Ein Kongruenz-Bitmatch-Problem

Thorsten Reinecke*

7. August 2005

Vorbemerkungen

Ich will kurz erläutern, wie ich auf das Bitmatch-Problem gestoßen bin: Der Faktorisierungsalgorithmus von Fermat kann durch Restklassenbetrachtungen beschleunigt werden. Dies war schon Fermat selbst bekannt.

Bei dem Versuch, diese Restklassenbetrachtungen am Telefon anschaulich zu erläutern, habe ich das Bitmatch-Problem formuliert. Es handelt sich um ein eigenständiges, verallgemeinertes Problem, das man völlig losgelöst vom Fermat-Algorithmus betrachten kann.

Derartige Kongruenzprobleme löst man üblicherweise mit Siebverfahren. Eine kurze Internetrecherche hat ergeben, dass bereits ab 1896 diverse Gerätschaften zur Lösung solcher Probleme entwickelt wurden, die allerdings alle darauf basieren, den Suchraum systematisch abzusieben (z.B. die *Machine á Congruences* von *Eugène Olivier Carissan, 1919*).

Es ist also sehr wahrscheinlich, dass das Problem in der Literatur unter anderem Namen bereits eingehend untersucht wurde. Für jeden Hinweis, der mir bei einer Komplexitätsabschätzung und/oder Lösung des Problems hilft, wäre ich dankbar. Wer kann mir Literaturverweise für dieses Problem angeben?

Mir jedenfalls ist **kein** effizienter Algorithmus zur Lösung des nachfolgend geschilderten Bitmatch-Problems bekannt...

Das Bitmatch-Problem

Es seien n Räder $R_1, R_2, R_3, ..., R_n$ gegeben. Jedes dieser Räder sei in Sektoren unterteilt. Jeder Sektor ist gefärbt, wobei genau zwei Farben, z.B. schwarz und rot zur Verfügung stehen. Die Anzahl der Sektoren des Rades R_i sei durch s_i bezeichnet. Die s_i seien paarweise teilerfremd. Jedes Rad sei so eingefärbt, dass beide Farben auf jedem Rad in etwa gleichhäufig vorkommen.

Die Räder seien durch eine Vorrichtung so miteinander verbunden, dass sie jeweils simultan um ein Feld vorwärts gedreht werden können. Eine vorgegebene Ausgangsstellung der Räder sei markiert.

Aufgabenstellung: Entwicklung eines Algorithmus, um die nächste *(und nicht etwa nur irgendeine!)* Stellung zu finden, an der alle Radpositionen rot eingefärbt sind.

Eingabe für den Algorithmus: Anzahl der Räder; die Färbungsinformation sowie die Ausgangsstellung der Räder.

Ausgabe des Algorithmus: Lösungsstellung der Räder und/oder Anzahl der notwendigen Schritte, um von der Ausgangsstellung zur Lösungsstellung zu gelangen.

*thre@thorstenreinecke.de

Beispiel

Drei Räder seien hier als Zeichenketten dargestellt. Eine Drehung der Räder entspricht dem Verschieben der Zeichenkette nach links, wobei rausgeschobene Zeichen rechts wieder angehängt werden. Wir betrachten jeweils das erste Zeichen der Zeichenketten. (Dies ist nur eine von vielen möglichen Kodierungen.)

R_1= **r**ss R_2= **s**srrs R_3= **r**srrsrs *(Ausgangsstellung)*
R_1= **s**sr R_2= **s**rrss R_3= **s**rrsrsr *(nach 1 Schritt)*
R_1= **s**rs R_2= **r**rsss R_3= **r**rsrsrs *(nach 2 Schritten)*
R_1= **r**ss R_2= **r**sssr R_3= **r**srsrsr *(nach 3 Schritten)*

Alle drei Räder stehen nach drei Schritten auf rot. Je nach Ausgangsstellung und Einfärbung erhalten wir natürlich unterschiedliche Ergebnisse. Je mehr Räder wir verwenden, desto mehr Schritte sind üblicherweise erforderlich, um alle Räder auf die Stellung rot zu bringen. Unter den oben gemachten Bedingungen verdoppelt sich die zu erwartende Schrittzahl in etwa mit jedem neu hinzugefügten Rad.

Wer sich weitere Beispiele am Computer anzeigen lassen möchte, für den steht ein kleines Demonstrationsprogramm in C auf meiner Homepage zur Verfügung.

Lösungsansätze

Ein simpler Ansatz

Es ist klar, dass man durch fortschreitendes Ausprobieren der Stellungen schließlich auf die Lösung kommt. Wenn die Anzahl der Räder klein ist, mag dies eine effiziente Methode sein.

Bereits für $n > 50$ erscheint diese Vorgehensweise jedoch als unpraktikabel. (Wenn man die Wahrscheinlichkeit, auf einem einzelnen Rad im nächsten Schritt ein rotes Feld zu erwischen, mit 50% kalkuliert und die Einzelwahrscheinlichkeiten als unabhängig annimmt, dann ist die Wahrscheinlichkeit, dass im nachfolgenden Schritt alle Felder rot markiert sind, 2^{-n}. Man kann daher einen Erfolg erst nach etwa 2^{n-1} Schritten "erwarten".)

Hackertricks

Moderne Prozessoren haben eine Registerbreite von 64 (und sogar 128 Bits). Dies bietet genügend Spielraum für eine Parallelverarbeitung der Felder. Mit SSE-Registern lassen sich 128 Felder auf einmal betrachten. Eine komponentenweise Und-Verknüpfung von 128 aufeinandfolgender Positionen der einzelnen Räder *(rot=1, schwarz=0)* macht den simplen Ansatz 128-mal schneller (benötigt aber entsprechend mehr Platzbedarf)!

Kombinatorische Ansätze

Man kann kleinere Räder miteinander kombinieren und daraus neue Räder erzeugen. Vermutlich lässt sich aber nicht vermeiden, dass diese "Kombinationsräder" sehr groß werden (Kleinstes gemeinsames Vielfaches der kombinierten s_i). Die "Kombinationsräder" hätten dafür aber auch erheblich weniger rote als schwarze Felder... (Abstandsbetrachtungen zwischen den roten Feldern werden nun sinnvoll!)

Ein zahlentheoretischer Ansatz

Man kann alle denkbaren Lösungspositionen mit dem Chinesischen Restsatz bestimmen und die kleinste Lösung anschließend auswählen. Dieser Ansatz ist jedoch nur dann praktikabel, wenn die Anzahl der roten Felder klein ist. Nach der gegebenen Problemstellung (und bei großem n) ist dies leider nicht der Fall.

Komplexitätstheoretische Fragen

- Liegt das Problem in **P**?

- Liegt es in **NP**? (Wir verwenden die modifizierte Fragestellung, ob die Schrittzahl eine gegebene Schranke unterschreitet. Dieses modifizierte Problem liegt ziemlich sicher in **NP**, [Skizze: Die Lösungsstellung kann geraten werden. Mit dem Chinesischen Restsatz wird anschließend die Schrittzahl errechnet und überprüft, ob sie unterhalb der vorgegebenen Grenze liegt.] Für die Original-Fragestellung gilt: Wie kann man beim Erraten einer denkbaren Lösungsposition den Nachweis führen, dass es sich auch um die kleinste Lösung handelt? [1])

- Vorige Fragestellung führt unmittelbar auf die Frage, ob sich das (modifizierte) Problem in P ∩coNP befindet. Gibt es einen NP-Algorithmus, der auch die Frage, ob die Schrittzahl (der minimalen Lösungsstellung!) eine vorgegebene Schranke nicht unterschreitet, akzeptiert?

- Ist das Problem **NP-vollständig**? (Untersuchung unter der modifizierten Fragestellung ist akzeptabel.)

- Wie steht es mit dem Average Case? Das ist die eigentlich interessante Fragestellung!

- Kann man eine Fallunterscheidung zwischen einfachen und schweren Fällen treffen?

- Ideen für eine Relaxiation, die sich anschließend auf das Problem zurückübertragen lässt?

- Gibt es einen Algorithmus, der mit einer geringen Irrtumswahrscheinlichkeit die nächstliegende Lösung in praktikabler Zeit präsentiert?

- Gibt es einen Algorithmus, der eine brauchbare untere/obere Schranke für die Schrittzahl annähern kann?

weitere Hinweise

Ein Spezialfall dieses Bitmatch-Problems tritt bei dem Versuch auf, den Fermat-Algorithmus zur Faktorisierung natürlicher Zahlen zu optimieren. Nachfolgend seien die Grundgedanken hierfür skizziert:

Gelte $N = p \cdot q$; eine Untersuchung der Eigenschaften von $p+q$ ergibt $p+q \equiv \Delta \pmod{t}$ $\Rightarrow p \equiv \frac{\Delta}{2} \pm \sqrt{(\frac{\Delta}{2})^2 - N} \pmod{t}$. Für alle Teiler t von $p+q-\Delta$ muss der Term $(\frac{\Delta}{2})^2 - N$ also ein quadratischer Rest modulo t sein. Insbesondere gilt dies für $\Delta = p+q$, da die 0 durch alle positiven natürlichen Zahlen ohne Rest teilbar ist. Für ein fixes N können wir daher z.B. eine Färbung der Räder unter Zuhilfenahme des Legendresymbols vornehmen, sofern wir für die s_i ungerade Primzahlen wählen: Die schwarzen Felder eines Rades R_i (mit $\Delta \in \{0, .., s_i - 1\}$) besitzen die Eigenschaft $\left(\frac{(\frac{\Delta}{2} \bmod s_i)^2 - N}{s_i}\right) = -1$; die übrigen Felder sind rot. Auch eine Startposition der Räder lässt sich vorgeben, z.B. $\Delta = \left\lfloor \sqrt{4N} \right\rfloor$.

Folgende Fälle wären interessant:

Falls ein effizienter Algorithmus für das Bitmatch-Problem existiert: Setze die Anzahl n der Räder so hoch an, dass die Irrtumswahrscheinlichkeit, dass die Lösung des Bitmatch-Problems nicht die Stellung für $p+q$ anzeigt, vernachlässigbar klein ist. Damit wäre ein probabilistischer Faktorisierungsalgorithmus für N gefunden. (Denn der Vorgang lässt sich einerseits sukzessive mit der "falschen" Endstellung als neuer Ausgangsstellung wiederholen; andererseits wäre auch eine beliebige Wahl neuer Räder möglich.)

[1] Man achte auf die Feinheiten! Die Lösungsstellung kann zwar geraten werden, aber in der Beweisskizze wird nicht verifiziert, ob es sich wirklich um die Lösungsstellung handelt. Es wird lediglich die schwächere Bedingung überprüft, ob eine gültige Kombination vorliegt, die die gegebene Schranke tatsächlich unterschreitet.

Eine Schachtelung von Schranken ist bei Nichtdeterminismus nicht ohne weiteres erlaubt, da man innerhalb der Schachtelung den Fall des nichtdeterministischen Nicht-Akzeptierens per Konstruktion ausschließen muß.

Für den (sehr unwahrscheinlichen) Fall, dass jemand nachweisen könnte, dass das Originalproblem nicht in NP liegt (während das modifizierte in NP liegt), wäre auch P≠NP nachgewiesen, da man für P=NP die Intervallschachtelung des mozifizierten Problems in P durchführen könnte.

Falls kein effizienter Algorithmus für das Bitmatch-Problem existiert: Verwende Heuristiken für das Bitmatch-Problem, um sukzessive neue Kandidaten für $p + q$ zu generieren. Hier ist definitiv eine erhebliche Beschleunigung des Fermat-Algorithmus möglich (und wird in diversen Implementierungen auch durchgeführt)! Ein wesentlicher Vorteil liegt darin, dass bis auf einige Initialisierungen und einige seltene Kandidatentests komplett mit herkömmlicher Integer-Arithmetik auf kleinen Restklassen gearbeitet werden kann, sofern die Bitmatch-Heuristik ebenfalls mit herkömmlicher Integer-Arithmetik auskommt.