

CS 432: Algorithms NP Completeness

Chapter 10 in Johnsonbaugh & Schaefer
Read and study! (This ain't simple.)

Slide credits: Thanks to David Luebke, Jim Cohoon

1

11/10/2005

NP-Completeness

- Some problems are *intractable*: as they grow large, we are unable to solve them in reasonable time
- What constitutes reasonable time? Standard working definition: *polynomial time*
 - On an input of size n the worst-case running time is $O(n^k)$ for some constant k
 - Polynomial time: $O(n^2)$, $O(n^3)$, $O(1)$, $O(n \lg n)$
 - Not in polynomial time: $O(2^n)$, $O(n^n)$, $O(n!)$

2

11/10/2005

Polynomial-Time Algorithms

- Are some problems solvable in polynomial time?
 - Of course: most every algorithm we've studied provides polynomial-time solution to some problem
 - We define **P** to be the class of problems solvable in polynomial time
- Are all problems solvable in polynomial time?

3

11/10/2005

Tractability

- Again, some problems are *undecidable*: no computer can solve them
 - E.g., Turing's "Halting Problem"
 - We're not going to talk about such problems here (Take a theory class to learn more!)
- Other problems are decidable, but *intractable*: as they grow large, we are unable to solve them in reasonable time
 - *What constitutes "reasonable time"?*

4

11/10/2005

Flashback: Growth Rates

- Review pages 430-431 for this important point:
 - Say s is the size of the largest problem we can solve with algorithm A given time t
 - How much larger a problem can we solve if we have 10x as much time (or a computer that's 10x faster)?
- Depends on the complexity of A
 - Linear? $10 \times s$ times as large a problem
 - Quadratic? $\sqrt{10} = 3.2 \times s$ times as large
 - Cubic? $\sqrt[3]{10} = 2.2 \times s$ times as large
 - 2^n ?
 - $s + \lg 10$ as large, or $s + 3.2$ as large
 - Note additive factor, not multiplicative

5

11/10/2005

NP-Complete Problems

- The *NP-Complete* problems are an interesting class of problems whose status is unknown
 - No polynomial-time algorithm has been discovered for an NP-Complete problem
 - No suprapolynomial lower bound has been proved for any NP-Complete problem, either
- We call this the *P = NP question*
 - The biggest open problem in CS

6

11/10/2005

An NP-Complete Problem: Hamiltonian Cycles

- An example of an NP-Complete problem:
 - A *hamiltonian cycle* of an undirected graph is a simple cycle that contains every vertex
 - The hamiltonian-cycle problem: given a graph G , does it have a hamiltonian cycle?
 - Describe a naive algorithm for solving the hamiltonian-cycle problem. Running time?
 - Have we studied a search algorithm that can be used to solve this? Running time?

7

11/10/2005

Other Problems to Know

- Hamilton Path/Cycle, TSP
- Graph Coloring
- Vertex cover
- Satisfiability
- Subset Sum
- Bin Packing
- Knapsack

8

11/10/2005

Some Definitions Before We Proceed

- Decision problems
 - Simple view: Problem is to answer with a "yes" or "no" answer for a given input
- Definition, page 431:
 - A **problem** is a set of finite-length questions (strings) with associated finite-length answers (strings).
 - A **decision problem**: all questions (instances) map to either yes or no
 - Positive instances vs. negative instances
 - A correct algorithm **accepts** all positive instances (says yes) and **rejects** negative instances (says no)

9

11/10/2005

Decision Problems and Related Problems

- Problem P1: Primality Problem
 - Is n a prime number?
 - Instances: set of natural numbers
 - Positive instances?
- Related Problem P2: Find smallest divisor
 - Not a decision problem.
 - P2 could be used to solve P1.
 - In some sense, P2 is "harder".
- Graph coloring
 - P3: k -colorability. Given k , this is a decision problem
 - P4: Find chromatic number? not a decision problem
 - Could use k -colorability to solve it.

10

11/10/2005

Reminder: Graph Coloring

- A "coloring" is an assignment of "colors" to the set of vertices so that if vw is an edge, then $C(v) \neq C(w)$
- Two forms:
 - Optimization problem: given G , find the smallest number of colors that can be used to color G
 - The smallest k is known as G 's *chromatic number*, $\chi(G)$. We say G is k -colorable.
 - Decision problem: given G and a positive integer k , is it possible to assign a valid k -coloring to G ?
 - Note: book uses term *function problem*
 - More general. An optimization is a type of...

11

11/10/2005

Decision vs. Optimization

- Clearly an optimization problem is related to a particular decision problem
 - Decision problem: Is there a solution as good or better than some given bound?
 - Optimal value: What is the value of the best possible solution?
 - Optimal solution: Find a solution with the optimal value!
- E.g. graph coloring
 - Decision problem: Can G be colored with k colors?
 - Optimal value: What is the chromatic number of G ?
 - Optimal solution: Find a coloring of vertices that uses $\chi(G)$ colors.

12

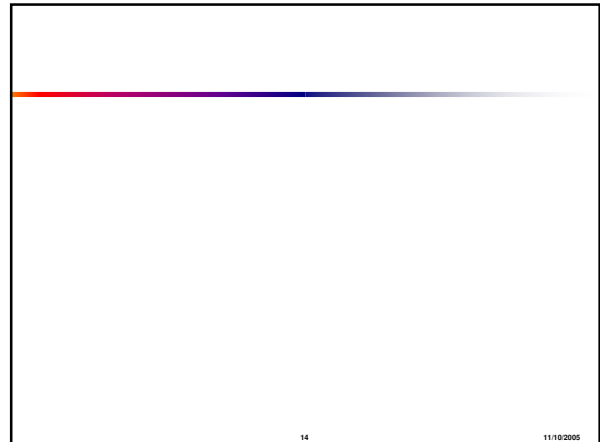
11/10/2005

Optimization and Decision Problems

- Optimization problems are at least as hard to solve as decision problems
 - E.g. if you can solve decision problem $\text{canColor}(G,k)$ then call it in a loop (1 to n) to find optimal value, $X(G)$
- Important: theory presented here for P, NP etc. is defined based on decision problems
 - But we'll see this isn't a problem...

13

11/10/2005



14

11/10/2005

Encodings, Input Sizes

- Our text takes a formal CS approach to these topics
 - Languages, accepting, encodings, etc.
- I choose in CS432 to be less formal
 - So I'll try to "translate" or simplify when I can
- So about pages 433-434 on encodings...

15

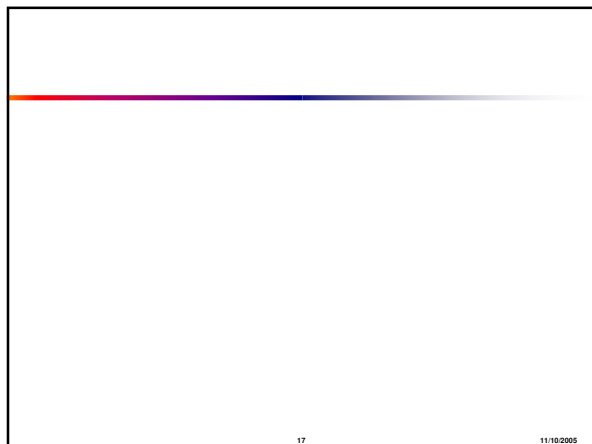
11/10/2005

Important: Input Size and P

- Sometimes a problem seems to be in P but really isn't
- Example: finding if value n is a prime
 - Just loop and do a mod: $\Theta(n)$, isn't it?
- Note that here "n" is not the count or number of data items.
 - There's just one input item.
 - But "n" is a value with a size that affects the execution time.
 - The size is the number of bits, which is $\log(n)$
 - $T(\text{size}) = n$ but size is $\log(n)$.
 - $T(\log n) = n = 10^{0.9n}$ This is really an exponential!
- Be careful when "n" is not a count of data items but a value
 - E.g. Dynamic programming problems (e.g. a table's dimension)

16

11/10/2005



17

11/10/2005

P and NP

- As mentioned, **P** is set of decision problems that can be solved in polynomial time
- **NP** (*nondeterministic polynomial time*) is the set of decision problems that can be solved in polynomial time by a *nondeterministic* computer
 - *What on earth is that?*
 - *Important: "NP" does not mean "not polynomial"!!!*

18

11/10/2005

Nondeterminism

- Think of a non-deterministic computer as a computer that magically “guesses” a solution, then has to verify that it is correct
 - If a solution exists, computer always guesses it
 - One way to imagine it: a parallel computer that can freely spawn an infinite number of processes
 - Have one processor work on each possible solution
 - All processors attempt to verify that their solution works
 - If a processor finds it has a working solution
 - So: **NP** = problems *verifiable* in polynomial time

19

11/10/2005

Nondeterminism (cont'd)

- Another way to think about it:
 - If you could always guess the answer but were required to verify your guess was really right, could you do this in polynomial time?
 - an *oracle*
- Just a second: solutions for decision problems are “yes” or “no”
 - How can guessing that help us?
 - The idea of verifying a solution in polynomial time is an informal definition of nondeterminism.

20

11/10/2005

Nondeterminism and our Text

- Pages 440-441
 - A *guess* function that makes a choice
 - Count it as one step (i.e. constant complexity)
 - If run again, could guess something else
 - Non-deterministic
 - Sequence of guesses is a *computation path* or *run*
 - If some run leads to a “yes”, then that sequence of choices is a *witness*
 - It proves that input is accepted by the algorithm

21

11/10/2005

Nondeterminism More Formally

- Non-deterministic algorithm **A** has 2 phases:
 - Non-deterministic phase: writes a string *s* (often called a *certificate*) somewhere
 - Deterministic phase: May use *s* and the input to return “yes” or “no” or nothing
- When run with same input, may produce different certificate *s*
- Total cost is cost of both steps
- Important: If **A** is given input *x* and for *some* execution it says “yes”, then the answer is “yes”.

22

11/10/2005

How to Think about Non-Det. Here

- Two ways:
 - If we ran a non-deterministic algorithm once and it made the right guess each time (wrote the correct certificate) if that was possible...
 - Or, if we ran it so many times over all possibilities, and one of those led to a “yes”...
- If some oracle could write out the “right” certificate, would we recognize it and say “yes”?
- (An aside: If not, then we’re not able to check a correct answer. That’s not good, is it?)

23

11/10/2005

Definition of Class NP

- Definition:
NP is the class of decision problems for which there is a polynomially bounded non-deterministic algorithm.
- Reminder: formally these are decision problems
 - But (for many problems) imagine a certificate that is the optimal solution
 - E.g. 3-colorability: *s* is a coloring
 - E.g. ham. path: *s* is a path
 - Can we “verify” these and say “yes” in poly. time?

24

11/10/2005

Proving Problems are in NP

- Self test:
 - *How do we typically prove a problem \in NP?*
- Examples:
 - *Is sorting in NP? (Not a decision problem! Could redefine it to be.)*
 - What could the certificate be? Could we verify it in poly. time?
 - *Is this in NP? Does a weighted graph G have a spanning tree with value $\leq k$?*
 - Think-Pair-Share activity: prove this belongs to NP
 - ◆ What's the certificate? What do the two phases do?

25

11/10/2005

Proving Problems are in NP (2)

- Note: The non-deterministic phase might do "nothing". Example:
- Problem: Does a graph G have a MST of total weight less than k?
 - Does this belong to NP? Yes! Outline of proof:
 - No need to generate a certificate or guess non-deterministically
 - Find MST using Kruskal's or Prim's algorithms
 - Compare weight of the MST to k
 - Thus we can verify a proposed yes/no answer in polynomial time
- Thus problems in P are easily shown to be in NP

26

11/10/2005

P and NP

- *Is $P \subseteq NP$? Why or why not?*
- Answer: all decision problems in P also belong to NP
 - Informally: you can solve them directly and compare the solution to the certificate
- But are they equal or is it a proper subset?
- In other words, is there a problem in NP that cannot be directly solved in polynomial time?
 - Is $P = NP$? Or not? (The big question!)

27

11/10/2005

Summary: P and NP

- Summary so far:
 - P = problems that can be solved in polynomial time
 - NP = problems for which a solution can be verified in polynomial time
 - Unknown whether $P = NP$ (most suspect not)
- We've seen problems that belong to NP that may not belong to P
 - Hamiltonian path/cycle, k-COL problems are in NP
 - Cannot solve in polynomial time
 - Easy to verify solution in polynomial time (*How?*)

28

11/10/2005

NP-Complete Problems

- We will see that NP-Complete problems are the "hardest" problems in NP:
 - If any *one* NP-Complete problem can be solved in polynomial time...
 - ...then *every* NP-Complete problem can be solved in polynomial time...
 - ...and in fact *every* problem in NP can be solved in polynomial time (which would show $P = NP$)
 - Thus: solve hamiltonian-cycle in $O(n^{100})$ time, you've proved that $P = NP$. Retire rich & famous.

29

11/10/2005

Reduction

- The crux of NP-Completeness is *reducibility*
 - Informally, a problem A can be reduced to another problem B if *any* instance of A can be "easily rephrased" as an instance of B, the solution to which provides a solution to the instance of A
 - *What do you suppose "easily" means?*
 - This rephrasing is called *transformation*
 - Intuitively: If A reduces to B, A is "no harder to solve" than B
 - Total cost: cost of transformation + cost to solve B

30

11/10/2005

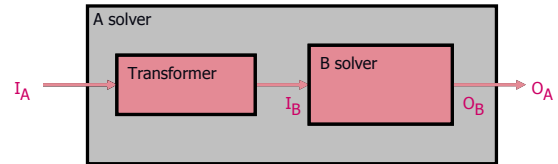
Reducibility

- An example:
 - A: Given a set of Booleans, is at least one TRUE?
 - B: Given a set of integers, is their sum positive?
 - Transformation: $(x_1, x_2, \dots, x_n) = (y_1, y_2, \dots, y_n)$ where $y_i = 1$ if $x_i = \text{TRUE}$, $y_i = 0$ if $x_i = \text{FALSE}$
- Another example:
 - Solving linear equations is reducible to solving quadratic equations
 - *How can we easily use a quadratic-equation solver to solve linear equations?*

31

11/10/2005

Reduction: A reduces to B



32

11/10/2005

NP-Hard and NP-Complete

- If A is *polynomial-time reducible* to B, we denote this $A \leq_p B$
- Definition of NP-Hard and NP-Complete:
 - If all problems $X \in \mathbf{NP}$ are reducible to A, then A is *NP-Hard*
 - We say A is *NP-Complete* if A is NP-Hard and $A \in \mathbf{NP}$
- If $A \leq_p B$ and A is NP-Complete, B is also NP-Complete
 - *You should be able to argue this is true from the definitions!*

33

11/10/2005

Why Prove NP-Completeness?

- Though nobody has proven that $\mathbf{P} \neq \mathbf{NP}$, if you prove a problem NP-Complete, most people accept that it is probably intractable
- Therefore it can be important to prove that a problem is NP-Complete
 - Don't need to come up with an efficient algorithm
 - Can instead work on *approximation algorithms*

34

11/10/2005

Proving NP-Completeness

- *What steps do we have to take to prove a problem A is NP-Complete?*
 - Pick a known NP-Complete problem B
 - Reduce B to A
 - Describe a transformation that maps instances of B to instances of A, s.t. "yes" for A = "yes" for B
 - Prove the transformation works
 - Prove it runs in polynomial time
 - Oh yeah, prove $A \in \mathbf{NP}$ (*What if you can't?*)

35

11/10/2005

Proving NP-Completeness

- (*We just said this:*) *What steps do we have to take to prove a problem A is NP-Complete?*
 - Pick a known NP-Complete problem B
 - Reduce B to A
- Why reduce B to A? Transformations are transitive
 - If B is NP-c, then all problems in NP reduce to B.
 - Then, if you find a transformation from B to A, the *composition* of two transformations would reduce any NP problem to A
 - The composition of two polynomials is polynomial

36

11/10/2005

Coming Up

- Given one NP-Complete problem, we can prove many interesting problems NP-Complete
 - Graph coloring (= register allocation)
 - Hamiltonian cycle
 - Hamiltonian path
 - Knapsack problem
 - Traveling salesman
 - Job scheduling with penalties
 - Many, many more