CS 170 HW 11

Due on 2018-04-16, at 11:59 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 $\leq d$, show how to efficiently find an independent set whose size is at least $1/(d + 1)$ times that of the largest independent set.

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

1. Pick the node v with the smallest degree 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) What is the last digit (i.e., the least significant digit) of 3^{4001}?

(b) Prove that for integers a_1, b_1, a_2, b_2, and n, if $a_1 \equiv b_1 \mod n$ and $a_2 \equiv b_2 \mod n$, then $a_1 \cdot a_2 \equiv b_1 \cdot b_2 \mod n$

(c) As in the last problem, show that if $a_1 \equiv b_1 \mod n$ and $a_2 \equiv b_2 \mod n$, then $a_1 + a_2 \equiv b_1 + b_2 \mod n$

(d) Give a polynomial time algorithm for computing $a^{b^c} \mod p$ for prime p and integers a, b, and c.

Solution:

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

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

(b) 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 $a_1 \cdot a_2 = k_1(r_2 n) + k_1 k_2 + r_1 n(r_2 n) + r_1 n(k_2) = k_1 k_2 + n(\dots) \equiv k_1 k_2 \mod n$. Likewise $b_1 \cdot b_2 = k_1(r_2' n) + k_1 k_2 + r_1' n(r_2' n) + r_1' n(k_2) = k_1 k_2 + n(\dots) \equiv k_1 k_2 \mod n$. So $a_1 \cdot a_2 \equiv b_1 \cdot b_2 \mod n$.

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

(d) First, we assume w.l.o.g. that a and b are between 0 and p. Otherwise, we can find $a \mod p$ and $b \mod p$ in time polynomial in $log(a)$, $log(b)$, and $log(p)$.

By Fermat's little theorem, $1 \equiv a^{p-1} \mod p$. This implies that $a^{b^c} \mod p \equiv a^{b^c \mod p-1} \mod p$. We can use the fast exponentiation algorithm from the book to quickly find $b^c \mod p-1$, then use the same algorithm to find $a^{b^c \mod p-1} \mod p$. Since $b^c \mod p-1$ is bounded by p, each of these operations is polynomial in $log(p)$. Since our input is $O(log(p) + log(a) + log(b))$, this implies that the whole algorithm is polynomial.

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 this rule?

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) Assume N is not prime. So N can be written as mn for $n, m > 1$. If $m \neq n$, then $(N-1)!$ contains the product of both m and n, and so $(N-1)! \equiv 0 \not\equiv -1 \mod N$. If $m = n$, then m divides $(N-1)!$, so $N = m^2$ divides $((N-1)!)^2$, so $((N-1)!)^2 \equiv 0 \mod N$. But if $(N-1)! \equiv -1 \mod N$, then $((N-1)!)^2 \equiv 1 \mod N$. So $(N-1)! \not\equiv -1 \pmod N$.

(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 x increases, the number of primes less than x is approximated by $x/(log(x))$. Such abundance makes it simple to generate a random n-bit prime:

- Pick a random n-bit number N.
- Run a primality test on N.
- If it passes the test, output N; 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 average number of coin tosses, show that $E = 1 + (1-p)E$)

Notice that this algorithm is different from other random algorithms we've seen, in that the randomness is in the runtime and not the correctness; It always returns a correct answer, but might take a long time to do so. Algorithms of this form are called *Las Vegas Algorithms*.

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 $E = 1/p$

In sampling an $n - bit$ number, we can expect there to be $2^n/log(2^n) = 2^n/n$ primes. So our chances p of randomly choosing a prime are $(2^n/n)/2^n = 1/n$.

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

6 (★★★★) Quantum Gates

(a) The Hadamard Gate acts on a single qubit and is represented by the following matrix:

$$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$$

Verify that this gate maps the basis states $|0\rangle$ and $|1\rangle$ to a superposition state that will yield 0 and 1 with equal probability, when measured. In other words, explicitly represent the bases as vectors, apply the gate as a matrix multiplication, and explain why the resulting vector will yield 0 and 1 with probabilities 1/2 each, when measured.

(b) Give a matrix representing a *NOT* gate. As in the previous part, explicitly show that applying your gate to the basis state $|0\rangle$ will yield the state $|1\rangle$ (and vice-versa).

(c) Give a matrix representing a gate that swaps two qubits. Explicitly show that applying this matrix to the basis state $|01\rangle$ will yield the state $|10\rangle$. Verify that this matrix is its own inverse.

Solution:

(a) The state $|1\rangle$ can be interpreted as the vector $[0,1]^T$. The results of applying the Hadamard gate to $|1\rangle$ is

$$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}0\\ 1\end{bmatrix} = \begin{bmatrix}\frac{1}{\sqrt{2}}\\ -\frac{1}{\sqrt{2}}\end{bmatrix}$$

When measured, this yields 1 with probability $(\frac{1}{\sqrt{2}})^2 = \frac{1}{2}$ and yields 0 with probability $(\frac{-1}{\sqrt{2}})^2 = \frac{1}{2}$.

Likewise, $|0\rangle$ can be interpreted as the vector $[1,0]^T$, so applying the Hadamard gate gives

$$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}1\\ 0\end{bmatrix} = \begin{bmatrix}\frac{1}{\sqrt{2}}\\ \frac{1}{\sqrt{2}}\end{bmatrix}$$

When measured, this yields 1 and 0 with probabilities $(\frac{1}{\sqrt{2}})^2 = \frac{1}{2}$, each.

(b) The following matrix represents a NOT gate:

$$\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}$$

To see this, we apply it to $|1\rangle$ to get:

$$\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}\begin{bmatrix}0\\ 1\end{bmatrix} = \begin{bmatrix}1\\ 0\end{bmatrix}$$

Likewise, applying it to $|0\rangle$ yields

$$\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}\begin{bmatrix}1\\ 0\end{bmatrix} = \begin{bmatrix}0\\ 1\end{bmatrix}$$

Thus, the gate is, in fact, a NOT gate.

(c) As in the book, We represent a two qubit state by $|\alpha\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle$. This is the same as the vector $[\alpha_{00}, \alpha_{01}, \alpha_{10}, \alpha_{11}]^T$. Swapping the two qubits is equivalent to swapping the middle two values, since swapping two of the same values is unnecessary. This can be done by the following matrix:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Applying this matrix to the basis state $|01\rangle$ yields

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

Simple matrix multiplication will show that the matrix is its own inverse.