**Problem Name:** Restore IP Addresses

**Topics:** String, Backtracking

**Companies:** Cisco, Microsoft, Oracle, Facebook, Amazon, Ebay, Twitter.

**Level:** Medium

**Language:** C++

**Problem Statement:** A valid IP address consists of exactly four integers separated by single dots. Each integer is between 0 and 255 (inclusive) and cannot have leading zeros.

* For example, "0.1.2.201" and "192.168.1.1" are valid IP addresses, but "0.011.255.245", "192.168.1.312" and "192.168@1.1" are invalid IP addresses.

Given a string s containing only digits, return *all possible valid IP addresses that can be formed by inserting dots into*s. You are not allowed to reorder or remove any digits in s. You may return the valid IP addresses in any order.

**Input Format:** The first and only line contain string s.

255255255255

**Output Format:** Print a vector of string containing all possible valid IP.

255.255.255.255

**Constraints:**

* 1 <= s.length <= 20
* s consists of digits only.

**Examples:**

**Input:** s = "25525511135"

**Output:** ["255.255.11.135","255.255.111.35"]

**Brute force Solution:**

**Explanation:** If the length of the string exceeds 12 it is impossible to obtain a valid partition of numbers for ip. Otherwise search through every partition and add that as a valid ip if it is indeed correct.

**Code:**

#include <bits/stdc++.h>

using namespace std;

set<string> ips;

void eval(int i,int j,int k,string& s){

    string ss = "";

    int n = s.length();

    vector<int>test = {-1,i,j,k,n-1};

    for(int x=1;x<5;x++){

        int cur = 0;

        for(int l=test[x-1]+1;l<=test[x];l++){

            cur\*=10;

            cur+=s[l]-'0';

        }

        if(cur<=255){

            string tf = s.substr(test[x-1]+1,test[x]-test[x-1]);

            if(tf!=to\_string(cur))return;

            ss+=tf;

            if(x<4)ss+=".";

        }

        else return;

    }

    ips.insert(ss);

}

vector<string> restoreIpAddresses(string s) {

    if(s.length()>12)return {};

    int n = s.length();

    for(int i = 0;i<n-1;i++){

        for(int j=i+1;j<n-1;j++){

            for(int k=j+1;k<n-1;k++){

                    eval(i,j,k,s);

            }

        }

    }

    vector<string> ans(ips.begin(),ips.end());

    return ans;

}

int main(){

      string s;

      cin>>s;

      vector<string> res;

      res = restoreIpAddresses(s);

      for(int i=0; i< res.size(); i++){

          cout<<res[i]<<" ";

      }

  }

**Time Complexity**: O(1)

**Space Complexity:** O(1)

**Optimized Solution:**

**Explanation:** A simple solution would be to just try every possible length for the 4 ip numbers. That's 3^4 = 81 possible values, since the total length of all the numbers must equal the string size, if you know your first 3 number lengths, the 4 is known. Therefore, there is actually only 3^3 = 27 possible values to check.

Checking 1.  len == 1 || '0' != digit[0] will handle that.

Checking 2. memcmp directly against a string. Fast and efficient, it allows the entire solution to be done without ever converting a string to a number or a number to a string.

**Code:**

#include <bits/stdc++.h>

using namespace std;

vector<string> restoreIpAddresses(string s) {

    vector<string> ret;

    for (int a = 1; a <= 3; ++a) {

      for (int b = 1; b <= 3; ++b) {

        for (int c = 1; c <= 3; ++c) {

          int d = s.size() - a - b - c;

          // Last number must use all remaining digits. Check;

          // 1. The size of the last number is valid

          // 2. Every number uses 1 digit for 0 and is less than 255 if using 3 digits

          if (1 <= d && d <= 3 &&

            (1 == a || '0' != s[0        ] && (3 != a || 0 < memcmp("256", &s[0        ], 3))) &&

            (1 == b || '0' != s[a        ] && (3 != b || 0 < memcmp("256", &s[a        ], 3))) &&

            (1 == c || '0' != s[a + b    ] && (3 != c || 0 < memcmp("256", &s[a + b    ], 3))) &&

            (1 == d || '0' != s[a + b + c] && (3 != d || 0 < memcmp("256", &s[a + b + c], 3))))

          {

            ret.emplace\_back(s.substr(0, a) + "." + s.substr(a, b) + "." + s.substr(a + b, c) + "." + s.substr(a + b + c));

          }

        }

      }

    }

    return ret;

  }

  int main(){

      string s;

      cin>>s;

      vector<string> res;

      res = restoreIpAddresses(s);

      for(int i=0; i< res.size(); i++){

          cout<<res[i]<<" ";

      }

  }

**Time Complexity**: O(1)

**Space Complexity:** O(1)