6.840 Dan Ports
Theory of Computation drkp@mit.edu
2004/11/04

10

Problem 4-1

The input is a tuple $\langle a, b, c, p\rangle$: the integers are represented in binary, so the length of the input is polynomial in the logarithm of the largest value. We wish to test whether $a^b \equiv c \pmod{p}$

Consider the following algorithm:

```
MOD-EXP
1  r ← a
2  β ← 0
3  for i ← ⌈log_2 b⌉ downto 0
4        do r ← r^2 (mod p)
5           β ← 2β
6           if the ith least significant bit of b is 1
7              then r ← ra (mod p)
8                   β ← β + 1
```

This algorithm produces the result in r after it terminates. To prove correctness, observe that the invariant $r = a^\beta \pmod{p}$ holds, and also the invariant that β contains bits $\lceil \log_2 b \rceil \cdots i$ of b, i.e. that β is a prefix of b. Then, when the algorithm terminates, $i = 0$, $\beta = b$, and $r = a^b \pmod{p}$.

Now consider the runtime of this procedure. First note that all integers involved have bit representation polynomial in the length of the input, so multiplication steps can be performed in polynomial time. Next observe that the **for** loop is executed at most $\lceil \log_2 b \rceil$ times, which is linear in the input length. So at most $2\lceil \log_2 b \rceil$ polynomial time multiplications are performed, so the algorithm runs in polynomial time and $MOD - EXP \in \mathsf{P}$.

6.840
Theory of Computation
2004/11/04

Dan Ports
drkp@mit.edu

10

Problem 4-2

a) Let ϕ be a CNF formula where each variable appears in at most 2 places. We give a procedure for determining in polynomial time whether it is satisfiable.

First, we scan through the formula (in $O(n^2)$ time) and divide the variables into three classes: those that appear only once; those that appear twice, either both unnegated or both negated; and those that appear twice, once negated and once unnegated. Note that in the first two cases, there is one assignment for each variable (x or $\overline{x}$) that will satisfy all clauses it appears in. So we set that variable to that value, remove it from ϕ, and remove the clauses it satisfies from ϕ. Then we are left with a formula ϕ where each variable appears in exactly two places, once negated and once non-negated.

Now we generate a bipartite graph with one vertex corresponding to each variable and one corresponding to each clause. We connect each variable to the clauses it can satisfy, tagging the edge with the necessary assignment of the variable. We then compute the maximum bipartite matching of this graph. If every clause is matched to some variable, ϕ is satisfiable, and we can choose the assignment corresponding to each edge used (and arbitrarily assign the values of any variables which were not matched). If the matching algorithm cannot match some clause, with a variable, then there it is unsatisfiable. The bipartite matching can be computed in polynomial time (using, e.g. a reduction to MAX-FLOW and Dinic's algorithm), so $CNF_2 \in \mathsf{P}$.

b) We show that CNF_3 is NP-complete by reduction from $3SAT$. Let ϕ be a formula in CNF form. We show how to reduce it in polynomial time to a CNF formula where each variable is used at most three times.

Suppose ϕ contains some variable x that is used $k > 3$ times. Replace x with the variables $\{x_1, x_2, \ldots, x_k\}$, and replace the ith occurrence of x with x_i. Now each of the x_i is used only once. To ensure that the new formula is equivalent (in terms of satisfiability) to the old, we must now ensure that they all take the same value. To do so, add the clauses

$$(x_1 \vee \overline{x_2}) \wedge (x_2 \vee \overline{x_3}) \wedge \cdots \wedge (x_{k-1} \vee \overline{x_k}) \wedge (x_k \vee \overline{x_1})$$

This is satisfied only if the x_i are either all true or all false.

We can repeatedly apply this procedure to remove all variables that are used more than k times. This is a polynomial-time operation, since $k_x - 1$ variables and k_x clauses are added, where k_x is the number of times variable x appears, and $\sum_x k_x$ is less than the total length of the problem. So $3SAT$ is polytime-reducible to CNF_3. It is obvious that $CNF_3 \in \mathsf{NP}$, so CNF_3 is NP-complete.

6.840 Dan Ports
Theory of Computation drkp@mit.edu
2004/11/04

Problem 4-3

(10)

Let $U = \{\langle M,\ x,\ \#^t\rangle \mid \mathsf{TM}M$ accepts x on at least one branch in t steps$\}$. Then U is NP-complete. First, $U \in \mathsf{NP}$. Let the certificate w be an accepting computation history for M on x. We can verify in the standard way that the computation history is valid for M's transition function, and that it has no more than t steps. This verification can be performed in polynomial time. ✓

Next, we show (directly) that U is NP-complete. Let $\mathcal{N}$ be some problem in NP. Then there is a nondeterministic TM M that decides $\mathcal{N}$ in polynomial time. Let $f(x)$ be the function that maps inputs x to the number of steps required for M to decide x; $f(x)$ is polynomial in $|x|$ since M is a polynomial time TM.

Let $\phi = \langle M,\ x,\ \#^{f(x)}\rangle$. Note that $|\phi| = O(x + f(x))$, which is polynomial in x. Then $\phi \in U$ iff M accepts x, since M always decides in at most $f(x)$ steps. So $\mathcal{N}$ is reducible to U. $\mathcal{N}$ was an arbitrary problem in NP, so U is NP-complete. ✓

8/10

6.840 Dan Ports
Theory of Computation drkp@mit.edu
2004/11/04

Problem 4-4

We reduce $3SAT$ to $3-COLOR$. Let ϕ be a formula in 3-CNF form. We design a graph, using the "palette," "variable", and "or" gadgets. that is 3-colorable iff ϕ is satisfiable.

We place one copy of the "palette" in our graph, with vertices labeled T, F, and μ. We use this to "name" the three colors. We create one instance of the "variable" subgraph for each variable x: two connected vertices which we call x and $\overline{x}$, and connect each of them to vertex μ. Observe that this ensures that if x is colored T, $\overline{x}$ must be colored F and vice versa.

Consider the "or" gadget. Call the two "dangling" vertices the inputs, and the vertex at the point of the triangle the output. Observe that the output must have the same color as one of the inputs: the two intermediate vertices must have some color other than the one associated with their attached vertex, and the output must have a different color than either of these intermediate vertices. So the only feasible colorings have the output vertex with the same color as one of the inputs. Note that we can combine two of these "or" gadgets to obtain a "3-or" gadget: we make the output vertex of one of the "or" gadgets the same vertex as one of the inputs of the other gadget. The resulting gadget has three input vertices, and the output vertex must have the same color as one of the inputs.

We use this to build a graph for the formula ϕ. For each clause $(x \vee y \vee z)$ (noting that x, y, and z may actually be negated variables), we place a "3-or" gadget in the graph, with the input vertices being the same vertices as the x, y, and z vertices in the "variable" gadget. We then connect the output vertex of every "3-or" gadget to the F vertex in the "palette".

Observe that this graph is 3-colorable iff ϕ is satisfiable. The variable vertices will be colored either T or F, and each clause's "3-or" gadget's output vertex can be colored T iff the coloring of the variable vertices is an assignment that satisfies the clause. The connection between the F and output vertices ensures that every output vertex must be T (it cannot be color μ), i.e. that the assignment satisfies every clause. If the graph is 3-colorable, we can read the satisfying assignment from the colors of the variable vertices; if it is not, no satisfying assignment exists.

Is the reduction poly-time? -1

This shows 3COLOR $\in$ NP-HARD. Must also show $\in$ NP to show $\in$ NP-COMPLETE. -1

6.840
Theory of Computation
2004/11/04

(10)

Dan Ports
drkp@mit.edu

Problem 4-5

Assume P = NP. We show that it is possible to factor integers in polynomial time.

Define the language $L = \{\langle n, m\rangle \mid n \text{ has a prime factor less than or equal to } m\}$. Clearly $L \in$ NP: the certificate is the prime factorization of n; we need only multiply the prime factors and verify that their product is n, and check whether the smallest prime factor is at most m. So $L \in$ P.

To determine the factorization of an integer n, we perform a binary search over the interval $[2, n]$ to find the smallest i such that $\langle n, i\rangle \in L$. Then i is a prime factor of n, so we output i. We then divide n by i. If n/i is 1, we are done; otherwise, we recursively apply this procedure to n/i.

The correctness of this procedure is clear; we show that it has polynomial runtime. Since every prime factor of n is less than n, i is always less than n, so checking $\langle n, i\rangle \in L$ takes polynomial time. On every iteration, we perform this check at most $O(\log n)$ times, which is polynomial in the input length, so the runtime of each iteration is polynomial. The number of primes is less than $\log_2 n$ since every prime is at most 2, and the input length is $\Omega(\log n)$, so the algorithm's total runtime is polynomial in the length of the input n.