M269

Programming & Efficiency Topics

Contents

1 Meeting Agenda

- Revue of session on Binary Trees
- Exponentials and Logarithms
- Programming points
- Height balanced (AVL) trees
- Future topics

2 Exponentials and Logarithms

2.1 Exponentials and Logarithms — Definitions

- **Exponential function** $y = a^x$ or $f(x) = a^x$
- $a^n = a \times a \times \cdots \times a$ (n a terms)
- **Logarithm** reverses the operation of exponentiation
- $\log_a y = x$ means $a^x = y$
- $\log_a 1 = 0$
- $\log_a a = 1$
- Method of logarithms propounded by John Napier from 1614
- Log Tables from 1617 by Henry Briggs
- Slide Rule from about 1620–1630 by William Oughtred of Cambridge
- *Logarithm* from Greek *logos* ratio, and *arithmos* number (Chanbers Dictionary 13th edition 2014)

2.2 Rules of Indices

1. $a^m \times a^n = a^{m+n}$
2. $a^m \div a^n = a^{m-n}$
3. $a^{-m} = \dfrac{1}{a^m}$
4. $a^{\frac{1}{m}} = \sqrt[m]{a}$
5. $(a^m)^n = a^{mn}$
6. $a^{\frac{n}{m}} = \sqrt[m]{a^n}$
7. $a^0 = 1$ where $a \neq 0$

- **Exercise** Justify the above rules
- What should 0^0 evaluate to ?
- See Wikipedia: Exponentiation
- The *justification* above probably only worked for whole or *rational* numbers — see later for exponents with *real* numbers (and the value of *logarithms*, *calculus*. . .)

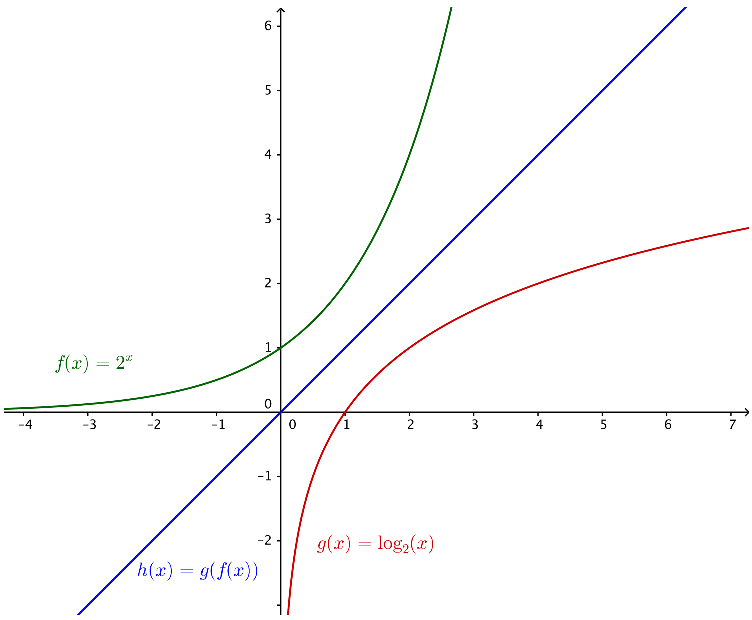
2.3 Logarithms — Motivation

- Make arithmetic easier — turns multiplication and division into addition and subtraction (see later)
- Complete the range of elementary functions for differentiation and integration

- An elementary function is a function of one variable which is the composition of a finite number of arithmetic operations ((+), (−), (×), (÷)), exponentials, logarithms, constants, and solutions of algebraic equations (a generalization of nth roots).
- The elementary functions include the trigonometric and hyperbolic functions and their inverses, as they are expressible with complex exponentials and logarithms.
- See A Level FP2 for Euler's relation $e^{i\theta} = \cos\theta + i\sin\theta$
- In A Level C3, C4 we get $\int \frac{1}{x} = \log_e |x| + C$
- e is Euler's number 2.71828...

2.4 Exponentials and Logarithms — Graphs

- See GeoGebra file `expLog.ggb`



2.5 Laws of Logarithms

- **Multiplication law** $\log_a xy = \log_a x + \log_a y$
- **Division law** $\log_a \left(\frac{x}{y}\right) = \log_a x - \log_a y$
- **Power law** $\log_a x^k = k \log_a x$

- **Proof of Multiplication Law**

$$
\begin{aligned}
x &= a^{\log_a x} \\
y &= a^{\log_a y} && \text{by definition of log} \\
xy &= a^{\log_a x} a^{\log_a y} \\
&= a^{\log_a x + \log_a y} && \text{by laws of indices}
\end{aligned}
$$

Hence $\log_a xy = \log_a x + \log_a y$ by definition of log

2.6 Arithmetic and Inverses

- *Notation helps or maybe not ?*
- **Addition** $\mathrm{add}(b, x) = x + b$
- **Subtraction** $\mathrm{sub}(b, x) = x - b$
- Inverse $\mathrm{sub}(b, \mathrm{add}(b, x)) = (x + b) - b = x$
- **Multiplication** $\mathrm{mul}(b, x) = x \times b$
- **Division** $\mathrm{div}(b, x) = x \div b = \frac{x}{b} = x/b$
- Inverse $\mathrm{div}(b, \mathrm{mul}(b, x)) = (x \times b) \div b = \frac{(x \times b)}{b} = x$
- **Exponentiation** $\exp(b, x) = b^x$
- **Logarithm** $\log(b, x) = \log_b x$
- Inverse $\log(b, \exp(b, x)) = \log_b(b^x) = x$
- *What properties do the operations have that work (or not) with the notation ?*

Arithmetic Operations — Commutativity and Associativity

- **Commutativity** $x \circledast y = y \circledast x$
- **Associativity** $(x \circledast y) \circledast z = x \circledast (y \circledast z)$
- $(+)$ and $(\times)$ are *semantically* commutative and associative — so we can leave the brackets out
- $(-)$ and $(\div)$ are not
- Evaluate $(3 - (2 - 1))$ and $((3 - 2) - 1)$
- Evaluate $(3/(2/2))$ and $((3/2)/2)$
- We have the *syntactic* ideas of left (and right) associativity
- We choose $(-)$ and $(\div)$ to be left associative
- $3 - 2 - 1$ means $((3 - 2) - 1)$
- $3/2/2$ means $((3/2)/2)$
- **Operator precedence** is also a choice (remember BIDMAS or BODMAS ?)
- If in doubt, put the brackets in

Exponentials and Logarithm — Associativity

- What should 2^{3^4} mean ?
- Let $b \wedge x \equiv b^x$
- Evaluate $(2 \wedge 3) \wedge 4$ and $2 \wedge (3 \wedge 4)$
- Evaluate $c = \log_b(\log_b((b \wedge b) \wedge x))$
- Evaluate $d = \log_b(\log_b(b \wedge (b \wedge x)))$
- Beware spreadsheets Excel and LibreOffice here
- $(2^3)^4 = 2^{12}$ and $2^{3^4} = 2^{81}$
- Exponentiation is not semantically associative
- We *choose* the syntactic left or right associativity to make the syntax nicer.
- Evaluate $c = \log_b(\log_b((b \wedge b) \wedge x))$
- $c = \log_b(x \log_b(b^b)) = \log_b(x \cdot (b \log_b b)) = \log_b(x \cdot b \cdot 1)$
- Hence $c = \log_b x + \log_b b = \log_b x + 1$
- Not symmetrical (unless b and x are both 2)
- Evaluate $d = \log_b(\log_b(b \wedge (b \wedge x)))$
- $d = \log_b((b \wedge x)(\log_b b)) = \log_b((b \wedge x) \times 1)$
- Hence $d = \log_b(b \wedge x) = x(\log_b b) = x$
- Which is what we want — so exponentiation is *chosen* to be right associative

2.7 Change of Base

- **Change of base**

  $$\log_a x = \frac{\log_b x}{\log_b a}$$

  Proof: Let $y = \log_a x$

  $$\begin{aligned} a^y &= x \\ \log_b a^y &= \log_b x \\ y \log_b a &= \log_b x \\ y &= \frac{\log_b x}{\log_b a} \end{aligned}$$

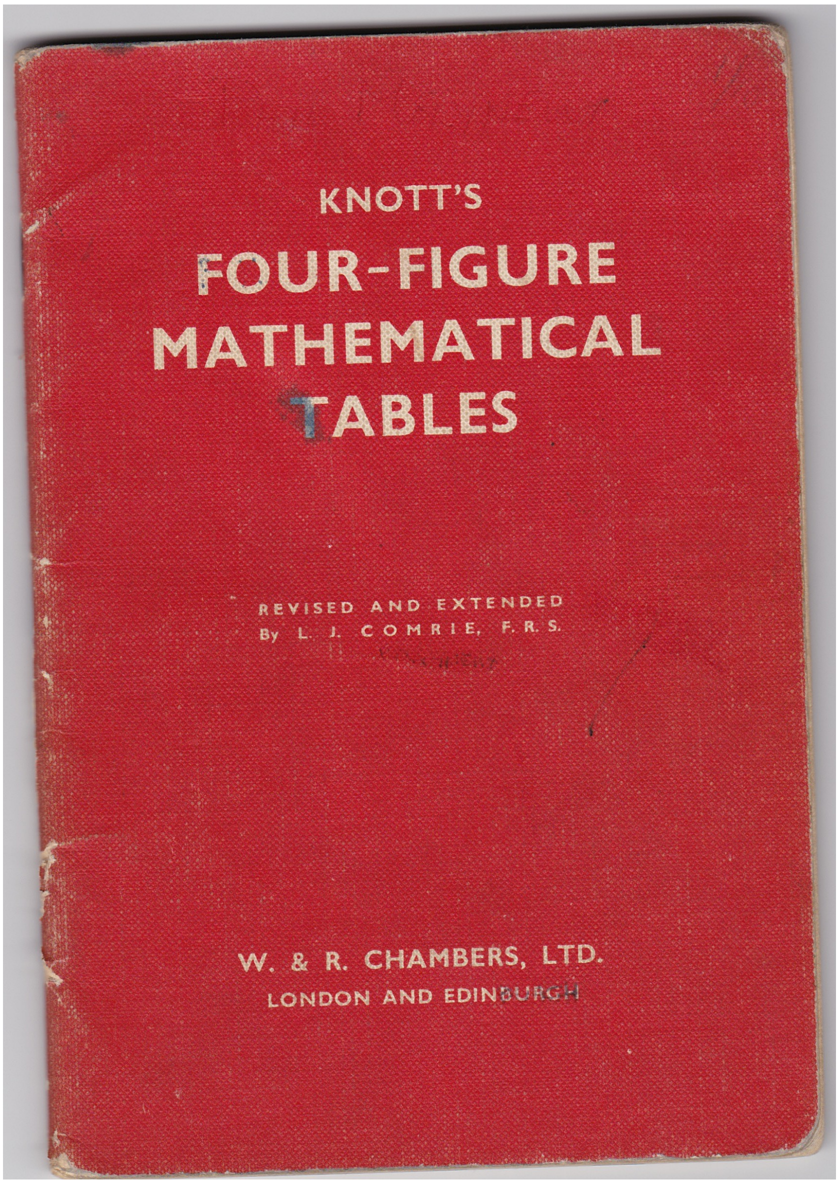
- Given x, $\log_b x$, find the base b
  - $b = x^{\frac{1}{\log_b x}}$
- $\log_a b = \dfrac{1}{\log_b a}$

3 Before Calculators and Computers

- We had computers before 1950 — they were *humans* with pencil, paper and some further aids:
- **Slide rule** invented by William Oughtred in the 1620s — major calculating tool until pocket calculators in 1970s
- **Log tables** in use from early 1600s — method of logarithms propounded by John Napier
- **Logarithm** from Greek *logos* ratio, and *arithmos* number

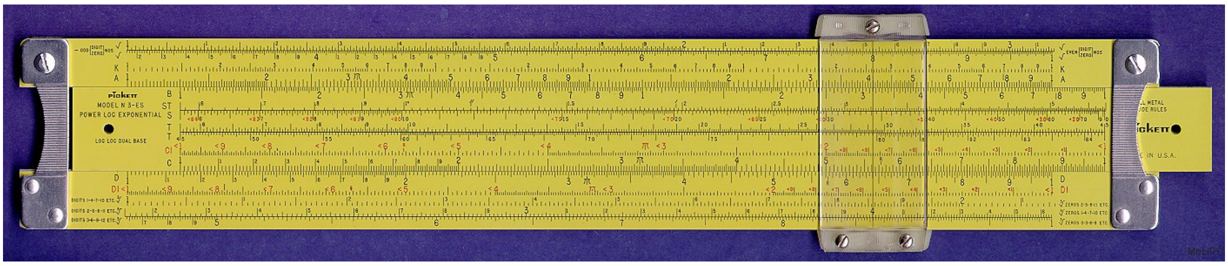
3.1 Log Tables

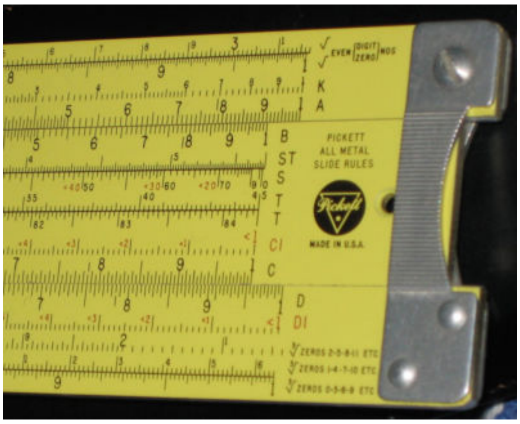
Knott's Four-Figure Mathematical Tables



3.2 Slide Rules

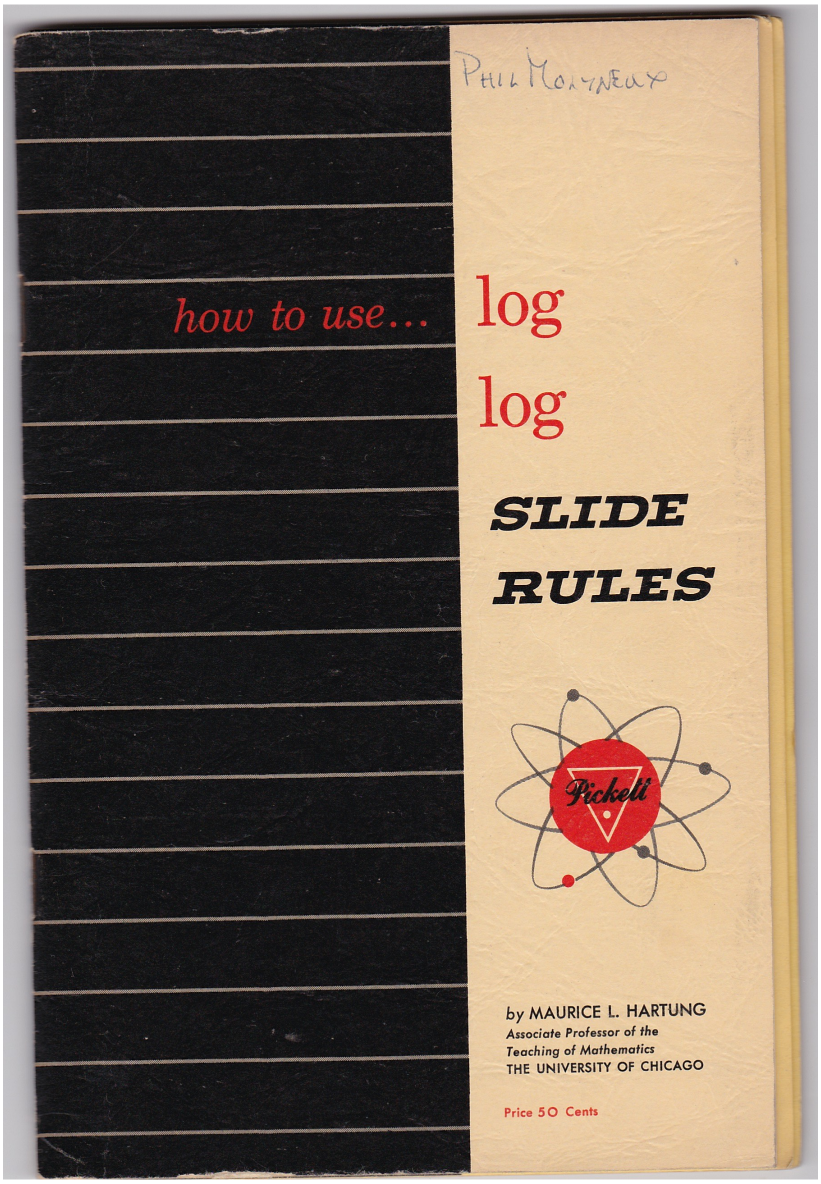
Pickett N 3-ES from 1967





- See Oughtred Society
- UKSRC
- Rod Lovett's Slide Rules
- Slide Rule Museum

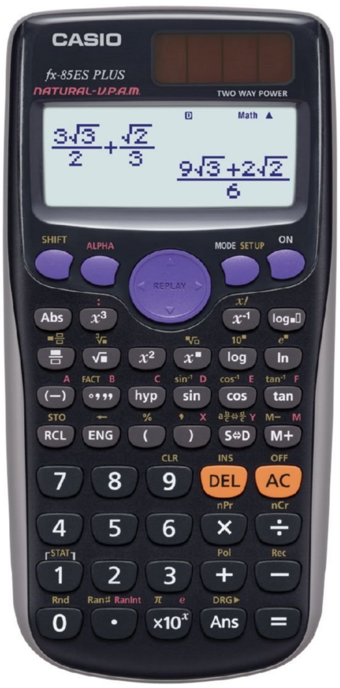
Pickett log log **Slide Rules Manual 1953**



3.3 Calculators

HP HP-21 Calculator from 1975 £69

Casio fx-85GT PLUS Calculator from 2013 £10



Calculator links

- **HP Calculator Museum** http://www.hpmuseum.org
- **HP Calculator Emulators** http://nonpareil.brouhaha.com
- **HP Calculator Emulators for OS X** http://www.bartosiak.org/nonpareil/
- **Vintage Calculators Web Museum** http://www.vintagecalculators.com

3.4 Example Calculation

- Evaluate 89.7×597
- **Knott's Tables**
- $\log_{10} 89.7 = 1.9528$ and $\log_{10} 597 = 2.7760$
- Shows *mantissa* (decimal) & *characteristic* (integral)
- Add 4.7288, take *antilog* to get $5346 + 10 = 5.356 \times 10^4$
- **HP-21 Calculator** — set display to 4 decimal places
- 89.7 [log] = 1.9528 and 597 [log] = 2.7760
- [+] displays 4.7288
- 10 [ENTER], [x ⇄ y] and [y^x] displays 53550.9000
- **Casio fx-85GT PLUS**
- [log] 89.7 [)] = 1.952792443 [+] [log] 597 [)] = 2.775974331 [=]
- 4.728766774 [Ans]+[10^x] gives 53550.9

4 Basic Computational Components

Computational Components — Imperative

Imperative or procedural programming has statements which can manipulate global memory — statements can be organised into procedures (or functions)

- **Sequence** of statements

```
stmnt ; stmnt
```

- **Iteration** to repeat statements

```
while expr :
  suite

for targetList in exprList :
  suite
```

- **Selection** choosing between statements

```
if expr : suite
elif expr : suite
else : suite
```

Functional programming treats computation as the evaluation of expressions and the definition of functions (in the mathematical sense)

- **Function composition** to combine the application of two or more functions — like sequence but from right to left (notation accident of history)

```
(f . g) x = f (g x)
```

- **Recursion** — function definition defined in terms of calls to itself (with *smaller* arguments) and base case(s) which do not call itself.

- **Conditional expressions** choosing between alternatives expressions

```
if expr then expr else expr
```

4.1 Writing Programs & Thinking

The Steps

1. Invent a *name* for the program (or function)
2. What is the *type* of the function ? What sort of *input* does it take and what sort of *output* does it produce ?
3. Invent *names* for the input(s) to the function (*formal parameters*)
4. *Think* of the definition of the function body.

The *Think* Step

0. Think of an example or two — what should the program/function do ?
1. Don't think too much at one go — break the problem down. Top down design, stepwise refinement.
2. What are the inputs — describe all the cases.
3. Investigate choices. What data structures ? What algorithms ?
4. Use common tools — bottom up synthesis.
5. Spot common function application patterns — generalise & then specialise.
6. Look for good *glue* — to combine little programs to make bigger ones.
7. Try out your first examples when you have written the program

5 Divide and Conquer

Binary Search — Exercise

Given the *Python* definition of `binarySearchRec` from the slides for the Unit 1 tutorial, trace an evaluation of `binarySearchRec(xs, 25)` where `xs` is

```
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
```

Binary Search Recursive

```
1  def binarySearchRec(xs,val,lo=0,hi=-1):
2    if (hi == -1):
3      hi = len(xs) - 1

5    mid = (lo + hi) // 2

7    if hi < lo:
8      return None
9    else:
10     guess = xs[mid]
11     if val == guess:
12       return mid
13     elif val < guess:
14       return binarySearchRec(xs,val,lo,mid-1)
15     else:
16       return binarySearchRec(xs,val,mid+1,hi)
```

Binary Search — Solution

```
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs, 25)
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs,25,8,14) by line 15
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs,25,8,10) by line 15
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs,25,8,8) by line 13
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs,25,8,7) by line 13
Return value: None by line 7
```

6 String Search

- *Sunday Quick Search* algorithm
- *Knuth-Morris-Pratt (KMP)* algorithm
- Exact String Matching Algorithms — animations in Java

6.1 Sunday Quick Search Algorithm

Example

- Sunday Quick Search example from M269 2014J TMA02
- Consider the alphabet C, G, A, T and a target string CGTACTCGTAGT.
- Calculate the shift table for this search, explaining in detail how you used the part of the algorithm that builds the table to derive your results.
- Given the target string CGTACTCGTAGT, the search string CGTACTCGGCGTAAAGT-GCGTCTT and the shift table calculated above

- describe, with diagrams if necessary, how the first attempt to match the target string against the search string fails
- and how the target string slides along the search string for the second matching attempt.

Sunday Quick Search Shift Table

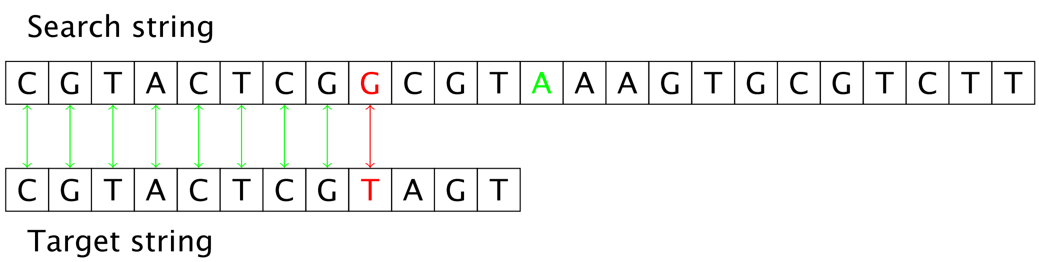
- The shift table for the Sunday Quick Search algorithm:
  - If the character does not appear in the target string T, the shift distance is one more than the length of T
  - If the character does appear in T the shift distance is the first position at which it appears, counting from right to left and starting at 1
- Shift table:

A	C	G	T
3	6	2	1

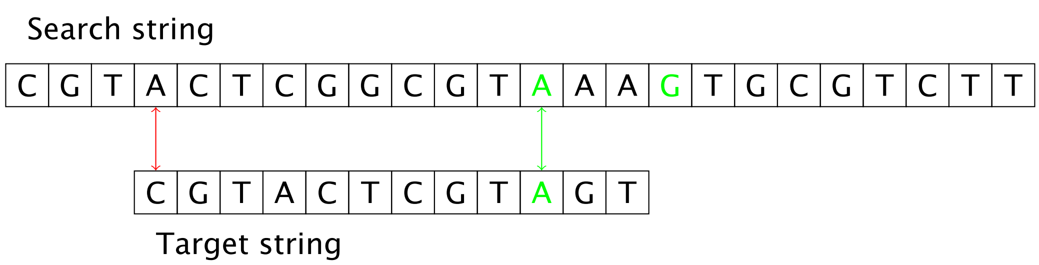
- See the example at Quick Search algorithm

Calculating the Shift

- Given the following alignment of *Search* and *Target* strings



- The next character in the *Search* string after the *Target* string is A so the shift will be 3 places



6.2 Knuth-Morris-Pratt (KMP) Algorithm

Example

- Knuth-Morris-Pratt (KMP) algorithm example from M269 2014J TMA02
- Consider the alphabet C, G, A, T and a target string CGTACTCGTAGT.
- Calculate the prefix table for this target string,

- explaining in detail how you derived the sixth, seventh and eighth entries in your table.

KMP Prefix Table

- The KMP *prefix table* takes the *target string*, t, and a *position*, p, in t and returns an integer k
- k is the length of the maximum proper prefix of t up to position p that is a suffix of t up position p
- We define a prefix function to take a *target string*, t, and a number $k \geq 0$, which is the number of items off the front of t
- Formally, if our position index is 0 based we have:
- $\text{prefixTable}(t, 0) = 0$
- $\text{prefixTable}(t, p) = \max\{k : k \leq p \wedge \text{prefix}(t, k) \sqsupset \text{prefix}(t, p + 1)\}$
- Here $\sqsupset$ means *proper suffix* — that is a suffix that does not include the whole string

KMP Shift Function

- M269 does not give the KMP shift function (or table) — we give it here for interest
- The shift function takes the *target string*, t, and the number of characters in the target string that have been matched, q and returns the shift.
- $\text{shift}(t, 0) = 1$
- $\text{shift}(t, q) = q - \text{prefixTable}(t, q - 1)$
- This assumes 0 based list indexing
- The notation here comes from Cormen et al. (2009, 2009, section 32.4, page 1004) — this uses 1 based list indexing and also provides code.
- Cormen uses the terminology *Text* for *Search string* and *Pattern* for *Target string*

KMP Prefix and Shift Table

	C	G	T	A	C	T	C	G	T	A	G	T	Target string
	0	1	2	3	4	5	6	7	8	9	10	11	Position (Index), p
0	1	2	3	4	5	6	7	8	9	10	11	12	Match, q
	0	0	0	0	1	0	1	2	3	4	0	0	prefixTable(t, p)
1	1	2	3	4	4	6	6	6	6	6	11	12	shift(t, q)

7 Future Work

- Reflections on today's topics — what were the important points ?
- Dates for next meeting: (??)
- Next topics
  - Unit 5 Optimisation
  - Graph algorithms
  - Critique of Phil's notes

8 Web Sites & References

8.1 Web Sites

- *Python Documentation* https://docs.python.org/3/ (26 January 2016)
- *Wikipedia Category: Python Libraries* https://en.wikipedia.org/wiki/Category:Python_libraries (26 January 2016)
- *Python Wiki: Useful Modules* https://wiki.python.org/moin/UsefulModules (26 January 2016)
- *Python Packaging User Guide* http://python-packaging-user-guide.readthedocs.org/e (26 January 2016)
- *Python Packaging Authority* https://www.pypa.io/en/latest/ (26 January 2016)
- *PyPI Python Package Index* https://pypi.python.org/pypi (26 January 2016)
- *Top 100 Tools for Learning 2015* http://c4lpt.co.uk/directory/top-100-tools/ (26 January 2016)
- *Pygal* Python charting http://www.pygal.org/en/latest/ (26 January 2016)
- *21 Ridiculously Impressive HTML5 Canvas Experiments* http://code.tutsplus.com/article ridiculously-impressive-html5-canvas-experiments--net-14210 (26 January 2016)
- *graph-tool* Efficient network analysis https://graph-tool.skewed.de (26 January 2016)
- *Python Patterns — Implementing Graphs* https://www.python.org/doc/essays/graphs/(26 January 2016)
- *Graphs in Python* http://www.python-course.eu/graphs_python.php
- *Stackoverflow Python Graph Library* http://stackoverflow.com/questions/606516/pytho graph-library (26 January 2016)
- *Dijkstra's algorithm for shortest paths* http://code.activestate.com/recipes/119466-dijkstras-algorithm-for-shortest-paths/

References

Adelson-Velskii, G M and E M Landis (1962). An algorithm for the organization of information. In *Doklady Akademia Nauk SSSR*, volume 146, pages 263–266. Translated from *Soviet Mathematics — Doklady*; 3(5), 1259–1263.

Bird, Richard (1998). *Introduction to Functional Programming using Haskell*. Prentice Hall, second edition. ISBN 0-13-484346-0.

Bird, Richard (2014). *Thinking Functionally with Haskell*. Cambridge University Press. ISBN 1107452643. URL http://www.cs.ox.ac.uk/publications/books/functional/.

Bird, Richard and Phil Wadler (1988). *Introduction to Functional Programming*. Prentice Hall, first edition. ISBN 0-13-484197-2.

Cormen, Thomas H.; Charles E. Leiserson; Ronald L. Rivest; and Clifford Stein (2009). *Introduction to Algorithms*. MIT Press, third edition. ISBN 0262533057. URL http://mitpress.mit.edu/books/introduction-algorithms.

Hudak, Paul; John Hughes; Simon Peyton Jones; and Phil Wadler (2007). A History of Haskell: Being Lazy with Class. In *Proceedings of the third ACM SIGPLAN conference on History of programming languages*, pages 12–1–12–55. ACM New York, NY, USA.

Lee, Gias Kay (2013). Functional Programming in 5 Minutes. Web. http://gsklee.im, URL http://slid.es/gsklee/functional-programming-in-5-minutes.

Lutz, Mark (2011). *Programming Python*. O'Reilly, fourth edition. ISBN 0596158106. URL http://learning-python.com/books/about-pp4e.html.

Lutz, Mark (2013). *Learning Python*. O'Reilly, fifth edition. ISBN 1449355730. URL http://learning-python.com/books/about-lp5e.html.

Marlow, Simon and Simon Peyton Jones (2010). Haskell Language and Library Specification. Web. URL http://www.haskell.org/haskellwiki/Language_and_library_specification.

Miller, Bradley W. and David L. Ranum (2011). *Problem Solving with Algorithms and Data Structures Using Python*. Franklin, Beedle Associates Inc, second edition. ISBN 1590282574. URL http://interactivepython.org/courselib/static/pythonds/index.html.

O'Donnell, John; Cordelia Hall; and Rex Page (2006). *Discrete Mathematics Using a Computer*. Springer, second edition. ISBN 1846282411. URL http://www.dcs.gla.ac.uk/~jtod/discrete-mathematics/.

Okasaki, Chris (1998). *Purely Functional Data Structures*. Cambridge University Press. ISBN 0-521-63124-6.

O'Sullivan, Bryan; John Goerzen; and Donald Stewart (2008). *Real World Haskell*. O'Reilly, first edition. ISBN 0596514980. URL http://book.realworldhaskell.org/.

Thompson, Simon (2011). *Haskell the Craft of Functional Programming*. Addison Wesley, third edition. ISBN 0201882957. URL http://www.haskellcraft.com/craft3e/Home.html.

van Rossum, Guido and Fred Drake (2011a). *An Introduction to Python*. Network Theory Limited, revised edition. ISBN 1906966133.

van Rossum, Guido and Fred Drake (2011b). *The Python Language Reference Manual*. Network Theory Limited, revised edition. ISBN 1906966141.