给定两个字符串数组，找出最长的相同subarray

Longest Common Substring problem

```
def lcs(S,T):
    m = len(S)
    n = len(T)
    counter = [[0]*(n+1) for x in range(m+1)]
    longest = 0
    lcs_set = set()
    for i in range(m):
        for j in range(n):
            if S[i] == T[j]:
                c = counter[i][j] + 1
                counter[i+1][j+1] = c
                if c > longest:
                    lcs_set = set()
                    longest = c
                    lcs_set.add(S[i-c+1:i+1])
                elif c == longest:
                    lcs_set.add(S[i-c+1:i+1])
    return lcs_set
```

https://www.bogotobogo.com/python/python_longest_common_substring_lcs_algorithm_generalized_suffix_tree.php

1. 一个矩阵，只有0和1，找到里面全为1的矩形的坐标。输入一定有效，保证有一个满足要求的矩形。用左上和右下坐标表示
比如：
0 0 0 0 0
0 1 1 0 0
0 0 0 0 0
结果就是返回
[1,1], [1,2]
2. follow up 有很多个这样的矩形， 返回所有的矩形的左上右下坐标
3. 不一定是矩形，找出所有连通的1.
1 0 0 1 1
0 0 0 1 1
1 0 0 0 1
这样的输入，返回一个大数组
[
[0,0],
[[0,3], [0,4], [1,3], [1,4], [2,4]],
[2,1]
]

Longest Word in Dictionary

Given a list of strings `words` representing an English Dictionary, find the longest word in `words` that can be built one character at a time by other words in `words`. If there is more than one possible answer, return the longest word with the smallest lexicographical order.

```
def longestWord(self, words):
    # Step 1: Construct a trie
    root = {"#":"#"} # root of the trie
    # Add words one by one
    for word in words:
        t = root
        for char in word:
            if char not in t:
                t[char] = {}
            t = t[char]
        t['#'] = '#' # Mark end of the word

    # Step 2: Find longest word with DFS
    def dfs(node, word_so_far):
        global longest_word
        if node == '#':
            return
       # Visit node only if a word ends here
        if "#" in node:
            # compare length and lexicographical order
            if len(longest_word) < len(word_so_far) or (len(longest_word) == len(word_so_far)
                                                        and word_so_far < longest_word):
                longest_word = word_so_far
            # visit all children
            for next_char in node:
                dfs(node[next_char], word_so_far+next_char)

    global longest_word
    longest_word = ''
    dfs(root, '')
    return longest_word
```

You and your friend are going to hang out together, but you are all very busy. Your friends tell you their busy times, you need to design a function to find a spare time which could give all the spare time for everyone.

输入输出格式什么的，全都自己定义。
我先问了一下，这个hang out是一天的还是几天的。然后说，先假设所有的time都是同一天的。
我先写了一个小函数，在输入都是integer的情况下，格式是 [[1,4],[2,5],[7,8]] 这样，排序之后，用一个变量keep latest busy time，如果新的busy time 大于已选 timeslots 的 latest busy time，就找到一段空闲。
然后她问有没有需要改进的地方，我就写了time conversion的函数，可以take 9:40, 11:20这样的输入，转化成整数，再进行操作。
又问了一下时间复杂度，能不能优化等。

题目很容易，就是有挺多细节需要注意的。比如输出的时候，如果是 9h 4min，要输出 9:04 而不是 9:4。比如第一个busy 时间是10:00开始busy 的，那10:00之前也要算上空闲。还有edge cases，如果大家都没有busy要输出全部时间等。做的时候要非常细心。。。

```
def spare_time(time_lst):
    ground = set(range(24))
    for time in time_lst:
        time = set(range(time[0],time[1]))
        ground -= time

    ground = list(ground)
    stack = []
    result = []
    for t in ground:
        if stack == [] or t == stack[-1] + 1:
            stack.append(t)
        else:
            result.append([stack[0],stack[-1]])
            stack = [t]
    result.append([stack[0],stack[-1]])
    return result
```

矩阵[[1,2,3],[4,5,6],[7,8,9]]输出 [[1],[2,4],[3,5,7],[6,8],[9]]和[[7],[4,8],[1,5,9],[2,6],[3]] https://leetcode.com/problems/diagonal-traverse/description/

```
def diagonal(matrix):
    n_rows, n_cols = len(matrix), len(matrix[0])
    i, j = 0, 0
    result = []
    c_i, c_j = 0, 0
    output = []
    while i != n_rows and j != n_cols:
        result.append(matrix[i][j])
        if i == n_rows-1:
            c_i += 1
            i = c_i
            j = c_j
            output.append(result)
            result = []
            continue
        if j == 0:
            c_j += 1
            i = c_i
            j = c_j
            output.append(result)
            result = []
            continue
        i += 1
        j -= 1
    return output

def inverse_diagonal(matrix):
    n_rows, n_cols = len(matrix), len(matrix[0])
    i, j = 2, 0
    c_i, c_j = 2, 0
    result, output = [], []
    while i != -1 and j != n_cols:
        result.append(matrix[i][j])
        if j == n_cols - 1:
            c_j += 1
            i = c_i
            j = c_j
            output.append(result)
            result = []
            continue
        if i == n_rows - 1:
            c_i -= 1
            i = c_i
            j = c_j
            output.append(result)
            result = []
            continue
        i += 1
        j += 1
    return output
```

Design Hit Counter (Leetcode 362)

Design a hit counter which counts the number of hits received in the past 5 minutes.
Each function accepts a timestamp parameter (in seconds granularity) and you may assume that calls are being made to the system in chronological order (ie, the timestamp is monotonically increasing). You may assume that the earliest timestamp starts at 1.
It is possible that several hits arrive roughly at the same time.

```
from collections import deque


class HitCounter(object):
    windowLen = 300

    def __init__(self):
        """
        Initialize your data structure here.
        """
        self.hitQueue = deque() # each item is a pair [timeStamp,hitCount] where hitCount is the number of hits at timeStamp
        self.hitCountInWindow = 0

    def _removeOldHits(self, timestamp):
        while self.hitQueue and self.hitQueue[0][0] <= timestamp - self.windowLen:
            self.hitCountInWindow -= self.hitQueue.popleft()[1]

    def hit(self, timestamp):
        """
        Record a hit.
        @param timestamp - The current timestamp (in seconds granularity).
        :type timestamp: int
        :rtype: void
        """
        if not ( self.hitQueue and self.hitQueue[-1][0]==timestamp ):
            self.hitQueue.append( [timestamp,0] )

        self.hitQueue[-1][1] += 1
        self.hitCountInWindow += 1

    def getHits(self, timestamp):
        """
        Return the number of hits in the past 5 minutes.
        @param timestamp - The current timestamp (in seconds granularity).
        :type timestamp: int
        :rtype: int
        """
        self._removeOldHits(timestamp)
        return self.hitCountInWindow



# Your HitCounter object will be instantiated and called as such:
# obj = HitCounter()
# obj.hit(timestamp)
# param_2 = obj.getHits(timestamp)
```

题目是给一个char矩阵和一个单词，找这个单词是否在矩阵中出现。follow up是这个的第二题。

```
def find_word(word, board):
    letter_positions = {}
    for (y, row) in enumerate(board):
        for (x, letter) in enumerate(row):
            letter_positions.setdefault(letter, []).append((x, y))

    def move_valid(position, last_position):
        if last_position is None or last_position == []:
            return True
        return (
            abs(position[0] - last_position[0]) <= 1 and
            abs(position[1] - last_position[1]) <= 1
        )

    def helper(word, used=None):
        if word == "":
            return []
        if used is None:
            used = []
        letter, rest = word[:1], word[1:]
        for position in letter_positions.get(letter) or []:
            if position in used:
                continue
            if not move_valid(position, used and used[-1]):
                continue
            path = helper(rest, used + [position])
            if path is not None:
                return [position] + path
        return None

    return True if helper(word) else False
```

Word Ladder (Leetcode 127)

Given two words (*beginWord* and *endWord*), and a dictionary's word list, find the length of shortest transformation sequence from *beginWord* to *endWord*, such that:

1. Only one letter can be changed at a time.
2. Each transformed word must exist in the word list. Note that *beginWord* is *not* a transformed word.

```
class Solution:
    def ladderLength(self, beginWord, endWord, wordList):
        visited = set()
        wordSet = set(wordList)

        queue = [(beginWord, 1)]

        while len(queue) > 0:
            word, count = queue.pop(0)
            if word == endWord:
                return count
            if word in visited:
                continue

            for i in range(len(word)):
                for j in range(0, 26): # try all possible one character permutations
                    char = ord('a') + j
                    changed_word = word[0:i] + chr(char) + word[i + 1:]
                    if changed_word in wordSet: # if permuted word is in word list then add children
                        queue.append((changed_word, count + 1))
            visited.add(word) # mark word as visited

        return 0 # if queue is exhausted and code reachers here then its impossible to reach endWord
```

Is Graph Bipartite?

Given an undirected `graph`, return `true` if and only if it is bipartite.
Recall that a graph is *bipartite* if we can split it's set of nodes into two independent subsets A and B such that every edge in the graph has one node in A and another node in B.
The graph is given in the following form: `graph[i]` is a list of indexes `j` for which the edge between nodes `i` and `j` exists. Each node is an integer between `0` and `graph.length - 1`. There are no self edges or parallel edges: `graph[i]` does not contain `i`, and it doesn't contain any element twice.

The graph is described as incident list.

```
def isBipartite(graph):  # graph is described by incident lists.
    n = len(graph)
    color = [0] * n

    def dfs(u, c):
        color[u] = c
        for v in graph[u]:
            if color[v] == c:
                return False
            elif not color[v] and not dfs(v, -c):
                return False
        return True


    for i in range(n):
        if not color[i]:
            if not dfs(i,1):
                return False
    return True

def convert_edge_list_to_incident_list(edge_list):
    incident_list = {}
    for a, b in edge_list:
        incident_list.setdefault(a, set([b])).add(b)
        incident_list.setdefault(b, set([a])).add(a)
    n = max(incident_list.keys())
    graph = [[0]] * (n+1)
    for key in incident_list:
        graph[key] = list(incident_list[key])
    return graph
```

Leetcode 157&158

```
'''
Read N Characters Given Read4
============================

The API: int read4(char *buf) reads 4 characters at a time
from a file.

The return value is the actual number of characters read.
For example, it
returns 3 if there is only 3 characters left in the file.

By using the read4 API, implement the function int
read(char *buf, int n) that
reads n characters from the file.

Note:
The read function will only be called once for each test
case.
'''
class Solution(object):
    def read(self, buf, n):
        count, readed = 0, [''] * 4
        while count < n:
           size = read4(readed)
           if not size:
                break
            size = min(size, n - count)
            for i in xrange(size):
                buf[count] = readed[i]
                count += 1
        return count
```

```
'''
Read N Characters Given Read4 II - Call multiple times
=======================================================
Note:
The read function may be called multiple times.
'''


class Solution(object):
    '''算法思路:

    要将上次 read4 读取后剩余的保存起来
    '''
    def __init__(self):
        self.left = []

    def read(self, buf, n):
        count, reads = 0, [''] * 4

        if self.left:
            while count < min(n, len(self.left)):
                buf[count] = self.left[count]
                count += 1

            self.left = self.left[count:]

        while count < n:
            size = read4(reads)
            if not size:
                break

            length = min(n - count, size)
            self.left = reads[length : size]

            for i in xrange(length):
                buf[count] = reads[i]
                count += 1

        return count


index = 0
string = 'abcdefg'

def read4(reads):
    global index

    count = 0
    while count < 4 and index < len(string):
        reads[count] = string[index]
        count += 1
        index += 1
    return count


s = Solution()
buf = [0] * 100
print s.read(buf, 3)
print s.read(buf, 4)
print s.read(buf, 3)
```

A log System with structure like this
ex.

```
|---register_button (10)
|    |---register_email (4)
|    |    |---email_already_exists (1)
|    |    |---register_success (3)
|    |---register_facebook (4)
|    |    |---register_success (4)
|    |---dropoff (2)
|---login_button (10)
|    |---login_email (4)
|    |    |---login_success (4)
|    |---login_facebook (4)
|    |    |---login_success (3)
|    |    |---login_failure (1)
|    |---dropoff (2)
```

now we have data with 3 properties

ex with A, B, C

user_id, timestamp, action
100, 1000, A
200, 1003, A
300, 1009, B
100, 1026, B
100, 1030, C
200, 1109, B
200, 1503, A

We want to output a graph to visualize it

Graph from input:

```
|---A (2)
|    |---B (2)
|    |    |---C (1)
|    |    |---A (1)
|---B (1)
```

1. define data structure and classes
2. construct the graph from logfile
3. print out the graph similar to above

Maximum product of 4 adjacent elements in matrix

Given a square matrix, find the maximum product of four adjacent elements of matrix. The adjacent elements of matrix can be top, down, left, right, diagonal or anti diagonal. The four or more numbers should be adjacent to each other.

Note: n should be greater than or equal to 4 i.e n >= 4

```
#Project Euler Problem 11

g = [map(int, s.split()) for s in open('pe11.in').readlines()]

maxp = 0
rows, cols, path_size = len(g), len(g[0]), 4
for i in range(rows):
    for j in range(cols - path_size + 1):
        phv = max(g[i][j] * g[i][j+1] * g[i][j+2] * g[i][j+3],
                  g[j][i] * g[j+1][i] * g[j+2][i] * g[j+3][i])
        if i < rows - path_size:
           pdd = max(g[i][j] * g[i+1][j+1] * g[i+2][j+2] * g[i+3][j+3],
                     g[i][j+3] * g[i+1][j+2] * g[i+2][j+1] * g[i+3][j])
        maxp = max(maxp, phv, pdd)

print "Greatest product of", path_size, "adjacent numbers:", maxp
```

给一个由单个字符串表示的log string, 每一行都是（PIN，BOARD, SEARCH). 将三个操作看做一个sequence，比如（P, B, S), 要求返回一个log file里面频率最高的sequence

hashmap + queue 秒了。

coding的follow up说是如果有很多很多这样的log files, 内存盛不下，我现在想找所有Logs里面最Popular的sequence， 应该怎么做。答，用map reduce。每个machine尽可能多的处理files, mapping 的时候emit 所有locally 最好的 (sequence, count) pairs， reduce的时候把sequence作为key来reduce。

```
def find_popular_sequence_0(s_lst):
    from collections import Counter
    c = Counter(s_lst)
    return c.most_common()[0][0]

def find_popular_sequence_1(s_lst):
    d = {}
    m_val, m_s = 0, ''
    for s in s_lst:
        if s not in d:
            d[s] = 1
        else:
            d[s] += 1
        if d[s] > m_val:
            m_val = d[s]
            m_s = s
    return m_s
```

A - 面试官
B - 我

A： 观察到了什么?
B：很多图片，宽度一样，长度不一样

A：在不同的屏幕（大小不同）上这些图片的安排位置怎么变化?
A：如何把他们安排的好看？或者说给定column数字，和list of pins(图片), 如何安排的好看?
B：好看的定义是?
A：按顺序处理图片，总是想把当前的图片放在 最短的那个column上

B：输入格式是？ 输出格式是?
A：输入pins = [{'id':1, 'height': 100}, {'id':2, 'height':300}, {'id':3, 'height':150}.....], col = 3
返回 [[1,.....],[2,...],[3,...]] list of list。 每个list含有一个col安排的图片的编号
B：minheap O(nlogk) n:number of pins k: col

A：如果不让用heap 你会用什么?
B：array + binarysearch

A：如果连binarysearch都不用， 给一个最最基础的
B：O(nk) 每次扫描所有col得到最小的index
A：什么时候我们想用 O(nk) 的方法
B：k 很小， logk, k 都是常数级别

这算是第一问吧。写完之后继续回到刚刚的网页。

A：你resize一下看看发生了什么
B：恩，位置发生了变化 而且还有特效， 很炫酷
A：如果是你 你怎么实现它。

3 sum

> 类型：固定区间斗鸡眼
> Time Complexity O(n^2)
> Space Complexity O(1)

```
class Solution(object):
    def threeSum(self, nums):
        if len(nums) < 3:
            return []
        nums.sort()
        res = []
        for i in range(len(nums) - 2):
            if i > 0 and nums[i-1] == nums[i]: continue
            l, r = i + 1, len(nums) - 1
            while l < r:
                s = nums[i] + nums[l] + nums[r]
                if s == 0:
                    res.append([nums[i], nums[l], nums[r]])
                    l += 1; r -= 1
                    while l < r and nums[l] == nums[l-1]: l += 1
                    while l < r and nums[r] == nums[r+1] : r -= 1
                elif s < 0:
                    l += 1
                else:
                   r -= 1
        return res
```

4 sum

```
def fourSum(self, nums, target):
    def findNsum(nums, target, N, result, results):
        if len(nums) < N or N < 2 or target < nums[0]*N or target > nums[-1]*N:  return
        if N == 2: # two pointers solve sorted 2-sum problem
            l,r = 0,len(nums)-1
            while l < r:
                s = nums[l] + nums[r]
                if s == target:
                    results.append(result + [nums[l], nums[r]])
                    l += 1
                    while l < r and nums[l] == nums[l-1]:
                        l += 1
                elif s < target:
                    l += 1
                else:
                    r -= 1
        else: # recursively reduce N
            for i in range(len(nums)-N+1):
                if i == 0 or (i > 0 and nums[i-1] != nums[i]):
                    findNsum(nums[i+1:], target-nums[i], N-1, result+[nums[i]], results)

    results = []
    findNsum(sorted(nums), target, 4, [], results)
    return results
```

Longest Increasing Subsequence

Given an unsorted array of integers, find the length of longest increasing subsequence.

Example:

Input: [10,9,2,5,3,7,101,18]
Output: 4
Explanation: The longest increasing subsequence is [2,3,7,101], therefore the length is 4.

```
class Solution:
    def lengthOfLIS(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        if not nums:
            return 0
        dp = [1] * len(nums)
        for i in range(1,len(nums)):
            dp[i] = max([dp[x] for x in range(i) if nums[x] < nums[i]]+[0]) + 1
        return max(dp)
```

```
# O(nlogn) solution with binary search
def lengthOfLIS(self, nums):
        def binarySearch(sub, val):
            lo, hi = 0, len(sub)-1
            while(lo <= hi):
                mid = lo + (hi - lo)//2
                if sub[mid] < val:
                    lo = mid + 1
                elif val < sub[mid]:
                    hi = mid - 1
                else:
                    return mid
            return lo
        sub = []
        for val in nums:
            pos = binarySearch(sub, val)
            if pos == len(sub):
                sub.append(val)
            else:
                sub[pos] = val
        return len(sub)
```

Diameter of Binary Tree

Given a binary tree, you need to compute the length of the diameter of the tree. The diameter of a binary tree is the length of the **longest** path between any two nodes in a tree. This path may or may not pass through the root.

```
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
# time complexity: O(V+E)
# space complexity: O(bm) b: branch m: maximum length of any path
class Solution(object):
    def diameterOfBinaryTree(self, root):
        """
        :type root: TreeNode
        :rtype: int
        """
        self.ans = 0

        def depth(p):
            if not p: return 0
            left, right = depth(p.left), depth(p.right)
            self.ans = max(self.ans, left+right)
            return 1 + max(left, right)

        depth(root)
        return self.ans
```

刚面碗求一个sorted array, 返回任意一个出现次数大于1/4的元素。
答案是在 0，1/4, 2/4, 3/4. 出向两边进行 binary search

```
#  0 1/4 2/4 3/4
def find_1_4(nums):  # nums is sorted
    def find_start_index(nums, l, r):  # binary search
        target = nums[r]
        if nums[l] == nums[r]:
            return l
        while l < r:
            mid = l + (r-l)//2
            if nums[mid] < target:
                l = mid + 1
            else:
                r = mid
        return -1

    if not nums:
        return None

    for i in range(0,3):
        index = find_start_index(nums, (len(nums)/4*i), len(nums)/4*(i+1))
        if index != -1 and index+len(nums)/4 <= len(nums)-1:
            if nums[index+len(nums)/4] == nums[index]:
                return nums[index]

    return None
```

Leetcode binary-tree-longest-consecutive-sequence

```
# Time:  O(n)
# Space: O(h)
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution(object):
    def longestConsecutive(self, root):
        """
        :type root: TreeNode
        :rtype: int
        """
        self.max_len = 0

        def longestConsecutiveHelper(root):
            if not root:
                return 0

            left_len = longestConsecutiveHelper(root.left)
            right_len = longestConsecutiveHelper(root.right)

            cur_len = 1
            if root.left and root.left.val == root.val + 1:
                cur_len = max(cur_len, left_len + 1)
            if root.right and root.right.val == root.val + 1:
                cur_len = max(cur_len, right_len + 1)

            self.max_len = max(self.max_len, cur_len, left_len, right_len)

            return cur_len

        longestConsecutiveHelper(root)
        return self.max_len
```

1. English words array ["Alice", "Bob", "Tom"] -> "Alice Bob and Tom" 要求Codepad 跑test case，我之前看过这题，虽然没练。没写出虫子一次过了
2. Follow up: Array and number 2 -> "Alice Bob and 1 more" 1 -> "Alice and 2 more"

```
def english_words_array_1(s_lst):
    if not s_lst:
        return
    if len(s_lst) == 1:
        return s_lst[0]
    return ' '.join(s_lst[:-1]) + ' and ' + s_lst[-1]

def english_words_array_2(s_lst, num):
    if not s_lst:
        return
    if len(s_lst) <= num:
        return ' '.join(s_lst[:-1]) + ' and ' + s_lst[-1]
    else:
        return ' '.join(s_lst[:num]) + ' and {} more'.format(len(s_lst)-num)
```

第一题： 给一个string，把里面的小写字母转成大写字母，类似于combination： abC -> AbC, aBC, abC, ABC. 用了backtrace，跑了一下没啥问题。
第二题： 设计一个like按钮，问db里面存储结构应该如何设计，怎样提高performance。
第三题： 设计一个api 用来平衡bst。

第二题：找int[][] matrix里的用0表示的box的范围。范围可以用左上，右下或者类似的表示。
第一小问：只有一个box
第二小问：有多个box。这一问应该和面试官讨论有两个box重合的情况。
第三小问：box是不规则的。每个box的定义变成这个box的所有的点的集合

24 game without (), that means you can not randomly select from 4 cards.

brute force solve it

跟leetcode 24 game 不一样。

附上自己写的版本

```
from fractions import Fraction
from operator import add, sub, mul, truediv

def game24(nums):

    def dfs(nums):
        if len(nums) == 1:
            return nums[0] == 24

        x = nums.pop(0)
        y = nums.pop(0)

        res = False
        for op in [add, sub, mul, truediv]:
            if op in [add, mul]:
                res = res or dfs([op(x, y)] + nums)

            elif op == sub:
                res = res or dfs([op(x, y)] + nums)
                res = res or dfs([op(y, x)] + nums)

            else:
                if y != 0:
                    res = res or dfs([op(x, y)] + nums)

                if x != 0:
                    res = res or dfs([op(y, x)] + nums)

        return res

    return dfs([Fraction(num) for num in nums])
```

类似meeting rooms，输入是一个int[][] meetings, int start, int end, 每个数都是时间，13：00 =》 1300， 9：30 =》 930， 看新的meeting 能不能安排到meetings
ex: {[1300, 1500], [930, 1200],[830, 845]}, 新的meeting[820, 830], return true; [1450, 1500] return false;

```
def convert(time):
    val = str(time)
    if len(val) ==  3:
        hr, m = val[0], val[1:]
    else:
        hr, m = val[:2], val[2:]
    return int(hr)+int(m)/60.


def convert_back(time):
    hr, m = str(time).split('.')[0], int(float(str(time).split('.')[1])/100*60)
    m = str(int(m)) if m != 0 else str(int(m))+'0'
    return hr+m


def arrage_meeting(lst, meeting):
    if not meeting or not lst:
        return False
    s0, s1 = int(convert(meeting[0])*100), int(convert(meeting[1])*100)
    s = set(range(s0,s1))
    ground = set()
    for t in lst:
        t0, t1 = int(convert(t[0])*100), int(convert(t[1])*100)
        ground = ground | set(range(t0,t1))
    available = set(range(0,2400)) - ground
    if s.issubset(available):
        return True
    else:
        return False
```

类似merge interval，唯一的区别是输出，输出空闲的时间段，merge完后，再把两两个之间的空的输出就好，注意要加上0 - 第一个的start time

```
# follow up
def available_time(lst):
    ground = set()
    for t in lst:
        t0, t1 = int(convert(t[0])*100), int(convert(t[1])*100)
        ground = ground | set(range(t0+1,t1))

    available = set(range(0,2400+1)) - ground
    available = sorted(list(available))
    result = []
    start = available[0]
    prev = start
    if len(available) == 1:
        return [convert_back(start/100.),convert_back(start/100.)]
    for t in available[1:]:
        if t > prev+1:
            result.append([start, prev])
            start = t
            prev = t
        else:
            prev = t
    if t > prev + 1:
        result.append([start, prev])
    else:
        result.append([start, t])
    return [[convert_back(t[0]/100.), convert_back(t[1]/100.)] for t in result]
```

给广告在每个domain上被click的次数 1point3acres.com/bbs
要求返回domain及其所有sub domain 被click的总次数
输入：[
["google.com", "60"],
["yahoo.com", "50"],
["sports.yahoo.com", "80"]
]
输出：[
["com", "190"], (60+50+80)
["google.com", "60"],
["yahoo.com", "130"] (50+80)
["sports.yahoo.com", "80"]
]

```
def parser(s):
    subdomain = s.split('.')[::-1]
    result = []
    for sub in subdomain:
        if result == []:
            result.append(sub)
            continue
        result.append('{}.{}'.format(sub, result[-1]))
    return result

def clicker_counter(clicker):
    d = {}
    for entry in clicker:
        keys = parser(entry[0])
        for key in keys:
            if key in d:
                d[key] += int(entry[1])
            else:
                d[key] = int(entry[1])

    return [[key, str(d[key])] for key in d]
```

给每个user访问历史记录，找出两个user之间longest continuous common history
输入： [
  ["3234.html", "xys.html", "7hsaa.html"], // user1
  ["3234.html", "sdhsfjdsh.html", "xys.html", "7hsaa.html"] // user2
], user1 and user2 （指定两个user求intersect）

输出：["xys.html", "7hsaa.html"]

user0 = ["/nine.html", "/four.html", "/six.html", "/seven.html", "/one.html"]
user2 = ["/nine.html", "/two.html", "/three.html", "/four.html", "/six.html", "/seven.html"]
user1 = ["/one.html", "/two.html", "/three.html", "/four.html", "/six.html"]
user3 = ["/three.html", "/eight.html"]

```
def find_longest_continuous_common_history(S, T):
    m = len(S)
    n = len(T)
    counter = [[0]*(n+1) for _ in range(m+1)]
    longest = 0
    lch = set()
    for i in range(m):
        for j in range(n):
            if S[i] == T[j]:
                c = counter[i][j] + 1
                counter[i+1][j+1] = c
                if c > longest:
                    lch = set()
                    longest = c
                    lch.add(tuple(S[i-c+1:i+1]))
                elif c == longest:
                    lch.add(tuple(S[i-c+1:i+1]))
    return lch
```

矩阵里的每个元素是1，但是其中分布着一些长方形区域， 这些长方形区域中的元素为0. 要求输出每个长方形的位置（用长方形的左上角元素坐标和右下角元素坐标表示）。
example：

input: [[1,1,1,1,1,1], [0,0,1,0,1,1], [0,0,1,0,1,0], [1,1,1,0,1,0], [1,0,0,1,1,1]]	output: [[1,0,2,1], [1,3,3,3], [2,5,3,5], [4,1,4,2]]

```
def find_rectangles(arr):
    # Deeply copy the array so that it can be modified safely
    arr = [row[:] for row in arr]
    rectangles = []
    for top, row in enumerate(arr):
        start = 0
        # Look for rectangles whose top row is here
        while True:
            try:
                left = row.index(0, start)  # find the first 0
            except ValueError:
                break
            # Set start to one past the last 0 in the contiguous line of 0s
            try:
                start = row.index(1, left)  # find the first 1 after first 0
            except ValueError:
                start = len(row)
            right = start - 1  # the position of the last 0
            bottom = top + 1
            while (bottom < len(arr) and
                    # No extra zeroes on the sides
                    (left == 0 or arr[bottom][left-1]) and
                    (right == len(row) - 1 or arr[bottom][right + 1]) and
                    # All zeroes in the row
                    not any(arr[bottom][left:right + 1])):
                bottom += 1
            # The loop ends when bottom has gone too far, so backtrack
            bottom -= 1
            rectangles.append([top, left, bottom, right])
            # Remove the rectangle so that it doesn't affect future searches
            for i in range(top, bottom+1):
                arr[i][left:right+1] = [1] * (right + 1 - left)
    return rectangles
```

follow up
如果 Matrix 中有多个由0组成的长方体，请返回多套值（前提每两个长方体之间是不会连接的，所以放心）

follow up 就是两层for循环遍历一遍2D matrix，然后用一个boolean[]把已经搜到的长方形记录下来，最后返回结果?

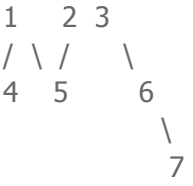
我是用的binary search。index = n x m。转坐标就是 matrix [index/m] [index%m]

不过还有第三问，就是connected components
第三问 基本上就是leetcode connected components,只不过是返回一个listoflist，每个list是一个component的所有点坐标

```
def numIslands(grid):
    result = []
    connected_componet = []
    def dfs(grid, i, j):
        if i<0 or j<0 or i>=len(grid) or j>=len(grid[0]) or grid[i][j] != '1':
            return
        grid[i][j] = '#'
        result.append([i,j])
        dfs(grid, i+1, j)
        dfs(grid, i-1, j)
        dfs(grid, i, j+1)
        dfs(grid, i, j-1)

    if not grid:
        return 0

#    count = 0
    for i in range(len(grid)):
        for j in range(len(grid[0])):
            if grid[i][j] == '1':
                dfs(grid, i, j)
#                  count += 1
                connected_componet.append(result)
                result = []
    return connected_componet
```



输入是int[][] input, input[0]是input[1] 的parent，比如 {{1,4}, {1,5}, {2,5}, {3,6}, {6,7}}会形成上面的图

第一问是只有0个parents和只有1个parent的节点

第二问是 两个指定的点有没有公共祖先

第三问是就一个点的最远祖先，如果有好几个就只需要输出一个就好，举个栗子，这里5的最远祖先可以是1或者2，输出任意一个就可以

```
inputlist = [[1,4],[1,5],[2,5],[3,6],[6,7]]

def findparent(graph):
    from collections import Counter
    parents, children = zip(*graph)
    nodes = range(min(min(children), min(parents)), max(max(children), max(parents))+1)
    n_parents = {}
    c = Counter(children)
    for node in nodes:
        if node in c:
            n_parents[node] = c[node]
        else:
            n_parents[node] = 0
    return [key for key in n_parents if n_parents[key]==0], [key for key in n_parents if n_parents[key]==1]
```

```
def get_all_parent(inputlist,n):
    parent=[]
    for i in inputlist:
        if i[1] == n:
            parent.append(i[0])
    if parent == []:
        return parent
    else:
        next_parent = []
        for i in parent:
            next_parent+=get_all_parent(inputlist,i)
        return (parent + next_parent)


def number_two(inputlist, a, b):
    a_parent=get_all_parent(inputlist,a)
    b_parent=get_all_parent(inputlist,b)
    return True if set(a_parent) & set(b_parent) else False

number_two(inputlist,4,5)



def get_last_parent(inputlist,n,level=0):
    parent=[]
    for i in inputlist:
        if i[1] == n:
            parent.append([i[0],level])
    if parent == []:
        return parent
    else:
```

```
        next_parent = []
        for i in parent:
            next_parent += get_last_parent(inputlist,i[0],level+1)
        return parent + next_parent

def number_three(inputlist, n):
    parents = get_last_parent(inputlist, n)
    max_level, last_parent = 0, None
    for i in parents:
        if i[1] >= max_level:
            last_parent = i[0]
    return last_parent
```

Basic Calculator I

```
def calculator_1(s):
    total = 0
    i = 0
    sign = 1
    while i < len(s):
        c = s[i]
        if c in '+-':
            sign = sign*(1,-1)[c=='-']
            i += 1
        elif c.isdigit():
            val = ''
            while i < len(s) and s[i].isdigit():
                val += s[i]
                i += 1
            total += sign * int(val)
            sign = 1
        else:
            i += 1
    return total
```

Basic Calculator II

Implement a basic calculator to evaluate a simple expression string. The expression string may contain open `(` and closing parentheses `)`, the plus `+` or minus sign `-`, **non-negative** integers and empty spaces .

```
def calculator_2(s):
    total = 0
    signs = [1,1]
    i = 0
    while i < len(s):
        c = s[i]
        if c.isdigit():
            start = i
            while i < len(s) and s[i].isdigit():
                i += 1
            total += signs.pop() * int(s[start:i])
            continue
        if c in '+-(':
            signs += signs[-1]*(1,-1)[c=='-'],
        if c == ')':
            signs.pop()
        i += 1
    return total
```

Basic Calculator III

```
# calculator 3
import re
def calc(a, b, op):
    if op == '+':
        for k, v in b.items():
            a[k] = a.get(k, 0) + v
        return a
    elif op == '-':
        for k, v in b.items():
            a[k] = a.get(k, 0) - v
        return a
    elif op == '*':
        t = {}
        for k1, v1 in a.items():
            for k2, v2 in b.items():
                newk = tuple(sorted(k1+k2))
                t[newk] = t.get(newk, 0) + v1 * v2
        return t

def basicCalculatorIV(expression, evalvars, evalints):
    vars = {n:v for n,v in zip(evalvars, evalints)}
    d = []  # operands
    op = []
    priority = {'(': 0, '+': 1, '-': 1, '*': 2}
    for t in re.findall(r'\(|\)|[a-z]+|[0-9]+|[\+\-\*]', expression):
        if t[0].isdigit():
            d.append({tuple():int(t)})
        elif t[0].isalpha():
            if t in vars:
                d.append({tuple():vars[t]})
            else:
                d.append({(t,): 1})
        elif t == '(':
            op.append(t)
        elif t == ')':
            while op and op[-1] != '(':
                d.append(calc(d.pop(-2), d.pop(-1), op.pop()))
            op.pop()
        elif t in '+-*':
            if not op or priority[t] > priority[op[-1]]:
                op.append(t)
            else:
                while op and priority[t] <= priority[op[-1]]:
                    d.append(calc(d.pop(-2), d.pop(-1), op.pop()))
                op.append(t)
    while op:
        d.append(calc(d.pop(-2), d.pop(-1), op.pop()))

    res = []
    for k in sorted(d[0].keys(), key=lambda x: (-len(x), x)):
        v = d[0][k]
        if v != 0:
            if not k:
                res.append(str(v))
            else:
                res.append('%s*%s' % (v, '*'.join(k)))
    return res

res = basicCalculatorIV('1+2', [],[])
```