The LOGSPACE Complexity Class

Last Updated May 12th, 2024

1 Log Space

In this lecture we study decision problems that may be decided using $\mathrm{O}(\log n)$ space, where n is the input size. However, to do this we must modify the Turing machine definition so that a machine now has two tapes: a read-only tape for holding the input, and a read-write *scratchwork tape* for the purpose of computing on behalf of deciding the input. A computation configuration is similar to that of the original TM model, but now the configuration must include a nonnegative integer that indicates the current location of the read-only tape head.

Definition 1.1. Decision problem A is a member of L iff it is decidable by a DTM that uses $\mathrm{O}(\log n)$ scratchwork-tape cells when deciding an instance x of A for which $n = |x|$. Similarly, A is a member of NL iff it is decidable by an NTM that uses $\mathrm{O}(\log n)$ scratchwork-tape cells when deciding an instance x of A for which $n = |x|$.

Example 1.2. Consider the language A consisting of all words of the form 0^n1^n for some $k \geq 0$. Then $A \in \mathtt{L}$ since a DTM M can count the number of 0's that begin a word, and then count the number of 1's that follow. If the two counts are equal and the 1's are not followed by a 0, then M accepts. The two counters each require $\mathrm{O}(\log n)$ amount of memory and therefore $A \in \mathtt{L}$.

Example 1.3. An instance of decision problem `Path` is a triple (G, s, t), where $G = (V, E)$ is a directed graph, $s, t \in V$, and the problem is to decide if there is a path in G that starts at s and ends at t. We may assume that G is represented in the following format:

$$u_1 : (v_{11}, \dots, v_{k_1 1}), \dots, u_n : (v_{1n}, \dots, v_{k_n n}),$$

were, e.g., the vertices $v_{11}, \dots, v_{k_1 1}$ are the **neighbors** of u_1 which is denoted as $N(u_1)$.

The following nondeterministic log-space algorithm proves that $\texttt{Path} \in \texttt{NL}$.

> Name: `can_reach`
>
> Inputs: i) directed graph $G = (V, E)$, ii) $s \in V$, iii) $t \in V$
>
> Output: 1 iff there is a path in G from s to t.
>
> If $s = t$, then return 1.
>
> $u = \texttt{guess}(N(s))$.
>
> Return $\texttt{can\_reach}(G, u, t)$.

The algorithm will certainly return 1 on at least one branch iff t is reachable from s.

In terms of memory used, the algorithm only needs to store a copy of t and the current value of u. Letting $n = |V|$, we see that both vertices may be encoded using $\mathrm{O}(\log n)$ bits using a uniform-length binary encoding scheme. Therefore, $\texttt{Path} \in \texttt{NL}$. $\square$

2 Log Space Reducibility

We would like to have a meaningful notion of reducibility when it comes to problems in either `L` or `NL`. Although polynomial-time reducibility seems very appropriate for the generally complex class of problems in both `NP` and `PSPACE`, since it turns out that *every* problem in `NL` is polynomial solvable, polynomial-time reducibility seems too strong for log-space problems (just as polynomial-*space* reducibility is too strong for `PSPACE`). Instead, we introduce the notion of log space reducibility.

Definition 2.1. A **transducer** T is a type of Turing machine that consists of a read-only input tape, a read-write scratchwork tape, and a write-only output tape. T is called a **log space transducer** iff it's scratchwork tape uses $\mathrm{O}(\log n)$ cells, where n is the size of the input to T.

Definition 2.2. Function $f : A \to B$ is said to be **log space computable** iff there is a log space transducer T for which $f(x) = T(x)$ for all $x \in A$.

Definition 2.3. Decision problem A is **log space mapping reducible** to decision problem B, written $A \leq_L B$, iff there exists a log-space-computable function $f : A \to B$ for which x is a positive instance of A iff $f(x)$ is a positive instance of B.

Theorem 2.4. If $A \leq_L B$ and $B \in \mathtt{L}$, then $A \in \mathtt{L}$.

Proof Idea. Assume $A \leq_L B$ via log space computable function $f : A \to B$, where f is computed by T. Let M_B be the log space computing TM that decides B. We now describe a log space computing TM Q that decides A.

1. On input x, Q's ultimate goal is to simulate M_B on input $f(x)$ and accept x iff M_B accepts $f(x)$.
2. Problem: $f(x)$ could be very large, as in polynomial with respect to $|x|$. Assume $|f(x)| \leq cn^k$ for constants $c, k > 0$. This assumption is validated Lemma 1.5 of the Space Complexity lecture.
3. Solution: repeat the following until the computation of M_B on input $f(x)$ has been completed.
   (a) For each step of the simulation of M_B on input $f(x)$, keep track of the location i of M_B's read-only tape head.
   (b) Before applying M_B's δ-transition function (which requires knowing the current input symbol at location i), simulate T up to when T writes the i th symbol w_i on to the output tape.
   (c) Use w_i for the purpose of applying M_B's δ-transition function to obtain the next configuration.
4. Accept x iff B accepts $f(x)$.

Program Q uses $\mathbf{O}(\log n)$ space.

1. Storing programs M_B and T requires O(1) space.
2. Storing the current configuration of M_B requires $\mathrm{O}(\log(cn^k)) = \mathrm{O}(\log n)$ space since the size of any configuration in the computation $M_B(f(x))$ is logarithmic with respect to the size of $f(x)$ which is bounded by cn^k.
3. Storing a configuration of T requires at most $\mathrm{O}(\log n)$ space since T itself uses at most $\mathrm{O}(\log n)$ space.
4. Similar to 2, the variable that holds the current location of M_B's input head requires $\mathrm{O}(\log(cn^k)) = \mathrm{O}(\log n)$ space.

3 $\mathtt{NL}$-Completeness

Definition 3.1. Decision problem B is said to be **$\mathtt{NL}$-complete** iff

1. $B \in \mathtt{NL}$
2. for every other decision problem $A \in \mathtt{NL}$, $A \leq_L B$.

A corollary to Theorem 2.4 (exercise) is that if any $\mathtt{NL}$-complete language is in $\mathtt{L}$, then $\mathtt{L} = \mathtt{NL}$.

Theorem 3.2. `Path` is `NL`-complete.

Proof Idea. Let $A \in \texttt{NL}$ be given and suppose NTM N decides A using log space scratchwork.

1. Assume that N has a unique accepting configuration c_a, regardless of input.
2. There is a constant $d > 0$ such that, for an input x of size n, the computation $M(x)$ uses configurations that may be written using at most $d \log n$ tape cells.
3. Given instance x of A, with $|x| = n$, define the **configuration graph** $G_x = (V, E)$, where
   (a) $c \in V$ iff c is a valid configuration for M (note: it's possible that c may never get used in the computation $M(x)$), and
   (b) $(c_1, c_2) \in E$ iff c_2 is a possible next configuration given that c_1 is the current configuration of some computation of M, assuming x as input.
4. There is a log space transducer T for which, given input $x \in A$, $T(x)$ outputs G_x in a format similar to the one described in Example 1.3. True, since
   (a) T may go through all possible legal configurations of N that have size at most $d \log n$.
   (b) For each legal configuration c encountered, T then uses N's δ-transition function to list all the configurations c' that are reachable from vertex c in one step, i.e. $(c, c') \in E$.
5. After writing G_x, T then writes c_x and c_a, where c_x is the initial configuration of the computation $N(x)$.
6. Therefore, x is a positive instance of A iff (G_x, c_x, c_a) is a positive instance of `Path`, since N accepts x iff there is a path of configurations from c_x to c_a. □

Corollary 3.3. $\texttt{NL} \subseteq \texttt{P}$.

Proof. Let $A \in \texttt{NL}$ be given and consider the reduction from A to `PATH` provided in the previous theorem. This is not only a log space reduction, but it is also a polynomial time reduction (exercise!). But, since $\texttt{Path} \in \texttt{P}$, it follows that $A \in \texttt{P}$. In other words, any decision problem that is polynomial-time reducible to a problem in `P`, must also be in `P` (exercise!). □

Corollary 3.4. $\texttt{TQBF} \not\in \texttt{NL}$.

Proof. By the Space Hierarchy Theorem, and the fact that every `NL` problem can be decided using a polynomial amount of space, it follows that `NL` is properly contained in `PSPACE`. But `TQBF` is `PSPACE`-complete and it can be shown that the mapping reduction used to prove this is in fact a log space reduction. Thereore, if `TQBF` were in `NL`, then all of `PSPACE` would be contained in `NL`, which contradicts the Space Hierarchy Theorem. □

Definition 3.5. The `UPath` decision problem is similar to `Path`, but now the input graph is assumed undirected.

Observations about `Path`.

- `UPath` $\in$ `NL` (same algorithm that was used for `Path`)
- It is not known if `UPath` is `NL`-complete
- Although `Path` can be trivially reduced to `UPath` in polynomial time (why?), there is no known way to reduce `Path` to `UPath` in log space.
- It is not known if either `UPath` or `Path` is in `L`
- However, there is a polynomial-time randomized log-space algorithm that decides if `UPath` with a probability of error that can be made arbitrarily small!

A polynomial-time randomized log-space algorithm that decides `UPath`

Name: `can_reach`

Inputs:

1. undirected graph $G = (V, E)$
2. $s \in V$
3. $t \in V$
4. polynomial $p(e, n)$, where $e = |E|$ and $n = |V|$
5. nonnegative integer `steps`

Output: 1 iff a path in G from s to t is discovered.

If $s = t$, then return 1.

If $\texttt{steps} = p(e, n)$, then return 0.

$u = \texttt{random}(N(s))$.

Return $\texttt{can\_reach}(G, u, p(e, n), \texttt{steps} + 1)$.

Theorem 3.6. There is a quadratic polynomial $p(e, n)$ such that, if, for sufficiently large n, if the above algorithm $\mathcal{A}$ is run for $p(e, n)$ steps and t is reachable from s, then with probability at least 1/2, then $\mathcal{A}$ returns 1.