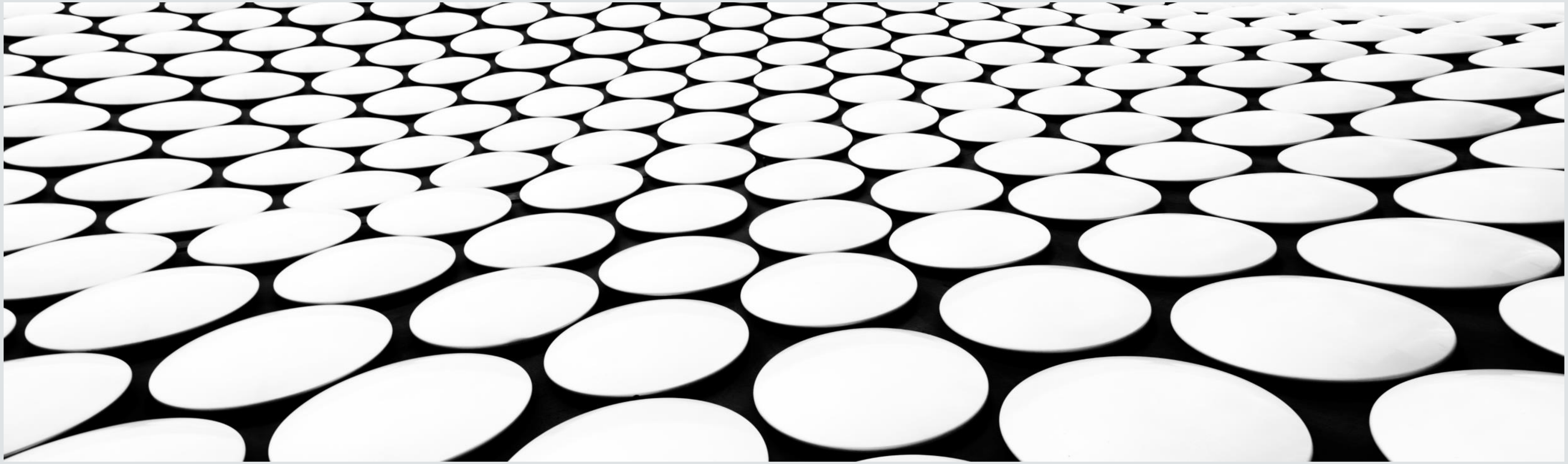
ALGORITHMS DESIGN PROJECT 1/2020

1167. BICOLORED HORSES

BY APITSARA CHOPPRADIT ID: 5710497



QUESTION

Every day, farmer Ion (this is a Romanian name) takes out all his horses, so they may run and play. When they are done, farmer Ion has to take all the horses back to the stables. In order to do this, he places them in a straight line and they follow him to the stables. Because they are very tired, farmer Ion decides that he doesn't want to make the horses move more than they should. So he develops this algorithm: he places the 1st P_1 horses in the first stable, the next P_2 in the 2nd stable and so on. Moreover, he doesn't want any of the **K** stables he owns to be empty, and no horse must be left outside. Now you should know that farmer Ion only has black or white horses, which don't really get along too well. If there are **i** black horses and **j** white horses in one stable, then the coefficient of unhappiness of that stable is **i*j**. The total coefficient of unhappiness is the sum of the coefficients of unhappiness of every of the K stables.

Determine a way to place the **N** horses into the **K** stables, so that the total coefficient of unhappiness is minimized.

QUESTION

Input

On the 1st line there are 2 numbers: **N** (1 ≤ N ≤ 500) and **K** (1 ≤ K ≤ N). On the next N lines there are N numbers. The i-th of these lines contains the color of the i-th horse in the sequence: 1 means that the horse is black, 0 means that the horse is white.

Output

You should only output a single number, which is the minimum possible value for the total coefficient of unhappiness.

Sample

Input	Ouput

6 3
1
1
0
1
0
1
	2

TECHNIQUES TO SOLVE

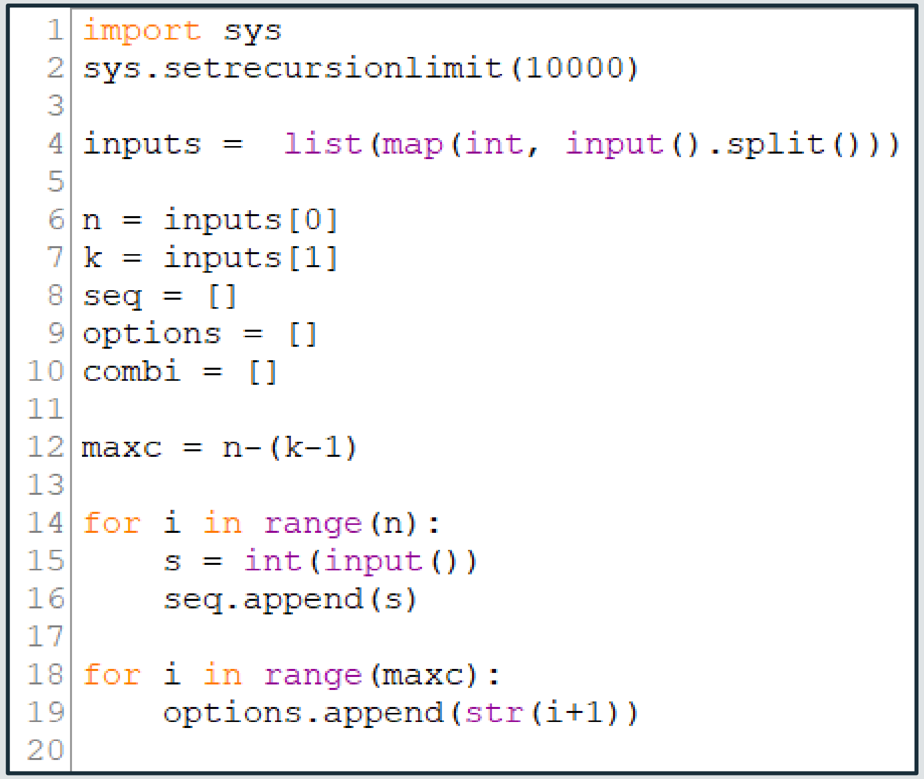
- To solve this problem, I choose to use the **greedy algorithms** and **recursion** as the techniques to solve ‘Bicolored Horse’ problem.
- I have used the **recursion** technique to find the combination of all possible ways to place ‘n’ horse in ‘k’ stable without any left empty stable.
  - For example; n = 6, k = 3 ---->[[2, 1, 3], [1, 1, 4], ……….]
- **The greedy algorithms** to find the minimum possible value for total coefficient of unhappiness of horses of all ‘k’ stable. To use the greedy algorithms, I have divided the one big problem to many small sub-problems and solve one-by-one to follows greedy method to solving problem heuristically of making the locally optimal choice at each sub-problem.
- Since, I do not have the need to revise the previous choice exclude the getting combination of all ways to place ‘n’ horse in ‘k’ stable. It is more efficient in terms of memory as it never look back or revise previous choices and computes the solution by making its choices in a serial forward fashion.

ONE BIG PROBLEM TO MANY SUB-PROBLEMS

- I have taken apart the ‘Bicolored Horse’ problem into many sub-problems to follow the greedy method like this.
  - Sub-problem 1: Generate all combinations of all possible ways to place ‘n’ horse in ‘k’ stable without any left empty stable. (Recursion)
  - Sub-problem 2: Check if each combination in the list are equal to total ‘n’ horse.
  - Sub-problem 3: Placing the inputted black & white horse sequence into the list by slicing the sequence based on combinations from sub-problem2. In order to find the placed horse sequence in ‘k’ stable of all possible ways.
  - Sub-problem 4: Find the total number of black & white horse in each placed sequence of each possible ways in each stable.
  - Sub-problem 5: Find the coefficient of unhappiness of each stable in each possible way.
  - Sub-problem 6: Find the minimum possible value of total coefficient of unhappiness of all stable from all possible ways.

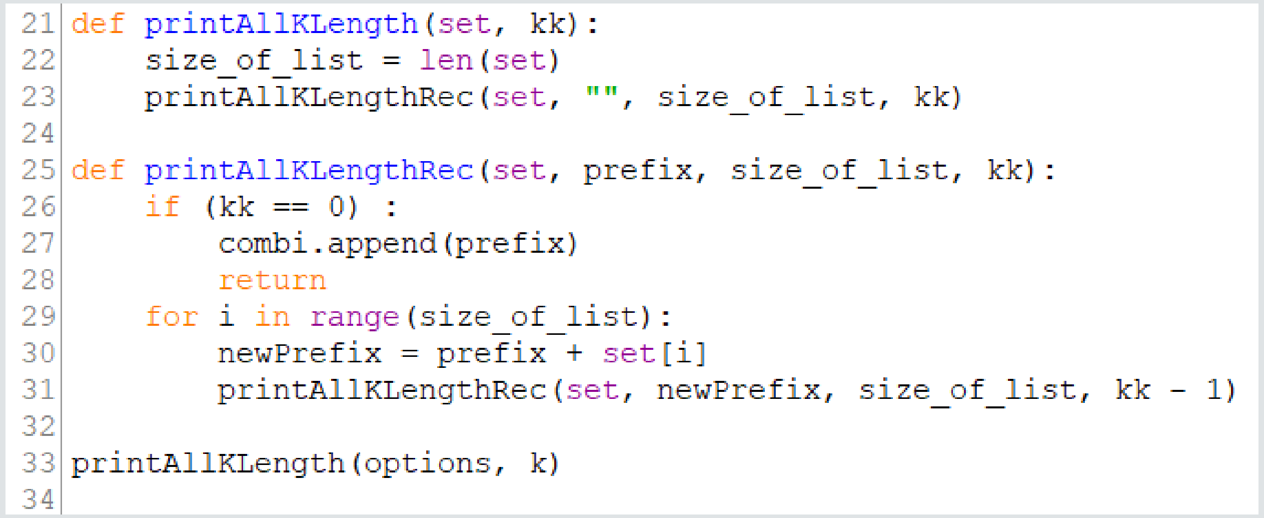
1. GETTING AN INPUT AND PREPARE INPUTTED DATA

- The forth line of code to get the input data into .py file and store it into ’inputs’ list
- ‘n’ = The total number of horses both of black & white horses. (Line 6)
- ‘k’ = The total number of stables that need to place horses. (Line 7)
- ‘seq’ = Keep the inputted sequences of black & white horses as a list. (Line 8)
- ‘options’ = Empty list create to store the options to place horses according to each option. (Line 9)
- ‘combi’ = Empty to store all generated combinations of all possible ways of place horses. (Line 10)
- ‘maxc’ = To find the maximum no of horse should be place because all stables cannot be empty and therefore must at least put one horse in one stable, so in one stable cannot put horse more than n-(k-1). (Line 12)
- Line 14-16: To run loop and append the input sequence into ‘seq’ list.
- Line 18-19: To run loop and append options to place horses based on ‘maxc’ value into ‘options’ list.



```
 1 import sys
 2 sys.setrecursionlimit(10000)
 3
 4 inputs =  list(map(int, input().split()))
 5
 6 n = inputs[0]
 7 k = inputs[1]
 8 seq = []
 9 options = []
10 combi = []
11
12 maxc = n-(k-1)
13
14 for i in range(n):
15     s = int(input())
16     seq.append(s)
17
18 for i in range(maxc):
19     options.append(str(i+1))
20
```

2. SUB-PROBLEM 1



```
21 def printAllKLength(set, kk):
22     size_of_list = len(set)
23     printAllKLengthRec(set, "", size_of_list, kk)
24 
25 def printAllKLengthRec(set, prefix, size_of_list, kk):
26     if (kk == 0) :
27         combi.append(prefix)
28         return
29     for i in range(size_of_list):
30         newPrefix = prefix + set[i]
31         printAllKLengthRec(set, newPrefix, size_of_list, kk - 1)
32 
33 printAllKLength(options, k)
34 
```

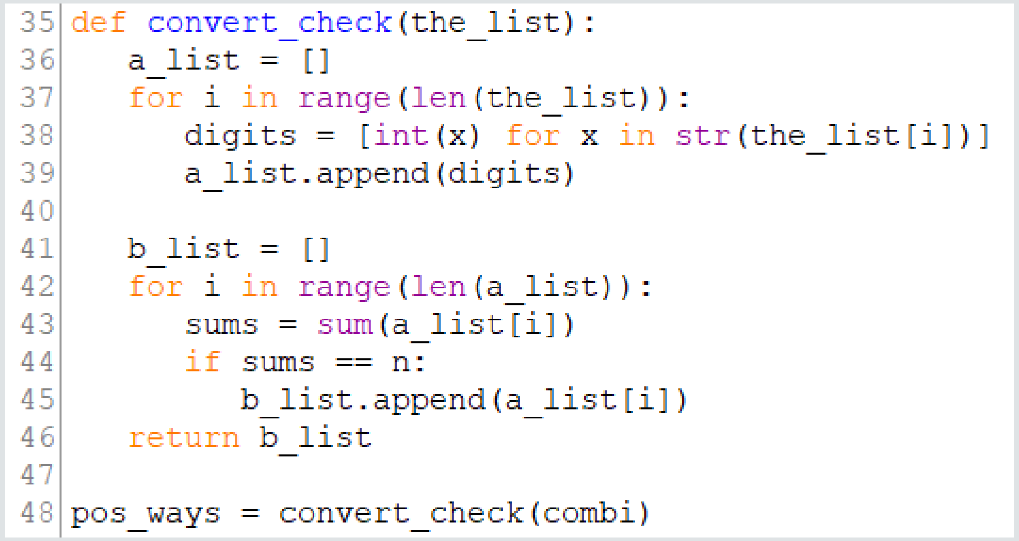
Generate all combinations of all possible ways to place ‘n’ horse in ‘k’ stable without any left empty stable. (Recursion)

‘options’ as a list that contain possible elements to generated combination.

‘k’ as the length of each combination.

- Line 21-23: The ‘printAllKLength()’ function to get the size of a passed set/list and It is mainly a wrapper over recursive function ‘printAllKLengthRec()’. This function pass a set/list ‘set’ and integer value ‘kk’ as parameters.
- Line 25-31: The ‘printAllKLengthRec()’ function to is the main recursive method to print all possible strings of length ‘kk’ as act as base case(‘kk’ == 0) to append ‘prefix’ into ‘combi’ list and return nothing. And the for loop choose the next character of input added from ‘set’ according to index ‘i’ and run the recursion until ‘kk’ = 0 since ‘kk’ is decreased, because a new character added.
- Line 33: Run the ‘printAllKLengthRec()’ function and pass ‘options’ list and ‘k’ value as parameters to perform he function and the element in ‘combi’ list will be string value.

3. SUB-PROBLEM 2



```
35 def convert_check(the_list):
36    a_list = []
37    for i in range(len(the_list)):
38       digits = [int(x) for x in str(the_list[i])]
39       a_list.append(digits)
40
41    b_list = []
42    for i in range(len(a_list)):
43       sums = sum(a_list[i])
44       if sums == n:
45          b_list.append(a_list[i])
46    return b_list
47
48 pos_ways = convert_check(combi)
```

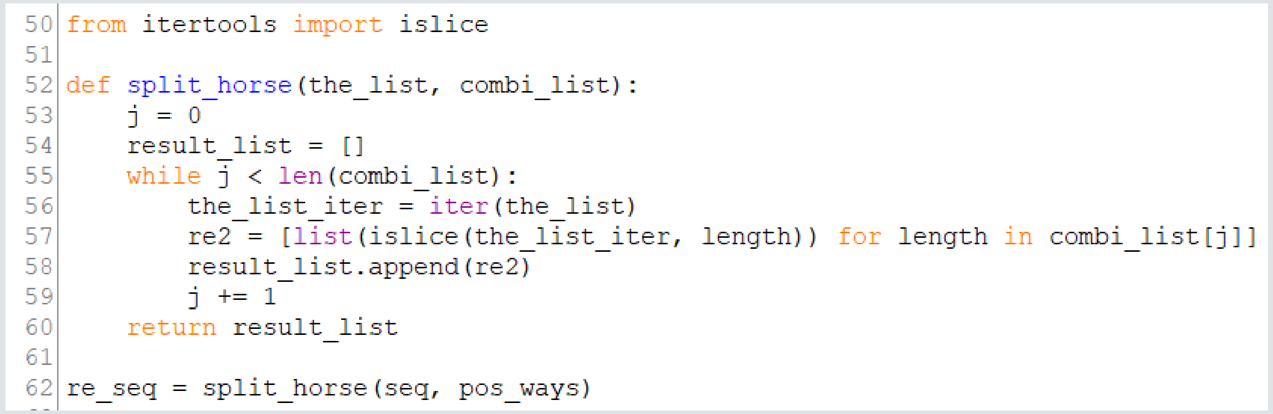
Check if each combination in the list are equal to total ‘n’ horse.

‘combi’ keep all possible combinations based on ‘options’ in list in the form of string element list.

‘b_list’ as temporary list to store the list elements as sub-list in one big list.

- Line 35: Pass list variable in ‘the_list’ parameter of ‘convert_check()’ function.
- Line 36-39: Create a temporary list ‘a_list’ to convert elements in ‘the_list’ from string value to int value in the foe loop and choose for one combination (Line 38) and append the result converted ‘digits’ to ‘a_list’ object.
- Line 41-45 : Create a temporary list ‘b_list’ to compute the sum of each combination(‘a_list[i]’) and check condition if the sum is equal to ‘n’ value (Total number of horse). If so, then append ‘a_list[i]’ combination and the ‘convert_check()’ function
- Line 48: ‘pos_ways’ to store the return list from running ‘convert_check()’ function which pass ‘combi’ as parameter.

4. SUB-PROBLEM 3



```
50 from itertools import islice
51
52 def split_horse(the_list, combi_list):
53     j = 0
54     result_list = []
55     while j < len(combi_list):
56         the_list_iter = iter(the_list)
57         re2 = [list(islice(the_list_iter, length)) for length in combi_list[j]]
58         result_list.append(re2)
59         j += 1
60     return result_list
61
62 re_seq = split_horse(seq, pos_ways)
```

Splitting and Placing the inputted black & white horse sequence ('seq' list) into the list by slicing the sequence by length that get from the elements of sub-list of combination list('pos_ways') from sub-problem2.

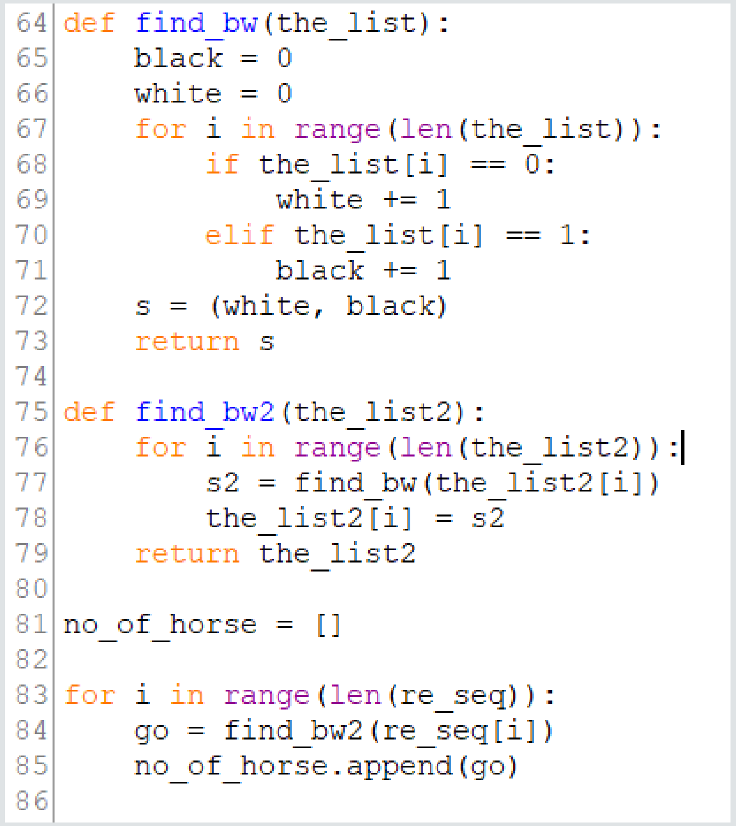
In order to find the placed horse sequence in 'k' stable of all possible ways.

- Line 50: To import function 'islice()' from itertools library into .py files.
- Line 52-60: The 'split_horse()' function to run the while loop to crate an iterator object with 'the_list' list before slice the 'the_list' list with islice() function that start from 'the_list_iter' and stop at the 'length' which get by choosing one element from 'combi_list' list. The sequence will get slice based on combination of one way which will be out in the new list('re2'). Then, the 're2' list will get append into 'result_list' which is a temporary list and is return from function as a list object.
- Line 62: 're_seq' to store the return list from running 'split_horse()' function which pass 'seq' list and 'pos_ways' list as parameter.

5. SUB-PROBLEM 4

- Line 64-73: Run the ‘find_bw()’ to find the total number of black & white horse in one stable of one combination. If elements of sub-list (‘the_list’[i]) == 0, then ‘white’ increase. If elements of sub-list (‘the_list’[i]) == 1, then ‘black’ increase. Then get ‘white’ and ‘black’ in the form of tuple objects and return ‘s’ from ‘find_bw()’ function.
- Line 75-79: Run the ‘find_bw2()’ as wrapper to run ‘find_bw()’ by the number of stable times (meaning if there are 3 stable in one sequence of one possible way then ‘find_bw()’ will run 3 times). The tuple object ‘s2’ as a result of running ‘find_bw()’ will substitute the element of ‘the_list2’ at the index ‘i’ and the ‘find_bw2()’ will return a list object.
- Line 81: ‘no_of_horse’ list object is created to store all number of black & white horses in each stable of all sequences of all possible ways.
- Line 83-85: The for loop to run ‘find_bw2()’ function at all sequences of all possible ways (meaning if there are 10 possible ways, ‘find_bw2()’ will run10 times) and the result ‘go’ will be append to ‘no_of_horse’ list object.

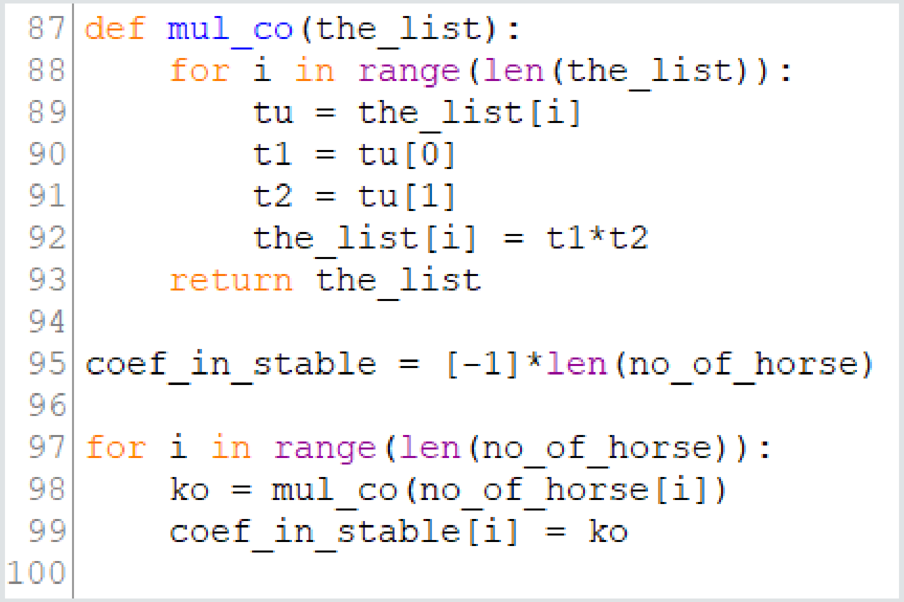
Find the total number of black & white horse in each placed sequence of each possible way in each stable.



```
64 def find_bw(the_list):
65     black = 0
66     white = 0
67     for i in range(len(the_list)):
68         if the_list[i] == 0:
69             white += 1
70         elif the_list[i] == 1:
71             black += 1
72     s = (white, black)
73     return s
74
75 def find_bw2(the_list2):
76     for i in range(len(the_list2)):
77         s2 = find_bw(the_list2[i])
78         the_list2[i] = s2
79     return the_list2
80
81 no_of_horse = []
82
83 for i in range(len(re_seq)):
84     go = find_bw2(re_seq[i])
85     no_of_horse.append(go)
86
```

6. SUB-PROBLEM 5

- Line 87-93: The ‘mul_co()’ function that have a for loop to get the tuple value that store no of white & black horses in one stable of one possible way. Then store in ‘tu’, and then get ‘t1’ and ‘t2’ by state the position of needed element in ‘tu’ list object. Later, compute the coefficient of unhappiness in one stable by ‘t1’ * ‘t2’ and substitute at the same index ‘i’ in ‘the_list’ list and ‘mul_co()’ function return list object.
- Line 95: The ‘coef_in_stable’ is created to store coefficient of unhappiness in one stable of all stable for all possible ways. This list object is created by the length of ‘no_of_length’ list object, since there are no change in the length of list from ‘no_of_horse’ list.
- Line 97-99: The for loop to run ‘mul_co()’ function in length of ‘no_of_horse’ times and store the result ‘ko’ in the same index ‘i’ of ‘no_of_horse’ in ‘coef_in_stable’ list object.



```
 87 def mul_co(the_list):
 88     for i in range(len(the_list)):
 89         tu = the_list[i]
 90         t1 = tu[0]
 91         t2 = tu[1]
 92         the_list[i] = t1*t2
 93     return the_list
 94 
 95 coef_in_stable = [-1]*len(no_of_horse)
 96 
 97 for i in range(len(no_of_horse)):
 98     ko = mul_co(no_of_horse[i])
 99     coef_in_stable[i] = ko
100 
```

Find the coefficient of unhappiness of each stable in each combination.

By compute ‘t1’ * ‘t2’ to find the coefficient of unhappiness of each stable.

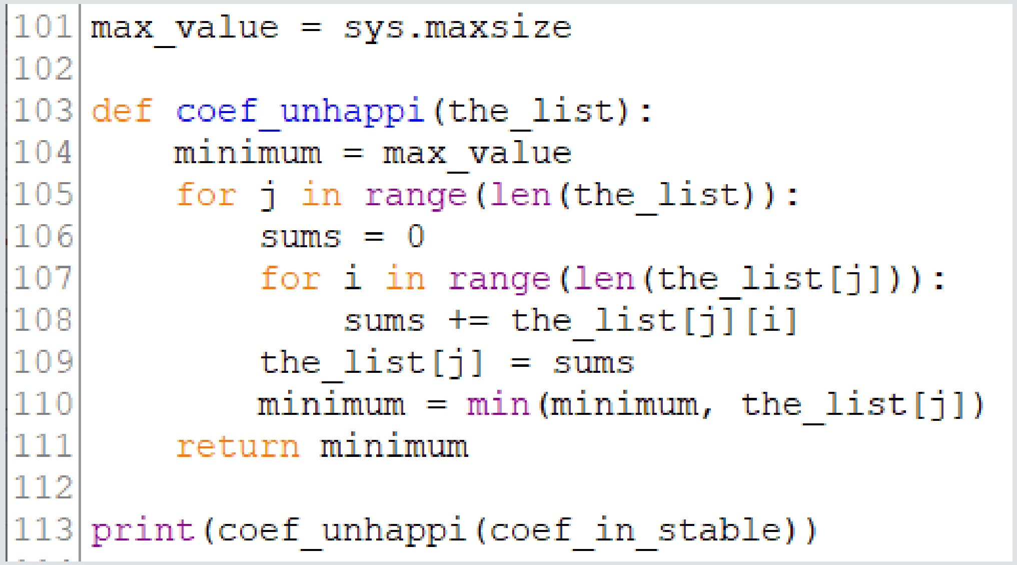
‘tu’ as list object that store number of white & black horse of each stable in one possible ways.

‘t1’ = no. of white horses, ‘t2’= no. of black horses

7. SUB-PROBLEM 6

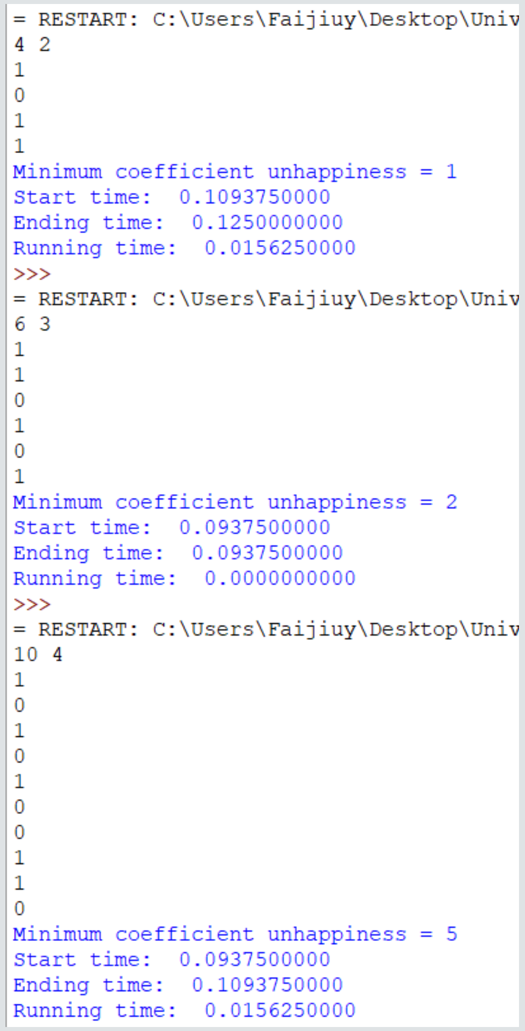
- Line 101: To set ‘max_value’ as the maximum value from sys.maxsize.
- Line 103-111: The ‘coef_unhappi()’ function begin with setting the ‘minimum’ as the maximum value. Run the first for loop for execute code in the loop for all possible ways ‘j’ times and create ‘sums’ to keep the sums of all stables’ coefficient of unhappiness.
    - The second for loop to add the coefficient of unhappiness of all stable in one possible ways to ‘sums’.
    - And substitute the ‘sums’ at the same index ‘j’ in ‘the_list’ object and find minimum value of coefficient of unhappiness of all possible ways to place ‘n’ horses in ‘k’ stable. And ‘coef_unhappi()’ return minimum value.
- Line 113: Printout the minimum possible value for the total coefficient of unhappiness by running ‘coef_unhappi()’ function.

Find the minimum possible value of total coefficient of unhappiness of all stable from all possible ways.

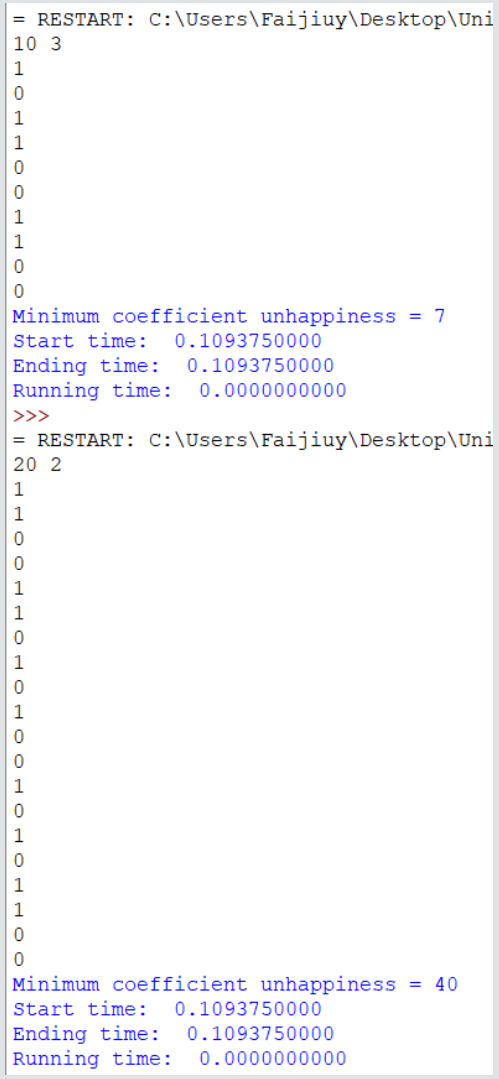


```
101 max_value = sys.maxsize
102
103 def coef_unhappi(the_list):
104     minimum = max_value
105     for j in range(len(the_list)):
106         sums = 0
107         for i in range(len(the_list[j])):
108             sums += the_list[j][i]
109         the_list[j] = sums
110         minimum = min(minimum, the_list[j])
111     return minimum
112
113 print(coef_unhappi(coef_in_stable))
```

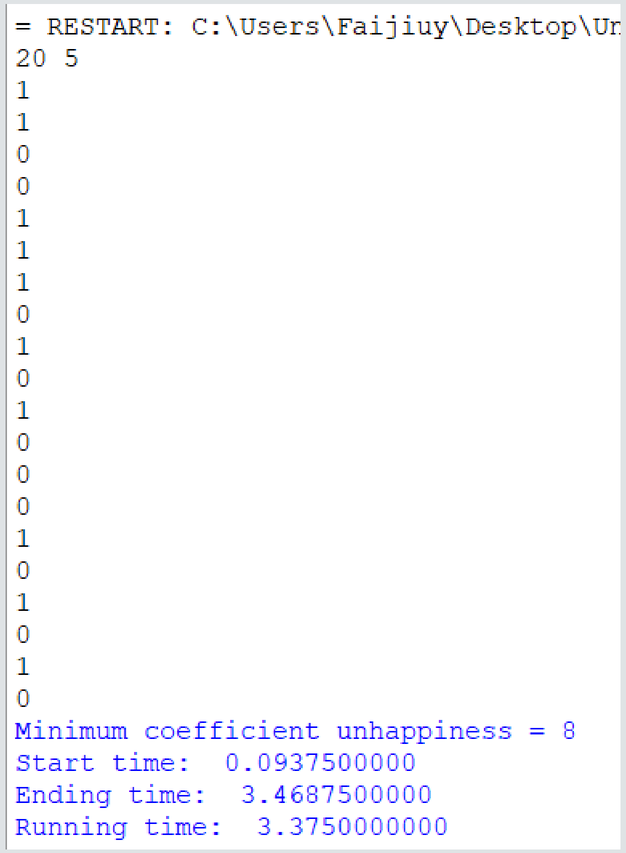
EXECUTION TIME ON MY LAPTOP



```
= RESTART: C:\Users\Faijiuy\Desktop\Univ
4 2
1
0
1
1
Minimum coefficient unhappiness = 1
Start time:  0.1093750000
Ending time:  0.1250000000
Running time:  0.0156250000
>>>
= RESTART: C:\Users\Faijiuy\Desktop\Univ
6 3
1
1
0
1
0
1
Minimum coefficient unhappiness = 2
Start time:  0.0937500000
Ending time:  0.0937500000
Running time:  0.0000000000
>>>
= RESTART: C:\Users\Faijiuy\Desktop\Univ
10 4
1
0
1
0
1
0
0
1
1
0
Minimum coefficient unhappiness = 5
Start time:  0.0937500000
Ending time:  0.1093750000
Running time:  0.0156250000
```

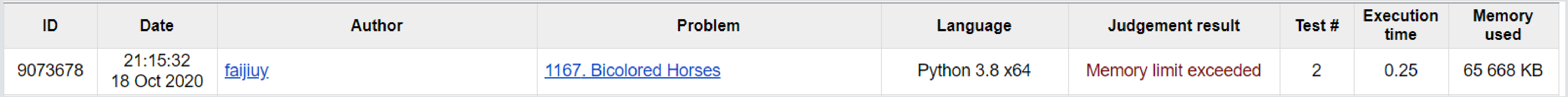


```
= RESTART: C:\Users\Faijiuy\Desktop\Uni
10 3
1
0
1
1
0
0
1
1
0
0
Minimum coefficient unhappiness = 7
Start time:  0.1093750000
Ending time:  0.1093750000
Running time:  0.0000000000
>>>
= RESTART: C:\Users\Faijiuy\Desktop\Uni
20 2
1
1
0
0
1
1
0
1
0
1
0
0
1
0
1
0
1
1
0
0
Minimum coefficient unhappiness = 40
Start time:  0.1093750000
Ending time:  0.1093750000
Running time:  0.0000000000
```



```
= RESTART: C:\Users\Faijiuy\Desktop\Ur
20 5
1
1
0
0
1
1
1
0
1
0
1
0
0
0
1
0
1
0
1
0
Minimum coefficient unhappiness = 8
Start time:  0.0937500000
Ending time:  3.4687500000
Running time:  3.3750000000
```

RESULT FROM TIMUS ONLINE JUDGE



ID	Date	Author	Problem	Language	Judgement result	Test #	Execution time	Memory used
9073678	21:15:32 18 Oct 2020	faijiuy	1167. Bicolored Horses	Python 3.8 x64	Memory limit exceeded	2	0.25	65 668 KB

REFERENCE

- https://www.geeksforgeeks.org/print-all-combinations-of-given-length/

THANK YOU!!!