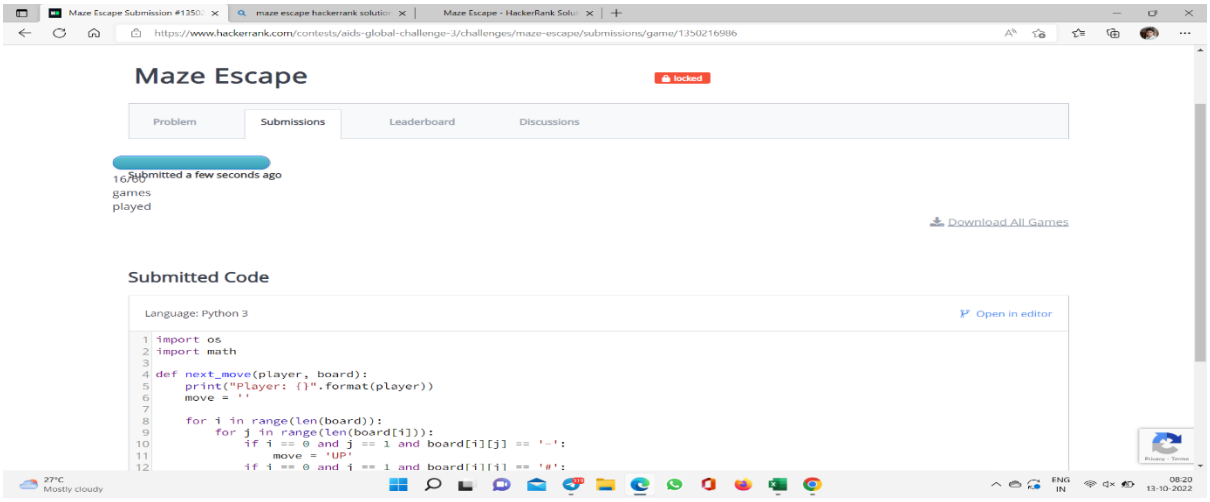
AIDS Global challenge-3

HackerRank

1.Maze Escape

Code:

```
import os

import math

def next_move(player, board):

    print("Player: {}".format(player))

    move = ''

    for i in range(len(board)):

        for j in range(len(board[i])):

            if i == 0 and j == 1 and board[i][j] == '-':

                move = 'UP'

            if i == 0 and j == 1 and board[i][j] == '#':

                move = 'RIGHT'



    print(move)

if __name__ == "__main__":

    # Set data

    player = int(input())

    board = [[j for j in input().strip()] for i in range(3)]

    next_move(player, board)
```



2.Lies Unknown:

Code:

```
from random import randint,seed
import sys


def try_max(equals, notequals, N, L, K, R):
    equals_f = sorted([(k, v) for k, v in equals.items()], key=lambda x: len(x[1]), reverse=True)
    for eq_entry in equals_f:
        As = eq_entry[0]
        if (len(equals[As]) > 0 and len(notequals[As]) < N*(K-1)//(K) + R):
            return As
    return None


def try_question(equals, notequals, N, L, K, R):


    loops = N*5
    while loops > 0:
        loops -= 1
```

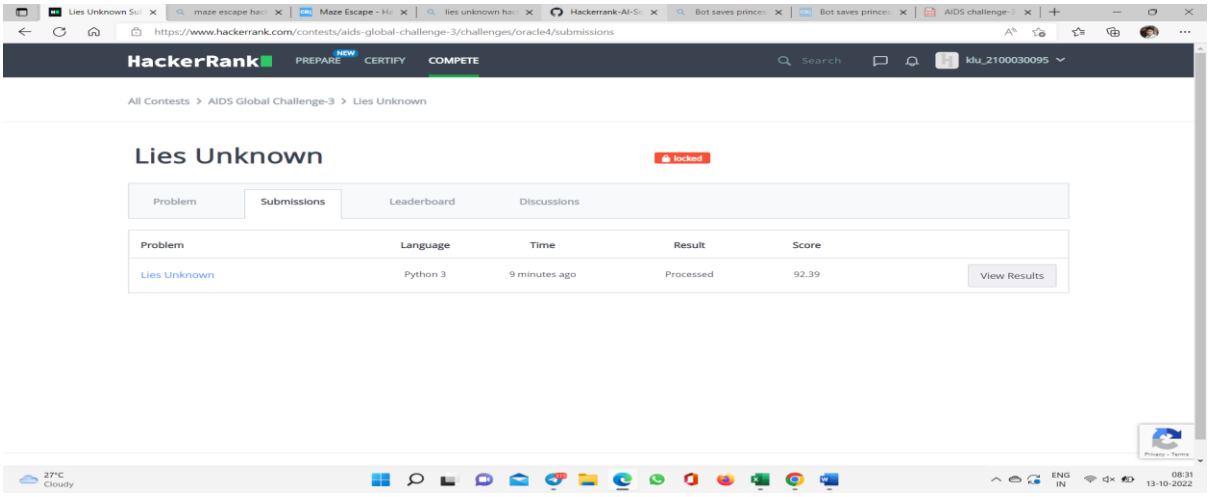
```
        As = try_max(equals, notequals, N, L, K, R)
        if (As == None):
            As = randint(1,N*(K-1)//(K))-1
        Bs = randint(1,N)-1

        if (Bs == As):
            continue
        if len(notequals[As]) >= N*(K-1)//K + R:
            continue
        if len(notequals[Bs]) >= N*(K-1) //K + R:
            continue

        if As in equals[Bs] or Bs in equals[As]:
            continue
        if As in notequals[Bs] or Bs in notequals[As]:
            continue
        return (As, Bs)
    return None

def tryGuess(equals_s, notequals, N,L, K):
    equals_f = [(k,v) for k,v in equals_s.items() if (len(v) >= N//K + L  )]
    if len(equals_f) > 0:
        return equals_f[0][0]
    else:
        return None
```

```
    def doGuess(equals_s, notequals, N, L, K, R):

        equals_f = sorted([(k,v) for k,v in equals_s.items()], key=lambda x :
len(x[1]), reverse=True)
        for eq_entry in equals_f:
            As = eq_entry[0]
            if (len(equals[As]) > 0 and len(notequals[As]) < N*(K-1)//(K)+R ):
                return As
        return None




    N, pl_flag, L, K, L_max = map(int, input().split())
    D = int(input())
    allconds = []

    min_guesses = (L + 1) * N / 2

    equals, notequals = {k:set() for k in range(N)},{k:set() for k in range(N)}
    for i in range(D):
        As, Bs, resp = input().split()
        A, B = int(As), int(Bs)
        allconds.append({"A": A, "B": B, "R": 1 if resp == 'YES' else 0})

    for cond in allconds:
        if cond["R"] == 1:
```

```
        equals[cond["A"]].add(cond["B"])
        equals[cond["B"]].add(cond["A"])
    else:
        notequals[cond["A"]].add(cond["B"])
        notequals[cond["B"]].add(cond["A"])

while True:
    added = False
    for k,v in equals.items():
        for ve in v:
            if ve not in equals[k]:
                equals[k].add(ve)
                added = True
            if k not in equals[ve]:
                equals[ve].add(k)
                added = True

    if not added:
        break




from itertools import combinations
if (K == 2):
    while True:
        added = False
```

```
        for k, v in notequals.items():


            eqc = combinations(v,2)
            for eq in eqc:
                if eq[0] not in equals[eq[1]]:
                    equals[eq[1]].add(eq[0])
                    added = True
                if eq[1] not in equals[eq[0]]:
                    equals[eq[0]].add(eq[1])
                    added = True
        if not added:
            break
max_guesses = min_guesses*5
tg = tryGuess(equals, notequals, N, L, K)
if (tg != None):
    print(tg)
else:
    done = False
    R = 0
    while not done:
        if (D < max_guesses):
            tq = try_question(equals, notequals, N, L, K, R)
        else:
            tq = None
        if (tq != None):
            print(tq[0], tq[1])
```

```
            done = True
        else:
            tg = doGuess (equals, notequals, N, L, K, R)
            if (tg != None):
                print(tg)
                done = True
        R += 1
```



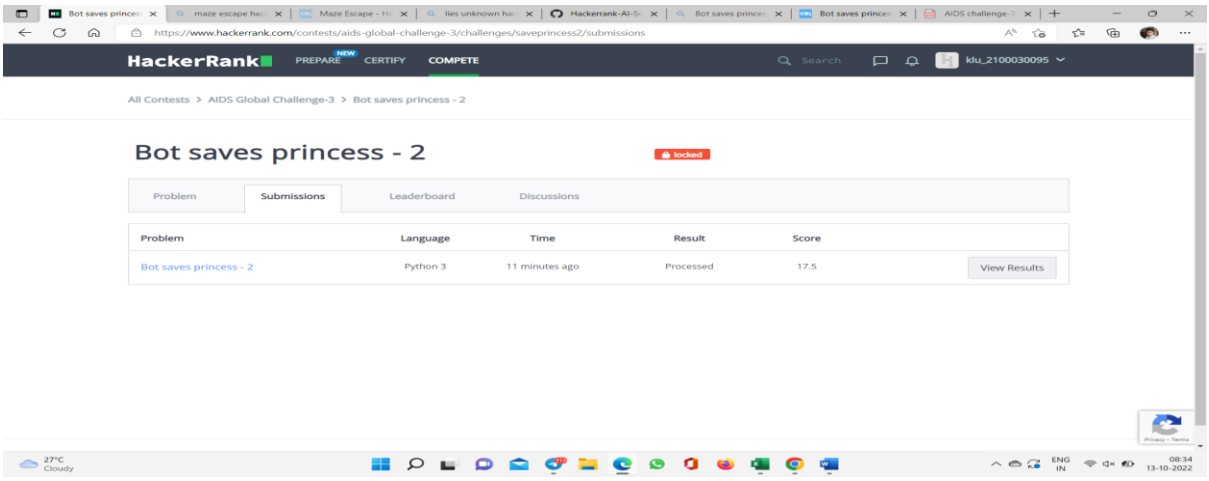
3.Bot saves princess-2

Code:

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

```
print(nextMove(n,r,c,grid))
```

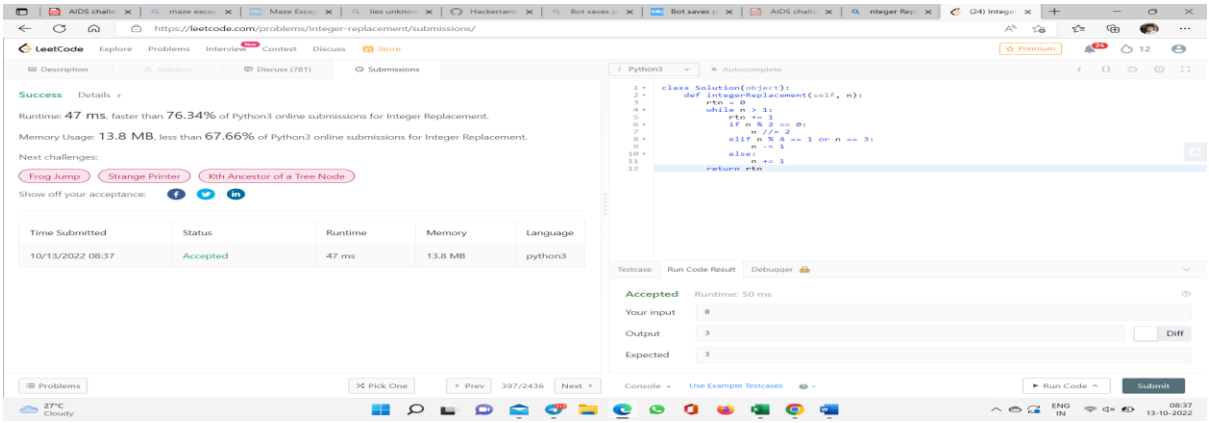


LeetCode:

1.Integer replacement:

Code:

```
class Solution(object):
    def integerReplacement(self, n):
        rtn = 0
        while n > 1:
            rtn += 1
            if n % 2 == 0:
                n //= 2
            elif n % 4 == 1 or n == 3:
                n -= 1
            else:
                n += 1
        return rtn
```



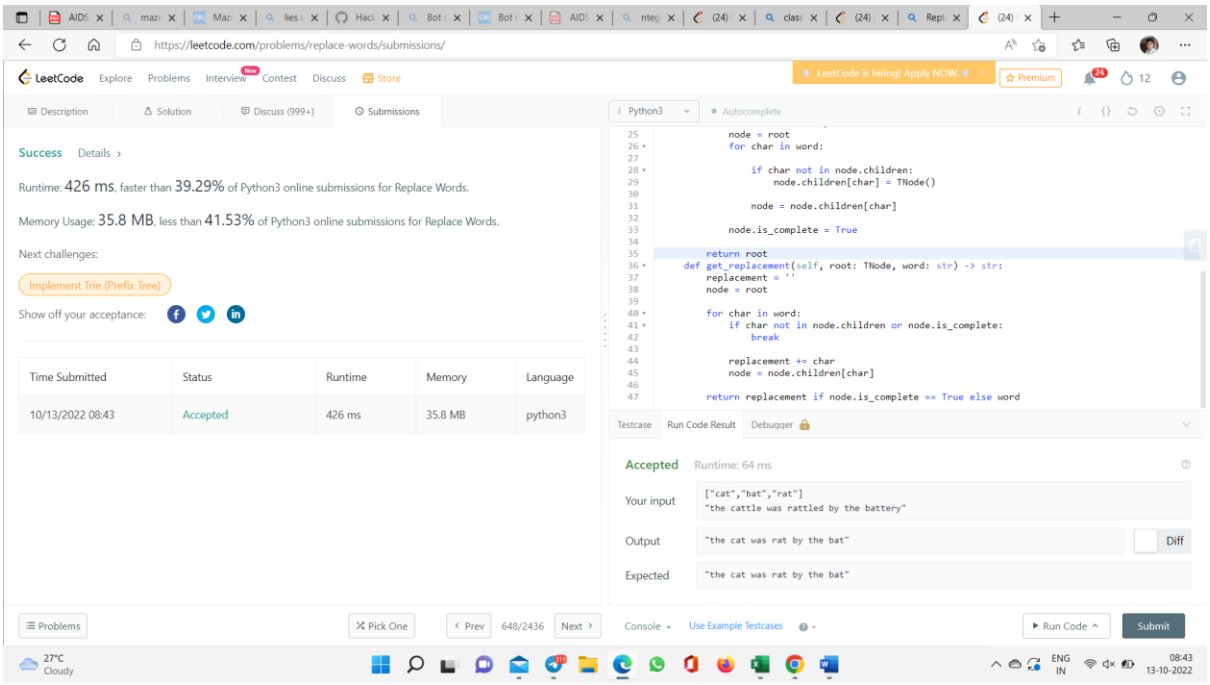
2.Replace Word:

Code:

```
class TNode:
    def __init__(self):
        self.children = {}
        self.is_complete = False


class Solution:
    def replaceWords(self, dictionary: List[str], sentence: str) -> str:


        tire_root = self.dict_to_trie(dictionary)


        words = sentence.split(' ')
        results = []


        for word in words:
            results.append(self.get_replacement(tire_root, word))
```

```
    return ' '.join(results)



def dict_to_trie(self, dictionary: List[str]) -> TNode:
    root = TNode()

    for word in dictionary:
        node = root
        for char in word:

            if char not in node.children:
                node.children[char] = TNode()

            node = node.children[char]

        node.is_complete = True

    return root
def get_replacement(self, root: TNode, word: str) -> str:
    replacement = ''
    node = root

    for char in word:
        if char not in node.children or node.is_complete:
            break
```

```
            replacement += char
            node = node.children[char]

        return replacement if node.is_complete == True else word
```



4.Minimum lines to represent a line chart

Code:

```
class Solution:
    def minimumLines(self, stockPrices) -> int:
        stockPrices.sort()
        ans = len(stockPrices) - 1
        for i in range(2, len(stockPrices)):
            a = (stockPrices[i][1] - stockPrices[i-1][1])
            b = (stockPrices[i][0] - stockPrices[i-1][0])
            c = (stockPrices[i-1][1] - stockPrices[i-2][1])
            d = (stockPrices[i-1][0] - stockPrices[i-2][0])
            ab = gcd(a, b)
```

```
        cd = gcd(c, d)
        a, b, c, d = a / ab, b / ab, c / cd, d / cd
        if a == c and b == d: ans -= 1
    return ans
```