Lesson:

Problems based on Recursion - 9

Pre-Requisites

- Recursion basics
- Working rules of recursive functions

List of Concepts Involved

- Given a string containing digits from 2-9 inclusive, return all possible letter combinations that the number could represent.
- Frog problem

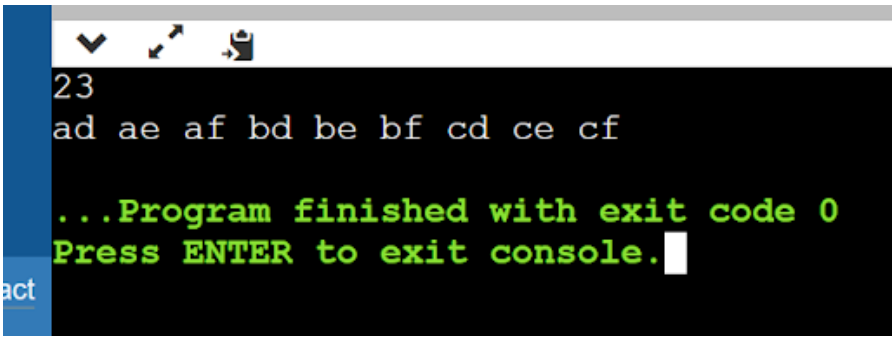
Problem 1 Given a string containing digits from 2-9 inclusive, return all possible letter combinations that the number could represent. Return the answer in any order.
A mapping of digits to letters (just like on the telephone buttons) is given below. Note that 1 does not map to any letters.

Example 1:
Input: digits = "23"
Output: ["ad","ae","af","bd","be","bf","cd","ce","cf"]

Example 2:
Input: digits = ""
Output: []

Example 3:
Input: digits = "2"
Output: ["a","b","c"]

Code link : https://pastebin.com/Qsse208V



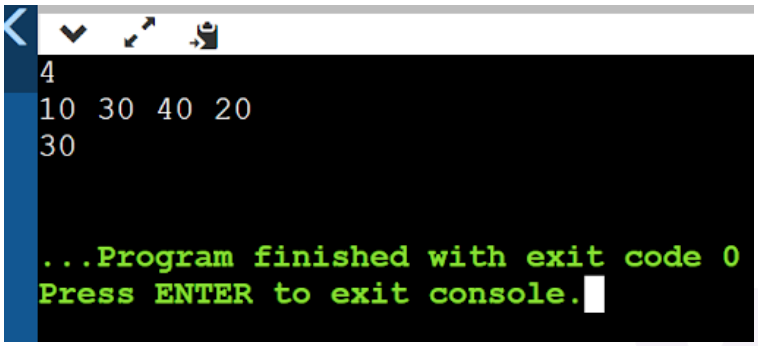
Explanation:

- The function “combination” consists of 5 parameters, the original string, the temporary Arraylist v which holds the possible letters associated with one number.
- Eg. for the 2nd index we have a string “abc”, for index 0 and 1 we have no strings so stored empty strings on their behalf. This is done so that whichever is the digit the same index as of digit holds the corresponding string.
- Other parameters are the current index, output string that will hold each of the possible sequences one by one and the answer arraylist that will hold all the resulting strings.
- The base case could be if the current index idx has crossed the length of the string, then we need to transfer the sequence so formed from the output string to the answer arraylist and since there are no further steps we can return from here.
- If the idx is within the string length then we need to extract digits in order to have their associated string or more formally the letters.
- To use that digit as an index we need to convert it from char datatype to integer therefore we have subtracted ‘0’ from the digit. By doing this we are basically subtracting the ascii values hence we get the integer version of that particular digit.
- Now we know that if the digit is 0 or 1 then it won't be making any contribution in the final answer so in that case we made a recursive call for the next index but have not appended anything to the output string.
- Other than these digits every digit has some letters corresponding to the digit. So we iterated over every possible character by applying a loop.
- Now for every iteration our task is to pick that corresponding character, add it in the output string and pass on the remaining task to recursive calls.
- So we did the same, we picked a particular character from the following string available(example: ‘a’ from “abc”) and called for the further index.
- So we made a further call with parameters as s , v , ans , idx + 1 , res + v.get(digit).charAt(j);
- This output string will keep on growing and once the length of the string is reached it will empty itself in the answer arraylist.
- Since the value if not passed by reference the string will clear itself after every returning call.

Problem 2 There are N stones, numbered 1,2,...,N. For each i (1≤i≤N), the height of Stone i is hi. There is a frog who is initially on Stone 1. He will repeat the following action some number of times to reach Stone N:
If the frog is currently on Stone i, jump to Stone i+1 or Stone i+2. Here, a cost of |hi - hj| is incurred, where j is the stone to land on.
Find the minimum possible total cost incurred before the frog reaches Stone N.
Input n= 4
arr[]=10 30 40 20
Output 30

Explanation:
If we follow the path 1 → 2 → 4, the total cost incurred would be |10-30| + |30-20| =30.

Code Link https://pastebin.com/6KLL60ui



Explanation:

- In the "best" function we have passed 3 parameters, one is the array 'hts'(array of heights), 2nd is the length of the array and the third one is the current index.
- The base case is that if the current index is greater than the length of the array then we have already reached our destination so we can return 0 from here since there are no further steps to make.
- Now we have two functionalities: either we can go one step forward or two steps forward in one go.
- If we try the first option that is taking one step in one go, then we add the current difference in both the heights(as required by the question) and we don't know the further step of how to calculate the final answer so we made a recursive call for the next iteration.
- If we have taken one step forward so we know that the next index would be i+1, rest parameters will observe no change.
- Here we have taken min(idx+1, n-1) , the reason being if we are standing at the n-1th step then saying hts[idx+1] will result in hts[n] which certainly does not exist.
- Now if you are trying to access any location in any array that does not exist then it will result in runtime error.
- To avoid that we have taken a minimum of n-1 and idx+1 or idx+2 (in case of two steps in one go) so that if idx+1 or idx+2 goes beyond the range of array it will automatically boil it down to n-1, the last index of array.
- Same is the explanation if we choose the 2nd option of taking 2 jumps in one go.
- In that case the only difference is that in that case you would have made a jump of 2 then your next index would be idx+2.

Upcoming Class Teasers:

- Bubble sort