Data Structures

R. K. Ghosh

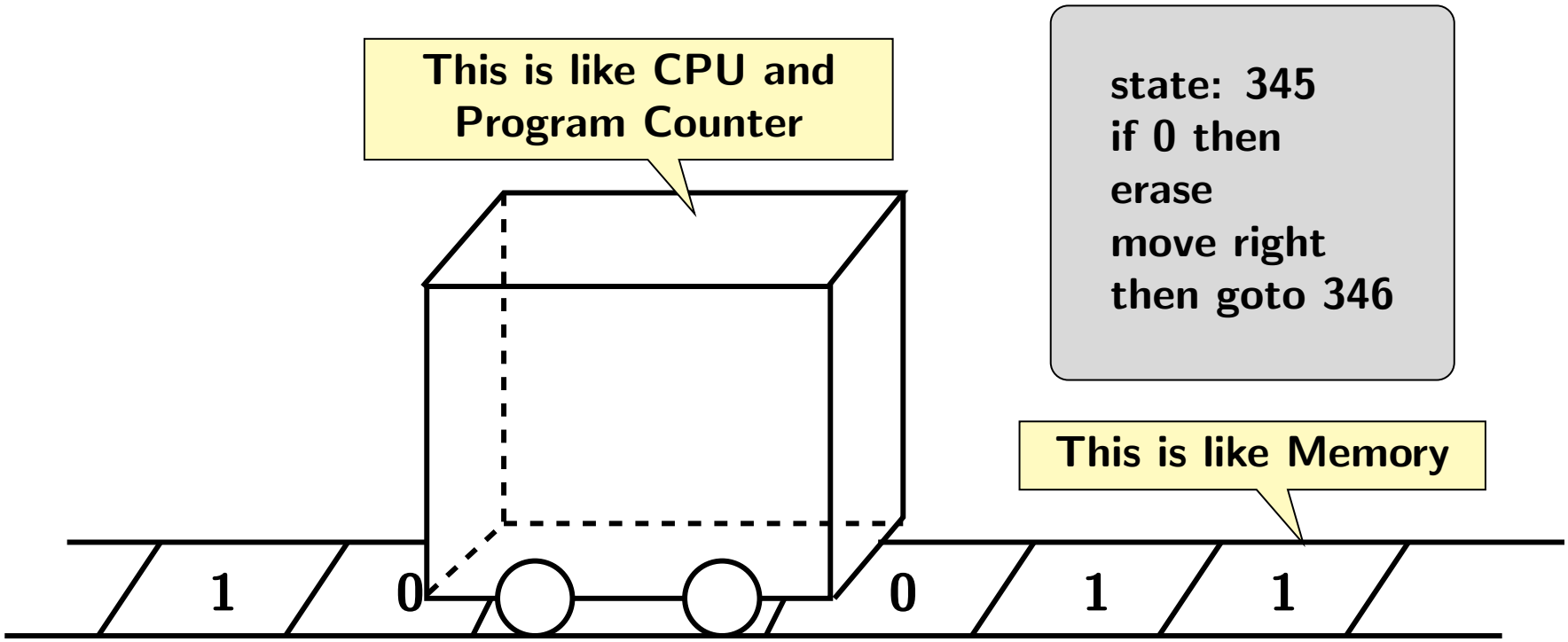
IIT Kanpur

Data structures: Computational Model

Turing Machine

- Turing proposed a generic definition of a computation.
- He viewed that any information can be written in coded form (a string of 0s and 1s).
- An infinite tape divided into cells, each holding a 0 or an 1 or some spaces.
- There is an automaton which has a knowledge of its current state.
- The automaton examines each cell on the tape at a time.
- Looks at a program book which tells it what to do in the current state.

Turing Machine



- After examining current input, TM either moves left or right.
- Changes its state as specified by the program.

Turing Machine

- Initial conditions: entire input string w is present on the tape surrounded by infinite number of blanks.
- Final state: if TM halts in final state then it accepts w
- TM halts in a non final state: it rejects w
- In general a transition is expressed as: $\delta(q, X) = (p, Y, D)$,
  - q: current state,
  - X: TM's RW-head at tape symbol X
  - Y: Output symbol, RW-head erases X and replaces it by Y.
  - p: New state
  - D: could be R or L specifying movement of RW-head

Computation versus Language

- Calculation: Takes an input value and outputs a value.
- Language: A set of string meeting certain criteria.
- So, language for a calculation basically a set of strings of the form ”<input, output>”, where output correspond to value calculated from the input.

Computation versus Language

L_{add} could consists of strings

<0+0, 0>	<0+1,1>	<0+2,2>	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
<5+7, 12>	<5+8,13>	<5+9,14>	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$

Membership question: Verifying a solution <13+12, 25> belongs to L_{add} or not?

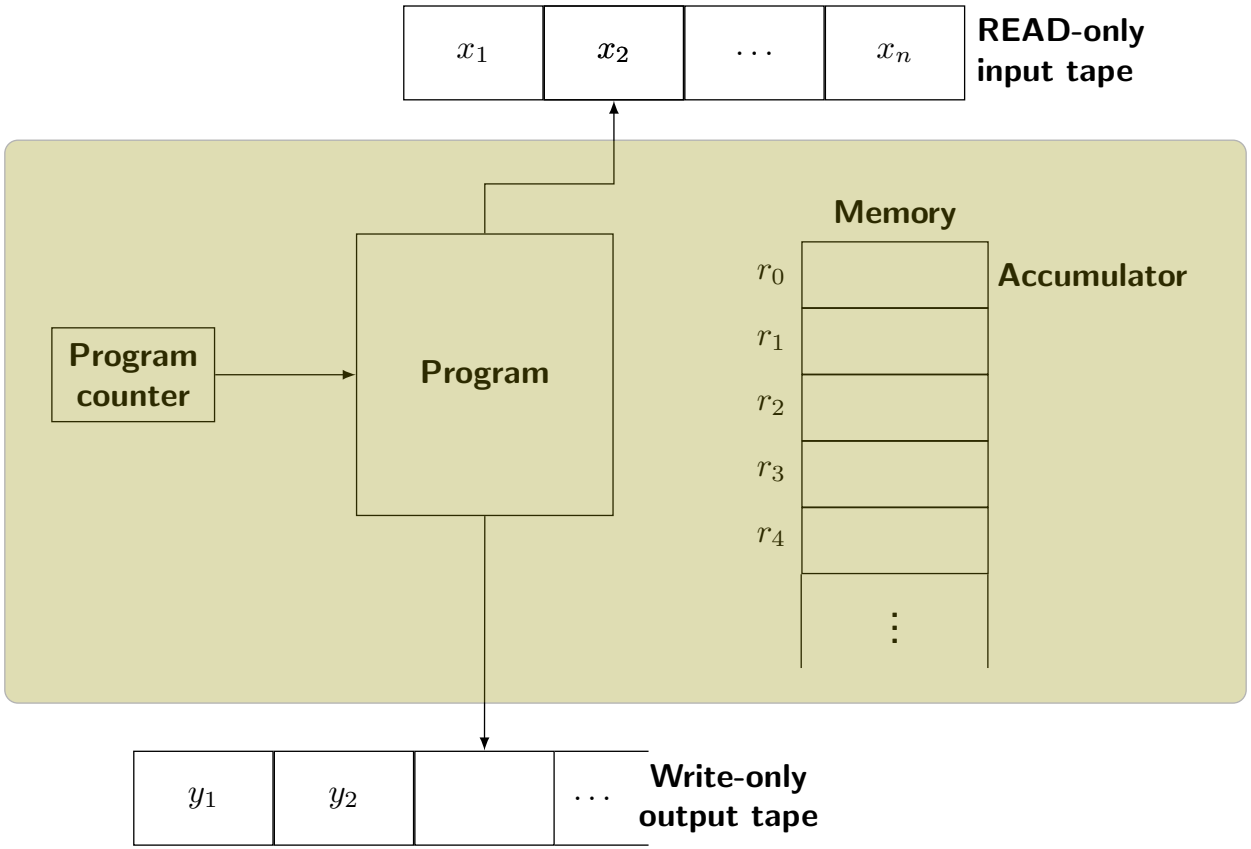
Random Access Machine

- Disconnect between a TM and real computer is sequential tape vs random access memory.
- A RAM is a simplified abstraction of real world computer
- It has an unbounded memory and capable of storing an arbitrarily large integer in each memory cell.
- RAM can access content of any random memory cell.
- However, to access a random cell, RAM needs to read the address for the cell in a different register.
- For description of algorithms it is practical to use RAM, since it is closest to a real program.

RAM Model

- Instructions are executed sequentially.
- Impractical to define instructions of each machine, and their corresponding costs.
- Therefore, a set of commonly found instructions in a computer are assumed:
  - **Arithmetic**: ADD, SUB, MULTI, DIV,
  - **Data movement**: LOAD, STORE, WRITE, READ
  - **Control**: JUMP, JGTZ, JZERO, HALT.
- Assume each instruction takes one unit of time.
- A RAM program is not stored in memory of RAM, so instructions cannot be modified.

RAM Model


x_1
x_2
$\cdots$
x_n
READ-only input tape
Memory
r_0
r_1
r_2
r_3
r_4
Accumulator
Program counter
Program
y_1
y_2
$\cdots$
Write-only output tape

RAM Model

- Programs of RAM not stored in the memory, so cannot be modified.
- All computation take place in register r_0 (accumulator)
- An operand can be one of the following type:
  - Immdiate Addressing ($= i$): integer i itself.
  - Direct Addressing (i): $c(i)$ contents of register r_i.
  - Indirect Addressing ($*i$): $c(c(i))$, if $c(c(i)) < 0$, machine halts.
- Initially $c(i) = 0$ for all $i \geq 0$.
- LC (PC) is set to first instruction of program P.
- After execution of k instruction LC = $k + 1$, automatically unless k instruction is JUMP, JGTZ, or JZERO.

Meaning of an Instruction & Program

- Value $v(a)$ of an operand a is defined as follows:
  - $v(=i) = i$, $v(i) = c(i)$, $v(*i) = c(c(i))$.
- Program is essentially defines a mapping of input tape to output tape.
- Since, program may not halt for some input, the mapping is only partial.

An Example

Pseudo Code

For algorithm that accepts strings (with end marker 0) having equal number of 1s and 2s.

```
begin
    d = 0;
    read x;
    while x ≠ 0 do begin // 0 is used as endmarker
        if x ≠ 1 then d = d − 1;
        else d = d + 1;
        read x;
    end;
    if d == 0 then write 1
end
```

An Example

RAM Program

```
          LOAD    =0        } d = 0
          STORE   2         }
          READ    1           read x
while:    LOAD    1         } while x ≠ 0 do
          JZERO   endwhile  }
          LOAD    1         }
          SUB     =1        } if x ≠ 1
          JZERO   one       }
          LOAD    2         }
          SUB     =1        } then d = d − 1
          STORE   2         }
          JUMP    endif     }
one:      LOAD    2         }
          ADD     =1        } else d = d + 1
          STORE   2         }
endif:    READ    1           read x
          JUMP    while
endwhile: LOAD    2         }
          JZERO   output    } if d = 0
          HALT              } then write 1
output:   WRITE   =1        }
          HALT
```

Assignment #2

Questions (Full Marks 35)

In each case you have to provide the theoretical solution in LaTeX. All programs should be submitted as per instructions provided in the course website.

1. Give a RAM Program for computing n^k, using squaring each time. [15]
2. Write a TM program for doubling of an input consisting of k consecutive 1s. Replace the input with $2k$ consecutive 1s. [10]
3. Write a TM program that accepts a binary number if it is divisible by 3. [10]

Complexity

- Two important measures for an algorithm: Running time and Space requirement.
- Worst case time complexity: For a given input size, the complexity is measured as the maximum of time taken over all possible inputs of that size.
- Average case time complexity: Equals to average of the time complexity over all input of a given size.
- Average case complexity is difficult to determine.
  - It requires assumptions about distribution of inputs.
  - These assumption may at times won't be mathematically tractable.

Notion of Running Time

- Sorting of 1000 elements takes more time than sorting of 3 elements.
- Even the same algorithm may take different amounts of time for the different inputs of same size.
  - Data **shifting** is not required in insertion sort for a **sorted** sequence.
  - But required for a **reverse sorted** sequence.

Example

```
for (i = 0; i < n; i++)
        sum += a[i];
```

Time Complexity

Description	Times executed
Initialization step	1
Comparison step	$n + 1$
Addition and assignment	$2n$
Increment step	n
Total	$4n + 2$.

Example

```
for(i = 0; i < n; i++)
    for(j = 0; j < n; j++)
        sum += b[i][j];
```

Time Complexity

Description	Times executed
Initialization	1 + n (1 for i, n for j)
Comparison step	$(n+1)$ + $n(n+1)$
Addition and assignment	$2n \times n$
Increment step for first loop	n
Increment step for second loop	n^2
Total	$4n^2 + 4n + 2$

Example

```
for (i = 0; i < n; i++)
    for (j = i + 1; j < n; j++)
        if (a[i] < a[j])
            swap(a[i], a[j]);
```

Time Complexity

- The input is n-element array, so code will result in:
  - $n - 1$ comparisons for a[0]
  - $n - 2$ comparisons for a[1], and so on.
  - In general, $n - i - 1$ comparisons for a[i]
- Therefor, total number of comparisons = $\sum_{i=0}^{n-1}(n - i - 1)$ or $\sum_{i=1}^{n-1} i = n(n-1)/2$

Comparison of Relative Execution Speeds

- Suppose an algorithm A takes time $5000n$ and another algorithm B takes time 1.1^n

Execution Speeds

Input	Algorithm A	Algorithm B
n	$5000n$	1.1^n
10	50000	5500
100	500,000	13,781
1000	5,000,000	2.5×10^{41}
1,000,000	5.10^9	$4.8.10^{41392}$

Comparing Algorithms

The Largest Input Size for a Problem

Find the largest problem size n that can be solved in 1 minute by each of the four algorithms with different running times (in microsconds) : (a) $\log n$, (b) $\sqrt{n}$, (c) n, (d) n^2, and (e) 2^n

Solution

(a) $\log n = 6 \times 10^7$, so $n = 2^{6*10^7}$

(b) $\sqrt{n} = 6 \times 10^7$, so $n = 36 * 10^{14}$

(c) $n = 6 \times 10^7$, so nothing to solve here.

(d) $n^2 = 6 \times 10^7$, so $n = \sqrt{n^2} = 7745$

(e) $2^n = 6 \times 10^7$, so $n = \log 6 \times 10^7 = 58$

Influence of Machine Speeds

Execution Speeds

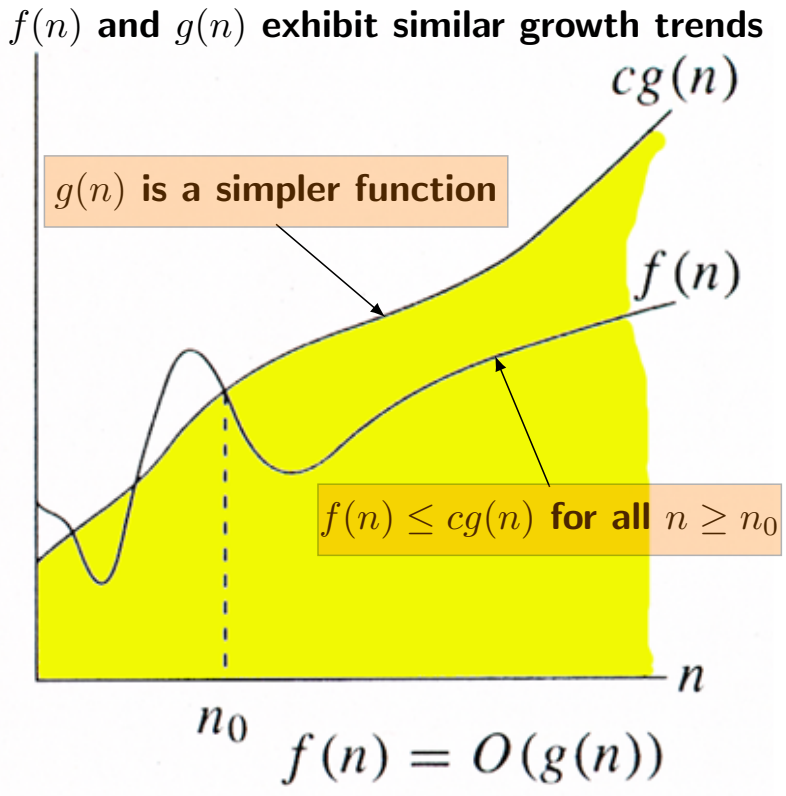
Complexity	Size of the Largest Problem Instance in 1 hour		
	With M1	**With M2**	**With M3**
n	N1	100N1	1000N1
n^2	N2	10 N2	31.6 N2
n^3	N3	4.64 N2	10 N2
2^n	N4	N4 + 6.64	N4 + 9.97
3^n	N5	N5 + 4.19	N5 + 6.29

- For 2^n case, in 1hr = $N4$ with slow computer
- For fast computer $100 \times 2^{N4} = 2^{Nx}$,
  $Nx = N4 + (\log 100 / \log 2) = N4 + 6.64$

Comparing Running Times

- Algorithms with exponential running times are inefficient and can solve only small problems of small sizes.
- Constant and logarithmic running times are most efficient.
- Sublinear and linear runing time are also very good.
- In general, algorithm with running times in polynomial of input size are considered efficient.
- Increasing machine speed does not help in scaling up.
- Therefore, the important measure of efficiency of a program is:
  How the number of steps (time) grows with the input size?

Big Oh Notation



Definition of Big Oh

- Growth function is defined by Big-O notation.
- $f(n)$ is **big-Oh** of $g(n)$, if
  - $g : R \to R$
  - $f : R \to R$,

  and there exist positive constants $c > 0$, and $n_0 \in N$ such that

  $f(n) \leq cg(n)$, for all $n \geq n_0$

Big Oh is Upper Bound

- $f(n)$ is bounded above by $g(n)$ from some point onwards. where
  - $g(n)$ is formulated as a simpler function.
  - $g(n)$ exhibits same trend in growth as $f(n)$.
- Since we are interested for large n, it is alright if $f(n) \leq cg(n)$ for $n > n_0$.

Example

$$\begin{aligned} f(n) = n^2 + 2n + 1 &\leq n^2 + 2n^2, \text{ if } n \geq 2 \\ &= 3n^2 \end{aligned}$$

Therefore, for $c = 3$ and $n_0 = 2$, $f(n) \leq cn^2$, whenever $n \geq n_0$.