Chapter 2: Algorithm Discovery and Design

Introduction

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

Objectives

- In this chapter, you will learn about:
  - Representing algorithms
  - Examples of algorithmic problem solving

Figure 2.1
The Addition Algorithm of Figure 1.2 Expressed in Natural Language

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

- 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
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
Conditional and 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

- **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)

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

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

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

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

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>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Get values for NAME, N, ENS, and T, T[99]</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>5</td>
<td>While both (Found = NO) and (i &lt; 10,000) do steps 4 through 7</td>
</tr>
<tr>
<td>4</td>
<td>If NAME is equal to the ith name on the list, 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>7</td>
<td>Else (NAME is not equal to N) then</td>
</tr>
<tr>
<td>8</td>
<td>Add 1 to the value of i</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>

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

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

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

Pattern-Matching Algorithm

- Get values for $n$ and $m$, the size of the text and the pattern, respectively
- Get values for both the text $T_1 \ldots T_n$ and the pattern $P_1 \ldots P_m$
- Set $k$, the starting location for the attempted match, to 1
- While $(k < n - m + 1)$ do
  - Set the value of $i$ to 1
  - If both $T_k = P_i$ and $(i < m)$ do
    - Set $M[match]$ to YES
    - Else
      - Increment $i$ by 1 (to move to the next character)
  - End of the loop
- If $M[match]$ = NO then
  - Print the message “There is no match at position”
- Print the value of $k$
- Increment $k$ by 1
- End of the loop
- Stop, we are finished