DD2458, Problemlösning och programmering under press

Föreläsning 10: Enumerativ Kombinatorik

Datum: 2007-11-20
Skribent(er): Tobias Kruseborn och Martin Thörne
Föreläsare: Per Austrin

Notation: $[n] = \{1, 2, \ldots n\}$

1 Delmängder

$\binom{n}{k}$ - antalet delmängder av $[n]$ av storleken k. Antalet sätt att välja ut en icke-ordnad delmängd med k element ur en mängd med n element kallas binomialtal.

$$\binom{n}{k} = \frac{n(n-1)(n-2)\cdots(n-k+1)}{k(k-1)(k-2)\cdots 1}$$

Totala antalet delmängder av $[n]$ är $2^n = \sum_{k=0}^{n} \binom{n}{k}$
Ett effektivt sätt att tabellera talen $\binom{n}{k}$ för $n \geq 0$ och $k \geq 0$ är i Pascals triangel:

$$\begin{array}{ccccccc}
\binom{0}{0} & & & & & & \\
\binom{1}{0} & \binom{1}{1} & & & & & \\
\binom{2}{0} & \binom{2}{1} & \binom{2}{2} & & & & \\
\binom{3}{0} & \binom{3}{1} & \binom{3}{2} & \binom{3}{3} & & & \\
\binom{4}{0} & \binom{4}{1} & \binom{4}{2} & \binom{4}{3} & \binom{4}{4} & &
\end{array}
\qquad
\begin{array}{ccccccc}
 & & & 1 & & & \\
 & & 1 & & 1 & & \\
 & 1 & & 2 & & 1 & \\
1 & & 3 & & 3 & & 1 \\
 & 4 & & 6 & & 4 & \\
1 & & & & & & 1
\end{array}$$

Observera att triangelns sidor består av endast ettor och att varje annat tal fås som summan av de två talen närmast ovanför i triangeln. Denna egenskap kan beskrivas som en rekursion för binomialtalen:

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

De två olika sätten att beräkna binomialtal – den direkta formeln och rekursionsekvationen – ger två olika naturliga implementationer:

```
BINOMIAL(n, k)
(1)   r ← 1
(2)   for i = 1 to k
(3)       r ← r(n − k + i)/i
(4)   return r
```

```
BINOMIAL(n, k)
(1)   if k < 0 or k > n
(2)      return 0
(3)   else if n = k or k = 0
(4)      return 1
(5)   else if b[n][k] ej beräknad
(6)      b[n][k] ← BINOMIAL(n − 1, k − 1) + BINOMIAL(n −
         1, k)
(7)   return b[n][k]
```

$b[][]$ är en global array där vi memoiserar värdena av de beräknade binomialtalen. Den första implementationen körs på linjär tid men kan dock få overflow. Den rekursiva implementationen kommer aldrig få overflow eftersom den aldrig hanterar tal som är större än det tal man vill beräkna. Den körs för övrigt på kvadratisk tid.

1.1 Multinomialtal

Multinomialtal är antal sätt att dela upp $[n]$ i k st delmängder, där den i:te har storleken n_i. Vi skriver multinomialtal som $\binom{n}{n_1,n_2,\ldots,n_k} = \frac{n!}{n_1!n_2!\cdots n_k!}$ där $\sum n_i = n$. Observera att binomialtalet $\binom{n}{k}$ är lika med multinomialtalet $\binom{n}{k,n-k}$

Formeln för beräkning av multinomialtal:

$$\binom{n}{n_1 \ldots n_k} = \sum_{i=1}^{k} \binom{n-1}{n_1, n_2, \ldots, n_i - 1, n_k} = \frac{n!}{n_1! \cdot n_2! \cdots n_k!}$$

Exempel:

$$\binom{5}{2,1,2} = \binom{4}{1,1,2} + \binom{4}{2,2} + \binom{4}{2,2,1} = \frac{5!}{2! \cdot 1! \cdot 2!}$$

2 Permutationer

En ordning av element i en mängd kallas för en permutation av elementen i mängden.

Ex. 1: *skriv upp alla permutationer av elementen i mängden* {1,2,3}:

Lösning: 123, 132, 213, 231, 312, 321. Det fanns alltså sex stycken permutationer av en mängd med tre element.

Allmänt: Låt $x = x_1x_2 \cdots x_n$ vara en sträng av längden n. Räkna ut hur många permutationer av x det finns.

I specialfallet att alla tecken är olika i mängden finns det $n!$ olika permutationer. Vi låter f_i = antal ggr i förekommer i strängen x.

Exempel: $x = cbaac$ (alla tecken är ej olika i mängden). Då får vi $f_a = 2, f_b = 1, f_c = 2$. Det totala antalet permutationer ges av

$$\binom{n}{f_a, f_b, \ldots}.$$

I vårt exempel ger detta: $\binom{n}{f_a,f_b,f_c} = \binom{5}{2,1,2} = 30$

Följande formel kan användas för att beräkna antalet permutationer som börjar på tecknet i:

$$\binom{n-1}{f_a, f_b, \ldots, f_i - 1, \ldots}$$

För $i = a, b$ och c får vi:

$a : \binom{4}{1,1,2} = 12,\ b : \binom{4}{2,2} = 6,\ c : \binom{4}{2,1,1} = 12$

Följande algoritm kan användas för att generera dessa permutationer.

PERMUTATION(x)
(1) hitta kortaste suffix $x_i \ldots x_n$ sådant att $x_i < x_{i+1}$, dvs $x_i < x_{i+1} \geq x_{i+2} \geq \ldots \geq x_n$
(2) Hitta största j sådant att $x_j > x_i$
(3) swap(x_i, x_j)
(4) reverse($x_i \ldots x_n$)

Exempel 1: Vi låter en sträng vara: '$bcfecb$'. Då får vi $i = 2, j = 4$. Kör vi swap får vi '$befccb$'. Och sedan efter reverse får vi: '$bebccf$'.

Exempel 2: Vi låter en sträng vara '$dcba$'. Här existerar inga suffix eftersom '$dcba$' är den sista permutationen.

I c++ (stl) finns det två olika implementationer för permutationer: prev_permutation och next_permutation.

Tillämpning: Generera alla $\binom{n}{k}$ delmängder av $[n]$ av storlek k. Detta är samma sak som att generera alla permutationer av en sträng med n st '0' och k st '1', där vi tar med element i, om tecknet på position i är en etta. Att ta nästa permutation blir då detsamma som att konstruera nästa delmängd av samma storlek, och vi har därmed ett sätt att generera alla delmängder av en viss storlek.

3 Partitioner

$B = \{B_1...B_k\}$ är en partition av $[n]$ i k st block om:

1. $\cup_{i=1}^{k} B_i = [n]$
2. $B_i \bigcap B_j = \emptyset$
3. $B_i \neq \emptyset$

Exempel på en partition av [8] i 4 block: $B = \{\{1,5,7\},\{2,8\},\{6\},\{3,4\}\}$

Hur många partitioner av $[n]$ finns det i k block? Antalet sätt att göra detta går mycket enkelt att formulera rekursivt:

- Att lägga n olika element i ett block går endast till på ett sätt dvs. lägga alla i ett och samma block.
- Att dela upp n olika element i n block går också endast till på ett sätt; de får bilda var sitt block.
- Om vi ska dela upp n element i k block kan vi antingen dela upp $n-1$ st av de första elementen i $k-1$ st block, och låta det sista elementet bilda ett eget block; eller dela upp de $n-1$ st första elementen i k st block och slänga det sista elementet på någon av de första blocken.

Det här ger upphov till följande rekursionsekvation:

1. $S(n,1) = S(n,n) = 1$
2. $S(n,k) = S(n-1,k-1) + k \cdot S(n-1,k)$ om $1 < k < n$.
3. $S(n,k) = 0$ annars

Dessa kallas Stirlingtal av andra ordningen. Stirlingtal kan beskrivas på på flera sätt, men inget är enklare än den rekursiva formeln. De första stirlingtalen är:

	1	2	3	4	5
1	1				
2	1	1			
3	1	3	1		
4	1	7	6	1	
5	1	15	25	10	1

Talet i rad n och kolumn k är $S(n,k)$.

4 Catalantalen

Catalantalen kan användas för att räkna antalet balanserade parentesuttryck för ett antal parentespar. Om vi har 3 parentespar kan vi skriva ett balanserat parentesuttryck på följande sätt:

$$< ()()(), ()(()), ((())), (()()), (())() >$$

Med andra ord har vi $P(3) = 5$.

Det här är en rekursionformel

$$P(n) = \sum_{k=0}^{n-1} P(k) \cdot P(n-k-1)$$

Där $P(0) = 1$ är basfallet
Formeln fås om man tänker sig att man har en rotparantes. I den parentesen finns det k paranteser och till höger om rotparantesen finns det $n-k-1$ parenteser.
Catalantalen kan även användas för att räkna antalet binära träd för ett antal noder. Om vi har 3 noder så kan vi rita upp följande binära träd:

Körtiden för beräkning är kvadratisk.
En bättre rekursionsformel för beräkning av catalantal

$$P(n) = \frac{\binom{2n}{n}}{n+1} = \frac{2 \cdot (2n-1) \cdot P(n-1)}{n+1}$$

Körtiden för denna beräkning är linjär.

5 Matrismetoden

Matrismetoden är en metod för att beräkna linjära rekursionsekvationer snabbt.
Ex. Fibonaccitalen

$$f_n = f_{n-1} + f_{n-2}$$

$$Basfallen : f_0 = 0, f_1 = 1$$

Uträkningen av fibonaccitalen är enkel då man endast behöver stega framåt och hålla de 2 senaste talen i minnet. Beräkning görs i linjär tid. Vi ska nu se hur man kan göra det ännu snabbare

Allmänt gäller för en linjär rekursionsekvation att

$$x_n = \sum_{i=1}^{k} a_i \cdot x_{n-i}$$

Där värdena $x_1 = b_1, x_2 = b_2, \ldots, x_k = b_k$ är kända. Beräkning av x_n:

Vi vill konstruera en matris A så att

$$A \cdot \begin{pmatrix} x_{n-1} \\ x_{n-2} \\ \cdot \\ \cdot \\ \cdot \\ x_{n-k} \end{pmatrix} = \begin{pmatrix} x_n \\ x_{n-1} \\ \cdot \\ \cdot \\ \cdot \\ x_{n-k+1} \end{pmatrix}$$

$$A = \begin{pmatrix} a_1 & a_2 & . & . & a_k \\ 1 & 0 & 0 & . & 0 \\ 0 & 1 & 0 & . & 0 \\ . & . & . & . & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Vi får då att:

$$\begin{pmatrix} x_n \\ x_{n-1} \\ . \\ . \\ . \\ x_{n-k+1} \end{pmatrix} = A^i \cdot \begin{pmatrix} x_{n-i} \\ x_{n-i-1} \\ . \\ . \\ . \\ x_{n-i-k+1} \end{pmatrix} = A^{n-k} \cdot \begin{pmatrix} b_k \\ . \\ . \\ . \\ b_1 \end{pmatrix}$$

Vi får då tidskomplexiteten vid naiv matrismultiplikation till $O(k^3 \lg n)$

6 Inklusion-Exklusion

Vi vill beräkna storleken på en union där elementen överlappar varandra.

$$\mid A_1 \cup A_2 \cup ... \cup A_n \mid$$

Ex.

$$\mid A_1 \cup A_2 \cup A_3 \mid = \mid A_1 \mid + \mid A_2 \mid + \mid A_3 \mid - \mid A_1 \cap A_2 \mid - \mid A_2 \cap A_3 \mid - \mid A_1 \cap A_3 \mid + \mid A_1 \cap A_2 \cap A_3 \mid$$

Allmänt så gäller att

$$\mid A_1 \cup A_2 \cup ... \cup A_n \mid = \sum_{\emptyset \neq I \subseteq [n]} (-1)^{|I|-1} \cdot \mid \bigcap_{i \in I} A_i \mid$$

Applicerbart om

- Det är enkelt att beräkna snittet och n litet säg, n mindre än 20
- Storleken av snittet bara beror på $\mid I \mid$, säg $\mid \bigcap_{i \in I} A_i \mid = F(\mid I \mid)$
  Vi får då:
  $\mid A_1 \cup A_2 \cup ... \cup A_n \mid = \sum_{k=1}^{n} \binom{n}{k} (-1)^{k-1} F(k)$

Ett exempel då inklusion-exklusion kan användas är derangements. Derangements är permutationer där inget av elementen befinner sig på sin ursprungliga plats. Detta kan jämföras med hattproblemet. Om alla som går till fest har med sig varsin hatt. Därefter när de lämnar festen får de med sig en slumpvald hatt. Vad är då sannolikheten att alla får en främmande hatt med sig hem. En approximation visar att denna sannolikhet är ungefär $\frac{1}{e}$.