Sommerakademie 2010 Rot an der Rot AG 1:

"Wieviel Platz brauchen Algorithmen wirklich?"

Nichtdeterministische Platzklassen

Ulf Kulau

August 23, 2010

Contents

1 Einführung

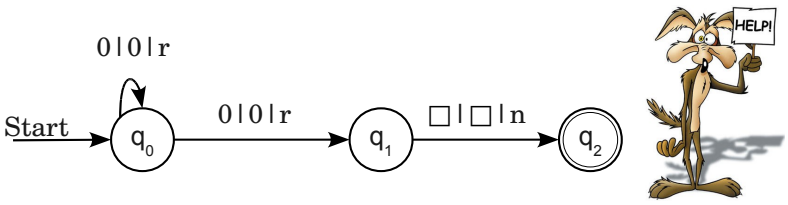
In einer Zeit, in der ein Gigabyte Speicher rund 7 Cent kostet[1], scheint das Nachdenken über logarithmischen Platzverbrauch reichlich überzogen[2]. Die Betrachtung nichtdeterministischer Platzklassen ist jedoch nicht praktischer Natur, da man nichtdeterministische Maschinen eh nicht wirklich bauen kann. Trotzdem sind diese theoretischen Konstrukte hilfreich, da die Laufzeiten und Platzanforderungen von nichtdeterministischen Maschinen deutlich geringer sind und wichtige Hinweise auf effizientere Lösungen vieler praktischer Probleme liefern[3]. Die eigentliche Kernfrage dieses Vortrags lautet damit: '*Wie viel Platz benötigt eine nichtdeterministische Maschine für ihre Berechnungen?*'

2 Nichtdeterminismus allgemein

Allgemein könnte man den Nichtdeterminismus beschreiben durch '*Maschinen, die sich nicht entscheiden können*'[2]. Während bei deterministischen Maschinen der einer Berechnung bei gegebener Eingabe klar definiert ist, kann es bei nichtdeterministischen Maschinen bei nur einem gelesenen Zeichen mehrere Nachfolgezustände geben[3].

2.1 Eine nichtdeterministische Maschine

Als kleines Beispiel soll eine nichtdeterministische Turing-Maschine (NTM) behandelt werden. Die nachfolgende Abbildung zeigt eine solche NTM [2]. Was macht diese Maschine, wenn sie sich im Zustand q_0 befindet und als Eingabezeichen eine 0 gelesen wird?



Diese Maschine ist im Zustand q_0 nichtdeterministisch. Angenommen, der oben beschriebene Fall tritt ein, dann könnte die Maschine folgende Berechnungen durchführen:

- in q_0 bleiben und nach rechts shiften
- nach q_1 gehen und ebenfalls nach rechts shiften

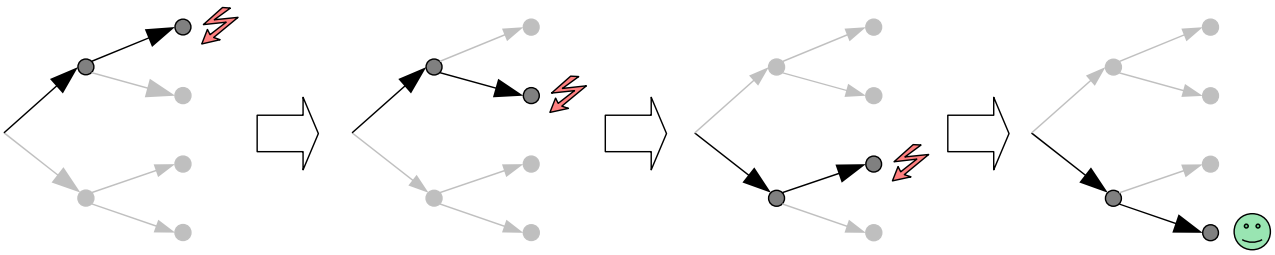
Im Gegensatz du deterministischen Maschinen gibt es also keine klar definierte 'Berechnung', sondern viele verschiedene Möglichkeiten der Berechnung, von denen mindestens eine in einem akzeptierenden Zustand (hier q_2) enden muss[2]. Mit anderen Worten akzeptiert eine NTM eine Eingabe, wenn es eine akzeptierende Berechnung gibt. Dadurch entsteht der Eindruck, dass eine nichtdeterministische Maschine ein Wort 'raten' kann[3]. Diese Eingenschaft kann genutzt werden und soll im folgendem Beispiel erläutert werden.

2.2 Deterministische und Nichtdeterministische Lösungsstrategien

Das Konzept nichtdeterminkistischer Lösungsstrategien baut auf der Fähigkeit der NTM des 'Ratens' auf. Die NTM dient sozusagen als Orakel und errät eine potenzielle Lösung, welche anschließend lediglich auf ihre Gültigkeit überprüft werden muss[3]. Bei deterministischen Lösungsansätzen muss

hingegen der Suchraum systematisch 'durchkämmt' werden (Brute-Force-Methode), was bei einem exponentiell mit der Eingabegröße wachsenden Suchraum nur für kleine Eingaben akzeptabel funktioniert. Die nachfolgende Grafik stellt deterministische und nichtdeterministische Lösungsstrategien anhand einer Graphensuche gegenüber[3]:

Deterministische Lösungsstrategie:
Der Suchraum wird systematisch durchkämmt (Brute-Force-Methode)



Erzeugen & Verifizieren | Erzeugen & Verifizieren | Erzeugen & Verifizieren | Erzeugen & Verifizieren

Nichtdeterministische Lösungsstrategie:
Der richtige Berechnungspfad wird geraten und die Lösung anschließend verifiziert (Orakelprinzip)

„Orakel“ | Raten & Verifizieren

[1] Das sich durch den Einsatz einer nichtdeterministischen Lösungsstrategie die Laufzeit erheblich verbessert wird deutlich, jedoch ist damit nicht die Frage geklärt, wie viel Platz bzw. Speicher eine nichtdeterministische Maschine für eine solche Berechnung benötigt.

3 Platzbedarf einer (nicht)deterministischen Maschine

Bei der Bestimmung des Platzbedarf einer (nicht)deterministischen Maschine (NTM/DTM) wird zunächst vereinbart, dass der Speicher für die Ein- bzw. Ausgabe keine Rolle spielt[2]. Betrachtet man die Gruppe der Entscheidungsprobleme, so existieren zwar exponentiell viele Möglichkeiten, wie ein Speichersegment belegt werden kann, aber jede einzelne dieser Belegungen kann in polynomiellen Platz gespeichert werden. Hieraus folgen die Komplexitätsklassen PSPACE für deterministische und NPSPACE für nichtdeterministische Maschinen (p beliebiges Polynom, n Eingabelänge)[3].

$$PSPACE = NPSPACE = SPACE(p(n)) \tag{1}$$

Die Beziehung PSPACE = NPSPACE lässt sich dadurch erklären, dass jede nichtdeterministische Maschine (NTM) unter erhöhtem Zeitaufwand von einer deterministischen Maschine (DTM) 'simuliert' werden kann, ohne dabei mehr Platz zu benötigen (Satz von Savitch)[3].

[1] `http://de.wikipedia.org/wiki/Paul_(Krake)`

3.1 Komplexitätsklasse NL

Übertreiben wird das Ganze noch ein wenig und fordern eine nichtdeterministische Maschine (NTM), welche ein gegebenes Entscheidungsproblem löst und für ihre Berechnungen lediglich logarithmischen Speicherplatz benötigen soll. Dies führt auf die Komplexitätsklasse NL[2]:

$$NL = SPACE(O(log(n))) \tag{2}$$

Im folgenden Beispiel soll geklärt werden, ob sich das gegebene REACH-Problem vollständig in NL befindet.

3.2 *REACH*: ein NL-vollständiges Problem

Ob es in einem gerichtetem Graphen G einen Weg von Startknoten s zum Zielknoten t gibt, ist ein klassisches Entscheidungsproblem und wird als $REACH$ oder GAP bezeichnet. Gibt es einen weg, so existiert auch eine akzeptierende Berechnung einer NTM[2].

$$REACH = \{code(G, s, t) | \text{es gibt einen Weg von s nach t in G}\} \tag{3}$$

Zunächst wird geprüft, ob die Komplexität des Erreichbarkeitsproblems lediglich logarithmisch viel Platz fordert, sprich ob $REACH \in NL$. Im zweiten Schritt wird dann geprüft, ob $REACH$ tatsächlich als 'schweres' Problem vollständig in NL liegt.

1. $REACH \in NL$
2. $REACH$ ist vollständig in NL

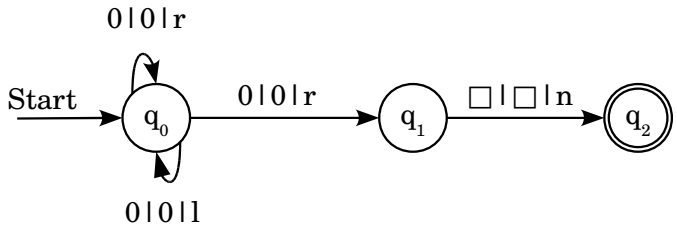
zu 1: Gegeben sei eine NTM und ein Graph G, welcher aus einer beliebigen Anzahl von Knoten $x_1, ..., x_n$, einem Startknoten s und einem Zielknoten t besteht. Die Knoten selbst sind dabei durch gerichtete Kanten miteinander verbunden. Die nichtdeterministische Turing-Maschine nutzt ihre 'Orakel-Eigenschaft'[3] und rät einen Weg von s nach t. Für ihre Berechnung spendieren wir der NTM zwei seprate 'Speichersegmente'[2], wobei der ein Speichersegment den Index des aktuell besuchten Knotens beinhaltet und das andere den Index des nichtdeterministisch geratenen Folgeknotens aufnimmt. In jedem Speicher steht also nur ein Zeichen, welches binär codiert $log_2(n)$ jeweils nur logarithmisch viel Platz in Anspruch nimmt.

$$\Longrightarrow REACH \in NL \tag{4}$$

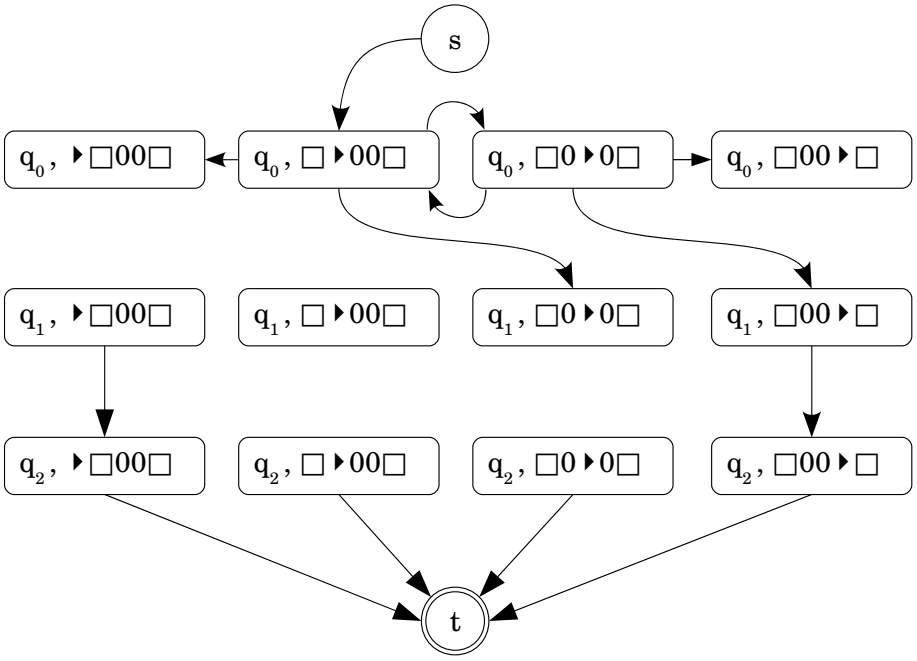
zu 2: Wie bereits bekannt ist, sind Probleme die in NL liegen auf logarithmischem Platz von einer nichtdeterminitischen Turing-Maschine lösbar. In 1) wurde gezeigt, dass $REACH \in NL$ liegt, jedoch muss für den Beweis der Vollständigkeit noch gezeigt werden, dass sich jede Sprache $A \in NL$ auf REACH reduzieren lässt[2]. Die Idee für einen Beweis basiert darauf, dass es für jede Sprache A eine NTM M gibt, welche auf logarithmischem Platz A akzeptiert. Man reduziert das Problem auf die Frage, ob bei einer beliebiegen Eingabe $x \in A$ gilt. Mit anderen Worten ist ein beliebiges Problem in NL zu finden, wenn es einen Weg vom Anfangszustand s zum akzeptierenden Zustand t im Konfigurationsgraphen der NTM M gibt[2].

Die Knoten eines Konfigurationsgraphen sind gegeben durch die Menge aller Konfigurationen der Turing-Maschine, welche durch die Eingabe x möglich ist. Eine Verbindung zwischen den Knoten, also die Kanten, existiert genau dann, wenn ein 'Übergang' zwischen zwei Konfigurationen besteht. Die Größe eines Konfigurationsgraphen ist polynomiell, da $NL \subseteq P$ gilt. Im nachfolgendem Beispiel wird ein Konfigurationsgraph einer NTM erstellt und ein Pfad von s nach t dargestellt[2]:

Nichtdeterministische Turing-Maschine M:



Konfigurationsgraph von M bei der Eingabe x=00 :



$$\Longrightarrow REACH \text{ ist vollständig in } NL \tag{5}$$

Platz für Notizen:

References

[1] *Online Shop 'Alternate'* `http://www.alternate.de/`, Zugriffsdatum: 31.07.2010

[2] *Vorlesungsskript Theoretische Informatik*, Wintersemesetr 2009, Prof. Till Tantau, Institut für Theoretische Informatik, Universität zu Lübeck

[3] *Theoretische Informatik*, Dirk. W. Hoffmann, Hanser Verlag, 2009

—