Перечисление элементарных комбинаторных объектов в лексикографическом порядке

Е. А. Максименко

30 июня 2007 г.

Аннотация

Учебное пособие по перечислению элементарных комбинаторных объектов: кортежей, перестановок, размещений, сочетаний и разбиений. Предлагается хранить эти объекты в виде целочисленных массивов и перечислять в лексикографическом порядке. Рассматриваются два способа перебора: рекурсивный и нерекурсивный. Упражнения на программирование оформлены как описания функций на языке C.

Содержание

0 Введение

Для большинства переборных задач *полный перебор* не является самым эффективным методом решения. Зачем же его изучать? Прежде всего, полный перебор годится в тех случаях, когда размеры задачи малы, а на разработку более эффективного алгоритма не хочется тратить силы. Кроме того, алгоритм полного перебора можно использовать при тестировании более эффективных алгоритмов. В некоторых случаях алгоритм полного перебора можно преобразовать в метод ветвей и границ. Наконец, изучение полного перебора помогает освоить элементарную комбинаторику и некоторые приёмы программирования (работу с массивами и рекурсию).

Существует два способа полного перебора: рекурсивный и некурсивный.

Рекурсивный полный перебор важен из-за того, что довольно часто его можно переделать в перебор с отсечением лишних ветвей («метод ветвей и границ»).

Идея нерекурсивного полного перебора состоит в следующем: из каждого объекта перебираемого множества создавать какой-то *следующий* объект, пока это возможно. Мы будем хранить перебираемые объекты как кортежи (упорядоченные наборы) целых чисел, а в качестве следующего объекта брать *лексикографически следующий*. Нерекурсивный способ перебора замечателен тем, что позволяет легко отделить порождение перебираемых объектов от их использования.

Данное пособие довольно близко по смыслу к первой главе книги [4], но содержит не готовые алгоритмы, а упражнения и подсказки к их решению. Большая часть упражнений сформулирована в виде описаний функций на языке C (см. [5]). Читателю предлагается написать эти функции, заботясь о быстродействии и лаконичности. Кроме этого, есть упражнения на понимание определений, в которых не требуется составлять программы.

Для изучения основ комбинаторики рекомендуются книги [1], [2], [3].

Огромную помощь в составлении и апробации упражнений из этого пособия оказали студенты факультета математики, механики и компьютерных наук Южного Федерального университета:

Е. Н. Андрюшкина,
К. Г. Минасян,
Т. С. Синяк.

1 Линейный порядок

В этом параграфе обсуждаются понятия, связанные с отношением линейного порядка. Эти понятия очень важны для дальнейшего изложения.

Определение 1.1. Бинарное отношение R на множестве X называют *нестрогим порядком* (или просто *порядком*), если оно обладает следующими свойствами:

транзитивность: $\forall a, b, c \in X \quad (aRb \quad \wedge \quad bRc) \quad \Longrightarrow \quad aRc;$

антисимметричность: $\forall a, b \in X \quad (aRb \quad \wedge \quad bRa) \quad \Longrightarrow \quad a = b;$

рефлексивность: $\forall a \in X \quad aRa.$

Определение *строгого порядка* отличается тем, что вместо рефлексивности требуется

антирефлексивность: $\nexists a \in X \quad aRa.$

Например, отношение $<$ на множестве целых чисел $\mathbb{Z}$ является строгим порядком, а отношение $\leqslant$ — нестрогим.

С каждым строгим порядком $<$ принято связывать нестрогий порядок $\leqslant$, определённый следующим образом:

$$x \leqslant y \qquad \Longleftrightarrow \qquad x < y \quad \vee \quad x = y.$$

Обратно, если дан нестрогий порядок $\leqslant$, то можно определить строгий:

$$x < y \qquad \Longleftrightarrow \qquad x \leqslant y \quad \wedge \quad x \neq y.$$

Определение 1.2. Строгий порядок $<$ на множестве X называют *линейным*, если он обладает следующим свойством:

сравнимость любых элементов:

$$\forall a, b \in X \qquad a = b \quad \vee \quad a < b \quad \vee \quad b < a.$$

Пару $(X, <)$ будем называть *линейно упорядоченным множеством*, если $<$ есть линейный строгий порядок на X.

Упражнение 1.1. На множестве $\mathbb{Z}^2$, которое состоит из всевозможных упорядоченных пар целых чисел, введём следующее отношение («покоординатное меньше»):

$$(x_1, x_2) < (y_1, y_2) \quad \iff \quad x_1 < y_1 \quad \wedge \quad x_2 < y_2.$$

Доказать, что это отношение является строгим порядком на $\mathbb{Z}^2$, но не является линейным строгим порядком на $\mathbb{Z}^2$.

Упражнение 1.2. Доказать, что система из трёх свойств: сравнимость любых элементов, антисимметричность и антирефлексивность, — равносильна следующему свойству:

свойство трихотомии: для любых a, b из X выполняется ровно одно из трёх условий: либо $a < b$, либо $a = b$, либо $b < a$.

Таким образом, бинарное отношение является строгим линейным порядком тогда и только тогда, когда оно транзитивно и удовлетворяет свойству трихотомии.

Определение 1.3. Пусть $(X, <)$ — множество с отношением строгого порядка, $Y \subset X$, $z \in X$. Говорят, что z является *верхней границей* (иначе говоря, *мажорантой*) множества Y, если для любого y из Y выполняется неравенство $y \leqslant z$. Аналогично определяются *нижние границы* (*миноранты*).

Определение 1.4. Пусть $(X, <)$ — множество с отношением строгого порядка, $Y \subset X$. Элемент z называют *наибольшим элементом* множества Y, если он принадлежит Y и является мажорантой Y. Элемент z называют *наименьшим элементом* множества Y, если он принадлежит Y и является минорантой Y.

Упражнение 1.3. Пусть X — множество с отношением строгого порядка $<$, $Y \subset X$. Если y_1 — наибольший элемент Y и y_2 — наибольший элемент Y, то $y_1 = y_2$. Аналогичное утверждение верно и для наименьшего элемента.

Упражнение 1.4. Пусть $(X, <)$ — линейно упорядоченное множество, m и M — два новых элемента, которых нет во множестве X, $\bar{X} = X \cup \{m, M\}$, и отношение $\prec$ определено на множестве $\bar{X}$ следующим правилом:

$$\begin{aligned} x \prec y \quad \iff \quad & (x \in X \quad \wedge \quad y \in X \quad \wedge \quad x < y) \quad \vee \\ & \vee \quad (x = m \quad \wedge \quad y \in X) \quad \vee \\ & \vee \quad (x \in X \quad \wedge \quad y = M) \quad \vee \\ & \vee \quad (x = m \quad \wedge \quad y = M). \end{aligned}$$

Доказать, что $\prec$ есть строгий линейный порядок на множестве $\bar{X}$. При этом m является наименьшим элементом в $\bar{X}$, а M — наибольшим.

Конструкцию, описанную в упражнении 1.4, особенно часто применяют в случаях $X = \mathbb{Z}$, $X = \mathbb{Q}$ и $X = \mathbb{R}$. При этом расширенное отношение порядка обозначают тем же символом $<$, что и прежнее, а для новых элементов m и M используют обозначения $-\infty$ и $+\infty$, соответственно.

Далее мы будем часто рассматривать подмножества множества $\mathbb{Z}$. Будем всегда предполагать, что множество $\mathbb{Z}$ и все его подмножества рассматриваются с обычным отношением порядка $<$. Далее, для любых a, b из $\mathbb{Z}$ будем использовать следующие обозначения:

$$[a, b)_{\mathbb{Z}} = \{x \in \mathbb{Z} \colon a \leqslant x < b\},$$
$$[a, b]_{\mathbb{Z}} = \{x \in \mathbb{Z} \colon a \leqslant x \leqslant b\}.$$

Упражнение 1.5. Для множества Y описать все мажоранты и миноранты в упорядоченном множестве X:
1) $X = \mathbb{Z}$, $Y = \{3, 5, 6, 9\}$;
2) $X = \mathbb{Z}$, $Y = \mathbb{Z}$;
3) $X = \mathbb{Z}$, $Y = \varnothing$;
4) $X = \mathbb{Q}$, $Y = \{r \in \mathbb{Q} \colon 3 < r < 5\}$;
5) $X = Y = \mathbb{Z} \cup \{-\infty, +\infty\}$;
6) $X = \mathbb{Z} \cup \{-\infty, +\infty\}$, $Y = \varnothing$;
7) $X = [0, 100]_{\mathbb{Z}}$, $Y = \{3, 5, 6, 9\}$;
8) $X = [0, 100]_{\mathbb{Z}}$, $Y = \varnothing$.

Отдельно остановимся на мажорантах и минорантах пустого множества.

Упражнение 1.6. Пусть $(X, <)$ — упорядоченное множество. Какие элементы множества X будут мажорантами пустого множества? Какие элементы множества X будут минорантами пустого множества?

Упражнение 1.7. Для каждого Y из упражнения 1.5 найти наибольший и наименьший элементы либо доказать их отсутствие.

Определение 1.5. Пусть $(X, <)$ — множество с отношением строгого порядка, $Y \subset X$, $z \in X$. Говорят, что элемент z есть *точная верхняя граница* (или *супремум*) множества Y, если он является наименьшим элементом во множестве всех верхних границ множества Y. Говорят, что элемент z есть *точная нижняя граница* (или *инфимум*) множества Y, если он является наибольшим элементом во множестве всех нижних границ множества Y.

Из результата упражнения 1.3 следует единственность супремума в случае его существования (аналогично для инфимума).

Упражнение 1.8. Пусть $(X, <)$ — множество с отношением строгого порядка, $Y \subset X$, $z \in X$. Доказать, что z является супремумом Y тогда и только тогда, когда выполняются условия:
1) $y \leqslant z$ для любого $y \in Y$;
2) для любого z' из X, такого что $z' < z$, существует такой y, что $y \in Y$ и $z' < y$.

Упражнение 1.9. Пусть $(X, <)$ — множество с отношением строгого порядка, $Y \subset X$, и в Y есть наибольший элемент y. Доказать, что y есть супремум множества Y в X.

Упражнение 1.10. Для каждого из следующих множеств Y найти его супремум и инфимум (если они существуют) в упорядоченном множестве X:
1) $X = \mathbb{Z}$, $Y = \{3, 5, 6, 9\}$;
2) $X = [0, 100]_{\mathbb{Z}}$, $Y = \{3, 5, 6, 9\}$;
3) $X = \mathbb{Z}$, $Y = \mathbb{Z}$;
4) $X = \mathbb{Z}$, $Y = \varnothing$;
5) $X = [0, 100]_{\mathbb{Z}}$, $Y = \varnothing$;
6) $X = \mathbb{Q}$, $Y = \{r \in \mathbb{Q} \colon 3 < r < 5\}$;
7) $X = Y = \mathbb{Z} \cup \{-\infty, +\infty\}$.

Отдельно остановимся на супремуме и инфимуме пустого множества.

Упражнение 1.11. Пусть $(X, <)$ — линейно упорядоченное множество, m — его наименьший элемент. Доказать, что супремум пустого множества в упорядоченном множестве $(X, <)$ равен m.

Упражнение 1.12. Пусть $(X, <)$ — линейно упорядоченное множество, в котором не существует наименьшего элемента. Доказать, что супремум пустого множества в упорядоченном множестве $(X, <)$ не существует.

Упражнение 1.13. Сформулировать и доказать утверждения для инфимума пустого множества, аналогичные утверждениям из упражнений 1.11 и 1.12.

Определение 1.6. Пусть $(X, <)$ — линейно упорядоченное множество; $a, b \in X$. Будем говорить, что b следует за a во множестве X относительно порядка $<$, если $a < b$ и не существует такого $c \in X$, что $a < c$ и $c < b$.

Упражнение 1.14. Доказать, что в X не существует элемента, следующего за a:
1) $X = [0, 10]_{\mathbb{Z}}$, $a = 10$;
2) $X = \mathbb{Q}$, $a = 3$.

Упражнение 1.15. Пусть $(X, <)$ — линейно упорядоченное множество, $a \in X$. Доказать, что если следующий за a элемент существует, то он единствен.

Упражнение 1.16. Дано линейно упорядоченное множество X и элемент a из X. Найти в X элемент b, следующий за a:
1) $X = \mathbb{Z}$, $a = 6$;
2) $X = 2\mathbb{Z} = \{\ldots, -4, -2, 0, 2, 4, \ldots\}$, $a = 6$;
3) $X = 3\mathbb{Z} = \{\ldots, -6, -3, 0, 3, 6, \ldots\}$, $a = 6$;
4) X — множество простых чисел, $a = 23$.

2 Простейшие функции для работы с массивами

Большинство функций, описанных в этом параграфе, используется далее при решении более сложных задач.

Если среди параметров присутствуют массив a и число n, то подразумевается, что n — длина (число элементов) массива a. Всюду предполагается, что $n \geqslant 0$.

Через $\mathbb{Z}^n$ обозначают множество всех кортежей (упорядоченных наборов) длины n, элементы которых являются целыми числами. Нумерацию элементов кортежа будем начинать с нуля. Таким образом, элементы множества $\mathbb{Z}^n$ имеют вид $a = (a_0, a_1, \ldots, a_{n-1})$, где $a_0, a_1, \ldots, a_{n-1} \in \mathbb{Z}$. Кортежи a и b из $\mathbb{Z}^n$ считаются *равными*, если $a_i = b_i$ для любого i из $[0, n)_{\mathbb{Z}}$.

Каждое состояние массива длины n с элементами типа `int` есть целочисленный кортеж длины n, элементы которого принадлежат множеству $[\texttt{INT\_MIN}, \texttt{INT\_MAX}]_{\mathbb{Z}}$. Здесь `INT_MIN` — наименьшее возможное значение переменной типа `int`; `INT_MAX` — наибольшее. Например, если на хранение переменных типа `int` отводится по 32 бита, т. е. `sizeof(int)` равно 4, то `INT_MAX` равно $2^{31} - 1 = 2147483647$, а `INT_MIN` равно $-2^{31} = -2147836648$.

Мы будем часто отождествлять массивы и записанные в них кортежи.

Напомним, что в стандартной библиотеке ввода-вывода языка C, которая имеет заголовочный файл `<stdio.h>`, стандартный выходной поток обозначается через `stdout`. Обычно вывод в `stdout` означает вывод на экран (в терминал пользователя). Для вывода в `stdout` можно использовать функцию `printf`.

Вот описания некоторых простейших функций для работы с массивами. Читателю предлагается написать эти функции и протестировать их работу.

```
void output_array(const int *a, int n)
```

Выводит элементы массива `a` в `stdout`, разделяя пробелами. После последнего элемента выводит символ перехода на новую строку.

```
void fill_array(int *a, int n, int x)
```

Заполняет массив `a` длины n значением x, т. е. записывает в массив `a` кортеж a из $\mathbb{Z}^n$, каждый элемент которого равен x.

Заметим, что адресная арифметика языка C позволяет использовать эту процедуру для заполнения части массива. Например, если в массиве `a` был записан кортеж $(7, 7, 7, 7, 7, 7)$, то после вызова процедуры `fill_array(a+2,3,8)` в массиве `a` будет записан кортеж $(7, 7, 8, 8, 8, 7)$.

`void fill_array_ident(int *a, int n)`

Записывает в массив `a` длины n кортеж $a \in \mathbb{Z}^n$, в котором $a_i = i$ для каждого i из $[0, n)_{\mathbb{Z}}$.

Пример: для $n = 4$ получится $a = (0, 1, 2, 3)$.

`void reverse(int *a, int n)`

«Переворачивает» массив `a` длины `n`.

Пример: делает из массива `2,7,1,4,5` массив `5,4,1,7,2`.

Заметим, что адресная арифметика языка C позволяет использовать эту процедуру для переворачивания части массива.

Пример: если в массиве `a` был записан кортеж $(3, 4, 5, 6, 7, 8)$, то после вызова `reverse(a+2,3)` в массиве `a` будет записан кортеж $(3, 4, 7, 6, 5, 8)$.

`int maximum(const int *a, int n)`

Возвращает максимальный элемент кортежа a. Предполагается, что $a_i \geqslant 0$ для каждого $i \in [0, n)_{\mathbb{Z}}$. Если $n = 0$, то возвращает -1.

`int count_value(const int *a, int n, int v)`

Подсчитывает, сколько раз значение v присутствует в кортеже a.

Пример: для $a = (4, 2, -1, 0, 1, -1)$ и $v = -1$ возвращает 2.

Ниже будут перечислены функции, которые находят среди индексов массива самый первый либо самый последний индекс, обладающий каким-то свойством. Возникает вопрос: что должны возвращать такие функции, если индексов с нужным свойством не существует? Другими словами, какие значения считать супремумом и инфимумом пустого множества? Как можно понять из решения упражнений 1.10, 1.11, 1.12 и 1.13, ответ зависит от того, в каком упорядоченном множестве X ведётся рассмотрение. Множество X должно, во-первых, содержать в себе множество $[0, n)_{\mathbb{Z}}$, а во-вторых, иметь наибольший и наименьший элементы. Рассмотрим следующие варианты:

a) $X = \mathbb{Z} \cup \{-\infty, +\infty\}$. Тогда $\sup \varnothing = -\infty$, $\inf \varnothing = +\infty$. Этот вариант математически безупречен, но не подходит нам из-за того, что у переменной типа `int` нет специальных значений $-\infty$ и $+\infty$.

b) $X = [0, n)_{\mathbb{Z}}$. Тогда $\sup \varnothing = 0$, $\inf \varnothing = n - 1$. Этот вариант плох тем, что $\sup \varnothing$ совпадает с супремумом множества $\{0\}$, а $\inf \varnothing$ — с инфимумом множества $\{n - 1\}$, т. е. для пустого множества могут получаться такие же результаты, как и для непустого.

c) $X = [\mathtt{INT\_MIN}, \mathtt{INT\_MAX}]_{\mathbb{Z}}$, где `INT_MAX` и `INT_MAX` — наименьшее и наибольшее значения, которые может принимать переменная типа `int`. Тогда $\sup \varnothing = \mathtt{INT\_MIN}$, $\inf \varnothing = \mathtt{INT\_MAX}$.

d) $X = [-1, n]_{\mathbb{Z}}$. Тогда $\sup \varnothing = -1$, $\inf \varnothing = n$.

Хотя вариант c) вполне приемлем, более удобен для программирования вариант d), которому и будем следовать. Дело в том, что при поиске наименьшего индекса, обладающего каким-то свойством, естественно просматривать массив слева направо, и если подходящего индекса не найдено, то индексная переменная остановится на значении n. Аналогично, если подходящего индекса не найдено при просмотре массива справа налево, то индексная переменная остановится на значении -1.

`int find_first(const int *a, int n, int x)`

Возвращает наименьший из таких индексов i, что $0 \leqslant i < n$ и $a_i = x$. Если таких индексов не существует, то возвращает n.
Примеры: для $a = (3, 0, 7, 0, 7, 7)$ и $x = 7$ возвращает 2, для $a = (3, 2, 3, 1)$ и $x = 7$ возвращает 4.

`int find_first_not(const int *a, int n, int x)`

Возвращает наименьший из таких индексов i, что $0 \leqslant i < n$ и $a_i \neq x$. Если таких индексов не существует, то возвращает n.
Примеры: для $a = (3, 3, 3, 3, 1, 3, 0)$ и $x = 3$ возвращает 4, для $a = (2, 2, 2)$ и $x = 2$ возвращает 3.

`int find_last(const int *a, int n, int x)`

Возвращает наибольший из таких индексов i, что $0 \leqslant i < n$ и $a_i = x$. Если таких индексов не существует, то возвращает -1.
Примеры: для $a = (2, 4, 1, 2, 1, 2, 4)$ и $x = 2$ возвращает 5, для $a = (1, 4, 3, 3)$ и $x = 2$ возвращает -1.

`int find_last_not(const int *a, int n, int x)`

Возвращает наибольший из таких индексов i, что $0 \leqslant i < n$ и $a_i \neq x$. Если таких индексов не существует, то возвращает -1.
Примеры: для $a = (3, 0, 3, 3)$ и $x = 3$ возвращает 1, а для $a = (3, 3, 3, 3)$ и $x = 3$ возвращает -1.

`int find_first_incr(const int *a, int n)`

Возвращает наименьший из таких индексов i, что $1 \leqslant i < n$ и $a_{i-1} < a_i$. Если таких индексов не существует, то возвращает n.
Примеры: для $a = (4, 3, 1, 2, 0, 5)$ возвращает 3, для $a = (6, 3, 2, 0)$ возвращает 4.

`int find_first_decr(const int *a, int n)`

Возвращает наименьший из таких индексов i, что $1 \leqslant i < n$ и $a_{i-1} > a_i$. Если таких индексов не существует, то возвращает n.

`int find_last_incr(const int *a, int n)`

Возвращает наибольший из таких индексов i, что $0 \leqslant i < n - 1$ и $a_i < a_{i+1}$. Если таких индексов не существует, то возвращает -1.

`int find_last_decr(const int *a, int n)`

Возвращает наибольший из таких индексов i, что $0 \leqslant i < n - 1$ и $a_i > a_{i+1}$. Если таких индексов не существует, то возвращает -1.

3 Хранение множества в булевском массиве

Рассмотрим задачу хранения и обработки произвольного подмножества S множества $[0, m)_{\mathbb{Z}}$, где число m постоянно. Такая задача далее будет возникать неоднократно. Нужно уметь быстро выполнять следующие операции: добавлять элементы к S, удалять элементы из S, проверять, принадлежит ли заданный элемент i множеству S, обходить все элементы множества S и все элементы его дополнения $[0, m)_{\mathbb{Z}} \setminus S$. В такой ситуации удобно использовать булевский массив длины m.

Для булевского типа и булевских констант будем использовать обозначения из языка `C++`: `bool`, `true`, `false`. В языке `C` их можно определить, например, так:

```
typedef char bool;
const bool false = 0;
const bool true = 1;
```

Определение 3.1. Пусть `b` — булевский массив длины m, $S \subset [0, m)_{\mathbb{Z}}$. Будем говорить, что *в булевском массиве* `b` *хранится множество* S, если при любом k из $[0, m)_{\mathbb{Z}}$ условие $k \in S$ равносильно тому, что ячейка `b[k]` имеет значение `true`.

Во всех перечисленных ниже функциях предполагается, что `b` — массив длины m с булевскими элементами.

`void store_empty_set(bool *b, int m)`

Заполняет булевский массив `b` длины m значением false. Таким образом, в конце работы этой процедуры в булевском массиве `b` хранится множество $\varnothing$.

`void store_set(bool *b, int m, const int *a, int n)`

Сохраняет в булевском массиве `b` множество значений, присутствующих в кортеже a из $[0, m)_{\mathbb{Z}}^n$. Другими словами, в конце работы этой процедуры булевская величина `b[k]` показывает, существует ли такой индекс i из $[0, n)_{\mathbb{Z}}$, что $a_i = k$.

Пример: для $m = 4$ и $a = (1, 3, 1)$ в массиве `b` будет записан булевский кортеж $(\mathsf{false}, \mathsf{true}, \mathsf{false}, \mathsf{true})$.

`void output_set(const bool *b, int m)`

Выводит в стандартный поток вывода элементы множества, хранимого

с помощью массива `b`.
Пример: для $b = (\mathsf{true}, \mathsf{false}, \mathsf{false}, \mathsf{true}, \mathsf{true})$ выводит `0 3 4`.

`int first_elem(const bool *b, int m)`

Возвращает наименьший элемент множества, хранимого в виде булевского массива `b`. Если это множество пусто, то возвращает m.
Пример: для $b = (\mathsf{false}, \mathsf{false}, \mathsf{true}, \mathsf{false}, \mathsf{true}, \mathsf{true})$ возвращает 2.

`int first_elem_from(const bool *b, int m, int j)`

Возвращает наименьший элемент множества $S \cap [j, m)_{\mathbb{Z}}$, где S — множество, хранимое в массиве `b`. Если множество $S \cap [j, m)_{\mathbb{Z}}$ пусто, то возвращает m.
Примеры: для $b = (\mathsf{true}, \mathsf{false}, \mathsf{true}, \mathsf{false}, \mathsf{true})$ и $j = 3$ возвращает 4;
для $b = (\mathsf{true}, \mathsf{false}, \mathsf{true}, \mathsf{false}, \mathsf{false})$ и $j = 3$ возвращает 5.

`int first_free(const bool *b, int m)`

Возвращает наименьший элемент множества $[0, m)_{\mathbb{Z}} \setminus S$, где S — множество, хранимое с помощью массива `b`. Если $[0, m)_{\mathbb{Z}} \setminus S = \varnothing$, то возвращает m.
Примеры: для $b = (\mathsf{true}, \mathsf{false}, \mathsf{false}, \mathsf{true})$ возвращает 1;
для $b = (\mathsf{true}, \mathsf{true}, \mathsf{true}, \mathsf{true})$ возвращает 4.

`int first_free_from(const bool *b, int m, int j)`

Возвращает наименьший элемент множества $[j, m)_{\mathbb{Z}} \setminus S$, где S — множество, хранимое с помощью массива `b`. Если $[j, m)_{\mathbb{Z}} \setminus S = \varnothing$, то возвращает m.
Примеры: для $b = (\mathsf{true}, \mathsf{false}, \mathsf{true}, \mathsf{true}, \mathsf{false})$ и $j = 2$ возвращает 4;
для $b = (\mathsf{true}, \mathsf{false}, \mathsf{false}, \mathsf{true}, \mathsf{true})$ и $j = 4$ возвращает 5.

`int set_to_array(const bool *b, int m, int *a)`

По порядку записывает в массив `a` элементы множества, хранимого в булевском массиве `b`, и возвращает число этих элементов. Предполагается, что массив `a` имеет достаточный размер.
Пример: для $b = (\mathsf{false}, \mathsf{true}, \mathsf{false}, \mathsf{true}, \mathsf{true})$ записывает в массив `a` кортеж $(1, 3, 4)$ и возвращает 3.

4 Лексикографический порядок

Определим на множестве $\mathbb{Z}^n$ *отношение лексикографического порядка.*

Определение 4.1. Пусть $a, b \in \mathbb{Z}^n$. Будем говорить, что кортеж a *лексикографически меньше* кортежа b, и писать "$a \prec b$", если существует такой индекс $i \in [0, n)_\mathbb{Z}$, что $a_i < b_i$ и $a_j = b_j$ для всех j из $[0, i)_\mathbb{Z}$. Будем говорить, что a *лексикографически больше* b, и писать "$a \succ b$", если $b \prec a$.

Например, если $a = (2, 3, 3, 0, 5)$, $b = (2, 3, 1, 4, 4)$, то $a \succ b$, т. е. $b \prec a$.

Вот основные функции для работы с лексикографическим порядком:

```
int first_differ(const int *a, const int *b, int n)
```

Находит первое различие в кортежах a и b длины n, т. е. возвращает наименьший из таких индексов i, что $0 \leqslant i < n$ и $a_i \neq b_i$. Если таких индексов не существует, то возвращает n.
Примеры: для $a = (2, 1, 1, 2)$, $b = (2, 1, 2, 0)$ возвращает 2,
для $a = b = (1, 3, 0, 3)$ возвращает 4.

```
int lex_compare(const int *a, const int *b, int n)
```

Производит лексикографическое сравнение кортежей a и b длины n:

$$\text{возвращает} \begin{cases} 1, & \text{если } a \succ b; \\ 0, & \text{если } a = b; \\ -1, & \text{если } a \prec b. \end{cases}$$

Упражнение 4.1. Доказать, что для лексикографического порядка выполняются свойства строгого линейного порядка (см. определение 1.2 и упражнение 1.2):

транзитивность: для любых $a, b, c \in \mathbb{Z}^n$ из условий $a \prec b$ и $b \prec c$ следует, что $a \prec c$;

закон трихотомии: для любых $a, b \in \mathbb{Z}^n$ выполняется ровно одно из следующих трёх условий: 1) $a \prec b$; 2) $a = b$; 3) $a \succ b$.

5 Перечисление кортежей

Обозначим через $[0, m)_{\mathbb{Z}}^n$ множество всех кортежей длины n, элементы которых принадлежат множеству $[0, m)_{\mathbb{Z}}$.

Множество $[0, m)_{\mathbb{Z}}^n$ вложено в $\mathbb{Z}^n$, поэтому на $[0, m)_{\mathbb{Z}}^n$ определён лексикографический порядок.

В соответствии с общим определением 1.6, будем говорить, что кортеж b является *лексикографически следующим* после кортежа a во множестве $[0, m)_{\mathbb{Z}}^n$, если $a, b \in [0, m)_{\mathbb{Z}}^n$, $a \prec b$ и не существует такого c из $[0, m)_{\mathbb{Z}}^n$, что $a \prec c$ и $c \prec b$.

Например, лексикографически следующим после $(3, 0, 4, 4)$ во множестве $[0, 6)_{\mathbb{Z}}^4$ является кортеж $(3, 0, 4, 5)$, а лексикографически следующим после $(3, 0, 4, 4)$ во множестве $[0, 5)_{\mathbb{Z}}^4$ является кортеж $(3, 1, 0, 0)$.

Упражнение 5.1. Вручную выписать в лексикографическом порядке все элементы множества $[0, m)_{\mathbb{Z}}^n$ сначала для $n = 4$ и $m = 2$, затем для $n = 3$ и $m = 3$.

Упражнение 5.2. Найти критерий в терминах элементов кортежей a и b, когда b является лексикографически следующим после a во множестве $[0, m)_{\mathbb{Z}}^n$. Сначала можно объяснить на примере, почему во множестве $[0, 7)_{\mathbb{Z}}^4$ кортеж $(3, 2, 0, 0)$ является лексикографически следующим за $(3, 1, 6, 6)$.

```
bool is_next_tuple(const int *a, const int *b, int n, int m)
```

Выясняет, является ли кортеж b лексикографически следующим после a во множестве $[0, m)_{\mathbb{Z}}^n$. Предполагается, что $a, b \in [0, m)_{\mathbb{Z}}^n$.

Упражнение 5.3. Сформулировать правило, как по заданному кортежу из $[0, m)_{\mathbb{Z}}^n$ строить лексикографически следующий. Например, во множестве $[0, 5)_{\mathbb{Z}}^5$ для кортежа $(4, 2, 0, 4, 4)$ лексикографически следующим является $(4, 2, 1, 0, 0)$. Нужно объяснить, как выбирается самый левый изменяемый элемент, как он изменяется и как изменяются последующие элементы.

```
int index_to_increase_tuple(const int *a, int n, int m)
```

Возвращает наибольший из таких индексов i, что для заданного кортежа a из $[0, m)_{\mathbb{Z}}^n$ можно построить лексикографически больший кортеж во множестве $[0, m)_{\mathbb{Z}}^n$, не изменяя элементов, индексы которых меньше i. Если таких индексов i не существует, то возвращает -1.
Пример: для $a = (3, 0, 4, 4)$ и $m = 5$ возвращает 1.

`bool next_tuple(int *a, int n, int m)`

Если во множестве $[0, m)^n_{\mathbb{Z}}$ существует кортеж, лексикографически следующий за данным кортежем a из $[0, m)^n_{\mathbb{Z}}$, то записывает его в тот же массив `a` и возвращает `true`, иначе оставляет массив `a` без изменений и возвращает `false`.

Примеры: для $m = 3$ и $a = (2, 0, 2, 2)$ записывает в массив `a` кортеж $(2, 1, 0, 0)$ и возвращает `true`;

для $m = 4$ и $a = (3, 3, 3, 3)$ оставлят массив `a` без изменений и возвращает `false`.

`void output_all_tuples(int n, int m)`

Выводит в стандартный поток вывода все кортежи из $[0, m)^n_{\mathbb{Z}}$ в лексикографическом порядке, используя следующие функции:

`fill_array`, `output_array` и `next_tuple`.

Теперь займёмся перечислением кортежей с помощью рекурсии:

`void output_tuples_recur(int *a, int n, int m, int i)`

Выводит в стандартный поток вывода в лексикографическом порядке все кортежи из $[0, m)^n_{\mathbb{Z}}$, у которых элементы с индексами $< i$ совпадают с заданными элементами массива `a`. В частности, при $i \geqslant m$ просто выводит тот кортеж, который передан в массиве `a`, а при $i = 0$ выводит в лексикографическом порядке все элементы $[0, m)^n_{\mathbb{Z}}$.

Объясним принцип работы этой рекурсивной функции. На i-м уровне рекурсии должны перебираться значения i-го элемента кортежа. Представим, например, что $n = 4$, $m = 3$ и $i = 2$. Значит, в качестве параметра a передан кортеж, в котором уже выбраны значения первых двух элементов. Пусть это $a = (2, 1, *, *)$. Звёздочки показывают, что значения последних двух элементов ещё не выбраны (нам неважно, что там записано, так как эти значения мы сейчас будем перезаписывать). Нужно перебрать все возможные значения элемента с индексом 2:

$$\begin{aligned} k = 0: &\quad a = (2, 1, 0, *), \\ k = 1: &\quad a = (2, 1, 1, *), \\ k = 2: &\quad a = (2, 1, 2, *), \end{aligned}$$

и для каждого варианта вызвать `output_tuples_recur`, указывая уровень рекурсии 3. На третьем уровне рекурсии будут по очереди перебираться все возможные значения элемента с индексом 3. Наконец, на самом глубоком уровне рекурсии (в нашем примере это $i = 4$) перебирать уже нечего, а нужно просто выводить переданный кортеж в стандартный поток вывода.

Изобразим это в виде схемы. Для каждого уровня рекурсии покажем, какой кортеж передаётся на этот уровень и как изменяется локальная переменная k:

$$
\begin{array}{r|r|l}
\qquad i = 2 \qquad & \qquad i = 3 \qquad & \qquad i = 4 \\
(2, 1, *, *) & & \\
k = 0 & (2, 1, 0, *) \quad & \\
& k = 0 & (2, 1, 0, 0) \to \texttt{stdout} \\
& k = 1 & (2, 1, 0, 1) \to \texttt{stdout} \\
& k = 2 & (2, 1, 0, 2) \to \texttt{stdout} \\
k = 1 & (2, 1, 1, *) \quad & \\
& k = 0 & (2, 1, 1, 0) \to \texttt{stdout} \\
& k = 1 & (2, 1, 1, 1) \to \texttt{stdout} \\
& k = 2 & (2, 1, 1, 2) \to \texttt{stdout} \\
k = 2 & (2, 1, 2, *) \quad & \\
& k = 0 & (2, 1, 2, 0) \to \texttt{stdout} \\
& k = 1 & (2, 1, 2, 1) \to \texttt{stdout} \\
& k = 2 & (2, 1, 2, 2) \to \texttt{stdout}
\end{array}
$$

Заметим, что конечным результатом вызова нашей рекурсивной функции с параметрами $n = 4$, $m = 3$, $i = 2$, $a = (2, 1, *, *)$ будет вывод всех кортежей из $[0, 3)^4_{\mathbb{Z}}$, начинающихся с $(2, 1)$.

Упражнение 5.4. Показать работу функции `output_tuples_recur` в виде схемы указанного вида для $n = 3$, $m = 2$, $i = 0$, $a = (*, *, *)$.

После написания функции `output_tuples_recur` можно воспользоваться ей для вывода *всех* элементов множества $[0, m)^n_{\mathbb{Z}}$:

```
void output_all_tuples_recur(int n, int m)
```

Выводит в стандартный поток вывода в лексикографическом порядке все кортежи из $[0, m)^n_{\mathbb{Z}}$. Использует функцию `output_tuples_recur`.

6 Перечисление перестановок

Определение 6.1. *Перестановкой* множества X называют любое биективное отображение этого множества в себя.

Определение 6.2. Множество всех перестановок множества $[0, n)_{\mathbb{Z}}$ будем обозначать через $\mathrm{Permuts}(n)$.

Перестановку $\varphi\colon [0, n)_{\mathbb{Z}} \to [0, n)_{\mathbb{Z}}$ будем отождествлять с кортежем a из $[0, n)_{\mathbb{Z}}^n$, в котором $a_i = \varphi(i)$ для любого $i \in [0, n)_{\mathbb{Z}}^n$.

Пример:

$$\text{отображение} \quad f\colon \begin{array}{cccc} 0 & 1 & 2 & 3 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ 2 & 1 & 3 & 0 \end{array} \quad \text{отождествляем с кортежем} \quad a = (2, 1, 3, 0).$$

Условия инъективности и сюръективности отображения f принимают следующий вид для кортежа a:

1) $a_i \neq a_j$ при любых различных $i, j \in [0, n)_{\mathbb{Z}}$;
2) для любого $j \in [0, n)_{\mathbb{Z}}$ существует такой индекс $i \in [0, n)_{\mathbb{Z}}$, что $a_i = j$.

На самом деле, можно доказать, что условия 1) и 2) равносильны, так что достаточно требовать одного из них.

Множество всех кортежей, удовлетворяющих этим условиям, будем обозначать через $\mathrm{Permuts}(n)$, как и множество соответствующих им перестановок.

```
bool is_permut(const int *a, int n)
```

Проверяет, является ли данный кортеж a из $\mathbb{Z}^n$ перестановкой множества $[0, n)_{\mathbb{Z}}$.

```
void output_all_permuts_slow(int n)
```

Выводит в `stdout` в лексикографическом порядке все элементы множества $\mathrm{Permuts}(n)$, используя функции `next_tuple` и `is_permut`.

Способ перечисления всех перестановок, который предложен в функции `output_all_permuts_slow`, очень неэффективен: между двумя лексикографически соседними перестановками может быть довольно много кортежей, которые приходится пропускать.

Как известно, число перестановок n-элементного множества равно $n!$ (см. далее упражнение 6.5):

$$|\mathrm{Permuts}(n)| = n! = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 2 \cdot 1.$$

Упражнение 6.1. Вычислить отношение $\dfrac{|\operatorname{Permuts}(n)|}{|[0,n)_{\mathbb{Z}}^{n}|}$ при $n = 8$.

Теперь нужно научиться сразу превращать произвольную перестановку в лексикографически следующую.

Упражнение 6.2. Вручную выписать в лексикографическом порядке все перестановки множества $[0,n)_{\mathbb{Z}}$ для $n = 3$ и $n = 4$. Проверить себя с помощью функции `output_all_permuts_slow`.

Упражнение 6.3. Сформулировать правило, по которому для каждой перестановки строится лексикографически следующая. Например, во множестве $\operatorname{Permuts}(5)$ для перестановки $(1,0,4,3,2)$ лексикографически следующей является $(1,2,0,3,4)$. Объяснить, как выбирается самый левый изменяемый элемент, как он изменяется и как изменяются последующие элементы.

`int max_index_to_incr_permut(const int *a, int n)`

Возвращает наибольший среди таких индексов i из $[0,n)_{\mathbb{Z}}$, что для заданной перестановки a из $\operatorname{Permuts}(n)$ можно построить лексикографически бо́льшую, не меняя элементов с индексами, меньшими i. Если таких индексов не существует, то возвращает -1.

`bool next_permut(int *a, int n)`

Если для заданной перестановки a из $\operatorname{Permuts}(n)$ существует лексикографически следующая перестановка из $\operatorname{Permuts}(n)$, то записывает её в тот же массив `a` и возвращает `true`; иначе оставляет массив `a` без изменений и возвращает `false`.

`void output_all_permuts(int n)`

Выводит в стандартный поток вывода все перестановки множества $[0,n)_{\mathbb{Z}}$ в лексикографическом порядке, используя следующие функции: `fill_array_ident`, `output_array` и `next_permut`.

Переходим к перечислению перестановок с помощью рекурсии.

Упражнение 6.4. Пусть $a \in \operatorname{Permuts}(5)$. Какие значения может принимать a_2, если $a_0 = 4$, $a_1 = 0$? Вручную перечислить в лексикографическом порядке все перестановки множества $[0,5)_{\mathbb{Z}}$, которые имеют вид $(4,0,*,*,*)$.

Из упражнения 6.4 можно сделать вывод, что на i-м уровне рекурсии нужно перебирать значения, не использованные на предыдущих уровнях.

Упражнение 6.5. Доказать, что $|\operatorname{Permuts}(n)| = n!$.

Эффективным способом для запоминания использованных и неиспользованных значений является булевский массив (см. § 3). Будем передавать булевский массив в качестве параметра рекурсивной функции.

`void output_permuts_recur(int *a, int n, int i, bool *b)`

Выводит в стандартный поток вывода в лексикографическом порядке все такие кортежи из $\operatorname{Permuts}(n)$, у которых элементы с индексами $< i$ совпадают с заданными элементами массива `a`. Предполагается, что $0 \leqslant i \leqslant n$, элементы массива `a` с индексами $< i$ попарно различны и принадлежат множеству $[0, m)_{\mathbb{Z}}$. В булевском массиве `b` передаётся множество $\{a_k \colon 0 \leqslant k < i\}$.

Вот схема работы функции `output_permut_recur` при $a = (2, 0, *, *)$, $i = 2$ (для краткости пишем 0 вместо false и 1 вместо true):

$i = 2$	$i = 3$	$i = 4$
$a = (2, 0, *, *)$ $b = (1, 0, 1, 0)$		
$k = 1$	$a = (2, 0, 1, *)$ $b = (1, 1, 1, 0)$	
	$k = 3$	$a = (2, 0, 1, 3) \to$ `stdout`
$k = 3$	$a = (2, 0, 3, *)$ $b = (1, 0, 1, 1)$	
	$k = 1$	$a = (2, 0, 3, 1) \to$ `stdout`

Упражнение 6.6. Изобразить схему работы функции `output_permut_recur` для $n = 3$, $a = (*, *, *)$, $i = 0$.

`void output_all_permuts_recur(int n)`

Выводит в `stdout` в лексикографическом порядке все элементы множества $\operatorname{Permuts}(n)$. Использует функцию `output_permuts_recur`.

7 Перечисление размещений

Определение 7.1. Пусть m, n — целые числа, $0 \leqslant n \leqslant m$. *Размещениями* длины n множества $[0, m)_{\mathbb{Z}}$ называют инъективные отображения множества $[0, n)_{\mathbb{Z}}$ в $[0, m)_{\mathbb{Z}}$. Множество всех размещений длины n множества $[0, m)_{\mathbb{Z}}$ обозначим через $\mathrm{Injects}(n, m)$.

Размещение $f\colon [0, n)_{\mathbb{Z}} \to [0, m)_{\mathbb{Z}}$ будем отождествлять с кортежем a из $[0, m)_{\mathbb{Z}}^n$, в котором $a_i = f(i)$. При этом условие инъективности принимает вид

$$\forall i, j \in [0, n)_{\mathbb{Z}} \qquad i \neq j \quad \Longrightarrow \quad a_i \neq a_j.$$

Множество всех таких кортежей будем обозначать через $\mathrm{Injects}(n, m)$, как и множество соответствующих им размещений.

Можно перечислять размещения, просто выделяя их среди всех кортежей:

```
bool is_inject(const int *a, int n, int m)
```

Проверяет, принадлежит ли данный кортеж a длины n множеству $\mathrm{Injects}(n, m)$.

```
void output_all_injects_slow(const int *a, int n, int m)
```

Выводит в `stdout` все размещения длины n множества $[0, m)_{\mathbb{Z}}$, используя функции `next_tuple` и `is_inject`.

Конечно, такой способ неэффективен.

Упражнение 7.1. Вычислить $\dfrac{|\mathrm{Injects}(n, m)|}{|[0, m)^n|}$ при $n = 5$, $m = 10$.

```
int max_index_to_incr_inject(const int *a, int n, int m)
```

Возвращает наибольший среди таких индексов i из $[0, n)_{\mathbb{Z}}$, что для заданного размещения a из $\mathrm{Injects}(n, m)$ можно построить лексикографически бо́льший элемент в $\mathrm{Injects}(n, m)$, не меняя элементов с индексами, меньшими i. Если таких индексов не существует, то возвращает -1.

```
bool next_inject(int *a, int n, int m)
```

Если для заданного a из $\mathrm{Injects}(n, m)$ во множестве $\mathrm{Injects}(n, m)$ существует лексикографически больший элемент, то записывает его в тот же массив `a` и возвращает `true`; иначе оставляет массив `a` неизменным и возвращает `false`.

`void output_all_injects(int n, int m)`

Выводит в `stdout` все элементы множества $\mathrm{Injects}(n, m)$ в лексикографическом порядке. Использует следующие функции: `fill_ident`, `output_array`, `next_inject`.

Рекурсивное перечисление размещений очень похоже на рекурсивное перечисление перестановок.

`void output_injects_recur(int *a, int n, int m, int i, bool *b)`

Выводит в стандартный поток вывода в лексикографическом порядке все размещения из $\mathrm{Injects}(n, m)$, у которых элементы с индексами $< i$ совпадают с заданными элементами массива `a`. Предполагается, что $0 \leqslant i \leqslant n$, элементы массива `a` с индексами $< i$ попарно различны и принадлежат множеству $[0, m)_{\mathbb{Z}}$. В булевском массиве `b` хранится множество $\{a_k : 0 \leqslant k < i\}$.

`void output_all_injects_recur(int n, int m)`

Выводит в `stdout` в лексикографическом порядке все элементы множества $\mathrm{Injects}(n, m)$. Использует функцию `output_injects_recur`.

8 Перечисление сочетаний

Всюду в этом параграфе будем считать, что $0 \leqslant n \leqslant m$.

Определение 8.1. *Сочетаниями* длины n из элементов множества X называют n-элементные подмножества X. Множество всех сочетаний длины n из элементов множества $[0, m)_{\mathbb{Z}}$ будем обозначать через $\mathrm{Subsets}(n, m)$.

Заметим, что каждому размещению можно сопоставить сочетание:

$$(a_0, a_1, \ldots, a_{n-1}) \quad \mapsto \quad \{a_0, a_1, \ldots, a_{n-1}\}.$$

При этом разным размещениям может соответствовать одно и то же сочетание. Пример: $(3, 0, 1)$ и $(1, 0, 3)$ — разные размещения, но соответствуют одному и тому же сочетанию.

Упражнение 8.1. Назвать все размещения, которые соответствуют тому же сочетанию, что и размещение $(3, 0, 1)$.

Упражнение 8.2. Выяснить, сколько различных размещений соответствует одному сочетанию из $\mathrm{Subsets}(n, m)$.

Из результатов упражнений 6.5 и 8.2 следует формула для числа элементов множества $\mathrm{Subsets}(n, m)$:

$$\frac{m!}{n!(m-n)!}.$$

Это число обозначают через C_m^n или $\binom{m}{n}$.

Чтобы сделать соответствие между размещениями и сочетаниями биективным, будем рассматривать лишь возрастающие размещения, т. е., что эквивалентно, возрастающие кортежи.

Итак, каждый кортеж из $[0, m)_{\mathbb{Z}}^n$ вида $a = (a_0, a_1, \ldots, a_{n-1})$, где $a_0 < a_1 < \ldots < a_{n-1}$, будем отождествлять с множеством $\{a_0, a_1, \ldots, a_{n-1}\}$.

Множество всех таких кортежей будем обозначать через $\mathrm{Subsets}(n, m)$, как и множество соответствующих сочетаний.

```
bool is_combin(const int *a, int n)
```

Выясняет, является ли данный кортеж a из $\mathbb{Z}^n$ возрастающим.

```
void output_all_combins_slow(int n, int m)
```

Выводит в `stdout` в лексикографическом порядке все элементы множества $\mathrm{Subsets}(n, m)$, используя функции `next_tuple` и `is_combin`.

Метод, предлагаемый в функции `output_all_combins_slow`, очень неэффективен.

Упражнение 8.3. Вычислить отношение $\dfrac{|\operatorname{Subsets}(n,m)|}{|[0,m)^n_{\mathbb{Z}}|}$ при $n=5$, $m=10$.

Нужно научиться для каждого возрастающего кортежа сразу строить лексикографически следующий возрастающий кортеж.

Упражнение 8.4. Вручную выписать в лексикографическом порядке все элементы множества $\operatorname{Subsets}(n,m)$ для $n=3$, $m=5$. Проверить себя с помощью функции `output_all_combins_slow`.

Упражнение 8.5. Сформулировать правило, по которому для каждого возрастающего кортежа строится лексикографически следующий. Например, во множестве $\operatorname{Subsets}(3,7)$ для возрастающего кортежа $(0,1,5,6)$ лексикографически следующим является $(0,2,3,4)$. Объяснить, как выбирается самый левый изменяемый элемент, как он изменяется и по какому правилу строятся последующие элементы.

`int index_to_incr_combin(const int *a, int n, int m)`

Возвращает наибольший из таких индексов i, что для кортежа a из множества $\operatorname{Subsets}(n,m)$ можно построить лексикографически больший кортеж из $\operatorname{Subsets}(n,m)$, не изменяя элементов, индексы которых меньше i. Если таких индексов i не существует, то возвращает -1.

`bool next_combin(int *a, int n, int m)`

Если в $\operatorname{Subsets}(n,m)$ существует кортеж, лексикографически следующий за a, то записывает его в тот же массив `a` и возвращает `true`; иначе оставляет массив `a` без изменений и возвращает `false`.

`void output_all_combins(int n, int m)`

Выводит в `stdout` все элементы множества $\operatorname{Subsets}(n,m)$ в лексикографическом порядке, используя следующие функции: `fill_array_ident`, `output_array` и `next_combin`.

Теперь займёмся перечислением возрастающих кортежей с помощью рекурсии.

`void output_combins_recur(int *a, int n, int m, int i, int v)`

Выводит в `stdout` в лексикографическом порядке все такие кортежи из

Subsets(n, m), у которых элементы с индексами $< i$ совпадают с заданными элементами массива `a`, а элементы с индексами $\geqslant i$ больше либо равны v. Предполагается, что $0 \leqslant i \leqslant n$ и элементы массива `a` с индексами $< i$ образуют возрастающий кортеж.

`void output_all_combins_recur(int n, int m)`

Выводит в `stdout` в лексикографическом порядке все элементы множества Subsets(n, m). Использует функцию `output_combins_recur`.

9 Перечисление разбиений

Определение 9.1. Множество P называют *разбиением* множества X, если элементы P являются непустыми подмножествами X и попарно не пересекаются, причём объединение всех элементов P равно X. Элементы множества P будем называть *долями* разбиения P. Множество всех разбиений множества $[0, n)_{\mathbb{Z}}$ обозначим через $\mathrm{Partitions}(n)$.

Например, множество $P = \{\{3, 1, 4\}, \{0, 2\}, \{5\}\}$ является разбиением множества $[0, 6)_{\mathbb{Z}}$, т. е. принадлежит множеству $\mathrm{Partitions}(6)$. Разбиение P состоит из трёх долей: $\{3, 1, 4\}$, $\{0, 2\}$, $\{5\}$.

Будем хранить разбиения в виде раскрасок.

Определение 9.2. *Раскраской* множества $[0, n)_{\mathbb{Z}}$ будем называть любой кортеж из $\mathbb{Z}^n$. Если a — раскраска множества $[0, n)_{\mathbb{Z}}$ и $i \in [0, n)_{\mathbb{Z}}$, то значение a_i будем называть *цветом* элемента i.

Определение 9.3. Каждой раскраске a множества $[0, n)_{\mathbb{Z}}$ сопоставим разбиение P_a, определённое следующим правилом: элементы i и j принадлежат одной доле P_a тогда и только тогда, когда $a_i = a_j$.

Например, раскраске $a = (1, 3, 1, 3, 3, 0)$ множества $[0, 6)_{\mathbb{Z}}$ соответствует разбиение $P_a = \{\{0, 2\}, \{1, 3, 4\}, \{5\}\}$, которое приводилось выше в виде примера.

Для каждого разбиения P можно построить такую раскраску a, что $P_a = P$: достаточно как-то пронумеровать доли разбиения и каждому элементу i из $[0, n)_{\mathbb{Z}}$ сопоставить номер («цвет») доли, которой он принадлежит. Но при этом разным раскраскам может соответствовать одно и то же разбиение.

Пример: $n = 6$, $a = (1, 3, 1, 3, 3, 0)$, $b = (2, 0, 2, 0, 0, 3)$, $P_a = P_b = \{\{0, 2\}, \{1, 3, 4\}, \{5\}\}$.

Определение 9.4. Раскраски a и b из $\mathbb{Z}^n$ будем называть *эквивалентными*, если $P_a = P_b$.

Легко сообразить, что эквивалентные раскраски получаются одна из другой переобозначением цветов. Формально: условие $P_a = P_b$ равносильно существованию такой биекции $f\colon \mathbb{Z} \to \mathbb{Z}$, что $a_i = f(b_i)$ при любом $i \in [0, n)_{\mathbb{Z}}$.

Например, раскраски $a = (3, 2, 0, 3, 0)$ и $b = (1, 6, 2, 1, 2)$ эквивалентны.

Очевидно, любая раскраска множества $[0, n)_{\mathbb{Z}}$ содержит не более n различных цветов. Поэтому далее будем считать, что для раскраски множества $[0, n)_{\mathbb{Z}}$ используются только цвета из множества $[0, n)_{\mathbb{Z}}$.

Определение 9.5. Набор $a \in \mathbb{Z}^n$ будем называть *правильной раскраской* множества $[0, n)_{\mathbb{Z}}$, если

$$\forall i \in [0, n)_{\mathbb{Z}} \qquad 0 \leqslant a_i \leqslant \max_{0 \leqslant j < i} a_j + 1.$$

При $i = 0$ в правой части возникает максимум пустого множества, который в данной ситуации нужно считать равным -1.

Упражнение 9.1. Пусть $P = \{\{5, 7, 1, 3\}, \{0, 4\}, \{6, 2\}\}$. Найти такую правильную раскраску a множества $[0, 8)_{\mathbb{Z}}$, что $P_a = P$.

Упражнение 9.2. Пусть $P \in \text{Partitions}(n)$. Доказать, что существует единственная правильная раскраска a множества $[0, n)_{\mathbb{Z}}$, для которой $P_a = P$.

Упражнение 9.3. Доказать, что правильная раскраска является лексикографически наименьшей среди всех эквивалентных ей неотрицательных раскрасок. Формально: пусть a — правильная раскраска множества $[0, n)_{\mathbb{Z}}$, $b \in [0, +\infty)_{\mathbb{Z}}^n$, $P_a = P_b$. Доказать, что $a \prec b$ или $a = b$.

`bool is_correct_coloring(const int *a, int n)`

Выясняет, является ли кортеж a правильной раскраской. Предполагается, что $a \in [0, n)_{\mathbb{Z}}^n$.

`void output_partition(const int *a, int n)`

Выводит в `stdout` разбиение P_a, соответствующее заданной правильной раскраске a множества $[0, n)_{\mathbb{Z}}$.
Пример: для $a = (0, 1, 0, 2, 1, 1)$ выводит `{{0,2},{1,4,5},{3}}`.

`void output_all_partitions_slow(int n)`

Выводит в `stdout` все правильные раскраски множества $[0, n)_{\mathbb{Z}}$ в лексикографическом порядке, используя следующие функции:
`is_correct_coloring`, `next_tuple`, `output_array`.
Вместо `output_array` можно использовать `output_partition`.

Конечно, способ перечисления правильных раскрасок, предлагаемый в функции `output_all_partitions_slow`, очень неэффективен.

Упражнение 9.4. С помощью функции `output_all_partitions_slow` найти число правильных раскрасок множества $[0, 4)_{\mathbb{Z}}$. Найти отношение этого числа к числу элементов множества $[0, 4)_{\mathbb{Z}}^4$.

При работе с раскрасками могут быть полезны следующие функции:

`int min_unused_value(const int *a, int n)`
: Возвращает минимальное неотрицательное целое число, которое не присутствует в кортеже a. Известно, что $a \in [0, n)_{\mathbb{Z}}^n$.

`int number_of_values(const int *a, int n)`
: Возвращает число различных значений, присутствующих в кортеже a. Известно, что $a \in [0, n)_{\mathbb{Z}}^n$.

`void transform_to_correct_coloring(int *a, int n)`
: Превращает заданную раскраску a в эквивалентную ей правильную раскраску.

Теперь нужно понять, как из заданной правильной раскраски строить лексикографически следующую.

Упражнение 9.5. Вручную перечислить в лексикографическом порядке все правильные раскраски множества $[0, n)_{\mathbb{Z}}$ для $n = 2$, $n = 3$ и $n = 4$. Проверить себя с помощью функции `output_all_partitions_slow`.

`int index_to_increase_partition(const int *a, int n)`
: Возвращает наибольший из таких индексов i, что для заданного кортежа a из $\text{Partitions}(n)$ можно построить лексикографически больший кортеж из $\text{Partitions}(n)$, не изменяя элементов, индексы которых меньше i. Если таких индексов i не существует, то возвращает -1.
Пример: для $a = (0, 1, 1, 2, 1, 3)$ возвращает 4.

`bool next_partition(int *a, int n)`
: Если для заданной правильной раскраски существует лексикографически следующая правильная раскраска, то записывает её в тот же массив `a` и возвращает `true`; иначе оставляет массив `a` без изменений и возвращает `false`.
Пример: для $a = (0, 1, 1, 2, 1, 3)$ записывает в `a` кортеж $(0, 1, 1, 2, 2, 0)$ и возвращает `true`.

`void output_all_partitions(int n)`
: Выводит в `stdout` все правильные раскраски множества $[0, n)_{\mathbb{Z}}$ в лексикографическом порядке.

Рекурсивное перечисление разбиений:

`void output_partitions_recur(int *a, int n, int i, int v)`

Выводит в `stdout` в лексикографическом порядке все правильные раскраски множества $[0, n)_{\mathbb{Z}}$, у которых элементы с индексами $< i$ совпадают с заданными элементами массива `a`. Предполагается, что элементы массива `a` с индексами $< i$ образуют правильную раскраску множества $[0, i)_{\mathbb{Z}}$, а v равно максимуму из этих элементов. В случае $i = 0$ предполагается, что $v = -1$.

`void output_all_partitions_recur(int n)`

Выводит в `stdout` в лексикографическом порядке все правильные раскраски множества $[0, n)_{\mathbb{Z}}$. Использует для этого рекурсивную функцию `output_partitions_recur`.

Список литературы

[1] **Виленкин Н. Я.** Популярная комбинаторика. — М.: «Наука», 1975. — 210 с.

[2] **Грэхем Р., Кнут Д., Паташник О.** Конкретная математика. Основание информатики: Пер. с англ. — М.: «Мир», 1988. — 703 с., ил. — ISBN 5-03-001793-3.

[3] **Ерусалимский Я. М.** Дискретная математика: теория, задачи, приложения. — М.: «Вузовская книга», 1998. — 280 с.

[4] **Липский В.** Комбинаторика для программистов: Перевод с польского. — М.: «Мир», 1988. — 200 с.

[5] **Kernighan B., Ritchie D.** The C programming language. Second edition. — AT&T Bell Laboratories. Murray Hill, New Jersey, 1988. — 272 p.