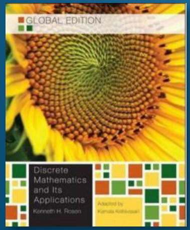
Algorithms

Chapter 3



- Taken from the instructor's resource of *Discrete Mathematics and Its Applications*, 7/e
- Edited by Shin Hong hongshin@handong.edu

Problems and Algorithms

- In many domains there are key general problems that ask for output with specific properties when a valid input is given
- Solution (algorithm)
  - State precisely the problem by specifying the input and the desired output, using the appropriate structures
  - Specify the steps of a procedure that takes a valid input and produces the desired output.

Algorithms

Abu Ja'far Mohammed Ibin Musa Al-Khowarizmi (780-850)

Definition: An *algorithm* is a finite sequence of precise instructions for performing a computation or for solving a problem.

Example: Describe an algorithm for finding the maximum value in a finite sequence of integers.

Solution: Perform the following steps:

1. Set the temporary maximum equal to the first integer in the sequence.
2. Compare the next integer in the sequence to the temporary maximum.
   - If it is larger than the temporary maximum, set the temporary maximum equal to this integer.
3. Repeat the previous step if there are more integers. If not, stop.
4. When the algorithm terminates, the temporary maximum is the largest integer in the sequence.

Specifying Algorithms in Pseudocode

- Pseudocode is an intermediate step between natural language description and code using a specific programming language.
- The form of pseudocode we use is specified in Appendix 3.
  - Similar with C++ and Java.

- Programmers can use the description of an algorithm in pseudocode to construct a program in a particular language.

- Pseudocode helps us analyze the time required to solve a problem using an algorithm, independent of the actual programming language used to implement algorithm.

Properties of Algorithms

- *Input*: An algorithm has input values from a specified set.
- *Output*: From the input values, the algorithm produces the output values from a specified set. The output values are the solution.
- *Correctness*: An algorithm should produce the correct output values for each set of input values.
- *Finiteness*: An algorithm should produce the output after a finite number of steps for any input.
- *Effectiveness*: It must be possible to perform each step of the algorithm correctly and in a finite amount of time.
- *Generality*: The algorithm should work for all problems of the desired form.

Finding the Maximum Element in a Finite Sequence

- The algorithm in pseudocode:

```
procedure max(a_1, a_2, …., a_n: integers)
    max := a_1
    for i := 2 to n
        if max < a_i then max := a_i
    return max  {max is the largest element}
```

- Does this algorithm have all the properties listed on the previous slide?

Some Example Algorithm Problems

- Three classes of problems will be studied in this section.
  1. *Searching Problems*: finding the position of a particular element in a list.
  2. *Sorting problems*: putting the elements of a list into increasing order.
  3. *Optimization Problems*: determining the optimal value (maximum or minimum) of a particular quantity over all possible inputs.

Searching Problems

Definition: The general *searching problem* is to locate an element x in the list of distinct elements $a_1, a_2, \ldots, a_n$, or determine it is not in the list.

- The solution to a searching problem is the location of the term in the list that equals x (that is, i is the solution if $x = a_i$) or 0 if x is not in the list.
  - For example, a library might want to check to see if a patron is on a list of those with overdue books before allowing him/her to checkout another book.
- Linear search vs. binary search.

Linear Search Algorithm

- The linear search algorithm locates an item in a list by examining elements in the sequence one at a time, starting at the beginning.
  - First compare x with a_1. If they are equal, return the position 1.
  - If not, try a_2. If $x = a_2$, return the position 2.
  - Keep going, and if no match is found when the entire list is scanned, return 0.

procedure *linear search*(x:integer, $a_1, a_2, \ldots, a_n$: distinct integers)
$i := 1$
while ($i \leq n$ and $x \neq a_i$)
    $i := i + 1$
if $i \leq n$ **then** *location* := i
else *location* := 0
return *location*{*location* is the subscript of the term that equals x, or is 0 if x is not found}

Binary Search

- Assume the input is a list of items in increasing order.
- The algorithm begins by comparing the element to be found with the middle element.
  - If the middle element is lower, the search proceeds with the upper half of the list.
  - If it is not lower, the search proceeds with the lower half of the list (through the middle position).
- Repeat this process until we have a list of size 1.
  - If the element we are looking for is equal to the element in the list, the position is returned.
  - Otherwise, 0 is returned to indicate that the element was not found.
- In Section 3.3, we show that the binary search algorithm is much more efficient than linear search.

Binary Search

- Here is a description of the binary search algorithm in pseudocode.

procedure binary search(x: integer, $a_1, a_2, \ldots, a_n$: increasing integers)
$i := 1$ {i is the left endpoint of interval}
$j := n$ {j is right endpoint of interval}
while $i < j$
  $m := \lfloor (i + j)/2 \rfloor$
  if $x > a_m$ then $i := m + 1$
  else $j := m$
if $x = a_i$ **then** *location* $:= i$
else *location* $:= 0$
return *location*{location is the subscript i of the term a_i equal to x, or 0 if x is not found}

Binary Search

Example: The steps taken by a binary search for 19 in the list:

1 2 3 5 6 7 8 10 12 13 15 16 18 19 20 22

1. The list has 16 elements, so the midpoint is 8. The value in the 8th position is 10. Since 19 > 10, further search is restricted to positions 9 through 16.

   1 2 3 5 6 7 8 10 12 13 15 16 18 19 20 22

2. The midpoint of the list (positions 9 through 16) is now the 12th position with a value of 16. Since 19 > 16, further search is restricted to the 13th position and above.

   1 2 3 5 6 7 8 10 12 13 15 16 18 19 20 22

3. The midpoint of the current list is now the 14th position with a value of 19. Since 19 ≯ 19, further search is restricted to the portion from the 13th through the 14th positions .

   1 2 3 5 6 7 8 10 12 13 15 16 18 19 20 22

4. The midpoint of the current list is now the 13th position with a value of 18. Since 19> 18, search is restricted to the portion from the 18th position through the 18th.

   1 2 3 5 6 7 8 10 12 13 15 16 18 19 20 22

5. Now the list has a single element and the loop ends. Since 19=19, the location 16 is returned.