Print all combinations of balanced parentheses

Difficulty Level : Hard • Last Updated : 13 Jun, 2022

Write a function to generate all possible n pairs of balanced parentheses.

Examples:

```
Input: n=1
Output: {}
Explanation: This the only sequence of balanced
parenthesis formed using 1 pair of balanced parenthesis.

Input : n=2
Output:
{}{}
{{}}
Explanation: This the only two sequences of balanced
parenthesis formed using 2 pair of balanced parenthesis.
```

with n pairs. So there are n opening brackets and n closing brackets.
So the subsequence will be of length 2*n. There is a simple idea, the i'th character can be '{' if and only if the count of '{' till i'th is less than n and i'th character can be '}' if and only if the count of '{' is greater than the count of '}' till index i. If these two cases are followed then the resulting subsequence will always be balanced.
So form the recursive function using the above two cases.

Algorithm:

1. Create a recursive function that accepts a string (s), count of opening brackets (o) and count of closing brackets (c) and the value of n.
2. if the value of opening bracket and closing bracket is equal to n then print the string and return.
3. If the count of opening bracket is greater than count of closing bracket then call the function recursively with the following parameters String *s + "}"*, count of opening bracket *o*, count of closing bracket *c + 1*, and n.
4. If the count of opening bracket is less than n then call the function recursively with the following parameters String *s + "{"*, count of opening bracket *o + 1*, count of closing bracket *c*, and n.

Implementation:

C++

```
// c++ program to print all the combinations of balanced
// parenthesis

#include <bits/stdc++.h>
using namespace std;
vector<string> generateParenthesis(int n)
```

```
        two.push_back("{}");
        return two;
    } // Returning vector if n==1
    if (n == 2) {
        two.push_back("{{}}");
        two.push_back("{}{}");
        return two;
    } // Returning vector if n==2, as these are base cases

    two = generateParenthesis(
        n - 1); // Recursively calling the function

    // Assigning the previous values of vector into the
    // present function
    for (int i = 0; i < two.size(); i++) {
        string buf = "{", bug = "{", bus = "{";
        buf = buf + two[i] + "}";
        bug = bug + "}" + two[i];
        bus = two[i] + bus + "}";
        ans.push_back(buf);
        ans.push_back(bus);
        ans.push_back(bug);
    }

    // Removing the duplicate as kind of this {}{} remains
    // same in either way
    ans.pop_back();

    return ans;
}
int main()
{
    vector<string>
        ff; // Vector to store all the combinations
    int n = 2;
    ff = generateParenthesis(n); // Calling the function

    for (int i = 0; i < ff.size(); ++i) {
        cout << ff[i] << endl; // Print all the combinations
        /* code */
    }
```

C

```c
// C program to Print all combinations
// of balanced parentheses
#include <stdio.h>
#define MAX_SIZE 100

void _printParenthesis(int pos, int n, int open, int close);

// Wrapper over _printParenthesis()
void printParenthesis(int n)
{
    if (n > 0)
        _printParenthesis(0, n, 0, 0);
    return;
}

void _printParenthesis(int pos, int n, int open, int close)
{
    static char str[MAX_SIZE];

    if (close == n) {
        printf("%s \n", str);
        return;
    }
    else {
        if (open > close) {
            str[pos] = '}';
            _printParenthesis(pos + 1, n, open, close + 1);
        }

        if (open < n) {
            str[pos] = '{';
            _printParenthesis(pos + 1, n, open + 1, close);
        }
    }
}

// Driver program to test above functions
int main()
```

```
    return 0;
}
```

Java

```
// Java program to print all
// combinations of balanced parentheses
import java.io.*;

class GFG {
    // Function that print all combinations of
    // balanced parentheses
    // open store the count of opening parenthesis
    // close store the count of closing parenthesis
    static void _printParenthesis(char str[], int pos,
                                  int n, int open,
                                  int close)
    {
        if (close == n) {
            // print the possible combinations
            for (int i = 0; i < str.length; i++)
                System.out.print(str[i]);
            System.out.println();
            return;
        }
        else {
            if (open > close) {
                str[pos] = '}';
                _printParenthesis(str, pos + 1, n, open,
                                  close + 1);
            }
            if (open < n) {
                str[pos] = '{';
                _printParenthesis(str, pos + 1, n, open + 1,
                                  close);
            }
        }
    }
```

```
            _printParenthesis(str, 0, n, 0, 0);
        return;
    }

    // driver program
    public static void main(String[] args)
    {
        int n = 3;
        char[] str = new char[2 * n];
        printParenthesis(str, n);
    }
}

// Contributed by Pramod Kumar
```

Python3

```
# Python3 program to
# Print all combinations
# of balanced parentheses

# Wrapper over _printParenthesis()


def printParenthesis(str, n):
    if(n > 0):
        _printParenthesis(str, 0,
                          n, 0, 0)
    return


def _printParenthesis(str, pos, n,
                      open, close):

    if(close == n):
        for i in str:
            print(i, end="")
        print()
        return
    else:
```

```
        if(open < n):
            str[pos] = '{'
            _printParenthesis(str, pos + 1, n,
                              open + 1, close)


# Driver Code
n = 3
str = [""] * 2 * n
printParenthesis(str, n)

# This Code is contributed
# by mits.
```

C#

```
// C# program to print all
// combinations of balanced parentheses
using System;

class GFG {
    // Function that print all combinations of
    // balanced parentheses
    // open store the count of opening parenthesis
    // close store the count of closing parenthesis
    static void _printParenthesis(char[] str, int pos,
                                  int n, int open,
                                  int close)
    {
        if (close == n) {
            // print the possible combinations
            for (int i = 0; i < str.Length; i++)
                Console.Write(str[i]);

            Console.WriteLine();
            return;
        }
        else {
            if (open > close) {
```

```
            if (open < n) {
                str[pos] = '{';
                _printParenthesis(str, pos + 1, n, open + 1,
                                  close);
            }
        }
    }

    // Wrapper over _printParenthesis()
    static void printParenthesis(char[] str, int n)
    {
        if (n > 0)
            _printParenthesis(str, 0, n, 0, 0);
        return;
    }

    // driver program
    public static void Main()
    {
        int n = 3;
        char[] str = new char[2 * n];

        printParenthesis(str, n);
    }
}

// This code is contributed by Sam007
```

PHP

```
<?php
// PHP program to Print
// all combinations of
// balanced parentheses
$MAX_SIZE = 100;

// Wrapper over
// _printParenthesis()
function printParenthesis($str, $n)
{
```

```
}

// Function that print
// all combinations of
    // balanced parentheses
    // open store the count of
    //  opening parenthesis
    // close store the count of
    //  closing parenthesis
function _printParenthesis($str, $pos, $n,
                           $open, $close)
{
    if($close == $n)
    {
        for ($i = 0;
             $i < strlen($str); $i++)
        echo $str[$i];
        echo "\n";
        return;
    }
    else
    {
        if($open > $close)
        {
            $str[$pos] = '}';
            _printParenthesis($str, $pos + 1, $n,
                              $open, $close + 1);
        }

        if($open < $n)
        {
        $str[$pos] = '{';
        _printParenthesis($str, $pos + 1, $n,
                          $open + 1, $close);
        }
    }
}

// Driver Code
$n = 3;
$str="     ";
```

Javascript

```
<script>
// javascript program to print all
// combinations of balanced parentheses

    // Function that print all combinations of
    // balanced parentheses
    // open store the count of opening parenthesis
    // close store the count of closing parenthesis
    function _printParenthesis( str , pos , n , open , close)
    {
        if (close == n)
        {

            // print the possible combinations
            for (let i = 0; i < str.length; i++)
                document.write(str[i]);
            document.write("<br/>");
            return;
        }
        else
        {
            if (open > close)
            {
                str[pos] = '}';
                _printParenthesis(str, pos + 1, n, open, close + 1);
            }
            if (open < n)
            {
                str[pos] = '{';
                _printParenthesis(str, pos + 1, n, open + 1, close);
            }
        }
    }

    // Wrapper over _printParenthesis()
    function printParenthesis( str , n)
    {
```

```
    // Driver program
    var n = 3;
    var str = new Array(2 * n);
    printParenthesis(str, n);

// This code is contributed by shikhasingrajput
</script>
```

Output

```
{{}}
{}{}
```

Let's see the implementation of the same algorithm in a slightly different, simple and concise way :

C++

```
// c++ program to print all the combinations of balanced
// parenthesis.
#include <bits/stdc++.h>
using namespace std;
// function which generates all possible n pairs of balanced
// parentheses. open : count of the number of open
// parentheses used in generating the current string s. close
// : count of the number of closed parentheses used in
// generating the current string s. s : currently generated
// string/ ans : a vector of strings to store all the valid
// parentheses.
void generateParenthesis(int n, int open, int close,
                         string s, vector<string>& ans)
{
    // if the count of both open and close parentheses
    // reaches n, it means we have generated a valid
    // parentheses. So, we add the currently generated string
    // s to the final ans and return.
    if (open == n && close == n) {
```

```
    // can put an open parentheses only if its count until
    // that time is less than n.
    if (open < n) {
        generateParenthesis(n, open + 1, close, s + "{",
                            ans);
    }
    // At any index i in the generation of the string s, we
    // can put a closed parentheses only if its count until
    // that time is less than the count of open parentheses.
    if (close < open) {
        generateParenthesis(n, open, close + 1, s + "}",
                            ans);
    }
}
int main()
{
    int n = 3;
    // vector ans is created to store all the possible valid
    // combinations of the parentheses.
    vector<string> ans;
    // initially we are passing the counts of open and close
    // as 0, and the string s as an empty string.
    generateParenthesis(n, 0, 0, "", ans);
    // Now, here we print out all the combinations.
    for (auto s : ans) {
        cout << s << endl;
    }
    return 0;
}
```

Java

```
// Java program to print all the combinations of balanced
// parenthesis.
import java.util.*;

class GFG {

    // function which generates all possible n pairs of
```

```
// the current string s. s : currently generated string/
// ans : a vector of strings to store all the valid
// parentheses.
public static void
generateParenthesis(int n, int open, int close,
                    String s, ArrayList<String> ans)
{

    // if the count of both open and close parentheses
    // reaches n, it means we have generated a valid
    // parentheses. So, we add the currently generated
    // string s to the final ans and return.
    if (open == n && close == n) {
        ans.add(s);
        return;
    }

    // At any index i in the generation of the string s,
    // we can put an open parentheses only if its count
    // until that time is less than n.
    if (open < n) {
        generateParenthesis(n, open + 1, close, s + "{",
                            ans);
    }

    // At any index i in the generation of the string s,
    // we can put a closed parentheses only if its count
    // until that time is less than the count of open
    // parentheses.
    if (close < open) {
        generateParenthesis(n, open, close + 1, s + "}",
                            ans);
    }
}

public static void main(String[] args)
{
    int n = 3;

    // vector ans is created to store all the possible
    // valid combinations of the parentheses.
```

```
        generateParenthesis(n, 0, 0, "", ans);

        // Now, here we print out all the combinations.
        for (String s : ans) {
            System.out.println(s);
        }
    }
}

// This code is contributed by ninja hattori
```

Python3

```
# Python program to print all the combinations of balanced parenthesis.

# function which generates all possible n pairs of balanced parentheses.
# open : count of the number of open parentheses used in generating the c
# close : count of the number of closed parentheses used in generating th
# s : currently generated string/
# ans : a vector of strings to store all the valid parentheses.


def generateParenthesis(n,  Open,  close,  s, ans):

      # if the count of both open and close parentheses reaches n, it mea
      # So, we add the currently generated string s to the final ans and
    if(Open == n and close == n):
        ans.append(s)
        return

      # At any index i in the generation of the string s, we can put an o
    if(Open < n):
        generateParenthesis(n, Open+1, close, s+"{", ans)

  # At any index i in the generation of the string s, we can put a closed
    if(close < Open):
        generateParenthesis(n, Open, close + 1, s+"}", ans)


n = 3
```

```
generateParenthesis(n, 0, 0, "", ans)

# Now, here we print out all the combinations.
for s in ans:
    print(s)

    # This code is contributed by ninja_hattori.
```

C#

```
// C# program to print all the combinations of balanced
// parenthesis.
using System;
using System.Collections;

public class GFG {

    // function which generates all possible n pairs of
    // balanced parentheses.
    // open : count of the number of open parentheses used
    // in generating the current string s. close : count of
    // the number of closed parentheses used in generating
    // the current string s. s : currently generated string/
    // ans : a vector of strings to store all the valid
    // parentheses.

    static public void generateParenthesis(int n, int open,
                                           int close,
                                           string s,
                                           ArrayList ans)
    {

        // if the count of both open and close parentheses
        // reaches n, it means we have generated a valid
        // parentheses. So, we add the currently generated
        // string s to the final ans and return.
        if (open == n && close == n) {
            ans.Add(s);
            return;
```

```
            // until that time is less than n.
            if (open < n) {
                generateParenthesis(n, open + 1, close, s + "{",
                                    ans);
            }

            // At any index i in the generation of the string s,
            // we can put a closed parentheses only if its count
            // until that time is less than the count of open
            // parentheses.
            if (close < open) {
                generateParenthesis(n, open, close + 1, s + "}",
                                    ans);
            }
        }

        static public void Main()
        {
            // Code

            int n = 3;

            // vector ans is created to store all the possible
            // valid combinations of the parentheses.
            ArrayList ans = new ArrayList();

            // initially we are passing the counts of open and
            // close as 0, and the string s as an empty string.
            generateParenthesis(n, 0, 0, "", ans);

            // Now, here we print out all the combinations.
            for (int i = 0; i < ans.Count; i++) {
                Console.WriteLine(ans[i]);
            }
        }
    }

    // This code is contributed by ninja_hattori.
```

Javascript

```
// function which generates all possible n pairs of balanced parentheses.
// open : count of the number of open parentheses used in generating the
// close : count of the number of closed parentheses used in generating t
// s : currently generated string/
// ans : an array of strings to store all the valid parentheses.
function generateParenthesis(n, open, close, s, ans)
{

    // if the count of both open and close parentheses reaches n, it mean
    // So, we add the currently generated string s to the final ans and r
    if(open == n && close == n){
        ans.push(s);
        return;
    }

    // At any index i in the generation of the string s, we can put an op
    if(open < n)
    {
        generateParenthesis(n, open + 1, close, s + "{", ans);
    }

    // At any index i in the generation of the string s, we can put a clo
    if(close < open)
    {
        generateParenthesis(n, open, close + 1, s + "}", ans);
    }

}

// Driver code
let n = 3;

// array of strings ans is created to store all the possible valid combin
// of the parentheses.
let ans = [];

// initially we are passing the counts of open and close as 0,
// and the string s as an empty string.
generateParenthesis(n, 0, 0, "", ans);

// Now, here we print out all the combinations.
```

```
// This code is contributed by shinjanpatra
</script>
```

Output

```
{{{}}}
{{}{}}
{{}}{}
{}{{}}
{}{}{}
```

Thanks to Shekhu for providing the above code.

Complexity Analysis:

- **Time Complexity:** O(2^n).
  For every index there can be two options '{' or '}'. So it can be said that the upper bound of time complexity is O(2^n) or it has exponential time complexity.
- **Space Complexity:** O(n).
  The space complexity can be reduced to O(n) if global variable or static variable is used to store the output string.

Please write comments if you find the above codes/algorithms incorrect, or find better ways to solve the same problem.

Previous

Generate integer from 1 to 7 with equal probability

RECOMMENDED ARTICLES

Page : 1 2 3

01 **Count pairs of parentheses sequences such that parentheses are balanced**
07, Jan 19

02 **Pairs involved in Balanced Parentheses**
14, Apr 18

03 **Check for balanced parentheses in an expression | O(1) space**
29, May 18

04 **Check for balanced parentheses in Python**
18, Jan 19

05 **Modify a numeric string to a balanced parentheses by replacements**
30, Jul 21

06 **Number of balanced parentheses substrings**
18, Jul 19

07 **Check for balanced parentheses in an expression | O(1) space | O(N^2) time complexity**
03, May 19

08 **Length of longest balanced parentheses prefix**
08, Dec 17

Vote for difficulty

Current difficulty : Hard

Easy | Normal | Medium | Hard | Expert

Improved By : Mithun Kumar, SmilingBuddha, ThangarajGFG, andrew1234, Akanksha_Rai, shikhasingrajput, hemanthlishetti999, krrishsai648, kalrap615, jaisaikuntala1, simranarora5sos, raviutsav0, shinjanpatra, ninja_hattori, Code_r

Article Tags : Parentheses-Problems, Samsung, Mathematical

Practice Tags : Samsung, Mathematical

Improve Article | Report Issue

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

GeeksforGeeks
A-143, 9th Floor, Sovereign Corporate Tower,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

NodeJS