

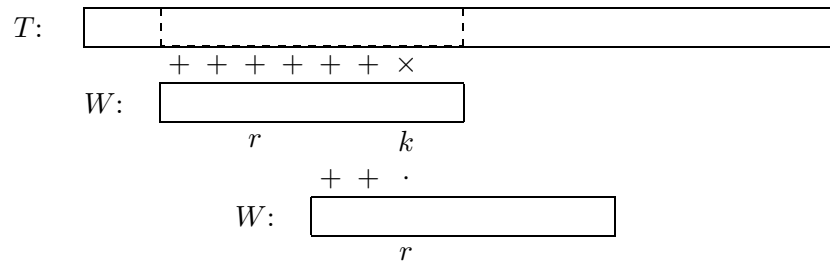
## Patroon herkennen met Knuth-Morris-Pratt

We proberen een woord  $W$  van lengte  $M$  te vinden in een (veel langere) tekst  $T$  van lengte  $N$ . We gaan er hier vanuit dat de  $M$  letters van  $W$  posities  $1, 2, \dots, M$  in  $W$  bezetten, en dat de  $N$  letters van de tekst posities  $1, 2, \dots, N$  in  $T$  bezetten (we beginnen dus niet op positie 0!).

In het naïeve algoritme leggen we voor  $i = 1, 2, \dots, N - M + 1$ , de eerste letter van  $W$  onder positie  $i$  van  $T$ . Vervolgens lopen we van links naar rechts het woord  $W$  af, totdat we een letter treffen die niet gelijk is aan de corresponderende letter in de tekst, of totdat we aan het eind van  $W$  zijn.

Als dit bij de  $k^e$  letter van  $W$  misgaat, is het kennelijk bij de  $k - 1$  voorgaande letters van  $W$  goedgegaan. We weten dan dus wat de corresponderende  $k - 1$  letters in de tekst  $T$  zijn, en we weten dat de huidige letter in de tekst ongelijk is aan de  $k^e$  letter van  $W$ . We kunnen deze kennis gebruiken, door het woord in één keer (eventueel) meerdere posities door te schuiven, en door ná het doorschuiven niet vooraan  $W$  te beginnen met vergelijken van letters.

Voor elke positie in  $W$  bepalen we daarom een *failure-link*: Stel dat de failure-link van positie  $k$  gelijk is aan  $r$ . Dan betekent dat: als het vergelijken van  $W$  met de tekst  $T$  misgaat bij de  $k^e$  letter van  $W$ , dan schuiven we het patroon in een keer  $k - r$  posities naar rechts, en we gaan nu de  $r^e$  letter van  $W$  vergelijken met dezelfde letter in de tekst waar het zojuist misging. We hoeven niet nog apart de letters op posities  $1, 2, \dots, r - 1$  van  $W$  met de corresponderende letters in de tekst te vergelijken, omdat we al weten dat die OK zijn.

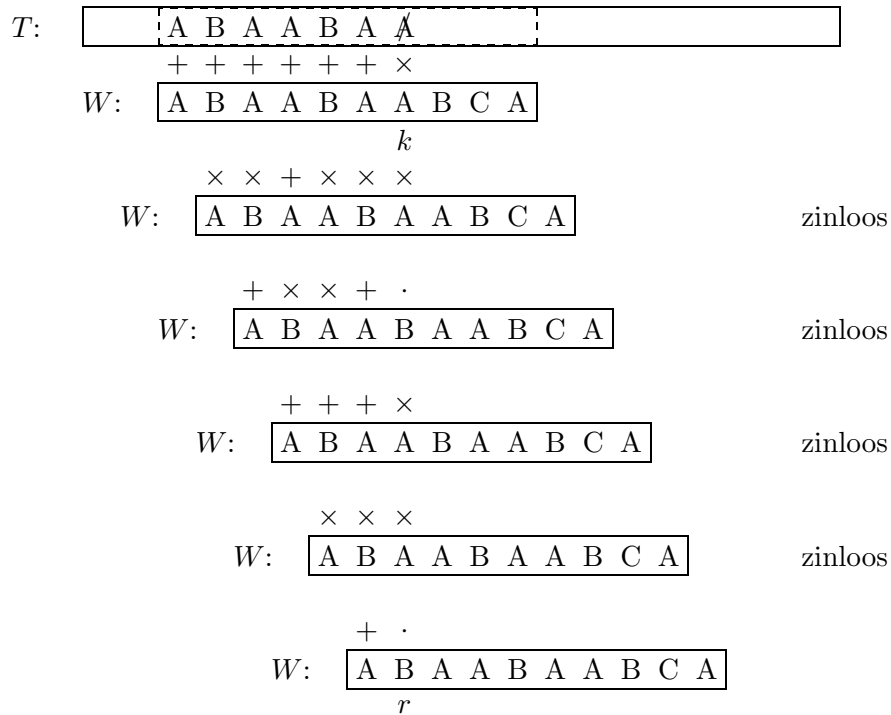


In bovenstaand plaatje gaat het vergelijken van de letters van  $W$  met de tekst mis bij  $k = 7$ . In dit geval is de failure-link voor  $k = 7$  gelijk aan  $r = 3$ . We schuiven  $W$  nu meteen  $k - r = 4$  posities door. We weten kennelijk dat minder posities doorschuiven geen zin heeft, dat  $W[1] = W[5]$ ,  $W[2] = W[6]$  en  $W[3] \neq W[7]$ .

**N.B.:** Dit is een slimme variant van het oorspronkelijke algoritme van Knuth-Morris-Pratt. In het oorspronkelijke algoritme worden bij misgaan van een vergelijking bij positie  $k$  in  $W$  alleen de letters op posities  $1, 2, \dots, k - 1$  gebruikt voor het bepalen van een failure-link. Er wordt dus geen gebruik gemaakt van de wetenschap dat de huidige tekstletter ongelijk is aan  $W[k]$ . Voor onze cursus hoef je alleen de slimme variant te kennen.

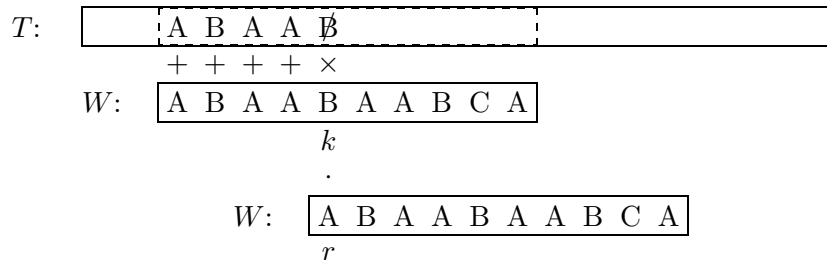
## Voorbeeld

Laat  $W = \text{ABAABAABCA}$ .

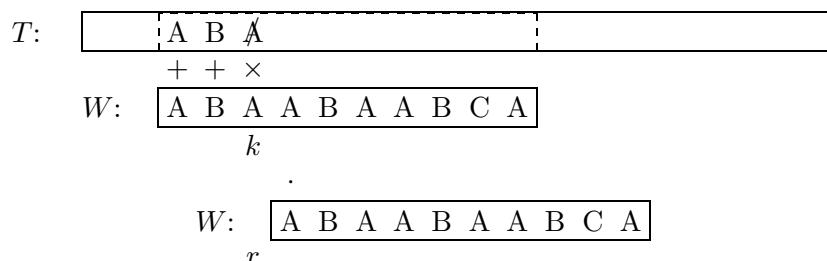


Voor dit woord  $W$  is de failure-link voor  $k = 7$  dus gelijk aan  $r = 2$ . We schuiven  $W$  meteen  $k - r = 5$  posities door, omdat minder doorschuiven geen zin heeft. We hoeven vervolgens niet de eerste letter van  $W$  met de corresponderende tekstletter te vergelijken, omdat we al weten dat die goed moet zijn. We gaan direct de  $r = 2^e$  letter met de huidige tekstletter vergelijken.

Een failure-link die gelijk is aan  $r = 1$  betekent, dat we bij een mislukte vergelijking de eerste letter onder de huidige tekstletter leggen. Bijvoorbeeld voor  $k = 5$ :

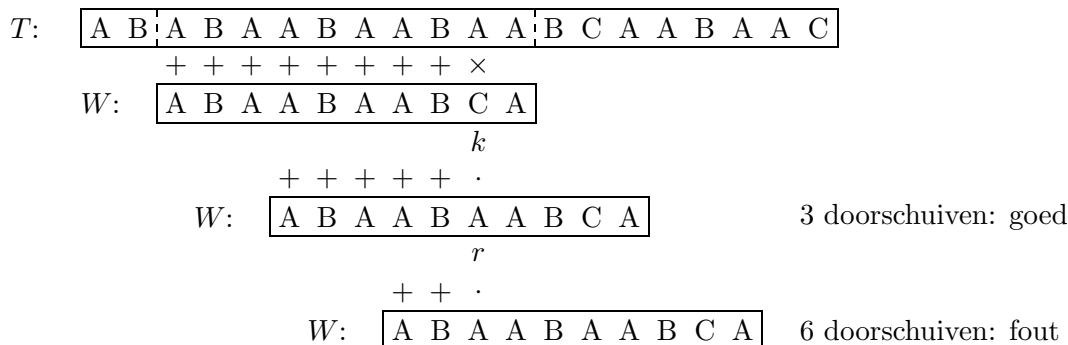


Een failure-link die gelijk is aan  $r = 0$  betekent dat we bij een mislukte vergelijking, het woord  $W$  direct na de huidige tekstletter leggen, *alsof* we de  $0^e$  letter van  $W$  (die natuurlijk niet bestaat) onder de huidige tekstletter leggen. Bijvoorbeeld voor  $k = 3$ :



De failure-link van positie  $k = 1$  is voor alle woorden  $W$  gelijk aan  $r = 0$ .

Pas op dat je het woord  $W$  niet te ver doorschuift. In ons voorbeeld is de failure-link voor  $k = 9$  gelijk aan  $r = 6$ . We moeten het woord  $W$  dus  $k - r = 3$  posities doorschuiven als de vergelijking voor  $W[9]$  misgaat. Als we in plaats daarvan het woord 6 posities doorschuiven, zou dat ook ‘passen’ tot en met de huidige tekstletter, maar dan zouden we een voorkomen van het woord  $W$  over het hoofd kunnen zien, zoals in het volgende voorbeeld:



Als we de failure-link van positie  $k$  afkorten met  $FL[k]$ , dan geldt voor ons voorbeeld:

$$FL[1]=0, FL[2]=1, FL[3]=0, FL[4]=2, FL[5]=1, FL[6]=0, FL[7]=2, FL[8]=1, FL[9]=6, FL[10]=0.$$

### Berekenen failure-links

Het berekenen van de failure-links kun je eenvoudig in  $\mathcal{O}(M^3)$ . We moeten immers voor alle  $M$  posities van woord  $W$  een failure-link berekenen. Voor positie  $k$  zal de failure-link gelijk zijn aan  $0, 1, 2, \dots$ , of  $k - 1$ . In het slechtste geval proberen we  $k$  mogelijke failure-links voor de ene positie  $k$ . Als we de waarde  $r$  proberen als failure-link voor positie  $k$  moeten we controleren dat  $W[1]-W[r - 1]$  netjes gelijk zijn aan  $W[k - r + 1]-W[k - 1]$ , en dat  $W[r] \neq W[k]$ . Dit laatste kost maximaal  $r$  vergelijkingen.

Zo wordt het totale aantal benodigde vergelijkingen voor het berekenen van de failure-links:

$$\sum_{k=1}^M \sum_{r=0}^{k-1} r = \sum_{k=1}^M \frac{1}{2} k(k - 1) = \mathcal{O}(M^3)$$

Het kan ook slimmer: er bestaat een algoritme om de failure-links in  $\mathcal{O}(M)$  tijd te berekenen. Daarmee wordt de totale tijdscomplexiteit van patroonherkennen met het algoritme van Knuth-Morris-Pratt (eerst berekenen failure-links, en vervolgens zoeken van het woord  $W$  met behulp van de failure-links):  $\mathcal{O}(M + N) = \mathcal{O}(N)$ . Dat is beter dan de tijdscomplexiteit (voor het slechtste geval) van patroonherkennen met het naïeve algoritme, die  $\mathcal{O}(M \cdot N)$  is.