WIKIPEDIA

Polynomial-time reduction

In computational complexity theory, a **polynomial-time reduction** is a method for solving one problem using another. One shows that if a hypothetical subroutine solving the second problem exists, then the first problem can be solved by transforming or reducing it to inputs for the second problem and calling the subroutine one or more times. If both the time required to transform the first problem to the second, and the number of times the subroutine is called is polynomial, then the first problem is polynomial-time reducible to the second.[1]

A polynomial-time reduction proves that the first problem is no more difficult than the second one, because whenever an efficient algorithm exists for the second problem, one exists for the first problem as well. Conversely, if no efficient algorithm exists for the first problem, none exists for the second either.[1] Polynomial-time reductions are frequently used in complexity theory for defining both complexity classes and complete problems for those classes.

Contents

Types of reductions

The three most common types of polynomial-time reduction, from the most to the least restrictive, are polynomial-time many-one reductions, truth-table reductions, and Turing reductions. The most frequently used of these are the many-one reductions, and in some cases the phrase "polynomial-time reduction" may be used to mean a polynomial-time many-one reduction.[2] The most general reductions are the Turing reductions and the most restrictive the Many-one reductions with Truth-table reductions occupying the space in between.[3]

Many-one reductions

A polynomial-time many-one reduction from a problem A to a problem B (both of which are usually required to be decision problems) is a polynomial-time algorithm for transforming inputs to problem A into inputs to problem B, such that the transformed problem has the same output as the original problem. An instance x of problem A can be solved by applying this transformation to produce an instance y of problem B, giving y as the input to an algorithm for problem B, and returning its output. Polynomial-time many-one reductions may also be known as **polynomial transformations** or **Karp reductions**, named after Richard Karp. A reduction of this type is denoted by $A \leq_m^P B$ or $A \leq_p B$.[4][1]

Truth-table reductions

A polynomial-time truth-table reduction from a problem A to a problem B (both decision problems) is a polynomial time algorithm for transforming inputs to problem A into a fixed number of inputs to problem B, such that the output for the original problem can be expressed as a function of the outputs for B. The function that maps outputs for B into the output for A must be the same for all inputs, so that it can be expressed by a truth table. A reduction of this type may be denoted by the expression $A \leq_{tt}^{P} B$.[5]

Turing reductions

A polynomial-time Turing reduction from a problem A to a problem B is an algorithm that solves problem A using a polynomial number of calls to a subroutine for problem B, and polynomial time outside of those subroutine calls. Polynomial-time Turing reductions are also known as **Cook reductions**, named after Stephen Cook. A reduction of this type may be denoted by the expression $A \leq_{T}^{P} B$.[4] Many-one reductions can be regarded as restricted variants of Turing reductions where the number of calls made to problem B is exactly one.

Completeness

A complete problem for a given complexity class **C** and reduction ≤ is a problem P that belongs to **C**, such that every problem A in **C** has a reduction $A \leq P$. For instance, a problem is **NP**-complete if it belongs to **NP** and all problems in **NP** have polynomial-time many-one reductions to it. A problem that belongs to **NP** can be proven to be **NP**-complete by finding a single polynomial-time many-one reduction to it from a known **NP**-complete problem.[6] Polynomial-time many-one reductions have been used to define complete problems for other complexity classes, including the **PSPACE**-complete languages and **EXPTIME**-complete languages.[7]

Every decision problem in **P** (the class of polynomial-time decision problems) may be reduced to every other nontrivial decision problem (where nontrivial means that not every input has the same output), by a polynomial-time many-one reduction. To transform an instance of problem A to B, solve A in polynomial time, and then use the solution to choose one of two instances of problem B with different answers. Therefore, for complexity classes within **P** such as **L**, **NL**, **NC**, and **P** itself, polynomial-time reductions cannot be used to define complete languages: if they were used in this way, every nontrivial problem in **P** would be complete. Instead, weaker reductions such as log-space reductions or **NC** reductions are used for defining classes of complete problems for these classes, such as the **P**-complete problems.[8]

Defining complexity classes

The definitions of the complexity classes **NP**, **PSPACE**, and **EXPTIME** do not involve reductions: reductions come into their study only in the definition of complete languages for these classes. However, in some cases a complexity class may be defined by reductions. If C is any decision problem, then one can define a complexity class **C** consisting of the languages A for which $A \leq_{m}^{P} C$. In this case, C will automatically be complete for **C**, but **C** may have other complete problems as well.

An example of this is the complexity class $\exists\mathbb{R}$ defined from the existential theory of the reals, a computational problem that is known to be **NP**-hard and in **PSPACE**, but is not known to be complete for **NP**, **PSPACE**, or any language in the polynomial hierarchy. $\exists\mathbb{R}$ is the set of problems having a polynomial-time many-one reduction to the existential theory of the reals; it has several other complete problems such as determining the rectilinear crossing number of an undirected graph. Each problem in $\exists\mathbb{R}$ inherits the property of belonging to **PSPACE**, and each $\exists\mathbb{R}$-complete problem is **NP**-hard.[9]

Similarly, the complexity class **GI** consists of the problems that can be reduced to the graph isomorphism problem. Since graph isomorphism is known to belong both to **NP** and co-**AM**, the same is true for every problem in this class. A problem is **GI**-complete if it is complete for this class; the graph isomorphism problem itself is **GI**-complete, as are several other related problems.[10]

See also

- Karp's 21 NP-complete problems

External links

- MIT OpenCourseWare: 16. Complexity: P, NP, NP-completeness, Reductions (https://www.youtube.com/watch?v=eHZifpgyH_4)

References

1. Kleinberg, Jon; Tardos, Éva Tardos (2006). *Algorithm Design*. Pearson Education. pp. 452–453. ISBN 978-0-321-37291-8.
2. Wegener, Ingo (2005), *Complexity Theory: Exploring the Limits of Efficient Algorithms*, Springer, p. 60, ISBN 9783540274773.
3. Mandal, Debasis; Pavan, A.; Venugopalan, Rajeswari (2014). *Separating Cook Completeness from Karp-Levin Completeness under a Worst-Case Hardness Hypothesis* (http://drops.dagstuhl.de/opus/volltexte/2014/4862/). 34th International Conference on Foundation of Software Technology and Theoretical Computer Science. ISBN 978-3-939897-77-4.
4. Goldreich, Oded (2008), *Computational Complexity: A Conceptual Perspective*, Cambridge University Press, pp. 59–60, ISBN 9781139472746
5. Buss, S.R.; Hay, L. (1988), "On truth-table reducibility to SAT and the difference hierarchy over NP", *Proceedings of Third Annual Structure in Complexity Theory Conference*, pp. 224–233, CiteSeerX 10.1.1.5.2387 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.5.2387), doi:10.1109/SCT.1988.5282 (https://doi.org/10.1109%2FSCT.1988.5282), ISBN 978-0-8186-0866-7.
6. Garey, Michael R.; Johnson, D. S. (1979), *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman.
7. Aho, A. V. (2011), "Complexity theory", in Blum, E. K.; Aho, A. V. (eds.), *Computer Science: The Hardware, Software and Heart of It*, pp. 241–267, doi:10.1007/978-1-4614-1168-0_12 (https://doi.org/10.1007%2F978-1-4614-1168-0_12), ISBN 978-1-4614-1167-3. See in particular p. 255.
8. Greenlaw, Raymond; Hoover, James; Ruzzo, Walter (1995), *Limits To Parallel computation; P-Completeness Theory*, ISBN 978-0-19-508591-4. In particular, for the argument that every nontrivial problem in P has a polynomial-time many-one reduction to every other nontrivial problem, see p. 48.
9. Schaefer, Marcus (2010), "Complexity of some geometric and topological problems" (http://ovid.cs.depaul.edu/documents/convex.pdf) (PDF), *Graph Drawing, 17th International Symposium, GS 2009, Chicago, IL, USA, September 2009, Revised Papers*, Lecture Notes in Computer Science, **5849**, Springer-Verlag, pp. 334–344, doi:10.1007/978-3-642-11805-0_32 (https://doi.org/10.1007%2F978-3-642-11805-0_32), ISBN 978-3-642-11804-3.
10. Köbler, Johannes; Schöning, Uwe; Torán, Jacobo (1993), *The Graph Isomorphism Problem: Its Structural Complexity*, Birkhäuser, ISBN 978-0-8176-3680-7, OCLC 246882287 (https://www.worldcat.org/oclc/246882287).