Andrew Hayes
21321503
a.hayes18@nuigalway.ie

CT2109 Assignment 4

2023-12-08

P, NP, NP-Hard, & NP-Complete Problems

1 Polynomial Problems

1.1 Definition

We say that a problem is **polynomial** if there exists a deterministic algorithm that runs in polynomial time which can compute the answer to the problem.[1] This means that the worst-case scenario for the algorithm can be represented by a polynomial expression that takes the size of the input for the algorithm (such as the number of elements in an array, for example) as a variable. This is denoted as $T(n) = O(n^k)$, where k represents a positive constant.[2] By "worst-case" we mean the situation that takes the greatest possible number of steps to compute, and thus (theoretically) the longest time possible. This worst-case is an upper bound on "how bad" the performance of the algorithm can be.

"Deterministic" means that, given the same input, the algorithm will always will always go through the same steps to produce the result. Deterministic algorithms can be contrasted with non-deterministic algorithms, which do not behave in this manner.[3]

Problems that can be solved in polynomial time are considered to be "tractable", meaning that they can be solved (relatively) quickly and with (relative) efficiency.[4]

1.2 Example

An example of a problem in P is sorting the elements of an array, e.g. integers, in some order, e.g., smallest to largest. Depending on the type of data in your array (random, natural) and the size of your array, there are a variety of sorting algorithms to choose from. A commonly-used optimal algorithm that is used for this problem is **Merge Sort**, which is highly efficient, provided that you are not concerned about space complexity. Merge sort is of $O(n\log n)$ time complexity, which means that it runs in polynomial time, as the worst-case scenario for the algorithm can be represented by a polynomial expression that takes the size of the input for the algorithm as a variable n. Because the problem of sorting an array can be solved by an algorithm that runs in polynomial time, we can say that the problem itself is in P.

2 Non-Deterministic Polynomial Problems

2.1 Definition

Non-deterministic polynomial problems are problems for which there is no known way to compute the answer "easily", i.e., using a deterministic polynomial algorithm.[5] The answers to non-deterministic polynomial problems must be computed using a non-deterministic algorithm, but can be checked for accuracy using a deterministic polynomial algorithm.[6] This means that, given the same input, the algorithm will not always go through the same steps to produce the result. This is because the algorithm will consist of two repeating steps. Firstly, a candidate answer is generated using some non-deterministic process, possibly random. This candidate answer is then verified using a deterministic polynomial algorithm. This repeats until the correct answer is found.[1,6]

In short, NP problems are problems whose answers can be checked for accuracy using a deterministic algorithm in polynomial time, but must be generated using a non-deterministic polynomial algorithm. Problems in NP are referred to as intractable meaning that they cannot be solved quickly or efficiently.[1] Problems in P are commonly referred to as "easy" and problems in NP are commonly referred to as "hard".[7]

2.2 Example

An example of a problem in NP is the **Travelling Salesman** problem. The Travelling Salesman problem looks for the shortest path that visits each of several points, commonly imagined as addresses in a town that must be visited by the eponymous "Travelling Salesman". The salesman wants to use the shortest possible path

that brings him to each address once and then back to the starting address (his own home). It is commonly modelled as a graph theory optimisation problem, with the addresses being represented as nodes, and the paths between each address being represented as edges. The main issue is that, to know the shortest path between two points, the salesman must traverse each path to determine which is the shortest. Because of this, the solution time for this problem grows exponentially with the number of addresses (n), which means that the problem is in NP, because it cannot be solved using a deterministic polynomial-time algorithm.[8]

3 NP-Hard Problems

3.1 Definition

An **NP-Hard** problem is any problem that is considered to be "at least as computationally difficult" as any problem in NP. A problem X is considered to be NP-hard if every problem in NP can be reduced to to X in polynomial time.[1,9]

NP-hard problems are significant because discovering a polynomial-time algorithm capable of solving at least one of any of the NP-hard problems would be the same thing as discovering a polynomial-time algorithm for every problem in NP, as any problem in NP can be reduced to any NP-hard problem in polynomial time. As of yet, no such algorithm has been discovered.[10]

3.2 Example

An example of an NP-hard problem is the **Halting Problem**. The Halting Problem is the problem of determining, given a program & an input to said program, whether the program will *halt* (terminate) or continue indefinitely. The problem was proven by Alan Turing in 1936 to be *undecidable*, meaning that there is no general algorithm that can compute the answer for all possible programs & their respective inputs.[1,11]

If we take any problem in NP, we know that there must be a polynomial-time algorithm that can check a potential solution to the problem for correctness. We could then construct a Turing machine which runs this algorithm to check whether a given input is a correct solution to the problem, where the machine will halt upon verifying the solution and continue to run indefinitely if it does not verify the solution. If a solution to the Halting Problem was found, we could then predict whether this machine would halt or run indefinitely for any input.

If the program will halt, we know that the input is a valid solution to the problem in NP. If the program will fail to halt, we know that the input is not a valid solution. This would mean that we had found a polynomial-time solution to the problem. This shows that any problem in NP can be reduced to the Halting Problem, which is why it is considered to be NP-hard.[9]

Because the Halting Problem is undecidable, there is no algorithm that can solve it for all possible programs & inputs, and therefore there is still no known algorithm that can solve all problems in NP efficiently.[11]

4 NP-Complete Problems

4.1 Definition

A problem X in NP is **NP-Complete** if every other problem in NP can be reduced to X in polynomial time and the answer to the problem X can be verified in polynomial time.[12] NP-complete problems facilitate the reduction of all problems in NP to an NP-hard problem, mentioned above in "NP-Hard Problems: Definition". It is believed that NP-complete problems cannot be solved in polynomial time. However, if an algorithm was discovered that was capable of solving an NP-complete problem in polynomial time, then it could be used to solve all problems in NP in polynomial time.

The discovery of a polynomial-time algorithm would prove that $P = NP$, which is the topic of discussion in the following section.

4.2 Example

A classic example of an NP-complete problem is the **Boolean Satisfiability Problem**, which is the problem of determining whether or not the variables of a given Boolean formula can be replaced with Boolean values (TRUE, FALSE) in a such a manner that the formula is consistent & equal to TRUE.[13]

The Cook-Levin theorem shows that this problem is NP-complete. While the proof is too long to include here, it can be summed up as follows: Any given problem in NP can be reduced to the Boolean Satisfiability Problem using a technique called Cook's Reduction, which works by constructing a Boolean formula that is equivalent to the input of the problem being reduced, an operation which can be performed in polynomial time. This constructed formula is then transformed into an equivalent Boolean Satisfiability Problem formula via a series of logical equivalences, an operation which can also be performed in polynomial time.[13]

Since this reduction can be performed in polynomial time, any given problem in NP can be reduced via this method to the Boolean Satisfiability Problem in polynomial time, proving that it is indeed NP-complete.

5 P versus NP Problem

The P versus NP, commonly abbreviated to $P = ?NP$ is the question of whether or not the set P of all deterministic polynomial problems is the same as the set NP of all non-deterministic polynomial problems, or, alternatively, it is the question of whether or not every problem in NP is also in P.[14] It is widely believed that $P \neq NP$.[15]

This problem is of massive importance to the fields of Computer Science & Mathematics, and is one of the Millennium Prize Problems.[14] The question of whether or not P is equal to NP was first put forward by Stephen Cook in 1971.[13] Since then, due to its enormous importance, it has been studied & researched at great lengths.[16]

One hugely important field that depends on the assumption (or hope) that $P \neq NP$ is modern public-key cryptography, which depends on the fact that verifying the product of two very large prime numbers is a problem that is in P, but the factorising of very large prime numbers is in NP, and would take a prohibitively long time to compute. If it transpired that P was in fact equal to NP, then the factorisation of large numbers could be done using a deterministic, polynomial time algorithm, which would effectively destroy almost all modern public-key cryptosystems.[14]

References

[1] Frank Glavin. *Topic Three: Algorithm Analysis & Dynamic Programming Part II*. Uploaded to Blackboard: 2023-02-17.

[2] David Terr. *Polynomial Time*. URL: https://mathworld.wolfram.com/PolynomialTime.html. (Accessed: 2023-03-03).

[3] Robert W. Floyd. *Nondeterministic Algorithms*. 1967-10. DOI: 10.1145/321420.321422.

[4] William L. Hosch. *NP-complete problem*. 2023-03-10. URL: https://www.britannica.com/science/NP-complete-problem#ref97458. (Accessed: 2023-03-03).

[5] Lance Fortnow. *The status of the P versus NP problem*. 2009. DOI: 10.1145/1562164.1562186.

[6] *Non-Deterministic Polynomial Time (NP)*. 2019-08-29. URL: https://www.techopedia.com/definition/21028/non-deterministic-polynomial-time-np. (Accessed: 2023-03-03).

[7] William L. Hosch. *P versus NP problem*. 2023-02-22. URL: https://www.britannica.com/science/P-versus-NP-problem. (Accessed: 2023-03-03).

[8] Stephan C. Carlson. *Travelling salesman Problem*. 2023-02-05. URL: https://www.britannica.com/science/traveling-salesman-problem. (Accessed: 2023-03-03).

[9] Eric W. Weisstein. *NP-Hard Problem*. URL: https://mathworld.wolfram.com/NP-HardProblem.html. (Accessed: 2023-03-03).

[10] Michael R. Garey & David S. Johnson. *Computers & Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979. ISBN: 0-7167-1045-5.

[11] Karleigh Moore et al. *Halting Problem*. URL: https://brilliant.org/wiki/halting-problem/. (Accessed: 2023-03-03).

[12] Erik Gregersen. *NP-Complete Problem*. 2023-03-10. URL: https://www.britannica.com/science/traveling-salesman-problem. (Accessed: 2023-03-12).

[13] Stephen Cook. *The complexity of theorem-proving procedures*. 1971. DOI: 10.1145/800157.805047.

[14] William L. Hosch. *P versus NP problem*. 2023-02-22. URL: https://www.britannica.com/science/P-versus-NP-problem. (Accessed: 2023-03-03).

[15] William I. Gasarch. *The P=?NP poll*. 2002-06. DOI: 10.1145/564585.564599.

[16] Michael Sipser. *The history & status of the P versus NP question*. 1992-07. DOI: 10.1145/129712.129771.