$\mathbf{P} \stackrel{?}{=} \mathbf{NP}$

Prove/Disprove it and be a millionaire

1 Introduction to Complexity Classes

In the previous lecture we learnt the notion of computability and complexity, i.e. running time of a TM. In this lecture we introduce you to the vast field of Computational Complexity Theory, by introducing to the two fundamental complexity classes P and NP.

2 Complexity Classes

So, what are these complexity classes after all? You would have noticed in your ESC101A classes as well that some problems are easy to solve and program while some are not that easy. That, along with the existence of uncomputable problems would give you a hint that problems are of varied difficulty. And it is the classification of the problems into different difficulty(complexity) classes that we study in Computational Complexity Theory. This division is based on the amount of resources consumed, usually time, space and energy.

3 DTIME

Let $T : N \to N$ be some function. A language L is in $\mathbf{DTIME}(T(n))$ iff there is a Turing machine that runs in time $c \cdot T(n)$ for some constant $c > 0$ and decides L.

3.1 P

P is the class of all languages decidable in polynomial time, i.e. they are efficiently solvable. Formally

$$\mathbf{P} = \cup_{c>1}\mathbf{DTIME}(n^c)$$

All the problems you have solved in your ESC101A lab and exams are in P. Well they were in P and that is why it was easy to solve them, that's one way of saying it.
Some examples include: Sorting, Searching, Graph Traversal(BFS and DFS) and Primality testing(this was recently proved by none other than our 2 IITK alumni and a professor Prof Manindra Agarwal, Nitin Saxena and Neeraj Kayal which won them Godel prize. You can check out Prof Agarwal's homepage as well.
Note that the class P is same irrespective of the variant of TM.

4 Solving vs Verifying and NP

4.1 Motivation for NP

You would have experienced it as well that the problem seems so easy when your chaapu friend explains it and all you can do is exclaim ,"bhak bc!" kat gaya. Well there's something more to it. The motivation behind the class NP is that verifying a solution to a problem is lot easier that solving it, we all believe and have experienced it. So, NP is basically the class of all problems whose solution can be verified easily. Note that if a problem can be solved easily it's solution can also be verified easily.

4.2 NP

Formally:
A language $L \subseteq \{0,1\}^*$ is in NP if there exists a polynomial $p : N \to N$ and a polynomial-time TM M (called the verifier for L) such that for every

$$x \in \{0,1\}^*, x \in L \iff \exists\, u \in \{0,1\}^{p(|x|)} \ s.t.\ M(x,u) = 1$$

If $x \in L$ and $u \in \{0,1\}^{p(|x|)}$ satisfy $M(x,u) = 1$, then we call u a certificate for x (with respect to the language L and machine M).
Examples: Traveling Salesman problem. Independent Set Problem, every problem in P, etc.
Clearly, $P \subseteq NP$, no certificate needed, $p(x) = 0$.

5 NDTM

Just like we had deterministic TM, we can have Non-deterministic TM(NDTM) as well. An NDTM has two transition functions q_0 and q_1 and at every stage of computation it applies one of them, independently. So on some input one sequence of choices of the transition function can lead to some output if it halts.
An NDTM M decides a language L if for all strings x in L, there exists some sequence of choices of the transition function on which M accepts x,i.e. outputs 1.
We say an NDTM M computes L in $T(n)$ time if for every string $x \in \{0,1\}^*$, M halts within $T(|x|)$ steps irrespective of the choices made.

5.1 NTIME

For every function $T : N \to N$ and $L \subseteq \{0,1\}^*$, we say that $L \in \mathbf{NTIME}(T(n))$ if there is a constant $c > 0$ and a $c \cdot T(n)$-time NDTM M such that for every $x \in \{0,1\}^*, x \in L \iff M(x) = 1$.

Theorem: $\mathbf{NP} = \cup_{c>1}\mathbf{NTIME}(n^c)$

6 Reducibility and Completeness

As introduced in previous lecture, we reduce a problem to other as show that we can solve one if we can solve the other. Here we formalise this notion.

A language $L \subseteq \{0,1\}^*$ is polynomial-time Karp reducible to a language $L' \subseteq \{0,1\}^*$ (sometimes shortened to just "polynomial-time reducible"), denoted by $L \leq_p L'$, if there is a polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that for every $x \in \{0,1\}^*, x \in L$ if and only if $f(x) \in L'$.

We say that L' is NP-hard if $L \leq_p L'$ for every $L \in NP$. We say that L' is NP-complete if L' is NP-hard and $L' \in NP$.

Hard problems, as their name say, are at least as hard as every problem in the class. Complete problems are among the hardest problems in the class.
Some trivial properties:

1. $L \leq_p L'$ and $L' \leq_p L$", then $L \leq_p L$".
2. If language L is NP-hard and $L \in P$, then $P = NP$.
3. If language L is NP-complete, then $L \in P$ if and only if $P = NP$.

The problems Traveling Salesman Problem and Independent Set problem are NP complete, but we first prove completeness of another language, and then reduce that language to these two languages showing they are NP complete (reduction showed their hardness, and they were already in NP so, NP complete).

7 SAT is NP complete

Boolean Formula: We all know what a Boolean formula in n variables look like. Example: $(x_1 \wedge x_2) \vee (\neg x_2 \vee x_3)$. A boolean formula is said to be satisfiable if some assignment to variables exists which makes the formula TRUE.
CNF form: Conjunctive Normal Form, you would have enocuntered it in your 12th standard also that any Boolean function can be represented in AND of OR or OR of AND form. This AND of OR representation is known as CNF of the Boolean Formula. A CNF formula has the form:

$$\bigwedge_i (\bigvee_j v_{ij})$$

SAT: SAT is the language of all satisfiable CNF Boolean Formula.

Theorem: SAT is NP-complete.
Proof Sketch:
First see that SAT is in NP, the assignment which satisfies the formula could work as a certificate.
Now we have to prove that SAT is NP hard.
Say some language $L \in NP$, this means that a polynomial time TM M exists s.t. $x \in L \iff \exists u$ (of some polynomial in input size length) s.t. $M(x,u) = 1$. Talk in terms of this verifier M, now a string $x \in L \iff \exists u$ s.t. M accepts (x,u). We can prove hardness of SAT by giving a transformation f s.t. $x \in L \iff$ f(x) $\in SAT$, i.e. for a string x

$$\exists u \; s.t. \; M(x,u) = 1 \iff f(x) \in SAT$$

So, we try to give a transformation which maps any string x to a Boolean Formula φ_x s.t. $x \in L$ iff φ_x is satisfiable.

Now it should be clear what we have to do, we have to give a Boolean Formula(CNF) φ_x for every string x s.t. $\exists u \; s.t. \; M(x,u) = 1 \iff f(x) \in SAT$. See that existence of certificate is related to satisfiability, i.e. existence of a satisfying assignment. This should click you now, how are we going to do it, we'll try to give a Boolean Formula whose satisfiability depends on M accepting some certificate for x.

We, in a way, turn the certificate into a satisfying assignment. The Boolean Formula that we create for x takes steps of execution of the verifier M on the certificate as input and returns True if the steps are consistent and M halts in 1. Our Boolean Formula, in a way checks whether M accepts x for some certificate. To be able to talk about it, we use two tape Oblivious TM, (otherwise this checking wouldn't have been done easily). Remember that oblivious TM are those whose head movements are dependent only on the size of input and the step number(first step, second step, etc.).

Snapshot of TM M - the tuple of the symbols under the two heads and the state of TM.
Given a sequence of snapshots corresponding to the steps of execution of M, we can inductively easily verify if that sequence is a valid one, i.e. it is actually the one that will be produced by M on the input (x,u). See that it's easy to check the validity of first snapshot. Let us assume then that upto i^{th} snapshot they are valid. Now we go for the $i+1^{th}$, we have to check if $i+1^{th}$ snapshot will indeed be the correct one. See that a snapshot will be valid if it correctly gives the tape's contents under the heads and the state. Now we know where the heads would be (oblivious TM), we only need to check the state and contents. Input tape being read only, should have the same unaltered symbol, for the work tape, if we know the last snapshot for which the work tape's head was at the same position(also easily known since 'oblivious' TM), we know what to expect there. And the state can be verified since we know the transition function, we can find the $i+1^{th}$ state from i^{th} snapshot.

The Boolean formula will take these snapshots as one input, and on a valid one, the formula will evaluate to be true, it will be a satisfying assignment.
Read out the details from the book.

A lot of interesting problems can be shown to be NP-complete by reducing SAT to them.

8 P vs NP and beyond

The problem of determining whether $P = NP$ is so central to the complexity theory that it is one of the millenium problems. It has far reaching mathematical and philosophical importance. Imagine if someone proves $P = NP$ (which is not what most people believe), then for decades almost entire workforce in Computer Science would focus on discovering efficient algorithms for those problems, now that they know one exists.