Letztes Mal:

- Bestimmung von LCS mit dynamischem Programmieren
- Das Problem der Stringsuche aka. „Strg+F"

$$S = s_1 s_2 s_3 \dots s_n$$
$$t = t_1 t_2 t_3 \dots t_m \qquad m \leq n$$

Ist t in s enthalten?

Naiver Algorithmus Laufzeit $O(mn)$

```
for i = 1 to n-m+1
    if s[i ... i+m-1] = t then
        return i
```

Kann man den Vergleich beschleunigen?

Gibt es eine Möglichkeit s und t kompakt zu repräsentieren, so dass man schnell vergleichen kann?

z.B. könnte man $s[i \dots i+m-1]$ und t Zahlen zuordnen, möglichst zufällig?

⇒ Verwende Hashfunktion h.

```
for i = 1 to n-m+1
    if h(s[i ... i+m-1]) = h(t) then
        if s[i ... i+m-1] = t then
            return i
```

Wenn $s[\dots]$ und t ungleich sind, sollte der Vergleich nun wesentlich schneller gehen (außer es gibt Kollisionen)

Aber: Wir müssen h berechnen. Für $h(t)$ müssen wir das nur einmal tun

Was ist mit $h(s[\dots])$?

Lösung: Wähle h geschickt.

$$h(s[i \dots i+m-1]) = \sum_{j=0}^{m-1} (|\Sigma|^{m-1-j} s_{i+j}) \bmod p$$

→ Interpretiere $s[\dots]$ als Zahl zur Basis $|\Sigma|$, reduziere mod p

Der Trick:

gegeben $h(s[i \dots i+m-1])$, wie findet man $h(s[i+1 \dots i+m])$?

$h(s[i+1 \dots i+m]) = [\,|\Sigma|\, h(s[i \dots i+m-1]) - |\Sigma|^m s_i + s_{i+m}] \bmod p$

Das sind $O(1)$ Operationen!

Wir können also $h(s[1 \dots m]), h(s[2 \dots m+1]), \dots, h(s[n-m+1 \dots n])$ in $O(n)$ Zeit berechnen.

Also Wenn Kollisionen „selten" sind, sollte der Algorithmus $O(m+n)$ Zeit benötigen (Heuristik).

Im worst-case ist die Laufzeit aber immer noch $O(mn)$. Man kann erwartete Laufzeit $O(m+n)$ erreichen, indem man h (bzw $p \in \mathbb{R}$) zufällig im Intervall $1, \dots, n^2 m \log n^2 m$ wählt

Bem: Es gibt Algorithmen, die $O(m+n)$ worst-case Laufzeit erreichen (z.B. KMP oder Suffixbäume) Aber: komplizierter

Interessante Idee: Verwende Hashfunktion als Fingerabdrücke (z.B. Email signieren, Download verifizieren, Passwörter speichern, Filesharing-Dateien identifizieren)

Wie findet man zufällige PZ? Wähle zufällige Zahl, teste ob sie prim ist.

Primzahltests existieren: Miller-Rabin Algorithmus, AKS

Es gilt der Primzahlsatz: Sei $\pi(n) = |\{p \in \mathbb{R} \mid p \leq n\}|$, dann gilt $\lim_{n \to \infty} \frac{\pi(n)}{\frac{n}{\log n}} = 1$

Wörterbuch für Strings

Problem: Wollen eine Menge von Zeichenketten speichern $S = \{s_1, s_2, \dots, s_k\}$, so dass geordnete Wörterbuchoperationen möglich sind.

Lösung: Verwende unsere bekannten Wörterbücher

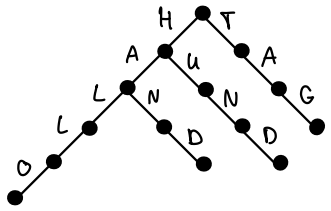
Unschön
- Vergleiche brauchen nicht konstante Zeit (Finden von s braucht $O(|s| \log k)$ Zeit)
- Die spezielle Struktur der Daten wird ignoriert.

→ Verwende ein spezialisiertes Wörterbuch für Strings
→ bessere Laufzeit
→ mehr Struktur → mehr Anwendungen

Ein solches Wörterbuch heißt Trie (retrieval)

Trie:
- Mehrwegbaum
- Jeder Knoten hat 1 bis $|\Sigma|$ viele Kinder
- Die Kanten sind beschriftet mit Symbolen aus Σ
- Die Blätter entsprechen den Wörtern in S
- Die Labels von der Wurzel zum jeweiligen Blatt ergeben das Wort.

Bsp $S = \{HALLO, HAND, HUND, HAUS, TAG\}$



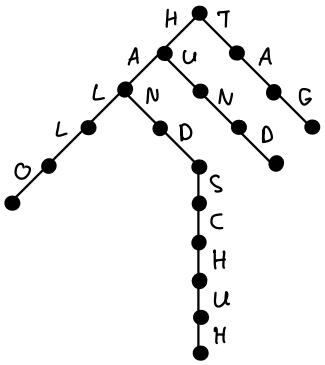
Speicherbedarf: $O(\sum_{i=1}^{k} |s_i|)$

Einfügen, Löschen, Suchen für ein Wort s $O(|s| \cdot |\Sigma|)$

→ besser: $O(|s| \log |\Sigma|)$, wenn wir Kinder für jeden Knoten in einem Wörterbuch speichern.

Problem 1: Was passiert, wenn ein Wort Präfix eines anderen ist?

z.B. füge HANDSCHUH ein.



→ HAND ist verloren gegangen

Fix 1: Markiere Knoten, die Wortenden entsprechen

→ macht Algorithmus kompliziert

Fix 2: Sorge dafür, dass das nicht passiert.

Führe ein spezielles Zeichen € ein (€ ∉ Σ) und hänge es an alle Strings in S

$S = \{HALLO€, HAND€, HUND€, HAUS€, TAG€, HANDSCHUH€\}$

Problem 2: Laufzeit / Platz: zu viele innere Knoten.

→ PATRICIA-Tries

Donald R. Morrison 1968 Practical Algorithms to Retrieve Information Coded in Alphanumeric

Anwendung: Assoziative Arrays