Harsha Dindigal
Hd4ka
10/23/15

CS 2110
Homework 3: Time Complexity
Due date: Friday, October 23, 2015

LEARNING OBJECTIVES:
- Describe the NP-completeness (P=NP?) problem
- Identify and describe the various problem classes (w.r.t. NP-completeness)
- Explain, and provide an example of, the negative impact of having a problem solvable by an exponential-time algorithm
- Explain, and provide an example of, the positive impact of having a problem solvable by an exponential-time algorithm

GRADING:
- A maximum of **100 points** can be obtained on this homework assignment.

SUBMITTING:
- Submit on Collab
- Submit **1 PDF** document as your homework *(if you don't know how, please ask!)*
- You may work **individually** or **in pairs** on this homework (more than two is not allowed)
- Your submitted homework must be typed
- At the top of your document be sure to include your name and computing ID
- If you choose to work with a partner, ensure both names and computing IDs are written on the submitted assignment
- The submission deadline is **11:30pm** on the date the assignment is due, mentioned above

Q1) [80 points] – every blank is worth *2 points*; except *4 points* for Q2 (2^{nd} blank), Q15 (1^{st} blank), and Q19.

The topic of this question is on NP-completeness. Complete these sentences by filling in the gaps with a word, a few words, an equation, or symbol as appropriate. Ensure that your answers are underlined and in another color other than black to facilitate grading this portion of the assignment.

Example:

Question: The topic of this section is on ________________.

Answer: The topic of this section is on NP-completeness.

1. **Intractable**, or *hard*, problems are unsolvable in a reasonable amount of time as ***n*** (the size of the input) gets large.
2. Typically, "reasonable time" is defined as an algorithm whose complexity is on an order of **polynomial** time. In other words, for an input size n the running time is on an order of $\mathbf{n^k}$ for some constant k.
3. **Tractable**, or *easy*, problems are solvable by **polynomial**-time algorithms.
4. **P** is the class of problems that have a **polynomial**-time solution ("solvable" in **polynomial** time).
5. Problems that are **undecidable** cannot be solved by a computer.
6. Many problems are decidable but **intractable**; that is, as the size of input ***n*** grows large the solution is unsolvable in a reasonable amount of time.
7. **NP** is the class of problems that are **verified** in polynomial time. That is, if given a "certificate" of a solution then it is possible to verify that the certificate is correct in polynomial time (based on input size ***n***).
8. Any problem in P is also in NP, since if a problem is in P then we can solve it in polynomial time without even being supplied a certificate. Therefore we can say

   P **is a subset of** NP.
9. The biggest open problem in CS is determining whether or not P is a **proper** subset of NP, that is, we need to answer the question is P = NP?
10. There are two kinds of problems: **decision** problems and **optimization** problems.

11. **NP-complete** problems are a class of problems whose status is **unknown**. No polynomial-time algorithm has been discovered for such problems, nor has anyone yet been able to prove that no polynomial-time algorithm can exist for any one of these problems.
12. **NP-complete** problems are in the NP class and are also the "hardest" problems to solve. It is important to note that if any *one* NP-complete problem can be solved in polynomial time, then *every* NP-complete problem has a **polynomial**-time algorithm. This would lead to the fact that *every* problem in NP can be solved in polynomial time (which would show P = NP!)
13. Theory on P, NP, etc is defined based on **decision** problems.
14. Revisiting previous definitions, P is a set of **decision** problems that can be solved in polynomial time.
15. NP (stands for **nondeterministic polynomial time**) is the set of **decision** problems that can be solved in polynomial time by a **nondeterministic** computer. (That is, solved by a polynomially bounded **nondeterministic** algorithm.)
    [Remember: "NP" does <u>not</u> mean "not polynomial"!!]
16. Since **NP-complete** problems are the harder problems to solve amongst the problems in the NP class, it follows that if *any* **NP-complete** problem can be solved in polynomial time, then *every* problem in NP has a polynomial-time algorithm (that solves it.)

17. Most theoretical computer scientists believe that the **NP-complete** problems are **probably intractable**. Therefore, if a problem can be established as NP-complete, there is no need to search for an efficient algorithm (because one doesn't exist). Instead, work on approximation algorithms that do run in polynomial time and produce near optimal results.
18. The crux of NP-completeness is reducibility.
19. If A is **polynomial-time reducible** to B, we denote this as $A \leq_p B$. It shows one problem is at least as hard as another.
20. Definition of NP-Hard and NP-Complete:

If all problems $X \in NP$ are reducible to A, then A is **NP-hard**

We say A is **NP-complete** if A is NP-Hard and $A \in NP$

21. Reductions may be a mechanism to prove problems do (or do not) belong to particular classes. Consider the following theoretical situation (given a polynomial-time reduction algorithm): If you can reduce a "hard" NP-complete problem, A, to another problem, B, that can be solved in polynomial time then the original problem, A, is, in a sense, "no harder to solve" than the other problem (and therefore solvable in **polynomial time**!)
22. To conclude, it is clear that research into the $P \neq NP$ question centers around the NP-complete problems. Most theoretical computer scientists believe that **P != NP** However, who knows when someone may come up with a polynomial-time algorithm for an NP-complete problem, thus proving that P = NP. Since no **polynomial-time** algorithm for ANY NP-complete problem has yet been discovered, a proof that a problem is NP-complete provides excellent evidence that it is **intractable**.

Q2) [20 points]

The topic of this question is on NP-completeness and algorithmic complexity.

(a) Describe why is it **problematic** to have an algorithmic solution to a problem that runs in exponential time. Give a specific example.

It is problematic to have an algorithmic solution to a problem that runs in exponential time because at a certain point, it will become intractable. As the solution is applied to higher and higher n-sizes, the run-time is exponentially rising to the point where it is no longer viable to everyday-time constraints people have. One example of this is the traveling salesman problem. At a certain point, when deciding to go between a certain number of cities, solutions take so long that it would be more beneficial just to change the nature of the problem/solution you're looking for.

(b) Describe why it might be **beneficial** to have an algorithmic solution to a problem that runs in exponential time. Give a specific example.
It might be beneficial to have an algorithmic solution to a problem that runs in exponential time because it might have more coverage than a polynomial-time solution. For example, algorithms that crack numerical passwords test out every possible password-combination. In this example, the solution algorithm is sure to check every possible combination ensuring a correct output.