Algorithmenanalyse - Komplexität und Berechenbarkeit

Seit der Entwicklung der ersten Rechenmaschinen haben die Computer gewaltige Fortschritte erzielt. Seitdem Computer im großen Stile eingesetzt werden, interessiert man sich mehr und mehr dafür, welcher Aufwand für die Bearbeitung eines theoretisch lösbaren Problems erforderlich ist. Der Aufwand oder die **Komplexität** beschreibt den für die Bearbeitung eines algorithmisch lösbaren Problems erforderlichen Berechnungsaufwand (Zeitkomplexität) bzw. den dafür erforderlichen Speicherplatzbedarf (Platzkomplexität). Als Maß für die Zeitkomplexität dient dabei eine mehr oder weniger genaue Abschätzung der Anzahl der Rechenschritte, die zur Lösung des Problems erforderlich sind. Der Zeitbedarf wird dabei in Abhängigkeit von der Länge der Eingabe n ermittelt.

Aufgabe: Hinsichtlich der Laufzeitabschätzung unterscheidet man die drei Varianten *worst case*, *best case* und *average case*. Klären Sie mithilfe des Onlinelexikons Wikipedia (http://de.wikipedia.org/wiki/Zeitkomplexit%C3%A4t) die Bedeutung dieser Begriffe! Welcher der Fälle ist gemeint, wenn man Untersuchungen zur Laufzeitkomplexität anstellt?

Komplexitätsklassen

Eine Komplexitätsklasse ist eine Gruppe von Problemen (Algorithmen), die sich bezüglich Laufzeit oder Platzbedarf ähnlich verhalten. Sie wird durch eine obere Schranke für den Bedarf einer bestimmten Ressource (z.B. Anzahl der notwendigen Berechnungsschritte zur Lösung eines Problems) gebildet. Dabei reicht es nicht, einfach die Laufzeit auf einem bestimmten Rechner zu messen. Ein anderer Rechner kann viel länger oder kürzer brauchen. Um eine rechnerunabhängige Maßzahl zu haben, betrachtet man die Schritte, die ein Algorithmus ausführt. Wenn n die Größe der Eingabe ist, dann lässt sich die Schrittzahl des Algorithmus beispielsweise beschreiben mithilfe einer Funktion T(n). Es interessiert aber meist nicht die genaue Schrittzahl, sondern eine ungefähre Abschätzung.

Verfahren, deren Zeitverhalten als Polynom in n ausgedrückt werden kann, haben **polynomialen Aufwand** und r ist der Grad des Polynoms
$T(n) = a_r n^r + a_{r-1} n^{r-1} + \ldots + a_1 n^1 + a_0$ mit $r \in N$, $a_r, \ldots, a_0 \in R$, $a_r \neq 0$.
Wenn r = 1, spricht man von **linearem Aufwand**. r = 2 bzw. r = 3 bedeuten **quadratischer** bzw. **kubischer Aufwand**.

Beispiele für Algorithmen mit polynomialem Aufwand sind Sortieralgorithmen.

Ein Verfahren besitzt **exponentiellen Aufwand**, wenn die folgende funktionale Beziehung gilt:
$T(n) = c * z^n$, wobei $z, c \in R$, $c \neq 0$, $z > 1$.

Besonders bei exponentiell aufwändigen Problemen gerät man schnell in so lange Rechenzeiten, dass es sich bei der üblichen menschlichen Lebensdauer nicht lohnt, auf ein Ergebnis zu warten (siehe Beispiel Handlungsreisender). Deshalb gelten Probleme mit exponentiellem Aufwand gemeinhin als praktisch unlösbar!

Aufgaben:

1. Stellen Sie rechnerisch (mithilfe einer Tabellenkalkulation, z.B. EXCEL) fest, ab welchem n das Verfahren mit dem polynomialen Aufwand
   a) $y = f(n) = n$,
   b) $y = f(n) = n \log_2 n$ (wird als polynomial betrachtet) bzw.
   c) $y = f(n) = n^2$
   gegenüber $y = f(n) = 2^n$ vorzuziehen ist!
   Skizzieren Sie außerdem die Grafen der vier Funktionen in ein und dasselbe Koordinatensystem! Stellen Sie den Rechenaufwand T(n) als Funktion der Anzahl der Elemente n dar!
2. Oftmals können für ein und dasselbe Problem viele Algorithmen zur Lösung angegeben werden. Vor der praktischen Implementierung stellt sich dann die Frage nach dem effizientesten Lösungsalgorithmus. Analysieren und vergleichen Sie verschiedene Algorithmen zum Sortieren von Zahlen hinsichtlich der Anzahl der verwendeten Rechenschritte! Verwenden Sie dazu die folgende Website: http://www.wollw.de/Sortieren/Sortieren.html

Grenzen der Berechenbarkeit

Man kann sich leicht vorstellen, dass bei zunehmender Eingabelänge n (z.B. Anzahl der zu sortierenden Elemente) der Aufwand für die abzuarbeitenden Rechenschritte und den benötigten Speicherplatz immer mehr ansteigt. Es stellt sich also die Frage, ob es Grenzen für die Berechnung der Lösung von Problemen gibt. Diese Fragestellung führt auf das Problem der **Berechenbarkeit** von Problemen.
In diesem Zusammenhang unterscheidet man **theoretische** und **praktische** Berechenbarkeit.

Eine Funktion f ist *theoretisch berechenbar*, wenn es ein effektives Rechenverfahren gibt, dass für eine beliebige Eingabe x aus dem Definitionsbereich den Funktionswert f(x) berechnet.

Diese Definition wird oft auch als *intuitiver Berechenbarkeitsbegriff* bezeichnet. Daraus ergibt sich folgende Problemstellung: Ist tatsächlich jedes Problem berechenbar, bzw. entscheidbar? Oder gibt es Probleme, die prinzipiell nicht berechenbar sind? Oder im Sinne der Informatik: Sind mithilfe eines Rechners alle Probleme berechenbar, oder hat auch eine noch so mächtige Rechenmaschine prinzipielle Grenzen hinsichtlich der Berechenbarkeit? Überraschenderweise hat sich gezeigt, dass allgemein gilt: Die Menge der Algorithmen reicht nicht aus, um jede (definierbare) Funktion berechnen zu können. Es gibt also nichtberechenbare Funktionen bzw. absolut unlösbare Probleme (Bsp. Halteproblem für PASCAL-Programme: Es gibt kein Pascalprogramm, das für ein beliebiges PASCALprogramm p und eine beliebige Eingabe x die Frage beantwortet, ob p – auf x angewendet – nach endlich vielen Schritten anhält oder nicht.).
Der Begriff der theoretischen Berechenbarkeit ist leider nur sehr schwierig zu verstehen und soll von uns nicht weiter verfolgt werden!
Eine Funktion ist auch dann theoretisch berechenbar, wenn jedes effektive Rechenverfahren mindestens eintausend Jahre benötigt, um den Funktionswert f(x) zu berechnen. Eine solche Funktion ist aber ganz sicher nicht praktisch berechenbar.

Eine Funktion f ist *praktisch berechenbar*, wenn es ein Computerprogramm gibt, das für ein beliebiges x aus dem Definitionsbereich den Funktionswert f(x) auf einem Rechner in polynomialer Zeit berechnet.

Beispiele für praktisch unlösbare Probleme

Das Problem des Handelsreisenden (TSP – Traveling Salesman Problem)

Ein Vertreter einer Firma, dessen Bezirk eine bestimmte Anzahl von Städten umfasst, beginnt seine Reise stets von seiner Basis aus, besucht jede Stadt genau einmal und kehrt dann zur Basis zurück. Um die Reisekosten zu minimieren, hat der Vertreter eine Tabelle der gegenseitigen Entfernungen der Städte seines Bezirks zusammengestellt. Es ist ein Algorithmus zu entwickeln, der die Reihenfolge der Städte für den kürzesten Reiseweg des Vertreters und die minimale Streckenlänge ausgibt.

Algorithmus: Handlungsreisender

```
Eingabe: Anzahl der Städte n, Entfernungstabelle
p := ( 1, 2, 3, . . . , n)
MinTour := p
MinStrecke := Tourlänge(p)
Permutiere(n) {und ermittle neue MinTour und MinStrecke}
Ausgabe: MinTour, Minstrecke
```

Aufgabe 1: Wie man sich leicht überlegen kann, gibt es bei einer Tour durch n Städte genau n! mögliche Touren. Wenn für die Berechnung der Weglänge jeder Reiseroute der Algorithmus eine Millisekunde benötigt, wie lange dauert dann die Berechnung der minimalen Tour für n = 5, 7, 10, 12, 14, 15, 20, 22 Städte? Geben Sie Ihre Ergebnisse in passenden Zeiteinheiten (Sekunde, Stunde, Tag, Jahr) an!

Fazit: Da die Anzahl der Permutationen einer n-elementigen Menge gleich n! ist, erfordert der Algorithmus einen zu n! proportionalen Zeitaufwand und ist somit praktisch nicht berechenbar. Und das obwohl sich ein Algorithmus angeben lässt!

Aufgabe 2: Im Internet findet man eine Reihe von Routenplanungsprogrammen, die den kürzesten Weg zu einem gewünschten Ziel bestimmen. Warum findet das Programm in kürzester Zeit eine Lösung, obwohl Routenplanung doch zu den praktisch nicht lösbaren Problemen zählt?

Das Schachspiel

Aufgabe: Erläutern Sie, inwiefern das Schachspiel ein praktisch unlösbares Problem darstellt! (Kleiner Leitfaden Informatik S. 515 f)