Всероссийская олимпиада школьников по информатике 2019-2020 учебного года.

Муниципальный этап, Амурская область

1. Разбор Задача A Треугольники

Автор: В.М. Гуровиц

Имя входного файла: a.in
Имя выходного файла: a.out
Максимальное время работы на одном тесте: 2 секунды
Максимальный объем используемой памяти: 64 мегабайта

Вася нарисовал выпуклый N-угольник и провел в нем несколько диагоналей таким образом, что никакие две диагонали не пересекаются внутри N-угольника. Теперь он утверждает, что весь N-угольник оказался разбит на треугольники. Напишите программу, которая проверяет истинность Васиного утверждения.

Формат входных данных

Во входном файле записано сначала число N - количество вершин N-угольника (3 <= N <= 1000). Далее записано число M - количество диагоналей, проведенных Васей. Далее записано M пар чисел, задающих диагонали (каждая диагональ задается парой номеров вершин, которые она соединяет). Гарантируется, что каждая пара чисел задает диагональ (то есть две вершины различны, и не являются соседними), а также что никакие две пары не задают одну и ту же диагональ. Никакие две диагонали не пересекаются внутри N-угольника.

Вершины N-угольника нумеруются числами от 1 до N.

Формат выходных данных

Если Васино утверждение верно, то выходной файл должен содержать единственное число 0.

В противном случае быть выведено сначала число K - количество вершин в какой-нибудь не треугольной части. Далее должно быть выведено K чисел - номера вершин исходного N-угольника, которые являются вершинами этой K-угольной части в порядке обхода этой части.

Примеры

a.in	a.out
3 0	0
4 1 1 3	0
6 2 1 3 5 3	4 1 3 5 6

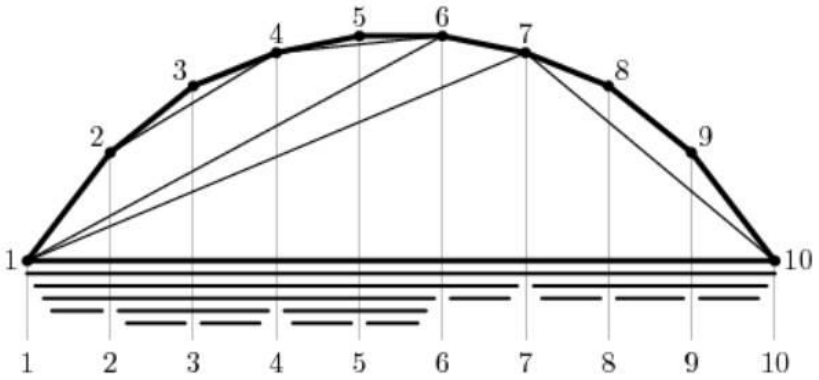
Указания к возможному разбору задачи (Автор: В.М. Гуровиц)

Решение задачи разбивается на две части: нужно сначала определить, есть ли не треугольные части, а затем найти одну из них.

Первая часть решения реализуется просто: N-угольник разбивается непересекающимися диагоналями на треугольники тогда и только тогда, когда количество диагоналей равно N-3. Это несложно доказать, сравнив сумму углов N-угольника с суммой углов треугольников, на которые он разбивается диагоналями.

Если же диагоналей меньше, чем N-3, то в многоугольнике есть хотя бы одна не треугольная грань. Ее поиском и займемся.

Всероссийская олимпиада школьников по информатике 2019-2020 учебного года.
Муниципальный этап, Амурская область



Разберем решение задачи на примере 10-угольника с вершинами 1, 2, 3, :, 9, 10 и диагоналями 1 - 6, 2 - 4, 4 - 6, 1 - 7, 7 - 10. Поскольку решение не зависит от того, как именно мы изобразим данный выпуклый многоугольник, расположим его на плоскости таким образом, как показано на рисунке: сторону 1 - 10 расположим горизонтально, а остальные вершины расположим над этой стороной.

Спроецируем все стороны и диагонали на горизонтальную прямую и далее будем рассматривать только эти проекции. Заметим, что поскольку стороны и диагонали многоугольника не пересекались, то и их проекции не пересекаются, то есть любые два отрезка-проекции либо не имеют внутренних точек, либо один из отрезков лежит внутри другого. Назовем отрезок 1 - 10 отрезком уровня 1. Максимальные отрезки, на которые он разбивается (1 - 7 и 7 - 10) назовем отрезками уровня 2. Отрезки, на которые разбиваются отрезки второго уровня, назовем отрезками третьего уровня (отрезок 1 - 7 разбивается на отрезки 1 - 6, 6 - 7). Отрезки третьего уровня в свою очередь разбиваются на отрезки четвертого уровня и т. д. Заметим, что на последних уровнях присутствуют только отрезки длины 1 - проекции сторон многоугольника.

Рассмотрим одну из частей, на которые диагонали разбивают многоугольник, например, 1 - 2 - 4 - 6. Одна из ее сторон (1 - 6) является отрезком третьего уровня, а остальные - это все отрезки четвертого уровня, на которые разбивается отрезок 1 - 6. Так устроена любая из частей, как треугольная, так и не треугольная. Таким образом, для того, чтобы найти не треугольную часть, достаточно найти отрезок, который разбивается на три или более отрезков следующего уровня. В нашем примере таких отрезка два: кроме отрезка 1 - 6 есть еще отрезок 7 - 10.

Перейдем к обсуждению реализации описанной идеи. Будем идти слева направо по горизонтальной прямой, на которую спроецированы все отрезки, и следить за началами и концами отрезков (см. рис.). Обратите внимание: каждый раз, когда мы встречаем начало отрезка, мы опускаемся на один уровень вниз, а когда мы встречаем конец отрезка, мы поднимаемся на один уровень вверх. Таким образом, нас интересует лишь, сколько начал и сколько концов отрезков проектируется в каждую из точек 1, 2, 3, : на горизонтальной оси. (Началом мы считаем вершину с меньшим номером). Определить это можно при считывании входных данных из файла. Пусть o[i] - количество начал, а c[i] - количество концов отрезков, проектирующихся в точку i. Заметим, что в каждую из точек 2, :, n-1 проектируется одно начало и один конец стороны многоугольника, в точку 1 проектируется два начала сторон, а в точку n - два конца. Напишем соответствующий код:

```
o[1]:=2;   for i:=2 to n-1 do o[i]:=1;   o[n]:=0;
c[1]:=0;   for i:=2 to n do c[i]:=1;     c[n]:=2;
```

Далее будем читать диагонали из входного файла и подсчитывать начала и концы:

```
for i:=1 to M do begin
  read(a,b);
  if a>b then begin inc(o[b]);inc(c[a]); end
    else begin inc(o[a]);inc(c[b]); end;
```

```
 end;
```

Переходим к реализации основной части алгоритма. Будем двигаться слева направо, просматривая в каждой точке сначала все концы, а затем все начала отрезков. Введем переменную level, в которой будем хранить текущий уровень отрезка.

```
 level:=0;
 for i:=1 to n do begin
   for j:=1 to c[i] do begin
     {Обрабатываем конец отрезка}
   end;
   for j:=1 to o[i] do begin
     {Обрабатываем начало отрезка}
   end;
 end;
```

Что происходит, когда мы встречаем конец отрезка на уровне level? Нам хочется узнать, на сколько отрезков был разбит этот отрезок. Если он был разбит лишь на два отрезка, то мы встретили треугольную грань, которая нас не интересует. Если он был отрезком последнего уровня, и не разбивался на отрезки, то он нас также не интересует. Если же этот отрезок был разбит на три или более отрезков следующего уровня, то мы нашли не треугольную грань. Таким образом, нам необходимо хранить количество отрезков, встретившихся нам на уровне level - обозначим это число l[level]. О том, как его вычислять, мы поговорим чуть позже. Напишем соответствующий код:

```
   if (l[level+1])>2 then begin
   <печатаем найденную многоугольную часть>
   exit;
 end
```

Если мы не нашли требуемую многоугольную часть, то мы обнуляем количество отрезков, найденных на уровне level+1 (они нам больше не нужны), и поднимаемся на один уровень вверх:

```
 else begin
   l[level+1]:=0;
   dec(level);
 end;
```

О том, как печатать многоугольную часть, мы поговорим чуть позже.А что происходит, когда мы встречаем начало отрезка? Мы должны, во-первых, увеличить уровень на 1, а во-вторых, увеличить количество найденных на этом уровне отрезков на 1:

```
 inc(depth);
 inc(l[depth]);
```

Еще раз подчеркнем: на каждом уровне level+1 мы считаем количество отрезков, являющихся частями одного и того же отрезка на уровне level. Как только отрезок на уровне level заканчивается, мы проверяем, на сколько частей он разбит, и если количество этих частей не больше двух, обнуляем l[level+1].

Пока мы еще не научились хранить и печатать нужные нам отрезки. Но прервемся ненадолго и оценим, какие ресурсы требует наша программа. Поскольку количество диагоналей не превосходит количества вершин, для хранений начал и концов всех отрезков нам понадобится массив, длина которого пропорциональна количеству вершин многоугольника N. Для заполнения этих массивов нам понадобится порядка N операций. Таким образом, наш алгоритм работает за линейное относительно N время и использует память, также пропорциональную N. Мы постараемся, заканчивая реализацию алгоритма, не выходить за указанные ограничения. Итак, нам нужно запоминать встречающиеся вершины. Достаточно запоминать только начала отрезков. Будем на каждом уровне запоминать в массиве first[level] первое встретившееся начало отрезка, в массиве

second[level] - второе встретившееся начало, а в массиве next[k] - k-е встретившееся начало любого уровня:

```
if l[level]=1 then first[level]:=i;
if l[level]=2 then second[level]:=i;
if l[level]>2 then next[l[level]]:=i;
```

На первый взгляд может показаться странным, что, скажем, 3-е по счету начало на любом уровне мы записываем в один и тот же элемент одного и того же массива, затирая тем самым информацию обо всех ранее найденных третьих началах. Но на самом деле она нам и не нужна. Пусть мы встретили конец некоторого отрезка на уровне level, и оказалось, что он разбит не менее чем на три части. Это значит, что каждая из этих частей в свою очередь разбита менее чем на три части (иначе мы бы остановились, дойдя до конца соответствующего отрезка), и каждая из частей этих частей также разбита менее чем на три части и т. д. Таким образом, с тех пор как мы встретили третий (четвертый, пятый, ...) отрезок на уровне level+1, мы более не встречали третьих (четвертых, пятых, ...) отрезков, то есть в массиве next хранятся номера вершин, служащих началами отрезков именно на уровне level+1. Таким образом, мы обходимся всего тремя дополнительными массивами first, second и next длины порядка N каждый. Теперь несложно вывести вершины многоугольной части, дополнив фрагмент программы, написанный нами выше:

```
if (l[level+1])>2 then begin
  {печатаем найденную многоугольную часть}
  write(i,' ',first[level+1],' ',second[level+1],' ');
  for k:=3 to l[level+1] do write(next[k],' ');
  exit;
end
```

Осталось объяснить, зачем мы печатаем число i. Дело в том, что кроме начал всех отрезков есть еще одна вершина, которая является вершиной нашей многоугольной части - самая правая вершина. Это как раз последний встреченный нами конец отрезка, то есть вершина с номером i.

Существует множество других алгоритмов, решающих задачу за время O(N log N) или $O(N^2)$, что удовлетворяет приведенным в задаче ограничениям. Но реализация этих алгоритмов не проще, а зачастую сложнее приведенной в этом разборе реализации линейного алгоритма.

Возможный вариант решения задачи

```
var
  k,N,d,i,j,depth,a,b : longint;
  o,c,l,first,second,next : array [1..1000] of longint;

begin
  assign(INPUT,'a.in');reset(INPUT);
  assign(OUTPUT,'a.out');rewrite(OUTPUT);
  read(N,d);
  if d=N-3 then begin write('0');        close(INPUT);close(OUTPUT);exit; end;
  o[1]:=2;   for i:=2 to n-1 do o[i]:=1;   o[n]:=0;
  c[1]:=0;   for i:=2 to n do c[i]:=1;     c[n]:=2;
  for i:=1 to d do begin
    read(a,b);
    if a>b then begin inc(o[b]);inc(c[a]); end
    else begin inc(o[a]);inc(c[b]); end;
  end;
  depth:=0;
  for i:=1 to n do begin
    for j:=1 to c[i] do begin
```

```
    dec(depth);
    if (l[depth+2])>2 then begin
      write(l[depth+2]+1,' ');
      write(i,' ',first[depth+2],' ',second[depth+2],' ');
      for k:=3 to l[depth+2] do write(next[k],' ');
      close(INPUT);close(OUTPUT);
      exit;
    end;
    l[depth+2]:=0;
  end;
  for j:=1 to o[i] do begin
    inc(depth);inc(l[depth]);
    if l[depth]=1 then first[depth]:=i;
    if l[depth]=2 then second[depth]:=i;
    if l[depth]>2 then next[l[depth]]:=i;
  end;
 end;
 close(INPUT);close(OUTPUT);
end.
```

Всероссийская олимпиада школьников по информатике 2019-2020 учебного года.

Муниципальный этап, Амурская область

2. Разбор задачи B Палиндром

Имя входного файла: b.in

Имя выходного файла: b.out

Максимальное время работы на одном тесте: 2 секунды

Максимальный объем используемой памяти: 64 мегабайта

Палиндром - это строка, которая читается одинаково как справа налево, так и слева направо.

Во входном файле записан набор больших латинских букв (не обязательно различных). Разрешается переставлять буквы, а также удалять некоторые буквы. Требуется написать программу, которая из данных букв по указанным правилам составит палиндром наибольшей длины, а если таких палиндромов несколько, то первый в алфавитном порядке

Формат входных данных

В первой строке входного файла записано число N (1 <= N <= 100000). Во второй строке записана последовательность из N больших латинских букв (буквы записаны без пробелов).

Формат выходных данных

В единственной строке выходного файла выдайте искомый палиндром.

Примеры

b.in	b.out
3 AAB	ABA
6 QAZQAZ	AQZZQA
6 ABCDEF	A

Итак, разбор задачи

Подсчитаем, сколько раз во входном файле встречается каждая буква. Для этого можно воспользоваться массивом:

```
k : array ['A'..'Z'] of longint;
```

Соответствующая часть программы будет выглядеть примерно так:

```
readln(N);
for i:=1 to N do begin
  read(ch);
  Inc(k[ch]);
end;
```

Если в палиндроме общее количество символов четно, то для каждого символа найдется симметричный ему, поэтому каждая буква должна встречаться четное количество раз. Если же общее количество символов нечетно, то для каждого символа, кроме стоящего в центре палиндрома, найдется пара, поэтому все буквы кроме одной, должны встречаться четное число раз. Таким образом, если некоторая буква встречается во входном файле четное число раз, то мы можем использовать все ее вхождения, а из букв, которые встречаются нечетное число раз лишь одну (первую по алфавиту) мы сможем использовать нечетное число раз. Определим, какая буква встречается нечетное количество раз первой:

```
o:='-'; {Буква, встречающаяся нечетное число раз}
for ch:='Z' to 'A' do
  if k[ch] mod 2=1 then o:=ch;
```

Если во входном файле нет букв, встречающихся нечетное число раз, то первый по алфавиту палиндром устроен следующим образом: сначала идет половина букв A,

затем половина букв B, :, затем половина букв Z, далее вторая половина букв Z, вторая половина букв Y, :, вторая половина букв A:

```
if o='-' then begin
  for ch:='A' to 'Z' do
    for i:=1 to k[ch] div 2 do
      write(ch);
  for ch:='Z' downto 'A' do
    for i:=1 to k[ch] div 2 do
      write(ch);
end;
```

Если во входном файле есть буквы, встречающиеся нечетное число раз, то первый по алфавиту палиндром устроен так же, как и палиндром четной длины, за одним исключением: в центре палиндрома расположена буква, записанная в переменную o:

```
if o<>'-'; then begin
  for ch:='A' to 'Z' do
   for i:=1 to k[ch] div 2 do
     write(ch);
 write(o);
  for ch:='Z' downto 'A' do
    for i:=1 to k[ch] div 2 do
      write(ch);
end;
```

Вариант возможного решения на Delphi

```
{$apptype console}
{$q+,r+,n-,o+}
const cinfile = 'b.in';
      coutfile = 'b.out';
type int = longint;
var a : array['A'..'Z'] of int;
    n, i : int;
    c, center : char;
begin
   assign(input, cinfile); reset(input);
   assign(output, coutfile); rewrite(output);
   fillchar(a, sizeof(a), 0);
   readln(n);
   for i:=1 to n do begin
      read(c);
      inc(a[c]);
   end;
   center:=#32;
   for c:='A' to 'Z' do begin
      if (a[c] and 1 <> 0) and (center = #32) then
         center:=c;
      a[c]:=a[c] shr 1;
      for i:=1 to a[c] do write(c);
   end;
   if center <> #32 then write(center);
   for c:='Z' downto 'A' do
      for i:=1 to a[c] do write(c);
   close(input); close(output);
end.
```

Всероссийская олимпиада школьников по информатике 2019-2020 учебного года.
Муниципальный этап, Амурская область

Всероссийская олимпиада школьников по информатике 2019-2020 учебного года.

Муниципальный этап, Амурская область

3. Разбор задачи C Представление числа

Автор: А.Ю.Гусаков

Имя входного файла: c.in
Имя выходного файла: c.out
Максимальное время работы на одном тесте: 2 секунды
Максимальный объем используемой памяти: 64 мегабайта

Учительница математики попросила школьников составить арифметическое выражение, так чтобы его значение было равно данному числу N, и записать его в тетради. В выражении могут быть использованы натуральные числа, не превосходящие K, операции сложения и умножения, а также скобки. Петя очень не любит писать, и хочет придумать выражение, содержащее как можно меньше символов. Напишите программу, которая поможет ему в этом.

Формат входных данных

В первой строке входного файла записаны два натуральных числа: N (1 <= N <= 10000) - значение выражения и K (1 <= K <= 10000) - наибольшее число, которое разрешается использовать в выражении.

Формат выходных данных

В единственной строке выходного файла выведите выражение с данным значением, записывающееся наименьшим возможным количеством символов.

Если решений несколько, выведите любое из них.

Примечание

При подсчете длины выражения учитываются все символы: цифры, знаки операций, скобки.

Примеры

c.in	c.out	Пояснение: длина получившегося выражения
7 3	3+1+3	5
15 20	15	2
176 1	(1+1+1+1)*(1+1+1+1)*(1+1+(1+1+1)*(1+1+1))	41

Разбор задачи

Задача решается применением метода динамического программирования. Прежде чем перейти к описанию идеи решения, введем несколько определений.

Будем называть выражение C выражением первого типа, если оно является числом от 1 до K или если последней операцией при его вычислении является сложение, то есть, C = A + B, где A и B - некоторые выражения.

Будем называть выражение C выражением второго типа, если оно является числом от одного до K или если последней операцией при его подсчете является умножение, то есть, C = A * B, где A и B - некоторые выражения. Например, (2+3*4)+5*7 - выражение первого типа, (2+3)*(3+2) - выражение второго типа, 1 - выражение, относящееся как к первому, так и ко второму типу.

Теперь обсудим идею решения. Пусть a[1,m] равно наименьшей длине выражения первого типа, значение которого равно m; a[2,m] равно наименьшей длине выражения второго типа, значение которого равно m. Легко видеть, что при m <= K значения a[1,m] и a[2,m] равны длине числа m. Пусть теперь m > K, и у нас уже вычислены все значения a[1,1], :, a[1,m-1], a[2,1], :, a[2,m-1].

Сначала вычислим a[1,m]. Поскольку соответствующее выражение представимо в виде C=A+B, нам достаточно перебрать все возможные значения и типы выражений A и B и выбрать оптимальную комбинацию. Иными словами, a[1,m]=min(a[i1,t]+a[i2,m-t]+1), где i1, i2 - 1 или 2, а t <= m/2 (можно считать, что значение A меньше или равно m/2, иначе A

и B можно поменять местами). В переменных how[1,m][1], how[1,m][2], how[1,m][3] будем хранить те значения i1, t, i2 соответственно, при которых достигается минимальное значение.

Теперь вычислим a[2,m]. Соответствующее выражение представимо в виде A*B, где A и B - либо выражения второго типа, либо выражения первого типа, взятые в скобки (либо какая-то их комбинация). Будем считать, что значение A меньше или равно , иначе поменяем местами A и B. Таким образом, a[2,m]=min(a[i1,t]+a[i2,m/t]+1+2*(4-i1-i2)), где i1, i2 - 1или 2, a t <= sqrt(2) и m делится на t.

Замечание. Число 4-i1-i2 равно количеству выражений первого типа среди A и B, а значит, равно количеству пар скобок, которое придется добавить.

В переменных how[2,m][1], how[2,m][2], how[2,m][3] будем хранить те значения i1, t, i2 соответственно, при которых достигается минимум.

Теперь мы умеем вычислять значения a[1,1], ..., a[1,N], a[2,1], ..., a[2,N]. Естественно, что искомое выражение является выражением первого или второго типов, поэтому его длина равна min(a[1,N], a[2,N]). Осталось только научиться его выводить. Для этого напишем рекурсивную процедуру Save(last, n), которая будет выписывать выражение типа last, значением которого является n. Во-первых, если n<=K, то нужно просто вывести число cur. Иначе нужно посмотреть на значение last.

Если last=1 (то есть выражение имеет вид A+B), то сначала выпишем первое слагаемое - Save(how[last,n][1],how[last,n][2]) - затем поставим знак '+' и выпишем второе слагаемое - Save(how[last,n][3],n-how[last,n][2]).

Если last=2 (то есть, имеем дело с произведением), то нужно действовать аналогично случаю last=1, но при этом не забыть, что если какой-то из множителей первого типа, то его надо окружить скобками.

Количество операций, выполняемых алгоритмом, пропорционально N2 что удовлетворяет указанным в условии задачи ограничениям.

Приведем теперь примерный полный текст решения с подробными комментариями:

```
const
 MaxN=10*1000;
var
 N,K:LongInt;
 a:array [1..2,1..MaxN] of LongInt;
 how:array [1..2,1..MaxN,1..3] of LongInt;
 m,i1,i2,t:LongInt;
function CountOfDigits(a:LongInt):byte;
//подсчитывает количество цифр в числе a
var
 res:LongInt;
begin
 res:=0;
 while a>0 do begin
   a:=a div 10;
   //каждое деление на 10 уменьшает количество цифр на одну
   inc(res);
 end;
 CountOfDigits:=res;
end;
procedure Save(last,n:LongInt);
//тип выражения и его значение
begin
 if (n<=K) and (a[last,n]=CountOfDigits(n)) then
```

```
  write(n)
//если n<=K, просто выводим n
 else if last=1 then begin
//имеем дело с суммой
  Save(how[last,n][1],how[last,n][2]);
//выводим первое слагаемое
  write('+');
  Save(how[last,n][3],n-how[last,n][2]);
//выводим второе слагаемое
 end else if last=2 then begin
//имеем дело с произведением
  if how[last,n][1]=1 then write('(');
  Save(how[last,n][1],how[last,n][2]);
//выводим первый множитель
  if how[last,n][1]=1 then write(')');
//если он первого типа, то окажется в скобках
  write('*');
  if how[last,n][3]=1 then write('(');
//аналогично поступаем со вторым множителем
  Save(how[last,n][3],n div how[last,n][2]);
  if how[last,n][3]=1 then write(')');
 end;
end;
Begin
 read(N,K);
 for m:=1 to K do begin
//случай m<=K
  a[1,m]:=CountOfDigits(m);
  a[2,m]:=CountOfDigits(m);
 end;
 for m:=K+1 to N do begin
//случай K < m <= N
// считаем a[1,m] и how[1,m], перебирая i1, i2, t, ...
  a[1,m]:=MaxLongInt;
//начальное значение
  for t:=1 to (m div 2) do
    for i1:=1 to 2 do
      for i2:=1 to 2 do
        if a[i1,t]+a[i2,m-t]+1<a[1,m] then begin
</pre>
<pre>
          a[1,m]:=a[i1,t]+a[i2,m-t]+1;
          how[1,m][1]:=i1;
          how[1,m][2]:=t;
          how[1,m][3]:=i2;
        end;
// считаем a[2,m] и how[2,m], перебирая i1, i2, t, ...
  a[2,m]:=MaxLongInt;
//начальное значение

  for t:=1 to trunc(sqrt(m)) do if m mod t=0 then
    for i1:=1 to 2 do
```

```
      for i2:=1 to 2 do
        if a[i1,t]+a[i2,m div t]+1+2*(4-i1-i2)<a[2,m] then begin
          a[2,m]:=a[i1,t]+a[i2,m div t]+1+2*(4-i1-i2);
          how[2,m][1]:=i1;
          how[2,m][2]:=t;
          how[2,m][3]:=i2;
        end;
 end;
 if a[1,N]<a[2,N] then
// выводим выражение наименьшей длины
   Save(1,N)
 else
   Save(2,N);
End.
```

Это далеко не единственный вариант реализации. Можно было обойтись без рекурсии. Для этого в элементах a[1,m] и a[2,m] надо хранить не длины выражений, а сами выражения (массив how в этом случае вообще не нужен). В конце программы необходимо просто вывести более короткое из выражени a[1,N] и a[2,N].

Можно грубо оценить длину наибольшей строки и понять, что памяти на такую реализацию хватит. В действительности, самая длинная строка при данных в условии ограничениях получается при N=9358, K=1: ее длина всего лишь 83 символа.

Оказывается, что приведенную динамическую схему можно упростить. Достаточно для каждого числа m хранить информацию лишь о самом коротком выражение, независимо от его типа. При этом, если для данного числа самое короткое выражение может быть как выражением типа 1, так и выражением типа 2, то нужно выбирать произведение, а не сумму.

4. Разбор задачи D Строки в книге

Подсчитаем сначала, сколько страниц будет заполнено целиком, если мы напечатаем только первые N строк. Для этого разделим N на K с остатком. Неполное частное как раз и будет равно искомому числу – количеству страниц, на которых напечатано по K строк. Если остаток от деления равен нулю, то строка с номером N окажется последней на странице с номером N div K. Если же остаток отличен от нуля, то эта строка окажется уже на следующей странице – на странице с номером (N div K) + 1 – и ее порядковый номер будет в точности равен этому остатку – N mod K. Осталось написать короткую программу:

```
read(K,N);
if N mod K = 0 then write(N div K, ' ', K)
else write (N div K + 1, ' ', N mod K);
```

Можно решить эту задачу и без использования условного оператора:

```
read(K,N);
write((n-1) div k) + 1, (n-1) mod k + 1);
```

Пример возможного кода рассуждения (без работы с файлами)

```
uses crt;
var k,n,ns,np:integer;
begin
clrscr;
write('Введите количество строк на странице k=');
```

```
readln(k);
write('Введите номер строки в тексте n=');
readln(n);
ns:=(n-1) div k+1;
np:=(n-1) mod k+1;
write('Номер страницы=',ns,' номер строки=',np);
readln
end.
```