Teaching Recursion With Counting Songs

Greg Michaelson
Heriot-Watt University
Riccarton, EH14 4AS, Scotland
G.Michaelson@hw.ac.uk

Abstract

Recursion is often seen as an advanced topic, unsuitable for early learners of computing. Here, we explore the use of simple counting songs to draw out patterns and abstractions for recursion, which are more widely applicable to solving problems involving sequences.

Author Keywords

recursion; computational thinking; patterns; abstraction; counting songs

ACM Classification Keywords

K.3.2. Computer and Information Science Education – Computer Science Education

Introduction

Recursion is a fundamental Computing technique but is often thought of as a scary, advanced topic. It is rarely covered at school level. Even in first level undergraduate text books on imperative programming, recursion is often only treated briefly in a later chapter.

But recursion is really straightforward. A recursive sub-program is one that calls itself. That's all.

Certainly, many people who are well used to iteration seem to find this characterization of recursion as somehow unnatural. Part of the difficulty may lie in the misleading conception that recursion involves defining something in terms of itself. While this is not wholly inaccurate, it does give the impression that recursion is some sort of trick: isn't defining something in terms of itself a tautology?

Another source of recursion's poor reputation is the poor choice of text book examples, typically numerical, through which it is often introduced. Indeed, many students may come to believe that the greatest common divisor, and the factorial and Fibonacci sequences, were dreamed up solely to illustrate an otherwise pointless technique.

If, however, we view recursion in the light of computational thinking(CT)[1] we can identify appropriate recursive patterns and abstractions. Then, recursion can come to feel as natural as iteration, once we know how to locate these patterns in concrete scenarios and introduce the corresponding abstractions.

In practice, we can use recursion anywhere we can find a sequence with some well specified property between items. So, here, rather than looking at boring old sequences of numbers, let us now think about how we can find recursion in children's songs that involve counting. And rather than starting with a formal definition of recursion, we will:

- analyse a first song – *Ten Green Bottles* - to tease out a recursive pattern;
- use our understanding of the recursion in the first song to analyse a second song – *One Man Went To Mow*.

Ten Green Bottles

Ten Green Bottles [2] is a well known English-language children's song. Here, though, we will look at the more succinct *3 Green Bottles*:

3 green bottles, hanging on the wall.

3 green bottles, hanging on the wall.

And if 1 green bottle should accidentally fall,

There'd be 2 green bottles, hanging on the wall.

2 green bottles, hanging on the wall.

2 green bottles, hanging on the wall.

And if 1 green bottle should accidentally fall,

There'd be 1 green bottle , hanging on the wall.

1 green bottle, hanging on the wall.

1 green bottle hanging on the wall.

And if 1 green bottle, should accidentally fall,

There'd be 0 green bottles hanging on the wall.

If we stand back, we can see that to sing the *song* about 3 green bottles, we:

- sing a *verse* about the 3 green bottles;
- sing the *song* about 2 green bottles.

To sing the *song* about 2 green bottles we:

- sing a *verse* about the 2 green bottles;
- sing the *song* about 1 green bottle.

And to sing the *song* about 1 green bottle we:

- sing a *verse* about the 1 green bottle;
- stop.

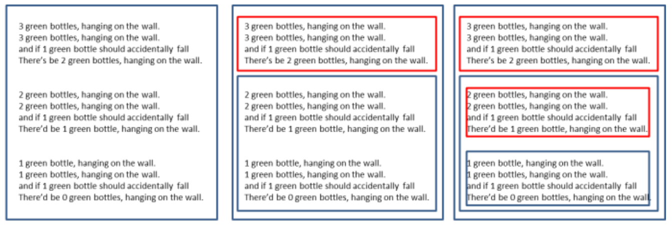
Now, in general, to sing the *song* about *N* green bottles, where *N* is bigger than 1, we:

- sing a *verse* about the *N* green bottles;
- sing the *song* about *N*-1 green bottles.

Then if *N* is 1 then we:

- sing a *verse* about the 1 green bottle;
- stop.

Thus, recursion is as much about *nesting* as *sequencing*. We can see this for the first two verses of *3 green bottles* in Figure 1:


3 green bottles, hanging on the wall.
3 green bottles, hanging on the wall.
and if 1 green bottle should accidentally fall
There's be 2 green bottles, hanging on the wall.

2 green bottles, hanging on the wall.
2 green bottles, hanging on the wall.
and if 1 green bottle should accidentally fall
There'd be 1 green bottle, hanging on the wall.

1 green bottle, hanging on the wall.
1 green bottles, hanging on the wall.
and if 1 green bottle should accidentally fall
There'd be 0 green bottles, hanging on the wall.

a) song for 3 bottles is...

b) verse for 3 bottles, + song for 2 bottles which is...

c) verse for 2 bottles + song for 1 bottle

Figure 1: recursive structure of *3 green bottles*

where a blue box shows a *song* and a red box shows a *verse*. At each level, we sing a *verse* and then sing the *song* at the nested level.

We have defined singing about *N* bottles in terms of singing about *N*-1 bottles. Each time, *N* gets smaller and smaller until there's nothing more to do.

Technically, we have found a:

- *recursion case*, where we deal with the first item and then apply the whole process to all the other items; and a…
- *base case*, when there's only one item, or indeed no items, left and we stop.

In computational thinking terms[3], we have:

- found a recursive *pattern* in the song;
- *generalized* the pattern to make a song *abstraction*.

One Man Went To Mow

Next, let us apply the same style of reasoning to the irritatingly non-terminating song *One Man Went To Mow*[2], which is redolent for me of excessively long car journeys as a child:

1 man went to mow, went to mow a meadow.

1 man and his dog, went to mow a meadow.

2 men went to mow, went to mow a meadow.

2 men, 1 man and his dog, went to mow a meadow.

3 men went to mow, went to mow a meadow.

3 men, 2 men, 1 man and his dog, went to mow a meadow.

etc

We will apply our reasoning to the song by looking for a pattern involving doing something with *N* men and then doing something with *N*+1 men. Thus, to sing a *song* about *N* men going to mow, we:

- sing a *verse* about *N* men going to mow;
- sing a *song* about *N*+1 men going to mow.

Once again we have used recursion to define singing about *N* men but this time in terms of *N*+1 men.

Technically, we have found a:

- *recursion case*, where we deal with the current number of items and then apply the whole process to an increasing number of items.

But we have no algorithmic way of stopping the song so this time we have no *base case*.

We can also see that the second line of each verse contains a sub-verse that counts down, a bit like for *Ten Green Bottles*.

So, to *count down* from *N* men we

- *mention N* men;
- *count down* from *N*-1 men.

and for one man we

- *mention* him and his dog;
- stop.

Here, we have found recursive generalisations for both the song and the verse.

Related Work

Greg Michaelson is Professor of Computer Science at Heriot-Watt University in Edinburgh, Scotland. His research interests are in the design and implementation of programming languages, and in the teaching of programming[8]. Most recently, he has co-authored the language-neutral *Haggis* Reference Language which is being used by the Scottish Qualifications Agency for the setting of all national examinations in Computing Science. Greg is a Fellow of the British Computer Society and an elected member of the Board of the European Association for Programming Languages and Systems.

Eisenberg[3] presents a lighthearted approach to recursion in a CT context through self-referential humour.

Shelly[4] discusses three traditional approaches to teaching recursion: as metaphors, through classic mathematical problems and via visual/kinesthetic methods such as program tracing.

There is a long standing and substantial literature on teaching recursion in the Computer Science curriculum In particular, Turbak et al[5] advocate the teaching of recursion prior to iteration, in the context of functional programming as a first approach, but through classic algorithms.

Recently there has been growing interest in teaching recursion through video games. For example, Chaffin et al[6] see recursion as "a rather notorious subject in the computer science field". They present the use of an interactive game to motivate recursion, through binary tree traversal. Lee et al[7], also use a video game but for a wider gamut of recursive techniques.

Conclusion

Let us conclude by considering the workshop questions:

1. *What is the relationship between programming and computational thinking, and are there any trade-offs regarding which should be the primary educational focus?*
2. *How can we design programming tools and curricula that are developmentally appropriate and foster motivation throughout childhood?*

First of all, I think that problem solving with CT should precede programming. CT is an excellent medium for arriving at programs, but the initial focus should be on analyzing problems in different domains because CT techniques are so widely applicable.

Secondly, I think that the types of problems we use in the early teaching of programming lack motivation, both for young persons and their teachers, and that by applying CT in unusual areas like music we can better engage with them. In particular we can more effectively convey both the challenge and fun of problem solving and programming.

Finally, I think that recursion is a natural way to characterize problems involving sequences and that, with instructor confidence, it can gainfully be taught at an early stage.

References

[1] E. Kao, Exploring Computational Thinking at Google, *CSTA Voice*, Vol 7, Issue 2, May, 2011, p6.

[2] *BBC - School Radio – Counting Songs* http://www.bbc.co.uk/schoolradio/subjects/mathematics/countingsongs - inspected 19/3/15.

[3] M. Eisenberg, Recursion – or, Better Computational Thinking Through Laughter, *International Journal of Computers for Mathematical Learning*, 13:171–174, 2008.

[4] T. Shelley, An Evaluation of Instructional Methods for Teaching Recursion, Stirrings, California State University Stanislaus, May 2009. https://www.csustan.edu/honors/stirrings - inspected 20/3/15

[5] F. Turbak, C. Royden, J. Stephan, and J. Herbst, Teaching Recursion Before Loops In CS1, *Journal of Computing in Small Colleges*, Volume 14, Number 4, pp 86-101, May 1999.

[6] A.Chaffin, K. Doran, D. Hicks, and T. Barnes, Experimental Evaluation of Teaching Recursion in a Video Game, *ACM ITiCSE'08*, Month 1–2, Madrid, Spain, 2009.

[7] E. Lee, V. Shan, B. Beth and C. Lin, A Structured Approach to Teaching Recursion Using Cargo-Bot, *ACM ICER'14*, August 11-13, Glasgow, Scotland, UK, 2014.

[8] G. Michaelson, Teaching Programming with Computational and Informational Thinking, *Journal of Pedagogic Development*, Vol. 5, Issue 1, pp51-65, March 2015.