1. flowchart

Follow the steps on the flowchart to write the code step by step, but notice that some of the diagrams on the flowchart are not fully drawn. Use the code to complete them.

2. matrix multiplication

2.1

For M1, it generates 5 rows and, for each row, generates 10 random integers between 0 and 50. These integers are appended to the M1 matrix. Similarly, for M2, it creates a 10x5 matrix using "for" loops.

2.2

The function first checks if the number of columns in M1 is equal to the number of rows in M2. If they are not equal, the function prints an error message and returns None. If the dimensions are valid for matrix multiplication, it initializes an empty list to store the result. It then iterates over the rows in M1 and, for each row, iterates over the columns in M2. For each element in the result matrix, it initializes a variable called element to 0. It then iterates over the columns in M1 and calculates the element by adding the product of corresponding elements from M1 and M2. Finally, it appends the element to the current row.

3. Pascal triangle

The 100^{th} row is [1,99,4851,156849……156849,4851,99,1]

The 200^{th} row is [1,199,19701,1293699……1293699,19701,199,1]

When k is 1, the first row only has one element 1; In other cases, since the first element of each line is 1, write 1 to the current_ line. After that, use the recursive method to obtain the value of the previous line, and then call a for loop to calculate the value of each element between the first element and the last element of each line. Finally, add the last element 1 of each line to the end of this line, and return the result.

4. add or double

Firstly, define a variable to store the minimum number of steps required; Assign the value of x to a new variable original_ input, because the value of x will eventually change. If x is 1, then the minimum number of steps is 0; If x is less than 1, greater than 100, or not an integer, an error is reported. In other cases, using a while loop, if x is between 1 and 100 (including 100), corresponding operations are made based on the parity of x. For each operation, the number of steps is increased by 1. Finally, output the results.

5. dynamic programming

5.1

I got inspired by reading the implementation method of decimal to ternary conversion:

https://blog.51cto.com/u_16213459/7819614

The overall idea is to select operators through an index of 8-bit ternary numbers.

First, a string variable is initialized with the numbers from 1 to 9.

Next, the code enters a loop that iterates from 0 to 6560, because there are 3 possible operations (+, -, and no operation) for each of the 8 positions in the expression, giving a total of 3^8 = 6561

combinations.

Within this loop, another loop is executed 8 times (from 0 to 7) to construct the expression. For each iteration, the current value of i is checked to determine the operation to be used. Encode the “ ”, “+”, “-” with numbers 0, 1, and 2 respectively.

After determining the operation for the current position, the variable i is updated by dividing it by 3. This allows the code to move onto the next operation for the next position in the expression. And the string “expr” is updated during each iteration. Once the loop completes, the last digit 9 is appended to the “expr” to complete the expression.

When the value of the expression equals to the target value, write the result to solutions.

5.2

The maximum number of solutions is 26 for target value 1 and 45.

The minimum number of solutions is 6 for target value 88.

Total_solutions is used to store the number of solutions. The code then iterates through a range of numbers from 1 to 100, and calls the find_expression function to obtain a list of expressions for each number. Then, it records the length of the solutions and assign it to count. Finally, write the count in order to Total_solutions.

Various formatting options are provided to customize the plot appearance, such as setting the line color, line width, and label. Finally, the plt.show() function displays the plot on the screen.