Chapter 2: Algorithm Discovery and Design

Invitation to Computer Science, C++ Version, Third Edition
Objectives

In this chapter, you will learn about:

- Representing algorithms
- Examples of algorithmic problem solving
Introduction

- This chapter discusses algorithms and algorithmic problem solving using three problems:
  - Searching lists
  - Finding maxima and minima
  - Matching patterns
Representing Algorithms

- Natural language
  - Language spoken and written in everyday life
  - Examples: English, Spanish, Arabic, etc.
  - Problems with using natural language for algorithms
    - Verbose
    - Imprecise
    - Relies on context and experiences to give precise meaning to a word or phrase
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

- High-level programming language
  - Examples: C++, Java
  - 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
```c++
{
    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
Pseudocode

- English language constructs modeled to look like statements available in most programming languages

- Steps presented in a structured manner (numbered, indented, etc.)

- 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

- Types of algorithmic operations
  - Sequential
  - Conditional
  - Iterative
Sequential Operations (continued)

- Computation operations
  - Example
    - Set the value of “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
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 and Iterative Operations (continued)

- 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
Conditional and Iterative Operations (continued)

- Iterative operations
  - Perform “looping” behavior; repeating actions until a continuation condition becomes false
  - Loop
    - The repetition of a block of instructions
Conditional and Iterative Operations (continued)

- 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 $k > n$
Average Miles per Gallon Algorithm (Version 2)

<table>
<thead>
<tr>
<th>Step</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Get values for <em>gallons used</em>, <em>starting mileage</em>, <em>ending mileage</em></td>
</tr>
<tr>
<td>2</td>
<td>Set value of <em>distance driven</em> to (<em>ending mileage</em> − <em>starting mileage</em>)</td>
</tr>
<tr>
<td>3</td>
<td>Set value of <em>average miles per gallon</em> to (<em>distance driven</em> ÷ <em>gallons used</em>)</td>
</tr>
<tr>
<td>4</td>
<td>Print the value of <em>average miles per gallon</em></td>
</tr>
<tr>
<td>5</td>
<td>If <em>average miles per gallon</em> is greater than 25.0 then</td>
</tr>
<tr>
<td>6</td>
<td>Print the message ‘You are getting good gas mileage’</td>
</tr>
<tr>
<td></td>
<td>Else</td>
</tr>
<tr>
<td>7</td>
<td>Print the message ‘You are NOT getting good gas mileage’</td>
</tr>
<tr>
<td>8</td>
<td>Stop</td>
</tr>
</tbody>
</table>

Figure 2.4
Second Version of the Average Miles per Gallon Algorithm
Conditional and Iterative Operations (continued)

- 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)**

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

Figure 2.5  
Third Version of the Average Miles per Gallon Algorithm
Conditional and Iterative Operations

- 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
Conditional and Iterative Operations (continued)

- Posttest loop
  - Continuation condition tested at the end of loop body
  - Loop body must be executed at least once
- Do/While loop
**Figure 2.6**

**Summary of Pseudocode Language Instructions**

**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
  - .
- While (“a true/false condition”)
Example 1: Looking, Looking, Looking

- Examples of algorithmic problem solving
  - **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: Looking, Looking, Looking (continued)

- **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 1: 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 1: 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 (like a name not found in the collection)
  - Uses the variable *Found* to exit the iteration as soon as a match is found
Sequential Search Algorithm

<table>
<thead>
<tr>
<th>Step</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Get values for NAME, N_1, \ldots, N_{10,000}, and T_1, \ldots, T_{10,000}</td>
</tr>
<tr>
<td>2</td>
<td>Set the value of i to 1 and set the value of Found to NO</td>
</tr>
<tr>
<td>3</td>
<td>While both (Found = NO) and (i ≤ 10,000) do steps 4 through 7</td>
</tr>
<tr>
<td>4</td>
<td>If NAME is equal to the i\text{th} name on the list N_i then</td>
</tr>
<tr>
<td>5</td>
<td>Print the telephone number of that person, T_i</td>
</tr>
<tr>
<td>6</td>
<td>Set the value of Found to YES</td>
</tr>
<tr>
<td></td>
<td>Else (NAME is not equal to N_i)</td>
</tr>
<tr>
<td>7</td>
<td>Add 1 to the value of i</td>
</tr>
<tr>
<td>8</td>
<td>If (Found = NO) then</td>
</tr>
<tr>
<td>9</td>
<td>Print the message ‘Sorry, this name is not in the directory’</td>
</tr>
<tr>
<td>10</td>
<td>Stop</td>
</tr>
</tbody>
</table>

Figure 2.9
The Sequential Search Algorithm
Example 1: Looking, Looking, Looking (continued)

- The selection of an algorithm to solve a problem is greatly influenced by the way the data for that problem are organized.
Example 2: 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 2: 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 2: Big, Bigger, Biggest (continued)

- Find Largest algorithm

- Uses iteration and indices like 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 $(i \leq n)$ do  
  If $A_i > $ *largest so far* then  
    Set *largest so far* to $A_i$  
    Set *location* to $i$  
    Add 1 to the value of $i$  
End of the loop  
Print out the values of *largest so far* and *location*  
Stop

**Figure 2.10**  
Algorithm to Find the Largest Value in a List
Example 3: 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 3: 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 3: Meeting Your Match (continued)

- Top-down design

  - When solving a complex problem:
    - Create high-level operations in 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 3: 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 $T_1 \ T_2 \ \ldots \ \ T_n$ and the pattern $P_1 \ P_2 \ \ldots \ \ P_m$
Set $k$, the starting location for the attempted match, to 1
While ($k \leq (n - m + 1)$) do
  Set the value of $i$ to 1
  Set the value of $Mismatch$ to NO
  While both ($i \leq m$) 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 2.12
Final Draft of the Pattern-Matching Algorithm
Summary

- Algorithm design is a first step in developing an algorithm

- Must also:
  - 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 analyzable

- 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