CS3201

Algorithm Design

Project Report

Problem: 1377. Lara Croft

Submitted To

Dr. Thitipong Tanprasert

Submitted By

5713074 Panusorn Srijamorn

5713356 Thanabodee Bunnuch

I. INTRODUCTION

Problem: 1377. Lara Croft

Time Limit: 1.0 second

Memory Limit: 64 MB

Difficulty: 295

Description:

A cemetery has a form of rectangle. There are N rows of graves, M ones in each row. The cemetery is enclosed with a high and deep fence.

Lara Petrovna Croft has penetrated into the cemetery through the sap at the Northwestern corner. It takes one night for Lara to dig a subway under one of the graves. If there is an intact grave straight ahead then Lara will lengthen the passage during the next night and will ravage the grave. If there is a cemetery fence or a ravaged grave on the way, then Lara will turn 90 degrees clockwise and will continue with her questionable affairs.

Treasures are located in two graves only. And we exactly know in which ones. But Lara doesn't. Lara has bought a package of champagne today. It means, that today she has found one of those graves. We wonder how long will it take her to find the other one?

Input:

The first line contains two numbers N and M ($2 \leq N,M \leq 100$) – the sizes of the cemetery. The second and the third lines contain the treasure graves coordinates. Assume that the North-Western grave has coordinates (1,1) and the South-Eastern – (N,M). Lara starts with the grave (1,1) moving to the East, i.e. towards the grave (1,2).

Output:

Output an amount of days that Lara will spend reaching for another grave with treasures.

Input	Output
5 4 2 2 5 3	6

Problem Author: Stanislav Vasilyev

Problem Source: IX Urals Programming Contest. Yekaterinburg, April 19-24, 2005

II. CODE OVERVIEW

LaraCroft1.py

```
import time

r = raw_input().split()
N = int(r[0])
M = int(r[1])
r = raw_input().split()
T1 = (int(r[0])-1, int(r[1])-1)
r = raw_input().split()
T2 = (int(r[0])-1, int(r[1])-1)
map = [[0 for x in range(M)] for x in range(N)]
Treasure = [[T1[0], T1[1]], [T2[0], T2[1]]]
ETreasure = []


def valid(r, c):
    return 0 <= r <= N-1 and 0 <= c <= M-1 and map[r][c] != 1


def isgoal(r, c):
    for i in range(len(Treasure)):
        if r == Treasure[i][0] and c == Treasure[i][1]:
            ETreasure.append(Treasure[i])
            del Treasure[i]
            return True
    return False


def dig(start):
    counter = 0
    r = start[0]
    c = start[1]
    direction = 'e'
    while not isgoal(r, c):
        map[r][c] = 1
        if direction == 'e' and valid(r, c + 1):
            c += 1
            counter += 1
        elif direction == 'e' and not valid(r, c + 1):
            direction = 's'
        if direction == 's' and valid(r+1, c):
            r += 1
            counter += 1
        elif direction == 's' and not valid(r+1, c):
            direction = 'w'

        if direction == 'w' and valid(r, c-1):
            c -= 1
            counter += 1
        elif direction == 'w' and not valid(r, c-1):
            direction = 'n'
        if direction == 'n' and valid(r-1, c):
            r -= 1
            counter += 1
        elif direction == 'n' and not valid(r-1, c):
            direction = 'e'

    return counter
st = time.clock()
dig((0, 0))
dig(ETreasure[0])
et = time.clock()

print et - st
```

II. CODE OVERVIEW

LaraCroft2.py

```
import time

r = raw_input().split()
N = int(r[0])
M = int(r[1])
r = raw_input().split()
Start = (int(r[0]), int(r[1]))
r = raw_input().split()
End = (int(r[0]), int(r[1]))
Explored = []
Treasure = [[Start[0], Start[1]], [End[0], End[1]]]
ETreasure = []


def valid(r, c):
    if 1 <= r <= N and 1 <= c <= M:
        for i in range(len(Explored)):
            if Explored[i][0] == r and Explored[i][1] == c:
                return False
        return True


def isgoal(r, c):
    for i in range(len(Treasure)):
        if r == Treasure[i][0] and c == Treasure[i][1]:
            ETreasure.append(Treasure[i])
            del Treasure[i]
            return True
    return False


def dig(start):
    counter = 0
    r = start[0]
    c = start[1]
    direction = 'e'
    while not isgoal(r, c):
        Explored.append([r, c])
        if direction == 'e' and valid(r, c + 1):
            c += 1
            counter += 1
        elif direction == 'e' and not valid(r, c + 1):
            direction = 's'
        if direction == 's' and valid(r + 1, c):

            r += 1
            counter += 1
        elif direction == 's' and not valid(r + 1, c):
            direction = 'w'
        if direction == 'w' and valid(r, c - 1):
            c -= 1
            counter += 1
        elif direction == 'w' and not valid(r, c - 1):
            direction = 'n'
        if direction == 'n' and valid(r - 1, c):
            r -= 1
            counter += 1
        elif direction == 'n' and not valid(r - 1, c):
            direction = 'e'
    return counter


st = time.clock()
dig((1, 1))
dig(ETreasure[0])
et = time.clock()

print et - st
```

III. CODE DEMONSTRATION

```
def valid(r, c):
    return 0 <= r <= N-1 and 0 <= c <= M-1 and map[r][c] != 1
```

valid(r, c) for LaraCroft1.py

Valid is used to check whether it is valid if we move to (r, c) by checking if (r, c) is not ravaged by simply check in 2d matrix and if (r, c) is still inside the cemetery.

```
def valid(r, c):
    if 1 <= r <= N and 1 <= c <= M:
        for i in range(len(Explored)):
            if Explored[i][0] == r and Explored[i][1] == c:
                return False
        return True
```

valid(r, c) for LaraCroft2.py

For another algorithm, we use another valid function the functional of this function stay the same but the way of determine whether (r, c) is ravaged or not will be different.

Instead we make use of 2d matrix we iterate through the list of explored point (list of tuple).

III. CODE DEMONSTRATION

```
def dig(start):
    counter = 0
    r = start[0]
    c = start[1]
    direction = 'e'
    while not isgoal(r, c):
        map[r][c] = 1
        if direction == 'e' and valid(r, c + 1):
            c += 1
            counter += 1
        elif direction == 'e' and not valid(r, c + 1):
            direction = 's'
        if direction == 's' and valid(r+1, c):
            r += 1
            counter += 1
        elif direction == 's' and not valid(r+1, c):
            direction = 'w'
        if direction == 'w' and valid(r, c-1):
            c -= 1
            counter += 1
        elif direction == 'w' and not valid(r, c-1):
            direction = 'n'
        if direction == 'n' and valid(r-1, c):
            r -= 1
            counter += 1
        elif direction == 'n' and not valid(r-1, c):
            direction = 'e'

    return counter
```

dig(r, c) for LaraCroft1.py & LaraCroft2.py

Dig has the starting point as a parameter(tuple) and return the number of night Lara have to dig until she found the next treasure. By giving the initial direction for Lara to start digging and then mark that (r, c) as ravaged and check if it is still valid to keep going on this direction if yes keep going, if not change the direction (90 degree clock wise) then continue digging.

III. CODE DEMONSTRATION

```
def isgoal(r, c):
    for i in range(len(Treasure)):
        if r == Treasure[i][0] and c == Treasure[i][1]:
            ETreasure.append(Treasure[i])
            del Treasure[i]
            return True
    return False
```

isgoal(r, c) for LaraCroft1.py & LaraCroft2.py

IsGoal is used to check whether the (r, c) is the treasure or not. If yes the function will append that (r, c) into the new list to be used further as new starting point for another treasure digging

IV. RESULT TABLE

Using LaraCroft1.py

Test case 1 (5.24915602955e-05)

Input	Output
5 5 1 1 3 3	24

Test case 2 (0.000250434594929)

Input	Output
10 10 1 1 5 5	96

Test case 3 (0.000696466232971)

Input	Output
20 20 1 1 10 10	396

Test case 4 (0.0116431559216)

Input	Output
100 100 1 1 50 50	9996

Test case 5 (1.01349267699)

Input	Output
1000 1000 1 1 500 500	999996

IV. RESULT TABLE

Using LaraCroft2.py

Test case 1 (0.00014779746586)

Input	Output
5 5 1 1 3 3	24

Test case 2 (0.000691187752047)

Input	Output
10 10 1 1 5 5	96

Test case 3 (0.00686847667733)

Input	Output
20 20 1 1 10 10	396

Test case 4 (3.6317511771)

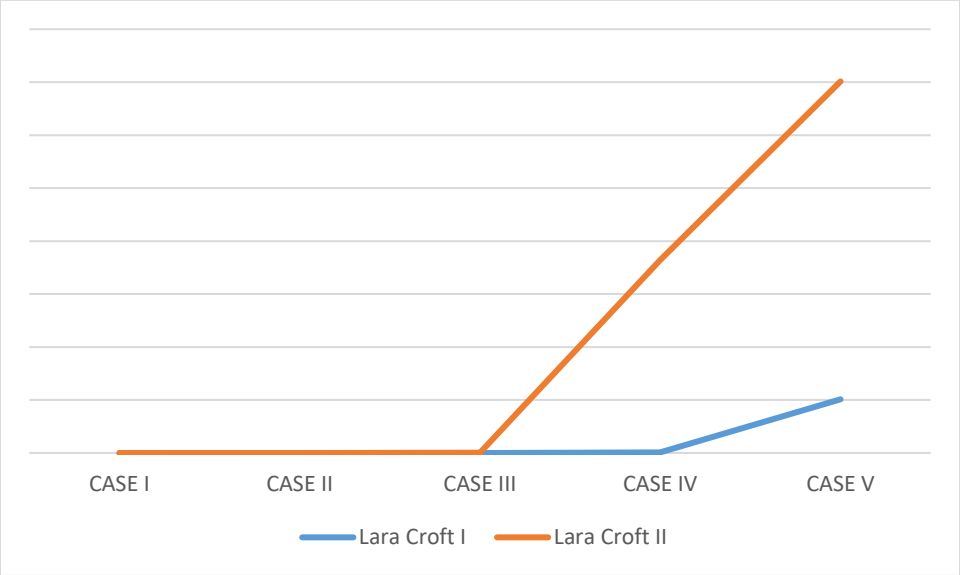
Input	Output
100 100 1 1 50 50	9996

Test case 5 (Take too long)

Input	Output
1000 1000 1 1 500 500	-

IV. RUNTIME ANALYSIS

	Test case 1	Test case 2	Test case 3	Test case 4	Test case 5
LaraCroft1.py	5.25E-05	0.000250435	0.000696466	0.011643156	1.013492677
LaraCroft2.py	0.000147797	0.000691188	0.006868477	3.631751177	-



V. SUMMARY

LaraCroft1.py: O(MN)

LaraCroft2.py: O(MN^2)

Author	Problem	Language	Judgement result	Test #	Execution time	Memory used
Thanabodee	1377. Lara Croft	Python 2.7	Accepted		0.046	376 KB

http://acm.timus.ru/problem.aspx?num=1377

http://acm.timus.ru/status.aspx?space=1&num=1377&status=accepted