Algorithms Analysis and Design

Week 1 - Diary

Ayan Agrawal (2020101034)

1). Introduction

1. In this course, we are studying and analyzing some important algorithms. For this, we must know what are algorithms. This is how we define them - They are sequence of steps which when followed end up in solving the given problem. But before we start, we need to clear some definitions and concepts.

- **Decision problems** - These problems have answer in either `YES` or `NO` (single-bit answer, i.e, 0 or 1) for any given input.
- **String** - A string is a finite sequence of symbols from alphabets sets. An empty string is denoted by ε.
- **Language** - A set of strings from all finite length strings combined forms a language according to Prof. Word "finite" is as of no use here because strings are meant to be finite length sequences.

  Each decision problem is characterized by a subset of the set of all possible inputs. This subset is formed by all the inputs for which answer is 1 (Yes). The superset of this set is **{0,1}***. This is called as ***Characteristic Language*** of that problem.

  The superscripted `*` after `{0,1}` is representing set of the strings formed by all possible combinations of 0 and 1. It is a widely used representation of Discrete mathematics.
- **Computational Problem** - It can be seen as a membership query in a language. Membership query implies that if a input string P $\in$ L (L is a language), it returns `YES` , otherwise `NO`.

  Some of the following problems can be considered as computation problems :-

  1. Which is the only even prime number?
  2. What is the GCD of set of even positive integers?
  3. Primality Test ($\text{Set of primes} = \{N \mid \text{N is prime}\}$)

  There are some challenges which we happen to face, related to these type of problems:-

  1. There may be infinite ways of posing the same problem

  - Here, for example, Which is smallest odd prime? What is the GCD (18,27)? What is $\sqrt{9}$? All these problems have solution as 3, which means all these problems are different still same from a computational p.o.v. So, there are infinite ways to pose a problem.

2. How do we pose a problem without solving it

Consider the problem of sorting, it is posed as :-

- *Given a sequence of N integers, among all possible permutations of this sequence , we need to select a sequence such that it is ascending in order*, *i.e.* ,$p_i < p_{i+1} \; \forall \; 1 \leq i < n$ where $p_i \text{ is the } i^{th} \text{element in } \{1, N\}$.

  Here, we can clearly see that we have in a way written solution to the above stated sorting problem. We need to iterate over all possible $n!$ permutations and pick a sequence such that it is ascending in order. This way of posing a problem when we still don't have a solution is helpful and we must perform it even if solution doesn't exist.

3. What are kinds of tools that we can use to solve a problem?

- Professor here talked about the *Mythological* Sort Algorithm considering various solutions that are allowed to solve the problem, meaning of "solving" changes. He quoted algorithm as:-

  1). Meditate until god appears

  2). Provide input numbers to god

  3). Obtain the sorted numbers from god

- This is not a *computational* solution as we mentioned above, therefore it is necessary to impose a *computational* constraint on problems.

2). Axioms of Computation

According to Prof. Kannan, there are some basic "axioms" or assumptions, we make when defining a solution.

- **It takes non-zero time to retrieve data from far-off locations,i.e., machines are not *omnipresent*.**
- **Only a finite amount of information can be stored and retrieved from a finite volume i.e., machines are not *omniscient*.**
- **A finite length code exerts only a finite amount of control i.e., machines are not *omnipotent*.**

NOTE: All these assumptions are made due to limitations of technological advancements of our time. If we are able to make more advancements in technology and break any of the above mentioned "axioms" , then we will in fact be able to develop such a computation model such that we will be able to solve much harder problems than we are able to do today.

3). How do we compare 2 computational solutions?

Now, we need a way to compare two different solutions to a problem and say deterministically "*which solution is better*?".

In the field of complexity theory, we usually do worst-case analysis,asymptotic analysis or average-case analysis. We analyse the performance of a solution by the means of input size. But, we do know this is not the best way of judging the 2 solutions. It maybe possible sometimes that algorithm `A` has constant time which is very large and is still less than time taken in a large-sized input in algorithm `B`. But, it is true that we may not give large input everytime. In this case, it might be better to compare the Best case of the algorithm.

To judge which algorithm is "better", we can say that the solution which uses lesser resources to compute is better. However, then arises one more challenge that computing a solution does not depend on only one resource. ***Time*** is a precious resource but ***space*** and ***power*** are also important resources.

However, most of the time, the dominating factor when comparing various solutions is the time taken to execute them as power can be generated and memory(space) can be created but time once lost never returns.

Lecture 2 :

We started the discussion on

- Are all the problems having a solution?
- How many computational problems exist and how many solutions actually exists

1. **Number of computational solutions**

- First of all, we tried to understand how many solutions exist in the world and it turns out that the number of solutions are countable.
  Let's try to prove our hypothesis, first let us assume all our solutions are in C programming language since programs in various languages are inter-convertible.

  We can consider the C programs as binary strings (finite length) , now we know that the set of binary strings consisting of $\{0, 1\}$ and of finite length are countable which in turn means Number of solutions existing are also countable as we can't store infinite length codes in finite memory (axiom).

A proof to prove binary strings of finite length are countable :-

$1 \rightarrow$ NULL String

$2 \rightarrow 0$

$3 \rightarrow 1$

$4 \rightarrow 00$

$5 \rightarrow 01$

$6 \rightarrow 10$

$7 \rightarrow 11$

and so on...

- Therefore, every binary string of finite length can be mapped to set of natural numbers. As, set of natural no.s is countable, hence binary strings of finite length are also countable.

  So, Number of solutions are finite, now lets check for number of problems exisiting.

2. **Number of computational problems**

- We saw in the last lecture that "Any problems can be posed as a membership query with T/F as answer" . Hence the total number of problems would atleast be equal to the cardinality of the power set of the computational solutions.
  But from Cantor's diagonalization proof, we already know that the cardinality is uncountable.
- Lets prove this using contradiction , we'll assume that the size of subset is countable. Hence, we assume all elements as a binary string and list them in order. Now we will try to construct a string that is in the power set but has not been taken into consideration,

  Suppose we have strings such as 000000, 000001, ...so on.

  Now, if we are building a new binary string, we would start with say 0 and then have 1 assuming we already have a string with 0 after 1st 0. On seeing the existing strings and making a new one like this, we would end up creating a whole new string out of the group of some selected strings.

  Hence, this is a contradiction as we would exploit the current set of strings. So, the number of strings set is uncountable which basically means "Number of computational problems are uncountable."

> Hence, we arrive at a very interesting fact that number of computational problems is uncountable but number of solutions existing is countable and hence there are problems to which a solution is unknown to us.

- After these 2 lectures, we got a nice understanding about the computational point of view of problems existing in our day-to-day lives.