Problem solving & Algorithms

Dr. Bibhudatta Sahoo
Communication & Computing Group
Department of CSE, NIT Rourkela
Email: bdsahu@nitrkl.ac.in, 9937324437, 2462358

Algorithm

- Algorithm: is a procedure that consists of a *finite set of instructions* which, given an *input* from some set of possible inputs, enables us to obtain an *output* if such an output exists or else obtain nothing at all if there is no output for that particular input through a *systematic execution* of the *instructions*.

Algorithm

- A clearly specified set of instructions to solve a problem.
- Characteristics:
  - **Input:** Zero or more quantities are externally supplied
  - **Definiteness:** Each instruction is clear and unambiguous
  - **Finiteness:** The algorithm terminates in a finite number of steps.
  - **Effectiveness:** Each instruction must be primitive and feasible
  - **Output**: At least one quantity is produced

Algorithm Description

- How to describe algorithms independent of a programming language
- Pseudo-Code = a description of an algorithm that is
  - more structured than usual prose but
  - less formal than a programming language
- (Or diagrams)
- Example: find the maximum element of an array.

Algorithm arrayMax(A, *n*):

Input: An array A storing n integers.

Output: The maximum element in A.

currentMax ← A[0]

for ***i***← **1 to *n* -1 do**

  if *currentMax* < A[i] **then** *currentMax* ← A[i]

return *currentMax*

Pseudocode Conventions

[1] Comments begin with **/ /** and continue until the end of line.

[2] Blocks are indicated with ·matching braces: **{** and **}**. A compound statement (i.e., a collection of simple statements] can be represented as a block. The body of a procedure also forms a block. Statements are delimited by **;** .

[3] An identifier begins with a letter. The data types of variables are not explicitly declared. The types will be clear from the context.

Pseudocode Conventions

[4] Assignment of values to variables is done using the assignment statement

(variable) := (expression);

[5] There are two boolean values **true** and **false**. In order to produce these values, the ***logical operators*** and, **or,** and **not** and the ***relational operators*** <,≤, =,≠,≥, and > are provided.

[6] Elements of multidimensional arrays are accessed using [and]. For example, if A is a two dimensional array, the (i, j)th element of the array is denoted as A[i,j]. Array indices start at zero.

Pseudocode Conventions

[7] The looping statements supported by the pseudo code are: for, while, and repeat - until.

The while loop takes the following form

```
While < condition> do
{
        <statement1>
}
```

8. A conditional statement has the following forms:

if $\langle condition \rangle$ **then** $\langle statement \rangle$
if $\langle condition \rangle$ **then** $\langle statement\ 1 \rangle$ **else** $\langle statement\ 2 \rangle$

Here $\langle condition \rangle$ is a boolean expression and $\langle statement \rangle$, $\langle statement\ 1 \rangle$, and $\langle statement\ 2 \rangle$ are arbitrary statements (simple or compound).

We also employ the following **case** statement:

case
{
 :$\langle condition\ 1 \rangle$**:** $\langle statement\ 1 \rangle$
 $\vdots$
 :$\langle condition\ n \rangle$**:** $\langle statement\ n \rangle$
 :else: $\langle statement\ n+1 \rangle$
}

Pseudocode Conventions

[9] Input and output are done using the instructions *read* and *write*. No format is used to specify the size of input or output quantities.

[10] There is only one type of procedure: *Algorithm*. An algorithm consists of a heading and a body. The heading takes the form

Algorithm *Name ((parameter list))*

```
Algorithm Max(A, n)
// A is an array of size n.
{
    Result := A[1];
    for i := 2 to n do
        if A[i] > Result then Result := A[i];
    return Result;
}
```

Not an algorithm

- [Selection sort]

for $i := 1$ **to** n **do**
{
    Examine $a[i]$ to $a[n]$ and suppose
    the smallest element is at $a[j]$;
    Interchange $a[i]$ and $a[j]$;
}

Algorithm 1.1 Selection sort algorithm

Algorithm finds and returns the maximum of *n* given numbers:

```
Algorithm Max(A, n)
// A is an array of size n.
{
    Result := A[1];
    for i := 2 to n do
        if A[i] > Result then Result := A[i];
    return Result;
}
```

```
for i := 1 to n do
{
    Examine a[i] to a[n] and suppose
    the smallest element is at a[j];
    Interchange a[i] and a[j];
}
```

Algorithm SelectionSort

```
Algorithm SelectionSort(a, n)
// Sort the array a[1 : n] into nondecreasing order.
{
    for i := 1 to n do
    {
        j := i;
        for k := i + 1 to n do
            if (a[k] < a[j]) then j := k;
        t := a[i]; a[i] := a[j]; a[j] := t;
    }
}
```

Recursive Algorithm

```
Algorithm TowersOfHanoi(n, x, y, z)
// Move the top n disks from tower x to tower y.
{
    if (n ≥ 1) then
    {
        TowersOfHanoi(n − 1, x, z, y);
        write ("move top disk from tower", x,
        "to top of tower", y);
        TowersOfHanoi(n − 1, z, y, x);
    }
}
```

Algorithm 1.3 Towers of Hanoi

Recursive Algorithm

```
Algorithm Perm(a, k, n)
{
    if (k = n) then write (a[1 : n]); // Output permutation.
    else // a[k : n] has more than one permutation.
        // Generate these recursively.
        for i := k to n do
        {
            t := a[k]; a[k] := a[i]; a[i] := t;
            Perm(a, k + 1, n);
            // All permutations of a[k + 1 : n]
            t := a[k]; a[k] := a[i]; a[i] := t;
        }
}
```

Algorithm 1.4 Recursive permutation generator

```
Algorithm RSum(a, n)
{
    count := count + 1; // For the if conditional
    if (n ≤ 0) then
    {
        count := count + 1; // For the return
        return 0.0;
    }
    else
    {
        count := count + 1;  // For the addition, function
                             // invocation and return
        return RSum(a, n − 1) + a[n];
    }
}
```

```
Algorithm Sum(a, n)
{
    s := 0.0;
    count := count + 1; // count is global; it is initially zero.
    for i := 1 to n do
    {
        count := count + 1; // For for
        s := s + a[i]; count := count + 1; // For assignment
    }
    count := count + 1; // For last time of for
    count := count + 1; // For the return
    return s;
}
```

```
Algorithm Sum(a, n)
{
    for i := 1 to n do count := count + 2;
    count := count + 3;
}
```