L (complexity)

In computational complexity theory, **L** (also known as **LSPACE** or **DLOGSPACE**) is the complexity class containing decision problems that can be solved by a deterministic Turing machine using a logarithmic amount of writable memory space.[1][2] Formally, the Turing machine has two tapes, one of which encodes the input and can only be read, whereas the other tape has logarithmic size but can be read as well as written. Logarithmic space is sufficient to hold a constant number of pointers into the input[1] and a logarithmic number of boolean flags, and many basic logspace algorithms use the memory in this way.

Contents

Complete problems and logical characterization

Every non-trivial problem in **L** is complete under log-space reductions,[3] so weaker reductions are required to identify meaningful notions of **L**-completeness, the most common being first-order reductions.

A 2004 result by Omer Reingold shows that USTCON, the problem of whether there exists a path between two vertices in a given undirected graph, is in **L**, showing that **L** = **SL**, since USTCON is **SL**-complete.[4]

One consequence of this is a simple logical characterization of **L**: it contains precisely those languages expressible in first-order logic with an added commutative transitive closure operator (in graph theoretical terms, this turns every connected component into a clique). This result has application to database query languages: *data complexity* of a query is defined as the complexity of answering a fixed query considering the data size as the variable input. For this measure, queries against relational databases with complete information (having no notion of nulls) as expressed for instance in relational algebra are in **L**.

Related complexity classes

L is a subclass of **NL**, which is the class of languages decidable in logarithmic space on a nondeterministic Turing machine. A problem in **NL** may be transformed into a problem of reachability in a directed graph representing states and state transitions of the nondeterministic machine, and the logarithmic space bound implies that this graph has a polynomial number of vertices and edges, from which it follows that **NL** is contained in the complexity class **P** of problems solvable in deterministic polynomial time.[5] Thus $\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{P}$. The inclusion of **L** into **P** can also be proved more directly: a decider using $O(\log n)$ space cannot use more than $2^{O(\log n)} = n^{O(1)}$ time, because this is the total number of possible configurations.

L further relates to the class **NC** in the following way: $\mathbf{NC}^1 \subseteq \mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{NC}^2$. In words, given a parallel computer C with a polynomial number $O(n^k)$ of processors for some constant k, any problem that can be solved on C in $O(\log n)$ time is in **L**, and any problem in **L** can be solved in $O(\log^2 n)$ time on C.

Important open problems include whether **L** = **P**,[2] and whether **L** = **NL**.[6] It is not even known whether **L** = **NP**.[7]

The related class of function problems is **FL**. **FL** is often used to define logspace reductions.

Additional properties

L is low for itself, because it can simulate log-space oracle queries (roughly speaking, "function calls which use log space") in log space, reusing the same space for each query.

Other uses

The main idea of logspace is that one can store a polynomial-magnitude number in logspace and use it to remember pointers to a position of the input.

The logspace class is therefore useful to model computation where the input is too big to fit in the RAM of a computer. Long DNA sequences and databases are good examples of problems where only a constant part of the input will be in RAM at a given time and where we have pointers to compute the next part of the input to inspect, thus using only logarithmic memory.

See also

- L/poly, a nonuniform variant of L that captures the complexity of polynomial-size branching programs

Notes

1. Sipser (1997), Definition 8.12, p. 295.
2. Garey & Johnson (1979), p. 177.
3. See Garey & Johnson (1979), Theorem 7.13 (claim 2), p. 179.
4. Reingold, Omer (2005). Undirected ST-connectivity in log-space (http://www.wisdom.weizmann.ac.il/~reingold/publications/sl.ps). STOC'05: Proceedings of the 37th Annual ACM Symposium on Theory of Computing. ACM, New York. pp. 376–385. doi:10.1145/1060590.1060647 (https://doi.org/10.1145%2F1060590.1060647). MR 2181639 (https://www.ams.org/mathscinet-getitem?mr=2181639). ECCC TR04-094 (https://eccc.weizmann.ac.il/report/2004/094/).
5. Sipser (1997), Corollary 8.21, p. 299.
6. Sipser (1997), p. 297; Garey & Johnson (1979), p. 180.
7. https://cs.stackexchange.com/questions/103073/is-it-possible-that-l-np

References

- Arora, Sanjeev; Barak, Boaz (2009). Computational complexity. A modern approach. Cambridge University Press. ISBN 978-0-521-42426-4. Zbl 1193.68112 (https://zbmath.org/?format=complete&q=an:1193.68112).
- Papadimitriou, Christos (1993). Computational Complexity (1st ed.). Addison Wesley. Chapter 16: Logarithmic space, pp. 395–408. ISBN 0-201-53082-1.
- Sipser, Michael (1997). Introduction to the Theory of Computation (https://archive.org/details/introductiontoth00sips). PWS Publishing. Section 8.4: The Classes L and NL, pp. 294–296. ISBN 0-534-94728-X.
- Garey, M.R.; Johnson, D.S. (1979). Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman. Section 7.5: Logarithmic Space, pp. 177–181 (https://archive.org/details/computersintract0000gare/page/). ISBN 0-7167-1045-5.
- Cook, Stephen A.; McKenzie, Pierre (1987). "Problems Complete for Deterministic Logarithmic Space" (http://www.cs.utoronto.ca/~sacook/homepage/cook_mckenzie.pdf) (PDF). Journal of Algorithms. **8** (3): 385–394. doi:10.1016/0196-6774(87)90018-6 (https://doi.org/10.1016%2F0196-6774%2887%2990018-6). ISSN 0196-6774 (https://www.worldcat.org/issn/0196-6774).