Das Wortproblem für kontextfreie Sprachen in subkubischer Zeit

Ahmad Yassin

Bachelorarbeit

25. Mai 2023

Gliederung

Gliederung

Einleitung

Wortproblem

Gegeben: Eine Grammatik $G = (S, \Sigma, N, P)$ und ein Wort w.
Frage: Ist $w \in L(G)$? Dabei beschreibt $L(G)$ die Sprache, die durch G erzeugt wird.

Einleitung

Wortproblem

Gegeben: Eine Grammatik $G = (S, \Sigma, N, P)$ und ein Wort w.
Frage: Ist $w \in L(G)$? Dabei beschreibt $L(G)$ die Sprache, die durch G erzeugt wird.

- Der Cocke-Younger-Kasami-Algorithmus (CYK) ist ein effizienter Algorithmus zur Lösung des Wortproblems für kontextfreie Sprachen.
- Die Zeitkomplexität des CYK-Algorithmus beträgt $O(n^3)$.
- Gibt es Algorithmen mit besserer Zeitkomplexität als der CYK-Algorithmus?

Einleitung

Wortproblem

Gegeben: Eine Grammatik $G = (S, \Sigma, N, P)$ und ein Wort w.
Frage: Ist $w \in L(G)$? Dabei beschreibt $L(G)$ die Sprache, die durch G erzeugt wird.

- Der Cocke-Younger-Kasami-Algorithmus (CYK) ist ein effizienter Algorithmus zur Lösung des Wortproblems für kontextfreie Sprachen.
- Die Zeitkomplexität des CYK-Algorithmus beträgt $O(n^3)$.
- Gibt es Algorithmen mit besserer Zeitkomplexität als der CYK-Algorithmus?
- Ja, es gibt viele, aber in dieser Arbeit werden wir uns den Leslie-G.-Valiant-Algorithmus (LGV) anschauen, der eine Zeitkomplexität von $O(n^{2.81})$ hat.

Ziel der Arbeit

- Der LGV-Algorithmus hat bessere Laufzeiten als der CYK-Algorithmus.

Ziel der Arbeit

- Der LGV-Algorithmus hat bessere Laufzeiten als der CYK-Algorithmus.
- Dazu werden beide Algorithmen implementiert.

Ziel der Arbeit

- Der LGV-Algorithmus hat bessere Laufzeiten als der CYK-Algorithmus.
- Dazu werden beide Algorithmen implementiert.
- Umwandlung in Chomsky-Normalform.

Ziel der Arbeit

- Der LGV-Algorithmus hat bessere Laufzeiten als der CYK-Algorithmus.
- Dazu werden beide Algorithmen implementiert.
- Umwandlung in Chomsky-Normalform.
- Die Implementierung erfolgt in Python Version 3.10.

Ziel der Arbeit

- Der LGV-Algorithmus hat bessere Laufzeiten als der CYK-Algorithmus.
- Dazu werden beide Algorithmen implementiert.
- Umwandlung in Chomsky-Normalform.
- Die Implementierung erfolgt in Python Version 3.10.
- Einsetzung dem Uni-Cluster, um die Laufzeiten zu messen.

Gliederung

Chomsky-Normalform

Definition (Chomsky-Normalform)

Eine kontextfreie Grammatik $G = (S, \Sigma, N, P)$ liegt in Chomsky-Normalform vor, wenn ihre Produktionen in einer von zwei einfachen Formen vorliegen:

1. $A \rightarrow BC$
2. $A \rightarrow a$

Chomsky-Normalform

Definition (Chomsky-Normalform)

Eine kontextfreie Grammatik $G = (S, \Sigma, N, P)$ liegt in Chomsky-Normalform vor, wenn ihre Produktionen in einer von zwei einfachen Formen vorliegen:

1. $A \rightarrow BC$
2. $A \rightarrow a$

Die Umwandlung in Chomsky-Normalform erfolgt in 4 Schritten:

- Eliminierung nutzloser Zeichen,
- Eliminierung von ε-Produktionen,
- Eliminierung von Einheitsproduktionen,
- Umwandlung in Chomsky-Normalform.

Eliminierung von nutzlosen Zeichen

Ein Zeichen X in einer Grammatik $G = (S, \Sigma, N, P)$ wird als nutzlos bezeichnet, wenn es keine Ableitung der Form $S \rightarrow^* \alpha X \beta \rightarrow^* w$ gibt.

Eliminierung von nutzlosen Zeichen

Ein Zeichen X in einer Grammatik $G = (S, \Sigma, N, P)$ wird als nutzlos bezeichnet, wenn es keine Ableitung der Form $S \rightarrow^* \alpha X \beta \rightarrow^* w$ gibt. Diese Zeichen werden in zwei Gruppen unterteilt:

1. Die Gruppe der nicht-erzeugenden Zeichen besteht aus nichtterminalen Zeichen, die keine Kette von terminalen Zeichen erzeugen können.
2. Die Gruppe der unerreichbaren Zeichen besteht aus terminalen und nichtterminalen Zeichen, die in keiner Ableitung von dem Startsymbol erreicht werden können.

Eliminierung von nutzlosen Zeichen

Ein Zeichen X in einer Grammatik $G = (S, \Sigma, N, P)$ wird als nutzlos bezeichnet, wenn es keine Ableitung der Form $S \rightarrow^* \alpha X \beta \rightarrow^* w$ gibt. Diese Zeichen werden in zwei Gruppen unterteilt:

1. Die Gruppe der nicht-erzeugenden Zeichen besteht aus nichtterminalen Zeichen, die keine Kette von terminalen Zeichen erzeugen können.
2. Die Gruppe der unerreichbaren Zeichen besteht aus terminalen und nichtterminalen Zeichen, die in keiner Ableitung von dem Startsymbol erreicht werden können.

Die Eliminierung der nutzlosen Zeichen erfolgt in zwei Schritten:

1. Suche nach nicht-erzeugenden Zeichen, um sie zu entfernen.
2. Suche nach unerreichbaren Zeichen, um sie zu entfernen.

Eliminierung von ε-Produktionen

Eine ε-Produktion ist eine Produktion der Form $A \to \varepsilon$ oder $A \to^* \varepsilon$.

Eliminierung von ε-Produktionen

Eine ε-Produktion ist eine Produktion der Form $A \rightarrow \varepsilon$ oder $A \rightarrow^* \varepsilon$.
Die Eliminierung erfolgt in zwei Schritten:

1. Suche nach eliminierbaren nichtterminalen Zeichen und fasse sie in einer Menge zusammen.
2. Entferne alle Produktionen, die auf Basis der erstellten Menge im vorherigen Schritt eliminiert werden können.

Eliminierung von ε-Produktionen

Eine ε-Produktion ist eine Produktion der Form $A \to \varepsilon$ oder $A \to^* \varepsilon$.
Die Eliminierung erfolgt in zwei Schritten:

1. Suche nach eliminierbaren nichtterminalen Zeichen und fasse sie in einer Menge zusammen.
2. Entferne alle Produktionen, die auf Basis der erstellten Menge im vorherigen Schritt eliminiert werden können.

Beispiel

Betrachte die Grammatik:

$$(S, \{a, b, c\}, \{S, X\}, \{S \to aSc|Sc|X, X \to aXb|Xb|\varepsilon\})$$

Nach der Eliminierung der ε-Produktionen ergibt sich:

$$(S, \{a, b, c\}, \{S\}, \{S \to aSc|Sc|X|ac|c, X \to aXb|Xb|ab|b\})$$

Eliminierung von Einheitsproduktionen

Eine Einheitsproduktion ist eine Produktion der Form $A \rightarrow B$, wobei A und B nichtterminale Zeichen sind.

Eliminierung von Einheitsproduktionen

Eine Einheitsproduktion ist eine Produktion der Form $A \to B$, wobei A und B nichtterminale Zeichen sind.

- Einheitsproduktionen der Form $A \to B$ können nicht entfernt werden.
- Sie werden durch die Ersetzung von B in der Produktion $A \to \alpha_1 | \alpha_2 | \ldots | \alpha_n$ behandelt.
- Dabei wird $B \to \alpha_1 | \alpha_2 | \ldots | \alpha_n$ verwendet.

Eliminierung von Einheitsproduktionen

Eine Einheitsproduktion ist eine Produktion der Form $A \rightarrow B$, wobei A und B nichtterminale Zeichen sind.

- Einheitsproduktionen der Form $A \rightarrow B$ können nicht entfernt werden.
- Sie werden durch die Ersetzung von B in der Produktion $A \rightarrow \alpha_1|\alpha_2|\ldots|\alpha_n$ behandelt.
- Dabei wird $B \rightarrow \alpha_1|\alpha_2|\ldots|\alpha_n$ verwendet.

Beispiel

Betrachte die Grammatik:

$$(S, \{a, b, c\}, \{S\}, \{S \rightarrow aSc|Sc|X|ac|c, X \rightarrow aXb|Xb|ab|b\})$$

Nach der Eliminierung der Einheitsproduktionen ergibt sich:

$$(S, \{a, b, c\}, \{S\}, \{S \rightarrow aSc|Sc|ac|c|aXb|Xb|ab|b, X \rightarrow aXb|Xb|ab|b\})$$

Umwandlung in Chomsky-Normalform

Grammatiken, die durch die vorherigen Schritte bearbeitet wurden, besitzen weder nutzlose Zeichen noch ε-Produktionen noch Einheitsproduktionen.

Umwandlung in Chomsky-Normalform

Grammatiken, die durch die vorherigen Schritte bearbeitet wurden, besitzen weder nutzlose Zeichen noch ε-Produktionen noch Einheitsproduktionen.
Solche Grammatiken enthalten zwei verschiedene Arten von Produktionen:

- $A \rightarrow a$ - diese Produktion gilt als Chomsky-Normalform Produktion.
- $A \rightarrow \alpha_1, \alpha_2, \ldots, \alpha_n$, wobei $\alpha_i \in \Sigma \cup N$ und $n \leq 2$ ist.

Umwandlung in Chomsky-Normalform

Grammatiken, die durch die vorherigen Schritte bearbeitet wurden, besitzen weder nutzlose Zeichen noch ε-Produktionen noch Einheitsproduktionen.
Solche Grammatiken enthalten zwei verschiedene Arten von Produktionen:

- $A \rightarrow a$ - diese Produktion gilt als Chomsky-Normalform Produktion.
- $A \rightarrow \alpha_1, \alpha_2, \ldots, \alpha_n$, wobei $\alpha_i \in \Sigma \cup N$ und $n \leq 2$ ist.
  - Ersetze alle $\alpha_i \in \Sigma$ durch $N_{\alpha_i} \rightarrow \alpha_i$ mit $N_{\alpha_i} \in N$.
  - Ersetze $A \rightarrow \alpha_1, \alpha_2, \ldots, \alpha_n$ durch $A \rightarrow \alpha_1 X_{\alpha_2}$, $X_{\alpha_2} \rightarrow \alpha_2 X_{\alpha_3}$, ..., $X_{\alpha_{n-1}} \rightarrow \alpha_{n-1}\alpha_n$ mit $X_{\alpha_i} \in N$.

Umwandlung in Chomsky-Normalform

Grammatiken, die durch die vorherigen Schritte bearbeitet wurden, besitzen weder nutzlose Zeichen noch ε-Produktionen noch Einheitsproduktionen.
Solche Grammatiken enthalten zwei verschiedene Arten von Produktionen:

- $A \rightarrow a$ - diese Produktion gilt als Chomsky-Normalform Produktion.
- $A \rightarrow \alpha_1, \alpha_2, \ldots, \alpha_n$, wobei $\alpha_i \in \Sigma \cup N$ und $n \leq 2$ ist.
  - Ersetze alle $\alpha_i \in \Sigma$ durch $N_{\alpha_i} \rightarrow \alpha_i$ mit $N_{\alpha_i} \in N$.
  - Ersetze $A \rightarrow \alpha_1, \alpha_2, \ldots, \alpha_n$ durch $A \rightarrow \alpha_1 X_{\alpha_2}$, $X_{\alpha_2} \rightarrow \alpha_2 X_{\alpha_3}$, ..., $X_{\alpha_{n-1}} \rightarrow \alpha_{n-1}\alpha_n$ mit $X_{\alpha_i} \in N$.

Beispiel

Betrachte die Grammatik:
$(S, \{a, b, c\}, \{S\}, \{S \rightarrow aSc|Sc|X|ac|c|aXb|Xb|ab|b, X \rightarrow aXb|Xb|ab|b\})$
Nach der Umwandlung in Chomsky-Normalform:
$(S, \{a, b, c\}, \{S\}, \{S \rightarrow AC|X_1|X_2|AB|AX_2|AX_1|b|c, X \rightarrow XB|AB|AX_1|b, X_1 \rightarrow XB, X_2 \rightarrow SC, A \rightarrow A, B \rightarrow B, C \rightarrow C\})$

Gliederung

CYK-Algorithmus in pseudocode

Der CYK-Algorithmus könnte in pseudocode wie folgt aussehen:

CYK-Algorithmus

Eingabe: Grammatik $G = (S, \Sigma, N, P)$ in Chomsky-Normalform und ein Wort $w = a_1 \dots a_n \in \Sigma^*$

Ausgabe: $\mathrm{w} \in \mathrm{L(G)}$ (falls w von G erzeugt wird), sonst $\notin \mathrm{L(G)}$

(i) for $\mathrm{i} := 1$ to n do

$\quad \mathrm{N_{i,i}} := \{\mathrm{A} \in \mathrm{N} \mid \mathrm{A} \to \mathrm{a_i} \in \mathrm{P}\}$

(ii) for $h := 1$ to $\mathrm{n} - 1$ do

$\quad$ for $i := 1$ to $n - h$ do

$\quad\quad \mathrm{N_{i,i+h}} = \bigcup_{j=i}^{i+h-1} \mathrm{N}_{i,j} \cdot \ \mathrm{N}_{j+1,i+h}$

(iii) if $S \in \mathrm{N_{1,n}}$

$\quad$ then Ausgabe $\mathrm{w} \in \mathrm{L(G)}$

else

$\quad$ Ausgabe w $\notin \mathrm{L(G)}$

CYK-Algorithmus in pseudocode

Die Operation $\cdot$ ist wie folgt definiert:

$N_1 \; \cdot \; N_2$

Sei $G = (S, \Sigma, N, P)$ eine Grammatik in Chomsky-Normalform und seien $N_1, N_2 \subseteq N$, dann gilt:

$$N_1 \; \cdot \; N_2 = \{A \mid \exists B \in N_1, \exists C \in N_2 : A \to BC \in P\}$$

CYK-Algorithmus in pseudocode

Die Operation $\cdot$ ist wie folgt definiert:

$N_1 \cdot N_2$

Sei $G = (S, \Sigma, N, P)$ eine Grammatik in Chomsky-Normalform und seien $N_1, N_2 \subseteq N$, dann gilt:

$$N_1 \cdot N_2 = \{A \mid \exists B \in N_1, \exists C \in N_2 : A \to BC \in P\}$$

	w_1	w_2	w_3	w_4	w_5
w_1	$L_{[1,1]}$	$L_{[1,2]}$	$L_{[1,3]}$	$L_{[1,4]}$	$L_{[1,5]}$
w_2	$L_{[2,1]}$	$L_{[2,2]}$	$L_{[2,3]}$	$L_{[2,4]}$	$L_{[2,4]}$
w_3	$L_{[3,1]}$	$L_{[3,2]}$	$L_{[3,3]}$	$L_{[3,4]}$	$L_{[3,5]}$
w_4	$L_{[4,1]}$	$L_{[4,2]}$	$L_{[4,3]}$	$L_{[4,4]}$	$L_{[4,5]}$
w_5	$L_{[5,1]}$	$L_{[5,2]}$	$L_{[5,3]}$	$L_{[5,4]}$	$L_{[5,5]}$

Komplexitätzeit und Implementierung

Algorithmus 1 Teil (i)

```
for i := 1 to n do                                        %O(n) Durchläufe
    for each A → α ∈ P do                                 %O(|P|) Durchläufe
        if A → α = B → w_i then
            Füge A zur Menge des Listenelements L_[i,i]
        else
            Abbruch (Das Wort w ist nicht ableitbar)
        end if
    end for
end for
```

Komplexitätzeit und Implementierung

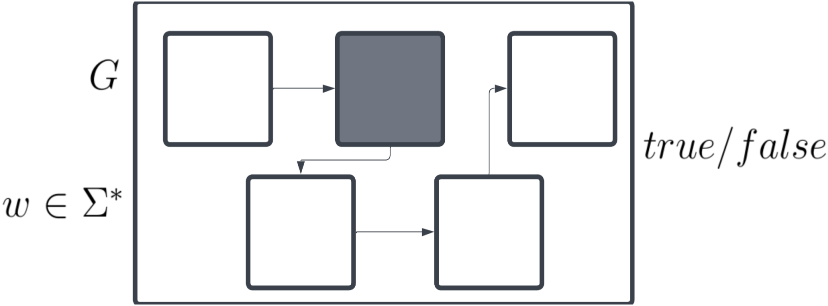
Algorithmus 2 Teil (ii)

for $h := 1$ *to* n **do** — $\%O(n)$ **Durchläufe**
  for $i := 1$ *to* $n - h$ **do** — $\%O(n)$ **Durchläufe**
    for $j := i$ *to* $i + h - 1$ **do** — $\%O(n)$ **Durchläufe**
      for each $A \to \alpha \in P$ **do** — $\%O(|P|)$ **Durchläufe**
        if $|\alpha| = 2 \wedge \alpha_1 \in L_{[i,j]} \wedge \alpha_2 \in L_{[j+1,i+h]}$ **then**
          Füge A zur Menge des Listenelements $L_{[i,i+h]}$
        end if
      end for
    end for
  end for
end for

CYK-Algorithmus Funktionalität durch graphische Effekte.

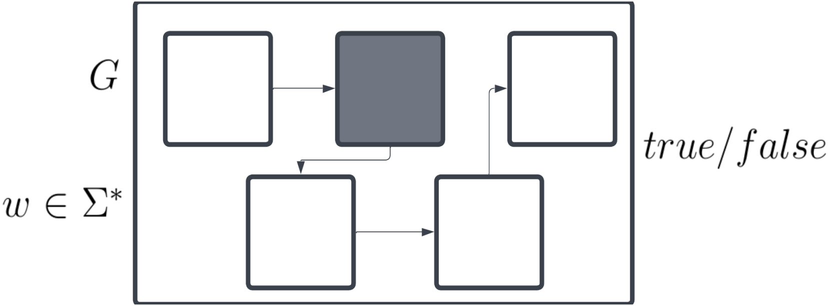
Gliederung

Leslie-G.-Valiant-Algorithmus (LGV)



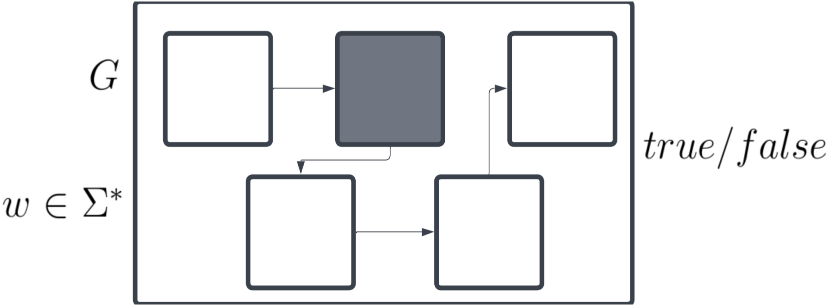
- Eingabe des LGV-Algorithmus:
  - $G = (S, \Sigma, N, P)$ in Chomsky-Normalform.
  - Wort der Länge $n = |w|$.

Leslie-G.-Valiant-Algorithmus (LGV)



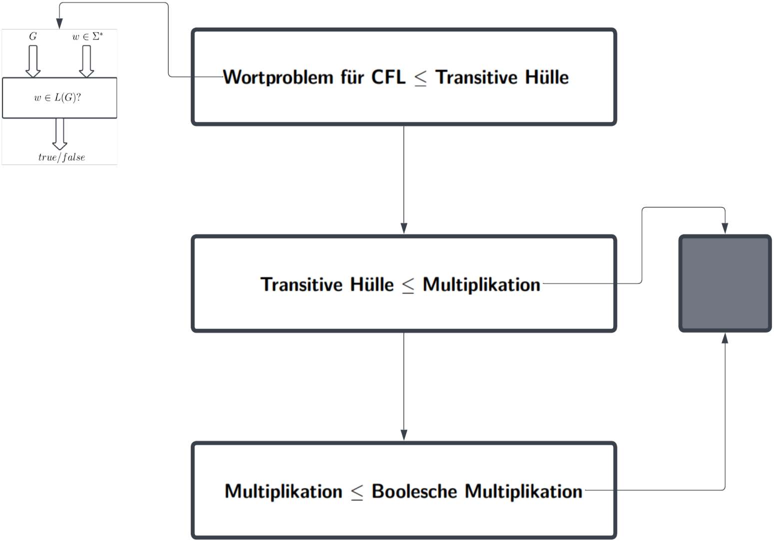
- Eingabe des LGV-Algorithmus:
  - $G = (S, \Sigma, N, P)$ in Chomsky-Normalform.
  - Wort der Länge $n = |w|$.
- Multiplikation von $n \times n$ booleschen Matrizen.
- Verwendung der Strassen-Matrix-Multiplikation.
- LGV-Algorithmus erreicht Zeitkomplexität von $O(n^{2.81})$.

Leslie-G.-Valiant-Algorithmus (LGV)



- Eingabe des LGV-Algorithmus:
  - $G = (S, \Sigma, N, P)$ in Chomsky-Normalform.
  - Wort der Länge $n = |w|$.
- Multiplikation von $n \times n$ booleschen Matrizen.
- Verwendung der Strassen-Matrix-Multiplikation.
- LGV-Algorithmus erreicht Zeitkomplexität von $O(n^{2.81})$.
- Dies kann durch eine Reihe von Reduktionen gezeigt werden.

Leslie-G.-Valiant-Algorithmus (LGV)



Preparation

- Die Definition der Matrixmultiplikation lautet folglich $c = a \cdot b$

$$c_{ik} = a_{i1} \cdot b_{1k} \cup \ldots \cup a_{in} \cdot b_{nk} = \bigcup_{j=1}^{n} a_{ij} \cdot b_{ik}$$

- Die transitive Hülle einer quadratischen Matrize a

$$a^{+} = a^{(1)} \cup a^{(2)} \cup \ldots$$

Preparation

- Die Definition der Matrixmultiplikation lautet folglich $c = a \cdot b$

$$c_{ik} = a_{i1} \cdot b_{1k} \cup \ldots \cup a_{in} \cdot b_{nk} = \bigcup_{j=1}^{n} a_{ij} \cdot b_{ik}$$

- Die transitive Hülle einer quadratischen Matrize a

$$a^{+} = a^{(1)} \cup a^{(2)} \cup \ldots$$

- Dabei wird $a^{(i)}$ wie folgt definiert:

$$a^{(i)} = \bigcup_{j=1}^{i-1} a^{(j)} \cdot a^{(i-j)} \; und \; a^{(1)} = a$$

- Die transitive Hülle für die Matrize a ist endlich.

Preparation

- Ein Hemiring $(H, \cup, \emptyset, \cdot)$
  - H ist die Potenzmenge von N,
  - $\cup$ sowohl kommutativ als auch assoziativ,
  - $\cdot$ weder kommutativ noch assoziativ,
  - $\emptyset$ ist das Nullelement eines Hemirings, d. h.
    - $\emptyset$ ist neutrales Element bezüglich dem Operator $\cup$
    - $\emptyset$ ist absorbierend bezüglich dem Operator $\cdot$

Preparation

- Ein Hemiring $(H, \cup, \emptyset, \cdot)$
  - H ist die Potenzmenge von N,
  - $\cup$ sowohl kommutativ als auch assoziativ,
  - $\cdot$ weder kommutativ noch assoziativ,
  - $\emptyset$ ist das Nullelement eines Hemirings, d. h.
    - $\emptyset$ ist neutrales Element bezüglich dem Operator $\cup$
    - $\emptyset$ ist absorbierend bezüglich dem Operator $\cdot$
- Zeitkomplexitätsfunktionen
  - $R(n)$ für das Lösen des Wortproblems.
  - $M(n)$ für die Multiplikation von $n \times n$ Matrizen.
  - $T(n)$ für das Finden der transitiven Hülle von $n \times n$ oberen Dreiecksmatrizen.
  - $BM(n)$ für das Multiplizieren von $n \times n$ booleschen Matrizen.

Wortproblem für CFL $\leq$ Transitive Hülle

- Sei $G = (S, \Sigma, N, P)$ eine Grammatik in Chomsky-Normalform und $w = w_1 w_2 \ldots w_n$ ein Wort über Σ^*.

Wortproblem für CFL $\leq$ Transitive Hülle

- Sei $G = (S, \Sigma, N, P)$ eine Grammatik in Chomsky-Normalform und $w = w_1 w_2 \dots w_n$ ein Wort über Σ^*.
- Nun wird eine $(n+1) \times (n+1)$ obere Dreieckmatrix b gebildet und wie folgt definiert:

$$b_{i,i+1} = \{A | A \rightarrow w_i \in P\} \; und \; b_{i,j} = \emptyset \; f\ddot{u}r \; alle \; j \neq i+1$$

Wortproblem für CFL $\leq$ Transitive Hülle

- Sei $G = (S, \Sigma, N, P)$ eine Grammatik in Chomsky-Normalform und $w = w_1 w_2 \dots w_n$ ein Wort über Σ^*.
- Nun wird eine $(n+1) \times (n+1)$ obere Dreieckmatrix b gebildet und wie folgt definiert:

$$b_{i,i+1} = \{A | A \to w_i \in P\} \; und \; b_{i,j} = \emptyset \; f\ddot{u}r \; alle \; j \neq i+1$$

- Aus der Definition der Multiplikationsoperation ergibt sich induktiv, dass die Elemente der transitiven Hülle b^+ ausschließlich diejenigen sind, die die Eigenschaft haben, dass

$$A \in b^+_{i,j} \Leftrightarrow A \to^* w_i \dots w_{j-1}$$

Wortproblem für CFL $\leq$ Transitive Hülle

- Sei $G = (S, \Sigma, N, P)$ eine Grammatik in Chomsky-Normalform und $w = w_1 w_2 \dots w_n$ ein Wort über Σ^*.
- Nun wird eine $(n+1) \times (n+1)$ obere Dreieckmatrix b gebildet und wie folgt definiert:

$$b_{i,i+1} = \{A | A \to w_i \in P\} \; und \; b_{i,j} = \emptyset \; f\ddot{u}r \; alle \; j \neq i+1$$

- Aus der Definition der Multiplikationsoperation ergibt sich induktiv, dass die Elemente der transitiven Hülle b^+ ausschließlich diejenigen sind, die die Eigenschaft haben, dass

$$A \in b^+_{i,j} \Leftrightarrow A \to^* w_i \dots w_{j-1}$$

- Es kann also festgestellt werden, ob $S \to^* w$, indem $a = b^+$ berechnet und gefragt wird, ob $S \in a_{1,n+1}$.

Wortproblem für CFL $\leq$ Transitive Hülle

Korrektheit der Reduktion.

- Das Wort w wird vom Startsymbol S abgeleitet $\Longleftrightarrow$
- das Startsymbol S in dem oberen rechten Eintrag $a_{1,n+1}$ der Matrize b^+ zu finden ist.

Wortproblem für CFL $\leq$ Transitive Hülle

Korrektheit der Reduktion.

- Das Wort w wird vom Startsymbol S abgeleitet $\iff$
- das Startsymbol S in dem oberen rechten Eintrag $a_{1,n+1}$ der Matrize b^+ zu finden ist.
- Unter Berücksichtigung des Aufwands für die Erstellung der oberen Dreieckmatrix b entsteht die folgende Zeitkomplexität.

Theorem

$R(n) \leq T(n+1) + O(n^2)$

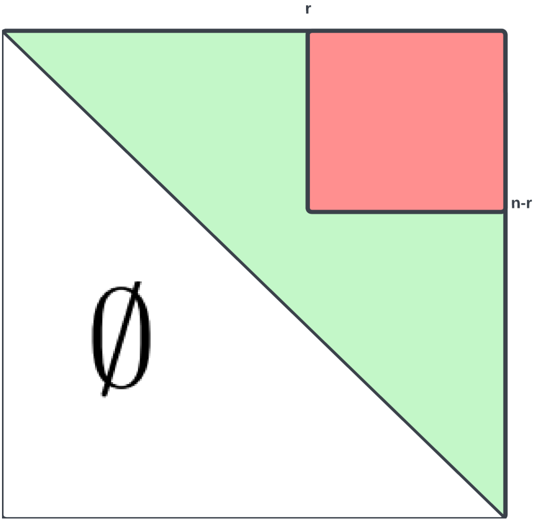
Transitive Hülle $\leq$ Multiplikation

Lemma

Sei b eine obere Dreiecksmatrix der Größe $n \times n$. Unter der Annahme, dass die transitive Hülle der Partitionen $[1 \leq i, j \leq r]$ und $[n - r < i, j \leq n]$ für ein $r > \frac{n}{2}$ bekannt sind, kann der Abschluss von b wie folgt berechnet werden:

1. Durchführung einer einzigen Matrixmultiplikation.
2. Finden der transitiven Hülle einer oberen Dreiecksmatrix der Größe $2(n - r) \times 2(n - r)$, für die die Hülle der Partitionen $[1 \leq i, j \leq n - r]$ und $[n - r < i, j \leq 2(n - r)]$ bekannt sind.

Transitive Hülle $\leq$ Multiplikation



Transitive Hülle $\leq$ Multiplikation

- Ein Term ist die Zusammensetzung von Matrixelementen unter $(\cdot)$
- Die binäre Operation $\cdot$ ist nicht assoziativ
- Terme können verschiedene Assoziationsordnungen haben
- $(2, 11)$-Terme
  - $(b_{2,5} \cdot (b_{5,7} \cdot b_{7,8})) \cdot b_{8,11}$
  - $(b_{2,5} \cdot (b_{5,7} \cdot (b_{7,8} \cdot b_{8,11})))$

Transitive Hülle $\leq$ Multiplikation

- Ein Term ist die Zusammensetzung von Matrixelementen unter $(\cdot)$
- Die binäre Operation $\cdot$ ist nicht assoziativ
- Terme können verschiedene Assoziationsordnungen haben
- $(2, 11)$-Terme
  - $(b_{2,5} \cdot (b_{5,7} \cdot b_{7,8})) \cdot b_{8,11}$
  - $(b_{2,5} \cdot (b_{5,7} \cdot (b_{7,8} \cdot b_{8,11})))$
- Induktiv könnte gezeigt werden, dass
  - $b_{k,l}^{(i)}$ ist die Vereinigung aller formal verschiedenen (k, l)-Terme mit genau i Komponenten
  - $b_{k,l}^{+}$ ist gleich die Vereinigung aller formal verschiedenen $(k, 1)$-Terme
- Lediglich die obere rechte Partition von b^{+} muss berechnet werden

Transitive Hülle $\leq$ Multiplikation

- In jedem (k, l)-Term gibt es einen minimalen (p, q)-Unterterm.
- Jeder dieser minimalen Teilterme hat
  - entweder nur eine Komponente, den (p, q)-Term
  - oder ist die Zusammensetzung von (p, s) (s, q) Terme.

Transitive Hülle $\leq$ Multiplikation

- In jedem (k, l)-Term gibt es einen minimalen (p, q)-Unterterm.
- Jeder dieser minimalen Teilterme hat
  - entweder nur eine Komponente, den (p, q)-Term
  - oder ist die Zusammensetzung von (p, s) (s, q) Terme.
- Die Vereinigung aller möglichen formal verschiedenen minimalen (p, q)-Unterterme ist

$$b_{pq} \cup \bigcup_{n-r<s\leq r} b_{ps}^{+} \cdot b_{sq}^{+}$$

Transitive Hülle $\leq$ Multiplikation

- In jedem (k, l)-Term gibt es einen minimalen (p, q)-Unterterm.
- Jeder dieser minimalen Teilterme hat
  - entweder nur eine Komponente, den (p, q)-Term
  - oder ist die Zusammensetzung von (p, s) (s, q) Terme.
- Die Vereinigung aller möglichen formal verschiedenen minimalen (p, q)-Unterterme ist

$$b_{pq} \cup \bigcup_{n-r<s\leq r} b_{ps}^{+} \cdot b_{sq}^{+}$$

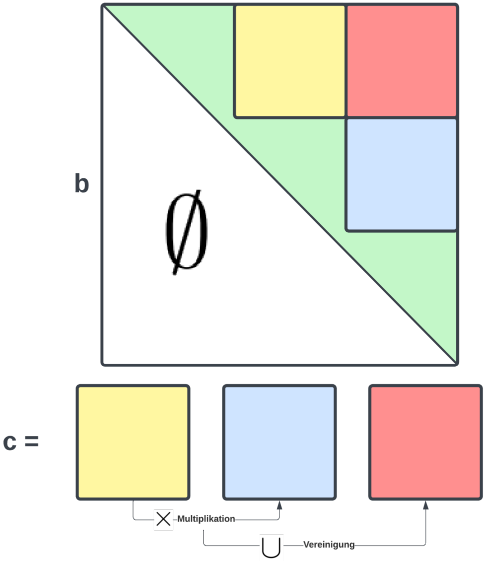
- b_{ps}^{+} und b_{sq}^{+} sind durch die Annahme bekannt

$$c^{'} = b_{ps}^{+} \cdot b_{sq}^{+}$$

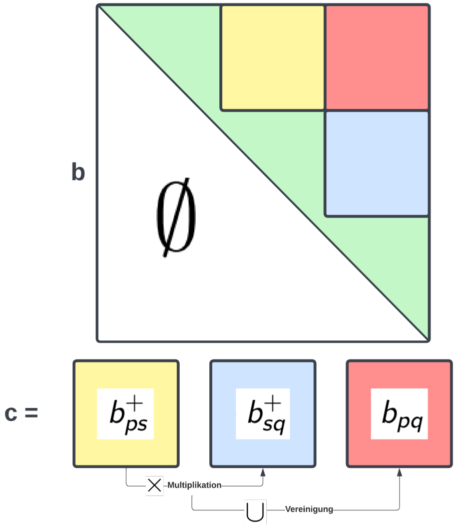
$$c = c^{'} \cup b_{pq}$$

- Matrix c ist gleich der Vereinigung aller formal verschiedenen minimalen (p, q)-Terme von b ist.

Transitive Hülle $\leq$ Multiplikation

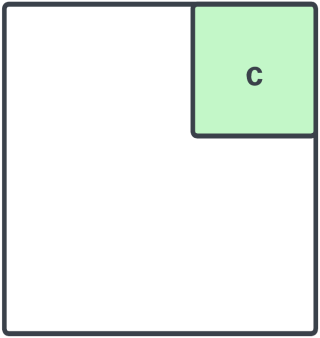


Transitive Hülle $\leq$ Multiplikation



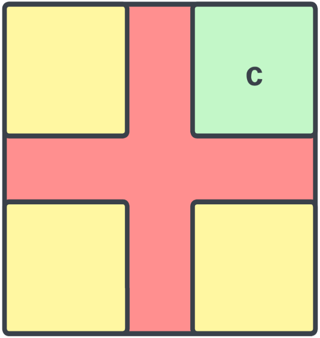
Transitive Hülle $\leq$ Multiplikation

- Ersetze die recht obere Partition von b durch c.



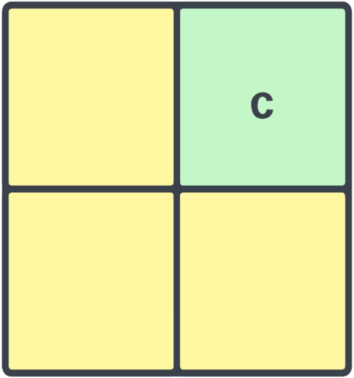
Transitive Hülle $\leq$ Multiplikation

- Markiere alle i-ten Zeilen und i-ten Spalten für $n - r < i \leq r$ in rot.



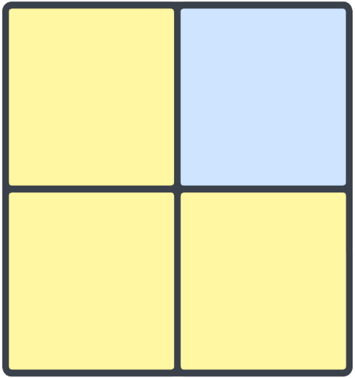
Transitive Hülle $\leq$ Multiplikation

- Eliminiere die rot markierten Zeilen und Spalten.



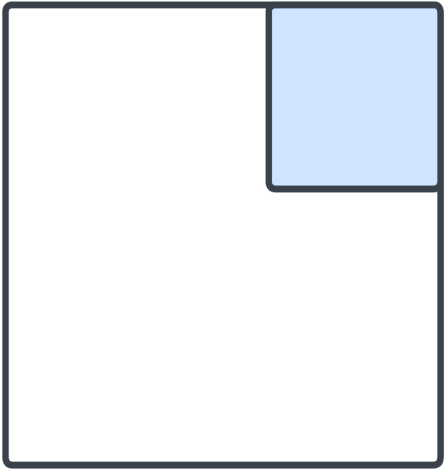
Transitive Hülle $\leq$ Multiplikation

- Berechne die transitive Hülle.



Transitive Hülle $\leq$ Multiplikation

- Ersetze die recht obere Partition von b durch die blau markierte Partition.



Transitive Hülle $\leq$ Multiplikation

Theorem

- $M(n)$ ist die Zeitkomplexität eines Matrixmultiplikationsalgorithmus

$$T(n) \leq M(n) \cdot f(n)$$

- wobei $f(n)$ eine konstante Funktion ist, falls für einige $\gamma > 2$ gilt. Außerdem ist in jedem Fall $f(n) = O(\log n)$.

Transitive Hülle $\leq$ Multiplikation

Theorem

- $M(n)$ ist die Zeitkomplexität eines Matrixmultiplikationsalgorithmus

$$T(n) \leq M(n) \cdot f(n)$$

- wobei $f(n)$ eine konstante Funktion ist, falls für einige $\gamma > 2$ gilt. Außerdem ist in jedem Fall $f(n) = O(\log n)$.

Beweis.

- Sei b eine $n \times n$ obere Dreieckmatrix
- P_k ist die Operation, die ihre Hülle finden soll, wenn jene ihrer folgenden Partitionen bereits bekannt sind
  - $[1 \leq i, j \leq n - \frac{n}{k}]$
  - $[\frac{n}{k} < i, j \leq n]$

Transitive Hülle $\leq$ Multiplikation

Forts. Beweis.

- Für $k = 2,\ 3$ und 4 kann rekursiv wie folgt definiert werden:
- P_2 :
  - (i) Anwendung von P_2 auf die Partition $[\frac{n}{4} < i, j \leq \frac{3n}{4}]$
  - (ii) Anwendung von P_3 auf die Partition $[1 \leq i, j \leq \frac{3n}{4}]$.
  - (ii) Anwendung von P_3 auf die Partition $[\frac{n}{4} < i, j \leq n]$.
  - (iii) Anwendung von $P4$.

Transitive Hülle $\leq$ Multiplikation

Forts. Beweis.

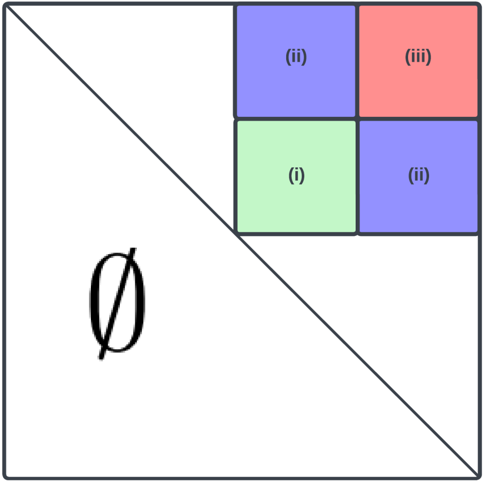
- Für $k = 2,\ 3$ und 4 kann rekursiv wie folgt definiert werden:
- P_2 :
  - (i) Anwendung von P_2 auf die Partition $[\frac{n}{4} < i, j \leq \frac{3n}{4}]$
  - (ii) Anwendung von P_3 auf die Partition $[1 \leq i, j \leq \frac{3n}{4}]$.
  - (ii) Anwendung von P_3 auf die Partition $[\frac{n}{4} < i, j \leq n]$.
  - (iii) Anwendung von $P4$.
- P_3 : Anwendung von dem Verfahren des Lemmas für $r = \frac{2n}{3}$.

Transitive Hülle $\leq$ Multiplikation

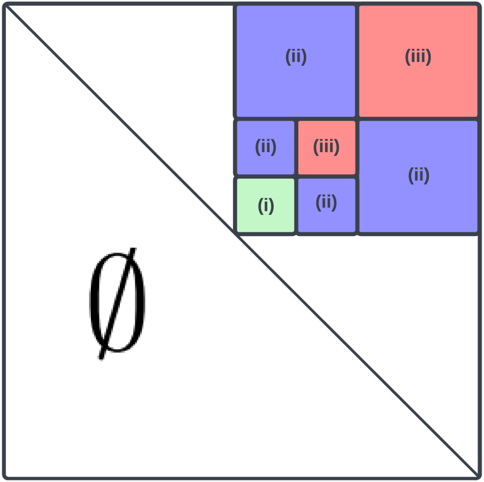
Forts. Beweis.

- Für $k = 2,\ 3$ und 4 kann rekursiv wie folgt definiert werden:
- P_2 :
  - (i) Anwendung von P_2 auf die Partition $[\frac{n}{4} < i, j \leq \frac{3n}{4}]$
  - (ii) Anwendung von P_3 auf die Partition $[1 \leq i, j \leq \frac{3n}{4}]$.
  - (ii) Anwendung von P_3 auf die Partition $[\frac{n}{4} < i, j \leq n]$.
  - (iii) Anwendung von $P4$.
- P_3 : Anwendung von dem Verfahren des Lemmas für $r = \frac{2n}{3}$.
- P_4 : Anwendung von dem Verfahren des Lemmas für $r = \frac{3n}{4}$.

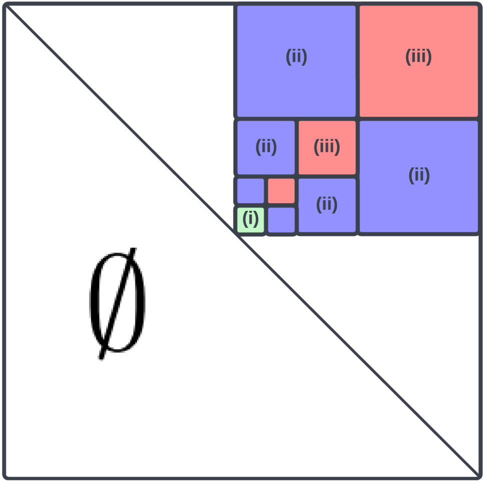
Transitive Hülle $\leq$ Multiplikation



Transitive Hülle $\leq$ Multiplikation



Transitive Hülle $\leq$ Multiplikation



Transitive Hülle $\leq$ Multiplikation

- $T_i(n)$ die Zeitschranke der Prozedur P_i

$$T_2(n) \leq T_2(\frac{n}{2}) + 2T_3(\frac{3n}{4}) + T_4(n),$$

$$T_3(n) \leq M(n) + T_2(\frac{2n}{3}) + O\left(n^2\right),$$

$$T_4(n) \leq M(n) + T_2(\frac{n}{2}) + O\left(n^2\right).$$

Setze T_3 und T_4 in T_2 ein

$$T_2(n) \leq 4T_2(n/2) + 3M(n) + O\left(n^2\right)$$

Transitive Hülle $\leq$ Multiplikation

- $T_i(n)$ die Zeitschranke der Prozedur P_i

$$T_2(n) \leq T_2(\frac{n}{2}) + 2T_3(\frac{3n}{4}) + T_4(n),$$

$$T_3(n) \leq M(n) + T_2(\frac{2n}{3}) + O\left(n^2\right),$$

$$T_4(n) \leq M(n) + T_2(\frac{n}{2}) + O\left(n^2\right).$$

Setze T_3 und T_4 in T_2 ein

$$T_2(n) \leq 4T_2(n/2) + 3M(n) + O\left(n^2\right)$$

- Unter der Annahme, dass n eine Potenz von 2 ist und
- $\gamma \geq 2$ ist Wachstumsfaktor derart, dass für alle m $M\left(2^{\mathrm{m}+1}\right) \geq 2^{\gamma} M\left(2^{\mathrm{m}}\right)$

$$T_2(n) \leq O\left(n^2 \log n\right) + 3M(n) \cdot \sum_{m=0}^{\log n} 2^{(2-\gamma)m}$$

Transitive Hülle $\leq$ Multiplikation

- Aber $\sum_{m=0}^{\infty} x^m$ konvergiert, wenn $|x| < 1$. Wenn also $\gamma > 2$,

$$T_2(n) \leq M(n) \cdot \text{ Konstante}$$

- Wenn $\gamma = 2$, dann

$$T_2(n) \leq M(n) \cdot \log n \cdot \text{ Konstante}$$

Transitive Hülle $\leq$ Multiplikation

- Aber $\sum_{m=0}^{\infty} x^m$ konvergiert, wenn $|x| < 1$. Wenn also $\gamma > 2$,

$$T_2(n) \leq M(n) \cdot \text{ Konstante}$$

- Wenn $\gamma = 2$, dann

$$T_2(n) \leq M(n) \cdot \log n \cdot \text{ Konstante}$$

- Der Abschluss von b, wobei deren Partitionen $[1 \leq i, j \leq \frac{n}{2}]$ und $[\frac{n}{2} < i, j \leq n]$ abgeschlossen sind.

$$T(n) = 2T(\frac{n}{2}) + T_2(n) + O\left(n^2\right).$$

$$T(n) \leq O\left(n^2\right) + T_2(n) \cdot \sum_{m=0}^{\log n} 2^{-m} \leq 2T_2(n) + O\left(n^2\right).$$

Multiplikation $\leq$ Boolesche Multiplikation

Theorem

$M(n) \leq BM(n) \cdot Konstante$

Beweis.

- Sei $G = (S, \Sigma, N, P)$ eine Grammatik in Chomsky-Normalform und seien a und b zwei $n \times n$ Matrizen mit $a_{ij}, b_{ij} \subseteq N$ für $i, j \in \{1, \ldots, n\}$

$$c = a \cdot b$$

Multiplikation $\leq$ Boolesche Multiplikation

Theorem

$M(n) \leq BM(n) \cdot Konstante$

Beweis.

- Sei $G = (S, \Sigma, N, P)$ eine Grammatik in Chomsky-Normalform und seien a und b zwei $n \times n$ Matrizen mit $a_{ij}, b_{ij} \subseteq N$ für $i, j \in \{1, \ldots, n\}$

$$c = a \cdot b$$

- $2|N|$ booleschen Matrizen $a[N_i], b[N_i]$ für $i = 1, \ldots, |N|$ mit N_i bzw. i-ten nichtterminalen Zeichen

$$a[N_i]_{jk} = 1 \; falls \; N_i \in a_{jk}, \qquad b[N_i]_{jk} = 1 \; falls \; N_i \in b_{jk}$$

Multiplikation $\leq$ Boolesche Multiplikation

Forts. Beweis.

Multiplikation $\leq$ Boolesche Multiplikation

Forts. Beweis.

- Für jedes Paar $N_i,\ N_j$ wird dann die boolesche Matrix $c[N_i, N_j]$ für $i.j \in \{1, \dots, |N|\}$ berechnet, wobei

$$c[N_i,\ N_j] = a[N_i] \times b[N_j]$$

Multiplikation $\leq$ Boolesche Multiplikation

Forts. Beweis.

- Für jedes Paar $N_i,\ N_j$ wird dann die boolesche Matrix $c[N_i, N_j]$ für $i.j \in \{1, \ldots, |N|\}$ berechnet, wobei

$$c[N_i,\ N_j] = a[N_i] \times b[N_j]$$

- Dies erfordert $|N|^2$ boolesche Matrixmultiplikationen. Offensichtlich kann c dann direkt erhalten werden, da nach der Konstruktion $N_k \in c_{ij}\ falls$

$$\exists N_i,\ N_j\ sodass,\ c[N_i,\ N_j] = 1\ and\ N_k \rightarrow N_i N_j \in P$$

Gliederung

Strassen

Matrix-Multiplikation

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

Strassen

Matrix-Multiplikation

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

Strassen-Matrix-Multiplikation

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} M_1 + M_2 - M_4 + M_6 & M_4 + M_5 \\ M_6 + M_7 & M_2 - M_3 + M_5 - M_7 \end{bmatrix}$$

Strassen

Matrix-Multiplikation

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

Strassen-Matrix-Multiplikation

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} M_1 + M_2 - M_4 + M_6 & M_4 + M_5 \\ M_6 + M_7 & M_2 - M_3 + M_5 - M_7 \end{bmatrix}$$

- $M_1 = (a_{12} - a_{22}) \cdot (b_{21} + b_{22})$
- $M_2 = (a_{11} + a_{22}) \cdot (b_{11} + b_{22})$
- $M_3 = (a_{11} - a_{21}) \cdot (b_{11} + b_{12})$
- $M_4 = (a_{11} + a_{12}) \cdot b_{22}$
- $M_5 = a_{11} \cdot (b_{12} - b_{22})$
- $M_6 = a_{22} \cdot (b_{21} - b_{11})$
- $M_7 = (a_{21} + a_{22}) \cdot b_{11}$

Implementierung Strassen

Algorithm 3 Strassen-Matrix-Multiplikation

Algorithmus

Input $(A,\ B)$

Output $A \times B$

```
if length(A) = 1, 2, 4, 8, 16, n^2 then        {Basisfall}
    A × B
else                                           {rekursiver Fall}
```

$$\left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right] \quad \left[\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right]$$

Implementierung Strassen

$m_1 = \text{Strassen-Matrix-Multiplikation}((a_{12} - a_{22}),\ (b_{21} + b_{22}))$
$m_2 = \text{Strassen-Matrix-Multiplikation}((a_{11} + a_{22}),\ (b_{11} + b_{22}))$
$m_3 = \text{Strassen-Matrix-Multiplikation}((a_{11} - a_{21}),\ (b_{11} + b_{12}))$
$m_4 = \text{Strassen-Matrix-Multiplikation}((a_{11} + a_{12}),\ b_{22})$
$m_5 = \text{Strassen-Matrix-Multiplikation}(a_{11},\ (b_{12} - b_{22}))$
$m_6 = \text{Strassen-Matrix-Multiplikation}(a_{22},\ (b_{21} - b_{11}))$
$m_7 = \text{Strassen-Matrix-Multiplikation}((a_{21} + a_{22}),\ b_{11})$
$c_{11} = m_1 + m_2 - m_4 + m_6$
$c_{12} = m_4 + m_5$
$c_{21} = m_6 + m_7$
$c_{22} = m_2 - m_3 + m_5 - m_7$
return $\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$
end if

Zeitkomplexität Strassen

- Sei $T_M(n)$ bzw. $T_A(n)$ die Anzahl der Multiplikationen und Additionen.

Zeitkomplexität Strassen

- Sei $T_M(n)$ bzw. $T_A(n)$ die Anzahl der Multiplikationen und Additionen.
- Für $T_M(n)$ gilt:

$$T_M(n) = \begin{cases} 1, & n = 1 \\ 7 \cdot T_M(n/2), & n > 1 \end{cases}$$

Zeitkomplexität Strassen

- Sei $T_M(n)$ bzw. $T_A(n)$ die Anzahl der Multiplikationen und Additionen.
- Für $T_M(n)$ gilt:

$$T_M(n) = \begin{cases} 1, & n = 1 \\ 7 \cdot T_M(n/2), & n > 1 \end{cases}$$

- Daraus ergibt sich:

$$T_M(n) = \underbrace{7 \cdot 7 \ldots 7}_{(\log n)-mal} = 7^{\log n} = n^{\log 7} \leq n^{2.8074}$$

Zeitkomplexität Strassen

- Sei $T_M(n)$ bzw. $T_A(n)$ die Anzahl der Multiplikationen und Additionen.
- Für $T_M(n)$ gilt:

$$T_M(n) = \begin{cases} 1, & n = 1 \\ 7 \cdot T_M(n/2), & n > 1 \end{cases}$$

- Daraus ergibt sich:

$$T_M(n) = \underbrace{7 \cdot 7 \ldots 7}_{(\log n)-mal} = 7^{\log n} = n^{\log 7} \leq n^{2.8074}$$

- Für $T_A(n)$ gilt:

$$T_A(n) = \begin{cases} 0, & n = 1 \\ a18(n/2)(n/2) + 7T_A(n/2), & n > 1 \end{cases}$$

Zeitkomplexität Strassen

- Sei $T_M(n)$ bzw. $T_A(n)$ die Anzahl der Multiplikationen und Additionen.
- Für $T_M(n)$ gilt:

$$T_M(n) = \begin{cases} 1, & n = 1 \\ 7 \cdot T_M(n/2), & n > 1 \end{cases}$$

- Daraus ergibt sich:

$$T_M(n) = \underbrace{7 \cdot 7 \ldots 7}_{(\log n)-mal} = 7^{\log n} = n^{\log 7} \leq n^{2.8074}$$

- Für $T_A(n)$ gilt:

$$T_A(n) = \begin{cases} 0, & n = 1 \\ a18(n/2)(n/2) + 7T_A(n/2), & n > 1 \end{cases}$$

- Damit erhalten wir $T_A(n) = O(n^{2.8074})$.

Boolesche Strassen

- Für die normale Matrix-Multiplikation könnte die boolesche Variante wie folgt aussehen:

$$c_{ij} = \bigvee_{k=1}^{n} a_{ik} \wedge b_{kj}$$

Boolesche Strassen

- Für die normale Matrix-Multiplikation könnte die boolesche Variante wie folgt aussehen:

$$c_{ij} = \bigvee_{k=1}^{n} a_{ik} \wedge b_{kj}$$

- Jedoch ist diese Formel nicht ausreichend für Strassen, da in dessen Berechnung auch Subtraktion involviert ist.

Boolesche Strassen

- Für die normale Matrix-Multiplikation könnte die boolesche Variante wie folgt aussehen:
$$c_{ij} = \bigvee_{k=1}^{n} a_{ik} \wedge b_{kj}$$
- Jedoch ist diese Formel nicht ausreichend für Strassen, da in dessen Berechnung auch Subtraktion involviert ist.
- Die boolesche Matrix-Multiplikation kann durch die Integer-Matrix-Multiplikation simuliert werden. Wenn die Multiplikation abgeschlossen ist, werden die 0-Einträge der Matrix C durch 0 und alle anderen Einträge durch 1 ersetzt.

Zeitkomplexität Boolesche Strassen

Algorithmus 4 Boolean-Strassen

Input $(A,\ B)$
Output $A \times B$

$c =$ Strassen-Matrix-Multiplikation-Algorithmus$(A,\ B)$ $\%O(n^{2.8074})$
c *ist eine* $n \times m$ *Matrix*
for $i := 1$ *to* n **do** $\%O(n)$ **Durchläufe**
  for $j := 1$ *to* m **do** $\%O(m)$ **Durchläufe**
    if $c[i][j] \neq 0$ **then**
      $c[i][j] = 1$
    end if
  end for
end for
return c

Gliederung

CYK-Algorithmus vs. LGV-Algorithmus

1. Test: Grammatiken mit einem nichtterminalen Zeichen und unterschiedlicher Anzahl von Produktionen

CYK-Algorithmus vs. LGV-Algorithmus

1. Test: Grammatiken mit einem nichtterminalen Zeichen und unterschiedlicher Anzahl von Produktionen

- $G_1 = (S, \{a\}, \{S\}, \{S \to SS|a\})$
- $G_2 = (S, \{a, b, c, d\}, \{S\}, \{S \to SS|a|b|c|d\})$

CYK-Algorithmus vs. LGV-Algorithmus

1. Test: Grammatiken mit einem nichtterminalen Zeichen und unterschiedlicher Anzahl von Produktionen

- $G_1 = (S, \{a\}, \{S\}, \{S \rightarrow SS|a\})$
- $G_2 = (S, \{a, b, c, d\}, \{S\}, \{S \rightarrow SS|a|b|c|d\})$

Grammatik / $\|w\|$	$\|N\| = 1$ $\|P\| = 2$		$\|N\| = 1$ $\|P\| = 5$	
	CYK	LGV	CYK	LGV
254	$9.429s$	$14.008s$	$10.658s$	$13.820s$
510	$82.134s$	$81.0641s$	$90.295s$	$79.939s$
1022	$649.915s$	$457.080s$	$717.655s$	$454.532s$
2046	$5192.263s$	$2571.086s$	$5740.248s$	$2585.514s$
4094	$41030.947s$	$14531.633s$	$48343.457s$	$14539.953s$
8190	$\dots s$	$\dots s$	$\dots s$	$\dots s$

CYK-Algorithmus vs. LGV-Algorithmus

2. Test: Grammatiken mit unterschiedlicher Anzahl von nichtterminalen Zeichen

CYK-Algorithmus vs. LGV-Algorithmus

2. Test: Grammatiken mit unterschiedlicher Anzahl von nichtterminalen Zeichen

- $G_3 = (S, \{a, b\}, \{S, B\}, \{S \to SB|BS|a|b, B \to b\})$
- $G_4 = (S, \{a, b\}, \{S, A, B\}, \{S \to SB|SA|a|b, B \to b\ , A \to a\})$

CYK-Algorithmus vs. LGV-Algorithmus

2. Test: Grammatiken mit unterschiedlicher Anzahl von nichtterminalen Zeichen

- $G_3 = (S, \{a, b\}, \{S, B\}, \{S \to SB|BS|a|b, B \to b\})$
- $G_4 = (S, \{a, b\}, \{S, A, B\}, \{S \to SB|SA|a|b, B \to b\ , A \to a\})$

Grammatik / $\|w\|$	$\|N\| = 2$		$\|N\| = 3$	
	CYK	LGV	CYK	LGV
254	$10.382s$	$23.382s$	$12.145s$	$35.979s$
510	$86.694s$	$134.923s$	$97.738s$	$207.755s$
1022	$730.844s$	$771.789s$	$804.681s$	$1180.179s$
2046	$5706.038s$	$4365.122s$	$6311.629s$	$6685.876s$
4094	$46464.972s$	$24703.334s$	$51015.653s$	$37991.584s$
8190	$\ldots s$	$\ldots s$	$\ldots s$	$\ldots s$

3. Test: Grammatik mit vier nichtterminalen Zeichen

CYK-Algorithmus vs. LGV-Algorithmus

3. Test: Grammatik mit vier nichtterminalen Zeichen

- $G_5 = (S, \{a, b\}, \{S, A, B, C\},$
  $\{S \rightarrow AB|AC, C \rightarrow SB, B \rightarrow b\ , A \rightarrow a\})$

CYK-Algorithmus vs. LGV-Algorithmus

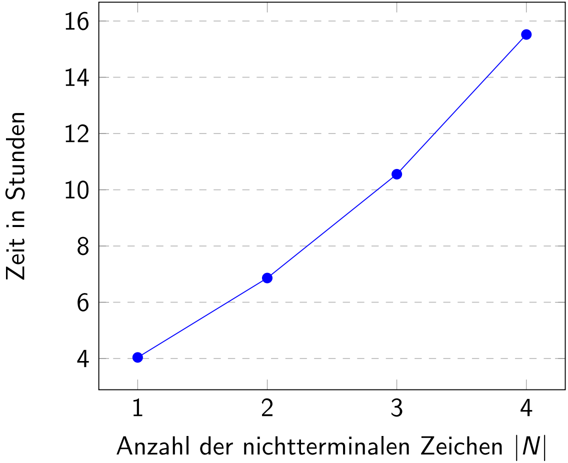
3. Test: Grammatik mit vier nichtterminalen Zeichen

- $G_5 = (S, \{a, b\}, \{S, A, B, C\},$
  $\{S \to AB|AC, C \to SB, B \to b\ , A \to a\})$

Grammatik / $\|w\|$	$\|N\| = 4$ CYK	LGV
254	$12.120s$	$53.519s$
510	$99.725s$	$305.683s$
1022	$824.670s$	$1744.574s$
2046	$6353.308s$	$9999.785s$
4094	$51319.035s$	$55878.540s$
8190	$\ldots s$	$\ldots s$

Einfluss der Nichtterminalen auf den LGV-Algorithmus

- LGV-Algorithmus Verhältnis bzgl. $|N|$ für ein Wort der Länge $|w| = 4094$



Einfluss der Nichtterminalen auf den LGV-Algorithmus

Fundamentalsatz der Zeitergebnisse

Für jede Grammatik mit $|N|$ nichtterminalen Zeichen ein Wort der Länge $|w| = n$ existiert, wo der LGV-Algorithmus den CYK-Algorithmus ab diesem Wort zeitlich besiegt.

Gliederung

Zusammenfassung

Wortproblem für kontextfreie Sprachen in subkubischer Zeit:

Zusammenfassung

Wortproblem für kontextfreie Sprachen in subkubischer Zeit:

- Umwandlung der kontextfreien Grammatiken in Chomsky-Normalform

Zusammenfassung

Wortproblem für kontextfreie Sprachen in subkubischer Zeit:

- Umwandlung der kontextfreien Grammatiken in Chomsky-Normalform
- Der CYK-Algorithmus hat eine Zeitkomplexität von $O(n^3)$

Zusammenfassung

Wortproblem für kontextfreie Sprachen in subkubischer Zeit:

- Umwandlung der kontextfreien Grammatiken in Chomsky-Normalform
- Der CYK-Algorithmus hat eine Zeitkomplexität von $O(n^3)$
- Der LGV-Algorithmus hat eine Zeitkomplexität von $O(n^{2.81})$

Zusammenfassung

Wortproblem für kontextfreie Sprachen in subkubischer Zeit:

- Umwandlung der kontextfreien Grammatiken in Chomsky-Normalform
- Der CYK-Algorithmus hat eine Zeitkomplexität von $O(n^3)$
- Der LGV-Algorithmus hat eine Zeitkomplexität von $O(n^{2.81})$
- Durchgeführte Tests zwischen den beiden Algorithmen auf dem Uni-Cluster

Zusammenfassung

Wortproblem für kontextfreie Sprachen in subkubischer Zeit:

- Umwandlung der kontextfreien Grammatiken in Chomsky-Normalform
- Der CYK-Algorithmus hat eine Zeitkomplexität von $O(n^3)$
- Der LGV-Algorithmus hat eine Zeitkomplexität von $O(n^{2.81})$
- Durchgeführte Tests zwischen den beiden Algorithmen auf dem Uni-Cluster
  - Verwendung großer Wörter mit Längen zwischen 254 und 8190
  - Auswahl von Grammatiken mit unterschiedlicher Anzahl an nichtterminalen Zeichen

Zusammenfassung

Wortproblem für kontextfreie Sprachen in subkubischer Zeit:

- Umwandlung der kontextfreien Grammatiken in Chomsky-Normalform
- Der CYK-Algorithmus hat eine Zeitkomplexität von $O(n^3)$
- Der LGV-Algorithmus hat eine Zeitkomplexität von $O(n^{2.81})$
- Durchgeführte Tests zwischen den beiden Algorithmen auf dem Uni-Cluster
  - Verwendung großer Wörter mit Längen zwischen 254 und 8190
  - Auswahl von Grammatiken mit unterschiedlicher Anzahl an nichtterminalen Zeichen
- Schlussfolgerung der Tests: Fundamentalsatz