М. С. Долинский,
Гомельский государственный университет имени Франциска Скорины, Республика Беларусь

ВВЕДЕНИЕ В РЕШЕНИЕ ЗАДАЧ С ПОМОЩЬЮ РЕКУРСИВНЫХ ПРОЦЕДУР И ФУНКЦИЙ

Аннотация

В статье описана методика изучения темы «Рекурсия» при подготовке школьников к олимпиадам по информатике. Технической основой является разработанная под управлением автора инструментальная система дистанционного обучения.

Ключевые слова: рекурсия, олимпиады по информатике, инструментальная система дистанционного обучения.

Контактная информация

Долинский Михаил Семенович, канд. тех. наук, доцент, доцент кафедры математических проблем управления Гомельского государственного университета имени Франциска Скорины, Республика Беларусь; *адрес:* 246000, Республика Беларусь, г. Гомель, ул. Советская, д. 104; *телефон:* (375-232) 77-70-69; *e-mail:* dolinsky@gsu.by

M. S. Dolinsky,
Gomel State University named after Francisk Skorina, the Republic of Belarus

INTRODUCTION TO PROBLEMS SOLVING WITH RECURSION

Abstract

The article describes the methodology to teach problem solving using recursion on Olympiads in informatics. Distance learning system is the effective technical base for teaching.

Keywords: recursion, teaching for programming, Olympiads in informatics, distance learning tools.

С сентября 1996 года на базе средней школы № 27 г. Гомеля (Республика Беларусь), а с сентября 1999 года дополнительно и на базе сайта дистанционного обучения DL.GSU.BY [4] ведется работа по факультативному обучению информатике и программированию школьников разных возрастов. Ключевая особенность этого обучения — его раннее начало, фактически с первого класса. Как следствие уже в пятом-шестом классах есть ученики, которым имеет смысл объяснять такие темы, как рекурсия. Поскольку традиционные подходы рассчитаны на обучение как минимум старшеклассников, то необходимы более простое и наглядное объяснение и более медленное продвижение по учебному материалу с явным выделением и обозначением всех этапов этого продвижения. Данная статья предлагает читателям соответствующий материал: текст содержит последовательное изложение необходимых сведений и задачи на закрепление. Проверка решений осуществляется автоматически на сайте DL.GSU.BY.

Мы рассмотрим понятие рекурсивной процедуры и решение следующих задач на одномерном массиве чисел a_1, a_2, ..., a_N:

- количество способов составления суммы M;
- количество способов составления суммы, не меньшей, чем M;
- нахождение максимальной суммы, не превосходящей M;
- нахождение суммы, минимально превышающей M.

Все эти задачи решаются последовательным анализом всех подмножеств заданного множества чисел (массива a), однако они не требуют запоминания и вывода тех подмножеств чисел, которые обеспечили решение.

Также мы рассмотрим запоминание подмножеств чисел, обеспечивающих решение, и следующие задачи:

- вывод всех способов составления суммы M;
- вывод одного из способов составления суммы M;
- вывод оптимального способа составления суммы M (накапливаются две суммы);
- вывод подмножества с максимальным числом подходящих элементов;
- разбиение массива a на подмножества, дающие суммы M_1, M_2, ..., M_k.

Задача 1. Количество способов составления суммы M.

Задача 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 из 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.

Заметим, что такая *процедура, которая вызывает сама себя, и называется* ***рекурсивной***.

Для того чтобы она работала корректно (не вызывала себя бесконечно), необходимо, чтобы:

- в ней были условия выхода (без рекурсивного вызова) при некоторых значениях;
- последовательные вложенные вызовы рекурсивной процедуры вели к этим значениям.

В нашем случае такими значениями являются:
$N = 0$ — нет больше элементов;
$M = 0$ — набрана требуемая сумма.

При каждом вызове мы идем к этим значениям, уменьшая N и оставляя прежним или уменьшая M.

Добавим, что такая процедура фактически перебирает множество всех подмножеств из заданных элементов массива a и считает, сколько раз сумма элементов получившегося подмножества равна заданному в условиях задачи числу M. В предложенной реализации процедура *Rec* работает с «конца» массива: сначала в подмножество добавляется/не добавляется элемент a_N, затем добавляется/не добавляется элемент a_{N-1}, ..., последним добавляется/не добавляется элемент a_1. При этом сами выбранные элементы не хранятся, поскольку в этом нет необходимости — достаточно просто в рекурсивную процедуру передавать новую сумму, которую дальше нужно собирать, — если элемент взяли, то сумма уменьшается на его величину, а если не взяли, то сумма остается прежней.

Задача 1.1.

Требуется написать программу, которая вводит числа M, N и последовательность из N чисел a_i и подсчитывает количество способов, которым из чисел a_i можно составить заданную сумму M.

Замечание. Варианты, в которых сумма составляется из одинаковых чисел, но находящихся на различных местах в последовательности, считать различными способами составления суммы.

На входе: числа M, N, последовательность из N чисел a_i.

На выходе: количество способов, которым из чисел a_i можно составить сумму M.

Пример ввода	*Пример вывода*
10 6 1 10 7 4 2 3	4
0 3 0 0 0	8

Методическое замечание. Здесь и далее при получении условия конкретной задачи учащимся предлагается перед чтением решения попытаться сначала самостоятельно воспользоваться приведенными предварительно общими теоретическими замечаниями.

Ниже представлен полный текст решения.

```
var
 a: array [1..100] of longint;
 i, M, N, ans: longint;

procedure Rec(N, M: longint);
begin
 if (N=0) and (M=0) then inc(ans);
 if N=0 then exit;
 Rec(N-1, M-a[N]);
 Rec(N-1, M)
end;

begin
 assign(input, 'input.txt'); reset(input);
 assign(output, 'output.txt'); rewrite(output);
 readln(M);
 readln(N);
 for i:=1 to N do read(a[i]);
 ans:=0;
 Rec(N, M);
 writeln(ans);
 close(input); close(output);
end.
```

Задача 2. Количество способов составления суммы, не меньшей, чем *M*.

Решение этой задачи, очевидно, требует замены только одной строки в решении задачи 1. Вместо:

```
if (N=0) and (M=0) then inc(ans);
```

требуется написать:

```
if (N=0) and (M<=0) then inc(ans);
```

Задача 3. Нахождение максимальной суммы, не превосходящей *M*.

Эту задачу можно решить с помощью следующей рекурсивной процедуры:

```
procedure Rec(N, Sum: longint);
begin
 if (N<0) or (Sum>M) then exit;
 if Sum>MaxSum then MaxSum:=Sum;
 Rec(N-1, Sum+a[N]);
 Rec(N-1, Sum);
end;
```

Вызов рекурсивной процедуры осуществляется так:

```
MaxSum:=-maxlongint;
Rec(N, 0);
```

Здесь:

- *MaxSum* — накапливаемая максимальная сумма, не превышающая граничное значение *M*. Она, как и граничное значение *M*, передается в рекурсивную процедуру *Rec* как глобальная переменная;
- *Sum* — текущая сумма выбранных элементов, передается в рекурсивную процедуру *Rec* параметром.

Рекурсивная процедура *Rec* выполняет следующие действия:

Если элементов больше нет или набранная сумма Sum превышает M,
то выходим из процедуры.
Если набранная сумма больше, чем накапливаемая максимальная сумма MaxSum,
то запоминаем ее.

Вызываем Rec, добавив к сумме элемент a[N] и перейдя к предыдущему элементу (N – 1).
Вызываем Rec, НЕ добавив к сумме элемент a[N] и перейдя к предыдущему элементу (N – 1).

Задача 4. Нахождение суммы, минимально превышающей *M*.

Эту задачу можно решить с помощью следующей рекурсивной процедуры:

```
procedure Rec(N, Sum: longint);
begin
 if Sum>=M then
  begin
   if Sum-M<ans then ans:=Sum-M;
   exit;
  end;
 if N=0 then exit;
 Rec(N-1, Sum+a[N]);
 Rec(N-1, Sum);
end;
```

Вызов ее осуществляется так:

```
ans:=maxlongint;
Rec(N, 0);
```

Рекурсивная процедура работает следующим образом:

Если текущая сумма Sum превысила H,
то сравниваем их разность с текущим ответом (ans),
если разность (Sum – H) меньше ans, заносим ее в ans.
Если больше нет элементов,
выходим из процедуры.
Вызываем процедуру Rec, добавив в сумму элемент a[N].
Вызываем процедуру Rec, не добавив в сумму элемент a[N].

Все эти задачи решаются последовательным анализом всех подмножеств заданного множества чисел (массива *a*), однако они не требуют запоминания и вывода тех подмножеств чисел, которые обеспечили решение. Далее рассмотрим задачи, в которых требуется запоминать подмножества чисел, обеспечивающие решение.

Задача 5. Вывод всех способов составления суммы *M*.

Эта задача решается с помощью следующей процедуры:

```
procedure Rec(N, Sum: longint);
begin
 if (N=0) and (Sum<>0) then exit;
 if Sum=0 then
  begin
   PrintB;
   inc(ans);
  end;
 inc(kb);
 b[kb]:=a[N];
 Rec(N-1, Sum-a[N]);
 dec(kb);
 Rec(n-1, Sum);
end;
```

Вызов процедуры осуществляется так:

```
ans:=0; kb:=0;
Rec(N, M);
```

Здесь:

- *b* — глобальный массив, который хранит текущие выбранные элементы;
- *kb* — глобальная переменная — количество элементов в массиве *b*;
- *N* — формальный параметр рекурсивной процедуры, номер текущего просматриваемого элемента и одновременно количество элементов, которые осталось просмотреть;
- *Sum* — формальный параметр рекурсивной процедуры, остаток суммы к набору.

Рекурсивная процедура *Rec* работает так:

Если все элементы просмотрены и сумма не набрана,
то выход.
Если сумма набрана,
то выводим текущее состояние массива B из KB элементов,
инкрементируем ответ.
Добавляем текущий элемент в массив b.
Вызываем Rec.
Забираем последний элемент из B.
Вызываем Rec (без добавления текущего элемента).

Задача 5.1.

Подсчитать количество различных представлений заданного натурального *N* в виде суммы не менее двух попарно различных положительных слагаемых.

Вывести все эти способы.

На входе: число *N*.

На выходе: все возможные представления числа *N* и количество таких представлений.

Пример ввода	*Пример вывода*
7	6+1 5+2 4+3 4+2+1 4

Замечание. Поскольку каждое из чисел 1, 2, ..., *N* – 1 меньше суммы *N*, которую требуется набрать, то слагаемых всегда будет не меньше двух.

Полный текст решения представлен ниже:

```
var
 a, b: array [1..100] of longint;
 i, M, N, ans, kb: longint;

procedure PrintB;
begin
 for i:=1 to kb-1 do write(b[i], '+');
 writeln(b[kb]);
end;

procedure Rec(Sum, K: longint);
begin
 if (k=0) and (Sum<>0) then exit;
 if Sum=0 then
  begin
   PrintB;
   inc(ans);
  end
 else
  begin
   if a[k]<=Sum then
    begin
```

```
    inc(kb); b[kb]:=a[k];
    Rec(Sum-a[k], k-1);
    dec(kb);
   end;
  Rec(Sum, k-1);
 end;
end;

begin
 readln(N);
 for i:=1 to N-1 do a[i]:=i;
 ans:=0;
 Rec(N, N-1);
 writeln(ans);
end.
```

Задача 6. Вывод одного из способов составления суммы *M*.

Задача решается с помощью такой процедуры:

```
procedure Rec(N, Sum: longint);
begin
 if (N=0) and (Sum<>0) then exit;
 if Sum=0 then PrintB_and_Halt;
 inc(kb);
 b[kb]:=a[N];
 Rec(N-1, Sum-a[N]);
 dec(kb);
 Rec(Sum, N-1);
end;
```

Вызов процедуры осуществляется так:

```
ans:=0; kb:=0;
Rec(N, M);
```

Отличие от предыдущего решения заключается в том, что в случае нахождения ответа вызывается процедура, которая выводит ответ и завершает выполнение программы, например, так:

```
procedure PrintB_and_Halt;
begin
 for i:=1 to kb-1 do write(b[i], '+');
 writeln(b[kb]);
 halt;
end;
```

Задача 7. Вывод оптимального способа составления суммы *M* (накапливаются две суммы).

Пример такой задачи выглядит так:

Имеется автомобиль с начальной силой тяги f_0 и начальной массой m_0. Имеется также N деталей, i-я из которых увеличивает силу тяги на f_i, а массу — на m_i. Ускорение автомобиля вычисляется по формуле $a = f/m$. Требуется добавить к автомобилю некоторые из деталей так, чтобы получить в итоге максимальное ускорение автомобиля, и вывести номера добавленных деталей.

Задача решается с помощью следующей рекурсивной процедуры:

```
procedure Rec(N, ft, mt: longint);
begin
 if ft/mt>a then
  begin
   a:=ft/mt;
   C:=B;
   kc:=kb;
  end;
 if N=0 then exit;
 inc(kb);
 b[kb]:=N;
 Rec(N-1, ft+f[N], mt+m[N]);
 dec(kb);
 Rec(N-1, ft, mt);
end;
```

Вызов процедуры осуществляется следующим образом:

```
Rec(N, f0, m0);
```

Здесь:

- $f0$ — начальная сила;
- $m0$ — начальная масса;
- ft — текущая сила;
- mt — текущая масса;
- a — текущее наилучшее ускорение (сначала равно $f0 / m0$);
- B — массив номеров деталей, взятых в текущей рекурсии;
- C — массив номеров деталей, повышающих ускорение.

Рекурсивная процедура *Rec* работает так:

Если текущее ускорение больше, запоминаем его и массив выбранных деталей.
Если больше нет деталей, выходим из процедуры.
Добавляем деталь в массив B, вызываем рекурсивную процедуру, забираем деталь из B.
Вызываем рекурсивную процедуру (без добавления текущей детали в B).

По завершении рекурсивной процедуры вызывается процедура *PrintC*, которая выводит или ответ (номера взятых деталей в порядке возрастания), или NONE — если ни одна комбинация деталей не приводит к повышению ускорения.

Заметим, что если массивы *B* и *C* объявлены как массивы одного и того же типа и размерности, например:

```
B, C: array [1..20] of longint;
```

то копирование массивов можно выполнить одним оператором присваивания:

```
C:=B;
```

Задача 7.1. Need For Speed (задача олимпиады USACO Bronze 2010 March).

Бесси готовит свой автомобиль к предстоящему Гран-при, но она хочет купить некоторые детали, чтобы повысить производительность автомобиля. В настоящее время ее автомобиль имеет массу M ($1 \le M \le 1000$) и способен перемещать себя вперед с силой F ($1 \le F \le 1\,000\,000$). В магазине есть N деталей ($1 \le N \le 20$), последовательно пронумерованных от *1* до *N*. Бесси может купить любое количество деталей, но не более чем по одному экземпляру каждого вида.

Деталь P_i добавляет силу F_i ($1 \le F_i \le 1\,000\,000$) и имеет массу M_i ($1 \le M_i \le 1000$). Второй закон Ньютона гласит, что $F = MA$, где F — сила, M — масса, A — ускорение.

Если Бесси хочет максимизировать общее ускорение своего гоночного автомобиля (и, другими словами, минимизировать массу), какие дополнительные детали она должна купить?

Рассмотрим авто с начальными значениями $F = 1500$ и $M = 100$.

Пусть имеется четыре вида деталей для улучшения:

i	F_i	M_i
1	250	25
2	150	9
3	120	5
4	200	8

Добавив только деталь 2, мы получим ускорение:

(1500 + 150) / (100 + 9) = 1650 / 109 = 15,13761.

Ниже приведена таблица, показывающая результирующее ускорение для всех возможных комбинаций добавления/недобавления четырех деталей (в первой колонке единица означает, что данная деталь добавляется, ноль — что деталь не добавляется).

Детали 1234	Суммарная F	Суммарная M	F/M	
0000	1500	100	15,0000	
0001	1700	108	15,7407	
0010	1620	105	15,4286	
0011	1820	113	16,1062	
0100	1650	109	15,1376	
0101	1850	117	15,8120	
0110	1770	114	15,5263	
0111	1970	122	16,1475	← highest F/M
1000	1750	125	14,0000	
1001	1950	133	14,6617	
1010	1870	130	14,3846	
1011	2070	138	15,0000	
1100	1900	134	14,1791	
1101	2100	142	14,7887	
1110	2020	139	14,5324	
1111	2220	147	15,1020	

Таким образом, наилучший вариант добавления — комбинация деталей 2, 3 и 4.

Формат ввода:

Строка 1: три разделенных пробелами целых числа: F, M, N.

Строки со 2-й по (N + 1)-ю: строка i + 1 содержит два разделенных пробелами целых числа: F_i и M_i.

Формат вывода:

Строки с 1-й по P-ю: номера дополнительных частей, по одному в строке, которые Бесси должна добавить. Если невыгодно что-либо добавлять, вывести NONE. Вывод нужно выполнять в порядке возрастания: т. е. если выгодно купить детали 2, 4, 6, 7, то можно выводить так:

2 4 6 7

и нельзя, например, так:

4 2 7 6.

Пример ввода (файл boost.in)	*Пример вывода (файл boost.out)*	*Детали вывода*
1500 100 4 250 25 150 9 120 5 200 8	2 3 4	Бесси должна добавить детали 2, 3, 4, чтобы максимизировать ускорение своего автомобиля

Полный текст решения представлен ниже:

```
var
 f, m, b, c: array [1..20] of longint;
 f0, m0, N, i, kb, kc: longint;
 a: real;

procedure PrintC;
var
 i: longint;
begin
 if kc>0 then
  for i:=kc downto 1 do writeln(c[i])
 else writeln('NONE');
end;

procedure Rec(ft, mt, N: longint);
begin
 if ft/mt>a then
  begin
   a:=ft/mt;
   C:=B;
   kc:=kb;
  end;
 if N=0 then exit;
 inc(kb);
 b[kb]:=N;
 Rec(ft+f[N], mt+m[N], N-1);
 dec(kb);
 Rec(ft, mt, N-1);
end;

begin
 assign(input, 'boost.in'); reset(input);
 assign(output, 'boost.out'); rewrite(output);
 readln(f0, m0, N);
 for i:=1 to n do readln(f[i], m[i]);
 a:=f0/m0; kc:=0; kb:=0;
 Rec(f0, m0, N);
 PrintC;
 close(input); close(output);
end.
```

Задача 8. Вывод подмножества с максимальным числом подходящих элементов.

Задача решается с помощью следующей рекурсивной процедуры:

```
procedure Rec(N: longint);
begin
 if N=0 then exit;
 if Good(N) then
  begin
   inc(kb);
   b[kb]:=a[N];
   Rec(N-1);
   if kb>MaxKB then MaxKB:=kb;
   dec(kb);
  end;
 Rec(N-1);
end;
```

Вызывается эта процедура так:

```
Rec(N);
```

Здесь *MaxKB* — глобальная переменная — максимальное количество элементов в массиве *B* (сначала равна 0).

Рекурсивная процедура *Rec* работает так:

Если все элементы просмотрены, выйти из процедуры.
Если добавление текущего элемента создает подходящее множество (определяется функцией Good),
то добавляем элемент в массив B, вызываем рекурсию.
Если текущее количество элементов в массиве B больше, чем MaxKB, запоминаем его.
Удаляем элемент из массива B.
Вызываем рекурсию (без добавления текущего элемента).

Задача 8.1. Escape (задача олимпиады USACO Bronze 2011 December).

Коровы планируют сбежать от фермера Джона, переплыв на плоту через реку. Проблема заключается в том, что плот может не выдержать всех желающих. N коров ($1 \le N \le 20$) имеют вес $w_1..w_N$. У коров плохо со сложением, они не умеют выполнять перенос. Вам требуется

определить размер наибольшей группы коров, вес которых можно сложить без переноса при сложении.

Формат ввода:

Строка 1: количество коров N ($1 \le N \le 20$).

Строки со 2-й по ($N+1$)-ю: каждая строка содержит вес одной коровы — целое число от 1 до 100 000 000.

Формат вывода:

Строка 1: максимальное количество коров, чей вес может быть сложен без переноса.

Пример ввода (файл escape.in)	*Пример вывода (файл escape.out)*	*Пояснения*
5 522 6 84 7311 19	3	Имеется пять коров с весом: 522, 6, 84, 7311, 19. Три веса: 522, 6, 7311 могут быть сложены без переноса. 522 6 + 7311 7839

Идея решения: рекурсивно формируем все суммы из «подходящих» слагаемых (когда не возникает переноса при сложении).

Проверку осуществляет функция *Good*:

```
function Good(s, w: longint): boolean;
var
 ts, tw: longint;
 tGood: boolean;
begin
 tGood:=true;
 while tGood and ((s>0) or (tw>0)) do
  begin
   ts:=s mod 10;
   tw:=w mod 10;
   tGood:=ts+tw<10;
   s:=s div 10;
   w:=w div 10;
  end;
 Good:=tGood;
end;
```

Здесь из слагаемых s, w выделяются и поразрядно складываются цифры, пока их сумма не превосходит 10.

Рекурсивная процедура, использующая описанную функцию *Good*, может выглядеть так:

```
procedure Rec(tN, S: longint);
begin
 if tN=0 then
  begin
   if kb>MaxKB then MaxKB:=kb;
   exit;
  end;
 if Good(S, w[tN]) then
  begin
   inc(kb);
   Rec(tN-1, S+w[tN]);
   dec(kb);
  end;
 Rec(tN-1, S);
end;
```

Если сумма сформирована, то запоминаем наибольшее количество слагаемых (*MaxKB*) и выходим.

Запускаем рекурсию с имеющейся суммой s и новым слагаемым $w[tN]$, если при их сложении не возникает переноса ни в одном разряде (предварительно увеличиваем на 1 количество слагаемых kb, после завершения — уменьшаем на 1).

Запускаем рекурсию, пропуская значение $w[tN]$.

Полный текст решения представлен далее:

```
var
 w: array [1..20] of longint;
 N, i, MaxKB, kb: longint;

function Good(s, w: longint): boolean;
var
 ts, tw: longint;
 tGood: boolean;
begin
 tGood:=true;
 while tGood and ((s>0) or (tw>0)) do
  begin
   ts:=s mod 10;
   tw:=w mod 10;
   tGood:=ts+tw<10;
   s:=s div 10;
   w:=w div 10;
  end;
 Good:=tGood;
end;

procedure Rec(tN, S: longint);
begin
 if tN=0 then
  begin
   if kb>MaxKB then MaxKB:=kb;
   exit;
  end;
 if Good(S,w[tN]) then
  begin
   inc(kb);
   Rec(tN-1, S+w[tN]);
   dec(kb);
  end;
 Rec(tN-1, S);
end;

begin
 assign(input, 'escape.in'); reset(input);
 assign(output, 'escape.out'); rewrite(output);
 readln(N);
 for i:=1 to N do readln(w[i]);
 MaxKB:=0;
 kb:=0;
 Rec(N, 0);
 writeln(MaxKB);
 close(input); close(output);
end.
```

Задача 8.2. Попарно взаимно простые (Гомельская областная олимпиада 2012 года).

Дано N чисел. Определить, какое максимальное количество чисел из этого набора являются попарно взаимно простыми.

Формат ввода: N — количество чисел ($1 \le N \le 20$), далее следует N чисел a_i.($2 \le a_i \le 1000$).

Формат вывода: количество чисел.

Пример ввода	*Пример вывода*
5 2 4 7 14 9	3

Идея рекурсивного решения задачи. Нужно рассмотреть множества всех подмножеств, составленных

из a_i, и из этих подмножеств выбрать такое, которое состоит из наибольшего числа взаимно простых элементов. При добавлении нового элемента проверяем его на взаимную простоту с ранее добавленными элементами.

Ниже представлен полный текст решения задачи:

```
var
 a, b: array [1..20] of longint;
 nodp: array [1..20, 1..20] of longint;
 N, i, MaxKB, kb: longint;

function NOD(a, b: longint): longint;
begin
 if a=0 then NOD:=b
 else if b=0 then NOD:=a
 else if a>b then NOD:=NOD(a mod b, b)
 else if a<b then NOD:=NOD(b mod a, a)
 else if a=b then NOD:=a;
end;

function MutSimple(k: longint): boolean;
var i: longint;
begin
 i:=1;
 while (i<=kb) and (NOD(b[i],a[k])=1) do inc(i);
 MutSimple:=i>kb;
end;

procedure Rec(k: longint);
begin
 if k=0 then exit;
 if MutSimple(k) then
  begin
   inc(kb);
   b[kb]:=a[k];
   Rec(k-1);
   if kb>MaxKB then MaxKB:=kb;
   dec(kb);
  end;
 Rec(k-1);
end;

begin
 readln(N);
 for i:=1 to N do read(a[i]);
 MaxKB:=0;
 Rec(N);
 writeln(MaxKB);
end.
```

Задача 9. Разбиение массива *a* на подмножества, дающие суммы M_1, M_2, ..., M_k.

Задача решается с помощью следующего рекурсивного решения.

Рекурсивно собираем сумму M_k, помечая, какие a_i для этого использовали. Если удалось, то вызываем ту же рекурсию для сбора уже M_{k-1}, используя только неиспользованные ранее a_i.

Если удалось добраться до $k = 0$ (т. е. собрали все M_i для i от k до 1) и все a_i использованы, то ответ YES. Если же какие-то a_i не использованы, то ответ NO и программа завершает свое выполнение.

Ответ NO также выдается при окончательном выходе из рекурсии, т. е. если все варианты разбиения a_i на подмножества просмотрены, а получить все суммы не удалось ни в одном разбиении.

Текст соответствующей рекурсивной процедуры:

```
procedure Rec(N, k, Sum, ks, ka: longint);
var i: longint;
begin
 if k=0 then
  begin
   if ka=nfix then writeln('YES')
   else writeln('NO');
   close(input); close(output);
   halt(0);
  end;
 if (Sum=M[k]) and (ks<>0) then
  Rec(N, k-1, 0, 0, ka);
 if N=0 then exit;
 for i:=1 to nfix do
  if x[i]=0 then
   begin
    x[i]:=1;
    Rec(N-1, k, sum+a[i], ks+1, ka+1);
    x[i]:=0;
   end;
end;
```

Вызов рекурсии осуществляется так:

```
nfix:=N;
Rec(N, k, 0, 0, 0);
writeln('NO');
```

Здесь:

- N, k — количества оставшихся элементов множеств a_i и M_i соответственно;
- *Sum* — текущая сумма выбранных a_i;
- *ks* — количество элементов a_i, из которых сформирована текущая сумма;
- *ka* — количество элементов a_i, использованных для формирования всех сумм;
- *nfix* — глобальная переменная, содержащая количество элементов в массиве *a*, это значение не меняется в процессе вычислений;
- x — массив, описывающий, какие элементы из массива *a* использованы ($x_i = 1$), а какие нет ($x_i = 0$).

Процедура *Rec* работает так:

Если все требуемые суммы собраны,
  то если все элементы массива a использованы,
    то выводим YES,
    иначе выводим NO.
  Завершаем выполнение программы.
Если текущая сумма M_k собрана и собраны не все суммы,
  то вызываем рекурсию для сбора следующей суммы: `Rec(N, k-1, 0, 0, ka);`
Если все элементы массива a использованы, выходим из рекурсии.
Для всех элементов из массива a:
  если он не использован,
  то помечаем его как использованный.
  Вызываем рекурсию: `Rec(N-1, k, sum+a[i], ks+1, ka+1);`
  помечаем его как неиспользованный.

Задача 9.1. Ksubset (Российская командная интернет-олимпиада 23 февраля 2008 года).

Одной из классических NP-полных задач является так называемая ***задача о сумме подмножества*** (subset-sum problem), которая формулируется следующим образом:

«Дано N чисел a_1, a_2, ..., a_N. Возможно ли выбрать несколько из них так, чтобы сумма была равна S?»

Эта задача допускает несколько обобщений. Одно из них мы назовем «Задача о сумме k подмножеств» и будем рассматривать далее:

«Пусть задано N чисел $a_1, a_2, ..., a_N$ и k чисел $s_1, s_2, ..., s_k$. Необходимо разбить числа $a_1, a_2, ..., a_N$ на k непустых наборов так, чтобы сумма чисел в первом из них была равна s_1, во втором — s_2, ..., в k-м — s_k. Каждое число может быть включено не более чем в один набор».

Напишите программу, решающую задачу о сумме k подмножеств.

Формат входного файла.

Первая строка входного файла содержит два целых числа N и k ($1 \le N, k \le 10$, $N^k \le 2^{24}$).

Вторая строка входного файла содержит N целых чисел $a_1, a_2, ..., a_N$ (для всех a_i верно неравенство $|a_i| \le 10^9$).

Третья строка входного файла содержит k целых чисел $s_1, s_2, ..., s_k$ (для всех s_i верно неравенство $|s_i| \le 10^9$).

Формат выходного файла.

Если искомое разбиение на наборы существует, то выведите в первой строке выходного файла слово YES, иначе — слово NO.

Пример ввода (файл ksubset.in)	*Пример вывода (файл ksubset.out)*
4 2 1 2 3 4 5 5	YES
4 2 1 2 3 4 1 8	NO

Идея решения заключается в следующем.

Рекурсивно собираем сумму s_k, помечая, какие a_i для этого использовали.

Если удалось собрать сумму, то вызываем ту же рекурсию для сбора уже s_{k-1}, используя только не использованные ранее a_i.

Если удалось добраться до $k = 0$ (т. е. собрали все s_i для i от k до 1) и все a_i использованы — ответ YES. Если же какие-то a_i не использованы — ответ NO.

Ответ NO также выдается по окончательному выходу из рекурсии, т. е. если все варианты разбиения a_i на подмножества просмотрены, а получить все суммы не удалось ни в одном разбиении.

Параметры рекурсии *Rec*(N, k, *Sum*, *ks*, *ka*):

N, k — количества оставшихся элементов множеств a_i и s_i соответственно;

Sum — текущая сумма выбранных a_i;

ks — количество элементов a_i, из которых сформирована текущая сумма;

ka — количество элементов a_i, использованных для формирования всех сумм.

Далее представлен полный текст решения:

```
var
a, s, x: array [1..10] of longint;
i, N, k, nfix: longint;

procedure Rec(N, k: longint;
          Sum: int64;
          ks, ka: longint);
var
 i: longint;
begin
 if k=0
  then
   begin
    if ka=nfix
     then writeln('YES') else writeln('NO');
    close(input); close(output);
    halt(0);
   end;
 if (Sum=s[k]) and (ks<>0)
  then Rec(N, k-1, 0, 0, ka);
 if N=0 then exit;
 for i:=1 to nfix do
  if x[i]=0
   then
    begin
     x[i]:=1;
     Rec(N-1, k, sum+a[i], ks+1, ka+1);
     x[i]:=0;
    end;
end;

begin
 assign(input, 'ksubset.in'); reset(input);
 assign(output, 'ksubset.out'); rewrite(output);
 readln(N, k);
 for i:=1 to N do read(a[i]);
 for i:=1 to k do read(s[i]);
 for i:=1 to N do x[i]:=0;
 nfix:=N;
 Rec(N, k, 0, 0, 0);
 writeln('NO');
 close(input); close(output);
end.
```

* * *

В данной статье представлена методика изучения темы «Рекурсия» с учащимися пятых—восьмых классов, предлагающая, по мнению автора, наиболее простой путь постепенного осознания механизма рекурсии и способа решения задач с ее помощью. Методика включает в себя последовательность усложняющихся задач, снабженных там, где это необходимо, предварительными общими пояснениями и последующими полными решениями этих задач. Школьникам предлагается перед чтением решения каждой задачи попытаться сначала самостоятельно воспользоваться приведенными общими теоретическими замечаниями.

Литературные и интернет-источники

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

2. *Долинский М. С.* Гомельская школа олимпиадного программирования // Информатика и образование. 2015. № 7.

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

4. *Долинский М. С., Кугейко М. А.* Гомельская инструментальная система дистанционного обучения // Информатика и образование. 2010. № 11.

5. *Долинский М. С., Кугейко М. А.* Компьютерные средства развития мышления у дошкольников и младших школьников // Информатика и образование. 2011. № 6.

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