Föreläsning 14: Probabilistiska algoritmer

Probabilistiska beslutsalgoritmer

Ex: Är polynomet

$f(x,y) = (x-3y)(xy-5x)^2 - 10x^3y + 25x^3 + 3x^2y^3 + 30x^2y^2 - 75x^2y = 0$ (identiskt?)

Test: Välj några värden x_i, y_i slumpmässigt och testa om $f(x_i, y_i) = 0$.

2 möjligheter

1. $f(x_i, y_i) \neq 0$. Då är $f \neq 0$.

2. $f(x_i, y_i) = 0$ för alla valda x_i, y_i. Vad är då sannolikheten för att $f = 0$?

Sats: Om $f(x_1, ..., x_m)$ inte är identiskt lika med 0 och varje variabel har grad högst d och M är ett heltal så är antalet nollställen i mängden $\{0, 1, ..., M-1\}^m$ högst mdM^{m-1}. Det ger:

$P[$ Slumpmässigt värde i $\{0, 1, ..., M-1\}^m$ är nollställe$] = \frac{1}{mdM^{m-1}} = \delta$.

Det ger att om vi har gjort k tester som visar att $f = 0$ så är $P[f = 0] \geq (1-\delta)^k$.

Monte Carlo-algoritmer

Antag att vi har ett språk L som vi vill kunna avgöra. En **Nej-baserad Monte Carlo-algoritm** F är en algoritm som är *polynomiell* i indatas storlek och som uppfyller:

1. Om $x \in L$ så är $F(x) = Ja$ med sannolikhet $> \frac{1}{2}$.

2. Om $\notin L$ så är $F(x) = Nej$ med sannolikhet 1.

Definition: Klassen RP består av alla språk (problem) som avgörs av en Nej-baserad Monte Carlo-algoritm. Det går att se att:

$P \subseteq RP \subseteq NP$.

Las Vegas-algoritmer

Givet ett språk L är en **Las Vegas-algoritm** en polynomiell algoritm som svarar Ja, Nej eller ? enligt följande:

1. $x \in L \Rightarrow F(x) = Ja$ eller ?.

2. $x \notin L \Rightarrow F(x) = Nej$ eller ?.

3. För alla x gäller att $F(x) =?$ med sannolikhet $< \frac{1}{2}$.

Om man kör en Las Vegas-algoritm k gånger är sannolikheten att man har fått ett definitivt och korrekt svar $> 1 - \frac{1}{2^k}$.

Språk (problem) som avgörs av en Las Vegas-algoritm bildar en klass som kallas ZPP.

En probabilistisk approximationsalgoritm

MAX-CNF-SAT: Givet en uppsättning $c_1, c_2, ..., c_m$ av 3-klausuler vill vi hitta en variabeltilldelning som satisfierar ett maximalt antal klausuler.
Problemet har uppenbart en motsvarande beslusvariant som är **NP**-fullständig.

Approximationsalgoritm: För varje x_i sätter vi $x_i \leftarrow Random(0,1)$. Kontrollera hur många klausuler som får värdet 1. Antalet blir APP. Värdet på APP kan variera mellan olika körningar av algoritmen. Vi vill ha en övre gräns för $\frac{OPT}{APP}$ som gäller **i medeltal**.

$P[c_i$ är satisfierad $] = \frac{7}{8}$

$E(APP) = \sum_i P[c_i$ är satisfierad $] = \sum_i \frac{7}{8} = \frac{7m}{8}$.

Eftersom $m \geq OPT$ fås $E(APP) \geq \frac{7}{8}OPT$. Så i medeltal är approximationskvoten $\leq \frac{8}{7}$.

Primtalstest

RSA bygger på att man kan hitta stora primtal (200 siffror). Vanliga sättet att hitta dem bygger på att man väljer ett udda tal slumpmässigt och testar om det är ett primtal.

Fermats sats: Om p är ett primtal och a är ett tal sådant att $a \nmid n$ gäller $a^{p-1} \equiv 1 (\mod p)$.

Speciellt kan vi välja $a = 2$. Om vi har ett tal n så att $2^{n-1} \not\equiv 1 (\mod n)$ så kan inte p vara ett primtal. Det ger följande enkla algoritm för att testa om n är ett primtal:

```
FERMAT(n)
(1)   k ← MOD-EXP(2, n − 1, n)
(2)   if k ≢ 1( mod n)
(3)       return FALSE
(4)   return TRUE
```

Om FERMAT returnerar FALSE vet vi säkert att n inte är primtal. Men FERMAT kan tyvärr returnera TRUE även om n inte är primtal. T.ex. är $2^{340} \equiv 1 \mod 341$ trots att 341 inte är primtal.

Vad man kan göra är att använda en s.k. **probabilistisk** algoritm som slumpmässigt väljer ett tal a i intervallet $[2, n-1]$ och gör ett Fermattest med a som bas.

```
PROB-FERMAT(n, s)
(1)   for j ← 1 to s
(2)       a ← RANDOM(2, n − 1)
(3)       k ← MOD-EXP(a, n − 1, n)
(4)       if k ≢ 1( mod n)
(5)           return FALSE
(6)   return TRUE
```

Algoritmen är probabilistisk på det sättet att den kan ge olika svar vid olika körningar även om man har samma tal n som indata. Men följande gäller:

Om n är ett primtal måste algoritmen returnera TRUE.. Om algoritmen returnerar FALSE så vet vi att n inte är primtal. FALSE är därför det enda säkra svar vi kan få.

$P(n$ är inte primtal $\mid$ Algoritmen svarar FALSE $) = 1$

Vad gäller för den betingade sannolikheten $P(n$ är primtal $\mid$ Algoritmen svarar TRUE $)$?
Det visar sig att för nästan alla n som inte är primtal gäller det att
$P($ Algoritmen svarar FALSE $) > \frac{1}{2}$.
(För primtal n gäller förstås $P($ Algoritmen svarar TRUE $) = 1$.) Problem ges av s.k. **Carmichaeltal**.

Carmichaeltal: Ett tal n som inte är primtal men som uppfyller $a^{n-1} \equiv 1$ (mod n) för alla $a \in [2, n-1]$.
$P($ Algoritmen svarar TRUE $|n$ är Carmichaeltal $) = 1$.

För att hantera sådana specialfall kan man istället använda följande algoritm.Den är tänkt att returnera FALSE **om n är primtal** och TRUE om n är sammansatt tal.

```
WITNESS(a, n)
(1)     Låt n − 1 = 2^t u, t ≥ 1, u udda
(2)     x_0 ← MOD-EXP(a, u, n)
(3)     for i ← 1 to t
(4)         x_i ← x_{i−1}^2  mod n
(5)         if x_i = 1 och x_{i−1} ≠ 1 och x_{i−1} ≠ n − 1
(6)             return TRUE
(7)     if x_t ≠ 1
(8)         return TRUE
(9)     return FALSE
```

För WITNESS gäller
$P($ Algoritmen svarar TRUE $|n$ är inte primtal $) > \frac{1}{2}$ för alla tal n. Genom att göra upprepade test med WITNESS kan man få godtycklig sannolikhet för korrekt svar. Algoritmen kallas för Miller- Rabins test.

```
MILLER-RABIN(n, s)
(1)     for j ← 1 to s
(2)         a ← RANDOM(1, n − 1)
(3)         if WITNESS(a, n)
(4)             return Inte primtal
(5)     return Primtal
```

Här gäller
$P($ Algoritmen svarar Primtal $|n$ är primtal$) = 1$.
$P($ Algoritmen svarar Inte primtal $|n$ är inte primtal $) > 1 - \frac{1}{2^s}$.

Vi kan också betrakta de ”omvända” betingade sannolikheterna:
$P(n$ är inte primtal $\mid$ Algoritmen svarar Inte primtal $) = 1$.

Sannolikheten $P(n$ är primtal | Algoritmen svarar Primtal) är knepigare. Formellt skulle den kunna beräknas som $P(n$ är primtal och algoritmen svarar primtal) $/P($ Algoritmen svarar Primtal). Det krävs dock att man känner till *a priori* fördelningen för sannolikheten att n är primtal. Om man antar att denna sannolikhet är p kan man med hjälp av Bayes lag uppskatta att den sökta sannolikheten är $> \frac{2^s}{2^s+(\frac{1}{p}-1)}$.

Det är sedan augusti 2002 känt att det finns effektiva deterministiska algoritmer som avgör om ett tal är primtal. De är betydligt mer komplicerade än t.ex. Miller-Rabins algoritm.