Python Coding Solution

Introduction of python

Q1 写一个程序，根据用户输入的小时数和小时工资额，来计算出应付工资
Enter Hours: 35
Enter Rate: 2.75
Pay: 96.25

simple answer：

```
hr = raw_input("Enter Hours: ")
rate = raw_input("Enter Hours: ")
print float(hr) * float(rate)
```

Basic Data Structure

Question 1: Finds all the common letters in two strings. Assume that there is no duplicate letter in a string.

```
s1 = "abc"
s2 = "cds"
if len(s1) == len(s2):
      for char1 in s1:
            for char2 in s2:
                  if char1 == char2:
                        print "common letter:", char1
```

Question 2: Given a string with only lower-case letters, determine if it is a palindrome.

For example,

"aabbaa" is a palindrome

"aabbcc" is not a palindrome

```
def isPalindrome(s):
  left, right = 0, len(s)-1
  while left < right:
    if s[left] != s[right]:
      return False
    left +=1
    right -= 1
  return True
```

Question 3: Reverse String. Given s = "hello", return "olleh".

```
def reverse_string(s):
  return s[::-1]
```

Question 4: Given a string *s* consists of upper/lower-case alphabets and empty space characters, return the length of last word in the string.

If the last word does not exist, return 0.

Note: A word is defined as a character sequence consists of non-space characters only.

For example,

Given s = "Hello World",
return 5.

```
def length_of_last_word(s):
        return len(s.split()[len(s.split())-1]) if s.split() != [] else 0
```

Question 5: Count word occurrences.

```
def words_to_frequencies(words):
  myDict = {}
  for word in words:
    if word in myDict:
      myDict[word] += 1
    else:
      myDict[word] = 1
  return myDict
```

Question 6. Most common word

```
def most_common_words(freqs):
  values = freqs.values()
  best = max(values)
  words = []
  for k in freqs:
    if freqs[k] == best:
    words.append(k)
  return (words, best)
```

Homework 1: 合并大小写的频次

```
mydic = {'a': 10, 'b': 34, 'A': 7, 'Z': 3}
→
mydic = {'a': 17, 'b': 34, 'z': 3}
```

```
my_dic = {key_expr: value_expr for value in collection if condition}
```

```
my_dic = {}
for value in collection:
  if condition:
```

```
    my_dic[key_expr] = value_expr

# 常规解法
new_mydic = {}
for k in mydic.keys():
  new_mydic[k.lower()] = mydic.get(k.lower(), 0) + mydic.get(k.upper(), 0)

# Dictionary comprehension
new_mydic = {
  k.lower(): mydic.get(k.lower(), 0) + mydic.get(k.upper(), 0)
  for k in mydic.keys()
}
```

Homework 2: Given an array of integers, return indices of the two numbers such that they add up to a specific target. You may assume that each input would have exactly one solution, and you may not use the same element twice.

Solution 1: O(n^2)
Brute Force.

Solution 2: time complexity: O(n) space complexity O(n)

```
def two_sum(nums, target):
  my_dic = {}
  for i in range(len(nums)):
    if nums[i] in my_dic:
      return (my_dic[nums[i]], i)
    my_dic[target - nums[i]] = i

j  target - nums[j] = nums[i]
```

Solution 3: time complexity: O(n) + O(nlogn)
space complexity: O(1)
相向而行的two pointers

```
# assume numbers is sorted.
def two_sum(numbers, target):
  left = 0
  right = len(numbers) - 1
  while left < right:
    if numbers[left] + numbers[right] < target:
      left += 1
    elif numbers[left] + numbers[right] > target:
      right -= 1
```

```
    else:
      return (left,right)
```

Homework 3: Given an array of integers and an integer k, you need to find the total number of continuous subarrays whose sum equals to k.

sum[i] means 前i个数之和

my_dic: key 代表 sum 和； value 代表：sum和出现的次数

```
def subarray_sum( nums, k):
  ans = sums = 0
  my_dic = {}
  for num in nums:
    if sums not in my_dic:
      my_dic[sums] = 0
    my_dic[sums] += 1
    sums += num
    ans += my_dic.get(sums - k, 0)
  return ans
```

```
sums[i]: sums[i] - k = sums → [j, m, n]
sums[i] - sums[j] = k
```

Binary Search

Q1. Fine target number in a sorted int array. If found, return the index. If not found, return None.

-- to find an element in an array → sorted array

case 1: nums[mid] == target → found!
case 2: nums[mid] < target → If exits, the target should be in the right half. **left = mid + 1**
case 3: nums[mid] > target → f exits, the target should be is in the left half. **right = mid - 1**

```
def binary_search(nums, target):
  left = 0
  right = len(nums) - 1
  while left <= right:
    mid = ( left + right) / 2  # 不用关心溢出问题
    if nums[mid] > target:
      right = mid - 1
    elif nums[mid] < target:
```

```
      left = mid + 1
    else:
      return mid
```

Q2 2D matrix, sorted on each row, first element of next row is larger than the last element of previous row. Now given a target number, return the position of the target number within the matrix.

```
def binary_search_in_2d(matrix, target):
  if matrix == None or len(matrix) == 0:
    return None
  N, M = len(matrix),len(matrix[0])
  left,right = 0,N*M-1
  while left <= right:
    mid = ( left + right) / 2
    row_num = mid / M
    col_num = mid % M
    if matrix[row_num][col_num] > target:
      right = mid - 1
    elif matrix[row_num][col_num] < target:
      left = mid + 1
    else:
      return (row_num,col_num)
```

Time complexity: O(log(N*M))

Q3 Find an element in the array that is closest to the target number. For example, nums = [1,2,5, 9] target = 3

```
def find_closest_num(nums, target):
  if len(nums) == 0:
    return -1
  left = 0
  right = len(nums) - 1
  while left < right - 1:
    mid = ( left + right) / 2
    if nums[mid] > target:
      right = mid
    elif nums[mid] < target:
      left = mid
    else:
      return mid
  return left if abs(nums[left] - target) < abs(nums[right] - target) else right
```

Python的三目运算符： 为真时的结果 if 判定条件 else 为假时的结果

Q4 Find the index of the first occurrence of an element.
Example, nums=[1,2,3,3,5] target = 3, → return 2

case 1: nums[mid] < target → the first occurrence is in the right part. **left = mid**
case 2: nums[mid] > target → the first occurrence is in the left part. **right = mid**
case 3: nums[mid] == target → mid may be the first occurrence but it is still possible to have the first occurrence in the left part. **right = mid**

case 2 and case 3 can be put together.

```
def find_first_occurence(nums, target):
  if len(nums) == 0:
    return -1
  left = 0
  right = len(nums) - 1
  while left < right - 1:
    mid = ( left + right) / 2
    if nums[mid] < target:
      left = mid
    else:
      right = mid
  if nums[left] == target:
    return left
  if nums[right] == target:
    return right
  return -1
```

Q5 Find the index of the last occurrence of an element.

Example, nums=[1,2,3,3,5] target = 3, → return 3

case 1: nums[mid] < target → the first occurrence is in the right part. **left = mid**
case 2: nums[mid] > target → the first occurrence is in the left part. **right = mid**
case 3: nums[mid] == target → mid may be the first occurrence but it is still possible to have the last occurrence in the right part. **left = mid**

Postprocessing is different with Q3!

```
def find_last_occurrence(nums, target):
  if len(nums) == 0:
    return -1
  left = 0
  right = len(nums) - 1
  while left < right - 1:
    mid = ( left + right) / 2
    if nums[mid] > target:
      right = mid
    else:
```

```
      left = mid
  if nums[right] == target:
    return right
  if nums[left] == target:
    return left
  return -1
```

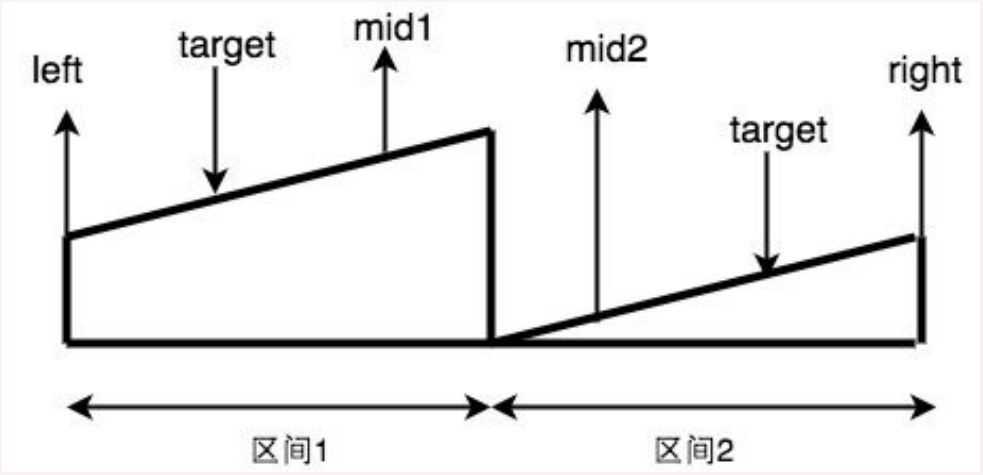
Q6. Find Peak Element. (A peak element is an element that is greater than its two neighbors.)

case 1: `nums[mid] > nums[mid+1] and nums[mid] > nums[mid-1]` → found!
case 2: `nums[mid] < nums[mid+1]` → the peak is in the right part. **left = mid + 1**
case 3: `nums[mid] > nums[mid+1]` → the peak is in the left part. **right = mid - 1**

```
def find_peak_element(nums):
    left = 0
    right = len(nums)-1
    while left < right-1:
        mid = (left+right)/2
        if nums[mid] > nums[mid+1] and nums[mid] > nums[mid-1]:
            return mid
        if nums[mid] < nums[mid+1]:
            left = mid + 1
        else:
            right = mid - 1

    return left if nums[left] >= nums[right] else right
```

Q7. Search in Rotated Sorted Array. You may assume no duplicate exists in the array.

`0 1 2 4 5 6 7` → `4 5 6 7 0 1 2`



Case 1: nums[mid] == target → found
Case 2: nums[left] <= nums[mid] → 说明 mid 落在区间1中,即图中 mid1 的位置。那么，如果 target < nums[mid] and target >= nums[left]，那么继续left和mid中间搜索；否则，在mid和right中间搜索；
Case 3: nums[left] > nums[mid]，说明 mid 落在区间2中，即图中mid2的位置。同理，如果 target > nums[mid] and `target <= nums[right]`, 那么在mid和right间搜索；否则，在left和mid中间搜索。

```
def binary_search_in_sorted_array(nums, target):
  if len(nums) == 0:
    return -1
  left = 0;
  right = len(nums) - 1
  while left < right - 1:
    mid = (left + right) / 2
    if target == nums[mid]:
      return mid
    if nums[mid] >= nums[left]:
      if target < nums[mid] and target >= nums[left]:
        right = mid - 1
      else:
        left = mid + 1
    elif nums[mid] < nums[left]:
      if target > nums[mid] and target <= nums[right]:
        left = mid + 1
      else:
        right = mid - 1
  if nums[left] == target:
    return left
  if nums[right] == target:
    return right
  return -1
```

Homework:
(Leetcode 278) You are a product manager and currently leading a team to develop a new product. Unfortunately, the latest version of your product fails the quality check. Since each version is developed based on the previous version, all the versions after a bad version are also bad.

Suppose you have `n` versions `[1, 2, ..., n]` and you want to find out the first bad one, which causes all the following ones to be bad.

You are given an API `bool isBadVersion(version)` which will return whether `version` is bad. Implement a function to find the first bad version. You should minimize the number of calls to the API.

```
def firstBadVersion(n):
  left = 1
  right = n
  while left + 1 < right:
    mid = (left + right) / 2
    if isBadVersion(mid):
      right = mid
    else:
      left = mid + 1
  if isBadVersion(left):
    return left
  return right
```

Array and Sorting

Q1 merge sort

```
def merge(a,b):
    """ Function to merge two arrays """
    c = []
    index_a = index_b = 0
    while index_a < len(a) and index_b < len(b):
        if a[index_a] < b[index_b]:
            c.append(a[index_a])
            index_a += 1
        else:
            c.append(b[index_b])
            index_b += 1
    if index_a < len(a):
        c.extend(a[index_a:])
    if index_b < len(b):
        c.extend(b[index_b:])
    return c

def merge_sort(x):
    """ Function to sort an array using merge sort algorithm """
    if len(x) == 0 or len(x) == 1:
        return x
    else:
        middle = len(x) / 2
        a = merge_sort(x[:middle])
        b = merge_sort(x[middle:])
```

```
        return merge(a,b)

alist = [2,1,0,3,-1, 5]
print merge_sort(alist)
```

Q1 quick sort

```
from random import randrange

def partition(lst, start, end, pivot):
    lst[pivot], lst[end] = lst[end], lst[pivot]
    store_index = start
    for i in xrange(start, end):
        if lst[i] < lst[end]:
            lst[i], lst[store_index] = lst[store_index], lst[i]
            store_index += 1
    lst[store_index], lst[end] = lst[end], lst[store_index]
    return store_index


def quick_sort(lst, start, end):
    if start >= end:
        return lst
    pivot = randrange(start, end + 1)
    new_pivot = partition(lst, start, end, pivot)
    quick_sort(lst, start, new_pivot - 1)
    quick_sort(lst, new_pivot + 1, end)
```

Q: design a class Student that can be sorted
print sort by age, then sort by grade(reverse)

```
hint: sorted(sorted(arr, key=???), key=???, reverse = True)
```

LinkedList(Queue, Stack) and Recursion I

Q: 逆波兰表达式计算(Reverse Polish notation)
a*(b+c) -> abc+*,

```
import operator
GLOBAL_OPERATE = { "+": operator.add,
          "-": operator.sub,
```

```
        "*": operator.mul,
        "/": operator.div
       }


def cal_rev_polish(alist):
    num_stack = Stack()
    for x in alist:
        if x not in operate:
            num_stack.push(x)
        else:
            right = num_stack.pop()
            left = num_stack.pop()
            num_stack.push(GLOBAL_OPERATE[x](left, right))
    return num_stack.pop()

print neg_polish([1,2,3,"+","-"])
```

Class 12: Binary Tree and Binary Search Tree

1. Invert Binary Tree

Invert a binary tree.

```
    4
  /   \
 2     7
/ \   / \
1  3 6   9
```

to

```
    4
  /   \
 7     2
/ \   / \
9  6 3   1
```

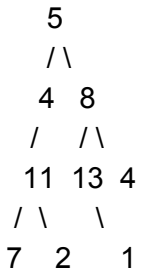
```
def invert(root):
      if root:
            invert = self.invertTree
            root.left, root.right = invert(root.right), invert(root.left)
            return root
```

2. Path Sum

Given a binary tree and a sum, determine if the tree has a root-to-leaf path such that adding up all the values along the path equals the given sum.

For example:

Given the below binary tree and sum = 22,



return true, as there exist a root-to-leaf path 5->4->11->2 which sum is 22.

```
def hasPathSum(root, sum):
        if not root:
            return False

        if not root.left and not root.right and root.val == sum:
            return True

        sum -= root.val

        return hasPathSum(root.left, sum) or hasPathSum(root.right, sum)
```

Class 15: Stack and Heap

Q: Design a min heap

```
Class MinHeap():

        def __init__(self):
                self.size = 0
                self.arr = []

        def push(self, value):
                self.arr.append(value)
                size += 1
                self.percolate_up(size - 1)

        def pop(self, value):
                self.arr[0], self.arr[size - 1] = self.arr[size - 1], self.arr[0]
                self.percolate_down(0)
                return_val = arr[size - 1]
```

```
            self.arr.remove(size - 1)
            size -= 1
            return return_val

    def percolate_up(self, index):
        while index > 0:
            parent_ind = (index - 1) / 2
            if  array[index] < array[parent_ind]:
                array[index], array[parent_ind] = array[parent_ind], array[index]
                index = paren_ind
            else:
                break

    def percolate_down(self, index):
        while index < size - 1:
            lchild_ind = 2 * index + 1
            rchild_ind = 2 * index + 2
            if arr[lchild_ind] > arr[index] and arr[rchild_ind] > arr[index]:
                break
            elif arr[lchild_ind] < arr[rchild_ind]:
                arr[index], arr[lchild_ind] = arr[lchild_ind], arr[index]
                index = lchild_ind
            else:
                arr[index], arr[rchild_ind] = arr[rchild_ind], arr[index]
                index = rchild_ind
```

Class 16 - Graph Search Algorithms

Iterative method based DFS:

```
def dfs(graph, start):
  visited, stack = set(), [start]
  visit_path = []
  while stack:
    vertex = stack.pop()
    if vertex not in visited:
      visited.add(vertex)
      visit_path.append(vertex)
      stack.extend(graph[vertex] - visited )
  return visit_path

print dfs(graph, 0)
```

Recursive method based DFS:

```
def dfs(graph, start, visit_path = [], visited=None):
  if visited is None:
    visited = set()
  visited.add(start)
  for vertex in graph[start]:
    if vertex not in visited:
      visit_path.append(vertex)
      dfs(graph, vertex, visit_path, visited)
  return visit_path

print dfs(graph, 0, [0])
```

Breadth First Search (BFS)

```
def bfs(graph, start):
  visited, queue = set(), [start]
  visit_path = []
  while queue:
    vertex = queue.pop(0)
    if vertex not in visited:
      visited.add(vertex)
      visit_path.append(vertex)
      queue.extend(graph[vertex] - visited)
  return visit_path

print bfs(graph, 0)
```

How to find the shortest path from start to end?

```
def bfs_shortest_path(graph, start_vertex, end_vertex):
  queue = [(start_vertex, [start_vertex])]
  visited = set()
  while queue:
    (vertex, path) = queue.pop(0)
    if vertex not in visited:
      for next in graph[vertex] - visited:
        if next == end_vertex:
          return path + [next]
        queue.append((next, path + [next]))
    visited.add(vertex)
```

```
  return None

print bfs_shortest_path(graph, 0, 3)
```

Question: check if there is a path with length <= K between two nodes in a graph. The length of the path is the number of edges on the path.

```
 def bfs_has_path_less_than_k(graph, start_vertex, end_vertex, k):
  queue1 = [start_vertex]
  queue2 = []
  visited = set()
  while ( queue1 or queue2) and k >= 0:
    while queue1 and k >= 0:
      vertex = queue1.pop(0)
      if vertex not in visited:
        if vertex == end_vertex:
          return True
        for next in graph[vertex] - visited:
          queue2.append(next)
      visited.add(vertex)
    k = k - 1
    while queue2 and k >= 0:
      vertex = queue2.pop(0)
      if vertex not in visited:
        if vertex == end_vertex:
          return True
        for next in graph[vertex] - visited:
          queue1.append(next)
      visited.add(vertex)
    k = k - 1
  return False


 def bfs_has_path_less_than_k(graph, start_vertex, end_vertex, k):
  queue = [start_vertex]
  visited = set()
  while queue and k >= 0:
    n = len(queue)
    while n:
      n = n - 1
      vertex = queue.pop(0)
      if vertex not in visited:
        if vertex == end_vertex:
```

```
          return True
        for next in graph[vertex] - visited:
          queue.append(next)
      visited.add(vertex)
    k = k - 1

print bfs_has_path_less_than_k(graph, 0, 3, 2)
```

Space complexity: O(V)
Time complexity: O(V+E)

Homework: write a function to check if an undirected graph is connected or not. Follow up: how about directed graph?

For undirected graph:

```
def is_connected(graph):
  start = 0 # start from an arbitrary vertex
  visited, queue = set(), [start]
  while queue:
    vertex = queue.pop(0)
    if vertex not in visited:
      visited.add(vertex)
      queue.extend(graph[vertex] - visited)
  return len(visited) == len(graph)

print is_connected(graph)
```

For directed graph:

1) Do a DFS/BFS traversal of graph starting from any arbitrary vertex v. If DFS/BFS traversal doesn't visit all vertices, then return false.
2) Reverse all edges
3) Do a DFS/BFS traversal of graph starting from vertex v. If DFS/BFS traversal doesn't visit all vertices, then return false.Otherwise return true.

```
def reverse_edges(graph):
  new_graph = {}
  for (start_vertex, end_vertices) in graph.items():
    for end_vertex in end_vertices:
      if end_vertex not in new_graph:
        new_graph[end_vertex] = [start_vertex]
      else:
```

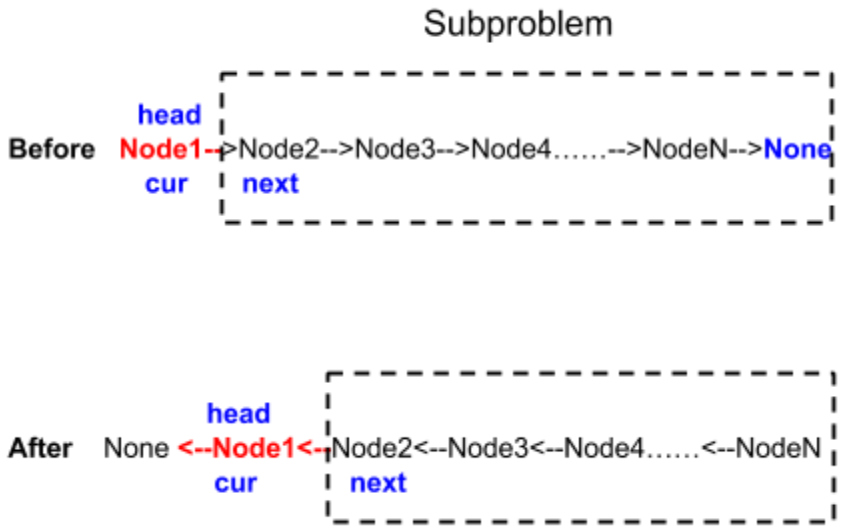
```
            new_graph[end_vertex].append(start_vertex)
    return new_graph

print reverse_edges(graph)
```

Class 19 - Python Coding - Recursion II

Q3. Reverse a linked list.

Node1->Node2-->Node3-->Node4……-->NodeN-->**NULL**
head

reversed: NULL <--Node1<-Node2<--Node3<--Node4……<--NodeN
head



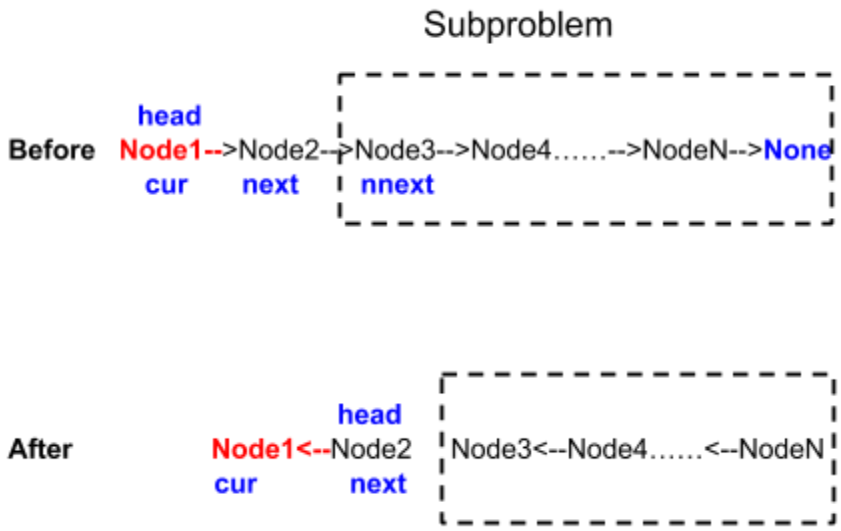
Base case: head == None or head.next == None
Recursion rule: 将当前剩下node作为一个subproblem来解决。subproblem中的tail是next。
next.next = cur
cur.next = None

```
class ListNode(object):
  def __init__(self, x):
    self.val = x
    self.next = None
```

```python
# head is a ListNode object; the following function return a ListNode object
def reverse_list(head):
  if not head or not head.next:
    return head
  next_node = head.next
  new_head = reverse_list(head.next)
  next_node.next = head
  head.next = None
  return new_head
```

Q4. Reverse a linked list (pair by pair).

Example: 1→ 2→ 3→ 4→ 5→ null
Output: 2→ 1→ 4→ 3→ 5→ null




```python
def reverse_list_in_pair(head):
  # base case
  if not head or not head.next:
    return head
  # recursion rule
  next_node = head.next
  head.next = reverse_list_in_pair(head.next.next)
  next_node.next = head
  return next_node
```

Q5. (从下往上返值) GetHeight of a binary tree

```
def get_height(root):
      if not root:
            return 0
      left = get_height(root.left)
      right = get_height(root.right)
      return 1 + max(left, right)
```

Q6. (从下往上返值) For each node in a tree, store the number of nodes in its left child subtree.

```
def find_left_total(root):
  if not root:
    return 0
  left_total = find_left_total(root.left)
  right_total = find_left_total(root.right)
  root.total_left = left_total
  return left_total + right_total + 1
```

Q7. (从下往上返值) Find the node with the max difference in the total number of descendents in its left subtree and right subtree

```
class TreeNode:
  def __init__(self, x):
    self.val = x
    self.left = None
    self.right = None

class ResultWrapper:
  def __init__(self):
    self.global_max = -1
    self.solution = None

def max_diff_node(root, res):
  if not root:
    return 0
  left_toal = max_diff_node(root.left, res)
  right_toal = max_diff_node(root.right, res)
  if abs(left_toal - right_toal) > res.global_max:
    res.global_max = abs(left_toal - right_toal)
    res.solution = root
  return left_toal + right_toal + 1

res = ResultWrapper()
root = TreeNode(0)
root.left = TreeNode(1)
```

```
root.left.left = TreeNode(3)
root.left.right = TreeNode(4)
root.right = TreeNode(2)
root.right.right = TreeNode(5)
root.right.right.right = TreeNode(6)

max_diff_node(root, res)
print res.solution.val
```

Q8. (从下往上返值) Lowest Common Ancestor (LCA)

```
def lowestCommonAncestor( root, p, q):
  if not root:
    return None
  if root == p or root == q:
    return root
  left = lowestCommonAncestor(root.left, p, q)
  right = lowestCommonAncestor(root.right, p, q)
  if left and right:
    return root
  return left if left else right
```

Class 24 Python Coding - Recursion III

DFS经典例题1 Print all subsets of a set S = {'a', 'b', 'c'}

```
def find_sub_set(input, index, solution):
  # base case. print solution
  if index == len(input):
    if len(solution) == 0:
      print "empty set"
    else:
      print solution
    return

  # recursion rules:
  # case 1: Add input[index] to subset.
  solution.append(input[index])
  find_sub_set(input, index + 1, solution)
  solution.pop()
  # case 2: add nothing
  find_sub_set(input, index + 1, solution)
```

```
find_sub_set(['a','b','c'], 0, [])
```

Time = O(2^n)

DFS经典例题2 ()()() find all valid permutation using the parenthesis provided.

Restriction here is whenever we insert a new ")" we need to make sure the number of "(" **added so far** is larger than the number of ")" added so far

```
def all_valid_permutation(n, l, r, solu_prefix):
  # base case
  if l == n and r == n:
    print solu_prefix
    return
  # recursion rules
  # case 1: add '(' on this level
  if l < n:
    solu_prefix.append('(')
    all_valid_permutation(n, l + 1, r, solu_prefix)
    solu_prefix.pop()

  # case 2: add ')' on this level
  if r < l:
    solu_prefix.append(')')
    all_valid_permutation(n, l, r+1, solu_prefix)
    solu_prefix.pop();

all_valid_permutation(2,0,0,[])
```

DFS经典例题3 Print all combinations of coins that can sum up to a total value k.
E.g. total value k = 99 cents
coin value(币值) = 25 10 5 1 cent

```
e.g. one possible way:
3 x 25 cents
2 x 10 cents
0 x 5 cents
4 x 1 cents
```

```
def find_combination(coins, money_left, level, solution ):
```

```
  # base case
  if level == 3:
    solution.append(money_left)
    print solution
    solution.pop()
    return
  # recursion rules
  for i in range(money_left/coins[level] + 1):
    solution.append(i)
    find_combination(coins, money_left - i * coins[level], level+1, solution)
    solution.pop()

coins = [25, 10, 5, 1]

find_combination(coins, 99, 0, [])
```

DFS经典例题4 . Given a string **with no duplicated letters**, how to print out all permutations of the string.

```
def permutation(input, index):
  # base case
  if index == len(input):
    print input
    return
  # recursion rules
  for i in range(index, len(input)):
    input[index], input[i] = input[i], input[index]
    permutation(input, index + 1);
    input[index], input[i] = input[i], input[index]

permutation(['a','b','c'], 0)
```

Class 26 - Python Coding -String

Q: Minimum Window Substring,给定两个字符串S和T，找到S中最短的子串S',使得S'包含T中所有char T = aab S = aaacbc S' aacb

```
def solve(s,t):
  dict = {}
  for ch in t:
    if ch in dict:
      dict[ch] += 1
```

```
    else:
      dict[ch] = 1
  start = 0
  end = 0
  valid_used = len(t)
  res_start = 0
  res_len = len(s)
  while end < len(s):
    if s[end] in dict:
      dict[s[end]] -= 1
      if dict[s[end]] >= 0:
        valid_used -= 1
    while valid_used == 0:
      if end - start < res_len:
        res_len = end - start
        res_start = start
      if s[start] in dict:
        dict[s[start]] += 1
        if dict[s[start]] > 0:
          valid_used += 1
      start += 1
    end += 1
  print s[res_start:res_start + res_len + 1]
```

solve("aaacbc", "aab")

Class 27 Python Coding - HashTable and String I

Longest Common Prefix

Write a function to find the longest common prefix string amongst an array of strings

```
def longest_common_prefix(strs):
      if not strs:
            return ""
        shortest = min(strs,key=len)
        for i, ch in enumerate(shortest):
            for other in strs:
                if other[i] != ch:
                    return shortest[:i]
        return shortest
```

Length of Last Word

Given a string *s* consists of upper/lower-case alphabets and empty space characters `' '`, return the length of last word in the string.

If the last word does not exist, return 0.

```
def length_of_last_word(str):
      n = len(str)
      lastLength = 0
      while n > 0 and str[n - 1]  == ' ':
            n -= 1
      while n > 0 and str[n - 1] != ' ':
            n -= 1
            lastLength += 1
      return lastLength
```

Class 32 - Python Coding - Dynamic Programming

Climbing Stairs

You are climbing a stair case. It takes n steps to reach to the top.
Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

Coin Change

You are given coins of different denominations and a total amount of money amount. Write a function to compute the fewest number of coins that you need to make up that amount. If that amount of money cannot be made up by any combination of the coins, return -1.
Example 1:
coins = [1, 2, 5], amount = 11
return 3 (11 = 5 + 5 + 1)
Example 2:
coins = [2], amount = 3
return -1.

Decode Ways

A message containing letters from A-Z is being encoded to numbers using the following mapping:
'A' -> 1
'B' -> 2

...
'Z' -> 26

Given an encoded message containing digits, determine the total number of ways to decode it.
For example,
Given encoded message "12", it could be decoded as "AB" (1 2) or "L" (12).
The number of ways decoding "12" is 2.

```
def numDecodings(s):
        #dp[i] = dp[i-1] if s[i] != "0"
        #       +dp[i-2] if "09" < s[i-1:i+1] < "27"
        if s == "": return 0
        dp = [0 for x in range(len(s)+1)]
        dp[0] = 1
        for i in range(1, len(s)+1):
            if s[i-1] != "0":
                dp[i] += dp[i-1]
            if i != 1 and s[i-2:i] < "27" and s[i-2:i] > "09":  #"01"ways = 0
                dp[i] += dp[i-2]
        return dp[len(s)]
```