Letter Combinations of a Phone Number

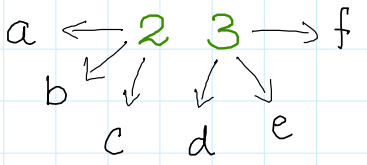
Pressing 2 can give "a", "b" or "c"

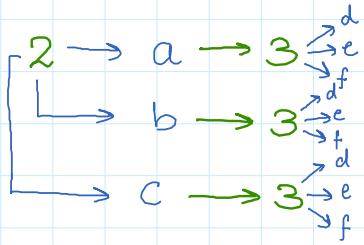
Pressing 23 can give : ad , ae , af , bd , be , bf , cd , ce , cf.

We have to return all possible combinations that can be formed by pressing a given phone number.

Note : Mapping exists only for numbers 2 - 9.

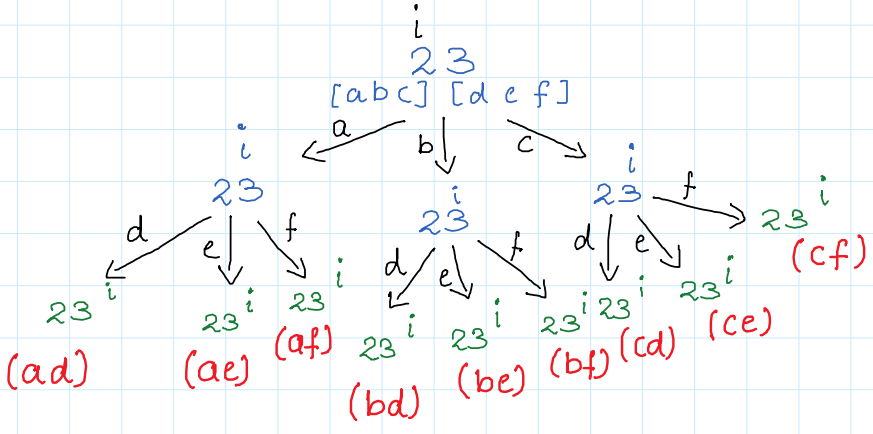
Approach : Just like subsets / subsequences (watch Lecture 37), we will iterate each digit in the phone number and for each alphabet belonging to the digit, we will choose it and move to the next digit recursively.





Example : Phone Number - 23

[a,b,c], [d,e,f]

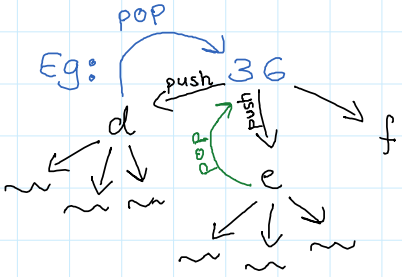


Code :

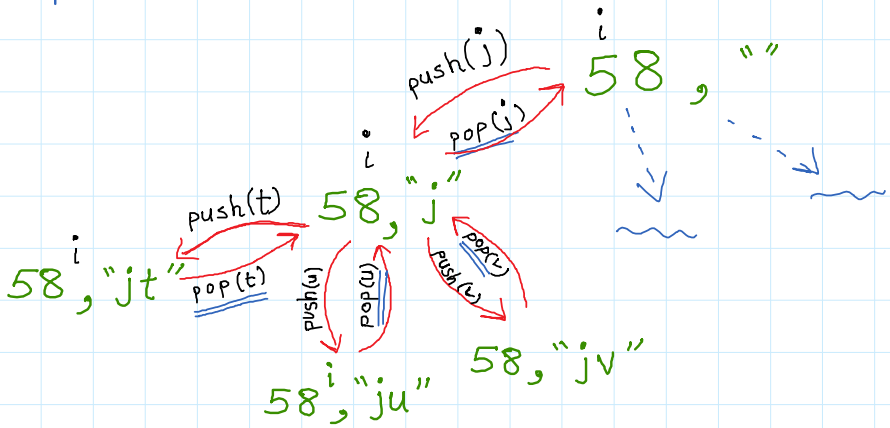
```
private:
    void solve(string digits, string mapping[], vector<string> &ans, string output, int i) {
        if(i >= digits.length()) {
            ans.push_back(output);
            return;
        }
        int number = digits[i] - '0';
        for(int j=0;j<mapping[number].length();j++) {
            output.push_back(mapping[number][j]);
            solve(digits, mapping, ans, output, i+1);
            output.pop_back();
        }

    }
public:
    vector<string> letterCombinations(string digits) {
        vector<string> ans;
        if(digits.length()==0)
            return ans;
        string output;
        int index = 0;
        string mapping[10] = {"", "", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"};
        solve(digits, mapping, ans, output, index);
        return ans;
    }
```

digits[i] will give me a character like "2" and not 2.

For all 'char' digits in the mapping of number, push it in our output string, get all combinations for this digit and then pop it to make way for the next digit.



Pop's Importance :



58 → [jkl, tuv]