Chapter 2: Algorithm Discovery and Design

BMM 105 Introduction to Computer Engineering, Fall 2018

Objectives

In this chapter, you will learn about

- Representing algorithms
- Examples of Algorithmic Problem Solving

Representing Algorithms

- Natural language
 - Language spoken and written in everyday life
 - Problems with using natural language for algorithms
 - Verbose
 - Imprecise
 - Relies on context and experiences to give precise meaning to a word or phase

Adding Two m-Digit Numbers



FIGURE 2.1

The Addition Algorithm of Figure 1.2 Expressed in Natural Language Initially, set the value of the variable *carry* to 0 and the value of the variable i to 0. When these initializations have been completed, begin looping as long as the value of the variable i is less than or equal to (m-1). First, add together the values of the two digits a_i and b_i and the current value of the carry digit to get the result called c_i . Now check the value of c_i to see whether it is greater than or equal to 10. If c_i is greater than or equal to 10, then reset the value of *carry* to 1 and reduce the value of c_i by 10; otherwise, set the value of *carry* to zero. When you are done with that operation, add 1 to i and begin the loop all over again. When the loop has completed execution, set the leftmost digit of the result c_m to the value of *carry* and print out the final result, which consists of the digits c_m $c_m-1 \ldots c_0$. After printing the result, the algorithm is finished, and it terminates.

Figure 2.1

The Addition Algorithm of Figure 1.2 Expressed in Natural Language

Representing Algorithms (continued)

- High-level programming language
 - Examples: C++, Java, Python
 - Problem with using a high-level programming language for algorithms
 - During the initial phases of design, we are forced to deal with detailed language issues

```
int i, m, Carry;
int[] a = new int[100];
int[] b = new int[100];
int[] c = new int[100];
m = Console.readInt();
for (int j = 0; j < = m-1; j++) {
         a[j] = Console.readInt();
         b[j] = Console.readInt();
Carry = 0;
i = 0;
while (i < m) {
         c[i] = a[i] + b[i] + Carry;
         if (c[i] > = 10)
```

Figure 2.2

The Beginning of the Addition Algorithm of Figure 1.2 Expressed in a High-Level Programming Language

```
Algorithm for Adding Two m-Digit Numbers
```

```
Given: m \ge 1 and two positive numbers each containing m digits, a_{m-1} a_{m-2} \dots a_n
and b_{m-1}, b_{m-2}, \dots b_{n}
Wanted: c_m c_{m-1} c_{m-2} \dots c_0, where c_m c_{m-1} c_{m-2} \dots c_0 = (a_{m-1} a_{m-2} \dots a_n) +
(b_{m-1}, b_{m-2}, \dots b_n)
Algorithm:
Step 1 Set the value of carry to 0.
Step 2 Set the value of i to 0.
Step 3 While the value of i is less than or equal to m-1, repeat the instructions in
           steps 4 through 6.
Step 4
                Add the two digits a; and b; to the current value of carry to get c;.
                If c_i \ge 10, then reset c_i to (c_i - 10) and reset the value of carry to 1;
Step 5
                otherwise, set the new value of carry to 0.
                Add 1 to i. effectively moving one column to the left.
Step 6
Step 7 Set c<sub>m</sub> to the value of carry.
Step 8 Print out the final answer, c_m c_{m-1} c_{m-2} \dots c_0.
Step 9 Stop.
```

Figure 1.2 Algorithm for Adding Two m-digit Numbers

Pseudocode

- English language constructs modeled to look like statements available in most programming languages
- Steps presented in a structured manner (numbered, indented, and so on)
- No fixed syntax for most operations is required

Pseudocode (continued)

- Less ambiguous and more readable than natural language
- Emphasis is on process, not notation
- Well-understood forms allow logical reasoning about algorithm behavior
- Can be easily translated into a programming language

Sequential Operations

- Computation operations
 - Example
 - Set the value "variable" to "arithmetic expression"
 - Variable
 - Named storage location that can hold a data value

Sequential Operations (continued)

- Input operations
 - To receive data values from the outside world
 - Example
 - Get a value for r, the radius of the circle
- Output operations
 - To send results to the outside world for display
 - Example
 - Print the value of Area

Average Miles per Gallon Algorithm (Version 1)

STEP	OPERATION
1	Get values for gallons used, starting mileage, ending mileage
2	Set value of distance driven to (ending mileage - starting mileage)
3	Set value of average miles per gallon to (distance driven ÷ gallons used)
4	Print the value of average miles per gallon
5	Stop

Figure 2.3 Algorithm for Computing Average Miles per Gallon

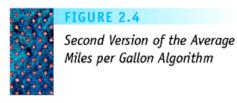
Conditional and Iterative Operations

- Sequential algorithm
 - Also called straight-line algorithm
 - Executes its instructions in a straight line from top to bottom and then stops
- Control operations
 - Conditional operations
 - Iterative operations

- Conditional operations
 - Ask questions and choose alternative actions based on the answers
 - Example
 - if x is greater than 25 then print x

else

print x times 100



Average Miles per Gallon Algorithm (Version 2) STEP OPERATION Get values for gallons used, starting mileage, ending mileage Set value of distance driven to (ending mileage - starting mileage) 2 3 Set value of average miles per gallon to (distance driven ÷ gallons used) Print the value of average miles per gallon 4 If average miles per gallon is greater than 25.0 then 5 Print the message 'You are getting good gas mileage' Else 7 Print the message 'You are NOT getting good gas mileage' 8 Stop

Figure 2.5 Second Version of the Average Miles per Gallon Algorithm

- Iterative operations while statement
 - Perform "looping" behavior, repeating actions until a continuation condition becomes false
 - Loop
 - The repetition of a block of instructions

Examples

```
• while j > 0 do

set s to s + a_j

set j to j - 1
```

repeat
print a_k
set k to k + 1
until k > n

- Components of a loop
 - Continuation condition
 - Loop body
- Infinite loop
 - The continuation condition never becomes false
 - An error

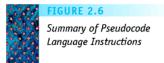
Average Miles per Gallon Algorithm (Version 3)

STEP	Operation
1	response = Yes
2	While (response = Yes) do steps 3 through 11
3	Get values for gallons used, starting mileage, ending mileage
4	Set value of distance driven to (ending mileage – starting mileage)
5	Set value of average miles per gallon to (distance driven ÷ gallons used)
6	Print the value of average miles per gallon
7	If average miles per gallon > 25.0 then
8	Print the message 'You are getting good gas mileage'
	Else
9	Print the message 'You are NOT getting good gas mileage'
10	Print the message 'Do you want to do this again? Enter Yes or No'
11	Get a new value for <i>response</i> from the user
12	Stop

Figure 2.7
Third Version of the Average Miles per Gallon Algorithm

- Pretest loop
 - Continuation condition tested at the beginning of each pass through the loop
 - It is possible for the loop body to never be executed
 - While loop

- Posttest loop
 - Continuation condition tested at the end of loop body
 - Loop body must be executed at least once
 - Do/While loop



```
COMPUTATION:
   Set the value of "variable" to "arithmetic expression"
INPUT/OUTPUT:
   Get a value for "variable", "variable". . .
   Print the value of "variable", "variable", ...
   Print the message 'message'
CONDITIONAL:
   If "a true/false condition" is true then
       first set of algorithmic operations
   Else
       second set of algorithmic operations
ITERATIVE:
While ("a true/false condition") do step i through step j
       Step i: operation
       Step j: operation
While ("a true/false condition") do
       operation
       operation
End of the loop
Do
       operation
       operation
While ("a true/false condition")
```

Figure 2.9

Examples of Algorithmic Problem Solving

- Go Forth and Multiply: Multiply two numbers using repeated addition
- Sequential search: Find a particular value in an unordered collection
- Find maximum: Find the largest value in a collection of data
- Pattern matching: Determine if and where a particular pattern occurs in a piece of text

Example 1: Go Forth and Multiply

Task

Implement an algorithm to multiply two numbers,
 a and b, using repeated addition

Algorithm outline

 Create a loop that executes exactly b times, with each execution of the loop adding the value of a to a running total

Multiplication via Repeated Addition

```
Get values for a and b
If (either a = 0 or b = 0) then
   Set the value of product to 0
Else
   Set the value of count to 0
   Set the value of product to 0
   While (count < b) do
       Set the value of product to (product + a)
       Set the value of count to (count+1)
   End of loop
Print the value of product
Stop
```

Figure 2.10
Algorithm for Multiplication via Repeated Addition

Example 2: Looking, Looking, Looking, Looking

- Task
 - Find a particular person's name from an unordered list of telephone subscribers
- Algorithm outline
 - Start with the first entry and check its name, then repeat the process for all entries

Example 2: Looking, Looking, Looking, Looking (continued)

- Algorithm discovery
 - Finding a solution to a given problem
- Naïve sequential search algorithm
 - For each entry, write a separate section of the algorithm that checks for a match
 - Problems
 - Only works for collections of exactly one size
 - Duplicates the same operations over and over

Example 2: Looking, Looking, Looking, Looking (continued)

- Correct sequential search algorithm
 - Uses iteration to simplify the task
 - Refers to a value in the list using an index (or pointer)
 - Handles special cases (such as a name not found in the collection)
 - Uses the variable Found to exit the iteration as soon as a match is found



FIGURE 2.9

The Sequential Search Algorithm

Sequential Search Algorithm

STEP	OPERATION
1	Get values for NAME, $N_1, \ldots, N_{10,000}$, and $T_1, \ldots, T_{10,000}$
2	Set the value of i to 1 and set the value of Found to NO
3	While both (Found = NO) and ($i \le 10,000$) do steps 4 through 7
4	If NAME is equal to the ith name on the list N, then
5	Print the telephone number of that person, T_i
6	Set the value of Found to YES
	Else (NAME is not equal to N _i)
7	Add 1 to the value of i
8	If (Found = NO) then
9	Print the message 'Sorry, this name is not in the directory'
10	Stop

Figure 2.13 The Sequential Search Algorithm

Example 2: Looking, Looking, Looking, Looking (continued)

 The selection of an algorithm to solve a problem is greatly influenced by the way the data for that problem is organized

Example 3: Big, Bigger, Biggest

Task

- Find the largest value from a list of values
- Algorithm outline
 - Keep track of the largest value seen so far (initialized to be the first in the list)
 - Compare each value to the largest seen so far, and keep the larger as the new largest

Example 3: Big, Bigger, Biggest (continued)

 Once an algorithm has been developed, it may itself be used in the construction of other, more complex algorithms

Library

- A collection of useful algorithms
- An important tool in algorithm design and development

Example 3: Big, Bigger, Biggest (continued)

Find Largest algorithm

Uses iteration and indices as in previous example

 Updates location and largest so far when needed in the loop



Find Largest Algorithm

Get a value for n, the size of the list
Get values for A_1, A_2, \ldots, A_n , the list to be searched
Set the value of largest so far to A_1 Set the value of location to 1
Set the value of i to 2
While $(i \le n)$ do

If $A_i > largest$ so far then

Set largest so far to A_i Set location to iAdd 1 to the value of iEnd of the loop
Print out the values of largest so far and location
Stop

Figure 2.14 Algorithm to Find the Largest Value in a List

Example 4: Meeting Your Match

Task

 Find if and where a pattern string occurs within a longer piece of text

Algorithm outline

- Try each possible location of pattern string in turn
- At each location, compare pattern characters against string characters

Example 4: Meeting Your Match (continued)

- Abstraction
 - Separating high-level view from low-level details
 - Key concept in computer science
 - Makes difficult problems intellectually manageable
 - Allows piece-by-piece development of algorithms

Example 4: Meeting Your Match (continued)

- Top-down design
 - When solving a complex problem
 - Create high-level operations in the first draft of an algorithm
 - After drafting the outline of the algorithm, return to the high-level operations and elaborate each one
 - Repeat until all operations are primitives

Example 4: Meeting Your Match (continued)

- Pattern-matching algorithm
 - Contains a loop within a loop
 - External loop iterates through possible locations of matches to pattern
 - Internal loop iterates through corresponding characters of pattern and string to evaluate match

Pattern-Matching Algorithm

```
Get values for n and m, the size of the text and the pattern, respectively
Get values for both the text T_1 T_2 ... T_n and the pattern P_1 P_2 ... P_m
Set k, the starting location for the attempted match, to 1
While (k \le (n - m + 1)) do
    Set the value of i to 1
    Set the value of Mismatch to NO
    While both (i \le m) and (Mismatch = NO) do
        If P_i \neq T_{k+(i-1)} then
           Set Mismatch to YES
        Else
            Increment i by 1 (to move to the next character)
    End of the loop
    If Mismatch = NO then
        Print the message 'There is a match at position'
        Print the value of k
    Increment k by 1
End of the loop
Stop, we are finished
```

Figure 2.16
Final Draft of the Pattern-Matching Algorithm

Summary

- Algorithm design is a first step in developing an algorithm
- Algorithm design must
 - Ensure the algorithm is correct
 - Ensure the algorithm is sufficiently efficient
- Pseudocode is used to design and represent algorithms

Summary

 Pseudocode is readable, unambiguous, and able to be analyzed

Algorithm design is a creative process; uses multiple drafts and top-down design to develop the best solution

Abstraction is a key tool for good design