WEEK 1

Lecture-1

INTRODUCTION:

In the first lecture we tried to answer some very important questions namely

- What are computational problems.
- Given a solution , how to prove its correctness
- How to compare two solutions and what should be the parameter to decide which solution is correct.
- Given all this,how to find the most optimal solution or algorithm for a problem and then how to prove it is the most optimal solution

Computational problems:

There are so many problems in the word like of global warming,climate change etc, but all of them cannot be computational problems.

We understand that there can be several ways of posing the same problem for example the following two problems:

- What is the smallest prime number.
- What is the smallest even natural number

Both are essentially the same problem posed in different manner so we can classify all such similar problems into one master problem which we can call as a computational problem.

Formally a computational problem is a mathematical object representing a collection of questions that computers might be able to solve.

Languages and Membership Queries:

We can define each problem as a membership problem where the solution will be a 1-bit answer true or false. That is

> 💡 Does my input X belong to the set of all inputs which give output one?

This can allow us to pose every computational problem as a membership query (that is whether the output belongs to the set of solution for the given problem or not) in 'languages'.

Languages:

Let us try to understand this using an example :

Say we want to check if the given array is sorted or not , we can make a language L_sorted where we can encode the sequence of binary numbers separated by some terminator.

So the array {1,2,3,4,5} belongs to the this set while the array {2,3,1,5,4} does not .So our original problem reduced to a membership query of whether the given set of bits belongs to the language L_sorted or not.

Axioms of Computing:

- Only finite information can be stored in a finite volume.
- A finite-length code can exert only a limited amount of control.
- It takes some finite non - zero time to get data from far-away locations.

How to prove Correctness:

Given a solution or algorithm , naively we can try to prove it's correctness by generating test cases but this might not be the best method to prove correctness of our code because there can always be some cases which we might miss.

Therefore we mush always try to prove our solution by techniques such as induction etc.

How to compare solutions;

Now given two problems we need to find a way to compare the two solutions . Firstly we need to decide the parameters on which we want to judge our solution and then give them priorities.

For example some of the parameters can be time,space,energy,input size ,etc.

Among them time is considered as the most important resource among all since time once gone cannot be brought back It is irreversible while we can generate more memory or power . Hence unless explicitly specified we always try to compare solutions based on the time it took for the solution to run.

Hence by the end of the lecture we had a thorough understanding of what constitutes computational problems and how to compare the various solutions to these problems.

Lecture-2

INTRODCTION:

In the second lecture we followed up on the basics of computational theory and tried to answer some questions like:

- Are all problems solvable?
- How many computational problems exist and how many solutions exist

Number of Computational Solutions:

We tried to understand how many solutions exist.

It turns out that the number of solutions are countable.

Let's try to prove our hypothesis, firstly without loss of generality let us assume all our solutions are in C programming language since programs in various languages are inter convertible so any problem can written in some programming language can be written in C also.

Now we discussed one axiom in last class that a computer program can only be of finite length and can exert only finite amount of control.

Hence we can consider the C programs as binary strings of finite lengths , now we know that the set of binary strings consisting of 0,1 and of finite length are countable and hence the number of solutions also have to be countable because we cannot store infinite length codes in finite amount of memory which was another axiom we started with .

Proof that binary strings of finite length are countable:

To prove this, we just need to show a bijective mapping between the set of all strings and the set of natural numbers and we can do this as follows:

1 —> Empty string

2 —> 0

3 —> 1

4 —> 00

5 —> 10

6 —> 01

7 —> 11

and so on

In this manner we can map every string to a natural number and hence the proof is complete.

Hence the number of solutions are finite.

Now let us check if the number of solutions are also finite .

Number of Computational Problems:

We studied in last lecture that any problems can be posed as a membership query with a true and false answer . Hence the number of problems would atleast be equal to the cardinality of the power set of the number of computational solutions.

But from Cantor's diagonalization proof, we know that the cardinialty is uncountable.

We can prove this using contradiction ,let us assume that the number of size of subset is countable. Hence we can treat all the elements again as binary string and list them in order. Now we try to construct a element that is part of the power set but hasn't been considered till now .

Say we have some strings of n length as follows:

000000000....

100000000...

010000000...

110000000..

001000000... and so on

Now to construct the new string we can iterate through all these strings and move our bit pointer to the right starting from extreme left.

If at the current position, the pointer points to 0 , then in the new string in that position we can place 1 and if there is a 1, then we can put 0 in our new string.

Hence we the end of our process we would have obtained a string of finite length that doesn't belong to the already chosen group of strings .

Hence contradiction

Therefore the cardinality of power set has to be uncountable

which implies that the number of problems are also uncountable.

Hence we arrive at a very intresting result that number of problems are uncountable but number of solutions are finite and hence there definitely exists problems solutions to which we don't know.

Is C program Turing complete:

In order to prove whether C program us Turing complete or not, we would atleast need to prove that it is atleast as powerful as a Turing machine. So if we can stimulate the Turing system using our C code then we can say we have a Turing complete system .

Thus from what we know about Turing machines , the language will have to support conditional branching and go-to instructions which we know are present in the C language or for that matter any programming language.

Hence any programming language is capable of being Turing complete.

Apart from all this , I also learned about Godel's proof of incompleteness and how Turing machine actually functions.

After these two lectures, we got a very good basic understanding of computational theory.