Seminararbeit zum Thema Algorithmentheorie

Harald Schweiger

26. Mai 2014

Inhaltsverzeichnis

1 Einführung

Algorithmentheorie ist ein weitreichendes Gebiet. Es besteht u. a. aus Sortier- und Graph-Algorithmen, Komplexitätstheorie, Datenstrukturen, Problemlösungsmethoden, Optimierungsproblemen und Entwurfsmuster für Algorithmen. In dieser Seminararbeit wird jeweils eine kurze Einführung in die drei zuletzt genannten Punkte gegeben. Für genauere Informationen können die Literatur-Quellen zurate gezogen werden.

2 Teile und Herrsche

Teilen und Herrschen (englisch *divide and conquer*) ist ein Entwurfsmuster für Algorithmen und besteht aus 3 Phasen. Es beschreibt die Idee, Probleme in kleinere Teilprobleme rekursiv aufzuteilen (**Divide**), wodurch sie lösbar werden (**Conquer**). Das Hauptproblem wird durch das Kombinieren der Teilprobleme gelöst (**Combine**). (vgl. [1, Seite 65])

Ein bekannter Anwendungsfall dieses Entwurfsmusters ist der Mergesort.

2.1 Mergesort

Der Mergesort ist ein Algorithmus, der ein Array der Größe n mit einer durchschnittlichen Komplexität $\mathcal{O}(n \log(n))$ sortiert. Das Array wird hierbei solange rekursiv halbiert, bis die Teil-Arrays maximal aus einem Element bestehen. Die Größe der Elemente dieser Arrays werden vom Anfang bis zum Ende miteinander verglichen und kombiniert. Am Ende ist das ursprüngliche Array sortiert. (vgl. [1, Seite 30-37])

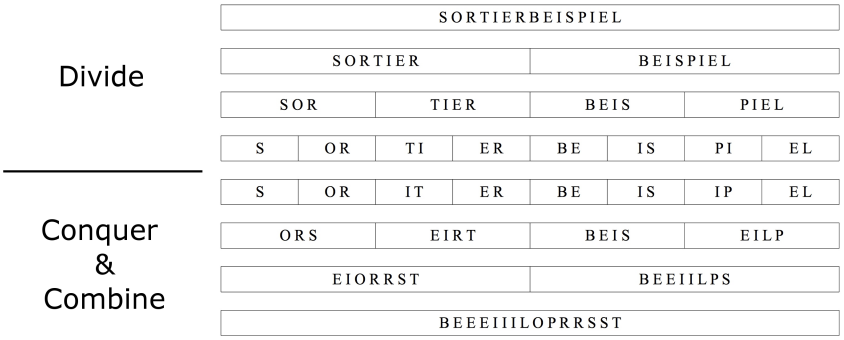

Abbildung 1: Die Buchstaben des Wortes SORTIERBEISPIEL werden mittels Mergesort sortiert.

3 Rücksetzverfahren

Das Rücksetzverfahren (englisch *Backtracking*) basiert auf dem *Versuch und Irrtum* Prinzip (englisch *Trial and Error*). Während der Ausführung des Programms trifft der Algorithmus Entscheidungen, die zur Problemlösung führen können. Stellt sich diese Entscheidung in den nächsten Schritten

als falsch heraus, geht man zum Ort der Entscheidung zurück und trifft stattdessen eine andere. (vgl. [4, Seite 847])

Für diese Problemlösungsmethode gibt es mehrere Anwendungen, wie z. B. das 8 Damenproblem oder der DPLL-Algorithmus. (vgl. [4, Seite 860])

3.1 Classical DPLL

Der DPLL, benannt nach den in der Entwicklung beteiligten Personen *Davis, Putnam, Logemann* und *Loveland*, berechnet die Erfüllbarkeit von konjunktiven Normalformen. Sprich die Literale müssen so belegt werden, dass alle Disjunktionsterme wahr in ihrer Aussage sind. Diese Belegungen werden auf der rechten Seite der $\|$ vermerkt. Bewahrt nur mehr ein Literal den Disjunktionsterm in seiner Aussage falsch zu werden, muss das Literal in seiner Belegung wahr werden (**UnitPropagate**). Lässt sich UnitPropagate nicht anwenden, entscheiden wir über eine Belegung beliebig (**Decide**). Wird im weiteren Verlauf ein Disjunktionsterm falsch in seiner Aussage (**Fail**), müssen wir unsere zuletzt getroffene Entscheidung negieren und von diesem Punkt aus erneut starten (**Backtrack**). (vgl. [2])

$$
\begin{array}{rcll}
\emptyset & \| & \neg 1 \vee \neg 2, 2 \vee 3, \neg 1 \vee \neg 3 \vee 4, 2 \vee \neg 3 \vee \neg 4, 1 \vee 4 \implies_{Cl} & \text{(Decide)} \\
1^d & \| & \neg 1 \vee \neg 2, 2 \vee 3, \neg 1 \vee \neg 3 \vee 4, 2 \vee \neg 3 \vee \neg 4, 1 \vee 4 \implies_{Cl} & \text{(U.P)} \\
1^d\ \neg 2 & \| & \neg 1 \vee \neg 2, 2 \vee 3, \neg 1 \vee \neg 3 \vee 4, 2 \vee \neg 3 \vee \neg 4, 1 \vee 4 \implies_{Cl} & \text{(U.P)} \\
1^d\ \neg 2\ 3 & \| & \neg 1 \vee \neg 2, 2 \vee 3, \neg 1 \vee \neg 3 \vee 4, 2 \vee \neg 3 \vee \neg 4, 1 \vee 4 \implies_{Cl} & \text{(U.P)} \\
1^d\ \neg 2\ 3\ 4 & \| & \neg 1 \vee \neg 2, 2 \vee 3, \neg 1 \vee \neg 3 \vee 4, 2 \vee \neg 3 \vee \neg 4, 1 \vee 4 \implies_{Cl} & \text{(Backtrack)} \\
\neg 1 & \| & \neg 1 \vee \neg 2, 2 \vee 3, \neg 1 \vee \neg 3 \vee 4, 2 \vee \neg 3 \vee \neg 4, 1 \vee 4 \implies_{Cl} & \text{(U.P)} \\
\neg 1\ 4 & \| & \neg 1 \vee \neg 2, 2 \vee 3, \neg 1 \vee \neg 3 \vee 4, 2 \vee \neg 3 \vee \neg 4, 1 \vee 4 \implies_{Cl} & \text{(Decide)} \\
\neg 1\ 4 \neg 3^d & \| & \neg 1 \vee \neg 2, 2 \vee 3, \neg 1 \vee \neg 3 \vee 4, 2 \vee \neg 3 \vee \neg 4, 1 \vee 4 \implies_{Cl} & \text{(U.P)} \\
\neg 1\ 4 \neg 3^d\ 2 & \| & \neg 1 \vee \neg 2, 2 \vee 3, \neg 1 \vee \neg 3 \vee 4, 2 \vee \neg 3 \vee \neg 4, 1 \vee 4 \implies_{Cl} &
\end{array}
$$

Abbildung 2: Durch DPLL erhalten wir ein Belegung, die diese konjunktive Normalform erfüllt.

4 Dynamische Programmierung

Normalerweise wenden wir Dynamische Programmierung für *Optimierungsprobleme*[1] an. Dynamische Programmierung löst, wie *divide and conquer* 2, Probleme durch ihre Teilprobleme. Jedoch wird Rechenarbeit erspart, indem schon einmal berechnete Teillösungen für andere Teilprobleme wiederverwendet werden. (vgl. [1, Seite 359])

[1] Def.: Das Finden der besten Lösung von allen Lösungskandiaten

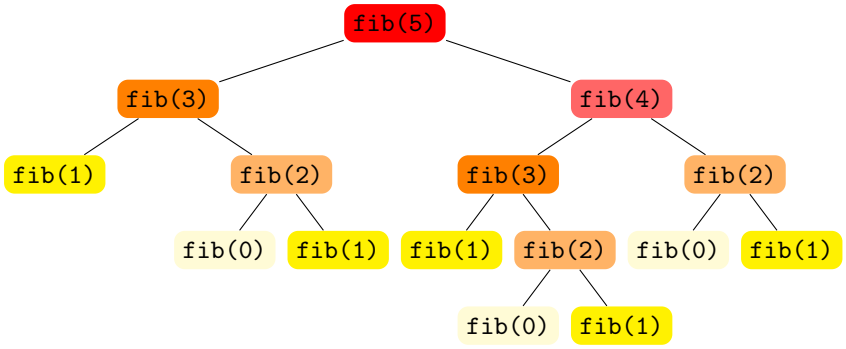

Abbildung 3: Veranschaulichung der sich wiederholenden rekursiven Aufrufe für die Berechnung der 5ten Fibonacci-Zahl.

4.1 Fibonacci-Folge

Die Berechnung der n-ten Fibonacci-Zahl ist eines der Beispiele, bei dem dynamisches Programmieren die Effizienz sehr verbessert. Die n-te Fibonacci-Zahl wird mit $f(n) = f(n-1)+f(n-2)$ für alle $n > 1$ berechnet. Andernfalls ist $f(n) = 1$. Eine genauere Einführung finden sie hier [3, Seite 92].

5 Schlusswort

Dieses Dokument gibt einen kurzen Einblick in drei Themengebiete der Algorithmentheorie. Zum besseren Verständnis wurden jeweils praktische Beispiele mit Abbildungen hinzugefügt.

Literatur

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. Cambridge, Mass. [u.a.]: MIT Press, 3 edition, 2009.

[2] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving sat and sat modulo theories. *Journal of the ACM (JACM)*, 53:937–977, November 2006.

[3] Christian Sternagel and Harald Zankl. Functional programming. University of Innsbruck, August 2013.

[4] Jürgen Wolf. *C von A bis Z : das umfassende Handbuch*. Bonn: Galileo Press, 3., aktual. u. erw. aufl. edition, 2011.