Генерация сочетаний

13 июня 2016 года, понедельник Java

Эту статью я написал несколько лет назад для другого сайта, но она так и не была опубликована. Тогда 7-я версия Java только-только появилась на свет, а 6-я была всё ещё актуальна. Статья адресована, тем, кто уже имеет некоторый опыт программирования на языке Java и не прочь разобраться в алгоритме генерации сочетаний. Я решил стряхнуть со статьи пыль и опубликовать её: пусть будет!

Здравствуйте, уважаемый читатель! В этой статье мы рассмотрим программу, генерирующую все сочетания из *n* элементов по *m*.

Необходимость генерировать сочетания появилась у меня после того, как ко мне обратился знакомый студент и попросил помочь ему решить следующую задачу по комбинаторике.

За круглым столом сидят 12 представителей различных государств, причём каждый из них враждует с соседями справа и слева. Из этих представителей нужно сформировать комиссию, состоящую из 5 человек, причём никакие два члена комиссии не должны враждовать между собой. Каким количеством способов можно сформировать такую комиссию?

Задачу эту я решил так. Пронумеруем всех представителей числами от 1 до 12 в порядке их размещения за столом (например, по часовой стрелке). Каждую комиссию будем описывать номерами представителей, которые в неё входят, перечисленными через запятую и помещёнными в фигурные скобки.

Тогда одна из комиссий, в которых нет врагов, будет выглядеть так: {1, 3, 5, 7, 9}. На основе этой комиссии можно сформировать ещё одну, если каждого члена заменить его соседом слева: {2, 4, 6, 8, 10}. И ещё такую операцию можно проделать 10 раз. Таким образом, получаем 12 вариантов.

Аналогично: формируем комиссию {1, 4, 7, 9, 11} и получаем на её основе ещё 11 комиссий. Всего 12 вариантов.

Наконец, формируем комиссию {1, 3, 6, 8, 10} и получаем на её основе ещё 11 комиссий. Общее количество комиссий — 12.

Итого: 12 + 12 + 12 = 36 вариантов. Это и есть окончательный ответ.

Я был уверен в правильности ответа, но всё же захотел проверить себя с помощью метода перебора. Для его реализации необходимо перебрать все возможные варианты комиссий из 5 человек и выбрать те из них, которые удовлетворяют условиям задачи. Поскольку вариантов много (а именно, 792, как будет показано далее), я решил прибегнуть к помощи компьютера.

И вот тут как раз и возникла необходимость в написании программы, генерирующей *сочетания*. Остановимся на этом понятии более подробно. Предположим, у нас имеются *n* элементов произвольной природы, различимых между собой. Сформируем из этих элементов всевозможные наборы по *m* элементов в каждом наборе, где *m* не превышает *n*, такие, что все элементы в каждом наборе различны и любые два набора различаются составом входящих в них элементов. Эти наборы и называются сочетаниями из *n* элементов по *m*.

Родственным понятием по отношению к сочетанию является *размещение*. Из *n* элементов будем теперь формировать всевозможные упорядоченные наборы по *m* элементов в каждом наборе, где *m* не превышает *n*, такие, что все элементы в каждом

наборе различны и любые два набора различаются составом или порядком входящих в них элементов. Эти наборы называются размещениями из n элементов по m.

Несложно подсчитать число всех размещений из n элементов по m, обозначаемое как Anm. Один из элементов, входящих в размещение, можно выбрать n возможными способами, второй — (n - 1) возможными способами и т. д. Наконец, последний, m-й элемент, можно выбрать (n-m) возможными способами. Перемножив эти количества способов, получаем искомое число всех размещений: Anm=n'n-1'...'n-m+1. С использованием факториалов число размещений записывается так: Anm=n!n-m!.

Давайте теперь вычислим Cnm — число сочетаний из n элементов по m. Разобьём все размещения из n элементов по m на группы таким образом, чтобы размещения, входящие в одну группу, имели бы одинаковый состав и различались только порядком элементов, а размещения, входящие в разные группы, различались составом элементов. Очевидно, что каждая такая группа, содержащая наборы одинакового состава, соответствует сочетанию из n элементов по m и наоборот: каждому такому сочетанию соответствует группа. Это означает, что число сочетаний равно количеству групп.

Для вычисления последнего, очевидно, нужно число размещений из n элементов по m, т. е. Anm, разделить на количество элементов в группе, равное числу размещений из m элементов по m, т. е. Amm. Имеем:

Cnm=AnmAmm=n'n-1'...'n-m+1m'm-1·...·1=n!m!'n-m!.

Очевидно, теперь, что комиссии — это и есть сочетания из 12 представителей по 5. Можно теперь подсчитать и число всех возможных комиссий. Оно равно 12!5!7!=792.

Таким образом, задача перебора всех комиссий, состоящих из 5 представителей, эквивалентна задаче перебора всех сочетаний из 12 элементов по 5. Её решением (в общем виде) мы сейчас и займёмся.

Пронумеруем n элементов номерами от 1 до n. Сочетания будем записывать в виде номеров элементов, входящих в эти сочетания, разделённых запятыми и помещённых в фигурные скобки.

Заметим, что порядок элементов в записи сочетания не важен. Например, {1, 2, 3, 4, 5} и {5, 4, 3, 2, 1} — это одно и то же сочетание. А раз порядок не важен, то договоримся записывать элементы в порядке возрастания номеров.

Введём способ упорядочивания сочетаний. Условимся считать, что сочетание $\{a_1, a_2, ..., a_m\}$ предшествует сочетанию $\{b_1, b_2, ..., b_m\}$, если $a_1 < b_1$ или $a_k = b_k$, где k = 1, 2, ..., l-1 и, при этом, $a_l < b_l$ (здесь предполагается, что l больше 1, но не превышает m).

То же самое можно сформулировать иначе: сочетание $\{a_1, a_2, ..., a_m\}$ предшествует сочетанию $\{b_1, b_2, ..., b_m\}$, если $a_1n^{m-1} + a_2n^{m-2} + ... + a_m < b_1n^{m-1} + b_2n^{m-2} + ... + b_m$.

Теперь, в соответствии с описанным способом упорядочивания, все сочетания можно единственным образом выстроить в одну цепочку, начинающуюся с сочетания {1, 2, ..., m} и заканчивающуюся сочетанием {n, n - 1, ..., n - m + 1}. Именно в таком порядке мы и будем генерировать сочетания.

Составим алгоритм, который по заданному сочетанию создаёт сочетание, непосредственно следующее за ним (разумеется, за исключением случая, когда алгоритму передаётся сочетание, “замыкающее” цепочку; в этом случае алгоритм ничего не возвращает).

Будем считать, что алгоритм принимает на входе сочетание $\{a_1, a_2, ..., a_m\}$.

1. Полагаем: $t = m$.
2. Если $a_t < n - t + 1$, то возвращаем сочетание $\{a_1, ..., a_{t-1}, a_t + 1, a_{t+1}, ..., a_m\}$ и завершаем работу.
3. Если t = 1, то завершаем работу, не возвращая никакого сочетания.
4. Уменьшаем t на единицу и переходим к пункту 2.

Итак, способ генерации сочетаний уже понятен. Передаём алгоритму первое сочетание, получаем второе, передаём второе — получаем третье и так до тех пор, пока не получим последнее. Можно начинать писать программу.

Построение изображения треугольника Серпиньского
05.04.2018

Программа состоит из единственного класса `Combinations`. Значения *m* и *n* хранятся в статических целочисленных константах `M` и `N` соответственно. Сочетание в программе представлено целочисленным массивом из `M` элементов. В ячейке массива с индексом *t* хранится номер элемента сочетания с индексом *t* + 1.

Описанный выше алгоритм реализован в статическом методе `generateCombinations()`. Метод принимает массив, хранящий сочетание, генерирует по переданному сочетанию новое, помещает его в тот же самый массив и возвращает этот массив.

Если методу передано значение `null`, то он возвращает массив с первым сочетанием. Если методу передан массив с последним сочетанием, то метод возвращает `null`.

Управление методом `generateCombinations()` осуществляется из метода `main()`. Сначала `generateCombinations()` вызывается со значением `null`, затем каждый массив, возвращаемый этим методом, передаётся методу снова, до тех пор, пока метод не вернёт `null`. Каждый сгенерированный массив преобразуется в строку статическим методом `toString()` класса `java.util.Arrays` и выводится на консоль.

Ниже приведён код класса `Combinations`.

```
import static java.lang.System.out;
import java.util.Arrays;
class Combinations
{
    private static final int M = 3;
    private static final int N = 5;
    private static int[] generateCombinations(int[] arr)
    {
        if (arr == null)
        {
            arr = new int[M];
            for (int i = 0; i < M; i++)
                arr[i] = i + 1;
            return arr;
        }
        for (int i = M - 1; i >= 0; i--)
            if (arr[i] < N - M + i + 1)
            {
                arr[i]++;
                for (int j = i; j < M - 1; j++)
                    arr[j + 1] = arr[j] + 1;
                return arr;
            }
        return null;
    }
    public static void main(String args[])
    {
        int[] arr = null;
        while ((arr = generateCombinations(arr)) != null)
                out.println(Arrays.toString(arr));
    }
}
```

Наша программа генерирует все сочетания из 5 элементов по 3. Запуск класса `Combinations` на выполнение приведёт к печати следующих результатов:

```
[1, 2, 3]
[1, 2, 4]
[1, 2, 5]
[1, 3, 4]
[1, 3, 5]
[1, 4, 5]
[2, 3, 4]
[2, 3, 5]
[2, 4, 5]
[3, 4, 5]
```

Как видим, программа работает корректно.

Итак, сочетания генерировать мы научились, вернёмся теперь к нашей задаче. Присвоим константам `M` и `N` значения 5 и 12 соответственно, а также изменим тело цикла `while` в методе `main()`. На этот раз будем выводить на печать только те сочетания, которым соответствуют комиссии, не содержащие ни одной пары соседей.

Этому критерию удовлетворяют только те сочетания, которые не содержат элементов с номерами, различающимися на единицу, а также не включают в себя элементов с номерами 1 и 12 одновременно.

Вот исходный код новой версии нашей программы:

```
import static java.lang.System.out;
import java.util.Arrays;
class Combinations2
```

```
{
    private static final int M = 5;
    private static final int N = 12;
    private static int[] generateCombinations(int[] arr)
    {
        if (arr == null)
        {
            arr = new int[M];
            for (int i = 0; i < M; i++)
                arr[i] = i + 1;
            return arr;
        }
        for (int i = M - 1; i >= 0; i--)
            if (arr[i] < N - M + i + 1)
            {
                arr[i]++;
                for (int j = i; j < M - 1; j++)
                    arr[j + 1] = arr[j] + 1;
                return arr;
            }
        return null;
    }
    public static void main(String args[])
    {
        int[] arr = null;
        while ((arr = generateCombinations(arr)) != null)
        {
             int i = 1;
             while ((i < M) && (arr[i] - arr[i - 1] != 1))
                 i++;
             if ((i == M) && (arr[0] != 1 || arr[M - 1] != N))
                 out.println(Arrays.toString(arr));
         }
    }
}
```

На этот раз запуск программы приведёт к следующему выводу на консоль:

```
[1, 3, 5, 7, 9]
[1, 3, 5, 7, 10]
[1, 3, 5, 7, 11]
[1, 3, 5, 8, 10]
[1, 3, 5, 8, 11]
[1, 3, 5, 9, 11]
[1, 3, 6, 8, 10]
[1, 3, 6, 8, 11]
[1, 3, 6, 9, 11]
[1, 3, 7, 9, 11]
[1, 4, 6, 8, 10]
[1, 4, 6, 8, 11]
[1, 4, 6, 9, 11]
[1, 4, 7, 9, 11]
[1, 5, 7, 9, 11]
[2, 4, 6, 8, 10]
[2, 4, 6, 8, 11]
[2, 4, 6, 8, 12]
[2, 4, 6, 9, 11]
[2, 4, 6, 9, 12]
[2, 4, 6, 10, 12]
[2, 4, 7, 9, 11]
[2, 4, 7, 9, 12]
[2, 4, 7, 10, 12]
[2, 4, 8, 10, 12]
[2, 5, 7, 9, 11]
[2, 5, 7, 9, 12]
[2, 5, 7, 10, 12]
[2, 5, 8, 10, 12]
[2, 6, 8, 10, 12]
[3, 5, 7, 9, 11]
[3, 5, 7, 9, 12]
[3, 5, 7, 10, 12]
[3, 5, 8, 10, 12]
[3, 6, 8, 10, 12]
[4, 6, 8, 10, 12]
```

Как мы видим, число искомых сочетаний действительно равно 36.

В следующей части статьи мы перепишем наши программы в объектно-ориентированном стиле с использованием интерфейсов `Iterator` и `Iterable`.

На этом всё. Спасибо за внимание!

Комментарий, сделанный в июне 2016 года. *Планируемое в предпоследнем абзаце продолжение статьи так и не было написано. Кто знает — может, я, всё-таки, когда-нибудь и возьмусь за его написание. Почему бы и нет?*

теги: java, алгоритмы, задача, комбинаторика, сочетания, старые статьи