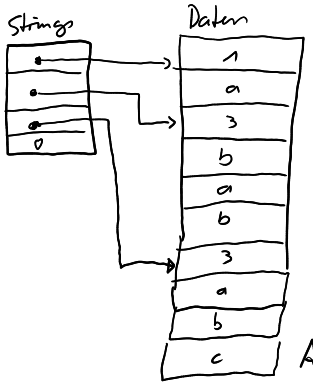
# Wir möchten Strings in Zeit $O(N+\sigma)$ sortieren → Indizes beginnen bei 1!

Gegeben: Menge von Strings $S = \{S_1, \dots, S_k\}$ der Längen $n_i = |S_i|$ über einem Alphabet der Größe $\sigma$.
Wir definieren $N := \sum n_i$ und sagen $n_{max}$ die größte Länge ist

Gesucht: Lex. Sortierung der Strings




„a", „bab", „abc"

Idee: Fangen hinten an (LSD-Radix-Sort) und betrachten nur die Strings, die auch ein entsprechendes Zeichen haben.


Das funktioniert weil kürzere Strings mit gleichem Präfix kleiner sind $a <_{lex} abc$

Algorithmus besteht aus 3 Teilen:
1) Vorverarbeitung für das Alphabet
2) —"— für die Längen der Strings
3) Das eigentliche sortieren

**Schritt 1** - Identifikation der relevanten Zeichen in jeder Iteration
↳ Erzeuge Paare $(l, S_i[l])$ für alle $i \in [1,k]$ und $l \in [1, n_i]$
In unserem Beispiel: (1,a), (1,b), (2,a), (3,b), (1,a), (2,b), (3,c)
→ Laufzeit für das Erzeugen: $O(N)$ Zeit
↳ Sortiere Paare zunächst nach der 2. Komponente und anschließend stabil nach der 1. Komponente → stabil: Wenn 2 Elemente den gleichen Schlüssel haben dann bleibt die Reihenfolge erhalten
Beispiel: 1. (1,a), (2,a), (1,a), (1,b), (3,b), (2,b), (3,c) → $O(N+\sigma)$ Zeit
2. (1,a), (1,a), (1,b), (2,a), (2,b), (3,b), (3,c) → $O(N)$ Zeit
→ Laufzeit für das Sortieren: $O(N+\sigma)$ Zeit
↳ NONEMPTY[l] enthält alle Zeichen, die als l-tes Zeichen in einem String vorkommen
Beispiel: NONEMPTY[1] = a, b
NONEMPTY[2] = a, b
NONEMPTY[3] = b, c
→ Laufzeit für das Erstellen von NONEMPTY ist $O(N)$ Zeit

**Schritt 2** - Identifikation der relevanten Strings in jeder Iteration
↳ Erstelle Array LENGTH[l], welches Pointer auf die Strings der Länge l enthält
Beispiel: LENGTH[1] = „a" ! Nur Pointer, nicht die Strings selber!
LENGTH[2] = ∅
LENGTH[3] = „bab", „abc"
→ Laufzeit: $O(N)$ Zeit da die max. Länge $s_{max}$ und $k$ durch $O(N)$ abgeschätzt werden können

**Schritt 3** - Sortieren
- Erstellen wir QUEUE (leer) für max. $k$ Elemente → $O(N)$ Zeit
- Erstellen Q (leer) für $\sigma$ Pointer → $O(\sigma)$ Zeit
- Für $l = n_{max}$ bis 1 machen → $O(1)$ Zeit
  ↳ Füge LENGTH[l] vorne an QUEUE an
  ↳ So lange QUEUE nicht leer !*
    ↳ Sei $S_i$ das erste Element in QUEUE
    ↳ Entferne $S_i$ aus QUEUE und füge es an $Q[S_i[l]]$ an (ein Zeichen) (*1)
  ↳ Für jedes j in NONEMPTY[l] mache
    ↳ Füge Q[j] hinten an QUEUE an
    ↳ Mache Q[j] leer

(*1) $O(N_e)$ Zeit, wobei $N_e$ die # der Strings ist mit $n_i \geq l$
$O(\sigma_e)$ Zeit, wobei $\sigma_e$ die # Zeich in Länge $l$ sind

Alternativ führe Schleife * unten zunächst für Strings in LENGTH[l] aus und dann für für QUEUE

⇒ Die Schleife wird $\sigma_{max}$ mal ausgeführt

→ $O(\sum_{\ell=1}^{\sigma_{max}} (N_\ell + \sigma_\ell))$ → Wir sehen in Summe max $N$ verschiedene Zeichen

$= O(N)$ → Wir suchen uns die Strings in Summe $N$ mal an

⇒ Der Alg. hat eine Laufzeit von $O(N+\sigma)$.

→ Wir haben alles initialisiert und dann:...

→ starte mit Länge 3 → Füge LENGTH[3] = „bab" und „abc" vorne an QUEUE an

↳ Q[b] = „bab"
Q[c] = „abc" → QUEUE = „bab", „abc"

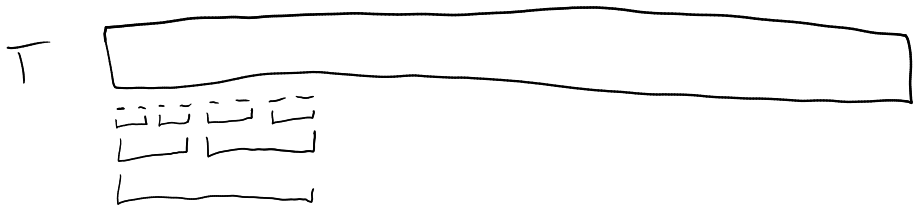
→ weiter mit Länge 2 → Füge LENGTH[2] von QUEUE an

↳ Q[a] = „bab"
Q[b] = „abc" → QUEUE = „bab" „abc"

→ enden mit Länge 1 → Füge LENGTH[1] = „a" vorne an QUEUE an

↳ Q[a] = „a", „abc"
Q[b] = „bab" → QUEUE = „a", „abc", „bab"

Zu Blatt 3




Idee Präfix-Doubling ist wichtig. Nicht die konkrete Variante des Algorithmuses.

## Lambda - Funktionen

```
void foo(...) {                                   ↓ Body
    int d = 12;
    auto add = [d](int a, int b) {                Capture, Parameter
        return a+b; };                            Body
```

class point { int x; int y; } → alternativ Operation überladen

```
};
std::vector<          > data; // voll mit Daten
(std::sort(data.begin(), data.end()); // sortierung nach Default-< vom Datentyp
```

```
std::sort(data.begin(), data.end(),
          [](point& a, point& b) {
              if(a.x == b.x) {
                  return a.y < b.y;
              }
              return a.x < b.x;
          });
```

const, const

(1,2) (1,1) (3,2) (3,3)

↑ std::sort benötigt echt kleiner "<" Vergleich