Parma University, Math Deaprtment

Algorithms and Data Structures
(models of computation)

prof. Marco Pellegrini

Preliminaries

- Materials: Books, Syllabus, Handouts
- Homeworks and Exams
- Contact: pellegrini@imc.pi.cnr.it, www.imc.pi.cnr.it/~pellegrini, tel 050-3152410
- Summary of first leture: models of computation, informal definition of algorithms, formal definitions, pseudocode.

Example: Euclid's Algorithm

Algorithm E (*Euclid's algorithm*).

Given two positive integers m and n, find their greatest common divisor (gcd), that is, the largest positive integer that divides both m and n.

E0. [Ensure $n \leq m$] If $m < n$ exchange m and n.

E1. [Find reminder] Divide m by n, let r be the remainder.

E2. [Is it zero?] If $r=0$, terminate and n is the answer.

E3. [Reduce] Set $m \leftarrow n$, $n \leftarrow r$, and go back to step E1.

Informal notion of Algorithm

Synonymouses: recepie, process, method, technique, procedure, routine.

1) Finiteness. Must terminate after a finite number of steps; must have a finite decription.

2) Definiteness. Each step must be defined precisely.

3) Input. Inputs come form a specified domain.

4) Output. Having a predefined relation to the input.

5) Effectiveness. Each step must be basic for an agent (person or device).

A formal definiton: I

An algorithm is a quadruple (Q,I,Ω,f).

Q is the set of states. $I \subset Q$ is the set of input states.

$\Omega \subset Q$ is the set of output states. $f: Q \to Q$ is a function that leaves Ω pointwise fixed.

For every $x \in I$ a computational sequence is as follows:

$x_0 = x$, $\quad x_{k+1} = f(x_k)$ $\quad$ for $0 \leq k$.

Let h be the smallest integer such that $f(x_h) \in \Omega$.

If h is defined the computation terminates in h steps with output x_h.

A formal definition II:

rewriting system

Let Σ be a finite set of symbols. Let Σ^* be the set of finite strings in the alphabet Σ. Let N be a positive integer. Encode $Q = \Sigma^* \times [0,N]$, $\quad I = \Sigma^* \times \{0\}$, $\quad \Omega = \Sigma^* \times \{N\}$.

For every $j \in [0,N[$, define θ_j and ϕ_j strings Σ^* and a_j, b_j in $[0,N]$.

$f(\sigma,j) = (\sigma , a_j)$ if θ_j is not a substring σ.

$f(\sigma,j) = (\alpha \phi_j \omega, b_j)$ if α is the shortest string s.t. $\sigma = \alpha \theta_j \omega$

$f(\sigma,N) = (\sigma,N)$.

A formal definition III

k-tape Turing machine

A k-tape TM is made of:

(1) a finite control set U,

(2) k semi-infinite tapes made of cells, each tape has a R/W head.

(3) Each cell holds a character from the alphabet Σ_t.

(4) The input aphabet is $\Sigma_i \subset \Sigma_t$, and the blank $b \in \Sigma_t \setminus \Sigma_i$.

(5) Initial state is $u_0 \in U$, final state is $u_f \in U$.

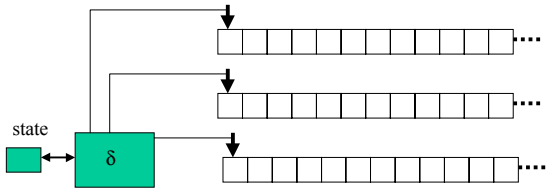
(6) The move function $\delta: U \times \Sigma_t^k \to U \times (\Sigma_t \times \{L,R,S\})^k$

Initially:

(a) the state is u_0;

(b) all R/W heads are in tape postion 1;

(c) The input string is in tape 1, from position 1 followed by blanks, all other tapes are blank.



Interpretation of a δ-move:

(1) read the currect state, the characters under the k heads as arguments of δ,

(2) interprete the value of δ as, the new state, characters to be written under the R/W head, moves of the R/W heads to the Left, Right or Stay.

Termination is when u_f is reached.

The output is a string of Σ_i^* in the last tape from postion 1.

Optionally we may ask the TM to be reset (i.e. all other tapes blank, alle heads in position 1.

A formal definition IV

Successor-RAM

(1) Each variable holds a natural number.

(2) A denumerable set of input variables $X_1,X_2,\ldots$

(3) A denumerable set of output variables $Z_1,Z_2,\ldots$

(3) A denumerable set of local variables $Y_1,Y_2,\ldots$

(4) A Program, that is a sequence of stements.

(5) An Instruction Register pointing at the next program instruction to be executed.

Instruction repertoire:

(a) [label] V ← V+1

(b) [label] V ← V-1

(c) [label] V ← V

(d) [label] if (V =0) goto label

(e) stop

Intially all variables are set to 0, except the variables holding the input. The IR points at the first instruction.

The algorithm stops when the *stop* instruction is reached; the last instruction of the program is always *stop.*

A classical RAM model

1) each cell or register holds a natural number

2) A semi-infinite input tape of cells $X_1,X_2,\ldots$ together with a Read head.

3) A semi-infinite output tape of cells $Z_1,Z_2,\ldots$ together with a Write head.

4) A semi-infinite memory of registers $R_1,R_2,\ldots$

5) An Accumulator Register, an Instruction register

6) A Program, a sequence of statements (with labels).

Instruction repertoire:

(a) Input/Output control (read, write, move heads)

(b) Load/Store from the Accumulator

(c) Arithmetic operations / Boolean operations.

(d) Branching (conditional, unconditional)

(e) Termination

Addressing modes:

(a) literal constant

(b) direct addressing

(c) indirect addressing

Operational Stements have a typical form:

([label] Operation code, address mode, operand).

Brunching Stements have a typical form:

([label] Operation code, label).

This RAM model is rather close to the actual functioning of computers from the 1950's onward. Except that:

(a) memory is a *finite* sequence of registers

(b) each cell can store a *finite* subset of integers in binary.

Time Cost Models

For $x \in I$, the Turing machine time cost $T_{TM}(x)$ is the number of steps of the computation of the TM having x as input.

For $x \in I$, the Unit Cost RAM time $T_{URAM}(x)$ is the number of stements executed in a computation of the RAM machine having x as input.

For $x \in I$, the Logaritmic cost RAM time $T_{LRAM}(x)$ is the sum of local costs of stements executed in a computation of the RAM machine having x as input.

The local cost function

Define the following auxilairy logarithmic like function:

$$l(0) = 1$$
$$l(i) = 1 + \lfloor \log |i| \rfloor$$

For (op,i) in literal mode: $t_{op} + l(i)$.

For (op,i) in direct mode: $t_{op} + l(i) + l(R[i])$.

For (op,i) in indirect mode: $t_{op} + l(i) + l(R[i]) + l(R[R[i]])$.

The Log cost model is more faithful, but harder to use.

Worst, best, average cases

Let A be any of the algorithms seen so far, and C any of the cost models seen so far we have $T_{AC}: I \to \mathbb{N}$.

Define a complexity measure $|| : I \to \mathbb{N}$. (e.g. input length)

Worst case time cost: $T^{W}_{AC}(n) = \max_{x \in I, |x|=n} T_{AC}(x)$

Best case time cost: $T^{W}_{AC}(n) = \min_{x \in I, |x|=n} T_{AC}(x)$

Let us associate a positive probability $p(x)$ to every $x \in I$

Average case time cost: $T^{Av}_{AC}(n) = \sum_{x \in I, |x|=n} p(x) T_{AC}(x)$

Arithmetic operations

• SRAM (Successor RAM): allows only inceremetns and decremetns.

• RAM (Standard RAM): allows sums and subtractions.

• MRAM (Multiplicative RAM): allows sums, subtraction, multiplication and division.

• MBRAM (Multiplicative Boolean RAM): allows sums, subtraction, multiplication, division and bit-wise operations.

Simulations

An algorithm A is composed of a machine model M and a Program P. If for every program P on M we have on a machine M' a program P' simulating P then we talk of *machine simulations*.

How does the Time complexity of algorithms behave across machine simulations?

All RAM models with Logarithmic Cost are *polynomially* related to TM.

The MRAM with Unit Cost is *exponentially* related to MRAM with Logarithmic Cost.

Sorting on RAM

Simulations involve *all* programs in a machine model. What can we say about programs realizing *specific* functions?

Sorting in the unit cost comparison-RAM requires $\Omega(n \log n)$ time.

Sorting in a RAM with +,-, and * (but no division, nor bit-wise operations) requires $\Omega(n \log n)$ time in the unti cost model.

BUT

Sorting in a RAM with +,-, * , division and bit-wise operations) is done in O(n) time in the unti cost model.

Bounded Word RAM

Since 1990's the sorting and searching problems are studied in a more advanced model called Bounded Word RAM.

Each register can hold up to b bits, thus store a number between 0 and 2^b -1.

The time complexity is a function of b and $n < 2^b$.

Unit Cost RAM Models are the machine of choice for the analysis of *specific* algorithms and this is safe provided tricks are not allowed, such as using long integers and precomputed lookup tables.

Pseudo-codes

Some textbooks describe algorithms using pseudo-codes similar to Pascal or Algol or C or Java etc..

The underlying model is still a (Unit Cost) RAM, and we assume the existence of a Compiler that translates the pseudo-code into a RAM program at cost O(1) per instruction.

Pseudocodes have:

(1) structured control, (2) basic types and aggregate types, (3) assignments, (4) arithemetic, (5) subroutines with parameter passing conventions, (6) comments (7) de-referencing.

Code (C) vs Pseudo-code (P-C)

(a) Occasionally we use natural language in P-C.

(b) Type declartations are avoided in P-C.

(c) Allowable basic types and aggregates are not limited.

(d) In P-C we use standard mathematical notation.

(e) In P-C we avoid issuses like Input/Output from periferals, error handling, memory management, etc..

(f) P-C is usually *imperative,* since functional of logical languages tend not to have efficient simulations on RAM machines.

Example: Insertion-Sort

```
INSERTION-SORT(A:array of integers)
 FOR j:=2 TO length[A]
  DO key := A[j]
    i := j-1
    WHILE i > 0 AND A[i] > key
    { DO A[i+1] := A[i]
      i := i-1
    END WHILE
    A[i+1] := key
 END FOR
```

Insertion-sort works incrementally, assuming A[1,j] sorted, finds the proper position of A[j+1] thus obtaining A[1,j+1] sorted.

Considerations on insertion-sort

- INSERTION-SORT terminates on evey input.
- INSERTION-SORT is correct, that is, always gives the elements of the input array A in sorted order in the output array A.
- INSERTION-SORT uses only the input storage plus 3 registers (i,j, key).
- We have used arrays with indices starting from 1.
- What happens with an array of one element only?

Time analysis of INSERTION-SORT

```
INSERTION-SORT(A:array)
 FOR j:=2 TO length[A]..... (n-1)
  DO key := A[j]..... (n-1)
    i := j-1.....…….(n-1)
    WHILE i > 0 AND A[i] > key…….2 (t2+…+tn)
     DO A[i+1] := A[i].....…….(t2-1+..+tn-1)
       i := i-1.....…….(t2-1+..+tn-1)
    END WHILE
    A[i+1] := key.....…….(n-1)
END FOR
```

Where n is the number of elements of A and t_j is the number of elements in A[1,j] that are ≥ A[j].

$T(n) = 4(n-1) + 4(t_2+,..,+t_n) -2(n-1)$

Since $1 \le t_j \le j$ we have:

(a) Best case: $t_j=1$ for every j, so T(n) = 6(n-1).

(b) Worst case: $t_j=j$ for every j, so T(n) = 4n-5 + n(n+1)/2

We have counted in a unit cost model for pseudo-code isntructions. To have the RAM unit cost we should take into consideration the compilation overhead.

Conclusions

We have seen:

(1) an example of an algorithm: Euclid's algorithm

(2) Informal notion of algorith

(3) Several formalizations (TM, RAM models)

(4) Simulations. Time complexity.

(5) Unit cost RAM models: pros and cons.

(6) Example: insertion-sort.