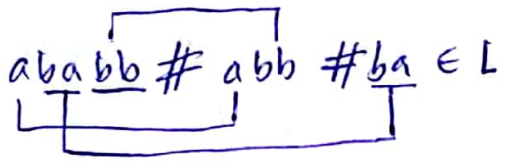
Komplexitätsanalyse

⇒ Wichtig für Klausur!

Schuffle-Membership

$$L = \{ w \# u \# v \mid u, v, w \in \{a,b\}^*, w \in u \sqcup\!\sqcup v \}$$

Z.B. Schuffle



leider funktioniert folgendes nicht:

$$ababb \# bab \# ba \notin L$$

DTIME

Idee: Wir probieren durch. Wir ordnen jedem Buchstaben aus w einen Bit zu, der dafür steht aus welchem Wort (u oder v) dieser Buchstabe kommt.

→ Wir probieren alle Zuordnungen durch

Arbeitsweise für Zuordnung:

- Starte mit $cod(b) = 0^{|w|}$
- Für jede Zuordnung: Prüfe ob Zuordnung funktioniert.

→ Matche die Buchstaben aus w jeweils mit u oder v.

→ Ersetzte gematchte Buchstaben x durch markierte Versionen x'

Was ist Schuffle?

Nimme Buchstaben u und v, sodass wir durch Schieben w bekommen.

Lasse die Reihenfolge u und v wie sie ist.

Wie findet man sowas kleverer?

kein festes Algo vorhanden :(

Wie sieht Zuordnung aus!

Die Zuordnung ist die Funktion

$$b : \{1, \dots, |w|\} \to \{0,1\}$$

Im Beispiel oben:

$cod(b) = 01100$

Wir arbeiten auf einem Band der Form

$w \# u \# v \# cod(b)$

→ Falls Matching nicht funktioniert, entferne alle Markierungen, zähle cod(b) um 1 hoch.
→ Falls passenden Matching gefunden → accept.

DTIME$_2$

Ein zweites Band bringt keine Vorteile, wenn wir einen gleichartigen Algo durchführen.

NTIME Wir können die Idee aus DTIME benutzen, aber die Zuordnung raten!
Dadurch fällt der Faktor $2^{O(n)}$ weg.
→ Wir erhalten eine Laufzeit $O(n^2)$.

NTIME$_2$

Idee: Speichere w auf Band 2, u # v auf Band 1.

- Markiere Buchstaben aus w nicht deterministisch mit 0 oder 1. (für u bzw. v)
- Lauf 2 mal durch Band 2 durch

→ Matche zuerst alles mit 0 markierte mit v (auf Band 1)
→ Danach matche alles mit 1 markierte mit v.

Wie sieht es bei der Laufzeit?

- Initialisierung: $O(n^2)$
- Zuordnung durch das Laufen:
  → $2^{|w|}$ Zuordnungen, es gilt $|w| \approx \frac{n}{2}$
  → $2^{O(n)}$ viele Zuordnungen.
- Matching berechnen: jeweils $O(n^2)$

Insgesamt: $O(n^2) + 2^{O(n)} \cdot O(n^2) = 2^{O(n)}$

Wird es noch besser laufen?

NTIME$_2$

Wie ist Laufzeit?

$O(n)$

DSPACE:

Lösung 1: Kopiere Eingabe auf Band 2.
Führe Algo aus DTIME aus.
⇒ $O(n)$ Speicherplatz.

Lösung 2: Wir müssen nicht das komplette Wort kopieren. Zähle stattdessen bis wohin w, u und v schon markiert wurden (benutzt $O(\log n)$ Speicher)

NSPACE:

Idee: Wir können die Kodierung von b fallen lassen, der wir diese einfach on-the-fly raten können.

Damit bleiben nur noch Zähler übrig der Größe $O(\log n)$ ⇒ $O(\log n)$ Speicherplatz.

Was für ein Problem steht da?
also mit cod(b)?
Eingabe: w # u # v
Band 2: iw # iu # iv # cod(b) #...

cod(b): linear in der Eingabe.
potenziell weitere Zähler

d.h. $O(n)$ Speicherplatz.

Was kann man hier zusammenfassen?

$$L \in \begin{cases} DTIME \in (2^{O(n)}) \\ NTIME \in (O(n^2)) \\ NTIME_2 \in (O(n)) \\ DSPACE \in (O(n)) \\ NSPACE \in (O(\log n)) \end{cases}$$

Noch besser?

$L \in NSPACE(O(\log n)) \xrightarrow{\text{Savitch}}$

$L \in DSPACE(O((\log n)^2))$