Dear reader, welcome to the next problem in the Recursion & Backtracking section named ‘[***Combinations - 1***](https://www.pepcoding.com/resources/data-structures-and-algorithms-in-java-levelup/recursion-and-backtracking/combinations-i-official/ojquestion)’.

Please, I request you to solve the previous problem ‘[Permutations - 1](https://www.pepcoding.com/resources/data-structures-and-algorithms-in-java-levelup/recursion-and-backtracking/permutation-i-official/ojquestion)’ before jumping on to this problem. Infact, I recommend you to watch these sets of problems in given order and try to complete the entire set together.

***Problem Statement:***

* You are given a number of boxes (nboxes) and a number of identical items (ritems).
* You are required to place the items in those boxes and print all such configurations possible.
* Items are identical and all of them are named 'i'.
* Note 1: Number of boxes is greater than number of items, hence some of the boxes may remain empty.
* Note 2: Check out the output format and write the recursive code as it is intended without changing signature. The judge can't force you but intends you to teach a concept.

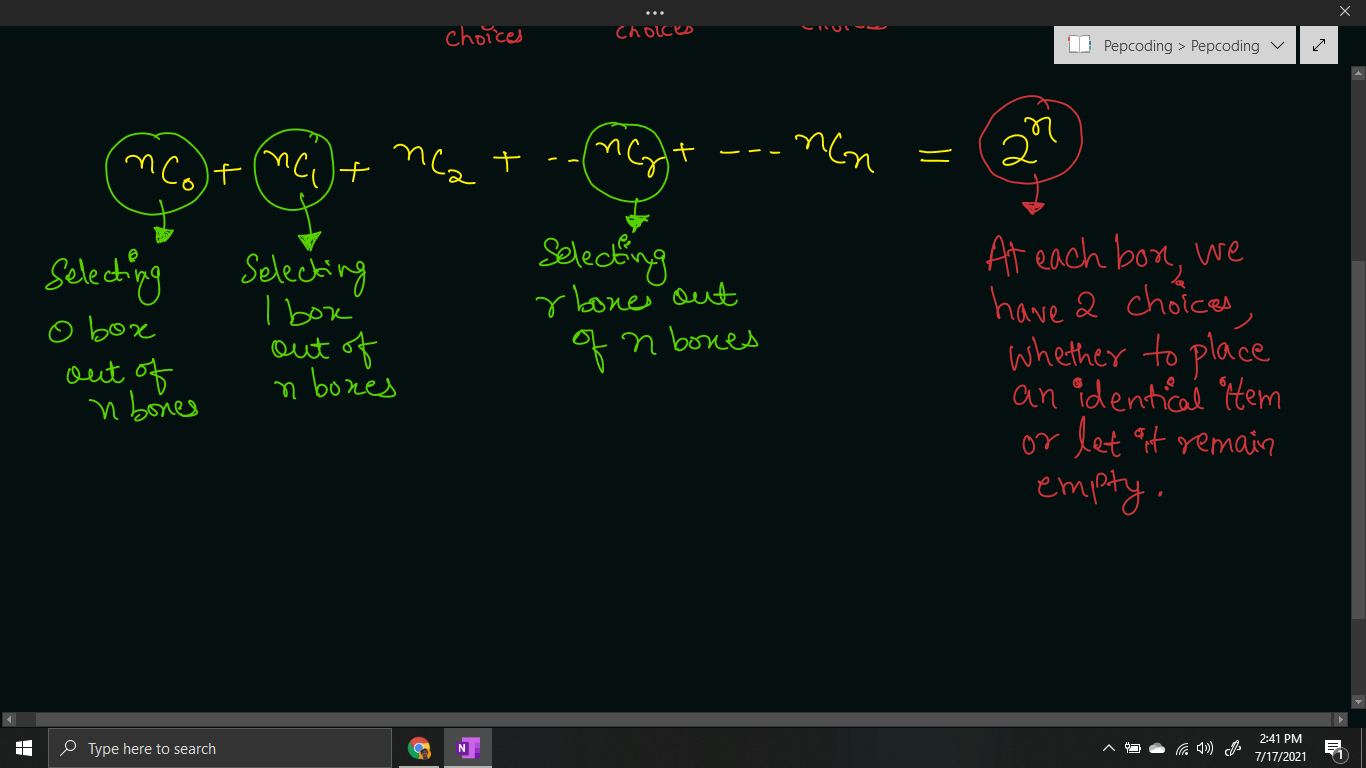
***Example:***

*Input*: Number of boxes (nboxes) = 5, number of identical items (ritems) = 3

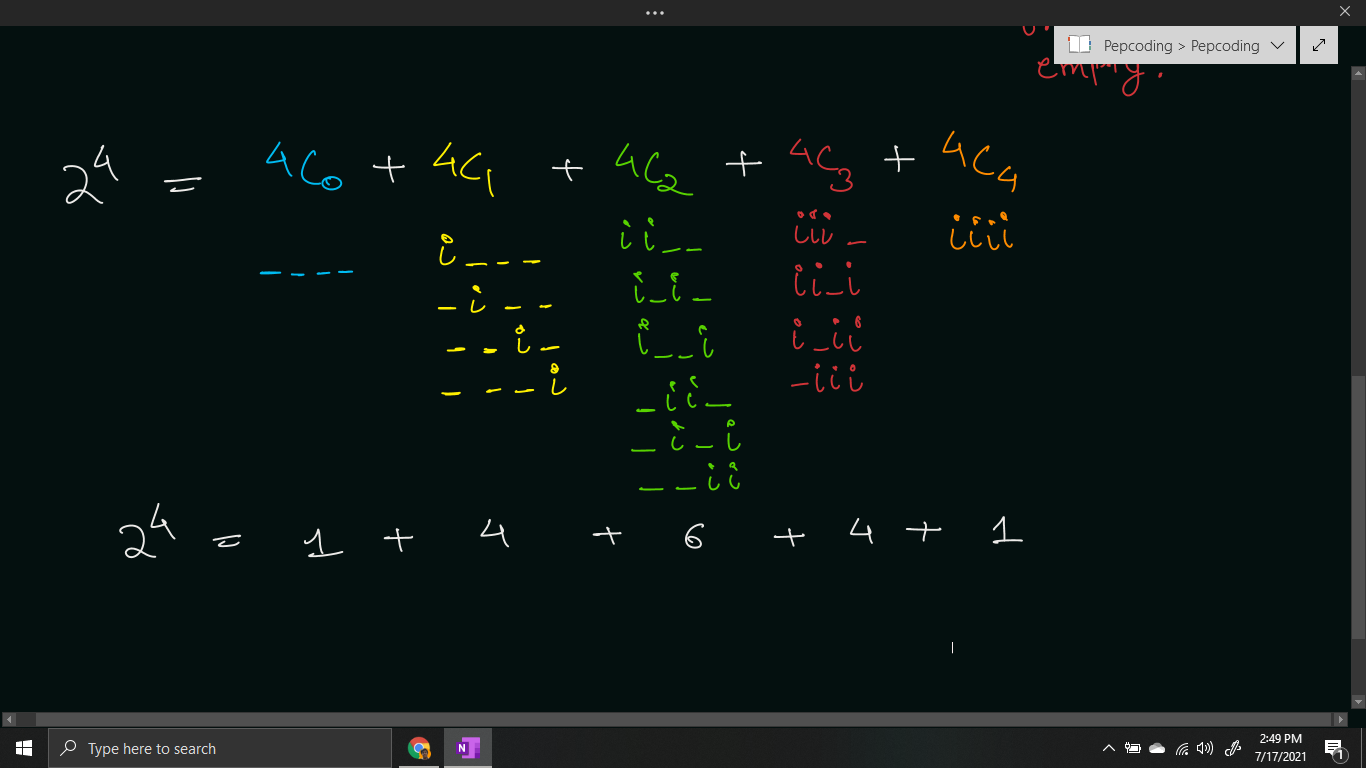
*Output*: [ iii-- , ii-i- , ii--i , i-ii- , i-i-i , i--ii , -iii- , -ii-i , -i-ii , --iii ]

***Solution***

Do you remember the following formula which you probably learned in high school mathematics?



Let us find all the combinations possible for n = 4 boxes.



We will print all those combinations where the number of identical items is equal to the total number (r) required, i.e. only print the combinations of ***nCr***.

Now, let us define the expectation and faith in our recursive functions, and then derive a recursive relation by meeting expectation with faith.

**Expectation**: We expect that the recursive function will give us all the combinations of r identical items. At each level, we are deciding for a given box whether an item should be placed in it or not. Hence, at first level, we are deciding for the 0th box.

combinations(int cb, int tb, int ssf, int ts, String asf)

Here, cb is the current box whose decision (whether to place an item or not) is to be taken, tb is the total number of boxes, ssf is the number of items placed so far, ts is the total number of identical items, and asf is the answer so far.

**Faith**: We will keep faith on the recursive function that will help us make all the choices for the remaining n-1 boxes.

combinations(nextBox, tb, ssf, ts, asf)

**Meeting Expectation with Faith:**

For the current box, we have two options, whether we should place an item in it or let it remain empty.

* If we place an item in the current box, then items selected so far will increase by 1, and we will append “i” (item in current box) to the answer string.
* If we do not place an item in the current box, then items selected so far will remain the same, and we will append “-” (empty box) to the answer string.

combinations(cb + 1, tb, ssf + 1, ts, asf + "i");

combinations(cb + 1, tb, ssf + 0, ts, asf + "-");

**Base Case**

Since we are picking all the boxes level by level, we should stop when decisions for all the boxes have been made. Hence, we should stop when the current box becomes greater than the total boxes (cb > tb).

Now, there is a catch. As discussed in the mathematical formula, we will generate all the combinations of 0 items, 1 items, 2 items, and so on. But we only require those combinations which have ts (total number of identical) items in the answer string. Hence, we will print the combination if ssf = ts.

if(cb > tb){

if(ssf == ts){

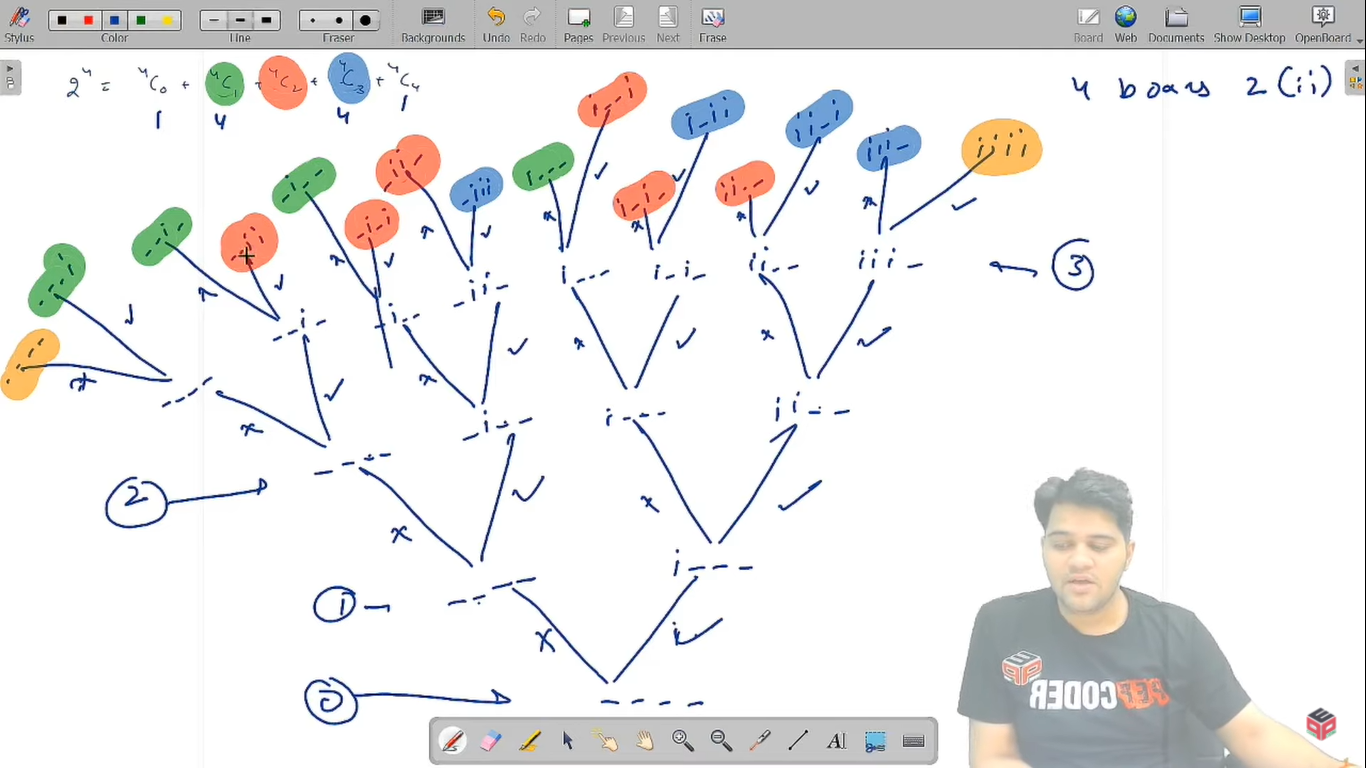
System.out.println(asf);

}

return;

}

In the recursion tree, the ***LEVELS are equivalent to the BOXES*** and the choices are to place an ITEM in the current box or not.



**Java Code**

import java.io.\*;

import java.util.\*;

public class Main {

public static void combinations(int cb, int tb, int ssf, int ts, String asf){

if(cb > tb){

if(ssf == ts){

System.out.println(asf);

}

return;

}

combinations(cb + 1, tb, ssf + 1, ts, asf + "i");

combinations(cb + 1, tb, ssf + 0, ts, asf + "-");

}

public static void main(String[] args) throws Exception {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int nboxes = Integer.parseInt(br.readLine());

int ritems = Integer.parseInt(br.readLine());

combinations(1, nboxes, 0, ritems, "");

}

}

Java Code is written and explained by our team in the [solution video](https://www.youtube.com/watch?v=wOaxJAtJ2Mo&list=TLGGVLXuUnA2Q18xNzA3MjAyMQ). Please refer to it for a better understanding of the algorithm and the implementation.

* What is the ***time complexity*** of the above code?

It is clearly visible from the recursion tree, that it can have a maximum depth of n levels (where n = number of boxes), and for each box, we have two options, whether to place an item or not, hence the total time complexity will be O(2 + 2 + 2 + …. n times) = **O(2n)**.

* What is the ***space complexity*** of the above code?

We are using recursion to print all the combinations. Since, the maximum depth is O(n), hence the space complexity will be **O(n)** where n = number of boxes. However, we are not using any auxiliary data structure.

Hope that you liked the article on **‘*Combinations - 1’***.

Subscribe to Pepcoding’s youtube channel for more such amazing video content on Data Structures & Algorithms. You can suggest any improvements to the article on our telegram channel, or on the youtube channel’s comment section.

Article Contributed by:

[Archit Aggarwal](https://www.linkedin.com/in/archit-aggarwal-6a7716189/)