Quiz

(*reference answers*)

Prof. Manisha Kulkarni, Prof. G.Srinivasaraghavan

On: April 19, 2023	**Time**: 90 Mins	**Max Marks**: 20

Q-1: We know that the asymptotic running time for both the product $m * n$ and the Extended GCD $\texttt{EGCD}(m, n)$ for any two integers m, n is $O(\texttt{len}(m) * \texttt{len}(n))$. (a) Argue why the actual running time (clock time) for $m*n$ is likely to be significantly faster (by a not-so-small, though bounded, constant factor) than $\texttt{EGCD}(m, n)$. (b) In this context design an algorithm for computing the inverses (modulo n) for a set of numbers $\alpha_1, ..., \alpha_{k+1} \in Z_n^*$ that computes just one inverse directly (presumably using EGCD) along with at most $3k$ products, modulo n. In other words, design an algorithm $\mathcal{A}$ which returns $(\alpha_1^{-1}, ..., \alpha_{k+1}^{-1})$ such that

$$T\left(\mathcal{A}(n, \alpha_1, ..., \alpha_{k+1})\right) \leq T\left(\texttt{EGCD}(\alpha, \beta)\right) + 3k.T\left(\alpha *_{\text{mod } n} \beta\right)$$

where $\alpha_1, ..., \alpha_{k+1} \in Z_n^*$ and α, β are arbitrary elements of Z_n^*.

Marks: 6

Answer:

(a) We discussed in the class (also see Theorem 4.2 of the Textbook by V.Shoup) that the running time of $\texttt{EGCD}(m, n)$ is given by

$$\begin{aligned} T\left(\texttt{EGCD}(m, n)\right) &\approx \texttt{len}(m) * \texttt{len}(n) + \texttt{len}(n)\left(\frac{\log n}{\log \phi} + 1\right) \\ &\approx T\left(m * n\right) + \texttt{len}(n)\left(\frac{\log n}{\log \phi} + 1\right) \end{aligned}$$

where ϕ is the Golden Ratio $(1 + \sqrt{5})/2$. The second term on the RHS is the overhead incurred by EGCD though it is asymptotically subsumed by $T\left(m * n\right)$.

(b) The following algorithm will compute the α_i^{-1}s in the required time. The main call is $\texttt{FindAllInverses}(n, \alpha_1, ..., \alpha_{k+1})$. $\texttt{AllInverses}(n, \alpha_1, ..., \alpha_i, P_2, \ldots, P_{k+1}, P_i^{-1})$ is an auxiliary function that is the core of the algorithm. It basically implements a simple inductive strategy to compute the α_i^{-1}s. The algorithm has been described recursively for ease of description. Clearly AllInverses gets invoked at most k times and as the annotations show, each invokation of AllInverses requires two multiplications. Therefore the time spent in AllInverses is at most $2k * T\left(\alpha *_{\text{mod } n} \beta\right)$. Therefore the total time is (including the time in FindAllInverses):

$$T\left(\texttt{EGCD}(\alpha, \beta)\right) + (2k + k) * T\left(\alpha *_{\text{mod } n} \beta\right)$$

satisfying the claim in the question.

■

```
1  def FindAllInverses(n, α_1, ..., α_{k+1}):
2  |  Compute P_i = ∏_{j=1}^{i} α_j  mod n for each 1 < i ≤ (k+1)     /* Time k * T(α *_{mod n} β) */
3  |  Compute P_{k+1}^{-1}  mod n                                     /* Time T(EGCD(α, β)) */
4  |  return AllInverses(n, α_1, ..., α_{k+1}, P_2, ..., P_{k+1}, P_{k+1}^{-1})
5
6  def AllInverses(n, α_1, ..., α_i, P_2, ..., P_{k+1}, P_i^{-1}):
7  |  if i == 1 then return (P_1^{-1} ≡ α_1^{-1})
8  |  P_{i-1}^{-1} ← α_i * P_i^{-1}    (Since P_i^{-1} = α_i^{-1} * ... * α_1^{-1}  mod n)   /* Time T(α *_{mod n} β) */
9  |  (α_1^{-1}, ..., α_{i-1}^{-1}) ← AllInverses(n, α_1, ..., α_{i-1}, P_2, ..., P_{k+1}, P_{i-1}^{-1})
10 |  α_i^{-1} ← P_i^{-1} * P_{i-1}                                  /* Time T(α *_{mod n} β) */
11 |  return (α_1^{-1}, ..., α_{i-1}^{-1}, α_i^{-1})
```

Q-2: Here's (algorithm `AnotherBinaryGCD`) a variant of the binary GCD. In the algorithm $\gg$ denotes a binary right shift, for dividing by 2. Also assume that at the start of the algorithm $a \geq b$.

```
1  def AnotherBinaryGCD(a, b):
2  |  Set e ← 0
3  |  while 2 | a  and  2 | b do a ≫ 2, b ≫ 2, e += 1
4  |  while b > 0 do
5  |  |  while 2 | a do a ≫ 2
6  |  |  while 2 | b do b ≫ 2
7  |  |  if a < b then Swap(a, b)
8  |  |  a ,  b ← (a + b) ≫ 2 ,  (a − b) ≫ 2
9  |  end
10 |  return 2^e.a
```

(a) Show that the algorithm `AnotherBinaryGCD` is guaranteed to terminate. **Marks: 2**

(b) Prove that the output of `AnotherBinaryGCD` is the correct `gcd` of a and b. **Marks: 2**

(c) Show how you can modify this algorithm to get an 'Extended' version that outputs the coefficients s, t along with the `gcd` d such that $d = as + bt$. **Marks: 5**

Answer:

(a) A simple invariant at the beginning of the while loop is obviously $a \geq b$ (if a becomes smaller than b in the previous iteration (due to statements 5,6) then the swap in statement 7 would restore the invariant). It is easy to see that if $a = b$ then the b becomes 0 in the next iteration (step 8) and the algorithm terminates. If $a > b$ then it is easy to see that $(a+b)/2 < a$. So a monotonically decreases (strictly) through the iterations, ensuring that it will eventually terminate.

(b) The only on trivial step here is Step 8. The first observation here is that this step is valid — because Steps 5,6 ensure that both a and b are odd when we reach step 8. We only need to show that

$$\texttt{gcd}(a, b) = \texttt{gcd}\left((a + b) \gg 2, (a - b) \gg 2\right)$$

. We show this by proving that the set of common divisors of a, b is identical to the set of common divisors of $\left(\frac{a+b}{2}, \frac{a-b}{2}\right)$. It is easy to see that if $d \mid a$ and $d \mid b$ for some d then

$d \mid \frac{a+b}{2}$ and $d \mid \frac{a-b}{2}$. This along with the observation that taking $x \equiv \frac{a+b}{2}$ and $y \equiv \frac{a-b}{2}$, (a, b) becomes $\left(\frac{x+y}{2}, \frac{x-y}{2}\right)$ proves our claim.

(c) It is easy to 'extend' this to produce s, t by following the Binary `EGCD` algorithm discussed in the class (also see Exercise 4.10 of the Textbook by V.Shoup). We follow the same scheme with variables r, r', s, s', t, t'. The only modification will be to account for the new 'reduction' step $(a, b) \leftarrow \left(\frac{a+b}{2}, \frac{a-b}{2}\right)$ (the 'usual' reduction step is $(a, b) \leftarrow (b, (a-b))$). The new update rule will be:

$$(r, r') \leftarrow \left(\frac{r+r'}{2}, \frac{r-r'}{2}\right)$$

$$(s, s', t, t') \leftarrow \begin{cases} \left(\frac{s+s'}{2}, \frac{s-s'}{2}, \frac{t+t'}{2}, \frac{t-t'}{2}\right) & \text{if } 2 \mid (s+s') \text{ and } 2 \mid (t+t') \\ \left(\frac{s+s'+\tilde{b}}{2}, \frac{s-s'+\tilde{b}}{2}, \frac{t+t'-\tilde{a}}{2}, \frac{t-t'-\tilde{a}}{2}\right) & \text{otherwise} \end{cases}$$

Just as we did for the version of Binary EGCD discussed in the class, it is easy to verify that the two invariances

$$r = \tilde{a}s + \tilde{b}t, \qquad r' = \tilde{a}s' + \tilde{b}t' \tag{1}$$

are indeed preserved with these updates, provided the divisions by 2 (in the 'otherwise') result in integers. Note that $\tilde{a}, \tilde{b}$ are the values of a, b just after the common powers of 2 have been removed in step 3. A case-based argument just as was done in the class would justify all the divisions by 2. To illustrate, suppose we denote the statement "x is odd" by $O(x)$ and 'x is even" by $E(x)$, we know that when we reach step 8, (i) $O(r)$ and $O(r')$, (ii) both $E(\tilde{a})$ and $E(\tilde{b})$ cannot be true and (iii) both $E(s+s')$ and $E(t+t')$ cannot be true. (iv) Equations 1 hold.
We will prove only that $E(s+s'+\tilde{b})$ and $E(t+t'-\tilde{a})$ and observe that for any two integers x, y, $E(x+y)$ iff $E(x-y)$. So the proofs will hold for for $E(s-s'+\tilde{b})$ and $E(t-t'-\tilde{a})$ as well.
Case 1: $E(\tilde{a})$. Then $E(\tilde{a}s), E(\tilde{a}s') \Rightarrow O(\tilde{b}t), O(\tilde{b}t') \Rightarrow O(t), O(t') \Rightarrow E(t+t') \Rightarrow O(s+s')$. Also $O(\tilde{b})$. Therefore $E(s+s'+\tilde{b})$ and $E(t+t'-\tilde{a})$. The case $E(\tilde{b})$ will be identical.
Case 2: $O(\tilde{a})$ and $O(\tilde{b})$. Then either $E(s)$ and $O(t)$ or $O(s)$ and $E(t)$ (otherwise r will not be odd). Similar statement holds for s' and t' as well.
(A) So Suppose $E(s)$ and $O(t)$. It cannot be that $E(s')$ and $O(t')$ (then condition (iii) above is violated). Therefore it must be $O(s')$ and $E(t')$. Then it must be $E(s+s'+\tilde{b})$ and $E(t+t'-\tilde{a})$.
(B) If $O(s)$ and $E(t)$. Then $E(s'), O(t') => E(s+s'+\tilde{b}), E(t+t'-\tilde{a})$

■

Q-3: Suppose Alice and Bob each have an n-bit message M_A, M_B which they believe are identical but wish to verify this with high probability using only a small amount of communication. Sending the message (M_A or M_B) as is, is not an option — n is too large for that to be practical. Here's the protocol they use to verify if $M_A = M_B$.

```
/* One of them (say Alice) initiates the protocol as below                    */
Select k primes p_1, ..., p_k uniformly at random from the first 2n primes
Send {p_1, ..., p_k, r_1, ..., r_k} to Bob                  /* r_i = M_A mod p_i */
/* Below is what Bob does after receiving the message above                   */
if ∀1 ≤ i ≤ k,  r_i == M_B mod p_i then
    Declare that (almost surely) M_A = M_B
else
    Declare that M_A ≠ M_B
end
```

Algorithm 1: Message Equivalence Verification Protocol

FYI (*This is not required for the answers to the questions below*): It is known that the m^{th} prime will be in the range $[(m(\ln m + \ln\ln m - 1),\ \ m(\ln m + \ln\ln m)]$. So the value of the largest among the p_is (and consequently the largest among the r_is) is at most $2n * (\ln(2n) + \ln\ln(2n))$. Therefore $\max\{\texttt{len}(p_i), \texttt{len}(r_i) \mid 1 \leq i \leq k\} \leq c.\log n$ for some small constant c. So Alice needs to send only $2ck \log n$ bits to Bob for verification as opposed to n bits if she were to send M_A as is to Bob.

Show that the protocol described above is effective. Specifically show that:

(a) The protocol does not make an error when $M_A = M_B$ — Bob will indeed correctly conclude that $M_A = M_B$.

(b) However when $M_A \neq M_B$ then $pr(M_A =_{\texttt{protocol}} M_B) \leq 2^{-k}$ — the probability of the protocol resulting in an incorrect conclusion is insignificant (bounded by 2^{-k}). *Hint*: Show that the number of distinct prime divisors of any integer $N > 0$ is at most $\log N$.

Marks: 5

Answer:

(a) If $M_A = M_B$ then obviously for any p, $M_A = M_B \mod p$. The protocol would indeed correctly conclude $M_A = M_B$.

(b) If $M_A \neq M_B$ but the protocol says $M_A = M_B$, then it is because $\forall 1 \leq i \leq k,\ \ r_i == M_B \mod p_i$. Therefore $\forall 1 \leq i \leq k,\ \ (M_A - M_B) \equiv 0 \mod p_i \Rightarrow p_i \mid (M_A - M_B)$. If D is the set of all prime divisors of $(M_A - M_B)$, then it means it so happened that $\forall 1 \leq i \leq k, p_i \in D$. Probability of selecting a p_i is therefore (since it is chosen at random from the first $2n$ primes) at most $|D|/(2n)$. So the probability that $\forall 1 \leq i \leq k, p_i \in D$ is at most $(|D|/(2n))^k$. To complete the claim in the question we only have to show that $|D| \leq n$.

Clearly for an arbitrary number $N = p_1^{e_1} * \ldots * p_t^{e_t} \geq 2 * 2 * \ldots * 2$ (2 repeated t times). Therefore $2^t \leq N \Rightarrow t \leq \log N$, where t is the number of distinct prime divisors of N. Since $M_A - M_B$ has n bits, $|M_A - M_B| \leq 2^n$. So the number of distinct divisors of $(M_A - M_B)$ is at most $\log 2^n = n$. Therefore $|D| \leq n$.

■