- 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？

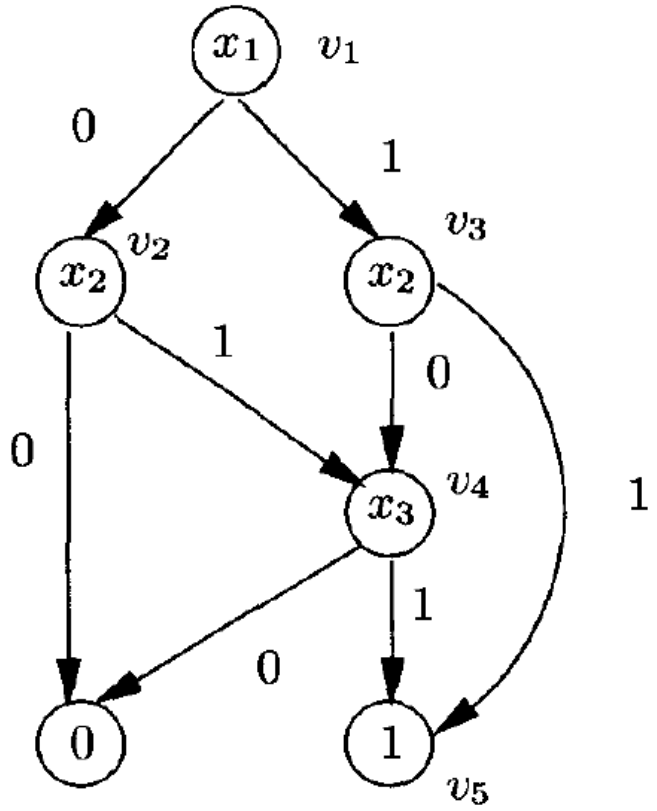
问题2：NEQ-1BP

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

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)

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 $1/2$. □

问题3：重访Rabin-Karp

- 你还记得Rabin-Karp算法吗？

Given a pattern $P[1..m]$, let p denote its corresponding decimal value.

We can compute p in time $\Theta(m)$ using Horner's rule (see Section 30.1):

$$p = P[m] + 10\,(P[m-1] + 10(P[m-2] + \cdots + 10(P[2] + 10P[1])\cdots))\,.$$

Similarly, we can compute t_0 from $T[1..m]$ in time $\Theta(m)$.

To compute the remaining values $t_1, t_2, \ldots, t_{n-m}$ in time $\Theta(n-m)$, we observe that we can compute t_{s+1} from t_s in constant time, since

$$t_{s+1} = 10(t_s - 10^{m-1}T[s+1]) + T[s+m+1]\,. \qquad (32.1)$$

...

In general, with a d-ary alphabet $\{0, 1, \ldots, d-1\}$, we choose q so that dq fits within a computer word and adjust the recurrence equation (32.1) to work modulo q, so that it becomes

$$t_{s+1} = (d(t_s - T[s+1]h) + T[s+m+1]) \bmod q\,, \qquad (32.2)$$

where $h \equiv d^{m-1} \pmod q$ is the value of the digit "1" in the high-order position of an m-digit text window.

```
RABIN-KARP-MATCHER(T, P, d, q)
 1  n = T.length
 2  m = P.length
 3  h = d^(m-1) mod q
 4  p = 0
 5  t_0 = 0
 6  for i = 1 to m                    // preprocessing
 7      p = (dp + P[i]) mod q
 8      t_0 = (dt_0 + T[i]) mod q
 9  for s = 0 to n - m                // matching
10      if p == t_s
11          if P[1..m] == T[s+1..s+m]
12              print "Pattern occurs with shift" s
13      if s < n - m
14          t_{s+1} = (d(t_s - T[s+1]h) + T[s+m+1]) mod q
```

问题3：重访Rabin-Karp (续)

- 你能将Rabin-Karp改造为Las Vegas吗？
- 你能将Rabin-Karp改造为Monte Carlo吗？
  - 是one-sided-error？还是two-sided-error？
- 改造之后，新算法比Rabin-Karp更快吗？

```
RABIN-KARP-MATCHER(T, P, d, q)
 1  n = T.length
 2  m = P.length
 3  h = d^(m−1) mod q
 4  p = 0
 5  t_0 = 0
 6  for i = 1 to m                    // preprocessing
 7      p = (dp + P[i]) mod q
 8      t_0 = (dt_0 + T[i]) mod q
 9  for s = 0 to n − m                // matching
10      if p == t_s
11          if P[1..m] == T[s + 1..s + m]
12              print "Pattern occurs with shift" s
13      if s < n − m
14          t_{s+1} = (d(t_s − T[s + 1]h) + T[s + m + 1]) mod q
```