Writing Algorithms

Louis-Noël Pouchet

pouchet@cse.ohio-state.edu

Dept. of Computer Science and Engineering, the Ohio State University

September 2010

888.11

Algorithms

Definition (Says wikipedia:)

An algorithm is an effective method for solving a problem expressed as a finite sequence of instructions.

It is usually a high-level description of a procedure which manipulates well-defined input data to produce other data

Algorithms are...

1. A way to communicate about your problem/solution with other people
2. A possible way to solve a given problem
3. A "formalization" of a method, that will be proved
4. A mandatory first step before implementing a solution
5. ...

A Few Rules

1. There are many ways to write algorithms (charts, imperative program, equations, ...)
   - Find yours! But...
   - ... **Always be consistent!**
2. An algorithm takes an input and produces an output
   - Those must be well-defined
3. An algorithm can call other algorithms
   - Very useful for a "top-down" description
   - But called algorithms must be presented too

A Syntax Proposal

- Generic imperative language that accepts recursive call
- Control structures: indentation delimits the scope
  - **for all** *element* $\in$ *Set* **do**
  - **for** *iterator* **=** *lowerbound* **to** *upperbound* **step** *increment* **do**
  - **while** *conditional* **do**
  - **do** ... **while** *conditional*
  - **if** *conditional* **then** ... **else**
  - **case** *element* **in** *value* **:**
  - **return** *value*
  - **break**
  - **continue**
- intructions: standard C++ syntax without pointers/reference
- function call: standard C++ syntax without pointers/reference
- exception: when your algorithm cannot safely terminate and/or respect the output specification

An example

Algorithm

algorithm **gcd**
input: *integer* a, b
output: *greatest common divisor of* a *and* b

if $a = 0$ **then**
  return b
while $b \neq 0$ **do**
  if $a > b$ **then**
    $a = a - b$
  else
    $b = b - a$
return a

Another example

Algorithm

***BuildSearchSpace**: Compute $\mathcal{T}$*

Input:

pdg*: polyhedral dependence graph*

Output:

$\mathcal{T}$*: the bounded space of candidate multidimensional schedules*

$d \leftarrow 1$
while $pdg \neq \emptyset$ **do**
  $\mathcal{T}_d \leftarrow$ *createPolytope(*$[-1,1]$,$[-1,1]$*)*
  for each *dependence* $\mathcal{D}_{R,S} \in pdg$ **do**
    $\mathcal{W}_{\mathcal{D}_{R,S}} \leftarrow$ *buildWeakLegalSchedules(*$\mathcal{D}_{R,S}$*)*
    $\mathcal{T}_d \leftarrow \mathcal{T}_d \cap \mathcal{W}_{\mathcal{D}_{R,S}}$
  end for
  for each *dependence* $\mathcal{D}_{R,S} \in pdg$ **do**
    $\mathcal{S}_{\mathcal{D}_{R,S}} \leftarrow$ *buildStrongLegalSchedules(*$\mathcal{D}_{R,S}$*)*
    if $\mathcal{T}_d \cap \mathcal{S}_{\mathcal{D}_{R,S}} \neq \emptyset$ **then**
      $\mathcal{T}_d \leftarrow \mathcal{T}_d \cap \mathcal{S}_{\mathcal{D}_{R,S}}$
      $pdg \leftarrow pdg$ - $\mathcal{D}_{R,S}$
    end if
  end for
end do

Recursive Algorithms

- Can be very useful / simpler to write
  - Do not worry about the efficiency of the implementation at this stage!
- Reflects well equational forms
- Possible design: assume a property at level n, how to ensure the property at level $n+1$
- Think about some specific data structures (eg, trees)

Vectors

- Generic container with random access capability via the index of the element
  - Example: `A[i]`, `A[i][j][function(i,j)]`
- Arbitrary size, automatically handled
- Accessor for its size (eg, $length(vector)$)

Stack and Queue

- Stack: LIFO
  - stack = push(stack, elt)
  - elt = pop(stack)
  - integer = size(stack)
- Queue: FIFO
  - queue = push(queue, elt)
  - elt = pop(queue)
  - integer = size(queue)

Graphs

- Set of nodes and edges, both can carry arbitrary information
  - edge = getEdge(graph, node1, node2)
  - list of nodes = getConnectedNodes(graph, node)
  - element = getNodeValue(graph, node)
  - element = getEdgeValue(graph, edge)
  - etc., and the associated functions to modify the graph structure
- Many, many problems in CS are amenable to graph representation...

Trees

- Trees are directed acyclic graphs
  - The functions to manipulate them are similar to graph ones
- Numerous refinement/specialization of trees
  - binary tree
  - search tree
  - ...

Algorithm writing 101

1. Determine the input and output
2. Find a correct data structure to represent the problem
   - Don't hesitate to convert the input to a suitable form, and to preprocess it
3. Try to reduce your problem to a variation of a well-known one
   - Sorting? Path discovery/reachability? etc.
   - Look in the litterature if a solution to this problem exists
4. Decide wheter you look for a recursive algorithm or an imperative one, or a mix
   - Depends on how you think, how easy it is to exhibit invariants, what is the decomposition in sub-problems, ...
5. Write the algorithm :-)
6. Run all your examples on it, manually, before trying to prove it

Reference

About manipulating data structures (arrays, trees, graphs):
Introduction to Algorithms, by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein

(I will assume this book has been read in full)

Exercise 1

Input:

- a vector V of n elements, unsorted
- a comparison function $boolean : f(elt : x, elt : y)$ which returns true if x precedes y

Ouput:

- a vector of n elements, sorted according to f

Exercise: write an algorithm which implements the above description

Exercise 2

Input:

- The starting address of a matrix of integer A of size $n \times n$
- The starting address of a matrix of integer B of size $n \times n$
- A function $matrix(16x16) : getBlock(address : X, int : i, int : j)$ which returns a sub-matrix (a block) of the matrix starting at address X, of size 16×16 whose first element is at position i, j

Ouput:

- An integer c, the sum of the diagonal elements of the product of A and B

Exercise: write an algorithm which implements the above description

Exercise 3

Input:

- An arbitrary binary search tree A with integer nodes
  - The left subtree of a node contains only nodes with keys less than the node's key.
  - The right subtree of a node contains only nodes with keys greater than the node's key.
  - Both the left and right subtrees must also be binary search trees.

Output:

- A balanced binary search tree B containing all elements in the nodes of A

Exercise: write an algorithm which implements the above description