Session: Logo & Mathematics

2005-08-31 PM

Mathematics Logo Style

Andrzej Walat

OEIIZK

Ul. Nowogrodzka 73, PL 02-06, Warsaw, Poland

andrzej@oeiizk.waw.pl

Abstract

"Logo has been the victim of its own success in the elementary schools. It has acquired a reputation as a trivial language for babies" Brian Harvey (1986). In this paper I take the position at the opposite extreme. I intend to show two examples of compelling problems appropriate for elder and ambitious upper secondary school students.

Keywords

Discrete mathematics, problem solving, dynamic programming, probability distribution, Logo

1. Introduction

There are many imposing examples of great achievements of using Logo while working with small children but relatively not so many suggestions what to do with elder students. In my opinion this imbalance in not good for Logo and, what is more important not good for the education of young people. This is why I have decided to develop a series of didactical materials for upper secondary students. In the frame of the European project CoLabs I have developed three educational microworlds (Walat 2005 a, b, c) comprised of printed materials – activity books – and Imagine projects, which offer upper secondary students a variety of activities helping them to understand and appreciate mathematics.

In this paper I am going to present only two of a countless variety of interesting mathematical problems which ambitious upper secondary students can successfully cope with by using Logo environment, and which are practically unmanageable with only pencil and paper.

2. The first example – finding the maximum sum

The first example is only a slight modification of the problem from the VI International Olympiad in Informatics (Stockholm 1994)

2.1. Problem

Look at the board (see Figure 1). The pawn starts wandering down the triangle from the top field. In each step it goes down left or right and, while wandering, it adds the numbers on the fields.

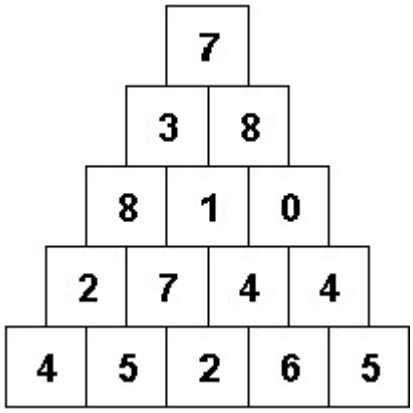

Figure 1. The example of a number triangle

Example: If the pawn chooses the way `rlll` (which means down right, down left, down left, down left), it will get 7 + 8 + 1 + 7 + 5 = 28 points altogether.

Find the maximum number of points that the pawn can get.

2.2. Solution

It seems to be a rather complex problem. In our exemplary triangle there are 16 different paths from the top to the bottom. However, in the case of a bigger triangle we would have an astronomical number of sums to compute. Are there any strategies which lead to the correct answer without long and tedious calculations? Yes there are. We can reduce the complexity of the problem by replacing a given triangle by its **reduct** – another smaller triangle with the same maximum sum of points. Look at the figure 2. To get a reduct of a complex triangle we replace the first two bottom lines by their **fusion**. In our example we remove the first line in the triangle and replace the first number in the second line from the bottom (it is number 2) by the maximum number of points we can get on the way down from this field (that is by 2 + 5 = 7). We replace the second number in the line (it is number 7) by 7 + 5 = 12, and so on.

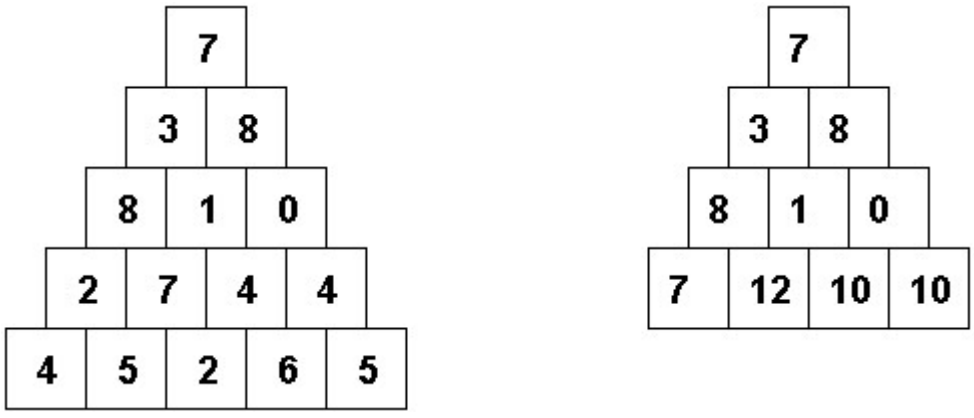

Figure 2. The first step of reduction

After the next steps of reduction we will get the triangle composed of only one field and the number in the field is the eventual output.

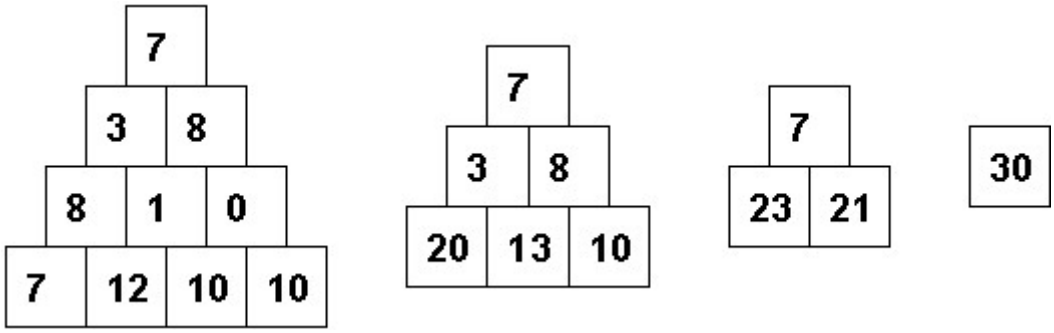

Figure 3. The next three steps of reduction

In Logo we can define a function which counts the maximal sum for a given triangle in the following way :

```
to maxsum :triangle
if atom? :triangle [output onlynumberin :triangle]
output maxsum reduct :triangle
end
```

In ordinary English we could say:

- If a given number triangle is an atomic one (the simplest triangle having only one number), then the output equals the only number in the triangle.
- Otherwise, reduce the given triangle and the maxsum of this reduct will be the outcome.

Now we have to define auxiliary procedures: `reduct`, `fusion`, `atom?`, and so on. We decided to represent number triangles as lists of lines (starting from the bottom). So our exemplary triangle will be represented by a list:

```
[[4 5 2 6 5][2 7 4 4][8 1 0][3 8][7]].
  to reduct :triangle
     op fput fusion item 2 :triangle item 1 :triangle bf bf :triangle
  end
  to fusion :l1 :l2
     if empty? :l1 [op []]
     op fput (first :l1) + (max item 1 :l2 item 2 :l2) fusion bf :l1 bf :l2
  end
  to atom? :triangle
     op count :triangle = 1
  end
  to onlynumberin :traingle
     op first first :triangle
  end
  to max :a :b
     op ifelse :a > :b [:a][:b]
  end
```

Now we can test our solution:

```
?print maxsum [[4 5 2 6 5][2 7 4 4][8 1 0][3 8][7]]
30
```

It works.

3. The second example

The second problem was posed to me by my friend – who is a physicist – Jan Dunin Borkowski.

3.1. Problem

There are 16 provinces in Poland and 16 towns which are their regional capitals. We will draw 16 times one of those 16 towns. What is more probable: event A – that the number of towns which will be chosen at least once will be greater than 8 or the opposite event B – that this number will be less or equal to 8?

3.2. The first solution

Let $P(t,d,s)$ denote the probability that after a given number d of drawings one of the given number t of towns, s different towns will be selected. We assume that t and d can be any nonnegative whole numbers, and that the third number s must not be greater than either t or d. Other data are not acceptable.

We can distinguish three cases:

1. The number of drawings $d = 1$. In this case, s must also be equal to 1 and so $P(t,d,s) = 1$.
2. $d > 1$ and s = 1, in this case: $P(t,d,s) = \frac{1}{t} * P(t,d-1,s)$.
3. In the third case, the value of $P(t,d,s)$ can be obtained by adding two numbers: $P(t,d-1,s-1) * \frac{t+1-s}{t}$ and $P(t,d-1,s) * \frac{s}{t}$ if $s < t$ or 0 if $s = t$.

All we have to do in order to get an executable definition is to write it in Logo.

```
to p :t :d :s
   if :d = 1 [op 1]
   if :s = 1 [op (p :t :d - 1 :s) / :t]
   op (p :t :d - 1 :s - 1) * (:t + 1 - :s) / :t
    + ifelse :s = :d [0][(p :t :d - 1 :s) * :s / :t]
end
```

Now we can use the computer to calculate the probabilities that after drawing one of the $t = 4$ towns $d = 4$ times, we will have selected $s = 1$, 2, 3 and 4 different towns.

```
?repeat 4 [pr se repc p 4 4 repc]
[1 0.015625]
[2 0.328125]
[3 0.5625]
[4 0.09375]
```

We can see that the probability of event *A* that the number of selected towns will be greater than 2 (half of all the towns) is much greater than the probablity of the opposite event – *B* that no more than half of the towns will be selected. This is also the case if the number of towns *t* = 16. We can check it by writing the command:

```
?repeat 16 [pr se repc p 16 16 repc]
```

We have built a simple and useful tool for answering a whole range of similar questions. But it has one disadvantage. If we choose a greater number of towns like *t* = 30, we will have to wait for the result of calculations ubelievably long.

3.3. The second solution

If we draw 4 times one of the 4 towns, then the number of towns selected at least once may equal to 1, 2, 3 or 4. It is a random variable. Let us denote it by $S_{4,4}$. The list of pairs which we have got as a result of the command: `repeat 4 [pr se repc p 4 4 repc]` represents the probability distribution of this variable, but we may prefer a shorter form of representation and use a list of numbers (instead of a list of pairs of numbers). Using ordinary fractions instead of decimal numbers we can write down the probability distribution of the random variable as a list of fractions [1/64 21/64 36/64 6/64].

Let us denote by $S_{4,3}$ the number of towns selected at least once after drawing one of the 4 towns three times. It can be calculated even by hand that this random variable has the following probability distribution: [1/16 9/16 6/16]. Let us look at how we could calculate the probability distribution of the random variable $S_{4,4}$ having given the probability distribution of the $S_{4,3}$.

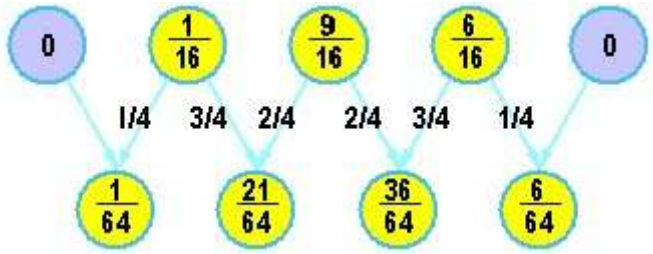

Figure 4. Calculating the next probability distribution

We can see that the list/vector [1/64 21/64 36/64 6/64] is a sum of two vectors: *V1* = [0 ¾ * 1/16 2/4 * 9/16 ¼ * 6/16] and *V2* = [1/4 * 1/16 2/4 * 9/16 ¾ * 6/16 0].

How could we calculate *V1* and *V2*? We can start with multiplying each element of a given vector *GV* = [1/16 9/16 6/16] by its position. Let us denote this vector operation by `pte` – position times element. Next, we divide the outcome by the number of towns *t* = 4 to get an auxiliary vector *AV* = [1/4 * 1/16 2/4 * 9/16 ¾ * 6/16]. We get *V2* if we put 0 after the last element of *AV*. We get *V1* if we put 0 before the first element of a difference of two vectors *GV* – *AV*.

Now we are ready to define a general operation which for a given probability distribution of the random variable $S_{t,d-1}$ returns the probabilty distribution of the $S_{t,d}$ variable in the case when $d \leq t$.

```
to nexta :t :gv
let "av pte :gv / :t
let "v1 fput 0 :gv - :av
let "v2 lput 0 :av
```

```
op :v1 + :v2
end
to pte :v
if empty? :v [op []]
op lput (last :v) * (count :v) pte bl :v
end
```

Let us check it:

```
?show nexta 4 (list 1/16 9/16 6/16)
[0.015625 0.3281125 0.5625 0.09375]
```

It works.

If we want to get a probability distribution of the random variable $S_{t,d}$ having given the probability distribution of $S_{t,d-1}$ for the case $d > t$, we have to apply another operation:

```
to nextb :t :gv
let "av pte :gv / :t
op (fput 0 bl (:gv - :av)) + :av
end
```

Let us check it:

```
?show nextb 4 [0.015625 0.3281125 0.5625 0.09375]
[0.00390625 0.175775 0.585931 0.234375]
```

Having `nexta` and `nextb` procedures we can define a general solution – an operation `pds` which returns the probability distribution of the random variable $S_{t,d}$ for any positive whole numbers t and d.

```
to pds :t :d
if :d = 1 [op [1]]
op ifelse :d > :t [nextb :t pds :t :d-1][nexta :t pds :t :d-1]
end
```

Let us check it:

```
?show pds 4 5
[0.00390625 0.175775 0.585931 0.234375]
?show time show pds 50 50 show time
[10 32 58 699]
[5.6295E-84 1.55287E-67 1.58423E-57 3.28694E-50 2.11861E-44 1.44517E-39
2.01619E-35 8.54047E-32 1.41776E-28 1.09769E-25 4.48626E-23 1.0602E-20
1.55219E-18 1.48508E-16 9.68541E-15 4.454E-13 1.48456E-11 3.66849E-10
6.84794E-9 9.80865E-8 1.09226E-6 9.56041E-6 6.63842E-5 0.000368488
0.00164553 0.00594237 0.0174245 0.041616 0.0811358 0.129295 0.168482
0.179446 0.156004 0.110446 0.063464 0.029465 0.010989 0.0032681
0.000767942 0.000140961 1.99261E-5 2.13077E-6 1.68484E-7 9.56454E-9
3.74759E-10 9.59659E-12 1.48261E-13 1.21649E-15 4.1948E-18 3.42432E-21]
[10 32 58 759]
```

We have got the solution after less than 0.1 sec, not after 1000 years.

In our first solution we operated on numbers, in the second one – on list representations of the probability distributions of the random variables. Is it this that makes the difference?

3.4. The third solution

Having a computer – a tool which is so quick, we can try to get probability distributions of relevant random variables in an experimental way. Let us define an operation `nst` which returns the number of selected towns. We assume that towns are numbered from 1 to *t*. These numbers will serve us as names of towns and at the same time as names of relevant variables. We will use here a nice feature that numbers can not only be the values of variables but also their names.

```
to nst :t :d
repeat :t [let repc 0]
repeat :d [make 1 + random :t 1]
let "nt 0
repeat :t [make "nt :nt + thing repc]
op :nt
end
```

The first instruction creates *t* variables with the names 1, 2, 3, ..., *t* and value 0. It means that before we start the drawing, each town has been selected 0 times. Next, we start a series of *d* drawings. We draw a number between 1 and *t* (the name of the town) and assign value 1 to the variable having the same name (which means that the town has been chosen at least once). Eventually, after finishing the drawing we add values of all variables (i.e. we count all selected towns).

Now we will define an operation `efd` – empirical frequency distribution, which returns the frequency distribution of the number of selected towns which we get after a given number of experiments *ne* of drawing one of the given number of *t* towns *d* times.

```
to efd :ne :t :d
repeat :t [let repc 0]
repeat :ne [inc nst :t :d]
let "lf []
repeat :t [make "lf lput thing repc :lf]
op :lf / :ne
end
```

This is a relatively short definition but it needs explaining. For the second time we have decided to use the numbers (between 1 and *t*) as names of local variables. This time a variable named *s* will serve as a counter of frequency – how many times we have selected *s* different towns. The first command creates all these variables and assigns them the value 0. Next, we repeat a given number *ne* of times an experiment of drawing *d* times one of the given number *t* of towns, and we increase the variable whose name is identical with the number of selected towns – if we selected 3 towns, we increase the variable named 3. After finishing all experiments we glue the values of all these variables (from 1 to *t*) into one object – the list of frequency. We get the output by dividing this list (this means each element of the list) by the number of experiments.

Let us compare this empirical frequency distribution with the relevant theoretical probability distribution, for an exemplary, relatively small case $t = d = 4$:

```
?show efd 1000 4 4 show pds 4 4
[0.015 0.322 0.569 0.094]
[0.015625 0.3281125 0.5625 0.09375]
```

The difference is insignificant. In big cases it is easier to compare two graphs instead of lists of numbers. For our purposes even a very simple procedure of drawing a graph of a given probability distribution will do.

```
to drawgraph :pds
let "l (count :pds) - 1
let "step 500 / :l
pu setpos se -500 1000 * first :pds pd
repeat :l [setpos se xcor + :step 1000 * item repc + 1 :pds]
end
```

Now if we write in the command line:

```
?cs setpw 5 setpc "black drawgraph pds 50 50
```

and next:

```
?cs setpw 3 setpc "red drawgraph efd 1000 50 50
```

we will see two graphs on the screen: the thick black graph of the theoretical probability distribution of the random variable $S_{50,50}$ and the thin red graph of the relevant empirical distribution.

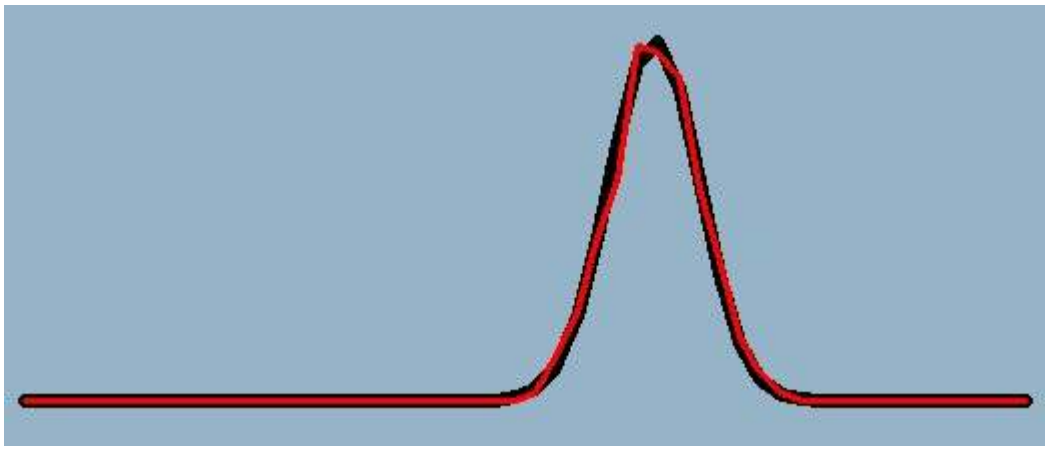

Figure 5. The theoretical and empirical distribution of the random variable $S_{50,50}$

4. Final remarks

4.1. The two quotations

Let us quote two remarks, the first one from the introduction to the *Concrete Mathematic*s, Graham at al (1989).

"One of the present authors had embarked on a series of books called The Art of Computer Programming, and in writing the first volume he (DEK) had found that there were mathematical tools missing from his repertoire; the mathematics he needed for a thorough, well-grounded understanding of computer programs was quite different from what he'd

learned as a mathematics major in college. So he introduced a new course, teaching what he wished somebody had taught him" (Graham at al 1989).

The second quotation comes from Wolfram (2002)

"…there are actually a vast range of abstract systems based on simple programs that traditional mathematic has never considered. And because these systems are in many ways simpler in construction than most traditional systems in mathematics it is possible with appropriate methods in effect to go farther in investigating them."

4.2. Conlusions

I believe that in the 21st century not only the authors of academic textbooks but also ordinary educated people will have to cope with situations when they need to use mathematics that they have not been taught. They will have to be able to invent it.
A great range of mathematics of tomorrow – non existent in actual school curricula – will be based on procedures instead of traditional equations. This is a very important kind of mathematics and it should not be more difficult (in fact it may be much simpler) than traditional mathematics. The only reason why it is absent from current curricula is that it is unmanageable without appropriate tools like Logo or any comparable environment.

The solution of the first problem and the second solution of the second problem are based on the idea of dynamic programming, which belongs both to the computer science and mathematics. Breaking barriers between disciplines is always a sign of revolution.

References

Graham R.L., Knuth D.E., Patshnik O. (1989). *Concrete Mathematics. A Foundation for Computer Science*. Addison-Wessley

Harel B. (1986). *Computer Science Logo Style, Volume 1: Intermediate Programming*. The MIT Press.

Walat A. (2005a) *Algebra of games* (to appear).

Walat A. (2005b) *The Laboratory of Randomness* (to appear).

Walat A. (2005c) *Visual modeling with Imagine Logo* (to appear).

Wolfram S. (2002). *A New Kind of Science*. Wolfram Media, Inc.