<https://leetcode.com/problems/letter-combinations-of-a-phone-number/>

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"]

**Constraints:**

* 0 <= digits.length <= 4
* digits[i] is a digit in the range ['2', '9'].

**Attempt 1: 2022-11-26**

**Solution 1:  Backtracking**

**Style 1: Recursion with StringBuilder (10 min)**

class Solution {

public List<String> letterCombinations(String digits) {

List<String> result = new ArrayList<String>();

if(digits.length() == 0) {

return result;

}

String[] dictionary = new String[] {"", "", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"};

helper(digits, 0, dictionary, result, new StringBuilder());

return result;

}

private void helper(String digits, int index, String[] dictionary, List<String> result, StringBuilder sb) {

if(index == digits.length()) {

result.add(sb.toString());

return;

}

String str = dictionary[digits.charAt(index) - '0'];

for(char c : str.toCharArray()) {

sb.append(c);

helper(digits, index + 1, dictionary, result, sb);

sb.setLength(sb.length() - 1);

}

}

}

Time Complexity: O(4^n)

Since there are no more than 4 possible characters for each digit, the number of recursive calls, T(n), satisfies T(n) < 4T(n - 1), where n is the number of digits in the number. This solves to T(n) = O(4^n).

Each base case entails making a copy of a string and adding it to the result. Since each such string has length n, each base case takes time O(n). Therefore, the time complexity is O(n \* 4^n).

Space Complexity: O(4^n)

**Style 2: Recursion with String (10 min)**

class Solution {

public List<String> letterCombinations(String digits) {

List<String> result = new ArrayList<String>();

if(digits.length() == 0) {

return result;

}

String[] dictionary = new String[] {"", "", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"};

helper(digits, 0, dictionary, result, "");

return result;

}

private void helper(String digits, int index, String[] dictionary, List<String> result, String tmp) {

if(index == digits.length()) {

result.add(tmp);

return;

}

String str = dictionary[digits.charAt(index) - '0'];

for(char c : str.toCharArray()) {

helper(digits, index + 1, dictionary, result, tmp + c);

}

}

}

Time Complexity: O(4^n)

Since there are no more than 4 possible characters for each digit, the number of recursive calls, T(n), satisfies T(n) < 4T(n - 1), where n is the number of digits in the number. This solves to T(n) = O(4^n).

Each base case entails making a copy of a string and adding it to the result. Since each such string has length n, each base case takes time O(n). Therefore, the time complexity is O(n \* 4^n).

Space Complexity: O(4^n)

**Refer to**

<https://leetcode.com/problems/letter-combinations-of-a-phone-number/discuss/780232/Backtracking-Python-problems%2B-solutions-interview-prep>

class Solution(object):

def letterCombinations(self, digits):

"""

:type digits: str

:rtype: List[str]

"""

dic = { "2": "abc", "3": "def", "4":"ghi", "5":"jkl", "6":"mno", "7":"pqrs", "8":"tuv", "9":"wxyz"}

res=[]

if len(digits) ==0:

return res

self.dfs(digits, 0, dic, '', res)

return res

def dfs(self, nums, index, dic, path, res):

if index >=len(nums):

res.append(path)

return

string1 =dic[nums[index]]

for i in string1:

self.dfs(nums, index+1, dic, path + i, res)

**Solution 2:  BFS**

class Solution {

public List<String> letterCombinations(String digits) {

List<String> result = new LinkedList<String>();

if(digits.length() == 0) {

return result;

}

String[] dictionary = {"", "", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"};

result.add("");

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

String str = dictionary[digits.charAt(i) - '0'];

// Important: Only remove the all current level elements (e.g

// if digits = "23" -> all current level (first level) elements

// is 'a' and 'b', and we remove these 2 from result list for

// updating, don't remove more elements, since those are already

// updated next level elements) and for each element plus next

// character on all potentials (e.g if digits = "23" -> if current

// level is 'a', next level potential is 'd', 'e' and 'f')), then

// all new elements will build new level together

int size = result.size();

for(int j = 0; j < size; j++) {

String cur = result.remove(0);

for(char c : str.toCharArray()) {

result.add(cur + c);

}

}

}

return result;

}

}

**Refer to**

<https://leetcode.com/problems/letter-combinations-of-a-phone-number/discuss/8064/My-java-solution-with-FIFO-queue/205699>

public List<String> letterCombinations(String digits) {

List<String> ans = new LinkedList<String>();

if (digits.isEmpty())

return ans;

String[] mapping = new String[] { "0", "1", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz" };

ans.add(0, "");

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

int x = Character.getNumericValue(digits.charAt(i));

int size = ans.size(); // number of nodes/strings already in the queue

for (int k = 1; k <= size; k++) {

String t = ans.remove(0);

for (char s : mapping[x].toCharArray())

ans.add(t + s);

}

}

return ans;

}