Описание алгоритмов к домашнему заданию № 4

Алгоритм построения бинарного кода Грея

Описанный далее алгоритм генерирует последовательность всех подмножеств *n*-элементного множества таким образом, что каждое последующее множество получается из предыдущего добавлением или удалением одного элемента.

Код Грея называется так же отраженным кодом. Рассмотрим построение кода на примере *n= 4.*

Будем считать старшим разрядом нулевой разряд. Он может принимать значения 0 и 1.

0000

0001 Далее старший разряд первый, который принимает значения 1, а младший разряд (нулевой) принимает значения в обратном порядке от предыдущего

0011

0010 Далее аналогичным образом: (старший разряд выделен размером, отражаемая часть жирным шрифтом)

0110
0111
0101
0100
1100
1101
1111
1110
1010
1011
1001

Пронумеруем полученные наборы от 0 до 14.

0	0000	**7**	0100
1	0001	**8**	1100
2	0011	**9**	1101
3	0010	**10**	1111
4	0110	**11**	1110
5	0111	**12**	1010
6	0101	**13**	1011
		14	1001

В первом наборе инвертировался разряд с номером 0, во втором – разряд с номером 1, в третьем – разряд с номером 0, в четвертом разряд с номером 2 и т.д.. Разложим числа от 1 до 14 на простые множители и подсчитаем количество двоек в разложении числа. В итоге получена та же самая последовательность.

Число	Разложение	Количество двоек в разложении числа	Число	Разложение	Количество двоек в разложении числа
1	1	0	8	2*2*2*2	3
2	2	1	9	3*3	0
3	3	0	10	2*5	1
4	2*2	2	11	11	0
5	5	0	12	2*2*3	2
6	2*3	1	13	13	0
7	7	0	14	2*7	1

1. Задать A – множество из n элементов.
2. Задать M = [000..]- подмножество булеана.
3. Вывести M;
4. Цикл $\left(i = \overline{1;2^n - 1}\right)$
   4.1. Найти k – количество двоек в разложении числа i ;
   4.2. Если $M[k] = 0$ То $\{M[k] = 1$
   Иначе $M[k] = 0$;
   4.3. Вывести M;
5. Конец цикла
6. Конец

Реализация кода Грея с помощью стека.

1. СТЕК ← пустой стек
2. Цикл ($j = \overline{n-1;0}$)
   2.1. $g_j = 0$
   2.2. СТЕК ← j
3. Конец цикла
4. Печать $(g_{n-1}, g_{n-2}, \ldots, g_0)$
5. Пока (СТЕК не пуст);
   5.1. СТЕК → a
   5.2. $g_a := \overline{g_a}$ {Инвертировать g_a }

5.3.Печать $(g_{n-1}, g_{n-2},\ldots,g_0)$

5.4. Цикл ($j = \overline{a-1;0}$)

5.4.1. СТЕК $\leftarrow j$

5.5. Конец цикла

6. Конец цикла

«Алгоритмы порождения комбинаторных объектов»

Генерация сочетаний

Генерация сочетаний в лексикографическом порядке

Будем рассматривать в качестве множества $X = \{1,2,\ldots,n\}$. Требуется сгенерировать все подмножества мощности $k, (0 \le k \le n)$ множества *X*.

Определим отношение лексикографического порядка ($\prec$) следующим образом. Пусть $a = (a_1, a_2,\ldots,a_n)$, $b = (b_1, b_2,\ldots,b_m)$. Будем говорить, что набор *a* предшествует набору *b*: $a \prec b \Leftrightarrow \exists r \ge 1 : a_r < b_r$ и $\forall i = \overline{1, r-1};\ a_i = b_i$.

Будем рассматривать сочетания *k* элементов из множества *X* как вектор $(c_1, c_2,\ldots,c_k)$, компоненты которого расположены в порядке возрастания слева направо (т.е. $c_i < c_{i+1}$ для любого *i*). Начиная с сочетания $(1,2,\ldots,k)$, следующие будем строить, просматривая текущее справа налево, чтобы найти самый первый элемент, не достигший максимального значения; этот элемент увеличим на единицу, а всем элементам справа от него присвоим номинальные наименьшие значения.

Лексикографический порядок порождения сочетаний не является алгоритмом с минимальными изменениями.

1. $c_0 := -1$
2. Цикл ($i := \overline{1,k}$)
   2.1. $c_i := i$
3. Конец цикла
4. $j := 1$
5. Пока ($j \ne 0$)
   5.1.Печать $(c_1, c_2,\ldots,c_k)$

5.2. $j := k$

5.3. Пока ($c_j = n - k + j$)

5.3.1 $j := j - 1$

5.4. Конец цикла

5.5. $c_j := c_j + 1$

5.6.Цикл $(i := \overline{j+1, k})$

5.6.1. $c_i := c_{i-1} + 1$

5.7. Конец цикла

6. Конец цикла

7. Конец

Генерация сочетаний с помощью кодов Грея

При генерации сочетаний из n элементов по k наименьшим возможным изменением при переходе от текущего сочетания к следующему является замена одного элемента другим. В терминах Грея это означает, что мы хотим выписать все n-разрядные кодовые слова, содержащие ровно k единиц, причем последовательные наборы отличаются ровно в двух разрядах (в одном из разрядов 0 заменяется на 1, а в другом — 1 на 0).

Пусть $G(n)$ — двоично-отраженный код Грея, а $G(n,k)\ (0 \le k \le n)$ — последовательность кодовых слов ровно с k единицами:

$$G(n,k)^T = \left(G(n,k)_1, G(n,k)_2, \ldots, G(n,k)_{C_n^k}\right)^T.$$

Эту последовательность можно рекурсивно определить следующим образом:

$$G(n,0) = (0\,0\ldots0);$$

$$G(n,n) = (1\,1\ldots1);$$

$$G(n,k) = \begin{pmatrix} 0 & G(n-1,k) \\ 1 & \overline{G(n-1,k-1)} \end{pmatrix}, \qquad (1.1)$$

где 0 — вектор-столбец размерности $C_{n-1}^k \times 1$, состоящий из нулей;

1 — вектор-столбец размерности $C_{n-1}^{k-1} \times 1$, состоящий из единиц;

$G(n-1,k)$ — матрица $C_{n-1}^k \times (n-1)$ кодовых слов, содержащих ровно k единиц;

$\overline{G(n-1,k-1)}$ — матрица $C_{n-1}^{k-1}\times(n-1)$ кодовых слов, содержащих ровно $k-1$ единиц, причем кодовые слова записаны в порядке, обратном порядку $G(n-1,k-1)$ ($\overline{G}$ — «перевернутая» матрица G).

На рис. 1.1 приведен пример построения кодовых слов Грея для генерации сочетаний из 4 элементов по 2.

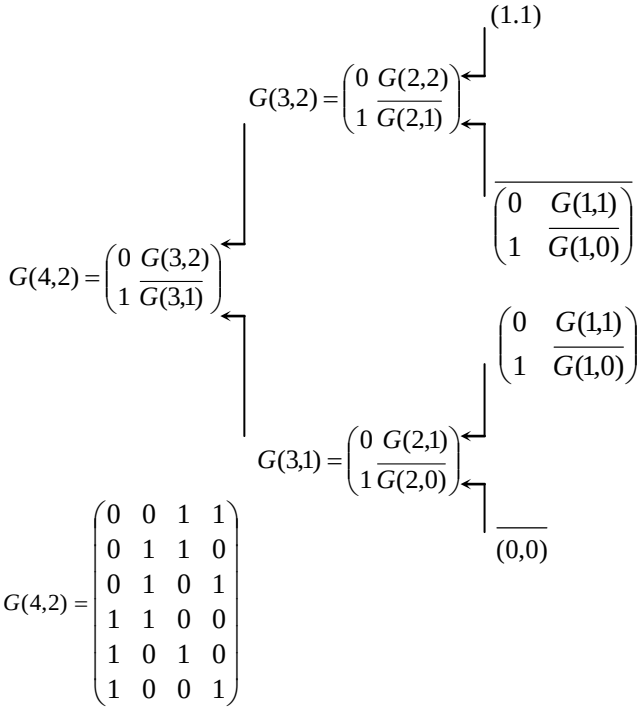

Рис. 1.1. Кодовые слова Грея для сочетаний из 4 по 2

Индукцией по n доказывается, что последовательность кодовых слов $G(n,k)$ получается удалением из кода Грея $G(n)$ всех кодовых слов с числом единиц, не равным k, причем в этой последовательности любые два соседних кодовых слов различаются только в двух позициях (*обратите внимание – алгоритм рекурсивный*).

Генерация перестановок

Генерация перестановок в лексикографическом порядке

Будем рассматривать исходное множество $X = \{1,2,\ldots,n\}$, и в качестве начальной перестановки возьмем $\pi' = (1,2,\ldots,n)$. Условие окончания работы — порождение перестановки $\pi'' = (n, n-1,\ldots 2,1)$, которая является последней в лексикографическом смысле среди всех перестановок множества *X*. Переход от текущей перестановки $\pi = (\pi_1, \pi_2, \ldots \pi_n)$ к следующей за ней будем осуществлять таким образом:

1) просматривая перестановку π справа налево, ищем самую первую позицию *i* такую, что $\pi_i < \pi_{i+1}$ (если такой позиции нет, значит текущая подстановка $\pi = \pi''$ и процесс генерации завершается);

2) просматривая π от π_i слева направо, ищем наименьший из элементов π_j такой, что $\pi_i < \pi_j (i < j)$;

3) меняем местами элементы π_i и π_j; затем все элементы $\pi_{i+1}, \pi_{i+2}, \ldots, \pi_n$ записываем в обратном порядке (т.е. меняем местами симметрично расположенные элементы π_{i+1+t} и π_{n-t}).

Пример. Пусть текущая перестановка π имеет вид $\pi = (3,5,7,6,4,2,1)$. На первом шаге найдены $\pi_i = 5, i = 2$; на втором — $\pi_j = 6, j = 4$; на третьем шаге меняем местами π_i и π_j: (3, 6, 7, 5, 4, 2, 1) и меняем местами элементы, начиная с третьей позиции: (3, 6, 1, 2, 4, 5, 7) — получили подстановку, следующую за текущей в лексикографическом порядке.

1.3.2. Генерация перестановок с помощью вложенных циклов

Будем говорить, что перестановка $\pi = \begin{pmatrix} 1 & 2 & \ldots & n \\ a_1 & a_2 & \ldots & a_n \end{pmatrix}$ является циклом длины *k* степени *d*, если ее элементы $a_i, i = \overline{1,k}$, получены из 1, 2, … , *k* циклическим сдвигом вправо на *d* позиций, остальные *n* – *k* элементов стационарны. Например, подстановка $\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 4 & 1 & 2 & 3 & 5 & 6 \end{pmatrix}$ является циклом длины 4 степени 1.

Алгоритм порождения подстановок с помощью вложенных циклов основан на следующей теореме.

Теорема 1. ***Любую подстановку*** π ***на множестве*** $X = \{1,2,\ldots,n\}$ ***можно представить в виде композиции***

$$\pi = \rho_n \circ \rho_{n-1} \circ \ldots \circ \rho_1 \qquad (1.2)$$

где ρ_i ***— циклическая подстановка порядка*** i.

Пример. Представим в виде (2.1) подстановку $\pi = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 2 & 4 & 1 \end{pmatrix}$, т.е. запишем

$$\pi = \rho_4 \circ \rho_3 \circ \rho_2 \circ \rho_1 = \begin{pmatrix} 1 & 2 & 3 & 4 \\ a_1 & a_2 & a_3 & a_4 \end{pmatrix} \circ \begin{pmatrix} 1 & 2 & 3 & 4 \\ b_1 & b_2 & b_3 & 4 \end{pmatrix} \circ$$

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ c_1 & c_2 & 3 & 4 \end{pmatrix} \circ \begin{pmatrix} 1 & 2 & 3 & 4 \\ d_1 & 2 & 3 & 4 \end{pmatrix}.$$

Очевидно, последний цикл является тождественной подстановкой. Определим ρ_4 : т.к. $\rho_3(4) = 4, \rho_2(4) = 4,$ то $\rho_4(4) = 1$ следовательно

$$\rho_4 = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 1 & 4 \end{pmatrix} \text{ — цикл порядка 4.}$$

Т.к. $\rho_2(3) = 3$, то $3 = \pi(1) = \rho_3(2) = \rho_3(\rho_4(1)) = \rho_2(2)$ и

$$\rho_2 = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 3 & 4 \end{pmatrix}.$$

Разложение подстановки π имеет вид:

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 2 & 4 & 1 \end{pmatrix} \circ \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 1 \end{pmatrix} \circ \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 1 & 4 \end{pmatrix} \circ \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 3 & 4 \end{pmatrix} \qquad (1.3)$$

Диаграмма композиции (1.3) приведена на рис. 2.2.

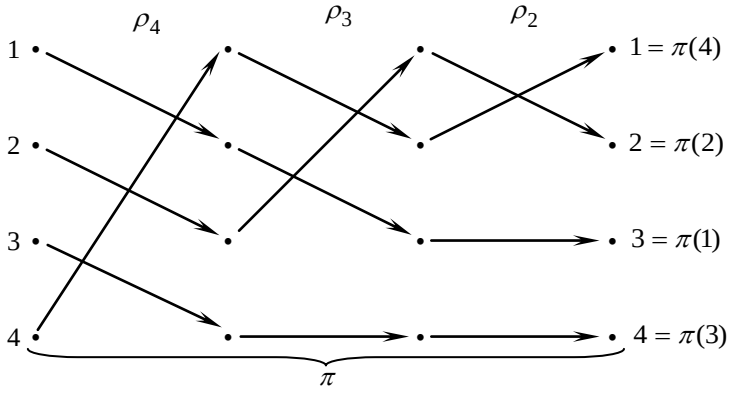

Рис. 2.2. Разложение в произведение вложенных циклов

Из теоремы 1 следует, что все перестановки можно получить систематическим перебором циклических сдвигов. В качестве начальной перестановки берем $\pi' = (1,2,\ldots,n)$ и сдвигаем на одну позицию вправо все элементы до тех пор, пока вновь не получим π'; теперь сдвигаем циклически первые $n-1$ элементов и снова повторяем сдвиг всех n элементов на одну позицию до тех пор, пока не получим уже имеющуюся перестановку; сдвигаем циклически ее первые $n-2$ элементов… и т.д., пока не переберем все $n!$ перестановок. Ниже приведен алгоритм вложенных циклов.

1. Цикл ($i = \overline{1,n}$)
    1.1. $\pi_i := i$;
2. Конец цикла
3. $k := 0$
4. Пока ($k \neq 1$)
    4.1.Печать $\pi = (\pi_1, \pi_2, \ldots \pi_n)$
    4.2. $k := n$
    4.3. Сдвиг первых k элементов на одну позицию
    4.4. Пока ($\pi_k = k$ и $k > 0$)
        4.4.1. $k := k - 1$
        4.4.2. Сдвиг первых k элементов на одну позицию
    4.5. Конец цикла
5. Конец цикла
6. Конец

Этот алгоритм не является эффективным, т.к. на каждом шаге требует большого количества (не меньше n) транспозиций (транспозиция — обмен местами двух элементов).

Транспозиция соседних элементов

Описанные выше алгоритмы генерации перестановок не являются алгоритмами с минимальными изменениями. Минимальным изменением при переходе от текущей перестановки к следующей является транспозиция двух элементов. Дадим рекурсивное описание такого алгоритма.

Если $n = 1$, то существует единственная перестановка $\pi^{(1)} = (1)$. Пусть $n > 1$ и последовательность перестановок

$\pi^{(1)}, \pi^{(2)}, \ldots, \pi^{(r)}, r = (n-1)!$ на множестве $(1,2,\ldots,n-1)$ построена. Для получения перестановок на множестве $(1,2,\ldots,n)$ будем вставлять элемент n на «промежутке» между элементами перестановки $\pi^{(i)}$ по следующему правилу: если номер i подстановки $\pi^{(i)}$ — нечетное число, то элемент n вставляется в промежутки справа налево, если I — четное число, то элемент n вставляется в промежутки между элементами $\pi^{(i)}$ слева направо.

Пример генерации перестановки при $n = 4$ приведен на рис. 1.3.

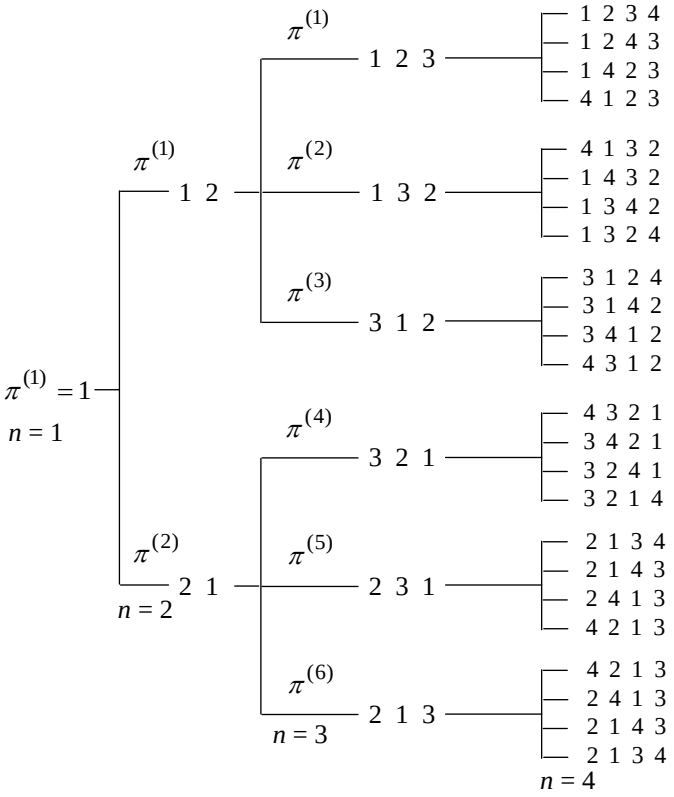

Рис. 1.3. Генерация перестановок транспозицией соседних элементов (n = 4)

Представленный на рис. 1.3 алгоритм является рекурсивным, не-

рекурсивная реализация называется алгоритмом Джонсона-Троттера (http://neerc.ifmo.ru/wiki/index.php?title=%D0%A3%D1%87%D0%B0%D1%81%D1%82%D0%BD%D0%B8%D0%BA:ZeRoGerc)