Contents

[1. Reverse Linked List in k groups 2](#_Toc464591043)

[2. Sort Linked list using Merge sort 2](#_Toc464591044)

[3. Find intersection of two LinkedLists 2](#_Toc464591045)

[4. Valid Parentheses 3](#_Toc464591046)

[5. WildCard 3](#_Toc464591047)

[6. Longest Valid Parentheses 3](#_Toc464591048)

[7. Expression evaluator 4](#_Toc464591049)

[8. Palindrome partitions 4](#_Toc464591050)

[9. New problem 5](#_Toc464591051)

# Reverse Linked List in k groups

<todo>

# Sort Linked list using Merge sort

**Tags:** LinkedList, Sorting, Merge sort, LeetCode

Merge sort’s merge operation usually requires additional space for merging into new array. But, using LinkstList that need can be avoided.

public int getLen(ListNode h) { }

public ListNode merge(ListNode l, ListNode r) { }

public ListNode mergeRecursive(ListNode l, ListNode r)

{

// Stack overflow problem on LeetCode with around 31k size length

}

public ListNode sortList(ListNode head, int len)

{

int mid = len/2;

ListNode left = sortList(firstHalf, mid);

ListNode right = sortList(secondHalf, len - mid);

return merge(left, right);

}

# Find intersection of two LinkedLists

**Tags:** LinkedList, LeetCode

Given that m and n are lengths of two LLs -

1. Time O(m\*n); Space O(1)

For each node in m, search entire LinkedList n. If you find a match return that node.

1. **Time O(m+n); Space O(1)**

Find length of each list; Then for longest list traverse abs(m-n) nodes; last step - traverse both lists one at a time till they are same;

1. Time O(m+n); space O(m+n)

Using hashing (set can be used too)

Traverse first list and hash each node; for second list while traversing each node check in hash if it is present; if matched report that node as intersection

1. Time O(m+n); space O(m+n)

Use stack; Idea is use the fact that both list share same size from the end.

While traversing both the list put them in 2 separate stacks; now start poping each element from two stacks simultaneously, if they match report that node as intersection.

# Valid Parentheses

**Tags:** Stack, LeetCode

Given a string containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

The brackets must close in the correct order, "()" and "()[]{}" are all valid but "(]" and "([)]" are not

Solution:

Use a stack to push (, { and ]. Pop during closing brackets.

# WildCard

**Tags:** Recursion

Given a string 123?, replace ? with ‘0’ or ‘1’. So, output will be 1230, 1231

# Longest Valid Parentheses

**Tags:** Stack, LeetCode

**Problem statement:**

Given a string containing just the characters '(' and ')', find the length of the longest valid (well-formed) parentheses substring.

For "(()", the longest valid parentheses substring is "()", which has length = 2.

Another example is ")()())", where the longest valid parentheses substring is "()()", which has length = 4.

**Solution [1]**: Time O(n); Space O(n)

Keep pushing **index** of ‘(‘ in the stack. On finding ‘)’ pop the index and calculate the length (current\_index – index\_of\_top\_element\_from\_stack)

# Expression evaluator

**Tags:** Leetcode, recursion

**Problem statement:**

Given a string that contains only digits 0-9 and a target value, return all possibilities to add binary operators (not unary) +, -, or \* between the digits so they evaluate to the target value.

Examples:

"123", 6 -> ["1+2+3", "1\*2\*3"]

"232", 8 -> ["2\*3+2", "2+3\*2"]

"105", 5 -> ["1\*0+5","10-5"]

"00", 0 -> ["0+0", "0-0", "0\*0"]

"3456237490", 9191 -> []

**Solution:**

Time: O(3^(n-1)); n – number of digits and 3 – number of operators

Space: O(n); size of stack

Using backtracking approach, one can quickly code it’s solution

# Palindrome partitions

**Tags:** LeetCode, recursion

**Problem statement:**

If given string is ‘aab’, then return palindrome partitions like this [ [“aa”, “b”], [“a”, “a”, “b”]]

**Solution:**

Time: O(n\* 2^n); n – Palindrome check; 2^n total substring partitioning.

Space: O(n) –

T(n) = T(n-1) + T(n-2) + … + 1

T(n+1) = T(n) + T(n-1) + … +1 = T(n) + T(n) = 2\*T(n)

Using backtracking, it can be solved easily.

public void findPartitions(string s, int start, List<IList<string>> res, List<string> buff) {

if (s.Length == start) {

res.Add(new List<string> (buff));

return;

}

for(int i=start ; i<s.Length ; i++) {

if (isPalindrome(s, start, i)) {

buff.Add(s.Substring(start, i-start+1));

findPartitions(s, i+1, res, buff);

buff.RemoveAt(buff.Count-1);

}

}

}

# Is Bst

**Tags:** LeetCode, homework

**Problem statement:** Check if a BST is a BST. That tree does not have any duplicates

**Answer:** Time O(n), Space O(n)

<http://articles.leetcode.com/determine-if-binary-tree-is-binary>

Start with left and right values and keep reducing based on sub tree we explore.

# Flip a Tree (create mirror image)

**Tags:** LeetCode, homework

**Problem statement:** Convert a tree into its mirror image

**Answer:**

[1] Recursive approach: Time O(n); Space O(h)

Change left and right from its root. After that make two recursive calls to same function with left and right subtrees.

[2] Iterative approach: Time O(n); Space O(n/2)

Use BFS using queue. Then while putting before left and right into queue, change pointer of left and right tree

<https://leetcode.com/articles/invert-binary-tree/>

# Print root to leaf path

**Tags:** LeetCode, Homework

**Solution:** Time O(n\*h) (n: Total nodes; h: string from root to leaf); Space O(h)

Recursive solution.

# New problem