Indian Institute of Information Technology
Design and Manufacturing, Kancheepuram
Chennai 600 127, India
An Autonomous Institute under MHRD, Govt of India
http://www.iiitdm.ac.in
Instructor
N.Sadagopan

COM 501 Advanced Data Structures and Algorithms

The Turing Machine and Computational Complexity

The science in `Computer Science` focuses on the systematic study of algorithms along with their formal properties. As an outcome, this study classifies the set of computational problems into *solvable* problems; those that have algorithms, and *unsolvable* problems; problems for which no algorithm exists.

Discussion on formal properties include `Mathematical Model`, `Input Representation (Encoding)`, `Input Size`, expected output with/without an objective function and run-time analysis as a function of the input size.

Mathematical Model is an abstract representation of a system under study. Models help in classifying the set of computational problems (which is an uncountable set) into a 'nice' set of equivalence classes. Often, models do not focus on a particular problem like *sorting*, *searching*, etc. Instead, it looks at the set of problems in a larger perspective and bring them under different umbrellas. For example, problems such as sorting, travelling salesman tour, coloring are classified as `permutation problems`. Since the solution we look for is a permutation (arrangement) of the input set. Similarly, searching and primality testing are classified as `decision problems`, and problems such as vertex cover, feedback vertex set fall into `subset problems`. `Enumeration problems` include: listing all onto functions, listing all spanning trees, etc. Essentially, models focus on the behavior of algorithms associated with problems and group those having a good similarity. Both vertex cover and feedback vertex set look for a subset satisfying some property. For both the problems, except the property testing phase, the identification of subsets (behavior) is same.

Needless to say, each model has its own merits, demerits and limitations. To overcome limitations of one model, various other models have been proposed in the literature. Commonly used models in the study of computing are

1. **Finite Automata:** Used to model decision problems. Membership testing (Is $x \in L$) and pattern matching (Is pattern 'text' present in a file) are common problems falling into this category. Since the model does not have memory to remember state information, questions such as whether a program has equal number of left and right curly braces cannot be verified.

2. **Pushdown Automata:** Variant of Finite Automata (FA) with a memory to remember state information. Membership testing with specific properties such as 'palindrome', 'matching parenthesis' can be modeled using PDA. However, computational problems such as $a+b$, 2^n cannot be modeled using PDA.

3. **Turing Machine:** Superior to PDA and hence, works as an accepting as well as computing device. Used to model all solvable computational problems. The TM can even model 'subroutine' and 'parameter passing'. Moreover, every valid program with finite termination conditions has a TM as its model.

4. **RAM Model:** Random Access Memory model is similar to the Turing machine on both computability and complexity aspects. The behavior of this model is close to the modern day computers.

Note: Models focus on implementation aspect of an algorithm and it plays a vital role in determining the complexity of the problem in hand. Further, input representation for each model is different, an appropriate

encoding of the input is must before using a model to solve a problem.

How are inputs represented ? Input representation is crucial in understanding the model with respect to a specific problem. Turing machine follows the `unary` representation. I.e., to represent the integer '5', we use '1 1 1 1 1'. RAM model uses binary representation and hence, computers use the `binary` representation to interact with its hardware. High-level programming languages follow decimal (base 10) number system for input representation.

Turing Machine

In this lecture, we shall discuss TM from the perspective of 'machine as an acceptor' and 'machine as a computing device'. A Turing machine (TM) consists of an input tape (to represent inputs) and a read/write head (a control unit) using which the tape symbols can be read/written. The input tape is divided into cells and each cell can contain exactly one symbol from a fixed alphabet $\{0, 1, \$\}$, where '0' and '1' are used to describe the input and \$ is a delimiter to mark the end of the tape.

The moves of the TM are described using transition identifiers (IDs). The ID $\delta(q_0, 1) = (q_1, A, R)$ describes that on reading the input '1' at the state q_0, the control switches to the state q_1 by writing the symbol 'A' in place of '1'. Further, the read/write head moves exactly one cell right. Similarly, $\delta(q_0, 1) = (q_0, 1, L)$ says that on reading '1' at q_0, the read/write head alone is shifted one cell left without changing the state and the tape symbol. The move $\delta(q_0, \$) = (q_0, \$, H)$ says on reading \$ at q_0, halt the Turing machine.

We shall now describe the transition function for the language $L = \{a^n\ b^n\}$. Note that TM description naturally yields an algorithm and hence, the description must be sound and complete. I.e., all valid inputs must be accepted and every invalid input must be rejected by the corresponding TM.

Problem 1: $L = \{a^n\ b^n\}$.

Transition Function:

$\delta(q_0, a) = (q_1, A, R)$
$\delta(q_1, a) = (q_1, a, R)$ $\quad$ $\delta(q_1, B) = (q_1, B, R)$
$\delta(q_1, b) = (q_2, B, L)$ $\quad$ $\delta(q_1, \$) = (q_{rej}, \$, H)$
$\delta(q_2, B) = (q_2, B, L)$
$\delta(q_2, a) = (q_2, a, L)$
$\delta(q_2, A) = (q_0, A, R)$
$\delta(q_0, B) = (q_3, B, R)$
$\delta(q_0, b) = (q_{rej}, b, H)$

$\delta(q_3, B) = (q_3, B, R)$
$\delta(q_3, a) = (q_{rej}, a, H)$
$\delta(q_3, b) = (q_{rej}, b, H)$
$\delta(q_3, \$) = (q_{acc}, \$, H)$

Remarks:
1. For all valid inputs such as `aabb`, `aaaabbbb`, the control starting from q_0, after a sequence of moves halts at q_{acc}.
2. For invalid inputs with more a's such as `aaabb`, `aaaa`, the control stops at q_1 when it exhausts a's and it reads \$. Subsequently, the TM halts at q_{rej}.
3. For invalid inputs with more b's such as `bbbb`, `aabbb`, the control is either in q_0 or q_3. Further, on reading 'b', the TM stops at q_{rej}.

In the above example, the TM behaves like an acceptor. Accept all strings $x \in L$ and stop the machine at q_{acc}, and reject all strings $x \notin L$ and stop the machine at q_{rej}. We next discuss TM as a computing device.

Problem 2: Input: m, n in unary, Output: $m + n$ in unary.

Transition Function:
$\delta(q_0, 1) = (q_0, 1, R)$
$\delta(q_0, 0) = (q_0, 1, R)$
$\delta(q_0, \$) = (q_1, 1, L)$
$\delta(q_1, 1) = (q_1, 0, H)$

For the input $m = 3, n = 2$, the tape initially contains

1 1 1 0 1 1 \$

Starting from q_0, after three moves, the read/write head reads '0' and changes the it to '1'. At this point of time, the tape contains $m + n + 1$ in unary. To get $m + n$; after two more moves, when the control sees \$, it moves left and changes the right most '1' to '0', and halts the machine. Thus, we get the unary representation of $m + n$.

Time complexity: To discuss the time complexity, we need to fix the primitive operation, an operation which is done frequently. Further, the time complexity is expressed as a function of the number of primitive operations. As far as TM is concerned, the operations involved are (i) movement of the tape to left/right (ii) changing the tape symbol. For most of the problems, the movement of R/W head dominates the time complexity as it is frequently performed over the other. We now analyze the number of R/W head movements for $m + n$.

`Input size:` $m + n + 1 + 1 = m + n + 2$; $m + n$ for the unary representation of m and n, and the additional two 1's are for delimiter '0' and the end symbol '\$'.

`R/W head movements:` $m - 1 + 1 + n + 1 + 1 = m + n + 2$. As per the above construction, to scan all m 1's, there are $m - 1$ R/W moves, one move to read '0', n moves to scan all n 1's, followed by one move to read the end symbol \$. Further, TM moves left and changes the last '1' to '0'. Note that, the number of R/W head movements is linear in the input size $(m+n+2)$. The above approach runs in polynomial time.

Problem 2: Input: a, b in unary, Output: $a - b$ in unary.

Transition Function:
The function returns $a - b$ if $a \geq b$ and stop the machine at q_7. Otherwise, it stops the machine at q_8.

$\delta(q_0, 1) = (q_1, 2, R)$
$\delta(q_1, 1) = (q_1, 1, R)$
$\delta(q_1, 0) = (q_2, 0, R)$
$\delta(q_2, 2) = (q_2, 2, R)$ $\delta(q_2, 1) = (q_4, 2, L)$
$\delta(q_4, 2) = (q_4, 2, L)$
$\delta(q_4, 0) = (q_3, 0, L)$
$\delta(q_3, 1) = (q_3, 1, L)$
$\delta(q_3, 2) = (q_0, 2, R)$
$\delta(q_2, \$) = (q_5, \$, L)$
$\delta(q_5, 2) = (q_5, 2, L)$
$\delta(q_5, 0) = (q_6, 0, L)$
$\delta(q_6, 1) = (q_6, 1, L)$
$\delta(q_6, 2) = (q_7, 1, H)$
$\delta(q_0, 0) = (q_8, 0, H)$

Note that for $a > b$, the number of R/W head movements is

$(b+1)(a+1)+b(a+2)+(a+1)$. In asymptotic sense, this is $O(ab) = O(a^2)$
For $a \leq b$
$a(a+1)+a(a+2) = O(a^2)$
Note that the input size is $a+b+2 = O(a)$. Therefore, the number of R/W moves for $a-b$ is polynomial in the input size.

Problem 3: Input: Array A of n numbers in unary, Output: Sorted array of A.

Approach in Unary Representation: Note that to sort $(3,2,1,4)$, the input representation in the unary representation is

1110110101111$

To sort; in the first iteration, we find the minimum of A by comparing $A[1]$ with the rest of A; in the second iteration, we find the second minimum of A, and so on. It is important to note that the comparison of two numbers is performed using $a-b$ subroutine as comparison cannot be done directly in Turing machines. Using $a-b$ routine, we declare $a > b$ if 1's are present in the result to the left of '0' ('0' acts as a delimiter for a and b) and $a < b$ if 1's are present in the result to the right of '0'. Further, the computation $a-b$ can be done using an additional tape.

During the first iteration, $A[1]$ is compared with the rest of A by invoking $a-b$ routine $n-1$ times, and the cost for each invoke is $O(a^2)$. The minimum obtained as a result of this iteration is written onto the output tape. Similarly, by incurring $O(a^2)$ effort for $(n-2)$ times, the second minimum is found. Overall, there are $O(n^2)$ comparisons and each comparison incurs $O(a^2)$ effort due to $a-b$ routine. Thus, the time complexity of sorting is $O(n^2a^2)$. Is this polynomial in the input size ?

Input Size: To represent, $(a_1,\ldots,a_n)$ in the unary representation, the number of bits required is $x = a_1+a_2+\ldots+a_n+(n-1)+1$. Note that $(n-1)$ in the sum corresponds to $(n-1)$ 0's used as delimiters and the last one is for the end symbol \$. Let $a = \max(a_1,\ldots,a_n)$, then the input size $x \leq n\cdot a+n = O(na)$. Also, note that $x \leq 2na \leq x^2$. This inequality is used in the run-time analysis while analysing problems such as sorting, search, etc. to say that the analysis is a polynomial function of the input size.
Since the sorting algorithm runs in $O(n^2a^2) = O((na)^2)$, it can be rewritten as $O((x^2)^2) = O(x^4)$, which is polynomial in the input size x. Note that we need an upper bound to bound na in $O((na)^2)$ as a function of x.

Approach in Binary Representation: In case of a binary representation, we work with Random Access Memory (RAM) model wherein we have a sequence of cells (random access memory) with the property that any cell can be accessed in constant time. Further, the addition, subtraction and comparison can be done using digital circuits at bit level in $O(\log n)$ time, where $\lfloor \log n \rfloor + 1 = O(\log n)$ is the number of bits required to represent a number n in the binary representation. Each cell can hold an integer in its binary form.

Input size: For the sorting problem, the input size to represent $(a_1,\ldots,a_n)$ is $\lfloor \log a_1 \rfloor + 1 + \ldots + \lfloor \log a_n \rfloor + 1 = O(n \log a)$, where $a = \max(a_1,\ldots,a_n)$.
Analysis of sorting: As discussed earlier, there are $O(n^2)$ comparisons and each comparison incurs $O(\log a)$. Thus, the time complexity is $O(n^2 \log a)$ which is polynomial in the input size.

Approach in Decimal Representation: Decimal representation follows RAM model with step count analysis approach for analysing the time complexity. That is, for sorting problem, the input size is n which is the number of elements in A and the time for comparing two elements is $O(1)$. Since there are $O(n^2)$ comparisons, the time for sorting is $O(n^2) \times O(1) = O(n^2)$ which is a polynomial in the input size.

Problem 4: Input: Integer n, Output: Factorial of n, $n!$.

Approach in unary: The unary representation follows the Turing machine model. Factorial of n is computed as a sequence of multiplications; $n \times (n-1) \times \ldots \times 1$. Further, the multiplication is modelled

using another Turing machine which does repeated additions.
`Time to perform` $a+b$ `in unary:` $O(a+b) = O(a)$ assuming $a \geq b$.

`Time to perform` ab `in unary:` The addition routine will be called b times. The first time, the addition routine returns $a+a$, the second time it returns $a+a+a$, and finally it returns ab. The overall cost is $O(ab)$.

`Time to perform` $n!$`:` During Iteration 1, we perform $n \times (n-1)$ for which the cost is $O(n(n-1))$. Iteration 2 incurs, $O(n(n-1)(n-2))$. After n iterations, we obtain $n!$ on the output tape in unary form after incurring $O(n(n-1)(n-1)\ldots 2\cdot 1)$ head movements. Thus, the factorial can be output after $O(n!)$ head moves which is exponential in the input size.

Approach in binary: The input n is represented in binary using $O(\log n)$ bits. Addition and multiplication of two numbers is performed using digital circuits at bit level. Addition of two numbers m and n (assuming $n \geq m$) can be performed in $O(\log n)$ bits. The multiplication of two numbers with $O(\log n)$ bits each incurs the following costs.
(a) Each bit of n is multiplied with each bit of m incurring $O(\log n) \times O(\log n) = O((\log n)^2)$ costs.
(b) The partial products must be added to get the final sum. The number of columns in the partial product is $2\log n$ and each column is of size at most $\log n$. Thus, addition cost is $2\log n \times \log n = O((\log n)^2)$.
(c) The overall cost of the multiplication routine for two numbers is $O((\log n)^2)$.
As part of factorial, the multiplication routine is called n times, and therefore, the time complexity of $n!$ is $O(n \cdot (\log n)^2)$.
Since the input size is $O(\log n)$, the time complexity when expressed as a function of input size is $O(2^{\log n}(\log n)^2)$, which is exponential in the input size.

Approach in Decimal: The classical recursive or iterative algorithm takes $O(n)$ steps to output $n!$. It is important to note that n in $O(n)$ does not refer to the input size, instead, it refers to the magnitude. Therefore, the run-time is polynomial in the magnitude.

Problem 5: Input: Integer n, Output: Check n is prime or not.
Approach in unary: Under unary representation, the approach is to check whether 2 is factor of n, 3 is a factor of n and so on. We shall now explain how this division by 2, division by 3 is performed in the Turing machine.
Division by 2: To check whether a number n is divisible by 2, encode n 1's in the input tape. Start moving the read/write head one cell right and for every second '1' in the tape, write '2' in the tape in place of '1'. At the end of this procedure, if we see no 1's in the tape after the last '2', then the number is divisible by 2 and 2 is factor. Otherwise, 2 is not a factor. The cost for this operation is $O(n)$ head moves.

Division by 3: Similar to division by 2, for three 1's, we mark '3' in the input tape in place of the third '1', and at the end if there are no 1's, then n is divisible by 3. Otherwise, 3 is not a factor.
We continue the above procedure for values 4,5, until $n-1$. If the division procedure returns 'NO' for all division routines, then the number is a prime. The overall cost of this approach is $n \times O(n) = O(n^2)$. Thus, primality checking is polynomial time solvable when the input is represented in unary.

Approach in binary: Note that the division checking is performed using repeated subtractions. To check whether '2' is a factor, we subtract '2' from n repeatedly ($\frac{n}{2}$ iterations). The cost of the subtraction is $\frac{n}{2}O(\log n)$. Thus, the run-time for factor checking is $\frac{n}{2}O(\log n) + \frac{n}{3}O(\log n) + \ldots + \frac{n}{n-1}O(\log n)$. On simplifying, we get $O(n \log n \log n) = O(2^{\log n} \log n \log n)$, exponential in the input size.

Approach in Decimal: A simple for loop for n iterations checks all factors in the range 2 to n by incurring $O(n)$ effort. This approach runs in time polynomial in the magnitude.

We shall summarize our discussion in the following table. 'Poly' refers to polynomial in the input size.

'Exp' refers to exponential in the input size. Decimal with constraints fixes a bound on the size of the maximum integer and due to which, the step count analysis assumes, the basic arithmetic operations such as addition, subtraction and multiplication can be done in $O(1)$ time. However, decimal with no constraints has no bound on the size of the integer and hence, the basic arithmetic operation cost varies based on the input size. For decimal with no constraints, the input size is $O(\log_{10} n)$ and therefore, the cost of addition is $O(\log_{10} n)$ and multiplication is $O((\log_{10} n)^2)$.

Problem	Unary representation	Binary	Base 10 (Decimal) with no constraints	Decimal with constraints
Addition(a, b)	Poly: $O(a)$	Poly: $O(\log_2 a)$	Poly: $O(\log_{10} a)$	Poly: $O(1)$
Multiplication(a, b)	Poly: $O(ab)$	Poly: $O((\log_2 a)^2)$	Poly: $O((\log_{10} a)^2)$	Poly: $O(1)$
Sorting	Poly	Poly	Poly	Poly
Factorial	Exp	Exp	Exp	Poly in magnitude
Primality checking	Poly	Exp	Exp	Poly in magnitude

Question: Consider a problem P with input size n. P has $O(n^k)$, k:fixed integer, algorithm in decimal representation. What would be the complexity of P in base-2 and unary representations. Would P take EXP time in unary representation.

Answer: Since P has $O(n^k)$ solution in decimal representation. It is clear that the space occupied by P to store the input and other intermediate results is $O(n^l)$ for some fixed integer l. This is true, because, if the underlying space used is exponential in the input size, then the overall time complexity is at least exponential in the input size as the algorithm must spend $O(1)$ at each of exponential locations. Also, the underlying computations of P may involve basic arithmetic/algebraic operations incurring $O(1)$ effort or it may invoke another subroutine polynomial number of times with each call to the subroutine incurs some polynomial time.

When we analyze P in base-2 representation, the cost of basic arithmetic/algebraic operations must be expressed as a function of the input size ($O(n \log_2 n)$). Since, we know that basic operations are polynomial in the input size for base-2 representation, P runs in polynomial in the input size under base-2 representation as well. I.e., the number of operations of P times the cost of each operation. Since both are polynomial in $O(n \log_2 n)$, our claim is true. It is important to recall that both decimal and base-2 work with RAM model for accessing and manipulating the inputs.

Under unary representation, the input size is the sum of the values of individual elements in the input. Note that basic arithmetic operations can be performed in $O(a^l)$, where l is a fixed integer and a is the maximum of n numbers. Further, in RAM model, accessing an arbitrary location is $O(1)$, however, in unary representation it is polynomial in the input size as the read/write head can move exactly one cell at a time. Since P performs polynomial number of basic operations and accesses the memory polynomial number of times, both these tasks still incur polynomial effort under unary representation. It is important to highlight that the degree of the polynomial increases when we move from decimal to base-2 to base-1. Thus, P has polynomial-time solution in all three representations, polynomial in the input size.