

CS215 REVIEW EXERCISES

not graded

Problem 1. (problem 7.15) Let UNARY-SSUM be the subset-sum problem in which all numbers are represented in unary. That is, each number k is encoded as a string of k “1”s.

- (a) Show that UNARY-SSUM is in P.
(b) Why does the NP-completeness proof for SUBSET-SUM in the book fail to show UNARY-SSUM is NP-complete? Be specific as you can about what is wrong with the content of the reduction. (Don’t just say, for example, “because UNARY-SSUM is in P, so if the reduction was correct, we would know that P=NP, and we don’t know that.”)

(a) *The standard dynamic programming algorithm for subset sum runs in time bounded by a polynomial in the size of the input and T , the target (when T is integer).*

(See <http://www.cs.ucr.edu/~neal/2004/cs141/index.cgi?SubsetSumByDP>.)

If the input is in unary, then here is a poly-time algorithm:

- 1. Check if the sum of the numbers is at least the target T .*
- 2. If not, reject.*
- 3. Otherwise, run the standard dynamic programming algorithm.*

You can verify that the algorithm is correct and runs in time polynomial in the size of the input.

(b) *Because the reduction is not polynomial-time. The numbers are of size around 2^n , so writing them out in unary takes exponential time.*

Problem 2. (a) (problem 7.17) Prove, using the definitions of P, NP, and NP-complete, that if P=NP, then each language L in P (except the languages Σ^* and \emptyset) is NP-complete.

(b) Prove formally that \emptyset is not NP-complete (according to the definition of NP-complete).

(a) *Suppose $P=NP$. Let L be any language in P. To show L is NP-complete, we need to show (1) L is in NP and (2) any language L' in NP reduces to L in polytime.*

(1) Since L is in P, L is in NP because $P \subseteq NP$.

(2) Let L' be a language in NP. Since (we are assuming) $P=NP$, L' is in P. So there is a poly-time machine M that decides L' . Since L is neither \emptyset nor Σ^ , there is a $x \in L$ and a $x' \notin L$. Here is the reduction f from L' to L :*

On input w :

- 1. Run M on w . If w accepts, output x . Otherwise output x' .*

The reduction is polynomial-time because M is. You can verify that the reduction is correct. (That is, $(\forall w)w \in L' \Leftrightarrow f(w) \in L$.)

(b) *If \emptyset were NP-complete, there would be a reduction f from SAT to \emptyset . Let F be any satisfiable formula. For the reduction to be correct, we would have to have $f(F) \in \emptyset$. But there is nothing in \emptyset , so this is impossible.*

Problem 3. (problem 7.24) Define $D = \{ \langle p(x_1, x_2, \dots, x_n) \rangle : p \text{ is a polynomial over variables } x_1, x_2, \dots, x_n \text{ and } p(c_1, c_2, \dots, c_n) = 0 \text{ for some } c_1, c_2, \dots, c_n \in N \}$. That is, D contains the multi-variable polynomials that have integer roots.

(a) Show that D is NP-hard.

Hint: $f(x)g(x) = 0$ iff $f(x) = 0$ or $g(x) = 0$, while $f(x)^2 + g(x)^2 = 0$ iff $f(x) = 0$ and $g(x) = 0$.

(b) What specifically is wrong with the following proof that D is in NP?

“Here is a verifier for D .”

$V(\langle p(x_1, \dots, x_n) \rangle, c) =$

1. Interpret c as a vector c_1, c_2, \dots, c_n of integers.
2. If $p(c_1, c_2, \dots, c_n) = 0$, accept.

Clearly step 1 and 2 can be evaluated in polynomial time, and clearly $D = \{ \langle p \rangle : (\exists c) V(\langle p \rangle, c) \}$ accepts. Thus, D has a polynomial-time verifier, and is in NP."

(a) We reduce 3SAT to it. Given a Boolean formula F in 3-cnf, construct a polynomial P as follows.

For each Boolean variable x_v in F , have a variable X_v in P .

Next, construct a set S of polynomials over the variables X_v as follows.

For each variable X_v , add a polynomial $X_v(1 - X_v)$ to the set S .

For each clause C in F , add a corresponding polynomial to the set. For example, if $C = x_1 \wedge x_2 \wedge \overline{x_3}$, then add the polynomial $X_1X_2(1 - X_3)$ to the set S .

Finally, define polynomial $p = \sum_{q \in S} q^2$.

Clearly this reduction is poly-time.

Next we verify its correctness.

(\Rightarrow) Suppose F is satisfiable. Fix a satisfying assignment. For each x_v that is assigned true, assign value 0 to the corresponding variable X_v in F . For each x_v that is assigned false, assign value 1 to the corresponding variable X_v in F .

Then every polynomial in the set S has value 0. (Each variable polynomial $X_v(1 - X_v)$ does, because $X_v \in \{0, 1\}$. Each clause polynomial such as $X_1X_2(1 - X_3)$ has value 0 because one of the terms has value 0.)

Thus, the polynomial p has value 0.

(\Leftarrow) Suppose p has an integer root, that is, a way of assigning an integer to each variable X_v so that p takes value 0.

Fix such an assignment. Since p has value 0, and $p = \sum_{q \in S} q^2$, it follows that each polynomial in S also has value 0.

Since there is a polynomial $X_v(1 - X_v)$ in S for each variable v , it follows that each variable X_v must be assigned 0 or 1.

If $X_v = 0$, then take $x_v = \text{true}$. If $X_v = 1$, then take $x_v = \text{false}$.

For each clause in F there is a corresponding polynomial in S that takes value 0. From this fact it is easy to verify that one of the literals in the clause must have value true (under the assignment above). Thus, F is satisfied by the assignment.

(b) The problem is that the integer roots may be arbitrarily large, so that the size of the certificate is not bounded by a polynomial in the size of D . Thus, the described verifier does not run in poly time.

Problem 4. (problem 7.26) See the book.

Answer postponed until after hwk redo's are turned in on Wednesday.

Problem 5. (problem 7.34) A legal coloring of a graph is an assignment of colors to the nodes so that no two adjacent nodes (sharing an edge) are assigned the same color.

Define 3COLOR = $\{ \langle G \rangle : \text{graph } G \text{ can be legally colored with three colors} \}$.

Show that 3COLOR is NP-complete. (For a hint, see the problem statement in the book.)

Here is a reduction from NAE-3SAT (from Papadimitriou's book).

The input to NAE-3SAT is a Boolean formula F in 3cnf form: a set of clauses C_1, C_2, \dots, C_m each with three literals. The question is whether there is an assignment to the variables that makes each clause have at least one true literal and at least one false literal.

For the reduction, we produce the following graph G .

For each variable x_i , make a triangle $[a, x_i, \bar{x}_i]$. Here a is a node shared among all triangles for all variables. For each clause C_i , we make another triangle $[C_{i1}, C_{i2}, C_{i3}]$. Here C_{ij} represents the j th literal in the clause. Finally, there is an edge from each C_{ij} to the corresponding node in the variable triangle for the literal.

Clearly the reduction is poly-time. We prove the formula F is NAE-satisfiable iff G is 3-colorable.

(\Rightarrow) Suppose F has an assignment of values to the variables that makes every clause have at least one true literal and at least one false literal.

We will use the colors gray, green, and red.

Color node a gray.

For each variable triangle $[a, x_i, \bar{x}_i]$, if x_i is true in the assignment, then color x_i green and \bar{x}_i red. Otherwise, color x_i red and \bar{x}_i green.

For each clause triangle $[C_{i1}, C_{i2}, C_{i3}]$, Color a C_{ij} corresponding to a true literal red, color a C_{ij} corresponding to a false literal green, and color the remaining C_{ij} gray.

It is easy to verify that this gives a legal 3-coloring.

(\Leftarrow) Suppose G has a legal 3-coloring.

By renaming the colors, we can assume that a is colored gray, and each node is colored either gray, green, or red.

For each variable triangle $[a, x_i, \bar{x}_i]$, either x_i is red and \bar{x}_i is green, or x_i is green and \bar{x}_i is red. In the former case, set x_i false. In the latter case, set x_i true.

Now consider any clause C_i , and consider the coloring of the clause triangle $[C_{i1}, C_{i2}, C_{i3}]$. These three nodes have to be colored different colors, so there is at least one red node and at least one green node. Since these nodes have edges to their corresponding literals in the variable triangles, those three variables must have one that is not red (which must be green, so that its variable is true) and one that is not green (which must be red, so that its variable is false).

Thus, the assignment gives at least one true and one false literal to each clause.

Problem 6. (problem 7.29) Show that if SAT is in P, then there exists a polynomial-time algorithm that, given any satisfiable Boolean formula $\phi(x_1, x_2, \dots, x_n)$, outputs a satisfying assignment (that is, it outputs $c_1, c_2, \dots, c_n \in \{\text{true}, \text{false}\}$ such that $\phi(c_1, \dots, c_n) = \text{true}$).

Hint: SAT being in P means that there is a polynomial-time algorithm A that, given any ϕ , outputs “yes” if ϕ is satisfiable, and “no” otherwise. But A doesn’t tell you the assignment. Use A to construct a poly-time algorithm A' that calls A several times and uses the answers to find an assignment.

Suppose SAT is in P. Let A be a polytime algorithm that, given a Boolean formula F , tells whether it is satisfiable.

Then the following algorithm produces a satisfying assignment, if there is one, in polytime:

1. Use A to check if F is satisfiable. If not, return.
2. Pick a variable x . Substitute “true” for x in F . Simplify the result to obtain formula F' (without variable x).
3. Use A to check whether F' is satisfiable.
4. If F' is, then assign x the value “true”.
5. Otherwise, assign x the value “false”, and take F' to be the formula obtained from F by substituting “false” for x in F .
6. Recurse to find an assignment for the remaining variables that satisfies F' .

Problem 7. (problem 7.33) What is wrong with the following proof that $P \neq \text{NP}$?

Here is an algorithm for SAT: “On input $\phi(x_1, \dots, x_n)$, try every possible assignment c (of true or false to the n variables). If any assignment c satisfies ϕ (i.e. $\phi(c) = \text{true}$), accept, else reject.”

This algorithm clearly requires exponential time. Thus, SAT has exponential time complexity. Therefore SAT is not in P. Because SAT is in NP, we conclude that $P \neq NP$.

To prove that SAT requires exponential time, we would have to show that all algorithms for SAT take exponential time (or worse). All we've shown is that this particular algorithm takes exponential time.

Problem 8. Recall that $\text{EQ}_{\text{CFG}} = \{\langle G_1, G_2 \rangle : G_1 \text{ and } G_2 \text{ are context-free grammars with } L(G_1) = L(G_2)\}$.

(a) (problem 5.2) Prove $\overline{\text{EQ}_{\text{CFG}}}$ (the complement of EQ_{CFG}) is Turing recognizable.

(b) (problem 5.1) Prove EQ_{CFG} is undecidable.

(c) Prove (using (a) and (b)) that EQ_{CFG} is not Turing recognizable.

(a) Here is a TM that recognizes $\overline{\text{EQ}_{\text{CFG}}}$:

On input G_1, G_2 :

1. Dovetail over all words $w \in \Sigma^*$, looking for a word w such that $w \in G_1$ but $w \notin G_2$ or vice versa.
2. If you find such a word, accept.

(To “dovetail” means the following:

1. for $i = 1, 2, 3, \dots$ do
2. for $j = 1, 2, \dots, i$ do
3. Spend i time units trying to find out if $w \in G_1$ but $w \notin G_2$ or vice versa.

If there is any word w that meets the condition, then eventually it will be found.)

(b) Recall Theorem 5.10: ALL_{CFG} is undecidable.

Here is a reduction from ALL_{CFG} to EQ_{CFG} :

1. On input G , output G, G' where $L(G') = \Sigma^*$.

Then $\langle G \rangle \in \text{ALL}_{\text{CFG}} \Leftrightarrow \langle G, G' \rangle \in \text{EQ}_{\text{CFG}}$.

By the reduction, if EQ_{CFG} were decidable, then ALL_{CFG} would be too.

(c) Recall from (a) the complement of EQ_{CFG} is Turing recognizable. If EQ_{CFG} were also Turing recognizable, then they would both be decidable. (Theorem 4.16)

Problem 9. (problem 5.5) Prove that A_{TM} is not mapping reducible to E_{TM} .

From the definition of mapping reducibility, if a language A mapping reduces to B , then \bar{A} mapping reduces to \bar{B} by the same reduction.

So, if A_{TM} was mapping reducible to E_{TM} , then $\overline{A_{\text{TM}}}$ would be mapping reducible to $\overline{E_{\text{TM}}}$.

Since the latter language is Turing recognizable, the former would be too. But we know it is not. (That is, $\overline{A_{\text{TM}}}$ is not Turing recognizable. We know this from Theorem 4.16 and the fact that A_{TM} is Turing recognizable but not decidable.)

Problem 10. (problem 5.9) Define a language B to be *Turing complete* if B is Turing recognizable and, for every Turing recognizable language A , A mapping reduces to B . Show that A_{TM} is Turing complete.

We have to show that for any Turing recognizable language L there exists a computable function f such that, for all $w \in \Sigma^$,*

$$w \in L \Leftrightarrow f(w) \in A_{\text{TM}}.$$

But this is easy: take $f(w) = \langle M, w \rangle$ where M is the recognizer for L .

(verify that $w \in L \Leftrightarrow f(w) \in A_{\text{TM}}$.)

Problem 11.

- (a) (problem 5.18) Define BINARY-PCP to be the PCP problem, restricted to pairs of binary strings:

Input: A collection of pairs $\{(a_i, b_i) : \text{each } a_i, b_i \in \{0, 1\}^*\}$.

Query: Is there a sequence of indices i_1, i_2, \dots, i_k such that $a_{i_1} a_{i_2} \dots a_{i_k} = b_{i_1} b_{i_2} \dots b_{i_k}$?

Show that BINARY-PCP is undecidable. Hint: reduce PCP over the alphabet used in the book to it.

- (b) (problem 5.17) Define UNARY-PCP to be the PCP problem, restricted to pairs of unary strings (strings over the alphabet $\Sigma = \{1\}$). Input: A collection of pairs $\{(a_i, b_i) : \text{each } a_i, b_i \in \{1\}^*\}$.

Query: As above.

Show that UNARY-PCP is decidable.

- (a) Given an instance $\{(a_i, b_i)\}$ of the general PCP problem (over an arbitrary alphabet Σ), construct an instance of BINARY-PCP as follows.

Let $\Sigma = \{c_1, c_2, \dots, c_k\}$.

For each c_i , define a corresponding string $\alpha_i = 01^i$.

Then, for each a_i , construct a'_i by replacing each character c in a_i by its corresponding α (as described above).

Likewise, for each b_i , define b'_i .

Then $\{(a'_i, b'_i)\}$ is an instance of PCP over alphabet $\{0, 1\}$.

We claim the above is a mapping reduction from PCP to BINARY-PCP. Clearly the reduction is computable.

It is easy to see that if $\{(a_i, b_i)\}$ has a match, then so does $\{(a'_i, b'_i)\}$.

We leave verifying the other direction to the reader.

- (b) We claim that the UNARY-PCP instance has a match if and only if there are pairs (a, b) and (a', b') where $|a| \geq |b|$ and $|a'| \leq |b'|$.

(This condition is easily checked by a TM.)

Clearly, if the condition is not true, then there can be no match. (Either all tops are larger than their bottoms, or all tops are smaller than their bottoms.)

On the other hand, if the condition is true, then consider taking some i copies of the domino (a', b') and j copies of the domino (a, b) .

The number of symbols on the top is then $i|a'| + j|a|$, while the number on the bottom is $i|b'| + j|b|$.

Taking $i = |a| - |b|$ and $j = |b'| - |a'|$, we get the top and the bottom have the same number of symbols (and $i, j \geq 0$). Thus, there is a match.

Problem 12. Describe, with proof, a polynomial-time reduction from $\text{FINITE}_{\text{TM}}$ to $\text{REGULAR}_{\text{TM}}$, where

$\text{FINITE}_{\text{TM}} = \{\langle M \rangle : M \text{ is a TM and } L(M) \text{ is finite}\}$.

$\text{REGULAR}_{\text{TM}} = \{\langle M \rangle : M \text{ is a TM and } L(M) \text{ is regular}\}$.

Here is the reduction:

Given $\langle M \rangle$, construct the following TM M' :

“On input w ,

1. If w is of the form $0^n 1^n$ where there exists $w' \in L(M)$ with length $|w'| = n$, then accept.”

Given M , the encoding $\langle M' \rangle$ can be output in time polynomial in $|M|$.

If $L(M)$ is finite, then $L(M')$ is finite, and hence regular.

On the other hand, if $L(M)$ is infinite, it is not hard to verify that $L(M')$ is not regular, because it is an infinite subset of $0^n 1^n$. (We can easily prove by the pumping lemma that such a language is not regular.)

Thus, $\langle M \rangle \in \text{FINITE}_{\text{TM}} \Leftrightarrow \langle M' \rangle \in \text{REGULAR}_{\text{TM}}$.

Thus, this is a polynomial-time reduction.

Problem 13. Give an example of a language that is:

(a) in P but not regular

Answer: $\{0^n 1^n\}$

(b) in NP but not NP-complete

Answer: \emptyset

(c) decidable but not in NP (and prove it)

Answer:

In fact, we show something stronger.

Define EXPTIME = to be the set of languages decidable in exponential time. That is, L is in EXPTIME iff there is a constant $c > 0$ and a Turing machine M that decides L in time 2^{n^c} (on inputs of size n).

We will construct a decidable language that is not in EXPTIME. Since EXPTIME contains NP, this is enough.

We construct such a language by diagonalization against all EXPTIME languages.

Let A be a Turing machine that recognizes A_{TM} .

Define the following Turing machine D :

“On input $\langle M \rangle$:

1. Say a state s of M is “useless” if there are no transitions into s and s is not the start state. Compute the number c of useless states. (Note: we are using useless states to “encode” a constant c in the TM description.)

2. Let $n = |\langle M \rangle|$.

4. Simulate M on input $\langle M \rangle$ for 2^{n^c} steps.

5. If M accepts within this time, reject, else accept.

By examination of D we can see that it always halts. So $L(D)$ is decidable. By inspection of D ,

$$L(D) = \{\langle M \rangle : M(\langle M \rangle) \text{ does not accept within } 2^{|\langle M \rangle|^c} \text{ steps}\} \quad (1)$$

(where c is the number of useless states in M).

Next we show that $L(D)$ is not in EXPTIME.

Suppose for contradiction that it is. Then there is an integer $c > 0$ and a TM M_D such that

$$L(D) = \{\langle M \rangle : M_D(\langle M \rangle) \text{ accepts within } 2^{|\langle M \rangle|^c} \text{ steps}\}.$$

By adding useless states to M_D (or removing them), we can get another machine M'_D with c useless states such that

$$L(D) = \{\langle M \rangle : M'_D(\langle M \rangle) \text{ accepts within } 2^{|\langle M \rangle|^c} \text{ steps}\}. \quad (2)$$

By (1),

$$\langle M'_D \rangle \in L(D) \Leftrightarrow M'_D(\langle M'_D \rangle) \text{ accepts within } 2^{|\langle M'_D \rangle|^c} \text{ steps}.$$

But by (2),

$$\langle M'_D \rangle \in L(D) \Leftrightarrow M'_D(\langle M'_D \rangle) \text{ does not accept within } 2^{|\langle M'_D \rangle|^c} \text{ steps}.$$

This gives a contradiction.

(d) Turing recognizable but not decidable

Answer: A_{TM}

(e) not Turing recognizable

Answer: $\overline{A_{TM}}$

(f) undecidable and unary (unary means $L \subseteq \{1\}^*$)

Answer:

Problem 14. Classify the following problems as in P, in NP, NP-complete, decidable, undecidable, Turing recognizable, not Turing recognizable. (Do not give redundant classifications. E.g. if a problem

(a) Given a CFG G and a DFA D , is $L(G) \subseteq L(D)$?

Answer:

1. Construct grammar G' such that $L(G') = L(G) \cap \overline{L(D)}$.

2. Accept if $L(G') = \emptyset$.

I believe both steps can be done in polynomial time (sorry, I don't have the details right now). So I believe this language is in P.

(b) Given a CFG G and a DFA D , is $L(D) \subseteq L(G)$?

Answer:

The language in (b) has ALL_{CFG} as a special case: "Given a CFG G , is $\Sigma^ \subseteq L(G)$?". That is, ALL_{CFG} easily reduces to this language.*

On the other hand, this language reduces to ALL_{CFG} . (Given G and D , construct grammar G' such that $L(G') = L(G) \cup \overline{L(D)}$. Then $L(G') = \Sigma^ \leftrightarrow L(D) \subseteq L(G)$.)*

So, this language has the same complexity as ALL_{CFG} .

Theorem 5.10 tells us ALL_{CFG} is undecidable. Note that $\overline{ALL_{CFG}}$ is Turing recognizable (given G , enumerate all strings looking for one that is not in $L(G)$, accept if you find one). Thus, ALL_{CFG} is not only not decidable, it is also not Turing recognizable (see Theorem 4.16). In sum, ALL_{CFG} is not Turing recognizable, but its complement is.

Thus, the language in (b) is the same: not Turing recognizable, but its complement is.

(c) Given the code for two C++ functions, do they compute the same function?

Answer: same as EQ_{TM} . Not Turing recognizable.

(d) Given the code for a C++ program, if you run it, will it halt?

Answer: same as $HALT_{TM}$. Turing recognizable, not decidable.

(e) Given a graph G and vertices s and t , does G contain a walk (a path that may revisit vertices) from s to t that traverses all vertices of G ?

Answer: Polynomial time.

(f) Given a graph G and vertices s and t , does G contain a walk from s to t such each vertex of G is visited at least once and at most 100 times?

Answer: NP-complete.

Given a graph G , s , t , make G' as follows. For each vertex v , add 99 copies of a triangle $[v, x, y]$ (so each x and y are used only once in any triangle). Then G has a HAMPATH from s to t iff G' has a path from s to t entering each vertex between 1 and 100 times.