Pina Kolling
piko0011

Summary: Lecture 3

Summary for the chapter *7.5* until page 150. [2, 3]

Background knowledge

The P vs. NP problem is an unsolved problem in complexity theory. Mathematical problems which are solved by a computer are classified as P or NP problems: All problems that can be solved efficiently by a computer belong to the class P. In the case of NP problems it is unknown whether they can be solved efficiently or not. In this context, efficient means that the required computing time of a solution algorithm grows at most polynomially (e.g. quadratically) with increasing complexity. The only thing that is currently clear is that a correct solution to an NP problem can be checked for correctness in polynomial time.
Developing an algorithm for an NP problem is usually very difficult. Computer scientists and mathematicians do not only try to work out effective algorithms, they also try to determine whether P = NP. In other words, they are trying to find out whether P and NP are really different problem classes or whether it is possible to solve NP problems in polynomial time as well. Scientists all over the world are trying to prove that P != NP.
The reason why experts wish NP problems to remain almost unsolvable is called cryptography. Unlike many other fields, complexity in cryptography is not only desirable, but necessary. It is important to know that most encryption methods used today are based solely on the fact that the effort to *guess* the key is too high. The problem of *guessing* is therefore an NP problem. Not only theoretically, but also practically, the proof of the solvability of NP problems means the end of all currently used encryption methods.
But there is still hope for cryptography. The NP problem has often been supposedly solved. Both P = NP and P != NP have been attempted to be proven many times. It turned out that every single solution and every single proof attempt so far turned out to be wrong or incomprehensible. [1, 3, 4]

Basic relations between complexity classes

The hierarchy theorem shows how deterministic classes of the same kind (time or space) relate to each other. Here are the relationships between classes of a different kind exmined: P and NP. If $f(n)$ is a proper complexity function:

$$\text{SPACE}(f(n)) \subseteq \text{NSPACE}(f(n)) \text{ and } \text{TIME}(f(n)) \subseteq \text{NTIME}(f(n)) \tag{1}$$

$$\text{NTIME}(f(n)) \subseteq \text{SPACE}(f(n)) \tag{2}$$

$$\text{NSPACE}(f(n)) \subseteq \text{TIME}(k^{\log n + f(n)}) \tag{3}$$

Proof for (1)**:**
Any deterministic Turing Machine is also a non-deterministic Turing Machine, which leads to $\text{SPACE}(f(n)) \subseteq \text{NSPACE}(f(n))$ and $\text{TIME}(f(n)) \subseteq \text{NTIME}(f(n))$.

Proof for (2)**:**
To prove (2), a language $L \in \text{NTIME}(f(n))$ is considered. It exists a precis non-deterministic Turing Machine M that decides L in time $f(n)$ and for the prove a deterministic Turing Machine M' has to be designed that decides L in space $f(n)$. Memory space can be reused, time cannot. There are d choices in every step: $1, ..., d$. For each choice the non-determinstic Turing Machine is simulated and the previous simulations can be deleted to reuse space. Only the currently simulated and the next simulation need to be saved and tracked.

Proof for (3)**:**
This proof involves a method for space-bounded Turing Machines, which is called **reachability method**. For this, a multistring non-deterministic Turing Machine M with input and output is given, which decides a lanugage L within space $f(n)$. Then things happen and graphs/graph

edges are constructed, M empties the tape and puts all the heads to the start and there is only a single node that is accepting.

Combining the previous knowledge, it can be concluded:

$$\mathrm{L} \subseteq \mathrm{NL} \subseteq \mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE}$$

and

$$\mathrm{L} \subset \mathrm{PSPACE}$$

This means, that at least on of the inclusions has to be proper.

Questions: What happens in the reachability method proof?!

Problem: GRAPH REACHABILITY

Given a graph G and two nodes $n_1, n_2 \in V$, is there path from n_1 to n_2?
(A graph $G = (V, E)$ is a finite set V of nodes and a set E of edges (node pairs).)

Savitch's theorem

$$\mathrm{REACHABILITY} \in \mathrm{SPACE}(\log^2 n)$$

Proof:
PATH($startnode, endnode, m$) checks if there is a path from startnode to endnode with the length of at most 2^m.
The parameters of the function can be split up (like divide and conquer):

$$\mathrm{PATH}(i, j, m) \Leftrightarrow \exists k : \mathrm{PATH}(i, k, m-1) \text{ and } \mathrm{PATH}(k, j, m-1)$$

Some internal node k is choosen and it is checked recursively if there is a path from *startnode* to k and from *endnode* to n. This is then recursively done with the new midpoint.

There can be $\log n$ many segments to work on – the recursion depth and the record size both are at most $\log n$. This leads to a recursion stack of $O(\log^2 n)$.
The complexity function is at least $\log n$.

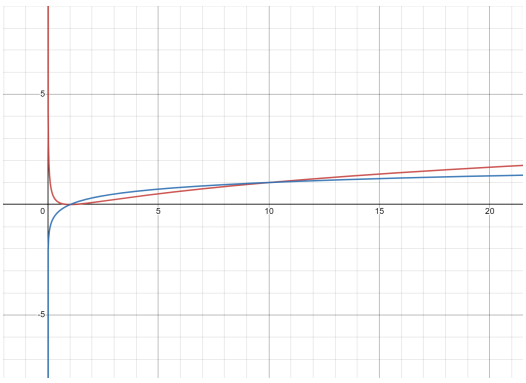

Figure 1: Red graph $f(n) = \log^2 n$ and blue graph $g(n) = \log n$

Questions: Why does the meaning of n change?

Pina Kolling
piko0011

Analysis

imagine the graph
graph can be too large to construct

TODO
Questions:

References

[1] Dr Datenschutz (Website). *P vs. NP: Ein Geschenk der Informatik an die Mathematik*. last opened 11.11.2022. URL: https://www.dr-datenschutz.de/p-vs-np-ein-geschenk-der-informatik-an-die-mathematik/#:~:text=Hierbei%20werden%20von%20einem%20Computer,effizient%20l%C3%B6sen%20lassen%20oder%20nicht..

[2] Martin Berglund. *Lecture notes in Computational Complexity*.

[3] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley Publishing Company, 1994.

[4] Prof. Dr. Thomas Schwentick. *Lecture notes in Grundbegriffe der theoretischen Informatik*. https://www.cs.tu-dortmund.de/nps/de/Studium/Ordnungen_Handbuecher_Beschluesse/Modulhandbuecher/Archiv/Bachelor_LA_GyGe_Inf_Modellv/_Module/INF-BfP-GTI/index.html.