NP problem

Monday, October 7, 2019 00:35

For the glory of God

Introduction

- In the class, we have discussed about problems that can be solved by either Greedy algorithm or Dynamic programming.
- We haven't yet made any attempt to actually quantify or characterize the range of problems that can't be solved efficiently.
  - ↳ These problems could be referred as NP, NP hard, and NP complete problems.
- In practice, there are literally thousands of NP-related problems arising in numerous areas.
- Therefore, we should know the NP problem stuffs and this hand-written note will explain it.

In computer view,

- Polynomial : Efficient
- Exponential : Inefficient

Decision problem

- In complexity theory, they are typically dealing with 'Decision problems' to see if the algorithm is efficient or not.
  - ↳ They are not interested in developing algorithms but focusing more on analyzing the complexity of algorithm.

Given an input and a question regarding a problem, determine if the answer is yes or no.

- You may be wondering what is the significance of decision problem?
  - First, it is a simple way to compare algorithms in a complexity manner.
  - Second, non-decision problems can also be transferred to a decision problem.

e.g. Decision problem format of TSP

: Given a complete graph, is there a cycle going through each vertex once with $\sum w(e) \leq k$?

↳ The answer should be either Yes or No.

P vs. NP problem

- Based on the fact that we are now only talking about decision problems, let us figure out P vs. NP problem.
  - ↳ One of the seven Millennium Prize Problems
- In general, P problems can be defined as;
  - ↳ Problems that are solvable in Polynomial time, i.e. running time is in $O(n^2)$
- On the other hand, NP problems are defined as; cf. NP stands for Non-deterministic Polynomial
  - ↳ Problems for which a candidate solution can be verified in polynomial time.
    - ↳ As noticed, we're not talking about solving problems but checking them.

↳ Here, recall that NP is intractable problems to solve. (Because we haven't found the algorithm suitable in polynomial time.)

↳ for example, let's say there is a problem that can't be solved but checked all possibilities in a polynomial time.

(By checking all possibilities, we may be able to realize that the selected one is actually the solution)

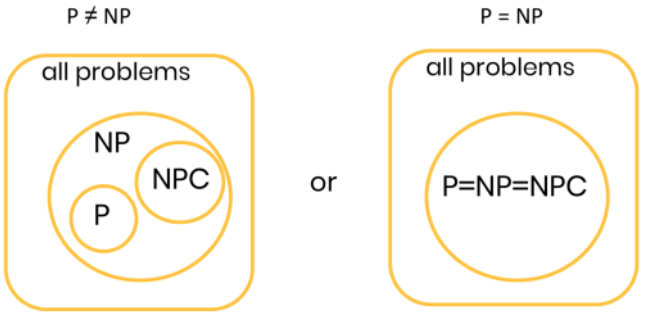
· Then, this question naturally arises ;

DOES P = NP ?

· The question of whether P = NP is fundamental in the area of algorithms and it's one of the most famous problems in computer science.

- The general belief is that P ≠ NP.
- However, there is not a lot of hard technical evidence for it.
- It is more based on the sense that P = NP would be too amazing to be true.

· A huge amount of effort has gone into failed attempt ; thus, many people can't answer the question but leave it as below ;



The definition of NP, NP-hard, and NP-complete

· As a reminder, we deal with problems that can't be solved efficiently.

· Also, we define the NP problems as the one where checking verification is running in polynomial time.

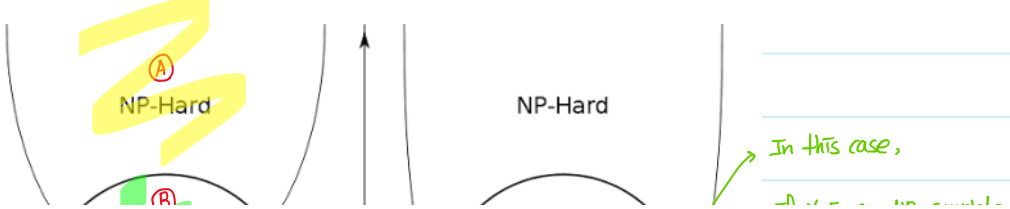
↳ Here, note that we can't solve the problems (In other words, there is no algorithm to handle the problem in polynomial time)

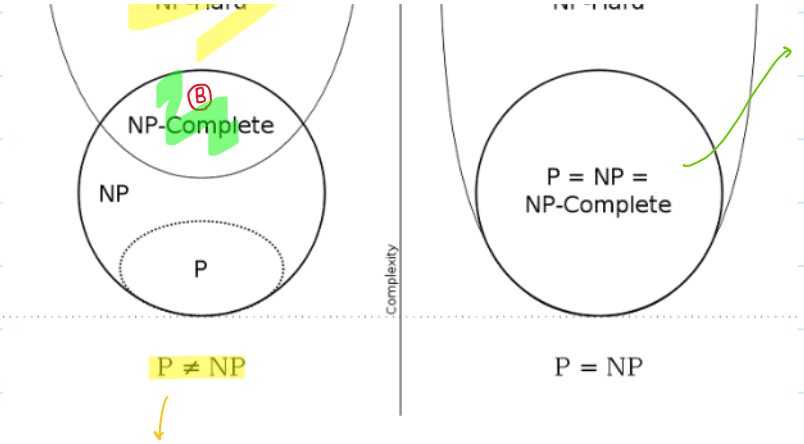
· People then have turned to a related but more approachable question ;

- What are the hardest problems in NP ?
- How should we formalize whether a problem can be solved (or checked) efficiently ?

↳ Basically, it might require us to classify / categorize the problems.

· Finally, the following figure arises from the scientists ;





In this case,

If X is an NP-complete,

X is solvable in polynomial time

(when P = NP)

Let us drive into this case because it's the general belief among scientists.

a) Class P

· Class P consists of decision problems that are solvable in polynomial time.

b) Class NP

· Class NP is the class of problems for which a candidate solution can be verified in polynomial time.

c) Class NP-hard

· This is the defining property of a class of problems that are informally 'At least as hard as the hardest problems'

· I know this definition is somehow ambigue. Let us get a little bit more details.

- We define the NP problems that can be verified in Polynomial time.

- First, (A) region is the NP-hard where we don't know if the problem can be verified in polynomial time.

  ↳ In other words, NP-hard problems do not have to be in NP.

- Second, (B) region is the NP-hard where we do know that it's NP-complete problem but we don't know if the problem is still able to be checked in Polynomial. ↳ we'll see in a second.

d) Class NP-complete

You see the difference

(B) is the hardest one and it's not in NP.

· Class NP-complete is a class of the hardest problems in NP class.

· In other words, if you can solve an NP-complete problem, then you can solve all NP problems.

· Someone already demonstrated that it's impossible to solve the NP-complete problems in polynomial time.

↳ Recall that { NP class : we haven't found out an algorithm that solves the problem in polynomial time.
NP-complete : we confirm that there is no algorithm to solve the problem in polynomial time.

Polynomial time reduction

· Recall that the NP-hard problem is defined as ;

- It is informally 'At least as hard as the hardest problems'

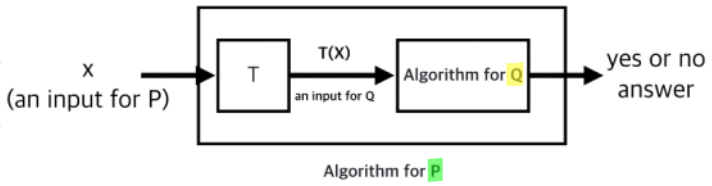
· We are going to formalize this through the notion of Reduction.

- Remember that we are comparing the relative difficulty of different problems.

- Suppose Q can be solved in polynomial time directly using the blackbox.

↳ Let's say it will return the correct answer.

e.g. All pairs shortest path : Multiple calls to Dijkstra



- In this example, we could say that :

↳ P is polynomial time reducible to Q.

↳ Q is at least as hard as P with respect to polynomial time.

↳ Problem P is easier than problem Q.

↳ By notation, $P \leq_p Q$

↳ It represents 'Polynomial reducible'

* cf.

If $X \leq_p Y$ and $Y \leq_p X$, $X \equiv_p Y$

If $X \leq_p Y$ and $Y \leq_p Z$, $X \leq_p Z$

· Therefore, if $Y \leq_p X$, then

If X can be solved in polynomial time, then Y can be solved in polynomial time.

If Y can't solved in polynomial time, then X can't be solved in polynomial time. (contrapositive)

Final notes

· NP problems really come up all the time.

· Since we can't solve them for many reasons discussed so far, we would use alternative ways to solve them in reality.

- Restrict the problem : Find the special case solvable in polynomial time

- Use a heuristic : Solve the problem reasonably

- Solve approximately : Close to the optimum

- Use an exponential time solution : if you really have to solve the problem

· In reality, Since we don't know whether a problem can be solved in polynomial time, the reduction is used to establish relative levels of difficulty among problems.

· In order for us to deal with computational intractability, we would use the following ingredients :

Reduction

§ Reduction

Characterization of the class of problems