**Data Structure Task 2 Report**

20200901 이효주

**Problem1.**

\* In the case of task 1, we used the <math.h> library in the code and could not compile on cspro, so we linked the math library to -lm.

1. **Problem definition)**

When tiles are laid in a rectangular bathroom of N times M given as an input, there is a condition that the size of the tile is given only by the square number of 2. Therefore, we first resize a given bathroom size so that it can be divided by a square of two, then calculate the minimum number of tiles for each divided area, and add them all together to calculate the total number of tiles.

1. **Approach)**

Find the largest value K among the square numbers of 2 equal to or less than the value of N and M given as the input value. Because you have to use the smallest number of tiles, you have to find the largest value you can put in a given rectangle. For example, if N=5 and M=6, then K=4.

And if N=0 or M=0, it will return 0, and if N=1 or M=1, it will return N\*M.

|  |  |  |  |
| --- | --- | --- | --- |
| 1X1 | 1X1 | 1X1 | 1X1 |

|  |
| --- |
| 1X1 |
| 1X1 |
| 1X1 |

As described above, when N or M is 1, the value of K, which is the square of 2, cannot be greater than 1, and thus the number of N or M must be returned.

If K is obtained, the number of tiles required to tile a square of size K is one. Except for the K\*K-sized bathroom, you can get (N, M-K), (N-K, M) again like the previous process and subtract (N-K, M-K) from it.  
  
Repeat steps 2-3 recursively for each segmented area. At this time, if N or M of the size of the divided area becomes 1, one tile is laid. Calculate the total number of tiles by obtaining the sum of the number of tiles for each segmented area.

Assuming that there is a 5 x 6 bathroom, if K = 4 and the space of K times K is subtracted, then N X (M-K) and (N-K) X M are recursively obtained and subtracted by (N-K) X (M-K).

|  |  |
| --- | --- |
| 4 x 4  (K = 4) | (N-K)  X  M |
| N X (M-K) | (N-K)  X  (M-K) |

1. **Pseudo code)**

// Define a function to find the minimum number of tiles required to cover a rectangular area

// with dimensions n and m

function find\_min\_tile(int n, int m):

// Check if dimensions are invalid (i.e., negative)

if n < 0 or m < 0:

return 0

// Case 1: If either dimension is 0 or 1, the minimum number of tiles is n \* m

if n == 0 or m == 0 or n == 1 or m == 1:

return n \* m

// Determine the maximum square tile size that can fit within the given area

k = log2(min(n, m))

size = 2^k

// Case 2: If the dimensions are already a square tile, the minimum number of tiles is 1

if n == size and m == size:

return 1

// Case 3: Divide the rectangular area into 4 quadrants, and recursively determine the minimum

// number of tiles required to cover each quadrant

result1 = find\_min\_tile(n, m - size)

result2 = find\_min\_tile(n - size, m)

result3 = find\_min\_tile(n - size, m - size)

// Add up the results of the 3 recursive calls and subtract the result for the overlapping quadrant

return 1 + result1 + result2 - result3

// Main function to get user input and output the result

function main():

// Get user input for the dimensions of the rectangular area

N, M = input()

// Call the find\_min\_tile function and output the result

result = find\_min\_tile(N, M)

print(result)

**Problem2.**

1. **Problem definition)**

It's a question of solving an equation that has the sum of s made up of n unknowns. Since all unknowns must be integers of 0 or more, the number of non-negative integer solutions should be calculated using recursion.

1. **Approach)**

First, two integers n and s are given as input values.  
If s is less than 0, the solution is negative, so 0 must be returned.  
If s is 0, then there will be no solution if n is 0, and if n is not 0, there will be one solution. For example, if x1 + x2 + x3 = 0, there would be one solution: x1 = x2 = x3 = 0. If s is 1, return n; if n is 1, return 1 because x1=s.  
  
Otherwise, the variable num is initialized to s and the variable sum is equal to or greater than 0, while num is reduced by 1, and the result of receiving n-1 and num as a factor is added to the sum to finally return the sum value.

For example, given that n=4 and s=2 are inputs, a solution of x satisfying x1+x2+x3+x4=2 can be obtained. At this time, if x1 is fixed and x2 + x3 + x4 = 2, x2 + x3 + x4 = 1, and x2 + x3 + x4 = 0 can be recursively obtained again. The final number of solutions can be returned by adding the obtained values.

1. **Pseudo code)**

// Define the function "find\_int\_value". This function receives two integer factors n and s.

function find\_int\_value(int n, int s):

// Check if s is less than 0, If s is less than 0, it returns 0 because it results in a negative integer solution.

if s < 0:

return 0

// Check if s is 0, If n is 0, return 0, otherwise return 1.

else if s = 0:

if n = 0: return 0

return 1

// Check if s is 1, If s is 1, return n. n solutions can come out (1, 0, 0...).

else if s = 1:

return n

// Check if n is 1, If n is 1, return 1.

else if n = 1:

return 1

// While num is greater than or equal to 0, num is reduced by 1, and the result of calling the find\_int\_value function by receiving n-1 and num as a factor is added to the sum.

else

int num = s, sum = 0;

while (num >= 0) {

sum += find\_int\_value(n-1, num);

num--

}

return sum

// Main function to get user input and output the result

function main():

// Get user input

n, s = input()

// Call the find\_int\_value function and output the result

result = find\_min\_tile(n, s)

print(result)