УДК 519.16:004.42

М.Ю. Перминова, В.В. Кручинин

Алгоритмы рекурсивной генерации ограниченных разбиений натурального числа

Описаны алгоритмы последовательной генерации, нумерации и генерации по номеру ограниченных разбиений. Алгоритмы построены на основе использования дерева разбиений, полученного с помощью рекуррентной формулы. Выполнен анализ предложенных алгоритмов и проведено их сравнение с известными.

Ключевые слова: алгоритм, генерация, разбиение натурального числа, нумерация, временная сложность.

Алгоритмы комбинаторной генерации разбиений натуральных чисел имеют важное научное и практическое значение [1–4]. Так, при вычислениях формулы Фа Ди Бруно и симметрических полиномов используются алгоритмы генерации разбиений [5, 6]. Известна связь между разбиениями и композициями натурального числа, что также приводит к необходимости генерировать разбиения. При построении многих классов деревьев на основе процедуры полного разбиения также используются алгоритмы генерации разбиений [3]. Уже получены и исследованы алгоритмы комбинаторной генерации всех разбиений и разбиений, имеющих M позиций. Однако алгоритмы комбинаторной генерации ограниченных разбиений авторам не известны. Ниже будут рассмотрены алгоритмы комбинаторной генерации одного класса ограниченных разбиений. Введем основные понятия. Под разбиением натурального числа n понимается всякая конечная невозрастающая последовательность натуральных чисел $\lambda_1,\lambda_2,...,\lambda_k$, для которой

$$\sum_{i=1}^{k}\lambda_i = n.$$

Числа λ_i называются частями разбиения [5]. Рассмотрим разбиения числа n с ограничениями $k \leqslant M$ и $\lambda_i \leqslant N$. Рассмотрим подробнее этот класс разбиений. Дж. Эндрюс в работе [7] получил рекуррентную формулу числа таких разбиений и записал производящую функцию. Рекуррентная формула для получения $p(N,M,n)$ – числа разбиений n не более чем на M частей, каждая из которых не превосходит N, имеет вид

$$p(N,M,n) = p(N,M-1,n) + p(N-1,M,n-M), \tag{1}$$

а производящая функция для ограниченных разбиений

$$G(N,M,x) = \frac{\prod\limits_{n=1}^{N+M}\left(1-x^n\right)}{\prod\limits_{n=1}^{M}\left(1-x^n\right)\prod\limits_{n=1}^{N}\left(1-x^n\right)}.$$

Метод построения алгоритмов. В основе метода построения алгоритмов комбинаторной генерации ограниченных разбиений лежит построение дерева решений (частный случай И/ИЛИ дерева). Эти деревья обладают следующими свойствами [3]:

1) каждому элементу комбинаторного множества соответствует единственный путь от корня к листу;

2) число листьев в дереве равно числу элементов комбинаторного множества;

3) на таком дереве можно задать экономные алгоритмы последовательной генерации элементов комбинаторного множества.

Рассмотрим правила построения дерева решений, основываясь на формуле (1). Это рекуррентное соотношение порождает двоичное дерево. В зависимости от правила порождения разбиения в узлы этого дерева записываются тройки (N,M,n), а в листья – 0 или 1. Правила следующие:

1. Если $n<0$ или $NM<n$, то в лист записывается 0.
2. Если $n=0$ или $NM=n$, то в лист записывается 1.
3. Получить левый узел – $(N,M-1,n)$.
4. Получить правый узел – $(N-1,M,n-M)$.

Пример дерева, построенного по данным правилам для $(3,3,5)$, показан на рис. 1.

ИЛИ-дерево на рис. 1 содержит в себе листы, в которых или ноль, или единица. Получить разбиения можно только по следу с листом, содержащим единицу. Соответственно, варианты с листами, содержащими ноль, отбрасываются и получается ИЛИ-дерево, по всем следам которого можно получить разбиения. Такое дерево будем называть деревом разбиений. Пример дерева разбиений с корнем $(3,3,5)$ приведён на рис. 2.

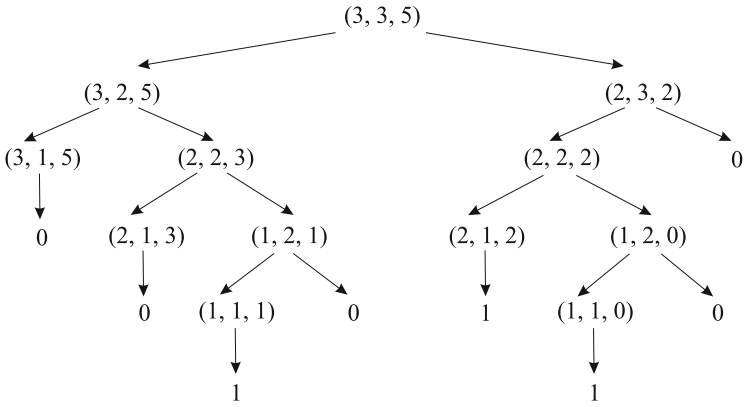
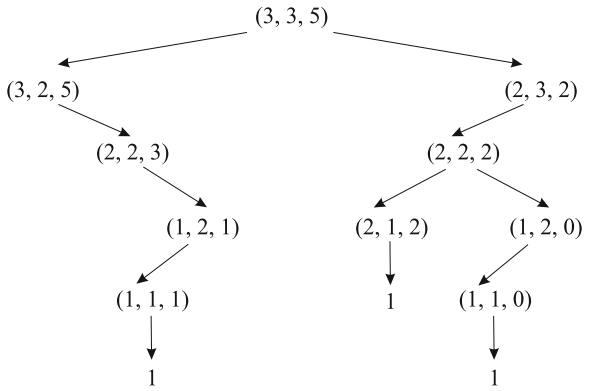


Рис. 1. ИЛИ-дерево для $(3,3,5)$



Используя метод построения алгоритмов комбинаторной генерации, получим алгоритмы последовательной генерации, генерации по номеру и алгоритм нумерации для рассмотренного класса разбиений.

Рис. 2. Дерево разбиений

Алгоритм последовательной генерации разбиений. Алгоритм последовательной генерации основан на использовании стека, в котором хранится путь от корня к листу дерева разбиений. Элемент стека хранит значения параметров N,M,n для текущего узла. Алгоритм First предназначен для нахождения пути от корня к самому левому листу дерева. Затем используется алгоритм Next, который производит преобразование стека по заданным правилам. По окончании работы данного алгоритма в стеке будет находиться самый близкий правый след или стек пуст.

Запишем основные параметры алгоритмов:

1) N,M,n – параметры разбиения;

2) Li – стек, элемент которого содержит параметры разбиения N,M,n с операциями: push – положить, pop – взять, first – копировать вершину стека;

3) V[i] – вектор, содержащий текущие параметры N,M,n.

Алгоритм Next находит следующий путь в поддереве и относительно стека Li модифицирует стек. Правила следующие:

1) лист всегда удаляется из стека;

2) определяется ветвь текущего узла. Если принадлежит левой ветви, то текущий узел заменяется на правый и запускается алгоритм First, если это правая ветвь, то текущий узел удаляется и процесс поиска продолжается;

3) если стек пуст, то процесс завершается.

Ниже приведены рекурсивный алгоритм First, который находит самый левый путь в поддереве и заносит этот путь в стек Li, и алгоритм Next. Для описания данных алгоритмов используется псевдокод, синтаксис которого похож на синтаксис системы Maxima.

Алгоритм First(N,M,n)

```
First(N,M,n):=
begin
 if n<0 then 0 else
 if n=0 then (push([N,M,n],Li),1) – нашли лист
  else
 if N*M<n then 0 else
 if N*M=n then (push([N,M,n],Li),1) – нашли лист
  else
   begin
    push([N,M,n],Li)
    if First(N,M-1,n)=0
    then First(N-1,M,n-M) – если влево нет сыновей, то
                            переходим на правую ветвь
   end
 end
```

Алгоритм Next()

```
Next():=
begin
 if Li=[] then 0 else – стек пуст?
  begin
   Lf:=pop(Li), – вытащить лист
  if Li=[] then 0 else – стек пуст?
   begin
   V:=first(Li), – взять текущий узел (N,M,n)
   if Lf[3]<V[3] then Next() – если правая ветвь
  else
   if First(V[1]-1,V[2],V[3]-V[2])=0
   then Next() – если левая ветвь
   end
 end
end
```

Алгоритм получения разбиения по следу, находящемуся в стеке Li

```
PrintPartitions():=
begin
 for l:=1 thru 20 do parti[l]:=0
 if Li=[] then 0 else
  begin
   V:first(Li)
   if V[3]>0 then for j:=1 thru V[2] do
   parti[j]:=V[3]/V[2],
   R:V[2]
    for i:=2 thru length(Li) do
    begin
     Z:=Li[i],
     if Z[3]>V[3] then
      begin
      for k:=1 thru V[2] do parti[k]:=parti[k]+1
      R:=V[2]
     end
     V:=Z
    end
  LL:=makelist(parti[l],l,1,R)
  print(LL)
 end
end
```

Пример работы алгоритма последовательной генерации

```
Li:[], – заводим стек
First(5,6,7) – получаем самый левый след
for i:=1 thru p(5,6,7) do – цикл по всем разбиениям
                            с параметрами 5, 6, 7
 begin
  PrintPartitions(),
  Next()
 end
```

Результат выдачи

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

Алгоритм генерации разбиения по номеру Unrank. Алгоритм генерации каждому уникальному числу $0 \le num < p(N,M,n)$ ставит однозначное соответствие разбиения π_k множества ограниченных разбиений $\mathrm{Par}(N,M,n)$. Параметры алгоритма следующие:

1) num – номер разбиения;
2) N,M,n – параметры разбиения;
3) parti – массив, содержащий числа разбиения.

Этот алгоритм рекурсивный. Правила работы алгоритма следующие: по значению *num* определяется ветвь и соответственно сын – если $num < p(N,M-1,n)$, то левый, иначе правый. Затем значение *num* корректируется для соответствующего сына узла и происходит рекурсивный переход на рассмотрение сына. Процесс продолжается до достижения листа дерева разбиений. После рекурсивного вызова Unrank производится вычисление частей ограниченного разбиения parti.

Алгоритм нумерации Rank. Алгоритм нумерации разбиения каждому разбиению π_k множества ограниченных разбиений $p(N,M,n)$ взаимно однозначно ставит в соответствие уникальный номер num_k. Этот алгоритм рекурсивный. Запишем параметры алгоритма:

1) N,M,n – параметры разбиения;
2) parti – массив, содержащий числа разбиения;
3) sizee(parti,M) – функция определяет размер текущего разбиения (параметр M).

Правила работы алгоритма следующие: по текущему разбиению, содержащемуся в parti, определяется параметр M, который имеет идентификатор mm. Если идентификатор mm меньше параметра алгоритма M, то переход по левой ветви. Если mm равен M, то переход по правой ветви, при этом корректируются значения частей parti. Текущее значение номера для левой ветви не меняется, а для правой ветви к текущему номеру прибавляется значение $p(N,M-1,n)$.

Алгоритм Rank

```
Rank(N,M,n,parti):=
 begin
  if n<0 then 0 else
  if N*M<n then 0 else
  if N*M=n then 0 else
  begin
   mm:=sizee(parti,M)-1
   if mm=0 then 0 else
   if mm<M then Rank(N,M-1,n,parti)
   else
    begin
     for i:0 thru M-1 do parti[i]:=parti[i]-1,
    Rank(N-1,M,n-M,parti)+p(N,M-1,n)
    end
 end
end
```

Алгоритм Unrank

```
Unrank(num,N,M,n,parti):=
begin
 if n<0 then 0 else
 if n=0 then 1 else
 if N*M<n then 0 else
 if N*M=n then for j:0 thru M-1 do parti[j]:=N else
 if num<p(N,M-1,n)
 then Unrank(num,N,M-1,n,parti)
  else
    begin
     Unrank(num - p(N,M-1,n),N-1,M,n-M,parti)
    for j:=0 thru M-1 do parti[j]:=parti[j]+1
   end
end
```

Анализ алгоритмов. Д. Кнут в своей книге [1] рассмотрел метод генерации всех разбиений числа n на фиксированное количество частей и предложил алгоритм для автоматизации данного метода. В данном случае число n разбивается ровно на M частей. Временная сложность этого алгоритма равна

$$h(n,M)=3S(n,M)+M,$$

где $S(n,M)$ – число разбиений числа n ровно на M частей.

Рассмотрим временную сложность $T(N,M,n)$ для представленного алгоритма последовательной генерации ограниченных разбиений (N,M,n). Этот алгоритм строит ограниченные разбиения на основе левостороннего обхода дерева разбиений. Это позволяет утверждать, что временная сложность алгоритма пропорциональна общему числу узлов в дереве разбиений. Поскольку это дерево строится на основе формулы (1), то данное дерево является двоичным. Тогда формула подсчета для числа узлов будет иметь вид

$$T(N,M,n)=\begin{cases}T(N,M-1,n)+T(N-1,M,n-M)+1 & \text{для узла,}\\ 1 & \text{для листа.}\end{cases}$$

Если бы листья двоичного дерева не содержали нули, то данное дерево было бы полным двоичным и общее число узлов равнялось бы удвоению числа листьев. В нашем случае это не так, поэтому число узлов в дереве больше чем $2p(N,M,n)$.

Известно [3], что для алгоритмов Rank и Unrank временная сложность одинаковая. Поэтому рассмотрим временную сложность генерации всех разбиений для алгоритма Unrank. Алгоритм для каждого разбиения начинает работать с рассмотрения текущего узла (N,M,n). Поэтому при генерации всех разбиений каждый узел дерева разбиений будет просмотрен $p(N,M,n)$ раз. Откуда временная сложность будет иметь следующий вид:

$$O(N,M,n)=\begin{cases} O(N,M-1,n)+O(N-1,M,n-M)+p(N,M,n) & \text{для узла,} \\ 0 & \text{для листа.} \end{cases}$$

Получить производящие функции для функций $T(N,M,n)$ и $O(N,M,n)$ авторам не удалось, поэтому было решено провести вычислительный эксперимент. На рис. 3 представлены графики зависимостей для функций $p(N,M,n)$, $T(N,M,n)$, $O(N,M,n)$ от n при $N=n/2$ и $M=n/2$. Кривая для $p(N,M,n)$ показывает рост числа композиций данного класса. Кривая для функции $O(N,M,n)$ растет быстрее, чем $T(N,M,n)$. Откуда следует, что функция $O(N,M,n)$ существенно более затратная по сравнению с $T(N,M,n)$.

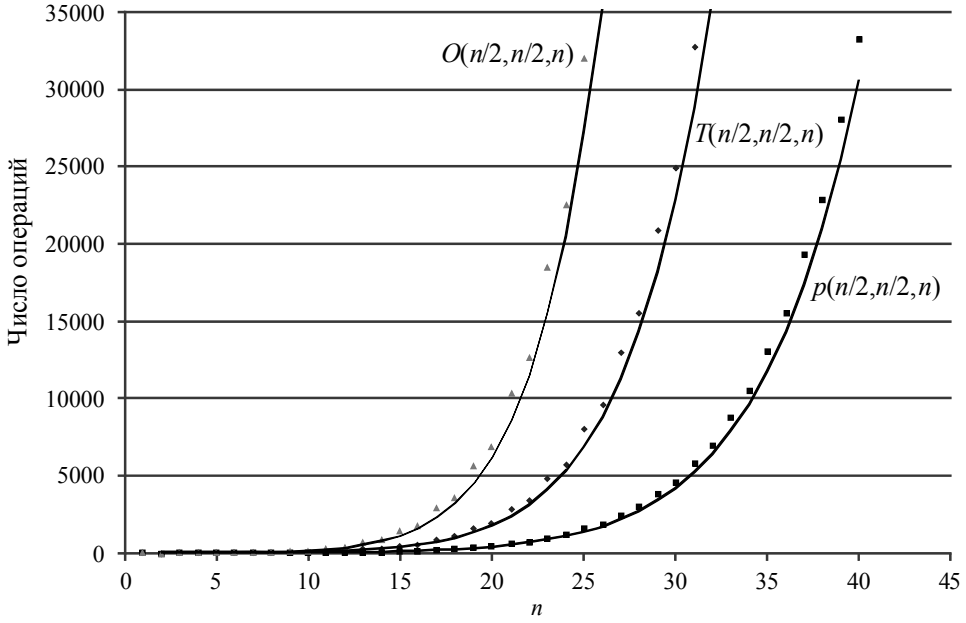


Рис. 3. График зависимостей числа разбиений $p(n)$, числа узлов в дереве разбиений $T(n)$ и временной сложности $O(n)$ генерации всех разбиений алгоритма Unrank от n

Рассмотрим теперь отношения функций $T(N,M,n)/p(N,M,n)$ и $O(N,M,n)/p(N,M,n)$, характеризующие число операций на одно разбиение; на рис. 4 представлены графики отношений при $N=n/2$ и $M=n/2$.

Вид графиков, представленных на рис. 4, позволяет сделать вывод, что:

1) временная сложность алгоритма последовательной генерации имеет логарифмический вид $a\ln n+b$, где a и b – некоторые константы. Сравнивая этот результат с временной сложностью алгоритма Кнута, который имеет фиксированное число операций на одно разбиение, можно сделать вывод о том, что алгоритм Кнута работает быстрее;

2) временная сложность для алгоритмов Unrank и Rank имеет вид, близкий к линейному.

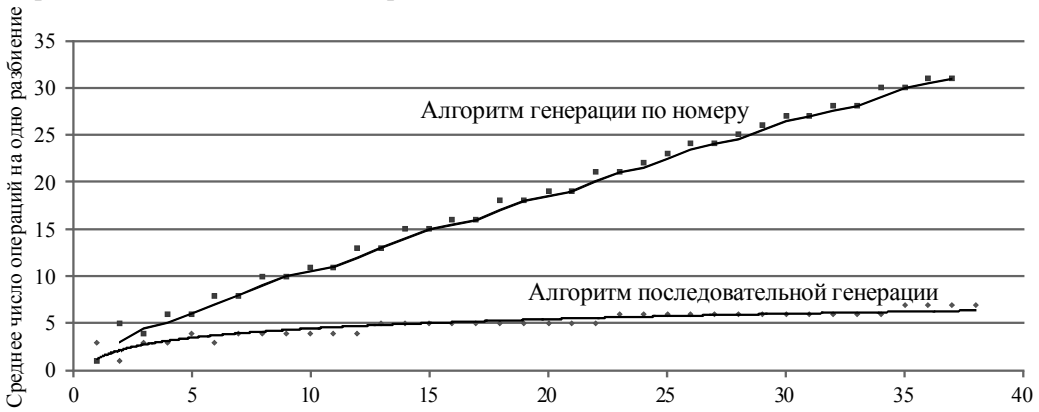


Рис. 4. График зависимостей среднего числа операций на одно разбиение для алгоритмов последовательной генерации и генерации по номеру

Заключение. Представленный алгоритм хуже уже известного алгоритма Кнута. Однако достоинством данного алгоритма последовательной генерации является тот факт, что он использует дерево разбиений. Это означает, что алгоритм First/Next можно использовать совместно с алгоритмами Rank и Unrank. Например, нам необходимо генерировать не все множество разбиений, а некоторое подмножество. С помощью модификации алгоритма Unrank первоначально создаем стек с заданным следом дерева и далее запускаем алгоритм последовательной генерации. В этом случае сложно воспользоваться алгоритмом Кнута, поскольку в нем используется механизм перестановок.

Литература

1. Кнут Д. Искусство программирования. Т. 4, вып. 3: генерация всех сочетаний и разбиений / Д. Кнут : пер. с англ. – М. : ООО «И.Д. Вильямс», 2007. – 208 с.

2. Kreher D.L. Combinatorial algorithms: Generation, Enumeration and Search / D.L. Kreher, D.S. Stinson. – Boca Raton: CRC Press, 1998. – 329 p.

3. Кручинин В.В. Методы построения алгоритмов генерации и нумерации комбинаторных объектов на основе деревьев И/ИЛИ. – Томск: В-Спектр, 2007. – 200 с.

4. Кручинин В.В. Алгоритмы генерации и нумерации композиций и разбиений натурального числа *n* // Доклады ТУСУРа. – 2008. – №2 (17). – С. 113–119.

5. Стенли Р. Перечислительная комбинаторика. Деревья, производящие функции и симметрические функции / Р. Стенли. – М.: Мир, 2009. – 767 с.

6. Кручинин В.В. Комбинаторика композиций и ее приложения / В.В. Кручинин. – Томск: В-Спектр, 2010. – 156 с.

7. Эндрюс Г. Теория разбиений / Г. Эндрюс; перев. с англ. – М.: Наука. Главная редакция физико-математической литературы, 1982. – 256 с.

Перминова Мария Юрьевна
Аспирант каф. промышленной электроники ТУСУРа
Тел.: 8 (383-2) 41-39-43 (вн. 4205)
Эл. почта: pmy@2i.tusur.ru

Кручинин Владимир Викторович
Д-р техн. наук, профессор каф. промышленной электроники ТУСУРа
Тел.: 8 (383-2) 42-30-67 (вн. 4170)
Эл. почта: kru@ie.tusur.ru

Perminova M.Y., Kruchinin V.V.

Recursive generation algorithms of limited partitions of natural number

The paper describes the algorithms of consecutive generation, numbering and generation according to number of limited partitions. Algorithms are constructed on the basis of use of the tree of partitions received by means of a recurrent formula. We analyzed the offered algorithms and compared them with the known ones.
Keywords: algorithm, generation, partition of natural number, numeration, time complexity.