- 作业讲解
  - JH第5章练习5.2.2.7、5.2.2.8

JH第5章练习5.2.2.7

- communication complexity
  - $2\log_2(n^c/\ln n^c) <= 2\log_2 n^c = 2c\log_2 n$
- Prob(accepts)
  - $>= 1-\ln n^c/n^{c-1}$

JH第5章练习5.2.2.8

- 先假设Alice和Bob共享一个长为n的随机串

Returning to the previous example of *EQ*, if certainty is not required, Alice and Bob can check for equality using only *O(log n)* messages. Consider the following protocol: Assume that Alice and Bob both have access to the same random string $z \in \{0,1\}^n$. Alice computes $z \cdot x$ and sends this bit (call it *b*) to Bob. (The $(\cdot)$ is the dot product in GF(2).) Then Bob compares *b* to $z \cdot y$. If they are the same, then Bob accepts, saying *x* equals *y*. Otherwise, he rejects.

Clearly, if $x = y$, then $z \cdot x = z \cdot y$, so $Prob_z[Accept] = 1$. If *x* does not equal *y*, it is still possible that $z \cdot x = z \cdot y$, which would give Bob the wrong answer. How does this happen?

If *x* and *y* are not equal, they must differ in some locations:

$$x = c_1 c_2 \ldots p \ldots p' \ldots x_n$$
$$y = c_1 c_2 \ldots q \ldots q' \ldots y_n$$
$$z = z_1 z_2 \ldots z_i \ldots z_j \ldots z_n$$

Where x and y agree, $z_i * x_i = z_i * c_i = z_i * y_i$ so those terms affect the dot products equally. We can safely ignore those terms and look only at where x and y differ. Furthermore, we can swap the bits x_i and y_i without changing whether or not the dot products are equal. This means we can swap bits so that x contains only zeros and y contains only ones:

$$x' = 00 \ldots 0$$
$$y' = 11 \ldots 1$$
$$z' = z_1 z_2 \ldots z_{n'}$$

Note that $z' \cdot x' = 0$ and $z' \cdot y' = \Sigma_i z'_i$. Now, the question becomes: for some random string z', what is the probability that $\Sigma_i z'_i = 0$? Since each z'_i is equally likely to be 0 or 1, this probability is just $1/2$. Thus, when x does not equal y, $Prob_z[Accept] = 1/2$. The algorithm can be repeated many times to increase its accuracy. This fits the requirements for a randomized communication algorithm.

JH第5章练习5.2.2.8 (续)

- 显然这一假设很难成立
  - Alice生成随机串后，传给Bob也需要代价
- 替代方案
  - 双方共享充分多的（cn个）随机串
  - Alice随机选一个，将其序号（长logn）传给Bob
  - 可证满足two-sided-error Monte Carlo的要求 (https://en.wikipedia.org/wiki/Communication_complexity)

- 教材讨论
  - JH第5章第3节第4小节

问题1：NEQ-POL

- NEQ-POL是解决什么问题的polynomial-time one-sided-error Monte Carlo算法？
- 你能解释这个算法的基本思路吗？
- 为什么是polynomial-time one-sided-error Monte Carlo？

Algorithm 5.3.4.4. NEQ-POL

Input:	Two polynomials $p_1(x_1, \ldots, x_m)$ and $p_2(x_1, \ldots, x_m)$ over $\mathbb{Z}_n$ with at most degree d, where n is a prime and $n > 2dm$.
Step 1:	Choose uniformly $a_1, a_2, \ldots, a_m \in \mathbb{Z}_n$ at random.
Step 2:	Evaluate $I := p_1(a_1, a_2, \ldots, a_m) - p_2(a_1, a_2, \ldots, a_m)$.
Step 3:	**if** $I \neq 0$ **then output**$(p_1 \not\equiv p_2)$ {accept} **else output**$(p_1 \equiv p_2)$ {reject}.

问题1： NEQ-POL (续)

Theorem 5.3.4.5. *Algorithm* NEQ-POL *is a polynomial time one-sided-error Monte Carlo algorithm that decides the nonequivalence of two polynomials.*

Proof. Since the only computation part of NEQ-POL is the evaluation of a polynomial in Step 2, it is obvious that NEQ-POL is a polynomial-time algorithm.

If $p_1 \equiv p_2$, then $p_1(a_1, \ldots, a_m) = p_2(a_1, \ldots, a_m)$ for all $a_1, a_2, \ldots, a_m$ from $\mathbb{Z}_n$ and so $I = 0$. So,

$$Prob(\text{NEQ-POL rejects } (p_1, p_2)) = 1.$$

If $p_1 \not\equiv p_2$, then $p_1(x_1, \ldots, x_m) - p_2(x_1, \ldots, x_m)$ is a nonzero polynomial. Following Lemma 5.3.4.2 and the fact $n \geq 2dm$,

$$Prob\,(p_1(a_1, a_2, \ldots, a_m) - p_2(a_1, a_2, \ldots, a_m) = 0) \underset{\text{Lemma 5.3.4.2}}{\leq} m \cdot d/n \leq \frac{1}{2}.$$

Thus,

$$Prob(\text{NEQ-POL accepts } (p_1, p_2)) =$$

$$Prob(p_1(a_1, \ldots, a_m) - p_2(a_1, \ldots, a_m) \neq 0) \geq 1 - \frac{m \cdot d}{n} \geq \frac{1}{2}.$$

□

问题1：NEQ-POL (续)

- 作为一种fingerprinting策略，NEQ-POL中的fingerprint是什么？
- 为什么说NEQ-POL也是一个two-sided-error Monte Carlo算法？

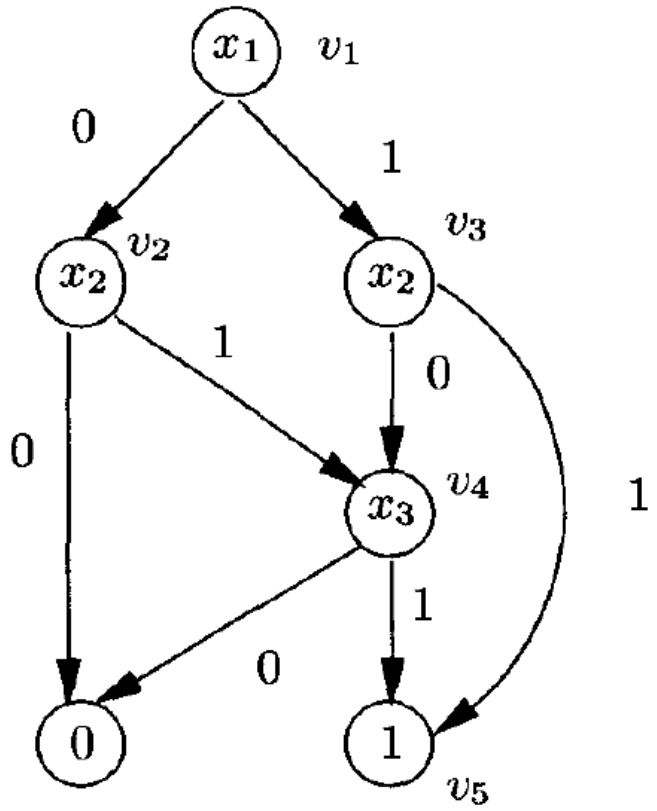
问题2：NEQ-1BP

- NEQ-1BP是解决什么问题的polynomial-time one-sided-error Monte Carlo算法？
  你理解这个问题了吗？
- 你能解释这个算法的基本思路吗？
  特别地：多项式是如何构造的？

Algorithm 5.3.4.9. NEQ-1BP

Input:	Two 1BPs A and B over the set of variables $\{x_1, x_2, \ldots, x_m\}$, $m \in \mathbb{N}$.
Step 1:	Construct the polynomials p_A and p_B.
Step 2:	Apply the algorithm NEQ-POL on $p_A(x_1, \ldots, x_m)$ and $p_B(x_1, \ldots, x_m)$ over some $\mathbb{Z}_n$, where n is a prime that is larger than $2m$.
Output:	The output of NEQ-POL.

问题2： NEQ-1BP (续)



问题2： NEQ-1BP (续)

- 为什么是polynomial-time one-sided-error Monte Carlo？

Lemma 5.3.4.8. *For every two* 1BP*s* A *and* B,
A *and* B *are equivalent if and only if* p_A *and* p_B *are identical.*

Proof. To see this we transform every polynomial of degree at most 1 into a special "normal" form similar to DNF for the representation of Boolean functions.[59] This normal form is the sum of "elementary multiplications" $y_1 y_2 \dots y_m$, where either $y_i = x_i$ or $y_i = (1 - x_i)$ for every $i = 1, 2, \dots, m$. Obviously, two polynomials of degree 1 are equivalent if and only if they have their normal forms identical. Moreover, every elementary multiplication of this normal form corresponds to one input assignment on which the corresponding 1BP computes "1". So, A and B are equivalent if and only if the normal forms of p_A and p_B are identical.

It remains to show that one can unambiguously assign the normal form to every polynomial[60] p_A of degree 1. First, one applies the distributive rules to get a sum of elementary multiplications. If an elementary multiplication y does not contain a variable x, then we exchange y by two elementary multiplications $x \cdot y$ and $(1 - x) \cdot y$. Obviously, an iterative application of this rule results in the normal form. □

Theorem 5.3.4.10. NEQ-1BP *is a polynomial-time one-sided-error Monte Carlo algorithm for the problem of nonequivalence of two* 1BP*s.*

Proof. The construction of p_A and p_B in Step 1 can be done in a time that is quadratic in the input size (representation of 1BPs). Since NEQ-POL works in polynomial time and the sizes of p_A and p_B (as inputs of NEQ-POL) are polynomial in the size of the input of NEQ-1BP, Step 2 is also done in polynomial time.

Due to Lemma 5.3.4.8 we know that A and B are equivalent if and only if p_A and p_B are equivalent. If A and B are equivalent (i.e., when p_A and p_B are equivalent), then NEQ-POL rejects (p_A, p_B) with a probability of 1, i.e., we have no error on this side. If A and B are not equivalent, then NEQ-POL accepts (p_A, p_B) with the probability of at least

$$1 - m/n.$$

Since $n > 2m$, this probability is at least $1/2$. □