DIGITALE
PROBLEMLÖSER

DIGITALE PROBLEMLÖSER

SCHNELLE ALGORITHMEN UND IHRE GRENZEN

FELIX JOOS

Algorithmen umgeben uns in unserem Alltag überall und ständig. Mit Hochdruck wird nach immer schnelleren Versionen davon zur Lösung immer komplexerer Probleme gesucht. Für manche Probleme aber wird es womöglich nie schnelle Algorithmen geben.

Einmal schnell das Handy aus der Tasche gezogen, das Navigationsprogramm geöffnet und nach der Route zum Lieblingsrestaurant gesucht. In Bruchteilen einer Sekunde spuckt die App eine mögliche Wegstrecke aus – die vermeintlich beste. Welchen Nutzen hätte solch eine App, wenn wir auch nur eine Minute auf die Antwort warten müssten? Unzählige technische Geräte lösen heute überall um uns herum kleine und größere Probleme. Die Hauptsache ist, es geht schnell, sehr schnell. Denn unser Anspruch ist es, alle Probleme umgehend zu lösen, alle Fragen rasch beantworten zu können. Und dafür braucht es neue, immer schnellere Algorithmen.

Das Sortierproblem

Ein „Algorithmus“ ist eine detaillierte Handlungsvorschrift, die einem (digitalen) Problemlöser vorschreibt,

wie er vorzugehen hat. Schauen wir uns ein Beispiel an. Gegeben ist eine Liste mit verschiedenen Zahlen, sagen wir 3, 1, 12, 4, 5, 8, 2, 11. Wie können wir diese Zahlen möglichst schnell aufsteigend anordnen?

Die Zeit, die es braucht, um diese Zahlenliste zu sortieren, können wir beispielsweise in Sekunden messen. Anstatt den Arbeitsaufwand in Sekunden zu messen, kann man auch wie folgt vorgehen: Wir zählen, wie viele sogenannte elementare Operationen der Algorithmus ausführt. Eine elementare Operation entspricht dabei zum Beispiel dem Vergleichen oder der Addition/Subtraktion/Multiplikation/Division zweier Zahlen oder dem Vertauschen zweier Zahlen. Ein Algorithmus könnte also beispielsweise die ersten beiden Zahlen vergleichen – also 3 und 1 – und sie danach vertauschen (zwei elementare Operationen).

Nun können wir verschiedene Aussagen treffen. Lassen Sie uns annehmen, dass eine Liste von n verschiedenen Zahlen gegeben ist, die wir sortieren möchten. An diesem Algorithmus kann man nun zum Beispiel eine „Worst-Case-Analyse" durchführen. Egal, wie die Liste von Zahlen am Anfang aussieht, ein bestimmter Algorithmus braucht höchstens $3n^2$ viele elementare Operationen. In aller Regel ist es bei solchen Betrachtungen egal, ob es $3n^2$ oder $20n^2$ ist. Man schreibt deshalb gerne $O(n^2)$. Und das bedeutet: Wir ignorieren alle Faktoren, die nicht von n, der Größe der Liste, abhängen und schreiben ein großes O davor. Eine andere Aussage wäre eine „Average-Case-Analyse": Im Durchschnitt (gemittelt über alle möglichen Listen) braucht ein Algorithmus höchstens $O(n \log n)$ viele elementare Operationen, um eine Liste zu sortieren (n^2 ist für große Werte von n viel größer als $n \log n$ und $n \log n$ ist im Wesentlichen so groß wie n, zumindest fast). Die besten Sortieralgorithmen haben eine Worst-Case-Laufzeit von $O(n \log n)$. Zur Veranschaulichung: Heutige Computer lösen Probleme innerhalb einer Sekunde, wenn der Algorithmus ungefähr 10^{10} elementare Operationen benötigt. Deswegen ist das Sortieren von rund einer Milliarde Zahlen (10^9) ziemlich schnell machbar.

Das Problem des Handlungsreisenden

Betrachten wir noch ein zweites Beispiel, das „Problem des Handlungsreisenden". Gegeben sind n verschiedene Städte, die der Handlungsreisende auf einer Rundreise genau einmal besuchen will. Was wir kennen, ist die Distanz zwischen je zwei Städten. Und die Fragen, die es zu lösen gilt, lauten: Was ist die kürzeste Rundreise, und gibt es eine Rundfahrt mit einer Strecke von höchstens 500 Kilometern?

An sich muss dafür gar nicht viel getan werden. Zunächst brauchen wir eine Liste aller Städte in der Reihenfolge, in der sie der Handlungsreisende nacheinander besuchen will. Allerdings gibt es ziemlich viele Möglichkeiten für solch eine Liste. Um genau zu sein: die Hälfte von $(n-1)! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot \ldots \cdot (n-2) \cdot (n-1)$.

JUNIORPROF. DR. FELIX JOOS leitet seit März 2020 die Arbeitsgruppe Theoretische Informatik am Institut für Informatik der Universität Heidelberg. Nach Mathematikstudium und Promotion an der Universität Ulm arbeitete er zunächst vier Jahre als Postdoktorand an der University of Birmingham (England). 2019 wurde er zum Juniorprofessor für Diskrete Mathematik an der Universität Hamburg berufen. Seither leitet er auch eine Forschungsgruppe, die im Rahmen des Emmy Noether-Programms für den akademischen Nachwuchs von der Deutschen Forschungsgemeinschaft (DFG) gefördert wird. 2020 erhielt Felix Joos den Lautenschläger-Preis für herausragende Nachwuchsforscher. Schwerpunkt seiner Forschungsarbeiten ist die Graphentheorie.

Kontakt: joos@informatik.uni-heidelberg.de

Leider ist dies bereits eine ziemlich große Zahl, selbst wenn wir nur 20 Städte betrachten, denn 19! ist ungefähr 10^{17}. Wenn wir wieder 10^{10} elementare Operationen zur Verfügung haben, sind das bei Weitem zu viele verschiedene Möglichkeiten, um alle Kombinationen zu testen und zu prüfen, ob eine Rundfahrt höchstens 500 Kilometer lang ist.

Geht es also auch etwas schneller? Wahrscheinlich nicht. Es gibt Algorithmen, die nur rund 2^n viele elementare Operationen brauchen. Allerdings ist das immer noch eine ziemlich schnell wachsende Zahl, zum Beispiel ist 2^{30} bereits rund 10^{10}.

Der entscheidende Unterschied zwischen den beiden beschriebenen Beispielen ist: Für das Sortierproblem kennen wir Algorithmen, deren Laufzeit durch ein Polynom in n nach oben beschränkt ist (also n oder n^2); die Laufzeiten von Algorithmen für das Problem des Handlungsreisenden hingegen sind nur durch eine Exponentialfunktion in n beschränkt (wie zum Beispiel 2^n). Exponentialfunktionen wachsen aber viel schneller als Polynome. Das haben wir spätestens in der Corona-Pandemie gelernt; der Wert von n^2 vervierfacht sich, wenn man n verdoppelt, allerdings vervierfacht sich der Wert von 2^n bereits, wenn n nur um zwei größer wird.

In der Theoretischen Informatik bezeichnen wir alle Algorithmen als „schnell lösbar", wenn es einen Algorithmus gibt, dessen Worst-Case-Laufzeit durch ein Polynom in n nach oben beschränkt werden kann – wie zum Beispiel das Sortieren: Hier gibt es einen Algorithmus mit Laufzeit $O(n^2)$. Die Menge aller solcher Probleme nennen wir P.

Es gibt allerdings eine Reihe von Problemen, für die wir keinen derart definierten schnellen Algorithmus kennen. Dazu gehört auch das Problem des Handlungsreisenden. Hier liegt die Schwierigkeit darin, zu zeigen, dass es tatsächlich keinen schnellen Algorithmus gibt. Denn der Nachweis der Existenz eines schnellen Algorithmus ist viel einfacher: Man muss ihn schließlich nur angeben, wohingegen der Nachweis einer Nichtexistenz grundlegende Schwierigkeiten mit sich bringt: Wie zeigt man nur, dass etwas nicht existiert?

Was machen wir also mit einem solch augenscheinlich schwierigeren Problem wie dem des Handlungsreisenden? Wir schrauben einfach unsere Erwartungen zurück. Stellen Sie sich vor, Sie bekommen von irgendwoher einen Lösungsversuch für das Rundfahrtproblem zugesteckt. Nun wollen wir wenigstens in der Lage sein zu überprüfen, ob dieser Lösungsvorschlag eine valide Lösung für unser Problem ist.

„Ein Algorithmus ist eine detaillierte Handlungsvorschrift, die einem (digitalen) Problemlöser vorschreibt, wie er vorzugehen hat.“

Das hört sich nach einer deutlich einfacheren Aufgabe an. In der Tat: Wir addieren die Distanzen zwischen den Städtepaaren, die der Handlungsreisende nacheinander besuchen will, und überprüfen, ob das Ergebnis unter 500 Kilometern liegt. Eine einfache Angelegenheit.

Gibt es also einen schnellen Algorithmus, wenn wir eine Lösung zugesteckt bekommen? Die Menge aller solcher Probleme nennen wir NP. Die Menge NP enthält P, da das Berechnen einer Lösung quasi die Überprüfung einer Lösung als Teilaufgabe enthält.

„Wir machen es uns auch zunutze, dass manche Probleme algorithmisch sehr schwer zu lösen sind.“

DIGITAL PROBLEM SOLVERS

FAST ALGORITHMS AND THEIR LIMITATIONS

FELIX JOOS

Algorithms are everywhere – we use numerous technical devices all day, every day. All these gadgets need fairly precise instructions telling them how to handle a particular task. These instructions are called algorithms. The enormous digital developments in the last few decades are strongly linked to advances regarding fast algorithms. Not surprisingly, processing speed is essential for many tasks; after all, who wants to use a satnav that needs several minutes to output a potential route? Not to mention if we miss a turn and instantly need directions for a new route. However, some algorithmic problems seem to be harder to solve at the conceptual level than others. One of the seven Millennium Prize Problems asks whether there are in fact two different classes of algorithms (this question is known as P=NP?). If this was not the case, it should be possible to quickly solve almost any algorithmic problem. This seems rather unlikely, but as yet there is no evidence either way.

At first glance, being able to quickly solve any algorithmic problem seems a good thing. However, our current cryptographic systems rely heavily on the assumption that some problems are too hard to be solved in a reasonable time frame. This in particular includes the following problem: given a number that is the product of two primes, compute the two unknown primes whose product is the given number. ●

JUNIOR PROF. DR FELIX JOOS has been heading the Theoretical Computer Science group at Heidelberg University's Institute for Computer Science since March 2020. He studied mathematics and earned his doctorate at the University of Ulm, then spent four years as a postdoc at the University of Birmingham (UK). In 2019 he accepted a junior professorship for discrete mathematics at the University of Hamburg. In the same year, he became head of a research group that is funded by the German Research Foundation (DFG) within the framework of the Emmy Noether Programme for junior academics. In 2020 Felix Joos was awarded the Lautenschläger Prize for outstanding early-career researchers. His main research interest is graph theory.

Contact: joos@informatik.uni-heidelberg.de

"An algorithm is a set of detailed instructions telling a (digital) problem solver how to proceed."

Was unterscheiden nun die Probleme in P und in NP? Was ist das grundlegend Verschiedene? Die Antwort lautet: Wir wissen es nicht.

Ist P = NP?

Im Jahr 2000 hat das „Mathematics Institute" in Cambridge (Massachusetts) eine Liste der sieben wichtigsten Probleme der Mathematik und Informatik zusammengestellt – in Anlehnung an die Liste der 23 wichtigen Probleme, die der deutsche Mathematiker David Hilbert im Jahr 1900 auf dem Weltkongress für Mathematik vorgestellt hatte. Für die Lösung jedes dieser sieben Probleme hat das Institut eine Million US-Dollar ausgelobt. Und eines dieser Probleme lautet: „Ist P = NP?"

Was würde es bedeuten, wenn P = NP ist? Es würde bedeuten: Für jedes Problem, für das man schnell überprüfen kann, ob eine vorgeschlagene Lösung auch wirklich eine Lösung ist, gibt es auch einen schnellen Algorithmus, der das ganze Problem lösen kann.

Die meisten Experten indes glauben, dass P nicht gleich NP ist. Wie wir uns bereits überlegt haben, ist es manchmal sehr schwer, die Nichtexistenz von Objekten nachzuweisen – und so könnte es auch hier sein. Allerdings wissen wir es nicht – und es gibt wenig vielversprechende Ansätze für die Lösung dieses Problems, einer der wichtigsten offenen Fragen in der Mathematik und Informatik.

Vielleicht hört es sich kurios an: Wir machen es uns auch zunutze, dass manche Probleme algorithmisch sehr schwer zu lösen sind. Auch hierzu ein Beispiel: Gegeben ist eine Zahl n, von der wir zusätzlich wissen, dass sie das Produkt zweier Primzahlen ist, beispielsweise 15, 21, 22 oder 187. Das heißt, n lässt sich schreiben als $p \cdot q$, wobei p und q (uns unbekannte) Primzahlen sind. Das zu lösende Problem lautet nun wie folgt: Berechne p und q.

Auch für nur mäßig große Zahlen p und q (in Computerdimensionen gedacht) brauchen selbst heutige Supercomputer Jahrmillionen, um p und q zu finden. Diese Schwerlösbarkeit nutzen wir tagtäglich aus, nämlich in der Kryptographie: Wir benutzen sie, um uns digitale Nachrichten zu schicken. Ein Hoch auf komplizierte Probleme!

Bei ganz vielen Problemen in NP ist relativ schnell klar: Wenn P wirklich nicht NP ist, dann liegen diese Probleme nicht in P. Wir möchten diese Probleme aber dennoch gerne lösen. Hierzu muss man die Theorie von der Praxis trennen. Viele theoretisch schwere Probleme – also Probleme, die nach unserer oben gegebenen Definition nicht mit einem schnellen Algorithmus zu lösen sind – sind bei Instanzen, die in der Praxis auftreten, recht gut lösbar. Ein Grund dafür kann sein, dass es in der Praxis gar nicht so viele komplexe Problemstellungen gibt, wie es das Problem a priori zulassen würde. In der Theorie arbeiten wir daran, die Instanzen aus mathematischer Sicht zu verstehen. Gibt es etwa Strukturen, die wir ausnützen können, um eine Lösung in bestimmten Fällen schnell zu berechnen?

In meiner Arbeitsgruppe befassen wir uns dazu vor allem mit „Graphen". Das sind im Grunde einfache Objekte: Sie bestehen aus einer endlichen Menge, das heißt, eine Menge von Punkten, Zahlen, Personen oder Bällen, die in Beziehung zueinander stehen können – ähnlich wie bei einer Mindmap. Für das Problem des Handlungsreisenden etwa kann man auf einem Blatt Papier jeder Stadt einen Punkt zuordnen, zwischen je zwei Städten eine Linie ziehen und diese Linie mit einer Zahl versehen, die die Entfernung beider Städte codiert. Es ist oft viel einfacher, mit einem solch abstrakten Objekt zu arbeiten, weil es unnötige Informationen ausblendet. Unsere Arbeitsgruppe benutzt jede Menge mathematischer Werkzeuge, um die Eigenschaften solcher Graphen besser zu verstehen – damit (theoretisch) bestimmte Probleme ganz schnell zu lösen sind. ●

„Ob P = NP ist, zählt zu den wichtigsten offenen Fragen der Mathematik und Informatik."