THEORY OF COMPUTATION
UNIT 4

Learning Objective

- Explain the concepts of Turing Machine and its extension.
- Explain the Concept of decidability and different undecidable problems
- Understand the concept of reducibility
- Explain the concept or recursion and recursion theorem

Complexity Theory

- Even when a problem is decidable , it may not be solvable in practice, if the solution requires enormous amount of time or memory.
- Therefore, an investigation of time, memory or some other resources is required for solving computational problem.
- Complexity theory :
  - ✓ Time complexity
  - ✓ Space complexity

Measuring Time Complexity

- Let M ne a DTM that halts on all inputs.
- The running time or time complexity of M is the function f:N->N, Where f(n) is the maximum number of steps that M uses on any input of length n.
- If f(n) is the running time of M, We say that M runs in f(n) an that M is an f(n) time Turing machine.
- "n" represents the length of input
- Exact running time of an algorithm is a complex expression so we usually estimate it.
- One convenient form of estimation is *asymptotic analysis*
  - ✓ *big-O notation*
  - ✓ *small-o notation*

big-O notation and small-o notation

- ***big-O notation*** : *Let f and g be functions f, g :N -> R^+, Say that f(n)=O(g(n)) if positive integers c and n_0 exist such that for every integer n>= n_0*

  f(n)<=cg(n)

- ***small-o notation*** : *Let f and g be functions f, g :N -> R^+, Say that f(n)=O(g(n)) if*

$$\lim_{n\to\infty}\frac{f(n)}{g(n)} = 0$$

In other words, f(n) =o(g(n)) means that, for any real number c>0, a number n_0 exist, where f(n) < cg(n) for all n >= n_0

Complexity Measure in NTM

- Let N be a non-deterministic Turing machine . The running time of N is the function f:N->N, Where f(n) is the maximum number of steps that N uses on any branch of its computation on any input of length n.
- Let t(n) be a function where t(n) >= n. Then every t(n) time non-deterministic single tape Turing machine has an equivalent $2^{O(f(n))}$ time deterministic Turing machine.

P Class complexity

- P is the complexity class containing decision problems which can be solved by a deterministic Turing machine using a polynomial amount of computation time, or polynomial time.
- In other words

$$P = \bigcup_k \text{Time}(n^k)$$

- P is often taken to be the class of computational problems which are "efficiently solvable" or "tractable".

P Class complexity

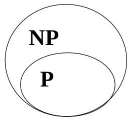
- Problems that are solvable in theory, but cannot be solved in practice, are called *intractable*.
- There exist problems in P which are intractable in practical terms; for example, some require at least $n^{1000000}$ operations.
- P is known to contain many natural problems, including the decision versions of linear programming, calculating the greatest common divisor, and finding a maximum matching. In 2002, it was shown that the problem of determining if a number is prime is in P.

NP Class complexity

- In computational complexity theory, NP ("Non-deterministic Polynomial time") is the set of decision problems solvable in polynomial time on a non-deterministic Turing machine.
- It is the set of problems that can be "verified" by a deterministic Turing machine in polynomial time.
- All the problems in this class have the property that their solutions can be checked effectively.

NP Class complexity

- This class contains many problems that people would like to be able to solve effectively, including
  - The Boolean satisfiability problem (SAT)
  - The Hamiltonian path problem (special case of TSP)
  - The Vertex cover problem.



Space Complexity

- Let M be a deterministic Turing machine that halts on all inputs. The *space complexity* of M is the function f:N->N, where f(n) is the maximum number of tape cells that M scans on any input of length n.
- If the space complexity of M is f(n), we say that M runs in space f(n)
- If M is non-deterministic Turing machine wherein all branches halt on all inputs, we define its space complexity f(n) to be the maximum number of tape cells that M scans on any branch of its computation for any input of length n.

Space Complexity Class

- Let f:N->R+ be a function. The *space complexity classes*, SPACE(f(n)) and NSPACE(f(n)) are defined as follows.
- SPACE(f(n)) = { L | L is a language decided by an O(f(n)) space deterministic Turing machine}.
- NSPACE(f(n)) = { L | L is a language decide by an O(f(n)) space nondeterministic Turing machine}

L Space Complexity

- **L** (also known as **LSPACE**) is the complexity class containing decision problems which can be solved by a deterministic Turing machine using a logarithmic amount of memory space.
- Logarithmic space is sufficient to hold a constant number of pointers into the input and a logarithmic number of boolean flags and many basic logspace algorithms use the memory in this way.

NL Space Complexity

- **NL** (Nondeterministic Logarithmic-space) is the complexity class containing decision problems which can be solved by a nondeterministic Turing machine using a logarithmic amount of memory space.
- **NL** is a generalization of **L**, the class for logspace problems on a deterministic Turing machine.
- Since any deterministic Turing machine is also a nondeterministic Turing machine, we have that **L** is contained in **NL**

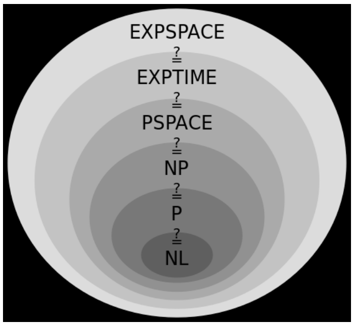
PSPACE Complexity

- **PSPACE** is the set of all decision problems that can be solved by a Turing machine using a polynomial amount of space.
- The set of all problems that can be solved by Turing machines using $O(t(n))$ space for some function t of the input size n, then we can define **PSPACE** formally as

$$\mathrm{PSPACE} = \bigcup_{k\in\mathbb{N}} \mathrm{SPACE}(n^k).$$

- **PSPACE** is a strict superset of the set of context-sensitive languages.

Relationship among complexity classes



Hierarchy Theorem

- One may have intuition that more time or more space should increase the class of problem that it can solve.
- Hierarchy theorems prove that this institution is correct.
- Hierarchy theorems are useful to prove that the time and space complexity class are not all the same.
- Complexity classes form the hierarchy whereby the large class contains more language than do the classes with smaller bounds.
- Hierarchy theorems is broadly categorized into two
  - Space hierarchy theorem
  - Time hierarchy theorem

Space Hierarchy Theorems

Theorem : (Space Hierarchy Theorem)
For any space constructible function $f : N \rightarrow N$, there exists a language A that is dicidable in space $O(f(n))$ but not in space $o(f(n))$.

proof :
Construct an $O(f(n))$ space algorithm B that decides a language $A \notin o(f(n))$ space

B = "On input w:

1. Let n be the length of w.
2. Compute f(n) using space constructibility, and mark off this much tape. If later stages ever attempt to use more, *reject*.
3. If w is not of the form <M>10* for some TM M, *reject*.
4. Simulate M on w while counting the number of steps used in the simulation. If the count ever exceeds $2^{f(n)}$, *reject*.
5. If M accepts, *reject*. If M rejects, *accept*."

Space Hierarchy Theorems

- M may have an arbitrary tape alphabet and B has a fixed tape alphabet, so we represent each cell of M with several cells on B's tape.
- Thus, if M runs in $g(n)$ space, then B uses $dg(n)$ space to simulate M, for some constant d that depends on M.
- B is a decider because it halts in a limited time. Let A be the language that B decides. A is decidable in $O(f(n))$

Space Hierarchy Theorem

- Assume some Turing machine M decides A in space g(n), where g(n) is o(f(n)).
- Since g(n) is o(f(n)) therefore from definition of o(f(n)), some constant n_0 exists, where dg(n) < f(n) for all n >= n_0.
- As the input length is n_0 or more , the simulation of M will complete.
- Therefore D will do the opposite of M on the same input, therefore M does not decide A which contradicts our assumption.
- Therefore A is not decidedable in o(f(n)) space.

Time Hierarchy Theorem

Theorem: (Time hierarchy Theorem)
For any time constructible function $t: N \to N$, there exists a language A that is decidable in time $O(t(n))$ but not in time $o(\frac{t(n)}{\log t(n)})$.

proof :
$B =$ " On input w

1. $n = |w|$
2. Store the value $\left\lceil t(n) / \log t(n) \right\rceil$ in a binary counter. Decrease this counter before each step used to carry out stages $3, 4, 5$
   If the counters $= 0$, then REJECT.
3. If w is not of the form $\langle M \rangle 10*$ for some TM M, REJECT.
4. Simulate M on w.
5. If M accepts, then REJECT;
   else if M rejects, then ACCEPT "

Savitch's Theorem

- For any function $\mathbf{f: N \rightarrow R^+}$, where $\mathbf{f(n) \geq n}$, we have

  $\mathbf{NSPACE(f(n)) \subseteq SPACE(f^2(n))}$.
- Proof :
  - As the proof of this theorem reveals, a deterministic TM can simulate a nondeterministic TM using only a little amount of extra space.
  - That is due to the fact that space can be recycled. As time cannot be recycled, the same trick fails to work with time (otherwise we would have a proof of P=NP).

Savitch's Theorem

- Let $\mathbf{t}$ be a positive integer, and let $\mathbf{c_1}$ and $\mathbf{c_2}$ be two configurations of $\mathbf{N}$.
- We cay that $\mathbf{c_1}$ ***can yield*** $\mathbf{c_2}$ in $\mathbf{\leq t}$ steps if $\mathbf{N}$ can go from $\mathbf{c_1}$ to $\mathbf{c_2}$ in $\mathbf{t}$ or
- fewer steps. The following is a deterministic recursive algorithm deci-
- ding the "can yield" problem when $\mathbf{t}$ is a power of $\mathbf{2}$ ($\mathbf{t=2^p}$ for some $\mathbf{p \geq 0}$).

Savitch's Theorem

- $\mathbf{CANYIELD(c_1, c_2, 2^p)}$ = "On input $\mathbf{c_1}$, $\mathbf{c_2}$ and $\mathbf{p}$, where $\mathbf{p \geq 0}$ and $\mathbf{c_1, c_2}$ are configurations that use at most $\mathbf{f(n)}$ space (i.e. in which the head is at a $\mathbf{\leq f(n)}^{th}$ cell, and all non-blank cells are among the first $\mathbf{f(n)}$ cells):
- 1. If $\mathbf{p=0}$, then test if $\mathbf{c_1=c_2}$ or $\mathbf{c_1}$ yields $\mathbf{c_2}$ in one step according to the rules of $\mathbf{N}$. *Accept* if either test succeeds; *reject* if both fail.
- 2. If $\mathbf{p>0}$, then for each configuration $\mathbf{c_m}$ of $\mathbf{N}$ that uses space $\mathbf{\leq f(n)}$:
- 3. Run $\mathbf{CANYIELD(c_1, c_m, 2^{p-1})}$.
- 4. Run $\mathbf{CANYIELD(c_m, c_2, 2^{p-1})}$.
- 5. If steps 3 and 4 both accept, then *accept*.
- 6. If haven't yet accepted, *reject*."

Savitch's Theorem

- We assume that (or otherwise modify **N** so that) **N** clears its tape before halting and goes to the beginning of the tape, thereby entering a (fixed) configuration called c_{accept}. And we let c_{start} be the start configuration of **N** on input **w**.
- Next, where **n** is the length of **w**, we select a constant **d** so that **N** has no more than $2^{df(n)}$ configurations that use **f(n)** cells of the tape.
- Then $2^{df(n)}$ provides an upper bound on the running time of **N** on **w** (for otherwise a configuration would repeat and thus **N** would go into an infinite loop, contrary to our assumption that this does not happen).
- Hence, **N** accepts **w** if and only if it can get from c_{start} to c_{accept}
- within $2^{df(n)}$ or fewer steps. So, the following (deterministic) machine **M** obviously simulates **N**, i.e. **M** accepts **w** iff **N** doeS:

Savitch's Theorem

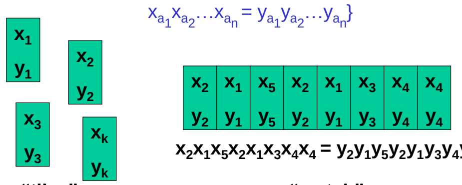
- **M** = "On input **w**:

  1. Output the result of **CANYIELD**(c_{start}, c_{accept}, $2^{df(n)}$) ."
- It remains to analyze the space complexity of **M**.
- Whenever **CANYIELD** invokes itself recursively, it stores the current stage number and the values of c_1,c_2 and **p** on a stack so that these values can be restored upon return from the recursive call. Each level of the recursion thus uses **O(f(n))** additional space.
- Next, each level of the recursion decreases **p** by **1**. And, as initially **p** starts out equal to **df(n)**, the depth of the recursion is **df(n)**.
- Therefore the total space used is $df(n) \times O(f(n)) = O(f^2(n))$, as promised.

Post Correspondence Problem

- Many undecidable problems unrelated to TMs and automata
- Classic example: Post Correspondence Problem

$$PCP = \{<(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)> :$$
$$x_i, y_i \in \Sigma^* \text{ and there exists } (a_1, a_2, \ldots, a_n) \text{ for which}$$
$$x_{a_1}x_{a_2}\ldots x_{a_n} = y_{a_1}y_{a_2}\ldots y_{a_n}\}$$



$x_2x_1x_5x_2x_1x_3x_4x_4 = y_2y_1y_5y_2y_1y_3y_4y_4$

"tiles" "match"

Probabilistic Turing Machine

- There are several popular definitions:
  - A nondeterministic Turing Machine (TM) which randomly chooses between available transitions at each point according to some probability distribution
  - A type of nondeterministic TM where each nondeterministic step is called a coin-flip step and has two legal next moves
  - A Turing Machine in which some transitions are random choices among finitely many alternatives
- Also known as a Randomized Turing Machine

Probabilistic Turing Machine

- There are (at least) three tapes
  - 1st Tape holds the input
  - 2nd Tape (also known as the random tape) is covered randomly (and independently) with 0's and 1's
    - ✓½ probability of a 0
    - ✓½ probability of a 1
  - 3rd Tape is used as the scratch tape
- **WHEN A PROBABILISTIC TM RECOGNIZES A LANGUAGE**
- Accept all strings in the language
- Reject all strings not in the language
- However, a probabilistic TM will have a probability of error

Probabilistic Turing Machine

- Each "branch" in the TMs computation has a probability
- Can have stochastic results
- Hence, on a given input it:
  - May have different run times
  - May not halt
- Therefore, it may accept the input in a given execution, but reject in another execution
- Time and space complexity can be measured using the worst case computation branch