Modern Complexity Theory (CS1.405)

Dr. Ashok Kumar Das

IEEE Senior Member
Web of Science (Clarivate™) Highly Cited Researcher 2022, 2023
Professor
Center for Security, Theory and Algorithmic Research
International Institute of Information Technology, Hyderabad

E-mail: *ashok.das@iiit.ac.in*
URL: `http://www.iiit.ac.in/people/faculty/ashok-kumar-das`
`https://sites.google.com/view/iitkgpakdas/`

Intractability

Intractability

- Some computational problems can theoretically be solved, but the time or space required for their solutions makes them impractical. These problems are referred to as *intractable*.
- Intractability varies depending on the computational resources available and the type of solution sought.
- **Problem 1.** A problem that is generally easy to solve but becomes difficult in rare instances is only intractable in the worst-case scenario.
- **Problem 2.** A problem might be solvable on a supercomputer but take an excessive amount of time to solve on a personal computer.
- **Goal.** We focus on problems whose worst-case complexity would be so enormous that any computer (***Quantum Computers***, in near future) that we could conceivably build would need more time than is thought to remain in the lifetime of the universe.

Definition (Big "oh" (O))

The function $f(n) = O(g(n))$ (read as "f of n is big oh of g of n") if and only if (iff) there exist positive constants c and n_0 such that $f(n) \leq c.g(n), \forall n \geq n_0$.

- Let $f(n) = 3n + 2$.
  Then $3n + 2 \leq 4n, \forall n \geq 2$.
  Hence, $f(n) = O(n)$.
- Let $g(n) = 1000n^2 + 100n - 6$.
  Then $g(n) = O(n^2)$, as $1000n^2 + 100n - 6 \leq 1001n^2$, for all $n \geq 100$.
- Let $h(n) = 6.2^n + n^2$.
  Then $h(n) = O(2^n)$ as $6.2^n + n^2 \leq 7.2^n$, for all $n \geq 4$.

Definition (Little "oh" (o))

The function $f(n) = o(g(n))$ (read as "f of n is little oh of g of n") if and only if (iff) there exist positive constants c and n_0 such that $f(n) < c.g(n), \forall n \geq n_0$.
In other words, the function $f(n) = o(g(n))$ iff $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.

- Let $f(n) = 3n + 2$.
  Then $f(n) = o(n^2)$,
  since $\lim_{n\to\infty} \frac{3n+2}{n^2} = \lim_{n\to\infty}(\frac{3}{n} + \frac{2}{n^2}) = 0$.
- Let $g(n) = 3n + 2$.
  Then $g(n) = o(n \log n)$, since
  $\lim_{n\to\infty} \frac{3n+2}{n \log n} = \lim_{n\to\infty}(\frac{3}{\log n} + \frac{2}{n \log n}) = 0$.

HIERARCHY THEOREMS

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

Definition (Space constructible)

A function $f : N \to N$, where $f(n)$ is at least $O(\log n)$, is called *space constructible* if the function that maps the string 1^n to the binary representation of $f(n)$ is computable in space $O(f(n))$. Here, $1^n = 111 \cdots 1$ (n times).

In other words, f is space constructible if some $O(f(n))$ space TM exists that always halts with the binary representation of $f(n)$ on its tape when started on input 1^n.

Example

All commonly occurring functions that are at least $O(\log n)$ are space constructible, including the functions $\log n$, $n \log n$, and n^2.

HIERARCHY THEOREMS

Theorem (Space hierarchy theorem)

For any space constructible function $f : N \to N$, a language A exists that is decidable in $O(f(n))$ space but not in $o(f(n))$ space.

HIERARCHY THEOREMS

Corollary

For any two real numbers ϵ_1 *and* ϵ_2 *with* $0 \leq \epsilon_1 < \epsilon_2$,

$$SPACE(n^{\epsilon_1}) \subset SPACE(n^{\epsilon_2}).$$

HIERARCHY THEOREMS

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY HYDERABAD

Corollary

$$NL \subset PSPACE.$$

Proof.

- By applying the Savitch's theorem, we have

$$NL \subseteq SPACE(\log^2 n).$$

- By the space hierarchy theorem, we have

$$SPACE(log^2 n) \subset SPACE(n).$$

Hence,

$$NL \subseteq SPACE(\log^2 n) \subset SPACE(n).$$

Thus, the corollary follows. □

HIERARCHY THEOREMS

Corollary

$$PSPACE \subset EXPSPACE.$$

HIERARCHY THEOREMS

Definition (Time constructible)

A function $t : N \to N$, where $t(n)$ is at least $O(n \log n)$, is called **time constructible** if the function that maps the string 1^n to the binary representation of $t(n)$ is computable in time $O(t(n))$.

In other words, t is time constructible if some $O(t(n))$-time TM exists that always halts with the binary representation of $t(n)$ on its tape when started on input 1^n.

Example

All commonly occurring functions that are at least $n \log n$ are time constructible, including the functions $n \log n$, $n\sqrt{n}$, n^2, and 2^n.

HIERARCHY THEOREMS

Theorem (Time hierarchy theorem)

For any time constructible function $t : N \to N$, a language A exists that is decidable in $O(t(n))$ time but not in time $o(t(n)/\log t(n))$.

HIERARCHY THEOREMS

Corollary

For any two functions $t_1 : N \to N$ *and* $t_2 : N \to N$, *where* $t_1(n)$ *is* $o(t_2(n)/\log t_2(n))$ *and* t_2 *is time constructible, the*

$$TIME(t_1(n)) \subset TIME(t_2(n)).$$

HIERARCHY THEOREMS

Corollary

For any two real numbers ϵ_1 *and* ϵ_2 *with* $1 \leq \epsilon_1 < \epsilon_2$,

$$TIME(n^{\epsilon_1}) \subset TIME(n^{\epsilon_2}).$$

HIERARCHY THEOREMS

Corollary

$$P \subset EXPTIME.$$

EXPONENTIAL SPACE COMPLETENESS

Definition (EXPSPACE-complete)

A language B is EXPSPACE-complete if

1. $B \in$ EXPSPACE, and
2. every A in EXPSPACE is poly-time reducible to B, that is, $A \leq_p B$, for all A in EXPSPACE (EXPSPACE-hard).

EXPONENTIAL SPACE COMPLETENESS

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

Problem: Let $\uparrow$ represent the exponentiation operation. If R is a regular expression and k is a non-negative integer, $R\uparrow$ is equivalent to the concatenation of R with itself k times. In other words,

$$R^k = R \uparrow k = R \circ R \circ \cdots R\,(k\ times).$$

Let $EQ_{REX\uparrow} = \{\langle Q, R\rangle | Q$ and R are equivalent regular expressions with exponentiation$\}$.
Prove that $EQ_{REX\uparrow}$ is EXPSPACE-complete.

EXPONENTIAL SPACE COMPLETENESS

PROOF First, we present a nondeterministic algorithm for testing whether two NFAs are inequivalent.

N = "On input $\langle N_1, N_2 \rangle$, where N_1 and N_2 are NFAs:

1. Place a marker on each of the start states of N_1 and N_2.
2. Repeat $2^{q_1+q_2}$ times, where q_1 and q_2 are the numbers of states in N_1 and N_2:
3. Nondeterministically select an input symbol and change the positions of the markers on the states of N_1 and N_2 to simulate reading that symbol.
4. If at any point a marker was placed on an accept state of one of the finite automata and not on any accept state of the other finite automaton, *accept*. Otherwise, *reject*."

If automata N_1 and N_2 are equivalent, N clearly rejects because it only accepts when it determines that one machine accepts a string that the other does not accept. If the automata are not equivalent, some string is accepted by one machine and not by the other. Some such string must be of length at most $2^{q_1+q_2}$. Otherwise, consider using the shortest such string as the sequence of nondeterministic choices. Only $2^{q_1+q_2}$ different ways exist to place markers on the states of N_1 and N_2; so in a longer string, the positions of the markers would repeat. By removing the portion of the string between the repetitions, a shorter such string would be obtained. Hence algorithm N would guess this string among its nondeterministic choices and would accept. Thus, N operates correctly.

EXPONENTIAL SPACE COMPLETENESS

Algorithm N runs in nondeterministic linear space. Thus, Savitch's theorem provides a deterministic $O(n^2)$ space algorithm for this problem. Next, we use the deterministic form of this algorithm to design the following algorithm E that decides $EQ_{\mathsf{REX}\uparrow}$.

E = "On input $\langle R_1, R_2 \rangle$, where R_1 and R_2 are regular expressions with exponentiation:

1. Convert R_1 and R_2 to equivalent regular expressions B_1 and B_2 that use repetition instead of exponentiation.
2. Convert B_1 and B_2 to equivalent NFAs N_1 and N_2, using the conversion procedure given in the proof of Lemma 1.55.
3. Use the deterministic version of algorithm N to determine whether N_1 and N_2 are equivalent."

Algorithm E obviously is correct. To analyze its space complexity, we observe that using repetition to replace exponentiation may increase the length of an expression by a factor of 2^l, where l is the sum of the lengths of the ex-

EXPONENTIAL SPACE COMPLETENESS

ponents. Thus, expressions B_1 and B_2 have a length of at most $n2^n$, where n is the input length. The conversion procedure of Lemma 1.55 increases the size linearly, and hence NFAs N_1 and N_2 have at most $O(n2^n)$ states. Thus, with input size $O(n2^n)$, the deterministic version of algorithm N uses space $O((n2^n)^2) = O(n^2 2^{2n})$. Hence $EQ_{\mathsf{REX}\uparrow}$ is decidable in exponential space.

Next, we show that $EQ_{\mathsf{REX}\uparrow}$ is EXPSPACE-hard. Let A be a language that is decided by TM M running in space $2^{(n^k)}$ for some constant k. The reduction maps an input w to a pair of regular expressions, R_1 and R_2. Expression R_1 is Δ^* where if Γ and Q are M's tape alphabet and states, $\Delta = \Gamma \cup Q \cup \{\#\}$ is the alphabet consisting of all symbols that may appear in a computation history. We construct expression R_2 to generate all strings that aren't rejecting computation histories of M on w. Of course, M accepts w iff M on w has no rejecting computation histories. Therefore, the two expressions are equivalent iff M accepts w. The

Thank You!!!