```
class Solution:
    def validPalindrome(self, s):
        return self.isPalindrome(list(s),False);
    def isPalindrome(self, s, deleted_one):
        for x in range (0, len(s)//2):
            if(s[x]!=s[len(s)-x-1]):
                if(not deleted_one):
                    try1=s[:x] + s[x+1 :]
                    try2=s[:len(s)-x-1] + s[len(s)-x :]
                    return self.isPalindrome(try1,True) or self.isPalindrome(try2,True)
                else:
                    return False
        return True

s=Solution();
print(s.validPalindrome("CAABAC"))
```

```
class MapSum:
    def __init__(self):
        self.map = {}


    def insert(self, key, val):
        self.map[key]=val
        print(self.map)


    def sum(self, prefix):
        total=0
        for key in self.map:
            if(key.startswith(prefix)):
                total+=self.map[key]
        return total;



# Your MapSum object will be instantiated and called as such:
obj = MapSum()
obj.insert('aa',2)
obj.insert('abb',2)
obj.insert('Ca',2)
param_2 = obj.sum('C')
print(param_2)
```

```
class Solution:
    def checkValidString(self, s):
        self.s=list(s)
        return self.isValid(0,0)

    def isValid(self, index, num_open):
        for x in range (index,len(self.s)):
            if(self.s[x]=='('):
                num_open+=1
            elif(self.s[x]==')'):
                if(num_open==0):
                    return False
                else:
                    num_open-=1
            elif(self.s[x]=='*'):
                try1=self.isValid(x+1,num_open)
                if(try1):
                    return True
                self.s[x]='('
                if(self.isValid(x,num_open)):
                    return True
                self.s[x]=')'
                if(self.isValid(x,num_open)):
                    return True
                return False
        return num_open==0

s=Solution();
print(s.checkValidString("()"))
```

```
class Solution:
    def judgePoint24(self, nums):
        for index1 in range (0,4):
            one=nums[index1]
            for index2 in range (0,4):
                if index1==index2:
                    continue
                two=nums[index2]
                for result1 in [one*two, one/two, one+two, one-two]:
                    for index3 in range (0,4):
                        if index1==index3 or index2==index3:
                            continue
                        three=nums[index3]
                        possible2=[result1*three,result1+three,result1-three,three-result1]
                        if(three!=0 and result1!=0):
                            possible2.append(result1/three)
                            possible2.append(three/result1)
                        for result2 in possible2:
                             for index4 in range (0,4):
                                if index1==index4 or index2==index4 or index3==index4:
                                    continue
                                four=nums[index4]
                                possible3=[result2*four,result2+four,result2-four,four-result2
]
                                if(four!=0 and result2!=0):
                                    possible3.append(result2/four)
                                    possible3.append(four/result2)
                                for result3 in possible3:
                                    if(result3==24):
                                        return True
                    for index3 in range (0,4):
                        if index1==index3 or index2==index3:
                            continue
                        three=nums[index3]
                        for index4 in range (0,4):
                            if index1==index4 or index2==index4 or index3==index4:
                                continue
                            four=nums[index4]
                            possible4=[three*four,three+four,three-four]
                            if(four!=0):
                                possible4.append(three/four)
                            for result2 in possible4:
                                if(result1*result2==24):
                                    return True
        return False

s=Solution()
print(s.judgePoint24([1,2,9,1]))
```

```
class Solution(object):

    def calPoints(self, ops):
        rounds=[]
        for op in ops:
            if(op==’+’):
                rounds.append(rounds[-1]+rounds[-2])
            elif(op==’D’):
                rounds.append(rounds[-1] * 2)
            elif(op==’C’):
                del rounds[-1]
            else:
                num=int(op)
                rounds.append(num)
        sum=0
        for num in rounds:
            sum+=num
        return sum



s=Solution();
print(s.calPoints(["5","2","C","D","+"]))
```

```
from datetime import datetime, timedelta

class Solution(object):
    def subtract_times(self):
        FMT = '%H:%M'
        tdelta = datetime.strptime("01:11", FMT) - datetime.strptime("01:11", FMT)
        print(tdelta)
    def nextClosestTime(self, time):

        digits=list(time)
        digits.remove(":")
        digits=list(map(int,digits))
        FMT = '%H:%M'
        min_diff=-1
        current_time=datetime.strptime(time, '%H:%M')
        next_closest="59:59"
        for a in digits:
            for b in digits:
                if(a*10+b>=24):
                    continue
                for c in digits:
                    if(c>=6):
                        continue
                    for d in digits:
                        new_time=str(a)+str(b)+":"+str(c)+str(d)

                        tdelta = datetime.strptime(new_time, FMT) - current_time
                        if(new_time==time):
                            tdelta+=timedelta(days=1)
                        if(tdelta.days<0):
                            tdelta+=timedelta(days=1)
                        if(min_diff==-1 or min_diff>tdelta):
                            min_diff=tdelta
                            next_closest=new_time
                            #print(next_closest)
        return next_closest




s=Solution();
print(s.nextClosestTime("00:00"))
```

```
class Solution:
    def func(self,s):
        pass

s=Solution();
print(s.func("sss"))
```

```
class Solution:
    def func(self,s):
        pass

s=Solution();
print(s.func("sss"))
```

```
class Solution:
    def repeatedStringMatch(self, A, B):
        chars=set()
        for c in A:
            chars.add(c)

        for c in B:
            if (c not in chars):
                return -1
        multiplier=1
        while(B not in A*multiplier):
            multiplier+=1
            if(len(A)*multiplier > len(B) *4):
                return -1
        return multiplier

sol=Solution()
print(sol.repeatedStringMatch("a",
"a"*1000))
```

```
# Definition for a binary tree node.
class TreeNode:
     def __init__(self, x):
        self.val = x
        self.left = None
        self.right = None

import math
from collections import defaultdict

class Solution:
    def longestUnivaluePath(self, root):
        if root is None:
            return 0
        self.startNode=0

        self.longestFromNode=defaultdict(lambda: 0)
        self.longestPath(root,math.inf,0)
        print(self.longestFromNode)
        length=1
        for key in self.longestFromNode:
            if(self.longestFromNode[key]>length):
                length=self.longestFromNode[key]
        return length - 1

    def longestPath(self, root, value,parent):
        if root is None:
            return 0
        if(root.val==value):
            pos1=self.longestPath(root.left, value,parent)
            pos2=self.longestPath(root.right, value,parent)
            mymax= 1+max(pos1,pos2)
            self.longestFromNode[parent]=mymax
            return mymax
        else:
            self.startNode+=1
            start=self.startNode
            pos1=self.longestPath(root.left, root.val, start)
            pos2=self.longestPath(root.right, root.val, start)
            mymax= 1+pos1+pos2
            self.longestFromNode[start]=mymax
            return 0




a=TreeNode(5)
b=TreeNode(5)
d=TreeNode(5)
e=TreeNode(5)
f=TreeNode(5)
a.left=b
a.right=d
b.left=e
b.right=f
e.right=TreeNode(5)
e.left=TreeNode(1)
f.right=TreeNode(1)
f.left=TreeNode(1)
d.right=TreeNode(1)
d.left=TreeNode(1)
sol=Solution()
print(sol.longestUnivaluePath(a))
```

```
class Solution:
    def knightProbability(self, N, K, r, c):
        self.prob={}
        self.N=N
        #K ,r,c -> prob
        return self.knightProbabilityHelper(K,r,c)
    def knightProbabilityHelper(self, K,r, c):
        #off
        if(r<0 or c<0 or r>=self.N or c>=self.N):
            return 0
        #done
        if(K==0):
            return 1
        else:
            if (K,r,c) in self.prob:
                return self.prob[(K,r,c)]
            sol= (.125*self.knightProbabilityHelper( K-1, r-2, c-1)
                + .125*self.knightProbabilityHelper( K-1, r+2, c+1)
                + .125*self.knightProbabilityHelper( K-1, r-2, c+1)
                + .125*self.knightProbabilityHelper( K-1, r+2, c-1)
                + .125*self.knightProbabilityHelper( K-1, r-1, c-2)
                + .125*self.knightProbabilityHelper( K-1, r+1, c+2)
                + .125*self.knightProbabilityHelper(K-1, r-1, c+2)
                + .125*self.knightProbabilityHelper( K-1, r+1, c-2))
            self.prob[(K,r,c)]=sol
            return sol

sol=Solution()
print(sol.knightProbability(3,2,0,0))
```

```
import math
import sys

sys.setrecursionlimit(10000)

class Solution:
    def maxSumOfThreeSubarrays(self, nums, k):
        self.dict={}
        self.nums=nums
        self.k=k
        return self.maxSumOfThreeSubarraysHelper(0, 3)[1]

    def maxSumOfThreeSubarraysHelper(self, index, arraysLeft):
        if(arraysLeft==0):
            return 0,[]
        if(index>len(self.nums)-self.k):
            return 0,[]
        if((index,arraysLeft) in self.dict):
            return self.dict[(index,arraysLeft)]
        sum=0
        for x in range(index,self.k+index):
            sum+=self.nums[x]

        sum2,indexes2=self.maxSumOfThreeSubarraysHelper( index+self.k, arraysLeft-1)
        pos1=sum+sum2
        sum3,indexes3=self.maxSumOfThreeSubarraysHelper( index+1, arraysLeft)
        pos2=sum3
        sol=0
        if(pos1>=pos2):
            sol=pos1
            ind=[index]+indexes2
        else:
            sol=pos2
            ind=indexes3
        self.dict[(index,arraysLeft)]=(sol,ind)
        return sol,ind

sol=Solution()
print(sol.maxSumOfThreeSubarrays([34890,35360,10252,56522,21768,39702,8000,55992,20392,44858,6
863,52979,50238,22694,50065,37407,25644,15195,6840,46920,52083,39988,15044,38572,24814,18261,1
5606,38593,3160,24405,9882,2622,40073,53516,9832,43694,50691,59048,39606,53407,23963,47620,490
17,31688,60383,3554,59878,53038,62651,61025,64684,11865,41871,13838,11950,44473,44786,35921,31
636,30956,40716,36956,37905,52139,52863,44699,12538,49272,39395,14813,61216,57789,7986,63677,4
5670,29128,5358,39501,12048,22852,52609,48441,37836,56734,8982,18660,45895,41011,53398,7637,30
131,31432,48804,26257,36011,21402,49867,20390,31760,58685,11163,3927,6591,62555,13313,17216,32
145,60226,55255,20435,12670,2156,50587,2617,52213,45180,10103,46108,31743,49411,5138,32906,245
54,1325,48724,20686,56142,7035,44746,27657,63180,22770,26597,26911,45355,24073,44936,30552,457
91,59249,42006,20481,55041,43513,8355,33689,30991,62873,6887,24293,36970,15902,42131,44401,582
67,24110,26784,41944,28771,38435,2632,28956,39760,21538,40985,5438,63784,52632,35382,61976,428
83,4054,11070,19411,23049,59,33501,30126,15336,41394,48484,31599,48070,27927,7274,46265,1558,6
259,22554,16295,61590,31194,35639,2332,15011,59266,40874,10266,21429,56604,46194,21911,27056,2
0618,21253,16393,62487,28875,29817,23006,41347,33195,10851,34054,21712,21690,20222,65508,14072
,30778,64808,40173,6567,59660,37969,18713,55841,29035,55400,28526,19749,44173,5658,63113,64738
,31033,20133,22936,50573,22942,56041,12505,59231,31966,9739,43640,48359,51440,35746,1745,4522,
57541,20189,31372,26794,52991,37708,57991,39877,42126,64905,35236,19185,40740,15684,27096,4873
2,22819,22617,20471,11963,49637,13253,7470,3762,32012,9901,34969,16610,61758,26584,14997,20576
,44548,61386,61852,63620,24178,15091,698,53015,25731,28856,43607,37052,48586,45847,15409,60322
,53435,22063,27853,47143,457,28,21986,52186,36346,20671,37960,42629,10081,40886,62301,17586,55
733,12958,59057,62972,62591,23953,40422,37591,1051,10512,22318,25340,44852,25207,52126,50647,1
0727,45200,27522,63963,2538,44932,31883,3940,2072,45015,26553,31984,24574,52565,5559,15356,636
17,16380,60322,10653,37980,57428,55221,3491,34594,9781,3654,7241,49404,27762,44068,7297,5365,5
3484,58662,58230,21173,60504,49744,13727,51388,49458,60355,50327,21613,21166,47286,44331,61368
,41151,22168,444,27148,13791,20001,10476,55713,61668,51276,61045,53348,26946,22493,5273,65130,
```

57053,9837,48181,57592,56071,9659,18219,36104,48482,44275,21337,36133,65488,6654,45515,52717,12306,753,25575,27553,39789,33853,41264,23479,8442,618,56362,36773,3254,38151,61065,54019,16429,47745,58169,54920,53783,29122,60772,27890,27841,60315,12100,57820,4499,5342,62901,52051,39841,54193,25160,20471,6127,51937,60551,18364,10458,19204,18708,41357,2999,7895,37248,59662,28559,25874,19554,12518,6192,1658,41197,10194,22904,19793,19681,23785,37219,17319,25234,20963,15643,1943,65448,41679,8568,50679,41834,53812,20378,27764,4157,50618,49591,52876,64827,20716,49655,7242,27409,12964,50152,13652,38662,30954,49661,27972,2650,23015,56578,40054,44607,51074,29425,5741,52606,44952,58252,60318,7660,56431,3302,29309,5370,751,13929,27259,10024,24426,55664,40191,64555,2008,64761,15812,63178,37102,31793,46921,35410,14245,37988,55131,27502,55201,6964,51777,50375,14091,22515,41076,9952,22677,44082,55076,30058,43752,35462,47174,32836,51727,1654,646,33678,3441,34303,40163,43153,390,7164,7729,28422,58197,35343,35806,4301,31687,19675,14551,51609,46930,65067,20636,26631,45259,55562,36844,3835,55623,60454,16624,58289,41413,38138,16093,689,61782,11276,16764,9393,47561,28169,26419,59773,16349,29361,64276,14528,63464,26594,22105,60500,8172,45755,33778,5492,14628,56950,41053,59653,28807,8799,4544,57075,57454,36714,41429,35163,26600,43255,58142,19049,47589,26517,62434,30988,15919,35084,17920,14530,24534,56603,9997,4011,15892,26471,33190,17772,42168,50010,52601,60888,46254,2963,50306,13274,30319,4717,9474,43194,42382,31727,9223,58516,38109,42170,45792,48541,52064,26689,20023,21034,7805,59653,51280,1966,43927,681,33109,4883,56806,21390,35711,38405,6039,25521,26728,17338,41677,41761,3785,13524,59717,47767,333,10688,32643,61016,2404,44060,36152,65477,30865,22846,22176,12,9192,34658,37989,38029,58527,51985,648,37617,64907,32875,44124,43322,18127,9672,64817,16083,52752,12158,32624,5278,18137,18948,57749,43245,36872,29893,51705,61068,13330,20477,8755,5451,29819,12655,42200,23739,62095,20499,63383,40397,25326,14542,47057,17244,62557,15225,41246,50216,32469,27744,21687,36737,4304,786,47530,63857,29206,4193,58774,27978,49208,63528,34773,55362,17338,30010,14862,20914,2657,54686,38134,48354,24003,53162,44461,31648,57779,26268,37613,19156,10137,19556,5681,17083,60278,34605,37577,5792,40686,24563,60286,3108,59964,38009,41515,1133,48503,34201,64755,27308,14402,37184,11482,57200,52628,52686,42553,45507,26370,34826,11214,7247,53075,5556,33620,56112,44669,7989,64838,3416,61585,41273,48497,24427,5340,19896,2693,9825,41940,34267,28952,44103,12140,44229,40767,7563,44044,31399,11040,35029,269,48738,13808,6539,20095,49855,48681,1080,2354,28033,5549,42369,53056,17733,53757,5906,12871,8229,22602,57791,39406,31538,5033,45303,44627,64269,13203,46493,19265,22491,20629,17473,51045,56873,45677,20257,34721,58791,39896,11202,15200,31426,40284,61042,31114,43106,55140,50190,47558,39299,43846,8049,29940,40572,62060,35632,40830,17268,20703,43815,42653,40406,51135,15659,45926,44701,61638,3831,22184,11698,56641,6471,22854,55119,64843,35147,10222,33897,63027,4873,36207,943,42495,60395,6958,40703,49289,61609,59988,63168,55050,44774,57718,19845,35630,7891,29479,39092,3813,53494,44065,5727,50439,35382,35035,17015,21420,3720,27894,36306,36276,23797,41218,51296,43989,17730,47275,8667,15467,21932,19839,11273,14568,37241,62759,63811,31878,18578,44179,586,32632,65046,44199,12762,39977,19421,42768,47650,3445,25707,8987,12927,15559,28342,18029,29105,57374,42571,19835,58367,61577,29087,11540,15015,60812,15536,23361,40591,29085,13450,45095,6484,60253,14917,25937,59461,27912,37533,13570,3984,24747,55323,45785,37967,38307,44964,25450,39438,45776,55092,30638,49465,17835,52290,18288,29928,22718,15089,39324,13485,63230,49861,36092,28140,18977,10810,22211,26815,40909,13259,6258,6536,252,26997,44892,15716,32247,1302,38220,26739,7210,56370,4368,23940,8155,13058,19007,47321,57182,58059,64501,32047,24754,49495,31800,55962,27614,41989,51054,133,33426,24468,58223,37914,40446,2932,31887,58077,31539,15585,54531,20287,13576,5397,62085,14252,39971,10822,63821,46749,14924,37928,26991,49727,27177,63725,58442,5305,21458,60147,38727,50902,42309,2623,58356,31863,22791,40269,8568,12880,44733,58066,45827,6620,31968,5662,52701,43327,25323,21898,26442,25718,8713,5852,8854,54585,12161,7912,25791,22419,36631,60223,38275,53558,4859,36111,63296,5873,50299,56885,64510,49716,24254,12409,7372,44778,8300,64143,1246,14593,19414,7292,9181,29665,36980,59456,23731,64486,35362,57368,41322,24864,50573,18944,34446,12837,9016,24215,27541,15065,7585,52410,8261,65325,14262,36844,63869,46305,13702,8847,31820,50497,15729,48975,61541,6821,44912,15984,33739,14578,20362,21070,64777,62537,48344,8285,25383,13851,36706,32651,46152,2966,48252,46585,57167,47634,55784,26430,39799,18210,42373,57326,44431,58151,37752,7350,13349,42207,63540,17182,9358,31894,31077,25904,40345,36245,39139,5545,21652,5130,24541,14922,28514,19487,11983,41302,56731,12634,59755,13210,59575,20628,47232,29309,13311,65205,50917,35335,62004,5437,1229,26647,61886,15438,62441,48085,308,23937,61509,30711,36680,49508,51035,56541,52752,25189,58696,48890,3230,24736,8618,59824,42031,48323,27812,58490,11363,9770,32387,28310,15451,8581,28230,26495,30005,35378,37724,24313,40648,42739,53139,10841,49677,43594,38606,54919,22022,45180,31514,64383,7809,203,30454,8842,2152,10276,35158,11217,23872,42623,45387,5672,10015,19612,13699,2658,1635,4787,50846,39280,62866,12689,24261,59320,26863,26888,37317,2502,44416,35729,34479,38731,24833,27703,28974,60777,22240,55567,30191,15925,13016,45072,38672,45635,57054,26513,27658,1894,15992,9770,26816,63806,22827,15606,47214,7836,36565,28422,17506,41014,25340,42439,57129,58332,57921,30505,27677,46327,3506,31627,8694,293,15572,12941,38261,13404,20794,22855,17937,28537,31135,14320,46731,41260,15098,56815,15687,42991,35027,22075,18624,18586,43255,21203,64488,32086,34838,34969,24721,64580,47698,36418,60782,14401,19282,17505,34661,13096,34442,49761,3046,63519,30

886,574,53241,11760,47012,39347,54528,51106,10737,61161,5328,12033,57256,34968,31946,16523,233
78,28489,47920,14616,53834,34439,1335,35896,59332,20747,63377,54220,6969,3925,55012,31661,1893
7,21982,15727,17633,50756,58325,45556,57689,555,1668,12102,3218,47490,46663,8638,40209,57944,2
9865,37809,28502,14277,14943,65387,56400,52137,3423,57906,55557,29457,17222,4488,50741,58649,3
6305,11711,63935,10897,24652,24519,16894,22287,33956,13484,40248,26549,35894,35364,57446,42344
,41197,36754,53320,51418,16097,33600,51450,34337,30786,35453,62606,52515,14505,1143,54039,2956
7,20702,22294,62402,20055,18905,26905,23933,46142,55499,29093,53177,52172,45514,11983,7311,421
69,47698,56258,15602,33726,8061,2580,9883,16192,37095,48921,18448,222,11183,64189,37570,41545,
14991,50299,5761,50774,17548,3336,34071,14300,29425,17832,58404,7744,60286,18709,8417,62923,39
231,21223,64516,63311,45737,19979,31739,12498,24807,45635,31436,9148,11533,19583,7596,18690,12
799,49710,3465,17004,20450,27327,6490,45340,7031,35896,43859,40597,51552,26268,48181,17341,362
41,8766,27117,58499,1147,31870,45597,60370,26829,62827,40865,12694,39815,28313,5511,17032,5370
5,30525,63838,26200,9009,4991,10553,61481,61569,48112,55814,9931,38920,16107,37095,56775,20806
,32067,38357,42823,27094,19388,9468,11003,9386,31973,11819,50817,47976,44057,8140,19967,26366,
23975,60470,52232,32577,31713,20292,38601,64370,60242,29224,27101,17381,18495,23206,59521,5492
6,1287,64797,20898,48624,48519,38273,61061,23692,24767,60733,12548,58579,32547,62482,46197,157
17,12569,45920,37926,3568,16183,19228,36921,46462,15064,17931,49299,27623,983,54498,23054,7357
,25770,46627,10216,53212,47272,32080,18063,42618,25762,17034,20211,28959,6694,24026,41312,1952
,40086,20759,19864,46877,30879,25204,6696,35688,7069,9244,29523,39330,32868,49486,13962,26676,
44898,63038,42638,11729,21844,5413,21336,45847,17318,21561,59487,28126,40025,9349,31645,1634,2
2965,44350,27467,44663,3224,26866,4697,8662,57538,1230,37345,31783,13519,19103,59307,2939,1742
5,7732,55667,2543,62139,43055,35066,32594,3273,30758,50884,44050,29467,9899,24554,64886,49654,
47930,25197,4026,42778,50997,43464,58749,605,10426,34766,36721,24327,54763,14379,7537,52097,16
694,32321,9827,8041,26530,29432,53799,7262,47733,49649,55319,20009,59735,63514,15046,61630,192
29,46671,17490,46216,37811,4243,2373,6755,39765,60624,28367,12498,16560,55836,29371,44086,4125
7,32009,16394,20680,61750,13411,16369,29866,3371,64129,58776,17098,30912,28349,32530,40481,375
48,54604,22311,25506,8325,42253,12074,55173,59293,23751,8929,13981,37311,51179,3867,26955,7918
,31766,24036,64352,60348,30021,56602,20587,5465,4742,9225,19518,50634,48191,17521,28219,56678,
35714,55075,64047,5281,37124,65483,56623,23471,59802,8587,57425,28888,37694,6797,45103,7204,35
40,7880,25403,58659,27927,38719,63138,45563,64766,16280,61823,47359,27775,52151,38374,57264,37
966,39325,19759,50168,6961,25684,9304,20437,15573,19587,16855,48921,65518,34807,1539,10352,625
57,26282,24759,40967,15790,40003,41482,483,26539,9623,52384,134,2216,31072,41567,1116,45513,20
153,28090,42032,15793,30047,22722,52370,54041,62842,51029,2153,48870,40889,48663,27296,46973,8
044,14744,47311,51490,3115,33711,5027,32516,65494,7212,1113,62707,51655,29303,14769,54467,5203
3,53456,360,118,51800,40877,47529,38434,11244,10999,57813,56953,10377,24668,580,63286,53255,42
603,31058,27895,46425,21925,44166,43535,54815,44400,1789,43620,28501,16444,62490,48201,26512,5
6562,22794,28198,36892,46178,58174,55909,45434,6914,42079,39526,5451,35488,23638,21323,42541,3
9602,48773,41094,7207,9480,9673,11031,8634,13023,56937,55391,60123,24363,54555,17496,9643,3645
0,49665,26352,15602,49833,10854,42330,52962,24034,31561,58838,51278,17238,51365,13557,65176,14
804,49110,50,65430,30566,32349,2070,46450,21783,45719,53894,59062,18609,1071,64536,55135,11767
,7300,1423,3905,24533,23485,5011,42410,708,36330,51512,21134,50259,7154,17905,23222,13218,5164
7,42039,55921,3101,16544,36748,48364,61949,13361,56296,24309,28654,31919,7029,34496,60462,1127
2,5258,43740,53570,19599,20138,48903,45084,33043,47283,13447,15505,55119,13158,6912,26610,3437
1,50320,1578,22621,51970,46615,61925,41895,21051,22271,17724,37066,55238,2990,8146,36736,37709
,29560,45424,1158,46315,18176,19009,29056,15365,62073,58167,61283,41975,46454,41554,6002,17795
,40966,39547,37697,58488,25254,9460,57891,40834,32505,15531,22821,28428,7411,40706,27511,7876,
58275,25009,57309,53182,58144,15157,46308,9281,4460,34862,12201,58417,5410,47055,17974,41188,8
386,55073,59440,49592,14836,59644,52736,57274,18552,12230,35315,12905,934,17118,51625,60206,44
831,42218,56719,23577,40980,55945,16299,42049,63139,34933,55844,53048,42072,59474,7488,27674,6
2349,34839,43306,109,41946,2858,23605,10106,43588,1362,63570,33322,54730,20285,40722,11899,93,
14796,23211,11991,60454,24334,20508,56876,28035,40534,5188,18401,20699,56265,49939,18385,17074
,475,47887,55006,62846,12672,46979,23410,48043,6873,23653,46751,44551,37428,46912,52863,60966,
6810,63534,40333,45685,35143,60137,45330,41007,46939,44,22044,29614,64052,62035,62938,13523,45
401,2819,23773,44881,39220,59475,29593,15586,5021,62116,49581,40786,7655,56247,27619,64230,295
26,48402,18041,47263,20100,3941,20180,4669,12264,41484,25288,14931,65274,21348,59153,38525,352
4,15867,13972,60770,7579,58297,42587,50579,27095,53049,49602,21832,43895,5479,10782,50368,4935
4,51975,3506,9688,27517,51063,62347,64617,8046,31725,2295,32096,55520,17972,35666,14518,5875,2
1206,54432,23507,57646,38624,9544,62487,17937,24403,22587,20280,29292,50215,13215,55691,20213,
12459,50980,34402,5701,51267,5755,31915,170,14609,23666,46249,53722,62277,25662,53168,9579,211
30,28639,28925,57254,38524,25342,22231,50182,11370,44336,48256,20307,30057,35704,64248,38570,3
9264,45162,2616,43214,46067,30535,29376,47990,29594,64970,35747,44465,16299,24430,53716,58100,
37533,62786,18031,64861,20638,22895,41285,9650,57357,65262,51081,49665,7814,53271,42043,51788,
10718,41155,63235,3177,14403,28844,3060,43446,65201,50239,22848,57480,30468,43883,17392,19894,

33416,37684,25944,48755,1128,33406,61636,64161,22027,10784,63520,60306,29751,35557,12533,28788
,33383,58055,42925,44712,56039,15540,50347,48490,23851,28864,23797,12198,44807,45545,15256,651
67,43739,23439,8062,31603,55498,9108,14403,4007,4060,64251,31956,8423,48505,64752,55221,33666,
14022,20594,22493,9244,41970,24837,24329,34540,34649,394,55630,33531,12797,40083,32047,39203,9
26,51536,24682,56504,3780,49282,64658,36557,21019,22267,60263,46901,45149,42772,63452,11010,51
701,25506,40369,38135,14781,54553,25790,39042,53709,24523,37753,36475,53292,6624,42428,54387,7
271,29487,36777,4912,32876,51509,13255,50772,52081,65077,58641,45821,17044,16535,28626,33209,5
5987,10477,54707,55745,49139,58676,42176,15510,31091,20446,46460,20547,6345,42669,64701,504,38
572,52720,50283,18365,47319,46196,45838,15386,29019,28637,659,55340,50608,8616,8880,878,34228,
57536,17404,42211,900,44997,46081,25197,57999,11265,58891,52627,25505,40723,34104,45707,52895,
55960,25274,21843,65440,39852,27770,29221,19149,10642,56053,31391,35458,51380,13022,26281,5087
5,39140,59444,53276,30215,15373,1960,39202,3628,19924,41597,8988,39055,10681,33645,39326,42341
,43949,2169,60233,18168,47383,48437,54352,15661,15581,50754,17070,25284,21812,10997,45446,6237
3,56477,3159,33107,49960,784,47773,3855,47772,10554,61756,64699,50256,12413,21652,23124,59204,
21948,12516,34868,28603,25039,12249,6873,30103,8499,27840,17489,4786,34263,13052,2998,59800,13
39,29485,11988,382,50191,15000,64708,26311,37586,16630,22436,15949,24940,64514,26796,7995,1026
5,32997,1958,16139,3283,19006,26666,57599,6380,30891,18435,33578,34367,23635,6717,32283,7462,1
7660,64749,4336,84,58555,51998,20784,35490,63742,38863,61850,23521,28276,24045,56471,27595,366
90,59824,34759,22977,8248,49418,27934,10434,13493,6809,58079,29335,12508,61223,61065,41286,111
90,31941,15763,53241,28008,58944,44385,45368,14729,63333,48109,45472,33150,13960,18002,33340,7
688,40817,57254,40735,9909,45466,24890,18562,37103,26742,38975,43571,65262,25947,7770,45720,29
228,44909,52289,9581,46341,52349,63608,7979,23422,59565,51738,21357,43941,50037,15035,55291,50
828,61154,2382,63369,659,35648,22581,47900,29012,43052,34236,8114,33273,18256,11957,38989,1878
6,56325,34909,32486,16923,23122,34064,51106,21161,29889,27136,56384,14792,60048,19157,17582,35
764,65358,24264,34998,29629,42776,43464,55649,16753,2056,25814,4477,34305,10908,35793,41606,48
955,47682,15934,38917,58130,35466,29114,43636,33599,22010,62811,60928,37703,50558,24293,55851,
30480,15454,27019,41434,5948,46545,35328,38191,46227,10593,53113,45985,12967,5562,5303,60648,2
3311,1471,40288,44199,53916,43036,64270,65167,5525,47854,9101,46448,25222,58116,20329,34415,14
057,45659,43580,21370,37436,38726,38503,31783,8861,23837,54503,1342,21520,23811,28724,64753,52
031,52706,11512,20038,59760,2329,39345,2805,46550,15028,36905,50054,6439,15219,33962,50340,380
76,46697,56727,58157,36041,16832,18768,39181,39382,28486,7743,62618,50899,11951,11418,46139,64
560,63485,2111,59065,64883,55797,57065,19301,60875,46421,24506,60318,17701,51680,60046,7760,21
442,55378,60798,58837,56605,22202,32559,8156,34125,4084,2049,43325,5802,38922,53595,46869,5814
9,53938,26632,62932,60908,59572,22318,14913,27112,29040,6588,40686,34285,50278,48604,34739,627
84,54641,59495,46925,45122,30658,47441,56979,42769,35312,13186,106,11339,58208,30787,14224,429
07,22832,46600,4100,25500,22645,2160,58456,58317,9461,41142,40615,35404,47610,33267,28734,1484
3,32105,53803,48175,58980,13925,54393,16103,391,15437,54799,39045,49231,44455,10071,63161,3998
3,21704,25728,26093,61871,32170,23644,35626,39992,57014,31924,2480,56697,30714,37248,60820,436
01,56801,22764,43927,20405,55415,25958,20828,10157,50959,23900,37905,21337,34168,47618,25460,4
6074,23804,61343,42385,40475,48601,12751,31614,54887,3927,60528,37890,55180,57845,22309,18731,
3392,8312,54048,38961,51364,36136,41899,37202,52149,52045,58963,35225,28252,29239,52530,65393,
62831,30954,60062,23096,6533,60637,15647,13638,47464,35603,13478,55282,36420,45149,2619,3998,5
5951,60313,44844,8424,48736,23547,64806,33043,57537,32829,43367,34973,31703,20905,40640,55156,
58737,48590,27190,47064,9681,37923,35818,54001,24763,54362,35426,35952,26552,47009,10576,34227
,12091,17366,33199,19411,27447,39090,33573,24551,51922,56800,28076,41360,40386,3028,21355,4705
7,51654,2229,8830,8465,64198,39702,8000,33149,28785,60173,3204,10296,26945,50407,27145,18152,4
6068,6646,3900,25823,1044,57566,1553,33458,40866,30951,55691,54745,15097,45392,20719,19312,112
40,9355,62573,36921,17062,39190,12553,1148,62144,61305,22714,7635,13661,13281,53410,37003,1510
9,37351,51768,55565,48987,53962,30068,64322,22125,27407,37767,32648,12758,12330,35512,55170,55
999,42432,50016,55675,45632,3895,26630,25469,5628,22042,7010,61322,23613,27841,22950,57613,256
05,49414,23249,41381,65354,33445,61975,35242,6434,27716,43991,31637,9552,22950,64637,14307,154
48,56196,14587,59513,3860,29103,61695,29294,40832,19376,47727,12099,20919,54871,4857,61015,422
62,28995,47699,5376,60166,47419,48719,53685,2497,60191,18351,7540,34380,8575,62195,3072,4377,1
2256,38825,9712,64375,8255,619,24385,4197,48849,16521,43631,43764,46768,63463,8507,14515,11957
,54642,9852,22943,43947,54117,50300,47174,64358,31070,61026,29317,56326,44176,47870,51246,4792
,50637,55174,11451,4435,61177,26814,27710,3389,14365,46457,59729,5266,52292,44325,12220,8203,3
1126,47926,44377,5118,57633,10976,57995,13062,13076,45370,60460,7914,43492,28597,33017,10691,2
7851,40691,42398,45084,16737,13488,48176,2327,61736,19859,43099,51383,42839,57381,35914,54245,
13885,61746,37061,50640,29779,59389,977,20134,51828,25505,15991,16361,2035,17247,35640,36369,2
4458,19092,63097,23669,61821,17044,62835,38730,21188,36820,20686,31620,4595,32470,733,41051,43
784,27578,28115,54041,22034,7333,57045,65125,19341,47731,22603,24348,20425,29580,5475,61429,93
62,23055,27602,41419,43102,53980,31394,62284,56719,57305,61603,4631,53641,5760,20251,56861,583
52,8792,29857,2585,57165,20677,23650,53423,346,5052,50180,7485,9875,44256,53630,23124,48048,43

118,24957,18003,239,3248,25829,64798,57186,44200,6589,19118,23196,47059,61646,20063,56248,1901
5,12022,6008,46864,58942,53347,33926,21960,26137,15143,8484,46949,38669,18348,25855,56199,4437
2,55705,60379,15861,50590,1081,1086,12467,34906,13881,64803,13965,55128,42291,48098,24165,5481
4,38872,16699,54855,22597,8068,32878,38242,30058,5169,50348,60624,52047,3712,64700,9740,20752,
61020,18019,6636,6887,21533,35168,51393,61731,8151,11466,11392,11035,34679,60195,9311,30615,35
863,41112,58394,35811,20842,18305,21214,25491,35803,37416,52897,50792,10368,40138,39843,19820,
2704,25356,6623,8122,49387,35225,21629,39104,34206,2319,24718,60385,2367,18419,32063,22367,432
93,63020,37066,51498,42409,43736,43171,63576,17890,49870,63377,21683,22863,7542,8440,53599,188
94,16086,36403,8857,19824,1572,27885,39964,17199,6714,60450,65358,34220,18092,34450,37188,5526
3,32388,59546,24072,45103,59029,44363,1599,42194,48707,10092,55529,48365,12552,43291,31842,329
51,64595,29740,52757,4027,39063,5815,7610,8173,61624,7337,39339,49180,39069,30185,20415,36040,
22421,8985,55426,34974,60478,37268,35644,29251,61313,55434,32943,57681,64971,55718,6978,9725,3
7906,29136,34376,18335,21126,48674,21788,16003,21414,55363,45768,15657,43397,46382,57621,47789
,13952,20605,40336,36604,35675,41131,24637,52746,33462,52762,61121,62336,51531,20376,28816,272
60,3555,43141,3441,40198,51959,31842,15135,10844,14880,3891,12468,26945,64129,13232,29485,4991
1,29728,7455,43925,2950,24215,42591,48160,19340,24468,57553,16205,63915,13195,62852,57275,1459
,1319,26152,58060,65532,9467,64171,33409,20252,57528,49840,42145,52442,30761,4529,18198,45839,
39450,24214,36930,39365,18270,24204,5299,42930,16778,10333,47745,57361,61226,9806,12141,59421,
43361,10073,33846,34567,32701,63681,19473,31357,4125,57359,45525,60945,9660,36258,45162,62032,
63908,10447,26765,13275,57094,32611,19854,52271,59171,40333,39844,44343,20134,37613,40644,4448
8,49276,47210,37751,11803,53091,44771,34639,59247,38001,13946,28823,50908,59459,19476,61288,26
389,31132,36459,28427,45554,18973,1647,59778,8053,13260,44510,43552,55761,1330,1160,63320,4193
9,25432,49005,57457,64743,23678,2507,34441,22732,7821,34278,11050,62431,63678,47338,7313,57664
,46055,42624,15482,52436,58372,58997,18599,10459,20988,21337,61242,56138,54593,45370,8330,9223
,64421,9607,33112,35179,487,34771,25017,64066,10820,6113,45777,41509,58620,35012,20751,47602,6
3326,34776,43567,18131,53970,47161,47021,1843,64448,3169,35331,13630,4221,2640,38016,61592,385
36,18080,13681,15550,22137,21850,55728,26641,23167,54254,6425,56822,33065,16884,34890,39640,12
980,43184,17942,11947,15641,43693,53992,32242,21833,30247,49075,47576,17716,41595,54878,14031,
18928,14039,60381,41514,17618,45680,63378,19201,130,44959,61734,13426,30349,47043,23512,12218,
51921,59768,29368,39510,22312,33498,46287,32677,37223,25628,64959,31980,35047,11097,58642,1828
8,51476,61514,56409,42100,46616,55564,31565,33908,36185,65487,20343,21914,43491,19281,41556,55
468,61113,5711,51758,62533,20952,31649,42191,59248,6403,35282,39551,45278,32167,14115,25996,44
898,34582,14993,9018,19744,51768,4265,53553,3804,49188,64623,43230,26846,31909,1651,28655,3124
9,59846,12140,18620,36707,7894,44011,39608,55100,42952,47055,35588,54221,28501,50888,12815,296
57,10346,24912,18363,26077,33631,58057,21111,57178,59557,46794,26964,30003,22564,31952,15063,8
956,52422,50037,9161,11870,26172,17877,13276,9856,39761,33447,19131,64615,56719,59543,6544,419
76,60664,52842,58522,42158,13782,10017,28484,14965,16411,23365,44391,22558,276,19844,12911,496
56,43135,53668,16671,41046,23065,36123,2797,38540,5055,11585,62148,48746,43160,35594,16912,594
20,15458,3990,14308,48522,55326,3594,47631,23704,10743,33257,52847,54388,1046,25768,31887,4530
6,22398,38144,60554,6612,60359,49107,20532,1585,11583,53492,54508,6541,44231,22326,7738,8220,6
747,41342,37424,52542,57876,61018,17568,34999,19601,23544,2332,42706,36275,56999,11026,32714,8
841,48653,44343,51233,48837,29535,61149,32389,37517,63700,29599,15937,47897,54893,17133,25719,
51407,17950,47129,22161,62079,22598,38052,48395,38423,39562,16822,46139,39624,42209,43775,6237
4,58127,55799,29830,55738,41323,49292,37582,13824,51805,43206,16129,2570,63200,9845,54398,1537
9,31881,30820,57285,4356,62892,8141,42815,36888,6220,38207,38570,39900,9502,36192,14073,49138,
32512,10766,13024,20474,46507,29203,16244,56422,24048,29653,6663,49402,241,5129,54278,12971,58
282,9808,62528,26677,60873,6687,24119,64056,39459,55878,14836,58309,21580,63266,11226,35803,21
461,31124,30083,30549,144,36797,46461,30438,25891,55981,2853,52061,26145,9368,40409,5339,62006
,62848,40568,25761,214,48719,50272,51962,17117,49173,47684,4880,54878,9472,38680,32701,14256,5
1988,32449,12671,45874,63169,34607,3414,63225,60480,21751,6688,54864,9121,17344,38846,27146,57
320,51302,44897,37718,47092,8043,1121,2900,18397,25567,2195,2541,46197,43304,48902,4558,34382,
64800,28536,42646,20208,46892,31550,52666,46724,788,32696,26123,31297,16146,31976,31788,29575,
54798,20796,25265,57374,52522,55413,15780,9358,3982,5264,58947,9257,27346,16792,40239,51748,93
67,31908,36952,51174,65125,31890,23257,41287,38315,55039,35410,46712,52199,57181,22761,314,478
24,53828,65342,57710,38351,469,34160,52191,31051,9172,51461,54764,43756,63771,60562,9978,18620
,53883,33452,57010,31619,52990,51917,61120,62040,10981,19550,51585,11230,5610,34626,207,42776,
44430,44494,53019,8865,43330,41068,817,32775,23668,25024,55506,37957,20646,18635,48464,6778,64
590,26142,3840,64041,40405,42261,1628,22507,62517,39653,7763,50699,47239,19135,940,19970,10945
,33355,54939,36543,539,444,22204,40755,15113,10315,1544,25624,61750,65259,57369,21427,58785,26
605,11376,25882,39801,42277,47407,36556,18329,50527,44208,2653,22407,60593,57007,31982,28833,6
4120,2572,31157,3195,45749,26515,5898,52812,3818,52010,16078,32453,55400,31461,5623,5432,16095
,21740,14012,27411,46501,3068,28611,65228,44443,39951,13724,48707,162,61070,63972,9379,55528,2
5413,3715,59458,65418,51716,1931,1918,3323,41094,33938,21509,30480,20659,58818,11091,45671,626

```
38,27219,12188,16275,1367,58610,24676,42226,61379,33600,1775,22711,40854,20068,59436,23325,351
65,17713,30338,7779,3309,39045,56695,45846,27244,18037,26099,46353,35573,60757,8179,64337,3113
6,20672,20261,42708,29953,59799,60502,44600,12077,7026,45791,26402,25954,42340,14120,6348,6157
6,62511,13338,14580,24281,24599,26198,38420,3085,16074,15868,40898,25182,37197,9400,21922,809,
26958,14446,55342,49989,38680,44937,20464,11833,8774,47411,7163,28874,48062,58640,22104,54798,
54483,18654,52393,63031,7456,15560,14345,31838,52456,9568,14351,18540,11970,982,21400,49065,42
261,46147,52265,45178,24153,64371,24736,58570,55490,51193,6409,21505,29622,58592,28461,32874,2
8338,7203,38962,33550,23219,45552,2386,19047,33139,25902,41340,39022,17185,38871,3384,55799,32
740,45233,54903,51143,38040,6265,42486,51779,55770,47287,23886,1768,9402,43504,8512,48646,7341
,60199,65331,20473,20759,19575,48264,40362,33925,41422,29041,47239,40406,6222,18930,24992,524,
34489,11803,23678,1652,12190,54665,1007,4717,50865,51143,51161,28361,43334,62716,19447,40832,4
8123,30757,52223,25239,4859,26218,52991,26877,46920,21852,28280,38123,30351,45050,38057,40249,
16813,6176,6300,11223,25029,46753,1392,55852,44394,51460,37783,61517,19612,52292,42652,31511,5
5304,47822,8291,56472,28833,47057,4102,45097,16504,64857,15636,57564,57418,41051,53555,44882,6
3251,6362,36214,2141,11698,31826,31293,37784,43395,49683,2548,35743,29559,43660,27237,32817,35
246,9451,22832,61543,28660,50038,47948,23269,42878,44669,6600,60235,12387,38587,32597,24905,14
120,59217,64551,7990,56463,44766,31969,12381,18300,61603,14011,8566,6985,64995,44387,1624,7777
,16945,50171,44605,34230,10614,10173,42412,13938,48710,59182,30220,60949,33715,20180,36670,338
28,30871,54013,62246,53342,6346,56245,16902,61341,2897,59187,62156,41906,48915,9999,58537,2156
6,5338,22646,42220,23825,3064,10296,38734,16654,53469,52905,20985,13831,62612,5296,19169,9885,
2889,38080,39190,44124,41725,29008,2317,42569,19905,2781,48070,41832,57787,24160,31216,9108,45
848,29064,54039,2230,65076,33828,11571,21202,21505,40498,64713,14462,53496,39675,12432,32242,5
3076,41262,35421,57004,11662,57614,8545,59397,47831,12073,38109,59948,18641,38252,55650,44534,
40072,39465,8408,49818,27584,21561,41189,10189,14720,57955,13875,9209,55604,10546,9009,42696,6
0387,25299,61978,72,27566,64003,30559,19306,9419,1880,12213,10319,36565,18885,34546,12840,2904
1,284,39997,43341,45491,9241,16575,42359,11953,13416,9662,64898,60804,23856,42946,16930,35229,
40417,43314,54700,49301,29946,50142,56062,45766,290,12552,33638,13149,9213,1118,49597,24874,45
545,51802,9696,56646,4859,56711,6057,63219,26538,60203,63218,13778,63226,57176,42415,53930,236
72,30334,31756,48495,20692,13125,32667,54983,24128,29153,42486,11505,16470,6912,44395,60503,57
875,27309,12939,44017,14674,45798,34163,48906,28310,25652,38877,15477,52198,64372,46737,51729,
15214,2060,10410,730,16895,28276,63061,23295,49063,45725,16209,37915,4288,58206,318,58695,4585
9,54991,49717,56386,32201,277,52664,37241,57286,4678,47042,30575,61385,33080,31323,46785,23025
,29636,45401,60196,22845,309,12141,8535,38941,45778,56454,38923,18786,63715,35883,56894,4906,1
7605,53330,32601,40890,58425,24091,59233,40638],659))
```

```
class Solution:
        def average(self,M, row_num, col_num):
                rows=len(M);
                cols=len(M[0]);

                num_valid_points=0;
                total=0;
                for i in range (row_num-1, row_num+2):
                        for j in range (col_num-1, col_num+2):
                                if(0<=i and i<rows and 0<=j and j<cols):
                                        num_valid_points+=1;
                                        total+=M[i][j];

                return math.floor(total/num_valid_points);
        def imageSmoother(self, M):
                rows=len(M);
                cols=len(M[0]);

                Res= [[0 for x in range(cols)] for y in range(rows)];

                for i in range (0,rows):
                        for j in range (0,cols):
                                Res[i][j]=self.average(M,i,j);
                return Res;
```

```
import math
class Solution:
        def widthOfBinaryTree(self, root):
                level = [[root,0]]
                nextLevel = []
                max_width=0

                while len(level)!=0:
                        leftmost=level[0][1];
                        rightmost=level[0][1];
                        for node in level:
                                if node[0].left != None:
                                        nextLevel.append([node[0].left, 2*node[1]])
                                if node[0].right != None:
                                        nextLevel.append([node[0].right, 2*node[1]+1])
                                if(node[1]<leftmost):
                                        leftmost=node[1]
                                if(node[1]>rightmost):
                                        rightmost=node[1]
                        width=rightmost-leftmost+1;
                        if(max_width<width):
                                max_width=width
                        level = nextLevel
                        nextLevel = [];

                return max_width
```

```
class Solution:
    def checkPossibility(self, nums):
        """
        :type nums: List[int]
        :rtype: bool
        """
        modified=False;
        for x in range (0,len(nums)-1):
                if( nums[x] <= nums[x+1]):
                        continue
                else:
                        if(modified):
                                return False
                        else:
                                modified=True
                                if(x==0 or nums[x-1] <=nums[x+1]):
                                        nums[x]=nums[x+1]
                                else:
                                        nums[x+1]=nums[x]
                                #print(nums)
        return True
```

```
class Solution:

        def pathSum(self, nums):
                """
                :type nums: List[int]
                :rtype: int
                """
                overall_total=0
                mydict={}
                for num in nums:
                        depth=num//100
                        pos=(num%100)//10-1
                        val=num%10
                        node_num=(1<<(depth-1))+pos
                        mydict[node_num]=val
                for key in mydict:
                        if (key*2 in mydict or key*2+1 in mydict):
                                continue #not leaf
                        total=mydict[key]
                        parent=key//2
                        while(parent in mydict):
                                total+=mydict[parent]
                                parent=parent//2
                        overall_total+=total
                return overall_total
```

```
class Solution:
    def constructArray(self, n, k):
        """
        :type n: int
        :type k: int
        :rtype: List[int]
        """
        total=n
        num_in_order=n-k
        num_diff=k
        arr=[]
        for x in range (0,num_diff):
            if(x%2==0):
                arr.append(total-x//2)
            else:
                arr.append(x//2+1)
        start=arr[num_diff-1]
        for x in range (0,num_in_order):
            if(num_diff%2==0):
                arr.append(start+x+1)
            else:
                arr.append(start-x-1)
        return arr
```

```
class Solution:
    def __init__(self):
        self.current_min = -1
        self.current_second_min=-1;


    def findSecondMinimumValue(self, root):
        if(root==None):
                return self.current_second_min;
        elif(  root.val < self.current_min or self.current_min == -1):
                self.current_second_min=self.current_min;
                self.current_min=root.val;
        elif( ( root.val < self.current_second_min or self.current_second_min == -1) and self.
current_min != root.val ):
                self.current_second_min=root.val;
        self.findSecondMinimumValue(root.left);
        self.findSecondMinimumValue(root.right);
        return self.current_second_min;
```

```
class Solution:
        def What(self, root, L, R):
                while(root and R < root.val):
                        root=root.left;
                while (root and root.val < L):
                        root=root.right;
                if(not root):
                        return None
                root.left=self.What(root.left,L,R);
                root.right=self.What(root.right,L,R);
                return root;
        def trimBST(self, root, L, R):
                root=self.What(root, L, R);
                root=self.What(root, L, R);
                return root;
```

```
class Solution:

    def maximumSwap(self, num):
        num=list(str(num))
        num_sorted=num[:];
        num_sorted.sort();
        num_sorted=num_sorted[::-1]
        diff_value=-1;
        diff_index=-1;
        diff2_value=-1;
        for x in range(0, len(num)):
            if(num[x]!=num_sorted[x]):
                diff_value=num[x];
                diff2_value=num_sorted[x];
                diff_index=x;
                break;
        if(diff_index==-1):
            return int("".join(num));
        diff2_index=-1;
        for x in range(len(num)-1,-1,-1):
            if(num[x]==diff2_value):
                diff2_index=x;
                break;
        sol="";
        for x in range(0, len(num)):
            if(x==diff2_index):
                sol+=diff_value
            elif(x==diff_index):
                sol+=diff2_value
            else:
                sol+=num[x]
        return int("".join(sol));
```

```
class Solution(object):
    def updateMatrix(self, matrix):
        self.matrix=matrix;
        self.rows=len(self.matrix)
        self.cols=len(self.matrix[0])
        self.total=self.rows*self.cols;
        self.result=[[-1 for i in range(self.cols)] for j in range(self.rows)]
        self.done=0
        self.count_zeros();
        self.zeros=len(self.zeros_locs)
        #if lots of zeroes
        if (self.zeros/self.total>.2):
            self.nonzero_find_closest_distance()
        else:
            self.zero_find_closest_distance()
        return self.result;
    def count_zeros(self):
        zeros_locs=[];
        for i in range (0,self.rows):
            for j in range (0,self.cols):
                if self.matrix[i][j]==0:
                    zeros_locs.append([i,j])
        self.zeros_locs=zeros_locs;
    def nonzero_find_closest_distance(self):
        for i in range (0,self.rows):
            for j in range (0,self.cols):
                if self.matrix[i][j]==0:
                    self.result[i][j]=0
                else:
                    sol=False
                    dist=0
                    while(not sol):
                        for delim in range (0,dist+1):
                            if(sol):
                                break
                            for x,y in [(delim,dist-delim),(-delim,dist-delim),(-delim,-dist+d
elim),(delim,-dist+delim) ]:
                                other_i,other_j=(x+i,y+j)
                                if self.in_bounds(other_i,other_j) and self.matrix[other_i][ot
her_j]==0:
                                    self.result[i][j]=dist
                                    sol=True
                                    break;
                        dist+=1

    def zero_find_closest_distance(self):
        for dist in range (0,self.rows+self.cols):
            for loc in self.zeros_locs:
                for delim in range (0,dist+1):
                    for x,y in [(delim,dist-delim),(-delim,dist-delim),(-delim,-dist+delim),(d
elim,-dist+delim) ]:
                        i,j=(x+loc[0],y+loc[1])
                        if self.in_bounds(i,j) and self.result[i][j]==-1:
                            self.result[i][j]=dist
                            self.done+=1
                            if(self.done==self.total):
                                return

    def in_bounds(self, i, j):
        return (0<=i and i<self.rows
            and 0<=j and j<self.cols)
```

```
s=Solution();
print(s.updateMatrix([[0,1,0],[2,1,2],[2,0,2]]))
```