Mastering Data Structures and Algorithms
Take-Home Assignment 1

Yiying Peng (Christine)

February 6, 2019

Problem

Assume you are given a function bool hit(int x, int y) that returns true if there was a hit to a ship and false if not. You are to write a function that returns the coordinates of a single ship on a grid of given size, the function takes at least one argument int gridSize and must return the coordinates of the 3 unit ship present in the board in the shortest time possible. Estimate the Time and memory complexity of your proposed solution. You are to select the data structure to return what you need as output and the prototype of the function.

Sol. My algorithm implementation is as follows.

```
SHIP_SIZE = 3
GRID_SIZE = 5
grid = []


'''THIS IS A HELPER FUNCTION'''
def load_grid():
    global grid
    with open('hw2_input.txt') as f:
        lines = f.readlines()
        assert len(lines) == GRID_SIZE
        grid = [line.strip() for line in lines]


'''THIS IS A HIDDEN FUNCTION'''
def hit(r, c):
    assert r >= 0 and r < GRID_SIZE and c >= 0 and c < GRID_SIZE
    return grid[r][c] == '*'


def check_row(r, c):
    res = []
    for nc in range(max(c-2, 0), min(c+2, GRID_SIZE-1)+1):
        if hit(r, nc):
            res.append((r, nc))
    if len(res) == 3:
        return res
    else:
        assert len(res) == 1
        return []


def check_col(r, c):
```

```
    res = []
    for nr in range(max(r-2, 0), min(r+2, GRID_SIZE-1)+1):
        if hit(nr, c):
            res.append((nr, c))
    if len(res) == 3:
        return res
    else:
        assert len(res) == 1
        return []


def find_ship():
    # horizontal ship
    for r in range(GRID_SIZE):
        for c in range(2, GRID_SIZE, 3): # [ooxooxo]
            if hit(r, c):
                res = check_row(r, c)
                if res:
                    return res
    # vertical ship
    for r in range(2, GRID_SIZE, 3): # [ooxooxo]^T
        for c in range(GRID_SIZE):
            if hit(r, c):
                res = check_col(r, c)
                if res:
                    return res


if __name__ == '__main__':
    load_grid()
    res = find_ship()
    print(res)
```

In order to verify the correctness of my algorithm, I implemented two additional functions:

- Helper function **load_grid()** reads a txt file with $n \times n$ grid, which is filled with either '_' as sea or '*' as part of the boat.
- Hidden function **hit(r, c)** takes row r and column c as arguments and determine whether (r, c) is part of the boat or not.

In terms of time complexity, it can be decompose as two parts: **find_ship()** and **check_row(r, c)** (or **check_col(r, c)**). For the first part, I have two grid scans with different directions, each of which needs $O(n \times (n/3))$, or said overall $O(\frac{2}{3}n^2)$. For the second part, it takes at most $O(5)$ time call function **hit(nr, c)**, where $nr \in [\max(c-2, 0), \min(c+2, \text{GRID\_SIZE})]$ (or **hit(r, nc)**, where $nc \in [\max(r-2, 0), \min(r+2, \text{GRID\_SIZE})]$). Jointly consider two parts, the time complexity is $O(\frac{2}{3}n^2) + O(5) = O(n^2)$.

In terms of space complexity, there is only at most three elements being added into **res**, which results in $O(3) = O(1)$ extra space requirement.

□