SA aus der BWT-Rücktransformation

$$T = \overset{0}{A}\overset{1}{N}\overset{2}{A}\overset{3}{N}\overset{4}{A}\overset{5}{S}\overset{6}{\$}$$

Wir wissen wo das letzte Suffix im SA steht

BWT = S $ N N A A A

LF = 6 0 4 5 1 2 3

(*) $LF(i) = j \Leftrightarrow A[i] = A[j] - 1$

lesen → A[0] = 6

schreiben ↓

- LF(0) = 6 → A[6] = 5
- LF(6) = 3 → A[3] = 4
- LF(3) = 5 → A[5] = 3
- LF(5) = 2 → A[2] = 2
- LF(2) = 4 → A[4] = 1
- LF(4) = 1 → A[1] = 0

Mithilfe von (*) und dem Wissen dass $A[0] = n-1$ ist, können wir das SA aus der BWT und dem LF-Mapping konstruieren

	0	1	2	3	4	5	6
A =	6	0	2	4	1	3	5
	$	ANANAS$	ANAS$	AS$	NANAS$	NAS$	S$

Können wir LF-Mapping überschreiben?

Ja, können wir
- ↳ Wir greifen auf jeden Eintrag im LF-Mapping nur einmal zu
- ↳ Der Zugriff erfolgt genau nach dem Schreiben ins SA an die entsprechende Stelle (*)

Wie erstellen wir das LF-Array effizient

↳ Wir möchten nur auf der BWT arbeiten und uns das LF-Array erstellen, wenn wir es benötigen

↳ Die Zeichen behalten die gleiche Reihenfolge in L und F

↳ F können wir aus L generieren indem wir L sortieren

```
        0 1 2 3 4 5 6
BWT =   S $ N N A A A     L
           ↓ stabil(!!) sortieren nach
             dem Zeichen
        1 4 5 6 2 3 0
        $ A A A N N S     F
        0 1 2 3 4 5 6
           ↓ sortiere nach grünen
             Positionen
        0 1 2 3 4 5 6
        S $ N N A A A
        6 0 4 5 1 2 3  → LF-Array
```

Die Text-Reihenfolge in F →

↳ ANANAS$

Konstruktion der BWT

① Konstruiere SA
② Lese SA und schreibe $t_{A[i]-1}$

- - - - - -

alternativ
↳ Konstruiere BWT direkt
z.B. mit DivSufSort

Rücktransformation

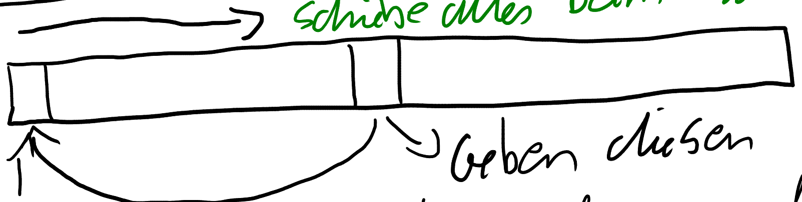
↳ Nutzen Trick von oben
↳ Also ohne LF-Array oder Generierung aller Rotationen.

Wichtig: Das ganze funktioniert nur, so lange wir $\$$ am Ende haben

Alternativ BWST
↳ „Scott"
↳ Immer bijektive Transformation.
Auch ohne $\$$ am Ende

Move-to-Front

schiebe alles beim Suchen um eine Position nach hinten



Geben diesen Index zurück

Wir wissen, dass hier das gesuchte Zeichen rein geschrieben wird

Beim Dekodieren funktioniert es analog

Huffman-Kodierung

T = ANANAS$

BWT = S$NNAAA

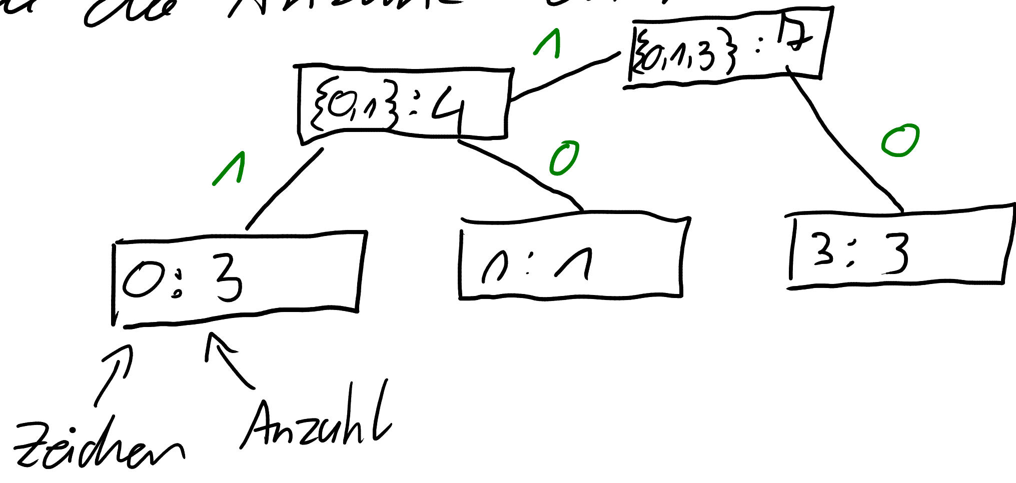
$\overset{0}{\$}\overset{1}{A}\overset{2}{N}\overset{3}{S} \rightarrow S\$AN \rightarrow \$SAN \rightarrow \dots$

MTF = 3130300 (7 · 2 Bit = 14 Bit)

1. Erzeuge Histogram → Zählen die Vorkommen der einzelnen Zeichen

Hist = 3 1 0 3 (Positionen 0 1 2 3)

2. Erzeuge Knoten für jedes Zeichen, die die Anzahl enthalten



3. Verschmelze die beiden Knoten mit den kleinsten Anzahlen

Um die Huffman-Codes für einzelne Zeichen zu bekommen lesen wir die Zeichen auf dem Pfad von der Wurzel zum Knoten:

0 → 11
1 → 10
3 → 0

MTF → 0100110 1111 11 Bit

→ Nachteil: Zum Dekodieren wird der Baum oder ein Wörterbuch benötigt
↳ alternative zum Baum

Canonical-Huffman-Codes

1. Berechne die Länge der Huffman-Codes für die Zeichen
2. Starten wir mit dem Code 0 der Länge 1
   ↳ Immer wenn wir ein neues Zeichen mit der aktuellen Huffman-Code-Länge kodieren wollen addieren wir 1
   ↳ Wenn wir eine längere Länge benötigen, dann adieren wir auch 1 und fügen passend viele 0en an

Länge 1: 3
Länge 2: 1,0

- Start mit 0 → Codewort für 3
- addiere 1 und füge eine 0 an 10 → Codewort für 1
- addiere 1 11 → Codewort für 0

Was ist an diesen Codes besser?
- Codes innerhalb einer Länge sind immer aufsteigend
- Anstelle eines Baumes müssen wir nur die Längen und die Zeichen übergeben
- Für unser Beispiel:

{1, 2} — # der Codes pro Länge

{3, 1, 0} — Zeichen in der Reihenfolge Länge und Zeitpunkt der Kodierung

Huffman-Codes implementieren

- Benutze Priority-Queue
- Speichere Knoten, die wir auch für den Baum erzeugen
- Verschmelzen wie beim Baum

ABER: Wir speichern die aktuelle Tiefe des, denn die brauchen wir für die Canonical-Huffman-Codes

- → Beim Erstellen speichern wir die Code-Wörter in der Reihenfolge der zu kodierenden Zeichen
  - ↳ code-words [. . . abcd.... . . .]
- → Beim Erstellen der Canonical-HC brauchen wir die Reihenfolge der Längen → code-length-order
- → Zum Dekodieren übergeben wir einfach die Tabelle denn die ist für ASCII-Texte klein und so müssen wir nicht neu berechnen.