CS 170 HW 12

Due on 2018-11-18, at 9:59 pm

1 (★) Study Group

List the names and SIDs of the members in your study group.

2 (★★★) Independent Set Approximation

In the Max Independent Set problem, we are given a graph $G = (V, E)$ and asked to find the largest set $V' \subseteq V$ such that no two vertices in V' share an edge in E.

Given an undirected graph $G = (V, E)$ in which each node has degree $\leq d$, give an efficient algorithm that finds an independent set whose size is at least $1/(d+1)$ times that of the largest independent set. Only the main idea and the proof that the size is at least $1/(d+1)$ times the largest solution's size are needed.

Solution: Initially, let G be the original graph and $I = \emptyset$. Repeat the process below until $G = \emptyset$:

1. Pick an arbitrary node v in G and let $I = I \cup \{v\}$.
2. Delete v and all its neighbors from the graph.
3. Let G be the new graph.

Notice that I is an independent set by construction. At each step, I grows by one vertex and we delete at most $d+1$ vertices from the graph (since v has at most d neighbors). Hence there are at least $|V|/(d+1)$ iterations. Let K be the size of the maximum independent set. Since $K \leq |V|$, we can use the previous argument to get:

$$|I| \geq \frac{|V|}{d+1} \geq \frac{K}{d+1}$$

3 (★★) Modular Arithmetic

(a) Prove that for integers a_1, b_1, a_2, b_2, and n, if $a_1 \equiv b_1 \pmod{n}$ and $a_2 \equiv b_2 \pmod{n}$, then $a_1 \cdot a_2 \equiv b_1 \cdot b_2 \pmod{n}$.

(b) As in the last problem, show that if $a_1 \equiv b_1 \pmod{n}$ and $a_2 \equiv b_2 \pmod{n}$, then $a_1 + a_2 \equiv b_1 + b_2 \pmod{n}$.

(c) What is the last digit (i.e., the least significant digit) of 3^{4001}?

Solution:

(a) By the definition of modular arithmetic, there are integers $k_1, \dots, k_4 \in \{0, \dots, n-1\}$ are r integers such that $a_1 = k_1 + r_1 n$, $b_1 = k_1 + r_1' n$, $a_2 = k_2 + r_2 n$, and $b_2 = k_2 + r_2' n$. This gives us $a_1 \cdot a_2 = k_1(r_2 n) + k_1 k_2 + r_1 n(r_2 n) + r_1 n(k_2) = k_1 k_2 + n(\dots) \equiv k_1 k_2 \pmod{n}$. Likewise $b_1 \cdot b_2 = k_1(r_2' n) + k_1 k_2 + r_1' n(r_2' n) + r_1' n(k_2) = k_1 k_2 + n(\dots) \equiv k_1 k_2 \pmod{n}$. So $a_1 \cdot a_2 \equiv b_1 \cdot b_2 \pmod{n}$.

(b) Using the same k and r values from the previous part, we get $a_1 + a_2 = k_1 + r_1 n + k_2 + r_2 n = k_1 + k_2 + n(r_1 + r_2) \equiv k_1 + k_2 \pmod{n}$. Likewise: $b_1 + b_2 = k_1 + r_1' n + k_2 + r_2' n = k_1 + k_2 + n(r_1' + r_2') \equiv k_1 + k_2 \pmod{n}$. So $a_1 + a_2 \equiv a_1 \cdot a_2 \pmod{n}$.

(c) The last digit of 3^{4001} is the same as the value of $3^{4001} \pmod{10}$. We can find this value through the following computation:

$$3^{4001} \equiv (3^4)^{1000} \cdot 3^1 \equiv (81)^{1000} \cdot 3^1 \equiv 1^{1000} \cdot 3^1 \equiv 3 \pmod{10}$$

4 (★★★★) Wilson's Theorem

Wilson's theorem says that a number N is prime if and only if

$$(N-1)! \equiv -1 \pmod{N}.$$

(a) If p is prime, then we know every number $1 \le x < p$ is invertible modulo p. Which of these numbers are their own inverse?

(b) By pairing up multiplicative inverses, show that $(p-1)! \equiv -1 \pmod{p}$ for prime p.

(c) Show that if N is *not* prime, then $(N-1)! \not\equiv -1 \pmod{N}$. [*Hint:* Consider $d = gcd(N, (N-1)!)$]

(d) Unlike Fermat's Little theorem, Wilson's theorem is an if-and-only-if condition for primality. Why can't we immediately base a primality test on Wilson's theorem?

Solution:

(a) For a number $1 \le n < p$ to be its own inverse modulo p it is necessary to have $n^2 \equiv 1 \mod p$. Equivalently, $n^2 - 1 = (n-1)(n+1) \equiv 0 \mod p$. Since p has no nontrivial factors, $(n-1), (n+1) = 0 \mod p$. Solving for n we get n equals 1 or $p-1 \mod p$. This shows that if a number is its own inverse, the number must be 1 or $p-1$.
1 is clearly its own inverse. $(p-1)^2 = p^2 - 2p + 1 \equiv 1 \mod p$, so $p-1$ is also its own inverse.

(b) Among the $p-1$ numbers, 1 and $p-1$ are their own inverses and the rest have a (different than themselves) unique inverse $\mod p$. Since inversion is a bijective function, each number a in $\{2, \cdots, p-2\}$ is the inverse of some number in the same range not equal to a. Thus,$(p-2)(p-3)\cdots 2 \equiv 1 \mod p \Rightarrow (p-1)! \equiv (p-1) \equiv -1 \mod p$.

(c) If N is not prime, then N can be written as mn for $n, m > 1$. In particular $N > 2$. If $m \neq n$, then $(N-1)!$ contains the product of both m and n, and so $(N-1)! \equiv 0 \not\equiv -1 \pmod{N}$. If $m = n$, then N is a perfect square. If $N = 4$, then $(N-1)! = 3! = 6 \equiv 2 \pmod{4}$. If $N > 4$ then $n \geq 3$, so $N - 1 = n^2 - 1 > 2n$. Hence N divides $(N-1)!$, so $(N-1)! \equiv 0 \pmod{N}$.

Alternate proof: Suppose $(N-1)! \equiv -1 \pmod{N}$. Then $(N-2)! \equiv 1 \pmod{N}$. Then for any $a \in \{1, \dots, N-2\}$, $(N-2)(N-1)\cdots(a+1)(a-1)\cdots 1$ is the inverse of a modulo N. $N-1$ is always its own inverse modulo N. Hence all integers $a \in \{1, \dots, N-1\}$ have inverses modulo N, and so N is prime.

(d) This rule involves calculating a factorial product which takes time exponential in the size of the input. Thus, the algorithm would not be efficient.

5 (★★) Random Prime Generation

Lagrange's prime number theorem states that as N increases, the number of primes less than N is $\Theta(N/\log(N))$. Consider the following algorithm for choosing a random n-bit prime.

- Pick a random n-bit number k.
- Run a primality test on k.
- If it passes the test, output k; else repeat the process.

Show that this algorithm will sample on average $O(n)$ random numbers before hitting a prime. (Hint: If p is the chance of randomly choosing a prime and E is the expected number of random samples, show that $E = 1 + (1-p)E$.)

Solution: Let E be the expected number of times a number will be sampled before a prime is chosen and p be the chance of randomly choosing a prime. After the first sample, there is a probability p chance that a prime was chosen (in which case only one number must have been sampled) and a probability $(1-p)$ chance that a prime wasn't chosen (in which case we can expect $1+E$ numbers to be chosen before a prime is found). This gives us the equation $E = 1 \cdot p + (1-p) \cdot (E+1)$, which we can rearrange to get $E = 1 + (1-p)E$ from the hint.

We can further rearrange this equation to get $E = 1/p$.

Of all n-bit numbers, $\Theta(2^n/\log(2^n)) = \Theta(2^n/n)$ are prime. So the probability p of randomly choosing a prime is $\Theta((2^n/n)/2^n) = \Theta(1/n)$.

Substituting this value of p in our equation for E, we get $E = \Theta(1/(1/n)) = \Theta(n)$, which gives us our solution.

6 (★★★) Streaming Algorithms

In this problem, we assume we are given an infinite stream of integers x_1,x_2, ..., and have to perform some computation after each new integer is given. Since we may see many integers, we want to limit the amount of memory we have to use in total. For all of the parts below, give a brief description of your algorithm and a brief justification of its correctness.

(a) Show that using only a single bit of memory, we can compute whether the sum of all integers seen so far is even or odd.

(b) Show that we can compute whether the sum of all integers seen so far is divisible by some fixed integer N using $O(\log N)$ bits of memory.

(c) Assume N is prime. Give an algorithm to check if N divides the product of all integers seen so far, using as few bits of memory as possible.

(d) Now let N be an arbitrary integer, and suppose we are given its prime factorization: $N = p_1^{k_1} p_2^{k_2} \dots p_r^{k_r}$. Give an algorithm to check whether N divides the product of all integers seen so far, using as few bits of memory as possible. Write down the number of bits your algorithm uses in terms of $k_1, \dots, k_r$.

Solution:

(a) We set our single bit to 1 if and only if the sum of all integers seen so far is odd. This is sufficient since we don't need to store any other information about the integers we've seen so far.

(b) Set $y_0 = 0$. After each new integer x_i, we set $y_i = y_{i-1} + x_i \mod N$. The sum of all seen integers at step i is divisible by N if and only if $y_i \equiv 0 \mod N$. Since each y_i is between 0 and $N-1$, it only takes $\log N$ bits to represent y_i.

(c) We can do this with a single bit b. Initially set $b = 0$. Since N is prime, N can only divide the product of all x_is if there is a specific i such that N divides x_i. After each new x_i, check if N divides x_i. If it does, set $b = 1$. b will equal 1 if and only if N divides the product of all seen integers.

(d) We can do this with $\lceil \log_2(k_1) \rceil + \lceil \log_2(k_2) \rceil + \cdots + \lceil \log_2(k_r) \rceil$ bits. For each i between 1 and r, we track the largest value $t_i \leq k_i$ such that p^{t_i} divides the product of all seen numbers. We start with $t_i = 0$ for all i. When a new number m is seen, we find the largest t_i' such that $p^{t_i'}$ divides m and set $t_i = \min\{t_i' + t_i, k_i\}$ We stop once $t_i = k_i$ for all i as this implies that N divides the product of all seen numbers.