

Mer avancerad enumeration

Cayleys sats

Antalet *numrerade* träd med n noder är n^{n-2} .

Detta verkar inte gå att visa med direkt konstruktion.

Två bevis:

- Bijektion till antal funktioner (enligt Cameron).
- Med hjälp av Prüferkod.

Bijektion till funktioner

Antalet funktioner

$$f : \{1, 2, \dots, n\} \rightarrow \{1, 2, \dots, n\} = n^n.$$

Vi representerar sådan funktioner med träd. Kräver flera steg:

1. Vi kan representera f med en *riktad* graf. Om $f(i) = j$ så går det en kant från i till j .
2. Vissa av talen (minst ett) kommer att ingå i cykler. Kalla mängden av dessa tal för C . Talen i C är tal i så att $f^k(i) = i$ för något k .
3. Om $i \notin C$ finns ett första tal j så att $f^k(i) = j$ och $j \in C$. Vi kallar j för i :s C -representant.
4. Den del av grafen som innehåller noderna med j som C -representant bildar ett träd med j som rot. Riktningen på kanterna kan strykas. Det är underförstått att riktningen går mot j .
5. (Den svåra delen.) Delen av grafen som består av C görs om. Den består först av en union av cykler. Denna cykelmängd kan tolkas som en *permutation*. Antag att c_1, c_2, \dots, c_k är talen i C sorterade i storleksordning. Låt oss anta att $f(c_i) = a_i$. Då kan vi beskriva f :s verkan på C med permutationen

$$\begin{pmatrix} c_1 & c_2 & \cdots & c_k \\ a_1 & a_2 & \cdots & a_k \end{pmatrix}$$

eller helt enkelt med sekvensen $(a_1 a_2 \dots a_k)$. Denna sekvens är en ordning av talen i C . Ersätt delgrafen över C med en oriktad stig som går i denna ordning (kant mellan a_i och a_{i+1} o.s.v.). Grafen utgör nu ett oriktat träd.

6. Vi representerar nu funktionen genom att presentera trädet från steg 5 och genom att dessutom ange de två noderna $U = a_1$ och $V = a_k$. Ur denna information går det att rekonstruera C , permutationen på C och därefter hela f .

Vi tänker nu i andra riktningen. Vi antar att vi har ett träd T och två noder U och V angivna. Vi kan då konstruera en funktion som representeras av trädet. Vi låter C bestå av noderna på stigen mellan U och V . Ordningen längs stigen läser vi av som permutationen på C . I varje nod k i C sitter ett (ev tomt) delträd påhängt. Detta delträd anger talen som har k som C representant. Om i och j är grannar i delträdet och j ligger närmst k sätter vi $f(i) = j$.

Detta ger en bijektion mellan funktioner och mängden av numrerade träd med två noder utmärkta (med ordning). Om $T(n)$ är antalet numrerade träd får vi $n^n = n^2 T(n)$

d.v.s.

$$T(n) = n^{n-2}.$$

Prüferkod

Vi kan representera ett numrerat träd med en lista av $n-2$ st heltal i intervallet $[1, n]$. Eftersom det finns n^{n-2} sådana sekvenser ger det satsen.

Vi har ett numrerat träd T . Vi konstruerar en lista L med följande algoritm:

```
PRÜFER( $T$ )
(1)    $L \leftarrow []$ 
(2)   while  $|T| > 2$ 
(3)       Låt  $l$  vara det minsta lövet i  $T$ 
(4)       Låt  $a$  vara  $l$ :s granne
(5)       Sätt in  $a$  sist i  $L$ 
(6)        $T \leftarrow T - \{l\}$ 
(7)   return  $L$ 
```

Det går att köra denna algoritm baklänges:

Starta med en lista $L = [a_1, a_2, \dots, a_{n-2}]$

```
BAKÅT-PRÜFER( $L$ )
(1)   Lägg till  $a_{n-1} = n$  sist i  $L$ 
(2)    $T \leftarrow \emptyset$ 
(3)   for  $i=1$  to  $n-1$ 
(4)       Låt  $b_i$  vara minsta talet
(5)       som inte ligger i
(6)        $[a_i, \dots, a_{n-1}] \cup [b_1, \dots, b_{i-1}]$ 
(7)        $T \leftarrow T \cup (b_i, a_i)$ 
(8)   return  $T$ 
```

Kommentarer kring Bakåt-Prüfer

Bevis för att Bakåt-Prüfer genererar ett träd:

Påstående: Varje tal i $[1, \dots, n]$ får minst en kant.

Bevis: Alla b -talen är olika. De är $n-1$ st. Talet n dyker inte upp bland b -talen. Alla b -tal får en kant. I sista steget har vi mängden $[n] \cup [b_1, \dots, b_{n-2}]$. Kanten (b_{n-1}, n) kommer också med och alla noder får därför en kant.

T är alltså en graf med n noder där alla noder har en kant och det finns $n-1$ kanter. T måste vara ett träd.

Bevis för att Prüfer och Bakåt-Prüfer är inverser till varandra:

(Skiss)

Påstående: Antag att T ger P-kod $[a_1, \dots, a_{n-2}]$. Låt T' vara T med det första lövet och dess kant borttaget. Då har T' P-kod $[a_2, \dots, a_{n-2}]$.

Påstående: Antag att L ger trädets T med Bakåt-Prüfer. Tag bort första talet a_1 ur L . Det ger L' som genererar trädets T' där (b_1, a_1) är borttagna.

Vi antar att P och B-P är inverser till varandra för träd av storlek $n-1$.

Det går att se att om P verkande på T tar bort (b_1, a_1) först så måste B-P verkande på L lägga till (b_1, a_1) först.

Skapa nu T' och L' enligt ovanstående påstående. Induktionsantagandet är nu att P och B-P är inverser till varandra på T', L' . Då är de inverser på T, L också. Induktion visar att de är inverser för alla träd.

Uppräkning av kombinatoriska objekt

Problem:

Vi vill skapa samtliga kombinatorisk objekt av en viss typ.

T.ex. kan vi vilja räkna upp alla permutationer av $[1, \dots, n]$ för att skicka som indata till ett sorteringsprogram som skall testas.

Vi ställer krav på en sådan uppräkning.

De viktigaste kraven

- Vi vill inte skapa en lista över objekten. Minnesutrymmet räcker inte!
- Vi vill generera objekten ett i taget och sedan "radera ut det".
- Vi vill ha en enkel regel som givet ett objekt genererar nästa.
- Varje objekt skall räknas upp exakt en gång.

För enklare typer av objekt går dessa krav oftast att uppfylla.

Uppräkning av delmängder

Vi vill räkna upp alla delmängder av $\{1, 2, \dots, n\}$.

En metod är följande:

Tolka varje mängd binärt.

T.ex. tolkas delmängden $\{1, 2, 6\}$ av

$\{1, 2, 3, 4, 5, 6, 7\}$ som 010011 (obs: baklänges).

Vi använder följande algoritm:

- Starta med $M_1 = \emptyset$. (Tolkas som $00 \dots 0$).
- Givet M_i fås M_{i+1} genom att addera 1 till M_i 's binära form.
- När vi nått $M_{2^n} =$ (Hela mängden) stannar vi.

I Cameron ges samma algoritm utan direkt användande av den binära formen.

Uppräkning av k-delmängder

En naturlig uppräkningsordning är *lexikografisk* ordning.

T.ex $n = 6, k = 3$ ger ordningen:

123, 124, 125, 126, 134, 135, 136, 145, 146, 156,

234, 235, 236, 245, 246, 256, 345, 346, 356, 456

En annan ordning som inte först verkar lika naturlig är *omvänd lexikografisk ordning*:

123, 124, 134, 234, 125, 135, 235, 145, 245, 345,

126, 136, 236, 146, 246, 346, 156, 256, 356, 456

Denna ordning kan beskrivas som att man "Vänder på mängderna. Sorterar dem baklänges. Vänder på mängderna. Vänder på sorteringen."

Ordningen har fördelen att mängderna som innehåller 6 kommer sist. Om man vill lägga till ett element kan man "fortsätta" på den tidigare ordningen.

Algoritmer

Båda dessa sorteringar kan beskrivas algoritmiskt. Det vi behöver är en regel som givet delmängd D_i skapar delmängd D_{i+1} .

Lexikografisk:

Låt $D_i = x_1x_2 \cdots x_k$. Låt s vara det största index så att $x_s + 1 \notin D_i$. Då är $D_{i+1} = x_1x_2 \cdots x_{s-1}(x_s + 1)(x_s + 2) \cdots (x_s + k - s + 1)$.

Omvänd lexikografisk:

Låt $D_i = x_1x_2 \cdots x_k$. Låt s vara det minsta index så att $x_s + 1 \notin D_i$. Då är $D_{i+1} = 123 \cdots (s - 2)(s - 1)(x_s + 1)x_{s+1} \cdots x_k$.

Mer avancerad uppräknig

Vi tittar på uppräknig av alla delmängder av $\{1, 2, \dots, n\}$ igen.

Med avståndet mellan två mängder A, B menar vi storleken av mängden $(A - B) \cup (B - A)$. Avståndet mellan A, B är ett om den ena mängden kan fås ur den andra genom att lägga till eller dra ifrån ett element.

Kan vi hitta en uppräknig M_1, M_2, \dots, M_{2^n} så att alla avstånd M_i, M_{i+1} är ett?

Lösning

Följande algoritm ger en sådan ordning:

Vi startar med $M_1 = \emptyset$ och slutar med $M_{2^n} = \{n\}$

Vi antar att M_i är beskriven på binär form som $b_1b_2 \cdots b_n$. Med \bar{b}_k menar vi b_k med omvänt värde. ($b_k = 0$ ger $\bar{b}_k = 1$ o.s.v.)

Om $b_1 + b_2 + \cdots + b_n$ är jämn sätter vi $M_{i+1} = \bar{b}_1b_2 \cdots b_n$.

Om $b_1 + b_2 + \cdots + b_n$ är udda bestämmer vi ett index k (unik) så att $b_k = 1$ och $b_j = 0$ för $j < k$. Sätt $M_{i+1} = b_1b_2 \cdots b_k\bar{b}_{k+1} \cdots b_n$.

Gray-koder

Ett bevis för att algoritmen fungerar är att se att den ger Gray-koden för familjen av delmängder. (Se kap 11.6 i Cameron).

En Gray-kod är en lösning till problemet att hitta uppräknig med avstånd ett. Gray-koder byggs upp rekursivt.

Låt G_{n-1} vara Gray-koden för delmängderna till $\{1, \dots, n-1\}$. Gray-koden G_n fås genom att ta de 2^{n-1} binära orden i G_{n-1} med en nolla sist. Vänd sedan på G_{n-1} och lägg till en etta sist. Tillsammans ger det G_n med 2^n binära ord.

Permutationer

Enklaste sättet att räkna upp permutationer av $(1, 2, \dots, n)$ är lexikografisk ordning. Ges av följande algoritm:

Starta med $(1, 2, \dots, n)$ och sluta med

$(n, n-1, \dots, 1)$.

Antag att $P_i = (x_1, x_2, \dots, x_n)$. Bestäm det största j så att $x_j < x_{j+1}$. Bestäm sedan det minsta $k > j$ så att $x_j < x_k$. Sätt

$P_{i+1} = (x_1, x_2, \dots, x_{j-i}, x_k, x_{j+1}, \dots, x_{k-1}, x_j, x_n, x_{n-1}, \dots, x_{k+1})$.

Mer om delbarhet

Problem: Avgör om $\binom{n}{k}$ är delbart med primtal p .

Från tidigare föreläsning:

$$f(m) = \lfloor \frac{m}{p} \rfloor + \lfloor \frac{m}{p^2} \rfloor + \dots + \lfloor \frac{m}{p^s} \rfloor$$

anger högsta p -potensen som delar m !

p delar $\binom{n}{k}$ om och endast om

$$f(n) - f(n-k) - f(k) > 0.$$

Ex: Visa att *binompk* alltid är delbart med p om $k < p$.

$$f(p) - f(p-k) - f(k) = 1 - 0 - 0 > 0.$$

Tillämpning:

$$(1+x)^p \equiv (1+x^p) \pmod{p}$$

Lucas sats

Följande sats ger betydligt mer information:

Antag att $n = a_0 + a_1p + a_2p^2 + \dots + a_sp^s$

och $k = b_0 + b_1p + b_2p^2 + \dots + b_sp^s$

och $0 \leq a_i, b_i < p$.

(Det betyder att vi har n och k angivna i talbasen p . Det är möjligt att $a_s = 1$ och $b_s = 0$.)

Vi vill beräkna $\binom{n}{k} \pmod{p}$.

Svaret är att $\binom{n}{k} \equiv \prod_{i=0}^s \binom{a_i}{b_i} \pmod{p}$

Bevis för Lucas sats

Det är en elegant tillämpning av potensserier.

Vi visar att om $n = a_1p + a_0$ och $k = b_1p + b_0$ så är

$$\binom{n}{k} \equiv \binom{a_1}{b_1} \binom{a_0}{b_0} \pmod{p}$$

$$(1+x)^n = (1+x)^{a_1p} (1+x)^{a_0}.$$

Nu använder vi att $(1+x)^p \equiv 1+x^p \pmod{p}$.

$$(1+x)^n \equiv (1+x^p)^{a_1} (1+x)^{a_0} = 1 + \dots + c_k x^k + \dots + x^n.$$

Vad blir c_k ? Vi vet att $k = b_1p + b_0$. Om vi tittar på produkten ser vi att c_k måste ha formen $\binom{a_1}{i} \binom{a_0}{j}$ där i, j är sådana att $ip + j = k$. Det ger att $c_k = \binom{a_1}{b_1} \binom{a_0}{b_0}$.

Men i vänsterledet får vi $\binom{n}{k}$ som x^k -koefficient.

Det visar att

$$\binom{n}{k} \equiv \binom{a_1}{b_1} \binom{a_0}{b_0} \pmod{p}$$

Beviset fungerar även om $a_1, b_1 > p$. Vi kan sätta ersätta a_1 med $a_1 + a_2p + \dots + a_sp^{s-1}$ och b_1 med $b_1 + b_2p + \dots + b_sp^{s-1}$.

Med induktion får vi fullständiga formen av Lucas sats.