Введение

С сентября 1996 года на базе СШ 27 г. Гомеля, а сентября 1999 года дополнительно и на базе сайта дистанционного обучения DL.GSU.BY (далее DL) ведется работа [1-6] по факультативному изучению информатики и программирования школьниками разных возрастов. Ключевое особенностью этого обучения является раннее начало обучения (фактически с 1-го класса). Как следствие, уже в 5/6-ом классе появляются ученики, которым приходится объяснять такие темы, как рекурсия. Поскольку традиционные подходы рассчитаны на обучение как минимум, старшеклассников, то приходится их модифицировать в сторону более простого и наглядного объяснения, и более медленного продвижения по учебному материалу, явно выделяя и обозначая все этапы этого продвижения. Данная статья предлагает читателям соответствующий материал по теме «Рекурсивная генерация комбинаторных объектов». Он содержит последовательное изложение необходимых сведений и закрепление их решением предложенных задач. Проверка решений осуществляется автоматически на сайте DL.GSU.BY.

В данной статье мы познакомимся с методикой генерации таких комбинаторных объектов как: множество всех подмножеств, сочетания, перестановки, перестановки с повторениями, лексикографически упорядоченные строки из различных символов заданного алфавита, правильные скобочные выражения, двоичный код Грея в N позициях, генерация чисел без ведущих нулей, генерация чисел из определённых цифр. Кроме того в статье приведены такие практически полезные приёмы как переменное количество операторов цикла и рекурсия с запоминанием (мемоизацией).

Разбор и решение приведенных затем в данной главе задач на вышеуказанные темы позволит подготовиться к последующему изучению способов решения более сложных задач на применение рекурсии в дальнейших разделах: деревья, графы, по рекурсивному определению, двумерные массивы, битовая обработка.

Множество всех подмножеств

Простейший пример (ранее уже описанный в первой главе) - задача "Количество способов составить точно сумму M". Она может быть решена с помощью рекурсивной процедуры:

```
procedure Rec(N,M : longint);
  begin
```

```
    if (N>=0) and (M= 0) then inc(ans);
    if (N=0)  and (M<>0) then exit;
    Rec(N-1,M-a[N]);
    Rec(N-1,M)
  end;
```

Вызов этой процедуры в главной программе осуществляется так:
Rec(N,M);

Процедура Rec выполняет следующие действия:
Если сумма набрана, увеличиваем ответ
Если элементов в массиве больше нет, а сумма не набрана, выходим их процедуры.
Вызываем процедуру с уменьшенным количеством элементов и взяв текущий элемент в сумму
Вызываем процедуру с уменьшенным количеством элементов, НЕ взяв текущий элемент в сумму

Вывод сочетаний из N по M

Рекурсивно генерируем все подмножества, сочетания анализируем, когда взяли нужное количество элементов из множества:

```
procedure Rec(i:longint);
  begin
    if kb=M
      then begin
             анализируем текущее сочетание
             exit;
           end;
    inc(kb); b[kb]:=a[i]; Rec(i+1); dec(kb);
    Rec(i+1);
  end;
```

Вызов рекурсивной процедуры Rec осуществляется так:
Rec(0);
Здесь
kb - количество чисел, добавленных в сочетание

b - массив, хранящий числа сочетания

Количество сочетаний из N по M

C(N,M) - количество сочетаний из M цифр на N позициях можно вычислить с помощью рекурсивной функции:

```
function C(N,M:longint) : longint;
  begin
    if ((M=0) and (N>0)) or
       ((M=N) and (M>0))
      then C:=1
      else if (M>N) and (N>=0)
             then C:=0
             else C:=C(N-1,M-1)+C(N-1,M)
  end;
```

Вызов этой функции можно осуществлять, например, так:

```
Writeln(C(N,M));
```

Перестановки

Рекурсивно генерируем все перестановки чисел от 1 до N. При необходимости в конкретных задачах эти числа от 1 до N можно использовать как номера элементов, из которых формируются перестановки.

```
procedure Rec(k:longint);
  var
    i : longint;
  begin
    if k=0
      then begin
             анализируем очередную перестановку
             exit;
           end;
```

```
    for i:=1 to N do
      if x[i]=0
        then begin
               x[i]:=1; inc(kb); b[kb]:=i; Rec(k-1);
               x[i]:=0; dec(kb);
             end;
  end;
```

Вызов процедуры

```
Rec(N);
```

Здесь k означает сколько элементов осталось добавить в перестановку.

Для генерации перестановок из N элементов используется глобальный массив x, хранящий признаки использования элементов для i от 1 до N. X[i]=0 означает, что i-ый элемент еще не включен в перестановку, а x[i]=1 означает, что i-ый элемент уже включен в перестановку. Массив b хранит номера элементов перестановки в порядке включения.

Процедура Rec работает так

Если k= 0

то составлена очередная перестановка

Выполняем ее анализ

Выходим из процедуры

Цикл по всем элементам перестановки

Если есть не добавленный в перестановку элемент

то помечаем его как добавленный, добавляем, вызываем Rec(k-1)

помечаем как не добавленный, удаляем из перестановки

Примечание:

Часто бывает, что анализ перестановки можно выполнять инкрементально (по ходу) , не храня всю перестановку, а передавая в рекурсивную процедуру результат инкрементального анализа. В этом случае рекурсивная процедура может выглядеть так

```
  procedure Rec(k, P :longint);
    var
      i : longint;
    begin
      if k=0
        then begin
               анализируем очередную перестановку
```

```
            exit;
          end;
    for i:=1 to N do
      if x[i]=0
        then begin
               вычисляем значение NewP по P и i
               x[i]:=1; Rec(k-1, NewP); x[i]:=0;
             end;
  end;
```

Вызов процедуры

```
Rec(N,P0);
```

Здесь P - параметр перестановки

P0 - его начальное значение

NewP - новое значение, вычисляемое в процедуре Rec по старому значению и последнему добавленному в перестановку элементу.

Перестановки с повторениями

Например, пусть требуется найти количество способов составить строку S из подстрок a[i], их можно повторять:

```
procedure Rec(S:string);
  var
    i,p : longint;
    x   : string;
  begin
    if S='' then begin inc(ans); exit; end;
    for i:=1 to N do
      if pos(a[i],s)=1
        then begin
               x:=s;
               delete(x,1,length(a[i]));
               Rec(x);
             end;
  end;
```

Вызов процедуры осуществляется так:

```
ans=0;
Rec(S);   { Здесь S = строка, которую необходимо составить}
```

Процедура Rec работает следующим образом:

Если S = пустая строка то инкрементируем ответ и выходим из процедуры

Для всех подстрок a[i]

Если подстрока a[i] есть в строке S с позиции 1

то удаляем эту подстроку из S

вызываем Rec с оставшейся частью строки

Переменное количество операторов цикла

Рассмотрим задачу, в которой требуется проанализировать все комбинации из цифр 0,1,2,3 на N позициях, то есть реализовать такую конструкцию:

```
for y[1]:=0 to 3 do
for y[2]:=0 to 3 do
...
for y[N]:=0 to 3 do
  анализ текущего состояния массива Y из N элементов,
  каждый из которых принимает одно из значений (0,1,2,3),
  значения могут повторяться
```

Это может быть выполнено с помощью следующей рекурсивной процедуры:

```
procedure Rec(N:longint);
  var
    yn : longint;
  begin
    if N=0 then Анализируем массив Y из N элементов; выход
    for yn:=0 to 3 do
      begin y[N]:=yn; Rec(N-1); end
  end;
```

Вызов процедуры Rec

```
Rec(N);   { N- количество "операторов цикла" }
```

Процедура Rec работает следующим образом:

Если N=0 то анализируем построенный массив Y из N элементов, выходим из процедуры
Цикл по всем нужным значениям
В Массив Y на позицию N заносим очередное значение
Вызываем рекурсию для предыдущей позиции (N-1)

Генерация лексикографически упорядоченных строк из различных символов заданного алфавита

Может быть выполнена с помощью следующей рекурсивной процедуры (здесь 19 - заданное количество символов в алфавите):

```
procedure Rec(N:longint; s: string);
  var
    i : longint;
  begin
    if N>19 then exit;
    if Строка s подходит then inc(Ans);
    for i:=N+1 to 19 do
      Rec(i,s+b[i]);
  end;
```

Вызов этой процедуры в главной программе выполняется так ():

```
Ans:=0; b:='bcdfghjklmnpqrstvxz'; {алфавит}
for i:=1 to 19 do
  Rec(i,b[i]);
writeln(Ans);
```

Вывести все правильные скобочные выражения

Пусть требуется найти вывести все правильные скобочные выражения, составленные из N открывающих и N закрывающих круглых скобок. Это может быть сделано с помощью такой рекурсивной процедуры

```
procedure Rec(L,B:longint; s : string);
  begin
    if L=0 then begin writeln(s); exit; end;
    if B<L then Rec(L-1,B+1,s+'(');
    if B>0 then Rec(L-1,B-1,s+')');
  end;
```

Вызов процедуры осуществляется так

```
Rec(2*n,0,'') {вывести все правильные скобочные выражения
        в лексикографическом порядке}
```

Здесь

S - формируемая рекурсивно правильная строка из открытых и закрытых скобок

L - количество скобок, которые осталось поставить

B - разница между количеством открывающих и закрывающих скобок (баланс)

Процедура Rec работает следующим образом:

Если L=0 то увеличиваем ответ, выходим из процедуры

Если баланс меньше чем количество оставшихся позиций,
то добавляем открывающую скобку

Если баланс больше нуля, то добавляем закрывающую скобку

Количество правильных скобочных выражений

Пусть требуется найти количество правильных скобочных выражений, составленных из N открывающих и N закрывающих круглых скобок. Это может быть сделано с помощью такой рекурсивной процедуры

```
procedure Rec(L,B:longint);
  begin
    if L=0 then begin inc(k); exit; end;
    if B<L then Rec(L-1,B+1);
    if B>0 then Rec(L-1,B-1);
  end;
```

Вызов выполняется так:

Rec(2*n,0)

Здесь

L - количество скобок, которые осталось поставить

B - разница между количеством открывающих и закрывающих скобок (баланс)

Процедура Rec работает следующим образом:

Если L=0 то увеличиваем ответ, выходим из процедуры

Если баланс меньше чем количество оставшихся позиций ,

то добавляем открывающую скобку

Если баланс больше нуля, то добавляем закрывающую скобку

Заметим, что в этой задаче "добавление" открывающих и закрывающих скобок осуществляется "виртуально" (реально строка не формируется, только соответствующим образом модифицируются переменные L и B, соответственно, количество оставшихся позиций и баланс)

Двоичный код Грея в N позициях

Это двоичный код, в котором каждое следующее число из N битов отличается от предыдущего ровно в одном разряде. Например, для N=3 код Грея имеет следующий вид:

000

001

011

010

110

111

101

111

Рекурсивная процедура, решающая такую задачу может выглядеть так:

```
procedure Rec(n:longint);
  var
    k,i : longint;
  begin
    if n=1 then begin s[1]:='0'; s[2]:='1'; exit end;
    Rec(n-1);
    k:=p2[n-1];
    for i:=1 to k do s[k+i]:='1'+s[k+1-i];
```

```
      for i:=1 to k do s[i]  :='0'+s[i];
    end;
```

Вызывается она так:

```
  Rec(N);
```

Здесь

S - глобальный массив строк, который будет содержать код Грея

Процедура Rec работает так

(Рекурсивное построение массива строк S кода Грея "по определению"):

Если n=1 то массив содержит строки 0, 1, выход из процедуры

Вызвать рекурсивное построение для n=n-1

А после этого (имея массив строк кода Грея для n-1) построить массив для n следующим образом:

- во вторую половину переписать строки в обратном порядке, добавив к ним символ 1 слева.

- в первой половине массива добавить символ '0' слева.

Генерация чисел без ведущих нулей

Пусть требуется сгенерировать и проанализировать все N-значные числа (без ведущих нулей). Рекурсивная процедура, которая решает поставленную задачу может выглядеть так:

```
  procedure Rec(N, параметр : longint);
    var
      y : longint;
    begin
      if N=0 then begin анализ exit; end;
      if N=1 then for y:=1 to 9 do Rec(N-1, новое значение п-ра 1)
      if N>1 then for y:=0 to 9 do Rec(N-1, новое значение п-ра 2)
    end;
```

Вызов процедуры осуществляется так:

Rec(N,начальное значение параметра);

Генерация чисел из определенных цифр

Рассмотрим на примере решения задачи, в которой генерируются последовательности чисел из цифр 2 4 6 8 и анализируется сумма цифр в последовательности:

```
procedure Rec(N,Sum:longint);
  begin
    if N=0 then begin Анализ exit; end;
    Rec(N-1,Sum+2);
    Rec(N-1,Sum+4);
    Rec(N-1,Sum+6);
    Rec(N-1,Sum+8);
  end;
```

Вызов этой процедуры осуществляется так:

Rec(N,0);

Процедура Rec работает так:

Если N=0 то анализируем полученную сумму и выходим из процедуры

Вызываем процедуру Rec добавив цифру 2

Вызываем процедуру Rec добавив цифру 4

Вызываем процедуру Rec добавив цифру 6

Вызываем процедуру Rec добавив цифру 8

Все делители числа X

Строим все простые делители числа x (можно непосредственно, поскольку по условию все они не больше 1000). Пусть kp будет их количество, а сами они содержатся в массиве p[i], где I от 1 до kp.

А далее рекурсивно строим все возможные делители числа x, сконструированные как произведения вида

p[1]^i1*p[2]^i2*…p[kp]^ikp,

где i1,i2,…ikp - числа от 0 до максимально возможного значения, оставляющее произведение делителем числа x.

Рекурсивная процедура конструирования всех возможных чисел, составленных произведением простых p[i] в произвольных степенях от 0 до максимально возможной выглядит так:

```
procedure Rec(r,n:qword);
 Var
   i : longint;
 begin
```

```
      используем очередной построенный делитель r
      for i:=n to kp do
        if (x mod (r*p[i]))=0
          then Rec(r*p[i],i);
    end;
```

Вызов рекурсивной функции осуществляется так:

```
 Rec(1,1);
```

Здесь r - текущее произведение (начинаем с 1)

n - номер делителя, с которого продолжается увеличение произведения (начинаем с 1)

Рекурсивная процедура работает так:

Используем очередной построенный делитель r

Для всех i от текущего делителя до последнего

*Если x делится на r*p[i]*

*То вызываем рекурсию с параметрами r*p[i] и i*

Рекурсия с запоминанием

ЗАДАЧА: Задано клетчатое поле 1xN клеток. Необходимо определить количество способов раскрасить это поле в два цвета так, чтобы никакие из K рядом находящихся клеток небыли закрашены одним цветом (каждая клетка может быть окрашена только в один из двух цветов).

РЕШЕНИЕ: Запоминать (мемоизировать) будем количество способов раскрасить поле из Current клеточек, заполнив одним цветом Fill клеточек (в массиве F[1..50,0..50]).

Если элемент массива уже вычислен, рекурсия не вызывается, а ответ берется из массива F, иначе, рекурсия вызывается, а вычисленный рекурсией ответ сохраняется в массиве F.

Рекурсивные вычисления таковы:

```
                    0, Если Fill>=k
Rec(Current,Fill) = 1, Если Current>N
                    Rec(Current+1,Fill+1)+Rec(Current+1,1)
```

Т.е., количество способов, если мы раскрасим тем же цветом + количество способов, если начнем красить другим цветом.

```
var
```

```
  F       : array [1..50,0..50] of int64;
  n,k,i,j : longint;

  function Rec(Current,Fill : longint):int64;
    var
      Res : int64;
    begin
      if Fill>=k            then begin Rec:=0; exit; end;
      if Current>N          then begin Rec:=1; exit; end;
      if F[Current,Fill]<>0
         then begin Rec:=f[Current,Fill]; exit; end;
      Res:=Rec(Current+1,Fill+1) + Rec(Current+1,1);
      F[Current,Fill]:=Res;
      Rec            :=Res;
    end;
begin
  readln(n,k);
  for i:=1 to n do
    for j:=0 to k do f[i,j]:=0;
  writeln(Rec(1,0));
end.
```

Заключение

В данной статье представлена методика изучения темы рекурсивная генерация комбинаторных объектов с учениками 5-8 классов, предлагающая, по мнению автора, наиболее простой способ постепенного осознания механизма рекурсии и способа решения задач с её помощью. Методика включает в себя последовательность усложняющихся задач, снабжённых, где необходимо, предварительными общими пояснениями и последующими полными решениями предлагаемых задач. Полные тексты условий задач и решений вынесены в дополнительные материалы в связи с ограничениями на размер статьи. Школьникам предлагается перед чтением решения каждой задачи попытаться вначале самостоятельно воспользоваться приведенными предварительно общими теоретическими замечаниями.

Литературные и интернет-источники

1. *Долинский М.С., Кугейко М.А.* Гомельская инструментальная система дистанционного обучения // Информатика и образование. 2010. № 11, с 69-74

2. *Долинский М.С., Кугейко М.А.* Компьютерные средства развития мышления у дошкольников и младших школьников // Информатика и образование. 2011. № 6, с.71-75

3. *Долинский М.С.* "Гомельская школа олимпиадного программирования" // Информатика и образование. 2015, № 7,с.82-87

4. *Долинский М.С.* Алгоритмизация и программирование на TURBO PASCAL: от простых до олимпиадных задач: Учебное пособие. СПб.: Питер, 2005

5. *Долинский М.С.* Решение сложных и олимпиадных задач по программированию: Учебное пособие. СПб.: Питер, 2006

6. Статистика результатов гомельчан на международных и республиканских олимпиадах по информатике 1997-2016. http://dl.gsu.by/olymp/result.asp