Announcements and Outline

Announcements

- Two misstated questions on term test
- Grading scheme for term test 3:
  1. Test will be graded out of 25 with a max of 30 (i.e. up to 20% bonus possible where now everyone has a better chance of getting bonus marks)
  2. Full credit (10 points) for seeing that Q1 was trivial; two points for saying “false” because of clauses containing $x \lor \overline{x}$
  3. Can obtain full credit for interpreting question in terms of approximation ratio.

Today’s outline

- Comments on the nature of the final exam
- Review and finish discussion of RWALK algorithm for 2SAT
- Brief discussion of 1-sided randomized compositeness algorithm
Papadimitriou’s random walk algorithm for 2-SAT

- It is not difficult to show that 2-SAT (determining if a 2CNF formula is satisfiable) is efficiently computable (reducible to directed ST connectivity) whereas we know that 3-SAT is NP complete.

- We will provide a conceptually simple 1-sided randomized algorithm (RWALK) running in time $O(n^2)$ to show that 2-SAT is computationally easy.

- The same basic approach can be used to derive a randomized algorithm (which in turn has led to a deterministic variant of the idea) for 3SAT that runs in time $(1.324)^n$. It is a big open question if one can get time $2^{o(n)}$ algorithm for 3-SAT.

- This random walk idea is the basis for a widely used class of algorithms known as Walk-Sat algorithms for SAT problems.
Random walk algorithm for 2-SAT

**RWALK algorithm for 2CNF formula** $F$

Choose a random or arbitrary truth assignment $\tau$

For $i = 1, 2, \ldots, (c \cdot n^2)$
   - % with a sufficiently large $c$ to obtain any desired probability of success

   If $\tau$ satisfies $F$ then
     - Report success and quit
   Else
     - Let $C$ be an unsatisfied clause and choose one of its literals $\ell_i$ at random
     - Flip the truth value of the literal $\ell_i$ to change $\tau$
   End If

End For

**Claim**

If $f$ is satisfiable, then with say probability at least $\frac{1}{2}$ the RWALK algorithm will succeed in finding a satisfying truth assignment.

- We can either increase $c$ or run RWALK many times to increase the probability of success.
Why RWALK works

Claim
Let $\tau^*$ be a truth assignment satisfying an $n$ variable 2CNF $F$. Then we can view RWALK as a random walk on a line graph (with nodes 1, 2, ..., $n$) that is trying to reach node $n$ where node $i$ indicates that $\tau$ matches $\tau^*$ in $i$ coordinates.

- Since the clause $C$ was not satisfied, at least one of its literals must be set different than $\tau^*$. (It could be that both literals are different.)
- This means that the probability (in terms of the walk on the line) of getting closer to node $n$ is at least $\frac{1}{2}$.
- It can happen that as we are randomly walking, we may come across another satisfying assignment but that will only shorten the time needed.
- What remains to be shown is that a random walk on the line with probability $\frac{1}{2}$ to move left or right will hit every point on the line in expected time $2n^2$.
- More generally, a uniform random walk (starting at any node) on a connected graph $G = (V, E)$ will hit all nodes in expected time $2|E|(|V| - 1)$. 
Randomized Compositeness/Primality Algorithm

One of the most influential randomized algorithms is a polynomial time method for determining if a number is prime/composite.

Quick modern history of primality testing

- Independently Solovay and Strassen, and Rabin (1974) gave two different polynomial time 1-sided error algorithms for determining if an \( n \) digit number \( x \) is prime.

- The algorithm always outputs PRIME if \( x \) is prime and outputs COMPOSITE with probability (say) \( \frac{1}{2} \) if \( x \) is composite.

- That is, the algorithm could error (saying PRIME when \( x \) is composite) with probability at most \( \frac{1}{2} \).
  - This error probability can be reduced by repeated independent trials of the algorithm.
  - That is, \( t \) trials would then yield an error probability at most \( \frac{1}{2^t} \).
The Rabin algorithm is related to deterministic polynomial time algorithm by G. Miller (1976) whose correctness requires the Extended Riemann Hypothesis (ERH), a famous well-believed conjecture in number theory.

Goldwasser and Kilian (1986) gave a polynomial time 0-sided error algorithm.

So why concern ourselves with randomized algorithms when the problem is solved?

- There are polynomials and there are polynomials
- The deterministic (or 0-sided algorithms) are not nearly as practical as the 1-sided algorithms
- These algorithms are an essential ingredient in much of modern cryptography where random primes are often needed.
- Note that while primality testing is theoretically (i.e. in P) and practically solvable, factoring is believed to be NP hard and even hard in some sense of “average case complexity”.
- Complexity based cryptography also depends on the hardness of problems such as factoring integers.
Some basic group theory and number theory

- \( Z_N^* = \{ a \in Z_N \mid gcd(a, N) = 1 \} \) is a commutative group under multiplication (mod \( N \))

- **Lagrange Theorem** If \( H \) is a subgroup of \( G \) then \( order(H) \) divides \( order(G) \).

- **Fermat’s Little Theorem**: If \( N \) is prime then for \( a \not\equiv 0 \) (mod \( N \)), \( a^{N-1} = 1 \) (mod \( N \))

- Furthermore, if \( N \) is prime, then \( Z_N^* \) is a cyclic group; that is, \( \exists g : \{ g, g^2, \ldots, g^{N-1} \} = Z_N \). This implies that for such a generator \( g \), \( g^i \not\equiv 1 \) for \( 1 \leq i < N - 1 \)

- If \( N \) is prime, then \( \pm 1 \mod N \) are precisely two distinct square roots of 1.

- **The Chinese Remainder Theorem**: If \( N_1 \) and \( N_2 \) are relatively prime, then for all \( \nu_1, \nu_2 \), there exists a unique non-negative \( w < N_1 \cdot N_2 \) such that \( w = \nu_1 \) (mod \( N_1 \)) and \( w = \nu_2 \) (mod \( N_2 \))
A simple but not quite correct algorithm

We need two computational facts:

1. \( a^i \pmod{N} \) can be efficiently computed by “repeated squaring mod \( N \)”. 
2. \( \gcd(a, b) \) can be efficiently computed by the Euclidean algorithm.

Simple randomized primality algorithm that “almost works”

- Choose \( a \in \mathbb{Z}_N \setminus \{0\} \) uniformly at random
- If \( \gcd(a, N) \neq 1 \) or \( a^{N-1} \pmod{N} \neq 1 \), then output “COMPOSITE”
- Otherwise output “PRIME”.
When the simple algorithm does (and doesn’t) work

- \( S = \{ a \in Z_N^* \mid a^{N-1} = 1 \pmod{N} \} \) is a subgroup of \( Z_N^* \)

- Hence either \( S = Z_N^* \) if \( S \) is a proper subgroup, or by the Lagrange Theorem, \( |S| \leq \frac{|Z_N^*|}{2} = \frac{N-1}{2} \) if \( S \) is not proper.

- Hence if the simple algorithm finds an \( a \) where \( gcd(a, N) = 1 \) but \( a^{N-1} \neq 1 \pmod{N} \), then \( S \) is proper and therefore at least \( \frac{1}{2} \) half of the elements in \( Z_N \setminus \{0\} \) will be certificates showing \( N \) is not prime.

- Hence the only numbers \( N \) that can defeat the simple algorithm are the Carmichael numbers (also called false primes); i.e. those \( N \) for which \( a^{N-1} = 1 \pmod{N} \) for all \( a \in Z_N^* \).

- It was only relatively recently (1994) when it was proven that there are infinitely many Carmichael numbers.
  - The first three Carmichael numbers are 561, 1105, 1729.
  - There are only 255 Carmichael numbers \( \leq 100,000,000 \).
The Miller-Rabin algorithm

If \( \gcd(a, N) \neq 1 \) then report \textbf{composite} and terminate

% This test isn’t really needed but we add it for clarity

Compute \( t, u \) such that \( N - 1 = (2^t u) \), \( u \) odd and \( t \geq 1 \).

\[ x_0 := 2^u \]

% all computations are mod \( N \)

Randomly choose \( a \in \mathbb{Z}_N \setminus \{0\} \)

For \( i = 1, \ldots, t \)

\[ x_i := x_{i-1}^2 \]

If \( x_i = 1 \) and \( x_{i-1} \notin \{-1, 1\} \) then report \textbf{composite} and terminate

End For

If \( x_t \neq 1 \) then report \textbf{composite} and terminate

Else report \textbf{prime}


- **Claim:** \( \mathbb{P}[\text{algorithm reports prime} \mid N \text{ is composite}] \leq \frac{1}{2} \)
- Proof relies on fact that \( N \) is Carmichael implies \( N = N_1 \cdot N_2 \) with \( \gcd(N_1, N_2) = 1 \)