Samuel Domiks
Pro-Seminar Theoretische Informatik
bei Prof. Dr. Helmut Alt

18.11.2014

Komplexitätsklasse P

Analyse von Algorithmen

Da wir das Thema Komplexitätsklasse P behandeln werden, ist vorher notwendig zu Wissen, wie man Algorithmen analysiert, sodass man dieses dann einordnen kann. Hierzu werden hier zur Erinnerung einige Definition angegeben, auf die wir nicht näher eingehen werden.

Definition 1.1

Let M be a deterministic Turing machine that halts on all inputs. The ***running time*** or ***time complexity*** of M is the function $f: \mathcal{N} \longrightarrow \mathcal{N}$, where $f(n)$ is the maximum number of steps that M uses on any input of length n. If $f(n)$ is the running time of M, we say that M runs in time $f(n)$ and that M is an $f(n)$ time Turing machine. Customarily we use n to represent the length of the input.

[1]

Definition 1..2

Let f and g be functions $f, g: \mathcal{N} \longrightarrow \mathcal{R}^+$. Say that $f(n) = O(g(n))$ if positive integers c and n_0 exist such that for every integer $n \geq n_0$

$$f(n) \leq c\, g(n).$$

When $f(n) = O(g(n))$ we say that $g(n)$ is an ***upper bound*** for $f(n)$, or more precisely, that $g(n)$ is an ***asymptotic upper bound*** for $f(n)$, to emphasize that we are suppressing constant factors.

[2]

Nun kommen wir zur Analyse von Algorithmen den wir mit einer Turing – Maschine wiedergeben werden. Wir haben eine entscheidbare Sprache $A = \{ 0^k 1^k \mid k \geqslant 0 \}$

deterministische Einband TM:

M_1 = "On input string w:

1. Scan across the tape and *reject* if a 0 is found to the right of a 1.
2. Repeat if both 0s and 1s remain on the tape:
3. Scan across the tape, crossing off a single 0 and a single 1.
4. If 0s still remain after all the 1s have been crossed off, or if 1s still remain after all the 0s have been crossed off, *reject*. Otherwise, if neither 0s nor 1s remain on the tape, *accept*."

[3]

nicht deterministische TM:

M_3 = "On input string w:

1. Scan across the tape and *reject* if a 0 is found to the right of a 1.
2. Scan across the 0s on tape 1 until the first 1. At the same time, copy the 0s onto tape 2.
3. Scan across the 1s on tape 1 until the end of the input. For each 1 read on tape 1, cross off a 0 on tape 2. If all 0s are crossed off before all the 1s are read, *reject*.
4. If all the 0s have now been crossed off, *accept*. If any 0s remain, *reject*."

[4]

Wie wir am obigen Beispiel sehen, ist der Algorithmus $M_1(n) = O(n^2)$, wobei $n = |w|$, denn M1 streicht bei Stufe 3 jeweils 2 Elemente, dafür muss er durch das komplette Wort durch, welches O(n) zeit benötigt , und dies n/2 mal wiederholt. Somit liegt M1 in $O(n^2)$. Algorithmus M2 liegt in $O(n)$, da dort jede Stufe nur einmal ausgeführt wird, und diese $O(n)$ – Zeit benötigt. Wie uns nun auffällt, ist die Zeitkomplexität eines Algorithmus je nach Berechnungsmodell unterschiedlich, sodass ein Algorithmus der in einem Berechnungsmodell $O(n)$ Zeit benötigt in einem anderen $O(n^2)$ benötigen kann. Dies ist der Unterschied zwischen der Berechenbarkeitstheorie und der Komplexitätstheorie. In der Berechenbarkeitstheorie sind alle sinnvollen Berechenbarkeitsmodelle äquivalent , sie entscheiden die selbe Klasse von Sprachen. In der Komplexitätstheorie ist die Zeitkomplexität abhängig vom gewählten Modell. Das vermutete Problem, welches Berechnungsmodell genutzt wird für die Analyse der

Samuel Domiks
Pro-Seminar Theoretische Informatik
bei Prof. Dr. Helmut Alt

18.11.2014

Zeitkomplexität eines Algorithmus tritt allerdings nicht auf, da für alle typischen deterministischen Modelle die Zeitkomplexität sich im Wesentlichen nicht unterscheidet.

Komplexitätsbeziehung zwischen den Modellen

Wir betrachten nun den Unterschied der Komplexität zwischen 3 Modellen, der ein-Band Turingmaschine, der k-Band Turingmaschine sowie der nichtdeterministisch Turingmaschine.

Let $t(n)$ be a function, where $t(n) \geq n$. Then every $t(n)$ time multitape Turing machine has an equivalent $O(t^2(n))$ time single-tape Turing machine.

[5]

Wie kommen wir nun auf $O(t^2(n))$? Wenn ein k-Band TM z.B. in $t(n)$ Zeit läuft, konstruieren wir einfach eine ein-Band TM die in $O(t^2(n))$ liegt.. Wir beginnen damit, die k – Band TM in eine ein-Band TM umzuwandeln, indem wir die k-Bänder der k-Band TM hintereinander schreiben. Außerdem merken wir uns die Kopfposition aller k-Bänder. Nun haben wir eine ein-Band TM, die pro Operation der k-Band TM, 2 mal durch geht, einmal um die Kopfposition aus zu lesen, und danach um die Operation auszuführen. Nun müssen wir noch das Problem lösen , dass wenn auf eins der k-Bänder etwas neues geschrieben wird. Dazu werde nach jedem Band der k-Band TM die wir auf unsere ein-Band TM schreiben, ein # eingefügt. Wenn nun darauf geschrieben werden soll, wird das komplette rechte Band um eins nach rechts verschoben, sodass dort wieder Platz ist. Nun analysieren wir das ganze. Um durch die k-Bänder in der ein-Band TM durch zugehen ist eine Zeit von $O(t(n))$ notwendig. Nun werden die t(n) Operationen ausgeführt, wofür die ein-Band TM für jede Operation jeweils – $O(t(n))$ Zeit benötigt, da bei jeder Operation nach rechts geschoben werden kann, was uns O(n) Zeit kostet. Somit liegt die ein-Band TM in $O(t^2(n))$.

Nun analysieren wir die nichtdeterministische TM im Vergleich zu einer ein-Band TM.

Let $t(n)$ be a function, where $t(n) \geq n$. Then every $t(n)$ time nondeterministic single-tape Turing machine has an equivalent $2^{O(t(n))}$ time deterministic single-tape Turing machine.

[6]

Um wieder nachvollziehen zu können, wie wir auf die Laufzeit von $2^{O(t(n))}$ kommen, müssen wir uns angucken wie die Umwandlung stattfindet. Wir können eine nicht deterministische TM, die Probleme entscheidet, in eine äquivalente k-Band TM umwandeln. Genauer gesagt in eine 3-Band TM, wobei das 1te Band die Eingabe der nichtdeterministischen TM simuliert, welches nicht verändert werden kann. Das 2te Band beinhaltet die Eingabe, wo die Operation der nicht deterministischen TM simuliert werden, und das letzte Band beinhaltet alle Entscheidungszweige der nicht deterministischen TM die zu einem Blatt im Entscheidungsbaum führen(D.h der String 1,2,3 steht für, das 1 Kind wird genommen, dann das 2 und als letztes das dritte). Nun werden die Operation auf Band 2 simuliert, und es werden die Entscheidung ausgewählt die im dritten Band als Entscheidungen im Entscheidungsbaum stehen. Wenn das fertig ist, wird der nächste lexikographische String genommen, bis ein akzeptierender Endzustand getroffen wird. Diese Möglichkeit benutzt die Idee der Breitensuche, damit der erst mögliche akzeptierende Zustand getroffen wird. Nun müssen wir diese k-Band TM noch analysieren. Wir wissen das jeder Zweig die maximale Länge $t(n)$, bei einer Eingabe der Länge n, haben kann. Jeder Knoten kann maximal b Kinder haben. Damit ist die maximal Anzahl der Blätter die der Baum haben kann $b^{t(n)}$. Bevor wir eine Stufe tiefer im Baum gehen, werden vorher alle Knoten der Stufe durchgegangen. Die Gesamtanzahl der Knoten im Baum beträgt $2*b^{t(n)}$, also $O(b^{t(n)})$. Um nun von der Wurzel zu einem Knoten zu kommen brauchen wir $O(t(n))$. Daraus folgt das das die k-Band TM in $O(t(n)b^{t(n)}) = 2^{O(t(n))}$ liegt. Nun wandeln wir die k-Band TM in eine ein-Band TM um, allerdings ändert sich die asymptotische Laufzeit nicht, da im Exponent nur ein konstanter Faktor dazukommen würde, nämlich 2.

Wie man an den beiden obigen beiden Beispielen sieht, gibt es dort einen großen Unterschied in der Laufzeit. So haben die ein – Band und k-Band TM einen polynomiellen Unterschied in der Laufzeit, die nichtdeterministischen im Gegensatz dazu einen exponentiellen Unterschied zu den deterministischen Modellen. Das führt uns nun zu P.

Samuel Domiks
Pro-Seminar Theoretische Informatik
bei Prof. Dr. Helmut Alt

18.11.2014

Komplexitätsklasse P

Für unsere Zwecke unterscheiden wir nicht zwischen unterschiedlichen polynomiellen Funktionen, sondern nur ob sie polynomiell oder exponentiell wachsen. Für uns macht es also keinen Unterschied ob sie in n^c , wobei $c \in \mathbb{N}$, bei c = 10 oder c = 10000 liegt, allerdings wenn es in c^n liegt. Exponentielle Laufzeiten entstehen meistens wenn wir durch einen Raum von Lösungen gehen müssen, sog. brute-force search. Alle vertretbaren deterministischen Berechnungsmodelle sind polynomiell äquivalent, was bedeutet das jeder dieser Berechnungsmodelle die anderen in nur einem polynomiellen Wachstum an Zeit simulieren kann. Nun können wir polynomielle Unterschiede ignorieren, da wir uns auf Aspekte der Zeitkomplexität konzentrieren die nicht von polynomiellen Unterschiede der Laufzeit beeinträchtigt werden. Damit können wir nun auch von den konkreten Berechnungsmodell trennen.

Definition 2.1

Let $t: \mathcal{N} \longrightarrow \mathcal{R}^+$ be a function. Define the ***time complexity class***, **TIME($t(n)$)**, to be the collection of all languages that are decidable by an $O(t(n))$ time Turing machine.

[7]

Definition 2.2

P is the class of languages that are decidable in polynomial time on a deterministic single-tape Turing machine. In other words,

$$\mathrm{P} = \bigcup_k \mathrm{TIME}(n^k).$$

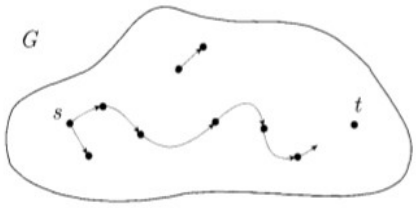
[8]

Die Klasse P ist deshalb wichtig, da sie stellvertretend für alle Berechenbarkeitsmodelle steht, die polynomiell äquivalent zu deterministischen ein-.Band TM sind und P bezieht sich auf alle Probleme die realistisch an einem Computer lösbar sind. Ob nun der Algorithmus wirklich praktisch einsetzbar ist, hängt von der Konstante c ab. Allerdings hat man herausgefunden, dass wenn man mal einen Algorithmus in polynomieller Zeit lösen kann, es Optimierung gibt, die die Konstante c möglichst klein werden lassen, da eine Laufzeit von $O(n^{10000})$ nicht wirklich praktikabel ist.

Beispiele zu P

1)
Wir wollen prüfen ob es in einem Graph einen Pfad von s nach t gibt.

$PATH = \{\langle G, s, t\rangle \mid G$ is a directed graph that has a directed path from s to $t\}$.



[9]

M = "On input $\langle G, s, t\rangle$ where G is a directed graph with nodes s and t:

1. Place a mark on node s.
2. Repeat the following until no additional nodes are marked:
3. Scan all the edges of G. If an edge (a, b) is found going from a marked node a to an unmarked node b, mark node b.
4. If t is marked, *accept*. Otherwise, *reject*."

[10]

Die einfachste Möglichkeit wäre eine brute-force search, allerdings gib es m^m ; $m = |Knoten|$, gibt. Das ist ungünstig, der oben stehende Algorithmus allerdings benutzt eine Tiefen-suche. Er markiert alle unmarkierten Knoten, bei denen es eine Kante von einem markierten Knoten zu einem nicht markierten Knoten gibt. Sobald s und t markiert sind, gibt es einen Pfad. Nun analysieren wir den Algorithmus. Stufe 1 und 4 brauchen jeweils nur einen Aufruf. Stufe 3 wird allerdings m mal aufgerufen, da pro Aufruf höchstens ein Knoten markiert wird, und es m Knoten gibt. Daraus folgt: $PATH \in P$
Diesen Algorithmus kann man auch mithilfe einer k-Band TM simulieren, werden wir allerdings nicht beachten, da diese ebenfalls in polynomieller Zeit ausführbar ist.

Samuel Domiks
Pro-Seminar Theoretische Informatik
bei Prof. Dr. Helmut Alt

18.11.2014

2)

Wir wollen herausfinden ob 2 Zahlen teilerfremd sind.

E = "On input $\langle x, y \rangle$, where x and y are natural numbers in binary:

1. Repeat until $y = 0$:
2. Assign $x \leftarrow x \bmod y$.
3. Exchange x and y.
4. Output x."

Algorithm R solves *RELPRIME*, using E as a subroutine.

R = "On input $\langle x, y \rangle$, where x and y are natural numbers in binary:

1. Run E on $\langle x, y \rangle$.
2. If the result is 1, *accept*. Otherwise, *reject*."

[11]

Diesmal benutzen wir den euklidischen Algorithmus um den GGT zweier Zahlen zu bestimmen, wenn dieser 1 ist, dann sind die beiden Zahlen teilerfremd, ansonsten nicht . Der euklidische Algorithmus teilt die beiden Zahlen durcheinander, und schaut sich den Rest an.Dies wird solange wiederholt bis der Rest einer Zahl 0 ist.

Bei diesem Algorithmus müssen wir nur E betrachten, denn wenn E in polynomieller Zeit läuft, so tut dies auch R. Nun betrachten wir was E mit dem Wert macht. Bei jeder Ausführung der 2ten Stufe, wird der Wert von x mindesten durch die Hälfte geteilt(außer eventuell bei der ersten Operation). Nun werden die Werte getauscht und das neue x, wird halbiert, so werden abwechselnd y und x halbiert. Daraus folgt das die max. Ausführung der Stufen 2 und 3 der kleiner von den beiden Werten $\log_2(x)$ oder $\log_2(y)$ ist. Dabei ist der Logarithmus proportional zur Länge der Darstellung, somit ist die Zahl der Ausführungen $O(n)$. Da jede Stufe nur polynomielle Zeit benötigt ist E, somit auch R, in polynomieller Zeit ausführbar

3)
Jede kontextfreie Sprache ist Mitglied von P

Dazu benutzen wir den CYK – Algorithmus .

D = "On input $w = w_1 \cdots w_n$:

1. If $w = \varepsilon$ and $S \rightarrow \varepsilon$ is a rule, *accept*. [handle $w = \varepsilon$ case]
2. For $i = 1$ to n: [examine each substring of length 1]
3. For each variable A:
4. Test whether $A \rightarrow \mathrm{b}$ is a rule, where $\mathrm{b} = w_i$.
5. If so, place A in $table(i, i)$.
6. For $l = 2$ to n: [l is the length of the substring]
7. For $i = 1$ to $n - l + 1$: [i is the start position of the substring]
8. Let $j = i + l - 1$, [j is the end position of the substring]
9. For $k = i$ to $j - 1$: [k is the split position]
10. For each rule $A \rightarrow BC$:
11. If $table(i, k)$ contains B and $table(k + 1, j)$ contains C, put A in $table(i, j)$.
12. If S is in $table(1, n)$, *accept*. Otherwise, *reject*."

[12]

Der Algorithmus entscheidet, ob ein Wort in der angegeben Grammatik(in CNF) ist oder nicht. Dabei werden Teilwörter gebildet und jeweils geprüft ob sich diese ableiten lassen.

Nun zur Analyse. Stufen 4 und 5 laufen $n*v$ -mal, wobei n die Anzahl der Variablen in G ist und eine fixe Konstante unabhängig von n. Daraus folgt $O(n)$. Stufe 6 läuft n-mal, wobei bei jedem Ausführen der Stufe 6 Stufe 7 n - mal läuft. Für jedes Ausführen der Stufe 7 laufen Stufe 9 n – mal, und für jedes Ausführen der Stufe 9 läuft Stufe 10 r-mal, wobei die Anzahl der Regeln darstellt und ebenfalls eine konstante ist. Daraus folgt dass der Algorithmus in $O(n^3)$ liegt.
Somit ist jede kontextfreie Sprache Mitglied von P, da diese in polynomieller Zeit entschieden werden kann.

Quellen:

[1] – [12] . Kapitel 7.1 -7.2
M. Sipser
Introduction to the Theory of Computation PWS Publ.Comp. 1997