**PLEASE REMEMBER TO NOT COPY EXACTLY AS THESE ARE HOW MY NOTES LOOK , DR JANSEN WILL READ THIS AND KNOW IF THEY ALL LOOK THE SAME**

Thursday 14th April

**Thursday 16th April.**

This was to be a very short lecture with only 8 slides to ensure we understand everything. We started by defining terms

**Computational Problem** - A computational problem is defined by a set of finite inputs over a finite input alphabet and for each input a set of correct finite outputs over a finite output alphabet. Example input directed graph with edge weights, nodes A and B output shortest path from A to B

**Optimisation Problem** - A optimisation problem is a computational problem where the output is the value of an optimal solution. Example input directed graph with edge weights, nodes A and B output length of shortest path from A to B Definition (Decision Problem) A decision problem is a computational problem where the output is ‘yes’ or ‘no’ (alternatively, ‘0’ or ‘1’ if we prefer binary encodings).

Example input directed graph with edge weights, nodes A and B, value k output yes if there is a path from A to B of length ≤ k, no otherwise

**Turing Machine**

A Turing machine has a finite set of possible states Q, an initial state q0 ∈ Q, a finite memory alphabet Γ that contains the blank character , a finite input alphabet Σ that does not contain , an infinite memory that is linearly organised, and a current position in the memory. Initially the input is in the memory, the current position is the first position of the input and all unused memory cells contain . Its functioning is defined by a program P : Q × Γ → Q × Γ × {L, R, ∗}. It operates in steps, in each step it is in some state q ∈ Q and reads the contents of the current cell a ∈ Γ. If P(q, a) = (r, b, d) then it replaces a by b, changes state from q to r and changes the current cell to its neighbour if d = L, to its right neighbour if d = R, leaving it unchanged if d = ∗.

We expect that for each input a correct output is in finite time. Again to be crystal clear We say ‘program P solves the computational problem Q’ if for any input that is an instance of Q P stops eventually and outputs a correct solution for that instance Definition (Computability) We call a computational problem computable if there exists a program that solves it.

Are there computational problems that are not computable? If so, are any of them practically important?

One way to solve this is using emulators- Important concept on the way to an answer emulation/simulation ‘computers pretending to be other computers’ ‘programs pretending to be other programs’

**Emulation**

Being a comp sci student and a former gamer, I will not write much about emulation as yself and most of my study group have a lot of experience using emulators to emulate consoles such as Playstation or Xbox to run on a pc. Or emulating classic game consoles. I get this subject quite well, below are a simple few notes to remember.

• halting problem

• computability

• halting problem not computable

 • The halting problem is a practical, important problem with significant applications.

• No computer can ever be able to solve the halting problem.

• There are problems that computers cannot solve and we should be aware of this so that we do not waste our time trying.