3.7. Подмножества множеств

3.7.1. Порождение подмножеств счетом в двоичной системе счисления

Порождение подмножеств множества $\{a_1, a_2, \ldots, a_n\}$ эквивалентно порождению n-разрядных двоичных наборов: a_i принадлежит подмножеству, если и только если i-й разряд равен единице. Таким образом, задача порождения всех подмножеств множества сводится к задаче порождения всех возможных двоичных последовательностей длины n, а задача порождения случайного подмножества сводится к задаче порождения случайного двоичного набора длины n (т. е. случайного числа i, $0 \leq i < 2^n$).

Ясно, что наиболее прямым способом порождения всех двоичных наборов длины n является счет в двоичной системе счисления (алгоритм 5).

$$\textbf{for } i \leftarrow 0 \textbf{ to } n \textbf{ do } b_i \leftarrow 0$$

$$\textbf{while } b_n \neq 1 \textbf{ do } \begin{cases} \text{Вывести } (b_{n-1}, b_{n-2}, \ldots, b_0) \\ i \leftarrow 0 \\ \textbf{while } b_i = 1 \textbf{ do } \begin{cases} b_i \leftarrow 0 \\ i \leftarrow i+1 \end{cases} \\ b_i \leftarrow 1 \end{cases}$$

Алгоритм 5. Счет в двоичной системе счисления

Перевод этого алгоритма на язык подмножеств множества $\{a_1, a_2, \ldots, a_n\}$ осуществляется очевидным образом. Добавляем фиктивный элемент a_{n+1} и получаем алгоритм 6.

$$
\begin{array}{l}
S \leftarrow \varnothing \\
\textbf{while } a_{n+1} \notin S \textbf{ do } \begin{cases} \text{Вывести } S \\ i \leftarrow 1 \\ \textbf{while } a_i \in S \textbf{ do } \begin{cases} S \leftarrow S - \{a_i\} \\ i \leftarrow i + 1 \end{cases} \\ S \leftarrow S \cup \{a_i\} \end{cases}
\end{array}
$$

Алгоритм 6. Порождение подмножеств счетом в двоичной системе счисления

Данный алгоритм линеен, так как проверка условия «$a_i \in S$» осуществляется один раз для каждого двоичного набора, еще раз для каждого второго набора, еще раз для каждого четвертого набора и т. д. или $2^n + 2^{n-1} + \ldots + 2 + 1 = 2^{n+1} - 1$ раз. Однако алгоритм не является алгоритмом минимального изменения и порядок, в котором порождаются подмножества, не имеет каких-либо специальных свойств, которые говорят в его пользу (естественный лексикографический порядок получается применением алгоритма поиска с возвратом).

Более эффективный алгоритм минимального изменения основан на порождении двоично-отраженного кода Грея. Теоретические основы получения двоично-отраженного кода Грея и детали реализации соответствующих алгоритмов можно посмотреть в [21].

3.7.2. k-подмножества (сочетания из n по k)

Сочетанием из n по k называется набор из k элементов, выбранных из множества, содержащего n различных элементов. Наборы, отличающиеся только порядком следования элементов (но не составом), считаются одинаковыми.

Число сочетаний вычисляется по формуле:

$$C_n^k = \binom{n}{k} = \frac{n!}{k!(n-k)!}.$$

Рассмотрим порождение всех сочетаний из n по k в лексикографическом порядке. Как обычно, делаем упрощающее предположение, что основным множеством является множество натуральных чисел $\{1, 2, \ldots, n\}$. Наиболее естественным является возрастающий лексикографический порядок. Например, $C_6^3 = 20$ сочетаний из 6 по 3, т. е. трехэлементные подмножества множества $\{1, 2, 3, 4, 5, 6\}$, записываются в лексикографическом порядке следующим образом:

123	134	146	236	345
124	135	156	245	346
125	136	234	246	356
126	145	235	256	456

Без детального рассмотрения математических аспектов разберем процесс порождения сочетаний. Начиная с сочетания $\{1, 2, \ldots, k\}$, следующее сочетание находится просмотром текущего сочетания справа налево с тем, чтобы определить позицию самого правого элемента, который еще не достиг своего максимального значения. Этот элемент увеличивается на единицу, и всем элементам справа от него присваиваются новые наименьшие возможные значения. Детали реализации показаны в алгоритме 7.

$$
\begin{array}{l}
c_0 \leftarrow -1 \\
\textbf{for } i \leftarrow 1 \textbf{ to } k \textbf{ do } c_i \leftarrow i \\
j \leftarrow 1 \\
\textbf{while } j \neq 0 \textbf{ do } \begin{cases} \text{Вывести } (c_1, c_2, \ldots, c_k) \\ j \leftarrow k \\ \textbf{while } c_j = n - k + j \textbf{ do } j \leftarrow j - 1 \\ c_j \leftarrow c_j + 1 \\ \textbf{for } i \leftarrow j + 1 \textbf{ to } k \textbf{ do } c_i \leftarrow c_{i-1} + 1 \end{cases}
\end{array}
$$

Алгоритм 7. Порождение сочетаний в лексикографическом порядке

Общую производительность алгоритма можно достаточно легко определить, если известно, сколько раз выполняется сравнение «$c_j = n - k + j$». Это сравнение осуществляется один раз для каждого сочетания (т. е. C_n^k раз), еще раз для каждого сочетания с $c_k = n$ (т. е. C_{n-1}^{k-1}), еще по разу для каждого сочетания с $c_{k-1} = n-1$ и $c_k = n$ (т. е. C_{n-2}^{k-2}), … и еще по одному разу для каждого сочетания с $c_1 = n-k+1, c_2 = n-k+2, \ldots, c_k = n$ (т. е. $C_{n-k}^{k-k} = 1$ раз). Таким образом, оно осуществляется

$$\sum_{j=0}^{k} C_{n-j}^{k-j} = C_{n+1}^{k}$$

раз. Зная это, легко видеть, что присваивание «$c_i \leftarrow c_{i-1} + 1$» выполняется $C_{n+1}^{k} - C_n^k = C_n^{k-1}$ раз. Сравнение «$j \neq 0$» выполняется $C_n^k + 1$ раз, и, таким образом, поведение алгоритма полностью определено.

Поскольку $C_{n+1}^{k} = C_{n}^{k}\dfrac{n+1}{n-k+1}$ и $C_{n}^{k-1} = C_{n}^{k}\dfrac{k}{n-k+1}$, алгоритм линеен, за исключением тех случаев, когда $k = n - o(n)$. В этом случае алгоритм можно применять для порождения сочетаний из n элементов по $n - k$; эти сочетания затем можно дополнить для получения сочетаний из n по k.

Наименьшим изменением, возможным в последовательных сочетаниях в списке всех сочетаний из n по k, является замена одного элемента другим. В терминах двоичных наборов это означает, что необходимо получить все n-разрядные наборы, содержащие точно k единиц, так что последовательные наборы различаются ровно в двух разрядах (в одном нуль заменяется единицей, в другом единица нулем). Порядок минимального изменения основан на двоично-отраженном n-разрядном коде Грея. Математические аспекты и детали реализации соответствующего алгоритма можно посмотреть в [21].

3.7.3. *Случайные сочетания из n по k*

Для порождения случайного подмножества R множества $\{a_1, a_2, \ldots, a_n\}$ с k элементами выбирается один из n элементов случайно и затем выбирается случайное $(k-1)$-подмножество из оставшихся $n-1$ элементов.

Алгоритм можно реализовать, используя дополнительный массив P для хранения номеров еще не выбранных элементов. После того, как выбраны первые $j-1 \le k$ элементов, $p_j, p_{j+1}, \ldots, p_n$ (см. алгоритм 8) будут записью $n-j+1$ элементов, которые еще не выбраны.

$$\begin{array}{l}
\textbf{for}\ \ j \leftarrow 1\ \ \textbf{to}\ \ n\ \ \textbf{do}\ \ p_j \leftarrow j \\
R \leftarrow \varnothing \\
\textbf{for}\ \ j \leftarrow 1\ \ \textbf{to}\ \ k\ \ \textbf{do}\ \ \begin{cases} r \leftarrow rand(j,n) \\ R \leftarrow R \cup \{a_{p_r}\} \\ p_r \leftarrow p_j \end{cases}
\end{array}$$

Алгоритм 8. Порождение случайного сочетания из n по k

Если порядок множества сохранять не обязательно, то массив P можно исключить, используя алгоритм, аналогичный алгоритму для случайных перестановок (см. алгоритм 4), который дает случайные перестановки на первых k позициях множества. Такая модификация представлена алгоритмом 9.

$$\textbf{for}\ \ j \leftarrow 1\ \ \textbf{to}\ \ k\ \ \textbf{do}\ \begin{cases} i \leftarrow rand(j, n) \\ a_i \leftrightarrow a_j \end{cases}$$

Алгоритм 9. Порождение случайного сочетания из n по k без массива P