

## Порождение элементарных комбинаторных объектов

(темы нет в учебном пособии)

Часто возникает необходимость порождения и исследования всех элементов некоторого класса комбинаторных объектов. Наиболее общие методы решения таких задач основаны на поиске с возвратом, однако во многих случаях объекты настолько просты, что целесообразнее применять специализированные методы. В данном разделе рассмотрим процедуры порождения некоторых комбинаторных объектов, которые обычно встречаются на практике. В каждом случае существуют две возможные цели: систематическое порождение всех возможных конфигураций и порождение равномерно распределенных случайных конфигураций.

Алгоритм систематического порождения состоит из трех компонент: выбор начальной конфигурации, преобразование одного объекта в следующий и условие окончания. Общая форма такого алгоритма будет иметь следующий вид:

```
Выбрать начальную конфигурацию
while not условие окончания do
  { Вывести объект
  { Преобразовать в следующий объект
```

В рамках такой схемы выбор начальной конфигурации должен учитывать тот факт, что алгоритм предпринимает еще одно преобразование после того, как выведен последний объект. Поскольку это избыточное преобразование не должно приводить к ошибке, некоторые из операций выбора начальной конфигурации могут присваивать значения на вид посторонним переменным, например, нулевому или  $(n + 1)$ -му элементу массива, в то время как массив для представления объекта обычно имеет элементы только с номерами от 1 до  $n$ . Обращение к этой посторонней переменной или изменение ее значения часто можно использовать как условие окончания.

В алгоритме порождения всех элементов множества нас прежде всего 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*, желательны, поскольку их эффективность – асимптотически наилучшая из возможных. Представляет также интерес количество изменений, которые происходят при переходе к последующим объектам. Иногда желательно, чтобы такого рода изменения были минимальными. Такие алгоритмы называются *алгоритмами с минимальным изменением*. Точное определение минимального изменения зависит от рассматриваемых комбинаторных конфигураций.

Задачи, требующие случайного порождения элементов в классе комбинаторных объектов, возникают при оценке сложности по методу Монте-Карло. Например, если мы не можем удовлетворительно проанализировать поведение алгоритма в среднем, для суждений об эффективности, возможно, придется испытать его на большом числе случайно выбранных входных конфигурациях.

Один из способов подхода к систематическому и случайному порождению состоит в задании определенного соответствия между целыми числами  $1, 2, \dots, N$  и  $N$  объектами. Систематическое порождение тогда осуществляется перечислением целых чисел от  $1, 2, \dots, N$  и обращением каждого числа в объект, в то время как случайное порождение осуществляется случайным порождением целого числа от  $1, 2, \dots, N$  и обращением его в объект. Процесс обращения, однако, может быть дорогостоящим, и обычно лучше его избегать, если это возможно.

### 3.6. Перестановки различных элементов

К часто порождаемым комбинаторным объектам относятся перестановки множества различных элементов. Без ограничения общности будем полагать, что элементами множества являются целые числа от 1 до  $n$ , всего имеем  $n!$  перестановок.

#### 3.6.1. Лексикографический порядок

Последовательность перестановок на множестве  $\{1, 2, \dots, n\}$  представлена в лексикографическом порядке, если она записана в порядке возрастания получающихся чисел. Например, лексикографическая последовательность перестановок трех элементов имеет вид 123, 132, 213, 231, 312, 321. В общем случае если  $\sigma = (\sigma_1, \sigma_2, \dots, \sigma_n)$  и  $\tau = (\tau_1, \tau_2, \dots, \tau_n)$  – перестановки, то говорят, что  $\sigma$  лексикографически меньше  $\tau$ , если и только если для некоторого  $k \geq 1$  имеет место  $\sigma_j = \tau_j$  для всех  $j < k$  и  $\sigma_k < \tau_k$ .

Необходимо отметить, что явное применение алгоритма поиска с возвратом к перестановкам порождает их в лексикографическом порядке.

Перестановки можно порождать одну за другой следующим образом. Начиная с перестановки  $(1, 2, \dots, n)$ , осуществляется переход от перестановки  $\Pi = (\pi_1, \pi_2, \dots, \pi_n)$  к ее последующей путем просмотра  $\Pi$  справа налево в поисках самой правой позиции  $i$ , в которой  $\pi_i < \pi_{i+1}$ . Найдя позицию  $i$ , производится поиск позиции  $j$ , в которой  $\pi_j$  есть наименьший элемент, расположенный справа от  $\pi_i$  и больший его. Затем осуществляется транспозиция элементов  $\pi_i$  и  $\pi_j$  и обращение отрезка  $\pi_{i+1}, \dots, \pi_n$  (элементы которого расположены в порядке убывания) путем транспозиции симметрично расположенных элементов (в результате элементы отрезка будут расположены в порядке возрастания).

Например, для  $n = 8$  и  $\Pi = (2, 3, 6, 8, 7, 5, 4, 1)$  имеем  $\pi_i = \pi_3 = 6$  и  $\pi_j = \pi_5 = 7$ . Выполнив транспозицию  $\pi_3$  и  $\pi_5$  и обращение  $\pi_4, \pi_5, \pi_6, \pi_7, \pi_8$ , получаем перестановку  $(2, 3, 7, 1, 4, 5, 6, 8)$ , следующую за перестановкой  $\Pi$  в лексикографическом порядке.

Детали реализации порождения перестановок в лексикографическом порядке показаны в алгоритме 1. Алгоритм начинает работу с вывода  $\Pi = (1, 2, \dots, n)$ , первой в лексикографическом порядке перестановки, и останавливается только, когда  $i = 0$ , что происходит, если и только если  $\pi_1 > \pi_2 > \dots > \pi_n$ , то есть после порождения  $\Pi = (n, n - 1, \dots, 1)$ , последней в лексикографическом порядке перестановки.

```

for  $j \leftarrow 0$  to  $n$  do  $\pi_j \leftarrow j$ 
 $i \leftarrow 1$ 
while  $i \neq 0$  do
    { Вывести  $\Pi = (\pi_1, \pi_2, \dots, \pi_n)$ 
      { найти самое правое место, где  $\pi_i < \pi_{i+1}$  }
       $i \leftarrow i - 1$ 
      while  $\pi_i > \pi_{i+1}$  do  $i \leftarrow i - 1$ 
      { найти  $\pi_j$ , наим. элемент справа от  $\pi_i$  и больший его }
       $j \leftarrow n$ 
      while  $\pi_i > \pi_j$  do  $j \leftarrow j - 1$ 
      { поменять местами  $\pi_i$  и  $\pi_j$  и перевернуть  $\pi_{i+1}, \dots, \pi_n$  }
       $\pi_i \leftrightarrow \pi_j$ 
       $r \leftarrow n$ 
       $s \leftarrow i + 1$ 
      while  $r > s$  do
          {  $\pi_r \leftrightarrow \pi_s$ 
             $r := r - 1$ 
             $s := s + 1$ 
          }
  
```

Алгоритм 1. Лексикографическое порождение перестановок

Общую эффективность этого алгоритма определяют две операции: число транспозиций и число сравнений между элементами перестановок. Воспользуемся результатами из [21].

Заметим, что в лексикографически упорядоченной последовательности перестановок существует полная подпоследовательность  $k!$  перестановок из  $k$  правых элементов, в которой остальные элементы не перемещаются. Это наблюдение позволяет получить рекуррентное соотношение для  $I_k$  и  $C_k$ , числа транспозиций « $\pi_i \leftrightarrow \pi_j$ » или « $\pi_r \leftrightarrow \pi_s$ » и числа сравнений « $\pi_i > \pi_{i+1}$ » или « $\pi_i > \pi_j$ » соответственно использованных алгоритмом 1 для порождения первых  $k!$  из  $n!$  перестановок. В частности, заметим, что для каждого из  $n$  возможных значений  $\pi_1$ , т. е. для каждой подпоследовательности перестановок, соответствующих  $n - 1$  самым правым компонентам, используется  $I_{n-1}$  транспозиций. Преобразование последней перестановки одной из этих подпоследовательностей в первую перестановку следующей подпоследовательности требует  $\lfloor (n+1)/2 \rfloor$  перестановок, и всего таких преобразований будет  $n - 1$ . Таким образом, за исключением преобразования, которое осуществляется при  $i = 0$ , имеется  $nI_{n-1} + (n-1)\lfloor (n+1)/2 \rfloor$  транспозиций, а поэтому

$$\begin{aligned} I_n &= nI_{n-1} + (n-1)\lfloor (n+1)/2 \rfloor \\ I_1 &= 0 \end{aligned}$$

Решение этого рекуррентного соотношения трудно найти прямо, но путем замены переменных можно сделать следующее. Пусть

$$S_n = I_n + \lfloor (n+1)/2 \rfloor;$$

тогда

$$\begin{aligned} S_1 &= 1, \\ S_n &= n(S_{n-1} + \varepsilon_n), \varepsilon_n = \begin{cases} 0, & \text{если } n \text{ нечетно,} \\ 1, & \text{если } n \text{ четно.} \end{cases} \end{aligned}$$

Решение этого соотношения легко получить:

$$S_n = n! \left( 1 + \frac{1}{2!} + \frac{1}{4!} + \frac{1}{6!} + \dots + \frac{1}{(2 \lfloor (n-1)/2 \rfloor)!} \right).$$

Поэтому

$$I_n = n! \left( \sum_{j=0}^{\lfloor (n-1)/2 \rfloor} \frac{1}{(2j)!} \right) - \left\lfloor \frac{n+1}{2} \right\rfloor.$$

Добавление  $\lfloor (n+2)/2 \rfloor$  транспозиций, осуществляемых при  $i = 0$ , дает всего

$$n! \left( \sum_{j=0}^{\lfloor (n-1)/2 \rfloor} \frac{1}{(2j)!} \right) + \varepsilon_n$$

транспозиций. Поскольку

$$\sum_{j=0}^{\lfloor (n-1)/2 \rfloor} \frac{1}{(2j)!} \approx \text{ch } 1 \approx 1,54308,$$

получаем, что алгоритм для порождения  $n!$  перестановок использует приблизительно  $1,54308n!$  транспозиций.

Аналогично, для числа сравнений можно составить рекуррентное соотношение

$$C_n = nC_{n-1} + \frac{(n-1)(3n-2)}{2}$$
$$C_1 = 0$$

и найти его решение (путем замены переменных, т. е. подставив  $T_n = C_n + (3n+1)/2$ . В результате получим, что

$$C_n \approx \left(\frac{3}{2}e - 1\right)n! \approx 3,07742n!$$

Таким образом, временная сложность алгоритма  $O(n!)$ .

### 3.6.2. Векторы инверсий

Пусть  $\Pi = (\pi_1, \pi_2, \dots, \pi_n)$  есть перестановка. Пара  $(\pi_i, \pi_j)$  называется *инверсией*  $\Pi$ , если  $i < j$  и  $\pi_i > \pi_j$ . Вектор инверсий перестановки  $\Pi$  – это последовательность целых чисел

$$D = (d_1, d_2, \dots, d_n)$$

таких, что  $d_j$  – число элементов  $\pi_i$ , таких, что  $(\pi_i, \pi_j)$  является инверсией. Другими словами.  $d_j$  – число элементов, больших  $\pi_j$  и стоящих в перестановке слева от него,  $0 \leq d_j < j$ .

Например, вектором инверсий перестановки

$$\Pi = (2, 3, 6, 8, 7, 5, 4, 1)$$

будет

$$\begin{array}{r} j \\ d_j \end{array} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 0 & 0 & 0 & 0 & 1 & 3 & 4 & 7 \end{array}$$

Вектор инверсий  $D = (d_1, d_2, \dots, d_n)$  однозначно определяет перестановку множества  $\{1, 2, \dots, n\}$ . Рассмотрим пример. Предположим, имеется вектор инверсий

$$\begin{array}{r} j \\ d_j \end{array} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 \\ 0 & 1 & 1 & 2 & 1 \end{array}$$

Вычислим соответствующую перестановку. Поскольку  $d_5 = 1$ , имеем  $\pi_5 = 4$ . Затем так как  $d_4 = 2$ , то  $\pi_4 = 2$ . Аналогично, так как  $d_3 = 1$ , то  $\pi_3 = 3$ . Поскольку  $d_2 = 1$ , то  $\pi_2 = 1$ , и поэтому  $\pi_1 = 5$ . Таким образом, перестановка имеет вид  $\Pi = (5, 1, 3, 2, 4)$ .



Поскольку вектор инверсий однозначно определяет перестановку, его можно использовать для порождения всех перестановок, начиная с вектора

$$\begin{array}{cccccc} j & 1 & 2 & 3 & \dots & n \\ d_j & 0 & 0 & 0 & \dots & 0 \end{array}$$

и завершая вектором

$$\begin{array}{cccccc} j & 1 & 2 & 3 & \dots & n \\ d_j & 0 & 1 & 2 & \dots & n-1 \end{array}$$

Однако, этот путь неэффективен, поскольку построение перестановки по ее вектору инверсий требует  $O(n^2)$  операций, что приводит к нелинейному алгоритму.

Польза вектора инверсий (особенно сумма его элементов) заключается в том, что он несет информацию о количестве «беспорядка» в перестановке, т. е. может использоваться в качестве некоторого количественного критерия для оценки степени неотсортированности перестановки. Это будет полезным при анализе некоторых алгоритмов сортировки.

### 3.6.3. Циклические перестановки

Одним из простейших видов перестановок является циклическая перестановка. Циклическая перестановка порядка  $k$  и степени  $d$  есть перестановка, в которой самые левые  $k$  элементов сдвинуты циклически вправо на  $d$  позиций, а положение остальных элементов фиксировано. Например,

$$\Pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 3 & 4 & 5 & 1 & 2 & 6 & 7 & 8 \end{pmatrix}$$

есть циклическая перестановка порядка 5 и степени 3.

Использование циклических перестановок дает способ порождения всех перестановок, который состоит в следующем. Начинается с перестановки  $(1, 2, \dots, n)$  и последовательно сдвигаются по циклу на один разряд все  $n$  элементов. Когда сдвиг по циклу на один разряд первых  $n$  элементов возвращает к ранее порожденной перестановке (получаем тождественную перестановку), сдвигаются по циклу первые  $n - 1$  элементов на один разряд. Если этот шаг возвращает к ранее порожденной перестановке, сдвигаются по циклу первые  $n - 2$  элементов на один разряд. После получения новой перестановки снова реализуется сдвиг по циклу всех  $n$  элементов. Детали процесса порождения перестановок показаны в алгоритме 2.

```
for  $i \leftarrow 1$  to  $n$  do  $\pi_i \leftarrow i$ 
 $k \leftarrow 0$ 
while  $k \neq 1$  do {
  Вывести  $\Pi = (\pi_1, \pi_2, \dots, \pi_n)$ 
   $k \leftarrow n$ 
  Сдвинуть первые  $k$  элементов на одну позицию
  while  $\pi_k = k \neq 1$  do {
     $k \leftarrow k - 1$ 
    Сдвинуть первые  $k$  элементов
    на одну позицию
```

Алгоритм 2. Порождение перестановок циклическим сдвигом

Для  $n = 4$  алгоритм порождает перестановки в следующем порядке (прямоугольником обведены тождественные перестановки, они алгоритмом не включаются в порождаемую последовательность перестановок):

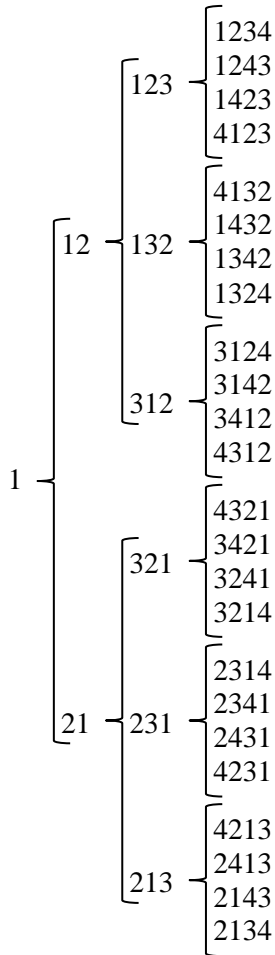
1234	3124	2314	2134	3214	1324	<span style="border: 1px solid black; padding: 2px;">2134</span>
4123	4312	4231	4213	4321	4132	<span style="border: 1px solid black; padding: 2px;">1234</span>
3412	2431	1423	3421	1432	2413	
2341	1243	3142	1342	2143	3241	
<span style="border: 1px solid black; padding: 2px;">1234</span>	<span style="border: 1px solid black; padding: 2px;">3124</span>	<span style="border: 1px solid black; padding: 2px;">2314</span>	<span style="border: 1px solid black; padding: 2px;">2134</span>	<span style="border: 1px solid black; padding: 2px;">3214</span>	<span style="border: 1px solid black; padding: 2px;">1324</span>	
		<span style="border: 1px solid black; padding: 2px;">1234</span>				

Порядок, определяемый этими вложенными циклами, нельзя породить эффективно. Преобразование одной перестановки в следующую может потребовать в худшем случае  $n^2$  транспозиций и всегда требует не менее чем  $n$  транспозиций. Это чрезвычайно неэффективно. Даже если циклический сдвиг рассматривать как элементарную операцию, преобразование в худшем случае все равно требует порядка  $n$  операций, что не дает линейного алгоритма. Данный алгоритм представляет некоторый исторический и теоретический интерес, имеет малую практическую ценность.

#### 3.6.4. Транспозиция смежных элементов

Для минимизации объема работы, необходимого для порождения последовательности перестановок, соседние перестановки в последовательности должны иметь минимальные различия. Лучшее, чего можно достигнуть, – это то, что любая перестановка в последовательности должна отличаться от предшествующей перестановки транспозицией двух соседних элементов.

Такую последовательность перестановок легко построить рекурсивно. Для  $n = 1$  единственная перестановка (1) удовлетворяет требованиям. Предположим, что имеем последовательность  $\Pi_1, \Pi_2, \dots, \Pi_{(n-1)!}$ , перестановок на множестве  $\{1, 2, \dots, n-1\}$ , в которой последовательные перестановки различаются только транспозицией смежных элементов. Будем расширять каждую из этих  $(n-1)!$  перестановок, вставляя элемент  $n$  на каждое из  $n$  возможных мест. Особенность состоит в том, чтобы расположить эти  $n!$  перестановок в порядке, удовлетворяющем требованию минимального изменения. Для этого  $n$  добавляется к  $\Pi_i$  последовательно во все позиции справа налево, если  $i$  нечетно, и слева направо, если  $i$  четно. Порядок порождаемых таким образом перестановок будет следующим:



Очевидно, что практически трудно породить весь список последовательностей целиком. Однако ту же последовательность перестановок можно породить итеративно, получая каждую перестановку из предшествующей ей перестановки и небольшого количества добавочной информации. Это делается с помощью трех векторов: текущей перестановки  $\Pi = (\pi_1, \pi_2, \dots, \pi_n)$ , обратной к ней перестановки  $P = (p_1, p_2, \dots, p_n)$  и направления сдвига  $D = (d_1, d_2, \dots, d_n)$ , где  $d_i = -1$ , если элемент  $i$  сдвигается влево,  $d_i = +1$ , если вправо, и  $d_i = 0$ , если элемент не сдвигается.

Прежде всего, напомним понятие обратной перестановки.

Под *суперпозицией* перестановок  $f$  и  $g$  понимают перестановку  $fg$ , определяемую как  $fg(i) = f(g(i))$ ,  $1 \leq i \leq n$ . Для суперпозиции двух перестановок, скажем

$$f = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 3 & 2 & 1 & 4 \end{pmatrix} \text{ и } g = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 5 & 3 & 1 & 4 \end{pmatrix},$$

достаточно изменить порядок столбцов в перестановке  $f$  таким образом, чтобы в первой строке получить последовательность, имеющуюся во второй строке перестановки  $g$ , тогда вторая строка перестановки  $f$  дает суперпозицию  $fg$ . В нашем случае

$$g = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 5 & 3 & 1 & 4 \end{pmatrix} \quad f = \begin{pmatrix} 2 & 5 & 3 & 1 & 4 \\ 3 & 4 & 2 & 5 & 1 \end{pmatrix} \quad fg = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 4 & 2 & 5 & 1 \end{pmatrix}.$$

Тождественная перестановка обычно обозначается

$$e = \begin{pmatrix} 1 & 2 & 3 & \dots & n \\ 1 & 2 & 3 & \dots & n \end{pmatrix}.$$

Каждая перестановка  $f$  однозначно определяет *обратную* к ней перестановку  $f^{-1}$ , такую, что  $ff^{-1} = f^{-1}f = e$ . Чтобы ее определить, достаточно поменять местами строки в записи перестановки  $f$  и упорядочить по первой строке. Например, для

$$f = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 4 & 2 & 1 & 5 \end{pmatrix} \text{ получаем } f^{-1} = \begin{pmatrix} 3 & 4 & 2 & 1 & 5 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 4 & 3 & 1 & 2 & 5 \end{pmatrix}.$$

Заметим, что по обратной перестановке легко определить позицию элемента в исходной перестановке. Например, по  $f^{-1}$  видно, что элемент 4 находится в позиции 2 в перестановке  $f$ , элемент 2 находится в позиции 3 и т. д.

Вернемся к алгоритму. Элемент сдвигается до тех пор, пока не достигнет элемента, большего, чем он сам; в этом случае сдвиг прекращается. В этот момент направление сдвига данного элемента изменяется на противоположное и передвигается следующий меньший его элемент, который можно сдвинуть. Поскольку хранится перестановка, обратная к  $\Pi$ , то в  $\Pi$  легко найти позицию следующего меньшего элемента.

Детали реализации представлены алгоритмом 3. Заметим, что выполняется присваивание  $\pi_0 \leftarrow \pi_{n+1} \leftarrow n + 1$ , чтобы прекратить передвижение  $n$ , и  $d_1 \leftarrow 0$ , чтобы в тех случаях, когда  $m$  становится равным 1 во внутреннем цикле, остаток внешнего цикла был правильно определен.

```

for  $i \leftarrow 1$  to  $n$  do {  $\pi_i \leftarrow p_i \leftarrow i$ 
                            $d_i \leftarrow -1$ 
                            $d_1 \leftarrow 0$ 
                            $\pi_0 \leftarrow \pi_{n+1} \leftarrow m \leftarrow n + 1$ 
                           while  $m \neq 1$  do { Вывести  $\Pi = (\pi_1, \pi_2, \dots, \pi_n)$ 
                                                 $m \leftarrow n$ 
                                                while  $\pi_{p_m+d_m} > m$  do {  $d_m \leftarrow -d_m$ 
                                                                  $m \leftarrow m - 1$ 
                                                                  $\pi_{p_m} \leftrightarrow \pi_{p_m+d_m}$ 
                                                                 {В ЭТОТ МОМЕНТ  $\pi_{p_m+d_m} = m$ }
                                                                  $p_{\pi_{p_m}} \leftrightarrow p_m$ 

```

Алгоритм 3. Порождение перестановок транспозицией смежных элементов



Этот алгоритм – один из наиболее эффективных алгоритмов для порождения перестановок. Алгоритм линеен, поскольку проверка условия во внутреннем цикле делается всего  $\sum_{i=1}^n i! = n! + o(n!)$  раз. Более эффективный алгоритм получается применением метода макрорасширений, поскольку в  $n - 1$  из каждых  $n$  сдвигов сдвигается элемент  $n$ .

### 3.6.5. Случайные перестановки

Любую из обсуждавшихся выше последовательностей можно использовать для порождения случайных перестановок, поскольку существует четкое соответствие между целыми числами и перестановками. Выбирая случайное число между 0 и  $n! - 1$ , мы тем самым выбираем перестановку. Однако в дополнение к задаче выбора, скажем, случайного целого между 0 и  $52! - 1$  существует еще задача превращения этого числа в перестановку. Это превращение требует порядка  $n^2$  операций.

Эффективный метод порождения случайных перестановок осуществляют последовательности из  $n - 1$  транспозиций. Начиная с любой перестановки  $\Pi = (\pi_1, \pi_2, \dots, \pi_n)$ , элемент  $\pi_n$  переставляется с одним из элементов  $\pi_1, \pi_2, \dots, \pi_n$ , выбираемым случайно. Затем  $\pi_{n-1}$  меняется местами с одним из элементов  $\pi_1, \pi_2, \dots, \pi_{n-1}$ , выбираемым случайно, и т. д. Детали реализации показаны в алгоритме 4.

**for**  $i \leftarrow n$  **downto** 2 **do**  $\begin{cases} j \leftarrow \text{rand}(1, i) \\ \pi_i \leftrightarrow \pi_j \end{cases}$

Алгоритм 1. Порождение случайной перестановки