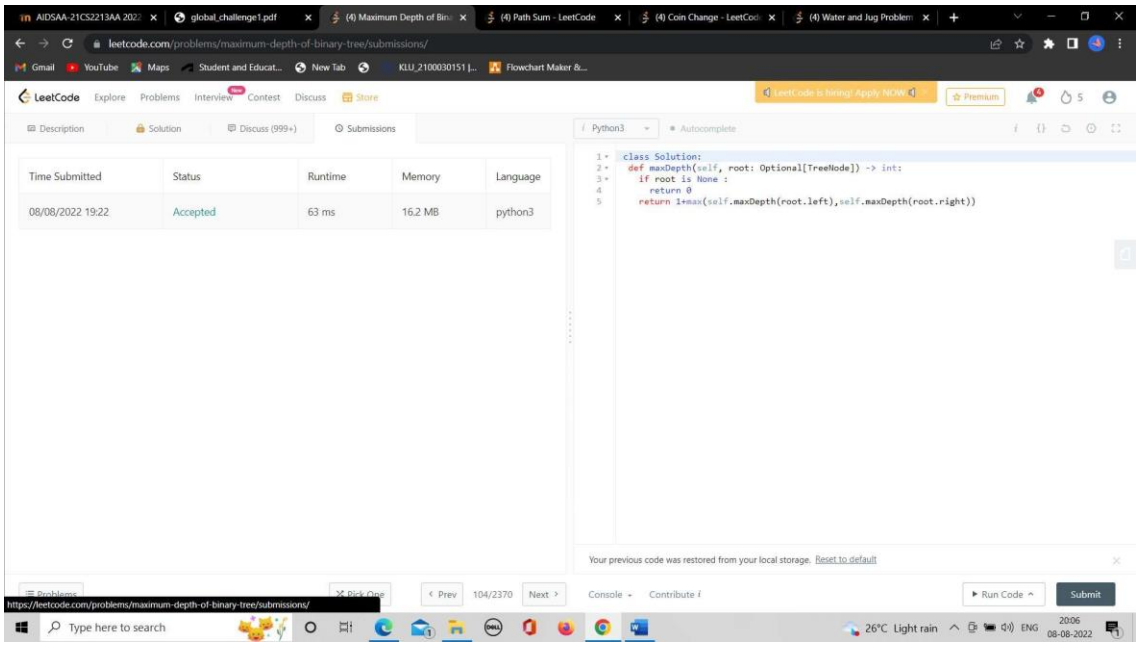
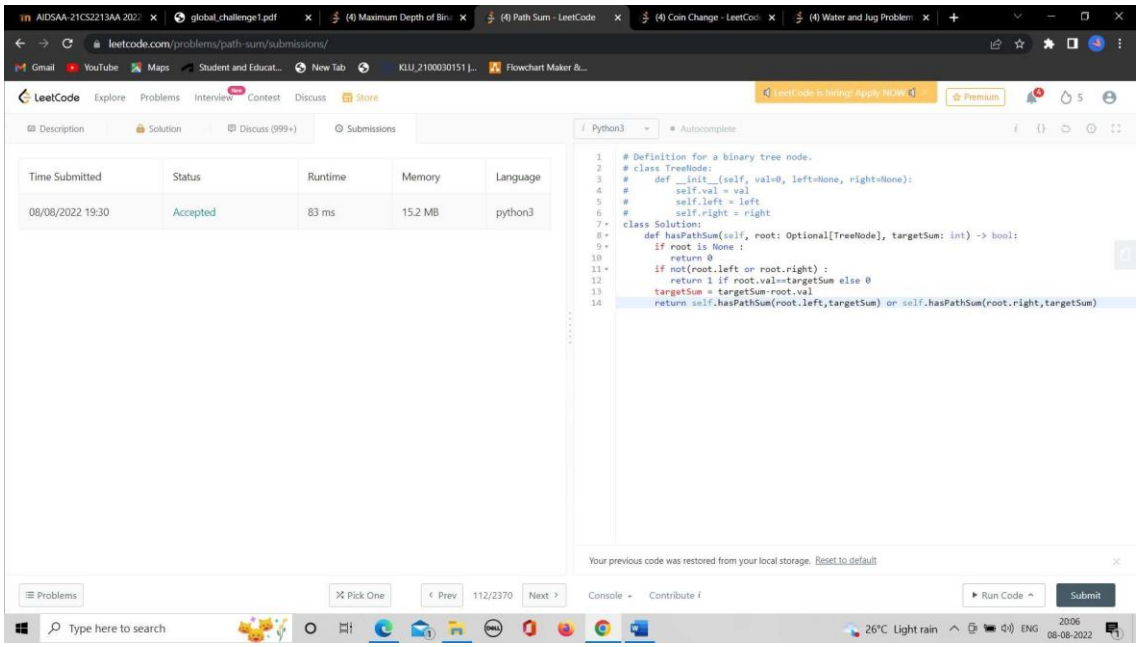
1

```
class Solution:
 def maxDepth(self, root: Optional[TreeNode]) -> int:
if root is None :
    return 0    return
1+max(self.maxDepth(root.left),self.maxDepth(root.right))
```



2 class Solution: def hasPathSum(self, root: Optional[TreeNode], targetSum: int) -> bool:

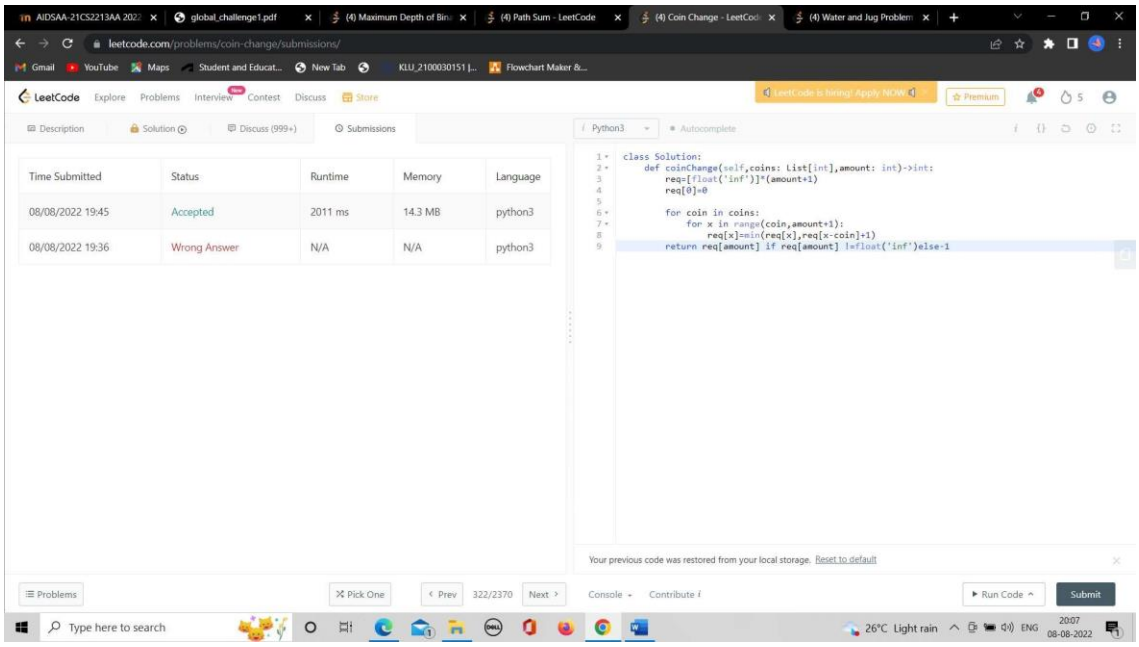
```
    if root is None :
        return 0    if not(root.left
or root.right) :
        return 1 if root.val==targetSum else 0
    targetSum = targetSum-root.val    return self.hasPathSum(root.left,targetSum) or
self.hasPathSum(root.right,targetSum)
```



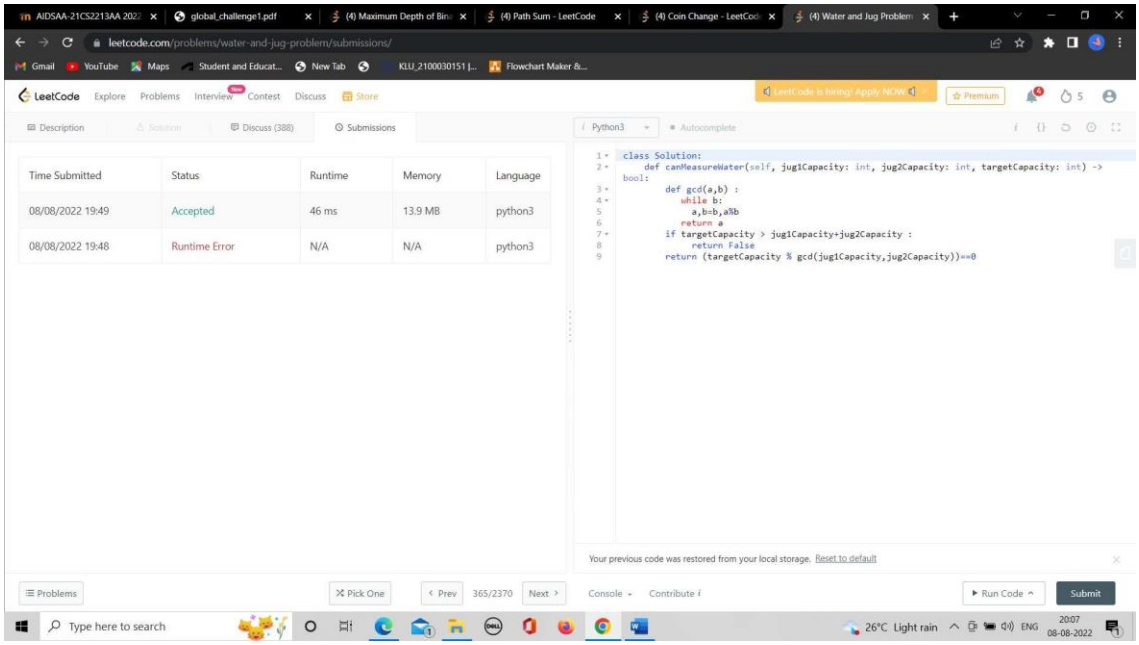
4 class

Solution:

```
    def coinChange(self,coins: List[int],amount: int)->int:
        req=[float('inf')]*(amount+1)
req[0]=0

        for coin in coins:
            for x in range(coin,amount+1):
                req[x]=min(req[x],req[x-coin]+1)          return
req[amount] if req[amount] !=float('inf')else-1
```



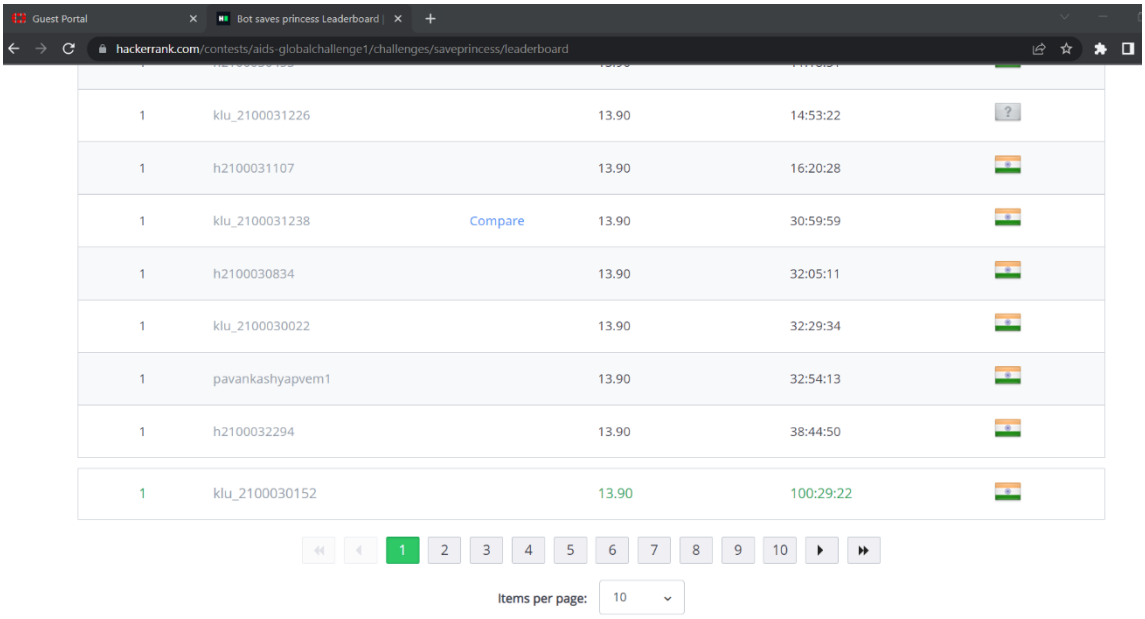

```
class Solution:
    def canMeasureWater(self, jug1Capacity: int, jug2Capacity: int, targetCapacity: int) -> bool:
        def gcd(a,b) :            while b:
a,b=b,a%b            return a        if targetCapacity >
jug1Capacity+jug2Capacity :
            return False
        return (targetCapacity %
gcd(jug1Capacity,jug2Capacity))==0
```



HACKER RANK-KLU_2100030152

```
1 def
displayPathtoPrincess(n,grid):
#first, find where p is located
flag = True     n = n-1
while(flag):        if grid[0][n] ==
'p':
            flag = False
a = 0           b = n
if grid[n][0] == 'p':
            flag = False
a = n           b = 0
if grid[0][0] == 'p':
            flag =
False           a =
0           b = 0

        if grid[n][n] ==
'p':            flag =
False           a = n
b = n
        #else:
            #print("Something broke?") #Why does this execute?
        y = a - int(n/2)
x = b - int(n/2)
while 0 != y:           if
y < 0:              y =
y+1
print("UP")           if y
> 0:              y = y-1
print("DOWN")
```

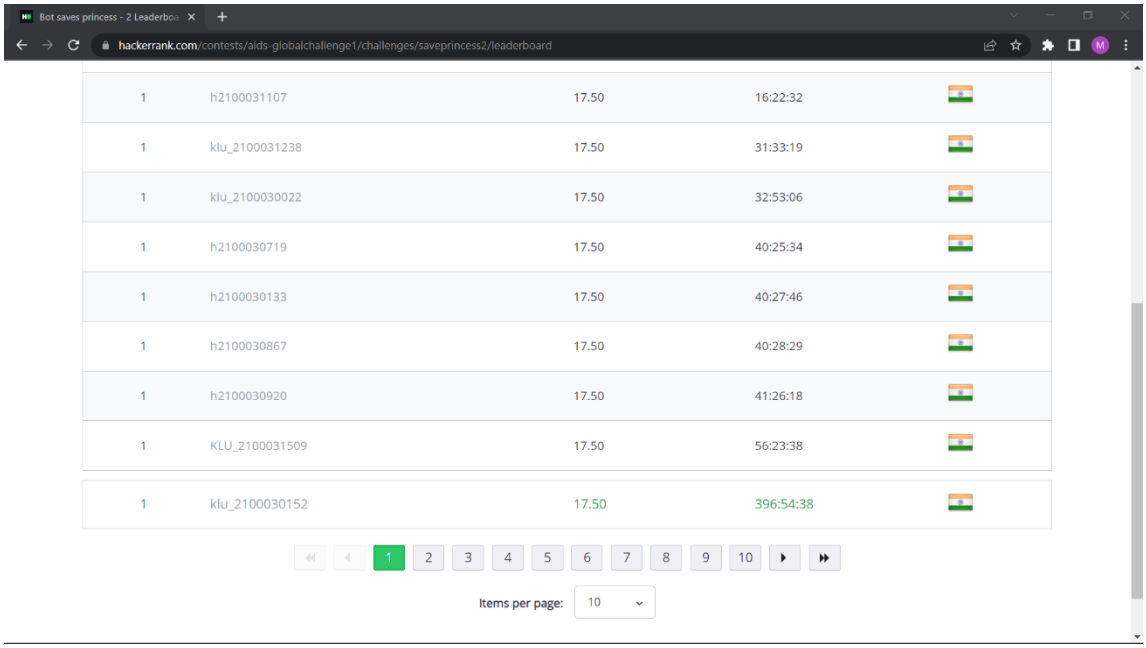
```
while 0 != x:            if
x <0:                x = x+1
print("LEFT")            if
x >0:                x = x-1
print("RIGHT")
m = int(input())
grid = []  for i in
range(0, m):
    grid.append(input().strip())

displayPathtoPrincess(m,grid)
```




```
2 def
nextMove(n,r,c,grid):
    pos_col_m = c
    pos_row_m = r
    pos_col_p =
pos_row_p = 0

    for i in range(n):
```

```
        line = len(grid[i])
        for j in range(line):
            if grid[i][j] == 'p':
                pos_row_p = i
                pos_col_p = j

    # Verify the positions
of the bot with the
princess
    if pos_row_m <
pos_row_p:
        pos_row_m =
pos_row_m + 1
        return 'DOWN'
    elif pos_row_m >
pos_row_p:
        pos_row_m =
pos_row_m - 1
        return 'UP'

    if pos_col_m <
pos_col_p:
        pos_col_m =
pos_col_m + 1
        return 'RIGHT'
    elif pos_col_m >
pos_col_p:
        pos_col_m =
pos_col_m - 1
        return 'LEFT'
```

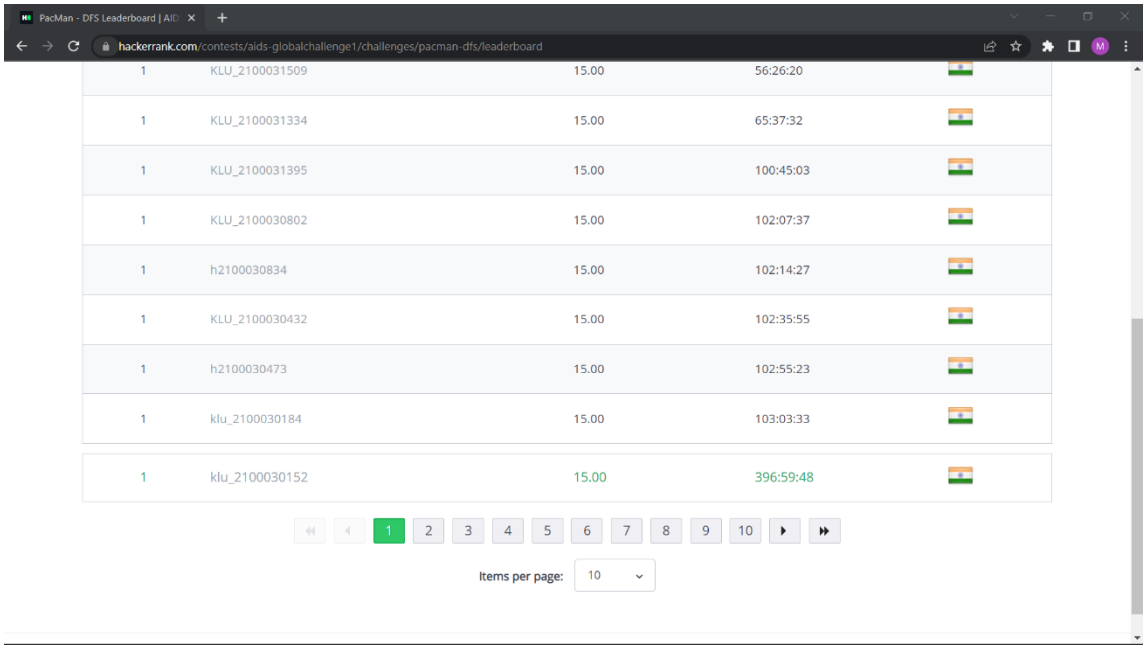
```
# Set the data
n = int(input())
r,c = [int(i) for i in
input().strip().split()]
grid = []
for i in range(0, n):
    grid.append(input())

# print the first move here
print(nextMove(n,r,c,grid
))
```




```
3 import
copy

pacman_a, pacman_b= list(map(int,
input().split())) food_a, food_b = list(map(int,
input().split())) h, m = list(map(int, input().split()))

grid = []
```

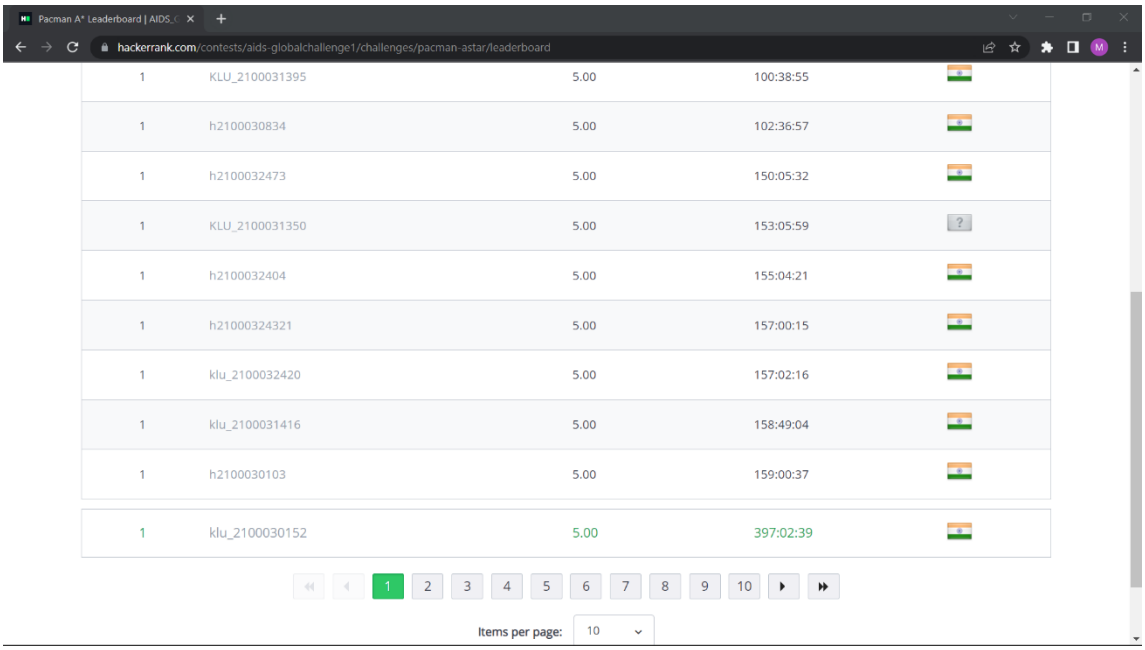
```
node_expanded = []
stack = []
answer_routes = None

for i in range(0, h):
    grid.append(list(map(str, input())))

directions = [[-1, 0], [0, -1], [0, 1], [1, 0]]

stack.append([pacman_a, pacman_b, []])
while len(stack) > 0:     a, b, r =
stack.pop()     routes = copy.deepcopy(r)
routes.append([a, b])

    node_expanded.append([a, b])

    if   a   ==   food_a   and   b   ==
food_b:         if answer_routes ==
None:                 answer_routes =
routes
            break

    for direction in directions:
        next_a, next_b = a + direction[0], b + direction[1]         if
next_a < 0 or next_a >= h or next_b < 0 and next_b >= h:
            continue

        if grid[next_a][next_b] == "-" or grid[next_a][next_b] == ".":
            grid[next_a][next_b]                  =                  '='
stack.append([next_a,              next_b,              routes])
```

```
print(str(len(node_expanded))) for point in
node_expanded:
    print(str(point[0]) + " " + str(point[1]))

print(str(len(answer_routes) - 1)) for
point in answer_routes:
print(str(point[0]) + " " + str(point[1]))
```



```
4 import copy

pacman_x, pacman_y = list(map(int, input().split()))
food_x, food_y = list(map(int, input().split()))
n, m = list(map(int, input().split()))
grid = []
queue = []
answer_routes = None

for i in range(0, n):
    grid.append(list(map(str, input())))

directions = [[-1, 0], [0, -1], [0, 1], [1, 0]]
```

```
queue.append([pacman_x, pacman_y, [], 0])
while len(queue) > 0:
    x, y, r, score = queue.pop(0)
    routes = copy.deepcopy(r)
    routes.append([x, y])

    if x == food_x and y == food_y:
        if answer_routes == None:
            answer_routes = routes
            break

    possible_moves = []
    for direction in directions:
        next_x, next_y = x + direction[0], y + direction[1]
        if next_x < 0 or next_x >= n or next_y < 0 and next_y >= n:
            continue

        if grid[next_x][next_y] == "-" or grid[next_x][next_y] == ".":
            grid[next_x][next_y] = '='
            possible_moves.append([next_x, next_y, score + abs(food_x - next_x) + abs(food_y -
next_y)])

    possible_moves.sort(key = lambda x: x[2])
    for move in possible_moves:
        queue.append([move[0], move[1], routes, score])

print(str(len(answer_routes) - 1))
for point in answer_routes:
    print(str(point[0]) + " " + str(point[1]))
```



5 import copy

```
pacman_x, pacman_y = list(map(int, input().split()))
food_x, food_y = list(map(int, input().split()))
n, m = list(map(int, input().split()))
grid = []
node_expanded = []
queue = []
answer_routes = None

for i in range(0, n):
    grid.append(list(map(str, input())))

directions = [[-1, 0], [0, -1], [0, 1], [1, 0]]

queue.append([pacman_x, pacman_y, []])
while len(queue) > 0:
    x, y, r = queue.pop(0)
    routes = copy.deepcopy(r)
    routes.append([x, y])

    node_expanded.append([x, y])
```

```
    if x == food_x and y == food_y:
        if answer_routes == None:
            answer_routes = routes
            break

    for direction in directions:
        next_x, next_y = x + direction[0], y + direction[1]
        if next_x < 0 or next_x >= n or next_y < 0 and next_y >= n:
            continue

        if grid[next_x][next_y] == "-" or grid[next_x][next_y] == ".":
            grid[next_x][next_y] = '='
            queue.append([next_x, next_y, routes])

print(str(len(node_expanded)))
for point in node_expanded:
    print(str(point[0]) + " " + str(point[1]))

print(str(len(answer_routes) - 1))
for point in answer_routes:
    print(str(point[0]) + " " + str(point[1]))
```