GIT Department of Computer Engineering CSE 222/505 - Spring 2022
Homework # Report

Sefa Çiçek
1801042657

1. System Requirements

- User has to choose question number on menu to see test results of question.

- jdk and jre are requested from operating system to execute this java program.

2. Diagrams

2.1 Class Diagram

3. Problem Solution Approach

Q1:
I used indexOf and index argument to traverse big String. If indexOf return value was not equal to -1, I incremented occurrence argument. To reach target occurrence, I stored occurrence index + length of small string in another parameter. On recursive part, I added index and recursive call return value to get actual index value of small string.

Q2:
I wrote wrapper function to initialize some arguments. On actual recursion function, I implemented principles of binary search algorithm.

Q3:
I have three condition for comparison of sum of subarray elements with target sum. If sum was greater than target, I changed beginning of index, sum and I called function again. If sum was equal to target, I printed subarray and then I called function for remaining part of the array. In case of the sum is less than target, I incremented begin index and I called function again.

Q5:
I have 2 recursion call. First one is called when one of the block set process is finished (required block num incrementation and start of block change). Another recursive call was used to print all configurations of any block set.
When index value(is used for traversing array) exceeds array size, I cleared some parts(recently added blocks) of the array and I moved the head part of block to the right.

Q6: (---Unfortunately it doesn't print all combinations. I tried a lot but couldn't figure out the problem.)
I have a function to initialize visited array (2D matrix - filled with -1) and tmp array(is used for comparison of results). I have wrapper function for specifying which position I start.
On recursive function, I checked validity of next moves. To check validity, I implemented 2 procedure.

- If next move coordinates are equal to -1, snake can move if other condition met.
- If next move coordinates not exceed matrix size and greater than 0, snake can move.

I used backtracking principle to refill matrix coordinate with -1, if snake move is not possible. Lastly, I have a boolean return value to print all possible results.

4. Test Cases

Q1

- Found in the FIRST INDEX of the array
- Found in the MIDDLE of the array
- Found in the LAST PART of the array
- Target occurrence is less than 0
- NOT FOUND in big string

Q2

- Lower bound < Fist index value
- Upper bound > Last index value
- Lower bound == Fist index value && Upper bound == Last index value
- Lower bound == any value && Upper bound == any value
- NOT FOUND any value - Lower bound < Fist index value && Upper bound < First index value
- FOUND ONLY one value in given range

Q3

- Subarray in first part
- Subarray in middle part
- Subarray in last part
- Subarray with 0
- Subarray without 0
- Subarray with negative values

Q5

- Example in pdf
- Block number < Length of array
- Block number == Length of array
- Block number > Length of array

Q6

- 1 X 1 Matrix
- 2 X 2 Matrix
- 3 X 3 Matrix
- 4 X 4 Matrix

5. Running Command and Results

Q1

```
Found in the FIRST INDEX of the array
-------------------------------------
Small string: bbb
Big string: bbbaaccbbbaaccbbb
Target occurence: 1
Result: 0

Found in the MIDDLE of the array
-------------------------------------
Small string: bbb
Big string: bbbaaccbbbaaccbbb
Target occurence: 2
Result: 7

Found in the LAST PART of the array
-------------------------------------
Small string: bbb
Big string: bbbaaccbbbaaccbbb
Target occurence: 3
Result: 14

Target occurrence is less than 0
-------------------------------------
Small string: bbb
Big string: bbbaaccbbbaaccbbb
Target occurence: -5
Result: -1

NOT FOUND in big string
-------------------------------------
Small string: zzz
Big string: bbbaaccbbbaaccbbb
Target occurence: 1
Result: -1
```

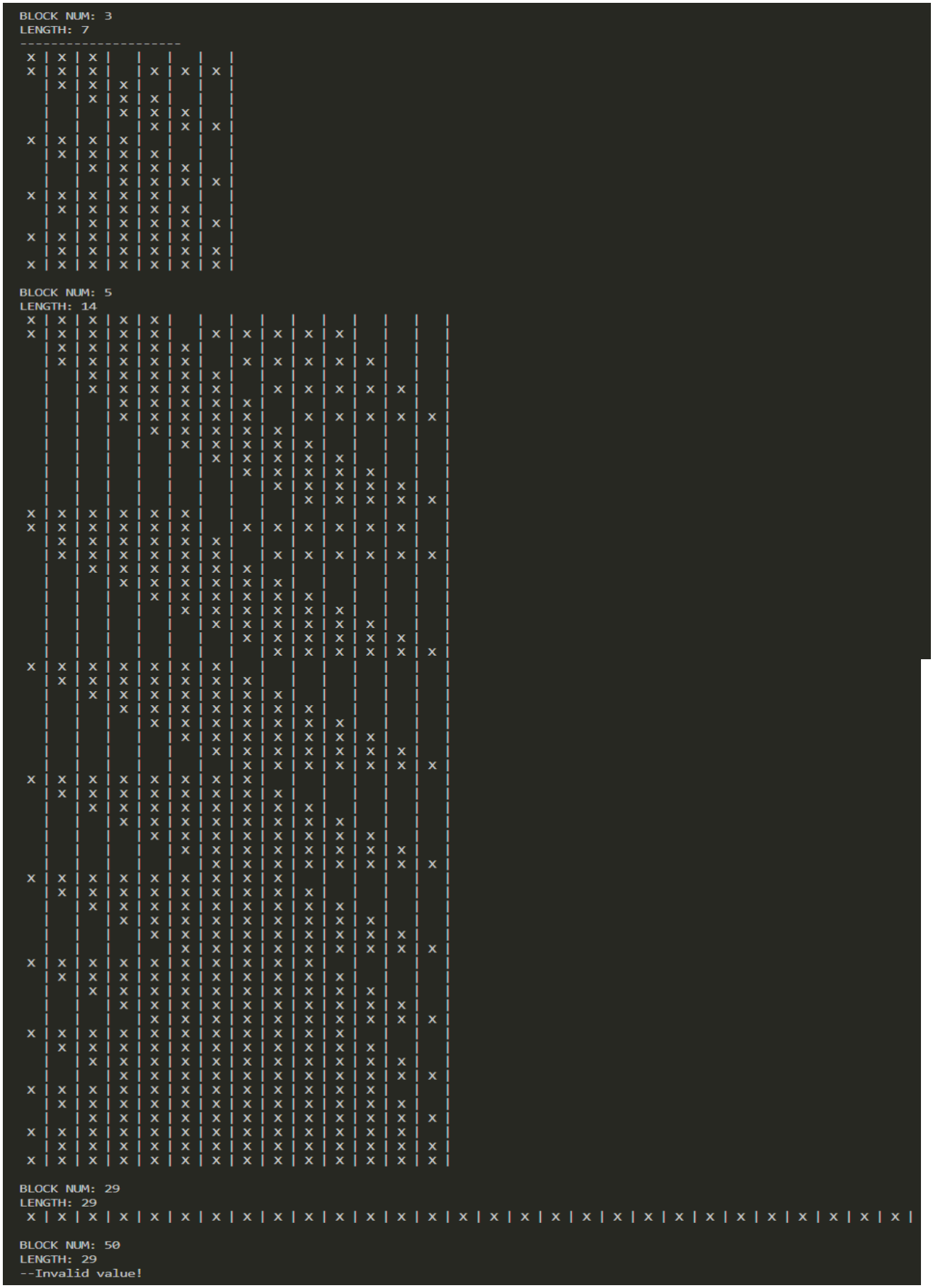
Q2

```
Lower bound < Fist index value
-----------------------------------
Sorted integer array: [ 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 ]
Lower bound: 5
Upper bound: 65
Result: 6

Upper bound > Last index value
-----------------------------------
Sorted integer array: [ 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 ]
Lower bound: 70
Upper bound: 200
Result: 4

Lower bound == Fist index value && Upper bound == Last index value
-----------------------------------
Sorted integer array: [ 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 ]
Lower bound: 10
Upper bound: 100
Result: 10

Lower bound == any value && Upper bound == any value
-----------------------------------
Sorted integer array: [ 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 ]
Lower bound: 20
Upper bound: 80
Result: 7

NOT FOUND any value - Lower bound < Fist index value && Upper bound < First index value
-----------------------------------
Sorted integer array: [ 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 ]
Lower bound: 0
Upper bound: 5
Result: 0

FOUND ONLY one value in given range
-----------------------------------
Sorted integer array: [ 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 ]
Lower bound: 45
Upper bound: 55
Result: 1
```

Q3

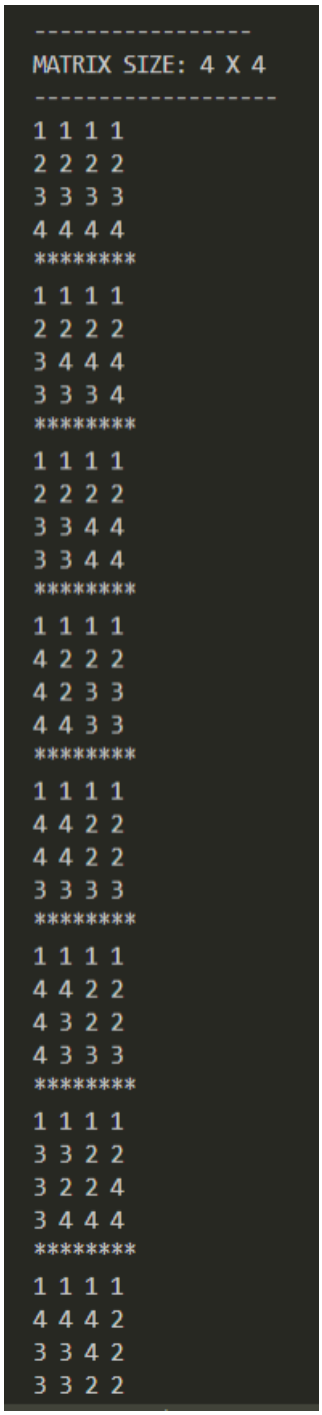
```
All test conditions are provided in these examples:
- Subarray in first part
- Subarray in middle part
- Subarray in last part
- Subarray with 0
- Subarray without 0
- Subarray with negative values
-----------------------------------
Unsorted integer array: [ -8, 1, 4, 0, 1, 3, 2, 2, -2, 7, 0, 1, 5 ]
Target sum: 5
Subarray/s:
-----------------
-8 1 4 0 1 3 2 2
1 4
1 4 0
4 0 1
3 2
-2 7
-2 7 0
5

Unsorted integer array: [ 0, 3, 0, -1, 4, 0 ]
Target sum: -8
Subarray/s:
-----------------

Unsorted integer array: [ 0, 3, 0, -1, 4, 0 ]
Target sum: 3
Subarray/s:
-----------------
0 3
0 3 0
3
3 0
0 -1 4 0
-1 4 0
```

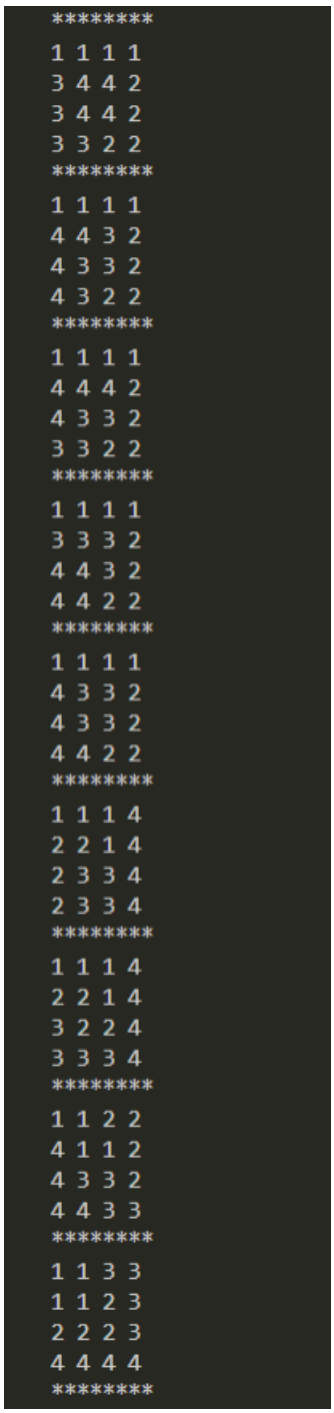
Q5

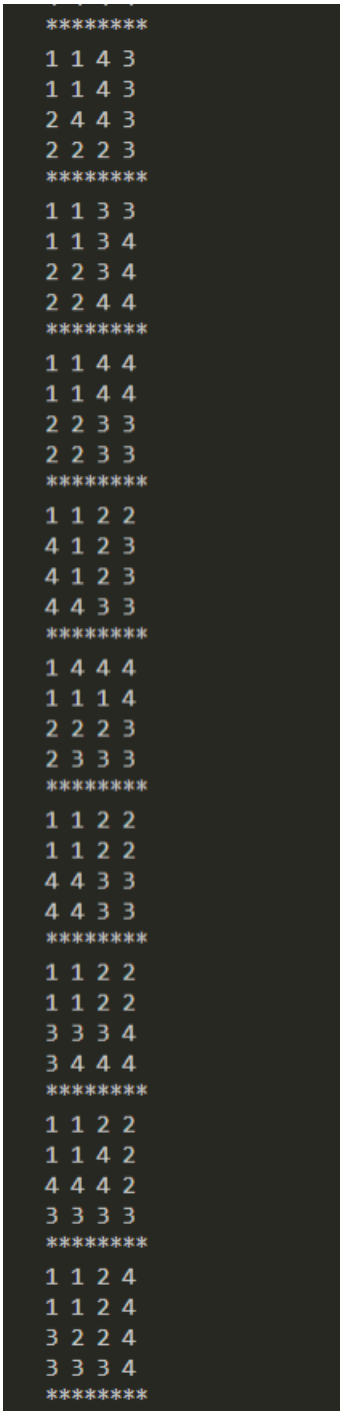

BLOCK NUM: 3
LENGTH: 7
BLOCK NUM: 5
LENGTH: 14
BLOCK NUM: 29
LENGTH: 29
BLOCK NUM: 50
LENGTH: 29
--Invalid value!

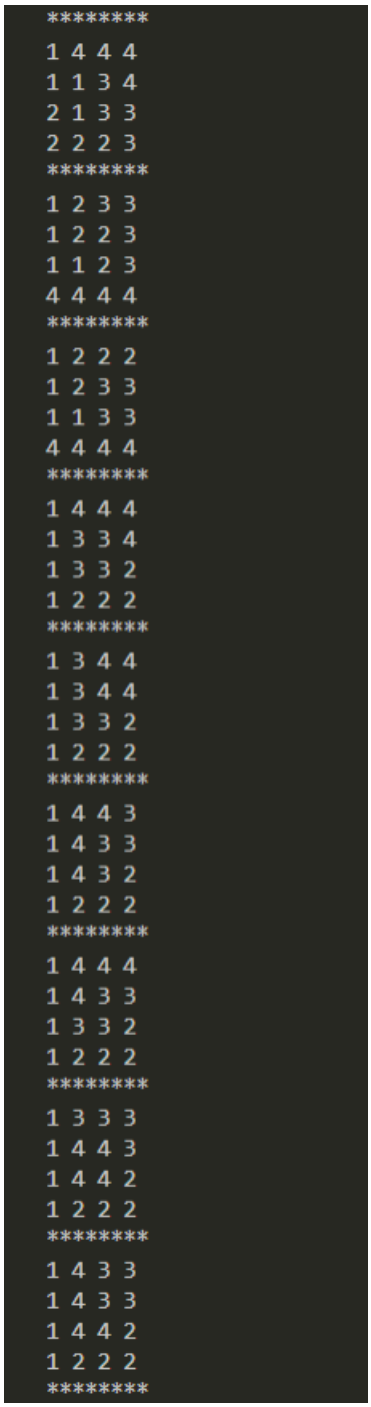
Q6

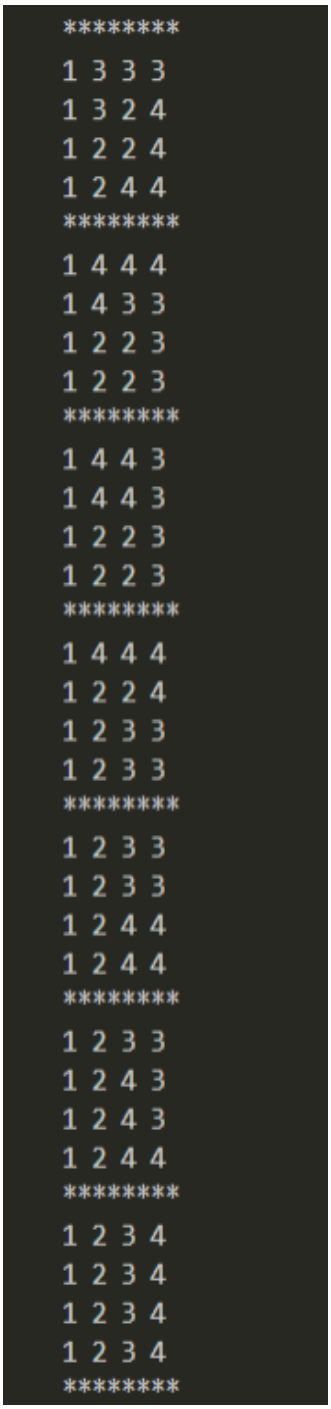
```
------------------
MATRIX SIZE: 1 X 1
------------------
1
********

-----------------
MATRIX SIZE: 2 X 2
------------------
1 1
2 2
********
1 2
1 2
********

-----------------
MATRIX SIZE: 3 X 3
------------------
1 1 1
2 2 2
3 3 3
********
1 1 1
3 3 2
3 2 2
********
1 1 3
2 1 3
2 2 3
********
1 2 2
1 1 2
3 3 3
********
1 3 3
1 3 2
1 2 2
********
1 2 3
1 2 3
1 2 3
********
```


MATRIX SIZE: 4 X 4
1 1 1 1
2 2 2 2
3 3 3 3
4 4 4 4
****
1 1 1 1
2 2 2 2
3 4 4 4
3 3 3 4
****
1 1 1 1
2 2 2 2
3 3 4 4
3 3 4 4
****
1 1 1 1
4 2 2 2
4 2 3 3
4 4 3 3
****
1 1 1 1
4 4 2 2
4 4 2 2
3 3 3 3
****
1 1 1 1
4 4 2 2
4 3 2 2
4 3 3 3
****
1 1 1 1
3 3 2 2
3 2 2 4
3 4 4 4
****
1 1 1 1
4 4 4 2
3 3 4 2
3 3 2 2


****
1 1 1 1
3 4 4 2
3 4 4 2
3 3 2 2
****
1 1 1 1
4 4 3 2
4 3 3 2
4 3 2 2
****
1 1 1 1
4 4 4 2
4 3 3 2
3 3 2 2
****
1 1 1 1
3 3 3 2
4 4 3 2
4 4 2 2
****
1 1 1 1
4 3 3 2
4 3 3 2
4 4 2 2
****
1 1 1 4
2 2 1 4
2 3 3 4
2 3 3 4
****
1 1 1 4
2 2 1 4
3 2 2 4
3 3 3 4
****
1 1 2 2
4 1 1 2
4 3 3 2
4 4 3 3
****
1 1 3 3
1 1 2 3
2 2 2 3
4 4 4 4
****


****
1 1 4 3
1 1 4 3
2 4 4 3
2 2 2 3
****
1 1 3 3
1 1 3 4
2 2 3 4
2 2 4 4
****
1 1 4 4
1 1 4 4
2 2 3 3
2 2 3 3
****
1 1 2 2
4 1 2 3
4 1 2 3
4 4 3 3
****
1 4 4 4
1 1 1 4
2 2 2 3
2 3 3 3
****
1 1 2 2
1 1 2 2
4 4 3 3
4 4 3 3
****
1 1 2 2
1 1 2 2
3 3 3 4
3 4 4 4
****
1 1 2 2
1 1 4 2
4 4 4 2
3 3 3 3
****
1 1 2 4
1 1 2 4
3 2 2 4
3 3 3 4
****


****
1 4 4 4
1 1 3 4
2 1 3 3
2 2 2 3
****
1 2 3 3
1 2 2 3
1 1 2 3
4 4 4 4
****
1 2 2 2
1 2 3 3
1 1 3 3
4 4 4 4
****
1 4 4 4
1 3 3 4
1 3 3 2
1 2 2 2
****
1 3 4 4
1 3 4 4
1 3 3 2
1 2 2 2
****
1 4 4 3
1 4 3 3
1 4 3 2
1 2 2 2
****
1 4 4 4
1 4 3 3
1 3 3 2
1 2 2 2
****
1 3 3 3
1 4 4 3
1 4 4 2
1 2 2 2
****
1 4 3 3
1 4 3 3
1 4 4 2
1 2 2 2
****


****
1 3 3 3
1 3 2 4
1 2 2 4
1 2 4 4
****
1 4 4 4
1 4 3 3
1 2 2 3
1 2 2 3
****
1 4 4 3
1 4 4 3
1 2 2 3
1 2 2 3
****
1 4 4 4
1 2 2 4
1 2 3 3
1 2 3 3
****
1 2 3 3
1 2 3 3
1 2 4 4
1 2 4 4
****
1 2 3 3
1 2 4 3
1 2 4 3
1 2 4 4
****
1 2 3 4
1 2 3 4
1 2 3 4
1 2 3 4
****