Clever Stuff For Common Problems

Going beyond simple algorithms

Trainer's Notes

Background

At primary school, computational concepts will often be introduced by way of analogy with ideas children are familiar with. Their first encounters with algorithms will be human instructions, perhaps to make a jam sandwich, or guide a child through an obstacle course. Programming constructs, such as repetition, might be illuminated by reference to nursery rhymes or similar repeating patterns. The key point is that familiarity with the idea is a pre-requisite to any later application in a programming environment.

By the end of Key Stage 3 children will have had experience creating programs that involve combinations of the 'big 3' constructs: sequence, selection and repetition. They will be familiar with the notion of a variable and possibly had practical experience of manipulating data in lists (or arrays). But for most students, this will be the limit of their practical application of algorithms.

This unit starts by looking at ways to consolidate list manipulation but suggests that at Key Stage 3 we ought also to be opening the children's eyes to some of the algorithms that help shape the world around them. These often depend on data structures that are more complex than lists. We wouldn't expect children to be capable of implementing these in a programming environment at KS3, but by gaining familiarity with the concepts and how they apply to solving real world problems, we broaden their horizons about the possible applications that can be developed by applying clever algorithms.

The aim of the day

The aim of the unit is to build appreciation of a variety of data structures which lend themselves to storing different information. The key structure and terminology teachers need to become familiar with is a graph (which can be thought of as a generalisation from the list they will be familiar with). Graphs can represent widely differing problems, but those problems may have common solutions. So if we can abstract the key fundamentals from a problem, we may be able to apply it to another different situation. Throughout the unit, there is an emphasis on how we can apply abstraction and decomposition to address everyday challenges.

Exploring some common 'graph traversals' in the activities illustrates how some are very hard to solve, whilst others, which look very similar are easy to develop algorithms for. The aim is to discover well known algorithms (such as Prim's and Kruskal's algorithms for finding a Minimum Spanning Tree) and illustrate some classic problems (such as the Travelling Salesman problem). Through these insights it is hoped a sense can be conveyed of how solutions to some problems don't scale as well as developing an appreciation of their role in providing solutions to common problems.

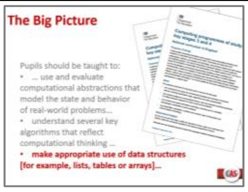
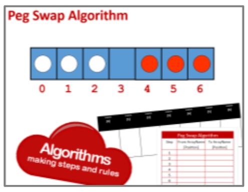
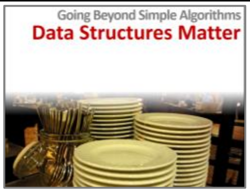
Throughout the material there are references to famous computer scientists, or classic books. There are lots of pointers to other material. The aim is to encourage colleagues to delve deeper and take ideas further.

Before delivering the unit, please check you are comfortable with the narrative and references to other material. These teacher's notes include a summary of each activity. Ensure you rehearse the delivery to familiarise yourself with transitions and animations. The slides include further detailed notes.

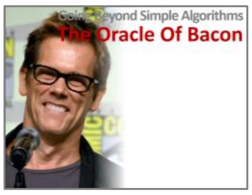
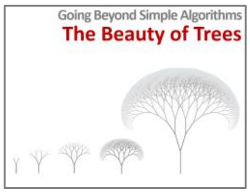
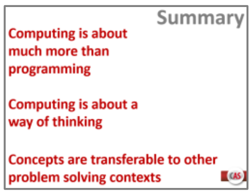
There are lots of exercises and supplementary material. The pace should be fast, with the assumption that the audience are experienced teachers, probably already teaching to GCSE level, with some familiarity with the concepts of Computer Science. As such they will probably not need to work through every activity in full. Sometimes it will be sufficient to only part complete an activity so teachers 'get it' and can see how it might be used. Judgement is required and the timings below are indicative, to help with planning. Always be flexible and encourage discussion and engagement. Details of each activity are given in the teachers notes. Further guidance on the narrative, slide transitions and animation can be found in the slide notes.

Indicative Timetable

The trainer's presentation is broken down into 8 sections:

 The Big Picture Pupils should be taught to: • … use and evaluate computational abstractions that model the state and behavior of real-world problems… • understand several key algorithms that reflect computational thinking … • make appropriate use of data structures [for example, lists, tables or arrays]… 20 minutes	Sets the subject matter in the context of the National Curriculum (5 mins). Illustrates a classroom activity (Peg Puzzles) which demonstrates an unplugged approach to developing an algorithm for list (array) manipulation (15 mins).  Peg Swap Algorithm 0 1 2 3 4 5 6 Algorithms making steps and rules
Going Beyond Simple Algorithms A Perfect Shuffle 40 minutes	Reinforces the importance of list manipulation. Introduces 3 ways to shuffle as a list manipulation exercise (10 mins). Investigates some surprising qualities of a perfect riffle shuffle (20 mins). Introduces a magic trick card force (5 mins).
 Going Beyond Simple Algorithms Data Structures Matter 35 minutes	Sets algorithms and data structures in the context of problem solving. Quickly illustrates a range of common data structures (5 mins). Introduces a classroom activity (Spit Not So) to illustrate how the choice of data structure affects the ease with which a problem is solved (5 mins). Main classroom activity contains two practical exercises, Knights Tour and Tour Guide. They take the ideas of data structures further, introducing the concepts of abstraction and a graph. The activity illustrates how two different problems can share an abstract representation and a common solution, introducing a common graph traversal – a Hamiltonian cycle (25 mins total).
Going Beyond Simple Algorithms Many Problems One Solution 60 minutes	Introduces 'the poor cartographers problem', representation as a graph, an algorithm to solve it and its application to a real world problem. Quibble, Crouch, Cornerstone and Quoink (15 mins) is an unplugged outdoor exercise to introduce map colouring. The Poor Cartographer formalises the previous activity, poses the question of the minimum number of colours required for map colouring. It considers how to represent countries as a graph, a way of representing the graph as a list of nodes, each with list of connections (an adjacency list) and explains the algorithm for determining the answer. (30 mins) Mobile Base Stations considers a real world problem. (15 mins)

COMPUTING AT SCHOOL
EDUCATE · ENGAGE · ENCOURAGE
Part of BCS, The Chartered Institute for IT

Going Beyond Simple Algorithms Toy Problems For The Real World 35 minutes	An activity to discover a famous algorithm, which is used to draw out the differences and similarities between several classic graph algorithms. Muddy City (15 mins) allows attendees to discover Prim's (an intuitive greedy) algorithm for themselves, introducing the idea of a minimum spanning tree. The Steiner Tree interlude (10 mins) can rely on the video explanation but is more impressive as a physical demonstration, requiring props made in advance. The supporting Create Maths worksheets (10 mins) are for class use, but need to be distributed and 'talked through'. They make connections with real world problems whilst distinguishing between minimum spanning trees, Hamiltonian cycles, the travelling salesman problem and Eulerian circuits. This session introduces the notion of tractable and intractable problems. Careful study is required beforehand. Ends with a quick promotion of Nrich maths materials.
 Going Beyond Simple Algorithms The Oracle Of Bacon 40 minutes	A fun activity, requiring internet access (10 mins) introduces shortest path problems and builds to explaining breadth first searches. It can be explained without internet connectivity for attendees if needed. Finding the shortest path is a paper activity (20 mins) aimed at articulating an algorithm, with a following explanation (10 mins) and link to real world problems. Ends with a quick promotion of the Bebras competition.
Class based research Why? COMPUTING AT SCHOOL CAS Research 10 minutes	A short discussion to promote classroom research and encourage reflective practice. Draw out suggestions for potential research areas and mention possible techniques. Ends with a quick promotion of the BCS Certificate in Computer Science Teaching
 Going Beyond Simple Algorithms The Beauty of Trees 20 minutes	Summarises how lists, trees and graphs are part of a family of structures. Exercise (Think Of A Number) focuses on constructing a binary tree. An in-order traversal retrieves the data in sorted order.
 Summary Computing is about much more than programming Computing is about a way of thinking Concepts are transferable to other problem solving contexts 5 minutes	Emphasise computational thinking. Distribute any materials and discuss ways to deliver smaller presentations. Mention further reading.

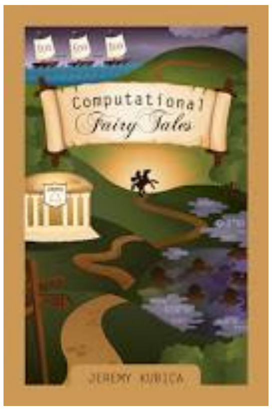
Above all else, remember that the aim is to empower attendees to offer similar sessions to colleagues. It should be inclusive, enjoyable and embody the CAS ethos of collegiality: There is no 'them', only us!

When someone books to attend the training session, send a prompt acknowledgment informing them when final confirmation and further details will be sent. Set a cut-off date, at which point you decide if there are enough bookings to make a viable session.

Once you have enough people booked, contact them again with brief details and suggested prior reading. Although not essential, by suggesting some prior reading you are indicating that this is in depth CPD which requires some commitment on the part of the attendees. It also gives you a chance to establish some dialogue with attendees prior to the event. With a week to go, you could mail a reminder and enquire about the reading and whether it would be useful for teaching. This helps keep the attendees focused on the event.

Prior Reading

Computational Fairy Tales is a book by Jeremy Kubica. It started life as a regular blog, where Jeremy introduced computer science concepts through short fairy tales. They are an excellent stimulus for ways to make computer science ideas accessible to KS3 children. The blog archive is available online: http://computationaltales.blogspot.co.uk. Each story takes less than 5 minutes to read. To get a flavour of the material please could you read the three stories below. All use the URL above, with only the path shown below:



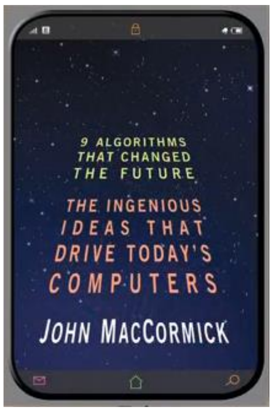
Swapping Array Values and the Swimmy Friends Pet Store:
/2011/06/swapping-array-values-and-swimmy.html
Arrays, Linked Lists, and Zed's Coffee Shop:
/2011/04/arrays-linked-lists-and-zeds-coffee.html
Ushers, Peanut Vendors, and Matrix Indices:
/2011/07/ushers-peanut-vendors-and-matrix.html

If you like what you read, do browse the rest of the site. It is full of ideas to help teach these concepts. We'll meet many of the data structures mentioned in other stories at the Tenderfoot session.

During the session, reference can be made to the prior reading in the card shuffle activities. This involves swapping values in an array. It allows the opportunity to clarify the difference between a list and an array. An extra challenge could be to ask attendees to research a way to swap numbers in an array which does not require a temporary variable (a=a+b, b=a-b, a=a-b).

Further Reading

After the session attendees should be encouraged to explore all the Computational Fairy Tales stories relating to the material covered. Although there are many, the 9 chapters covering Anne's visit to G'raph are most relevant. Links can be found by scrolling down to the heading Graphs (Data Structures and Algorithms) at http://computationaltales.blogspot.co.uk/p/posts-by-topic.html.



A very accessible book for those wishing to know a little more about 'real world' algorithms is 'Nine Algorithms That Changed The Future' by John MacCormick. In it, he details the development of the key algorithms that shaped the development of the modern web; search engine indexing, page rank, public-key cryptography and digital signatures, error correcting codes, multi-access databases, compression and more.

Each chapter is written in a very clear, step by step fashion. the use of analogy and explanation of the workings of each algorithm through key 'tricks' means this is a resource that can be used to develop material directly for lessons.

Advance Preparation

Well before the session is due to take place ensure that the venue has internet access and check how attendees will access it, either by logging on to institution machines or bringing their own laptops. If BYOD, ensure that is made clear in any prior publicity. Check that the venue has a projector and speakers.

Ensure there is outside space for the Quibble, Crouch, Cornerstone and Quoink activity, and the hosts are happy for you to chalk on the space. If not, an alternative might be to use string to do the activity indoors (assuming space allows it).

Ensure you have the following general material:

- Facilities for taking notes (paper and pens)
- A3 Computational Thinking Posters
- CAS Publicity: Copies of SwitchedON, BCS Certificate flyers and any local information

Attendees Materials

Activity	Materials (Per Attendee)	
Peg Swap Puzzle	Array template (paper)	☐
	Algorithm sheet (paper)	☐
	8 counters (4 of each colour, includes spare)	☐
Perfect Shuffle	Array template (paper)	☐
	Investigation (paper)	☐
	21 Card Trick (paper)	☐
	Note: p6-16 of The Magic of Computer Science	
Data Structures Matter	Spit Not So Cards (card, guillotined) Solution sheet folded. Note: 9 sets produced by printing one copy of the file	☐
	Knights Tour board and Tour Guide map (paper) Note: 2 sets printed in one copy of the file	☐
Many Problems, One Solution	Quibble, Crouch Positions Card (card, 2 copies)	☐
	Poor Cartographer maps (paper)	☐
	Mobile Phone Mast worksheet (paper)	☐
Toy Problems For The Real World	Muddy City map (paper)	☐
	Working For Efficiency worksheets (paper)	☐
Finding Kevin Bacon	Finding Keven Bacon exercise (paper)	☐
	Sample Bebras questions (paper)	☐
The Beauty Of Trees	Binary Tree Traverse exercise (paper)	☐

See overleaf for Trainers materials

Trainers Materials

Activity	Resources	
Peg Swap Puzzle	Demonstration peg game	☐
	Array template and Algorithm sheet (laminated / card) + counters	☐
	Teacher Notes (laminated / card)	☐
Perfect Shuffle	Packs of cards (Ace -8 in each suit)	☐
	Array template (laminated / card)	☐
	Investigation (laminated / card)	☐
	21 Card Trick (laminated / card)	☐
	Magic of Computer Science book	☐
	Teacher Notes (laminated / card)	☐
Data Structures Matter	Spit Not So Cards (guillotined and laminated) + large Solution sheet folded	☐
	Knights Tour board (laminated / card) + chess piece (or counter)	☐
	Cs4Fn Knights Tour booklet	☐
	Teacher Notes (laminated / card)	☐
Many Problems, One Solution	Pavement chalks (or string)	☐
	Quibble, Crouch Positions (laminated / card)	☐
	Sets of coloured pencils (1 set of 6 between 4 attendees)	☐
	Poor Cartographer maps (laminated / card)	☐
	Mobile Phone Mast worksheet (laminated / card)	☐
	Teacher Notes (laminated / card)	☐
Toy Problems For The Real World	Muddy City map (laminated / card)	☐
	Steiner Points Perspex props, bowl and washing up liquid Props must immerse in bowl (optional)	☐
	Working For Efficiency worksheets (laminated / card)	☐
	Teacher Notes (laminated / card)	☐
Finding Kevin Bacon	Finding Kevin Bacon exercise (laminated / card)	☐
	Sample Bebras questions (laminated / card)	☐
	Bebras Questions 2015 booklet	☐
	Teacher Notes (laminated / card)	☐
The Beauty Of Trees	Binary Tree Traverse exercise (laminated / card)	☐
	Teacher Notes (laminated / card)	☐
Reflective Practitioner	BCS Certificate Flyers	☐
	Trainers Notes (laminated / card)	☐

The unit presentation is designed to support a full one day session, delivered to CAS Master Teachers and other curriculum champions. It will likely be fast paced, delivered to experienced teachers.

It is envisaged that those attendees will take the material and deliver shorter sessions, either as half day, twilight of CAS Hub inputs. It is expected these will take longer to cover each activity as the material will be unfamiliar to teachers new to Computer Science. Please find time to discuss with attendees possible ways to use the material and encourage them to offer further sessions in their locality.

Resources

All supporting material is available in zipped folders, corresponding to each session in the day.

Each session folder includes:

- a presentation to support the activities, including slides (from the full session) with modified notes
- all activity sheets in both pdf and proprietary format (usually Microsoft Word or Publisher)
- a set of Teachers Notes explaining the material

There are two further folders; the research slides, and master slides to use for an Introduction and Conclusion. If using material at shorter session please consider combining session material and adding slides to introduce the 'big picture' at the start and to discuss being a reflective practitioner at the end. Please try to stick to the CAS House Style which is outlined on the opening introduction slide.

Half Day / Twilight CPD Sessions

It is suggested the material could be delivered as three separate shorter sessions.

- Introduction
- Peg Puzzle
- Perfect Shuffle
- Spit Not So
- Knights Tour
- Reflection / Conclusion

The Peg Puzzle, Perfect Shuffle and Data Structures Matter activities.

This focuses on manipulation of arrays and introduces the importance of other data structures and abstraction.

There's plenty of time to practice the magic trick too!

- Introduction
- Knights Tour
- Quibble / Crouch
- Poor Cartographer
- Mobile Phone Masts
- Reflection / Conclusion

The three activities in Many Problems, One Solution (which could be preceded by the Knights Tour activity).

This focuses on abstraction and generalisation.

Implementing the map colouring algorithm makes links with other data structures, principally lists and stacks.

- Introduction
- Muddy City
- Oracle Of Bacon
- Beauty Of Trees
- Working For Efficiency
- Reflection / Conclusion

The Toy Problems For The Real World, Oracle Of Bacon and Beauty Of Trees activities.

This focuses on the various graph traversals and associated problems.

Of course, Master Teachers can combine sessions as they feel fit, and many of the activities are short enough to introduce at CAS Hub or school departmental meetings. We hope you find them useful.

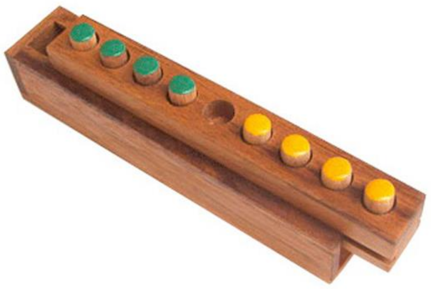
Solving The Peg Swap Puzzle

Teacher Notes to support Tenderfoot Unit 2: Clever Stuff For Common Problems – Going beyond simple algorithms

Many children will be familiar with a peg swap (leapfrog) puzzle from their childhood. It is a useful activity to introduce the notion of algorithms and data structures. The practical challenge gives children experience of manipulating values in an array.

Preparation required:
Array and Algorithm sheets + 6 counters (3 of two different colours) for each student.
Wooden puzzles if available.

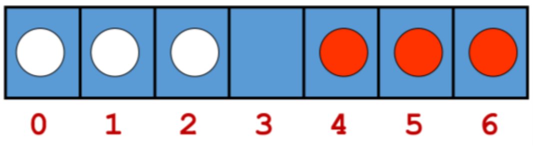
Playing Leapfrog

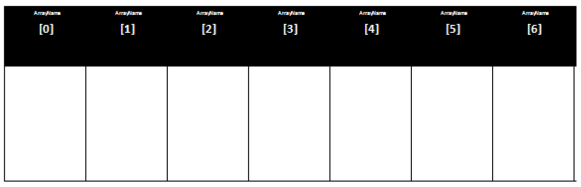


It's likely that the most complex structures a child will manipulate in programs at KS3 will be a list or array. This, in itself is often a difficult conceptual step, children struggling to distinguish between the value in a location and its index position. Before introducing programming challenges, children generally benefit from being exposed to the concept in other ways. Peg puzzles can be a kinaesthetic way into investigating data structures such as arrays. A peg puzzle like this can be made simply with wood and small sections of dowel in two colours. The aim is to swap the pegs, moving each colour alternately. There are two types of move; into an adjacent space or jumping over an opposing colour into a space. These are the only two moves allowed.

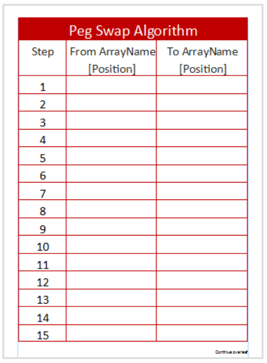
The first challenge for students is to find a way to solve the problem. Once some have found a solution, introduce a competitive element as an extension, whilst others are still trying to solve it. What is the minimum number of moves to complete a swap? Children should begin to see a pattern required to complete this efficiently. Pattern recognition like this is a key concept underpinning Computational Thinking. Once students appreciate a pattern, they should be able to extend their reasoning and tackle a larger, similar problem.

Once all the children have figured out how to solve the smaller puzzle, ask them explain their solution with the puzzle displayed on the board (with index positions initially hidden). Invite them to come up and explain. They may point to pegs initially, but it will become apparent that they need to refer to spaces as well. Discuss possible ways to do this before suggesting (if the children don't) numbering each space. Once we can identify the spaces, do we need to identify the pegs as well? We can articulate an algorithm by recording the move, rather than the peg eg space 4 to space 3.





ArrayName [0]	ArrayName [1]	ArrayName [2]	ArrayName [3]	ArrayName [4]	ArrayName [5]	ArrayName [6]



Peg Swap Algorithm

Step	From ArrayName [Position]	To ArrayName [Position]
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		

Continue overleaf

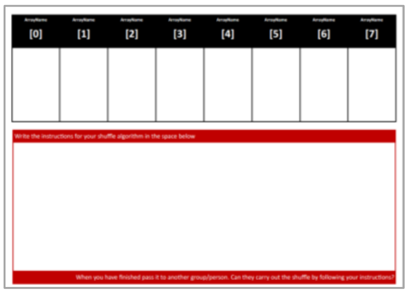
Hand out an array template, exercise sheet and counters to help them visualise and record their algorithm. If they struggle recording the steps an applet on the cs4fn website (www.cs4fn.org/algorithms/swappuzzle) will record and count the moves as they do them. This can also be useful for a homework extension, perhaps using a larger 9 space board.

The exercise emphasises a way to introduce difficult programming concepts through simple problem solving activities. Often, working things through by hand is an important precursor to any coding. Using an array template to manipulate values helps embed the idea of index positions. There are many other opportunities for a similar approach – developing an algorithm to shuffle cards being one example.

Devising an algorithm to shuffle cards is a lot easier than one to sort them. It provides a good introduction to manipulating values in an array. It also provides an introduction into investigating perfect shuffles, which leads to some magical connections between ideas in computer science.

Preparation required:
Shuffle Array Template + cards Ace to 8 in one suit for each pair / small group.
Perfect Shuffle Investigation sheet for each student.

A Shuffle Algorithm



ArrayName [0]	ArrayName [1]	ArrayName [2]	ArrayName [3]	ArrayName [4]	ArrayName [5]	ArrayName [6]	ArrayName [7]

Write the instructions for your shuffle algorithm in the space below

When you have finished pass it to another group/person. Can they carry out the shuffle by following your instructions?

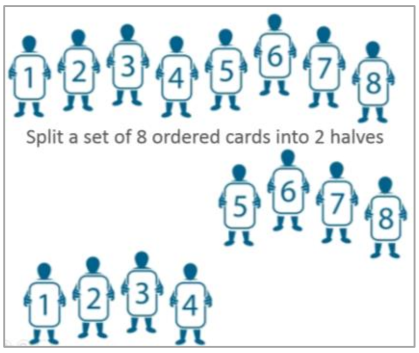
Children need repeated exposure to the same ideas in a variety of contexts before they sink in. This activity is designed to develop familiarity with manipulating arrays. It uses an array template with 8 positions and asks students to design an algorithm to shuffle cards. It works well as a small group or paired activity. Each group will need the cards Ace through to 8 in one particular suit.

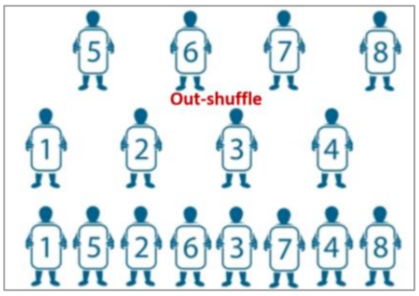
There are many ways to shuffle cards, or other lists of items. There are no right answers, but the challenge is to articulate their method as an algorithm in the space below the array template. The aim is to think about the stages involved in a shuffle, articulate them in some form of pseudocode, specifying elements in the array by reference to their index position. Once they have articulated the steps involved, they can dry run their algorithm (or that of their neighbours) to test it.

If they are stuck for ideas, the presentation slides provide three possibilities. The first, repeatedly swapping random positions is the easiest to implement. Implementing the Knuth, or Fisher Yates shuffle is much more challenging. A couple of slides are included for you to use or study. They also provide an example for children of how to write their algorithm. Articulating (and coding) a Knuth shuffle is a good exercise for more able students and for staff learning about array manipulation. The key to understanding this shuffle is identifying the remaining list – or more specifically, the last position in the remaining list.

The third suggestion, a riffle shuffle is very challenging. Implementing a riffle shuffle is a good exercise for very able children. It also provides the basis for the investigation below.

Investigating Perfect Shuffles





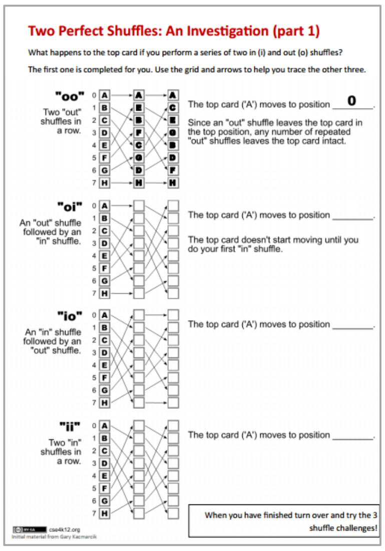
Most children will be familiar with how a riffle shuffle works. There are also some interesting properties associated with perfectly interleaved riffle shuffles, which they can explore. This is a pattern matching activity designed to reveal a surprising insight.

We can use the same array and 8 cards as before. It requires an explanation of an out-shuffle (leaving the top and bottom cards in place) and an in-shuffle, moving the top and bottom cards 'in'. The demonstration slides reveal the answer so children can check their result.

Once they understand, the first challenge is to perform 3 out-shuffles. There is a certain wonderment in finding the cards have returned to their original position.

With 3 in-shuffles the order hasn't returned, so encourage them to keep investigating. They should find it takes another 3 in-shuffles to return to the original order.

Controlling The First Card



Two Perfect Shuffles: An Investigation (part 1)

What happens to the top card if you perform a series of two in (i) and out (o) shuffles?

The first one is completed for you. Use the grid and arrows to help you trace the other three.

"oo" Two "out" shuffles in a row.

The top card ('A') moves to position 0.

Since an "out" shuffle leaves the top card in the top position, any number of repeated "out" shuffles leaves the top card intact.

"oi" An "out" shuffle followed by an "in" shuffle.

The top card ('A') moves to position ________.

The top card doesn't start moving until you do your first "in" shuffle.

"io" An "in" shuffle followed by an "out" shuffle.

The top card ('A') moves to position ________.

"ii" Two "in" shuffles in a row.

The top card ('A') moves to position ________.

When you have finished turn over and try the 3 shuffle challenges!

cse4k12.org
Initial material from Gary Kacmarcik

We can move the first card (in array[0]) to array[1] with one in-shuffle. Challenge the students to try to move the first card to each subsequent position using a sequence of 'in' and 'out' shuffles. Start by trying to move it to array[2]. The answer is In-shuffle, followed by Out-shuffle. If some find a solution quickly, ask them to see if any other sequence of moves would achieve the same result.

Combining two or more shuffles is an exercise best started in class but could continue for homework. The object is to familiarise children with array index positions whilst being able to trace cumulative changes. It isn't easy but the Perfect Shuffle Investigation gives a clear steer, with children only having to follow directions to derive the answer on the first side.

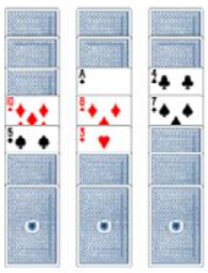
The second part of the investigation gets more complicated with less and less help available. Nonetheless, once completed, or even part way through, it should become apparent to the pupils that the pattern of In(I) Out(O) shuffles required to move to each index position is given by the binary number of the target e.g. to move to array[6] (binary 110) requires I I O. The exercise ends by asking for predictions. We are looking to develop the idea of pattern matching, and from that generalising so we can predict outcomes, based on what we have observed.

The 21 Card Trick

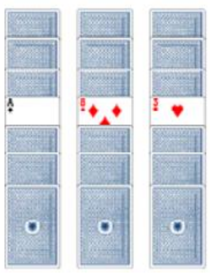
Moving cards around with a sequence of perfect shuffles is quite magical and the sort of manipulation involved in card forces. Unfortunately, perfect shuffles take a lot of practice! Paul Curzon and Pete McGowan have produced a series of magic books where each trick or illusion has explicit links to CS. The first volume focuses on card tricks. Thankfully, most require far less practice.

The 21 card trick is a classic card force, the connection with the perfect shuffle being easy to make. It can generate the sort of classroom 'wow' we wish to encourage amongst teachers. It involves dealing out 21 cards in a 3 by 7 matrix as shown right. Through 3 more successive deals, the chosen card is moved to the middle of the matrix. In performing this, we have an introduction, if needed to a 2 dimensional array and another possible algorithm to trace (or program) for very able students.

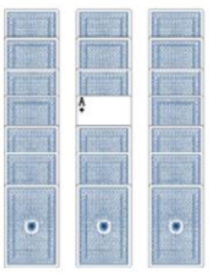
After Deal Number 2



After Deal Number 3



After Deal Number 4



What we are trying to do here is reveal the links between seemingly disparate ideas in CS. Manipulating arrays makes a connection with binary. Unexpected patterns emerge that are surprising and stimulate a desire to delve deeper. Maybe students could experiment with a full set of cards?

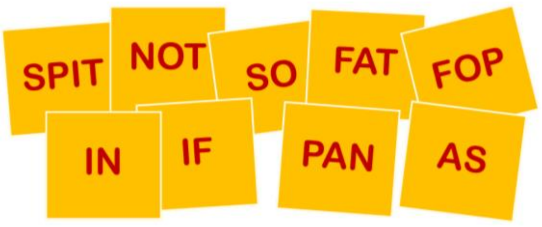
If teachers wish to understand more about the maths involved, or indeed the link between perfect shuffles and clever methods to move data in memory, the lecture by Brent Morris (youtu.be/GV9W-vvhh-0) is a valuable use of 50 minutes.

The Magic of Computer Science is a card trick special. The 'self working' card tricks are all good examples of algorithms – following a set a defined steps to achieve the same result each time. They would make an excellent subject for a Hub meeting, with attendees volunteering to do a trick each. The magic book is included in the resources.

This activity demonstrates that different ways to hold data (data structures) can have a profound impact on our ability to solve a problem.

Preparation required:
1 set of 9 Spit Not So cards, guillotined for each pair of students.
1 ‘Top Secret’ aid for each pair.

Spit Not So



Doing things well depends on the way we order information. That is the key intended learning outcome from this activity. Split group into pairs. Each pair has a set of 9 cards spread randomly between them. Display the rules on the board using the slide included. Select one person in each pair to take the first turn.

Taking alternate turns, each student takes a card. The objective of the game is to be the first player to collect three cards with words that contain a common letter. For example, SPIT, NOT and FAT all contain the letter T. Each time a player wins a game, they are awarded 1 point. All cards are returned to the middle and a new game starts. The players take turns to be the player taking the first card.

Each pair should play as many games as possible in a few minutes, keeping score. Try to encourage a fast pace between games, maybe setting a competitive element to see which pair can complete ‘x’ number of games first. When you stop the activity ask those who are winning in each pair to put their hands up. If a pair have an equal number of wins, play one more game to ensure one is in the lead, whilst you discuss the game with the group, asking if they found it easy / difficult etc.

The Secret

NOT	IN	PAN
SO	SPIT	AS
FOP	IF	FAT

Explain you are going to give those currently losing some top secret help. Select those who didn’t put up their hand and gather them around your desk, or ideally just outside the room. Establish if they have played noughts and crosses. If not, demonstrate the game. Hand each a copy of the Top Secret aid and insist on them keeping their instructions hidden in their hand. They will need a pencil to keep a note on their aid. The aid arranges the words in a 3 by 3 grid, just like a noughts and crosses board. Words containing the same letter form a line, either vertically, horizontally or diagonally. By recording the words they take, and their opponents, using X and O, children can now play Spit Not So as if it was a game of noughts and crosses. They may not always win, but most children know how to force at least a draw in noughts and crosses.

The students should return to their pairs. Explain to the class that you have given them some secret training and that we are going to play again. Start with a blank score and give students several more minutes to play some more.

When you stop the groups again, ask those with the Top Secret aid whether it has changed their fortunes, before discussing as a class. Ask the children to explain why the Top Secret aid helped them.

Reinforce the point that the way we structure data has a very big bearing on how successful we might be. We shouldn’t underestimate the challenges facing children when learning to code. At KS3, repeated exposure to single variables and one dimensional lists or arrays is probably about right, but by introducing more complex data structures (without worrying about how we implement them in code) we can begin to open their eyes to clever algorithms that can solve everyday problems.

Problem Representation and Abstraction

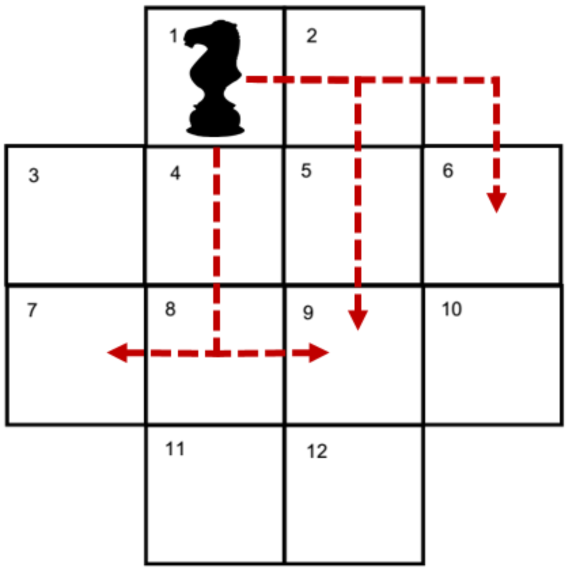
Teacher Notes to support Tenderfoot Unit 2: Clever Stuff For Common Problems – Going beyond simple algorithms

These two class activities are designed to demonstrate how the representation of a problem can help or hinder the solution. They focus on representing problems through graphs, graph traversals and using abstraction to draw out the essential elements of a problem.

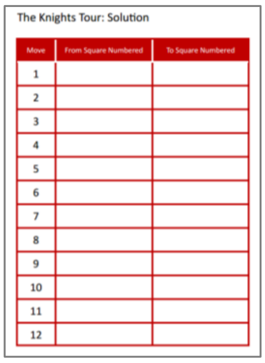
Preparation required:
Knights Tour resources (board, solution sheet and Tour Guide map) + 1 counter per student.
Plain paper and pencils.

A Knights Tour

A Knights Tour is a puzzle, recently been reproduced as a cs4fn booklet free to download from http://teachinglondoncomputing.org



Give each student a board and Knight piece (counter). This can be cut from the board if needed. Place the Knight on square 1. Check everyone is familiar with the way a Knight can move: from square 1, the piece could move 3 possible ways, to squares 7, 9 or 6. The challenge is to work out a series of moves so the knight visits each square once (and once only) and ends up back at its original position.

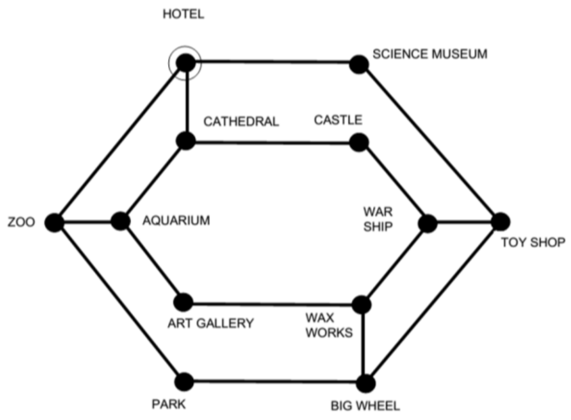


The Knights Tour: Solution

Move	From Square Numbered	To Square Numbered
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		

Give out the sheets to record the moves. You could perhaps offer a small prize for the first to solve it. There are many solutions, so if a child solves it quickly, challenge them to find as many different ways as possible. It isn't easy, however, so allow perhaps 15 minutes for this activity and then stop everyone. Establish if anyone / how many did succeed and discuss how hard they found it.

Tour Guide



If the Knights Tour was a little challenging, helping a Tour Guide plan an outing to visit every attraction might be easier. Ask a child to give out the maps as the students will solve this one very quickly. Alternatively project onto a whiteboard and ask children to come and outline their solutions, or simply read the order of attractions to visit.

There are many solutions, one is highlighted on the slides provided. Solving the problem isn't the point of the exercise. Discuss with the class why it was much easier to solve this problem, compared to the Knights Tour. Answers may include the fact that you can trace a trail, having a map makes it easy to see etc.

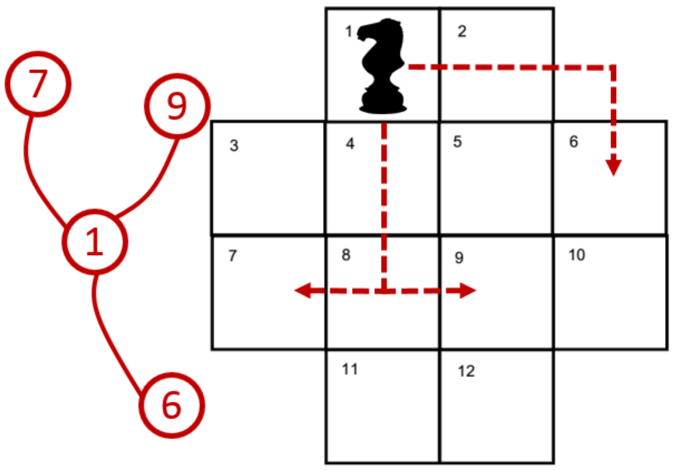
In essence, the way the problem is represented makes it easy to solve. The map or diagram is an example of a structure computer scientists call a graph. A graph consists of a series of nodes connected by edges and they are remarkably useful for representing a range of problems.

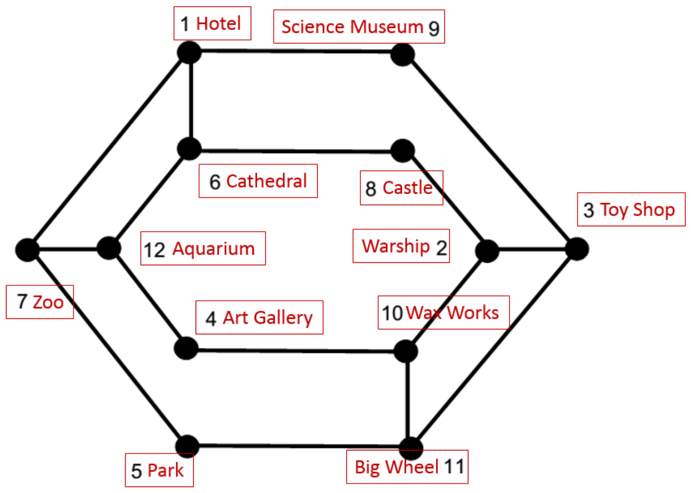
A Knights Tour Revisited

Deriving a graph from a problem statement is a good example of representational abstraction. The graph captures all the relevant information required to solve the problem, and remove s any superfluous detail.

Discuss with the class whether we could use a graph to represent the Knights Tour. What superfluous detail could we remove? Do we need to know which squares, for example, are adjacent to each other?

All we really need to know, to solve this problem is which squares the Knight can reach from any location. A node can represent each square, linked by an edge to every other square the Knight can reach, so node 1 is connected to node 7 and to node 9 and node 6 as shown.





Armed with that knowledge, give out plenty of rough paper and challenge the students to complete the graph representation. The answer is available on a slide for students to check their attempts against. Ensure they check carefully.

Often they may not look alike, but a little re-arranging of nodes will show it is the same. The layout is secondary to the connections. By arranging it in the way shown on the slide should make something obvious. Do they notice anything about the graph? It is identical to the London Tour Guide problem!

By writing the specification of the problem in general terms, we can begin to see similarities. Once we see the similarities in problems, we can apply a general solution to both. This notion of generalisation, of recognising patterns is a central concept in Computational Thinking.

The solution to both these problems, visiting every node and returning to complete a circuit is known as a Hamiltonian Cycle. It is named after the Irish physicist and Mathematician, William Rowan Hamilton (1805-1865), who devised a puzzle to visit every corner of a dodecahedron, as shown on the final slide. The twelve sides of the dodecahedron can be represented as a graph and a cycle traced through it.

Tracing a route through a graph can make good homework exercises, but the difficulty depends on the size of the graph. Determining whether a graph contains a Hamiltonian Cycle is one of a class of very hard problems for computers. Visiting every node of a graph is just one of several common operations which collectively can be called graph traversals. As you might expect various algorithms have been developed to perform different types of traversals.

Developing an efficient way of finding some types of traversal remains a key challenge in computer science. The key point for us though, is to stress that by abstracting away details and recognising similarities we have the potential to use a general algorithm to solve a particular problem.

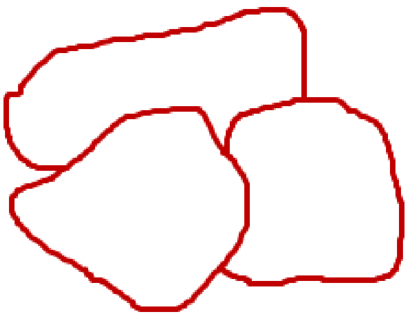
Three activities that introduce 'map colouring' problems, abstract representation as graphs, the algorithm to identify the minimum colours required to colour a map and its application to a real world problem.

Preparation required:
1 Quibble, Crouch Positions Sheet
Set of pavement chalks or lots of string for first activity.

Quibble, Crouch, Cornerstone and Quoink

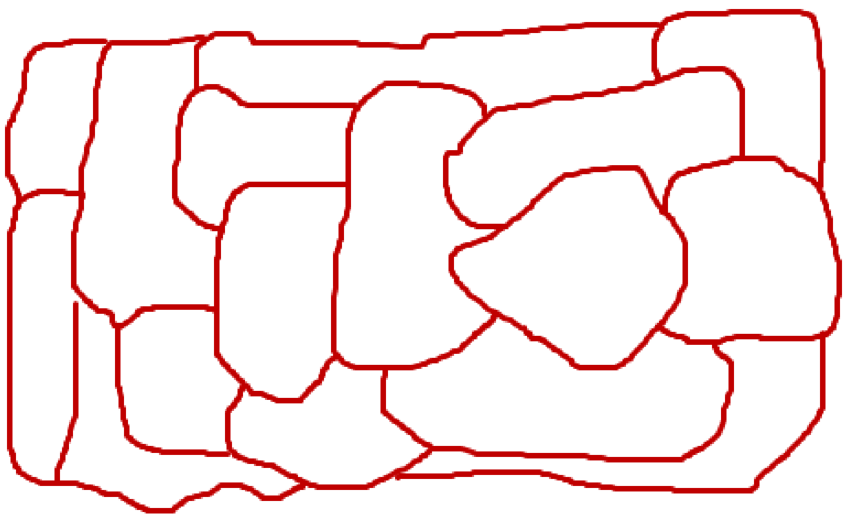
This activity is weather dependent, although it could be done in an open indoor space. It will require some pavement chalks (or string) and probably works best in groups of approximately 12 to 17. The idea was taken from the now defunct MegaMath project. MegaMath was a project of the Computer Research and Applications Group at Los Alamos National Laboratory.

Each child in turn draws a small area in the playground to stand in. An example of the first three areas of a 'map' are shown right.

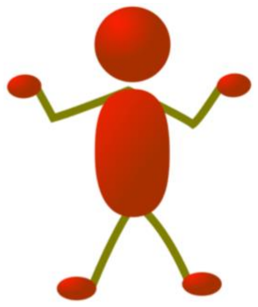


Each area should be connected to at least one of the existing areas. Encourage children to draw shapes that connect to more than one other area if possible (without getting too ridiculous!).

Gradually build up the map, with each child adding and then standing in their area until everyone is standing on the map.



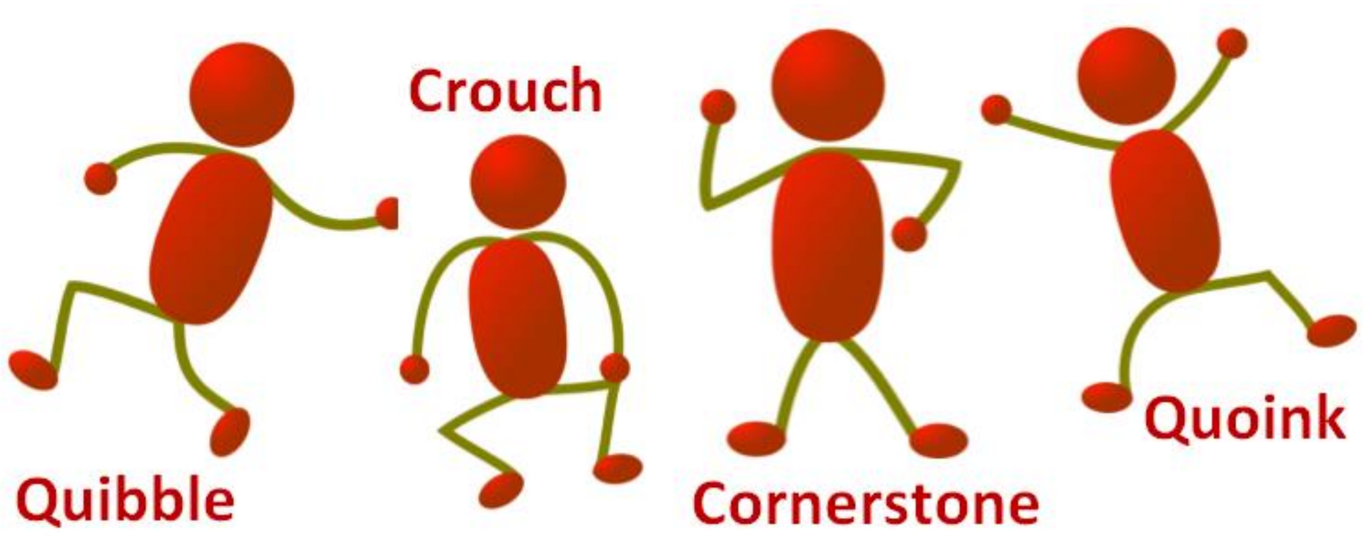
If you have any particularly boisterous children, you may wish to retain a couple initially to help you direct operations and check everyone is doing it right. Once the activity starts, you can always add them in at a later stage by adding extra spaces to the map. This can be a useful extension, particularly if they solve the initial challenge quickly.



Each child will start in a confused state. Model the stance shown left. This indicates they don't know what to do.

The challenge is for the students to arrange themselves into one of four positions, Quibble, Crouch, Cornerstone and Quoink.

Each stance is explained overleaf.



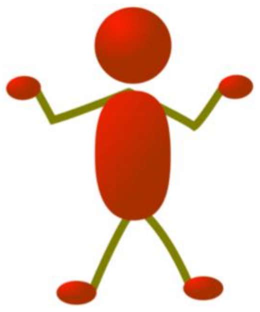
The challenge is for the students to arrange themselves into one of the four positions:

Quibble: standing on one leg (they can swap legs if they are tired!)

Crouch: squatting,

Cornerstone: standing straight and tall, hand on hip and clenched first salute and

Quoink: jiggling about and waving their arms above their heads.



The only rule is that they must not adopt the same position as any of their neighbours that their shape connects with.

If they don't know what to do, or can't find a position to satisfy the rule, they stay in a confused state.

The challenge, as a group is to see how long it takes until no-one is still confused.

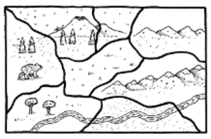
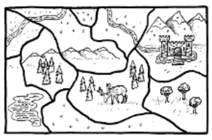
If silly positions are potentially too disruptive, each child can be issued with cards bearing the position names which they simply hold aloft.

If the group find a solution quickly, remove one of the positions and see if they can find a solution with just three different positions.

A final extension might be to create a map by drawing closed loops, overlaid on each other. Challenge the group to find the smallest number of positions needed to satisfy the rule that no two adjoining areas adopt the same stance. The discrete areas of a map created this way can always be filled with just two positions.

Back in the classroom you can discuss the success (or otherwise) of their efforts. If you have a camera, a snapshot of the 'map' might be useful to display on the board.

Preparation required:
3 maps for each student, spare plain paper.
Several sets of coloured pencils

The Poor Cartographer's Problem

Three simple maps are included to introduce map colouring and the 'poor cartographer's problem'. Use the larger map first – possibly as a group activity. The challenge for students is to find the minimum number of colours required to colour each country. Countries that share borders should not be the same colour. We want children to discover the 'has to be' rule, and identify ways to check existing colours before adding another. In this case, the map can be coloured using just two colours. We can also use it to clarify the point about countries meeting at a point, which does not constitute a shared border.

The two smaller maps requires three and four colours respectively. Set these as independent challenges. An extension task might be to ask students to design their own map that requires five colours to complete. Whilst students may think they have a designed such a map, all maps drawn on a flat surface require only four colours.

It can be pointed out that the conjecture that any map can be coloured using only four colours was formulated in 1852, but it was not proved until 1976. Computer science is full of unsolved problems, and knowing that the four-colour theorem was proved after more than 120 years of research is encouragement for those Computer Scientists looking for solutions to hard problems today. Another extension challenge is to draw a map consisting of overlaid, closed loops. What is the minimum number of colours needed for this? Any map constructed in this fashion only needs two colours.

As a class you can consider some of the problems we might face tackling a harder problem, such as the map of Europe. What strategies might we adopt? Would we start with any specific country e.g. the one with the most borders? How do we know which country has the most borders anyway in a large map? We need to abstract away the unnecessary detail in a map, then develop an algorithm to solve it. The associated slides demonstrate how to develop a graph where nodes represent countries and edges their shared borders.

To apply the algorithm we can represent the graph as an 'adjacency list'. This is a list of nodes, each with an attached list of connections. The slides walk through implementing the algorithm. This is a two stage process: Stage 1 determines the best order to colour the nodes and Stage 2 assigns a legal colour. The slides allow you to work through this as a class, pausing for questions, each step initiated 'on click'.

For Stage 1 the node with the smallest number of connections is selected first. Take care to point out how, when a node is selected, it is removed from the adjacency lists of any nodes remaining. This is the key stumbling block to implementing the algorithm. Where several nodes have the smallest number of connections, any can be selected. Repeatedly applying this leads to a new list of ordered nodes being created.

Stage 2 involves selecting nodes in reverse order from the new list. There is the potential here to investigate how we might reverse the order of a list. Had we built the list as a stack, the first node selected in stage 1 would have been at the bottom and the last at the top. Taking items from the top of the stack selects them in the reverse order to that when added. The last one in, is the first one out. For each node selected, its connections are listed, and a colour assigned. We will always assign a colour used already if we can. However, we cannot select a colour assigned to another node, if that node is listed as connected.

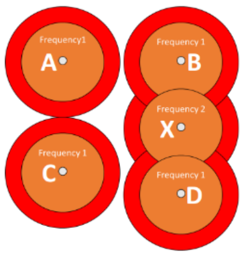
Preparation required:
Mobile Phone Mast Problem activity sheet for each student.

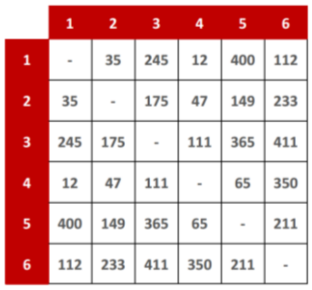
Mobile Phone Mast Problem

This activity illustrates how an algorithm for one problem (map colouring) can be applied to a seemingly unrelated one. It provides a real world context, reinforces the idea of abstract representation of the problem as a graph, and it encourages children to have a go at implementing the map colouring algorithm independently.

Mobile phone use has exploded over the last twenty years. To satisfy demand, network operators have built ever increasing numbers of base station transmitters / receivers. The problem they face will be familiar to anyone who has a wireless doorbell. Where many devices use wireless communication, to avoid interfering with each other, they must use different frequencies. Small devices work in the unregulated frequency ranges. They usually avoid interference by having several frequencies they can potentially operate on.

Telecommunications is a more serious matter. Telecoms operators must operate within a regulated range. Each operator must purchase the right to transmit on a particular frequency. Each base station has a transmission range as shown on the schematic. Base stations transmitting on the same frequency cannot overlap without causing interference. To cover the gap, an extra station (X), working on a different frequency is required. Each telecoms provider will want to maximise mobile phone reception, but minimise the number of frequencies it has to purchase.





	1	2	3	4	5	6
1	-	35	245	12	400	112
2	35	-	175	47	149	233
3	245	175	-	111	365	411
4	12	47	111	-	65	350
5	400	149	365	65	-	211
6	112	233	411	350	211	-

The table shows six base stations owned by a hypothetical telecoms provider. The figures indicate the distance between each mast. Masts that are within 150 miles of each other must operate on different frequencies. Each frequency costs £50,000 to purchase. How can we go about calculating how many frequencies the operator needs to buy? In the discussion the students will hopefully recognise that we might be able to construct a graph to represent the problem.

What sort of problem is it though? Is it a map colouring problem? Once established that it is, and we already know how to solve these point out this is an example of generalising a problem. But what would we put as nodes and edges? The nodes seem obvious, each base station, but the edges need some thought. We don't need to know about the base stations that are over 150 miles apart. That is unnecessary detail. All we need to represent is the base stations that are within 150 miles. Armed with this knowledge, it should be possible to draw the graph, and apply the same algorithm as before. A student worksheet is included to encourage them to tackle this independently.

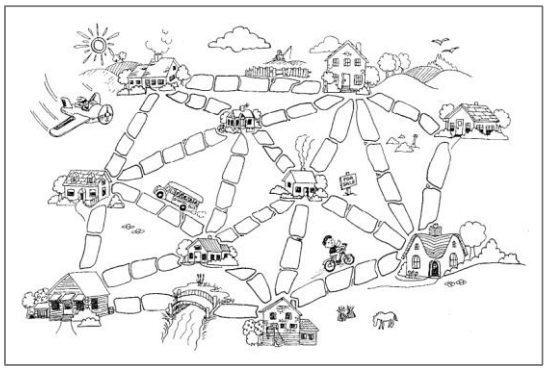
We may have algorithms for solving particular classes of problems, but recognising a problem as similar to another is part of the art of computer science. There is no set of rigid rules for solving problems. Recognising that a problem is identical to another is partly experience. Deciding what to include in an abstraction, for example, is a judgement. The ability of students to make sensible judgements, will depend on the intellectual toolkit they develop. That in turn will be based on familiarity with computational concepts that occur again and again, such as abstraction and generalisation. Combining abstraction with algorithms and automating the algorithm through computer programs, (a computational abstraction) is a powerful tool.

This simple class activity leads to students discovering either Prim's or Kruskal's algorithm for finding a Minimum Spanning Tree (MST). It goes on to consider the similarities and differences between a MST and other classic graph traversal algorithms.

Preparation required:
Muddy City puzzle sheet for each small group

A Muddy City

This simple exploration allow pupils to discover famous algorithms that are used today in a wide range of real world problems. Split into small groups and distribute the map. Explain that it has no roads, making it difficult (and muddy) to get around. Enough streets must be paved so it is possible to travel from any house to any other house, possibly via other houses. The paving should be done at a minimum cost. The diagram indicates the number of paving stones needed to connect each house. The bridge requires a paving slab. What is the smallest number of paving stones needed?



Give groups 10 minutes or so to come up with answers. 23 slabs are needed. Ask groups to indicate their solution on a projected map. There are several correct solutions. Two are shown as exemplars, but the key task is to get groups to articulate their strategy for solving it. Allow time for students to articulate and discuss alternatives. There are many possibilities but intuitively, students will probably come up with selecting the shortest connection first as a starting point. Two possible approaches generally follow, both known as 'greedy' algorithms – they always take the best option available. Each results in an optimal solution, known as a Minimum Spanning Tree (MST).

Prim's Algorithm

Prims algorithm selects the lowest connector from the two houses first connected. It is demonstrated in the subsequent slides, which can be introduced as a series of questions, each click building up the MST. It is worth becoming familiar with the slide notes which contain the narrative. Although known as Prims algorithm it was developed initially by a Czech mathematician, Vojtěch Jarník in 1930. Only later was it also discovered independently by computer scientists Robert Prim (1957) and Edsger Dijkstra (1959).

Kruskal's Algorithm

Prim's algorithm is not the only one for finding a Minimum Spanning Tree. Another, often discovered intuitively by children was developed by American computer scientist Joseph Kruskal in 1956. Challenge pupils to spot the difference between this approach and Prim's. A simple walkthrough is provided. As with Prim's we select the shortest edge as a starting point, but Kruskals algorithm takes the shortest connection, regardless of whether it joins the existing path.

Minimum Spanning Trees provide the shortest connection utilising existing points. Either algorithm will result in an optimal solution. They are widely used in many real world scenarios, a good example might be planning the network cabling in the school. Knowing they have discovered a 'real' algorithm can be very motivating for pupils. Can they think of other situations where finding a MST would be useful?

Similar Problems

Many problems share similarities with a MST. A Steiner Tree shortens distances by introducing new interim points but an optimal solution is difficult to compute. A CS Unplugged outdoor activity, Ice Roads could make an extension activity. An explanation by NumberPhile: www.youtube.com/watch?v=dAyDi1aa40E is worth watching. Soap bubbles will naturally form a local Steiner Tree. It is simple to demonstrate with a little preparation.

Teacher notes

create maths

Getting there : Working for efficiency

Description

This topic looks at simplified versions of three different network problems that are encountered in practical logistical planning.

Activity 1: Paper rounds
Activity 2: Cable connections
Activity 3: Deliveries

Edge-traceable graphs are explored in Paper rounds. The starter activity offers a simple version of a practical context in which to explore the concept. The famous mathematician Euler noticed that, to be edge-traceable, a network must have at most two odd nodes.

The worksheet offers opportunities for further exploration and then asks the pupils to search for a rule and to prove their results. At some point it is worth asking: can you have just one odd node?

Using tracing paper supports experimenting. For lower attaining pupils, slipping the worksheet into a plastic wallet will allow experimentation using felt tip pens, wiping off incorrect attempts, which may be easier.

Encourage pupils to look for what the edge-traceable networks have in common. Which graphs can be traced whatever starting point is chosen? Which graphs can be traced only if you start, and end, at particular points? Pupils will test out their theories better if they work together.

Cable connections provides a practical example of a minimum connector problem. It is quite possible to find a good or even an optimal solution using trial and improvement methods but the task provides the opportunity to introduce pupils to the key mathematical idea of an algorithm. Prim's algorithm, to find this minimum spanning tree, is quite within the grasp of many pupils, even at key stage 3.

For some of the problems on the worksheet, there is more than one optimum solution.

Prim's algorithm works as follows:

Step 1: Choose an arbitrary location, say Grimsby, and connect it to the nearest place, in this case, Westgate. Note the distance.

Step 2: Find the nearest place not yet in the solution, in this case either Frenchy place or Wheekby Road. Whenever you get a choice of connections of equal length, choose either one. Here we have arbitrarily chosen Frenchy place.

Continue this process until all the locations are connected.

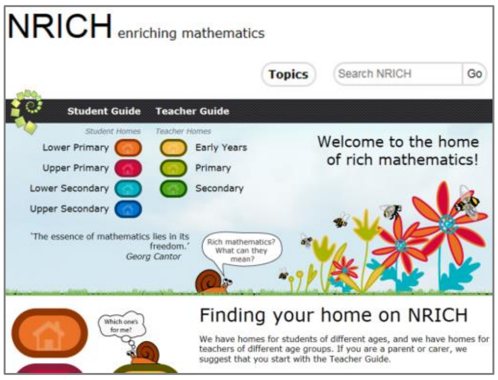
Working for efficiency Page 1

The Create Maths 'Working For Efficiency' resources use real world graph problems. 'Cable Connections' involves finding a MST. Problems often appear similar, but can be subtly different. In Muddy City, the MST challenge finds the shortest tree that connected all nodes. 'Deliveries' looks for the shortest cycle - famously known as the Travelling Salesman Problem. It is famous because no known algorithm can solve it in a reasonable amount of time, even for a small number of locations. It is a common misconception that as computers get faster and faster, they will eventually be able to solve any problem such as this by 'brute force'. But even with 8 locations, all connected, a fast computer might take around 5 minutes to check every possible combination. Just increasing that to 10 requires 5 hours, and with 20 locations, it takes nearly 200 million years!

If this seems ridiculous, read the New Zealand CS Field Guide: www.csfieldguide.org.nz/en/chapters/complexity-tractability.html. It explains the concept of intractability well and has a nice interactive to show how such problems quickly becomes intractable. Nonetheless, having just discovered Prims algorithm, there is no harm in children having a go. With small graphs it is possible to get 'good' solutions, even if we can't prove they are optimal. Again, the focus is on developing a greedy strategy. The website www.math.uwaterloo.ca/tsp/games/index.html has two games for children to explore. Well worth investigating.

Is a task such as planning a paper round (the 3rd Create Maths activity) like any of the previous problems? Can we use a previous strategy to solve this? Sadly we can't. In this case, we need a solution that visits every edge once, rather than every node, and returns to the starting point. These tours are known as Eulerian Circuits, named after the mathematician Leonhard Euler (1707 -1783) and his first statement of this classic problem, the Bridges of Konigsberg. Edge traceable graphs make great 'pencil puzzles'. Students try to trace a route along all the lines without taking their pencil off the paper.

Like Hamiltonian Cycles, finding an Eulierian Circuit is a very difficult problem for a computer. But being able to say whether a graph contains an edge traceable circuit is much easier. It involves an investigation of the properties of the graph. Encourage children to study the number of connections at each node. Can they spot the pattern? The Create Maths teacher notes provide further pointers for structuring this investigation. Investigations of this sort are ideal for developing the ability to generalise: observing specific examples, spotting similarities and developing a general hypothesis that can be applied to all similar problems.



Welcome to the home of rich mathematics!

'The essence of mathematics lies in its freedom.' Georg Cantor

Finding your home on NRICH

We have homes for students of different ages, and we have homes for teachers of different age groups. If you are a parent or carer, we suggest that you start with the Teacher Guide.

These algorithms fall under the heading of Discrete (or Decision) Maths and many lie at the heart of Computer Science. The Nrich Maths project (http://nrich.maths.org) also has many activities that can introduce Computational Thinking in KS3. The entire repository is searchable by Topic. A good starting point is to select Topics – Decision Mathematics – Network / Graph Theory. The results can then be filtered by Key Stage. Each resource has hints to get children going, solutions and teachers resources. The teacher notes, in particular, are very useful, providing links to extensions and clear explanations as to the purpose of the exercise.

A humorous web based activity leads on to looking at ways to develop an algorithm to determine the shortest path between two points.

Preparation required:
Finding Kevin Bacon exercise sheet for each pupil or small group

Six Degrees Of Kevin Bacon

This session starts with a fun activity that can be used to introduce another 'classic' algorithm and ends with a recap of how different data structures all 'fit together'.

Twenty years ago, inspired by "six degrees of separation," (the theory that nobody is more than six relationships away from any other person in the world) a game was dreamed up by movie buff Brian Turtle, called Six Degrees Of Kevin Bacon. The game involves trying to connect any movie star to the actor Kevin Bacon, linking them by other actors who have appeared in the same film. The presentation shows an example using Alexandra Daddario, who played Annabeth in Percy Jackson films. It explains how we calculate her Bacon Number as 2.

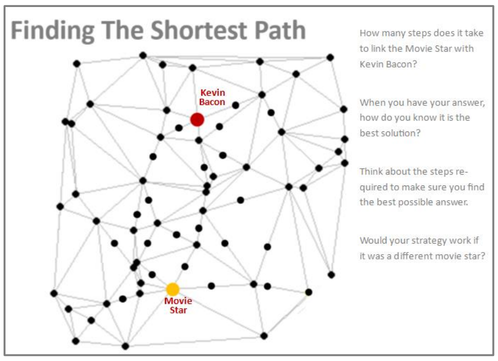
The internet movie database lists nearly 3 million actors and actresses, and about 2 million movies. A distribution of the statistics for 'Bacon numbers' is shown indicating that finding a movie star with a Bacon number of 4 or over is challenging! (Note the chart will be out of date as it is updated regularly). Such was the popularity of the game, Google have built Bacon number functionality into its search engine. Here's a fun class challenge! Pupils use Google to try to find a movie star with a Bacon number greater than 3. The star children search for must have appeared in a movie, not TV or music videos.

It's not just Google who calculate a movie star's Bacon Number. There is a dedicated website, called The Oracle of Bacon which does the same thing and more. The presentation uses the website to calculate the link between Daniel Radcliffe and Kevin Bacon, like shown previously in Google.

Although we have the same Bacon Number, the connections are different to those given in Google. Invite suggestions as to why this might be the case. The answer lies in understanding how a Bacon Number is calculated. So how does it do it?

All 3 million actors and actresses on the internet movie database are linked in a gigantic graph. Each node is a movie star. Each edge between two nodes is the film they both appeared in. An animation builds the links so Alexandra Daddario is linked to Logan Lerman (Percy Jackson) through the film Sea of Monsters. She is also linked to Douglas Smith, Katelyn Mager, Leven Rambin and every other actor/actress in that film. And of course, it is not just Alexandra who is linked to each star, but they are all linked to each other. The animation then links to stars who appeared in Noah… to give a sense of how complicated the graph will become.

In reality, the graph is far more complicated than the animation, and would look more like the graphic in the presentation, with Kevin Bacon at the centre (the different colours represent people in the same film clustered together).



Finding The Shortest Path

How many steps does it take to link the Movie Star with Kevin Bacon?

When you have your answer, how do you know it is the best solution?

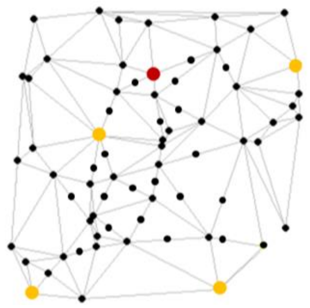
Think about the steps required to make sure you find the best possible answer.

Would your strategy work if it was a different movie star?

So a very large graph is the structure that holds all the data, but what about the algorithm to link a movie star? Let's try to work out an algorithm. As a whole class activity, with the diagram displayed as shown, imagine the red node is Kevin Bacon. The yellow node is another movie star entered on the website. We need to find the shortest path between the two.

Hand out the graph for students to try to find the smallest number of connections to link the two.

If students think they have found it, invite them to connect the path on the board. If they are struggling, announce you have found a link using eight steps – can they beat that? (They should, the shortest is 7 and the answer is given on the following slide).



Conclude by encouraging a class discussion. What lessons can we learn from this exercise? The best solution may not always be the most obvious. The shortest path starts by moving away from the destination for example. Ask how they got their answer. Did they develop a systematic approach? We are looking for them to articulate an algorithm that checks possible routes and is general enough to apply to any situation. Once students have articulated a possible approach, consider if it could be applied to the general situation of finding the shortest path for any Movie Star to Kevin Bacon (as shown by different yellow dots).

We can generalise the problem even further. The Oracle of Bacon allows you to calculate not just a star's Bacon number, but their degree of separation from any other movie star. So here we can check, for example the degree of separation of Alexandra Daddario from Daniel Radcliffe (Harry Potter). She has a Daniel Radcliffe Number of 2.

We can use a simpler graph to demonstrate the approach. The animation in the presentation walks through how a search builds up. From the starting node we identify every node connected to it and assign a value of 1. Then we take each node assigned a 1 in turn, identify every node connected to it, and label them with a value of 2 (if it has no value already assigned to it). A detailed explanation of the subsequent steps, and subtleties is in the slide notes. Check and rehearse carefully before using! It explains how Google and The Oracle of Bacon can get the same Bacon Number but with different links. It advances 'on click' so pause to ask pupils to predict which connections will be made if required.

Finding The Shortest Path

Having found the destination, we can identify a shortest path. There may be others just as short, but no path can be shorter. An algorithm like this is known as a Breadth First Search because it searches every node one step away from the start, before it tries any further away.

Is Kevin Bacon at the centre of the Hollywood universe? To answer that involves calculating every star's Bacon number, then working out the average. The table on the slide shows the distribution. Kevin Bacon is, on average 3.009 degrees of separation from every other star. Challenge the pupils to find a star who has a lower average number. Who can find the best 'centre' of the movie universe?

Do read the explanation of how the site works. It gives an idea of the size of the graph structures that power applications such as these, as well as ways to link this work to other aspects of computing. Note in particular, the details about caching requests to speed up the ability to answer future similar queries. Calculating the shortest path to every movie star would take a long time!

Breadth first searches lie at the heart of shortest path algorithms. Such algorithms lie behind many common applications, such as Sat Nav's and route finding software. Encourage students to think about how the graph behind Google maps or AA route finder would be constructed. Every intersection of every tiny road is in there. The sheer size of these graphs makes route finding a demanding task. Shortest path algorithms have become much more efficient over the last 20 years. If they hadn't many route finding apps wouldn't be able to run on devices such as Smartphones which have far less memory than most PC's. Watch Simon Peyton-Jones (30 mins) talking about recent developments; research.microsoft.com/apps/video/?id=146561

The presentation ends with two exemplars from the Bebras competition 2015: www.beaver-comp.org.uk. The first (p31) was aimed at KS3, a simple shortest path problem, which pupils can probably do in their head. The second (p59) is much more involved. Answers and explanations are contained in a separate handout. The complete booklet of questions is included in the resources, graded from youngest to oldest. Those on p25, 31, 34, 44, 47, 49, 50, 52, 55 and 59 are particularly pertinent to graphs and tree structures.

An initial demonstration shows how compound data structures, such as lists and trees, can be viewed as subsets of the more general graph structure. A short class exercise demonstrates how a particular subset of a tree structure, a binary tree, can be constructed from a random set of numbers. A specific traversal returns the numbers in sorted order, without moving them within the structure.

Preparation required:
Binary Tree Traverse Exercise sheet for each pupil.
Plain paper.

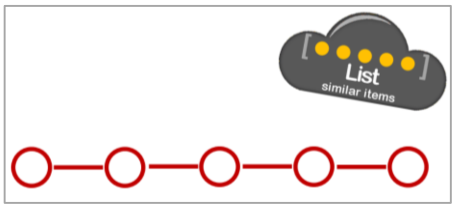
Variables, Lists, Trees and Graphs

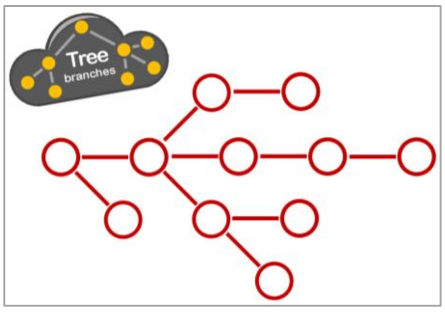
The famous computer scientist, Niklaus Wirth wrote a book in 1975 entitled Algorithms + Data Structures = Programs. A better way to think of it might be that computation acting on information leads to solutions to problems. Most students are familiar now with the term algorithm and the key constructs from which algorithms are built; sequence, selection and repetition. On the other side of the equation are data structures for holding information, such as lists, graphs and trees. The presentation shows how they 'fit together' as a family.

Variable
a single entity

The presentation does this by considering the development of a rail network. Initially we may have only one station. A single entity, such as a station can be stored as a named variable. Single variables can be thought of as the foundation of all other data structures. We can build more complex structures by aggregating variables – making a compound data structure.

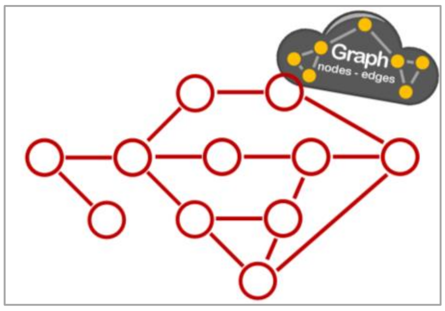
Once we connect two stations, a compound data structure makes sense. Because they are similar items a list would work well. And a list would continue to be appropriate whilst extra stations were added to this one line. A linear structure such as this could also cope with new stations inserted in the line, rather than appended to the end. But what happens when branch lines are added?





This new sort of arrangement is better thought of as a tree – with different paths or lines branching off. Trees can have many branches, but re distinguished by only having one path to a leaf node. They can also be rooted (or not) as we shall see with an exercise using a binary tree.

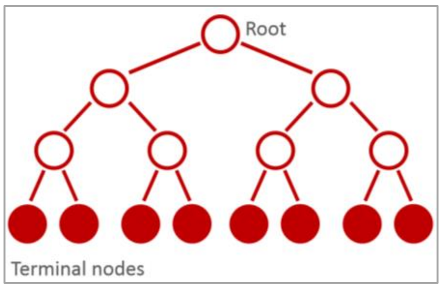
As the network continues to grow, branching paths may no longer be adequate for representing the railway. We may have lines that loop round, for example. To represent a rail network with circuits in it we need to be thinking in terms of graph structures, the most general representation. So Trees and Lists can be thought of as particular subsets of the more general graph structure.



We often introduce algorithmic constructs to children away from the computer. Developing familiarity with these makes the challenge of coding algorithms easier. Similarly, developing an appreciation of the role of data structures away from the computer helps develop an appreciation of their role in developing clever algorithms. The exercise overleaf introduces a novel way to solve a common problem.

Think Of A Number

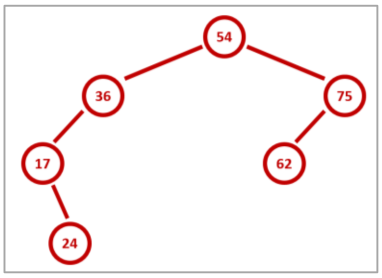
A binary tree is a particular subset of the more general tree structure. Firstly it is a rooted tree, which means a node is identified from which the tree springs. Somewhat confusingly, in computing, trees are usually drawn upside down. The root is at the top.



In a binary tree the root node has two and only two child nodes, hence the name. Each child node is referred to as the left and right. Each subsequent node also has two child nodes though it is possible that every node may not be used. Formally though, in a binary tree every node has two child nodes, apart from terminal nodes, which have no children.

You can play do this as a class activity with a child at the board drawing the binary tree as it grows or you can have each child creating their own tree on paper. The activity involves children picking numbers at random and can go on as long as you want. For simplicity, it is probably worth having limits, say between 1 and 100, and avoiding duplicate values.

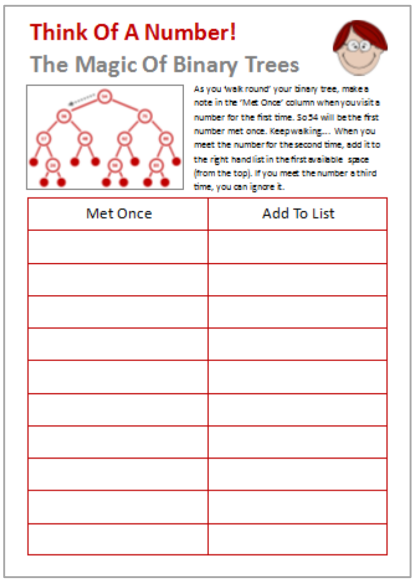
To demonstrate, we use ten numbers, selected at random, though, for the purpose of the animation, they had to be selected in advance. The presentation builds the binary tree by following simple rules. The first numbers are 54, 36, 75, 17, 24, 62 …



The first number, 54 becomes the root node. Because 36 is less than 54, it become the left child node and as 75 is greater than 54, it become the right child node. So where does 17 go? Starting at the root, we move left, because 17 is less than 54. It is also less than 36 so becomes the left child of 36. How about 24? Less than 54, and 36, but more than 17, so the right child of 17. 62 goes right of 54, but left of 75. Obviously we could go on and on, the slides completing the binary tree for the list of numbers.

In a binary tree, each node has two child nodes apart from terminal nodes. As every node containing a value should have two children, we can add empty terminal nodes to complete the tree.

Once built, we 'walk around' the structure (traverse the tree). Starting at the root, going anti-clockwise initially, pupils simply trace a line around the tree, making a note each time they meet a number, then moving it to a second column when they meet it a second time. Give each student a sheet to record the results.



Think Of A Number!

The Magic Of Binary Trees

As you 'walk round' your binary tree, make a note in the 'Met Once' column when you visit a number for the first time. So 54 will be the first number met once. Keep walking… When you meet the number for the second time, add it to the right hand list in the first available space (from the top). If you meet the number a third time, you can ignore it.

Met Once	Add To List

If they are struggling, there is a hidden slide which builds up the answer. It can be used for a whole class as a child traces the route on a whiteboard. The subsequent 3 slides walk through the process step by step, visiting each node in turn. The first slide takes the animation up to the return to the root node. Then up to the second visit of Node 75, the third slide completing the traversal. Use (or hide) whichever is most suitable.

Once completed, can the students see what has happened? By following this algorithm we can retrieve an ordered list without having to sort the values first.

Another Tenderfoot Unit: Doing Stuff And Doing Stuff Well considers ways to evaluate the efficiency of algorithms. This activity can be used when reflecting on the efficiency of different sorting algorithms. In this case, no values are moved or swapped (as they must be in a list). The sort is accomplished by the way the tree is built and the order in which nodes are visited in a traversal. Beautiful!

A new subject offers lots of potential for research on the part of teachers. Throughout the activities in this unit there is encouragement to consider issues, reflect on classroom practice and engage in action research. Not only does it contribute to practitioner's professional development, but can provide a key part of the evidence required for teacher accreditation via the BCS Certificate in Computer Science Teaching.

Preparation required:
Publicity for the BCS Certificate available for all attendees.
Familiarity with the questions to be posed and discussions to facilitate.

Points To Ponder

Scattered throughout the trainer's presentation are 'Points To Ponder' slides. Usually at the end of an activity, they have a dual purpose, both practical and professional. From a practical point of view, by posing a question for short discussion, they provide the presenter with a few minutes to prepare for the next activity and gather their thoughts. Moreover, they encourage attendees to converse with each other. The aim is not to engage in a lengthy discussion, but to plant ideas that could be pursued as classroom research. Near the end of the presentation, we hope you will raise the value of such action research.

In this unit, the questions posed at the end of an activity are:

The Perfect Shuffle: What are the affordances of kinaesthetic activities?

Data Structures Matter: There are many ways to illustrate abstraction… and many types. How can we move from examples of abstraction to applying it as a computational technique?

Many Problems, One Solution: Can we teach problem solving? There is no set of rigid rules for solving problems. Recognising that a problem is identical to another is partly experience. Deciding what to include in an abstraction, for example, is a judgement. So what are we teaching? Are we just exposing children to lots of examples in the hope something 'rubs off'?

Toy Problems For The Real World: Can Computing provide a different perspective on maths? Can it help develop mathematical understanding? This particular question is very pertinent given the current primacy of performance tables and pressures in many schools to cut curriculum time for other subjects.

Class Based Research

Why? Before embarking on the last activity, it is worth pausing to prompt a discussion about the value of classroom research both to support continuing professional development and to inform the wider teaching community. Use the terms "teacher enquiry", "action research", "reflective practitioner" and "practitioner research". Teaching computer programming is a relatively new activity. Many teachers are experiencing it for the first time. Those with experience can offer valuable support, advice and information to other colleagues. There are lots of opportunities for colleagues, old and new to contribute to an emerging pedagogy.

Inform attendees of the CAS Teacher Inquiry in Computing Education project. It provides a forum and focus for research in the field of computing. It can be accessed from the home page of the CAS website, under the link to projects: http://www.computingatschool.org.uk/

Research an area in which teachers have ready access to the data but be aware of researcher bias and local ethical considerations. Don't try to prove something you believe – always think that you are "bringing a better understanding" of the situation – it helps avoid bias.

What? Emphasise the key words such as “find out”, “evaluate” and “compare” to spark ideas. Each of the following give examples for a possible focus for research:

Find out about pupil preconceptions. Why do they think they are learning about computational thinking?

Evaluate an unplugged resource. Focus your research on something you are trying out. Start by creating a question. Develop sub-questions that bring more detail or further focus your research.

Compare school data about the pupils with performance in computational thinking activities. There is a lot of data in school. Devise an analysis that will enhance your job and provide interesting perspectives. Maybe the Bebras competition could provide an interesting comparison here?

Design, implement and evaluate a scheme of work that focuses on algorithmic thinking and design. Evaluating a unit of work you develop has obvious practical benefits.

How? Briefly tell the audience of different methods. Choose one of the examples from the suggestions above, or those raised in ‘Points To Ponder’ and discuss which methods might be used:

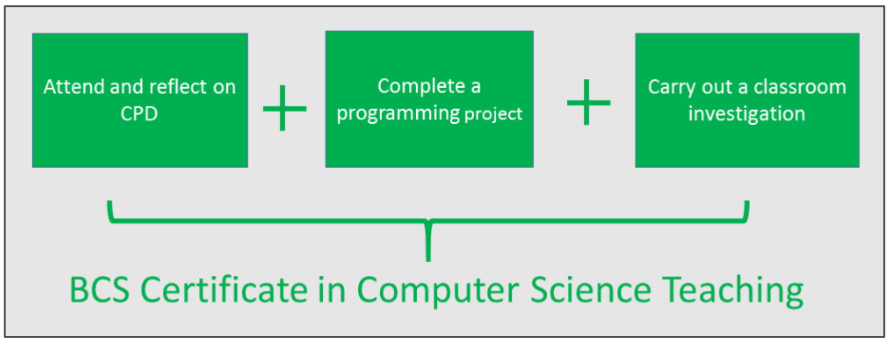
Questionnaires (including online) and surveys, interviews, focus groups (and online forums, wikis), scrutiny of documents including pupils’ work and blogs, scrutiny of numerical data and observation are all common techniques.

Less common but very effective might be analysis of professional conversations (in meetings, by email or via forums), analysis of pupil interactions/engagement in VLEs and forums, discourse analysisand content analysis.

Remember – all research has ethical considerations. Remind attendees of a school’s requirements and permissions regarding information about pupils and parents, in particular the idea of informed consent and freedom to withdraw their data from the research process. Remind them that if they embark on a University course then they will be governed by their ethics policy. Similarly, if completing the BCS Certificate then there is an explicit ethical requirement.

BCS Certificate In Computer Science Teaching

End with a reminder of the BCS Certificate in Computer Science Teaching – ensure hand-outs are available. One part of the evidence for the award is classroom research undertaken by the teacher. The other two areas involve attending CPD – like today, and completing a manageable programming project.


Attend and reflect on CPD
+
Complete a programming project
+
Carry out a classroom investigation
BCS Certificate in Computer Science Teaching

If that sounds a little daunting, remind teachers they can opt for either an independent or guided route. The latter means they have support from a mentor who can help guide them through the requirements. It is a valued award, giving professional recognition, accredited by the BCS, The Chartered Institute for IT. More importantly, it’s designed to help teachers in the work they are developing in school.

Encourage teachers to seek support from their schools to gain accreditation. Ensure they raise it as part of the performance management cycle.

More information is available at: http://www.computingatschool.org.uk/certificate.