Letter Combinations of a Phone Number

Given a digit string, return all possible letter combinations that the number could represent.

A mapping of digit to letters (just like on the telephone buttons) is given below.


```
Input:Digit string "23"
Output: ["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"].
```

Note:
Although the above answer is in lexicographical order, your answer could be in any order you want.

Solution 1

```
    public List<String> letterCombinations(String digits) {
    LinkedList<String> ans = new LinkedList<String>();
    String[] mapping = new String[] {"0", "1", "abc", "def", "ghi", "jkl", "mno",
"pqrs", "tuv", "wxyz"};
    ans.add("");
    for(int i =0; i<digits.length();i++){
        int x = Character.getNumericValue(digits.charAt(i));
        while(ans.peek().length()==i){
            String t = ans.remove();
            for(char s : mapping[x].toCharArray())
                ans.add(t+s);
        }
    }
    return ans;
}
```

written by lirensun original link here

Solution 2

This is my solution, FYI

```
vector<string> letterCombinations(string digits) {
    vector<string> res;
    string charmap[10] = {"0", "1", "abc", "def", "ghi", "jkl", "mno", "pqrs", "t
uv", "wxyz"};
    res.push_back("");
    for (int i = 0; i < digits.size(); i++)
    {
        vector<string> tempres;
        string chars = charmap[digits[i] - '0'];
        for (int c = 0; c < chars.size();c++)
            for (int j = 0; j < res.size();j++)
                tempres.push_back(res[j]+chars[c]);
        res = tempres;
    }
    return res;
}
```

written by peerlessbloom original link here

Solution 3

```
vector<string> letterCombinations(string digits) {
    vector<string> result;
    if(digits.empty()) return vector<string>();
    static const vector<string> v = {"", "", "abc", "def", "ghi", "jkl", "mno", "
pqrs", "tuv", "wxyz"};
    result.push_back("");   // add a seed for the initial case
    for(int i = 0 ; i < digits.size(); ++i) {
        int num = digits[i]-'0';
        if(num < 0 || num > 9) break;
        const string& candidate = v[num];
        if(candidate.empty()) continue;
        vector<string> tmp;
        for(int j = 0 ; j < candidate.size() ; ++j) {
            for(int k = 0 ; k < result.size() ; ++k) {
                tmp.push_back(result[k] + candidate[j]);
            }
        }
        result.swap(tmp);
    }
    return result;
}
```

Simple and efficient iterative solution.

Explanation with sample input "123"

Initial state:

- result = {""}

Stage 1 for number "1":

- result has {""}
- candiate is "abc"
- generate three strings "" + "a", ""+"b", ""+"c" and put into tmp, tmp = {"a", "b","c"}
- swap result and tmp (swap does not take memory copy)
- Now result has {"a", "b", "c"}

Stage 2 for number "2":

- result has {"a", "b", "c"}
- candidate is "def"
- generate nine strings and put into tmp, "a" + "d", "a"+"e", "a"+"f", "b" + "d", "b"+"e", "b"+"f", "c" + "d", "c"+"e", "c"+"f"
- so tmp has {"ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf" }
- swap result and tmp
- Now result has {"ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf" }

Stage 3 for number "3":

- result has {"ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf" }

- candidate is "ghi"
- generate 27 strings and put into tmp,
- add "g" for each of "ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"
- add "h" for each of "ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"
- add "h" for each of "ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"
- so, tmp has {"adg", "aeg", "afg", "bdg", "beg", "bfg", "cdg", "ceg", "cfg" "adh", "aeh", "afh", "bdh", "beh", "bfh", "cdh", "ceh", "cfh" "adi", "aei", "afi", "bdi", "bei", "bfi", "cdi", "cei", "cfi" }
- swap result and tmp
- Now result has {"adg", "aeg", "afg", "bdg", "beg", "bfg", "cdg", "ceg", "cfg" "adh", "aeh", "afh", "bdh", "beh", "bfh", "cdh", "ceh", "cfh" "adi", "aei", "afi", "bdi", "bei", "bfi", "cdi", "cei", "cfi" }

Finally, return result.

written by asbear original link here

From Leetcoder.