CSC-257
Theory Of Computation
(BSc CSIT, TU)

Ganesh Khatri
kh6ganesh@gmail.com

Computational Complexity

- Complexity Theory is a central field of the theoretical foundations of Computer Science.
- It is concerned with the study of the complexity of computational tasks.
- That is, a typical Complexity theoretic study looks at a task (or a class of tasks) and at the computational resources required to solve this task.
- The complexity of computational problems can be discussed by choosing a specific abstract machine as a model of computation and considering how much resource machine of that type require for the solution of that problem
- Complexity Measure is a means of measuring the resource used during a computation.
- In case of Turing Machines, during any computation, various resources will be used, such as space and time

Computational Complexity

- A problem is regarded as inherently difficult if solving the problem requires a large amount of resources, whatever the algorithm used for solving it.
- The theory formalizes this intuition, by introducing mathematical models of computation to study these problems and quantifying the amount of resources needed to solve them, such as time and storage.
- Other complexity measures are also used, such as the amount of communication (used in communication complexity), the number of gates in a circuit (used in circuit complexity) and the number of processors (used in parallel computing).
- In particular, computational complexity theory determines the practical limits on what computers can and cannot do.

Time and Space Complexity of a Turing Machine

- The model of computation we have chosen is the Turing Machine.
- When a Turing machine answers a specific instance of a decision problem we can measure time as number of moves and the space as number of tape squares, required by the computation.
- The most obvious measure of the size of any instance is the length of input string.
- The worst case is considered as the maximum time or space that might be required by any string of that length.

Time and Space Complexity of a Turing Machine

- The time and space complexity of a TM can be defined as :
- Let T be a TM, the time complexity of T is the function T_t defined on the natural numbers as; for n ε N, $T_t(n)$ is the maximum number of moves T can make on any input string of length n.
- If there is an input string x such that for |x| = n, T loops forever on input T, $T_t(n)$ is undefined
- The space complexity of T is the function S_t defined as $S_t(n)$ is the maximum number of the tape squares used by T for any input string of length n.
- If T is multi-tape TM, number of tape squares means maximum of the number of individual tapes.
- If for some input of length n, it causes T to loop forever, $S_t(n)$ is undefined.

Intractability

- Intractability is a technique for solving problems not to be solvable in polynomial time.
- The problems that can be solved by any computational model, probably TM, using no more time than some slowly growing function size of the input are called "tractable:, i.e. those problems solvable within reasonable time and space constraints (polynomial time).
- The problems that cannot be solved in polynomial time but requires super polynomial (exponential) time algorithm are called intractable or hard problems.
- There are many problems for which no algorithm with running time better than exponential time is known some of them are, traveling salesman problem, Hamiltonian cycles, and circuit satisfiability, etc.

Intractability

- To introduce intractability theory, the class P and class NP of problems solvable in polynomial time by deterministic and non-deterministic TM's are essential.
- A solvable problem is one that can be solved by particular algorithm i.e. there is an algorithm to solve this problem.
- But in practice, the algorithm may require a lot of space and time.
- When the space and time required for implementing the steps of the particular algorithm are (polynomial) reasonable, we can say that the problem is tractable.
- Problems are intractable if the time required for any of the algorithm is at least f(n), where f is an exponential function of n

Complexity Classes

- In computational complexity theory, a complexity class is a set of problems of related resource-based complexity.
- A typical complexity class has a definition of the form - “The set of problems that can be solved by an abstract machine M using O(f(n)) of resource R, where n is the size of the input”

Complexity Classes

- **Class P :**
  - The class P is the set pf problems that can be solved by deterministic TM in polynomial time.
  - A language L is in class P if there is some polynomial time complexity T(n) such that L = L(M), for some Deterministic Turing Machine M of time complexity T(n).
- **Class NP :**
  - The class NP is the set of problems that can be solved by a non-deterministic TM in polynomial time.
  - Formally, we can say a language L is in the class NP if there is a non-deterministic TM, M, and a polynomial time complexity T(n), such that L= L(M), and when M is given an input of length n, there are no sequences of more than T(n) moves of M

Complexity Classes

- **NP Complete Problems :**
- In computational complexity theory, a problem is NP-complete when :
  - A nondeterministic Turing machine can solve it in polynomial-time
  - A deterministic Turing machine can solve it in large time complexity and can verify its solutions in polynomial time.
- **NP Hard Problems :**
  - A problem is NP-hard if an algorithm for solving it can be translated into one for solving any NP-problem (nondeterministic polynomial time) problem.
  - NP-hard therefore means "at least as hard as any NP-problem," although it might, in fact, be harder

Types of Computational Problems

- **Abstract Problem :**
  - Abstract problem A is binary relation on set I of problem instances, and the set S of problem solutions. For e.g. Minimum spanning tree of a graph G can be viewed as a pair of the given graph G and MST graph T.
- **Decision Problems :**
  - Decision problem D is a problem that has an answer as either “true”, “yes”, “1” or “false”, ”no”, “0”.
  - For e.g. if we have the abstract shortest path with instances of the problem and the solution set as {0,1}, then we can transform that abstract problem by reformulating the problem as “Is there a path from u to v with at most k edges”.
  - In this situation the answer is either yes or no

Types of Computational Problems

- **Optimization Problem :**
  - We encounter many problems where there are many feasible solutions and our aim is to find the feasible solution with the best value.
  - This kind of problem is called optimization problem.
  - For e.g. given the graph G, and the vertices u and v find the shortest path from u to v with minimum number of edges.
  - The NP completeness does not directly deal with optimizations problems; however we can translate the optimization problem to the decision problem.

Types of Computational Problems

- **Function Problems :**
  - In computational complexity theory, a function problem is a computational problem where a single output (of a total function) is expected for every input, but the output is more complex than that of a decision problem, that is, it isn't just YES or NO.
  - Notable examples include the Traveling salesman problem, which asks for the route taken by the salesman, and the Integer factorization problem, which asks for the list of factors.
  - Function problems can be sorted into complexity classes in the same way as decision problems.
  - For example FP is the set of function problems which can be solved by a deterministic Turing machine in polynomial time, and FNP is the set of function problems which can be solved by a non-deterministic Turing machine in polynomial time.