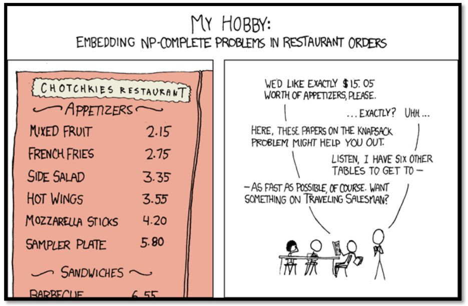
CSE 421: Algorithms

Winter 2014
Lecture 22: P, NP, and reductions

Reading:
Sections 8.1-8.3



computational complexity

- Classify problems according to the amount of computational resources used by the best algorithms that solve them
- Recall:
  - worst-case running time of an algorithm
    - max # steps algorithm takes on any input of size n

relative complexity of problems

Need a notion that allows us to compare the complexity of problems.

Want to make statements of the form:

"If we could solve problem B in polynomial time then we can solve problem A in polynomial time"

"Problem B is at least as hard as problem A"

polynomial-time reduction

- A $\leq_P$ B if there is an algorithm for A using a 'black box' (subroutine) that solves B that
  - Uses only a polynomial number of steps
  - Makes only a polynomial number of calls to a subroutine for B
- Thus, poly time algorithm for B implies poly time algorithm for A
  - Not only is the number of calls polynomial but the size of the inputs on which the calls are made is polynomial!
- If you can prove there is no fast algorithm for A, then that proves there is no fast algorithm for B

a math joke

- An engineer
  - is placed in a kitchen with an empty kettle on the table and told to boil water; she fills the kettle with water, puts it on the stove, turns on the gas and boils water.
  - she is next confronted with a kettle full of water sitting on the counter and told to boil water; she puts it on the stove, turns on the gas and boils water.
- A mathematician
  - is placed in a kitchen with an empty kettle on the table and told to boil water; he fills the kettle with water, puts it on the stove, turns on the gas and boils water.
  - he is next confronted with a kettle full of water sitting on the counter and told to boil water: he empties the kettle in the sink, places the empty kettle on the table and says, "I've reduced this to an already solved problem."

special kind of poly-time reduction

- We will always use a restricted form of polynomial-time reduction often called a "Karp" or **many-one reduction**
- **A** $\leq_P^1$ **B** if and only if there is an algorithm for **A** given a black box solving **B** that on input **x**
  - Runs for polynomial time computing an input **f(x)**
  - Makes one call to the black box for **B**
  - Returns the answer that the black box gave

  We say that the function **f** is the reduction.

reductions by simple equivalence

- Show: Independent-Set $\leq_P$ Clique
- Independent-Set:

  Given a graph **G=(V,E)** and an integer **k**, is there a subset **U** of **V** with **|U|** $\geq$ **k** such that no two vertices in **U** are joined by an edge?
- Clique:

  Given a graph **G=(V,E)** and an integer **k**, is there a subset **U** of **V** with **|U|** $\geq$ **k** such that every pair of vertices in **U** is joined by an edge?

Independent-Set $\leq_P$ Clique

- Given **(G,k)** as input to Independent-Set where **G=(V,E)**
- Transform to **(G',k)** where **G'=(V,E')** has the same vertices as **G** but **E'** consists of **precisely** those edges that are not edges of **G**
- **U** is an independent set in **G**

  $\Leftrightarrow$ **U** is a clique in **G'**

more reductions

- Show: Independent Set $\leq_P$ Vertex-Cover
- Vertex-Cover:
  – Given an undirected graph G=(V,E) and an integer k is there a subset W of V of size at most k such that every edge of G has at least one endpoint in W? (i.e. W covers all edges of G)?
- Independent-Set:
  – Given a graph G=(V,E) and an integer k, is there a subset U of V with |U| $\geq$ k such that no two vertices in U are joined by an edge?

reduction idea

- Claim: In a graph G=(V,E), S is an independent set iff V-S is a vertex cover
- Proof:
  ⇒ Let S be an independent set in G

  Then S contains at most one endpoint of each edge of G

  At least one endpoint must be in V-S

  V-S is a vertex cover

  ⇐ Let W=V-S be a vertex cover of G

  Then S does not contain both endpoints of any edge (else W would miss that edge)

  S is an independent set

reduction

- Map (G,k) to (G,n-k)
  – Previous lemma proves correctness
- Clearly polynomial time
- We also get that
  – Vertex-Cover $\leq_P$ Independent Set

reducing a special case to a general case

- Show: Vertex-Cover $\leq_P$ Set-Cover
- Vertex-Cover:
  – Given an undirected graph G=(V,E) and an integer k is there a subset W of V of size at most k such that every edge of G has at least one endpoint in W? (i.e. W covers all edges of G)?
- Set-Cover:
  – Given a set U of n elements, a collection $S_1, \ldots, S_m$ of subsets of U, and an integer k, does there exist a collection of at most k sets whose union is equal to U?

the simple reduction

- Transformation **f** maps
  (G=(V,E), k) to (U, S_1,…,S_m, k')
  - U ← E
  - For each vertex v∈V create a set S_v containing all edges that touch v
  - k'←k
- Reduction **f** is clearly polynomial-time to compute
- We need to prove that the resulting algorithm gives the right answer.

proof of correctness

Two directions:

- If the answer to Vertex-Cover on (G,k) is YES then the answer for Set-Cover on f(G,k) is YES
- If the answer to Set-Cover on f(G,k) is YES then the answer for Vertex-Cover on (G,k) is YES

decision problems

- Computational complexity usually analyzed using decision problems
  - answer is just 1 or 0 (yes or no).
- Why?
  - much simpler to deal with
  - *deciding* whether G has a path from s to t, is certainly no harder than *finding* a path from s to t in G, so a lower bound on deciding is also a lower bound on finding
  - Less important, but if you have a good decider, you can often use it to get a good finder.

polynomial time

Define ***P*** (polynomial-time) to be

- the set of all decision problems solvable by algorithms whose worst-case running time is bounded by some polynomial in the input size.

beyond P?

- There are many other natural, practical problems for which we don't know any polynomial-time algorithms
- For example: decisionTSP
  - Given a weighted graph G and an integer k, does there exist a tour that visits all vertices in G having total weight at most k?

satisfiability

- Boolean variables $x_1,\ldots,x_n$
  - taking values in {0,1}. 0=false, 1=true
- Literals
  - x_i or $\neg x_i$ for $i=1,\ldots,n$
- Clause
  - a logical OR of one or more literals
  - e.g. $(x_1 \vee \neg x_3 \vee x_7 \vee x_{12})$
- CNF formula
  - a logical AND of a bunch of clauses
- k-CNF formula
  - All clauses have exactly k variables

satisfiability

- CNF formula example
  $(x_1 \vee \neg x_3 \vee x_4) \wedge (x_2 \vee \neg x_4 \vee x_3) \wedge (x_2 \vee \neg x_1 \vee x_3)$
- If there is some assignment of 0's and 1's to the variables that makes it true then we say the formula is satisfiable
  - the one above is, the following isn't
  - $x_1 \wedge (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge \neg x_3$
- 3-SAT: Given a CNF formula F with 3 variables per clause, is it satisfiable?

common property of these problems

- There is a special piece of information, a **short certificate** or proof, that allows you to **efficiently verify** (in polynomial-time) that the YES answer is correct. This certificate might be very hard to find
- e.g.
  - **DecisionTSP**: the tour itself,
  - **Independent-Set**, **Clique**: the set U
  - **3-SAT**: an assignment that makes F true.

The complexity class NP

NP consists of all decision problems where

- You can **verify** the YES answers efficiently (in polynomial time) given a short (polynomial-size) **certificate**

and

- **No certificate** can fool your polynomial time verifier into saying YES for a NO instance

more precise definition of NP

- A decision problem is in NP iff there is a polynomial time procedure **verify**(.,.), and an integer **k** such that
  - for every input **x** to the problem that is a YES instance there is a certificate **t** with $|t| \leq |x|^k$ such that **verify**(x,t) = YES

  and
  - for every input **x** to the problem that is a NO instance there does **not** exist a certificate **t** with $|t| \leq |x|^k$ such that **verify**(x,t) = YES

CLIQUE is in NP

procedure **verify**(**x**,**t**)
  if **x** is a well-formed representation of
   a graph **G** = (**V**, **E**) and an integer **k**,
  and
   t is a well-formed representation of a vertex subset **U** of **V** of size **k**,
  and
   U is a clique in **G**,
  then output "**YES**"
  else output "**I'm unconvinced**"

is it correct?

keys to showing a problem is in NP

- What's the output? (must be YES/NO)
- What must the input look like?
- Which inputs need a YES answer?
  - Call such inputs YES inputs/YES instances
- For every given YES input, is there a certificate that would help?
  - OK if some inputs need no certificate
- For any given NO input, is there a fake certificate that would trick you?

what we know

- Nobody knows if all problems in NP can be done in polynomial time, i.e. does P = NP ?
  - one of the most important open questions in all of science.
  - huge practical implications
- Every problem in P is in NP
- Every problem in NP is in exponential time

solving NP problems without hints

The only **obvious algorithm** for most of these problems is **brute force**:

- try all possible certificates and check each one to see if it works.
- *Exponential* time:

  2^n truth assignments for **n** variables

  n! possible TSP tours of **n** vertices

  $\binom{n}{k}$ possible **k** element subsets of **n** vertices

  etc.