Data Structure Task 1 Report

20200901 이효주

Problem1. Roman numeral notation and addition

1. Problem definition)

First, this problem is given a number of test cases (≤1,000) as the input value, and then two Roman numerals are entered while rotating for loop as many test cases as the number of test cases. Add the two Roman numerals entered and output the addition result as Roman numerals and decimal numbers.

2. Approach)

The key to this problem is to convert the input Roman numerals to decimal numbers and to convert the addition result back to Roman numerals.

In the process of converting Roman numerals to decimal numbers, read Roman numerals in order, subtract the value of the current character from the value of the next character if the current character is smaller than the next character, and add the value of the current character if the current character is greater than or equal to the next character. If you add these values repeatedly by the length of the Roman numerals, you will find a decimal value.
For example, in the case of IX, if the current index is I, I will be less than X, so the result value of X-I will be 9. Conversely, if the current index of VI is V, V is greater than I, so the result value of V+I will be 6.

In the process of converting decimal values into Roman numerals, decimal values were considered based on I, V, X, L, C, D, and M, and 4, 40, 400, 9, 90, and 900 additional exceptions were considered. For each Roman numeral of the entered decimal value, find the largest Roman numeral greater than or equal to that value, add it to the result, and subtract the corresponding value from the input value. Repeat this procedure to convert decimal values to Roman numerals.

3. Flow chart & Pseudo code)

// roman character define

Function roma(r: character) {

1. The function roma(r) returns the Roman numeral corresponding to the letter c as an integer.
2. If the letter c is not included in the Roman numeral string, return -1.

Switch (r) {

'I' -> return 1;

'V' -> return 5;

'X' -> return 10;

'L' -> return 50;

'C' -> return 100;

'D' -> return 500;

'M' -> return 1000;

Default -> return -1; } **}**

// roman to integer

Fuction r_to_int(r: string) {

1. Initialize a variable i to 0, which represents the current index of the string being processed.
2. Initialize a variable result to 0, which will hold the final integer value.
3. Initialize a variable len to the length of the input Roman numeral string.
4. While (i <= len) :

    a. Get the integer value of the Roman numeral character at index i using the function roma().

    b. Get the integer value of the Roman numeral character at index i+1 using the function roma().

    c. If a is less than b:

    i. Subtract a from b and add the result to result.

    ii. Increment i by 2 to skip over the next Roman numeral character.

d. Otherwise, add a to result.

i. Increment i by 1 to process the next Roman numeral character.

5. Return the final integer value stored in result.

}

// integer to roman

Function int_to_r(integer a){

1. Create a function called int_to_r that takes an integer input 'a' and returns a string output 'roman'.

2. Allocate memory for 'roman' string with a capacity of 2000 characters.

3. Set 'i' to 0 as the current index of the 'roman' string.

4. While (a > 0) :

if ‘a' >= 1000

then add 'M' to 'roman' and subtract 1000 from 'a'.

Else if 'a' >= 900

add 'C' and 'M' to 'roman' and subtract 900 from 'a'.

Else if >= 500

add 'D' to 'roman' and subtract 500 from 'a'.

Else if 'a' >= 400

add 'C' and 'D' to 'roman' and subtract 400 from 'a'.

Else if 'a' >= 100

add 'C' to 'roman' and subtract 100 from 'a'.

Else if 'a' >= 90

add 'X' and 'C' to 'roman' and subtract 90 from 'a'.

Else if 'a' >= 50

add 'L' to 'roman' and subtract 50 from 'a'.

Else if 'a' >= 40

add 'X' and 'L' to 'roman' and subtract 40 from 'a'.

Else if 'a' >= 10

add 'X' to 'roman' and subtract 10 from 'a'.

Else if 'a' >= 9

add 'I' and 'X' to 'roman' and subtract 9 from 'a'.

Else if 'a' >= 5

add 'V' to 'roman' and subtract 5 from 'a'.

Else if 'a' >= 4

add 'I' and 'V' to 'roman' and subtract 4 from 'a'.

Else

add 'I' to 'roman' and subtract 1 from 'a'.

5. Add "NULL" letter to the end of the roman and return the 'roman' string

}

// main function

Int main(){

1. Declare variables:

Integer num, integer i

roman1 as a two-dimensional character array of size 1001 by 101

roman2 as a two-dimensional character array of size 1001 by 101

2. Read the number of test cases (num) from standard input using scanf.

3 for-loop (i = 0~num)

read two Roman numerals (roman1[i] and roman2[i]) from standard input

4. for-loop (i = 0~num)

- Convert roman1[i] and roman2[i] to integers using the r_to_int function and store the results in a1 and a2, respectively.

- Compute the sum of a1 and a2 and store the result in result_sum.

- Convert result_sum to a Roman numeral using the int_to_r function and store the result in result_roman.

- Print the input Roman numerals, the computed sum in Roman numerals, and the computed sum in integers using printf.

- Free the memory allocated for result_roman using free.

5. Return 0 to indicate successful completion of the program.

}

Problem2. A smart MOUSE

1. Problem definition)

This problem is to find the maximum value of the sum of the columns that pass from the upper left to the lower right in the matrix of M x N. However, the distance traveled from the starting point to the ending point should be the minimum. Multiple test cases may be given, and each test case is entered with a given matrix. The maximum sum obtained shall be printed for each test case.

2. Approach)

The key to this problem is to find the maximum value of the sum of the columns passed by the matrix of M x N, and I felt the need to approach it in a different way because solving the total number of cases with repeat statements creates so much overlap.

Therefore, the key to this code is to use Dynamic Programming to receive a two-dimensional array T, and to find the maximum value of the sum of the numbers of all passing columns when moving from T[0][0] to T[row-1][col-1].

To this end, the code initializes the sum of the two-dimensional arrays using the first input T, and sum[i][j] stores the maximum value of the sum of all the columns that pass when moving from T[0][0] to T[i][j]. The two-dimensional array called sum was used to store previous values.

However, when coloroum and row are 0, since it is an outer area, the case of volume=0 or row=0 was considered separately. sum[i][0](i>0) is equal to sum[i-1][0] and T[i][0], and sum[0][j](j>0) is equal to sum[0][j-1] and T[0][j].

And if coloum and row are not 0, the larger value of sum[i-1][j] and sum[i][j-1] is selected, and T[i][j] is added to the value. Considering the case of Sum[i][j], you can choose the maximum value between the two values, whether to come down from the top or from left to right. Compare the values of sum[i-1][j] and sum[i][j-1], select a large value, add the value currently in T[i][j], and store it in sum[i][j].

The sum[row-1] [col-1] obtained thus will be the final result, which was stored in the array result and output at the end.

3. Pseudo code)

```
set MAX = 100
set TEST = 10000

function main {
    read num from input

    set testnum = num
    set result as an array of size num

    while ( testnum > 0 )
        read row, col from input // matrix size

        set T as a 2D array of size row x col
        set sum as a 2D array of size MAX x MAX
```

```
for i from 0 to row-1 do:
    for j from 0 to col-1 do:
        read T[i][j] from input

for i from 0 to MAX-1 do:
    for j from 0 to MAX-1 do:
        set sum[i][j] = 0

set sum[0][0] = T[0][0]

for i from 1 to row-1 do:
    set sum[i][0] = sum[i-1][0] + T[i][0]

for j from 1 to col-1 do:
    set sum[0][j] = sum[0][j-1] + T[0][j]

for i from 1 to row-1 do:
    for j from 1 to col-1 do:
        if sum[i-1][j] > sum[i][j-1] then:
            set sum[i][j] = sum[i-1][j] + T[i][j]
        else:
            set sum[i][j] = sum[i][j-1] + T[i][j]

set result[num-testnum] = sum[row-1][col-1]
decrement testnum
```

```
for i from 0 to num-1 do:
    print result[i]

return 0 }
```

4. Flow Chart in table

0	1	0	1
0	0	4	0
2	0	0	2
0	2	0	0
0	0	3	0

The example given in the problem is stored in the T[row][col] matrix above.
Accordingly, a two-dimensional array sum[row][col] to store the results is declared.

0	1	1	2
0			
2			
2			
2			

First, the sum value is obtained when row and column are 0. If Row is 0, that is, if sum[0][i], we have to go in the right direction, so we add the value of sum[0][j-1] and the current T[0][j] and add the result to sum[0][j]. The same is true if the column is 0.

0	1	1	2
0	1		
2			
2			
2			

If you think of Sum[1][1] from the top, from left to right, that is, from sum[0][1] or sum[1][0], you can compare the sum value and store it in the sum array by adding the current T[1][1] value to the larger value of sum[0][1]

0	1	1	2
0	1	5	5
2	2	5	7
2	4	5	7
2	4	8	8

If this is repeated by row and col, the result value present in sum[4][3] will be the final output value.