LogSpace

Weitere Probleme in LogSpace z.B.:

- Isomorphie von Bäumen (gerichtet und ungerichtet)
- Entscheiden, ob ein Graph zusammenhängend ist (gerichtet und ungerichtet)
- Pattern matching: gegeben Wort w und Pattern (Wort) p, entscheide ob p Teilwort von w ist
- Addition, Subtraktion, Multiplikation, Division von natürlichen Zahlen
- Auswertung von AL-Formeln (gegeben Formel und Belegung, erfüllt Belegung die Formel?)

Kapitel 4

Nicht-Determinismus und LogSpace

NLogSpace

Auch von LogSpace gibt es eine nicht-deterministische Variante:

$$\text{NLogSpace} := \text{NSpace}(\log(n))$$

Aus Savitch's Theorem folgt nicht, dass LogSpace = NLogSpace, nur

$$\text{NLogSpace} \subseteq \text{DSpace}(\log(n)^2)$$

In der Tat ist nicht bekannt, ob LogSpace = NLogSpace, man weiss nur:

Theorem

$\text{LogSpace} \subseteq \text{NLogSpace} \subseteq \text{P}$

Es macht also Sinn, NLogSpace-Härte und -Vollständigkeit zu betrachten

LogSpace Reduktionen

Polynomialzeit-Reduktionen sind hier nicht sinnvoll, da

für alle $L, L' \in$ NLOGSPACE mit L' nicht-trivial: $L \leq_p L'$

(*nicht-trivial*: es gibt positive Instanzen und negative Instanzen)

Definition LogSpace Reduktion

Reduktion f von L auf L' ist *LogSpace-Reduktion* wenn sie LogSpace-berechenbar ist. Wir schreiben $L \leq_{\log} L'$.

Beachte:

- $\leq_p$ bezieht sich immer auf Polynomialzeit
  (Polynomialplatz zu mächtig für Reduktionen)
- $\leq_{\log}$ bezieht sich immer auf Logplatz
  (Logzeit erlaubt nicht mal das Lesen der Eingabe)

LogSpace-Reduktionen

Lemma

1. "LogSpace-reduzierbar" ist transitive Relation:
   $L_1 \leq_{\log} L_2$ und $L_2 \leq_{\log} L_3$ impliziert $L_1 \leq_{\log} L_3$
2. NLOGSPACE, P, NP, PSPACE sind abgeschlossen unter LogSpace-Reduktionen: $L' \in \mathcal{C}$ und $L \leq_{\log} L'$ impliziert $L \in \mathcal{C}$;

Punkt 1 folgt aus: wenn f und g LogSpace-berechenbar, dann auch $f(g)$

Bemerkungen:

- die meisten Polyzeit-Reduktionen für NP-Vollständigkeit lassen sich auf LogSpace-Reduktionen verbessern
- $\leq_p = \leq_{\log}$ gdw. LOGSPACE = P

NLogSpace-Vollständigkeit

Definition NLogSpace-Härte, NLogSpace-Vollständigkeit

Problem L ist

- *NLogSpace-hart* wenn $L' \leq_{\log} L$ für alle $L' \in$ NLogSpace;
- *NLogSpace-vollständig* wenn L NLogSpace-hart und in NLogSpace.

Theorem

GAP ist NLOGSPACE-vollständig

Beweis zeigt sehr engen Zusammenhang zwischen NLogSpace und GAP, schon fast: NLogSpace ist GAP!

NLogSpace-Vollständigkeit

Bemerkungen:

- Die LogSpace vs. NLogSpace Frage ist: ist GAP für gerichtete Graphen in LogSpace?

- Omer Reingold hat 2004 in einem gefeierten Resultat gezeigt, dass GAP für *ungerichtete* Graphen in LogSpace ist

NLogSpace-Vollständigkeit

Weitere NLogSpace-vollständige Probleme z.B.:

- 2SAT
- Das Wortproblem für lineare Grammatiken
- Entscheiden, ob ein Graph durch 2 Cliquen überdeckt werden kann
- Entscheiden, ob ein gerichteter Graph stark zusammenhängend ist, ob also jeder Knoten von jedem Knoten erreichbar ist

Kapitel 5

co-NLogSpace

co-NLogSpace

Da NLogSpace nicht-deterministische Klasse, macht es Sinn,

co-NLogSpace zu betrachten

co-NLOGSPACE $:= \{\overline{L} \mid L \in$ NLOGSPACE$\}$

Es war lange Zeit unbekannt, ob NLogSpace = co-NLogSpace

Theorem von Immerman und Szelepcsényi

NLOGSPACE = co-NLOGSPACE

Das Resultat lautet sogar:
für alle $s \in \Omega(\log(n))$ gilt NSpace(s) = co-NSpace(s)

Immerman / Scelepcsenyi

Da GAP NLogSpace-vollständig, reicht es zu zeigen:

$$\overline{\text{GAP}} \in \text{NLogSpace}$$

Wir brauchen also eine LogSpace NTM für nicht-Erreichbarkeit in gerichteten Graphen!

Wir gehen in zwei Schritten vor:

1. NLogSpace Algorithmus für: gegeben G, u_0, v_0, c mit c die Anzahl der von u_0 aus erreichbaren Knoten, ist v_0 nicht erreichbar von u_0?
2. Zeigen, dass c in NLogSpace berechnet werden kann.

Immerman / Scelepcsenyi - Schritt 1

Überblick:

- M iteriert über alle Knoten v und rät für alle $v \neq v_0$, ob v von u_0 erreichbar ist
- Wenn v als erreichbar geraten wurde, überprüfe die Erreichbarkeit durch das sukzessive Raten eines Pfades (Länge max. $|V|$) von u_0 nach v
- Wenn das fehlschlägt, verwerfe (keine “false positives”)
- M zählt in binär die Anzahl x der als erreichbar geratenen Knoten
- Wenn $x = c$, akzeptiere, sonst verwerfe (keine “false negatives”)

Beachte: wenn M akzeptiert, dann $x = c$, also sind alle nicht als erreichbar geratenen Knoten (inkl. v_0) wirklich nicht erreichbar

Immerman / Scelepcsenyi - Schritt 2

Überblick:

- wir entwerfen LogSpace-NTM M, bei der eine Berechung den korrekten Wert für c ausrechnet und alle anderen verwerfen
- diese kann dann der vorigen NTM “vorgeschaltet” werden
- Sei $V_i \subseteq V$ die Menge der Knoten, die von u_0 aus in i Schritten erreicht werden und $|V_i| = c_i$ für $0 \leq i \leq |V|$
- Wir wollen c_m mit $m := |V|$; berechnen $c_0, \cdots, c_m$
- Berechnung von c_i: Iteriere über alle $v \in V$, bestimme ob $v \in V_i$, zähle binär
- Bestimmen ob $v \in V_i$ ähnlich Schritt 1: rate sukzessive alle Knoten in V_{i-1}, identifiziere “false positives” mit Erreichbarkeitstest und ”false negatives” durch Vergleich mit dem schon berechneten c_{i-1}
  Es bleibt zu prüfen, ob v von Knoten in V_{i+1} in einem Schritt erreichbar

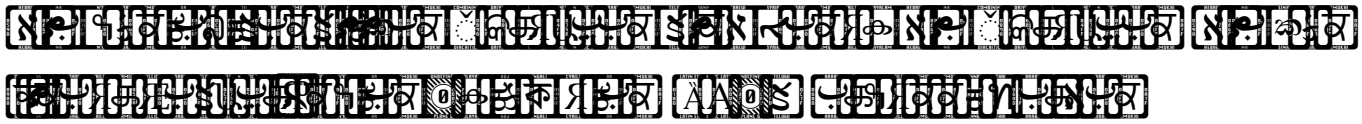
Immerman / Scelepcsenyi

Randbemerkung:

in seiner allgemeinen Formulierung löst das Theorem von Immerman und Scelepcsenyi noch eine weitere wichtige offene Frage.

Theorem

Die Klasse der kontextsensitiven Sprachen (Typ 1 - Sprachen) ist unter Komplement abgeschlossen.

- 
- 
-