Основные комбинаторные принципы

Комбинаторика

Комбинаторика – раздел дискретной математики, ориентированный на решение задач выбора и расположения элементов некоторого множества в соответствии с заданными правилами и ограничениями.

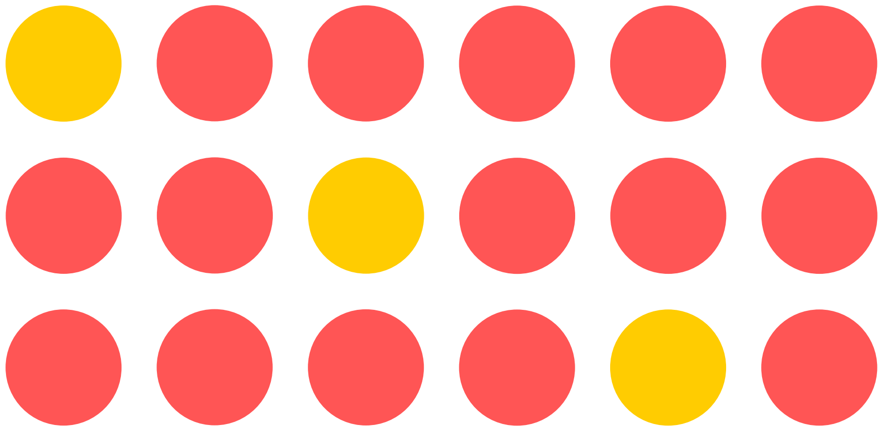
Комбинаторика

Каждое такое правило определяет способ построения некоторой комбинаторной конфигурации, поэтому комбинаторика занимается

- изучением свойств комбинаторных конфигураций;
- условиями их существования;
- алгоритмами построения комбинаторных конфигураций;
- оптимизацией этих алгоритмов.

Две основные задачи комбинаторики

1. Подсчёт комбинаций.
2. Генерация комбинаторных объектов.



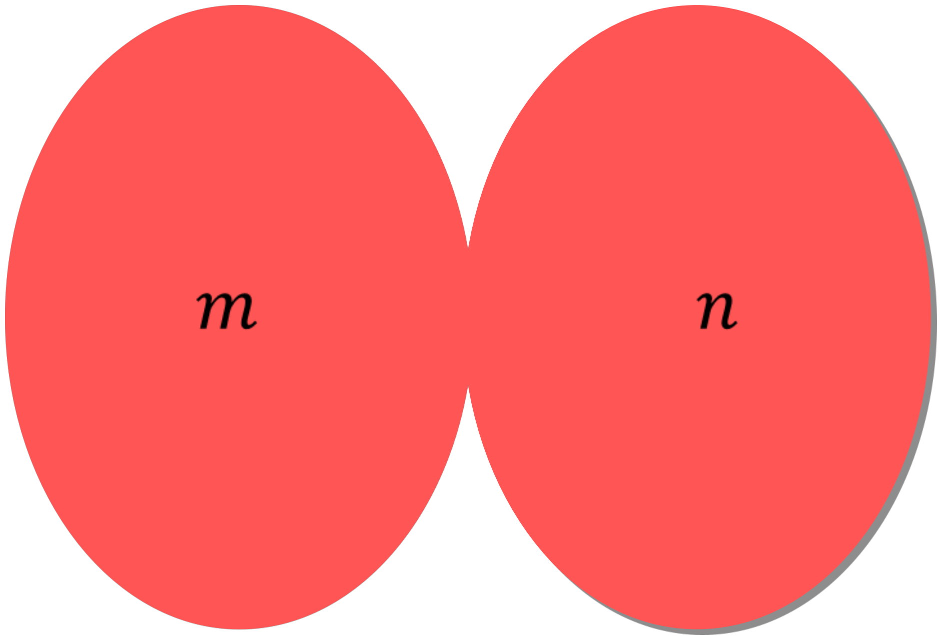
Правила комбинаторики

Правило суммы

Пусть $A \cap B = \emptyset$.
Если элемент $a \in A$ можно выбрать m способами,
а элемент $b \in B$ - n способами, то выбор элемента

$$x \in A \cup B$$

можно осуществить $m + n$ способами.



Правило произведения

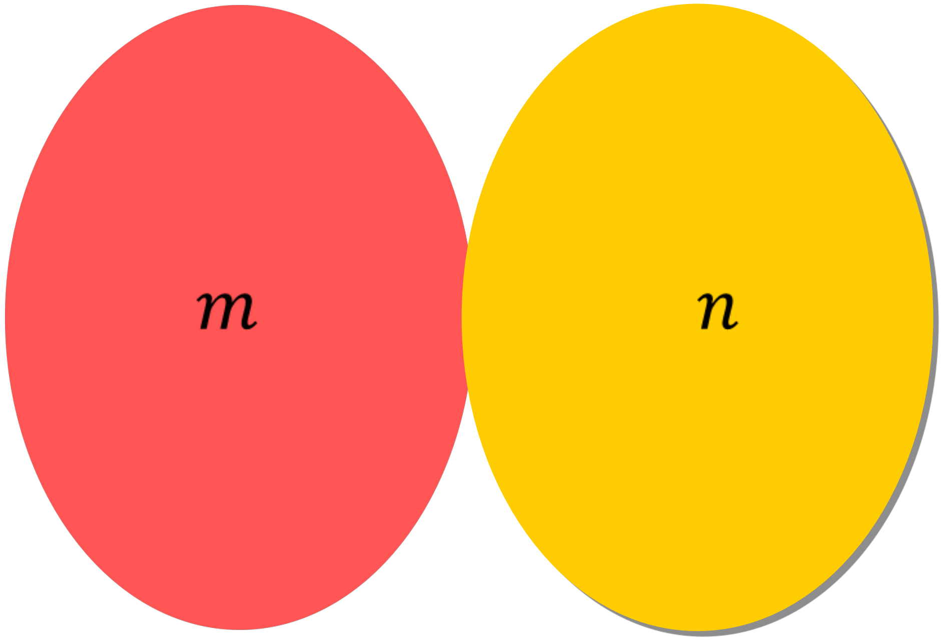
Пусть $A \cap B = \emptyset$.
$a \in A$ можно выбрать m способами, $b \in B$ можно выбрать n способами.
Выбор пары

$$(a, b) \in A \times B$$

можно осуществить

$$|A \times B| = m \cdot n$$

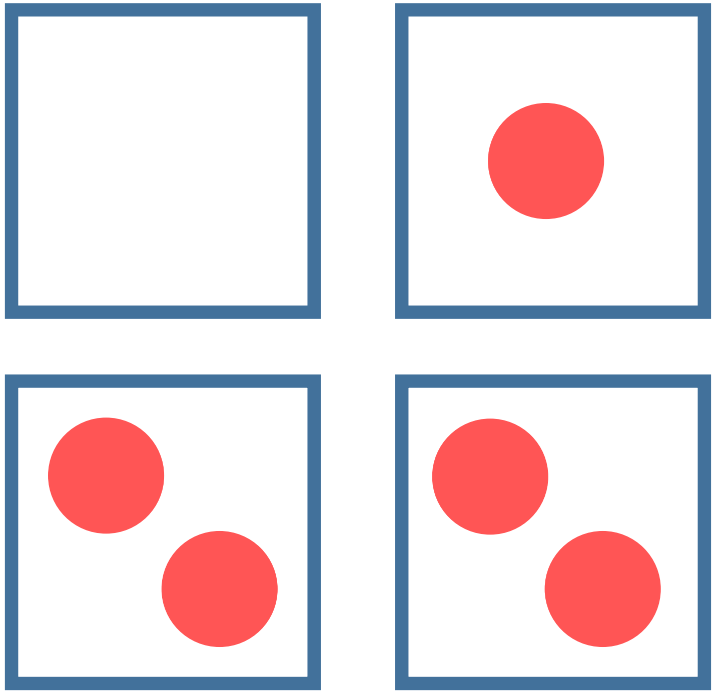
способами.



Принцип Дирихле

Если в 4 клетках сидит 5 кроликов, то по крайней мере в одной клетке сидит не менее двух кроликов.

Если $n + 1$ элемент разбит на n множеств, то по крайней мере одно множество содержит не менее двух элементов.



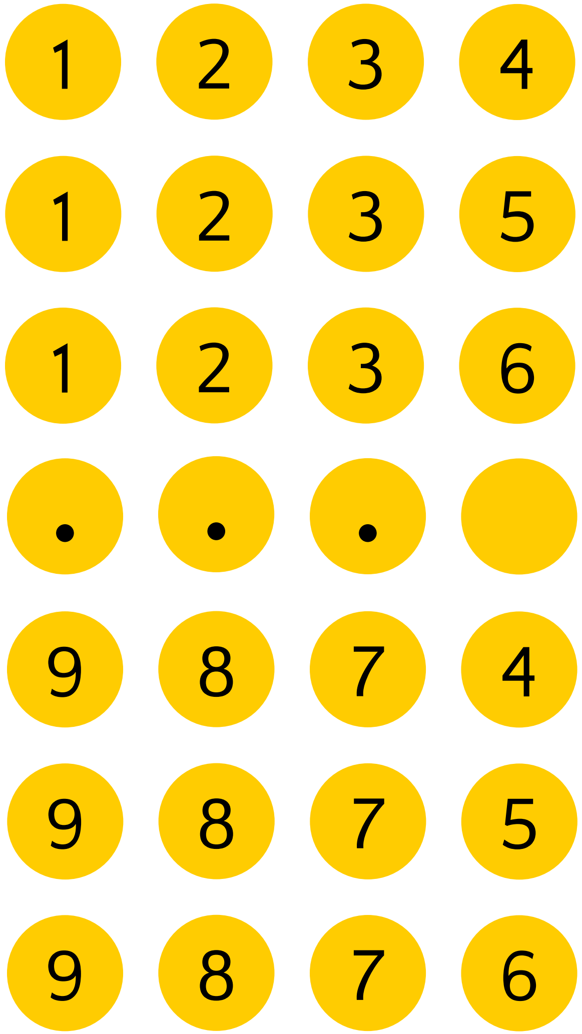
Основные комбинаторные конфигурации

Размещения

Имеются предметы n различных видов $a_1, a_2, \ldots, a_n$. Из них составляют всевозможные расстановки длины k. Например,

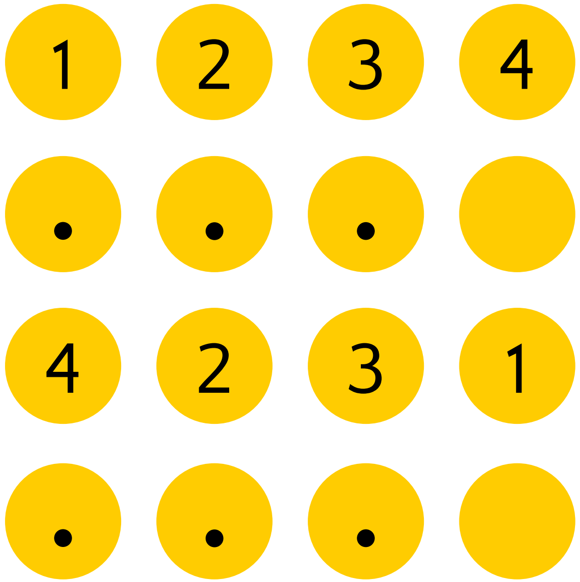
$$a_1 a_2 a_3 a_6$$

- расстановка длины 4.



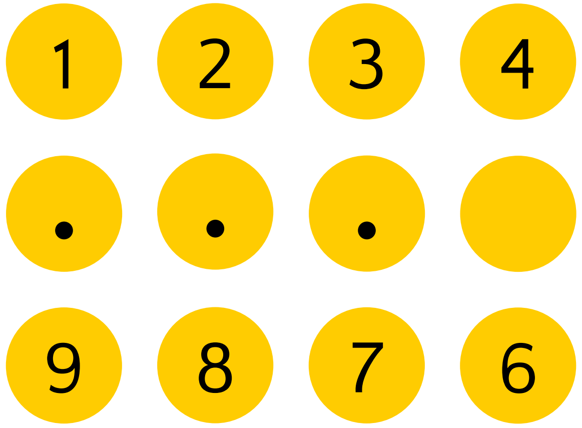
Размещения

Две расстановки считаются различными, если они отличаются видом входящих в них элементов или порядком их в расстановке.



Размещения

Такие расстановки называются **размещениями без повторений**, а их число обозначают A_n^k.



$$A_9^4 = 9 \cdot 8 \cdot 7 \cdot 6 = 3024.$$

$$A_n^k = n \cdot (n-1) \cdot \ldots \cdot (n-k+1).$$

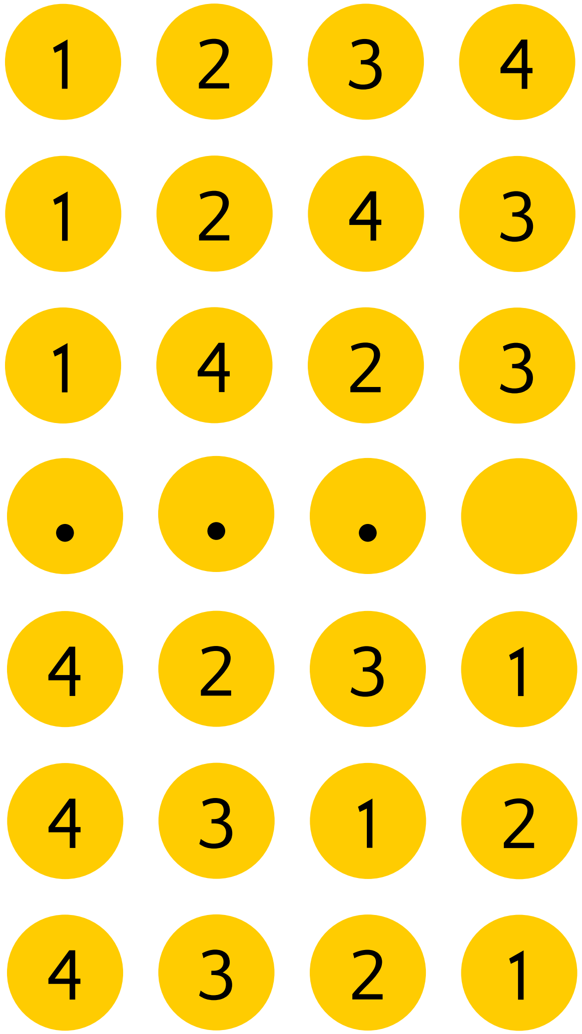
Размещения. Пример

Множество состоит из 17 разных элементов. Выбираются 3 элемента с учётом порядка.

1. Сколькими способами могут быть выбраны 3 элемента?
2. Сгенерировать всевозможные способы выбора трёх элементов.

Перестановки

При составлении размещений без повторений из n по k мы получали расстановки, отличающиеся друг от друга либо составом, либо порядком элементов.
Но если брать расстановки, которые включают все n элементов, то они могут отличаться друг от друга лишь порядком входящих в них элементов.

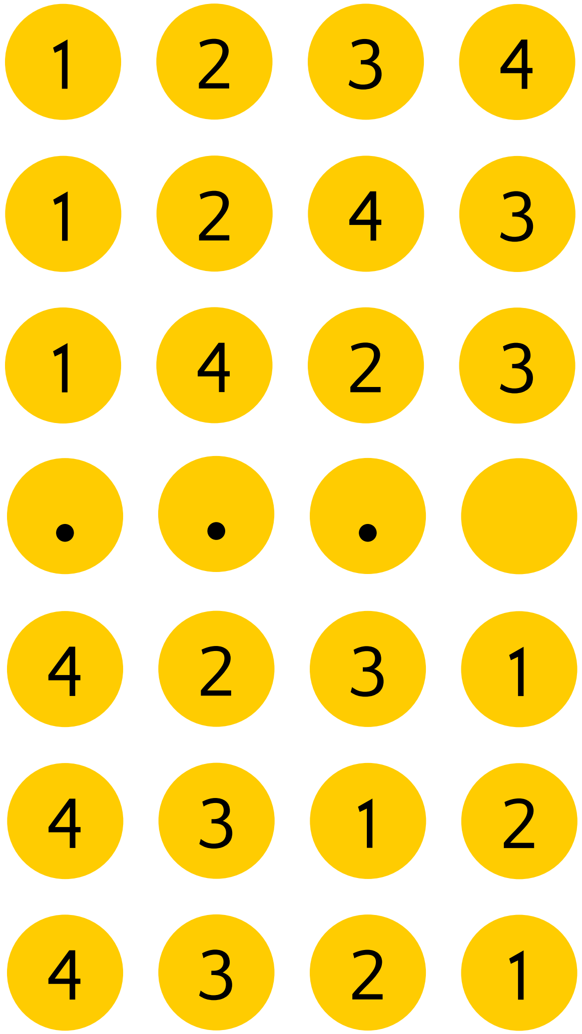


Перестановки

Такие расстановки называются **перестановками** из n элементов, а их число обозначается P_n.

$$P_4 = 4 \cdot 3 \cdot 2 \cdot 1 = 4! = 24$$
$$P_n = n!$$

$$100! = 9.33 \cdot 10^{157}.$$



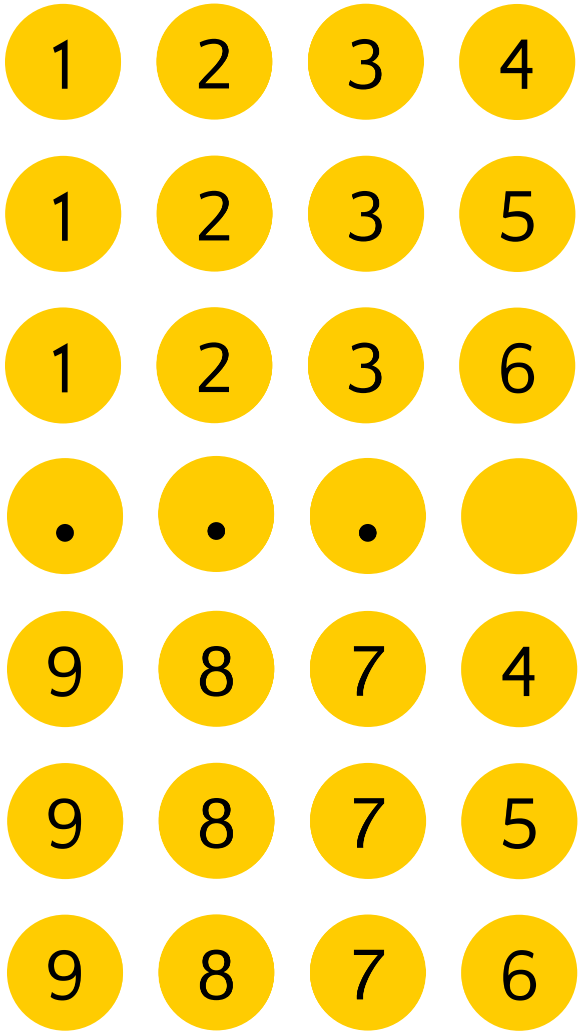
Перестановки. Пример

Множество состоит из 8 элементов. Выбираются все 8 элементов с учётом порядка.

1. Сколькими способами мы можем это сделать?
2. Сгенерировать всевозможные способы выбора 8 элементов.

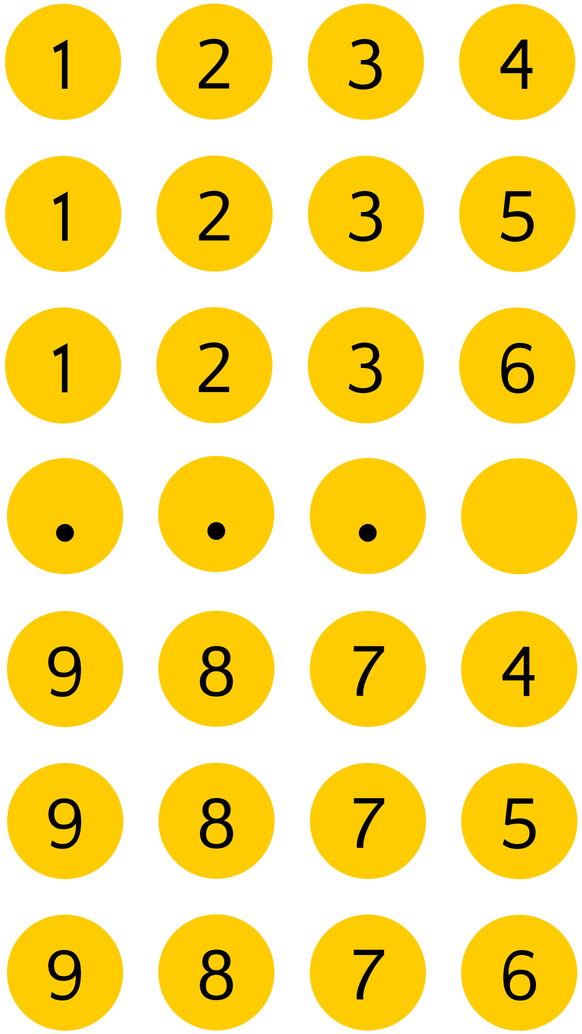
Сочетания

В тех случаях, когда нас не интересует порядок элементов в расстановке, а интересует лишь её состав, то говорят о сочетаниях.



Сочетания

Сочетаниями из n различных элементов по k называют все возможные расстановки длины k, образованные из этих элементов и отличающиеся друг от друга составом, но не порядком элементов.

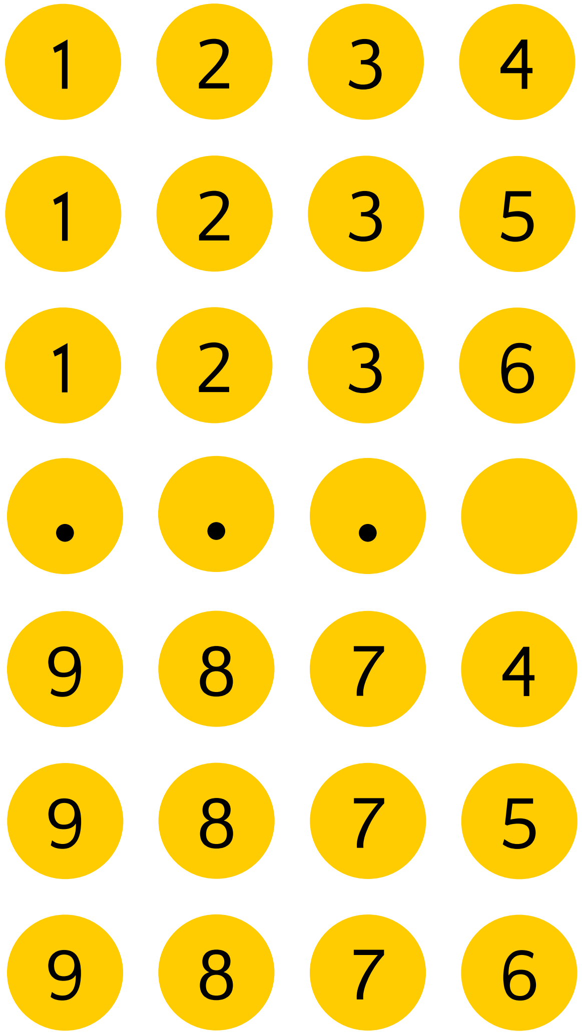


Сочетания

Общее число сочетаний обозначают через C_n^k.

$$C_9^4 = \frac{9 \cdot 8 \cdot 7 \cdot 6}{4 \cdot 3 \cdot 2 \cdot 1}.$$

$$C_n^k = \frac{n \cdot (n-1) \cdot \ldots \cdot (n-k+1)}{k \cdot (k-1) \cdot \ldots \cdot 1}.$$



Сочетания. Пример

Множество состоит их 14 элементов. Мы составляем всевозможные подмножества мощности 3.

1. Сколькими способами мы можем это сделать?
2. Сгенерировать всевозможные подмножества.

Генерация комбинаторных объектов

Порождение комбинаторных объектов

Все рассматриваемые методы систематического порождения комбинаторных объектов будут сводиться

1. К выбору начальной конфигурации, задающей первый генерируемый объект.
2. Трансформации полученного объекта в следующий.
3. Проверке условия окончания, которое определяет момент прекращения вычислений.

Порождение комбинаторных объектов

При этом особый интерес будут представлять алгоритмы генерации объектов в порядке минимального изменения, когда два «соседних» порождаемых объекта различаются в подходящем смысле «минимально».

Лексикографический порядок

На множестве всех перестановок n-элементного множества определим бинарное отношение $\leq$ следующим образом:

$$(\alpha_1, \alpha_2, \dots, \alpha_n) \leq (\beta_1, \beta_2, \dots, \beta_n) \Leftrightarrow$$
$$\Leftrightarrow \exists k \geq 1: (\alpha_k < \beta_k) \text{ и } \forall i < k \ (\alpha_i = \beta_i).$$

$$(1, 3, 2, 4, 5) \leq (1, 3, 4, 2, 5).$$

Лексикографический порядок

Отношение $\leq$ удовлетворяет следующим аксиомам:

1. Рефлексивность. $(\alpha \leq \alpha)$.
2. Антисимметричность. $(\alpha \leq \beta) \wedge (\beta \leq \alpha) \Rightarrow \alpha = \beta$.
3. Транзитивность. $(\alpha \leq \beta) \wedge (\beta \leq \gamma) \Rightarrow (\alpha \leq \gamma)$.
4. Сравнимость. $(\alpha \leq \beta) \vee (\beta \leq \alpha)$.

Лексикографический порядок

Отношение $\leq$ есть линейный порядок на множестве перестановок.

Такой порядок называется **лексикографическим.**

$$123, 132, 213, 231, 312, 321$$

Генерация перестановок в лексикографическом порядке

Алгоритм Нарайаны (Pandita Narayana)

Будем говорить, что перестановка β непосредственно следует за перестановкой α относительно лексикографического порядка если выполняются следующие условия:

1. $\alpha < \beta$, т. е. $\alpha \leq \beta$ и $\alpha \neq \beta$.
2. Не существует такой перестановки γ, что $\alpha < \gamma < \beta$.

Алгоритм Нарайаны. Идея

Генерация перестановок в лексикографическом порядке

1. Начинаем с тождественной перестановки $(1,2,\dots,n)$.
2. Переходим от уже построенной перестановки $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_n)$ к непосредственно следующей за ней перестановке $\beta = (\beta_1, \beta_2, \dots, \beta_n)$.
3. Останавливаемся, как только получим наибольшую перестановку $(n, n-1, \dots, 2, 1)$ (относительно лексикографического порядка).

Алгоритм Нарайаны. Пример

Перестановка

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

За ней следует

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

Как получить следующую перестановку?

Алгоритм Нарайаны. Трансформация

Просматриваем справа налево перестановку

$$\alpha = (\alpha_1, \alpha_2, \dots, \alpha_n)$$

в поисках самой правой позиции i такой, что

$$\alpha_i < \alpha_{i+1}.$$

Если такой позиции нет, то

$$\alpha_1 > \alpha_2 > \cdots > \alpha_n,$$

т. е.

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

и генерировать больше нечего.

Алгоритм Нарайаны. Трансформация

Пусть такая позиция i есть

$$\alpha_i < \alpha_{i+1} > \alpha_{i+2} > \cdots > \alpha_n.$$

Далее ищем первую позицию j при переходе от позиции n к позиции i такую, что

$$\alpha_i < \alpha_j \quad (i \ < \ j).$$

Затем меняем местами элементы α_i и α_j, а в полученной перестановке $\alpha' = (\alpha'_1, \alpha'_2, \dots, \alpha'_n)$ отрезок $\alpha'_{i+1}\alpha'_{i+2} \dots \alpha'_n$ переворачиваем.

Построенную перестановку обозначим через β.

Алгоритм Нарайаны. Пример

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

$$\alpha = (2, 6, \mathbf{5}, 8, 7, 4, 3, 1).$$

$$\alpha = (2, 6, \mathbf{5}, 8, \mathbf{7}, 4, 3, 1).$$

Тогда $\alpha_i = 5$ и $\alpha_j = 7$.

Поменяем местами эти элементы

$$\alpha' = (2, 6, \mathbf{7}, 8, \mathbf{5}, 4, 3, 1).$$

Перевернём отрезок $(8, 5, 4, 3, 1)$ и получим перестановку

$$\beta = (2, 6, 7, \mathbf{1}, \mathbf{3}, \mathbf{4}, \mathbf{5}, \mathbf{8}).$$

Алгоритм Нарайаны

$\alpha_1 = (1, 2, 3, 4)$	$\alpha_i = 3$	$\alpha_j = 4$	$\alpha'_1 = (1, 2, 4, 3)$
$\alpha_2 = (1, 2, 4, 3)$	$\alpha_i = 2$	$\alpha_j = 3$	$\alpha'_2 = (1, 3, 4, 2)$
$\alpha_3 = (1, 3, 2, 4)$	$\alpha_i = 2$	$\alpha_j = 4$	$\alpha'_3 = (1, 3, 4, 2)$
$\alpha_4 = (1, 3, 4, 2)$	$\alpha_i = 3$	$\alpha_j = 4$	$\alpha'_4 = (1, 4, 3, 2)$
$\alpha_5 = (1, 4, 2, 3)$			

Алгоритм Нарайаны

```
for j = 0 to n do α_j = j;
i = 1;
while i ≠ 0 do
begin
  write(α_1, α_2, … , α_n);
  i = n − 1;
  while α_i > α_{i+1} do i = i − 1;
  j = n;
  while α_j < α_i do j = j − 1;
  swap(α_i, α_j);
  k = i + 1;
  m = i + ⌊(n − i)/2⌋;
  while k ≤ m do
   begin
    swap(α_k, α_{n−k+i+1});
    k = k + 1;
  end
end
```

Алгоритм Нарайаны. Использование

```
#include <algorithm>
#include <iostream>
#include <vector>
#include <boost/timer.hpp>

int main()
{
    std::vector<size_t> v = { 7, 1, 12, 5, 9, 20, 15 };
    boost::timer t;
    t.restart();
    std::sort(v.begin(), v.end());
    do
    {
        copy(v.begin(), v.end(), std::ostream_iterator<size_t>(std::cout, " "));
        std::cout << std::endl;
    } while (std::next_permutation(v.begin(), v.end()));
    double duration = t.elapsed();

    std::cout << duration << std::endl;
}
```

Алгоритм Нарайаны

Теорема. Алгоритм Нарайаны корректен и строит все перестановки без повторений в лексикографическом порядке за время $O(n!)$.

Эффективное порождение перестановок

Алгоритм Джонсона – Троттера

Любая перестановка в последовательности должна отличаться от предшествующей транспозицией двух соседних элементов.

Алгоритм Джонсона – Троттера. Рекурсия

Алгоритм Джонсона – Троттера

Данную последовательность перестановок можно порождать итеративно, получая каждую перестановку из предшествующей ей и небольшого количества добавочной информации.

Алгоритм Джонсона – Троттера

Делается с помощью трёх векторов:

1. Текущей перестановки $\pi = (\pi_1, \pi_2, \dots, \pi_n)$.
2. Обратной к ней перестановки $p = (p_1, p_2, \dots, p_n)$.
3. Записи направления d_i, в котором сдвигается каждый элемент i равен
   -1, если он сдвигается влево;
   +1, если вправо;
   0, если не сдвигается.

Алгоритм Джонсона – Троттера

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 π, то в π легко найти место следующего меньшего элемента.

Алгоритм Джонсона – Троттера. Пример

Алгоритм Джонсона – Троттера

$\overrightarrow{\pi} := (\overleftarrow{1}, \overleftarrow{2}, \ldots, \overleftarrow{n});$

$m := 0;$

while $m \neq 1$ **do**

Пока имеется мобильное число m

begin

  write$(\pi_1, \ldots, \pi_n);$

  $m := n;$

Находим наибольшее мобильное число m

  while (m не кандидат для перемещения в π **and** $m > 1$) **do** $m := m - 1;$

  π: $m \leftrightarrow m^*$; (считаем $1^* = 1$)

Меняем местами m и соседнее число, на которое указывает стрелка у m

  $\overrightarrow{\pi}$: над всеми элементами перестановки π, большими m, меняем стрелку на противоположную по направлению

end.

Алгоритм Джонсона – Троттера

for $i = 1$ **to** n **do**

$\pi_i = p_i = i;$

$d_i = -1;$

$d_1 = 0;$

$\pi_0 = \pi_{n+1} = m = n + 1;$ {*метки границ*}

while $m \neq 1$ **do**

write $\pi = (\pi_1, \pi_2, \dots, \pi_n);$

$m = n;$

while $\boldsymbol{\pi}_{p_m + d_m} > m$ **do**

$d_m = -d_m;$

$m = m - 1;$

$\pi_{p_m} \leftrightarrow \pi_{p_m + d_m};$ {*изменить* π}

$p_{\pi_{p_m}} \leftrightarrow p_m$ {*изменить* $p = \pi^{-1}, \pi_{p_m + d_m} = m$}

Алгоритм Джонсона – Троттера

Теорема. Алгоритм Джонсона – Троттера корректен и строит все перестановки без повторений за время $O(n!)$

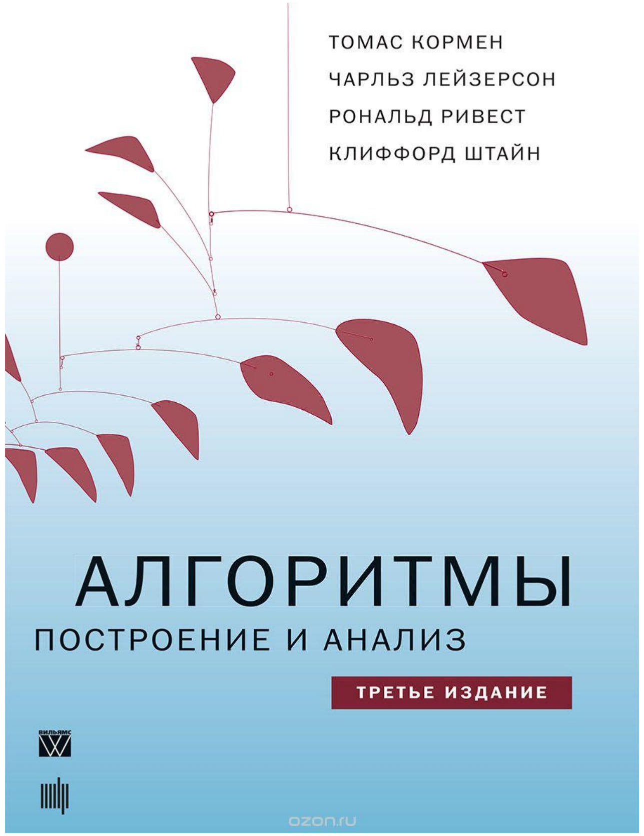
Порождение случайных перестановок

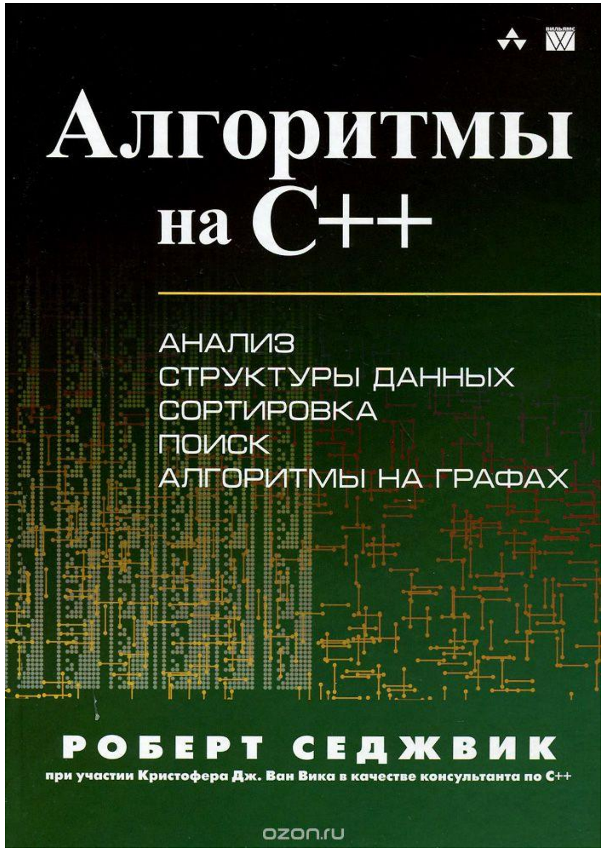
Порождение случайных перестановок

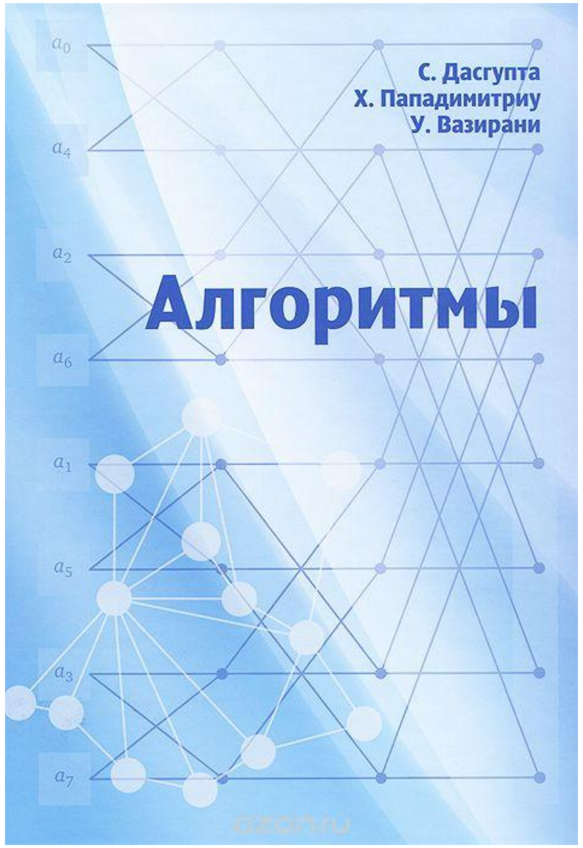
Нужен метод, тасующий карточную колоду.

1. Колода должна быть идеально перемешана.
2. Перестановки карт должны быть равновероятными.
3. Вы можете использовать идеальный генератор случайных чисел.

Литература


ТОМАС КОРМЕН
ЧАРЛЬЗ ЛЕЙЗЕРСОН
РОНАЛЬД РИВЕСТ
КЛИФФОРД ШТАЙН
АЛГОРИТМЫ
ПОСТРОЕНИЕ И АНАЛИЗ
ТРЕТЬЕ ИЗДАНИЕ
ozon.ru


Алгоритмы
на C++
АНАЛИЗ
СТРУКТУРЫ ДАННЫХ
СОРТИРОВКА
ПОИСК
АЛГОРИТМЫ НА ГРАФАХ
РОБЕРТ СЕДЖВИК
при участии Кристофера Дж. Ван Вика в качестве консультанта по C++
ozon.ru


С. Дасгупта
Х. Пападимитриу
У. Вазирани
Алгоритмы

Лабораторная работа 1

1.1. [# 25] Задача коммивояжёра. Для графа, заданного матрицей смежности найти гамильтонов цикл минимальной суммарной стоимости. Решить задачу для $N = 10$, $N = 15$, где N – количество вершин в графе. Оценить время работы программы для входа $N = 20$ и $N = 50$.

Порождение перестановок вершин для поиска гамильтонова цикла провести алгоритмом Нарайаны.

Лабораторная работа 1

1.1. [# 25] Квадратичная задача о назначениях. Есть множество n предприятий, которые могут быть расположены в n местах. Для каждой пары мест задано расстояние и для каждой пары производств задано количество материала, перевозимого между двумя производствами. Требуется расставить производства по местам (два производства нельзя размещать в одном месте) таким образом, что сумма расстояний, умноженных на соответствующие потоки, будет минимальной. Решить задачу для $N = 10$, $N = 15$, где N – количество вершин в графе. Оценить время работы программы для входа $N = 20$ и $N = 50$.

Лабораторная работа 1

n – число объектов и мест их назначения;
c_{ij} – затраты на передачу единицы потока ресурсов из пункта i в пункт j;
q_{kp} – объём ресурсов, направляемых от объекта k к объекту p;
Элементы c_{ij} и q_{kp} задаются в виде соответствующих матриц C и Q размерности $n \times n$.
R – общие затраты, необходимые для обмена ресурсами между всеми объектами.

$$R \to \min.$$

Понижающие коэффициенты

Сдача работы без тестов – коэффициент 0.5.
К задачам с графами необходимы изображения графов для каждого теста.
Тесты не покрывают всевозможные ситуации от 0.5 до 1.0.

Сдача в течении недели после выдачи – коэффициент 1.0.
Сдача в течении двух недель после выдачи – коэффициент 0.8.
Сдача через две недели после выдачи – коэффициент 0.6.

Замер времени

В комбинаторных лабораторных работах требуется выводить время работы программы в зависимости от количества входных данных. Укажите время работы в секундах для
$n = 5; n = 10; n = 20; n = 50; n = 100.$
Для больших данных напишите функцию выполняющую замер времени.
Сравните с возрастом Земли – 4.54 миллиарда лет, с возрастом Вселенной – 13.8 миллиардов лет.

Основные комбинаторные принципы