Assignment 1

May 12, 2021

1 Assignment 1

Due: 5:00pm AEST Friday 16th of April

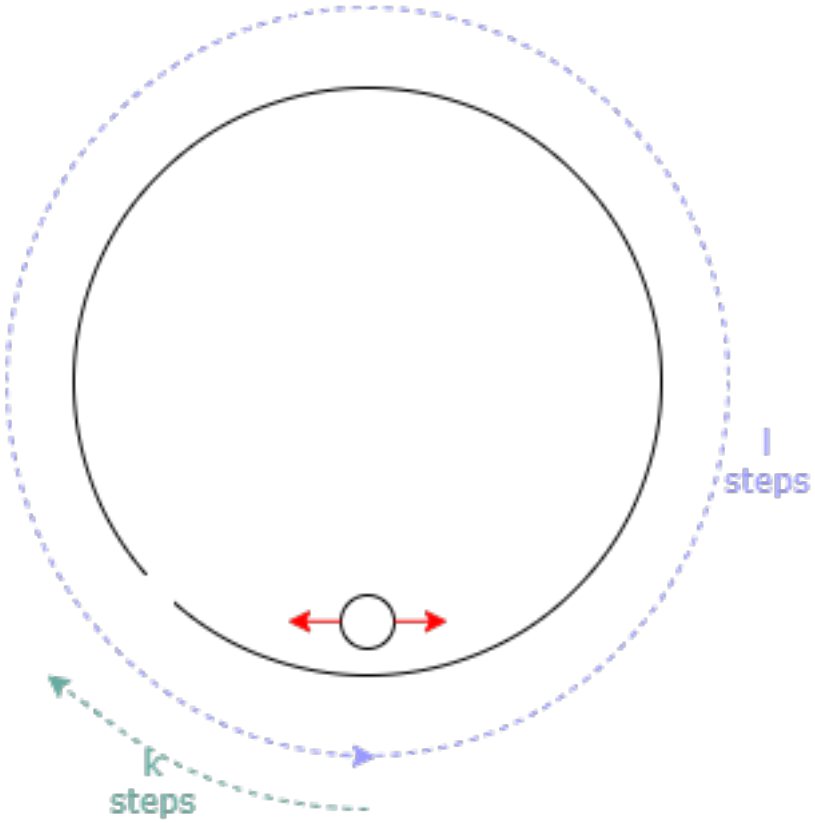
1.0 Introduction

This assignment contains 3 problems. You will be required to write pseudocode and C code, as well as provide a detailed analysis of your algorithms. You will submit your solutions for the C programming component of this assignment via `dimefox submit` and the written component via the LMS.

This assignment has a total of 10 marks and will contribute 10% to your final grade for this subject.

1.1 Problem 1

A shortsighted cow, named Sam, cannot find enough grass on its current pasture. It remembers that the pasture's enclosing fence has a gap. Unfortunately, the fence is very long: for a full circle, it takes Sam l steps to walk along the fence. Sam can only see the gap if it is right next to it (remember the cow is shortsighted). In this question, you will design different algorithms that will enable Sam to find the gap that is k steps away from its current position. Sam is always located next to the fence. We call its start position *origin*. You may assume that l is much greater than k.



1.1.1 Part A

We assume that the cow can go only in one direction along the fence. It has to pick one of the two directions (say left or right) and continues until it has found the gap. Derive the worst case complexity of this algorithm.

Part A Answer The worst case occurs in two cases, the case where the gap is immediately on the right side and the cow picks left as its initial direction, and the symmetric case. In both cases the time complexity is O(l). In both cases the gap lies O(1) ddistance away from the cow, so doesn't violate k << l. A more flexible specification of this, leaving k free, is O(l - k) which is still O(l) as k << l.

1.1.2 Part B

Sam's best friend, an eagle called Indigo, can see the gap and knows the number of steps Sam has to take, i.e., the value of k is known in this question. Unfortunately, Indigo is never sure whether it saw the gap with its left or right eye. Hence, Indigo can tell Sam only the number of steps Sam has to take but does not know the right direction. Design an algorithm with $O(k)$ worst case complexity such that Sam can find the gap. Explain why your has $O(k)$ time complexity.

Part B Answer Sam picks a random direction.

1. Case 1: Sam picks the right direction, moves k steps and finds the gap.
2. Case 2: Sam picks the wrong direction, moves k steps, finds there is no gap, turns around and moves $2k$ steps.

Since $\max\{k, 3k\} \leq 3k$, the worst case complexity is O(k).

1.1.3 Part C

In this part, we assume that k is not known. Sam applies the following strategy: walk 1 step to the right, turn around if no gap is found, and move to the start. Then walk 2 steps to the left and return to the start if no gap is found. Then walk 3 steps to the right and return to the start, and so forth.

1. First, work out the general algorithm and write it in pseudocode.
2. Second, work out the worst case time complexity of this algorithm in detail.
3. Explain whether or not this strategy is better than the one you analyzed in Part A.

Part C Answer An answer to Part C's pseudocode is:

```
direction ← RIGHT
choice ← NONE
distance ← 1
while choice = NONE:
        walk(direction, distance)
        if seesGap:
                choice ← direction
        else:
                if direction = RIGHT:
                        walk(LEFT, distance)
                        direction ← LEFT
                else:
                        walk(RIGHT, distance)
                        distance ← distance + 1
                        direction ← RIGHT
```

The computational complexity is $O(k^2)$. The complexity is computed as: $k + \Sigma_{i=1}^{k}(2 \times i) = k + k \times (k+1) = k \times (k+2) \in O(k^2)$. The additional k if we went k into the wrong direction, and still have to go k in the correct direction.

We cannot make the claim that the strategy is better than the strategy analyzed in Part A. If we add the information that $k^2 << l$, we can, but otherwise there is insufficient information to claim that either the strategy in Part A or the strategy in Part C is better.

1.1.4 Part D

Design an algorithm that requires $O(k)$ time efficiency to find a gap and show that its efficiency is indeed $O(k)$. You do not need to write the algorithm in pseudocode (you can if you want) but you have to describe the algorithm clearly.

Part D Answer An answer to Part D's pseudocode is:

```
direction ← RIGHT
choice ← NONE
distance ← 1
while choice = NONE:
        walk(direction, distance)
        if seesGap:
                choice ← direction
        else:
                if direction = RIGHT:
                        walk(LEFT, distance)
                        distance ← distance * 2
                        direction ← LEFT
                else:
                        walk(RIGHT, distance)
                        distance ← distance * 2
                        direction ← RIGHT
```

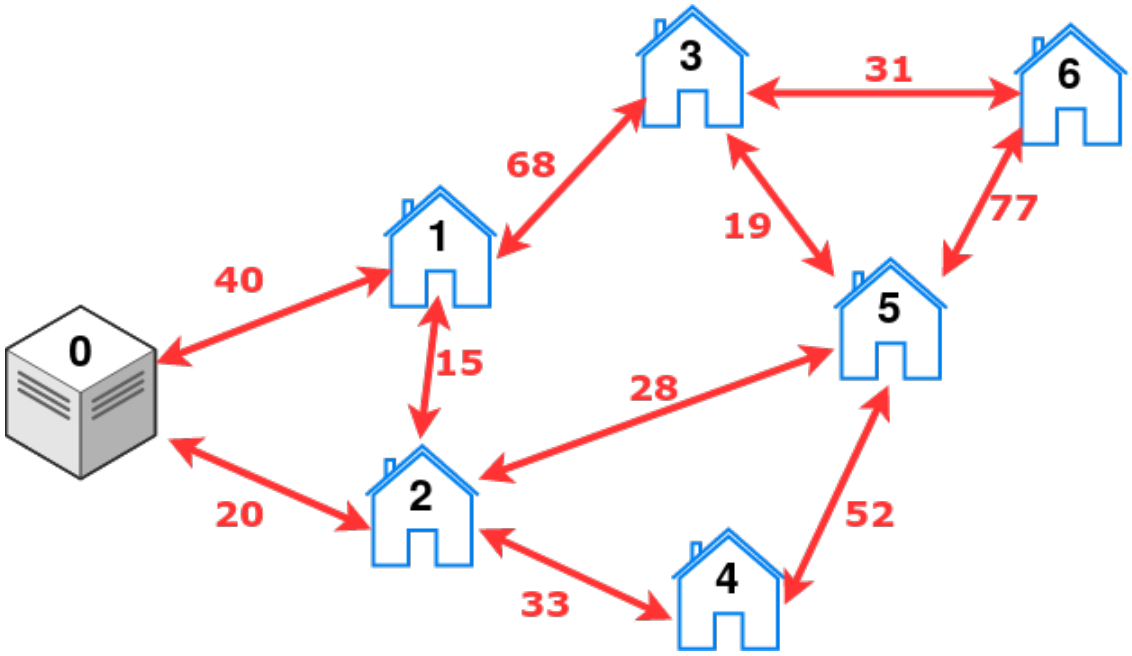
We see that the algorithm will terminate when distance surpasses k, or on the loop which follows (in the case that the direction chosen was wrong). Because each step doubles the distance, we will surpass k by loop m + 1, when $k \leq 2^m \leq 2k$. Furthermore of note is that the final iteration of the loop will only require exactly k steps because the gap will be seen before reaching the planned distance. The sum of steps taken is hence given by $k + \sum_{i=0}^{m} 2^{i+1}$, which includes a geometric series with sum $2^{m+2} - 2$. As earlier noted, $k \leq 2^m \leq 2k$, hence $5k - 2 \leq 2^{m+2} + k \leq 9k - 2$. Because the upper and lower bounds for the total number of steps in this sum are both $\Theta(k)$, the total number of steps is as well.

1.2 Problem 2

An Internet service provider wants to deliver service to a village. They have two installation options:

- Cabled installation. This means connecting the data center via underground cables. Connections can be either made from the data center to a house or between houses. The cost is equal to the distance between houses (or between a data center and a house).
- Radio-based installation. This means installing antennas in each house, which then receive the internet signal through a satellite. The cost of an antenna installation is fixed per house but can vary from village to village.

The figure below shows an example of a village, where distances are drawn as red arrows with their corresponding value.



For this village, the corresponding textual format is shown below:

```
25
6 10
0 1 40
0 2 20
1 2 15
1 3 68
2 4 33
2 5 28
3 5 19
3 6 31
4 5 52
5 6 77
```

where:

- First line contains the **antenna cost**.
- Second line shows the **number of houses** followed by the **number of connections** in the village.
- Remaining lines show connections where the first and second number are the houses (or `0` if it is the data center) and the third number is the actual distance.

1.2.1 Part A

Devise a program in C that given a village map and antenna installation cost in the format described above, returns the installation option with lower cost in total. Your program should output the lowercase character `c` if cabled has the lower installation cost and `r` if radio-based has the lower cost instead.

Part A Answer See provided sample solution code.

1.2.2 Part B

Explain why your program works. You can use pseudocode to help explain your solution.

Part B Answer The cost of servicing each house using antennas is simply the number of houses multiplied by the cost of each antenna.

For the cabled cost, we can model the houses and data centre as vertices in a graph and identified distances as edges in the graph. The subset of cables which spans all vertices in this graph forms a spanning tree. Any spanning tree (or graph) connects all vertices transitively, so all houses will be connected to the data centre. The smallest set of distances (and hence length of cable) which spans the graph is the minimum spanning tree. We can use Prim's algorithm for this.

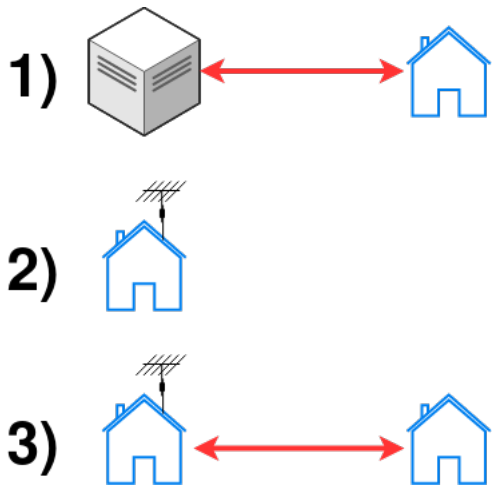
In part C, an alternative algorithm (Kruskal's algorithm) has been chosen and implemented - this algorithm also finds a minimum spanning tree, but instead of exploring by selecting the closest node, we select the smallest edge and add it to the MST if the edge doesn't connect two already connected components.

1.2.3 Part C

Same as Part A, but now the company updated their infrastructure and it can now provide a mix of cabled and radio-based installations for the village. This means that a house will have internet access if any of the following are true:

1. There is a path to the data center,
2. The house has an antenna or
3. There is a path to a house with an antenna.

The figure below shows an example of each of the cases above. All houses below are considered connected.



Modify your program in order to output the **minimum total cost** for this mixed installation.

Part C Answer The solution for part C is simple, we simply add an antenna cost edge back to the data centre before running Kruskal's algorithm as usual. This is the solution we used. Alternatives (such as removing the edges with greater cost than the antenna cost, running Prim's algorithm on each connected component, keeping track of all vertices already spanned by the Minimum Spanning Forest) are also valid.

1.3 Problem 3

A processor manufacturer has come up with a design for a new processor technology designed for cryptography which is able to simultaneously check if any elements in an array of integers are divisible by a particular number. In marketing this chip, the processor engineering team have asked you to write a program to compare how the chip performs across two different benchmarks which simplify a fraction.

The first of these benchmarks uses Euclid's algorithm to find the greatest common factor between the two numbers, the second works by dividing all common prime factors appearing in both numbers until no more common factors remain. It finds these primes using the Sieve of Eratosthenes.

Both programs are given below in pseudocode.

First algorithm:

```
euclid(numerator, denominator)
    b ← numerator
    a ← denominator
    while b ≠ 0
        t ← b
        b ← a mod b
        a ← t

    print (numerator / a) "/" (denominator / a)
```

Second algorithm:

```
eratosthenes(numerator, denominator)
    num ← numerator
    den ← denominator

    numCandidates ← min(num, den)

    primes ← array with index 1..numCandidates, filled with 1s

    i ← 1
    while i < numCandidates
        i ← i + 1
        if primes[i]
            primes[j: j / i > 1, j mod i = 0] = 0
            while num mod i == 0 and den mod i == 0
                num ← num / i
                den ← den / i

    print (num) "/" (den)
```

The two chips under comparison have the following properties:

1. Assignments cost 1 operation.
2. Divisions cost 5 operations.
3. Each modulo operation costs 5 operations.
4. An if branch costs 1 operation.
5. Every while branch check costs 1 operation.
6. Accessing an element of an array or a particular variable otherwise costs 0 operations.
7. All other operations are also 0 cost operations.

The new chip has the following difference: 1. The operation in the 13th line of the second algorithm, primes[j: j / i > 1, j mod i = 0] = 0, costs 1 operation.

While on the old chip this must be implemented using a while loop. When calculating the operations required for this, assume both conditions are always evaluated. So this will take 5 + 5 + 1 operations. Assume for a prime i that this is performed by jumping i steps forward.

To compare the performance, **implement both algorithms in C**, adding to the pseudocode the number of operations required on each chip. For each algorithm find the number of operations required on each chip for 10000 pairs of integers, varying the numerator and denominator from 1 to 100. Collect statistics on the minimum, average and maximum number of operations taken and print these out. Your program should print out *only* these summary statistics. Add the summaries of these tests to your report.

Problem 3 Answer Correct answers for this had observable properties present in the sample solution provided to do with the intuitive complexity differences in each algorithm.

1.4 Completing the Programming Problems

We have provided you with some skeleton code for this assignment, you may freely change any files but ensure output matches the form given in problem2a.c, problem2c.c and problem2d.c.

```
provided_files/
    Makefile
    problem2a.c
    problem2c.c
    problem3.c
    utils.c
    utils.h
    graph.c
    graph.h
    pq.c
    pq.h
    list.c
    list.h
    tests/
        p2a-in-1.txt
        p2a-out-1.txt
        p2a-in-2.txt
        p2a-out-2.txt
        ...
        p2c-in-3.txt
        p2c-out-3.txt
```

If you create new C modules (which you are encouraged to do) then you must change the Makefile so that your program can be compiled on dimefox using the make command.

To run the programs you must first compile with `make` followed by one of problem2a, problem2c or problem3. For problem2a and problem2c you can send the test case in via standard input redirection. For problem3 you can simply run the program.

For example:

```
make problem2a
make problem2c
make problem3
./problem2a < p2a-in-1.txt
./problem2c < p2c-in-1.txt
./problem3
```

The pXX-in-X.txt files contain the input your program will be provided for each test case, and the pXX-out-X.txt files contain the expected output. Your program must match the expected output exactly, so *must not print to stdout* otherwise.

1.5 Programming Problem Submission

You will submit your program via `dimefox submit`. Instructions for how to connect to the `dimefox` server can be found on the LMS. Further instructions on submitting your files will be provided soon.

It is recommended that you test your program on `dimefox` before submitting to make sure that your program compiles without warnings and runs as expected. If your program performs differently on the server to how it performs locally, you may find the tool valgrind useful.

Note that programs which do not implement the algorithm they claim to or do not run within the required time bound will receive fewer marks than the receipt may suggest.

Any attempt to manipulate the submission system and/or hard-code solutions to pass the specific test cases we have provided will result in a mark of 0 for the whole assignment.

1.6 Completing the Written Problems

You will submit your solutions to Problems 1a, 1b, 1c, 1d, 2b, and the summary of Problem 3 via the LMS.

For Problems 1 and 2, which ask for pseudocode, we expect you to provide the same level of detail as the lectures and workshops do. Pseudocode which lacks sufficient detail or is too detailed (e.g., looks like C code) will be subject to a mark deduction.

Your submission should be typed and not handwritten and submitted as a single .pdf file. Pseudocode should be formatted similarly to lectures/workshops, or presented in a monospace font. You should aim to keep each written problem part to less than one full page.

1.7 Mark Allocation

The total number of marks in this assignment is 10. The maximum marks for each problem are:

- **Problem 1** 4 marks (1 per part)
- **Problem 2** 3 marks (1 per part)
- **Problem 3** 2 marks

There is one additional mark awarded for "structure and style" for the C programming component. Of particular importance will be the structure of your C program in terms of modules (.c and .h files), and how you separate your types and functions across these files.

In total there are 5 marks for the written problems (Problem 1 and Problem 2 (b)) and 5 marks for the C programming problems (Problem 2 (a, c) and Problem 3).

We have provided **3 test cases** for each part of Problem 2 (a, c).

The marks awarded for each part of Problem 2 will be calculated by:

$$Marks = max\{1 - 0.5 \times TestCasesFailed, 0\}$$

So passing 0 or 1 of the test cases will award 0 marks, 2 test cases will award 0.5 marks and all 3 will award 1 mark (the maximum available).

1.8 Late Policy

A late penalty of 20% per day will be applied to submissions made after the deadline. The 20% applies to the *number of total marks*. This also applies *per component*, i.e.,

$$\begin{aligned} Grade = max\{ProgrammingGrade - 0.2 \times DaysLate \times 5, 0\} + \\ max\{WrittenSubmissionGrade - 0.2 \times DaysLate \times 5, 0\} \end{aligned}$$

For example if you are 2 days late with the programming component but only 1 day late with the analysis component, your grade for the programming component will be reduced by $0.4 \times 5 = 2$ and the grade for the analysis component will be reduced by $0.2 \times 5 = 1$.

1.9 Academic Honesty

You may make use of code provided as part of this subject's workshops or their solutions (with proper attribution), however you may not use code sourced from the Internet or elsewhere. Using code from the Internet is grounds for academic misconduct.

All work is to be done on an individual basis. All submissions will be subject to automated similarity detection. Where academic misconduct is detected, all parties involved will be referred to the School of Engineering for handling under the University Discipline procedures. Please see the Subject Guide and the "Academic Integrity" section of the LMS for more information.