Recall From Last Time

Theorem: All CFL's are in $\mathbf{sAC}^1$.

Facts: ITADD, MULT, ITMULT and DIVISION on n-bit integers are all in $\mathbf{ThC}^0$.

Th The following problems are complete for

$\mathbf{PSPACE} = \mathbf{NPSPACE} = \mathbf{ATIME}[n^{O(1)}]$:

QSAT, GEOGRAPHY, SUCCINCT REACH.

Barrington's Theorem

A *permutation* of a finite set S is a one-to-one onto function from S to itself, The *composition* of two permutations is the permutation we get by performing first one and then the next. Composition is not commutative – the set of all permutations of five elements form the *non-commutative group* S_5 with composition as the operation.

The S_5 iterated multiplication problem (S_5-MULT) is to input n elements of S_5 (in order) and determine their composition. Clearly a DFA can do this, so S_5-MULT is a regular language.

Theorem 25.1 *S_5-MULT is complete for* **NC**^{1}. *Specifically, if C is an n-input circuit of depth d and fan-in two, we can take a string x of length n and construct a sequence of 4^d permutations that multiplies to a non-identity permutation iff $C(x) = 1$.*

Notation: A *five-cycle* $(abcde)$ is the permutation that takes a to b, b to c, c to d, d to e, and e to a, where {a,b,c,d,e} = {1,2,3,4,5}.

Lemma: There exist five-cycles σ and τ such that $\sigma\tau\sigma^{-1}\tau^{-1}$ is a five-cycle. (This permutation is called the *commutator* of σ and τ.)

Proof: $(12345)(13542)(54321)(24531) = (13254)$.

Fact: (basic group theory) If α and β are both five-cycles, then $\alpha = \gamma\beta\gamma^{-1}$ for some permutation γ.

Proof: (of Barrington's Theorem) Use induction on the depth d of the circuit. For each gate g we'll construct a sequence $s(g)$ such that $s(g)$ evaluates to the five-cycle (12345) if g evaluates to 1 and $s(g)$ evaluates to the identity otherwise. By the Fact, if we can get one five-cycle we can get any other with a sequence of the same length.

Base Case: $d = 0$ and the gate is an input. Look up the literal and let $s(g)$ consist of one permutation, (12345) if the literal is true and the identity if it is false.

NOT Gates: If h is the NOT of g, compose $s(g)$ with (54321). This gives the identity if g is true and (54321) if g is false. Using the Fact, normalize to give (12345) if h is true and the identity if h is false.

AND Gates: Suppose h is the AND of g_1 and g_2 and each of g_1 and g_2 have depth d. Using $s(g_1)$ and $s(g_2)$, we construct four sequences of length 4^d each:

- a_1 yields (12345) if g_1 is true and the identity otherwise,
- a_2 yields (13542) if g_2 is true and the identity otherwise,
- b_1 yields (54321) if g_1 is true and the identity otherwise, and
- b_2 yields (24531) if g_2 is true and the identity otherwise.

Calculation: $a_a a_2 b_1 b_2$ yields (13254) if g_1 and g_2 are both true, and the identity otherwise.

Conclusion: If C is a depth $O(\log n)$ circuit, we get a sequence of length $4^{O(\log n)}$, which is polynomial. We have reduced the circuit evaluation problem to an S_5-MULT instance that is only polynomial size. ♠

Application to PSPACE

Fact: **PSPACE** is characterized by circuits of polynomial *depth*.

Corollary: Any **PSPACE** problem can be reduced to an instance of S_5-MULT of length $2^{n^{O(1)}}$.

Corollary: (Cai-Furst) Any **PSPACE** problem can be solved by a log-space Turing machine that:

- has access to a read-only clock
- wipes its memory every poly-many steps, except for *three safe bits*.

Randomization

We see in an algorithms course that it is sometimes to our advantage to use randomness in solving a problem. For example, Quicksort has good average-case but bad worst-case behavior. Flipping our own coins can (with high probability) keep us out of the bad cases.

In a competitive situation we may be worried about our opponent predicting our move. Flipping our own coins may make this impossible and guarantee us some minimum level of expected success.

Random sampling of a large space may give us a good idea of the results of an impractically large exhaustive search. There is a large body of mathematics telling us what inferences we may reliably make from such sampling.

Is randomness a powerful tool in general? In complexity theory we attack this question by looking at problems where randomization seems practical, and comparing classes of such problems to deterministic and nondeterministic classes.

In a moment, we'll look at *primality testing*, which until recently *was* the most famous example of a problem that could be solved in polynomial time with high probability by a randomized algorithm, but which was not known to be in ¶. This example has been taken from us, however, by Agarwal, Saxena, and Kayal, who last summer gave a deterministic algorithm to test a number for primality in polynomial time.

[P] gives two other interesting randomized algorithms:

- Testing whether a matrix of polynomials has determinant identically zero, and
- Using a random walk through the space of settings to find a satisfying instance of a 2-CNF formula

The Primality Problem

$$\text{PRIME} \quad = \quad \{m \in \mathbf{N} \mid m \text{ is prime}\}$$

Proposition 25.2 $\overline{\text{PRIME}} \in \mathbf{NP}$

Proof:

$$m \in \overline{\text{PRIME}} \;\Leftrightarrow\; m < 2 \quad \vee$$
$$(\exists xy)((1 < x, y < m) \;\wedge\; x \cdot y = m)$$

♠

Question: Is PRIME $\in$ **NP**?

Fact 25.3 (Fermat's Little Theorem): *Let p be prime and $0 < a < p$, then,*

$$a^{p-1} \equiv 1 \pmod{p}$$

$$\mathbf{Z}_n^\star \quad = \quad \{a \in \{1, 2, \ldots, n-1\} \mid \text{GCD}(a, n) = 1\}$$

$Z_n^\star$ is the multiplicative group of integers mod n that are relatively prime to n.

Euler's Phi: $\varphi(n) \quad = \quad |\mathbf{Z}_n^\star|$

Proposition 25.4 *If $n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$ is the prime factorizaton of n, then*

$$\varphi(n) \quad = \quad n(p_1 - 1)(p_2 - 1) \cdots (p_k - 1)/(p_1 p_2 \cdots p_k)$$

Theorem 25.5 *(*Euler's Theorem): *For any n and any $a \in \mathbf{Z}_n^\star$,*

$$a^{\varphi(n)} \equiv 1 \pmod{n}$$

Fact 25.6 *Let $p > 2$ be prime. Then $\mathbf{Z}_p^\star$ is a cyclic group of order $p-1$. That is,*

$$\mathbf{Z}_p^\star \quad = \quad \{a, a^2, a^3, \ldots, a^{p-1}\}$$

$$m \in \text{PRIME} \quad \Leftrightarrow \quad (\exists a \in \mathbf{Z}_m^\star)(\text{ord}(a) = m-1)$$

Theorem 25.7 *[Pratt]*
PRIME $\in$ **NP**.

Proof:
Given m,

1. Guess a, $1 < a < m$
2. Check $a^{m-1} \equiv 1 \pmod{m}$ by repeated squaring.
3. Guess prime factorization,
$$m - 1 \; = \; p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$$
4. Check for $1 \leq i \leq k$,
$$a^{m-1/p_i} \not\equiv 1(\text{mod}\, m)$$
5. Recursively check that $p_1, p_2, \ldots, p_k$ are prime.

$$T(n) \quad = \quad O(n^2) + T(n-1)$$

$$T(n) \quad = \quad O(n^3) \qquad\qquad ♠$$

Corollary 25.8 *PRIME and Factoring are in* **NP**$\cap$*co*-**NP**.

More Primality Testing

Let $a \in \mathbf{Z}_m^\star$

a is a *quadratic residue* mod m iff,

$$(\exists b)(b^2 \equiv a \pmod{m})$$

For p prime let,

$$\left(\frac{a}{p}\right) = \begin{cases} 1 & \text{if } a \text{ is a quadratic residue mod } p \\ -1 & \text{otherwise} \end{cases}$$

Generalize to $\left(\frac{a}{m}\right)$ when m is not prime,

$$\left(\frac{a}{mn}\right) = \left(\frac{a}{m}\right)\left(\frac{a}{n}\right)$$

Fact 25.9 [Gauss]**(Quadratic Reciprocity)** *For odd* a, m,

$$\left(\frac{a}{m}\right) = \begin{cases} \left(\frac{m}{a}\right) & \textit{if } a \equiv 1 \pmod 4 \textit{ or } m \equiv 1 \pmod 4 \\ -\left(\frac{m}{a}\right) & \textit{if } a \equiv 3 \pmod 4 \textit{ and } m \equiv 3 \pmod 4 \end{cases}$$

$$\left(\frac{2}{m}\right) = \begin{cases} 1 & \textit{if } m \equiv 1 \pmod 8 \textit{ or } m \equiv 7 \pmod 8 \\ -1 & \textit{if } m \equiv 3 \pmod 8 \textit{ or } m \equiv 5 \pmod 8 \end{cases}$$

Thus, we can calculate $\left(\frac{a}{m}\right)$ efficiently.

$$\begin{aligned}
\left(\frac{107}{351}\right) &= -\left(\frac{351}{107}\right) \quad = \quad -\left(\frac{30}{107}\right) \\
&= -\left(\frac{2}{107}\right)\left(\frac{15}{107}\right) \quad = \quad -\left(\frac{107}{15}\right) \\
&= -\left(\frac{2}{15}\right) \quad = \quad -1
\end{aligned}$$

$$107 \equiv 351 \equiv 15 \equiv 3 \pmod{4}$$

$$107 \equiv 3 \pmod 8; \qquad 15 \equiv 7 \pmod 8$$

Fact 25.10 [Gauss] *For p prime, $a \in \mathbf{Z}_p^\star$,*

$$\left(\frac{a}{p}\right) \quad \equiv \quad a^{\frac{p-1}{2}} \pmod p$$

Fact 25.11 *If m is not prime then,*

$$\left|\left\{a \in \mathbf{Z}_m^\star \;\mid\; \left(\frac{a}{m}\right) \equiv a^{\frac{m-1}{2}} \pmod m\right\}\right| \;<\; \frac{m-1}{2}$$

Solovay-Strassen Primality Algorithm:

```
1. Input is odd number $m$
2. For $i := 1$ to $k$ **do** {
3.         choose $a < m$ at random
4.         **if** GCD$(a, m) \neq 1$ **return**("not prime")
5.         **if** $\left(\frac{a}{m}\right) \not\equiv a^{\frac{m-1}{2}} \pmod m$ **return**("not prime")
6. }
7. **return**("probably prime")
```

Theorem 25.12
- *If m is prime then Solovay-Strassen(m) returns "probably prime".*
- *If m is not prime, then the probability that Solovay-Strassen(m) returns "probably prime" is less than $1/2^k$.*

Corollary 25.13 PRIME $\in$ *"Truly Feasible"*

The new Agarwal-Saxena-Kayal algorithm is a great theoretical achievement (solving an open problem dating back to at least 1970) but is *not* the best way in practice to test primality. Solovay-Strassen (or the related Miller-Rabin algorithm) runs faster and gives you numbers that are reasonably certain to be prime. If you want a prime of a given size, you generate random numbers until you get one that passes the test many times. There are enough primes so that you can expect this not to take too long.

There is a nice FAQ about this new result at:

```
crypto.cs.mcgill.ca/
~stiglic/PRIMES_P_FAQ.html
```

FACTORING is still believed to be hard for conventional computation. But – there is a 1994 algorithm due to Shor that solves FACTORING in poly time on a *quantum computer*. Only very small quantum computers have so far been built, but one of them has successfully factored the number 15.

What does it mean for a problem to be solvable in ”random polynomial time”? We can define a probabilistic TM easily enough by having an NDTM M flip a coin for each of its classes. There are then *four* different poly-time complexity classes defined in the literature!

We define Prob(M, x) to be the probability that M accepts x. Remember that A is in **NP** if there exists M such that Prob$(M, x) > 0$ iff $x \in A$. The new classes have similar definitions:

- A is in **RP** if there exists M such that Prob$(M, x) \geq 1/2$ for all $x \in A$ and Prob$(M, x) = 0$ for all $x \notin A$.
- A is in **ZPP** if both A and $\overline{A}$ are in **RP**.
- A is in **BPP** if there exists M such that Prob$(M, x) > 2/3$ for all $x \in A$ and Prob$(M, x) < 1/3$ for all $x \notin A$.
- A is in **PP** if there exists M such that Prob$(M, x) > 1/2$ iff $x \in A$.

Amplification

Which of these classes are practical? After all, you don't *want* to put up with a significant probability of a wrong answer.

RP, the class generalizing the Solovay-Strassen primality algorithm, is pretty good. As we've seen, repeated independent trials can reduce our error probability to exponentially small.

ZPP is even better in one sense, because if we try *both* **RP** algorithms repeatedly, in a *constant expected number of trials* we will get a *guaranteed answer*. This is historically called a ”Las Vegas” algorithm (provably correct, probably fast) as opposed to an **RP** or **BPP** ”Monte Carlo” algorithm (probably correct, provably fast).

PP, on the other hand, is *completely useless* in practice. If the probability of acceptance is very very close to $1/2$, the number of trials needed to make a statistical prediction of whether it is over or under 1/2 could be exponential.

But for **BPP** we are in good shape!

Proposition 25.14 *If* $S \in$ **BPP** *then there is a probabilistic, polynomial-time algorithm* A' *such that for all* n *and all inputs* w *of length n,*

$$\textbf{if } (w \in S) \textbf{ then } \text{Prob}(A'(w) = 1) \geq 1 - \frac{1}{2^n}$$

$$\textbf{if } (w \notin S) \textbf{ then } \text{Prob}(A'(w) = 1) \leq \frac{1}{2^n}$$

Proof: Iterate A polynomially many times and answer with the majority. The probability the mean is off by $\frac{1}{3}$ decreases exponentially with n — the formal proof uses Chernoff bounds. ♠

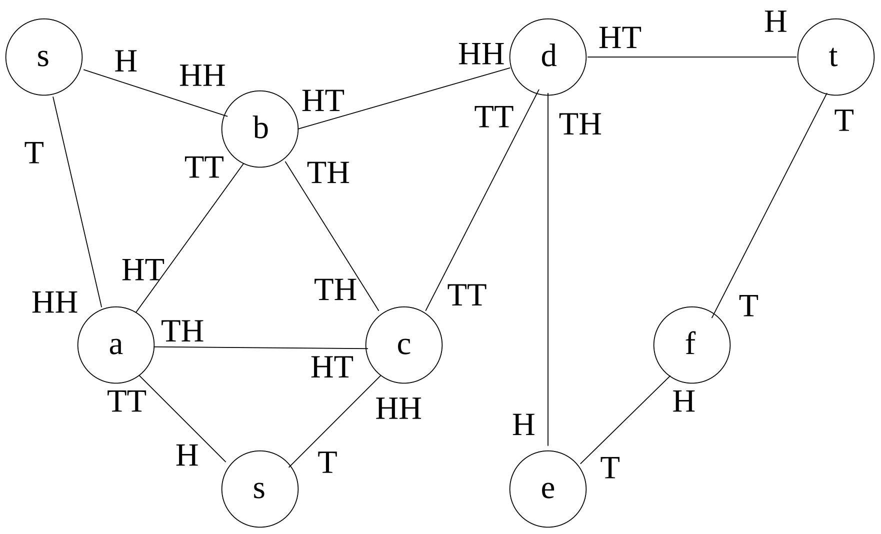
Is **BPP** equal to **P**???

Probably, because pseudo-random number generators are good.

It's probably possible to build a poly-time deterministic generator that gives numbers that are *indistinguishable from random* to any poly-time procedure. Such a generator would allow us to *derandomize* **BPP**.

It's not hard to show that if a language is in **BPP**, it has *non-uniform* poly-size circuits, i.e., it is in the non-uniform class **PSIZE**. This is because if we make the probability small enough that a random string causes the **BPP** algorithm to be wrong on a given input, we can ensure that some string exists that is right on *all inputs*. There *exists* a circuit that has this string hard-wired into it.

$$\mathrm{REACH}_u \quad = \quad \{G, \text{ undirected} \mid s \underset{G}{\overset{\star}{\to}} t\}$$

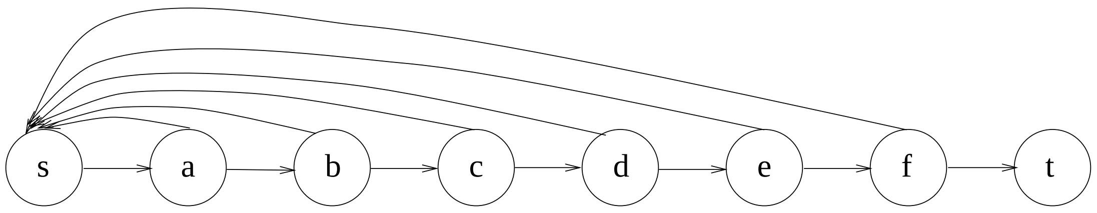


Fact 25.15 *Let $T(i)$ be the expected number of steps in a random walk to visit all vertices in connected graph G, starting from i. Then,*

$$T(i) \ \leq \ 2e(n-1)$$

Corollary 25.16

$$\mathrm{REACH}_u \ \in \ \mathrm{BPL}$$



A look at this *directed* graph should convince you that a random walk on it is *not* likely to reach all vertices in polynomial time. To get to vertex t from s you would have to guess right about n times in a row.

It's very plausible that REACH_u is in **L**, and one might hope to prove it by derandomizing the random walk. (There must exist a single sequence of choices of size $O(n^3)$ that visits every node of *any* undirected labelled n-node graph.) But randomization doesn't seem to help much with the general REACH problem.