Module #5 - Algorithms

University of Florida
Dept. of Computer & Information Science & Engineering
COT 3100
Applications of Discrete Structures
Dr. Michael P. Frank

Slides for a Course Based on the Text
Discrete Mathematics & Its Applications
(5th Edition)
by Kenneth H. Rosen

2004/3/2 (c)2001-2002, Michael P. Frank

Module #5 - Algorithms

Module #5: Algorithms

Rosen 5th ed., §2.1
~31 slides, ~1 lecture

2004/3/2 (c)2001-2002, Michael P. Frank
Chapter 2: More Fundamentals

- §2.1: Algorithms (Formal procedures)
- §2.2: Complexity of algorithms
  - Analysis using order-of-growth notation.
- §2.3: The Integers & Division
  - Some basic number theory.
- §2.6: Matrices
  - Some basic linear algebra.

2.1: Algorithms

- The foundation of computer programming.
- Most generally, an algorithm just means a definite procedure for performing some sort of task.
- A computer program is simply a description of an algorithm in a language precise enough for a computer to understand, requiring only operations the computer already knows how to do.
- We say that a program implements (or “is an implementation of”) its algorithm.
Module #5 - Algorithms

Algorithms You Already Know

• Grade school arithmetic algorithms:
  – How to add any two natural numbers written in
decimal on paper using carries.
  – Similar: Subtraction using borrowing.
  – Multiplication & long division.
• Your favorite cooking recipe.
• How to register for classes at UF.

Programming Languages

• Some common programming languages:
  – Newer: Java, C, C++, Visual Basic, JavaScript,
    Perl, Tcl, Pascal
  – Older: Fortran, Cobol, Lisp, Basic
  – Assembly languages, for low-level coding.
• In this class we will use an informal,
Pascal-like “pseudo-code” language.
• You should know at least 1 real language!
Module #5 - Algorithms

Algorithm Example (English)

- Task: Given a sequence \( \{a_i\} = a_1, \ldots, a_n \), \( a_i \in \mathbb{N} \), say what its largest element is.
- Set the value of a temporary variable \( v \) (largest element seen so far) to \( a_1 \)'s value.
- Look at the next element \( a_i \) in the sequence.
- If \( a_i > v \), then re-assign \( v \) to the number \( a_i \).
- Repeat previous 2 steps until there are no more elements in the sequence, & return \( v \).

Module #5 - Algorithms

Executing an Algorithm

- When you start up a piece of software, we say the program or its algorithm are being run or executed by the computer.
- Given a description of an algorithm, you can also execute it by hand, by working through all of its steps on paper.
- Before ~WWII, “computer” meant a person whose job was to run algorithms!
Module #5 - Algorithms

Executing the Max algorithm

- Let \( \{a_i\} = 7, 12, 3, 15, 8 \). Find its maximum...
- Set \( v = a_1 = 7 \).
- Look at next element: \( a_2 = 12 \).
- Is \( a_2 > v \)? Yes, so change \( v \) to 12.
- Look at next element: \( a_2 = 3 \).
- Is \( 3 > 12 \)? No, leave \( v \) alone....
- Is \( 15 > 12 \)? Yes, \( v = 15 \)....

Module #5 - Algorithms

Algorithm Characteristics

Some important features of algorithms:
- **Input.** Information or data that comes in.
- **Output.** Information or data that goes out.
- **Definiteness.** Precisely defined.
- **Correctness.** Outputs correctly relate to inputs.
- **Finiteness.** Won’t take forever to describe or run.
- **Effectiveness.** Individual steps are all do-able.
- **Generality.** Works for many possible inputs.
- **Efficiency.** Takes little time & memory to run.
Our Pseudocode Language:  \[A2\]

**procedure** \(\text{name}(\text{argument}: \text{type})\)

- Declares that the following text defines a procedure named \text{name} that takes inputs (\text{arguments}) named \text{argument} which are data objects of the type \text{type}.
  - Example:
    \[\text{procedure} \ \text{maximum}(L: \text{list of integers})\]
    \[\{\text{statements defining} \ \text{maximum} \ldots\}\]

**for** \(\text{variable} : = \text{initial value} \) to \(\text{final value}\)

**while** \(\text{condition}\)

**if** \(\text{condition}\) then \(\text{statement}\) [else \(\text{statement}\)]

**return** \(\text{expression}\)
Module #5 - Algorithms

variable \( : = \) expression

- An assignment statement evaluates the expression \( \text{expression} \), then reassigns the variable \( \text{variable} \) to the value that results.
  - Example:
    \[ v := 3x + 7 \]
    (If \( x \) is 2, changes \( v \) to 13.)
- In pseudocode (but not real code), the \( \text{expression} \) might be informal:
  - \( x := \) the largest integer in the list \( L \)

Module #5 - Algorithms

Informal statement

- Sometimes we may write a statement as an informal English imperative, if the meaning is still clear and precise: “swap \( x \) and \( y \)”
- Keep in mind that real programming languages never allow this.
- When we ask for an algorithm to do so-and-so, writing “Do so-and-so” isn’t enough!
  - Break down algorithm into detailed steps.
Module #5 - Algorithms

```
begin statements end
```

- Groups a sequence of statements together:
  ```
  begin
  statement 1
  statement 2
  ...
  statement n
  end
  ```

- Allows sequence to be used like a single statement.
- Might be used:
  - After a `procedure` declaration.
  - In an `if` statement after `then` or `else`.
  - In the body of a `for` or `while` loop.

Module #5 - Algorithms

```
{comment}
```

- Not executed (does nothing).
- Natural-language text explaining some aspect of the procedure to human readers.
- Also called a `remark` in some real programming languages.
- Example:
  - `{Note that v is the largest integer seen so far.}`
Module #5 - Algorithms

**if** condition **then** statement

- Evaluate the propositional expression condition.
- If the resulting truth value is true, then execute the statement statement; otherwise, just skip on ahead to the next statement.
- Variant: if cond then stmt1 else stmt2
  Like before, but iff truth value is false, executes stmt2.

Module #5 - Algorithms

**while** condition **statement**

- Evaluate the propositional expression condition.
- If the resulting value is true, then execute statement.
- Continue repeating the above two actions over and over until finally the condition evaluates to false; then go on to the next statement.
Module #5 - Algorithms

**while** condition statement

- Also equivalent to infinite nested ifs, like so:
  
  ```
  if condition
  begin
    statement
    if condition
    begin
      statement
      ...(continue infinite nested if's)
    end
  end
  ```

Module #5 - Algorithms

**for** var := initial to final stmt

- *Initial* is an integer expression.
- *Final* is another integer expression.
- Repeatedly execute stmt, first with variable var := initial, then with var := initial+1, then with var := initial+2, etc., then finally with var := final.
- What happens if stmt changes the value that initial or final evaluates to?
**Module #5 - Algorithms**

**for** \( var : \text{initial} \text{ to final} \text{ stmt} \)

- **For** can be exactly defined in terms of **while**, like so:

\[
\begin{align*}
\text{begin} \\
\quad var : = \text{initial} \\
\quad \text{while} \quad var \leq \text{final} \\
\quad \text{begin} \\
\quad \quad \text{stmt} \\
\quad \quad var : = var + 1 \\
\quad \text{end} \\
\text{end}
\end{align*}
\]

**procedure**(argument)

- A **procedure call** statement invokes the named **procedure**, giving it as its input the value of the **argument** expression.
- Various real programming languages refer to procedures as **functions** (since the procedure call notation works similarly to function application \( f(x) \)), or as **subroutines**, **subprograms**, or **methods**.
Module #5 - Algorithms

Max procedure in pseudocode

procedure max(a₁, a₂, …, aₙ: integers)
    \( v := a₁ \)  \{ largest element so far \}
    for \( i := 2 \) to \( n \)  \{ go thru rest of elems \}
        if \( a_i > v \) then \( v := a_i \)  \{ found bigger? \}
    \{ at this point \( v \)'s value is the same as the largest integer in the list \}
return \( v \)

Another example task

• Problem of searching an ordered list.
  – Given a list \( L \) of \( n \) elements that are sorted into a definite order (e.g., numeric, alphabetical),
  – And given a particular element \( x \),
  – Determine whether \( x \) appears in the list,
  – and if so, return its index (position) in the list.
• Problem occurs often in many contexts.
• Let’s find an efficient algorithm!
Module #5 - Algorithms

Search alg. #1: Linear Search

**procedure** *linear search*

\((x: \text{ integer}, a_1, a_2, \ldots, a_n: \text{ distinct integers})\)

\(i := 1\)

\(\text{while } (i \leq n \land x \neq a_i)\)

\(i := i + 1\)

\(\text{if } i \leq n \text{ then } \text{location} := i\)

\(\text{else } \text{location} := 0\)

**return** \(\text{location}\) \{index or 0 if not found\}

---

Module #5 - Algorithms

Search alg. #2: Binary Search

- Basic idea: On each step, look at the *middle* element of the remaining list to eliminate half of it, and quickly zero in on the desired element.
Module #5 - Algorithms

Search alg. #2: Binary Search

procedure binary search
  (x:integer, a_1, a_2, ..., a_n: distinct integers)
  i := 1 {left endpoint of search interval}
  j := n {right endpoint of search interval}
  while i < j begin {while interval has >1 item}
      m := ⌊(i+j)/2⌋ {midpoint}
      if x > a_m then i := i + 1 else j := m
  end
  if x = a_i then location := i else location := 0
  return location

Module #5 - Algorithms

Practice exercises

• 2.1.3: Devise an algorithm that finds the sum of all the integers in a list.  [2 min]
• procedure sum(a_1, a_2, ..., a_n: integers)
  s := 0 {sum of elems so far}
  for i := 1 to n {go thru all elems}
      s := s + a_i {add current item}
  {at this point s is the sum of all items}
  return s
Module #5 - Algorithms

Review [2.1]: Algorithms

- Characteristics of algorithms.
- Pseudocode.
- Examples: Max algorithm, linear search & binary search algorithms.
- Intuitively we see that binary search is much faster than linear search, but how do we analyze the efficiency of algorithms formally?
- Use methods of algorithmic complexity, which utilize the order-of-growth concepts from §1.8.

Module #5 - Algorithms

Review: max algorithm

procedure max($a_1, a_2, \ldots, a_n$: integers)
  $v := a_1$ \{largest element so far\}
  for $i := 2$ to $n$ \{go thru rest of elements\}
    if $a_i > v$ then $v := a_i$ \{found bigger?\}
  \{at this point $v$’s value is the same as the largest integer in the list\}
return $v$
Module #5 - Algorithms

Review: Linear Search

procedure linear search
\( (x: \text{integer}, a_1, a_2, \ldots, a_n: \text{distinct integers}) \)
\[ i := 1 \]
\[ \text{while } (i \leq n \land x \neq a_i) \]
\[ \quad i := i + 1 \]
\[ \text{if } i \leq n \text{ then } \text{location} := i \]
\[ \text{else } \text{location} := 0 \]
\[ \text{return } \text{location} \{\text{index or 0 if not found}\} \]

Review: Binary Search

- Basic idea: On each step, look at the middle element of the remaining list to eliminate half of it, and quickly zero in on the desired element.
Module #5 - Algorithms

Review: Binary Search

**procedure** binary search  
(x:integer, a₁, a₂, …, aₙ: distinct integers)  
i := 1  {left endpoint of search interval}  
j := n  {right endpoint of search interval}  
while i<j begin  {while interval has >1 item}  
m := ⌊(i+j)/2⌋  {midpoint}  
if x>aₘ then i := m+1 else j := m  
end  
if x = aᵢ then location := i else location := 0  
return location