Maxwell Maia 21236277

Assignment 4
P, N, NP, NP-Hard and NP-Complete Problem

Problems can be solved by writing an algorithm for them. Problems may solvable but not yet solved in polynomial time by an algorithm – we may have not discovered the algorithm yet. This topic deals with a way of classifying problems to aid with research to help find ways to be able to computationally answer decision problems or speed up the time required for a computer to solve these problems.

1.

Polynomial Problems

Definition

A problem that is solvable by a deterministic algorithm in "polynomial time" is a polynomial problem. An algorithm is deterministic when the input given to the algorithm results in the same output every time it is run with that input (Tutorials Point). An algorithm is solvable in "polynomial time" if the number of time steps required to perform the algorithm is of the time complexity $O(n^k)$. n is the length of the input of the problem and k is a non-negative integer (Weisstein). In other words, the number of steps (timesteps) in the algorithm is bounded by a polynomial function (Britannica). The time required to solve the problem grows no faster than a polynomial function of the input size. Some polynomial function examples are:

1. $n^2 + 6n + 21$	=> $O(n^2)$
2. n log n	=> O(n log n)
3. n^{100}	=> $O(n^{100})$

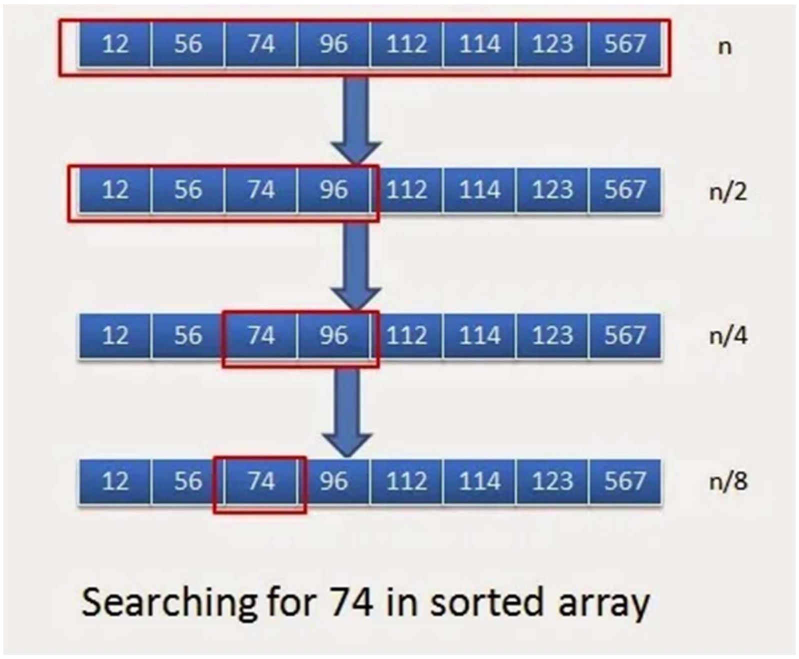
Polynomial problems are considered to be tractable. They can be solved efficiently for large input sizes.

Example

Binary search is polynomial problem. It has a time complexity of O(log n), where n is the size of the input array. The input data has to be in sorted order.

The algorithm divides the array in half repeatedly until the target is found. On each iteration of the algorithm, the middle element of the current interval is compared with the target value. If the target value is greater than the middle element: a new interval is set so as to remove the bottom half of the search space. If instead the target value is less than the middle element: a new interval is set so as to remove the bottom half of the search space. This eliminates half of the remaining possibilities at each step. (Geeks for Geeks).

See the diagram below showing the process of finding 74 in the array of elements. Each array you see is another recursive step all on the same array. The current interval is highlighted in red. As the iterations go by, the search space halves. This gives the log n nature of the time complexity.



"BinarySeasrch.webp" by Danish Ali.
Source: https://masterprograming.com/binary-search-in-hindi/

Binary search is considered a polynomial problem because it’s complexity is of the function O(log n). O(log n) grows slower than O(n). In a worst case scenario – when the target is not in the array – the algorithm has to check log n elements. A linear search would have to check n elements. Log n is more efficient.

2.

Non-deterministic Polynomial Problems

Definition

Non-deterministic polynomial problems are a class of computational problems in computer science.

A non-deterministic polynomial problem is a problem for which a solution can be verified in polynomial time but a solution cannot be found in polynomial time using an algorithm that always gives the correct answer. That is to say they cannot be solved deterministically in polynomial time but they can be verified deterministically in polynomial time.

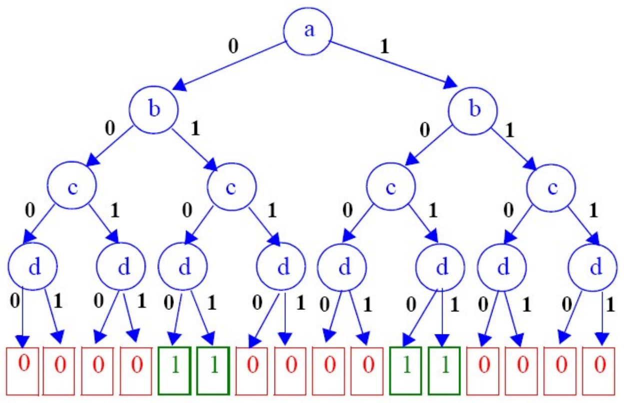
These problems can be solved in polynomial time by a non-deterministic Turing machine. A non-deterministic Turing machine is a model of computation where the next state is not fully determined by its action and the current symbol it sees. (Wikipedia).

Non-deterministic polynomial problems cannot be computed by our classical computers because our computers are based in deterministic operations. If it was non-deterministic then somehow the computer would be able to test multiple possibilities simultaneously to find the answer to an operation. (Engati). Classical computers cannot do this but they can test multiple possibilities one after the other to find the answer to an operation. This is a brute-force search. Brute force search is one way to solve non-deterministic problems. Another way is to use heuristics which are algorithms that make intelligent guesses bases on some prior knowledge. (Geeks for Geeks).

Example

The Boolean satisfiability problem is a decision problem that falls under the category of non-deterministic polynomial problems. It is the problem of determining whether there exists an assignment of Boolean values to variables in a given Boolean formula that will make the whole formula evaluate to true. Given a logical formula using Booleans and logical operators, is there a particular set of values of the Booleans that make the formula true? Determining this yes or no answer is the basis of the satisfiability problem. (Abdul Bari).

$$f(a,b,c,d) = \begin{array}{l}(a \vee b \vee c)\cdot(a \vee b \vee \bar{c})\cdot(\bar{a} \vee c \vee d)\\ (\bar{a} \vee c \vee \bar{d})\cdot(\bar{b} \vee \bar{c} \vee d)\cdot(\bar{b} \vee \bar{c} \vee d)\end{array}$$



"An example of a satisfiable SAT instance showing its corresponding decision tree" by Ali Orooji and Yashar Ganjali is licensed under CC BY-NC-SA 4.0.
Source: https://www.researchgate.net/figure/An-example-of-a-satisfiable-SAT-instance-showing-its-corresponding-decision-tree_fig1_45267434

This figure shows a formula with 4 Booleans and all the combinations of their values in the formula. The paths that lead to a true are marked with green. We can see in this example that the answer of the yes or no question is yes. There is a set of values that make the formula true.

The Boolean satisfiability problem is a non-deterministic polynomial problem.

A non-deterministic Turing machine can solve the Boolean satisfiability problem in polynomial time. A non-deterministic Turing machine is a theoretical model of computation that tries all possible paths of computing a problem simultaneously. (Wikipedia). Each Boolean can either be true or false. This leads to multiple paths of computation which all need to be checked to see whether the formula evaluates to true. The number of paths to check increases exponentially when increasing the amount of Boolean variables in the formula, thus the time complexity is $O(2^n)$ if solved deterministically. (Abdul Bari). So solving this problem in polynomial time would require computing multiple paths simultaneously, therefore being solved by a non-deterministic Turing machine in polynomial time.

For the Boolean satisfiability problem, we can check a solution in polynomial time. To check a solution we just need to evaluate the formula using the proposed Boolean variables that the non-deterministic algorithm calculated and check whether the result is true. The evaluation of a Boolean formula takes polynomial time.

Since any given solution can be verified in polynomial time and a non-deterministic Turing machine can solve the Boolean satisfiability problem in polynomial time, the Boolean satisfiability problem is a non-deterministic polynomial problem.

3.

NP-Hard Problems

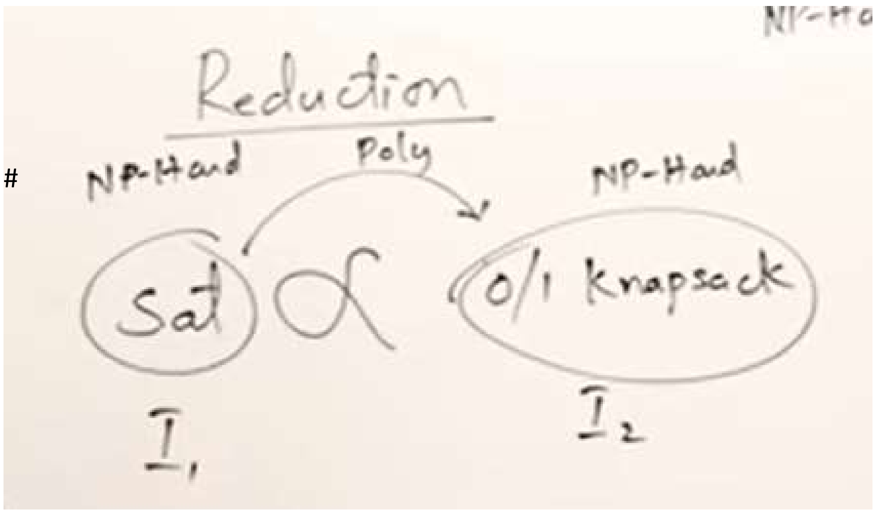
Definition

Problem A is considered an NP-Hard problem if the algorithm that solves problem A can be translated (by any reduction algorithm of polynomial time) into the algorithm that solves another problem which has already been classified as an NP-Hard problem (Weisstein). The base problem of NP-Hard problems is the Satisfiability problem. The satisfiability problem is solved by an algorithm that takes exponential time. (Adbul Bari).

Any problem that can be converted to the satisfiability problem by reducing it with an algorithm taking at most polynomial time, is “at least as hard as the satisfiability problem”. Thus, since satisfiability is an NP-Hard problem, and a problem reduced to satisfiability is at least as hard as the satisfiability problem, then that problem is an NP-Hard problem too. This reduction works in both directions. (Abdul Bari).

Also, any problem than can be converted to any NP-Hard problem by a polynomial time reduction is considered an NP-Hard Problem because it would be “at least as hard as any NP-Hard problem”.

The following diagram shows that the satisfiability can be reduced to the 0/1 knapsack problem using a polynomial time reduction algorithm. Since the satisfiability problem is NP-Hard, the 0/1 knapsack problem is therefore NP-Hard too.

#


(Abdul Bari)

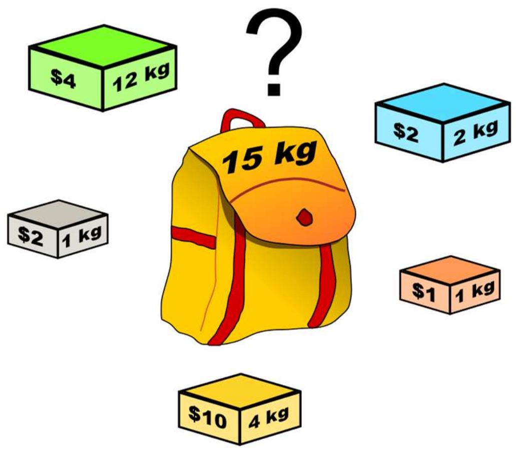
If a deterministic solution in polynomial time is found for ANY NP-Hard problem then all other NP-Hard problems will have a known deterministic solution in polynomial time too, since NP-Hard problems can be reduced to each other using a polynomial time algorithm. (Abdul Bari). This is the power behind this classification system: if you solve one of them, they all get solved.

For NP-Hard problems, there is no known algorithm that can solve the problem in polynomial time. Often times there are heuristic and approximation algorithms that can find good solutions to NP-Hard problems in polynomial time.

Example

The Knapsack problem

Which items should you choose to maximise the amount of money in the knapsack without going over the 15kg limit?



While the problem may seem simple, it quickly becomes difficult as the number of items increases.

We have to consider all combinations of picking the boxes and then choose the one with the highest value. The number of possible combinations of picking n boxes it 2^n. Therefore, the brute force of approach has the time complexity of $O(n^2)$.

Using dynamic programming the approach has a time complexity of O(n*W), where n is the number of items and W is the capacity of the knapsack. (Wikipedia).

The knapsack problem is a NP-Hard because the Satisfiability problem can be reduced to the Knapsack problem using a polynomial time algorithm. (Abdul Bari).

If the there are multiple of each box available, then the solution to choose is 3 yellow boxes and 3 grey boxes. If there is only 1 of each box, then the solution is to choose all boxes except for the green box.

The knapsack problem has many real-world applications, such as in resource allocation, production planning, and finance.

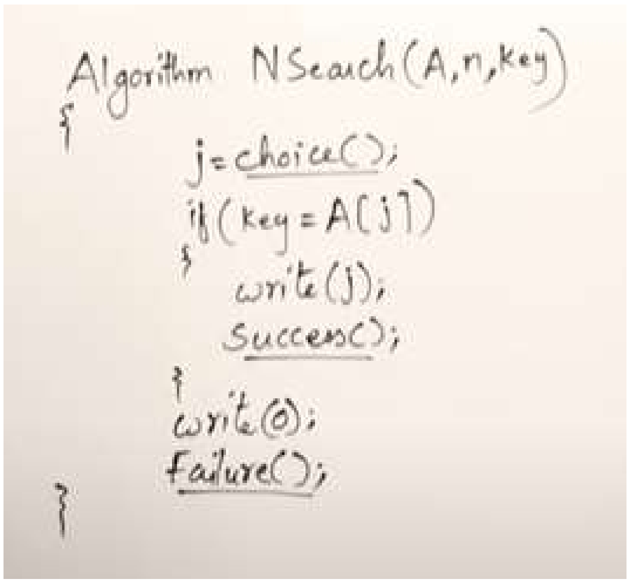
4.

NP-Complete Problems

Definition

NP-Complete problems are problems that we haven't yet found an efficient solution algorithm for (intractable). NP-Complete problems that need to be solved quickly are handled by using a polynomial time algorithm that approximates the solution. (Britannica).

NP-Complete problems, are NP-Hard problems that have a nondeterministic polynomial time algorithm written for them. To show what a nondeterministic algorithm looks like see the following nondeterministic pseudocode of a searching algorithm that searches a key from an array of size n:



```
Algorithm NSearch(A,n,key)
{
        j=choice();
        if(key=A[j])
        {
            write(j);
            Success();
        }
        write(0);
        Failure();
}
```

(Abdul Bari)

How does the choice() method know that the key element is in index j? We have code that can solve this for us, but not in constant time. This part of the algorithm is nondeterministic as we have not found a way of writing the code for choice() that finds index j in constant time. Once we find that code, then the whole searching algorithm can be solved in constant time. And the whole algorithm will become deterministic.

Using this manner of writing a nondeterministic algorithm we can write a non-deterministic algorithm for a NP-Hard problem that would solve the problem in polynomial time. Once this nondeterministic polynomial time algorithm is written, the problem is NP-Complete. (Abdul Bari).

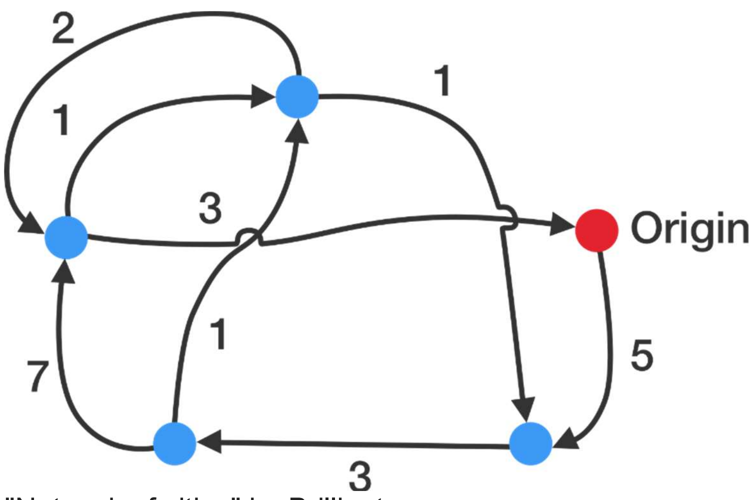
The ability to solve all other problems in the NP-Complete class once a solution for one NP-Complete problem has been found is present and is true just like it is for NP-Hard problems. (Britannica)

We do not know whether any polynomial-time algorithms will be found for NP-complete problems. Determining whether or not these problems are tractable or intractable is highly valuable as solving them would save computers lots of time. (Britannica). Since computing power costs tech companies lots of money a solution to these algorithms would be highly valuable.

Example

A NP-Complete problem is a NP-Hard problem that has a non-deterministic polynomial algorithm written for it.

The travelling salesman problem is NP-Complete. The travelling salesman problem attempts to find the shortest possible route that visits a set of given cities and also returns to the starting city. Given a set of cities, a distance between each pair of cities, what is the shorted path that the salesman should take to visit each city exactly once and returns to the starting city? (Brilliant).



"Network of cities" by Brilliant.
Source: https://brilliant.org/wiki/traveling-salesperson-problem/

The solution to the problem is to try every single path and pick the fastest.

There are (n-1)! Paths that the travelling salesman could take. This problem becomes hard to solve in a brute-force way quickly. If there were 20 cities, then the number of possible paths would be 19! which is $1,22*10^{17}$ paths. (Brilliant). The problem is intractable for a large number of cities.

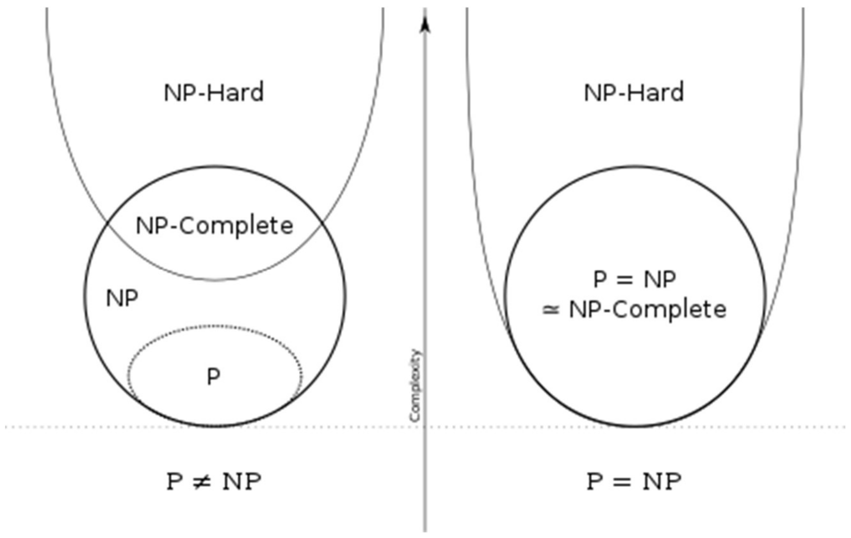
Using dynamic programming where the time complexity is: O(n!). (Geeks for Geeks). This is not polynomial. This algorithm has a non-deterministic polynomial algorithm and is NP-Hard, it is considered NP-Complete.

5.

<u>P versus NP</u>

<u>Definition</u>

Are all problems that can be verified in polynomial time also able to be solved in polynomial time? P is a set of decision problems that can be solved by a deterministic algorithm in polynomial time. NP is the set of decision problems that an answer of “yes” can be verified in polynomial time by a deterministic algorithm. P versus P aims to answer the question: is the set P, equal to the set NP? (Wikipedia).



"P np np-complete np-hard.svg" by Abiomedic is licensed under CC BY-SA 3.0.
Source: https://commons.wikimedia.org/wiki/File:P_np_np-complete_np-hard.svg

If P is equal to NP: all non-deterministic polynomial problems have a polynomial time algorithm that can solve them. (Abdul Bari).

If P is not equal to NP: there are non-deterministic problems that do not have polynomial time algorithms. (Abdul Bari).

History

Computer scientists and even mathematicians have been trying to solve the theoretical question: is P equal to NP, since the 1970's. Stephan Cook is a Canadian mathematician who proved that the Boolean satisfiability problem is NP-complete in 1971. (Wikipedia).

Richard Karp proved that 21 problems, including the travelling salesman problem and the knapsack problem are NP-complete.

Once a problem is categorized as NP complete, finding a deterministic algorithm that solves the problem in polynomial time would help prove that P is equal to NP. However this has not happened yet.

There has been much research over the past few decades yet the P versus NP problem is still not solved.

Bibliography

Wikipedia contributors. (2023, March 7). Nondeterministic Turing machine. In Wikipedia, The Free Encyclopedia. Retrieved 14:18, March 13, 2023, from https://en.wikipedia.org/wiki/Nondeterministic_Turing_machine

Tutorials Point. (n.d.). Difference between Deterministic and Non-Deterministic Algorithms. Retrieved March 13, 2023, from https://www.tutorialspoint.com/difference-between-deterministic-and-non-deterministic-algorithms

GeeksforGeeks. (n.d.). Binary Search. Retrieved March 13, 2023, from https://www.geeksforgeeks.org/binary-search/

Engati. (n.d.). Non-deterministic Algorithm. Retrieved March 13, 2023, from https://www.engati.com/glossary/non-deterministic-algorithm

GeeksforGeeks. (n.d.). Brute-Force Approach and Its Pros and Cons. Retrieved March 13, 2023, from https://www.geeksforgeeks.org/brute-force-approach-and-its-pros-and-cons/

Weisstein, E.W. (n.d.). Polynomial Time. In MathWorld. Retrieved March 13, 2023, from https://mathworld.wolfram.com/PolynomialTime.html

Weisstein, E.W. (n.d.). NP-Hard Problem. In MathWorld. Retrieved March 13, 2023, from https://mathworld.wolfram.com/NP-HardProblem.html

Wikipedia contributors. (2023, March 10). Knapsack problem. In Wikipedia, The Free Encyclopedia. Retrieved 14:33, March 13, 2023, from https://en.wikipedia.org/wiki/Knapsack_problem

Encyclopedia Britannica. (n.d.). P versus NP problem. Retrieved March 13, 2023, from https://www.britannica.com/science/P-versus-NP-problem#ref1119246

Encyclopedia Britannica. (n.d.). NP-complete problem. Retrieved March 13, 2023, from https://www.britannica.com/science/NP-complete-problem

Abdul Bari. (2018, February 18). 8. NP-Hard and NP-Complete Problems. [Video]. YouTube. https://www.youtube.com/watch?v=e2cF8a5aAhE&ab_channel=AbdulBari

Wikipedia contributors. (2023, March 7). NP-completeness. In Wikipedia, The Free Encyclopedia. Retrieved 14:50, March 13, 2023, from https://en.wikipedia.org/wiki/NP-completeness

Brilliant. (n.d.). Traveling Salesperson Problem. Retrieved March 13, 2023, from https://brilliant.org/wiki/traveling-salesperson-problem/

GeeksforGeeks. (n.d.). Travelling Salesman Problem using Dynamic Programming. Retrieved March 13, 2023, from https://www.geeksforgeeks.org/travelling-salesman-problem-using-dynamic-programming/

Wikipedia contributors. (2023, March 7). Cook–Levin theorem. In Wikipedia, The Free Encyclopedia. Retrieved 14:50, March 13, 2023, from https://en.wikipedia.org/wiki/P_versus_NP_problem

Wikipedia contributors. (2023, March 7). Cook–Levin theorem. In Wikipedia, The Free Encyclopedia. Retrieved 14:50, March 13, 2023, from https://en.wikipedia.org/wiki/Cook%E2%80%93Levin_theorem

Wikipedia contributors. (2023, March 7). Karp's 21 NP-complete problems. In Wikipedia, The Free Encyclopedia. Retrieved 14:50, March 13, 2023, from https://en.wikipedia.org/wiki/Karp%27s_21_NP-complete_problems