x25c
Combination 242: 4x25c
Combination 243: 50x1c + 1x50c
Combination 244: 45x1c + 1x5c + 1x50c
Combination 245: 40x1c + 2x5c + 1x50c
Combination 246: 35x1c + 3x5c + 1x50c
Combination 247: 30x1c + 4x5c + 1x50c
Combination 248: 25x1c + 5x5c + 1x50c
Combination 249: 20x1c + 6x5c + 1x50c
Combination 250: 15x1c + 7x5c + 1x50c
Combination 251: 10x1c + 8x5c + 1x50c
Combination 252: 5x1c + 9x5c + 1x50c
Combination 253: 10x5c + 1x50c
Combination 254: 40x1c + 1x10c + 1x50c
Combination 255: 35x1c + 1x5c + 1x10c + 1x50c
```

```
Combination 256: 30x1c + 2x5c + 1x10c + 1x50c
Combination 257: 25x1c + 3x5c + 1x10c + 1x50c
Combination 258: 20x1c + 4x5c + 1x10c + 1x50c
Combination 259: 15x1c + 5x5c + 1x10c + 1x50c
Combination 260: 10x1c + 6x5c + 1x10c + 1x50c
Combination 261: 5x1c + 7x5c + 1x10c + 1x50c
Combination 262: 8x5c + 1x10c + 1x50c
Combination 263: 30x1c + 2x10c + 1x50c
Combination 264: 25x1c + 1x5c + 2x10c + 1x50c
Combination 265: 20x1c + 2x5c + 2x10c + 1x50c
Combination 266: 15x1c + 3x5c + 2x10c + 1x50c
Combination 267: 10x1c + 4x5c + 2x10c + 1x50c
Combination 268: 5x1c + 5x5c + 2x10c + 1x50c
Combination 269: 6x5c + 2x10c + 1x50c
Combination 270: 20x1c + 3x10c + 1x50c
Combination 271: 15x1c + 1x5c + 3x10c + 1x50c
Combination 272: 10x1c + 2x5c + 3x10c + 1x50c
Combination 273: 5x1c + 3x5c + 3x10c + 1x50c
Combination 274: 4x5c + 3x10c + 1x50c
Combination 275: 10x1c + 4x10c + 1x50c
Combination 276: 5x1c + 1x5c + 4x10c + 1x50c
Combination 277: 2x5c + 4x10c + 1x50c
Combination 278: 5x10c + 1x50c
Combination 279: 25x1c + 1x25c + 1x50c
Combination 280: 20x1c + 1x5c + 1x25c + 1x50c
Combination 281: 15x1c + 2x5c + 1x25c + 1x50c
Combination 282: 10x1c + 3x5c + 1x25c + 1x50c
Combination 283: 5x1c + 4x5c + 1x25c + 1x50c
Combination 284: 5x5c + 1x25c + 1x50c
Combination 285: 15x1c + 1x10c + 1x25c + 1x50c
Combination 286: 10x1c + 1x5c + 1x10c + 1x25c + 1x50c
Combination 287: 5x1c + 2x5c + 1x10c + 1x25c + 1x50c
Combination 288: 3x5c + 1x10c + 1x25c + 1x50c
Combination 289: 5x1c + 2x10c + 1x25c + 1x50c
Combination 290: 1x5c + 2x10c + 1x25c + 1x50c
Combination 291: 2x25c + 1x50c
Combination 292: 2x50c
Combination 293: 1x100c
```

[ ]:
```
# Get the results
results = numberOfWaysToMakeChange(50, denoms)

# Note to self, add formatter to HTML output <pre> element for no wrap and side
↪scrolling:  style="white-space: pre; overflow-x: scroll;"
```

```
Below is a table showing the number of unique ways to make change for each
amount from 0 to n (columns) using increasingly more coins (rows):
NOTE: this table is best displayed in VS Code Jupyter Nodetook, or you can
```

```
copy/paste into some text editor that won't wrap the text onto multiple lines

     Allowed Coins       |  0  |  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
================================================================================
================================================================================
================================================================================
================================================================================
=========
[1                     ] |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |
1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1
|  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |
1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1
|  1
[1, 5                  ] |  1  |  1  |  1  |  1  |  1  |  2  |  2  |  2  |  2  |
2  |  3  |  3  |  3  |  3  |  3  |  4  |  4  |  4  |  4  |  4  |  5  |  5  |  5
|  5  |  5  |  6  |  6  |  6  |  6  |  6  |  7  |  7  |  7  |  7  |  7  |  8  |
8  |  8  |  8  |  8  |  9  |  9  |  9  |  9  |  9  | 10  | 10  | 10  | 10  | 10
| 11
[1, 5, 10              ] |  1  |  1  |  1  |  1  |  1  |  2  |  2  |  2  |  2  |
2  |  4  |  4  |  4  |  4  |  4  |  6  |  6  |  6  |  6  |  6  |  9  |  9  |  9
|  9  |  9  | 12  | 12  | 12  | 12  | 12  | 16  | 16  | 16  | 16  | 16  | 20  |
20  | 20  | 20  | 20  | 25  | 25  | 25  | 25  | 25  | 30  | 30  | 30  | 30  | 30
| 36
[1, 5, 10, 25          ] |  1  |  1  |  1  |  1  |  1  |  2  |  2  |  2  |  2  |
2  |  4  |  4  |  4  |  4  |  4  |  6  |  6  |  6  |  6  |  6  |  9  |  9  |  9
|  9  |  9  | 13  | 13  | 13  | 13  | 13  | 18  | 18  | 18  | 18  | 18  | 24  |
24  | 24  | 24  | 24  | 31  | 31  | 31  | 31  | 31  | 39  | 39  | 39  | 39  | 39
| 49
[1, 5, 10, 25, 50      ] |  1  |  1  |  1  |  1  |  1  |  2  |  2  |  2  |  2  |
2  |  4  |  4  |  4  |  4  |  4  |  6  |  6  |  6  |  6  |  6  |  9  |  9  |  9
|  9  |  9  | 13  | 13  | 13  | 13  | 13  | 18  | 18  | 18  | 18  | 18  | 24  |
24  | 24  | 24  | 24  | 31  | 31  | 31  | 31  | 31  | 39  | 39  | 39  | 39  | 39
| 50
[1, 5, 10, 25, 50, 100] |  1  |  1  |  1  |  1  |  1  |  2  |  2  |  2  |  2  |
2  |  4  |  4  |  4  |  4  |  4  |  6  |  6  |  6  |  6  |  6  |  9  |  9  |  9
|  9  |  9  | 13  | 13  | 13  | 13  | 13  | 18  | 18  | 18  | 18  | 18  | 24  |
24  | 24  | 24  | 24  | 31  | 31  | 31  | 31  | 31  | 39  | 39  | 39  | 39  | 39
| 50
```

[ ]:
```python
# Print the output seperately (so that can view the above table better in HTML document)
print(results)
```

```
Number of unique coin combinations to make 50 cents using coins [1, 5, 10, 25,
50, 100]: 50 (see unique combinations below):

Combination 1  : 50x1c
Combination 2  : 45x1c + 1x5c
Combination 3  : 40x1c + 2x5c
Combination 4  : 35x1c + 3x5c
Combination 5  : 30x1c + 4x5c
Combination 6  : 25x1c + 5x5c
Combination 7  : 20x1c + 6x5c
Combination 8  : 15x1c + 7x5c
Combination 9  : 10x1c + 8x5c
Combination 10 : 5x1c + 9x5c
Combination 11 : 10x5c
Combination 12 : 40x1c + 1x10c
Combination 13 : 35x1c + 1x5c + 1x10c
Combination 14 : 30x1c + 2x5c + 1x10c
Combination 15 : 25x1c + 3x5c + 1x10c
Combination 16 : 20x1c + 4x5c + 1x10c
Combination 17 : 15x1c + 5x5c + 1x10c
Combination 18 : 10x1c + 6x5c + 1x10c
Combination 19 : 5x1c + 7x5c + 1x10c
Combination 20 : 8x5c + 1x10c
Combination 21 : 30x1c + 2x10c
Combination 22 : 25x1c + 1x5c + 2x10c
Combination 23 : 20x1c + 2x5c + 2x10c
Combination 24 : 15x1c + 3x5c + 2x10c
Combination 25 : 10x1c + 4x5c + 2x10c
Combination 26 : 5x1c + 5x5c + 2x10c
Combination 27 : 6x5c + 2x10c
Combination 28 : 20x1c + 3x10c
Combination 29 : 15x1c + 1x5c + 3x10c
Combination 30 : 10x1c + 2x5c + 3x10c
Combination 31 : 5x1c + 3x5c + 3x10c
Combination 32 : 4x5c + 3x10c
Combination 33 : 10x1c + 4x10c
Combination 34 : 5x1c + 1x5c + 4x10c
Combination 35 : 2x5c + 4x10c
Combination 36 : 5x10c
Combination 37 : 25x1c + 1x25c
Combination 38 : 20x1c + 1x5c + 1x25c
Combination 39 : 15x1c + 2x5c + 1x25c
Combination 40 : 10x1c + 3x5c + 1x25c
Combination 41 : 5x1c + 4x5c + 1x25c
Combination 42 : 5x5c + 1x25c
Combination 43 : 15x1c + 1x10c + 1x25c
Combination 44 : 10x1c + 1x5c + 1x10c + 1x25c
Combination 45 : 5x1c + 2x5c + 1x10c + 1x25c
```

```
Combination 46 : 3x5c + 1x10c + 1x25c
Combination 47 : 5x1c + 2x10c + 1x25c
Combination 48 : 1x5c + 2x10c + 1x25c
Combination 49 : 2x25c
Combination 50 : 1x50c
```

[ ]:
```python
# Test using different coins
denoms  = [1, 2, 10, 20, 50, 100]
results = numberOfWaysToMakeChange(100, denoms)

# Note to self, add formatter to HTML output <pre> element for no wrap and side scrolling:  style="white-space: pre; overflow-x: scroll;"
```

```
Below is a table showing the number of unique ways to make change for each
amount from 0 to n (columns) using increasingly more coins (rows):
NOTE: this table is best displayed in VS Code Jupyter Nodetook, or you can
copy/paste into some text editor that won't wrap the text onto multiple lines

     Allowed Coins       |  0  |  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   | 59   | 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
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
=====================================================================
[1                   ] |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |
1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1
|  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |
1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1
|  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |
1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1
|  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |
1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1  |  1
[1, 2                ] |  1  |  1  |  2  |  2  |  3  |  3  |  4  |  4  |  5  |
5  |  6  |  6  |  7  |  7  |  8  |  8  |  9  |  9  | 10  | 10  | 11  | 11  | 12
| 12  | 13  | 13  | 14  | 14  | 15  | 15  | 16  | 16  | 17  | 17  | 18  | 18  |
19  | 19  | 20  | 20  | 21  | 21  | 22  | 22  | 23  | 23  | 24  | 24  | 25  | 25
| 26  | 26  | 27  | 27  | 28  | 28  | 29  | 29  | 30  | 30  | 31  | 31  | 32  |
32  | 33  | 33  | 34  | 34  | 35  | 35  | 36  | 36  | 37  | 37  | 38  | 38  | 39
```

```
| 39   | 40   | 40   | 41   | 41   | 42   | 42   | 43   | 43   | 44   | 44   | 45   | 45   |
46   | 46   | 47   | 47   | 48   | 48   | 49   | 49   | 50   | 50   | 51
[1, 2, 10              ] |   1  |   1  |   2  |   2  |   3  |   3  |   4  |   4  |   5  |
5   |   7  |   7  |   9  |   9  | 11   | 11   | 13   | 13   | 15   | 15   | 18   | 18   | 21
| 21   | 24   | 24   | 27   | 27   | 30   | 30   | 34   | 34   | 38   | 38   | 42   | 42   |
46   | 46   | 50   | 50   | 55   | 55   | 60   | 60   | 65   | 65   | 70   | 70   | 75   | 75
| 81   | 81   | 87   | 87   | 93   | 93   | 99   | 99   | 105  | 105  | 112  | 112  | 119  |
119  | 126  | 126  | 133  | 133  | 140  | 140  | 148  | 148  | 156  | 156  | 164  | 164  |
172  | 172  | 180  | 180  | 189  | 189  | 198  | 198  | 207  | 207  | 216  | 216  | 225  |
225  | 235  | 235  | 245  | 245  | 255  | 255  | 265  | 265  | 275  | 275  | 286
[1, 2, 10, 20          ] |   1  |   1  |   2  |   2  |   3  |   3  |   4  |   4  |   5  |
5   |   7  |   7  |   9  |   9  | 11   | 11   | 13   | 13   | 15   | 15   | 19   | 19   | 23
| 23   | 27   | 27   | 31   | 31   | 35   | 35   | 41   | 41   | 47   | 47   | 53   | 53   |
59   | 59   | 65   | 65   | 74   | 74   | 83   | 83   | 92   | 92   | 101  | 101  | 110  |
110  | 122  | 122  | 134  | 134  | 146  | 146  | 158  | 158  | 170  | 170  | 186  | 186  |
202  | 202  | 218  | 218  | 234  | 234  | 250  | 250  | 270  | 270  | 290  | 290  | 310  |
310  | 330  | 330  | 350  | 350  | 375  | 375  | 400  | 400  | 425  | 425  | 450  | 450  |
475  | 475  | 505  | 505  | 535  | 535  | 565  | 565  | 595  | 595  | 625  | 625  | 661
[1, 2, 10, 20, 50      ] |   1  |   1  |   2  |   2  |   3  |   3  |   4  |   4  |   5  |
5   |   7  |   7  |   9  |   9  | 11   | 11   | 13   | 13   | 15   | 15   | 19   | 19   | 23
| 23   | 27   | 27   | 31   | 31   | 35   | 35   | 41   | 41   | 47   | 47   | 53   | 53   |
59   | 59   | 65   | 65   | 74   | 74   | 83   | 83   | 92   | 92   | 101  | 101  | 110  |
110  | 123  | 123  | 136  | 136  | 149  | 149  | 162  | 162  | 175  | 175  | 193  | 193  |
211  | 211  | 229  | 229  | 247  | 247  | 265  | 265  | 289  | 289  | 313  | 313  | 337  |
337  | 361  | 361  | 385  | 385  | 416  | 416  | 447  | 447  | 478  | 478  | 509  | 509  |
540  | 540  | 579  | 579  | 618  | 618  | 657  | 657  | 696  | 696  | 735  | 735  | 784
[1, 2, 10, 20, 50, 100] |   1  |   1  |   2  |   2  |   3  |   3  |   4  |   4  |   5  |
5   |   7  |   7  |   9  |   9  | 11   | 11   | 13   | 13   | 15   | 15   | 19   | 19   | 23
| 23   | 27   | 27   | 31   | 31   | 35   | 35   | 41   | 41   | 47   | 47   | 53   | 53   |
59   | 59   | 65   | 65   | 74   | 74   | 83   | 83   | 92   | 92   | 101  | 101  | 110  |
110  | 123  | 123  | 136  | 136  | 149  | 149  | 162  | 162  | 175  | 175  | 193  | 193  |
211  | 211  | 229  | 229  | 247  | 247  | 265  | 265  | 289  | 289  | 313  | 313  | 337  |
337  | 361  | 361  | 385  | 385  | 416  | 416  | 447  | 447  | 478  | 478  | 509  | 509  |
540  | 540  | 579  | 579  | 618  | 618  | 657  | 657  | 696  | 696  | 735  | 735  | 785
```

[ ]:
```python
# Print the output seperately (so that can view the above table better in HTML document)
print(results)
```

```
Number of unique coin combinations to make 100 cents using coins [1, 2, 10, 20,
50, 100]: 785 (see unique combinations below):

Combination 1   : 100x1c
Combination 2   : 98x1c + 1x2c
Combination 3   : 96x1c + 2x2c
Combination 4   : 94x1c + 3x2c
Combination 5   : 92x1c + 4x2c
```

```
Combination 6  : 90x1c + 5x2c
Combination 7  : 88x1c + 6x2c
Combination 8  : 86x1c + 7x2c
Combination 9  : 84x1c + 8x2c
Combination 10 : 82x1c + 9x2c
Combination 11 : 80x1c + 10x2c
Combination 12 : 78x1c + 11x2c
Combination 13 : 76x1c + 12x2c
Combination 14 : 74x1c + 13x2c
Combination 15 : 72x1c + 14x2c
Combination 16 : 70x1c + 15x2c
Combination 17 : 68x1c + 16x2c
Combination 18 : 66x1c + 17x2c
Combination 19 : 64x1c + 18x2c
Combination 20 : 62x1c + 19x2c
Combination 21 : 60x1c + 20x2c
Combination 22 : 58x1c + 21x2c
Combination 23 : 56x1c + 22x2c
Combination 24 : 54x1c + 23x2c
Combination 25 : 52x1c + 24x2c
Combination 26 : 50x1c + 25x2c
Combination 27 : 48x1c + 26x2c
Combination 28 : 46x1c + 27x2c
Combination 29 : 44x1c + 28x2c
Combination 30 : 42x1c + 29x2c
Combination 31 : 40x1c + 30x2c
Combination 32 : 38x1c + 31x2c
Combination 33 : 36x1c + 32x2c
Combination 34 : 34x1c + 33x2c
Combination 35 : 32x1c + 34x2c
Combination 36 : 30x1c + 35x2c
Combination 37 : 28x1c + 36x2c
Combination 38 : 26x1c + 37x2c
Combination 39 : 24x1c + 38x2c
Combination 40 : 22x1c + 39x2c
Combination 41 : 20x1c + 40x2c
Combination 42 : 18x1c + 41x2c
Combination 43 : 16x1c + 42x2c
Combination 44 : 14x1c + 43x2c
Combination 45 : 12x1c + 44x2c
Combination 46 : 10x1c + 45x2c
Combination 47 : 8x1c + 46x2c
Combination 48 : 6x1c + 47x2c
Combination 49 : 4x1c + 48x2c
Combination 50 : 2x1c + 49x2c
Combination 51 : 50x2c
Combination 52 : 90x1c + 1x10c
Combination 53 : 88x1c + 1x2c + 1x10c
```

```
Combination 54 : 86x1c + 2x2c + 1x10c
Combination 55 : 84x1c + 3x2c + 1x10c
Combination 56 : 82x1c + 4x2c + 1x10c
Combination 57 : 80x1c + 5x2c + 1x10c
Combination 58 : 78x1c + 6x2c + 1x10c
Combination 59 : 76x1c + 7x2c + 1x10c
Combination 60 : 74x1c + 8x2c + 1x10c
Combination 61 : 72x1c + 9x2c + 1x10c
Combination 62 : 70x1c + 10x2c + 1x10c
Combination 63 : 68x1c + 11x2c + 1x10c
Combination 64 : 66x1c + 12x2c + 1x10c
Combination 65 : 64x1c + 13x2c + 1x10c
Combination 66 : 62x1c + 14x2c + 1x10c
Combination 67 : 60x1c + 15x2c + 1x10c
Combination 68 : 58x1c + 16x2c + 1x10c
Combination 69 : 56x1c + 17x2c + 1x10c
Combination 70 : 54x1c + 18x2c + 1x10c
Combination 71 : 52x1c + 19x2c + 1x10c
Combination 72 : 50x1c + 20x2c + 1x10c
Combination 73 : 48x1c + 21x2c + 1x10c
Combination 74 : 46x1c + 22x2c + 1x10c
Combination 75 : 44x1c + 23x2c + 1x10c
Combination 76 : 42x1c + 24x2c + 1x10c
Combination 77 : 40x1c + 25x2c + 1x10c
Combination 78 : 38x1c + 26x2c + 1x10c
Combination 79 : 36x1c + 27x2c + 1x10c
Combination 80 : 34x1c + 28x2c + 1x10c
Combination 81 : 32x1c + 29x2c + 1x10c
Combination 82 : 30x1c + 30x2c + 1x10c
Combination 83 : 28x1c + 31x2c + 1x10c
Combination 84 : 26x1c + 32x2c + 1x10c
Combination 85 : 24x1c + 33x2c + 1x10c
Combination 86 : 22x1c + 34x2c + 1x10c
Combination 87 : 20x1c + 35x2c + 1x10c
Combination 88 : 18x1c + 36x2c + 1x10c
Combination 89 : 16x1c + 37x2c + 1x10c
Combination 90 : 14x1c + 38x2c + 1x10c
Combination 91 : 12x1c + 39x2c + 1x10c
Combination 92 : 10x1c + 40x2c + 1x10c
Combination 93 : 8x1c + 41x2c + 1x10c
Combination 94 : 6x1c + 42x2c + 1x10c
Combination 95 : 4x1c + 43x2c + 1x10c
Combination 96 : 2x1c + 44x2c + 1x10c
Combination 97 : 45x2c + 1x10c
Combination 98 : 80x1c + 2x10c
Combination 99 : 78x1c + 1x2c + 2x10c
Combination 100: 76x1c + 2x2c + 2x10c
Combination 101: 74x1c + 3x2c + 2x10c
```

```
Combination 102: 72x1c + 4x2c + 2x10c
Combination 103: 70x1c + 5x2c + 2x10c
Combination 104: 68x1c + 6x2c + 2x10c
Combination 105: 66x1c + 7x2c + 2x10c
Combination 106: 64x1c + 8x2c + 2x10c
Combination 107: 62x1c + 9x2c + 2x10c
Combination 108: 60x1c + 10x2c + 2x10c
Combination 109: 58x1c + 11x2c + 2x10c
Combination 110: 56x1c + 12x2c + 2x10c
Combination 111: 54x1c + 13x2c + 2x10c
Combination 112: 52x1c + 14x2c + 2x10c
Combination 113: 50x1c + 15x2c + 2x10c
Combination 114: 48x1c + 16x2c + 2x10c
Combination 115: 46x1c + 17x2c + 2x10c
Combination 116: 44x1c + 18x2c + 2x10c
Combination 117: 42x1c + 19x2c + 2x10c
Combination 118: 40x1c + 20x2c + 2x10c
Combination 119: 38x1c + 21x2c + 2x10c
Combination 120: 36x1c + 22x2c + 2x10c
Combination 121: 34x1c + 23x2c + 2x10c
Combination 122: 32x1c + 24x2c + 2x10c
Combination 123: 30x1c + 25x2c + 2x10c
Combination 124: 28x1c + 26x2c + 2x10c
Combination 125: 26x1c + 27x2c + 2x10c
Combination 126: 24x1c + 28x2c + 2x10c
Combination 127: 22x1c + 29x2c + 2x10c
Combination 128: 20x1c + 30x2c + 2x10c
Combination 129: 18x1c + 31x2c + 2x10c
Combination 130: 16x1c + 32x2c + 2x10c
Combination 131: 14x1c + 33x2c + 2x10c
Combination 132: 12x1c + 34x2c + 2x10c
Combination 133: 10x1c + 35x2c + 2x10c
Combination 134: 8x1c + 36x2c + 2x10c
Combination 135: 6x1c + 37x2c + 2x10c
Combination 136: 4x1c + 38x2c + 2x10c
Combination 137: 2x1c + 39x2c + 2x10c
Combination 138: 40x2c + 2x10c
Combination 139: 70x1c + 3x10c
Combination 140: 68x1c + 1x2c + 3x10c
Combination 141: 66x1c + 2x2c + 3x10c
Combination 142: 64x1c + 3x2c + 3x10c
Combination 143: 62x1c + 4x2c + 3x10c
Combination 144: 60x1c + 5x2c + 3x10c
Combination 145: 58x1c + 6x2c + 3x10c
Combination 146: 56x1c + 7x2c + 3x10c
Combination 147: 54x1c + 8x2c + 3x10c
Combination 148: 52x1c + 9x2c + 3x10c
Combination 149: 50x1c + 10x2c + 3x10c
```

```
Combination 150: 48x1c + 11x2c + 3x10c
Combination 151: 46x1c + 12x2c + 3x10c
Combination 152: 44x1c + 13x2c + 3x10c
Combination 153: 42x1c + 14x2c + 3x10c
Combination 154: 40x1c + 15x2c + 3x10c
Combination 155: 38x1c + 16x2c + 3x10c
Combination 156: 36x1c + 17x2c + 3x10c
Combination 157: 34x1c + 18x2c + 3x10c
Combination 158: 32x1c + 19x2c + 3x10c
Combination 159: 30x1c + 20x2c + 3x10c
Combination 160: 28x1c + 21x2c + 3x10c
Combination 161: 26x1c + 22x2c + 3x10c
Combination 162: 24x1c + 23x2c + 3x10c
Combination 163: 22x1c + 24x2c + 3x10c
Combination 164: 20x1c + 25x2c + 3x10c
Combination 165: 18x1c + 26x2c + 3x10c
Combination 166: 16x1c + 27x2c + 3x10c
Combination 167: 14x1c + 28x2c + 3x10c
Combination 168: 12x1c + 29x2c + 3x10c
Combination 169: 10x1c + 30x2c + 3x10c
Combination 170: 8x1c + 31x2c + 3x10c
Combination 171: 6x1c + 32x2c + 3x10c
Combination 172: 4x1c + 33x2c + 3x10c
Combination 173: 2x1c + 34x2c + 3x10c
Combination 174: 35x2c + 3x10c
Combination 175: 60x1c + 4x10c
Combination 176: 58x1c + 1x2c + 4x10c
Combination 177: 56x1c + 2x2c + 4x10c
Combination 178: 54x1c + 3x2c + 4x10c
Combination 179: 52x1c + 4x2c + 4x10c
Combination 180: 50x1c + 5x2c + 4x10c
Combination 181: 48x1c + 6x2c + 4x10c
Combination 182: 46x1c + 7x2c + 4x10c
Combination 183: 44x1c + 8x2c + 4x10c
Combination 184: 42x1c + 9x2c + 4x10c
Combination 185: 40x1c + 10x2c + 4x10c
Combination 186: 38x1c + 11x2c + 4x10c
Combination 187: 36x1c + 12x2c + 4x10c
Combination 188: 34x1c + 13x2c + 4x10c
Combination 189: 32x1c + 14x2c + 4x10c
Combination 190: 30x1c + 15x2c + 4x10c
Combination 191: 28x1c + 16x2c + 4x10c
Combination 192: 26x1c + 17x2c + 4x10c
Combination 193: 24x1c + 18x2c + 4x10c
Combination 194: 22x1c + 19x2c + 4x10c
Combination 195: 20x1c + 20x2c + 4x10c
Combination 196: 18x1c + 21x2c + 4x10c
Combination 197: 16x1c + 22x2c + 4x10c
```

Combination 198: 14x1c + 23x2c + 4x10c
Combination 199: 12x1c + 24x2c + 4x10c
Combination 200: 10x1c + 25x2c + 4x10c
Combination 201: 8x1c + 26x2c + 4x10c
Combination 202: 6x1c + 27x2c + 4x10c
Combination 203: 4x1c + 28x2c + 4x10c
Combination 204: 2x1c + 29x2c + 4x10c
Combination 205: 30x2c + 4x10c
Combination 206: 50x1c + 5x10c
Combination 207: 48x1c + 1x2c + 5x10c
Combination 208: 46x1c + 2x2c + 5x10c
Combination 209: 44x1c + 3x2c + 5x10c
Combination 210: 42x1c + 4x2c + 5x10c
Combination 211: 40x1c + 5x2c + 5x10c
Combination 212: 38x1c + 6x2c + 5x10c
Combination 213: 36x1c + 7x2c + 5x10c
Combination 214: 34x1c + 8x2c + 5x10c
Combination 215: 32x1c + 9x2c + 5x10c
Combination 216: 30x1c + 10x2c + 5x10c
Combination 217: 28x1c + 11x2c + 5x10c
Combination 218: 26x1c + 12x2c + 5x10c
Combination 219: 24x1c + 13x2c + 5x10c
Combination 220: 22x1c + 14x2c + 5x10c
Combination 221: 20x1c + 15x2c + 5x10c
Combination 222: 18x1c + 16x2c + 5x10c
Combination 223: 16x1c + 17x2c + 5x10c
Combination 224: 14x1c + 18x2c + 5x10c
Combination 225: 12x1c + 19x2c + 5x10c
Combination 226: 10x1c + 20x2c + 5x10c
Combination 227: 8x1c + 21x2c + 5x10c
Combination 228: 6x1c + 22x2c + 5x10c
Combination 229: 4x1c + 23x2c + 5x10c
Combination 230: 2x1c + 24x2c + 5x10c
Combination 231: 25x2c + 5x10c
Combination 232: 40x1c + 6x10c
Combination 233: 38x1c + 1x2c + 6x10c
Combination 234: 36x1c + 2x2c + 6x10c
Combination 235: 34x1c + 3x2c + 6x10c
Combination 236: 32x1c + 4x2c + 6x10c
Combination 237: 30x1c + 5x2c + 6x10c
Combination 238: 28x1c + 6x2c + 6x10c
Combination 239: 26x1c + 7x2c + 6x10c
Combination 240: 24x1c + 8x2c + 6x10c
Combination 241: 22x1c + 9x2c + 6x10c
Combination 242: 20x1c + 10x2c + 6x10c
Combination 243: 18x1c + 11x2c + 6x10c
Combination 244: 16x1c + 12x2c + 6x10c
Combination 245: 14x1c + 13x2c + 6x10c

Combination 246: 12x1c + 14x2c + 6x10c
Combination 247: 10x1c + 15x2c + 6x10c
Combination 248: 8x1c + 16x2c + 6x10c
Combination 249: 6x1c + 17x2c + 6x10c
Combination 250: 4x1c + 18x2c + 6x10c
Combination 251: 2x1c + 19x2c + 6x10c
Combination 252: 20x2c + 6x10c
Combination 253: 30x1c + 7x10c
Combination 254: 28x1c + 1x2c + 7x10c
Combination 255: 26x1c + 2x2c + 7x10c
Combination 256: 24x1c + 3x2c + 7x10c
Combination 257: 22x1c + 4x2c + 7x10c
Combination 258: 20x1c + 5x2c + 7x10c
Combination 259: 18x1c + 6x2c + 7x10c
Combination 260: 16x1c + 7x2c + 7x10c
Combination 261: 14x1c + 8x2c + 7x10c
Combination 262: 12x1c + 9x2c + 7x10c
Combination 263: 10x1c + 10x2c + 7x10c
Combination 264: 8x1c + 11x2c + 7x10c
Combination 265: 6x1c + 12x2c + 7x10c
Combination 266: 4x1c + 13x2c + 7x10c
Combination 267: 2x1c + 14x2c + 7x10c
Combination 268: 15x2c + 7x10c
Combination 269: 20x1c + 8x10c
Combination 270: 18x1c + 1x2c + 8x10c
Combination 271: 16x1c + 2x2c + 8x10c
Combination 272: 14x1c + 3x2c + 8x10c
Combination 273: 12x1c + 4x2c + 8x10c
Combination 274: 10x1c + 5x2c + 8x10c
Combination 275: 8x1c + 6x2c + 8x10c
Combination 276: 6x1c + 7x2c + 8x10c
Combination 277: 4x1c + 8x2c + 8x10c
Combination 278: 2x1c + 9x2c + 8x10c
Combination 279: 10x2c + 8x10c
Combination 280: 10x1c + 9x10c
Combination 281: 8x1c + 1x2c + 9x10c
Combination 282: 6x1c + 2x2c + 9x10c
Combination 283: 4x1c + 3x2c + 9x10c
Combination 284: 2x1c + 4x2c + 9x10c
Combination 285: 5x2c + 9x10c
Combination 286: 10x10c
Combination 287: 80x1c + 1x20c
Combination 288: 78x1c + 1x2c + 1x20c
Combination 289: 76x1c + 2x2c + 1x20c
Combination 290: 74x1c + 3x2c + 1x20c
Combination 291: 72x1c + 4x2c + 1x20c
Combination 292: 70x1c + 5x2c + 1x20c
Combination 293: 68x1c + 6x2c + 1x20c

Combination 294: 66x1c + 7x2c + 1x20c
Combination 295: 64x1c + 8x2c + 1x20c
Combination 296: 62x1c + 9x2c + 1x20c
Combination 297: 60x1c + 10x2c + 1x20c
Combination 298: 58x1c + 11x2c + 1x20c
Combination 299: 56x1c + 12x2c + 1x20c
Combination 300: 54x1c + 13x2c + 1x20c
Combination 301: 52x1c + 14x2c + 1x20c
Combination 302: 50x1c + 15x2c + 1x20c
Combination 303: 48x1c + 16x2c + 1x20c
Combination 304: 46x1c + 17x2c + 1x20c
Combination 305: 44x1c + 18x2c + 1x20c
Combination 306: 42x1c + 19x2c + 1x20c
Combination 307: 40x1c + 20x2c + 1x20c
Combination 308: 38x1c + 21x2c + 1x20c
Combination 309: 36x1c + 22x2c + 1x20c
Combination 310: 34x1c + 23x2c + 1x20c
Combination 311: 32x1c + 24x2c + 1x20c
Combination 312: 30x1c + 25x2c + 1x20c
Combination 313: 28x1c + 26x2c + 1x20c
Combination 314: 26x1c + 27x2c + 1x20c
Combination 315: 24x1c + 28x2c + 1x20c
Combination 316: 22x1c + 29x2c + 1x20c
Combination 317: 20x1c + 30x2c + 1x20c
Combination 318: 18x1c + 31x2c + 1x20c
Combination 319: 16x1c + 32x2c + 1x20c
Combination 320: 14x1c + 33x2c + 1x20c
Combination 321: 12x1c + 34x2c + 1x20c
Combination 322: 10x1c + 35x2c + 1x20c
Combination 323: 8x1c + 36x2c + 1x20c
Combination 324: 6x1c + 37x2c + 1x20c
Combination 325: 4x1c + 38x2c + 1x20c
Combination 326: 2x1c + 39x2c + 1x20c
Combination 327: 40x2c + 1x20c
Combination 328: 70x1c + 1x10c + 1x20c
Combination 329: 68x1c + 1x2c + 1x10c + 1x20c
Combination 330: 66x1c + 2x2c + 1x10c + 1x20c
Combination 331: 64x1c + 3x2c + 1x10c + 1x20c
Combination 332: 62x1c + 4x2c + 1x10c + 1x20c
Combination 333: 60x1c + 5x2c + 1x10c + 1x20c
Combination 334: 58x1c + 6x2c + 1x10c + 1x20c
Combination 335: 56x1c + 7x2c + 1x10c + 1x20c
Combination 336: 54x1c + 8x2c + 1x10c + 1x20c
Combination 337: 52x1c + 9x2c + 1x10c + 1x20c
Combination 338: 50x1c + 10x2c + 1x10c + 1x20c
Combination 339: 48x1c + 11x2c + 1x10c + 1x20c
Combination 340: 46x1c + 12x2c + 1x10c + 1x20c
Combination 341: 44x1c + 13x2c + 1x10c + 1x20c

```
Combination 342: 42x1c + 14x2c + 1x10c + 1x20c
Combination 343: 40x1c + 15x2c + 1x10c + 1x20c
Combination 344: 38x1c + 16x2c + 1x10c + 1x20c
Combination 345: 36x1c + 17x2c + 1x10c + 1x20c
Combination 346: 34x1c + 18x2c + 1x10c + 1x20c
Combination 347: 32x1c + 19x2c + 1x10c + 1x20c
Combination 348: 30x1c + 20x2c + 1x10c + 1x20c
Combination 349: 28x1c + 21x2c + 1x10c + 1x20c
Combination 350: 26x1c + 22x2c + 1x10c + 1x20c
Combination 351: 24x1c + 23x2c + 1x10c + 1x20c
Combination 352: 22x1c + 24x2c + 1x10c + 1x20c
Combination 353: 20x1c + 25x2c + 1x10c + 1x20c
Combination 354: 18x1c + 26x2c + 1x10c + 1x20c
Combination 355: 16x1c + 27x2c + 1x10c + 1x20c
Combination 356: 14x1c + 28x2c + 1x10c + 1x20c
Combination 357: 12x1c + 29x2c + 1x10c + 1x20c
Combination 358: 10x1c + 30x2c + 1x10c + 1x20c
Combination 359: 8x1c + 31x2c + 1x10c + 1x20c
Combination 360: 6x1c + 32x2c + 1x10c + 1x20c
Combination 361: 4x1c + 33x2c + 1x10c + 1x20c
Combination 362: 2x1c + 34x2c + 1x10c + 1x20c
Combination 363: 35x2c + 1x10c + 1x20c
Combination 364: 60x1c + 2x10c + 1x20c
Combination 365: 58x1c + 1x2c + 2x10c + 1x20c
Combination 366: 56x1c + 2x2c + 2x10c + 1x20c
Combination 367: 54x1c + 3x2c + 2x10c + 1x20c
Combination 368: 52x1c + 4x2c + 2x10c + 1x20c
Combination 369: 50x1c + 5x2c + 2x10c + 1x20c
Combination 370: 48x1c + 6x2c + 2x10c + 1x20c
Combination 371: 46x1c + 7x2c + 2x10c + 1x20c
Combination 372: 44x1c + 8x2c + 2x10c + 1x20c
Combination 373: 42x1c + 9x2c + 2x10c + 1x20c
Combination 374: 40x1c + 10x2c + 2x10c + 1x20c
Combination 375: 38x1c + 11x2c + 2x10c + 1x20c
Combination 376: 36x1c + 12x2c + 2x10c + 1x20c
Combination 377: 34x1c + 13x2c + 2x10c + 1x20c
Combination 378: 32x1c + 14x2c + 2x10c + 1x20c
Combination 379: 30x1c + 15x2c + 2x10c + 1x20c
Combination 380: 28x1c + 16x2c + 2x10c + 1x20c
Combination 381: 26x1c + 17x2c + 2x10c + 1x20c
Combination 382: 24x1c + 18x2c + 2x10c + 1x20c
Combination 383: 22x1c + 19x2c + 2x10c + 1x20c
Combination 384: 20x1c + 20x2c + 2x10c + 1x20c
Combination 385: 18x1c + 21x2c + 2x10c + 1x20c
Combination 386: 16x1c + 22x2c + 2x10c + 1x20c
Combination 387: 14x1c + 23x2c + 2x10c + 1x20c
Combination 388: 12x1c + 24x2c + 2x10c + 1x20c
Combination 389: 10x1c + 25x2c + 2x10c + 1x20c
```

```
Combination 390: 8x1c + 26x2c + 2x10c + 1x20c
Combination 391: 6x1c + 27x2c + 2x10c + 1x20c
Combination 392: 4x1c + 28x2c + 2x10c + 1x20c
Combination 393: 2x1c + 29x2c + 2x10c + 1x20c
Combination 394: 30x2c + 2x10c + 1x20c
Combination 395: 50x1c + 3x10c + 1x20c
Combination 396: 48x1c + 1x2c + 3x10c + 1x20c
Combination 397: 46x1c + 2x2c + 3x10c + 1x20c
Combination 398: 44x1c + 3x2c + 3x10c + 1x20c
Combination 399: 42x1c + 4x2c + 3x10c + 1x20c
Combination 400: 40x1c + 5x2c + 3x10c + 1x20c
Combination 401: 38x1c + 6x2c + 3x10c + 1x20c
Combination 402: 36x1c + 7x2c + 3x10c + 1x20c
Combination 403: 34x1c + 8x2c + 3x10c + 1x20c
Combination 404: 32x1c + 9x2c + 3x10c + 1x20c
Combination 405: 30x1c + 10x2c + 3x10c + 1x20c
Combination 406: 28x1c + 11x2c + 3x10c + 1x20c
Combination 407: 26x1c + 12x2c + 3x10c + 1x20c
Combination 408: 24x1c + 13x2c + 3x10c + 1x20c
Combination 409: 22x1c + 14x2c + 3x10c + 1x20c
Combination 410: 20x1c + 15x2c + 3x10c + 1x20c
Combination 411: 18x1c + 16x2c + 3x10c + 1x20c
Combination 412: 16x1c + 17x2c + 3x10c + 1x20c
Combination 413: 14x1c + 18x2c + 3x10c + 1x20c
Combination 414: 12x1c + 19x2c + 3x10c + 1x20c
Combination 415: 10x1c + 20x2c + 3x10c + 1x20c
Combination 416: 8x1c + 21x2c + 3x10c + 1x20c
Combination 417: 6x1c + 22x2c + 3x10c + 1x20c
Combination 418: 4x1c + 23x2c + 3x10c + 1x20c
Combination 419: 2x1c + 24x2c + 3x10c + 1x20c
Combination 420: 25x2c + 3x10c + 1x20c
Combination 421: 40x1c + 4x10c + 1x20c
Combination 422: 38x1c + 1x2c + 4x10c + 1x20c
Combination 423: 36x1c + 2x2c + 4x10c + 1x20c
Combination 424: 34x1c + 3x2c + 4x10c + 1x20c
Combination 425: 32x1c + 4x2c + 4x10c + 1x20c
Combination 426: 30x1c + 5x2c + 4x10c + 1x20c
Combination 427: 28x1c + 6x2c + 4x10c + 1x20c
Combination 428: 26x1c + 7x2c + 4x10c + 1x20c
Combination 429: 24x1c + 8x2c + 4x10c + 1x20c
Combination 430: 22x1c + 9x2c + 4x10c + 1x20c
Combination 431: 20x1c + 10x2c + 4x10c + 1x20c
Combination 432: 18x1c + 11x2c + 4x10c + 1x20c
Combination 433: 16x1c + 12x2c + 4x10c + 1x20c
Combination 434: 14x1c + 13x2c + 4x10c + 1x20c
Combination 435: 12x1c + 14x2c + 4x10c + 1x20c
Combination 436: 10x1c + 15x2c + 4x10c + 1x20c
Combination 437: 8x1c + 16x2c + 4x10c + 1x20c
```

Combination 438: 6x1c + 17x2c + 4x10c + 1x20c
Combination 439: 4x1c + 18x2c + 4x10c + 1x20c
Combination 440: 2x1c + 19x2c + 4x10c + 1x20c
Combination 441: 20x2c + 4x10c + 1x20c
Combination 442: 30x1c + 5x10c + 1x20c
Combination 443: 28x1c + 1x2c + 5x10c + 1x20c
Combination 444: 26x1c + 2x2c + 5x10c + 1x20c
Combination 445: 24x1c + 3x2c + 5x10c + 1x20c
Combination 446: 22x1c + 4x2c + 5x10c + 1x20c
Combination 447: 20x1c + 5x2c + 5x10c + 1x20c
Combination 448: 18x1c + 6x2c + 5x10c + 1x20c
Combination 449: 16x1c + 7x2c + 5x10c + 1x20c
Combination 450: 14x1c + 8x2c + 5x10c + 1x20c
Combination 451: 12x1c + 9x2c + 5x10c + 1x20c
Combination 452: 10x1c + 10x2c + 5x10c + 1x20c
Combination 453: 8x1c + 11x2c + 5x10c + 1x20c
Combination 454: 6x1c + 12x2c + 5x10c + 1x20c
Combination 455: 4x1c + 13x2c + 5x10c + 1x20c
Combination 456: 2x1c + 14x2c + 5x10c + 1x20c
Combination 457: 15x2c + 5x10c + 1x20c
Combination 458: 20x1c + 6x10c + 1x20c
Combination 459: 18x1c + 1x2c + 6x10c + 1x20c
Combination 460: 16x1c + 2x2c + 6x10c + 1x20c
Combination 461: 14x1c + 3x2c + 6x10c + 1x20c
Combination 462: 12x1c + 4x2c + 6x10c + 1x20c
Combination 463: 10x1c + 5x2c + 6x10c + 1x20c
Combination 464: 8x1c + 6x2c + 6x10c + 1x20c
Combination 465: 6x1c + 7x2c + 6x10c + 1x20c
Combination 466: 4x1c + 8x2c + 6x10c + 1x20c
Combination 467: 2x1c + 9x2c + 6x10c + 1x20c
Combination 468: 10x2c + 6x10c + 1x20c
Combination 469: 10x1c + 7x10c + 1x20c
Combination 470: 8x1c + 1x2c + 7x10c + 1x20c
Combination 471: 6x1c + 2x2c + 7x10c + 1x20c
Combination 472: 4x1c + 3x2c + 7x10c + 1x20c
Combination 473: 2x1c + 4x2c + 7x10c + 1x20c
Combination 474: 5x2c + 7x10c + 1x20c
Combination 475: 8x10c + 1x20c
Combination 476: 60x1c + 2x20c
Combination 477: 58x1c + 1x2c + 2x20c
Combination 478: 56x1c + 2x2c + 2x20c
Combination 479: 54x1c + 3x2c + 2x20c
Combination 480: 52x1c + 4x2c + 2x20c
Combination 481: 50x1c + 5x2c + 2x20c
Combination 482: 48x1c + 6x2c + 2x20c
Combination 483: 46x1c + 7x2c + 2x20c
Combination 484: 44x1c + 8x2c + 2x20c
Combination 485: 42x1c + 9x2c + 2x20c

Combination 486: 40x1c + 10x2c + 2x20c
Combination 487: 38x1c + 11x2c + 2x20c
Combination 488: 36x1c + 12x2c + 2x20c
Combination 489: 34x1c + 13x2c + 2x20c
Combination 490: 32x1c + 14x2c + 2x20c
Combination 491: 30x1c + 15x2c + 2x20c
Combination 492: 28x1c + 16x2c + 2x20c
Combination 493: 26x1c + 17x2c + 2x20c
Combination 494: 24x1c + 18x2c + 2x20c
Combination 495: 22x1c + 19x2c + 2x20c
Combination 496: 20x1c + 20x2c + 2x20c
Combination 497: 18x1c + 21x2c + 2x20c
Combination 498: 16x1c + 22x2c + 2x20c
Combination 499: 14x1c + 23x2c + 2x20c
Combination 500: 12x1c + 24x2c + 2x20c
Combination 501: 10x1c + 25x2c + 2x20c
Combination 502: 8x1c + 26x2c + 2x20c
Combination 503: 6x1c + 27x2c + 2x20c
Combination 504: 4x1c + 28x2c + 2x20c
Combination 505: 2x1c + 29x2c + 2x20c
Combination 506: 30x2c + 2x20c
Combination 507: 50x1c + 1x10c + 2x20c
Combination 508: 48x1c + 1x2c + 1x10c + 2x20c
Combination 509: 46x1c + 2x2c + 1x10c + 2x20c
Combination 510: 44x1c + 3x2c + 1x10c + 2x20c
Combination 511: 42x1c + 4x2c + 1x10c + 2x20c
Combination 512: 40x1c + 5x2c + 1x10c + 2x20c
Combination 513: 38x1c + 6x2c + 1x10c + 2x20c
Combination 514: 36x1c + 7x2c + 1x10c + 2x20c
Combination 515: 34x1c + 8x2c + 1x10c + 2x20c
Combination 516: 32x1c + 9x2c + 1x10c + 2x20c
Combination 517: 30x1c + 10x2c + 1x10c + 2x20c
Combination 518: 28x1c + 11x2c + 1x10c + 2x20c
Combination 519: 26x1c + 12x2c + 1x10c + 2x20c
Combination 520: 24x1c + 13x2c + 1x10c + 2x20c
Combination 521: 22x1c + 14x2c + 1x10c + 2x20c
Combination 522: 20x1c + 15x2c + 1x10c + 2x20c
Combination 523: 18x1c + 16x2c + 1x10c + 2x20c
Combination 524: 16x1c + 17x2c + 1x10c + 2x20c
Combination 525: 14x1c + 18x2c + 1x10c + 2x20c
Combination 526: 12x1c + 19x2c + 1x10c + 2x20c
Combination 527: 10x1c + 20x2c + 1x10c + 2x20c
Combination 528: 8x1c + 21x2c + 1x10c + 2x20c
Combination 529: 6x1c + 22x2c + 1x10c + 2x20c
Combination 530: 4x1c + 23x2c + 1x10c + 2x20c
Combination 531: 2x1c + 24x2c + 1x10c + 2x20c
Combination 532: 25x2c + 1x10c + 2x20c
Combination 533: 40x1c + 2x10c + 2x20c

```
Combination 534: 38x1c + 1x2c + 2x10c + 2x20c
Combination 535: 36x1c + 2x2c + 2x10c + 2x20c
Combination 536: 34x1c + 3x2c + 2x10c + 2x20c
Combination 537: 32x1c + 4x2c + 2x10c + 2x20c
Combination 538: 30x1c + 5x2c + 2x10c + 2x20c
Combination 539: 28x1c + 6x2c + 2x10c + 2x20c
Combination 540: 26x1c + 7x2c + 2x10c + 2x20c
Combination 541: 24x1c + 8x2c + 2x10c + 2x20c
Combination 542: 22x1c + 9x2c + 2x10c + 2x20c
Combination 543: 20x1c + 10x2c + 2x10c + 2x20c
Combination 544: 18x1c + 11x2c + 2x10c + 2x20c
Combination 545: 16x1c + 12x2c + 2x10c + 2x20c
Combination 546: 14x1c + 13x2c + 2x10c + 2x20c
Combination 547: 12x1c + 14x2c + 2x10c + 2x20c
Combination 548: 10x1c + 15x2c + 2x10c + 2x20c
Combination 549: 8x1c + 16x2c + 2x10c + 2x20c
Combination 550: 6x1c + 17x2c + 2x10c + 2x20c
Combination 551: 4x1c + 18x2c + 2x10c + 2x20c
Combination 552: 2x1c + 19x2c + 2x10c + 2x20c
Combination 553: 20x2c + 2x10c + 2x20c
Combination 554: 30x1c + 3x10c + 2x20c
Combination 555: 28x1c + 1x2c + 3x10c + 2x20c
Combination 556: 26x1c + 2x2c + 3x10c + 2x20c
Combination 557: 24x1c + 3x2c + 3x10c + 2x20c
Combination 558: 22x1c + 4x2c + 3x10c + 2x20c
Combination 559: 20x1c + 5x2c + 3x10c + 2x20c
Combination 560: 18x1c + 6x2c + 3x10c + 2x20c
Combination 561: 16x1c + 7x2c + 3x10c + 2x20c
Combination 562: 14x1c + 8x2c + 3x10c + 2x20c
Combination 563: 12x1c + 9x2c + 3x10c + 2x20c
Combination 564: 10x1c + 10x2c + 3x10c + 2x20c
Combination 565: 8x1c + 11x2c + 3x10c + 2x20c
Combination 566: 6x1c + 12x2c + 3x10c + 2x20c
Combination 567: 4x1c + 13x2c + 3x10c + 2x20c
Combination 568: 2x1c + 14x2c + 3x10c + 2x20c
Combination 569: 15x2c + 3x10c + 2x20c
Combination 570: 20x1c + 4x10c + 2x20c
Combination 571: 18x1c + 1x2c + 4x10c + 2x20c
Combination 572: 16x1c + 2x2c + 4x10c + 2x20c
Combination 573: 14x1c + 3x2c + 4x10c + 2x20c
Combination 574: 12x1c + 4x2c + 4x10c + 2x20c
Combination 575: 10x1c + 5x2c + 4x10c + 2x20c
Combination 576: 8x1c + 6x2c + 4x10c + 2x20c
Combination 577: 6x1c + 7x2c + 4x10c + 2x20c
Combination 578: 4x1c + 8x2c + 4x10c + 2x20c
Combination 579: 2x1c + 9x2c + 4x10c + 2x20c
Combination 580: 10x2c + 4x10c + 2x20c
Combination 581: 10x1c + 5x10c + 2x20c
```

Combination 582: 8x1c + 1x2c + 5x10c + 2x20c
Combination 583: 6x1c + 2x2c + 5x10c + 2x20c
Combination 584: 4x1c + 3x2c + 5x10c + 2x20c
Combination 585: 2x1c + 4x2c + 5x10c + 2x20c
Combination 586: 5x2c + 5x10c + 2x20c
Combination 587: 6x10c + 2x20c
Combination 588: 40x1c + 3x20c
Combination 589: 38x1c + 1x2c + 3x20c
Combination 590: 36x1c + 2x2c + 3x20c
Combination 591: 34x1c + 3x2c + 3x20c
Combination 592: 32x1c + 4x2c + 3x20c
Combination 593: 30x1c + 5x2c + 3x20c
Combination 594: 28x1c + 6x2c + 3x20c
Combination 595: 26x1c + 7x2c + 3x20c
Combination 596: 24x1c + 8x2c + 3x20c
Combination 597: 22x1c + 9x2c + 3x20c
Combination 598: 20x1c + 10x2c + 3x20c
Combination 599: 18x1c + 11x2c + 3x20c
Combination 600: 16x1c + 12x2c + 3x20c
Combination 601: 14x1c + 13x2c + 3x20c
Combination 602: 12x1c + 14x2c + 3x20c
Combination 603: 10x1c + 15x2c + 3x20c
Combination 604: 8x1c + 16x2c + 3x20c
Combination 605: 6x1c + 17x2c + 3x20c
Combination 606: 4x1c + 18x2c + 3x20c
Combination 607: 2x1c + 19x2c + 3x20c
Combination 608: 20x2c + 3x20c
Combination 609: 30x1c + 1x10c + 3x20c
Combination 610: 28x1c + 1x2c + 1x10c + 3x20c
Combination 611: 26x1c + 2x2c + 1x10c + 3x20c
Combination 612: 24x1c + 3x2c + 1x10c + 3x20c
Combination 613: 22x1c + 4x2c + 1x10c + 3x20c
Combination 614: 20x1c + 5x2c + 1x10c + 3x20c
Combination 615: 18x1c + 6x2c + 1x10c + 3x20c
Combination 616: 16x1c + 7x2c + 1x10c + 3x20c
Combination 617: 14x1c + 8x2c + 1x10c + 3x20c
Combination 618: 12x1c + 9x2c + 1x10c + 3x20c
Combination 619: 10x1c + 10x2c + 1x10c + 3x20c
Combination 620: 8x1c + 11x2c + 1x10c + 3x20c
Combination 621: 6x1c + 12x2c + 1x10c + 3x20c
Combination 622: 4x1c + 13x2c + 1x10c + 3x20c
Combination 623: 2x1c + 14x2c + 1x10c + 3x20c
Combination 624: 15x2c + 1x10c + 3x20c
Combination 625: 20x1c + 2x10c + 3x20c
Combination 626: 18x1c + 1x2c + 2x10c + 3x20c
Combination 627: 16x1c + 2x2c + 2x10c + 3x20c
Combination 628: 14x1c + 3x2c + 2x10c + 3x20c
Combination 629: 12x1c + 4x2c + 2x10c + 3x20c

```
Combination 630: 10x1c + 5x2c + 2x10c + 3x20c
Combination 631: 8x1c + 6x2c + 2x10c + 3x20c
Combination 632: 6x1c + 7x2c + 2x10c + 3x20c
Combination 633: 4x1c + 8x2c + 2x10c + 3x20c
Combination 634: 2x1c + 9x2c + 2x10c + 3x20c
Combination 635: 10x2c + 2x10c + 3x20c
Combination 636: 10x1c + 3x10c + 3x20c
Combination 637: 8x1c + 1x2c + 3x10c + 3x20c
Combination 638: 6x1c + 2x2c + 3x10c + 3x20c
Combination 639: 4x1c + 3x2c + 3x10c + 3x20c
Combination 640: 2x1c + 4x2c + 3x10c + 3x20c
Combination 641: 5x2c + 3x10c + 3x20c
Combination 642: 4x10c + 3x20c
Combination 643: 20x1c + 4x20c
Combination 644: 18x1c + 1x2c + 4x20c
Combination 645: 16x1c + 2x2c + 4x20c
Combination 646: 14x1c + 3x2c + 4x20c
Combination 647: 12x1c + 4x2c + 4x20c
Combination 648: 10x1c + 5x2c + 4x20c
Combination 649: 8x1c + 6x2c + 4x20c
Combination 650: 6x1c + 7x2c + 4x20c
Combination 651: 4x1c + 8x2c + 4x20c
Combination 652: 2x1c + 9x2c + 4x20c
Combination 653: 10x2c + 4x20c
Combination 654: 10x1c + 1x10c + 4x20c
Combination 655: 8x1c + 1x2c + 1x10c + 4x20c
Combination 656: 6x1c + 2x2c + 1x10c + 4x20c
Combination 657: 4x1c + 3x2c + 1x10c + 4x20c
Combination 658: 2x1c + 4x2c + 1x10c + 4x20c
Combination 659: 5x2c + 1x10c + 4x20c
Combination 660: 2x10c + 4x20c
Combination 661: 5x20c
Combination 662: 50x1c + 1x50c
Combination 663: 48x1c + 1x2c + 1x50c
Combination 664: 46x1c + 2x2c + 1x50c
Combination 665: 44x1c + 3x2c + 1x50c
Combination 666: 42x1c + 4x2c + 1x50c
Combination 667: 40x1c + 5x2c + 1x50c
Combination 668: 38x1c + 6x2c + 1x50c
Combination 669: 36x1c + 7x2c + 1x50c
Combination 670: 34x1c + 8x2c + 1x50c
Combination 671: 32x1c + 9x2c + 1x50c
Combination 672: 30x1c + 10x2c + 1x50c
Combination 673: 28x1c + 11x2c + 1x50c
Combination 674: 26x1c + 12x2c + 1x50c
Combination 675: 24x1c + 13x2c + 1x50c
Combination 676: 22x1c + 14x2c + 1x50c
Combination 677: 20x1c + 15x2c + 1x50c
```

```
Combination 678: 18x1c + 16x2c + 1x50c
Combination 679: 16x1c + 17x2c + 1x50c
Combination 680: 14x1c + 18x2c + 1x50c
Combination 681: 12x1c + 19x2c + 1x50c
Combination 682: 10x1c + 20x2c + 1x50c
Combination 683: 8x1c + 21x2c + 1x50c
Combination 684: 6x1c + 22x2c + 1x50c
Combination 685: 4x1c + 23x2c + 1x50c
Combination 686: 2x1c + 24x2c + 1x50c
Combination 687: 25x2c + 1x50c
Combination 688: 40x1c + 1x10c + 1x50c
Combination 689: 38x1c + 1x2c + 1x10c + 1x50c
Combination 690: 36x1c + 2x2c + 1x10c + 1x50c
Combination 691: 34x1c + 3x2c + 1x10c + 1x50c
Combination 692: 32x1c + 4x2c + 1x10c + 1x50c
Combination 693: 30x1c + 5x2c + 1x10c + 1x50c
Combination 694: 28x1c + 6x2c + 1x10c + 1x50c
Combination 695: 26x1c + 7x2c + 1x10c + 1x50c
Combination 696: 24x1c + 8x2c + 1x10c + 1x50c
Combination 697: 22x1c + 9x2c + 1x10c + 1x50c
Combination 698: 20x1c + 10x2c + 1x10c + 1x50c
Combination 699: 18x1c + 11x2c + 1x10c + 1x50c
Combination 700: 16x1c + 12x2c + 1x10c + 1x50c
Combination 701: 14x1c + 13x2c + 1x10c + 1x50c
Combination 702: 12x1c + 14x2c + 1x10c + 1x50c
Combination 703: 10x1c + 15x2c + 1x10c + 1x50c
Combination 704: 8x1c + 16x2c + 1x10c + 1x50c
Combination 705: 6x1c + 17x2c + 1x10c + 1x50c
Combination 706: 4x1c + 18x2c + 1x10c + 1x50c
Combination 707: 2x1c + 19x2c + 1x10c + 1x50c
Combination 708: 20x2c + 1x10c + 1x50c
Combination 709: 30x1c + 2x10c + 1x50c
Combination 710: 28x1c + 1x2c + 2x10c + 1x50c
Combination 711: 26x1c + 2x2c + 2x10c + 1x50c
Combination 712: 24x1c + 3x2c + 2x10c + 1x50c
Combination 713: 22x1c + 4x2c + 2x10c + 1x50c
Combination 714: 20x1c + 5x2c + 2x10c + 1x50c
Combination 715: 18x1c + 6x2c + 2x10c + 1x50c
Combination 716: 16x1c + 7x2c + 2x10c + 1x50c
Combination 717: 14x1c + 8x2c + 2x10c + 1x50c
Combination 718: 12x1c + 9x2c + 2x10c + 1x50c
Combination 719: 10x1c + 10x2c + 2x10c + 1x50c
Combination 720: 8x1c + 11x2c + 2x10c + 1x50c
Combination 721: 6x1c + 12x2c + 2x10c + 1x50c
Combination 722: 4x1c + 13x2c + 2x10c + 1x50c
Combination 723: 2x1c + 14x2c + 2x10c + 1x50c
Combination 724: 15x2c + 2x10c + 1x50c
Combination 725: 20x1c + 3x10c + 1x50c
```

Combination 726: 18x1c + 1x2c + 3x10c + 1x50c
Combination 727: 16x1c + 2x2c + 3x10c + 1x50c
Combination 728: 14x1c + 3x2c + 3x10c + 1x50c
Combination 729: 12x1c + 4x2c + 3x10c + 1x50c
Combination 730: 10x1c + 5x2c + 3x10c + 1x50c
Combination 731: 8x1c + 6x2c + 3x10c + 1x50c
Combination 732: 6x1c + 7x2c + 3x10c + 1x50c
Combination 733: 4x1c + 8x2c + 3x10c + 1x50c
Combination 734: 2x1c + 9x2c + 3x10c + 1x50c
Combination 735: 10x2c + 3x10c + 1x50c
Combination 736: 10x1c + 4x10c + 1x50c
Combination 737: 8x1c + 1x2c + 4x10c + 1x50c
Combination 738: 6x1c + 2x2c + 4x10c + 1x50c
Combination 739: 4x1c + 3x2c + 4x10c + 1x50c
Combination 740: 2x1c + 4x2c + 4x10c + 1x50c
Combination 741: 5x2c + 4x10c + 1x50c
Combination 742: 5x10c + 1x50c
Combination 743: 30x1c + 1x20c + 1x50c
Combination 744: 28x1c + 1x2c + 1x20c + 1x50c
Combination 745: 26x1c + 2x2c + 1x20c + 1x50c
Combination 746: 24x1c + 3x2c + 1x20c + 1x50c
Combination 747: 22x1c + 4x2c + 1x20c + 1x50c
Combination 748: 20x1c + 5x2c + 1x20c + 1x50c
Combination 749: 18x1c + 6x2c + 1x20c + 1x50c
Combination 750: 16x1c + 7x2c + 1x20c + 1x50c
Combination 751: 14x1c + 8x2c + 1x20c + 1x50c
Combination 752: 12x1c + 9x2c + 1x20c + 1x50c
Combination 753: 10x1c + 10x2c + 1x20c + 1x50c
Combination 754: 8x1c + 11x2c + 1x20c + 1x50c
Combination 755: 6x1c + 12x2c + 1x20c + 1x50c
Combination 756: 4x1c + 13x2c + 1x20c + 1x50c
Combination 757: 2x1c + 14x2c + 1x20c + 1x50c
Combination 758: 15x2c + 1x20c + 1x50c
Combination 759: 20x1c + 1x10c + 1x20c + 1x50c
Combination 760: 18x1c + 1x2c + 1x10c + 1x20c + 1x50c
Combination 761: 16x1c + 2x2c + 1x10c + 1x20c + 1x50c
Combination 762: 14x1c + 3x2c + 1x10c + 1x20c + 1x50c
Combination 763: 12x1c + 4x2c + 1x10c + 1x20c + 1x50c
Combination 764: 10x1c + 5x2c + 1x10c + 1x20c + 1x50c
Combination 765: 8x1c + 6x2c + 1x10c + 1x20c + 1x50c
Combination 766: 6x1c + 7x2c + 1x10c + 1x20c + 1x50c
Combination 767: 4x1c + 8x2c + 1x10c + 1x20c + 1x50c
Combination 768: 2x1c + 9x2c + 1x10c + 1x20c + 1x50c
Combination 769: 10x2c + 1x10c + 1x20c + 1x50c
Combination 770: 10x1c + 2x10c + 1x20c + 1x50c
Combination 771: 8x1c + 1x2c + 2x10c + 1x20c + 1x50c
Combination 772: 6x1c + 2x2c + 2x10c + 1x20c + 1x50c
Combination 773: 4x1c + 3x2c + 2x10c + 1x20c + 1x50c

```
Combination 774: 2x1c + 4x2c + 2x10c + 1x20c + 1x50c
Combination 775: 5x2c + 2x10c + 1x20c + 1x50c
Combination 776: 3x10c + 1x20c + 1x50c
Combination 777: 10x1c + 2x20c + 1x50c
Combination 778: 8x1c + 1x2c + 2x20c + 1x50c
Combination 779: 6x1c + 2x2c + 2x20c + 1x50c
Combination 780: 4x1c + 3x2c + 2x20c + 1x50c
Combination 781: 2x1c + 4x2c + 2x20c + 1x50c
Combination 782: 5x2c + 2x20c + 1x50c
Combination 783: 1x10c + 2x20c + 1x50c
Combination 784: 2x50c
Combination 785: 1x100c
```

3 Q3 (20 points) Triple Step

A child can hop up a staircase with n steps, taking either 1, 2, or 3 steps at a time. Write a method to count how many ways the child can run up the stairs.

Hints: * Approach this problem from the top down. What was the child's final hop? * If we knew the number of paths to each step before step 100, could we use that information to calculate the number of steps to reach step 100? * To compute the number of steps required to reach 100, we can add the number of steps required to reach 99, 98, and 97. This is because the child can hop 1, 2, or 3 steps at the end. The question is whether we should add or multiply these values. In other words, should it be f(100) = f(99) + f(98) + f(97) or f(100) = f(99) * f(98) * f(97)? * If you have an inefficient recursive program, you can try optimizing it through memorization.

Questions * Part 1. Explain your thought process, data structures used, and complexity (time and space) using big O notation.

- Part 2. Code and test your solution.
- Part 3: Explain what do you need to change in your code to modify the problem fron 1,2,and 3 steps to 1,2,3… m steps (with 0 < m <= n)

Part 1 Here

To solve this problem, I decided to implement a method very similar to dynamic programming for the fibbonacci sequency. This time, however, instead of summing the previous 2 values, I needed to sum the previos m values where m is the length of the step_sizes array (for example [1, 2, 3]). The reason for this is that I know the last unique step I can take to get to my target step is one of my step_size values away. So, the number of unique ways to get to my target step is just the sum of the number of unique ways to get within 1 step_size of the target step. A fundamental assumption of this code is that the order of steps matters (i.e. taking 1 step then 3 is unique compared to 3 steps then 1).

As with using dynamic programming for the fibbonacci sequence, it is easiest to first define my base case of how many ways to do nothing (get to step 0, or don't move) or to get to the first step. I am assuming the step_sizes array will always start with 1 and increment by 1 up to a max step_size of m. Then I just need to loop over all integers between 2 (first uncomputed step) and the target step and compute the sum of the unique ways to get to the previous m steps. I need to take extra caution when computing the number of unique ways to get to a step that is within a step_size of the base of the staircase - for those unique instances I need to add an extra unique way besides the sum of the previous m steps because the child can just step from the base of the staircase directly onto the target step. Also, for steps within a step_size of the base of the staircase I have to be careful not to request an index outside the bounds of my ways array (there is no logical sense of a negative step) - this is handled simply through an if statement (see code below).

I only needed to use a list data structure to implement this code, and then I used list slicing to get the previous m values to compute the next value.

The time complexity of this algorithm is O((n-2)*m) -> O(n*m) where n is the target step and m is the maximum step size. This is because I am using dynamic programming and only need to loop over each step one time to compute it's value. The complexity is multiplied by m because each step requires me to access the previous m elements from the ways array - and this is done for each step (except for the first few, which require less than m lookups).

The space complexity of this algorithm is O(n+1) -> O(n) because I am storing a value for each step. If I wanted to simplify this space complexity, I would, at minimum, only need to store m values in an array (only need to keep track of the previous m values to compute the next step).

[ ]:
```python
# Function to count the number of ways a child can run up the stairs in 1, 2,␣
↪3, ..., m step increments (default of [1, 2, 3])
def count_ways(target_step, max_step_size=3):

    # Initialize the list of ways to climb the stairs with the simplest base␣
↪cases
    ways = [0, 1]

    # Define the allowed step sizes
    step_sizes = [i for i in range(1, max_step_size + 1)]

    # Loop over all steps from 1 to the target step
    for step in range(2, target_step + 1):

        # Figure out how many previous steps are within one step of the current␣
↪step
        num_prev_steps_within_one_step = len(step_sizes) if step >␣
↪len(step_sizes) else step

        # The ways to get the the current step is the sum of the unique ways to␣
↪get to a step within one step of the current step (plus one additional step␣
↪if current step is in the step_sizes list - can do the whole thing uniquely␣
↪by taking a single step)
        ways.append(sum(ways[step - num_prev_steps_within_one_step : step] + [1␣
↪if step in step_sizes else 0]))

    # Return the answer
    return ways[target_step]
```

```
# Test the code
target_step   = 10
max_step_size = 3
print(f'The number of ways to climb {target_step} steps given steps sizes of␣
↪{[i for i in range(1, max_step_size + 1)]} is: {count_ways(target_step,␣
↪max_step_size)}')
# print(ways)
```

```
The number of ways to climb 10 steps given steps sizes of [1, 2, 3] is: 274
```

```
[ ]: # Test the code
target_step   = 10
max_step_size = 4
print(f'The number of ways to climb {target_step} steps given steps sizes of␣
↪{[i for i in range(1, max_step_size + 1)]} is: {count_ways(target_step,␣
↪max_step_size)}')
# print(ways)
```

```
The number of ways to climb 10 steps given steps sizes of [1, 2, 3, 4] is: 401
```

Part 3 Here

I would not need to change anything to make my algorithm work for any max step stize m that fits the criteria of an integer value between 0 and n (target step). This is because I already made the algorithm general enough to accept an arbitrary maximum step size based on the criteria in the problem statement. One modification (as I mentioned in Part 1) I could do to reduce the space complexity is I could change the ways array to be only the size m and then keep track of only the previous m values that are needed to compute the number of unique ways to get to the next step. Then, after I compute the new step's value, I just shift all the elements to the left and change the last value to the new computed value.

4 Q4 (40 points) - Book word search

A search engine needs to be designed for a book that contains chapters and subchapters, each having section numbers and contents (in addition to the title). The search feature should retrieve information based on a specific term within the content section/title of chapter/subchapters. The search results should be sorted according to the number of occurrences of this term in each subchapter's content/title. You can use the same novel book selected in Group Project 2 and add more functionalities to your own structure to complete the search engine. Each chapter should have a title and content. The search engine should search for the word in both the titles and contents.

The search engine needs to have the ability to::

- To proceed with the search, please enter only one word at a time as the search term.
- Identify occurrences of search words by traversing chapter/subchapter content/title.
- Generate a ranked list of chapters based on the number of times the search term appears within their content sections, with the chapters having the highest occurrence count appearing at the top.

The output should display:

- Chapter title and section number.
- Number of times the search term appears in the content/title of each chapter.
- List the top 20 results sorted by frequency.

The question is asking for algorithms and/or methods that can efficiently search through the chapters of a book and rank the search results based on how frequently the search term appears in the content. This means that a search engine should be able to parse through the subchapter content, count the occurrences of the specified term, and present search results that are ranked by how often the term appears within the content of the subchapters. Let me know if you need me to clarify anything else.

Hint: * Create a dictionay for each chapter/subchapter. The dictionary will include the term and the frequency. * Modify your Book/Chapter class add the content, sectionNumber and a indexOfTerm (key:word,value:Frequency). * As a proof of concept of yout solution, add 2-3 paragraph for 2-3 chapters of your book manyally.

Questions * Part 1. Explain your approach. Please provide insight into your thought process, including the data structures, algorithms, and complexities (in terms of both time and space) using Big O Notation. Separate your analysis for the Indexation and Searching processes. * Part 2. Code and test your solution.

Initial section below is just some pre-processing to get the book initialized. As I will discuss below, I used my class from the group project 2 and augmented it with the additional features I needed for this problem (see discussion below).

[ ]:
```
# import statements
import re

# Hardcode some content for the book
book_toc = '''1. Introduction to Python        1
1.1 Introduction        1
1.2 Software        1
1.3 Development Tools        4
1.3.1 Advanced Python Tools        5
2. Data Types and Variables        25
2.1 Python Integer Values        25
3. Operators        53
3.1 Python Expressions and Operators        53'''
book_content = '''There is no content in the textbook for this chapter, but␣
↪instead content is in the following sub-chapters.
```

```
Python is a free general- purpose programming language with beautiful syn-tax. ␣
↪It  is  available  across  many  platforms  including  Windows,  Linux  and ␣
↪macOS. Due to its inherently easy to learn nature along with object oriented␣
↪features, Python is used to develop and demonstrate applications quickly. It␣
↪has the batteries included philosophy wherein the standard programming␣
↪language  comes  with  a  rich  set  of  built-  in  libraries.  It's  a ␣
↪known  fact  that  developers spend most of their time reading code rather␣
↪than writing it and Python  can  speed  up  software  development.  Hosting ␣
↪solutions  for  Python  applications are also very cheap. The Python␣
↪Software Foundation (PSF) nur-tures the growth of the Python programming␣
↪language. A versatile language like  Python  can  be  used  not  only  to ␣
↪write  simple  scripts  for  handling  file  operations but also to develop␣
↪massively trafficked websites for corporate IT organizations [1].
According to IBM Research: Software development refers to a set of com-puter ␣
↪science  activities  dedicated  to  the  process  of  creating,  designing, ␣
↪deploying and supporting software. Software itself is the set of␣
↪instructions or programs that tell a computer what to do. It is independent␣
↪of the hard-ware and makes computers programmable. There are three basic␣
↪types:
It is no longer surprising to hear that Python is one of the most popular␣
↪lan-guages  among  developers  and  in  the  data  science  community. ␣
↪While  there  are numerous reasons behind Python's popularity, it is␣
↪primarily because of two core reasons: [redacted]. There  are  numerous ␣
↪reasons  to  use  Python  for  data  science.  For  now  we'll  discuss ␣
↪some  of  the  Python  tools  most  widely  used  by  developers,  coders ␣
↪and data scientists across the world. These tools are useful for many␣
↪different purposes if you know how to use them correctly. So, without␣
↪further delay, let's look at the best Python tools out there!
Selenium:  This  is  undoubtedly  one  of  the  best  Python  development ␣
↪tools.  It  is  an  open-  source  automation  framework  for  web ␣
↪applications.  With  Selenium, you can write test scripts in many other␣
↪programming languages, including Java, C#, PHP, Perl, Ruby and .NET.
↪Furthermore,  you  can  perform  tests  from  any  browser  (Chrome, ␣
↪Firefox,  Safari, Opera and Internet Explorer) in all of the three major␣
↪operating sys-tems  -  Windows,  macOS  and  Linux.  You  can  also ␣
↪integrate  Selenium  with  tools like JUnit and TestNG for managing test␣
↪cases and generating reports.
There is no content in the textbook for this chapter, but instead content is in␣
↪the following sub-chapters.
In Python, integers are zero, positive or negative whole numbers without a␣
↪fractional part and with unlimited precision, for example 0, 100, -10. The␣
↪fol-lowing are valid integer literals in Python:
There is no content in the textbook for this chapter, but instead content is in␣
↪the following sub-chapters.
```

```
A literal value like 34 and a variable like x are examples of simple␣
↪expressions. Values and variables can be combined with operators to form␣
↪more complex expressions.  In  Section  2.1  we  saw  how  we  can  use  the␣
↪ +  operator  to  add  integers  and  concatenate  strings.  Program  3.1 ␣
↪(addition.py)  shows  how  the  addition operator (+) can used to add two␣
↪integers provided by the user [11].
'''

# Replace all tabs with spaces in book content
book_content = book_content.replace('\t', ' ')

# Make all characters lowercase
book_content = book_content.lower()

# Remove all special characters and punctuation
to_delete = '''!()-[]{};:'"\,<>./?@#$%^&*_~ï¿½1234567890+-*'''
for char in to_delete:
    book_content = book_content.replace(char, '')

# Change all duplicate spaces to single spaces
book_content = re.sub(' +', ' ', book_content)

toc_lines     = book_toc.split('\n')
content_lines = book_content.split('\n')
```

[ ]:

```
####----------------------------------------------------------------------------------####
#### Code Below Taken from Group Project 2 as Suggested in Question Prompt     ␣
↪        ####
#### Added additional attributes to the node class for content and dictionary␣
↪of terms ####
#### Also added a method to print toc with content (for debugging purposes)    ␣
↪        ####
####----------------------------------------------------------------------------------####

# Create the Node class for the general tree
class Node:

    # Initialize the class, had attributes for chapter text, chapter id, and␣
↪chapter level and a list of children
    def __init__(self, text, id, level, page, content):
        self.chapter_text  = text
        self.chapter_id    = id
        self.chapter_level = level
        self.page_number   = page
        self.content       = content
        self.dict_of_terms = {}
        self.children = []
```

```
    # Methods to add children
    def add_child(self, obj):
        self.children.append(obj)

    # Method to get children based on list index
    def get_child(self, idx):
        return self.children[idx]

    # Method to print the tree using pre-order traversal
    def print_toc(self, out, max_char=100):
        txt = f'{"  " * (self.chapter_level+1)}{self.chapter_id} {self.
↪chapter_text} '
        out.append(f'{txt}{"." * (max_char - len(txt) - len(self.page_number) -␣
↪2)} {self.page_number}') # Adds indentation based on chapter level
        for child in self.children:
            child.print_toc(out)

    # Method to print the tree using pre-order traversal
    def print_toc_with_content(self, out, max_char=100):
        txt = f'{"  " * (self.chapter_level+1)}{self.chapter_id} {self.
↪chapter_text} '
        out.append(f'{txt}{"." * (max_char - len(txt) - len(self.page_number) -␣
↪2)} {self.page_number}') # Adds indentation based on chapter level
        out.append(f'{"  " * (self.chapter_level+1)}{self.content}')
        for child in self.children:
            child.print_toc_with_content(out)
```

```
[ ]: ####----------------------------------------------------------------------####
     #### Code Below Taken from Group Project 2 as Suggested in Question Prompt ####
     ####----------------------------------------------------------------------####

     # Initialize the book with the title
     book = Node('Python for Beginners', '', -1, '0', '')

     # Loop over all lines
     for i, line in enumerate(toc_lines):

         # Remove any carriage returns from the end of the lines
         line = line.rstrip()

         # Get the page number at the end of the line
         page_number = re.findall('[0-9]*$', line)[0]

         # Use clean line to delete page numbers at the end of the line
         clean_line = re.sub('[ \t]*[0-9]*$', '', line)
```

```python
    # Use Regular expression to find only the chapter level values
    chapter_id = re.findall('^[0-9.]+', clean_line)[0]

    # Remove chapter id values from the clean line
    clean_line = re.sub('^[0-9. ]+', '', clean_line)

    # Find the number of occurances of te pattern '.[0-9]' in the chapter id
    chapter_level = len(re.findall('\.[0-9]', chapter_id))

    # If the chapter level is 0 (main chapter)
    if chapter_level == 0:

        # Parse the chapter id to get the chapter number
        chapter_numbers = [int(chapter_id.split('.')[0])]

        # Add the chapter node to the book node as a child
        book.add_child(Node(clean_line, chapter_id, chapter_level, page_number,␣
↪content_lines[i]))

    else:

        # Start with the root node (book)
        prev = book

        # Walk down the tree using chapter numbers in the chapter id, excluding␣
↪the last one (corresponds to the new node being added)
        for chapter_number in chapter_numbers[:-1]:

            # Get the correct child based on chapter number
            prev = prev.get_child(chapter_number-1)

        # After looping, will have the parent node for the new node, add the␣
↪new node as a child
        prev.add_child(Node(clean_line, chapter_id, chapter_level, page_number,␣
↪content_lines[i]))

# Get the output from the print_tree method
out = []
book.print_toc(out)

# Print the output
for line in out:
    print(line)
```

```
 Python for Beginners
… 0
  1. Introduction to Python
```

```
… 1
    1.1 Introduction
… 1
    1.2 Software
…
1
    1.3 Development Tools
… 4
      1.3.1 Advanced Python Tools
… 5
  2. Data Types and Variables
… 25
    2.1 Python Integer Values
… 25
  3. Operators …
… 53
    3.1 Python Expressions and Operators
… 53
```

[ ]:
```python
# Get the output from the print_tree method
out = []
book.print_toc_with_content(out)

# Print the output
for line in out:
    print(line)
```

```
 Python for Beginners
… 0

  1. Introduction to Python
… 1
  there is no content in the textbook for this chapter but instead content is in
the following subchapters
    1.1 Introduction
… 1
    python is a free general purpose programming language with beautiful syntax
it is available across many platforms including windows linux and macos due to
its inherently easy to learn nature along with object oriented features python
is used to develop and demonstrate applications quickly it has the batteries
included philosophy wherein the standard programming language comes with a rich
set of built in libraries its a known fact that developers spend most of their
time reading code rather than writing it and python can speed up software
development hosting solutions for python applications are also very cheap the
python software foundation psf nurtures the growth of the python programming
language a versatile language like python can be used not only to write simple
scripts for handling file operations but also to develop massively trafficked
websites for corporate it organizations
```

1.2 Software
…
1

according to ibm research software development refers to a set of computer science activities dedicated to the process of creating designing deploying and supporting software software itself is the set of instructions or programs that tell a computer what to do it is independent of the hardware and makes computers programmable there are three basic types

1.3 Development Tools
… 4

it is no longer surprising to hear that python is one of the most popular languages among developers and in the data science community while there are numerous reasons behind pythons popularity it is primarily because of two core reasons redacted there are numerous reasons to use python for data science for now well discuss some of the python tools most widely used by developers coders and data scientists across the world these tools are useful for many different purposes if you know how to use them correctly so without further delay lets look at the best python tools out there

1.3.1 Advanced Python Tools
… 5

selenium this is undoubtedly one of the best python development tools it is an open source automation framework for web applications with selenium you can write test scripts in many other programming languages including java c php perl ruby and netfurthermore you can perform tests from any browser chrome firefox safari opera and internet explorer in all of the three major operating systems windows macos and linux you can also integrate selenium with tools like junit and testng for managing test cases and generating reports

2. Data Types and Variables
… 25

there is no content in the textbook for this chapter but instead content is in the following subchapters

2.1 Python Integer Values
… 25

in python integers are zero positive or negative whole numbers without a fractional part and with unlimited precision for example the following are valid integer literals in python

3. Operators …
… 53

there is no content in the textbook for this chapter but instead content is in the following subchapters

3.1 Python Expressions and Operators
… 53

a literal value like and a variable like x are examples of simple expressions values and variables can be combined with operators to form more complex expressions in section we saw how we can use the operator to add integers and concatenate strings program additionpy shows how the addition operator can used to add two integers provided by the user

Part 1 Here

To initialize this problem, I just used the Node class from my Goup Project 2 code. This is very similar to the pre-existing code that was in the template, but was in a format I was more familiar with. I did modify my GP2 class to include the additional attributes (content and dictionary of terms) that are needed for this question. I also added a debugging method that printed the content of each chapter along with the table of contents so that I could make sure that my pre-processing steps (getting content into the book) were working properly. The output of that code is shown above.

Book Indexing: For creading the dictionary of word occurance frequency, all I needed to do was create a function that started from the root node (in this case, the book title node) and then recursively looped over all the children of each node until all nodes (chapters) have been processed. At each node, the process for populating the dictionary of terms was very simple and was actually the same as I used previously on question 1 - loop over all words, if the word doesn't exist, initialize the frequency to 1, if the word does exist in the dictionary, then increment the frequency number. After this procedure is complete, I have a fully populated dictionary of terms for each node (chapter/subchapter) that contains all words that have occured and their frequency of occurance.

For data structures, I used the recommended dictionary to create the dictionary of terms with the key being the unique word and the value being the frequency of occurance.

The time complexity of this code is O(N*m_n) where N is the number of chapters/subchapters and m_n is the number of words for each specific chapter/subchapter. This is because we must loop over every chapter and every word in every chapter to compute the frequency of occurance of each word.

The space complexity of this code is O(Nu_n*2) -> O(Nu_n) where N is the number of chapters/subchapters and u_n is the number of unique words per chapter. The *2 which simplifies out is because each word also requires a frequency, but that is not kept for proper big O notation.

Word Searching: Once the indexing is complete, it is very easy to search the book for a specific word occurance in each chapter/subchapter. All I needed to do was create a function that looped over every node in the book and checked to see if the search word exists in the dictionary of terms. If it did, then I added it to a list of outputs along with the corresponding chapter that was searched. Finally, I created a second function just to format the output (was simpler to do it with 2 functions since I used recursion for the searching) and only look at the top 20 results. Since there weren't more than 20 chapters/subchapters, this last filtering step didn't actually do anytihng. If I had input the full content of the entire book, however, then the top 20 filtering would have been useful.

For data structures, I used a list to keep track of all the results from each chapter and then merged that list into a final text output printed for the user.

The time complexity of this code is O(N) where N is the number of chapters/subchapters assuming the dictionary of terms hash tables have no collisions (and thus dictionary lookup complexity would be O(1)). In the worst case scenario (if the dictionary hash tables have only collisions, and thus their time complexity is O(u_n)), the time complexity would be O(N*u_n) where N is the number of chapters/subchapters and u_n is the number of unique words per chapter.

For space complexity, this code does not add any additional data into memory - it only looks at existing data in memory and prints a result. If the result was stored in memory instead of printed, then the space complexity would be O(1) since we are limiting our output to the top 20 results.

```
[ ]: # populate the dictionary with words and frequencies by chapter/subchapter
def IndexBook(book):

    # Parse the node's content into a list of words
    words = book.content.split(' ')

    # Loop over all words
    for word in words:

        # If the word is not in the dictionary, add it with an initial␣
↪frequency of 1, else increment the frequency
        if word not in book.dict_of_terms.keys():
            book.dict_of_terms[word] = 1
        else:
            book.dict_of_terms[word] += 1

    # Loop over all children
    for child in book.children:

        # Recursively call the function on the child
        IndexBook(child)
```

```
[ ]: # the search will review all content of all chapters/subchapters and will␣
↪provide a list of the top 20 (relevance by frequency)
# the list will include at least Chapter/subchapter (number) and frequency.
def search_book_dict_of_terms(book, word):

    # Initialize a list of tuples to store the results for each chapter/
↪subchapter where the word exists
    results = []

    # If the word is in the dictionary of terms, add it to the results list
    if word in book.dict_of_terms.keys():
        results.append((book.dict_of_terms[word], f'{book.chapter_id} {book.
↪chapter_text}')) # Note I have combined the chapter_id and chapter_text␣
↪attributes for readibility of the output

    # Loop over all children
    for child in book.children:

        # Recursively call the function on the child and extend the results␣
↪list (extend is used to add the tuples from the child to the results list␣
↪since recursive output is a list itself - don't want to add a list to a list)
        results.extend(search_book_dict_of_terms(child, word))

    # Return the results list
    return sorted(results)
```

```
# This function just formats the output, calls the helper function above to get␣
↪the results
def SearchTerm(book, word):
    return '\n'.join([f'word "{word}" occurs {freq:>2} times in {chapter}' for␣
↪freq, chapter in sorted(search_book_dict_of_terms(book,word), reverse=True)[:
↪20]]) # Note that we are filtering out the top 20 results here
```

[ ]:
```
# First, we need to run the indexing fucntion to populate the dictionary of␣
↪terms for each chapter/subchapter
IndexBook(book)
```

[ ]:
```
# Now, we can search for a word and get the results
print(SearchTerm(book, 'python'))

# Now, we can search for a word and get the results
print(SearchTerm(book, 'the'))
```

```
word "python" occurs  7 times in 1.1 Introduction
word "python" occurs  4 times in 1.3 Development Tools
word "python" occurs  2 times in 2.1 Python Integer Values
word "python" occurs  1 times in 1.3.1 Advanced Python Tools
word "the" occurs  5 times in 1.3 Development Tools
word "the" occurs  5 times in 1.1 Introduction
word "the" occurs  3 times in 3.1 Python Expressions and Operators
word "the" occurs  3 times in 1.2 Software
word "the" occurs  2 times in 3. Operators
word "the" occurs  2 times in 2. Data Types and Variables
word "the" occurs  2 times in 1.3.1 Advanced Python Tools
word "the" occurs  2 times in 1. Introduction to Python
word "the" occurs  1 times in 2.1 Python Integer Values
```