2. Add Two Numbers

April 5, 2016 linked-list (/articles/?tag=linked-list)

Question | Editorial Solution

Question

> You are given two **non-empty** linked lists representing two non-negative integers. The digits are stored in reverse order and each of their nodes contain a single digit. Add the two numbers and return it as a linked list.
>
> You may assume the two numbers do not contain any leading zero, except the number 0 itself.
>
> `Input: (2 -> 4 -> 3) + (5 -> 6 -> 4)`
> `Output: 7 -> 0 -> 8`

Solution

Intuition

Keep track of the carry using a variable and simulate digits-by-digits sum starting from the head of list, which contains the least-significant digit.

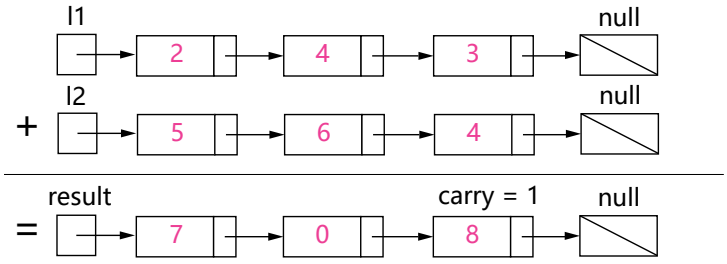

Figure 1. Visualization of the addition of two numbers: $342 + 465 = 807$*.*
Each node contains a single digit and the digits are stored in reverse order.

Algorithm

Just like how you would sum two numbers on a piece of paper, we begin by summing the least-significant digits, which is the head of $l1$ and $l2$. Since each digit is in the range of $0 \ldots 9$, summing two digits may "overflow". For example $5 + 7 = 12$. In this case, we set the current digit to 2 and bring over the $carry = 1$ to the next iteration. $carry$ must be either 0 or 1 because the largest possible sum of two digits (including the carry) is $9 + 9 + 1 = 19$.

The pseudocode is as following:

- Initialize current node to dummy head of the returning list.
- Initialize carry to 0.
- Initialize p and q to head of $l1$ and $l2$ respectively.
- Loop through lists $l1$ and $l2$ until you reach both ends.
    - Set x to node p's value. If p has reached the end of $l1$, set to 0.
    - Set y to node q's value. If q has reached the end of $l2$, set to 0.
    - Set $sum = x + y + carry$.
    - Update $carry = sum / 10$.
    - Create a new node with the digit value of $(sum \bmod 10)$ and set it to current node's next, then advance current node to next.
    - Advance both p and q.
- Check if $carry = 1$, if so append a new node with digit 1 to the returning list.
- Return dummy head's next node.

Note that we use a dummy head to simplify the code. Without a dummy head, you would have to write extra conditional statements to initialize the head's value.

Take extra caution of the following cases:

Test case	Explanation
$l1 = [0, 1]$ $l2 = [0, 1, 2]$	When one list is longer than the other.
$l1 = []$ $l2 = [0, 1]$	When one list is null, which means an empty list.
$l1 = [9, 9]$ $l2 = [1]$	The sum could have an extra carry of one at the end, which is easy to forget.

Java

```
public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
    ListNode dummyHead = new ListNode(0);
    ListNode p = l1, q = l2, curr = dummyHead;
    int carry = 0;
    while (p != null || q != null) {
        int x = (p != null) ? p.val : 0;
        int y = (q != null) ? q.val : 0;
        int sum = carry + x + y;
        carry = sum / 10;
        curr.next = new ListNode(sum % 10);
        curr = curr.next;
        if (p != null) p = p.next;
        if (q != null) q = q.next;
    }
    if (carry > 0) {
        curr.next = new ListNode(carry);
    }
    return dummyHead.next;
}
```

Complexity Analysis

- Time complexity : $O(\max(m, n))$. Assume that m and n represents the length of $l1$ and $l2$ respectively, the algorithm above iterates at most $\max(m, n)$ times.
- Space complexity : $O(\max(m, n))$. The length of the new list is at most $\max(m, n) + 1$.

Follow up

What if the the digits in the linked list are stored in non-reversed order? For example:

$$(3 \to 4 \to 2) + (4 \to 6 \to 5) = 8 \to 0 \to 7$$

guojing_neo commented 6 days ago

func doAdd(l1 *ListNode, l2 *ListNode, step int) *ListNode{
if l1 == nil {
return l2
}

```
if l2 == nil {
    return l1
}

l := new(ListNode)

l.Val = (l1.Val + l2.Val + step) % 10
step = (l1.Val + l2.Val + step) / 10

l.Next=doAdd(l1.Next,l2.Next,step)
return l
```

}

func addTwoNumbers(l1 *ListNode, l2 *ListNode) *ListNode {
return doAdd(l1,l2,0)
}

how about the golang solution?

shannonliang312 commented last month

JS:
/**

- You are given two non-empty linked lists representing two non-negative integers. The digits are stored in reverse order and each of their nodes contain a single digit. Add the two numbers and return it as a linked list.

You may assume the two numbers do not contain any leading zero, except the number 0 itself.

Input: (2 -> 4 -> 3) + (5 -> 6 -> 4)
Output: 7 -> 0 -> 8
*/

/**

- Definition for singly-linked list.
- function ListNode(val) {
-

```
this.val = val;
```

-

```
this.next = null;
```

- }
  /
  /*
- @param {ListNode} l1
- @param {ListNode} l2
- @return {ListNode}
  */
  var addTwoNumbers = function(l1, l2) {
  var res = new ListNode(0); //头结点
  var current_node = res; //当前节点
  var next_node = res; //下一节点
  var tmp1 = l1; //l1链表当前节点
  var tmp2 = l2; //l2链表当前节点
  var tmp_sum = 0; //当前位数的和
  var tmp_carry = 0; //进位

  while(tmp1 !== null || tmp2 !== null) {
  if(tmp1 !== null && tmp2 !== null) {
  tmp_sum = tmp1.val + tmp2.val + tmp_carry;

tmp1 = tmp1.next;
tmp2 = tmp2.next;
} else if(tmp1 === null && tmp2 !== null) {
tmp_sum = tmp2.val + tmp_carry;
tmp2 = tmp2.next;
} else if(tmp2 === null && tmp1 !== null) {
tmp_sum = tmp1.val + tmp_carry;
tmp1 = tmp1.next;
}

```
//判断进位
if(tmp_sum >= 10) {
    tmp_sum = tmp_sum - 10;
    tmp_carry = 1;
} else {
    tmp_carry = 0;
}

current_node.val = tmp_sum;
//判断最高位
if(tmp1 !== null || tmp2 !== null) {
    next_node = new ListNode(0);
    current_node.next = next_node;
    current_node = next_node;
    current_node.next = null;
} else if(tmp_carry === 1) {
    next_node = new ListNode(1);
    current_node.next = next_node;
    current_node = next_node;
    current_node.next = null;
}
```

```
}

return res;
```

};

sinwan commented last month

Solution with recursion in JavaScript:
(discuss.leetcode.com/user/sinwan)

```
var addTwoNumbers = function(l1, l2) {
var listNode;
if(l1 && l2){
if(l1.val + l2.val >= 10) {
listNode = new ListNode(l1.val + l2.val - 10);
if (l1.next === null) {
l1.next = new ListNode(1);
} else {
l1.next.val++;
}
} else {
listNode = new ListNode(l1.val + l2.val);
}
listNode.next = addTwoNumbers(l1.next, l2.next);
} else if (l1) {
if (l1.val == 10) {
listNode = new ListNode(l1.val - 10);
if (l1.next === null) {
l1.next = new ListNode(1);
} else {
l1.next.val++;
}
} else {
listNode = new ListNode(l1.val);
}
listNode.next = addTwoNumbers(l1.next, null);
} else if (l2) {
listNode = new ListNode(l2.val);
listNode.next = l2.next;
} else {
listNode = null;
}
return listNode;
};
```

ZhengChengGui commented last month

My solution in C++:
(discuss.leetcode.com/user/zhengchenggui)

```
class Solution {
public:
ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
ListNode *result = new ListNode{l1->val};
ListNode *key = result;
while (l1->next) {
l1 = l1->next;
key->next = new ListNode{ l1->val };
key = key->next;
}
key = result;
```

```
        addNumber(key, l2->val);
        while (l2->next) {
                if (key->next)
                        addNumber(key->next, l2->next->val);
                else {
                        key->next = new ListNode{ 0 };
                        addNumber(key->next, l2->next->val);
                }
                key = key->next;
                l2 = l2->next;
        }

        return result;
}
void addNumber(ListNode *node, int a) {
        node->val += a;
        if (node->val > 9) {
                node->val -= 10;
                if (node->next)
                        addNumber(node->next, 1);
                else
                        node->next = new ListNode{ 1 };
        }
}
```

```
};
```

pianyao commented last month

Mine in Java:
(discuss.leetcode.com/user/pianyao)

```
public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
ListNode result = new ListNode(l1.val + l2.val);
ListNode prev = result;
l1 = l1.next;
l2 = l2.next;
while (l1 != null || l2 != null) {
int val1 = l1 != null ? l1.val : 0;
int val2 = l2 != null ? l2.val : 0;
prev.next = new ListNode(val1 + val2);
prev = prev.next;
l1 = l1 != null ? l1.next : null;
l2 = l2 != null ? l2.next : null;
}
```

```
    ListNode curr = result;
    while (curr != null) {
        if (curr.val > 9) {
            curr.val = curr.val - 10;
            if (curr.next != null) {
                curr.next.val++;
            } else {
                curr.next = new ListNode(1);
            }
        }
        curr = curr.next;
    }
    return result;
}
```

Send Feedback (mailto:admin@leetcode.com?subject=Feedback)

fsw0422 commented last month

```
def addTwoNumbers(self, l1, l2):
    p1 = l1
    p2 = l2
    output = []
    stack = [0]
    while p1 != None or p2 != None:
        n1 = n2 = 0
        if p1 != None: n1 = p1.val
        if p2 != None: n2 = p2.val

        r = stack.pop()
        if n1 + n2 + r >= 10:
            output.append(n1 + n2 + r - 10)
            stack.append(1)
        else:
            output.append(n1 + n2 + r)
            stack.append(0)

        if n1 + n2 + r >= 10:
            stack.append(1)

        if p1 != None: p1 = p1.next
        if p2 != None: p2 = p2.next

    rem = stack.pop()
    if rem != None and rem != 0: output.append(rem)
    return output
```

is there a way to reduce the Big O though?

yvette.ying.wang commented last month

public class Solution {
public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
ListNode p = l1;
ListNode q = l2;
ListNode l3 = new ListNode (0);
ListNode r = l3;

```
    int addup = 0;
    int carry = 0;
    boolean pflag = true;
    boolean qflag = true;

    while (p != null || q != null) {
        if (p == null && q != null) {
            addup = q.val + carry;
            pflag = false;
        }
        if (p != null && q == null) {
            addup = p.val + carry;
            qflag = false;
        }
        if (p != null && q != null) {
            addup = p.val + q.val + carry;
        }
        if (addup < 10) {
                r.next = new ListNode(addup);
                carry = 0;
            }
            else {
                r.next = new ListNode (addup - 10);
                carry = 1;
        }
            r = r.next;
            if (pflag) {
                p = p.next;
            }
        if (qflag) {
            q = q.next;
        }
    }
    if (carry == 1) {
        r.next = new ListNode (1);
    }
    return l3.next;
}
```

}

q986171791 commented 2 months ago

My solution in javascript
(discuss.leetcode.com/user/q986171791)

```
var addTwoNumbers = function(l1, l2) {
       var getModAndCarry = function (num) {
           var result = {
               m:num,
               carry:0
           };
           if(num < 10){
               return result;
           }
           result.m = num % 10;
           result.carry = (num - result.m) / 10;
           return result;
       }

       var l1Cur = l1,
               l2Cur = l2,
               result = [],
               ret = [],
               i = 0;
       //为每个节点加上 pre 节点代表上个节点
       while(true){
           //存储结果
           result.push(l1Cur.val + l2Cur.val);

           if(l1Cur.next == null && l2Cur.next == null){
               break;
           }
           l1Cur = l1Cur.next || {val:0,next:null};
           l2Cur = l2Cur.next  || {val:0,next:null};
       }
       if(result.length == 1 && result[0] == 0){
           return new ListNode(0);
       }
       var pre = 0;
       var curNode = null;
       result.forEach(function (num,i) {
           var cur = getModAndCarry(num+pre);
           pre = cur.carry;
           ret.push(new ListNode(cur.m));
           if(i == result.length - 1){
               if(cur.carry != 0){
                   ret.push(new ListNode(cur.carry));
               }
           }
       })
       ret.forEach(function (node,i) {
           if(ret[i] && ret[i + 1]){
               ret[i]['next'] = ret[i + 1];
           }
       })
       return ret[0];
   };
```

huge0x0 commented 2 months ago

My solution in CPP:

(discuss.leetcode.com/user/huge0x0)

ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
ListNode* result;
result=new ListNode(0);
ListNode* p=result;
int temSum=0;
while(l1!=nullptr&&l2!=nullptr){
temSum=l1->val+l2->val+(temSum>9);
p->next=new ListNode(temSum%10);
p=p->next;
l1=l1->next;
l2=l2->next;
}
l1=l2==nullptr?l1:l2;

```
    while(l1!=nullptr){
        temSum=l1->val+(temSum>9);
        p->next=new ListNode(temSum%10);
        p=p->next;
        l1=l1->next;
    }

    if(temSum>9)
        p->next=new ListNode(1);
    return result->next;
}
```

solana_casas commented 2 months ago

My solution in Java :)

(discuss.leetcode.com/user/solana_casas)

public class Solution {
public ListNode addTwoNumbers(ListNode l1, ListNode l2) {

```
    if (l1 == null) return null;

    ListNode tmp = l1;

    int x;
    while (l1 != null){
        x = l1.val + l2.val;
        if (x > 9){
            l1.val = x % 10;
            if(l1.next == null){
                l1.next = new ListNode(1);
                if (l2.next == null) l2.next = new ListNode(0);
            }else l1.next.val += 1;
        }else{
            l1.val = x;
        }

        if(l1.next == null && l2.next != null) l1.next = new ListNode(0);
        if(l2.next == null && l1.next != null) l2.next = new ListNode(0);

        l1 = l1.next;
        l2 = l2.next;
    }
    return tmp;
}
```

}

View original thread (https://discuss.leetcode.com/topic/54)

Load more comments...