LeetCode 399 Evaluate Division 变种汇率题

Frequency = 21

Equations are given in the format `A / B = k`, where `A` and `B` are variables represented as strings, and `k` is a real number (floating point number). Given some queries, return the answers. If the answer does not exist, return `-1.0`.

Example:

Given `a / b = 2.0, b / c = 3.0.`

queries are: `a / c = ?, b / a = ?, a / e = ?, a / a = ?, x / x = ? .`

return `[6.0, 0.5, -1.0, 1.0, -1.0 ].`

Idea: build a graph and search

```
def calcEquation(equations, values, query):
    def seeker(a, b, path=[]):
        if a not in dct or b not in dct:
            return 0
        if b in dct[a]:
            return dct[a][b]
        else:
            for c in dct[a]:
                if c not in path and seeker(c, b, path+[c]):
                    return dct[a][c] * seeker(c, b, path+[c])

    dct = {}
    for i in range(len(equations)):
        nums = equations[i]
        div = float(values[i])
        if nums[0] in dct:
            dct[nums[0]][nums[1]] = div
        else:
            dct[nums[0]] = {nums[0]: 1, nums[1]: div}
        if nums[1] in dct:
            dct[nums[1]][nums[0]] = 1/div
        else:
            dct[nums[1]] = {nums[1]:1, nums[0]: 1/div}

    res = []
    for pair in query:
        if seeker(pair[0], pair[1]):
            res.append(seeker(pair[0], pair[1]))
        else:
            res.append(-1)
    return [float(n) for n in res]
```

一个2维数组里的有人和自行车，要每个人匹配到一辆自行车，人和自行车的距离越短越好，没有距离相同的情况。
Frequency = 13

```
class Solution(object):
    def find_bike(self, matrix):
        if not matrix:
            return []

        distances = []
        for i in range(len(matrix)):  # people
            for j in range(len(matrix[0])):  # bike
                if matrix[i][j] != 0:
                    distances.append((matrix[i][j], i, j))
        distances.sort(key=lambda x: x[0])
        self.res, self.assigned_people, self.assigned_bike = [], set([]), set([])

        def dfs(candidates, path=[]):
            if not candidates:
                if len(path) == len(matrix):
                    self.res = path
                    return True
                else:
                    return False

            for i in range(len(distances)):
                distance, people, bike = distances[i]
                if people not in self.assigned_people and bike not in self.assigned_bike:
                    self.assigned_people.add(people)
                    self.assigned_bike.add(bike)
                    if not dfs(distances[i+1:], path+[[people, bike]]):
                        self.assigned_people.remove(people)
                        self.assigned_bike.remove(bike)


        dfs(distances)

        return self.res

matrix = [[0,1,2,0],
          [3,4,5,6],
          [9,0,0,8],
          [0,7,0,0]]

print Solution().find_bike(matrix)
```

```
'''
there is a 2d array and gbikes are located in that location.
there is a person and he wants to know the nearest location
of the bike which is available for him(there can be more
than 1 nearest bike). person can only move left , right ,
up or down. output should be the distance in int.
'''

class Solution1(object):
    def find_bike(self, grid, a, b):
        if not grid:
            return None

        def isValidMove(grid, i, j):
            if 0 <= i < len(grid) and 0 <= j < len(grid[0]) and grid[i][j] != '#':
                return True
            else:
                return False


        dis, level = 0, [(a,b)]
        while level:
            for i, j in level:
                if grid[i][j] == 1:
                    return dis
                grid[i][j] = '#'

            next_level = []
            for i, j in level:
                for x, y in [(i-1, j), (i+1, j), (i, j-1), (i, j+1)]:
                    if isValidMove(grid, x, y):
                        next_level.append((x, y))
            level = next_level
            dis += 1

        return None

grid = [[1,0,0,1],
        [0,0,1,0],
        [0,0,0,0],
        [0,0,0,1]]

print Solution1().find_bike(grid, 2, 0)
```

Leetcode 684 Redundant Connection Frequency 10

In this problem, a tree is an **undirected** graph that is connected and has no cycles.
The given input is a graph that started as a tree with N nodes (with distinct values 1, 2, ..., N), with one additional edge added. The added edge has two different vertices chosen from 1 to N, and was not an edge that already existed.
The resulting graph is given as a 2D-array of `edges`. Each element of `edges` is a pair `[u, v]` with `u < v`, that represents an **undirected** edge connecting nodes `u` and `v`.
Return an edge that can be removed so that the resulting graph is a tree of N nodes. If there are multiple answers, return the answer that occurs last in the given 2D-array. The answer edge `[u, v]` should be in the same format, with `u < v`.
Example 1:

```
Input: [[1,2], [1,3], [2,3]]
Output: [2,3]
Explanation: The given undirected graph will be like this:
  1
 / \
2 - 3
```

```
class Solution(object):
    def findRedundantConnection(self, edges):
        """
        :type edges: List[List[int]]
        :rtype: List[int]
        """
        graph = collections.defaultdict(set)

        def dfs(source, target):
            if source not in seen:
                seen.add(source)
                if source == target: return True
                return any(dfs(node, target) for node in graph[source])

        for u, v in edges:
            seen = set()
            if u in graph and v in graph and dfs(u, v):
                return u, v
            graph[u].add(v)
            graph[v].add(u)
```

Leetcode 685 Redundant Connection II

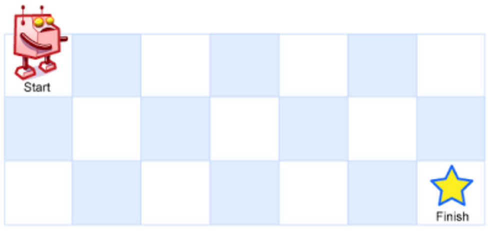
```
class Solution(object):
    def union(self, a, b):
        self.uf[self.find(b)] = self.find(a)

    def find(self, a):
        while self.uf[a] != a:
            a = self.uf[a]
        return a

    def detectCycle(self, V):
        self.visited[V] = True
        for i in range(len(self.adjList[V])):
            nextV = self.adjList[V][i]
            if self.visited[nextV]:
                return (V, nextV)
            ret = self.detectCycle(nextV)
            if ret[0]:
                return ret
        return (None, None)

    def findRedundantDirectedConnection(self, edges):
        """
        :type edges: List[List[int]]
        :rtype: List[int]
        """
        self.uf = [0] + [i + 1 for i in range(len(edges))]
        self.adjList = [[] for i in range(len(edges) + 1)]      # Adjancency 
List
        hasFather = [False] * (len(edges) + 1)                  # Whether a 
vertex has already got a parent
        criticalEdge = None

        for i, edge in enumerate(edges):
            w, v = edge[0], edge[1]
            self.adjList[w].append(v)
            if hasFather[v]:
                criticalEdge = (w, v)                           # If a vertex 
has more than one parent, record the last edge
            hasFather[v] = True
            if self.find(w) == self.find(v):                    # If a loop 
is found, record the edge that occurs last
                cycleEdge = (w, v)
            self.union(w, v)

        if not criticalEdge:                                    # Case 1
            return cycleEdge
        self.visited = [False] * (len(edges) + 1)
        (w, v) = self.detectCycle(criticalEdge[1])
        return (w, v) if w else criticalEdge                   # Case 2 and 3
```

Leetcode 62
Frequency 9

A robot is located at the top-left corner of a m x n grid (marked 'Start' in the diagram below).

The robot can only move either down or right at any point in time. The robot is trying to reach the bottom-right corner of the grid (marked 'Finish' in the diagram below).

How many possible unique paths are there?



```
class Solution(object):
    def uniquePaths(self, m, n):
        """
        :type m: int
        :type n: int
        :rtype: int
        """
        dp = [[0]*n for _ in range(m)]
        for i in range(m):
            dp[i][0] = 1
        for j in range(n):
            dp[0][j] = 1
        for i in range(1, m):
            for j in range(1, n):
                dp[i][j] = dp[i-1][j] + dp[i][j-1]

        return dp[-1][-1]
```

著名变种：DP。
给定一个矩形的长宽，用多少种方法可以从左上角走到右上角 （每一步，只能向正右、右上 或 右下走）

```
def uniquePath(m, n):
    dp = [[0]*n for _ in range(m)]
    dp[0][0] = 1
    for j in range(n):  # 后遍历列，先遍历行，记得找规律
        for i in range(m):  # (i, j-1), (i-1, j-1), (i+1, j-1)
            if 0 <= j-1 < n:
                if 0 <= i < m:
                    dp[i][j] += dp[i][j-1]
                if 0 <= i-1 < m:
                    dp[i][j] += dp[i-1][j-1]
                if 0 <= i+1 < m:
                    dp[i][j] += dp[i+1][j-1]
    return dp[0][-1]
```

-follow up：**如果给矩形里的三个点，要求解决上述问题的同时，遍历这三个点 （切割矩形，一个一个地做DP，然后相加）**

-follow up：**如何判断这三个点一个是合理的，即存在遍历这三个点的路经**
  Suppose each point can be expressed as (i,j), and (i_1, j_1) is valid. Then, (i_2, j_2) is valid if and only if (i_2 == i_1) or (j_2-j_1)/(i_2-i_1) >= 1 or (j_2-j_1)/(i_2-i_1) <= -1

-follow up：**如果给你一个H，要求你的路径必须向下越过H这个界，怎么做**
楼主这题试一试用镜像做，也就是说重点不在(W, 0), 而在(W, 2H)， W是矩阵的宽度

Leetcode 843 Guess the Word
Frequency 9
O(n)

```
class Solution(object):
    def findSecretWord(self, wordlist, master):

        def pair_matches(a, b):         # count the number of matching characters
            return sum(c1 == c2 for c1, c2 in zip(a, b))

        def most_overlap_word():
            counts = [[0 for _ in range(26)] for _ in range(6)]     # counts[i][j] is nb of words
with char j at index i
            for word in candidates:
                for i, c in enumerate(word):
                    counts[i][ord(c) - ord("a")] += 1

            best_score = 0
            for word in candidates:
                score = 0
                for i, c in enumerate(word):
                    score += counts[i][ord(c) - ord("a")]           # all words with same chars
in same positions
                if score > best_score:
                    best_score = score
                    best_word = word

            return best_word

        candidates = wordlist[:]        # all remaining candidates, initially all words
        while candidates:

            s = most_overlap_word()     # guess the word that overlaps with most others
            matches = master.guess(s)

            if matches == 6:
                return

            candidates = [w for w in candidates if pair_matches(s, w) == matches]   # filter
words with same matches
```

Leetcode 890 Find and Replace Pattern Frequency 8

You have a list of `words` and a `pattern`, and you want to know which words in `words` matches the pattern.
A word matches the pattern if there exists a permutation of letters `p` so that after replacing every letter `x` in the pattern with `p(x)`, we get the desired word.
(Recall that a permutation of letters is a bijection from letters to letters: every letter maps to another letter, and no two letters map to the same letter.)
Return a list of the words in `words` that match the given pattern.
You may return the answer in any order.

Example 1:

```
Input: words = ["abc","deq","mee","aqq","dkd","ccc"],
pattern = "abb"
Output: [“mee","aqq"]
```

```
class Solution(object):
    def findAndReplacePattern(self, words, pattern):
        """
        :type words: List[str]
        :type pattern: str
        :rtype: List[str]
        """
        def match(word):
            map_word, map_pattern = {}, {}
            for w, p in zip(word, pattern):
                if w not in map_word:
                    map_word[w] = p
                if p not in map_pattern:
                    map_pattern[p] = w
                if (map_word[w], map_pattern[p]) != (p, w):
                    return False
            return True

        return [word for word in words if match(word)]
```

Leetcode 489 Robot Room Cleaner
Frequency 7

Given a robot cleaner in a room modeled as a grid.
Each cell in the grid can be empty or blocked.
The robot cleaner with 4 given APIs can move forward, turn left or turn right.
Each turn it made is 90 degrees.
When it tries to move into a blocked cell, its bumper sensor detects the obstacle and it stays on the current cell.
Design an algorithm to clean the entire room using only the 4 given APIs shown below.

```
interface Robot {
  // returns true if next cell is open and robot moves into the cell.
  // returns false if next cell is obstacle and robot stays on the current cell.
  boolean move();

  // Robot will stay on the same cell after calling turnLeft/turnRight.
  // Each turn will be 90 degrees.
  void turnLeft();
  void turnRight();

  // Clean the current cell.
  void clean();
}
```

Example:

```
Input:
room = [
  [1,1,1,1,1,0,1,1],
  [1,1,1,1,1,0,1,1],
  [1,0,1,1,1,1,1,1],
  [0,0,0,1,0,0,0,0],
  [1,1,1,1,1,1,1,1]
],
row = 1,
col = 3
```

```
class Solution(object):
    def cleanRoom(self, robot):
        """
        :type robot: Robot
        :rtype: None
        """
        directions = [(0, 1), (1, 0), (0, -1), (-1, 0)]

        def goBack(robot):
            robot.turnLeft()
            robot.turnLeft()
            robot.move()
            robot.turnRight()
            robot.turnRight()

        def dfs(pos, robot, d, lookup):
            if pos in lookup:
                return
            lookup.add(pos)

            robot.clean()
            for _ in directions:
                if robot.move():
                    dfs((pos[0]+directions[d][0],
                         pos[1]+directions[d][1]),
                        robot, d, lookup)
                    goBack(robot)
                robot.turnRight()
                d = (d+1) % len(directions)

        dfs((0, 0), robot, 0, set())
```

Leetcode 855 Exam Room
Frequency 6

In an exam room, there are `N` seats in a single row, numbered `0, 1, 2, ..., N-1`.
When a student enters the room, they must sit in the seat that maximizes the distance to the closest person. If there are multiple such seats, they sit in the seat with the lowest number. (Also, if no one is in the room, then the student sits at seat number 0.)

```
class ExamRoom(object):
    def __init__(self, N):
        self.N = N
        self.students = []

    def seat(self):
        # Let's determine student, the position of the next
        # student to sit down.
        if not self.students:
            student = 0
        else:
            # Tenatively, dist is the distance to the closest student,
            # which is achieved by sitting in the position 'student'.
            # We start by considering the left-most seat.
            dist, student = self.students[0], 0
            for i, s in enumerate(self.students):
                if i:
                    prev = self.students[i-1]
                    # For each pair of adjacent students in positions (prev, s),
                    # d is the distance to the closest student;
                    # achieved at position prev + d.
                    d = (s - prev) / 2
                    if d > dist:
                        dist, student = d, prev + d

            # Considering the right-most seat.
            d = self.N - 1 - self.students[-1]
            if d > dist:
                student = self.N - 1

        # Add the student to our sorted list of positions.
        bisect.insort(self.students, student)
        return student

    def leave(self, p):
        self.students.remove(p)
```

设计一个Map, get() 和 set() function,
每个set同时还set一个expiration time, get 的时候如果expire了就返回null,
这个用普通hashmap就能解决
follow up 加一个clean function，clean所有expire的entry，我就加了一个heap解决，到这里大概30分钟不到

```
import heapq
class ExpirationMap(object):
    def __init__(self):
        self.nodeForKey = {}
        self.nodeForExpiration = {} # key: expiration
        self.ExpirationHeap = [] # (expiration, key)

    def Get(self, key, timeStamp):
        value = self.nodeForKey[key]
        expiration = self.nodeForExpiration[key]
        if timeStamp > expiration:
            return None
        else:
            return value

    def Set(self, key, value, expiration):
        if key not in self.nodeForKey:
            heapq.heappush(self.ExpirationHeap, (expiration, key))
        # we won't update the expiration in the heap,
        # if key has existied until clean() is called
        self.nodeForExpiration[key] = expiration
        self.nodeForKey[key] = value


    def Clean(self, timeStamp):
        while self.ExpirationHeap:
            expiration, key = heapq.heappop(self.ExpirationHeap)
            if expiration == self.nodeForExpiration[key]:  # record is matched
                if timeStamp < expiration:  # the smallest expiration is larger than timeStamp
                    heapq.heappush(self.ExpirationHeap, (expiration, key))
                    break
            else:
                heapq.heappush(self.ExpirationHeap, (self.nodeForExpiration[key], key))
```

多个不重复的长方形内随机取点

```
import random
def randomPoint(recs):
    areas = [(upper_right[1]-lower_left[1])*(upper_right[0]-lower_left[0])
                    for lower_left, upper_right in recs]
    area_sum = sum(areas)

    # find what rectangle should choose based on its area
    r = random.randint(0, area_sum)
    for i in range(len(areas)):
        r -= areas[i]
        if r <= 0:
            break

    # find a random point within the picked rectangle
    lower_left, upper_right = recs[i]
    return random.randint(lower_left[0], upper_right[0]),
random.randint(lower_left[1], upper_right[1])

print randomPoint([[(0,0),(3,3)], [(4,0),(5,6)]])
```

follow up：多个可能重复的长方形内随机取点

Leetcode 853 Car Fleet
Frequency 5

`N` cars are going to the same destination along a one lane road. The destination is `target` miles away.
Each car `i` has a constant speed `speed[i]` (in miles per hour), and initial position `position[i]` miles towards the target along the road.
A car can never pass another car ahead of it, but it can catch up to it, and drive bumper to bumper at the same speed.
The distance between these two cars is ignored – they are assumed to have the same position.
A car fleet is some non-empty set of cars driving at the same position and same speed. Note that a single car is also a car fleet.
If a car catches up to a car fleet right at the destination point, it will still be considered as one car fleet.

How many car fleets will arrive at the destination?

```
class Solution(object):
    def carFleet(self, target, position, speed):
        n = len(speed)
        arrival_time = [(target-position[i])/float(speed[i]) for i in range(n)]
        data = sorted(zip(arrival_time, position), reverse=True)
        count = 0
        curr_max = -1
        for _, position in data:
            if position > curr_max:
                curr_max = position
                count += 1
        return count
```

Leetcode 853 Minimum Cost to Hire K Workers Frequency 5

There are `N` workers. The `i`-th worker has a `quality[i]` and a minimum wage expectation `wage[i]`.

Now we want to hire exactly `K` workers to form a paid group. When hiring a group of K workers, we must pay them according to the following rules:

1. Every worker in the paid group should be paid in the ratio of their quality compared to other workers in the paid group.
2. Every worker in the paid group must be paid at least their minimum wage expectation.

Return the least amount of money needed to form a paid group satisfying the above conditions.

```
class Solution(object):
    def mincostToHireWorkers(self, quality, wage, K):
        from fractions import Fraction
        workers = sorted((Fraction(w, q), q, w)  for q , w in zip(quality, wage))
        ans = float('inf')
        pool = []
        sumq = 0
        for ratio, q, w in workers:
            heapq.heappush(pool, -q)
            sumq += q

            if len(pool) > K:
                sumq += heapq.heappop(pool)

            if len(pool) == K:
                ans = min(ans, ratio*sumq)

        return float(ans)
```

Leetcode 750 Number of Corner Rectangles Frequency 5

Given a grid where each entry is only 0 or 1, find the number of corner rectangles.
A corner rectangle is 4 distinct 1s on the grid that form an axis-aligned rectangle. Note that only the corners need to have the value 1. Also, all four 1s used must be distinct.
Example 1:

```
Input: grid = 
[[1, 0, 0, 1, 0],
 [0, 0, 1, 0, 1],
 [0, 0, 0, 1, 0],
 [1, 0, 1, 0, 1]]
Output: 1
Explanation: There is only one corner rectangle, with 
corners grid[1][2], grid[1][4], grid[3][2], grid[3][4].
```

counts[i][j] is used to keep track of the number of rows, found so far, in which column i and column j are 1.
Every time you find a new row that has i and j set to 1, counts will tell you how many rectangles you can form with this new row.

```
def countCornerRectangles(self, grid):
        res = 0
        counts = [[0] * len(grid[rows]) for rows in range(len(grid))]
        for row in range(0, len(grid)):
            for j in range(1, len(grid[0])):
                if grid[row][j] == 1:
                    for i in range(0, j):
                        if grid[row][i] == 1:
                            res += counts[i][j]
                            counts[i][j] += 1
        return int(res)
```

Leetcode 815 Bus Routes
Frequency 5

We have a list of bus routes. Each `routes[i]` is a bus route that the i-th bus repeats forever. We start at bus stop `S` (initially not on a bus), and we want to go to bus stop `T`. Travelling by buses only, what is the least number of buses we must take to reach our destination? Return -1 if it is not possible.

```
Example:
Input:
routes = [[1, 2, 7], [3, 6, 7]]
S = 1
T = 6
Output: 2
```

Intuition

Instead of thinking of the stops as nodes (of a graph), think of the buses as nodes. We want to take the least number of buses, which is a shortest path problem, conducive to using a breadth-first search.

```
class Solution(object):
    def numBusesToDestination(self, routes, S, T):
        if S == T:
            return 0
        routes = map(set, routes)
        graph = collections.defaultdict(set)
        for i, r1 in enumerate(routes):
            for j in range(i+1, len(routes)):
                r2 = routes[j]
                if any(r in r2 for r in r1):
                    graph[i].add(j)
                    graph[j].add(i)

        seen, targets = set(), set()
        for node, route in enumerate(routes):
            if S in route:
                seen.add(node)
            if T in route:
                targets.add(node)

        queue = [(node, 1) for node in seen]
        for node, depth in queue:
            if node in targets:
                return depth
            for nei in graph[node]:
                if nei not in seen:
                    seen.add(nei)
                    queue.append((nei, depth+1))
        return -1
```

Leetcode 659 Split Array into Consecutive Subsequences Frequency 5

You are given an integer array sorted in ascending order (may contain duplicates), you need to split them into several subsequences, where each subsequences consist of at least 3 consecutive integers. Return whether you can make such a split.

Example 1:

```
Input: [1,2,3,3,4,5]
Output: True
Explanation:
You can split them into two consecutive subsequences :
1, 2, 3
3, 4, 5
```

使用两个哈希表map，第一个map用来建立数字和其出现次数之间的映射count，第二个用来建立可以加在某个连续子序列后的数字及其可以出现的次数之间的映射tail。对于第二个map，举个例子来说，就是假如有个连，[1,2,3]，那么后面可以加上4，所以就建立4的映射。这样我们首先遍历一遍数组，统计每个数字出现的频率，然后我们开始遍历数组，对于每个遍历到的数字，首先看其当前出现的次数，如果为0，则继续循环；如果tail中存在这个数字的非0映射，那么表示当前的数字可以加到某个连的末尾，我们将当前数字的映射值自减1，然后将下一个连续数字的映射值加1，因为当[1,2,3]连上4后变成[1,2,3,4]之后，就可以连上5了；如果不能连到其他子序列后面，我们来看其是否可以成为新的子序列的起点，可以通过看后面两个数字的映射值是否大于0，都大于0的话，说明可以组成3连儿，于是将后面两个数字的映射值都自减1，还有由于组成了3连儿，在tail中将末尾的下一位数字的映射值自增1；如果上面情况都不满足，说明该数字是单牌，只能划单儿，直接返回false。最后别忘了将当前数字的count映射值自减1。退出for循环后返回true，参见代码如下：

```
class Solution(object):
    def isPossible(self, nums):
        count = collections.Counter(nums)
        tails = collections.Counter()
        for x in nums:
            if count[x] == 0:
                continue
            elif tails[x] > 0:
                tails[x] -= 1
                tails[x+1] += 1
            elif count[x+1] > 0 and count[x+2] > 0:
                count[x+1] -= 1
                count[x+2] -= 1
                tails[x+3] += 1
            else:
                return False
            count[x] -= 1
        return True
```

给一个国王家的family tree （n-ary tree），王位继承是先传国王最年长的儿子，假如最年长儿子死了，就传给死儿子最年长的儿子。。。如果这些人都不存在，再考虑国王次年长的儿子，以此类推。要求设计这样一棵树，死掉的人不要求删除，实现birth（）和输出王位继承顺序的method（死掉的人不在继承顺序结果里）。（及其变种）

Frequency 5

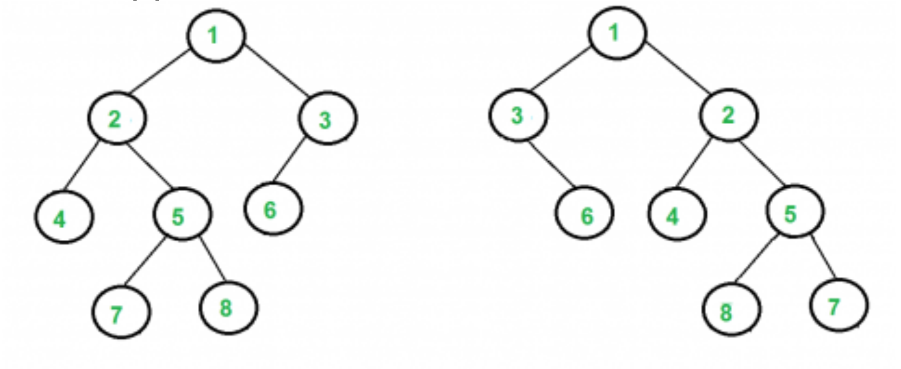
```
class TreeNode(object):
    def __init__(self, name, death=False):
        self.name = name
        self.death = death
        self.children = []

class FamilyTree(object):
    def __init__(self, king):
        self.king = king  # a root
        self.people = {self.king.name: self.king} # shallow copy

    def birth(self, father_name, new_kid):
        self.people[father_name].children.append(new_kid)
        self.people[new_kid.name] = new_kid

    def find_heir(self):
        def _dfs(root):
            if not root:
                return
            for kid in root.children:
                if not kid.death:
                    self.res.append(kid.name)
                _dfs(kid)
        self.res = []
        _dfs(self.king)
        return self.res
```

Tree Isomorphism Problem

Frequency 5

Write a function to detect if two trees are isomorphic. Two trees are called isomorphic if one of them can be obtained from other by a series of flips, i.e. by swapping left and right children of a number of nodes. Any number of nodes at any level can have their children swapped. Two empty trees are isomorphic.

For example, following two trees are isomorphic with following sub-trees flipped: 2 and 3, NULL and 6, 7 and 8.



```
# A Binary tree node
class Node:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None
def isIsomorphic(n1, n2):
    if n1 is None and n2 is None:
        return True
    if n1 is None or n2 is None:
        return False
    if n1.data != n2.data :
        return False
    # There are two possible cases for n1 and n2 to be isomorphic
    # Case 1: The subtrees rooted at these nodes have NOT
    # been "Flipped".
    # Both of these subtrees have to be isomorphic, hence the &&
    # Case 2: The subtrees rooted at these nodes have
    # been "Flipped"
    return ((isIsomorphic(n1.left, n2.left)and
            isIsomorphic(n1.right, n2.right)) or
            (isIsomorphic(n1.left, n2.right) and
            isIsomorphic(n1.right, n2.left))
            )
```

给定一个multiple tree，以及需要删除的多个nodes，要求返回一个node的list，以便在nodes被删除之后可以找到这些nodes的child。

Frequency 5

例子：

```
              a
     /     /     \      \
   b      d      c      f
 /   \   \
h     z   i
```

假如删除b和f，返回{a, h, z, i}

假如删除a和b，返回{d, c, f, h, z, i}

可以用level traverse解。

```python
class MultipleTreeNode(object):
    def __init__(self, x):
        self.val = x
        self.children = []

def findDeleteNodeChildren(root, nodeLst):
    if not root or not nodeLst:
        return []

    level, res = [root], set([])
    removeLst = nodeLst[:]
    while level and removeLst:
        next_level = []
        for node in level:
            if node in removeLst:
                res |= set([n.val for n in node.children])
                removeLst.remove(node)
            next_level += node.children
        level = next_level

    return list(res - set([node.val for node in nodeLst]))
```

可乐饮料机，有一系列按钮，每个按钮按下去会得到一定体积范围的可乐。先给定一个目标体积范围，问不限制按按钮次数，能否确定一定能得到目标范围内的可乐?
举例：有三个按钮，按下去得到的范围是[100, 120], [200, 240], [400, 410],
假设目标是[100,110]那答案是不能。因为按下一，可能得到120体积的可乐，不在目标范围里。
假设目标是[90, 120]，那答案是可以。因为按下一，一定可以得到此范围内的可乐。
假设目标是[300, 360], 那答案是可以，因为按下一再按二，一定可以得到此范围内
假设目标是[310, 360], 那答案是不能，因为按下一再按二，有可能得到300，永远没可能确定得到这个范围内的可乐。
假设目标是[1,9999999999]，那答案是可以。随便按一个都确定满足此范围。

Frequency 5

```
def colaMachine(buttons, lowerbound, upperbound):
    weight = upperbound - lowerbound
    dp = [0] * (weight+1)
    for i in range(len(buttons)):
        curr_lowerbound, curr_upperbound = buttons[i]
        curr_weight = curr_upperbound - curr_lowerbound
        for j in range(1, weight+1):
            if j-curr_weight >= 0 and \
               dp[j-curr_weight]+curr_lowerbound > dp[j] and \
               dp[j-curr_weight]+curr_lowerbound+j <= upperbound:
                dp[j] = dp[j-curr_weight] + curr_lowerbound
                if dp[j] >= lowerbound:
                    return True
    return False
```

判断target字符串是否可以由给定字符串转换得到。转换的规则是：每次转换要变所有的相同字母
比如：abca -> cdec 可以 abca -> abea -> cbec -> cdec

Frequency 5

idea:

用hashmap，存两个s1和s2(target)的对应关系，一旦s1中的char需要对应到多个s2中的char时，返回错误，全部结束未返回错误的话，返回正确

重点是考虑target string里有26个字母时，没有中间值转换。

```
def wordConverter(source, target):
    if len(source) != len(target):
        return False

    if len(set(ch for ch in target))== 26 and source != target:
        return False

    d = {}
    for i, ch_in_source in enumerate(source):
        if ch_in_source not in d or d[ch_in_source] == target[i]:
            d[ch_in_source] = target[i]
        else:
            return False
        print d
    return True

print wordConverter('abca', 'cbad')
```

Frequency 4

1. 下围棋，判断棋盘上一个点是不是被包围了。 （找有没有和黑棋相连的空格）

   用了dfs 写的挺顺的。 然后问testcase， 画了各种形状的围棋图

2. 一个map分为n层，每层都m个node， 每个node有值，每一层跟下一层的node是full connected，edge的值不一定相同，求从第一层到最后一层的mini cost
3. ABC 小哥和大黑哥， 问了一个拿纸牌游戏， 纸牌上面有值，比如说 100， 1， -1， 2， 200， 1. 然后两个人轮流拿，直到拿完。 但是每次只能拿从左边数起的前三个，但是如果你要拿第三个，就必须前两个都拿了，你要拿第二个，就必须第一个也拿了，大家都最优策略，问最后第一个人能拿多少分。dfs 加 cache做了。
4. 一个image以2D byte array的方式储存（byte[][] image），每个象素点是1个bit（0或1）。现在要求每行的象素点做对称翻转。我的做法是先把每行的byte对称翻转，然后再把每个byte各自翻转。

   如其中一行byte[] row = {11010100，00101010}
   第一步{00101010，11010100}
   第二步{01010100，00101011}

   时间复杂度是O(m*n*8), follow up如何优化时间，应该是在翻转每个byte上把O（8）的复杂度降低，但是不要求使用复杂的位运算
5. 已知screen的高和宽，给你最小和最大的fontSize，要求给定一个string，将string用竟可能大的fontSize显示在screen里。已知两个API getHeight(int fontSize), getWidth(char c, int fontSize)，可以得到每个character在不同fontSize下的高和宽。和面试官交流后，确认string可以拆分成几行显示在screen中，先提出暴力解法，然后用二分法优化.
6. 莉蔻把零散 （打砖块）
7. 莉蔻儿五三及其变种（meeting room）
8. random generate a maze
9. 白人大哥，给一个list intervals of integer ([2,9], [1,3], [10,20] ..), 给一个integer X, 查找X在不在给的interval 里面。答曰一遍遍历，o(n). 他问能不能

优化。这个list 是无序的，只有排好序才能优化，但是排序就要o(nlogn)。在我追问下，他说了如果我们有很多的查询x在不在，能不能优化每次的查询？那就是一次性处理这个list intervals. 我先给的解法是遍历一遍intervals, 把所有数字存到hashset里，查询用hashset， O(1)。他貌似对这个解法不满意，说这个方法如果数字很大，很占内存，等等。我就顺着他说先sort, 然后binary search. binary search 前要先remove overlap.然后开始写代码。

10. 实现一个iterator的iterator

Frequency 3

676.

Implement Magic Dictionary

Implement a magic directory with `buildDict`, and `search` methods.
For the method `buildDict`, you'll be given a list of non-repetitive words to build a dictionary.
For the method `search`, you'll be given a word, and judge whether if you modify **exactly** one character into **another** character in this word, the modified word is in the dictionary you just built.

```
class MagicDictionary(object):
    def __init__(self):
        self.magic = []

    def buildDict(self, dict):
        for key in dict:
            self.magic.append(key)

    def search(self, word):
        for Lst in [zip(word, key) for key in self.magic if len(key)==len(word)]:
            c = 0
            for (a,b) in Lst:
                if a!=b:
                    c+=1
                if c>1:
                    break
            if c==1:
                return True
        return False
```

849.
Maximize Distance to Closest Person

In a row of `seats`, `1` represents a person sitting in that seat, and `0` represents that the seat is empty.
There is at least one empty seat, and at least one person sitting.
Alex wants to sit in the seat such that the distance between him and the closest person to him is maximized.
Return that maximum distance to closest person.

```
class Solution(object):
    def maxDistToClosest(self, seats):
        ans = 0
        for seat, group in itertools.groupby(seats):
            if not seat:
                K = len(list(group))
                ans = max(ans, (K+1)/2)

        return max(ans, seats.index(1), seats[::-1].index(1))
```

LeetCode — 418. Sentence Screen Fitting

Given a `rows x cols` screen and a sentence represented by a list of **non-empty** words, find **how many times** the given sentence can be fitted on the screen.

Note:

- A word cannot be split into two lines.
- The order of words in the sentence must remain unchanged.
- Two consecutive words **in a line** must be separated by a single space.
- Total words in the sentence won't exceed 100.
- Length of each word is greater than 0 and won't exceed 10.
- 1 ≤ rows, cols ≤ 20,000.

```
def wordsTyping(sentence, rows, cols):
    s = ' '.join(sentence)

    count = 0
    n = len(s)
    for i in range(rows):
        count += cols
        if s[count % n] == ' ':
            count += 1
        else:
            while count > 0 and s[(count-1) % n] != ' ':
                count -= 1

    return count / n
```

68. Text Justification

Given an array of words and a width maxWidth, format the text such that each line has exactly maxWidth characters and is fully (left and right) justified.

You should pack your words in a greedy approach; that is, pack as many words as you can in each line. Pad extra spaces `' '` when necessary so that each line has exactly maxWidth characters.

Extra spaces between words should be distributed as evenly as possible. If the number of spaces on a line do not divide evenly between words, the empty slots on the left will be assigned more spaces than the slots on the right.

For the last line of text, it should be left justified and no **extra** space is inserted between words.

Note:

- A word is defined as a character sequence consisting of non-space characters only.
- Each word's length is guaranteed to be greater than 0 and not exceed maxWidth.
- The input array `words` contains at least one word.

Example 1:

```
Input:
words = ["This", "is", "an", "example", "of", "text",
"justification."]
maxWidth = 16
Output:
[
   "This    is    an",
   "example  of text",
   "justification.  "
]
```

```
class Solution(object):
    def fullJustify(self, words, maxWidth):
        # Algorithm: Add as many words as possible to line with
        # spaces in between and increase size of spaces as needed.
        result = []
        word_idx = 0
        while word_idx < len(words):
            next_line = []
            char_count = word_count = space_count = 0

            # Add words and a space while we have words and if there are
available characters
            while (
                word_idx < len(words) and
                len(words[word_idx]) <= (maxWidth - (char_count +
space_count))
            ):
                next_word = words[word_idx]
                next_line.append(next_word)
                next_line.append(" ")
                char_count += len(next_word)
                word_count += 1
                space_count += 1
                word_idx += 1

            # Ignore the last space in our count of valid spaces
            space_count -= 1

            # Get the number of free characters left (not including spaces)
            char_difference = maxWidth - char_count

            # Case 1: if we're out of words or have only
            # one word on the line, then just increase the
            # number of spaces after the last word in the line.
            #
            # The number of spaces added should be the difference
            # in max width and non-space characters less the number
            # of spaces not accounted for in the difference.
            if word_idx == len(words) or word_count == 1:
                next_line[-1] *= (char_difference - space_count)

            # Case 2: otherwise, increase the space size after each
            # character as evenly as possible. Prefer left spaces to
            # apply extra spaces. Drop the last space at the end
            else:
                next_line.pop()
                num_spaces = char_difference / space_count
                num_slots_with_extra_spaces = char_difference % space_count

                for space_idx in range(1, word_count+space_count, 2):
                    if num_slots_with_extra_spaces:
                        next_line[space_idx] *= num_spaces+1
                        num_slots_with_extra_spaces -= 1
                    else:
                        next_line[space_idx] *= num_spaces

            result.append("".join(next_line))
        return result
```

844. Backspace String Compare

Given two strings `S` and `T`, return if they are equal when both are typed into empty text editors. `#` means a backspace character.

Example 1:

```
Input: S = "ab#c", T = "ad#c"
Output: true
Explanation: Both S and T become “ac”.
```

```
# Time: O(M+N)
# Space: O(M+N)

class Solution(object):
    def backspaceCompare(self, S, T):
        """
        :type S: str
        :type T: str
        :rtype: bool
        """
        def reader(S):
            res = []
            for s in S:
                if s != '#':
                    res.append(s)
                elif res:
                    res.pop()
            return res

        return reader(S) == reader(T)

# Can be optimized by two-pointer method
```

394.

Decode String

Given an encoded string, return it's decoded string.

The encoding rule is: `k[encoded_string]`, where the encoded_string inside the square brackets is being repeated exactly k times. Note that k is guaranteed to be a positive integer.

Examples:

```
s = "3[a]2[bc]", return "aaabcbc".
s = "3[a2[c]]", return "accaccacc".
s = "2[abc]3[cd]ef", return “abcabccdcdcdef".
```

```python
class Solution(object):
    def decodeString(self, s):
        """
        :type s: str
        :rtype: str
        """
        stack = []
        stack.append(["", 1])
        num = ""
        for ch in s:
            if ch.isdigit():
                num += ch
            elif ch == '[':
                stack.append(["", int(num)])
                num = ""
            elif ch == ']':
                st, k = stack.pop()
                stack[-1][0] += st*k
            else:
                stack[-1][0] += ch
        return stack[0][0]
```

334.

Increasing Triplet Subsequence

Given an unsorted array return whether an increasing subsequence of length 3 exists or not in the array.

```
class Solution(object):
    def increasingTriplet(self, nums):
        """
        :type nums: List[int]
        :rtype: bool
        """
        first = second = float('inf')
        for n in nums:
            if n <= first:
                first = n
            elif n <= second:
                second = n
            else:
                return True
        return False
```

LeetCode 774. Minimize Max Distance to Gas Station

On a horizontal number line, we have gas stations at positions stations[0], stations[1], ..., stations[N-1], where N = stations.length.
Now, we add K more gas stations so that D, the maximum distance between adjacent gas stations, is minimized.
Return the smallest possible value of D.
Example:
Input: stations = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], K = 9
Output: 0.500000

如果我们假设知道了答案会怎么样？因为知道了最大间隔，所以如果目前的两个station之间的gap没有符合最大间隔的约束，那么我们就必须添加新的station来让它们符合最大间隔的约束，这样一来，对于每个gap我们是能够求得需要添加station的个数。如果需求数<=K，说明我们还可以进一步减小最大间隔，直到需求数>K。

```
class Solution(object):
    def minmaxGasDist(self, st, K):
        """
        :type stations: List[int]
        :type K: int
        :rtype: float
        """
        lf = 1e-6
        rt = st[-1] - st[0]
        eps = 1e-7
        while rt - lf > eps:
            mid = (rt + lf) / 2
            cnt = 0
            for a, b in zip(st, st[1:]):
               cnt += (int)((b - a) / mid)
            if cnt <= K: rt = mid
            else: lf = mid
        return rt
```

337.

House Robber III

There is only one entrance to this area, called the "root." Besides the root, each house has one and only one parent house. After a tour, the smart thief realized that "all houses in this place forms a binary tree". It will automatically contact the police if two directly-linked houses were broken into on the same night.

Determine the maximum amount of money the thief can rob tonight without alerting the police.

Example 1:

Input: `[3,2,3,null,3,null,1]` **Output:** `7`

```
     3
    / \
   2   3
    \   \
     3   1
```

```
class Solution(object):
    def rob(self, root):
        def superrob(node):
            # returns tuple of size two (now, later)
            # now: max money earned if input node is robbed
            # later: max money earned if input node is not robbed

            # base case
            if not node: return (0, 0)

            # get values
            left, right = superrob(node.left), superrob(node.right)

            # rob now
            now = node.val + left[1] + right[1]

            # rob later
            later = max(left) + max(right)

            return (now, later)

        return max(superrob(root))
```

[leetcode] 340. Longest Substring with At Most K Distinct Characters

Given a string, find the length of the longest substring T that contains at most k distinct characters.
For example, Given s = “eceba” and k = 2,
T is "ece" which its length is 3.

思路: 一个hash表和一个左边界标记. 遍历字符串将其加入到hash表中, 不同字符多于k个了, 就从左边开始删字符. 直到hash表不同字符长度等于k.此时字符串的长度就是当前字符和左边界的距离.

```
def longest_substring_distinct_K(s, k):
    d = {}
    left, max_length = 0, 0
    for i in range(len(s)):
        if s[i] in d:
            d[s[i]] += 1
        else:
            d[s[i]] = 1
        while len(d) > k:
            d[s[left]] -= 1
            if d[s[left]] == 0:
                d.pop(s[left])
            left += 1
        max_length = max(max_length, i-left+1)
    return max_length
```

设计个log start，finish finish后 输出log id跟内容 但finish的时候 如果start前面还有start 就不能输出
直到没有pending了 就全输出
比如
start(1, time1)
start(2, time2)
start(3, time3).
finish(2, time2)
finish(3, time3)
finish(1, time1)
输出 1， 2， 3
为啥 因为finish2得时候 前面还有个1 被start过
finish3的时候 2虽然没了 前面还有个1 被start过

priority queue 加hashmap。queue按start sort。map记录process有没有finish。

486.
Predict the Winner

Given an array of scores that are non-negative integers. Player 1 picks one of the numbers from either end of the array followed by the player 2 and then player 1 and so on. Each time a player picks a number, that number will not be available for the next player. This continues until all the scores have been chosen. The player with the maximum score wins.

Given an array of scores, predict whether player 1 is the winner. You can assume each player plays to maximize his score.

Example 1:

```
Input: [1, 5, 2]
Output: False
```

```
class Solution(object):
    def PredictTheWinner(self, nums):
        """
        :type nums: List[int]
        :rtype: bool
        """
        dp = [[0] * len(nums) for _ in range(len(nums))]
        def getScore(nums, left, right):
            # maximize the difference  (my_score - op_score) of nums[l:r+1]
            if left == right:
                return nums[left]
            if dp[left][right] > 0:
                return dp[left][right]
            dp[left][right] = max(nums[left] - getScore(nums, left+1, right),
                                  nums[right] - getScore(nums, left, right-1))
            return dp[left][right]
        return getScore(nums, 0, len(nums)-1) >= 0
```

215.

Kth Largest Element in an Array

Find the **k**th largest element in an unsorted array. Note that it is the kth largest element in the sorted order, not the kth distinct element.

Example 1:

```
Input: [3,2,1,5,6,4] and k = 2
Output: 5
```

```
import heapq
class Solution(object):
    def findKthLargest(self, nums, k):
        min_heap = [-float('inf')] * k
        heapq.heapify(min_heap)
        for num in nums:
            if num > min_heap[0]:
                heapq.heappop(min_heap)
                heapq.heappush(min_heap, num)
        return min_heap[0]
```

312. Burst Balloons

Given `n` balloons, indexed from `0` to `n-1`. Each balloon is painted with a number on it represented by array `nums`. You are asked to burst all the balloons. If the you burst balloon `i` you will get `nums[left] * nums[i] * nums[right]` coins. Here `left` and `right` are adjacent indices of `i`. After the burst, the `left` and `right` then becomes adjacent.

Find the maximum coins you can collect by bursting the balloons wisely.

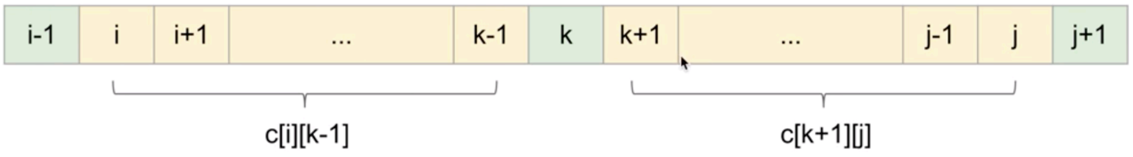
Note:

- You may imagine `nums[-1] = nums[n] = 1`. They are not real therefore you can not burst them.
- 0 ≤ `n` ≤ 500, 0 ≤ `nums[i]` ≤ 100

Example:

```
Input: [3,1,5,8]
Output: 167
Explanation: nums = [3,1,5,8] --> [3,5,8] -->   [3,8]   -->  [8]  --> []
             coins =  3*1*5      +  3*5*8    +  1*3*8      + 1*8*1   = 167
```

c[i][j] = maxCoins(nums[i]~nums[j])
ans = c[1][n]
c[i][j] = max{c[i][k-1] + nums[i-1]*nums[k]*nums[j+1] + c[k+1][j] | i <= k <= j }



```
class Solution(object):
    def maxCoins(self, nums):
        n = len(nums)
        nums = [1]+nums+[1]
        # max score we can get from nums[i:j+1]
        dp = [[0] * (n+2) for _ in range(n+2)]

        for l in range(1, n+1):
            for i in range(1, n-l+2):
                j = i + l -1
                for k in range(i, j+1):
                    dp[i][j] = max(dp[i][j], dp[i][k-1] +
nums[i-1]*nums[k]*nums[j+1] + dp[k+1][j])
        return dp[1][n]
```

769.

Max Chunks To Make Sorted

Given an array `arr` that is a permutation of `[0, 1, ..., arr.length - 1]`, we split the array into some number of "chunks" (partitions), and individually sort each chunk. After concatenating them, the result equals the sorted array.

What is the most number of chunks we could have made?

Example 1:

```
Input: arr = [4,3,2,1,0]
Output: 1
```

Intuition and Algorithm

Let's try to find the smallest left-most chunk. If the first `k` elements are `[0, 1, ..., k-1]`, then it can be broken into a chunk, and we have a smaller instance of the same problem.

We can check whether `k+1` elements chosen from `[0, 1, ..., n-1]` are `[0, 1, ..., k]` by checking whether the maximum of that choice is `k`.

```
class Solution(object):
    def maxChunksToSorted(self, arr):
        ans = ma = 0
        for i, x in enumerate(arr):
            ma = max(ma, x)
            if ma == i: ans += 1
        return ans
```

505. The Maze II

There is a ball in a maze with empty spaces and walls. The ball can go through empty spaces by rolling up, down, left or right, but it won't stop rolling until hitting a wall. When the ball stops, it could choose the next direction.

Given the ball's start position, the destination and the maze, find the shortest distance for the ball to stop at the destination. The distance is defined by the number of empty spaces traveled by the ball from the start position (excluded) to the destination (included). If the ball cannot stop at the destination, return -1.

The maze is represented by a binary 2D array. 1 means the wall and 0 means the empty space. You may assume that the borders of the maze are all walls. The start and destination coordinates are represented by row and column indexes.

```
def shortestDistance(maze, start, destination):
    def go2End(maze, start, direction):
        new_start = (start[0]+direction[0], start[1]+direction[1])
        row_num, col_num = len(maze), len(maze[0])
        if new_start[0] < 0 or new_start[0] >= row_num or new_start[1] < 0 or new_start[1] >= 
col_num or maze[new_start[0]][new_start[1]]==1:
            return start
        return go2End(maze, new_start, direction)

    visited = {}
    directions = [[0,1],[0,-1],[1,0],[-1,0]]
    ans = float('inf')
    visited[start] = 0
    queue = [start]
    while queue:
        pos = queue.pop(0)
        if pos == destination:
            ans = min(ans, visited[pos])
        else:
            for i in range(4):
                res = go2End(maze, pos, directions[i])
                line_length = abs(pos[1]-res[1]) if i <= 1 else abs(pos[0]-res[0])
                if res not in visited or visited[res] > visited[pos] + line_length:
                    visited[res] = visited[pos] + line_length
                    queue.append(res)
    return -1 if ans == float('inf') else ans

maze = [[0,0,1,0,0],
        [0,0,0,0,0],
        [0,0,0,1,0],
        [1,1,0,1,1],
        [0,0,0,0,0]]
print shortestDistance(maze, (0,4), (4,4))
```

96.

Unique Binary Search Trees

Given n, how many structurally unique **BST's** (binary search trees) that store values 1 ... n?

Example:

```
Input: 3
Output: 5
Explanation:
Given n = 3, there are a total of 5 unique BST's:

   1         3     3      2      1
    \       /     /      / \      \
     3     2     1      1   3      2
    /     /       \                 \
   2     1         2                 3
```

```
class Solution(object):
    def numTrees(self, n):
        """
        :type n: int
        :rtype: int
        """
        if n == 1:
            return 1

        dp = [0] * (n+1)
        dp[0] = 1
        dp[1] = 1
        for i in range(2, n+1):
            n_tree = 0
            for j in range(i):
                n_tree += dp[j]*dp[i-j-1]
            dp[i] = n_tree
        return dp[n]
```

834.

Sum of Distances in Tree

An undirected, connected tree with `N` nodes labelled `0...N-1` and `N-1` `edges` are given. The `i`th edge connects nodes `edges[i][0]` and `edges[i][1]` together. Return a list `ans`, where `ans[i]` is the sum of the distances between node `i` and all other nodes.

Example 1:

```
Input: N = 6, edges = [[0,1],[0,2],[2,3],[2,4],[2,5]]
Output: [8,12,6,10,10,10]
Explanation:
Here is a diagram of the given tree:
  0
 / \
1   2
   /|\
  3 4 5
We can see that dist(0,1) + dist(0,2) + dist(0,3) +
dist(0,4) + dist(0,5)
equals 1 + 1 + 2 + 2 + 2 = 8.  Hence, answer[0] = 8,
and so on.
```

题解：每个节点保存两个值，一个是其子树的节点个数（包括自身节点也要计数）nodesum[ ]，一个是其子树各点到它的距离 dp[ ]，那么我们假设根节点为 u ，其仅有一个儿子 v ， u 到 v 的距离为 1 ，而 v 有若干儿子节点，那么 dp[v] 表示 v 的子树各点到 v 的距离和，那么各个节点到达 u 的距离便可以这样计算： dp[u] = dp[v] + nodesum[v] *1; （式子的理解，v 的一个儿子节点为 f，那么 f 到达 u 的距离为 (sum[f ->v] + sum [v- > u])*1 ，dp[v] 包含了 sum[f->v]*1，所以也就是式子的分配式推广到各个子节点计算出来的和）。我们已经知道了各个节点到达根节点的距离和，那么从根节点开始递推下来可以得到各个点的距离和。另开一个数组表示每个节点的到其他节点的距离和，那么对于根节点u来说， dissum[u] = dp[u]。以 u 的儿子 v 为例，v 的子节点到 v 不必经过 v->u 这条路径，因此 dissum[u] 多了 nodesum[v] * 1，但是对于不是 v 的子节点的节点，只到达了 u ，因此要到达 v 必须多走 u->v 这条路径，因此 dissum[u] 少了 (N - nodesum[v]) * 1) ,所以 dissum[v] =

dissum[u] - nodesum[v] * 1 + (N - nodesum[v]) * 1，按照这个方法递推下去就可以得到各个点的距离和。

```
class Solution(object):
    def sumOfDistancesInTree(self, N, edges):
        graph = collections.defaultdict(set)
        for u, v in edges:
            graph[u].add(v)
            graph[v].add(u)

        count = [1] * N
        ans = [0] * N
        def dfs(node=0, parent=None):
            for child in graph[node]:
                if child != parent:
                    dfs(child, node)
                    count[node] += count[child]
                    ans[node] += ans[child] + count[child]

        def dfs2(node=0, parent=None):
            for child in graph[node]:
                if child != parent:
                    ans[child] = ans[node] - count[child] + N-
count[child]
                    dfs2(child, node)

        dfs()
        dfs2()
        return ans
```