Restore IP Addresses - QOTD 21 Jan 23

Leetcode Link : Click

GFG Link : CLick

★ $Approach - 1$ (recursive & backtracking)

Time : $O(3^n)$

> because at every level we have 3 choices to make

Space $O(3^n)$

> recursive call stack

approach :-

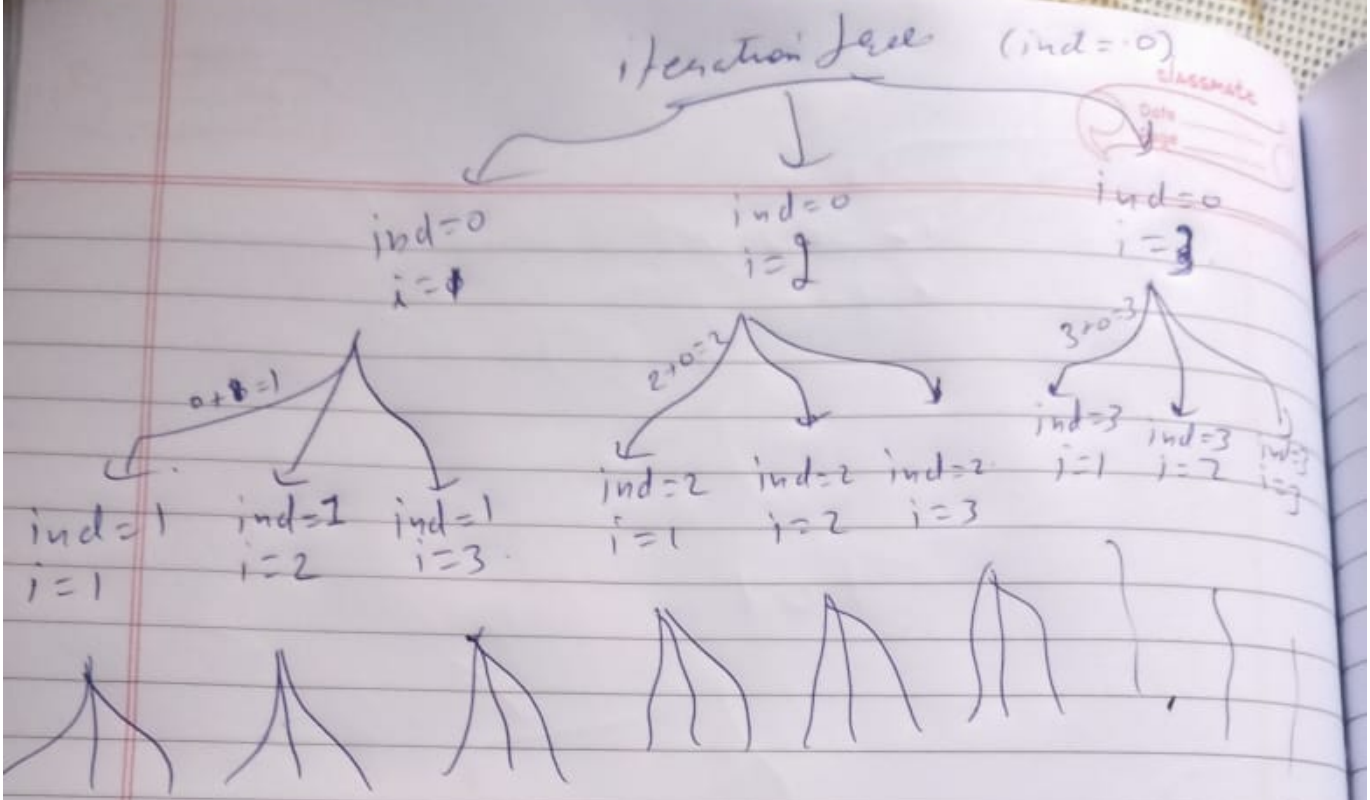
```
/* ✔️🌟Approach - 1 (recursive & backtracking)

    basic idea  :- for s = "1011" we have 3 choices at 1st level, i.e wheter we
include then [1] , we can include [10] , and  also [101] , if we go futther [1011]
this is out of the range 255 , so at every level we got 3 choices to make thats y we
can use recursion and backtracking here

    explanation :-

            ->// Fun.2 : solve()

                step 1 : base case - if the ans is of length 4 , and all the
characters of s are consumed, then push the ans into result
                step 2 : run a loop from 1 to 3
                    step 3 : break - if index + i > s.size() i.e if this substring
doesnt exist
                    step 4 : else - fetch out the substring from 's' and convert it
into int and save it
                    step 5 : check if the part is in range, and with no leading zeros
then push it into ans
                        step 6 : if the parts length is more then 1 with leading 0
then break the loop (i.e no need for check for greater value of i)
                        step 7 : if the parts length 1, then push it into ans (even
when its 0)
                            step 8 : recursively call for index + i, and pop the back
while coming back

         ->// Fun.1 : main function
             step 1 : create vector<string> ans and result, and index= 0
             step 2 : call function.2 solve()
             step 3 : return result

*/
```

code :-

```
private:
    // Fun.2 : solveRec()
    void solveRec(string &s, vector<string> &result, vector<string> &ans, int index){

        // step 1 : base case - if the ans is of length 4 , and all the characters of
s are consumed, then push the ans into result
        if(ans.size() == 4){
            if(index == s.length()){
                string temp = "";
                temp += ans[0] + "." + ans[1] + "." + ans[2] + "." + ans[3];
                result.push_back(temp);
            }
            return;
        }

        // step 2 : run a loop from 1 to 3
        for(int i = 1; i <= 3; i++){

            // step 3 : break - if index + i > s.size() i.e if this substring doesnt
exist
            if(index + i > s.length()) break;

            // step 4 : else - fetch out the substring from 's' and convert it into
int and save it
            string part = s.substr(index,i);

            // step 5 : check if the part is in range, and with no leading zeros then
push it into ans
            if(stoi(part) <= 255 && stoi(part) >= 0){

                // step 6 : if the parts length is more then 1 with leading 0 then
break the loop (i.e no need for check for greater value of i)
                if(part.length() > 1 && part[0] == '0') break;

                //  step 7 : if the parts length 1, then push it into ans (even when
its 0)
                else{
                    ans.push_back(part);

                    // step 8 : recursively call for index + i, and pop the back while
coming back
                    solveRec(s, result, ans, index + i);
                    ans.pop_back();
                }
            }
        }
    }
public:
    // Fun.1 : main function
    vector<string> restoreIpAddresses(string s) {
```

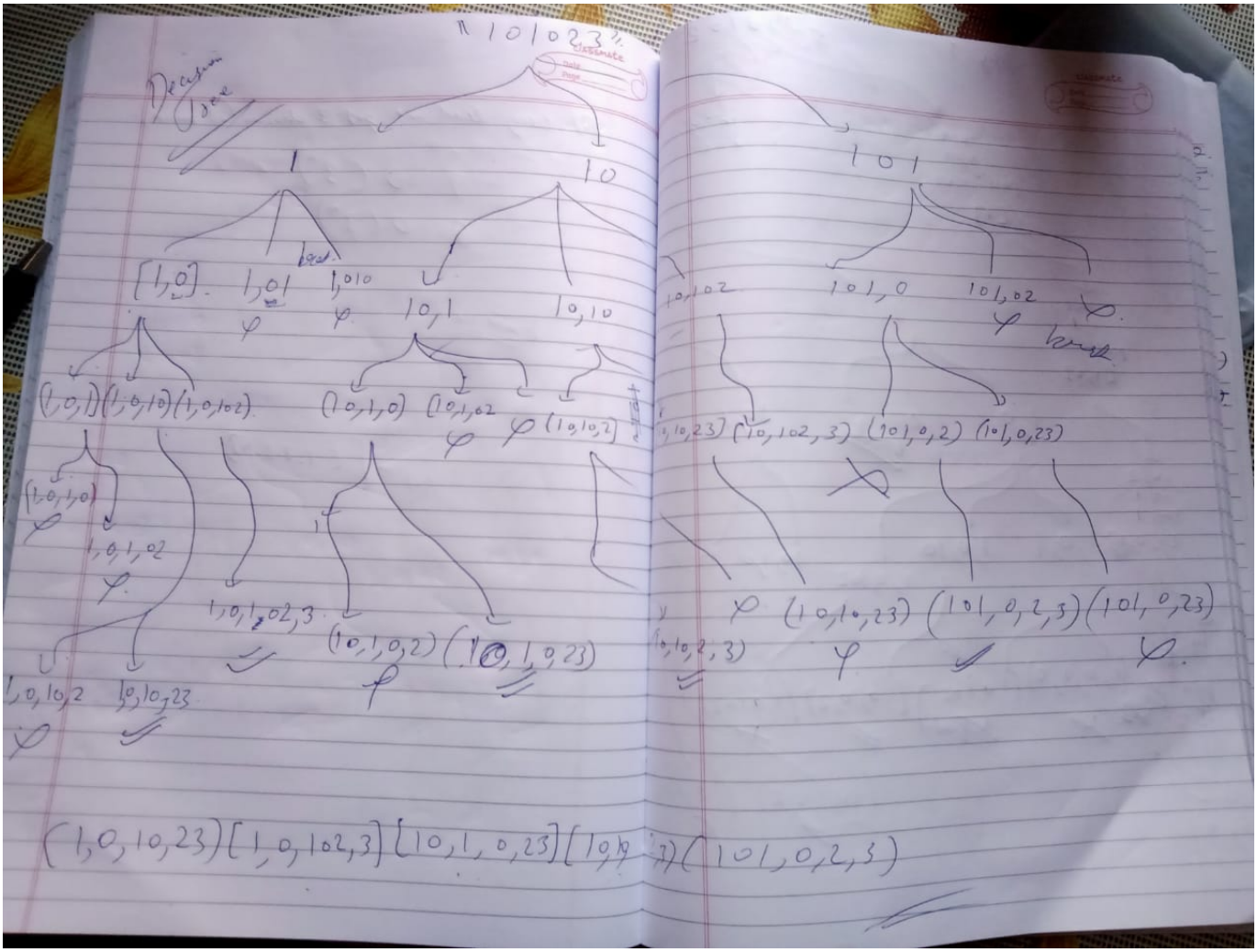
```
        // step 1 : create vector<string> ans and result, and index= 0
        vector<string> ans;
        vector<string> result;
        int index = 0;

        // step 2 : call function.2 solve()
        solveRec(s, result, ans, index);

        // step 3 : return result
        return result;

    }
```

Decision Tree :-

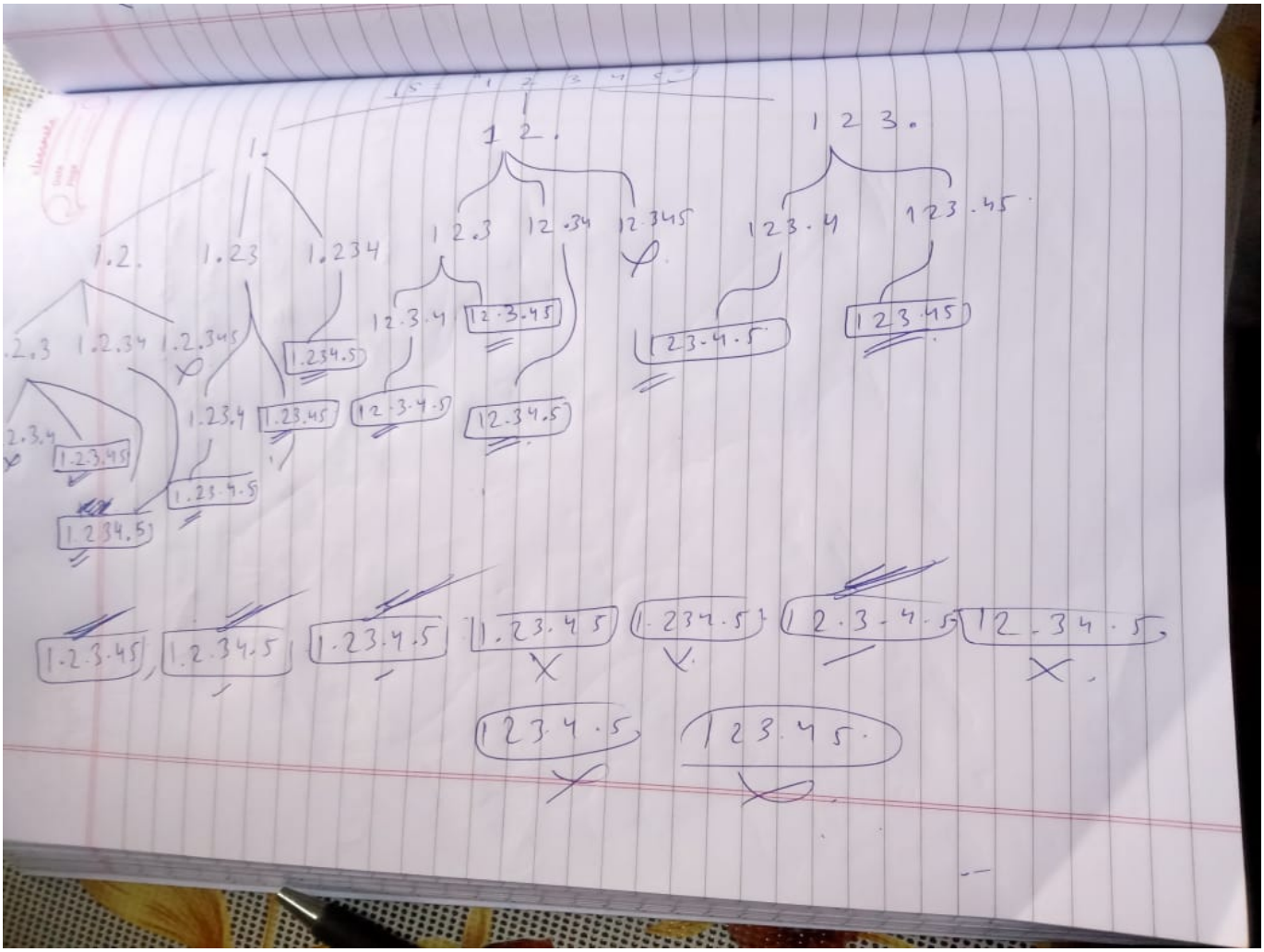

iteration level (ind=0)
ind=0
i=1
ind=0
i=2
ind=0
i=3
0+1=1
2+0=2
3+0=3
ind=1
i=1
ind=1
i=2
ind=1
i=3
ind=2
i=1
ind=2
i=2
ind=2
i=3
ind=3
i=1
ind=3
i=2

$T: O(3^n)$

$S: O(3^n)$


10102,3
1
10
[1,0]
1,01
1,010
10,1
10,10
(1,0,1)(1,0,10)(1,0,102)
(10,1,0)
(10,1,02
(1,10,2]
(1,0,1,0)
1,0,1,02
1,0,1,02,3
(10,1,0,2)
(1,0,10,2
1,0,10,23
(1,0,10,23)
(1,0,102,3]
(10,1,0,23]
(1,0,1,0,2,3)

101
10,102
101,0
101,02
(10,102,3)
(101,0,2)
(101,0,23)
(10,10,23)
(101,0,2,3)
(101,0,23)


1.
1.2.
1.23
1.234
1.2.3
1.2.34
1.2.345
1.234.5
1.23.4
1.23.45
1.23.4.5
1.234.5
1.2.3.4
1.2.3.45
1.2.34.5
1.23.4.5
1.23.45
1.234.5
12.
12.3
12.34
12.345
12.3.4
12.3.45
12.34.5
123.
123.4
123.45
123.4.5
123.45
1.2.3.45
1.2.34.5
1.23.4.5
1.23.45
1.234.5
12.3.4.5
12.34.5
123.4.5
123.45