Recursion-III

Problem Statement - Return subset of an array

Approach:

The idea is simple, that if there are n number of elements inside an array, there are two choices for every element. Either include that element in the subset or do not include it.

Using the above idea form a recursive solution to the problem.

Algorithm:

- This method takes an array of integers input and the startIndex from which to start generating subsets.
- The base case is when startIndex equals the length of the input array. In this case, it returns a 2D array containing a single empty subset (an array of length 0).
- It makes a recursive call to subsetsHelper with an incremented startIndex to obtain subsets without including the current element.
- The method then constructs a new 2D array (output) to store subsets.
- It copies the subsets obtained from the recursive call (smallerOutput) to the new array.
- It adds subsets that include the current element (input[startIndex]) by iterating through the subsets obtained from the recursive call and adding the current element at the beginning of each subset.

This is a wrapper method that initializes the process by calling subsetsHelper with the input array and a starting index of 0.

```
public class solution {
     public static int[][] subsetsHelper(int[] input, int startIndex){
          if(startIndex == input.length){
               int[][] output = new int[1][0];
               return output;
          }
          int[][] smallerOutput = subsetsHelper(input, startIndex + 1);
          int[][] output = new int[2*smallerOutput.length][];

          int k = 0;
          for(int i = 0; i < smallerOutput.length; i++){
               output[k] = new int[smallerOutput[i].length];
```

```
                for(int j = 0; j<smallerOutput[i].length; j++){
                    output[k][j] = smallerOutput[i][j];
                }
                k++;
            }
            for(int i = 0; i < smallerOutput.length; i++){
                output[k] = new int[smallerOutput[i].length+1];
                output[k][0] = input[startIndex];
                for(int j = 1; j <= smallerOutput[i].length; j++){
                    output[k][j] = smallerOutput[i][j-1];
                }
                k++;
            }
            return output;
        }

        public static int[][] subsets(int input[]) {
            return subsetsHelper(input, 0);
        }
}
```

Problem Statement - Return Subsequences

Given a string (let's say of length n), return all the subsequences of the given string. Subsequences contain all the strings of length varying from 0 to n. But the order of characters should remain the same as in the input string. Note:

Approach:

- This method takes a string str as input.
- The base case is when the length of the string becomes 0. In this case, it returns an array containing an empty string, representing the empty subsequence.
- It makes a recursive call to subsequence with the substring obtained by excluding the first character of the input string.
- It constructs a new array (output) to store subsequences. The size of the array is twice the length of the array obtained from the recursive call, as each subsequence can either include or exclude the first character of the input string.
- It copies the subsequences obtained from the recursive call (smallerOutput) to the new array.

- It adds subsequences that include the first character of the input string by iterating through the subsequences obtained from the recursive call and adding the first character at the beginning of each subsequence.
- The main work is done by the recursive subsequence method, which explores all possible combinations of including or excluding characters from the input string to generate all possible subsequences.

```
public class solution {

	public static String[] subsequence(String str){
		if(str.length() == 0){
			String output[] = {""};
			return output;
		}

		String[] smallerOutput = subsequence(str.substring(1));
		String[] output = new String[2*smallerOutput.length];

		for(int i = 0; i < smallerOutput.length; i++){
			output[i] = smallerOutput[i];
		}

		for(int i = 0; i < smallerOutput.length; i++){

			output[i + smallerOutput.length] = str.charAt(0) + smallerOutput[i];
		}

		return output;

	}
}
```

Problem Statement: Print All Subsequences

Given a string (let's say of length n), print all the subsequences of the given string. Subsequences contain all the strings of length varying from 0 to n. However, the order of characters should remain the same as in the input string.

Approach:

- The base case is when the input string s becomes empty. In this case, it prints the current subsequence ans and returns from the recursive call.
- In each recursive call, it considers two possibilities for the next character of the subsequence:
  - Include the first character of the input string in the current subsequence (ans + s.charAt(0)).
  - Exclude the first character of the input string in the current subsequence (ans).
- The method calls itself recursively with the updated input string (excluding the first character) and the updated subsequence.
- printSubsequences Method:
- This is a wrapper method that initializes the process by calling printSubsequencesHelper with the input string and an empty initial subsequence.

```
public class solution {

	public static void printSubsequencesHelper(String s, String ans) {
		if(s.length() == 0) {
			System.out.println(ans);
			return;
		}
		printSubsequencesHelper(s.substring(1), ans + s.charAt(0));
		printSubsequencesHelper(s.substring(1), ans);
	}

	public static void printSubsequences(String input) {
		printSubsequencesHelper(input, "");
	}
}
```

Problem Statement: Print Keypad Combinations

Given an integer n, using the phone keypad find out and print all the possible strings that can be made using digits of input n

The phone keypad looks like this:

Approach: It can be observed that each digit can represent 3 to 4 different alphabets (apart from 0 and 1). So the idea is to form a recursive function. You can follow the given steps:

- The code contains two overloaded versions of the printKeypad method.
- The first version is a helper method named printKeypad(int input, String output).
    i. The base case is when input becomes 0. In this case, it prints the current combination output and returns.
    ii. It calculates the set of characters corresponding to the last digit of input using the singleDigit method.
    iii. It recursively calls itself for the remaining digits in input with each character from the set appended to the current output.
- The second version is a wrapper method named printKeypad(int input).
    i. It initializes the process by calling the helper method with the input number and an empty initial output string.
- The main work is done by the recursive printKeypad method, which explores all possible combinations of characters corresponding to the digits in the input number.
- The recursion continues until the base case is reached when input becomes 0.

```
public class solution {

	public static String[] singleDigit(int n){
		if(n<=1 || n>10){
			System.exit(0);
```

```
        }
        if(n==2){
                String output[] ={"a","b","c"};
                return output;
        }
        else if(n==3){
                String output[] ={"d","e","f"};
                return output;
        }
        else if(n==4){
                String output[] ={"g","h","i"};
                return output;
        }
        else if(n==5){
                String output[] ={"j","k","l"};
                return output;
        }
        else if(n==6){
                String output[] ={"m","n","o"};
                return output;
        }
        else if(n==7){
                String output[] ={"p","q","r","s"};
                return output;
        }
        else if(n==8){
                String output[] ={"t","u","v"};
                return output;
        }
        else{
                String output[] ={"w","x","y","z"};
                return output;
        }
    }

    public static void printKeypad(int input, String output){
        if(input == 0){
                System.out.println(output);
                return;
        }

        String singleDigitOutput[] = singleDigit(input % 10);
        for(int i = 0; i < singleDigitOutput.length; i++){
                printKeypad(input / 10, singleDigitOutput[i] + output);
        }
    }
```

```
	public static void printKeypad(int input){
		printKeypad(input, "");
	}
}
```