COMP20007 DOA 2021 S1 Assignment 1

Lang (Ron) Chen 1181506

README

This file contains my responses to all of problem 1, problem 2b and the report for problem 3 for this assignment.

Problem 1

Part A

Assuming that "Sam can only see the gap if it is right next to it" means Sam is only able to see the fence (whether its closed or open) that it is currently at:

The worst-case complexity of the algorithm in part A is O(l). This is because the worst case is that the gap is one fence left of Sam's current fence, but he chose to move right (or vice versa in terms of directions). With the basic operation being the number of steps, he makes (can also use "making a check" as basic operation but since they are the same for this part of the question but subsequent parts of this question are more easily solved using "steps" as the basic operation, I have chosen to use "steps" here), he will need to make l basic operations for a fence of length l. Hence, the worst-case time complexity of this algorithm is O(l).

Part B

A possible algorithm is this:

Sam will choose a random direction to start walking in. If he walks k steps yet fails to see the fence at that spot (wrong direction), he can walk back 2k steps (first k steps back to the original and then k steps in the other direction) and be sure to find the exit.

The worst-case scenario is if Sam walks in the opposite direction to which it can make k steps to the fence. Hence, in the worst-case Sam needs to make 3k steps, which is asymptotically equivalent to O(k).

THIS GAP WAS DELIBERATELY LEFT HERE SO PART C PSEUDOCODE FITS ON ONE PAGE

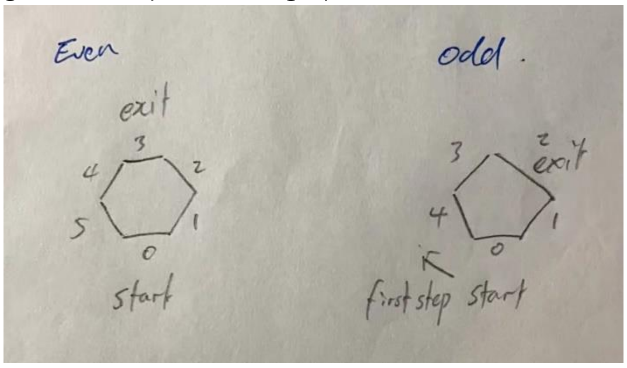
Part C

1. Psueudcode

```
Function SamAlgorithmPartC(Fence with an exit, Sam's starting position)
//Helps Sam finds the exit
//Input: A data structure (e.g. circular array) representing Fence with exit and an index (e.g an integer) representing Sam's starting position
//Output: Sam just exits (or returns True)
i ← 1
direction ← random pick of left and right
check whether at exit                if True do exit
while True do
        for a ← 1 to i inclusive do
                move Sam 1 step in direction
                check whether at exit if a ≥ i - 1          if True do exit
        move Sam i steps in opposite direction of direction
          ← opposite direction of direction
        i ← i + 1
```

2. Worst case time complexity

There are two cases of worst case for this algorithm (let the fence that Sam starts at be 0): 1. l is even and k is at (l/2); 2. l is odd and k = ⌊l/2⌋ counting clockwise (or to the right), but Sam starts off anticlockwise (to the left). Vice versa applies.



For l is even: number of steps (base operation) required in worst case

$$= \left(\sum_{i=1}^{\frac{l}{2}} 2i\right) - \frac{I}{2} = \frac{l^2}{4}$$ which is asymptotically equivalent to O(l^2).

The sum is defined this way because the total steps of Sam's expeditions are summed as 2(1) + 2(2) + … + 2(l/2), ending at l/2 because in the algorithm Sam traverses both directions and thus is bound to have found the exit by l/2. The subtraction at the end is because when l is even, Sam will find the exit at l/2 and does not need to return to the origin in his last expedition (one expedition defined as moving x steps from origin and then back to the origin), and hence one round of l/2 can be subtracted from the sum.

For L is odd: number of steps required in worst case $= (\sum_{i=1}^{\lfloor \frac{l}{2} \rfloor} 2i) + \lfloor \frac{l}{2} \rfloor = \frac{l^2}{4} + l$ which is also asymptotically equivalent to O(l^2).

The reason for the sum is the same as above, while the addition of ⌊l/2⌋ is because l is odd:

after the expedition of ⌊l/2⌋ steps (defined above as having returned to the origin), Sam will need to make a further one-way trip of ⌊l/2⌋ in the opposite direction (the one way expedition length should be ⌊l/2⌋+ 1, but he will find the exit at ⌊l/2⌋), so one bout of ⌊l/2⌋ is added to the sum.

Hence in the worst case this algorithm is asymptotic to O(l^2).

3. Explanation/Comparison to Part A

The best case for both algorithms is O(1) (e.g. "Sam's original position was the exit" or "k = 1 and Sam started in the correct direction", the list is not exhaustive), however the worst case for the algorithm in part C is O(l^2) while in part A is Θ(l). Hence the algorithm for part A is Ω(1) and O(l^2) while part C is Ω(1) and O(l^2). Thus, it is worse than the strategy in part A.

Part D

The algorithm to achieve this aim is for Sam to first take 1 step in one direction, and if fails to find the exit returns home and walks 2 steps in the other direction and if still fail return to origin. However, the third and fourth time Sam will walk 4 and 8 steps respectively. (i.e. the steps Sam takes are 2^0, 2^1, 2^2, 2^3 etc). This yields O(k) time because:

Let k = 2^m for some nonnegative integer m and the basic operation being the number of steps Sam takes.

Hence the sum for the steps he will take is

$$2(1) + 2(2) + 2(4) + \dots + 2(k) + k$$

$$= 2(2^0) + 2(2^1) + 2(2^2) + \dots + 2(2^{\log_2 k}) + (2^{\log_2 k})$$

$$= 2(\textstyle\sum_{i=0}^{\log_2 k} 2^i) + (2^{\log_2 k})$$

$$= 2(\sum_{i=0}^{\log_2 k} 2^i) + k$$

$$= 2\left(\frac{2^{\log_2(k)+1} - 1}{2-1}\right) + k \qquad \textit{as per equation 5 in Levitin's Appendix A}$$

$$= 2(2k-1) + k$$

$$= 5k - 2$$

For cases where k does not equal 2^m, simply round k up to the next 2^m – the only difference would be that on the final trip the exit would be found before k steps are taken, so a constant could be subtracted off the sum, but it makes no difference asymptotically.

The reason for the addition after the sum is if Sam starts off in the "wrong" direction (worst possible case), he needs to do the full expedition of k length in the wrong direction and then add an extra trip of k (not 2k because he is certain to find the exit at k) to the total steps. As 5k-2 is asymptotically equivalent to O(k) time, using this algorithm Sam will certainly be able to leave the paddock within O(k) time efficiency.

<u>Problem 2</u>

Part B

To solve my problem, I calculated the cost of the installing antenna for each house and the cost for connecting cables so that each house is ultimately connected to the data centre, and then compared them to return the lower cost option.

Calculating the cost for installing antennas was simple – the number of houses was multiplied to the cost of installing an antenna.

Calculating the lowest cost to connect all houses was slightly more complex as it involved Prim's algorithm, and summing the costs of all edges that were picked by the algorithm to form the minimal weight graph. This algorithm works for solving this question because the question implicitly implies it wants to find the lowest cost for using cables for all houses: only by finding the lowest/most competitive option for connecting all houses can the internet providers make a meaningful comparison as to whether all-cables or all-radios are the cheaper choice (their goal is to find the cheapest option overall). The lowest cost connection of all houses using cables coincides with the minimal weight graph for where houses are vertices of the graph and potential connections (with costs) are considered as the weighted edges, so Prim's algorithm (and summing the edge costs for the outcome) is the precise solution for calculating the cost of connecting all houses with cables.

Hence, this is why my program worked.

<u>Problem 3</u>

Part C

The statistics for the two chips working on the two algorithms are as follows (as according to the rules for calculating operations specified in the question):

Old chip (Euclid):

Minimum operations: 22

Average operations: 48.843400

Maximum operations: 103

New chip (Euclid)

Minimum operations: 22

Average operations: 48.843400

Maximum operations: 103

Old chip (Sieve)

Minimum operations: 6

Average operations: 818.965800

Maximum operations: 2764

New chip (Sieve)

Minimum operations: 6

Average operations: 244.945500

Maximum operations: 717

From the statistics, it can be concluded that while the New chip allows for a better performance for the Sieve algorithm, bringing the average and worst case down significantly, but it is still outperformed by the Euclid algorithm (either chip – which doesn't make a difference) in terms of average and worst case; in fact, the worst case of Euclid outperforms the average case of Sieve on both chips. However, the best case for Sieve for both new and old chip is better than the best case of Euclid. Overall, it can be said that the new chip is a milestone improvement, but its capabilities are not demonstrated well by the comparison of these two algorithms