Кручинин Владимир Викторович

ГЕНЕРАЦИЯ СОЧЕТАНИЙ, РАЗЛОЖЕНИЙ И СЧАСТЛИВЫХ БИЛЕТОВ

ВВЕДЕНИЕ

Факультативные занятия по информатике с талантливыми школьниками и студентами требуют творческого подхода к их организации. Важными элементами здесь являются сложные и нестандартные задачи. Огромное число таких задач можно получить, рассматривая разделы комбинаторики.

Одним из направлений перечислительной комбинаторики [1; 2] является разработка и исследование алгоритмов генерации комбинаторных объектов. Оказывается, что комбинаторному объекту можно поставить в соответствие номер и по данному номеру построить соответствующий объект. Ниже последовательно рассматривается группа связанных алгоритмов, предназначенных для генерации сочетаний, разложений и счастливых билетов.

1. АЛГОРИТМ ГЕНЕРАЦИИ СОЧЕТАНИЙ

Сочетанием множества элементов из $E = \{a_1, a_2, ..., a_n\}$ по m называется подмножество из m элементов, принадлежащих E и отличающихся друг от друга составом, но не порядком элементов. Число сочетаний вычисляется по формуле

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

Сочетание можно представить двоичным числом, где n – количество разрядов в числе, m – количество единиц в n-разрядном двоичном числе. Тогда наличие единицы в i-разряде будет означать, что элемент a_i войдет в сочетание.

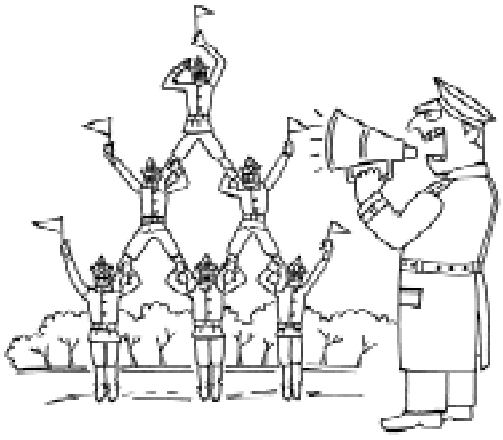
Существует множество различных алгоритмов генерации сочетаний [3]. Ниже будет описан алгоритм генерации, основанный на представлении сочетания в виде дерева. Рассмотрим построение дерева для сочетания C_n^m, для этого запишем следующее тождество:

$$C_n^m = C_{n-1}^m + C_{n-1}^{m-1}.$$

Применяя это тождество для заданного сочетания C_n^m, можно построить следующее двоичное дерево (рисунок 1).

Дерево строится до тех пор, пока каждый лист этого дерева не станет элементом тождества C_k^0 или C_k^k, для которых $C_k^0 = C_k^k = 1$.

Очевидно, что число листьев такого дерева равно C_n^m. Дерево является двоичным, так как каждый узел, кроме листьев, дерева имеет ровно два сына. Если выход-



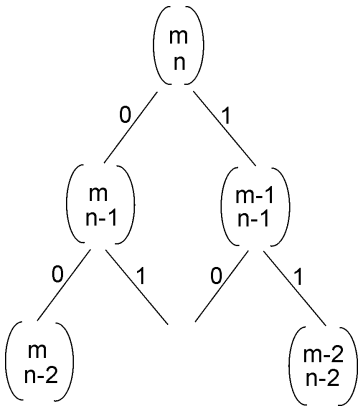


Рисунок 1. Двоичное дерево декомпозиции сочетания.

ные дуги узла нагрузить числами, левая – 0, правая – 1, то путь от корня к листу будет идентифицировать некоторое двоичное число, это двоичное число будет соответствовать некоторому сочетанию. Пример такого дерева записан для представления сочетания C_4^2 на рисунке 2. Количество листьев равно 6, что соответствует числу сочетаний. Заметим, что количество путей из корня этого дерева до листьев также равно числу сочетаний. Тогда, производя левосторонний обход дерева (см. рисунок 2) и записывая соответствующие цифры для каждого пути, получим первый столбец таблицы 1.

Для первой строчки не хватает двух единиц, для второй – одной единицы, для третьей – нуля, для четвертой – единицы для пятой – нуля, для шестой – двух нулей.

Можно предложить следующее правило дописывания единиц или нулей: если лист содержит m равное n, то справа дописывается m единиц; если лист содержит m равное нулю, то справа дописывается n нулей.

Используя эти простые правила, можно перечислить все сочетания для заданных значений m и n. Построим алгоритм генерации сочетания по трем параметрам, где m, n – параметры сочетания, а num – номер сочетания. Очевидно, что для num должно выполняться следующее соотношение:

$$1 \le num \le C_n^m .$$

Решающее правило будет следующим:

1. Если $num \le C_{n-1}^m$, то записываем ноль и переходим на рассмотрение узла C_{n-1}^m.

2. Если $num > C_{n-1}^m$, то записываем единицу, $num = num - C_{n-1}^m$ и переходим на рассмотрение узла C_{n-1}^{m-1}.

3. Если n равно m дописываем m нулей.

4. Если m равно 0 дописываем n единиц.

Тогда алгоритм будет следующий:

```
algorithm GCombination (num, m, n)
  begin
    if m=0 then {записать n нулей}
      for (i = 1, n)  v ← 0
      else
      if m = n then {записать m единиц}
        for(i = 1, m)  v ← 1
  else
  if  k ≤ C(n-1)^m then {движение по левой ветви}
    begin
      v ← 0
      GCombination (num, m, n – 1)
    end
  else
    begin {движение по правой ветви}
      v ← 1
      GCombination (num, m – 1, n – 1)
    end
end
```

Здесь num – номер сочетания, m, n – параметры сочетания, v – вектор, содержащий двоичное представление сочетания. Операция $v \leftarrow b$ означает дописывание справа значение бита b в двоичном числе v. Как видно, алгоритм рекурсивный, глубина рекурсии зависит от глубины дерева соче-

таний и равна параметру n, следовательно, сложность данного алгоритма равна $O(n)$.

2. АЛГОРИТМ ГЕНЕРАЦИИ РАЗЛОЖЕНИЙ

Как известно [4], разложением называется число N, представленное суммой k чисел. $a_1 + a_2 + ... + a_k = N$. Общее число разложений вычисляется по формуле:

$$C_{N+k-1}^{k-1}.$$

Очевидно, что для генерации разложений можно воспользоваться алгоритмом генерации сочетаний *GCombination (num*, $k-1$, $N+k-1$). Однако интерпретация вектора v будет иной. Значение элемента вектора, равное единице, будет означать знак суммы, количество нулей между единицами это собственно число. Тогда 00100001000 => 2 + 4 + 3 или 00000110000 => 5 + 0 + 4.

Усложним задачу, внеся ограничение на число $a_i \le l$. Попробуем разработать алгоритм генерации разложений с таким ограничением. Параметр l задает ограничение на количество нулей, следующих подряд в сочетании в векторе v. Можно воспользоваться деревом представления сочетания и из него построить дерево для представления разложений с ограничением $a_i \le l$. Это можно сделать, отсекая ветви, у которых количество нулей, следующих подряд больше или равно l. Поскольку число разложений с ограничением будет меньше C_{N+k-1}^{k-1}, то необходимо задать алгоритм подсчета. Этот алгоритм рекурсивный и основан на неявном представлении двоичного дерева для разложений с заданным ограничением

Таблица 1.

00	(11)
010	(1)
011	(0)
100	(1)
101	(0)
11	(00)

$a_i \le l$. Параметры алгоритма: j – количество подряд нулей при прохождении в двоичном дереве; m – количество плюсов и n разрядность вектора v ($m = k-1$, $n = N + k - 1$).

```
algorithm CDecomposition (j, m, n)
  begin
    if m = 0 then
      if n ≤ l then return 1 else return 0
    if m=n then return 1
      if j > 0 then return
              CDecomposition (j – 1, n – 1, m) +
              CDecomposition (j, n – 1, m – 1)
  else return CDecomposition (l, n – 1, m – 1)
end
```

Этот алгоритм работает следующим образом: если m равно нулю, то мы добрались до листа двоичного дерева и далее проверяем значение n (см. правило дописывания нулей). Если n меньше или равно l, считаем этот вариант. Если m равно n, то используем правило дописывания единиц и также считаем вариант. Если $j > 0$, то счи-

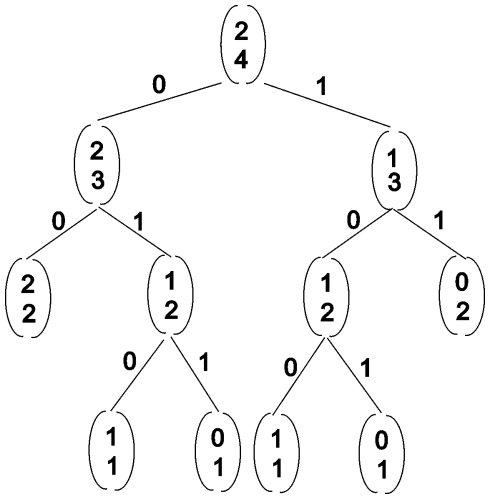


Рисунок 2. Пример двоичного дерева для сочетания C_4^2.

таем варианты и слева и справа, причем, если движемся влево, то j уменьшаем на единицу. Если вправо, то это означает, что мы дописываем в вектор единицу и переходим на рассмотрение следующего числа a_i, таким образом, параметр j должен принять значение l. Если j становится равным нулю, то запрещаем движение по левой ветви дерева.

Рассмотрим теперь алгоритм генерации разложения, который использует алгоритм подсчета вариантов *CDecomposition*, параметр *num* задает номер разложения, остальные параметры такие же, как в вышеописанном алгоритме подсчета *CDecomposition*, l – задает ограничение.

```
algorithm GenerateDecomposition (num, j, n, m)
begin
  if m = 0 then
    begin
      if n > l then return
      for (i = 1, n)  v ← 0
      return
    end
  if m = n then
    begin
      for (i = 1, n)  v ←1
      return
    end
    b = CDecomposition (j – 1, n – 1, m);
    if  num ≤ b  and k > 0 then
      begin {переход по левой ветви}
       v ← 0
      GenerateDecomposition (num, j – 1, n –1, m)
      end
    else
      begin
      v ←1
       if k = 0 then {ветвь одна}
       GenerateDecomposition (num, j, n –1, m – 1)
       else {переход по правой ветви}
       GenerateDecomposition (num – b, l, n – 1, m – 1)
       return
      end
end
```

Таблица 2.

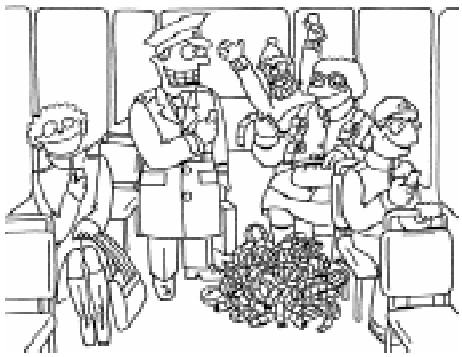
num	Двоичный вектор *v*	Разложение
0	00010001	3 + 3 + 0
1	00010010	3 + 2 + 1
2	00010100	3 + 1 + 2
3	00011000	3 + 0 + 3
4	00100010	2 + 3 + 1
5	00100100	2 + 2 + 2
6	00101000	2 + 1 + 3
7	01001000	1 + 2 + 3
8	01000100	1 + 3 + 2
9	10001000	0 + 3 + 3

Работа алгоритма генерации *GenerateDecomposition* (*num, j, m, n*) для $j = l = 3$, $m = 2$, $n = 8$ приведена в таблице 2.

Используя описанное дерево разложений и алгоритм генерации *GenerateDecomposition,* можно построить следующие алгоритмы:

1. Алгоритм генерации разложений с ограничением на $a_i > l$.
2. Алгоритм генерации разложений с ограничением $l \le a_i \le L$.
3. Алгоритм генерации разбиений.
4. Алгоритм генерации разложения только для четных (нечетных) чисел.
5. Алгоритм генерации разбиений, для которых имеется некоторое подмножество совпадений $a_i = a_k$, $i \ne k$.

3. АЛГОРИТМ ГЕНЕРАЦИИ СЧАСТЛИВЫХ БИЛЕТОВ



Задача подсчета счастливых билетов рассматривается в различных публикациях по занимательной математике [5–8]. Ниже предлагается алгоритм генерации счастливых билетов, основанный на использовании алгоритма *GenerateDecomposition* (*num*, *j*, *n*, *m*).

Вся задача генерации счастливых билетов разбивается на 28 классов, по числу

различных сумм из трех цифр (от 0 до 27). Число счастливых билетов в классе будет определяться произведением числа правого и левого разложений с соответствующими параметрами (для нашего случая $j = l = 9$, $n = i + 2$, $m = 2$, где i – задает номер класса). Тогда запишем алгоритм подсчета числа счастливых билетов.

```
algorithm CountLuckyTicket
  begin
    count=0
  for (i = 0, 27)
    begin
      b = CDecomposition (l, i+2, 2)
      {подсчет числа разложений для i-го класса}
      count=count + b^2
    end
  return count
end
```

В таблице 3 представлен результат подсчета счастливых билетов с помощью алгоритма *CountLuckyTicket.*

Тогда алгоритм генерации счастливых билетов будет следующий

```
algorithm GenerateLuckyTicket(num)
{num – номер счастливого билета по порядку}
begin
  for (i = 0, 27) {определяем номер класса}
  begin
    b=CDecomposition(9, i+2, 2)
    if num – b^2 < 0 break
    num = num –b^2
    end
    num left = num/b {определяем номера}
                     {генерации разложений}
    num right= num mod b
    GenerateDecomposition (num left + 1, 9, i + 2, 2)
                  {генерируем левое разложение}
    left = v
    GenerateDecomposition (num right + 1, 9, i + 2, 2)
                  {генерируем правое разложение}
    right = v
    Translate (left, right)
end
```

Здесь функция *Translate* (*left*, *right*) преобразует битовые вектора *left* и *right* в три цифры и печатает счастливый номер билета. В таблице 4 отражены фрагменты работы алгоритма *GenerateLuckyTicket* (*num*). В левом столбце находятся порядковые но-

Таблица 3.

Сумма цифр (класс)	Число разложений	Число билетов в классе
0	1	1
1	3	9
2	6	36
3	10	100
4	15	225
5	21	441
6	28	784
7	36	1296
8	45	2025
9	55	3025
10	63	3969
11	69	4761
12	73	5329
13	75	5625
14	75	5625
15	73	5329
16	69	4761
17	63	3969
18	55	3025
19	45	2025
20	36	1296
21	28	784
22	21	441
23	15	225
24	10	100
25	6	36
26	3	9
27	1	1
	Итого	55252

Таблица 4.

Номер по порядку	Счастливый билет
0	0 + 0 + 0 = 0 + 0 + 0
1	1 + 0 + 0 = 1 + 0 + 0
2	1 + 0 + 0 = 0 + 1 + 0
3	1 + 0 + 0 = 0 + 0 + 1
4	0 + 1 + 0 = 1 + 0 + 0
...	...
25000	4 + 5 + 4 = 0 + 4 + 9
25001	4 + 4 + 5 = 9 + 4 + 0
25002	4 + 4 + 5 = 9 + 3 + 1
25003	4 + 4 + 5 = 9 + 2 + 2
25004	4 + 4 + 5 = 9 + 1 + 3
25005	4 + 4 + 5 = 9 + 0 + 4
...	...
55246	9 + 8 + 9 = 9 + 8 + 9
55247	9 + 8 + 9 = 8 + 9 + 9
55248	8 + 9 + 9 = 9 + 9 + 8
55249	8 + 9 + 9 = 9 + 8 + 9
55250	8 + 9 + 9 = 8 + 9 + 9
55251	9 + 9 + 9 = 9 + 9 + 9

мера счастливых билетов. Справа сами счастливые билеты.

ЗАКЛЮЧЕНИЕ

Рассмотренные алгоритмы генерации сочетаний и размещений можно использовать в различных предметных областях, где необходимо строить генераторы, в частности, в тестировании устройств, компьютерных программ и т. д.

Описанный алгоритм генерации сочетаний использовался автором при разработке компьютерных программ генерации меню-вопросов для организации компьютерного тестирования [9].

Рассмотренный выше пример алгоритма генерации счастливых билетов показывает подходы для построения генерирующих алгоритмов на основе имеющихся алгоритмов генерации комбинаторных объектов.

На диске к журналу находятся тексты описанных программ на языке С.

Литература

1. *Гульден Я., Джексон Д.* Перечислительная комбинаторика. М.: Наука, 1990. 504 с.

2. *Стенли Р.* Перечислительная комбинаторика. М.: Мир, 1990. 440 с.

3. *Akl S.G.* Adaptive and optimal parallel algorithms for enumerating permutations and combinations, The Computer Journal, 30(1987). P. 433–436.

4. *Ландо С.К.* Комбинаторика. М.: Изд-во независимого москов. ун-та, 1994. 78 с.

5. *Савин А.П., Финк Л.М.* Разговор в трамвае. «Квант», № 7 , 1975. С. 67–70.

6. *Финк Л.М.* Ещё раз о счастливых билетах . «Квант», № 12, 1976. С. 68–70.

7. *Виленкин Н.Я.* Счастливые троллейбусные билеты. «Популярная комбинаторика», М.: Наука, 1975. С. 147–148.

8. *Ландо С.К.* Счастливые билеты. Сборник «Математическое просвещение», № 2, 1998. С. 127–132.

9. *Кручинин В.В.* Генераторы в компьютерных учебных программах. Томск: изд-во Томск. ун-та, 2003. 200 с.

Кручинин Владимир Викторович,
кандидат технических наук,
заместитель директора
Томского межвузовского центра
дистанционного образования
по науке.