南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Assignment 1

Report

张子严
UID: 12132873
量子科学与工程研究院
2021年9月29日

Question 1. Flowchart

According to the Truth Table below, we have 5 possible cases.

	a>b	b>c	a>c	OUTPUT
1	F	F		c, b, a
2	F	T	F	c, a, b
3	F	T	T	a, c, b
4	T	F	F	c, a, b
5	T	T		a, b, c

Hence, we set 5 input to test our program.

Case 1.

Input: a = 1, b = 2, c = 3

Output: 3, 2, 1

Case 2.

Input: a = 1, b = 3, c = 2

Output: 2, 1, 3

Case 3.

Input: a = 2, b = 3, c = 1

Output: 2, 1, 3

Case 4.

Input: a = 2, b = 1, c = 3

Output: 3, 2, 1

Case 5.

Input: a = 3, b = 2, c = 1

Output: 3, 2, 1

Resource:

```
(base) Ziyans-MacBook-Pro:ESE5023_Assignments_12132873 ziyanzhang$ python3 PS1_1.py
3, 2, 1
2, 1, 3
2, 1, 3
3, 2, 1
3, 2, 1
(base) Ziyans-MacBook-Pro:ESE5023_Assignments_12132873 ziyanzhang$ cat PS1_1.py
def flowchart(a, b, c):
    if a > b:
        if b > c:
            print('%s, %s, %s' % (a, b, c))
            return 0
        else:
            if a > c:
                print('%s, %s, %s' % (a, c, b))
                return 0
            print('%s, %s, %s' % (c, a, b))
    else:
        if b > c:
            if a > c:
                print('%s, %s, %s' % (a, c, b))
                return 0
            print('%s, %s, %s' % (c, a, b))
            return 0
        print('%s, %s, %s' % (c, b, a))
        return 0


if __name__ == '__main__':
    flowchart(1, 2, 3)
    flowchart(1, 3, 2)
    flowchart(2, 3, 1)
    flowchart(2, 1, 3)
    flowchart(3, 2, 1)
```

Question 2. Matrix Multiplication

2.1 Make two matrices M1 and M2

Resource:

```
(base) Ziyans-MacBook-Pro:ESE5023_Assignments_12132873 ziyanzhang$ cat PS1_2.py
import numpy as np


def create_matrix():
    m1 = np.random.randint(0, 51, size=(5, 10))
    m2 = np.random.randint(0, 51, size=(10, 5))
    return m1, m2


def Matrix_multip(matrix_1, matrix_2):
    k = 0
    x = 0
    result = []
    row1, length1 = np.shape(matrix_1)
    row2, length2 = np.shape(matrix_2)
    for q in range(row1):
        for i in range(row1):
            for j in range(row2):
                x += int(matrix_1[k][j]) * int(matrix_2[j][i])
            result.append(x)
            x = 0
        k += 1
    return np.reshape(result, (5, 5))


if __name__ == '__main__':
    m1, m2 = create_matrix()
    print('Matrix 1:')
    print(m1, end='\n\n')
    print('Matrix 2:')
    print(m2)
    print(np.matmul(m1, m2))  # Output the correct answer calculated by built-in function
    print(Matrix_multip(m1, m2))  # Output reshaped matrix by our function
```

For the first question, we use `np.random.randint( )` function to randomly generate two matrices named M1 and M2.

2.2 Matrix Multiplication

For question 2.2, we use `numpy` built-in function `np.matmul( )` to verify our approach with simple operator. Results are showed below. We can see that our algorithm works well.

```
(base) Ziyans-MacBook-Pro:ESE5023_Assignments_12132873 ziyanzhang$ python3 PS1_2.py
Matrix 1:
[[33  0  4 23 27 18  0 29 25 16]
 [38 13 11 28 38 10 21 34 18 39]
 [12 16 47 20 32 44 16 29 43 34]
 [37 33 19 29 29 19 19 13  5  8]
 [ 5 10 37 35 24 10 39 48 21 47]]

Matrix 2:
[[27 26 36 11 49]
 [10 29 14 39 22]
 [38 14 30 44 33]
 [23 46 29 46 42]
 [23 38 24 23  3]
 [ 7  9 14 27 42]
 [43 32 31 50  4]
 [41 32 41 29  1]
 [46 21  5 36 49]
 [25 47 16 11 50]]
[[5058 5365 4445 4621 5606]
 [7262 8312 6503 6954 7171]
 [8479 7907 6474 9231 9175]
 [5298 6297 5442 6639 6003]
 [8854 8984 7195 9120 7231]]
[[5058 5365 4445 4621 5606]
 [7262 8312 6503 6954 7171]
 [8479 7907 6474 9231 9175]
 [5298 6297 5442 6639 6003]
 [8854 8984 7195 9120 7231]]
(base) Ziyans-MacBook-Pro:ESE5023_Assignments_12132873 ziyanzhang$
```

Question 3. Pascal triangle

Resource:

```
def pascal_triangle(k):
    """
    Find nth row of the pascal triangle
    :param n:
    :return: a list of row nth of pascal triangle
    """
    length = k - 1
    mid = length//2
    row = []
    for i in range(mid+1):
        num = factorial(length)//(factorial(i)*factorial(length - i))
        row.append(num)
    if k % 2 == 0:
        row.extend(row[::-1])
    else:
        row.extend(row[1::-1])
    return row


if __name__ == '__main__':
    print(pascal_triangle(int(input())))
```

According to the theory, for the nth line of Pascal triangle, it can be represented as two symmetric combination lists that consist of $c_n^i, where\ i = 1,2,3,...,1/2n.$

We have our results of Pascal_triangle(100) and Pascal_triangle(200) below.

```
(base) Ziyans-MacBook-Pro:ESE5023_Assignments_12132873 ziyanzhang$ python3 PS1_3.py
100
[1, 99, 4851, 156849, 3764376, 71523144, 1120529256, 14887031544, 171200862756, 1731030945644, 15579278510796, 126050526132804, 924370524973896, 6186171974825304, 38000770702498296, 215337700647490344, 1130522928399324306, 5519611944537877494, 25144898858450330806, 107196674080761936594, 428786696323047746376, 1613054714739084379224, 5719012170438571889976, 19146258135816088501224, 60629817430084280253876, 181889452290252840761628, 516853642107196237061720, 1399667836569723427057428, 3599145865465003098147672, 8817019464832834471891280, 20560637875127661376774632, 45764000431735762419272568, 97248500917438495140954207, 197443926105102399225573693, 383273503615787010261407757, 711793649572175876199757263, 1265410932572757113244012912, 2154618614921181030658724688, 3515430371713505892127392912, 5498493658321124600506947888, 8247740487481686900760421832, 11868699725888281149874753368, 16390109145274293016493707032, 21726423750712434928840495368, 27651812046361280818524266832, 33796659167774898778196326128, 39674339023040098565708730672, 44739148260023940935799206928, 48467410615025936013782474172, 50445672227278209666740624862 8, 50445672227278209666740624862 8, 48467410615025936013782474172, 44739148260023940935799206928, 39674339023040098565708730672, 33796659167774898778196326128, 27651812046361280818524266832, 21726423750712434928840495368, 16390109145274293016493707032, 11868699725888281149874753368, 8247740487481686900760421832, 5498493658321124600506947888, 3515430371713505892127392912, 2154618614921181030658724688, 1265410932572757113244012912, 711793649572175876199757263, 383273503615787010261407757, 197443926105102399225573693, 97248500917438495140954207, 45764000431735762419272568, 20560637875127661376774632, 8811701946483283447189128, 3599145865465003098147672, 1399667836569723427057428, 516853642107196237061720, 181889452290252840761628, 60629817430084280253876, 19146258135816088501224, 5719012170438571889976, 1613054714739084379224, 428786696323047746376, 107196674080761936594, 25144898858450330806, 5519611944537877494, 1130522928399324306, 215337700647490344, 38000770702498296, 6186171974825304, 924370524973896, 126050526132804, 15579278510796, 1731030945644, 171200862756, 14887031544, 1120529256, 71523144, 3764376, 156849, 4851, 99, 1]
```

We can zoom in for a clearer view.

Question 5. Dynamic programming

Resource:

```
(base) Ziyans-MacBook-Pro:ESE5023_Assignments_12132873 ziyanzhang$ cat PS1_5.py
import matplotlib.pyplot as plt


class Solution:
    def __init__(self, target):
        self.answer = []
        self.target = target
        self.string = ''

    def Find_expression(self, s, pointer, path=''):
        if len(s) == 0 and pointer == self.target:
            self.answer.append(path)
        for i in range(len(s)):
            self.Find_expression(s[i + 1:], pointer + int(s[:i + 1]), path + '+' + s[:i + 1])
            self.Find_expression(s[i + 1:], pointer - int(s[:i + 1]), path + '-' + s[:i + 1])

    def __len__(self) -> int:
        count = 0
        for string in self.answer:
            if string[0] == '+':
                count += 1
        return count

    def __str__(self):
        for string in self.answer:
            if string[0] == '+':
                string = string[1:]
                print(string, end='=')
                print(eval(string))
        return ''


def get_key(my_dict, n):
    result = []
    for item in my_dict.items():
        if item[1] == n:
            result.append(item[0])
    return result
```

```
if __name__ == '__main__':
    target = int(input())
    total_solution = []
    a = Solution(target)
    a.Find_expression("123456789", 0)
    print('Possible Solutions:')
    print(a)
    print('Total number:%d' % len(a))

    # Question 2

    y_dict = {}
    for i in range(1, 101):
        a = Solution(i)
        a.Find_expression("123456789", 0)
        y_dict[i] = len(a)
    max_key = get_key(y_dict, max(y_dict.values()))
    min_key = get_key(y_dict, min(y_dict.values()))
    x = []
    y = []
    for key in y_dict.keys():
        x.append(key)
    for value in y_dict.values():
        y.append(value)
    plt.plot(x, y)
    plt.text(1, 26, 'Max(1,26)', color='r')
    plt.text(45, 26, 'Max(45,26)', color='r')
    plt.text(88, 6, 'Min(88,6)', color='r')
    plt.title('Distribution')
    plt.show()
(base) Ziyans-MacBook-Pro:ESE5023_Assignments_12132873 ziyanzhang$
```

Question 1.

For the function `Find_expression( )` we set 50 as the input.

Input: 50

Output:

```
(base) Ziyans-MacBook-Pro:ESE5023_Assignments_12132873 ziyanzhang$ python3 PS1_5.py
50
Possible Solutions:
1+2+3+4-56+7+89=50
1+2+3-4+56-7+8-9=50
1+2-3+4+56+7-8-9=50
1+2+34-5-6+7+8+9=50
1+2+34-56+78-9=50
1+2-34+5-6-7+89=50
1-2+3-45+6+78+9=50
1-2-3+4+56-7-8+9=50
1-2-3-4-5-6+78-9=50
1-2+34+5+6+7+8-9=50
1-2+34-5-67+89=50
1-2-34-5-6+7+89=50
1-23+4+5-6+78-9=50
1-23-4-5-6+78+9=50
12+3+4-56+78+9=50
12-3-4-5+67-8-9=50
12-3+45+6+7-8-9=50

Total number:17
```

Question 2.

The Distribution Chart is showed below. Axis-x denotes the number, axis-y denotes the aggregation of possible solutions.