Speicherplatz-Komplexität

Speicherplatz-Komplexität

Warum sollte uns die Ressource „Speicherplatz“ interessieren?

Um

- die Komplexität der Berechnung von **Gewinnstrategien** für viele nicht-triviale 2-Personen Spiele zu charakterisieren,
- „nicht-klassische“ Berechnungsarten wie **Randomisierung** oder **Quanten-Berechnungen** mit deterministischen Berechnungen zu vergleichen,
- die Klasse **parallelisierbarer Probleme** besser zu verstehen.
- die Komplexität der folgenden Probleme für **NFAs** zu klären:
  - (?) Akzeptiert ein NFA eine gegebene Eingabe?
  - (?) Sind zwei NFAs äquivalent?
  - (?) Minimiere einen NFA.
- die Komplexität des **Wortproblems für kontextfreie Sprachen** oder kontextsensitive Sprachen zu untersuchen,

Wie misst man Speicherplatz?

I-O Turingmaschinen

Eine **I-O Turingmaschine** M besitzt drei ein-dimensionale Bänder mit jeweils einem Kopf, wobei jeder Kopf in einem Schritt (höchstens) zur linken oder rechten Nachbarzelle der gegenwärtig besuchten Zelle wandern darf.

1. Das erste Band ist das **read-only Leseband**, das die Eingabe speichert.
2. Das zweite Band ist das **read-write Arbeitsband**, das aus $s(n)$ Zellen besteht.
3. Das dritte Band ist das **write-only Ausgabeband**. (Wir interpretieren das Drucken einer 0 (1), als das Verwerfen (Akzeptieren) der Eingabe).

Die Anzahl aller während der Berechnung für Eingabe w besuchten Zellen des zweiten Bands bezeichnen wir mit $\mathtt{DSPACE}_M(w)$.

Speicherplatz: Determinismus

Wir messen den Speicherplatz

M sei eine I-O Turingmaschine.

(a) Der Speicherplatzbedarf von M für Eingabelänge n ist

$$\texttt{DSPACE}_M(n) = \max\{\texttt{DSPACE}_M(w) \mid w \in \Sigma^n\}.$$

(b) Für $s : \mathbb{N} \to \mathbb{N}$ ist

$$\texttt{DSPACE}(s) = \{L \subseteq \Sigma^* \mid L(M) = L \text{ für eine I-O TM } M \text{ mit } \texttt{DSPACE}_M = \mathcal{O}(s)\}$$

die Klasse aller auf Platz $\mathcal{O}(s(n))$ lösbaren Entscheidungsprobleme.

(c) Die Komplexitätsklasse $\texttt{DL}$ besteht aus allen mit logarithmischem Speicherplatzbedarf erkennbaren Sprachen, also

$$\texttt{DL} = \texttt{DSPACE}(\log_2 n).$$

(d) Die Komplexitätsklasse $\texttt{PSPACE}$ besteht aus allen mit polynomiellem Speicherplatzbedarf erkennbaren Sprachen, also

$$\texttt{PSPACE} = \bigcup_{k \in \mathbb{N}} \texttt{DSPACE}(n^k).$$

Speicherplatz: Nichtdeterminismus

Speicherplatz nichtdeterministischer Rechnungen

Für eine nichtdeterministische TM M und eine Eingabe w ist $\mathtt{NSPACE}_M(w)$ der maximale Speicherplatz einer Berechnung von M auf w.

(a) Der Speicherplatzbedarf von M für Eingabelänge n ist

$$\mathtt{NSPACE}_M(n) = \max\{\mathtt{NSPACE}_M(w) \mid w \in \Sigma^n\}.$$

(b) Die Klasse aller mit Speicherplatz $\mathcal{O}(s)$ lösbaren Probleme ist

$$\mathtt{NSPACE}(s) = \{L \subseteq \Sigma^* \mid \text{es gibt eine nichtdeterministische I-O TM } M \text{ mit } L(M) = L \text{ und } \mathtt{NSPACE}_M = \mathcal{O}(s)\}.$$

(c) Die Komplexitätsklasse $\mathtt{NL}$ besteht aus allen nichtdeterministisch mit logarithmischem Speicherplatzbedarf erkennbaren Sprachen, also

$$\mathtt{NL} = \mathtt{NSPACE}(\log_2 n).$$

(d) Die Komplexitätsklasse $\mathtt{NPSPACE}$ besteht aus allen nichtdeterministisch, mit polynomiellem Speicherplatzbedarf erkennbaren Sprachen, also

$$\mathtt{NPSPACE} = \bigcup_{k \in \mathbb{N}} \mathtt{NSPACE}(n^k).$$

Speicherplatz: Alternation

Speicherplatz alternierender Berechnungen

Der maximale Speicherplatzbedarf einer Berechnung eines alternierenden Algorithmus A auf Eingabe x ist

$$\mathrm{aspace}_A(x).$$

(a) Der Speicherplatz von A für Eingabelänge n ist

$$\mathrm{aspace}_A(n) := \max\{\mathrm{aspace}_A(x) \ : \ x \in \Sigma^n\}$$

(b) Sei $s : \mathbb{N} \to \mathbb{N}$ gegeben. Dann definiere

$$\mathtt{ASPACE}(s) := \{L \subseteq \Sigma^* \quad : \quad L = L(A) \text{ für eine alternierende I-O TM } A \text{ mit } \mathrm{aspace}_A = \mathcal{O}(s) \ \}.$$

(c) Definiere „alternierenden polynomiellen Speicherplatz“ durch

$$\mathtt{APSPACE} := \bigcup_{k \in \mathbb{N}} \mathtt{ASPACE}(n^k).$$

Sind diese Klassen denn überhaupt interessant?

Sind diese Klassen überhaupt interessant? Und wie!

Es ist

(a) $\mathtt{ASPACE}(\log_2 n) = \mathtt{P}$,

(b) $\mathtt{APSPACE} = \mathtt{EXP}$ und

(c) $\mathtt{AP} = \mathtt{PSPACE}$.

Siehe Übungsaufgaben.

Speicherkomplexität: Die wichtigen Fragestellungen

- ? Ist DL eine echte Teilmenge von NL?
  - (∗) Wie sehen die für DL schwierigsten Probleme in NL aus?
  - (∗) Für welches kleinste s gilt

$$\mathrm{NL} \subseteq \mathrm{DSPACE}(s).$$

  - (∗) Wenn $L \in \mathrm{NL}$, ist dann auch $\overline{L} \in \mathrm{NL}$? D.h. ist NL abgeschlossen unter Komplement?
- ? Ist NL eine Teilmenge von P, und wenn ja, ist NL eine echte Teilmenge?
- ? Sind alle Sprachen in NL parallelisierbar?
- ? Ist P eine echte Teilmenge von PSPACE?
  - (∗) Wie sehen die für P schwierigsten Probleme in PSPACE aus?
  - (∗) Ist $\mathrm{NP} \subseteq \mathrm{PSPACE}$? Kann man allgemeine probabilistische Berechnungen, die in polynomieller Zeit ablaufen, in PSPACE simulieren?
  - (∗) Lassen sich Quantenberechnungen in PSPACE simulieren?
- ? Wie ordnen sich die Klassen regulärer, kontextfreier und kontextsensitiver Sprachen in die Speicherplatzklassen ein?
  - (∗) Die Chomsky-Hierarchie.

Sub-Logarithmischer Speicherplatz

Sub-logarithmischer Speicher

Für Eingaben der Länge n wird logarithmischer Speicher $\mathcal{O}(\log_2 n)$ benötigt, um sich an eine Eingabeposition zu erinnern.

Und wenn nur Speicher $o(\log_2 n)$ zur Verfügung steht?

(a) Aus welchen Sprachen besteht die Komplexitätsklasse $\mathtt{DSPACE}(0)$?
- TMs ohne Speicher sind Zwei-Weg Automaten.

$$\mathtt{DSPACE}(0) = \text{ Die Klasse der regulären Sprachen.}$$

(b) Leben nicht-reguläre Sprachen in $\mathtt{DSPACE}(\log_2 \log_2 n)$?
- $\text{bin}(i)$ sei die Binärdarstellung der Zahl i ohne führende Nullen.
- Für Eingabealphabet $\Sigma = \{0, 1, \$\}$ definiere

$$\text{BIN} = \{\text{bin}(1)\$\text{bin}(2)\$\cdots\$\text{bin}(n) \mid n \in \mathbb{N}\}.$$

- $\text{BIN} \in \mathtt{DSPACE}(\log_2 \log_2 n)$: $\text{bin}(i)$ besteht für $i \leqslant n$ aus $\leqslant \lceil \log_2 n \rceil$ Bits.
- BIN ist nicht regulär.

$$\mathtt{DSPACE}(o(\log_2 \log_2 n)) = \text{ Die Klasse der regulären Sprachen.}$$

Logarithmischer Speicherplatz

Logarithmischer Speicherplatz

DL und NL gehören zu den wichtigsten Speicherplatz-Klassen.

- Die Berechnungskraft ist durchaus signifikant, da die Maschinen sich jetzt Positionen in der Eingabe merken können.
- Viele Eigenschaften, die für DL und NL gelten, verallgemeinern sich auf beliebige Speicherplatzklassen.

  Dieses Phänomen werden wir im Satz von Savitch und im Satz von Immerman-Szlepscenyi beobachten.
- Zusammenhang zur Berechenbarkeit in logarithmischer paralleler Zeit.

Wir beginnen mit deterministisch logarithmischem Platz.

Programmiertricks für $\mathtt{DL}$

Das Bandalphabet –solange konstant groß – kann beliebig gewählt werden.

- Statt nur ein Arbeitsband können wir uns **konstant viele Arbeitsbänder** (mit jeweils einem Lese-Schreibkopf) erlauben.
  - Wir können uns das Arbeitsband in Spuren unterteilt denken.
- Statt nur einen Lesekopf auf dem Eingabeband können wir uns **konstant viele Leseköpfe** erlauben.
  - Speichere die Position jedes Lesekopfes auf dem Arbeitsband
  - und benutze einen Zähler, um die Position zu erreichen.

Die Sprache der Palindrome

PALINDROM $\in \mathtt{DL}$. (PALINDROM ist die Menge aller Palindrome über dem Alphabet $\{0, 1\}$.)

1. Arbeite mit zwei Leseköpfen:
   - der erste Kopf bleibt auf der ersten Eingabeposition stehen,
   - der zweite Kopf läuft zur letzten Eingabeposition.
2. Die beiden Köpfe vergleichen alle Buchstabenpaare $(w_i, w_{|w|-i+1})$, indem sie aufeinander zulaufen.

$\mathtt{DL}$ und $\mathtt{NL}$,
Was kann man auf logarithmischem Platz rechnen?

In $\mathtt{DL}$ liegen

1. alle **regulären** Sprachen,
2. **einige kontextfreien Sprachen** wie etwa
   - die Palindromsprache,
   - die Dycksprache mit beliebig vielen Klammertypen,
3. sogar **einige** kontextsensitive Sprachen wie etwa $\{a^n b^n c^n \mid n \in \mathbb{N}\}$,
4. die Addition und Multiplikation von Zahlen in Binärdarstellung (und damit auch die Auswertung von Polynomen)
5. die Matrizenmultiplikation,
6. sogar **U-REACHABILITY**, die Menge aller **ungerichteten** Graphen G, die einen Weg von Knoten 1 nach Knoten 2 besitzen,
   - Der Graph G werde durch seine Adjazenzmatrix repräsentiert.
   - Der Nachweis ist kompliziert.

D-REACHABILITY die Menge aller **gerichteten** Graphen G, die einen Weg von Knoten 1 nach Knoten 2 besitzen, gehört vermutlich **nicht** zu $\mathtt{DL}$.

Wie mächtig ist $\mathtt{DL}$? (2/3)

(a) Die I-O Turingmaschine M arbeite mit Speicherplatz $\mathbf{s}$. Dann ist die Laufzeit von M für Eingaben der Länge n durch $\mathbf{n} \cdot \mathbf{2}^{\mathcal{O}(\mathbf{s(n)})}$ beschränkt.

(b) Es gelte $s(n) \geqslant \log_2 n$. Dann ist

$$\mathtt{DSPACE}(s) \subseteq \bigcup_{k \in \mathbb{N}} \mathtt{DTIME}(2^{k \cdot s}).$$

(c) Es ist $\mathtt{DL} \subseteq P$.

(∗) Teil (b) folgt aus Teil (a), wenn wir die Annahme $s(n) \geqslant \log_2 n$ beachten.

(∗) Teil (c) folgt aus Teil (b).

(∗) Idee für Teil (a): Bei zu langer Laufzeit sollte eine platz-beschränkte Maschine in eine Endlosschleife geraten!

Wie mächtig ist $\mathtt{DL}$? (3/3)

Für eine I-O Turingmaschine besteht eine **Konfiguration** aus

1. dem gegenwärtigen Zustand,
2. der Position des Lesekopfs,
3. der Position des Kopfs auf dem Arbeitsband und
4. dem Inhalt des Arbeitsbands.

- Wenn eine I-O Turingmaschine M eine Konfiguration zweimal annimmt, dann steckt M in einer Endlosschleife!
- Für Speicher s und Eingabelänge n gibt es höchstens

  $$\mathcal{O}(n \cdot s \cdot 2^{\mathcal{O}(s)}) = n \cdot 2^{\mathcal{O}(s)}$$

  Konfigurationen.
- Die Laufzeit von M ist durch die Anzahl der Konfigurationen beschränkt.

Wie mächtig ist NL?

- D-REACHABILITY gehört wahrscheinlich nicht zu $\mathtt{DL}$, wohl aber zu $\mathtt{NL}$.
  - Eine nichtdeterministische Maschine rät einen Weg von Knoten 1 nach 2.
- Gehört D-UNREACHABILITY zu $\mathtt{NL}$? Ein gerichteter Graph gehört zu D-UNREACHABILITY, wenn *G keinen* Weg von Knoten 1 nach Knoten 2 besitzt.
- Das Wortproblem für NFAs (akzeptiert ein gegebener NFA eine gegebene Eingabe) gehört zu $\mathtt{NL}$, wahrscheinlich aber nicht zu $\mathtt{DL}$.

  Tatsächlich sind das Wortproblem für NFAs und D-REACHABILITY gleich-schwer.
- 2-SAT, das Erfüllbarkeitsproblem für KNF-Formeln mit höchstens zwei Literalen pro Klausel gehört zu $\mathtt{NL}$. Gehört 2-SAT zu $\mathtt{DL}$?
- Es gibt kontextfreie Sprachen, die zu $\mathtt{NL}$ aber wahrscheinlich nicht $\mathtt{DL}$ gehören.
  - Gehören alle kontextfreien Sprachen zu $\mathtt{NL}$?

Ist $\mathtt{DL}$ eine echte Teilmenge von $\mathtt{NL}$?

LOGSPACE-Reduktionen

LOGSPACE-Reduktionen

Ist D-REACHABILITY deterministisch mit logarithmischem Speicherplatz erkennbar?

- Wir zeigen, dass eine positive Antwort die Gleichheit $\mathtt{DL} = \mathtt{NL}$ erzwingt:
  - Die wahrscheinliche Antwort ist also negativ.
- Insbesondere zeigen wir, dass D-REACHABILITY eine schwierigste Sprache in $\mathtt{NL}$ ist, wobei „Schwierigkeit“ gemessen wird durch

  LOGSPACE-Reduktionen.

L ist LOGSPACE-reduzierbar auf K, geschrieben

$$L \leqslant_{\mathrm{LOG}} K,$$

falls es eine (deterministische) I-O Turingmaschine M mit logarithmischem Speicherplatzbedarf gibt, so dass für alle Eingaben $w \in \Sigma_1^*$,

$$w \in L \iff M(w) \in K.$$

NL-Vollständigkeit

(a) Die Sprache K heißt NL-hart, falls $L \leqslant_{\mathrm{LOG}} K$ für alle Sprachen $L \in \mathrm{NL}$.
(b) Die Sprache K heißt NL-vollständig, wenn K NL-hart ist und $K \in \mathrm{NL}$.

Die wesentlichen Eigenschaften der LOGSPACE-Reduktion:

- Wenn $L \leqslant_{\mathrm{LOG}} K$ und wenn $K \in \mathrm{DL}$, dann ist $L \in \mathrm{DL}$.
  - $\mathrm{DL} = \mathrm{NL}$, wenn irgendeine NL-vollständige Sprache in DL liegt.
    Die Sprache K sei NL-vollständig. Dann sind äquivalent
    $$K \in \mathrm{DL} \iff \mathrm{DL} = \mathrm{NL}.$$
- Wenn $M \leqslant_{\mathrm{LOG}} K$ und $K \leqslant_{\mathrm{LOG}} L$, dann ist $M \leqslant_{\mathrm{LOG}} L$.
  - Wie weist man nach, dass eine Sprache L NL-vollständig ist?
    Wenn K NL-hart ist und wenn $K \leqslant_{\mathrm{LOG}} L$, dann ist auch L NL-hart.

D-REACHABILITY ist $\mathtt{NL}$-vollständig

D-REACHABILITY ist $\mathtt{NL}$-vollständig (1/2)

M sei eine *nichtdeterministische* Turingmaschine mit logarithmischem Speicherplatz. Wir definieren das zentrale Konzept des

Berechnungsgraphen $G_M(w)$ von M auf Eingabe w.

- Die *Knoten* von $G_M(w)$ entsprechen den Konfigurationen.
- Wir fügen eine *Kante* von Konfiguration c nach Konfiguration d ein, wenn M (mit Eingabe w) in einem Schritt von c nach d gelangen kann.

$G_M(w)$ kann von einer *deterministischen* I-O Turingmaschine mit logarithmischem Speicherplatz berechnet werden.

- Stelle deterministisch, auf logarithmischem Platz, fest, ob ein 1-Schritt Übergang von einer Konfiguration c_1 zu einer Konfiguration c_2 möglich ist.
- Kein Problem, denn der Speicher für c_1 wie auch für c_2 ist logarithmisch.

1. D-REACHABILITY liegt offensichtlich in $\mathtt{NL}$.

2. *Zeige*: $L \leqslant_{\mathrm{LOG}}$ D-REACHABILITY für eine beliebige Sprache $L \in \mathtt{NL}$.
   - Es ist $L = L(M)$ für eine nichtdeterministische I-O Turingmaschine M mit logarithmischem Speicher.
   - Der Berechnungsgraph $G_M(w)$ kann von einer deterministischen I-O TM berechnet werden.
     - ★ Der Knoten der Anfangskonfiguration erhält den Namen 1.
     - ★ Zusätzlich füge eine Kante von jeder akzeptierenden Haltekonfiguration zu einem neuen Knoten ein, dem wir den „Namen“ 2 geben.

$$w \in L \iff G_M(w) \in \text{ D-REACHABILITY.}$$

Weitere $\mathtt{NL}$-vollständige Sprachen

1. Das Leerheitsproblem für DFAs (Akzeptiert ein gegebener DFA irgendeine Eingabe?)
2. Die Simulation von NFAs (Akzeptiert ein gegebener NFA eine gegebene Eingabe?)
3. 2-SAT
4. Das Wortproblem für bestimmte kontextfreie Sprachen.

Beweis: In den Übungen.

Später: Wenn L $\mathtt{NL}$-vollständig ist, dann ist auch die Komplementsprache $\mathtt{NL}$-vollständig.

Wie mächtig ist $\mathtt{NL}$?

$\mathtt{DL} \subseteq \mathtt{NL} \subseteq \mathtt{P} \subseteq \mathtt{NP}$.

- Die Beziehungen $\mathtt{DL} \subseteq \mathtt{NL}$ sowie $\mathtt{P} \subseteq \mathtt{NP}$ sind offensichtlich.
- Es genügt der Nachweis von $\mathtt{NL} \subseteq \mathtt{P}$.
  - Für eine beliebige Sprache L in $\mathtt{NL}$ gilt $L \leqslant_{\mathrm{LOG}}$ D-REACHABILITY.
  - Wenn $L \leqslant_{\mathrm{LOG}} K$ und $K \in \mathtt{P}$, dann ist auch $L \in \mathtt{P}$. Warum?
    - ⋆ Deterministische I-O Turingmaschinen mit logarithmischem Speicher besitzen eine polynomielle Laufzeit.
  - Da D-REACHABILITY in $\mathtt{P}$ liegt, folgt somit auch $L \in \mathtt{P}$.

Die Chomsky-Hierarchie: Worum geht's?

Die Chomsky-Hierarchie

1. Grammatiken ohne jede Einschränkung heißen **Typ-0**-Grammatiken. Die entsprechende Sprachenfamilie ist

$$\mathcal{L}_0 = \Big\{ L(G) \mid G \text{ ist vom Typ } 0 \Big\}$$

2. Eine Grammatik G mit Produktionen der Form

$$u \to v \quad \text{mit } |u| \leqslant |v|$$

heißt **Typ 1** oder **kontextsensitiv**. Die zugehörige Sprachenfamilie ist

$$\mathcal{L}_1 = \Big\{ L(G) \mid G \text{ ist vom Typ } 1 \Big\} \cup \Big\{ L(G) \cup \{\epsilon\} \mid G \text{ ist vom Typ } 1 \Big\}$$

3. **Kontextfreie** Grammatiken werden als Grammatiken vom **Typ 2** bezeichnet. Die zugehörige Sprachenfamilie ist

$$\mathcal{L}_2 = \Big\{ L(G) \mid G \text{ hat Typ } 2 \Big\}$$

4. Eine **reguläre** Grammatik heißt auch **Typ-3**-Grammatik. Die zugehörige Sprachenfamilie ist

$$\mathcal{L}_3 = \Big\{ L(G) \mid G \text{ hat Typ } 3 \Big\}$$

Die Chomsky Hierarchie und Platzkomplexität

Es kann gezeigt werden:

(a) $\mathcal{L}_0$ ist die Klasse aller rekursiv aufzählbaren Sprachen.

(b) Es gilt $\mathcal{L}_1 = \mathtt{NSPACE}(n)$ und $\mathcal{L}_1$ ist die Klasse aller Sprachen, die von nichtdeterministischen Turingmaschinen auf linearem Platz erkannt werden.

(c) Es ist $\mathtt{NL} \subseteq \mathtt{LOGCFL} \subseteq \mathtt{DSPACE}(\log_2^2 n)$. LOGCFL ist der Abschluß kontextfreier Sprachen unter LOGSPACE-Reduktionen. Insbesondere gilt also $\mathcal{L}_2 \subseteq \mathtt{DSPACE}(\log_2^2 n)$.

(d) Die Klasse der regulären Sprachen stimmt mit der Klasse $\mathtt{DSPACE}(0)$ überein, es gilt also $\mathcal{L}_3 = \mathtt{DSPACE}(0)$.

(e) $\mathcal{L}_3 \subset \mathcal{L}_2 \subset \mathcal{L}_1 \subset \mathcal{L}_0$ und alle Inklusionen sind echt.

Diagonalisierung

Platz-Konstruierbarkeit

Eine Funktion

$$s : \mathbb{N} \to \mathbb{N}$$

heißt genau dann

platz-konstruierbar,

wenn

- $s(n) = \Omega(\log_2 n)$ und
- es eine deterministische I-O TM gibt, die für Eingabe 0^n genau $s(n)$ Zellen des Arbeitsbands markiert und $\leqslant s(n)$ Speicherplatz benötigt.

Wir zeigen, dass man mit mehr zur Verfügung stehendem Speicher mehr Entscheidungsprobleme lösen kann.

Die zentrale Idee: *Diagonalisierung*.

Eine Speicherplatz-Hierarchie (1/2)

Die Funktion s sei platz-konstruierbar $\Longrightarrow$

$$\mathtt{DSPACE}(o(s)) \text{ ist eine echte Teilmenge von } \mathtt{DSPACE}(s).$$

Die *Diagonalisierungsmethode*: M^* ist nicht auf Platz $o(s)$ simulierbar.

1. M^* bestimmt die Länge n der Eingabe w.
2. M^* steckt auf dem Arbeitsband $2s(n)$ Zellen ab.
   /* Dies ist mit Speicherplatz $s(n)$ möglich, da s platz-konstruierbar ist. */
3. Wenn
   $$w \neq \langle M \rangle 0^k$$
   für eine I-O Turingmaschine M und eine Zahl $k \in \mathbb{N}$ ist, dann verwirft M^*.
   /* $\langle M \rangle$ bezeichnet die „Gödelnummer" der Turingmaschine M. */
4. M^* simuliert M auf Eingabe $w = \langle M \rangle 0^k$ und beginnt die Simulation mit dem Kopf in der Mitte des abgesteckten Bereichs.
   - Wenn M irgendwann den abgesteckten Bereich verlässt oder mehr als $2^{s(n)}$ Schritte benötigt, dann verwirft M^*. /* Achtung: Schleife! */
   - Ansonsten *akzeptiert* M^* die Eingabe w genau dann, wenn M **verwirft**.

/* Auf Platz $\mathcal{O}(s(n))$: Zähle bis $2^{s(n)} - 1$ und simuliere M. */

Eine Speicherplatz-Hierarchie

(a) M^* hält immer und kommt mit Speicherplatzbedarf $\mathcal{O}(s(n))$ aus. $\Longrightarrow$ $L(M^*) \in \texttt{DSPACE}(s)$.

(b) Und wenn $L(M^*) = L(M)$ für eine I-O TM M mit Speicherplatz $o(s)$?

- Für *genügend große Eingabelänge n* rechnet M stets in Zeit $\leqslant 2^s \Longrightarrow$ M^* kann M für Eingaben $w = \langle M \rangle \mathbf{0^k}$ mit *genügend großem* k simulieren.
- Schritt (4) garantiert, dass sich $L(M)$ und $L(M^*)$ unterscheiden: Speicherplatz $o(s)$ ist unzureichend für die Berechnung von $L(M^*)$. $\square$

Der Satz von Savitch (1970)

Der Satz von Savitch

(a) D-REACHABILITY $\in \mathtt{DSPACE}(\log_2^2 n)$.

(b) Die Funktion s sei platz-konstruierbar. Dann ist

$$\mathtt{NSPACE}(s) \subseteq \mathtt{DSPACE}(s^2)$$

und insbesondere folgt

$$\mathtt{NL} \subseteq \mathtt{DSPACE}(\log_2^2 n) \quad \text{und} \quad \mathtt{PSPACE} = \mathtt{NPSPACE}.$$

- Leider können wir D-REACHABILITY weder mit Tiefensuche noch mit Breitensuche lösen:
  - Sowohl der Stack der Tiefensuche wie auch die Queue der Breitensuche verlangen bis zu linearem Speicherplatz.

  Wir erfinden ein neues speicher-effizientes Traversierungsverfahren.
- Teil (b) stellt sich als direkte Konsequenz von Teil (a) heraus: D-REACHABILITY nimmt eine Schlüsselrolle ein.

D-REACHABILITY $\in \mathtt{DSPACE}(\log_2^2 n)$ (1/2)

Der **Algorithmus von Savitch** überprüft, ob der Eingabegraph G einen Weg von *Knoten u* nach *Knoten v* der *genauen Länge m* besitzt.

```
Savitch (u, v, m):
    Wenn m = 1, dann akzeptiere, wenn (u, v) eine Kante ist.
    Sonst akzeptiere wenn es einen Knoten w gibt, so dass
    Savitch(u, w, m/2) und Savitch(w, v, m/2) akzeptieren.
```

✓ Der Masteraufruf erfolgt für $u = 1, v = 2$ und $m = n - 1$ bei n Knoten.

? Wie viel Speicherplatz wird benötigt?

Analyse von Speicherplatz und Laufzeit:

- Speicherplatzverbrauch:
  - Ein Stack der maximalen Höhe $\lceil \log_2 n \rceil$ genügt.
  - Jedes Element des Stacks entspricht zwei Knoten und einer Zahl $m \leqslant n$: Logarithmischer Speicher reicht für Stackelemente.
  - Insgesamt benötigen wir Speicherplatz $\mathcal{O}(\log_2^2 n)$.
- Die Laufzeit ist höchstens exponentiell im Speicherplatz und deshalb durch $2^{\mathcal{O}(\log_2^2 n)}$ beschränkt. Furchtbar, aber was soll's?

Warum gilt $\mathrm{NSPACE}(s) \subseteq \mathrm{DSPACE}(s^2)$, wenn s platzkonstuierbar ist?

$\mathrm{NSPACE}(s) \subseteq \mathrm{DSPACE}(s^2)$

Es sei $s(n) \geqslant \log_2 n$ und s sei platz-konstruierbar.

- M sei eine beliebige nichtdeterministische Turingmaschine mit Speicherplatzbedarf s und w sei eine Eingabe.
- Wir konstruieren eine deterministische Turingmaschine M^*, die M auf Eingabe w mit Speicherplatz $\mathcal{O}(s^2)$ simuliert.
  - Da s platz-konstruierbar ist, kann M^* einen Speicherplatz von $s(n)$ Zellen abstecken und damit alle Konfigurationen von M systematisch erzeugen.
  - M^* führt den Algorithmus von Savitch aus und zwar

    für den Berechnungsgraphen $G_M(w)$ mit Knoten 1,2 entsprechend gewählt.

  - M^* akzeptiert $w \quad :\Longleftrightarrow \quad$ es gibt einen Weg in $G_M(w)$ von Knoten 1 nach 2.

M^* benötigt Speicherplatz $\mathcal{O}(s^2)$: Speicherplatz $\mathcal{O}(\log_2^2 K)$ genügt, um D-REACHABILITY für Graphen mit $K = 2^{\mathcal{O}(s(n))}$ Knoten zu lösen.

Der Satz von Immerman und Szelepcsenyi (1988)

(a) D-UNREACHABILITY, das Komplement von D-REACHABILITY, liegt ebenfalls in $\mathtt{NL}$.

(b) Die Funktion s sei platz-konstruierbar. Dann ist

$$\mathtt{NSPACE}(s) = \mathtt{coNSPACE}(s),$$

wobei

$$\mathtt{coNSPACE}(s) = \{\overline{L} \mid L \in \mathtt{NSPACE}(s)\}$$

aus den Komplementen der Sprachen in $\mathtt{NSPACE}(s)$ besteht.

$\Longrightarrow$ Nichtdeterministischer Speicherplatz ist abgeschlossen unter Komplementbildung.

Der Satz von Immerman und Szelepcsenyi (2/4)

Der Graph G sei die Eingabe für D-UNREACHABILITY.

1. Angenommen, wir können das *Anzahlproblem* in $\mathtt{NL}$ lösen, also
   die Anzahl **m** der von Knoten 1 aus erreichbaren Knoten bestimmen.
   Wir konstruieren eine nichtdeterministische Maschine *M*, die D-UNREACHABILITY mit logarithmischem Speicher akzeptiert.
2. Im zweiten Schritt lösen wir das *Anzahlproblem* in $\mathtt{NL}$.

- m Knoten seien vom Knoten 1 aus erreichbar.
- Die Idee und ihre Implementierung.
  - M rät *nacheinander* von 1 aus erreichbare Knoten $v_1 < v_2 < \cdots < v_m$. Wie?
    - Für v_i rät M einen Weg $1 \xrightarrow{*} v_i$.
    - Wenn dies erfolgreich war, rät M danach v_{i+1} mit $v_i < v_{i+1}$ und wiederholt ihr Vorgehen für v_{i+1}.
  - Wenn Knoten 2 von $v_1, \ldots, v_m$ verschieden ist, dann ist Knoten 2 **nicht** von Knoten 1 aus erreichbar und M akzeptiert.

Der Satz von Immerman und Szelepcsenyi (3/4)

Wir müssen das *Anzahlproblem* für den Graphen G lösen.

- Knoten 1 erreiche m_i Knoten durch Wege der Länge höchstens i.
  - Offensichtlich ist $m_0 = 1$ und $m_{n-1} =?$ ist zu bestimmen.
- Zeige, dass m_{i+1} in NL berechnet werden kann, wenn m_i bekannt ist.
  - Setze zu Anfang $m_{i+1} = 0$.
  - Verarbeite die Knoten **k** in *aufsteigender* Reihenfolge.
    - Rate nacheinander die m_i von 1 durch einen Weg der Länge höchstens m_i erreichbaren Knoten $v_{i,r}$ mit $v_{i,1} < \cdots < v_{i,m_i}$ und verifiziere diese Eigenschaft. Scheitert die Verifikation für irgendein $v_{i,r}$, dann verwerfe.
    - Überprüfe für jedes r ob

      $$(v_{i,r}, k) \text{ eine Kante ist oder } v_{i,r} = k \text{ gilt.}$$

      Wenn ja, setze $m_{i+1} := m_{i+1} + 1$ und $k := k + 1$.

      Ansonsten rate $v_{i,r+1}$ und fahre mit der Untersuchung von k fort.
    - Verwerfe, wenn m_i in der Iteration für k nicht inkrementiert wurde, aber weniger als m_i Knoten geraten wurden.

- Die Sprache L werde von einer nichtdeterministischen Turingmaschine M mit Speicherplatzbedarf $s \geqslant \log_2 n$ erkannt. (s sei platzkonstruierbar).
- Wir bauen eine nichtdeterministische TM M^*, die das Komplement $\overline{L}$ mit Speicherplatzbedarf $\mathcal{O}(s)$ erkennt.

- M^* muss genau dann akzeptieren, wenn es keine akzeptierende Berechnung gibt, wenn also $G_M(w)$ zu D-UNREACHABILITY gehört.
  - M^* wendet den Algorithmus von Immerman und Szelepscenyi für den Berechnungsgraphen $G_M(w)$ an.
  - M^* muss klären, welche Kanten in $G_M(w)$ einzusetzen sind:
    Platz $\mathcal{O}(s)$ reicht, da die Konfigurationen von M nur Platz $\mathcal{O}(s)$ benötigen.
- Also genügt insgesamt Platz $\mathcal{O}(s)$.

PSPACE-Vollständigkeit

$\mathrm{P} \stackrel{?}{=} \mathrm{PSPACE}$: PSPACE-Vollständigkeit

Die schwierigsten Sprachen in PSPACE für die *polynomielle Reduktion*:

(a) Eine Sprache L heißt PSPACE-hart, falls $K \leqslant_P L$ für alle Sprachen $K \in \mathrm{PSPACE}$ gilt.

(b) L heißt PSPACE-vollständig, falls L PSPACE-hart ist und falls $L \in \mathrm{PSPACE}$.

Die wesentlichen Eigenschaften der polynomiellen Reduktion:

- Wenn $L \leqslant_P K$ und wenn $K \in \mathrm{P}$, dann ist auch $L \in \mathrm{P}$.
  - Als Konsequenz: $\mathrm{P} = \mathrm{PSPACE}$, wenn irgendeine PSPACE-vollständige Sprache in P liegt.

    Die Sprache L sei PSPACE-vollständig. Dann sind äquivalent

$$L \in \mathrm{P} \iff \mathrm{P} = \mathrm{PSPACE}.$$

- Wenn $M \leqslant_P K$ und $K \leqslant_P L$, dann ist $M \leqslant_P L$.
  - Als Konsequenz: Wie zeigt man neue PSPACE-vollständige Sprachen?

    Wenn K PSPACE-hart ist und wenn $K \leqslant_P L$, dann ist auch L PSPACE-hart.

QBF: Quantifizierte Boolesche Formeln

Quantifizierte Boolesche Formeln

Eine *quantifizierte Boolesche Formel* ϕ besteht aus einem

- *Quantorenteil* mit Existenz- und Allquantoren
- gefolgt von einer *aussagenlogischen Formel* α. Jede in α vorkommende Variable wird von genau einem Quantor gebunden.

$$\text{QBF} = \{\langle\phi\rangle \mid \phi \text{ ist eine wahre quantifizierte Boolesche Formel }\}.$$

- Die Formel $\phi \equiv \exists p \forall q \left((p \vee \neg q) \wedge (\neg p \vee q)\right)$ ist falsch, denn
  - sie drückt die Äquivalenz von p und q aus, aber es gibt keinen Wahrheitswert für p, der mit 0 und 1 äquivalent ist.
  - Also ist $\phi \notin$ QBF.
- Die Formel $\psi \equiv \forall p \exists q \left((p \vee \neg q) \wedge (\neg p \vee q)\right)$ ist hingegen wahr, denn
  - zu jedem Wahrheitswert für p gibt es einen äquivalenten Wert für q.
  - Also ist $\psi \in$ QBF.

QBF $\in \mathrm{DSPACE}(n)$

- QBF ist ein Spiel des Existenzquantors $\exists$ gegen den Allquantor $\forall$.
- Bei n Variablen ist das Spiel nach höchstens n Zügen vorbei $\Longrightarrow$ Der „Spielbaum" hat Tiefe höchstens n.
- Werte den Spielbaum in $\mathrm{DSPACE}(n)$ aus.

QBF ist PSPACE-hart:

- Die Sprache $L = L(M) \in \mathrm{PSPACE}$ werde von einer deterministischen Turingmaschine M mit Speicherplatzbedarf $\mathcal{O}(n^k)$ berechnet.
- Zeige

$$L \leqslant_P QBF :$$

  Baue für jede Eingabe w von L in polynomieller Zeit eine quantifizierte Boolesche Formel ϕ_w mit

$$w \in L \iff \phi_w \text{ ist wahr.}$$

Wir müssen eine Formel ϕ_w bauen mit

$$w \in L \iff \phi_w \text{ ist wahr.}$$

- Wir wissen: Es ist $L = L(M)$ für eine deterministische TM M mit Speicherplatzbedarf $s = \mathcal{O}(n^k)$.
- Wie im NP-Vollständigkeitsbeweis von KNF-SAT führen wir aussagenlogische Variablen ein wie
  - $\mathit{Kopf}^t(z)$ für die Kopfposition. $\mathit{Kopf}^t(z)$ *soll* genau dann wahr sein, wenn der Kopf zum Zeitpunkt t auf Zelle z steht,
  - $\mathit{Zelle}^t(z, a)$ für den Zelleninhalt. $\mathit{Zelle}^t(z, a)$ *soll* genau dann wahr sein, wenn die Zelle z zum Zeitpunkt t mit dem Buchstaben a beschriftet ist und
  - $\mathit{Zustand}^t(q)$ für den aktuellen Zustand. $\mathit{Zustand}^t(q)$ *soll* genau dann wahr sein, wenn q der Zustand zum Zeitpunkt ist.
- Aber M kann bis zu $2^{\mathcal{O}(n^k)}$ Schritte ausführen!!!

QBF ist PSPACE-vollständig (3/4)

(a) Für Konfigurationen c und d schreibe

$$c \xrightarrow{t} d,$$

wenn M nach höchstens t Schritten, von Konfiguration c aus startend, Konfiguration d erreicht.

(b) Baue eine höchstens polynomiell (polynomiell in „was“?) lange Formel $\psi_t(c, d)$, die genau dann wahr ist, wenn $c \xrightarrow{t} d$ gilt.

- Zur Erinnerung: M benötigt höchstens Speicher s.
- c_0 sei die Anfangskonfiguration und c_a sei die eindeutig bestimmte akzeptierende Haltekonfiguration von M. Dann gilt

$$w \in L \iff \psi_{2^s}(c_0, c_a).$$

- Setze

$$\phi_w := \psi_{2^s}(c_0, c_a).$$

Und jetzt der Clou:

$$\psi_{2^{t+1}}(c,d) \;\equiv\; \exists e \forall f \forall g(\; ((f = c \wedge g = e) \vee (f = e \wedge g = d)) \rightarrow \psi_{2^t}(f,g)\;).$$

- $\exists e$ entspricht der *Folge* der Existenz-Quantoren zu den Variablen
  - Kopfposition, Zelleninhalt und Zustand der Konfiguration e.

  Ähnliches gilt für $\forall f$ und $\forall g$.
- $\psi_{2^{t+1}}(c,d)$: Eine Berechnung der Länge höchstens 2^{t+1} kann in zwei Berechnungen der Länge höchstens 2^t aufgespalten werden.
  - Der All-Quantor erlaubt eine *simultane* Überprüfung der beiden Berechnungen von c nach e und von e nach d.
- Dementsprechend wächst die Formellänge additiv um höchstens $\mathcal{O}(n^k)$, also höchstens um den Speicherplatzbedarf von M.

$\mathcal{O}(n^{2k})$ ist eine obere Schranke für die Länge der Formel $\phi_w = \psi_{2^s}(c_0, c_a)$.

Das GEOGRAPHIE-Spiel

In „Geographie“ wählen Alice und Bob abwechselnd noch nicht genannte Städtenamen: Jede Stadt muss mit dem letzten Buchstaben der zuvor genannten Stadt beginnen.

- Die Eingabe: Ein gerichteter Graph $G = (V, E)$ und ein ausgezeichneter Knoten $s \in V$.
- Die Spielregeln:
  - Zwei Spieler Alice und Bob wählen abwechselnd eine noch nicht benutzte Kante aus E.
    - Alice beginnt und wählt eine Kante mit Startknoten s.
    - Jede anschließend gewählte Kante muß im Endknoten der zuvor gewählten Kante beginnen.
    - Kein Knoten darf zweimal besucht werden.
  - Der Spieler verliert, der als erster nicht mehr ziehen kann.

Geographie ist PSPACE-vollständig.

PSPACE und die Komplexität von 2-Personen Spielen

- Wir haben schon beobachtet, dass sich QBF als ein Spiel zwischen dem Existenzquantor und dem Allquantor auffassen lässt.
- Wenn Spiele auf beliebige Spielgröße verallgemeinert werden, wie etwa auf $n \times n$ Bretter, sind viele weitere Vollständigkeits-Ergebnisse bekannt **www.ics.uci.edu/~eppstein/cgt/hard.html**:
  - Dame, Go, Othello, Schach, Sokoban sind Beispiele PSPACE-harter Spiele.
- PSPACE-Härte ist ein Gütesiegel, dass es sich um ein interessantes, weil sehr schwieriges Spiel handelt.

Leider kann die Komplexitätstheorie keine definitiven Aussagen über Spiele machen, deren Spielgröße fest ist – wie etwa eine fixierte Brettgröße.

Zusammenfassung

Zusammenfassung (1/2)

- In $\mathtt{DSPACE}(o(\log_2 \log_2 n))$ berechenbare Sprachen stimmen mit den regulären Sprachen überein.
- Die erste nicht-triviale Klasse ist die Klasse $\mathtt{DL}$ aller auf logarithmischem Platz berechenbaren Sprachen.
  - U-REACHABILITY gehört zu $\mathtt{DL}$.
  - D-REACHABILITY ist $\mathtt{NL}$-vollständig, also ein für $\mathtt{DL}$ schwierigstes Problem in $\mathtt{NL}$.
    - ⋆ Insbesondere folgt, dass $\mathtt{NL}$ in $\mathtt{P}$ enthalten ist.
    - ⋆ Weitere $\mathtt{NL}$-vollständige Probleme sind 2-SAT, das Leerheitsproblem für DFAs und das Problem, NFAs zu simulieren.
- Der Satz von Savitch zeigt

$$\text{D-REACHABILITY} \in \mathtt{DSPACE}(\log_2^2 n)$$

  sowie $\mathtt{NSPACE}(s) \subseteq \mathtt{DSPACE}(s^2)$ für platz-konstruierbare Funktionen s.
- Das Komplementverhalten ist „nicht typisch für Nichtdeterminismus“, denn

$$\mathtt{NSPACE}(s) = \mathtt{coNSPACE}(s)$$

  gilt, falls s platz-konstruierbar ist.

Zusammenfassung (2/2)

- PSPACE-Härte bzw. PSPACE-Vollständigkeit ist eine Charakterisierung „richtig“ schwieriger Entscheidungsprobleme. Zu den Beispielen gehören:
  - Das Problem der Bestimmung einer Gewinnstrategie für nicht-triviale Zwei-Personen Spiele (wie QBF oder Geographie),
  - Entscheidungsprobleme für reguläre Ausdrücke oder NFA, wie Universalität, das Äquivalenzproblem oder die Minimierung.
- Später: Die Klasse PSPACE enthält alle Entscheidungsprobleme, die durch randomisierte Algorithmen oder Quanten-Algorithmen in polynomieller Zeit lösbar sind.
- Wir haben die Chomsky-Hierarchie betrachtet.
  - Die von Typ-0 Grammatiken erzeugbaren Sprachen stimmen mit den rekursiv aufzählbaren Sprachen überein.
  - Die kontextsensitiven Sprachen sind noch zu komplex, da ihr Wortproblem PSPACE-vollständig sein kann.
  - Kontextfreie Sprachen können NL-hart sein, ihr Wortproblem ist aber in $\mathrm{DSPACE}(\log_2^2 n)$ lösbar.
  - Die Klasse der regulären Sprachen stimmt überein mit $\mathrm{DSPACE}(s)$, falls $s = o(\log_2 \log_2 n)$.