

МИНИСТЕРСТВО ПРОСВЕЩЕНИЯ УССР
ХЕРСОНСКИЙ ГОСУДАРСТВЕННЫЙ ПЕДАГОГИЧЕСКИЙ ИНСТИТУТ
им. Н.К.КРУПСКОЙ

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ
ПО ПОДГОТОВКЕ СТУДЕНТОВ К ОБЕСПЕЧЕНИЮ
КОМПЬЮТЕРНОЙ ГРАМОТНОСТИ УЧАЩИХСЯ
для студентов специальностей 2104 и 2105
"Математика" и "Физика"

Утверждено
на заседании кафедры математики
Протокол № 18 от 30.06.87 г.

Херсон ХГПИ 1988

Методические рекомендации по подготовке студентов к обеспечению компьютерной грамотности учащихся для студентов специальностей 2104 и 2105 "Математика" и "Физика"/Сост. А.В.Спиваковский, А.П.Ковтушенко. - Херсон: ХПТИ, 1988. - 60 с.

Составители А.В.Спиваковский, канд.физ.-мат.наук
А.П.Ковтушенко

Рецензенты П.Ф.Жук, канд. физ.-мат.наук
В.А.Прядко, канд. техн.наук

Данные методические рекомендации предназначены для учителей информатики и студентов физико-математических факультетов педагогических институтов.

Цель работы - привить студентам навыки применения средств вычислительной техники в учебном процессе.

Использование вычислительной техники в обучении предполагает организационную и методическую подготовку учебного процесса. При этом используются автоматические обучающие программы, ЭВМ для контроля знаний и ЭВМ как инструмент.

В предлагаемой работе подробно изложены описания алгоритмов решения задач курса "Теория чисел" и приведены программы, реализующие данные алгоритмы. Работа не содержит описания используемого языка программирования - языка Бейсик; она рассчитана на читателя уже знакомого с основами этого языка.

Курс "Теория чисел" с применением портативных ЭВМ должен быть прочитан в IV семестре после изучения курса "Основы информатики и вычислительной техники" в I-III семестрах и прохождения практики.

Использование вычислительной техники предполагает, во-первых, изучение предлагаемых ключевых алгоритмов, и во-вторых, разработку студентами программ для решения задач из курса "Теория чисел" на ЭВМ.

Использование вычислительной техники в других учебных курсах возможно на основе тех же принципов - решение задач, требующих существенного объема вычислений, причем ключевые алгоритмы, предложенные в качестве образца, должны быть особенно тщательно изучены, так как на их основе учащиеся будут разрабатывать программы для решения частных задач.

В качестве технической базы в работе использован ДВК-1, язык Бэйсик которого совпадает с минимальным подмножеством любой другой известной нам версии данного языка программирования.

Таким образом, приводимые в работе программы могут быть использованы на любой персональной ЭВМ без доработок. Разработан также пакет более совершенных программ на базе дискоточной версии Бэйсик ДВК-2М.

1. ДЕЛЕНИЕ С ОСТАТКОМ. ПРОСТЫЕ ЧИСЛА

Определение 1.1. Пусть a и b , причем $b \neq 0$, - целые числа. Назовем b делителем a , если существует целое q такое, что $a = bq$. В этом случае a называется кратным b , а q - частным от деления a на b .

За a каждый раз стоит некоторое конкретное число, делители которого нужно найти, причем этому действию вполне соответствует определенный порядок поиска делителей данного конкретного числа - алгоритм поиска делителей.

Рассмотрим АЛГОРИТМ 1.1. Изменяя значение B от 1 до A , проверяем правильность соотношения $[A/B] = A/B$, где $[]$ - целая часть числа. Те значения B , которые окажутся делителями числа A , сразу напечатаем.

Определение 1.2. Натуральное число называется простым, если оно больше 1 и не имеет делителей кроме 1 и себя самого.

Определение 1.3. Натуральное число называется составным, если оно имеет хотя бы один делитель, отличный от 1 и самого числа.

Для распознавания простых чисел служит АЛГОРИТМ 1.2, который отличается от предыдущего только действием при обнаружении делителя. При этом выдается сообщение - "A - составное" и задача решена. Если делитель не найден, выдается сообщение - "A - простое".

Определение 1.4. Каноническим разложением целого числа a , большего 1, называется представление его в виде:

$$a = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_k^{\alpha_k},$$

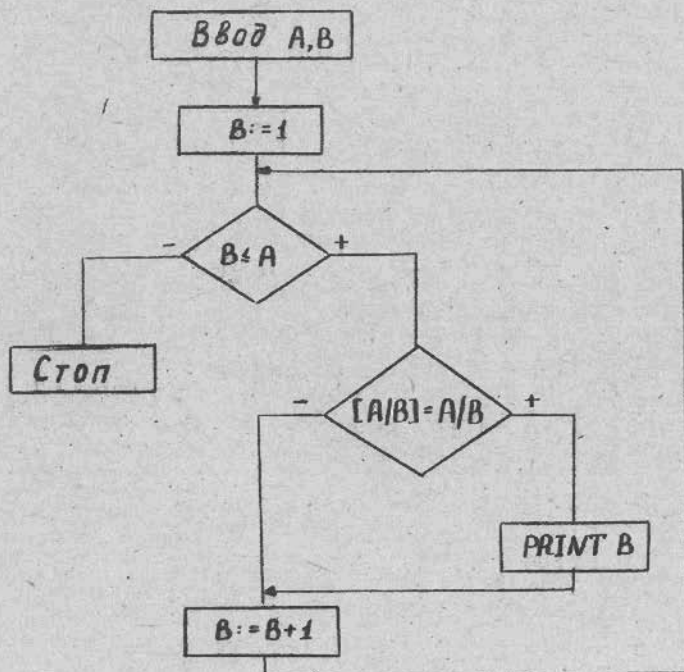
где p_k - попарно различные простые числа; α_k - натуральные числа.

Конечно, p_i - делители числа a , но одного этого для их поиска недостаточно. Для того, чтобы начать поиск p_k необходимо знать, что наименьший положительный делитель, отличный от 1, - всегда простое число!

```

05 REM ДЕЛИТЕЛИ ЧИСЛА
10 PRINT "ВВЕДИТЕ НАТУРАЛЬНОЕ ЧИСЛО:"
20 INPUT A
30 FOR B=1 TO A
40 IF INT(A/B)=A/B THEN PRINT B;
50 NEXT B

```

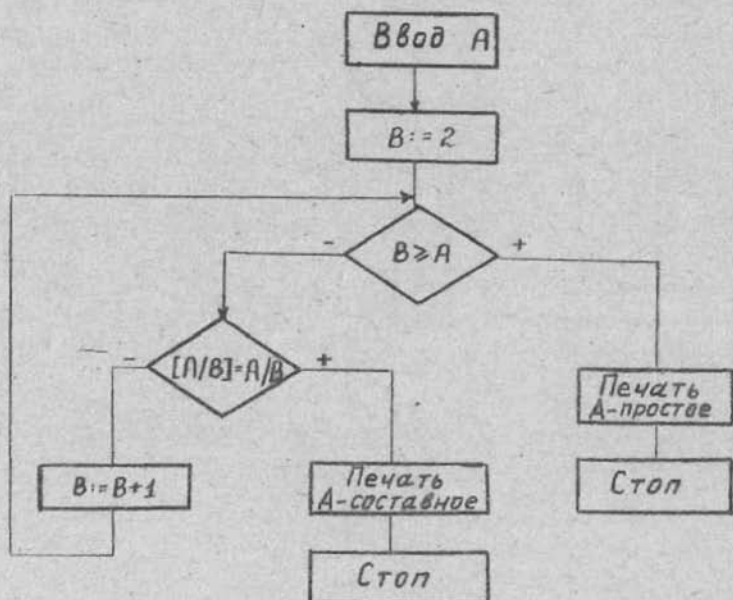


Алгоритм I.I

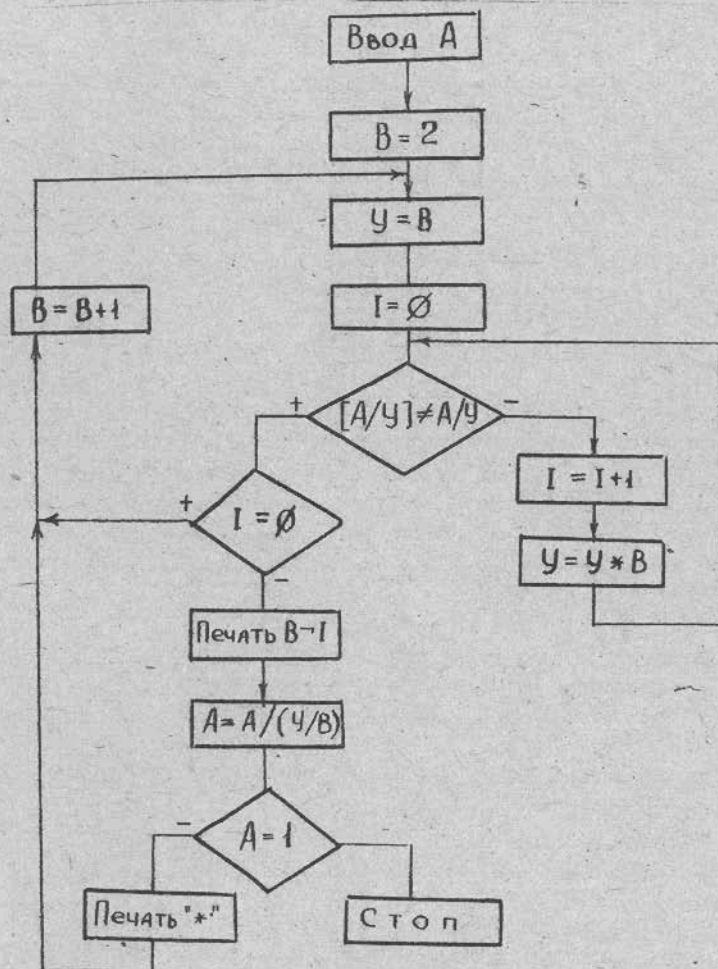
```

5 REM ОПРЕДЕЛЕНИЕ ПРОСТОТЫ ЧИСЛА
10 PRINT "ВВЕДИТЕ НАТУРАЛЬНОЕ ЧИСЛО"
20 INPUT A
30 B=2
40 IF B>=A THEN PRINT A;"- ЧИСЛО ПРОСТОЕ" : STOP
50 IF INT(A/B)=A/B THEN B=
60 B=B+1
70 GO TO 40
80 PRINT A;"- ЧИСЛО СОСТАВНОЕ"
90 STOP

```



Алгоритм I.2



Алгоритм 1.3а

Для того чтобы определить α_i , придется использовать тот факт, что если $P_i^{\alpha_i}$ - делитель числа a , а $P_i^{\alpha_i+1}$ - нет, то для нахождения α_i необходимо определить наименьшее $P_i^{\alpha_i+1}$, которое не является делителем числа a .

В АЛГОРИТМЕ 1.3 будем искать делители среди всех натуральных чисел в порядке возрастания, а при нахождении делителя поделим a на $P_i^{\alpha_i}$, после чего новое значение a не будет делиться на P_i и потому следующий наименьший делитель опять будет простым числом. Признаком завершения работы будет тот факт, что очередное значение a станет равным единице, т.е. список $P_k^{\alpha_k}$ исчерпан. Отметим, что имя I в программе соответствует L_1 , имя $Y - P_i^{\alpha_i+1}$.

/АЛГОРИТМ 1.36/.

```

5REMКАНОНИЧЕСКОЕПРЕДСТАВЛЕНИЕНАТУРАЛЬНОГОЧИСЛА
10PRINT"ВВЕДИТЕНАТУРАЛЬНОЕЧИСЛО"
20INPUTA
30B=2
40Y=B
50I=0
60IFINT(A/Y)<>A/YTHEN80
70I=I+1:Y=Y*B:GOTO60
80IFI=0THEN130
90PRINTB;"^";I
100A=A/(Y/B)
110IFA=1THEN150
120PRINT"*"
130B=B+1
140GOTO40
150END

```

Алгоритм 1.36

Рассмотренные алгоритмы 1.1-1.3 обладают общим недостатком - они крайне расточительны.

В АЛГОРИТМЕ 1.4 - печать простых чисел, не превосходящих заданного числа; для поиска делителя вводится ограничение не A , а только \sqrt{A} , поскольку известно следующее.

Теорема 1.1. Если натуральное число N не делится ни на одно простое число, не превосходящее \sqrt{N} , то оно простое. Но при этом приходится учитывать тот факт, что извлечение корня на ЭВМ делается с погрешностью, например $\sqrt{4} \approx 2$. Поэтому для исключения ошибок приходится вместо \sqrt{N} ставить $\sqrt{N} + 0,5$, что гарантирует от причисления квадрата простого числа к простым числам.

Нетрудно заметить, что, хотя в теореме говорится о делимости на простые числа, в АЛГОРИТМЕ I.4 все еще делим на натуральные числа, т.е. выполняем лишнюю работу.

Более эффективен по затрачиваемому времени алгоритм отсеивания составных чисел, известный еще древнегреческому математику Эратосфену /275 - 196 гг. до н.э./ и называемый сегодня решето Эратосфена:

1) в ряду натуральных чисел зачеркнем все четные числа, кроме самого числа 2;

2) зачеркнем все натуральные числа, делящиеся на число 3, кроме самого числа 3;

3) будем поступать аналогично со всеми следующими в порядке возрастания незачеркнутыми числами /т.е. само число оставляем, а все ему кратные числа вычеркиваем/;

4) когда, повторяя п.3, доберемся до числа, большего \sqrt{N} (где N - максимальное число в той таблице, из которой вычеркиваем числа), у нас останутся только простые числа.

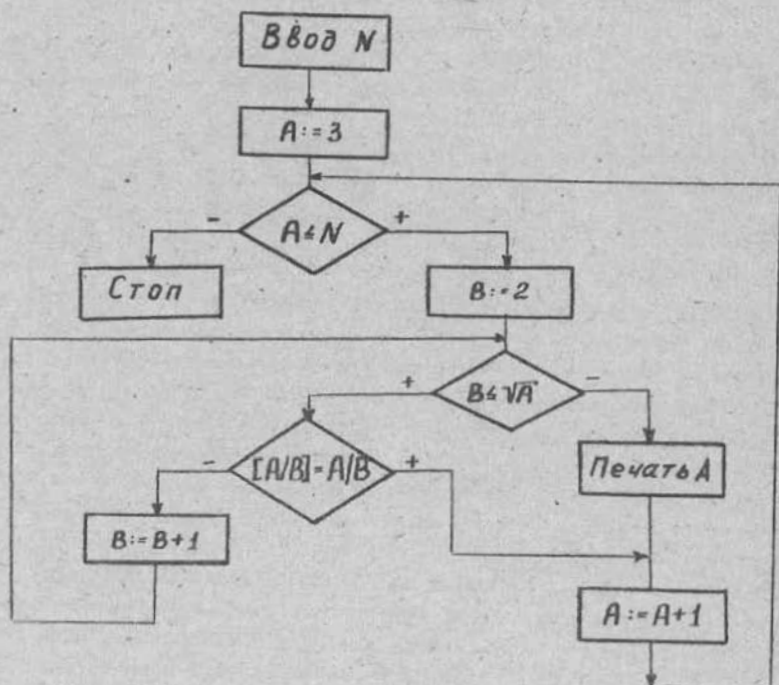
В АЛГОРИТМЕ I.5 выполняем предложенные действия, но с оговоркой. В предложенном виде решето Эратосфена расточительно использует уже не время, а память машины. Поскольку нужно хранить все рассматриваемые натуральные числа, этих чисел не может быть больше чем емкость данного массива (в нашем случае емкость A - 256 чисел). Поэтому рассматриваем не всю таблицу, а только ее верхнюю часть - от N до $N+255$. В этой части таблицы вычеркиваем составные числа /см. строки - ~~98-108~~ АЛГОРИТМА I.5в/. При этом X - это натуральное число, кратное которому вычеркиваем; I - это делитель числа $Q = X \cdot I$, которое вычеркиваем; $X < \sqrt{N+255}$ - значения натуральных чисел, кратные которым необходимо просмотреть при вычеркивании.

Рассмотрим теперь АЛГОРИТМ I.6 - улучшенный алгоритм решета Эратосфена, который не потребует памяти для хранения составных чисел, т.е. в данном случае бесполезных. Для хранения простых чисел, которые будут использованы как делители, используем массив по имени A , а все простые числа печатаем сразу же по их обнаружении. Имя K - натуральное число, проверяемое на наличие делителей. Делителями может быть простое число, не большее числа \sqrt{K} . Простые числа будут заноситься в массив A в порядке обнаружения, т.е. в порядке возрастания. Поэтому все те простые числа, которые претендуют на роль делителя, находятся в начале таблицы A с номером от I до $M-I$, где M - номер простого числа, квадрат которого превосходит K , поскольку из неравенства $M^2 > K$ следует неравенство $M > \sqrt{K}$. Значение M для каждого K вычисляется в строках ~~98-108~~ АЛГОРИТМА I.6в.

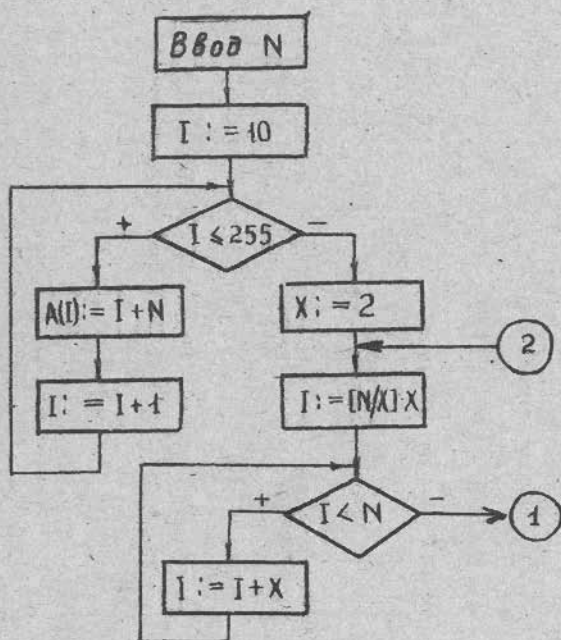
```

5 REM ПЕЧАТЬ ТАБЛИЦЫ ПРОСТЫХ ЧИСЕЛ
10 PRINT "ПЕЧАТАЕМ ПРОСТЫЕ ЧИСЛА ОТ 1 ДО "
20 INPUT N
30 PRINT 1,2,
40 A=3
50 IF A<=N THEN 70
60 STOP
70 B=2
80 IF B<=SQR(A) THEN 110
90 PRINT A,
100 A=A+1 : GO TO 50
110 IF INT(A/B)=A/B THEN 100
120 B=B+1
130 GO TO 80
140 END

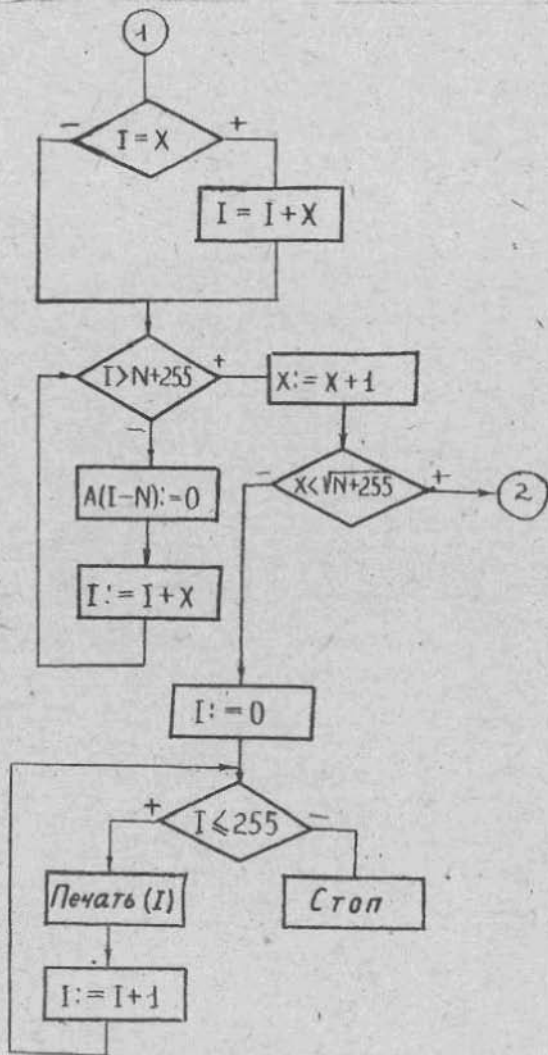
```



Алгоритм 1.4



Алгоритм 1.5а



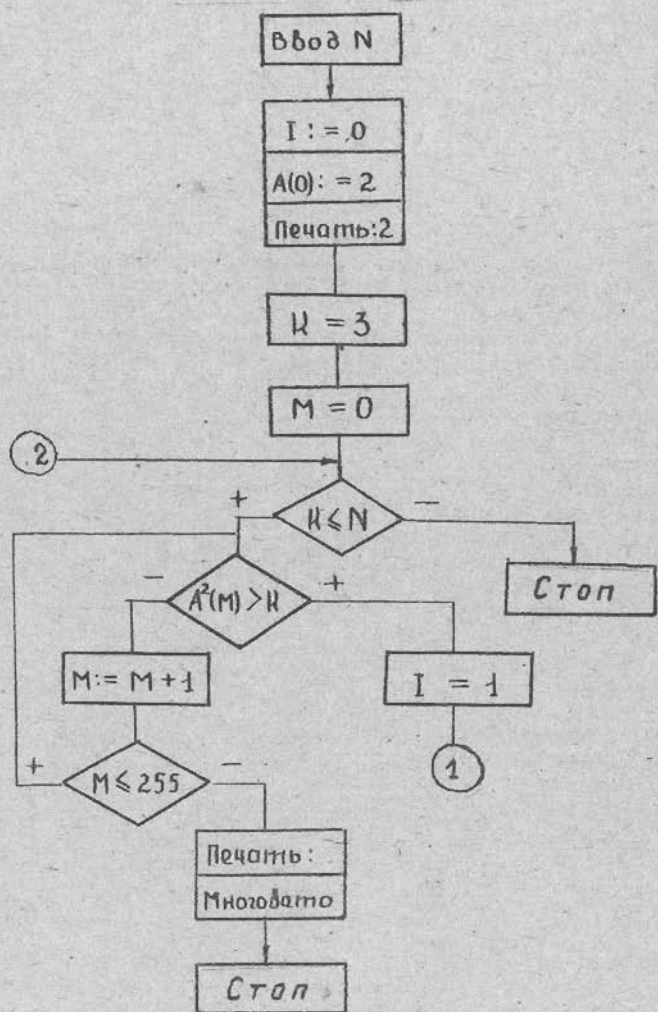
Алгоритм 1.56

```

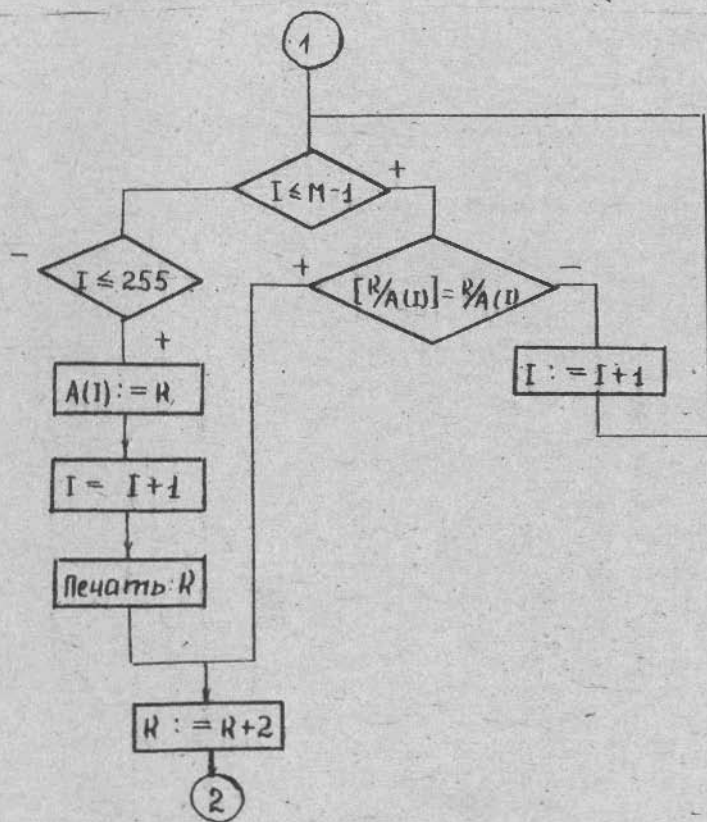
5 REMPEMETO3PATOCФЕНА
10 PRINT "ПЕЧАТАЕМРЕМЕТO3PATOCФЕНАДТНАОН+255"
15 PRINT "ВВЕДИТЕНАТУРАЛЬНОЕЧИСЛО"
20 INPUT N
30 IF N<=0 THEN PRINT "НАОЛЖНОВЫТЬНАТУРАЛЬНЫМ" : GOTO 15
40 PRINT "РЕМЕТO3PATOCФЕНАДТ" IN "ДО" IN N+255
50 DIM A(255)
60 FOR I=0 TO 255
70 A(I)=I+N
80 NEXT I
90 X=2
100 J=INT(N/X)*X
110 IF J<N THEN J=J+X : GOTO 110
120 IF J=X THEN J=J+X
130 IF J>N+255 THEN 170
140 A(J-N)=0
150 J=J+X
160 GOTO 130
170 X=X+1
180 IF X<8 OR (N+255) THEN 100
190 FOR I=0 TO 255
200 PRINT A(I)
210 NEXT I
220 PRINT
230 END

```

Алгоритм I.5в



Алгоритм I.6a



Алгоритм 1.66

```

10RENTАВЛИЦАПРОСТЫХЧИСЕЛСИСПОЛЬЗОВАНИЕММАССИВА
20PRINT"ПЕЧАТАЕМПРОСТЫЕЧИСЛАДО"
30INPUTN
40DIMA(255)
50I=0:A(0)=2:PRINT2
60K=3:M=0
70IFK<=NTHEN90
80PRINT"ВСЕI" :GOTO210
90IFA(M)=A(M)>KTHEN130
100M=M+1
110IFM<=255THEN90
120PRINT"ВЫЛИЗАПРЕДЕЛММАССИВА" :STOP
130I=0
140IFI<=M-1THEN180
150IFL<255THENL=L+1:A(L)=K
160PRINTK
170K=K+2:GOTO70
180IFINT(K/A(I))=K/A(I) THEN170
190I=I+1
200GOTO140
210END

```

Алгоритм I.6в

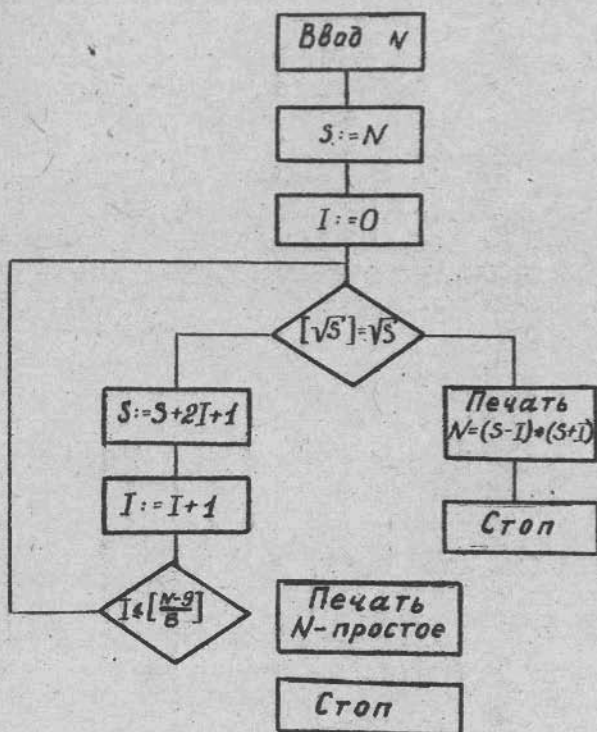
Так как кроме двойки нет ни одного четного простого числа, будем увеличивать K , начиная с числа 3, на два, т.е. K будет пробегать значения нечетных натуральных чисел. В связи с тем, что запомнить в качестве делителей можно только ограниченное количество простых чисел, в нашем случае не более 256, данный алгоритм не может выработать простое число большее чем 1619 в квадрате /число 1619 - 256-е простое число в натуральном ряду/, т.е. 2621161. Для исключения ошибки в случае, если N больше 2621161, нужно использовать строки 110-120 АЛГОРИТМА I.6в. Отметим, что проверка простоты числа K осуществляется в строках 180-200. Любопытно, что указатель J - "какое по счету простое число берется в качестве делителя" - начинает работу с $J = I/A(I) = 3/$, а не с $J = 0 / A(0) = 2 /$. Как помним, наше K принимает только нечетные значения!

В конце данного раздела предлагается самостоятельно разобрать АЛГОРИТМ I.7, представляющий процесс факторизации натурального числа.

```

5REM ФАКТОРИЗАЦИЯ НАТУРАЛЬНОГО ЧИСЛА
10PRINT "ВВЕДИТЕ НАТУРАЛЬНОЕ ЧИСЛО"
20INPUT N
30S=N
40FOR I=0 TO INT((N-9)/6)
50Y=INT(SQR(S)+.5)
60IF Y*Y=S THEN 100
70S=S+2*I+1
80NEXT I
90PRINT N; " - ПРОСТОЕ ЧИСЛО" : GOTO 110
100PRINT N; " = (Y-I) * (Y+I)"
110END

```



Алгоритм I.7

I7

2. НАИБОЛЬШИЙ ОБЩИЙ ДЕЛИТЕЛЬ. НАИМЕНЬШЕЕ ОБЩЕ КРАТНОЕ

Определение 2.1. А. Общим делителем целых чисел a_1, a_2, \dots, a_n называется любое целое d , являющееся делителем каждого из этих чисел. Б. Наибольшим общим делителем (НОД) целых чисел a_1, a_2, \dots, a_n называется такое натуральное d , которое делится на любой общий делитель этих чисел.

Определение дает исчерпывающий ответ на вопрос, что такое НОД, но практически вряд ли приемлемо для поиска НОД конкретных чисел. Кроме того сам алгоритм поиска делителей методом перебора натуральных чисел и даже только простых чисел не очень прост /см. разд. I/.

Древнегреческим математиком Эвклидом (3 в. до н.э.) был предложен очень простой и поэтому хороший алгоритм поиска НОД, идея которого основывается на том факте, что если $a \neq b$ имеют НОД, равный c , то и $|a - b|$ будет иметь этот же делитель. Заменяя большее из a и b на $|a - b|$ будем последовательно уменьшать a и b до тех пор, пока значения a и b не совпадут; а НОД для равных чисел, очевидно, равен одному из них.

Идея Эвклида может быть представлена АЛГОРИТМОМ 2.1. Необходимо отметить, что за внешней простотой алгоритма скрыт один и весьма существенный недостаток - не для всех пар целых чисел a и b он будет работать достаточно быстро. Если, например, возьмем $a = 123622$ и $b = 2$, то для получения делителя 2 из a придется вычесть b целых 61810 раз. Поэтому для практических целей лучше использовать АЛГОРИТМ 2.2, где будем не просто вычитать из большего числа меньшее, а вместо большего будем ставить остаток от деления большего числа на меньшее. При этом учтем, что если одно из чисел окажется делителем другого, то получим остаток, равный нулю; а в этом случае, очевидно, алгоритм работать не будет. По этой причине установлена защита от нулевого остатка /строки 70 и 100/ - если значение вспомогательной переменной AI или BI окажутся равными нулю, то НОД исходных данных A и B есть ненулевое значение другой вспомогательной переменной. Последнее утверждение можно понимать и в том смысле, что процесс нахождения НОД заканчивается, когда найден последний, отличный от нуля, остаток от деления в алгоритме Эвклида.

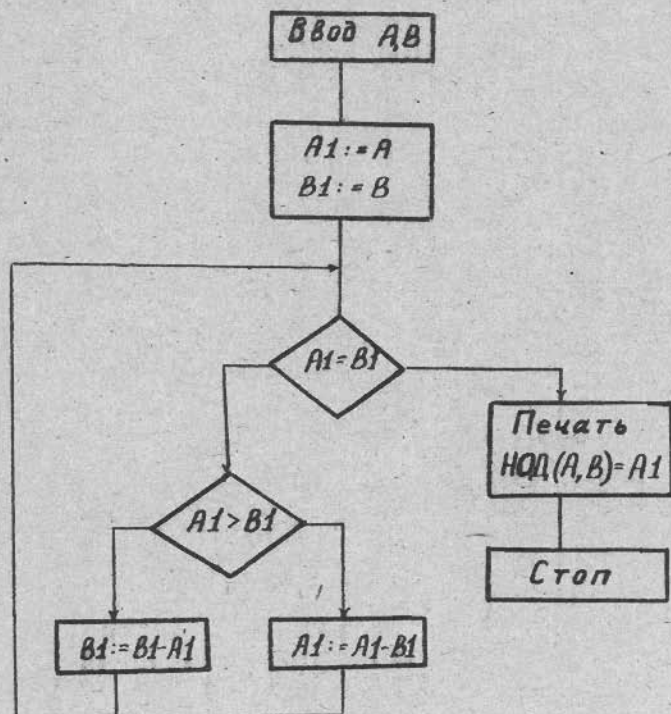
Для нахождения НОД нескольких чисел нет необходимости искать сразу общий делитель для всех заданных чисел. Следует воспользоваться следующим известным результатом.

Теорема 2.1. НОД чисел $a_1, a_2, \dots, a_{n-1}, a_n$ равен НОД числа, равного НОД чисел a_1, a_2, \dots, a_{n-1} и числа a_n .

```

5REM НОД ДВУХ ЧИСЕЛ (МЕТОД ПОСЛЕДОВАТЕЛЬНОГО ВЫЧИТАНИЯ)
10PRINT "ВВЕДИТЕ ДВА НАТУРАЛЬНЫХ ЧИСЛА, НОД КОТОРЫХ ВАС ИНТЕРЕСУЕТ"
20INPUT A,B
30A1=A:B1=B
40IFA1=B1 THEN 70
50IFA1>B1 THEN A1=A1-B1:GOTO 40
60B1=B1-A1:GOTO 40
70PRINT "НОД ("A1","B1")="A1
80END

```

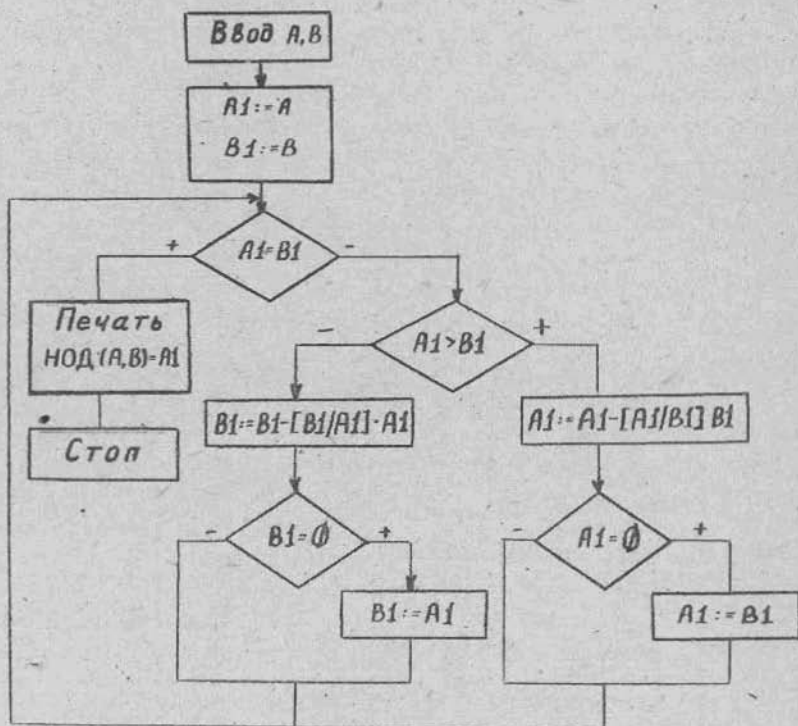


Алгоритм 2.1

```

5 REM НОД ДВУХ ЧИСЕЛ
  (МЕТОД ПОСЛЕДОВАТЕЛЬНОГО ДЕЛЕНИЯ)
10 PRINT "ВВЕДИТЕ ДВА НАТУРАЛЬНЫХ ЧИСЛА,
      НОД КОТОРЫХ ВАС ИНТЕРЕСУЕТ"
20 INPUT A,B
30 A1=A : B1=B
40 IF A1=B1 THEN 120
50 IF A1<B1 THEN 90
60 A1=A1-INT(A1/B1)*B1
70 IF A1=0 THEN A1=B1
80 GO TO 40
90 B1=B1-INT(B1/A1)*A1
100 IF B1=0 THEN B1=A1
110 GO TO 40
120 PRINT "НОД("A1","B1")="A1
130 END

```



Алгоритм 2.2

Таким образом сначала следует найти НОД 1-го и 2-го чисел, затем НОД полученного числа и 3-го числа, и продолжать этот процесс до тех пор, пока не выбраны все заданные числа. АЛГОРИТМ 2.3 для поиска НОД массива чисел реализует эту идею, причем алгоритм поиска НОД двух чисел оформлен в виде подпрограммы, т.е. для рабочего состояния названный алгоритм должен быть дополнен вычислительной частью АЛГОРИТМА 2.2. Конкретнее - АЛГОРИТМ 2.2 передвинут со строк 40-100 на строки 200-270 и дополнен оператором `RETURN`. В качестве рабочих переменных используются B и C .

Определение 2.2. Пусть a_1, a_2, \dots, a_n - различные от нуля целые числа. Наименьшим общим кратным (НОК) этих чисел называется наименьшее положительное число, кратное всем этим числам.

Для того чтобы найти НОК нескольких чисел вручную, необходимо сначала разложить их на простые множители. Можно аналогично решать эту задачу и на ЭВМ, но значительно проще решение, основанное на следующих теоремах.

Теорема 2.2. НОК массива чисел a_1, a_2, \dots, a_n равен НОК числа, равного НОК чисел a_1, a_2, \dots, a_{n-1} и числа a_n .

Теорема 2.3. Наименьшее общее кратное двух чисел равно их произведению, деленному на их наибольший общий делитель.

Таким образом, зная НОД двух чисел, найти их НОК очень просто. Кроме того, на основании теоремы 2.2, поиск НОК нескольких чисел строится точно так же, как и НОД, т.е. с помощью последовательного вычисления НОК пары чисел. АЛГОРИТМ 2.4 построен так же как и АЛГОРИТМ 2.3 с использованием АЛГОРИТМА 2.5.

3. КОНЕЧНЫЕ ЦЕЛЫЕ ДРОБИ

Определение 3.1. Конечной цепной дробью называется число, записанное в виде

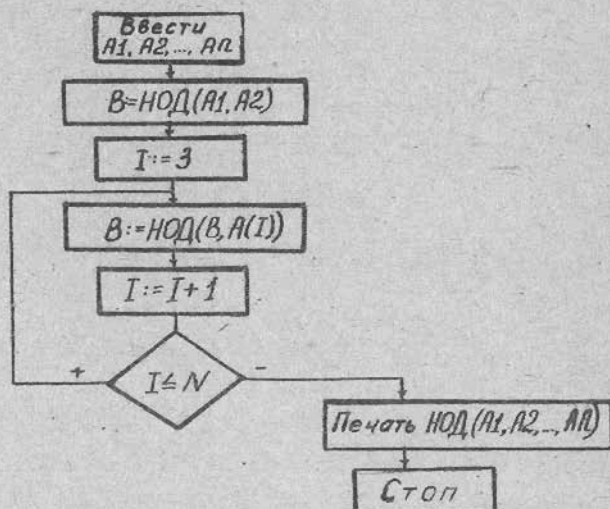
$$a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{a_3 + \dots + \frac{1}{a_{s-1} + \frac{1}{a_s}}}}},$$

где a_0, a_1, \dots, a_s - натуральные числа.

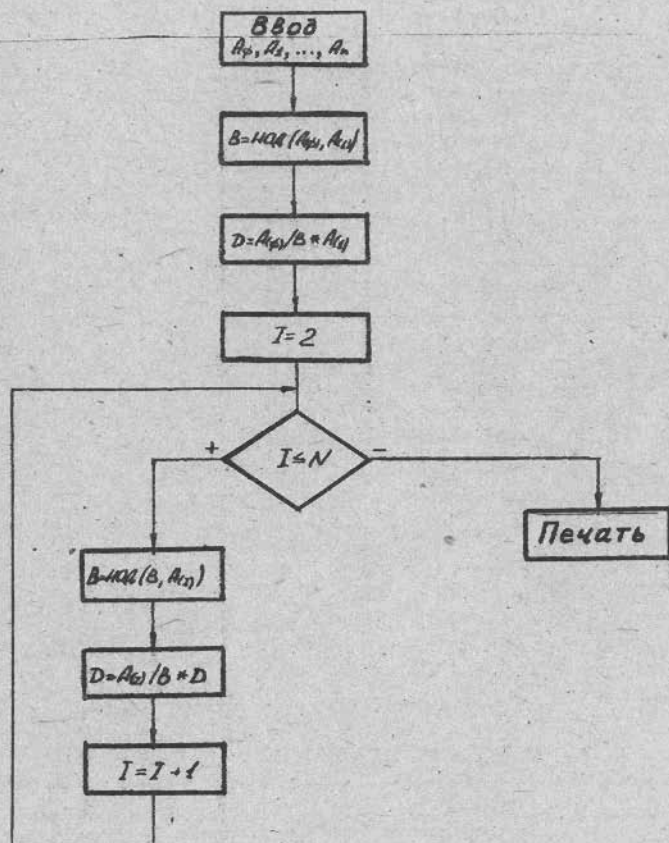
```

ВРЕМЯ ЧИСЛЕНИЯ ОДНЕСКОЛЬКИХ ЧИСЕЛ
10 PRINT "СКОЛЬКО ЧИСЕЛ" : INPUT N : IF N > 100 THEN 10
20 DIM A(99)
30 FOR I = 0 TO N - 1
40 PRINT I + 1 : " - ОЧИСЛО" : INPUT A(I)
50 NEXT I
60 LET B = A(0) : LET C = A(1)
70 GOSUB 200
80 FOR I = 2 TO N - 1
90 LET C = A(I) : GOSUB 200
100 NEXT I
110 PRINT "НОД (" : A(0) :
120 FOR I = 1 TO N - 1
130 PRINT ", " : A(I) :
140 NEXT I
150 PRINT ") = " : B
160 STOP
200 IF C = B THEN RETURN
210 IF C > B THEN 250
220 LET B = B - INT(B/C) * C
230 IF B = 0 THEN LET B = C
240 GOTO 200
250 LET C = C - INT(C/B) * B
260 IF C = 0 THEN LET C = B
270 GOTO 200
280 END

```



Алгоритм 2.3



Алгоритм 2.4а

```

5REM ВЪВЕДЕНИЕ ИМЕНА НЕКОЛЬКИХ ЧИСЕЛ
10PRINT "СКОЛЬКО ЧИСЕЛ" : INPUT N
20IF N > 20 THEN 10
30DIM A(19)
40FOR I = 0 TO N - 1
50PRINT "ВВЕДИТЕ " I + 1 " - Е ЧИСЛО"
60INPUT A(I)
70NEXT I
80LET B = A(0) : LET C = A(1)
90GOSUB 200
100LET B = A(0) / B * A(1) : LET D = B
110FOR I = 2 TO N - 1
120LET C = A(I) : GOSUB 200
130LET B = A(I) / B * D : LET D = B
140NEXT I
150PRINT "НОК = " D
160STOP
200IF C = B THEN RETURN
210IF C > B THEN 250
220B = B - INT(B/C) * C
230IF B = 0 THEN LET B = C
240GOTO 200
250C = C - INT(C/B) * B
260IF C = 0 THEN LET C = B
270GOTO 200

```

* Алгоритм 2.46

```

200IF C = B THEN RETURN
210IF C > B THEN 250
220B = B - INT(B/C) * C
230IF B = 0 THEN LET B = C
240GOTO 200
250C = C - INT(C/B) * B
260IF C = 0 THEN LET C = B
270GOTO 200

```

Алгоритм 2.5

Для удобства и краткости будем записывать цепную дробь в виде (a_0, a_1, \dots, a_s) , т.е. списка коэффициентов. Так как цепная дробь является рациональным числом, она может быть представлена в виде P/Q , где P и Q - натуральные числа. Может быть поставлена и обратная задача.

Теорема 3.1. Любое рациональное число равно некоторой цепной дроби.

Из формы записи дроби вытекает способ ее построения:

$$\begin{aligned} P &= Q a_0 + z_1 \\ 0 &= z_1 a_1 + z_2 \\ z_1 &= z_2 a_2 + z_3 \\ &\vdots \\ z_{s-1} &= z_s a_s \end{aligned}$$

и z_s является последним остатком от деления, отличным от нуля. Из приведенных соотношений следует алгоритм, в основе которого лежит алгоритм деления с остатком /см. разд. I и 2/, а признаком завершения его работы является нахождение последнего, отличного от нуля остатка от деления P/Q . При переходе от одного шага к следующему, делитель занимает место делимого / Q занимает место P , затем z_1 занимает место Q и т.д./.. АЛГОРИТМ 3.1 выполняет эту работу, печатая коэффициенты цепной дроби.

Теперь перейдем к созданию алгоритма для превращения цепной дроби в рациональное число. Свертывание цепной дроби происходит согласно следующим соотношениям:

$$\begin{aligned} R_0 &= 0 \\ R'_0 &= 1 \\ R'_1 &= Q_s R_1 + R_0 \\ R_2 &= Q_{s-1} R'_1 + R_1 \\ R'_2 &= Q_{s-2} R_2 + R'_1 \\ &\vdots \\ R_{s+2} &= Q_0 R_{s+1} + R_s \end{aligned} \quad (3.1)$$

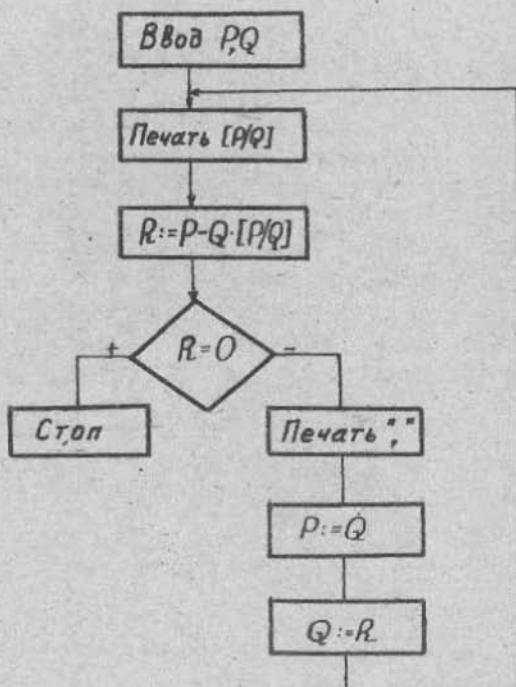
и $P/Q = R_{s+2}/R_{s+1}$.

Обратим внимание на то, что в каждом уравнении участвуют три подряд стоящие остатка R_{i-2} , R_{i-1} , R_i , т.е. для вычисления очередного R_i достаточно знать R_{i-1} и R_{i-2} , что позволяет при составлении алгоритма хранить промежуточные значения трех остатков не в массиве с числом ячеек $S+3$, а в трех именах. В АЛГОРИТМЕ 3.2 остатку R_i соответствует имя $R2$, остатку $R_{i-1} - R1$, а остатку $R_{i-2} - R0$. После вычисления очередного

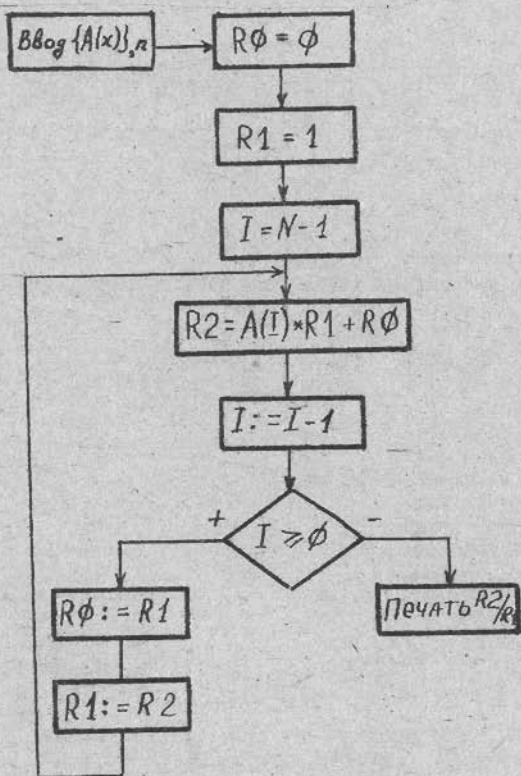
```

10REM ЦЕПНАЯ ДРОБЬ
20INPUTP,Q
30PRINTINT(P/Q)
40LETR=P-Q*INT(P/Q)
50IFR=0THEN100
60PRINT", "
70LETP=Q
80LETQ=R
90GOTO30
100END

```



Алгоритм 3.1



Алгоритм 3.2а

R_2 необходимо "сдвинуть" эту тройку: R_0 присваивается значение R_1 , при этом старое, уже не нужное, значение R_0 затирается, R_1 присваивается значение R_2 , причем имя R_2 освобождается для следующей подстановки, после чего можно увеличивать I на 1 - значения остатков уже соответствуют этому новому I .

На основе уже законченных алгоритмов построения и свертывания цепной дроби достаточно просто решается ряд других интересных задач.

Определение 3.2. n -я подходящей дробью $0 \leq n \leq S$ к конечной цепной дроби (3.1) будем называть величину $A_n = (\alpha_0, \alpha_1, \dots, \alpha_n)$.

Подходящие дроби вычисляются АЛГОРИТМОМ 3.3а; поскольку они - наилучшие рациональные приближения дроби P/Q , то для иллюстрации будем печатать для каждого значения подходящей дроби ее ошибку относительно исходной дроби.

```

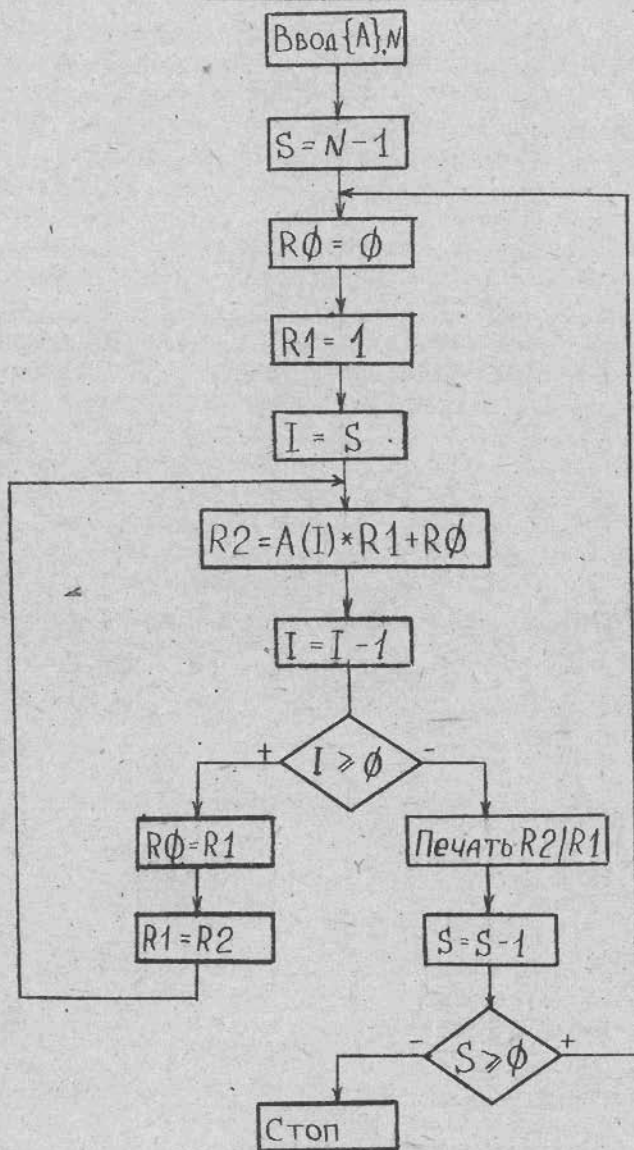
10REM СВЕРКА ЦЕПНОЙ ДРОБИ
20DIM A(40)
30PRINT "ВВЕДИТЕ ЧИСЛО ЭЛЕМЕНТОВ ЦЕПНОЙ ДРОБИ"
40INPUT N
50FOR I=0 TO N-1
60PRINT "ВВЕДИТЕ "; I+1; " - ЭЛЕМЕНТ"
70INPUT A(I)
80NEXT I
90LETR0=0:LETR1=1
100LET I=N-1
110LETR2=A(I)*R1+R0
120LET I=I-1
130IF I=0 THEN 150
140PRINT R2;" / " ; R1; STOP
150LETR0=R1:LETR1=R2
160GO TO 110
170END

```

Алгоритм 3.2б

Если известна цепная дробь, то подходящие дроби можно вычислить АЛГОРИТМОМ 3.3. Его отличает от алгоритма свертывания цепной дроби только дополнительная часть - организация цикла по переменной S , где S - число коэффициентов в очередной подходящей дроби.

В АЛГОРИТМЕ 3.4 строки 40 - 90 организуют построение цепной дроби; строки 110 - 140 и 200 - 210 организуют свертывание подходящей дроби, строки 150 - 190 организуют перебор всех подходящих дробей.



Алгоритм 3.3а

Еще одно приложение цепных дробей – задача поиска дробного представления рационального числа. При этом используем тот факт, что в АЛГОРИТМЕ 3.5 значения P и Q не обязательно должны быть целыми числами. В этом случае несмотря на дробные значения чисел P и Q получаем корректные коэффициенты a_0, a_1, \dots, a_s , которые можно использовать для свертывания цепной и подходящих дробей.

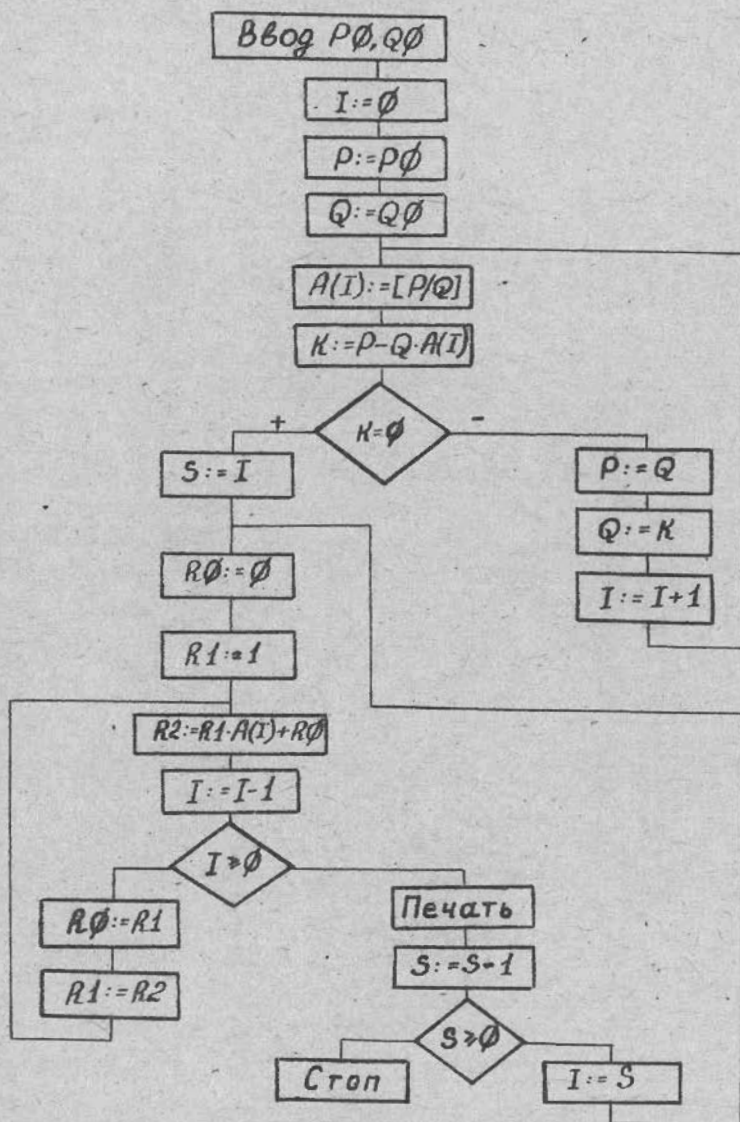
Цепные дроби уже изначально имели важное практическое применение. Начиная с середины семнадцатого столетия известный физик, астроном и математик Христиан Гюйгенс /1629-1695 гг./ попытался построить действующую модель движения небесных тел. При этом сложные соотношения движения планет по небосводу приходилось имитировать поворотами зубчатых колес. Таким образом, скорость движения планет относительно друг друга определялась соотношением числа зубцов на колесах; такие соотношения имели конкретные значения. Следовательно, надо было нарезать столько зубцов, сколь близко можно подойти к известному конкретному значению, не забывая при этом, что число зубцов не может быть слишком большим. При решении такой задачи фактически и возник аппарат цепных дробей, поскольку было доказано, что подходящие дроби – наилучшие приближения к рациональному числу в смысле следующего определения.

```

10REM ПОДХОДЯЩИЕ ДРОБИ
20DIM A(40)
30INPUT N
40FOR I=0 TO N-1
50PRINT "ВВЕДИТЕ " I+1; " ЭЛЕМЕНТ "
60INPUT A(I)
70NEXT I
80LET S=N-1
90LET I=0
100LETR0=0:LETR1=1
110LETR2=A(I)*R1+R0
130LET I=I+1
140IFI=0 THEN 200
150PRINT R2; "/" R1;
160PRINT "СЧЕТОМ " I+1; " ЭЛЕМЕНТОВ "
170LET S=S-1
180IFI=S=0 THEN 90
190STOP
200LETR0=R1:LETR1=R2
210GOTO 110

```

Алгоритм 3.36



Алгоритм 3.4а

Определение 3.3. Рациональная дробь P/Q называется наилучшим приближением к действительному числу α , если не существует ни одной рациональной дроби X/Y со знаменателем, меньшим или равным Q , и которая была бы ближе к α , чем P/Q .

```

10 REM НАИЛУЧШЕЕ ПРИБЛИЖЕНИЕ
20 INPUT P,Q
30 DIM A(255)
40 LET I=0: LET P=P0: LET Q=Q0
50 LET A(I)=INT(P/Q)
60 LET K=P-Q*A(I)
70 IF K=0 THEN 100
80 LET P=Q: LET Q=K
90 LET I=I+1: GOTO 50
100 LET S=I
110 LET R0=0: LET R1=1
120 LET R2=A(I)*R1+R0
130 LET I=I-1
140 IF I=0 THEN 200
150 PRINT R2;" / " R1
160 PRINT "СТОЧНОСТЬ В" TAB(20) ABS(R2/R1-P0/Q0)*100;" %"
170 LET S=S-1
180 IF S=0 THEN LET I=S: GOTO 110
190 STOP
200 LET R0=R1: LET R1=R2
210 GOTO 120

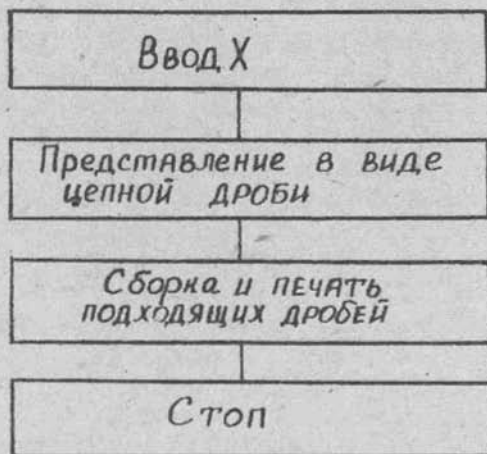
```

Алгоритм 3.46


```

10PRINT "ВВЕДИТЕ ПОЛОЖИТЕЛЬНОЕ РАЦИОНАЛЬНОЕ ЧИСЛО"
20INPUT X
30DIM A(255)
40LET I=0: LET P=X: LET Q=1
50LET A(I)=INT(P/Q)
60LET R=P-Q*INT(P/Q)
70IF R=0 THEN 90
80LET P=Q: LET Q=R: LET I=I+1: GOTO 50
90FOR S=I-1 TO 0 STEP -1
110LET R1=1: LET R0=0
120FOR J=8 TO 0 STEP -1
130LET R2=A(J)*R1+R0
140LET R0=R1: LET R1=R2
150NEXT J
160PRINT S+1; "-КОЭФФИЦИЕНТ" X; "=" R1 "/" R0
170IF X>R1/R0 THEN PRINT "+ "
180PRINT X-R1/R0
190NEXT S
200STOP

```



Алгоритм 3.5

4. СРАВНЕНИЯ

Определение 4.1. Целые числа a и b называются сравнимыми по модулю m , где m - целое положительное число, если их разность $a - b$ делится на m .

Определение 4.2. Полной системой вычетов по некоторому модулю называется система чисел, взятых по одному из каждого класса по этому модулю.

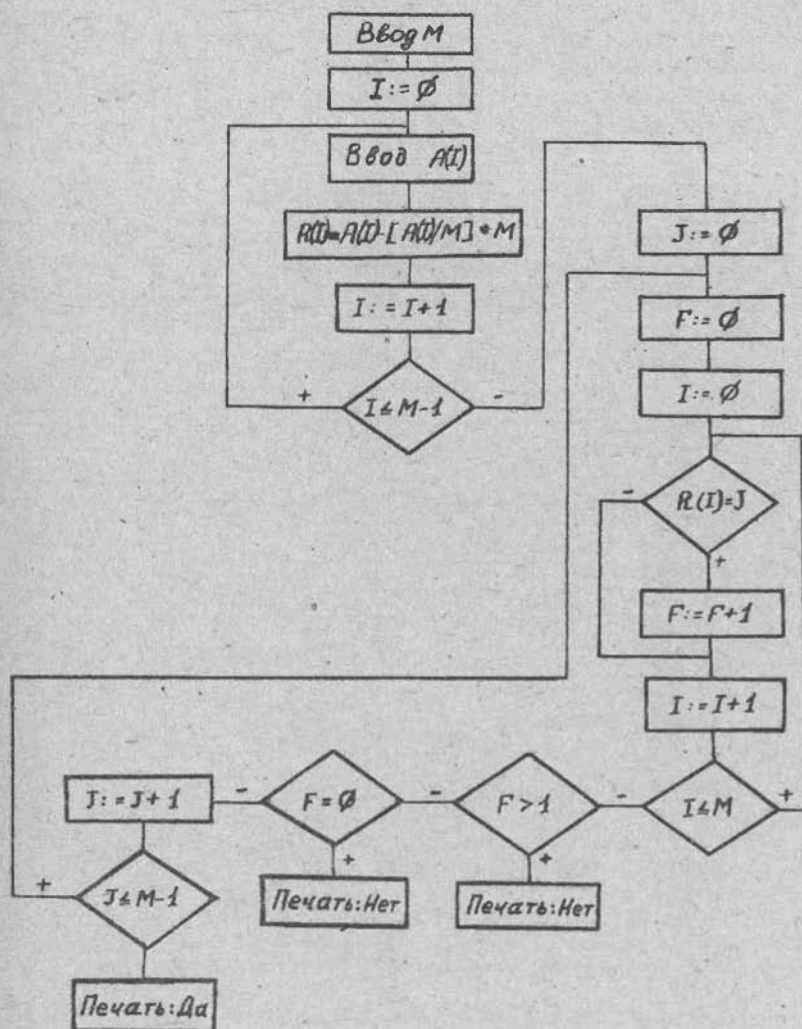
Таким образом, полной системой вычетов является максимально возможное множество чисел, попарно не сравнимых по заданному модулю. Очевидно, все эти числа при делении на модуль дают различные остатки /иначе разность чисел, имеющих одинаковые остатки, делилась бы на модуль нацело, а следовательно, эти числа лежали бы в одном классе/. Остается сделать вывод о том, что число возможных остатков от деления на заданное число m совпадает с самим m . Именно на этом построен АЛГОРИТМ 4.1. Вначале каждому числу ставим в соответствие его остаток от деления на модуль m /строки 40 - 90/, а затем проверяем наличие необходимых остатков. Очевидно, можно сделать вывод, что система чисел не является полной системой вычетов, как только обнаружим, что какой-то остаток присутствует более одного раза, либо его вовсе нет. Число остатков, равных J , в массиве R подсчитываем в имени F /строки 100 - 140/. Если после просмотра массива F не есть 1, то алгоритм выходит на один из остановов с негативным ответом /строки 150, 160/.

Определение 4.3. Приведенной системой вычетов по некоторому модулю называется система чисел, взятых по одному из каждого класса, взаимно простых с модулем.

Для построения приведенной системы вычетов сделан АЛГОРИТМ 4.2. В нем имеется цикл перебора всех остатков /строки 40 - 100/, где I - очередной остаток. Для определения взаимной простоты I и исходного натурального N применен алгоритм Эвклида /строки 50 - 80/, если НОД есть 1 - числа взаимно простые. Так как при работе алгоритма Эвклида могут быть изменены значения переменных, для I и N используются "дублиры" - A и B .

Определение 4.4. Функцией Эйлера $\varphi(m)$ называется число классов по модулю m взаимно простых с этим модулем.

Из последующего определения следует, что разобранный алгоритм кроме приведенной системы вычетов находит и $\varphi(m)$, но, используя следующие теоретические результаты, можно построить алгоритм вычисления функции Эйлера намного эффективнее.



Алгоритм 4.1а

Теорема 4.1. Функция Эйлера мультипликативна, т.е. $\varphi(ab) = \varphi(a)\varphi(b)$, при $\text{НОД}(a, b) = 1$.

Теорема 4.2. Пусть p - простое число, $\alpha \geq 1$ - произвольное натуральное число; тогда $\varphi(p^\alpha) = p^{\alpha-1}(p-1)$.

Таким образом, для нахождения функции Эйлера необходимо построить каноническое представление числа:

$$m = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_n^{\alpha_n},$$

а затем вычислить $\varphi(m) = p_1^{\alpha_1-1}(p_1-1) p_2^{\alpha_2-1}(p_2-1) \dots p_n^{\alpha_n-1}(p_n-1)$

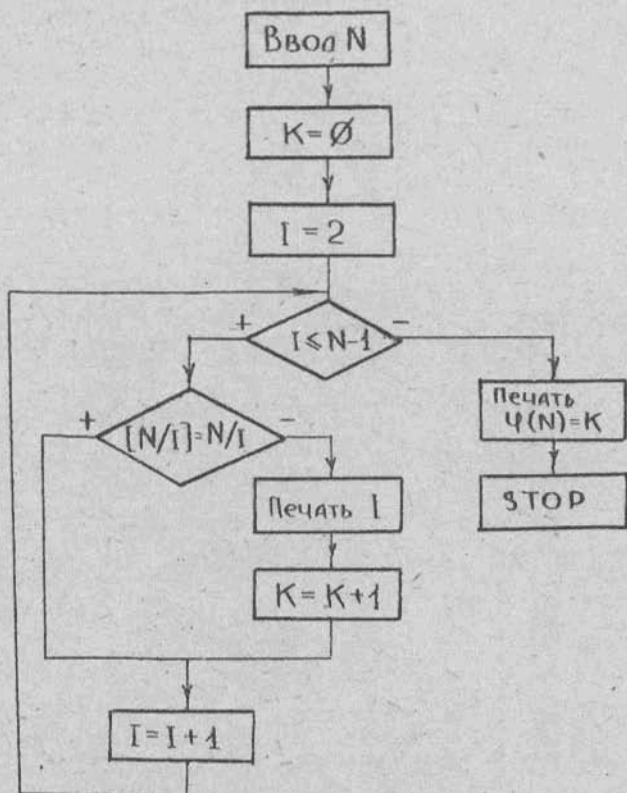
На практике при составлении алгоритма выясняется, что необходимо в каждый момент времени знать только один сомножитель канонического разложения $p_i^{\alpha_i}$, и как только он найден (см. строки 40 - 80), следует $p_i^{\alpha_i}(p_i-1)$ сразу же домножить на F , что реализовано в строке 80. При этом в АЛГОРИТМЕ 4.3 имя Y есть $p_i^{\alpha_i-1}$, а цикл организован так же как при построении канонического разложения.

```

10REM ПОЛНАЯСИСТЕМАВЫЧЕТОВ
20DIM A(255),R(255)
30PRINT"МОДУЛЬСИСТЕМЫ"
40INPUTM
50FORI=0TOM-1
60PRINTI+1"-МИЗЛЕМЕНТСИСТЕМЫ"
70INPUTA(I)
80LETR(I)=A(I)-INT(A(I)/M)*M
90NEXTI
100FORJ=0TOM-1
110LETF=0
120FORI=0TOM-1
130IFR(I)=JTHENLETF=F+1
140NEXTI
150IFF>1THEN200
160IFF=0THEN220
170NEXTJ
180PRINT"СИСТЕМАВЫЧЕТОВПОЛНАЯ"
190STOP
200PRINT"ПРИВЕДЕННИИВЫЧЕТ-"J;"НАЙДЕН"IFI"РАЗ"
210STOP
220PRINT"ПРИВЕДЕННИИВЫЧЕТ-"J;"НЕНАЙДЕН"
230STOP

```

Алгоритм 4.16

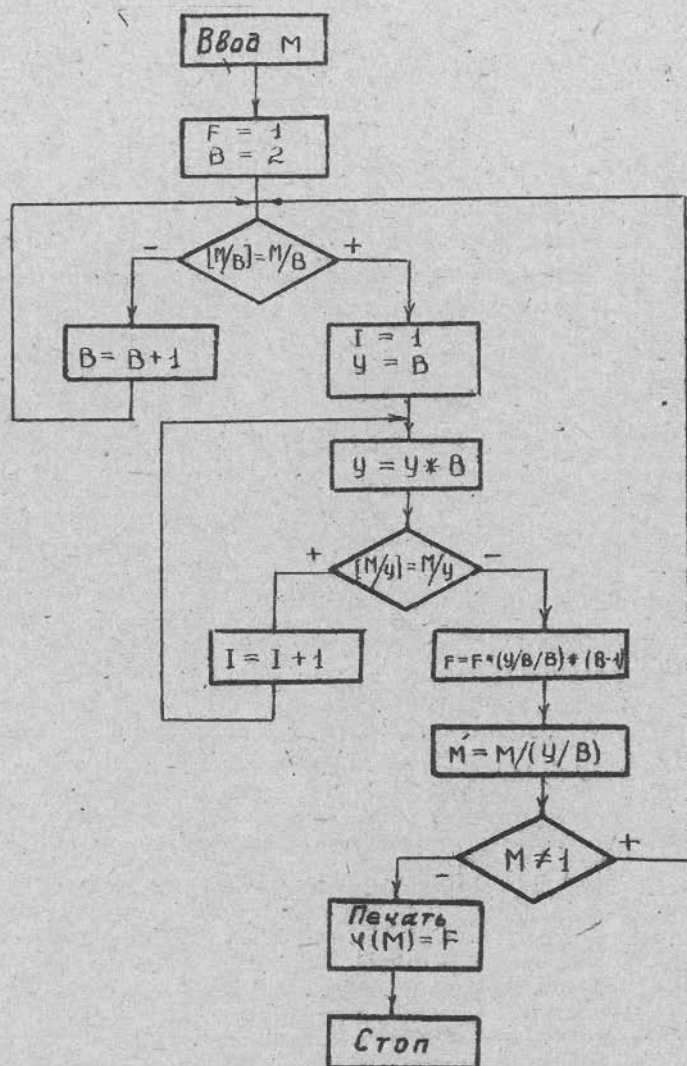


Алгоритм 4.2а

```

5 REM ПРИВЕДЕННАЯ СИСТЕМА ВЪЕЗДОВ
10 PRINT "НАТУРАЛЬНОЕ ЧИСЛО"
20 INPUT N: LET N0 = N
30 LET K = 0
40 FOR I = 1 TO N - 1
50 LET A = N: LET B = I
60 IF A = B THEN GOTO 90
70 IF A > B THEN LET A = A - B: GOTO 60
80 LET B = B - A: GOTO 60
90 IF A = 1 THEN PRINT I, : LET K = K + 1
100 NEXT I: PRINT
110 PRINT "F("N;") = "K
120 STOP
  
```

Алгоритм 4.2б



Алгоритм 4.3а

Функция Эйлера имеет большое значение в теории чисел, а следующие две теоремы Ферми - Эйлера, являются ключевыми в теории сравнений и широко применяются при решении задач.

Теорема 4.3. Для любого простого числа p и любого целого числа $a \geq 1$, не делящегося на p , справедливо сравнение

$$a^{p-1} \equiv 1 \pmod{p}.$$

Теорема 4.4. Для любого модуля m и любого целого числа $a \geq 1$, взаимно простого с m , справедливо сравнение

$$a^{\varphi(m)} \equiv 1 \pmod{m}.$$

Используя последние результаты, решим задачу о нахождении остатка от деления X^N на M .

```
5REM ФУНКЦИЯ ЭЙЛЕРА
10PRINT "НАТУРАЛЬНОЕ ЧИСЛО:"
20INPUT M: LET M=M0
30LET F=1: LET B=2
40IF INT(M/B) < M/B THEN LET B=B+1: GOTO 40
50LET Y=B
60LET Y=Y*B
70IF INT(M/Y)=M/Y THEN GOTO 60
80LET Y=Y/B
90LET F=F*(Y/B)*(B-1): LET M=M/Y
100IF M<>1 THEN 40
110PRINT "F("M0;")="F
120STOP
```

Алгоритм 4.36

Очевидно, если N - достаточно большое число, по сути единственным способом решения поставленной задачи будет использование функции Эйлера - $(X^{\varphi(M)})^K \equiv 1 \pmod{M}$, т.е. степень можно заменить на остаток от деления N на $\varphi(M)$. Но так поступать имеем право, когда числа X и M взаимно просты; иначе приходится сокращать M до тех пор пока $\text{НОД}(M, X)$ не станет равным 1 (см. строки 30-60 АЛГОРИТМА 4.4. При этом переменная $X1 - \text{НОД}(M, X)$, в $X0$ "накапливаются" частные от деления X . Как только $X0$ станет больше M заменяем $X0$ на остаток от деления $X0$ на M /строка 90/. Если в процессе сокращения M на НОД /а при каждом сокращении число N уменьшается на единицу, т.е. сомножитель X

исчезает, а его частное не теряется, а домножается на X^0 / число N стало равным нулю, можно считать, что остаток уже фактически найден. Добившись сокращения M , имеем право вычислить функцию Эйлера /строки 170-290/ и заменить число N остатком от деления. Для получения окончательного результата находим K такое, что $K = N - \alpha \varphi(M)$, и заменяем X^N на X^K , где $K < \varphi(M)$ и α - целое число.

```

5 REM ОСТАТОК ОТ ДЕЛЕНИЯ X В СТЕПЕНИ N
10 PRINT "ВВЕДИТЕ X, N, M": INPUT X, N, M
20 LET M1=M
30 LET X1=X: LET X2=M
40 GOSUB 500
50 LET X0=1: LET X3=1
60 IF X1=1 THEN 170
70 LET X0=X0*X/X1: LET X3=X3*X1
80 LET M=M/X1
90 IF X0>M THEN LET X0=X0-M*INT(X0/M)
100 LET N=N-1
110 IF M<>1 THEN 130
120 PRINT "ОСТАТОК РАВЕН НУЛЮ" : GOTO 10
130 IF N=0 THEN 460
140 LET X1=X1: LET X2=M
150 GOSUB 500
160 GOTO 60
170 LET F=1: LET B=2: LET M1=M
180 IF INT(M/B)=M/B THEN 230
190 LET B=B+1
200 IF SQR(M)+1>B THEN 180
210 LET B=M: LET Y=M
220 GOTO 270
230 LET Y=B
240 IF INT(M/(Y*B))<>M/(Y*B) THEN 270
250 LET Y=Y*B
260 GOTO 240
270 LET F=F*Y/B*(B-1)
280 LET M=M/Y: LET B=B+1
290 IF M>1 THEN 180
300 LET X=X-M1*INT(X/M1)
310 LET N=N-F*INT(N/F): LET M=M1
320 LET I=0: LET B=1
330 IF M<B THEN 370
340 IF I>N THEN 450
350 LET B=B*X: LET I=I+1
360 GOTO 330
370 FOR I=1 TO N-I*INT(N/I)
380 LET X0=X0*X
390 IF X0>M THEN LET X0=X0-M*INT(X0/M)

```

```

400NEXT I1
410LETX=B-INT(B/M)*M
420LETN=INT(N/I)
430GOTO320
450LETX0=X0*B
460IFX0>MTHENLETX0=X0-M*INT(X0/M)
470PRINT"OCTATOK-"*X0*X3
480GOTO10
500IFX1=X2THENRETURN
510IFX1>X2THEN560
520LETX2=X2-X1*INT(X2/X1)
540IFX2=0THENRETURN
550GOTO510
560LETX1=X1-X2*INT(X1/X2)
570IFX1=0THENLETX1=X2*RETURN
580GOTO510

```

Алгоритм 4.4 (Окончание)

Теперь следует обратить внимание на конечный результат. К примеру, для вычисления остатка от деления 6^4 на 10, сокращаем $M = 10$ до $M = 5$ и ищем остаток $6^3 \cdot 3$ на 5. Но получаемый остаток — это еще не ответ, поскольку частные от деления 6^4 на 10 и $6^3 \cdot 3$ на 5 равны, а остатки — нет. Поэтому следует скорректировать остаток от деления $6^3 \cdot 3$ на 5 умножением его на 2, т.е. на тот НОД, на который было сокращено число M . Этот сомножитель накапливаем в переменной $X3$ /строка 70/ и используем при выдаче ответа /строка 470/.

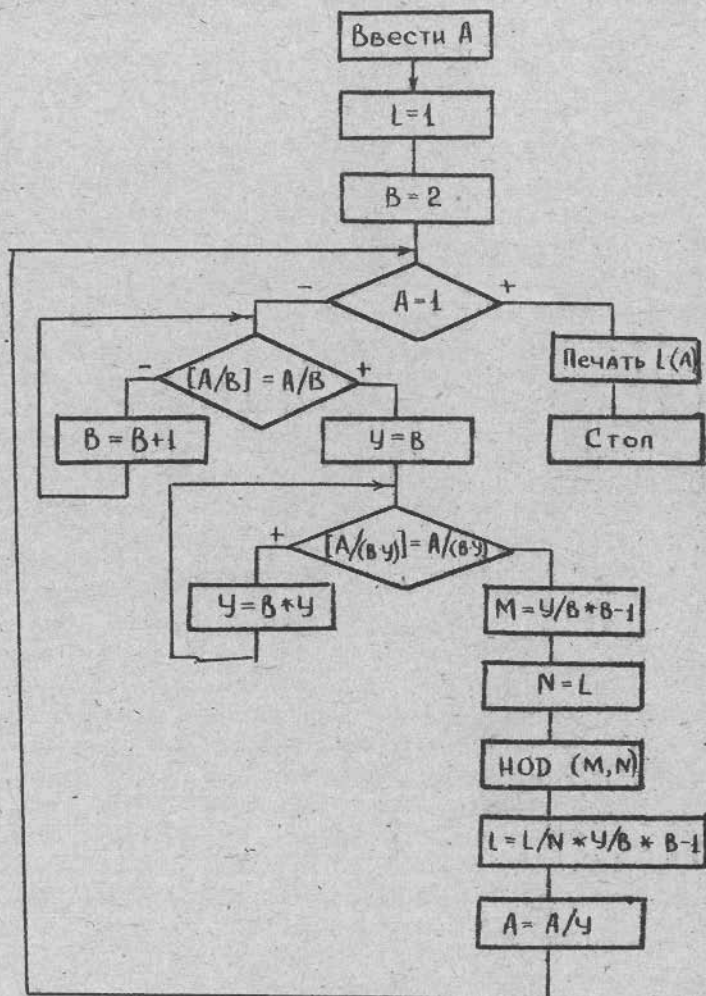
Кроме вычисления функции Эйлера в программе организован еще один цикл /строки 320–430/, время прохождения которого зависит от того, сколько сомножителей X осталось для вычисления остатка. Обратим внимание на то, что время вычисления функции Эйлера от N не зависит, поэтому время работы программы следует оценить следующим образом: при небольших значениях N эффективность программы незначительна, с ростом N /до сотен тысяч/ будем иметь ощутимый выигрыш во времени, поскольку это время не зависит от исходного значения N . Выигрыш будет тем весомее, чем меньше остаток от деления N на $\varphi(M)$, поэтому было бы желательно иметь аналог $\varphi(M)$, меньший $\varphi(M)$. И такой аналог есть. Наименьшим положительным значением K , таким, что $a^K \equiv 1(M)$ является обобщенная функция Эйлера $L(M)$.

Определение 4.5. Обобщенной функцией Эйлера $L(M)$ называется функция, определяемая для всех натуральных значений M следующим образом: $L(1) = 1$, а при $M > 1$

$$L(M) = [P_1^{\alpha_1 - 1} (P_1 - 1) \dots P_s^{\alpha_s - 1} (P_s - 1)],$$

где компоненты наименьшего общего кратного - комбинация компонент канонического разложения $M = p_1^{\alpha_1} \dots p_s^{\alpha_s}$.

Обобщенная функция Эйлера вычисляется АЛГОРИТМОМ 4.5, который отличается от алгоритма вычисления функции Эйлера тем, что вычисляем НОК вместо произведения.



Алгоритм 4.5а

В качестве тренировки составим следующий алгоритм.

Упражнение. Составить алгоритм вычисления остатка X^N на M с использованием обобщенной функции Эйлера.

```

10 REM ОБОБЩЕННАЯ ФУНКЦИЯ ЭЙЛЕРА
20 PRINT "ВВЕДИТЕ НАТУРАЛЬНОЕ ЧИСЛО"
30 INPUT A: LET A=A0
40 LET L0=1: LET B=2
50 IF A=1 THEN 210
60 IF INT(A/B)=A/B THEN 80
70 LET B=B+1: GOTO 60
80 LET Y=B
90 IF INT(A/(Y*B)) < A/(Y*B) THEN 120
100 LET Y=Y*B
110 GOTO 90
120 LET M=Y/B*(B-1): M0=M
130 LET L=L0
140 IF M=L THEN 180
150 IF M>L THEN 170
160 LET L=L-M: GOTO 140
170 LET M=M-L: GOTO 140
180 LET L0=L0*M0/L
190 LET A=A/Y
200 GOTO 50
210 PRINT "L ("A0")="L0
220 STOP
    
```

Алгоритм 4.5б

Пусть дан многочлен $f(x) = C_0 x^n + C_1 x^{n-1} + \dots + C_n$. Рассмотрим сравнение $f(x) \equiv O(M)$, которое будем называть сравнением с неизвестной величиной X . Поставим задачу о нахождении множества всех значений X /если они существуют/, удовлетворяющих последнему сравнению.

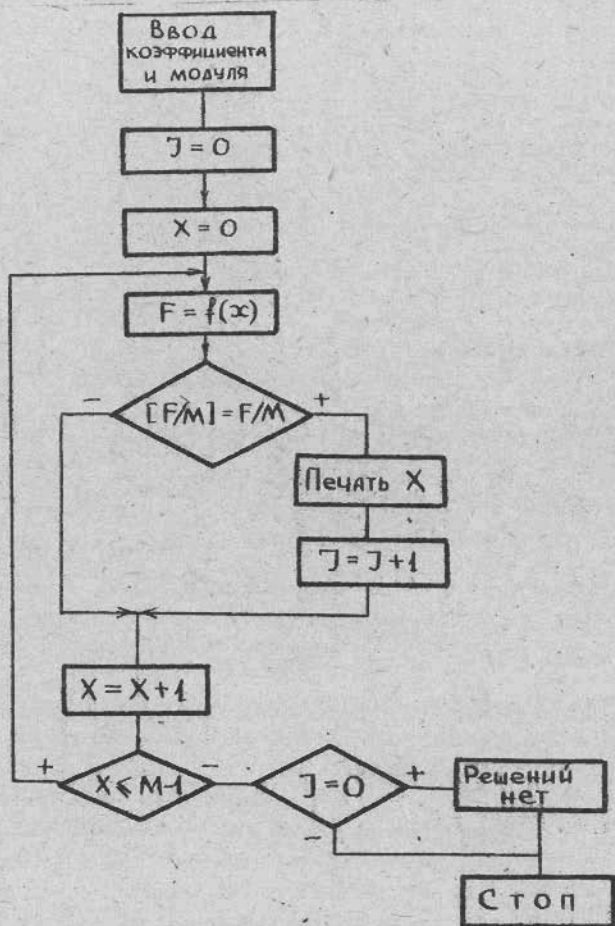
Теорема 4.5. Если некоторое целое число a удовлетворяет сравнению $f(x) \equiv O(M)$, то и весь класс \bar{a} по модулю M состоит из чисел, удовлетворяющих этому сравнению.

Последняя теорема определяет и способ получения всех решений и вид решений. Так, для решения сравнения достаточно перебрать все наименьшие вычеты по данному модулю, что и реализовано в АЛГОРИТМАХ 4.6а и 4.6б.

Более общей является задача решения системы сравнений:

$$\begin{cases} f_1(x) \equiv O(M_1) \\ f_2(x) \equiv O(M_2) \\ \vdots \\ f_s(x) \equiv O(M_s) \end{cases}$$

где $f_1(x)$, $f_2(x)$, ..., $f_s(x)$ — многочлены с целыми коэффициентами.



Алгоритм 4.6а


```

5 REMPEШЕНИЕ СРАВНЕНИЯ
10 PRINT "ВВЕДИТЕ СТЕПЕНЬ МНОГОЧЛЕНА И МОДУЛЬ"
20 INPUT N, M: DIM A(255)
30 FOR I = N TO 0 STEP -1
40 PRINT "X^"; I; " = "; INPUT A(I)
50 NEXT I: LET J = 0
60 FOR K = 0 TO M - 1
70 LET F = 0
80 FOR I = N TO 0 STEP -1
90 LET F = F * X + A(I)
100 NEXT I
110 IF INT(F/M) < F/M THEN 130
115 PRINT J+1; " - РЕШЕНИЕ "; F: LET J = J+1
120 PRINT X; " (MOD "; M; ") = "
130 NEXT K
140 IF J = 0 THEN PRINT "РЕШЕНИЙ НЕТ"
150 STOP

```

Алгоритм 4.66

Отметим, что вместе с каждым числом a , удовлетворяющим данной системе, решением будет любое число класса \bar{a} по модулю $M_1 = \text{НОК}[M_1, M_2, \dots, M_s]$. АЛГОРИТМ 4.7 решает данную задачу. При этом ввод коэффициентов многочленов и модулей осуществляется в строках 10-120. В строках 130-210 определяется НОК $[M_1, M_2, \dots, M_s]$, и наконец, в строках 220-330 вычисляются значения многочленов и проходят проверку все классы по модулю M_1 .

Еще более общий случай - система сравнений с несколькими неизвестными:

$$\begin{aligned}
 f_1(x_1, \dots, x_s) &\equiv 0(M_1); \\
 &\vdots \\
 f_s(x_1, \dots, x_s) &\equiv 0(M_s).
 \end{aligned}
 \tag{4.1}$$

Если все неизвестные x_1, \dots, x_s присутствуют в каждом уравнении, то можно утверждать, что решениями системы сравнений будут комплексы классов $(\bar{a}_1, \bar{a}_2, \dots, \bar{a}_s)$ по модулю $M = \text{НОК}[M_1, \dots, M_s]$. Действительно, если найдем набор чисел (a_1, a_2, \dots, a_s) таких, чтобы они удовлетворяли последней системе, то, зафиксировав в качестве (x_1, \dots, x_s) частные решения (a_1, a_2, \dots, a_s) , можем использовать рассуждения, проведенные для системы сравнений с одним неизвестным, т.е. решением системы сравнений окажутся все числа класса \bar{a}_1 по модулю $M = \text{НОК}[M_1, M_2, \dots, M_s]$. Рассмотрев аналогичным образом все неизвестные, приходим к следующему результату: решением последней системы является комплекс классов $(\bar{a}_1, \bar{a}_2, \dots, \bar{a}_s)$ по модулю $M = \text{НОК}[M_1, M_2, \dots, M_s]$. Если

же в некотором из сравнений J все коэффициенты при неизвестной x_i окажутся нулевыми, то, на наш взгляд, в последний результат следует внести поправки /хотя это и необязательно/.

Зафиксируем все неизвестные, кроме x_i и подставим найденный набор чисел $(a_1, a_2, \dots, a_{i-1}, a_{i+1}, \dots, a_s)$ в J -е сравнение. Получим сравнение, не содержащее неизвестных. Следовательно, решением для неизвестной x_i будет класс \bar{a}_i , но уже по модулю $M_i = \text{НОК}[M_1, \dots, M_{j-1}, M_{j+1}, \dots, M_s]$.

```

10PRINT"ВВЕДИТЕ КОЛИЧЕСТВО СРАВНЕНИЙ"
20INPUTS
30DIM A(40,40),M(40),N(40)
40FOR I=1 TO S
50PRINT"ВВЕДИТЕ СТЕПЕНЬ"II;"-ГО МНОГОЧЛЕНА"
60INPUT N(I-1)
70FOR J=N(I-1) TO STEP-1
80PRINT"X^"IJ;"="I:INPUT A(I-1,J)
90NEXT J
100PRINT"ВВЕДИТЕ N("II;"-)"
110INPUT M(I-1)
120NEXT I
130LET A=M(0):LET M1=M(0)
140FOR I=1 TO S-1
150LET B=M(I)
160IFA=B THEN 200
170IFA>B THEN 190
180LET B=B-A:GOTO 160
190LET A=B:GOTO 160
200LET M1=M1/A*M(I)
210NEXT I:LET K=1
220FOR X=0 TO M1-1
230FOR I=0 TO S-1
240LET F=0
250FOR J=0 TO N(I)
260LET F=F*X+A(I,J)
270NEXT J
280IF INT(F/M1)<>F/M1 THEN 320
290NEXT I
300PRINT"X("IKI;" )="IXI;" (MOD"IM1I;" )"
310LET K=K+1
320NEXT X
330IF K=1 THEN PRINT"РЕШЕНИЙ НЕТ"
340STOP

```

Алгоритм 4.7

Определение 4.6. Решением системы /4.1/ называется комплекс классов $(\bar{a}_1, \dots, \bar{a}_J)$ по соответствующим модулям M_1, \dots, M_J , где $\bar{a}_i = \text{НОК}[M_1, \dots, M_{j-1}, M_{j+1}, \dots, M_J]$, где j - номер сравнения с нулевыми коэффициентами при неизвестной x_i .

Таким образом, отличие алгоритма решения системы сравнений с несколькими неизвестными от предыдущих алгоритмов принципиально проявляется в двух моментах:

1) необходимо вычислять не $M = \text{НОК}[M_1, \dots, M_J]$, а модуль для каждого неизвестного;

2) для нахождения решения системы перебирать не классы одного неизвестного, а все комбинации классов неизвестного.

При написании программы на Бэйсике возникает еще одна проблема, как описать в программе, как разместить в памяти машины систему сравнений с несколькими неизвестными? Бэйсик допускает использование только двумерных массивов, а этого недостаточно для размещения системы сравнений естественным образом. Предлагается следующая вынужденная компоновка системы, представляющая особый интерес:

каждый многочлен содержит не более 16 слагаемых;

$A(0, i)$ - коэффициент при i -м слагаемом;

$A(1, i)$ - степень x_1 при i -м слагаемом;

$A(2, i)$ - степень x_2 при i -м слагаемом;

$A(3, i)$ - степень x_3 при i -м слагаемом;

$A(4, i)$ - степень x_4 при i -м слагаемом;

система содержит до восьми сравнений, причем слагаемые J -многочлена размещены при $16 \cdot (J - 1) \leq i \leq 16J - 1$.

Для неизвестных используем массив $X(3)$, для модулей каждого неизвестного - массив $M(3)$. Для хранения признака присутствия неизвестного x_i в сравнении $J - Z_{ij}$ - массив $R(3, 7)$. Причем признаком присутствия неизвестного в массиве будем считать $Z_{ij} = 1$, а признаком отсутствия - $Z_{ij} = 0$. АЛГОРИТМ 4.8 реализует решение настоящей задачи. В строках 10 и 20 вводится число сравнений S и число неизвестных T системы. Поскольку в массивах языка Бэйсик минимальный индекс нулевой, уменьшаем S и T на 1. В результате S - индекс последнего сравнения в массиве, а T - индекс последнего неизвестного. В строке 30 описаны массивы программы - $A(4, 127)$ - для многочленов сравнений, $N(7)$ - для хранения числа слагаемых многочленов, $M(7)$ - для модулей сравнений. В строках 50-60 вводится число слагаемых очередного сравнения, причем, как и в случае с S и T , это число $N(J)$ уменьшается на 1. В массиве $R(3, 7)$

перед вводом сравнения столбец, соответствующий вводимому сравнению, обнуляется /строки 70-90/, что позволяет при вводе ненулевой степени неизвестного сразу же устанавливать признак присутствия неизвестного /строка 160/. В строках 100-180 вводится многочлен очередного сравнения в соответствии с уже описанной компоновкой; в строках 190-200 - модуль очередного сравнения, а в целом строки 10-210 организуют ввод решаемой системы сравнений с несколькими неизвестными.

```

SРЕМЕШЕНИЕСИСТЕМЫСРАВНЕНИИСНЕСКОЛЬКИМИНЕИЗВЕСТНЫМИ
10PRINT"СКОЛЬКОСРАВНЕНИЙ":INPUTS:LET S=S-1
20PRINT"СКОЛЬКО НЕИЗВЕСТНЫХ":INPUTT:LETT=T-1
30DIMA(4,127),X(3),M(3),R(3,7),N(7),M0(7)
40FORJ=0TOS
50PRINT"СКОЛЬКО СЛАГАЕМЫХ":J+1"СРАВНЕНИЙ"
60INPUTN(J):LETN(J)=N(J)-1
70FORI=0TOT
80LETR(I,J)=0
90NEXTI
100FORK=0TON(J)
110PRINT"ВВЕДИТЕ"K+1"КОЭФФИЦИЕНТ"
120INPUTA(0,16*J+K)
130FORI=1TOT+1
140PRINT"СТЕПЕНЬ X":I
150INPUTA(I,16*J+K)
160IFA(I,16*J+K)(<)0THENLETR(I-1,J)=1
170NEXTI
180NEXTK
190PRINT"МОДУЛЬ X":J+1"СРАВНЕНИЯ"
200INPUTM0(J)
210NEXTJ
220FORI=0TOT
230LETC=1:LETM(I)=1
240FORJ=0TOS
250IFR(I,J)=0THEN330
260LETB=M0(J)
270IFC=BTHEN310
280IFC>BTHEN300
290LETB=B-C:GOTO270
300LETC=C-B:GOTO270
310M(I)=M0(J)/C*M(I)
320C=M(I)
330NEXTJ
340NEXTI
350FORI=0TOT
360LETX(I)=0
370NEXTI:X0=1

```

Алгоритм 4.8

```

380FORJ=0TOS
390B=0
400FORK=0TON(J)
410LETC=A(0,16*J+K)
420FORI=0TOT
430FORL=1TOA(I+1,16*J+K)
440LETC=C*X(I)
450NEXTL
460NEXTI
470LETB=B+C
480NEXTK
490IFINT(B/M0(J))(>)B/M0(J)THEN550
500NEXTJ
510PRINTX0;"-РЕШЕНИЕ"
520FORI=0TOT
530PRINT"X";I+1;"=";"X(I)";"(MOD;"M(I)";")
540NEXTI:X0=X0+1
550I=T
560IFX(I)>M(I)-1THEN580
570X(I)=X(I)+1:GOTO380
580X(I)=0:I=I-1
590IFI>0THEN560
600IFX0=1THENPRINT"РЕШЕНИЙНЕТ"
610STOP

```

Алгоритм 4.8 (Окончание)

Следующий этап задачи предполагает вычисление модулей каждого неизвестного, так как не в каждом сравнении обязательно присутствуют все неизвестные. Этот этап алгоритма выполняется в строках 220-340. В организованном крайними строками 220 и 340 цикле по переменной I перебираются все неизвестные $X(I)$. Необходимо найти наименьшее общее кратное тех модулей сравнений $M0(J)$, в которых хотя бы раз встречалась переменная $X(I)$, т.е. задача сводится к нахождению НОК массива чисел. Но поскольку заранее неизвестно в каком сравнении у нас будет присутствовать $X(I)$, в качестве первого значения НОК массива берем не НОК двух чисел, а единицу /строка 230/. Далее ищем сравнение, в котором присутствует $X(I)$ /строка 250/. Если такое сравнение обнаружено, вычисляем очередное НОК:

во-первых, вычисляем НОД, используя при этом рабочие переменные C и B ;

во-вторых, вычисляем НОК по формуле в строке 310.

Поскольку в имени C должна храниться рабочая копия вычисленного НОК, необходимо перед продолжением вычисления НОК массива восстановить соответствие C и $M(I)$, что и сделано в строке 320.

Далее следует собственно решение системы сравнений. Поскольку необходимо перебрать все возможные сочетания классов переменных, то не избежать очень громоздкой конструкции, обеспечивающей соответствие числа неизвестных числу циклов. В указанном случае система была ограничена четырьмя неизвестными и поэтому была возможность организовать соответствующую конструкцию. Однако при решении системы с числом неизвестных, большим числа четыре, необходимо было бы менять не только описание массивов, но и текст самой программы, т.е. исчезло бы свойство массовости данного алгоритма.

Для того чтобы избежать такого результата, используем следующий прием – организуем перебор всех возможных комбинаций классов, неизвестных в одном цикле. Так, для хранения текущих классов неизвестных используется массив X , начальное положение обеспечивается в строках 350-370, где всем классам присваивается значение нуля. Одновременно устанавливается $X0 = I$ – номер очередного искомого решения. Следующее сочетание классов формируется в строках 550-590, причем изменение комплекса производится, начиная с последнего, т.е. с $X(I)$ -го неизвестного. Если рассматриваемое неизвестное $X(I)$ не достигло своего максимального класса $M(I)-1$, увеличиваем $X(I)$ на 1 и возвращаемся на подстановку решения /строка 570/. Если неизвестное достигло максимального класса, переходим к изменению другого неизвестного; при этом $X(I)$ должно быть установлено в нуль – формируется как бы единица следующего разряда /строка 580/. Признаком завершения перебора всех комплексов является тот факт, что номер неизвестного, которое требуется увеличить, выйдет за пределы индексов неизвестных, т.е. $I < 0$. Проверка этого факта реализуется в строке 590.

В строках 380-500 осуществляется подстановка очередного предполагаемого решения:

в соответствии с компоновкой вычисляются многочлены; полученные значения проверяются делением на соответствующий модуль и в случае, если результат деления не целое число /строка 490/, переходим на формирование следующего комплекса, в противном случае имеем очередное решение системы, которое печатается /строки 510-540/. При этом номер искомого решения $X0$ увеличивается на единицу. Затем происходит переход к формированию следующего комплекса.

Следует отметить, что если и перед завершением работы программы $X0$ есть единица, то система не имеет решений, о чем и сообщает строка 600.

Данный алгоритм имеет существенный недостаток: эффективность программы, т.е. время счета, очень сильно зависит не столько от самих модулей системы, сколько от значения НОК этих модулей. Например, если у нас три сравнения, объединенные в систему, имеют модули 3, 5, 7 и в каждом сравнении присутствуют по три переменные, то модуль решения - число $3 \cdot 5 \cdot 7 = 105$. Следовательно, должно быть проверено на решение $105 \cdot 105 \cdot 105$ комплексов, что недопустимо много. Такой факт стал возможным, поскольку не проанализировали, которому из трех сравнений не удовлетворяет данный комплекс.

Значительно проще и быстрее найти решение каждого сравнения, а затем - их пересечение. Следующий АЛГОРИТМ 4.9 реализует эту идею.

Сравнения вводятся так же, как и в АЛГОРИТМЕ 4.8: первоначальным сочетаниям классов неизвестных присваиваются нулевые значения и последовательно выполняется подстановка неизвестных в сравнения. Но вот уже выбор следующего комплекса зависит от того, на каком этапе был отвергнут предыдущий комплекс.

Проверка начинается с подстановки предполагаемого решения в I-е сравнение. Если сочетание классов по модулю I-го сравнения не удовлетворяет ему, то I-е сравнение не удовлетворяет и всей системе. При этом следующий комплекс ищется опять среди классов по модулю I-го сравнения. Если очередное возможное решение удовлетворяет I-му сравнению, но не удовлетворяет 2-му, возникает вопрос, какое сочетание элементов выбрать в качестве следующего комплекса?

Пусть $(\bar{x}_1, \bar{x}_2, \dots, \bar{x}_r)$ - совокупность классов по модулю M_1 I-го сравнения, а M_2 - модуль 2-го сравнения. Решение системы данных двух сравнений нужно искать среди классов по модулю $M_{12} = \text{НОК}[M_1, M_2]$, а решению $(\bar{x}_1, \bar{x}_2, \dots, \bar{x}_r)$ по модулю M_1 соответствует одно или несколько решений вида $(\bar{x}_1 + K_1 M_1, \bar{x}_2 + K_2 M_1, \dots, \bar{x}_r + K_r M_1)$ по модулю M_{12} . При этом K_1, K_2, \dots, K_r - целые числа, а все найденные решения будут решениями I-го сравнения по модулю M_1 . При переходе от найденного решения для двух сравнений $(\bar{x}_1, \bar{x}_2, \dots, \bar{x}_r)$ по модулю M_{12} к решению для трех сравнений происходит аналогичное преобразование. Решение будет ищется по модулю $M_{13} = \text{НОК}[M_1, M_2, M_3]$, а соответствующие классы могут быть получены увеличением неизвестных на число, кратное M_{12} , что, очевидно, сохранит их как решения первых двух сравнений. Аналогично просматриваются остальные сравнения и формируется окончательный результат. Единственное дополнение, о котором следует напомнить, уже описывалось для АЛГОРИТМА 4.8, поскольку не все сравнения обязательно содержат все неизвестные. Введем описан-

ный порядок изменения модулей системы сравнений для каждого сравнения и для каждого неизвестного в отдельности. Кроме того шаг изменения неизвестного при подстановке на проверку по изложенной причине может быть разным, и поэтому необходимо вместе с модулем неизвестного хранить и шаг изменения неизвестного.

```

5$РЕШЕНИЕСИСТЕМЫСРАВНЕНИЙСЧЕСКОЛЬКИМИНЕИЗВЕСТНЫМИ
10PRINT"СКОЛЬКОСРАВНЕНИЙ":INPUTS:LETS=S-1
20PRINT"СКОЛЬКОНЕИЗВЕСТНЫХ":INPUTT:LETT=T-1
30DIM A(4,127),X(3,7),H(3,7),M1(3),R(3,7),N(7)
35DIM M(7),P(7),LX(7)
40FORJ=0TOS
50PRINT"СКОЛЬКОСЛАГАЕМЫХ":J+1;"СРАВНЕНИЙ"
60INPUTN(J):N(J)=N(J)-1
70FORI=0TOT
80R(I,J)=0
90NEXTI
100FORK=0TON(J)
110PRINT"ВВЕДИТЕ":K+1;"КОЭФФИЦИЕНТ"
120INPUTA(0,16*J+K)
130FORI=1TOT+1
140PRINT"СТЕПЕНЬX":I
150INPUTA(I,16*J+K)
160IFA(I,16*J+K)(>0)THENR(I-1,J)=1
170NEXTI
180NEXTK
190PRINT"МОДУЛЬ":J+1;"СРАВНЕНИЯ"
200INPUTM(J)
210NEXTJ
220FORI=0TOS
230LETP(I)=I
240NEXTI
250FORI=0TOS-1
260FORJ=0TOS-1-I
270IFM(P(J))(<=M(P(J+1)))THEN310
280LETC=P(J+1)
290LETP(J+1)=P(J)
300LETP(J)=C
310NEXTJ
320NEXTI
330FORI=0TOT
340LETC=1:LETD=1
350FORJ=0TOS
360LETK=P(J)

```

Алгоритм 4.9

```

370IFR(I,K)=0THEN460
380LETD=M0(K)
390IFD=CTHEN430
400IFD)CTHEN420
410LETC=C-D=80T0390
420LETD=D-C=80T0390
430LETM(I,K)=M0(K)/C*B=M1(I)=M(I,K)
440LETR(I,K)=B
450LETB=M(I,K):LETC=B=80T0470
460LETM(I,K)=0
470LETX(I,K)=0
480NEXTJ
490NEXTI
500LETX0=1
510LETJ=0
520LETK0=P(J):LETB=0
530FORK1=0TON(K0)
540LETC=A(0,16*K0+K1)
550FORI=0TOT
560FORL=1TOA(I+1,16*K0+K1)
570LETC=C*X(I,J)
580NEXTL
590NEXTI
600LETB=B+C
610NEXTK1
620IFINT(B/M0(K0))(>)B/M0(K0)THEN730
630IFJ=8THEN690
640FORI=0TOT
650LETX(I,J+1)=X(I,J)
660NEXTI
670LETJ=J+1
68080T0520
690PRINTX0;"-РЕШЕНИЕ"
700FORI=0TOT
710PRINT"X" I+1;"=" I X(I,J) I" (MOD" M1(I) I" "-"
720NEXTI:LETX0=X0+1
730LETI=T
740IFX(I,J)=M(I,K0)-R(I,K0)THEN760
750LETX(I,J)=X(I,J)+R(I,K0):80T0520
760IFR(I,K0)=0THEN780
770LETX(I,J)=X(I,J)-INT(X(I,J)/R(I,K0)):R(I,K0)
780LETI=I-1
790IFI=0THEN740
800LETJ=J-1
810IFJ=0THENLETK0=P(J):80T0730
820IFX0=1THENPRINT"РЕШЕНИЙНЕТ"
830STOP

```

Алгоритм 4.9 (Окончание)

Еще одно соображение учтено в АЛГОРИТМЕ 4.9. Если число решений не очень велико, время работы программы будет зависеть от того, насколько велик, в первую очередь, модуль I -го сравнения, затем насколько велико число шагов M_{i2}/M_i (при взаимно простых M_i и M_{i2} — просто от M_{i2} / и т.д. Поэтому в некоторых случаях важен порядок ввода сравнений системы, т.е. целесообразно вводить сравнения в порядке возрастания их модулей, начиная с меньшего и заканчивая наибольшим. Если же этот порядок нарушается и модули не взаимно простые, то выясняется, что использование этого алгоритма никакого выигрыша не даёт.

Например, если $M_i = 8$, а $M_{i2} = 2$, то никакого ускорения работы программы не произойдет, так как решениями системы могут быть только решения, подобранные к I -му сравнению.

Поэтому, перед началом работы алгоритма необходимо расположить сравнения в порядке неубывания модулей сравнений. Для этого в алгоритме вводится косвенная адресация, т.е. вычисляется необходимая перестановка сравнений, нужный порядок запоминается, а все содержимое массивов не изменяется.

Рассмотрим реализацию предложенного алгоритма — ПРОГРАММУ 4.9. Строки 220-240 заполняют массив P косвенными адресами, т.е. номерами сравнений из решаемой системы. Далее сортируют эти косвенные адреса — их располагают в соответствии с модулями сравнений /строки 250-320/. Таким образом, на первом месте в $P(0)$ указывается номер сравнения с наименьшим модулем, на втором в $P(1)$ — номер сравнения с модулем не меньшим первого, и т.д. Следующий этап — вычисление модулей для каждого неизвестного в каждом сравнении — $M(3,7)$ и шагов изменения каждого неизвестного в каждом сравнении — $R(3,7)$. Массив $R(3,7)$ используется несколько иначе. Так, раньше $R(I,J) = 0$ было признаком отсутствия $X(I)$ в сравнении с номером J , теперь $R(I,J)$ после выполнения строк 360-520 будет указывать на шаг изменения неизвестного в данном сравнении. Конкретнее, если неизвестное отсутствует в сравнении, шаг принимается равным 0.

Этот фрагмент программы почти аналогичен фрагменту вычисления модулей для неизвестных в АЛГОРИТМЕ 4.8. Для вычисления НОД используются имена D и C , для вычисления очередного НОК используется имя B . Отличие состоит, во-первых, в том, что сравнения обрабатываются не в порядке следования, а в порядке неубывания модулей сравнений, т.е. по косвенному адресу $K = P(J)$; во-вторых, в том, что все промежуточные значения модулей неизвестных запоминаются в

массиве $M(I, K)$, а предшествующие модули - в $R(I, K)$. Исключением является случай, когда неизвестное отсутствует в сравнении; при этом строки 370 и 460 $R(I, K)$ устанавливается в 0 и осуществляется переход к следующему сравнению. Начальные значения $X(I, K)$ подстановок для всех сравнений обнуляются /строка 470/. С вычислением модулей это не связано. Фактически это уже подготовка к перебору всех подстановок, но если ее делать отдельно, то потребуется еще раз организовать циклы по I и K .

Строки 500-510 последние перед циклом поиска решений. Имя $X0$ указывает на номер искомого решения, J - на косвенный номер сравнения, т.е., куда нужно делать подстановку. Подстановка решения производится в строках 520-620. Если подстановка неудачна, управление передается фрагменту выработки следующей подстановки, т.е. в строки 730-810; если подстановка удачна, то проверяется, было ли сравнение $K0 = P(J)$ последним в системе; если да, то переходим к печати очередного решения, что обеспечивают строки 690-720; если нет, то полученную подстановку в качестве исходной переписываем для следующего сравнения /строки 640-660/. Затем увеличиваем номер проверяемого сравнения /строка 670/ и переходим на проверку подстановки.

Ключевым для понимания данной программы является фрагмент выработки следующей подстановки /строки 730-810/. Изменение подстановки начинается с последнего неизвестного, имеющего номер T /строка 760/. I - номер неизвестного, которых необходимо изменить.

Во-первых, в строке 740, учитывая модуль данного неизвестного $M(I, K0)$ и шаг изменения неизвестного $R(I, K0)$ проверяется, можно ли его увеличить. Если можно, то это выполняется в строке 750 и полученная подстановка отправляется на проверку.

Во-вторых, если нельзя - требуется уточнение. Если шаг $R(I, J)$ есть 0, то неизвестное отсутствует в J -м сравнении, и можно сразу рассматривать $I-1$ неизвестное /строка 760/.

В-третьих, если неизвестное имеется в данном сравнении и его увеличить невозможно, то /строка 770/ восстанавливается первоначальное значение неизвестного и переходим к другому неизвестному.

В-четвертых, если /строка 780/ изменим номер неизвестного, то необходимо проверить, существует ли такое неизвестное? Если его номер I меньше 0, то это признак того, что для данного сравнения все возможные подстановки данного частного решения уже проводились и необходимо вернуться на предыдущее сравнение /строки 800-810/.

Подстановка, которая была оставлена в столбце $K\emptyset$ массива $X(I, K\emptyset)$ для предыдущего сравнения, удовлетворяет этому сравнению, иначе не обратились бы к проверке этой подстановки для следующего сравнения системы. Но все решения системы сравнений, если они были, на основе этой подстановки уже получены, т.е. необходимо изменить эту подстановку, и потому из строки $8I\emptyset$ переход выполняется на строку $73\emptyset$ - изменение очередной подстановки. При этом необходимо восстановить косвенный адрес $K\emptyset$ (вся обработка выполняется с использованием этого адреса).

В-пятых, если при вычислении номера предшествовавшего сравнения получим отрицательное J /строка $8I\emptyset$ /, то это - признак того, что перебраны все подстановки для I -го сравнения, а следовательно, и все возможные подстановки для системы сравнений. После этого остается выяснить наличие хотя бы одного решения системы и выдать соответствующее сообщение, за что отвечает строка $82\emptyset$.

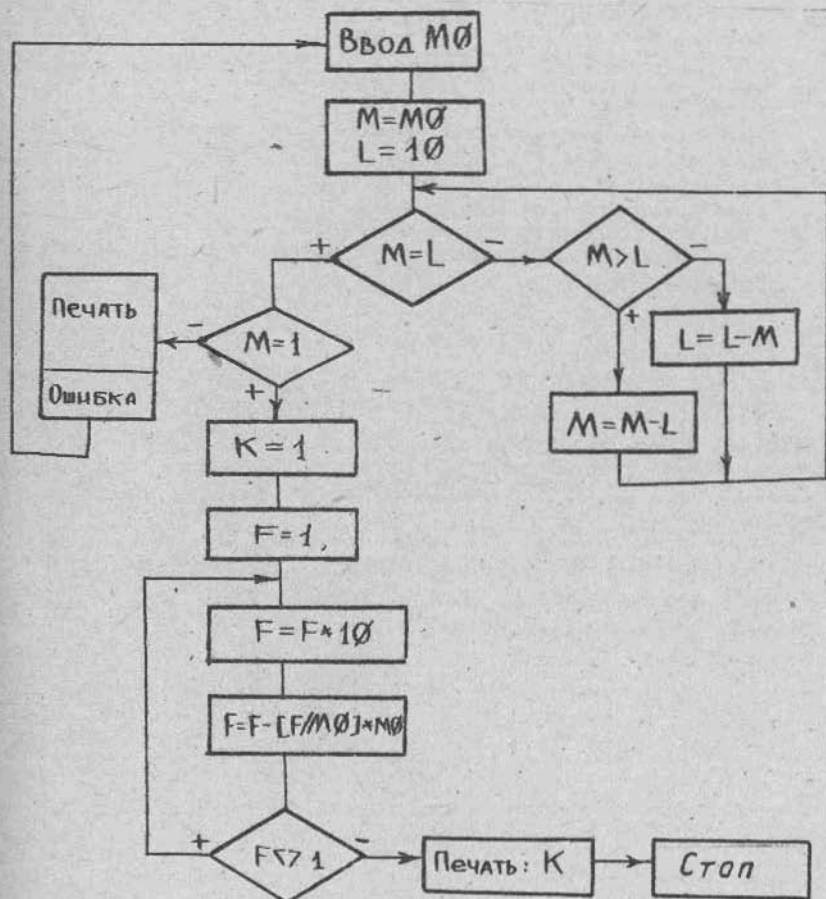
Есть несколько возможностей для улучшения данной программы. Например, вполне можно было бы обойтись одномерным массивом $X(I)$ вместо $X(I, K\emptyset)$ без существенных изменений программы; отказаться от массива $M(I, K\emptyset)$. Этот массив и массив $R(I, K\emptyset)$ имеет почти одинаковое содержание, но со сдвигом по индексу $K\emptyset$ на I /здесь требуется некоторая изобретательность/.

В целом приведенная программа достаточно полно характеризует возможности языка Бейсик и того архаичного стиля программирования, который неизбежно навязывается при написании программ на этом языке.

Интересным приложением теории сравнений является отыскание признаков делимости на некоторое натуральное число M , взаимно простое с числом IO .

Теорема 4.6. Пусть M - число взаимно простое с числом IO , K - наименьшее натуральное число, такое, что $IO^K = 1(M)$ и записано в системе счисления с основанием IO . Число N делится на M тогда и только тогда, когда на M делится сумма чисел, которые получаются при разбиении справа налево цифровой записи числа N на грани по K цифр в каждой.

Для решения задачи необходимо вычислить значение K , т.е. найти наименьшее среди натуральных чисел решение сравнения $IO^K = 1(M)$. АЛГОРИТМ 4.10 выполняет эту работу. Если введенное число не взаимно просто с IO , то выдается соответствующее сообщение и предлагается ввести новое число.



Алгоритм 4.10а

```

5 REM ВМ ЧИСЛЕНИЕ ПРИЗНАКА ДЕЛИМОСТИ
10 PRINT "ПРИЗНАК ДЕЛИМОСТИ НА -" : L=10
20 INPUT M:M=M
30 IF M=L THEN GOTO 60
40 IF M>L THEN LET M=M-L:GOTO 30
50 L=L-M:GOTO 30
60 IF M=1 THEN GOTO 90
70 PRINT "ЧИСЛО "M" НЕ ВЗАИМНО ПРОСТО С 10 И 7"
80 GOTO 10
90 K=1:F=1
100 F=F*10:F=F-INT(F/M)*M
110 IF F<>1 THEN LET K=K+1:GOTO 100
120 PRINT "СУММА ГРАНЕЙ ПО "K" ДИФ"
130 END

```

Алгоритм 4.106

СПИСОК ЛИТЕРАТУРЫ

1. Ершов А.П., Звенигородский Г.А., Первин Ю.А. Школьная информатика /концепции, состояния, перспективы/. - Новосибирск, 1979.
2. Основные направления реформы общеобразовательной и профессиональной школы /Правда. - 1984. - 14 апреля.

ОГЛАВЛЕНИЕ

1. Деление с остатком. Простые числа	4
2. Наибольший общий делитель. Наименьшее общее кратное	18
3. Конечные цепные дроби	21
4. Сравнения	34
Список литературы	58

Методические рекомендации
по подготовке студентов к обеспечению
компьютерной грамотности учащихся
для студентов специальностей 2104 и 2105
"Математика и "Физика"

Составители Александр Владимирович Спиваковский
Александр Петрович Ковтушенко

Редактор Л.И.Шульгач
Корректоры Р.И.Петрова
С.М.Куник
Н.Ф.Слонина
Т.Н.Сенник-Шевчук

Подп. к печ. 04.02.98 . Формат 60×84^{1/16}. Бумага
тип. №3 . Печать офсетная. Усл. печ. л. 3,49 . Усл. кр.-отт. 36 .
Уч.-изд. л. 3,49 . Изд. № 4576 . Тираж 200 .
Зак. № 12031 . Бесплатно.

Херсонский Государственный педагогический институт
им. Н. К. Крупской
325000, г. Херсон, ул. «40-річчя Жовтня», 27

ГП ППО «Укрвузполиграф».
252151, г. Киев, ул. Волинская, 60.