

## Лекция 1

### МЕТОДЫ РЕШЕТА

*Решето* представляет собой метод комбинаторного программирования, который рассматривает конечное множество элементов и исключает все элементы этого множества, не представляющие интереса.

Методы решета полезны в теоретико-числовых исследованиях. Например, один из наиболее известных методов отыскания простых чисел носит название “Решето Эратосфена”. Пусть нам необходимо найти все простые числа меньше 25. Рассмотрим все натуральные числа от 1 до 25. Вычеркнем из них числа кратные 2 (исключая 2), т.е. числа 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24. Заметим, что начали мы вычеркивать числа, кратные 2 с числа 4. Теперь вычеркнем среди оставшихся чисел числа кратные 3 (исключая само число 3). Теперь мы начинаем вычеркивать с числа 9, так как число 6 является четным, и было вычеркнуто на предыдущем шаге. Таким образом, мы вычеркиваем 9, 15, 21. Далее начинаем вычеркивать числа кратные 5 (исключая 5) и видим, что первое невычеркнутое число кратное 5 – это 25. Нетрудно видеть, что все оставшиеся невычеркнутыми числа, т.е. 1, 2, 3, 5, 7, 11, 13, 17, 19, 23 являются простыми.

Из рассмотренного примера следует, что для нахождения всех простых чисел, не превосходящих заданного числа  $N$ , нужно вычеркивать все числа кратные простым делителям до наибольшего простого числа  $\leq \lfloor \sqrt{N} \rfloor$ . При этом каждый раз вычеркивание нужно начинать с числа равного квадрату соответствующего простого делителя.

Занумеруем все элементы в исходном множестве натуральными числами и будем хранить только так называемый характеристический вектор, где  $i$ -й разряд равен 1, если он является решением и равен 0 в противном случае. Для нашего примера характеристический вектор  $X$  имеет вид

$$X = 1110101000101000101000100.$$

Алгоритм “Решето Эратосфена” приведен на Рис.1.

Легко понять, почему методы решета могут быть полезны. Поскольку мы работаем с характеристическим вектором, то на множествах, состоящих буквально из миллионов элементов, возможен поиск без явного порождения и исследования каждого элемента множества. Кроме того, принимая во внимание, что целое число в ЭВМ представлено в двоичной форме, характеристический вектор может быть представлен как одно или несколько целых чисел. Таким образом, все арифметические и булевы операции с разрядами этого вектора можно производить параллельно, выполняя их над соответствующими целыми числами, обеспечивая тем самым значительную экономию времени.

### НЕРЕКУРСИВНОЕ МОДУЛЬНОЕ РЕШЕТО

Решето Эратосфена можно интерпретировать как поиск чисел, которые одновременно являются членами одной из арифметических прогрессий, а именно

$\{2k + 1\}$  – нечетные числа,

$\{3k + 1\}$ ,  $\{3k + 2\}$  – числа, которые не кратны трем,

$\{5k + 1\}$ ,  $\{5k + 2\}$ ,  $\{5k + 3\}$ ,  $\{5k + 4\}$  – числа, которые не кратны пяти,

.....

$\{pk + 1\}$ ,  $\{pk + 2\}$ , ...,  $\{pk + p - 1\}$ , где  $p$  – наибольшее простое число  $\leq \lfloor \sqrt{N} \rfloor$ .

Рассмотрим одну головоломку. Женщина продавала розы. Когда она брала розы парами, то оставалась одна роза. Если она брала розы по три, четыре, пять или шесть, то тоже каждый раз оставалась одна роза. Но когда она брала по семь роз, то в остатке ничего не было. Необходимо найти все возможные значения числа роз, меньшие 1000.

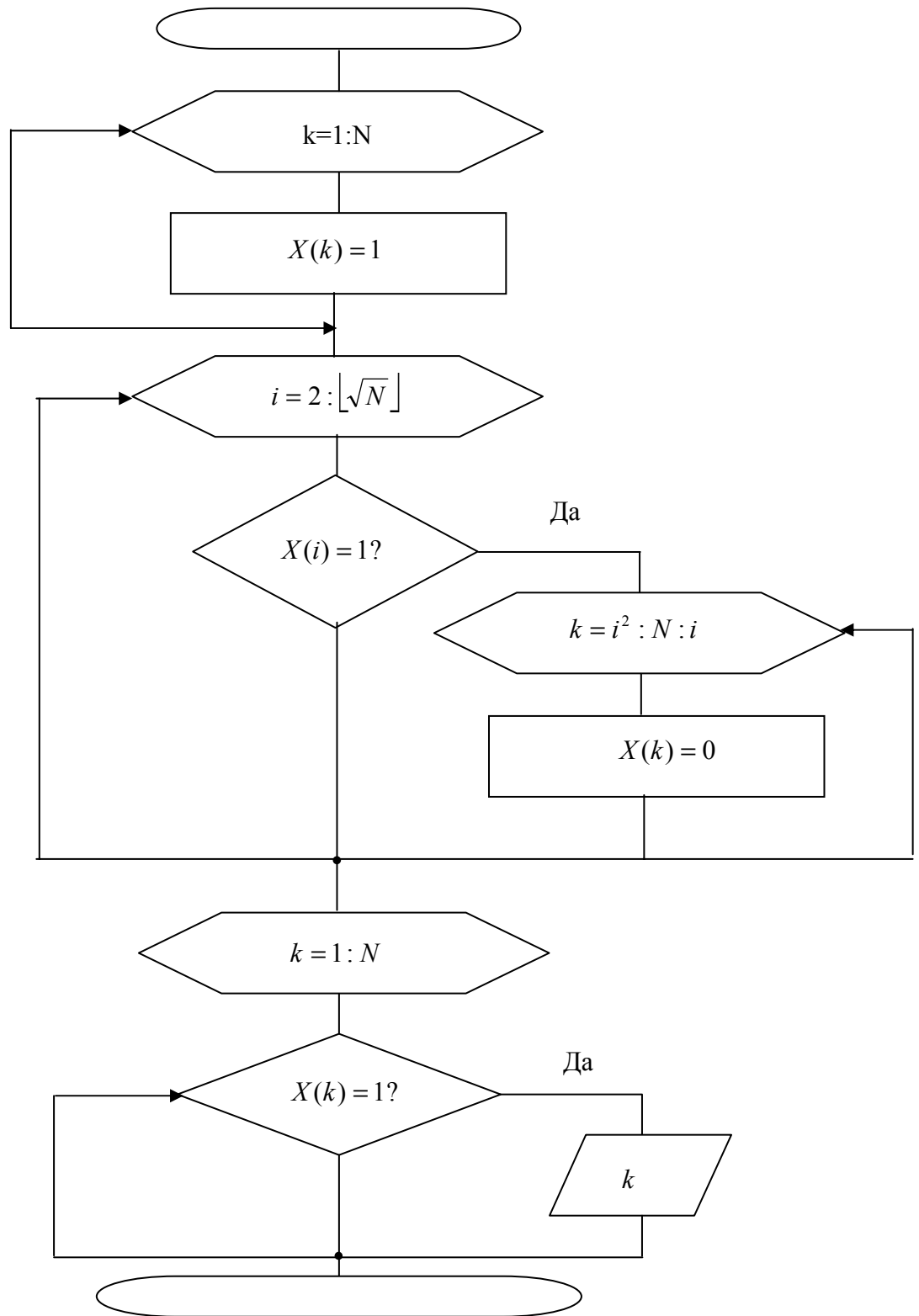


Рис.1. Решето Эратосфена

Очевидно, что число роз является одновременно членом каждой из арифметических прогрессий

$$\begin{aligned} &\{2k + 1\}, \\ &\{3k + 1\}, \\ &\{4k + 1\}, \\ &\{5k + 1\}, \\ &\{6k + 1\}, \\ &\{7k\}, \quad k = 1, 2, \dots \end{aligned}$$

Для решения этой задачи можно использовать решето, выписывая целые числа  $1, \dots, 1000$  и затем удаляя все элементы, не принадлежащие различным прогрессиям. Оставшиеся числа и будут возможными решениями.

Решето Эратосфена и решето для головоломки про розы являются специальными случаями обобщенного модульного решета. Пусть  $m_1, m_2, \dots, m_t$  – множество из  $t$  целых чисел, называемых модулями. Для каждого  $m_i$  рассмотрим  $n_i$  арифметических прогрессий

$$m_i k + a_{ij}, \quad j = 1, 2, \dots, n_i, \quad i = 1, \dots, t.$$

Задача состоит в отыскании всех целых чисел, заключенных между числами  $A$  и  $B$ , которые для каждого  $m_i$  одновременно принадлежат одной из  $n_i$  прогрессий.

В решете Эратосфена мы имеем  $m_1 = 2, m_2 = 3, m_3 = 5, \dots, m_i = p$  ( $p$  – наибольшее простое число  $\leq \lfloor \sqrt{N} \rfloor$ ,  $n_i = m_i - 1$  и  $a_{ij} = j$ ).

В задаче о розах  $m_i = i + 1, n_i = 1, 1 \leq i \leq 6, a_{i1} = 1, 1 \leq i \leq 5, a_{61} = 0$ .

## РЕКУРСИВНОЕ РЕШЕТО

Существует много решет, в которых модули  $m_1, m_2, \dots$  заранее не заданы, значение  $m_i$  будет зависеть от чисел, не удаленных после просеивания по модулю  $m_{i-1}$ . Примером может служить решето, порождающее счастливые числа. Из списка чисел  $1, 2, 3, 4, 5, \dots$  удаляется каждое второе число, в результате получается список  $1, 3, 5, 7, 9, \dots$ . Так как 3 является первым числом (исключая 1), которое не использовано в качестве просеивающего, то из оставшихся чисел удаляем каждое третье число, получаем  $1, 3, 7, 9, 13, 15, 19, 21, \dots$ . Теперь удаляем каждое седьмое из оставшихся чисел и получаем  $1, 3, 7, 9, 13, 15, 21$ . Числа, которые никогда не удаляются из списка называются счастливыми.

Несмотря на большую схожесть решета Эратосфена и решета для получения счастливых чисел, последнее реализуется труднее. Основная трудность при решении задачи о счастливых числах состоит в том, что числа, исключаемые на  $k$ -м шаге, зависят от еще не исключенных элементов первоначального множества. В предположении, что для представления чисел используется двоичная последовательность, теперь нужно отсчитывать, скажем, каждый седьмой единичный разряд в характеристическом векторе вместо вообще седьмого разряда. Задача упрощается, если использовать *бирки*. Вместо использования полного машинного слова для представления решений, которые мы должны просеять, мы используем часть этого слова для хранения счетчика единиц в остальной части слова. Например, в ЭВМ со словом, состоящим из четырех байтов, каждый из которых состоит из восьми разрядов, мы можем использовать слово так, как показано на Рис 2. Самый правый байт слова содержит счетчик числа единичных разрядов в трех левых байтах. С использованием бирок отыскание  $t$ -го невычеркнутого элемента

( $t$ -го единичного разряда) легко осуществляется путем суммирования бирок до тех пор, пока их сумма  $S$  не станет большей или равной  $t$ . Искомый элемент будет  $(S - t + 1)$ -м единичным разрядом справа в левых байтах последнего слова, чья бирка суммировалась.

Навешивание бирок не приводит к экономии времени, если из каждого слова исключается один или более разрядов. По этой причине имеет смысл осуществить просеивание для нескольких шагов без бирок и начинать их использовать, когда подлежащие отсеиванию элементы становятся достаточно редкими.

Навешивание бирок можно расширить до бирок более высокого порядка, выбирая целое  $m \geq 2$  и подсчитывая сумму бирок в блоках из  $m$  слов.

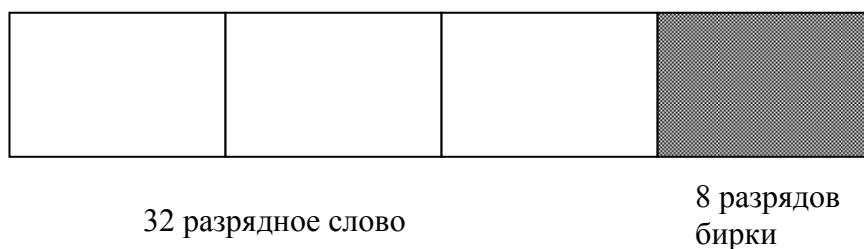


Рис.2. Использование бирок

## ГЕНЕРАЦИЯ ЭЛЕМЕНТАРНЫХ КОМБИНАТОРНЫХ ОБЪЕКТОВ

В комбинаторных алгоритмах часто необходимо порождать и исследовать все элементы некоторого класса комбинаторных объектов. В алгоритме порождения всех элементов множества нас интересует общее количество времени, требующегося для порождения всего множества. В частности, в некоторых алгоритмах можно порождать все множество за время, пропорциональное его мощности. Такие алгоритмы, называемые линейными, имеют асимптотически наилучшую эффективность. Представляет также интерес количество изменений, которые происходят при переходе к последующим объектам. Например, мы рассмотрим алгоритм порождения перестановок, в котором последовательные перестановки различаются лишь транспозицией соседних элементов перестановки. Такие алгоритмы называются *алгоритмами с минимальным изменением*.

### ПЕРЕСТАНОВКИ РАЗЛИЧНЫХ ЭЛЕМЕНТОВ

Без потери общности будем полагать, что элементами множества являются целые числа от 1 до  $n$ , т.е. рассматриваются только перестановки целых чисел от 1 до  $n$ . Очевидно, что число перестановок можно подсчитать по формуле

$$n(n-1)(n-2)\dots 1 = n!,$$

так как для первого элемента перестановки существует  $n$  возможностей, для второго –  $n-1$ , для третьего –  $n-2$  и для последнего возможен только один вариант. Например, все перестановки для  $n = 3$ :

123, 132, 213, 231, 312, 321.

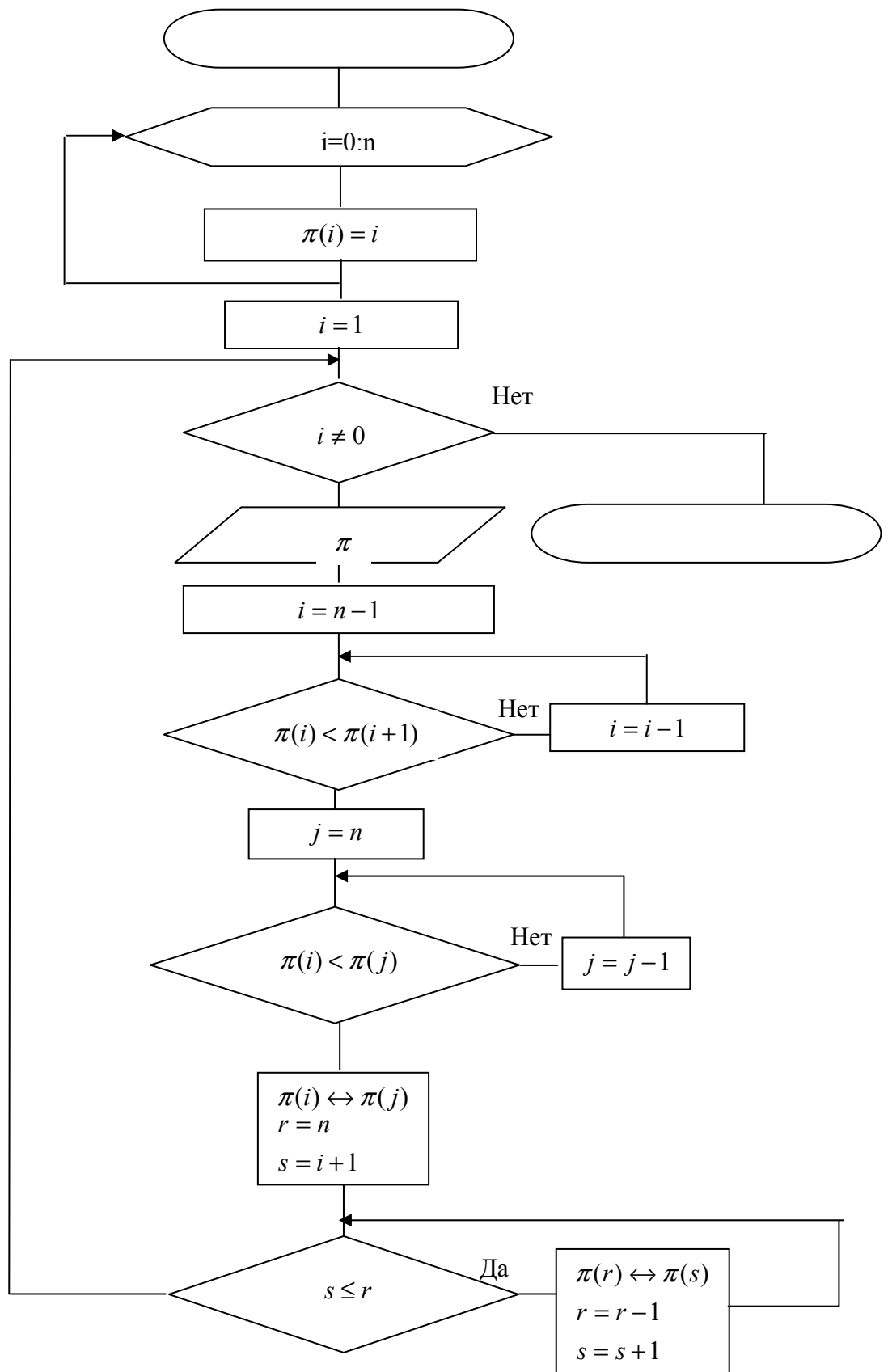


Рис.3. Генерация перестановок в лексикографическом порядке

В общем случае мы говорим, что перестановка  $\sigma = (\sigma_1, \sigma_2, \dots, \sigma_n)$  лексикографически меньше, чем другая перестановка  $\tau = (\tau_1, \tau_2, \dots, \tau_n)$ , если и только если для некоторого  $k \geq 1$  мы имеем  $\sigma_j = \tau_j$  для всех  $j < k$  и  $\sigma_k < \tau_k$ . Последовательность перестановок на множестве  $\{1, 2, \dots, n\}$  представлена в лексикографическом порядке, если записана в порядке возрастания получающихся чисел. Приведенная выше последовательность перестановок на множестве  $\{1, 2, 3\}$  записана в лексикографическом порядке. Алгоритм порождения перестановок в лексикографическом порядке приведен на Рис.3.

Начиная с перестановки  $(1, 2, \dots, n)$ , мы переходим от перестановки  $\pi = (\pi_1, \pi_2, \dots, \pi_n)$  к следующей за ней путем просмотра  $\pi$  справа налево в поисках самой правой позиции, в которой  $\pi_i < \pi_{i+1}$ . Найдя ее, мы теперь ищем  $\pi_j$ , наименьший элемент, расположенный справа от  $\pi_i$  и больший его. Затем осуществляется перестановка элементов  $\pi_i$  и  $\pi_j$  и отрезок  $\pi_{i+1}, \dots, \pi_n$  (элементы которого расположены в порядке убывания) переворачивается. Например, для  $n = 6$  и  $\pi = (2, 4, 5, 6, 3, 1)$  мы имеем  $\pi_i = 5$  и  $\pi_j = 6$ . Меняя их местами, получаем перестановку  $(2, 4, 6, 5, 3, 1)$ , переворачивая  $\pi_{i+1}, \dots, \pi_n$ , получаем  $(2, 4, 6, 1, 3, 5)$ , т.е. перестановку, следующую за исходной в лексикографическом порядке.

Алгоритм начинает работу с печати  $\pi = (1, 2, \dots, n)$ , первой в лексикографическом порядке перестановки, и останавливается только, когда  $i = 0$ , что происходит, если и только если  $\pi_1 > \pi_2 > \dots > \pi_n$ , т.е. когда распечатана  $\pi = (n, n-1, \dots, 1)$ , последняя в лексикографическом порядке перестановка.

Алгоритм порождения перестановок в лексикографическом порядке не является лучшим с точки зрения минимизации вычислений. Очевидно, что последовательность перестановок, в которой соседние перестановки различаются так мало, как только это возможно – это лучшее на что можно надеяться с точки зрения минимизации объема вычислений, необходимых для порождения последовательности. Для того, чтобы такое различие было минимально возможным, любая перестановка в нашей последовательности должна отличаться от предшествующей ей перестановкой двух соседних элементов. Такую последовательность перестановок легко построить рекуррентно. Для  $n = 1$  перестановка  $\pi = (1)$  удовлетворяет нашим требованиям. Предположим, что мы имеем последовательность перестановок на множестве  $\{1, 2, \dots, n-1\}$ , в которой последовательные перестановки различаются только перестановкой соседних элементов. Мы будем расширять каждую из этих  $(n-1)!$  перестановок, вставляя элемент  $n$  на каждое из  $n$  возможных мест. Для того, чтобы расположить эти  $n$  перестановок в порядке минимального изменения будем поступать следующим образом. Будем добавлять  $n$  к перестановке  $\pi_i$  последовательно во все позиции справа налево, если  $i$  нечетно и слева направо, если  $i$  четно. Порядок порождаемых перестановок для  $n = 4$  будет следующим:

1	}					1	2	3	4	
				1	2	3	1	2	4	3
							1	4	2	3
							4	1	2	3
							4	1	3	2
		1	2	1	3	2	1	4	3	2
							1	3	4	2
							1	3	2	4
							3	1	2	4
				3	1	2	3	1	4	2
							3	4	1	2
							4	3	1	2
							4	3	2	1
				3	2	1	3	4	2	1
							3	2	4	1
							3	2	1	4
							2	3	1	4
		2	1	2	3	1	2	3	4	1
							2	4	3	1
							4	2	3	1
					4	2	1	3		
		2	1	3	2	4	1	3		
					2	1	4	3		
					2	1	3	4		

Однако приведенный алгоритм требует порождения всего списка перестановок, начиная с  $n=1$ , что требует больших затрат памяти. Попробуем порождать ту же последовательность перестановок итеративно, получая каждую перестановку из предшествующей ей и небольшого количества добавочной информации. Для этого будем использовать текущую перестановку  $(\pi_1, \pi_2, \dots, \pi_n)$ , перестановку обратную к ней  $(p_1, p_2, \dots, p_n)$  и записи направления движения  $(d_1, d_2, \dots, d_n)$ , где  $d_i = -1$ , если элемент  $i$  сдвигается влево,  $d_i = 1$ , если элемент  $i$  сдвигается вправо и  $d_i = 0$ , если элемент  $i$  остается на месте. Прежде чем привести описание алгоритма, определим перестановку обратную данной. Перестановку можно задать в виде матрицы с двумя строками. Порядок элементов в первой строке фиксирован, а вторая строка соответствует собственно перестановке, например

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

Тогда перестановка обратная к данной может быть определена следующим образом: меняем строки местами

$$\begin{pmatrix} 3 & 1 & 2 \\ 1 & 2 & 3 \end{pmatrix},$$

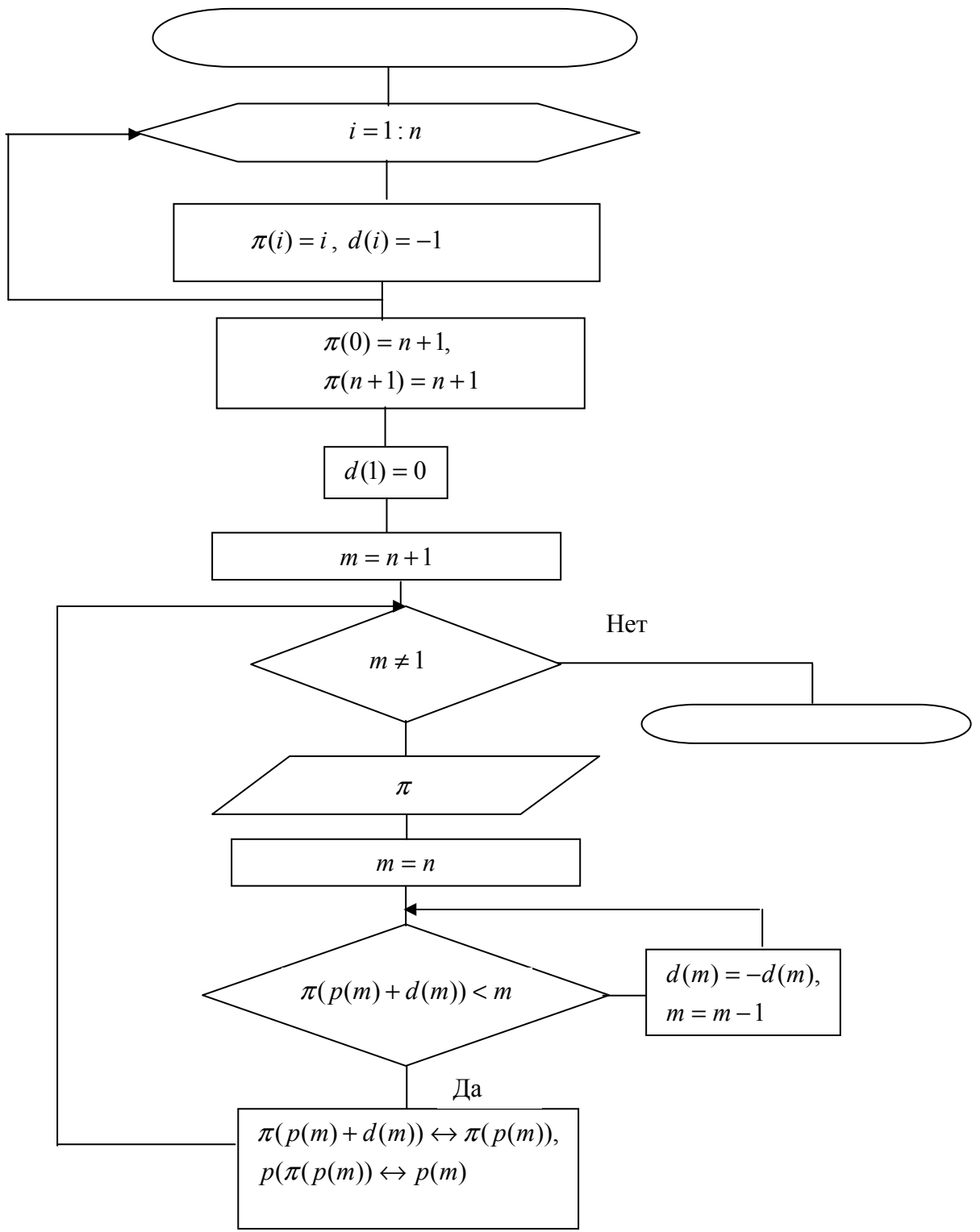


Рис.4. Генерация перестановок в порядке минимального изменения

восстанавливаем исходный порядок элементов в первой строке, при этом вторая строка переставляется вместе с первой,

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

Полученная перестановка 231 называется обратной к 312. Отметим, что перестановка

$$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$$

называется тождественной перестановкой. Нетрудно видеть, что элементы обратной перестановки представляют собой индексы элементов прямой перестановки. А именно, в нашем примере  $p_1 = 2$ , а  $\pi_2 = 1$ , то есть 1 стоит в прямой перестановке на втором месте.

Теперь продолжим описание алгоритма. Начинаем с элемента  $i = n$ , элемент сдвигается до тех пор, пока не достигнет элемента большего, чем он сам. После этого направление движения данного элемента меняется на противоположное, и передвигается следующий меньший его элемент, который можно сдвинуть. Так как хранится перестановка обратная к данной, то легко найти место следующего меньшего элемента.

Алгоритм порождения перестановок в порядке минимального изменения приведен на Рис.4.

## СОЧЕТАНИЯ

Теперь мы обсудим порождение подмножеств фиксированной мощности  $k$  из  $n$  элементов, т.е. сочетания из  $n$  элементов по  $k$  штук. Число сочетаний из  $n$  элементов по  $k$  обозначается  $\binom{n}{k}$  и вычисляется по формуле

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

Выведем эту формулу. Очевидно, что число перестановок длины  $k$  из  $n$  элементов (их называют размещениями) определяется как  $n(n-1)(n-2)\dots(n-k+1)$ . Однако, при рассмотрении перестановок мы различали порядок элементов, т.е. например, перестановки 123 и 321 считались различными. Теперь порядок элементов для нас не важен и подмножества 123, 321, 213 и т.д. мы рассматриваем как одно подмножество. Таким образом, для того, чтобы вычислить число возможных сочетаний мы должны уменьшить число перестановок длины  $k$  из  $n$  элементов в  $k!$  раз, т.е. в число раз равное числу возможных перестановок из  $k$  элементов. В результате имеем

$$\binom{n}{k} = \frac{n(n-1)(n-2)\dots(n-k+1)}{k!} = \frac{n!}{(n-k)!k!}.$$

Алгоритм порождения сочетаний в лексикографическом порядке приведен на Рис.5. Начиная с сочетания  $(1,2,\dots,k)$ , следующее сочетание находится просмотром текущего сочетания справа налево для того, чтобы найти место самого правого элемента, который еще не достиг своего максимального значения. Этот элемент увеличивается на 1, а всем элементам справа от него присваиваются новые наименьшие возможные значения. Например, сочетания из 4 элементов по 2, расположенные в лексикографическом порядке, имеют вид

12 23 34  
13 24  
14

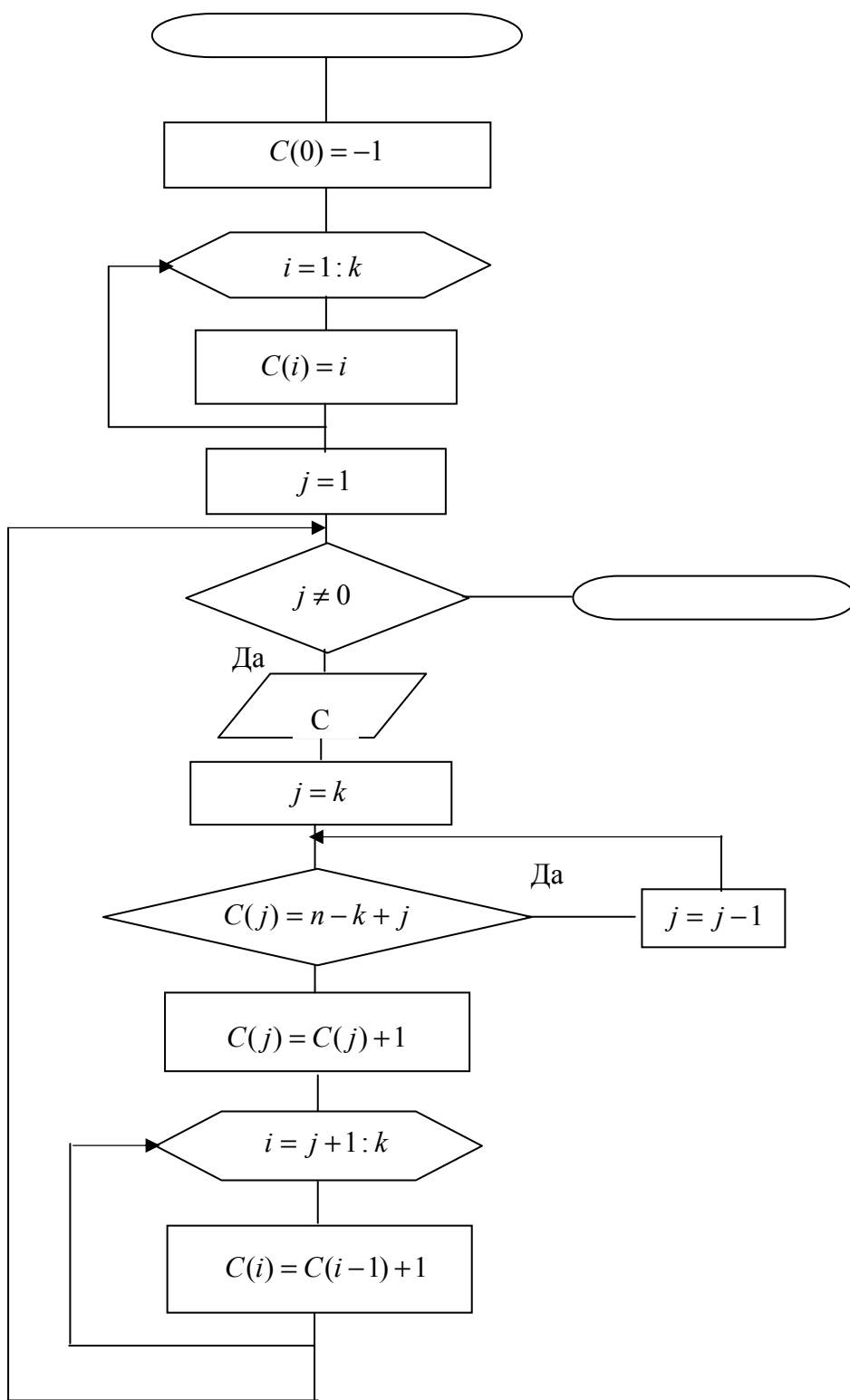


Рис.5. Порождение сочетаний

## РАЗБИЕНИЯ ЦЕЛЫХ ЧИСЕЛ

Мы рассмотрим задачу порождения разбиений положительного целого числа  $n$  в последовательность неотрицательных целых чисел  $\{p_1, p_2, \dots, p_n\}$  таких, что  $n = p_1 + p_2 + \dots + p_n$ . Порядок  $p_i$  не важен и  $p_i > 0$ . Алгоритм порождения разбиений числа  $n$  с  $l$  компонентами в возрастающем лексикографическом порядке приведен на Рис.6.

Начинаем с  $p_1 = p_2 = \dots = p_{l-1} = 1$ ,  $p_l = n - l + 1$  и продолжаем следующим образом. Для получения следующего разбиения из текущего просматриваем элементы справа налево, останавливаясь на самом правом  $p_i$ , таком, что  $p_i - p_i \geq 2$ . Заменяем  $p_j$  на

$p_i + 1$  для  $j = i, i + 1, \dots, l - 1$  и после этого заменяем  $p_l$  на  $n - \sum_{j=1}^{l-1} p_j$ .

Пусть  $n = 12$ ,  $l = 5$  и разбиение имеет вид  $\{1, 1, 3, 3, 4\}$ . Так как разница между элементом 4 и самой правой 1 больше 2, то следующее разбиение имеет вид  $\{1, 2, 2, 2, 5\}$ . Когда ни один элемент разбиения не отличается от последнего больше, чем на 1, мы заканчиваем процедуру для данной длины разбиения.

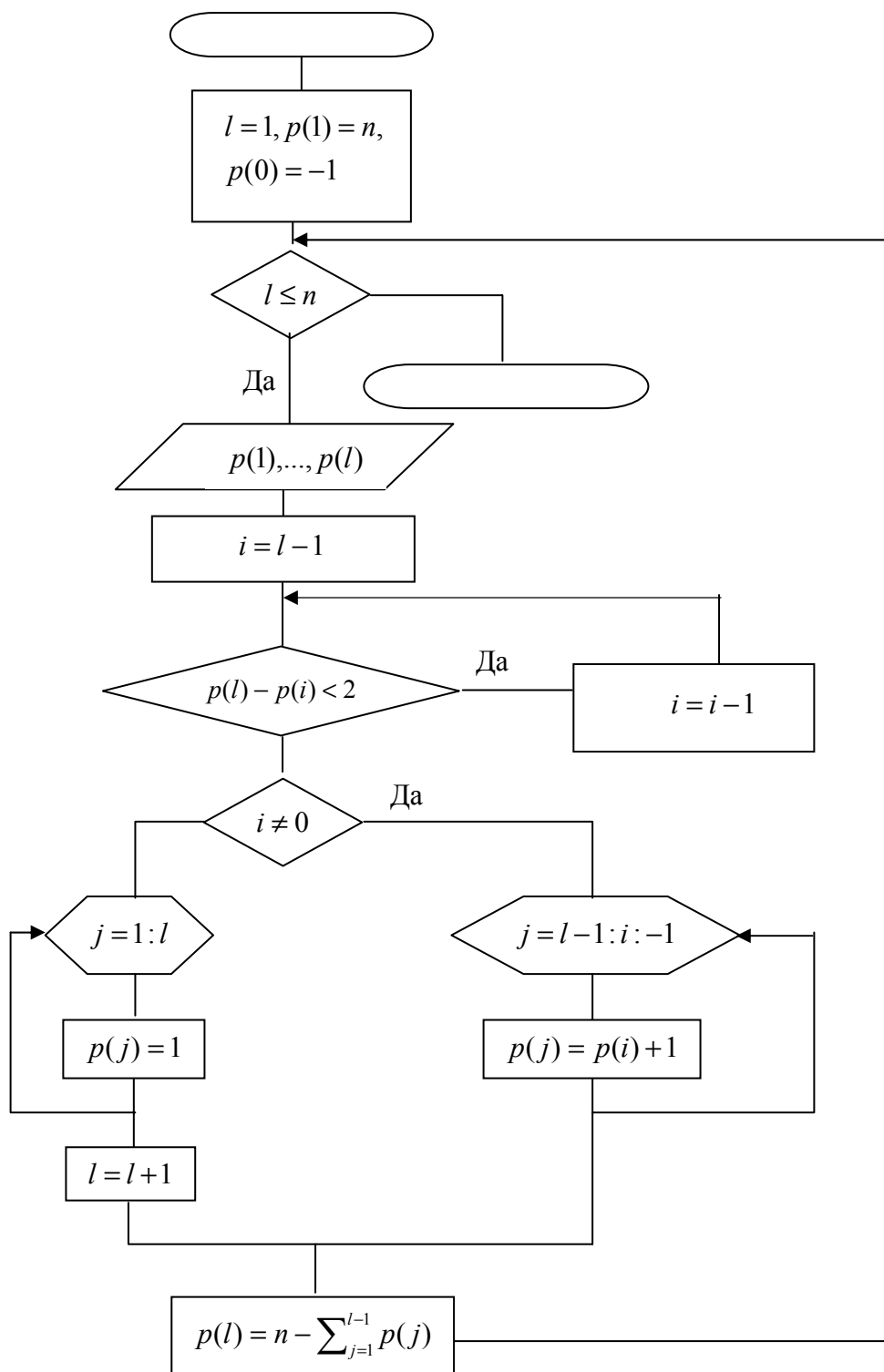


Рис.6. Порождение разбиений целых чисел