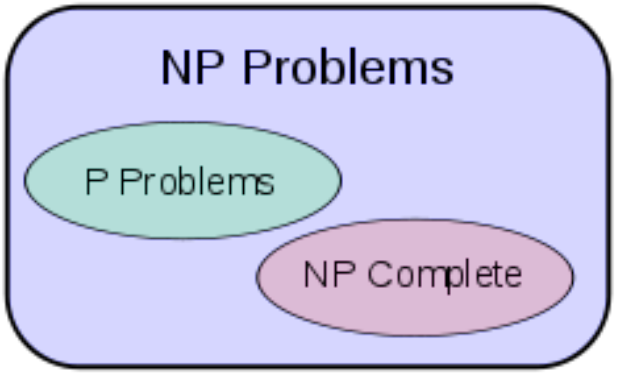
National Institute of Technology, Delhi

Name: UTAM KUMAR
Roll no.: **181210055**
CSE 2nd Year

What are P and NP problems?



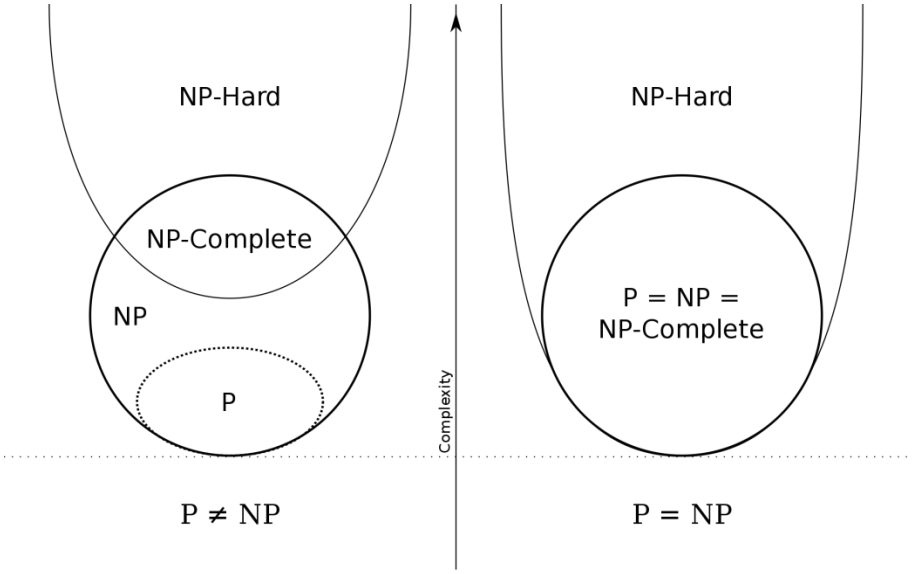
- P (polynomial time) contains all decision problems that can be solved by a deterministic Turing machine using a polynomial amount of computation time, or polynomial time. They are the relatively 'easier' set of problems.
- NP (non-deterministic polynomial time) refers to the class of problems that can be solved in polynomial time by a non-deterministic computer. It refers to the class of problems that currently, has no way of finding a quick (polynomial time) enough answer, BUT can be quickly verified (in polynomial time) if one provides the solution to the problem.

NP-Complete and NP-Hard

- Amongst the NP problems, there exists a King of all problems which researchers call NP-Complete problems. Formally, they are a set of problems to each of which any other NP problem can be reduced (addressed below) in polynomial time and whose solution may still be verified in polynomial time. This means that any NP problem can be transformed into a NP-Complete problem.

- Thus if any one NP-Complete problem can be solved in polynomial time, then every NP-Complete problem can be solved in polynomial time, and every problem in NP can be solved in polynomial time (i.e. P=NP). The most famous example would be the Traveling Salesmen problem.

NP-Hard Problems



There also exists a set of problems called NP-Hard problems. These problems are at least as hard as NP problems, but without the condition that requires it to be solved in polynomial time. This suggests that NP-Hard problems may not necessarily be part of the NP class. An example would be solving a chess board — given a state of a chess board, it is almost impossible to tell if a given move at the given state is in fact the optimal move. Formally, there exists no polynomial time algorithm to verify a solution to a NP-Hard problem.

If we put the two together, a NP-Complete problem implies it being NP-Hard, but a NP-Hard problem does NOT imply it being NP-Complete.

Real Life Example-Register Allocation

- Optimal register allocation (allocating registers such that the numbers of memory references are minimized) is based on coloring the interference graph – This is an NP-hard issue.
- Graph coloring is a relatively simple method which can be used for some of the scheduling problems, e.g. for register allocation.
- To apply graph-coloring to register allocation, we first need construct an interference graph.
- Next, we color the interference graph using K different colors, where K is the number of registers available for allocation. No pair of nodes which are connected by an edge may be assigned the same color.
- If it is impossible to color the interference graph with the given K colors, then we will have to keep some of the values (represented by corresponding vertices) in the memory (for at least part of their lifetime).
- The compiler should generate the code such that a live value will either reside in a register or in a memory location. Before the program overwrites a register which stores a still-live value, that value must be saved to a memory location. This is called spilling the register, and the memory location to save the spilled value called its spill location.
- If P=NP, then some computer programs would get slightly faster, since compilers use graph coloring for register allocation. We would be able to allocate for large numbers of registers exactly. Existing compilers using an approximate solution would get better output, and those using an exact solution would get faster.