

Homework 7

Due: April 13, 2005

Vinod Vaikuntanathan

Readings: Sections 7.1, 7.2, 7.3

Problem 1: Answer each of the following with TRUE or FALSE. You do not need to justify your answers. (Note: when dealing with sets like $O(f(n))$, $\Omega(f(n))$, etc., we use the symbols = and \in interchangeably.)

- | | |
|---|---|
| 1. $3 = O(n)$ – TRUE | 10. $1000n = o(n^3)$ – TRUE |
| 2. $12n = O(n)$ – TRUE | 11. $3^n = o(4^n)$ – TRUE |
| 3. $n^4 = O(n^3 \log^3(n))$ – FALSE | 12. $1000 = o(n)$ – TRUE |
| 4. $3n \log(n) + 1000n = O(n^2)$ – TRUE | 13. $n = o(\log^2(n))$ – FALSE |
| 5. $3^n = O(2^n)$ – FALSE | 14. $\frac{1}{2} = o(1)$ – FALSE |
| 6. $3^n = 2^{O(n)}$ – TRUE | 15. $\log_2(n) = \Theta(\log_{10}(n))$ – TRUE |
| 7. $2^{2^n} = O(2^{2^n})$ – TRUE | 16. $3^n = \Theta(4^n)$ – FALSE |
| Note: There was a formatting problem in this part. The problem was supposed to read “Is $2^{2^n} = O((2^2)^n)$?”, and the answer to that is FALSE. | 17. $n^3 = \Theta(8^{\log_2(n)})$ – TRUE |
| 8. $n = o(3n)$ – FALSE | 18. $n^2 = \Omega(n^3)$ – FALSE |
| 9. $n^n = O(n!)$ – FALSE | 19. $\log(n) = \Omega(\log(\log(n)))$ – TRUE |
| | 20. $4^{2^n} = \Omega(2^{4^n})$ – FALSE |

Problem 2: (Sipser problem 7.12)
Let

$$MODEXP = \{\langle a, b, c, p \rangle \mid a, b, c \text{ and } p \text{ are binary integers such that } a^b \equiv c \pmod{p}\}.$$

Show that $MODEXP$ is in P . (Note that the first and the most obvious algorithm you would come up would run in time *exponential in the input length*. Hint: Try it first when b is a power of 2.)

Solution 2: Let n be the length of the input. Our strategy will be to compute $a^b \pmod{p}$ and compare it to c . To compute $a^b \pmod{p}$, do the following:

- Write b in the binary representation. $b = b_1 b_2 \dots b_{n-1} b_n$, where the b_i 's are 0 or 1.
- Initialize $P := 1$.
- For $j = 1$ to n , do:
 - Compute $P := P^2 \times a^{b_j} \pmod{p}$.
(Note: If $b_j = 0$, $P := P^2 \pmod{p}$, and if $b_j = 1$, $P := P^2 \times a \pmod{p}$).
- Check if $P \stackrel{?}{=} c$. If so, output YES, else output NO.

Proof: (that the above algorithm works) We prove the correctness of the above procedure by induction.

Lets look at the value of P after the i^{th} iteration of the for loop (Call this value $P[i]$). Denote by $b|_i$ the number equivalent to the binary sequence $b_1b_2 \dots b_i$.

We claim that $P[i]$ is precisely $a^{b|_i} \pmod{p}$. Suppose this were true for some i . We want to prove that $P[i + 1] = a^{b|_{i+1}} \pmod{p}$. This follows from the following two observations.

- Note that $b|_{i+1} = b|_i \times 2 + b_{i+1}$ (This is by definition of $b|_i$'s and the property of binary representation).
- After the $(i + 1)^{th}$ iteration of the for loop, we have that $P[i + 1] = P[i]^2 \times a^{b_{i+1}}$. By the induction hypothesis, $P[i] = a^{b|_i} \pmod{p}$. Therefore, $P[i + 1] = (a^{b|_i})^2 \times a^{b_{i+1}} \pmod{p} = a^{b|_i \times 2 + b_{i+1}} \pmod{p} = a^{b|_{i+1}} \pmod{p}$.

To analyze the running time, we need the following (easily verifiable) facts about the complexity of elementary arithmetic operations:

- Multiplication of 2 n -bit numbers takes time at most $O(n^2)$ bit operations.
- Computing $a \pmod{b}$ for 2 n -bit numbers a and b takes at most $O(n^2)$ steps.

Each iteration of the for loop requires (at most) 3 multiplications and 1 mod-operation, and therefore takes time $O(n^2)$. The whole procedure therefore, takes time $O(n^3)$.

Problem 3: (Based on Sipser problem 7.14) Prove that P is closed under:

1. The concatenation operation.
2. The star operation.

Solution 3:

1. **CONCATENATION** – Let M_a and M_b be polynomial time algorithms deciding languages A and B , respectively. Say the running time of M_a is some polynomial $p_a(n)$ and that of B is $p_b(n)$. We describe a polynomial time algorithm N deciding AB as follows:

N = “On input $x = x_1x_2 \dots x_n$, where each x_i is a single character,

1. For $i = 1$ to $n - 1$, run M_a on $x_1 \dots x_i$ and M_b on $x_{i+1} \dots x_n$. If, for any i , both algorithms accept, accept.
2. Otherwise, reject.”

Analysis: In step 1, we try all $n - 1$ possible divisions, and each of these take an upper-bound of time polynomial in n to run on M_a on $x_1 \dots x_i$ and M_b on $x_{i+1} \dots x_n$. Considering that the actual inputs to M_a and M_b are of size n , the time taken is upperbounded by $p_a(n) + p_b(n)$. Thus, the overall running time is $O(n(p_a(n) + p_b(n)))$, which is also polynomial in n .

2. **STAR** – Let the input $x = x_1 \dots x_n$.

The idea is to use *dynamic programming*. Create an $n \times n$ matrix A such that the $(i, j)^{th}$ entry $A_{i,j}$ is 1 if and only if the string $x_i x_{i+1} \dots x_j \in L^*$. (Note that this makes sense only when $j \geq i$). To start with, it is easy to fill the A_{ii} entries of this matrix for all $1 \leq i \leq n$, since this is equivalent to deciding if the string $x_i \in L^*$, which is equivalent to deciding if $x_i \in L$ (this last decision problem is in P).

Fill the $(i, j)^{th}$ entry of the matrix iteratively, given $A_{i,k}$ and $A_{k,j}$ for all k such that $i \leq k < j$.

- If for some k such that $i \leq k < j$, both $A_{i,k}$ and $A_{k+1,j}$ are 1, then set $A_{i,j} = 1$.
- If $x_i x_{i+1} \dots x_j \in L$, then set $A_{i,j} = 1$.

- Else set $A_{i,j} = 0$.

(Note: If $A_{ik} = 1$ and $A_{kj} = 1$ for some k , it means that $x_i x_{i+1} \dots x_{k-1} \in L^*$ and $x_k \dots x_j \in L^*$. This means that $x_i \dots x_k \dots x_j \in L^*$. Here, we use the elementary fact that if $x \in L^*$ and $y \in L^*$, then $xy \in L^*$. Also, if $x_i \dots x_j \in L$, then $x_i \dots x_j \in L^*$ – another elementary fact.)

It is easy to see that this procedure is non-circular and it does terminate. At the end, we get the value of $A_{1,n}$ which, by definition, is 1 if and only if $x \in L^*$.

Problem 4: Prove that NP is closed under:

1. The intersection operation.
2. The concatenation operation.

Solution 4:

1. **INTERSECTION** – Let M_a and M_b be polynomial time algorithms deciding languages A and B , respectively. Say the (non-deterministic) running time of M_a is some polynomial $p_a(n)$ and that of B is $p_b(n)$. We describe a (non-deterministic) polynomial time algorithm N deciding $A \cap B$ as follows:

N = “On input x ,

1. Run M_a and M_b on x . If both M_a and M_b accept, accept.
2. Otherwise, reject.”

Analysis: The non-deterministic time-complexity of this procedure is clearly $O(p_a(n) + p_b(n))$.

2. **CONCATENATION** – Let M_a and M_b be (non-deterministic) polynomial time algorithms deciding languages A and B , respectively. Say the (non-deterministic) running time of M_a is some polynomial $p_a(n)$ and that of B is $p_b(n)$. We describe a (non-deterministic) polynomial time algorithm N deciding AB as follows:

N = “On input $x = x_1 x_2 \dots x_n$, where each x_i is a single character,

1. Guess a location i ($1 \leq i < n$).
2. Run M_a on $x_1 \dots x_i$ and M_b on $x_{i+1} \dots x_n$. If both M_a and M_b accept, accept.
3. Otherwise, reject.”

Analysis: If $x \in AB$, then (by definition), there exists a location i ($1 \leq i < n$) such that $x_1 x_2 \dots x_i \in A$ and $x_{i+1} \dots x_n \in B$. Thus, there exists a correct (non-deterministic) guess which makes N accept on input an $x \in AB$. On the other hand, if $x \notin AB$, there is no such right guess, and N rejects. The time taken is $O(p_a(n) + p_b(n))$.

Note: Alternatively, the same algorithm used in Problem 3(a) works for this part too, but it gives a slightly worse running time.

Problem 5: Prove that the following languages are in NP. You may use either the guess-and-check (certificate/verifier) method, or else describe a nondeterministic Turing machine that decides the language in time polynomial in the length of the input.

1. (From Sipser exercise 7.11)

$$ISO = \{ \langle G, H \rangle \mid G \text{ and } H \text{ are undirected graphs and } G \text{ and } H \text{ are isomorphic} \}$$

(Two graphs are *isomorphic* if, by renaming the nodes of one, we get a graph that is identical to the other.)

Solution 5: We prove that $ISO \in NP$ using the guess/check method.

Guess: Let L_G be the names of the nodes in G and L_H be the names of the nodes in H . Guess a bijection $\pi : L_H \rightarrow L_G$.

Check:

(1) For every pair of vertices i and j ($i, j \in L_H$) which are *connected* by an edge in H , if the vertices $\pi(i)$ and $\pi(j)$ in G *have an edge* between them, then this check passes.

(2) For every pair of vertices i and j ($i, j \in L_H$) which are *not connected* by an edge in H , if the vertices $\pi(i)$ and $\pi(j)$ in G *do not have an edge* between them, then this check passes.

If both these checks pass, then accept. Else, reject.

If G and H are isomorphic, then (by definition), there is an isomorphism π between the labels of the vertices in H and those in G . Thus there is a certificate that the verifier accepts. On the other hand, if G and H are not isomorphic, there is no such certificate.

2. $TRIPLE-SAT = \{ \langle \phi \rangle \mid \phi \text{ is a Boolean formula and } \phi \text{ has at least three distinct satisfying assignments} \}$
(Boolean formulas are defined on p. 271 of Sipser's book.)

Solution 5:

Guess: Let the number of variables in the formula be n . Guess three assignments A_1, A_2, A_3 to the variables in the formula. (An assignment can be represented as a length- n vector with boolean values as its entries).

Check:

Check that the three assignments are distinct. (That is, the corresponding vectors of Boolean values are unequal). Also, check if the formula ϕ evaluates to true, for all the three assignments. If both these checks pass, accept. Else, reject.

The proof that this is correct, is self-evident and is omitted.

3. A crossword puzzle construction problem is specified by a finite set $W \subseteq \Sigma^*$ of words, and an $n \times n$ matrix A whose entries are either 0 or 1 (intuitively, a 0 corresponds to a blank square, and a 1 corresponds to a black square). The goal is to use the words in W to fill in the blank squares. Formally, suppose E is the set of all pairs (i, j) such that A_{ij} , the $(i, j)^{th}$ entry of A , is 0. We want to find a mapping $f : E \rightarrow \Sigma$ such that the letters assigned to any maximal horizontal or vertical contiguous sequence of members of E form, in order, a word of W . If this is possible, we say that (W, A) is a *constructable crossword system*.

$$CROSSWORD = \{ (W, A) \mid W \subseteq \Sigma^* \text{ and } A \text{ is an } n \times n \text{ } 0-1 \text{ matrix and } (W, A) \text{ is a constructable crossword system.} \}$$

(For instance, the set $W = \{a, b, ab, ba, aba\}$ over the alphabet $\{0, 1\}$ and the matrix A as in the figure form a constructable crossword system. One of the crosswords so constructed is the matrix B in the figure.)

Solution 5:

Guess: Guess a function $f : E \rightarrow \Sigma$.

Check:

Check, for each maximal horizontal/vertical sequence of "blank" squares that, the word (formed by concatenating the corresponding symbols), is in the dictionary W .

The proof that this is correct, is self-evident and is omitted. The certificate (guess) is of size at most n^2 , and is therefore, polynomial in the size of the input. Verifying takes at most $O(n^2|W|)$ time, since there are at most n^2 maximal contiguous rows/columns in A , and for

1	0	0	0
0	0	1	1
1	0	0	0
1	1	0	0

A

	a	b	a
a	b		
	a	b	a
		a	b

B

$$W = \{ a, b, ab, ba, aba \}$$

each of them, checking if the word in the row/column is in W takes (by a naive linear search) time $O(|W|)$.

Grading Guidelines:

Problem	Points/Guidelines
1	2 points – 0.1 point for every part. Incorrect answers get no credit.
2	2 points – 1.5 points for the algorithm, and 0.5 points for the correctness proof/running time analysis.
3	0.5 + 1.5 points – Problem 3(a) carries 0.5 points, and problem 3(b) carries 1.5 points. If the boundary conditions are incorrect or are not stated explicitly (in problem 3b), 0.5 points will be deducted.
4	0.5 + 0.5 points – This problem is relatively straightforward.
5	0.5 × 3 points – Again, this is pretty straightforward. If guess/check method is used, the guess part and the check part should be explicitly stated. In any case, the running time of the procedure should be (at least informally) analyzed.