Programming Design and Implementation

Lecture 10: Real World Applications

Updated: 7th June, 2020

Mark Upston
Discipline of Computing
School of Electrical Engineering, Computing and Mathematical Sciences (EECMS)

Outline

What is an Algorithm?

- Al Khwarizmi (9^{th} century Persian mathematician, Bagdad) wrote a textbook (in Arabic) about basic methods for adding, multiplying, and dividing numbers, extracting square roots, and calculating digits of π.
  - Al Khwarizmi, when written in Latin, the name became Algorismus / Algoritmi
- An algorithm is any well-defined computational procedure that
  - Takes some value as input
  - Produces some value as output
  - Solves a specified computational problem
- An algorithm
  - Must be correct (i.e., always gives the right result)
  - Should be tractable & terminate (i.e., gives a result in reasonable time)
  - Can be specified in English, as computer program, or as hardware design

Our Definition of an Algorithm

- An algorithm is a set of detailed, unambiguous, ordered steps specifying a solution to a problem
  - Steps must be stated precisely, without ambiguity
  - Enter at the start & exit at the bottom
  - English description independent of any programming language
  - Non trivial problem will need several stages of refinement
  - Various methodologies available
  - Must be desk-checked for correctness

Algorithm - Pseudo Code

- In Curtin computing, Algorithms are expressed in Pseudo Code:
  - *But they don't have to be, as you will find out in this lecture*
  - English like phrases which describe the algorithm steps
  - The pseudo code is evolved from a rough description to something which almost looks like a programming language
  - Pseudo code development is about refinement
    - Developing an algorithm is a journey where the problem
  - Algorithm design is an art that takes a lot of practice

Problem Example: Sorting

- Sorting Problem *(A problem we solved recently)*

  Input: A sequence of n numbers (a_1, a_2, ..., a_n)
  Output: A reordering (b_1, b_2, ..., b_n) of the INPUT sequence such that $b_1 \leq b_2 \leq \ldots \leq b_n$
- Example:
  - **Input:** (30, 20, 41, 51, 3, 20)
  - **Output:** (3, 20, 20, 30, 41, 51)

Algorithm: Insertion Sort

```
SUBMODULE: insertionSort
IMPORT: array (ARRAY OF X)
EXPORT: array (ARRAY OF X)
ASSERTION: array will be sorted using Insertion Sort
ALGORITHM:
    FOR nn := 1 TO array.length - 1 INC BY 1
        ii := nn
        temp := array[ii]
        WHILE (ii > 0) AND (array[ii-1] > temp)
            array[ii] := array[ii-1]
            ii := ii - 1
        END WHILE
        array[ii] := temp
    END FOR
END insertionSort
```

Application

1	2	3	4	5	6
5	**2**	4	6	1	3

$\longrightarrow$

1	2	3	4	5	6
2	5	**4**	6	1	3

1	2	3	4	5	6
2	4	5	**6**	1	3

$\longrightarrow$

1	2	3	4	5	6
2	4	5	6	**1**	3

1	2	3	4	5	6
1	2	4	5	6	**3**

$\longrightarrow$

1	2	3	4	5	6
1	2	3	4	5	6

Problem Example: GCD

- GCD - <u>G</u>reatest <u>C</u>ommon <u>D</u>ivisor *(Learnt in High School)*

  Input: Integers `X` and `Y`
  Output: The largest integer `Z` that divides both `X` and `Y`, i.e.,
  `Z = GCD(X, Y)`
  - Note: `GCD(X, Y) = GCD(Y, X)`

Algorithm

1. Find all prime factors of **<u>both</u>** `X` and `Y`
2. Multiply all **<u>common</u>** prime factors to form `Z`

Algorithm - Prime Numbers

```
// Test to find a prime number (n)
prime := TRUE
for ii := 2 TO n INC BY 1
    IF n MOD ii EQUALS 0
        prime := FALSE
```

Application

- Example:
  - **Input:** X = 1035, Y = 759
  - **Output:** Z = `GCD(1035, 759)` = 69
- Find the GCD of X = 1035 and Y = 759

1. $X = 1035 = 3^2 * 5 * 23$
1. $Y = 759 = 3 * 11 * 23$
   - The common prime factors are: **3** and **23**
2. $Z = 3 * 23 =$ **69**

Problem Example: GCF

- GCF - **G**reatest **C**ommon **F**actor *(Learnt in High School)*

  Also known as the Least Common Multiple (LCM)

  Input: Integers X and Y
  Output: The smallest integer **`Z`** divisible by both **`X`** and **`Y`**, i.e., **`Z = LCM(X, Y)`**

  - Note: **`LCM(X, Y) = LCM(Y, X)`**

Algorithm

1. Find all prime factors of **<u>both</u>** **X** and **Y**
2. Multiply **<u>all</u>** prime factors to form **Z**
   - For each prime factor common to **X** and **Y**, use the largest power.

Application

- Example:
  - **Input:** X = 1035, Y = 759
  - **Output:** Z = `GCF(1035, 759)` = 11385
- Find the GCF of X = 1035 and Y = 759

1. $X = 1035 = 3^2 * 5 * 23$
1. $Y = 759 = 3 * 11 * 23$
2. $Z = 3^2 * 5 * 11 * 23 = \underline{\mathbf{11385}}$

Algorithm (2)

- We can use the solution of `GCD(X, Y)` to compute `LCM(X, Y)` as follows
- `LCM(X, Y) = (X * Y) / GCD(X, Y)`

Application (2)

- Example:
  - **Input:** X = 1035, Y = 759
  - **Output:** `LCM(1035, 759) = (1035 * 759)/GCD(1035, 759)`
    - From the previous example, we have `GCD(1035, 759) = 69`
- = (1035 * 759) / 69 = <u>**11385**</u>

Problem Example: Integer Multiplication

- Integer Multiplication *(Learnt in Primary School)*
  Input: Integers X and Y
  Output: **Z = X * Y**

Algorithm

1. Multiply each digit with every other digit, carrying values.
2. Add the results

Application

- Example:
  - **Input:** X = 12, Y = 34
  - **Output:** Z = 408

```
    1 2
    3 4 x
  -------
    4 8
  3 6 0 +
 ---------
  4 0 8
```

Algorithm (2) - Al Khwarizmi' Algorithm

1. Divide the first number by 2 (in Col 1)
2. Double the second number (in Col 2)
3. Repeat until the first number becomes 1
4. Add all rows in Col 2 that has odd number in Col 1

Application (2)

- Example:

12 * 34 = 408

Col 1	Col 2
12	34
6	68
*3	136
*1	272
Result =	408

25 * 70 = 1750

Col 1	Col 2
*25	70
12	140
6	280
*3	560
*1	1120
Result =	1750

Problem Example: Addition of Consecutive Numbers

- Addition of **n** consecutive numbers 1, 2, 3, ..., n

  Input: Integer **n**

  Output: 1 + 2 + 3 + ... + **n** - 1 + **n**

Algorithm

1. Consecutively add the numbers.

Application

- Example:
  - **Input:** **n** = 10
  - **Output:** 55

  1 + 2 + 3 + ... + 9 + 10 = **<u>55</u>**

Algorithm (2) - Gauss

- Some say Carl Friedrich Gauss knew the algorithm when he was 8 years old

1. Follow the formula: $\frac{n(n+1)}{2}$.

Application (2)

- Example:
  - **Input:** $\mathbf{n} = 10$
  - **Output:** 55
- $1 + 2 + 3 + \ldots + \mathbf{n} = \frac{n(n+1)}{2}$
- Thus, $1 + 2 + 3 + \ldots + 9 + 10 = \frac{10(10+1)}{2} = \underline{\mathbf{55}}$

What are some Other Problems?

- How many digits are there in Pi (π)? $\pi = 3.14159265\ \ldots$
  - In 2020, the record was more than 31 trillion digits
  - We have already discussed 2 algorithms to calculate π *(or $\frac{\pi}{4}$)*
- How many digits are there in Phi (ϕ - pronounced fi)? Million digits!
  - ϕ is the Golden Ratio
    - Also called the Golden Number, Golden Proportion, Golden Mean, Golden Section
  - $\phi = 1 + \frac{1}{\phi}$ & $\phi^2 - \phi - 1 = 0$ & $\phi = \frac{1+\sqrt{5}}{2} = 1.618803398874989$
  - Or $\frac{1}{\phi} = 0.618803398874989$
  - The ratio of each successive pair of Fibonacci numbers approximates phi, e.g., $\frac{2584}{1597} = 1.618033813$

Levenshtein Distance

- In information theory, linguistics and computer science, the Levenshtein distance is a string metric for measuring the difference between two sequences.
- Informally, the Levenshtein distance between two words is the minimum number of single-character edits (insertions, deletions or substitutions) required to change one word into the other.
- The Levenshtein distance between two strings a, b (of length $|a|$ and $|b|$ respectively) is given by $\mathrm{lev}_{a,b}(|a|, |b|)$ where

$$\mathrm{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \mathrm{lev}_{a,b}(i-1,j)+1 \\ \mathrm{lev}_{a,b}(i,j-1)+1 \\ \mathrm{lev}_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Applications of Levenshtein Distance

- For example, the Levenshtein distance between `"kitten"` and `"sitting"` is **3**, since the following three edits change one into the other, and there is no way to do it with fewer than three edits:

1. <u>**k**</u>`itten` → <u>**s**</u>`itten` (substitution of `"s"` for `"k"`)
2. <u>**s**</u>`itten` → `sitt`<u>**i**</u>`n` (substitution of `"i"` for `"e"`)
3. `sittin` → `sittin`<u>**g**</u> (insertion of `"g"` at the end)

Algorithm (Using a Table Based Approach)

```
SUBMODULE: calculateLevenshtein
IMPORT: x (String), y (String)
EXPORT: result (Integer)
ASSERTION: Will return the difference between 2 Strings.
ALGORITHM:
    dp := (2D ARRAY SIZE OF (LENGTH x), (LENGTH y))
    FOR ii := 0 TO LENGTH x INC BY 1
        FOR jj := 0 TO LENGTH y INC BY 1
            IF ii = 0
                dp[ii][jj] = jj
            ELSE IF jj = 0
                dp[ii][ii] = ii
            ELSE
                dp[ii][jj] = min <- (dp[ii - 1][jj - 1] +
                        costOfSubstution <- (x[ii - 1], y[jj - 1]),
                        dp[ii - 1][jj], dp[ii][jj - 1])
            END IF
        END FOR
    END FOR
    result := dp[LENGTH x][LENGTH y]
END calculateLevenshtein
```

Matrix Output

		k	**i**	**t**	**t**	**e**	**n**
	0	1	2	3	4	5	6
s	1	1	2	3	4	5	6
i	2	2	1	2	3	4	5
t	3	3	2	1	2	3	4
t	4	4	3	2	1	2	3
i	5	5	4	3	2	2	3
n	6	6	5	4	3	3	2
g	7	7	6	5	4	4	**3**

Burrows-Wheeler Transform

- The Burrows-Wheeler transform (BWT, also called block-sorting compression) rearranges a character string into runs of similar characters.
- This is useful for compression, since it tends to be easy to compress a string that has runs of repeated characters by techniques such as move-to-front transform and run-length encoding.
- More importantly, the transformation is reversible, without needing to store any additional data except the position of the first original character.
- The BWT is thus a "free" method of improving the efficiency of text compression algorithms, costing only some extra computation.

Applications of Burrows-Wheeler Transform

- The transform is done by sorting all the circular shifts of a text in lexicographic order and by extracting the last column and the index of the original string in the set of sorted permutations of S.

1. Given an input string **S** = **^BANANA|**
2. Rotate it N times
3. Where **N** = 8 is the length of the **S** string considering also the symbol ^ representing the start of the string and the red | character representing the `'EOF'` pointer; these rotations, or circular shifts, are then sorted lexicographically
4. The output of the encoding phase is the last column **L** = **BNN^AA|A** after step 3, and the index (0-based) **I** of the row containing the original string **S**, in this case **I** = 6

Applications of Burrows-Wheeler Transform (2)

<table>
<tr><th colspan="5">Transformation</th></tr>
<tr><th>1. Input</th><th>2. All rotations</th><th>3. Sort into lexical order</th><th>4. Take the last column</th><th>5. Output</th></tr>
<tr>
<td>^BANANA|</td>
<td>^BANANA|
|^BANANA
A|^BANAN
NA|^BANA
ANA|^BAN
NANA|^BA
ANANA|^B
BANANA|^</td>
<td>ANANA|^B
ANA|^BAN
A|^BANAN
BANANA|^
NANA|^BA
NA|^BANA
^BANANA|
|^BANANA</td>
<td>ANANA|^B
ANA|^BAN
A|^BANAN
BANANA|^
NANA|^BA
NA|^BANA
^BANANA|
|^BANANA</td>
<td>BNN^AA|A</td>
</tr>
</table>

Algorithm

- The following pseudocode gives a simple (though inefficient) way to calculate the BWT and its inverse. It assumes that the input String **s** contains a special character 'EOF' which is the last character and occurs nowhere else in the text.

```
function BWT (String s)
    create a table, rows are all possible rotations of s
    sort rows alphabetically
    return (last column of the table)
```

```
function inverseBWT (string s)
    create empty table
    repeat length(s) times
        // first insert creates first column
        insert s as a column of table before first column of the table
        sort rows of the table alphabetically
    return (row that ends with the 'EOF' character)
```

Applications of Algorithms in the Real World

- The algorithms that are shown in this lecture are just a snippet of what is used in the real world.
- Other curious algorithms to research are:
  - Needleman-Wunsch Algorithm
  - Trigraph Matching
  - Jaro
  - Elastic Potential Energy
  - Projectile Motion
  - OPRs and Least Squares Approximation
- Have a go at implementing some of these
- Your Convolution operation that we have been applying throughout this semester is another example of an algorithm that is crucial to us today!

Don't Forget!

- Assignment is due soon (See Specification)
  - Ensure you go to your registered practical next week, it is the only way we you can demonstrate your assignment (and receive a mark!)

Next Week

- The next Lecture will address the following:
  - Revision