Indian Institute of Information Technology
Design and Manufacturing, Kancheepuram
Chennai 600 127, India
An Autonomous Institute under MHRD, Govt of India
http://www.iiitdm.ac.in
COM 501 Advanced Data Structures and Algorithms

Instructor
N.Sadagopan
Scribe:
P.Renjith

Lecture- Complexity Class- P and NP

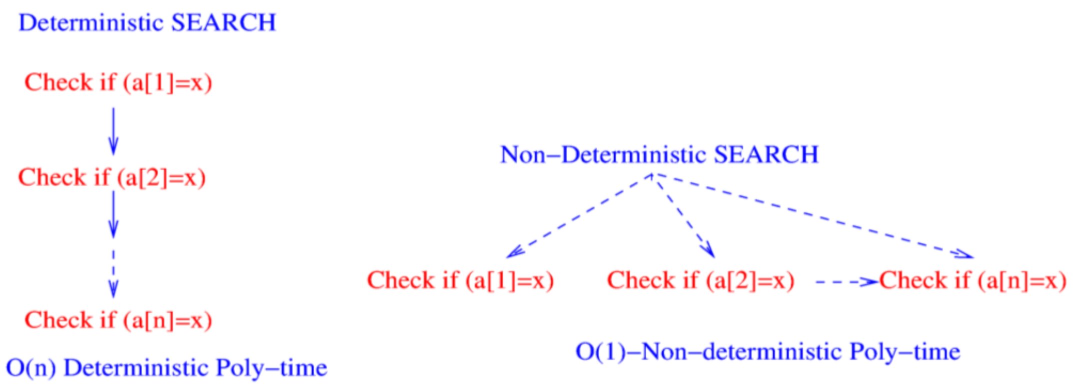
Computational problems can be broadly classified into *solvable* and *unsolvable* problems. Solvable problems are problems that have an algorithm running in finite time whereas for unsolvable problems, every algorithm takes infinite time to complete. For example, printing all natural numbers, printing all real numbers, and halting problem are unsolvable. Problems such as sorting and searching are solvable. Restricting our attention to solvable problems, we can think of many deterministic approaches to solve a given problem. Among different approaches possible, polynomial-time solutions are of practical interest. That is, algorithms running in time $n^k, k \in Z^+$, where n is the input size. A natural question is whether all solvable problems can have polynomial-time solutions? Interestingly, enumeration problems such as enumerating all possible spanning trees of an arbitrary graph of size n, enumeration of all subsets of a set of size n etc., have exponential-time complexity. The former incurs $\Omega(n^{n-2})$ time-effort and the latter incurs $\Omega(2^n)$ time-effort. This shows that not all solvable problems have polynomial-time solutions.

This brings a classification among solvable computational problems, namely *tractable* and *intractable* problems. A problem is tractable if there exists a polynomial-time algorithm. On the other hand, intractable problems of input size n have $\Omega(c^n), c > 1$. I.e., every algorithm for such problems incurs $\Omega(c^n)$. An immediate and a natural question at this context is that can every solvable problem be classified into tractable and intractable classes. Interestingly, there are computational problems like travelling salesperson problem, maximum independent set problem, maximum clique problem, minimum vertex cover problem, minimum steiner tree problem, etc., for which, as of this date, there is no known polynomial-time algorithm, neither they have a exponential-time lower bound. That is, such problems cannot be included either in tractable or intractable class. We call such problems as HARD problems.

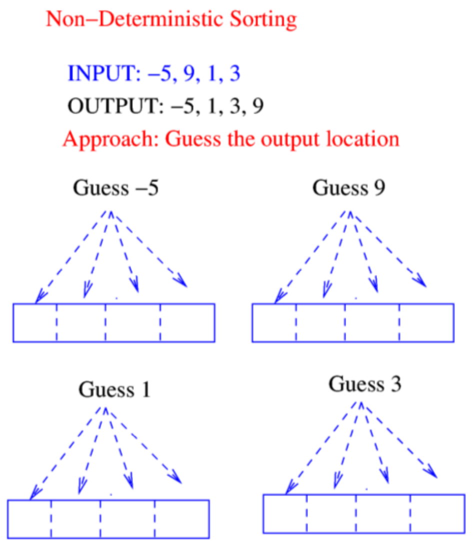
Algorithms in practice is usually made to run on a deterministic machine whose underlying computational graph is a path. Can the computational model be altered so that it is different from the traditional deterministic computations? Motivated by this line of research, we introduce the non-deterministic model in which the computational graph is a tree instead of a path. The non-deterministic computation takes a non-deterministic move (guess) at each step of the computation. It branches to multiple partial solutions at each move. That is, it guesses the (partial) solution at each step, the guess is always correct and finally a correct solution is obtained at one of the leaves of the computational tree. The sequence of correct guesses constitute a solution which is a path in the associated tree. We can see the non-deterministic model as a powerful model that is capable of computing more tasks at once and produces the correct solution if the input has one such solution.

Let us revisit tractable (intractable) problems. In particular, tractable problems are problems that have polynomial-time algorithms under a deterministic computational model and intractable problems have exponential-time algorithms under a deterministic computational model. To cope-up with HARD problems, we shall introduce non-deterministic model and ask, do HARD problems have polynomial-time algorithms under this variant. This line of study was introduced by researchers Stephen Cook and Leonid Levin. Subsequently, they introduced complexity classes P and NP. The complexity class P contains the set of problems that are solvable in deterministic polynomial time and the complexity class NP contains the set of problems that are solvable in non-deterministic polynomial time. Classical problems such as minimum spanning tree and shortest path are belong to class P. We shall next see some examples for the non-deterministic computation.

Example 1: Non-deterministic Linear search. Input: Array A, key k. Check whether $k \in A$.
In a deterministic linear search, the elements in the array are searched one after the other, and a successful match is acknowledged. In the non-deterministic computation model, the computational graph resembles a star. The location of the key element, if present is returned by the non-deterministic search (guessing the solution) in constant time.



Example 2: Non-deterministic Sorting
Note: The input array is in $A[1..n]$ and output array is in $B[1..n]$.
The non-deterministic sorting algorithm guesses for each $A[i]$ the position in $B[\ ]$ array. It correctly guesses for each element in $A[\ ]$ and in total the model makes $O(n)$ guesses. Therefore the non-deterministic sorting is performed in linear time. Note that under deterministic model, any comparison based sorting algorithm has a lower bound of $\Omega(n \log n)$



Example 3:
Hamiltonian cycle problem
Input:-Graph G
Question: Does G have a Hamiltonian cycle?
A trivial deterministic algorithm for this problem generates all the permutations of the vertices of G and checks whether a permutation forms a spanning cycle. Thus, this approach incurs $O(n!)$. In the non-deterministic model, it correctly guesses the first vertex, followed by the second, and so on. A computational graph is shown in Figure.1. The guessing part guesses the right path in the computational tree, which gives

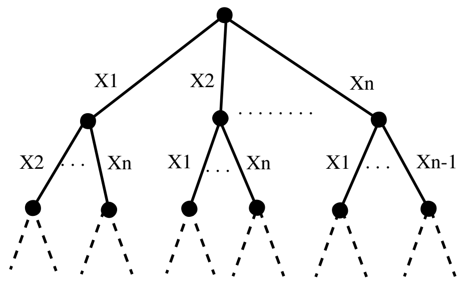


Figure 1: Non-Deterministic computation graph: Hamiltonian cycle problem

a permutation at the leaf node in polynomial time. Checking whether a permutation is a spanning cycle or not can be done using a deterministic polynomial-time algorithm. Therefore, there exist a non-deterministic polynomial-time algorithm to solve this problem and hence it is in NP.

Example 4:
Boolean Satisfiability
Input: Boolean formula F on variables $x_1, \ldots, x_n$
Question: Is F satisfiable?
For example, for the variables $\{x_1, x_2, x_3, x_4\}$ and $F = (x_1 \vee \bar{x_2} \vee v_3) \wedge (x_1 \vee x_2 \vee \bar{x_4}) \wedge (x_2 \vee x_3 \vee x_4)$, find a truth assignment for $\{x_1, x_2, x_3, x_4\}$ such that $F(\{x_1, x_2, x_3, x_4\})$ is evaluated to be true. A trivial deterministic algorithm try all 2^n possibilities and thus runs in $\Omega(2^n)$ time. For the non-deterministic approach, it correctly guesses the truth assignment for each variable $x_i, 1 \leq i \leq n$, as shown in the Figure 2. Note that in the computational graph, each leaf corresponds to a truth assignment for the variables $x_1, \ldots, x_n$. Since the guess at each step made by the non-deterministic model is always correct, the path chosen results in an assignment which is in turn a satisfiable assignment if F has one such assignment. Also there exist a

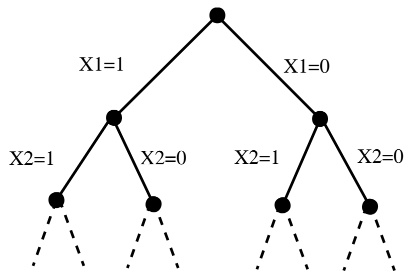


Figure 2: Non-Deterministic computation graph: Satisfiability problem

deterministic polynomial-time algorithm to check the validity of the truth assignment in polynomial time. Therefore, 3-SAT is in NP.

Example 5:
Input: Graph G with the vertex set $V(G) = \{v_1, \ldots, v_n\}$ and the edge set $E(G)$
Question: Find a Minimum Vertex Cover. A set $S \subset V(G)$ is called a vertex cover of G if for every edge $uv \in E(G)$ either u or v or both are in S.
In a deterministic approach, one has to find all subsets which are possible candidates for a minimum vertex cover, and check whether the subset is a valid vertex cover or not. Note that this method incurs exponential effort. In a non-deterministic approach, it guesses the vertices one after the other which are part of the minimum vertex cover. To find the minimum vertex cover one has to check all the leaves and finally output the minimum one. In this process, even though the model is a non-deterministic one, the polynomial time is not achieved due to the exponential number of leaves. One way to overcome this situation is to work with

the decision version of the problem, which is defined as follows.

Input: Graph G, Integer k
Question: Find a vertex cover of size at most k.

Note that the optimization version (minimum vertex cover) can be solved by making a polynomial number (here linear) of invocations to the decision version of minimum vertex cover problem. That is, ask for a vertex cover of size 1 in the first invocation, 2 in the second and so on till n. It can be observed that the decision version gives `NO` for the initial run and at a later point, there exist a run on size k that gives `YES`. Further, it will say YES for $k+1, k+2, \ldots.$ From this it can be easily inferred that the minimum vertex cover is of size k.

Moreover, using this decision version, note that the overhead involved in scanning the entire leaves for finding the minimum is not needed. Instead, once the correct guess returns a subset of vertices S, we can easily check whether S is a vertex cover or not in deterministic polynomial time. Therefore, minimum vertex cover is in NP.

A deterministic model analogy to the non-deterministic computational model is a parallel processing model that has an array of parallel processors. Note that the run time of non-deterministic algorithms can not be compared with that of the deterministic algorithms as they differ in the underlying computational model. Following are certain characteristics of the non-deterministic and deterministic models.
Characteristics of non-deterministic algorithms

– The computational graph of the non-deterministic computation form a tree.
– An algorithm which realizes the non-deterministic computation is called a *non-deterministic algorithm.*
– Every move of the algorithm from one state to another is not unique.

Characteristics of deterministic algorithms

– For a deterministic computation the computational graph is a path.
– An algorithm which realizes the deterministic computation is called a *deterministic algorithm.*
– Every move of the algorithm from one state to another is unique.

Class P and NP

Since optimization problems have equivalent decision problems, we shall focus on decision problems and let us define the complexity classes restricted to decision problems.

P={ X : X is a decision problem and X is solvable in deterministic polynomial time }

NP={ X : X is a decision problem and X is solvable in non-deterministic polynomial time }

Recall that the computational model NP guesses the solution and the solution is further checked using a deterministic polynomial-time algorithm. For example, for Hamiltonian cycle problem, after making $O(n)$ guesses, the algorithm checks whether the guess is indeed a Hamiltonian cycle using a deterministic polynomial-time verification algorithm. We have no clue on how the guessing is done, even though the model ensures a correct solution if it has one. Therefore, we focus only on the verification algorithm and present an alternative definition for NP. This brings a more easier method to check whether a given problem is in NP or not;

NP={ X : there exist a deterministic polynomial-time algorithm to verify an instance of X}

According to the above definition, the input to verification algorithm consists of parameters such as a graph (formula), a certificate, an integer. For example, for Hamiltonian problem:
Input:-Graph G, $S = (v_7, v_1, v_4, \dots, v_n, v_2, \dots, v_7)$
Question: Is S a Hamiltonian cycle?

The verification algorithm proceeds as follows; check whether there exist an edge between the adjacent vertices in the given permutation, i.e., $v_7 v_1, v_1, v_4 \in E(G)$, and so on. Clearly, such a check can be done in deterministic polynomial time. Similarly, is the case for problems like travelling salesman problem, independent set of size at most k, clique of size at most k, etc. Note that all decision problems which are solvable in deterministic polynomial time can also be verified in polynomial time. By the above definition, all problems in class P is also in NP. As a result, P $\subseteq$ NP. It is an open question whether P is a strict subset of NP.

Some more interesting problems in class NP
SAT, 3-SAT, 3-colorability, Travelling Salesman problem, Hamiltonian Cycle, Hamiltonian Path, Minimum Steiner tree, Minimum Dominating set, Minimum connected dominating set, Minimum Vertex Cover, Maximum Independent Set, Maximum Clique, Puzzles like Sudoku, Minesweeper.

Class NP consists of thousands of combinatorial problems, and the set NP is still growing in size. Surprisingly, no concrete progress has been made to show P=NP or P $\neq$ NP. While this being the status of NP one side, researchers Cook and Levin focused on identifying 'Hardest' problems in NP. A problem which is at least as difficult as every other problem in NP. In the next session, we shall see how to categorize such problems.

Reducibility between Combinatorial problems

For decision problems X and Y, X is reduced to Y if the following holds. Instances of X is `true` if and only if there exist a corresponding instance of Y evaluated to be `true`. It follows that to solve the problem X, we can use problem Y as a black box. Moreover, if the problem Y is polynomial-time solvable and the reduction is also possible in polynomial time, then the for every instance of problem X, we reduce it to Y, solve Y, and the solution to Y is used to obtain a solution to X. Therefore, the problem X can also be solved in polynomial time. Polynomial-time reduction is represented as $X \leq p\, Y$. Note that all the instances of X has an image in Y by the reduction. The converse is not necessarily true. That is, there may exist instances in Y which do not correspond to any instance of X. Reductions are useful in comparing the hardness between problems. That is, $X \leq p\ Y$ implies that the problem Y is at least as hard as problem X.

The class *NP-Hard* is a class of problems (not necessarily in NP) that are at least as hard as all the problems in the class NP. Therefore to show that a problem Y is in the class NP-Hard, we have to show a reduction from all the problems in the class NP to Y, $X \leq p\ Y$ for every X in NP. Stephen Cook and Leonid Levin showed that circuit satisfiability, boolean satisfiability, and 3-SAT are NP-Hard. They proved that there exist a polynomial-time reduction from every problem in NP to 3-SAT. Note that if we show a polynomial-time reduction from 3-SAT to a problem Y, then there exists a polynomial-time reduction from all the problems in NP to Y through 3-SAT using transitivity relation. That is, $X \leq p$ 3-SAT $\leq p\ Y$, for every problem X in NP. It follows from the definition that Y is NP-Hard. To show a problem Y is NP-Hard, it is sufficient to choose a known NP-Hard problem and show a polynomial-time reduction to Y.

Clique is NP-Hard
Input: Graph G with the vertex set $V(G) = \{v_1, \dots, v_n\}$ and the edge set $E(G)$, Integer k
Question: Does there exist a clique of at least size k. A clique is a set of mutually adjacent vertices in G.

Claim: Clique problem is NP-Hard

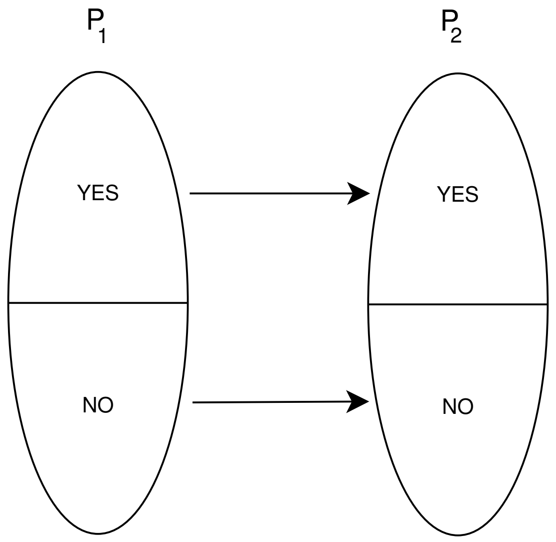


Figure 3: $X \leq pY$

Proof: We shall present a polynomial-time reduction from 3-SAT to clique. An instance of a 3-SAT consists of clauses $C = \{C_1, \ldots, C_m\}$ and variables $x_1, \ldots, x_n$ is reduced to the clique problem as follows: corresponding to each literal (variables or their negations) $x_i \in C_j$, $1 \leq i \leq n, 1 \leq j \leq m$, there exists a vertex in graph G. $E(G) = \{x_i x_j : x_j \neq \overline{x_i}$ and x_i, x_j are in different clauses $\}$. We shall see that there exist a satisfying truth assignment for 3-SAT if and only if there exist a clique of size m in G. For *if* part, since there are m clauses, each clause should be evaluated to 1 in any true assignment. For this, there exist a literal assigned 1 in each clause. Let $U = u_1, \ldots, u_m$ be the literals such that u_i is the literal assigned 1 in the clause C_i Clearly, there cannot be $u_i, u_j \in U$ such that $u_j = \overline{u_i}$. This implies that the vertices corresponding to the literals in U are mutually adjacent as no two literals in U belong to the same clause. Therefore, there exist a clique of size m in G.
For *only if* part, consider a clique of size m in the reduced graph G. Let the clique K be induced on vertices $v_1, \ldots, v_m$. Since there are no edges between the literals of a clause, there are no two vertices in K which corresponds to two literals of same clause. Also observe that there are no two vertices $v_i v_j$ in K with corresponding literals x_i, x_j such that $x_j = \overline{x_i}$. This implies that the literals $u_i, \ldots, u_m$ corresponding to the vertices in clique K can be assigned 1 to get a satisfiable assignment in the 3-SAT formula. Since 3-SAT is NP-Hard and from the above polynomial-time reduction (from 3-SAT to clique problem), it follows that clique problem is NP-Hard. □

Independent set and Vertex cover are NP-Hard
Input: Graph, G, Integer k
Question: Does there exist an independent set of size at least k. An independent set is a set of mutually disjoint vertices in G.

Claim: Independent set problem is NP-Hard.
Finding a clique of size k in G is reduced in polynomial time to finding an independent set problem in the complement graph of G. The complement graph $\overline{G}$ of G is such that $V(\overline{G}) = V(G)$ and $uv \in E(\overline{G})$ if and only if $uv \notin E(G)$. A clique of size k in G is mapped to an independent set of size k in $\overline{G}$. Therefore, the independent set problem is NP-Hard.

Claim: Vertex cover problem is NP-Hard.
Maximum independent set of size k in G is reduced to minimum vertex cover of size $n - k$ in G. That is, S

is a minimum vertex cover in G if and only if $G \backslash S$ is a maximum independent set in G. This implies that vertex cover problem is NP-Hard.

Note: C-SAT $\leq p$ 3-SAT $\leq p$ Clique $\leq p$ IS $\leq p$ VC.
C-SAT - Circuit satisfiability problem.
3-SAT - 3 CNF satisfiability problem.
VC- Vertex Cover problem
IS - Independent Set problem

NP Complete Problems

The difficult problems in class NP.
Problem X is NP Complete if
X $\in$ NP and
X is NP-Hard
Since there exist a polynomial-time verification algorithm for the independent set, clique, vertex cover problems, they are all in NP, and from the reductions presented above, it follows from the definition that independent set, clique, and vertex cover problems are NP-complete.

Questions

1. What will be the consequence of the following:
   Sorting $\leq p$ 3-SAT
   3-SAT $\leq p$ Sorting

Acknowledgements: Lecture contents presented in this module and subsequent modules are based on the text books mentioned at the reference and most importantly, author has greatly learnt from lectures by algorithm exponents affiliated to IIT Madras/IMSc; Prof C. Pandu Rangan, Prof N.S.Narayanaswamy, Prof Venkatesh Raman, and Prof Anurag Mittal. Author sincerely acknowledges all of them. Special thanks to Teaching Assistants Mr.Renjith.P and Ms.Dhanalakshmi.S for their sincere and dedicated effort and making this scribe possible. Author has benefited a lot by teaching this course to senior undergraduate students and junior undergraduate students who have also contributed to this scribe in many ways. Author sincerely thanks all of them.

References:
1. E.Horowitz, S.Sahni, S.Rajasekaran, Fundamentals of Computer Algorithms, Galgotia Publications.
2. T.H. Cormen, C.E. Leiserson, R.L.Rivest, C.Stein, Introduction to Algorithms, PHI.
3. Sara Baase, A.V.Gelder, Computer Algorithms, Pearson.