БЪЛГАРСКА АКАДЕМИЯ НА НАУКИТЕ . BULGARIAN ACADEMY OF SCIENCES

ПРОБЛЕМИ НА ТЕХНИЧЕСКАТА КИБЕРНЕТИКА И РОБОТИКАТА, 51
PROBLEMS OF ENGINEERING CYBERNETICS AND ROBOTICS, 51

София . 2001 . Sofia

On the "Greedy" Algorithm

Krassimir Penev, Petar Petrov***

**Academy for the Advancement of Science and Technology, USA*
***Senior System Analyst, USA*

Introduction

Born on the periphery of the Mathematical Science, today the combinatorics blooms as one of the most rapidly developing branches of Mathematics. A testimony to that is not only the enormous number of scientific publications on it but also the increasing interest of physicists, chemists, biologists and engineers to the applications of various combinatorial structures. New and new problems of the Applied Mathematics are not only a powerful incentive to look for concrete solutions but also to form specific methods and results. Particularly close is the relationship between Combinatorics and Computer Science and Information Technologies – probably closer than any other branch of Mathematics. It is important to note that the applications of the Information Technologies should be based on solid combinatorial knowledge.

This article is about a couple of applications of one of the most natural and effective combinatorial algorithms – the so-called "greedy" algorithm. The examples are intentionally picked from different fields on order to underline its universal mathematical significance. Solving these problems is in fact a proof for the correctness of algorithmic procedures, which leads to the practical application of the greedy algorithm as a method for solving combinatorial problems as well as a means of exploring combinatorial problems with computer programs.

Egyptian Fractions

The ancient Egyptian papyri tell us something interesting. To perform operations on proper fractions, the Egyptians would represent them as a sum of fractions with a numerator of one and different denominators. For example

$$\frac{3}{7} = \frac{1}{3} + \frac{1}{11} + \frac{1}{231}.$$

That is why the fractions of $1/n$, $n=2, 3, \ldots,$ have been attracting the mathematicians' attention ever since those early days. For the historical tradition these fractions are called Egyptian.

Can any proper fraction be represented as a sum of Egyptian fractions with different denominators, though? Led by practical purposes the Egyptian mathematics did not tackle such a question. Not until many centuries after the golden Egyptian age, in 1202, did someone solve this problem. That was Leonardo of Pisa, better known as Fibonaci. Although this problem has many solutions, here we consider the earliest only – Leonardo's proof. Most probably it is the first nontrivial application of the greedy algorithm.

Problem 1. Prove that every rational number a/b from the interval $(0, 1)$ can be represented as a sum of different Egyptian fractions, i.e. as a sum of the following kind

$$\frac{1}{c_1} + \frac{1}{c_2} + \ldots + \frac{1}{c_n}, \tag{1}$$

where $c_1, c_2, \ldots, c_n$ are different positive integers.

Solution. Let a/b be a proper fraction. We will apply induction with respect to the numerator a. If $a=1$, a/b itself is an Egyptian fraction and the assertion is true. Suppose that every proper fraction whose numerator is less than a can be represented as a sum of different Egyptian fractions. Since $a/b \in (0, 1)$, there exists a unique natural number $n \geq 2$, so that

$$\frac{1}{n} \leq \frac{1}{b} < \frac{1}{n-1}. \tag{2}$$

It is clear that $1/n$ is the biggest Egyptian fraction smaller than or equal to a/b. This is the step where the greedy algorithm comes into play.

Consider the difference

$$\frac{a}{b} - \frac{1}{n} = \frac{an-b}{bn}. \tag{3}$$

Inequalities (2) are equivalent to $0 \leq an - b < a$, hence $(an - b)/bn$ is a proper fraction with a numerator smaller than a. According to the inductive assumption it can be represented in the form of (1) with different denominators $c_1, c_2, \ldots, c_n$. Since

$$\frac{a}{b} - \frac{1}{n} = \frac{an-b}{bn} = \sum_{i=1}^{n} 1/c_i,$$

then

$$\frac{a}{b} = \frac{1}{n} + \frac{1}{c_1} + \frac{1}{c_2} + \ldots + \frac{1}{c_n},$$

and hence a/b. is represented as a sum of Egyptian fractions. It remains to prove that n is different from $c_1, c_2, \ldots, c_n$. Indeed, from $0 \leq an - b < a$ and $a/b < 1$, we have

$$\frac{an-b}{bn} < \frac{a}{bn} < 1\,\frac{1}{n} = \frac{1}{n}.$$

Thus the fraction $(an-b)/bn$ is less than $1/n$ and so is each of the fractions $1/c_i$, which completes the induction.

Formally speaking the proof uses induction and no algorithm seems to be involved. This only a matter of presentation, though. The above solution leads to the effective procedure for representing a/b as a sum of different Egyptian fractions. The main thing in it is that at each step we subtract from the fraction the *greatest possible* Egyptian fraction. The inductive proof is nothing but the proof for the correctness of the algorithm.

On the other hand the above mentioned greatest possible Egyptian fraction is readily determined. Inequalities

$$\frac{1}{n} \le \frac{1}{b} < \frac{1}{n-1}$$

are equivalent to $n \le b/a \le n-1$, and thus n is the smallest integer greater than or equal to a/b. Instead of the formal procedure description here is a program, which represents a proper fraction as a sum of Egyptian fractions.

All programs were written using *Microsoft Visual Basic for Applications* because of its popularity and availability. They can be executed under the standard *Microsoft Visual Basic*, as well as under each of products included in *Microsoft Office*. We skip the error handling in order to minimize the text of the programs.

```
Option Explicit
Option Base 1

Dim a As Double          'The numerator
Dim b As Double          'The denominator
Dim aNs() As Double      'The array for the denominators
                         'of the Egyptian Fractions
Dim lArrayDim As Long   'The dimension of aNs()

Sub Egipt()
   Dim i As Long
   Dim sPrint As String

   lArrayDim = 0

   'Pick the random proper fraction
   Call RandomFraction

   'Prepare the result for output
   sPrint = a & "/" & b & " = "

   'While the numerator is greater than 1
   Do While a > 1
     Call FindFraction
   Loop

   'Output the result
   If lArrayDim > 0 Then
      For i = 1 To lArrayDim
         sPrint = sPrint & "1/" & Int(aNs(i)) & " + "
```

```
      Next i
    End If
    sPrint = sPrint & a & "/" & b
    Debug.Print sPrint
End Sub

'A function that calculates the next Egyptian fraction
'and writes its denominator in the array sNs()
Sub FindFraction()
    Dim n As Double

    If (b / a) = Int(b / a) Then
      'If the fraction a/b has the numerator equals to 1
      b = b \ a:  a = 1
      Exit Sub
    Else
      'Calculate n from n >= b/a > n-1
      n = Int(b / a) + 1
    End If

    'Write n in the array with the results
    lArrayDim = lArrayDim + 1
    ReDim Preserve aNs(lArrayDim)
    aNs(lArrayDim) = n

    'Calculate the numerator and the denominator
    'of the new fraction a/b, which is equal to a/b - 1/n
    a = a * n - b:  b = b * n
End Sub

'Generating the proper fraction
'with a denominator between 2 and 1000
Sub RandomFraction()
    Const LowerBound = 2
    Const UpperBound = 1000

    Randomize
    b = Int((UpperBound - LowerBound + 1) * Rnd + LowerBound)
    a = Int((b - LowerBound + 1) * Rnd + LowerBound)
End Sub
```

Sample output:

```
58/229 = 1/4 + 1/306 + 1/140148
13/25 = 1/2 + 1/50
151/324 = 1/3 + 1/8 + 1/130 + 1/42120
67/91 = 1/2 + 1/5 + 1/28 + 1/1820
17/47 = 1/3 + 1/36 + 1/1692
39/46 = 1/2 + 1/3 + 1/69
121/252 = 1/3 + 1/7 + 1/252
67/340 = 1/6 + 1/33 + 1/11220
```

A Problem in Combinatorial Geometry

Here is a problem in which neither the idea of using the greedy algorithm nor the proof of the correctness of the procedure is obvious.

Problem 2. Consider two sets of line segments as follows: the segments in the first set are colored in blue where as the segments in the second are colored in red. The total length of the segments of each color is 1. Find the smallest possible line segment on which all given segments can be placed so that every two segments of the same color have no points in common except possibly for their endpoints, and every two segments of different colors either have no points in common or one contains the other.

Solution. This problem is significantly harder. To figure out the answer first we consider two simple sets of red and blue segments. The first one consists of two red segments of length 1/2 and the second one consists of two blue segments of lengths ε and $1-\varepsilon$, where ε is a positive number less than 1/2. The "big" blue segment, that of length $1-\varepsilon$, can not accommodate the two red. That is why we have to place the big blue and one red segment next to each other. Thus in this case the smallest segment satisfying the conditions is of length $(1-\varepsilon)+1/2 = 3/2-\varepsilon$. Since this reasoning holds for every, the smallest segment satisfying all cases is of length $\geq 3/2$.

The nontrivial part of the problem is to prove that any system of red and blue segments can be placed on a segment of length 3/2. First we will describe an algorithmic procedure for arranging the segments, which uses the greedy algorithm. Then we will prove the correctness of this procedure.

Let's consider two sets of blue and red segments with the properties described in the problem statement and proceed as follows:

1. Pick segment d of maximum length, which we will call *basic* from now on.

2. Now we start placing on d line segments of the other color (i.e. different from the color of d) so that:

- At each step we pick the longest segment d' of the other color, which has not been placed yet.
- The first picked segment d' is placed so that its left endpoint coincides with that of d, and the left endpoint of any next picked segment d' coincides with the right endpoint of the previous one.
- We stop placing segments when the next picked segment – the longest not placed yet of color different from that of d– can not be accommodated entirely on d.

We repeat the same procedure with the rest of red and blue segments until they are all placed. On every repetition of step 1 we place the next basic segment so that its left endpoint coincides with the right endpoint of the previous basic segment.

From the description of this algorithm it is clear that we follow the rule of the problem statement that every two segments of the same color have no points in common except possibly for their endpoints, and every two segments of different colors either do not have points in common or one contains the other. It is far from obvious though that the so placed segments are entirely accommodated in segment l of length less than or equal to 3/2 .

Without loss of generality suppose that the last placed basic segment is blue. We can consider l as the union of *all blue segments* and of the "*free*" *parts of all basic red segments*. By the free part of a red basic segment we understand its part which is not covered with blue segments. The total length of all blue segments is 1. It remains to prove that the total length of the free parts of the basic red segments is less than or equal to 1/2 .

Consider any red basic segment r. We can say that on r there is at least one placed blue segment b– the next in order of length after r among the blue segments not placed yet. We see this by noting that if this were not true, after placing r, we would have red segments only, thus the last basic segment would be red which is not true. Then the availability of free part f on r means that the next in order of length blue segment *c* was longer than *f*. There exists such a blue segment because the last basic segment is blue. But our picking of consecutive segments follows the greedy algorithm: at each step the longest possible segment is chosen. Hence the blue segment *c*, which was not accommodated in the free part *f*, is not longer than the already placed blue segment *b*. That is why the free part *f* is not bigger in length than *c*, and it follows that the noncovered (with blue) part of the red basic segment *r* is not longer than its covered part.

Then the total of the lengths of all free parts of the red basic segments is not bigger than 1/2 of the total length of the red basic segments and thus – 1/2 of the total length of all red segments, which is 1. The proof of the correctness of the procedure is complete. Following is the practical application of this procedure as a program. We skip the text of the program that generates a sorted array with the sum of its elements equal to 1 – GenSegments, because it is outside the subject of this article.

```
Option Explicit
Option Base 1

Const ic_RED = 1
Const ic_BLUE = 2

Sub Segments()
    Dim i As Long
    Dim iColor As Integer

    'Arrays for the red and blue segments
    Dim sngRed() As Single, sngBlue() As Single
    'Indexes in the red and blue arrays
    Dim lRedIndex As Long, lBlueIndex As Long

    'Arrays for the current and alternative segments
    Dim sngCurr() As Single, sngOpp() As Single
    'Indexes in the current and alternative arrays
    Dim lCurrIndex As Long, lOppIndex As Long

    'The length of the resultant segment
    Dim sngResult As Single

    'Generating the sorted red and blue arrays
    sngRed = GenSegments:  sngBlue = GenSegments

    'Output of both arrays
    Dim sRed As String, sBlue As String

    Debug.Print: Debug.Print "Red:", "Blue:"
    For i = 1 To IIf(UBound(sngRed) > UBound(sngBlue), UBound(sngRed),
UBound(sngBlue))
        If i > UBound(sngRed) Then sRed = "" Else sRed = Format(sngRed(i),
"0.0000")
        If i > UBound(sngBlue) Then sBlue = "" Else sBlue = Format(sngBlue(i),
"0.0000")

        Debug.Print sRed, sBlue
    Next i
```

```
Debug.Print "Basic:", "Arranged:"
sngResult = 0#
lRedIndex = 1: lBlueIndex = 1

'Pick the array with the longest segment
If sngRed(lRedIndex) > sngBlue(lBlueIndex) Then
    iColor = ic_RED

   'Set the current and alternative values
    sngCurr = sngRed:  lCurrIndex = lRedIndex
    sngOpp = sngBlue:  lOppIndex = lBlueIndex
Else
    iColor = ic_BLUE

   'Set the current and alternative values
    sngCurr = sngBlue: lCurrIndex = lBlueIndex
    sngOpp = sngRed:   lOppIndex = lRedIndex
End If

'An infinite loop
Do While True
   'Add the next in order of length segment to the result
    sngResult = sngResult + sngCurr(lCurrIndex)
    Debug.Print Format(sngCurr(lCurrIndex), "0.0000") & _
            IIf(iColor = ic_RED, " r", " b")

   'Placing of the alternative segments on the basic segment
   If Not CoverSegment(sngCurr(), sngOpp(), lCurrIndex, lOppIndex, iColor)
Then
      'EXIT the infinite loop
      Exit Do
   End If

    'Go to the next segment
    lCurrIndex = lCurrIndex + 1

   'If there is no segments in the current array –
   'EXIT the infinite loop
    If lCurrIndex > UBound(sngCurr) Then  Exit Do

   'Change the color if it is necessary
    If sngOpp(lOppIndex) > sngCurr(lCurrIndex) Then
      Select Case iColor
       Case ic_RED
         'Reset the indexes
          lRedIndex = lCurrIndex: lBlueIndex = lOppIndex

         'Set the current and alternative values
          iColor = ic_BLUE
          sngCurr = sngBlue: lCurrIndex = lBlueIndex
          sngOpp = sngRed:   lOppIndex = lRedIndex
```

```
      Case ic_BLUE
        'Reset the indexes
         lBlueIndex = lCurrIndex: lRedIndex = lOppIndex

        'Set the current and alternative values
         iColor = ic_RED
         sngCurr = sngRed:  lCurrIndex = lRedIndex
         sngOpp = sngBlue:  lOppIndex = lBlueIndex
    End Select
  End If
Loop

'Add the segments not placed yet if there are any
For i = lCurrIndex + 1 To UBound(sngCurr)
   sngResult = sngResult + sngCurr(i)
   Debug.Print Format(sngCurr(i), "0.0000") & _
          IIf(iColor = ic_RED, " r", " b")
Next i
For i = lOppIndex To UBound(sngOpp)
   sngResult = sngResult + sngOpp(i)
   Debug.Print Format(sngOpp(i), "0.0000") & _
          IIf(iColor = ic_RED, " r", " b")
Next i

'Output the result
Debug.Print "Result:": Debug.Print Format(sngResult, "0.0000")
End Sub

'A function for placing of segments on the basic segment
Public Function CoverSegment(sngCurr() As Single, sngOpp() As Single, _
                 lCurrIndex As Long, lOppIndex As Long, _
                iColor As Integer) As Boolean
   Dim sngRest As Single

   'The difference between the lengths of basic segment and
   'the placed on it segments of alternative color
   sngRest = sngCurr(lCurrIndex)

   'Exit function if there are no segments in the alternative array
   If lOppIndex > UBound(sngOpp) Then
      CoverSegment = False
     Exit Function
   End If

   'While the length of the noncovered part is greater than
   'the length of the next segment of alternative color
   Do While sngRest >= sngOpp(lOppIndex)
      'The length of the noncovered part
      sngRest = sngRest - sngOpp(lOppIndex)
      Debug.Print , Format(sngOpp(lOppIndex), "0.0000") & _
             IIf(iColor = ic_RED, " b", " r")
```

```
      'Go to the next segment
       lOppIndex = lOppIndex + 1

      'Exit function if there are no segments in the alternative array
       If lOppIndex > UBound(sngOpp) Then
          CoverSegment = False
         Exit Function
      End If
    Loop

    CoverSegment = True
End Function
```

Sample Output:

Red:	Blue:		Basic:	Arranged:
0.4636	0.3458	|	0.4636 r	0.3458 b
0.4486	0.2168	|	0.4486 r	0.2168 b
0.0476	0.2132	|		0.2132 b
0.0402	0.1966	|	0.1966 b	0.0476 r
	0.0188	|	0.1966 b	0.0476 r
	0.0088	|		0.0402 r
		|		0.0402 r
Result:		|	0.0188 b	
1.1364		|	0.0088 b	

References

1. Honsberger, R. Mathematical Gems II, Vol. 2. Mathematical Association of America, 1976.
2. Honsberger, R. Mathematical Gems III, Vol. 9. Mathematical Association of America, 1985.
3. Knuth, D. The Art of Computer Programming, Vol. 1-3. Addison-Wesley, 1999.
4. Riordan, J. Introduction to Combinatorial Analysis, New York, Wiley, 1998.

О Greedy алгоритме

Красимир Пенев, Петар Петров

Academy for the Advancement of Science and Technology, USA
***Senior System Analyst, USA*

(Резюме)

Статья с научно-приложным характером и исследует самые применения Greedy алгоритма как метод решения комбинаторных проблем. Представлены решения двух задач. Одна из них представляет рациональные числа при помощи Египетских дроб, а другая – для арранжировки отрезков. Решения задач являются доказательством корректности предложенных алгоритмических процедур.