1 Collatz Conjecture

If a number is odd, the next transform is 3*n+1
If a number is even, the next transform is n/2
The number is finally transformed into 1.
The step is how many transforms needed for a number turned into 1.
Given an integer n, output the max steps of transform number in [1, n] into 1.

```
import math
def collatz(n):
    temp = [0] * (n+1)
    res, temp[1], d= 1, 1, {}

    def find_steps(a):
        count = 0
        while a > 1:
            if a in d:
                return d[a] + count
            if a%2:
                a = 3*a + 1
            else:
                a //= 2
            count += 1
        return count

    for i in range(2, n+1):
        if bin(i).count('1') == 1:
            temp[i] = int(math.log(i, 2))
        elif i%2 == 0:
            temp[i] = temp[i>>1] + 1
        else:
            temp[i] = find_steps(i)
        d[i] = temp[i]
        res = max(res, temp[i])
    return res
```

2 Implement Queue with Limited Size of Array

Implement a queue with a number of arrays, in which each array has fixed size.

```
class MyQueue(object):

  def __init__(self, k):
    self.queue = []
    self.size = k
```

```
    def enqueue(self, x):
        if len(self.queue) < self.size - 1:
            self.queue.append(x)
        else:
            if len(self.queue) == self.size - 1:
                self.queue.append([x])
            else:
                q = self.queue
                while len(q) == self.size:
                    q = q[-1]
                if len(q) == self.size-1:
                    q.append([x])
                else:
                    q.append(x)

    def dequeue(self):
        l = len(self.queue)
        if l == 0:
            return False
        elif l == 1 and isinstance(self.queue[0], list):
            self.queue = self.queue[0]
        return self.queue.pop(0)

    def empty(self):
        return len(self.queue) == 0
```

3 List of List Iterator

Given an array of arrays, implement an iterator class to allow the client to traverse and remove elements in the array list.

This iterator should provide three public class member functions:

boolean hasNext() return true if there is another element in the set

int next() return the value of the next element in the array

void remove() remove the last element returned by the iterator. That is, remove the element that the previous next() returned. This method can be called only once per call to next(), otherwise an exception will be thrown.

```
class MyListItr():

    def __init__(self, l):
        self.l = l
        self.called = False
        self.c = 0
```

```
    def next(self):
        if self.hasNext():
            val = self.l[self.c]
            self.called = True
            self.c += 1
            return val
        else:
            return None

    def hasNext(self):
        if not self.l or self.c == len(self.l):
            return False
        while isinstance(self.l[self.c], list):
            self.l = self.l[:self.c] + self.l[self.c] + self.l[self.c+1:]
        self.called = False
        return True

    def remove(self):
        if not self.called:
            raise Exception('Call next() first.')
        self.l = self.l[:self.c-1] + self.l[self.c:]
        self.c -= 1
        self.called = False
```

4 Display Page (Pagination)

Given an array of CSV strings representing search results, output results sorted by a score initially. A given host may have several listings that show up in these results. Suppose we want to show 12 results per page, but we don't want the same host to dominate the results.

Write a function that will reorder the list so that a host shows up at most once on a page if possible, but otherwise preserves the ordering. Your program should return the new array and print out the results in blocks representing the pages.

Given an array of csv strings, output results separated by a blank line.

5 Travel Buddy

I have a wish list of cities that I want to visit to, my friends also have city wish lists that they want to visit to. If one of my friends share more than 50% (over his city count in his wish list), he is my buddy.

Given a list of city wish list, output buddy list sorting by similarity.

6 File System

Write a file system class, which has two functions: create, and get

create("/a",1)
get("/a") //get 1
create("/a/b",2)
get("/a/b") //get 2
create("/c/d",1) //Error, because "/c" is not existed
get("/c") //Error, because "/c" is not existed

```
import sys
class FileSystem(object):
    def __init__(self):
        self.pathMap = {}
        self.callbackMap = {}
    def create(self, path, value):
        if path in self.pathMap:
            return False
        lastIdx = path.rfind('/')
        if path[:lastIdx] and not path[:lastIdx] in self.pathMap:
            return False
        self.pathMap[path] = value
        return True
    def set(self, path, value):
        if not path in self.pathMap:
            return False
        self.pathMap[path] = value
        curPath = path
        while curPath:
            if curPath in self.callbackMap:
                func = self.callbackMap[curPath]
                func(curPath)
            lastIdx = path.rfind('/')
            curPath = curPath[:lastIdx]
        return True
    def get(self, path):
        if not path in self.pathMap:
            return None
        return self.pathMap[path]
    def watch(self, path, callback):
        if not path in self.pathMap:
            return False
        self.callbackMap[path] = callback
        return True
def callback0(path):
```

```
    print('callback on path {0}'.format(path))
    sys.exit()

def callback1(path):
    print('callback on path {0}'.format(path))
    sys.exit()
```

7 Palindrome Pairs

Given a list of unique words, find all pairs of distinct indices (i, j) in the given list, so that the concatenation of the two words, i.e. words[i] + words[j] is a palindrome.

```
class Solution(object):
    def palindromePairs(self, words):
        wordict = {word: i for i, word in enumerate(words)}
        res = []
        for i in range(len(words)):
            for j in range(len(words[i])+1):
                tmp1 = words[i][:j]
                tmp2 = words[i][j:]
                if wordict.get(tmp1[::-1], -1) not in (-1, i) and tmp2 == tmp2[::-1]:
                    res.append([i,wordict[tmp1[::-1]]])
                if j > 0 and wordict.get(tmp2[::-1], -1) not in (-1, i) and tmp1 == tmp1[::-1]:
                    res.append([wordict[tmp2[::-1]],i])
        return res
```

8 Find Median in Large Integer File of Integers

Find the median from a large file of integers. You can not access the numbers by index, can only access it sequentially. And the numbers cannot fit in memory.

```
class Solution:
    def FindMedianinLargeIntegerFileofIntegers(self):
        def search(nums, k, left, right):
            if (left >= right):
                return left
            res = left
            guess = left + (right - left) / 2
            count = 0
            for num in nums:
                if (num <= guess):
                    count += 1
                    res = max(res, num)

            if (count == k):
```

```
            return res
          else if (count < k):
            return search(nums, k, max(res + 1, guess), right)
          else:
            return search(nums, k, left, res)

      len = 0
      min_v, max_val = float('inf'), float('-inf')
      for num in nums:
         len += 1
         min_v = min(min_v, num)
         max_v = max(max_v, num)

      if (len % 2 == 1):
         return search(nums, len / 2 + 1, min_v, max_v)
       else:
         return (search(nums, len/2, min_v, max_v) + search(nums, len/2 + 1, min_v, max_v))/2
```

9 IP Range to CIDR

Given an IPv4 IP address p and an integer n, return a list of CIDR strings that most succinctly represents the range of IP addresses from p to (p + n).

```
class Solution(object):
   def ip2number(self, ip):
      numbers = list(map(int, ip.split(".")))
      n = (numbers[0]<<24) + (numbers[1]<<16) + (numbers[2]<<8) + numbers[3]
      return n

   def number2ip(self,n):
      return ".".join([str(n>>24&255), str(n>>16&255),str(n>>8&255), str(n&255)])

   def ilowbit(self,x):
      for i in range(32):
         if(x & (1<<i)):
            return i

   def lowbit(self,x):
      return 1<<self.ilowbit(x)

   def ipToCIDR(self, ip, n):
      number = self.ip2number(ip)
      result = []
      while n>0:
```

```
            lb = self.lowbit(number)
            print(lb)
            while lb>n:
                lb = lb//2

            n = n - lb

            result.append(self.number2ip(number) + "/" + str(32-self.ilowbit(lb)))
            number = number + lb;
        return result
```

10 CSV Parser

Write a method to parse input string in CSV format.

11 Text Justification

```
class Solution:
    def fullJustify(self, words, maxWidth):
        res = []
        cur_letter, cur_length, cur = 0, 0, []

        for word in words:
            if len(word) + cur_length + cur_letter > maxWidth:
                if cur_letter == 1:
                    cur[0] += ' '*(maxWidth-cur_length)
                else:
                    for i in range(maxWidth-cur_length):
                        cur[i%(cur_letter-1)] += ' '
                res.append(''.join(cur))
                cur, cur_length, cur_letter = [word], len(word), 1
            else:
                cur, cur_length, cur_letter = cur + [word], cur_length + len(word), cur_letter+1
        return res + [' '.join(cur).ljust(maxWidth)]
```

12 Regular Expression

```
def isMatch(s, p):
    table = [[False] * (len(s) + 1) for _ in range(len(p) + 1)]
    table[0][0] = True

    for i in range(2, len(p) + 1):
        table[i][0] = table[i - 2][0] and p[i - 1] == '*'
```

```
    for i in range(1, len(p) + 1):
        for j in range(1, len(s) + 1):
            if p[i - 1] != "*":
                table[i][j] = table[i - 1][j - 1] and  (p[i - 1] == s[j - 1] or p[i - 1] == '.')
            else:
                table[i][j] = table[i - 2][j] or table[i - 1][j]
                if p[i - 2] == s[j - 1] or p[i - 2] == '.':
                    table[i][j] |= table[i][j - 1]

    return table[-1][-1]
```

13 Water Drop/Water Land/Pour Water

Input is a array represent how the height of water is at each position, the number of water will be poured, and the pour position. Print the land after all water are poured.

Example: input land height int[]{5,4,2,1,3,2,2,1,0,1,4,3} The land is looks ike:

+
++ +
++ + ++
+++ +++ ++
++++++++ +++ GO to the left, find the first location going up, add water
++++++++++++
012345678901
water quantity is 8 and pour at position 5. The land becomes:
+
++ +
++www+ ++
+++w+++www++
++++++++w+++
++++++++++++
012345678901

```
def getWaterLevel(height, position, count):
    if not height:
        return
    n = len(height)
    water = [0] * n
    while count:
        putLocation = position
        left = position
        right = position
        while left >= 1:
            if height[left - 1] + water[left - 1] > height[left] + water[left]:
                break
```

```
            left -= 1
        if height[left] + water[left] < height[position] + water[position]:
            putLocation = left
        else:
            while right < n - 1:
                if height[right + 1] + water[right + 1] > height[right] + water[right]:
                    break
                right += 1

            if height[right] + water[right] < height[position] + water[position]:
                putLocation = right
        water[putLocation] += 1
        count -= 1

    highest = 0
    for i in range(n):
        highest = max(highest, height[i] + water[i])
    res = []
    for h in range(highest, 0, -1):
        temp = []
        for i in range(n):
            if(height[i] + water[i] < h):
                temp.append(" ")
            elif height[i] < h:
                temp.append("w")
            else:
                temp.append("*")
        res.append(temp)
```

14 Hilbert Curve

Given (x, y, iter), in which (x, y) is position at x-axis and y-axis, and iter is how many iterations will be. Output is in iteration iter, how many steps are required to move from (0, 0) to (x, y) in iteration iter.

```
def hilbertCurve(x, y, iter):
    if (iter == 0) return 1;
    areaCnt = (1 << (iter * 2 - 2));
    borderLen = (1 << (iter - 1));

    if (x >= borderLen and y >= borderLen):
        return areaCnt * 2 + hilbertCurve(x - borderLen, y - borderLen, iter - 1);
    else if (x < borderLen anf y >= borderLen):
```

```
    return areaCnt + hilbertCurve(x, y - borderLen, iter - 1);
  else if (x < borderLen and y < borderLen):
    return hilbertCurve(y, x, iter - 1);
  else:
    return areaCnt * 3 + hilbertCurve(borderLen - 1 - y, 2 * borderLen - 1 - x, iter - 1);
```

15 Simulate Diplomacy #这个题真的没看懂

Input (army_name, location, action())

action() could be:

move(new_location) then army_name to new_location, If there is an army at new_location, then company strength of two aramy:

The army have higher strength stay at new_location, the lower army is disappeared.

If two army have the same strength, both are disppeared.

hold() stay at the same location

support(army_name) supoort another army. The supported army have one more strength.

The army's strength are intialized as the same.

16 Meeting Time

Input is a number of meetings (start_time, end_time)。 Output is a number of time intervals (start_time, end_time), where there is no meetings.

For exmaple, input is [[1, 3], [6, 7]], [[2, 4]],□[[2, 3], [9, 12]]]

output [[4, 6], [7, 9]]

```
def meeting_time(t):
        t = sum(t, [])
        s = min(x[0] for x in t)
        e = max(x[1] for x in t)

        res = [0] * (e+1)
        for tt in t:
                res[tt[0]] += 1
                res[tt[1]] -= 1

        for i in range(1, len(res)):
                if res[i] == 0:
                        res[i] = res[i-1]
                else:
                        res[i] += res[i-1]

        i, r = s, []
        while i < e:
```

```
            if res[i] == 0:
                start = i
                while i < e and res[i] == 0:
                    i += 1
                r.append([start, i])
            i += 1
        return r
```

17 Round Prices

Given an array of numbers A = [x1, x2, ..., xn] and T = Round(x1+x2+... +xn). We want to find a way to round each element in A such that after rounding we get a new array B = [y1, y2, ..., yn] such that y1+y2+...+yn = T where yi = Floor(xi) or Ceil(xi), ceiling or floor of xi.

We also want to minimize sum |x_i-y_i|

```
def rounding_number(n):
    floor = [int(x) for x in n]
    total_diff = sum(n) - sum(floor)
    d = [[n[i] - floor[i], floor[i], i] for i in range(len(n))]
    d.sort(key=lambda x: x[0], reverse=True)
    for i in range(int(round(total_diff))):
        d[i][1] += 1
    d.sort(key=lambda x: x[2])
    return [x[1] for x in d]
```

18 Sliding Game

You're given a 3x3 board of a tile puzzle, with 8 tiles numbered 1 to 8, and an empty spot. You can move any tile adjacent to the empty spot, to the empty spot, creating an empty spot where the tile originally was. The goal is to find a series of moves that will solve the board, i.e. get [[1, 2, 3], [4, 5, 6], [7, 8, -]…

```
class SlidingGame:
    def __init__(self, matrix):
        self.dirs =[[1, 0],[0, 1],[-1, 0],[0, -1]]
        self.matrix = matrix
        self.m = len(matrix)
        self.n = len(matrix[0])
        recoveredMatrix = [[0] * self.n for i in range(self.m)]
        for i in range(self.m):
            for j in range(self.n):
                if (matrix[i][j] == 0):
                    self.originX = i
                    self.originY = j
                recoveredMatrix[i][j] = i * self.n + j
```

```
    self.recovered = self.getMatrixString(recoveredMatrix)

def getSolution(self):
    pathWord = ["Down", "Right", "Up", "Left"]

    items = []
    matrix = []
    visited = set()

    stringMatrix = self.getMatrixString(self.matrix)
    items.append([self.originX, self.originY])
    matrix.append(stringMatrix)
    visited.add(stringMatrix)
    path = []

    while items:
        size = len(items)
        for i in range(size):
            curElement = items.pop(-1)
            curMatrixString = matrix.pop(-1)
            x = curElement[0]
            y = curElement[1]

            if curMatrixString == self.recovered:
                return path

            for k in range(len(self.dirs)):
                xx = x + self.dirs[k][0]
                yy = y + self.dirs[k][1]

                if (xx < 0 or xx >= self.m or yy < 0 or yy >= self.n):
                    continue

                newMatrix = self.recoverMatrixString(curMatrixString)
                newMatrix[xx][yy], newMatrix[x][y] = newMatrix[x][y], newMatrix[xx][yy]
                newMatrixString = self.getMatrixString(newMatrix)
                if newMatrixString in visited:
                    continue

                items.append([xx, yy]);
                matrix.append(newMatrixString);
                path += [pathWord[k]];
                visited.add(newMatrixString)
```

```
        return path

    def getMatrixString(self, matrix):
        res = ''
        for m in matrix:
            res += ''.join([str(x) for x in m])
        return res

    def recoverMatrixString(self, strs):
        parts = list(strs)
        res = [[0] * self.n for i in range(self.m)]
        for i in range(self.m):
            for j in range(self.n):
                res[i][j] = int(parts[i * self.n + j])
        return res
```

19 Maximum Number of Nights You Can Accommodate

Given a set of numbers in an array which represent number of consecutive nights of AirBnB reservation requested, as a host, pick the sequence which maximizes the number of days of occupancy, at the same time, leaving at least 1 day gap in between bookings for cleaning. Problem reduces to finding max-sum of non-consecutive array elements.

20 Find Case Combinations of a String

Find all the combinations of a string in lowercase and uppercase. For example, string "ab" >>> "ab", "Ab", "aB", "AB". So, you will have 2^n (n = number of chars in the string) output strings. The goal is for you to test each of these strings and see if it matchs a hidden string.

```
def case_combination(s):
        s = list(s.lower())
        res =[s[0], s[0].upper()]
        for i in range(1, len(s)):
                c, temp = s[i], []
                for r in res:
                        temp += [r+c, r+c.upper())
                res = temp
        return res
```

21 Menu Combination Sum

Given a menu (list of items prices), find all possible combinations of items that sum a particular value K. (A variation of the typical 2sum/Nsum questions).

```
class Solution:
    def MenuCombinationSum(self, prices, target):
        def search(res, centsPrices, start, centsTarget, curCombo, prices):
            if (centsTarget == 0):
                res.append(curCombo[:])
                return

            for i in range(start, len(centsPrices)):
                if (i > start and centsPrices[i] == centsPrices[i - 1]):
                    continue
                if (centsPrices[i] > centsTarget):
                    break
                search(res, centsPrices, i + 1, centsTarget - centsPrices[i], curCombo + [prices[i]],
prices)

        res = []
        if not prices or target <= 0:
            return res

        centsTarget = target * 100
        prices.sort()
        centsPrices = [int(round(x * 100)) for x in prices]

        search(res, centsPrices, 0, centsTarget, [], prices)
        return res
```

22 K Edit Distance

Find all words from a dictionary that are k edit distance away.

```
import collections

class TrieNode:
    def __init__(self):
        self.children = {}
        self.isLeaf = False

class Trie:
    def __init__(self):
        self.root = TrieNode()

    def insert(self, s):
        p = self.root
```

```
        for c in s:
            if c not in p.children:
                p.children[c] = TrieNode()
            p = p.children[c]
        p.isLeaf = True

class Solution:
    def KEditDistance(self, words, target, k):
        def search(curr, target, k, root, prevDist):
            if root.isLeaf:
                if prevDist[-1] == k:
                    res.add(curr)
                else:
                    return

            for c in root.children:
                currDist = [0 for i in range(len(target)+1)]
                currDist[0] = len(curr) + 1
                for j in range(1, len(prevDist)):
                    if target[j - 1] == c:
                        currDist[j] = prevDist[j - 1]
                    else:
                        currDist[j] = min(min(prevDist[j - 1], prevDist[j]), currDist[j - 1]) + 1
                search(curr + c, target, k, root.children[c], currDist)

        res = set()
        if (not words or len(words) == 0 or not target or len(target) == 0 or k < 0):
            return res

        trie = Trie()

        for word in words:
            trie.insert(word)

        root = trie.root
        prev = [i for i in range(len(target)+1)]

        search("", target, k, root, prev)

        return res
```

23 Boggle Game

Given 2d matrix of letters, and a word dictionary, find a path which has largest number of words (existed inside the dictionary).

```
class TrieNode:
    def __init__(self):
        self.children = {}
        self.isLeaf = False

class Trie:
    def __init__(self):
        self.root = TrieNode()

    def insert(self, s):
        p = self.root
        for c in s:
            if c not in p.children:
                p.children[c] = TrieNode()
            p = p.children[c]
        p.isLeaf = True

class Solution:
    def __init__(self):
        self.res = []

    def getAllWords(self, board, words):
        trie = Trie()
        for word in words:
            trie.insert(word)

        m = len(board)
        n = len(board[0])
        for i in range(m):
            for j in range(n):
                visited = [[False] * n for _ in range(m)]
                path = []
                self.findWords([], board, visited, path, i, j, trie.root)
        return self.res


    def findWords(self, result, board, visited, words, x, y, root):
        m = len(board)
        n = len(board[0])
        for i in range(x, m):
```

```
            for j in range(y, n):
                nextWordIndexes = []
                path = []
                self.getNextWords(nextWordIndexes, board, visited, path, i, j, root)
                for indexes in nextWordIndexes:
                    word = ""
                    for i, j in indexes:
                        word += board[i][j]

                    words.append(word)
                    if len(words) > len(self.res):
                        self.res = words[:]

                    self.findWords(result, board, visited, words, i, j, root)

                    for i, j in indexes:
                        visited[i][j] = False
                    words.pop(-1)
            # y = 0


    def getNextWords(self, words, board, visited, path, i, j, root):
        if i < 0 or i >= len(board) or j < 0 or j >= len(board[0]) or visited[i][j] or board[i][j] not in
root.children:
            return

        root = root.children[board[i][j]]
        if root.isLeaf:
            newPath = path
            newPath.append((i,j))
            words.append(newPath[:])
            print(visited)
            return

        visited[i][j] = True
        self.getNextWords(words, board, visited, path + [(i,j)], i + 1, j, root)
        self.getNextWords(words, board, visited, path + [(i,j)], i - 1, j, root)
        self.getNextWords(words, board, visited, path + [(i,j)], i, j + 1, root)
        self.getNextWords(words, board, visited, path + [(i,j)], i, j - 1, root)
        visited[i][j] = False
```

24 Minimum Cost with At Most K Stops

Given a flight itinerary consisting of starting city, destination city, and ticket price (2d list) - find the optimal price flight path to get from start to destination. (A variation of Dynamic Programming Shortest Path)

```
def min_cost(flights, start, end, k):
    info = collections.defaultdict(set)
    for tour, cost in flights:
        st, ed = tour.split("->")
        info[st].add((ed, cost))

    cur_level = {start: 0}
    ans = 0x7FFFFF
    for _ in xrange(k + 1):
        next_level = {}
        for port, cur_cost in cur_level.iteritems():
            for nx, cost in info[port]:
                if nx == end:
                    ans = min(ans, cost + cur_cost)
                else:
                    if nx not in next_level:
                        next_level[nx] = cost + cur_cost
                    else:
                        next_level[nx] = min(next_level[nx], cost + cur_cost)
        cur_level = next_level
    return ans
```

25 String Pyramids Transition Matrix

Given a list of leaf nodes in a pyramid, and a map which indicates what's the possible parent node given a left and right node. Return true if the one of leaf node could turn into the root node, Otherwise, return false.

```
import collections

class Solution(object):
    def pyramidTransition(self, bottom, allowed):
        allow = collections.defaultdict(list)
        for a in allowed:
            allow[a[:2]].append(a[2])

        def dfs(bottom):
            if len(bottom) == 1:
                return True
```

```
            new = []
            for i in range(len(bottom)-1):
                choices = []
                if bottom[i:i+2] in allow:
                    choices = allow[bottom[i:i+2]]
                if not choices:
                    return False
                if not new:
                    new = [[x] for x in choices]
                else:
                    temp = []
                    for r in new:
                        for c in choices:
                            temp.append(r+[c])
                    new = temp[:]
            return any(dfs(''.join(x)) for x in new)

        return dfs(bottom)
```

26 Finding Ocean

Given: An array of strings where L indicates land and W indicates water, and a coordinate marking a starting point in the middle of the ocean.

Challenge: Find and mark the ocean in the map by changing appropriate Ws to Os.

An ocean coordinate is defined to be the initial coordinate if a W, and any coordinate directly adjacent to any other ocean coordinate.

void findOcean(String[] map, int row, int column);

Replace 1/0 to W/O here

```
def numOfIslands2(grid):
    def fill_land(i, j):
        if 0 <= i < len(grid) and 0 <= j < len(grid[0]) and grid[i][j] == '1':
            grid[i][j] = '0'
            for x, y in [(1,0), (-1,0), (0,1), (0,-1)]:
                fill_land(i+x, j+y)
        return
    res = 0
    for i in range(len(grid)):
        for j in range(len(grid[0])):
            if grid[i][j] == '1':
                res += 1
                fill_land(i, j)
    return res
```

27 Preference List

Given a list of everyone's preferred city list, output the city list following the order of everyone's preference order.

For example, input is [[3, 5, 7, 9], [2, 3, 8], [5, 8]]. One of possible output is [2, 3, 5, 8, 7, 9].

```
class Node:
    def __init__(self, val):
        self.val = val
        self.IN = set()
        self.OUT = set()


def preference_list(l):
    nodes = {}

    for ll in l:
        for n in ll:
            if n not in nodes:
                nodes[n] = Node(n)

    for ll in l:
        for i in range(len(ll)-1):
            nodes[ll[i]].OUT.add(ll[i+1])
            nodes[ll[i+1]].IN.add(ll[i])

    res = []
    while nodes:
        for node in nodes:
            if not nodes[node].IN:
                res.append(node)
                for node2 in nodes[node].OUT:
                    if node in nodes[node2].IN:
                        nodes[node2].IN.discard(node)
                nodes.pop(node)
                break

    return res
```

28 Minimum Vertices to Traverse Directed Graph

Given a directed graph, represented in a two dimensional array, output a list of points that can be used to traverse every points with the least number of visited vertices.

```
    def minVertices(self, edges):
        p = {}

        def find(x):
            return x if x == p[x] else find(p[x])

        def union(x, y):
            px = find(x)
            py = find(y)
            p[py] = px

        for x, y in edges:
            if x not in p and y not in p:
                p[x] = x
                p[y] = y
                union(x, y)
            elif x in p and y not in p:
                p[y] = p[x]
            elif y in p and x not in p:
                p[x] = p[y]
            else:
                union(x, y)

        res = set()
        for x, y in edges:
            res.add(find(x))
            res.add(find(y))

        return list(res)
```

29 10 Wizards

There are 10 wizards, 0-9, you are given a list that each entry is a list of wizards known by wizard. Define the cost between wizards and wizard as square of different of i and j. To find the min cost between 0 and 9.

```
def min_distance(wizards, start=0, end=9):
    cur_level = {start: 0}
    ans = 0x7FFFFFFF
    for _ in xrange(10):
        next_level = {}
        for idx, cur_cost in cur_level.iteritems():
```

```
            if idx >= len(wizards):
                continue
            for nx in wizards[idx]:
                cost = cur_cost + (nx - idx) ** 2
                if nx == end:
                    ans = min(ans, cost)
                else:
                    if nx not in next_level:
                        next_level[nx] = cost
                    else:
                        next_level[nx] = min(next_level[nx], cost)
        cur_level = next_level
    return ans
```

30 Number of Intersected Rectangles

Given lots of rectangles, output how many of them intersect.

31 Guess Number

The system has a secret four digits number, in which each digit is in range of one to 6 [1, 6].

Given a four digital number, the system also provide a API that returns the number of correct matched digits.

Design a method to guess this secret number.

```
import socket

class MySocket:
    def __init__(self, sock=None):
        if sock is None:
            self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        else:
            self.sock = sock

    def connect(self, host, port):
        self.sock.connect((host, port))

    def mysend(self, msg):
        sent = self.sock.send(msg)
        if sent == 0:
            raise RuntimeError("socket connection broken")

    def myreceive(self):
        chunk = self.sock.recv(2048)
```

```
        if chunk == b'':
            raise RuntimeError("socket connection broken")
        return chunk


def guess_game():
    s = MySocket()
    s.connect()
    s.mysend('START\n'.encode())
    r = s.myreceive()

    # Find starting point
    start = [1,1,1,1]
    while True:
        r = take_guess(s, start)
        if r[0] == r[2] and r[0] == '0':
            break
        else:
            start = [x+1 for x in start]

    print(start)
    res = []
    c = start[0]
    for i in range(4):
        for j in range(1, 7):
            if j != c:
                guess = start[:i] + [j] + start[i+1:]
                r = take_guess(s, guess)
                if r[0] == '1':
                    res.append(j)
                    break

    return ''.join([str(x) for x in res])

def take_guess(s, guess):
    guess = ''.join([str(x) for x in guess])
    s.mysend(('GUESS ' + guess + '\n').encode())
    r = s.myreceive()
    r = r.decode()
    return r


print(guess_game())
```