Hoofdstuk 12

Indexeren van Vaste Tekst

Sommige zoekoperaties op strings gebeuren op een *vaste tekst T*, met lengte *t* waarin frequent moet gezocht worden naar een *patroon P*, met lengte *p*. Best doe je dan voorbereidend werk op T, zodat het zoeken efficiënter loopt.

We tonen aan dat *O(t)* voorbereidingswerk zoeken verbeterd tot *O(p)*.

Als voorbereidend werk slaat men alle **suffixen** van de tekst op in een gegevensstructuur. *Als een patroon voorkomt, dan moet het een prefix zijn van één van die suffixen*. We zijn dus op zoek naar een gegevensstructuur die toelaat snel die prefixen te vinden. Dit is niet evident, want er zijn *t* suffixen, met in totaal $O(t^2)$ karakters.

We duiden de suffix die begint op locatie *i* aan als $suff_i$.

12.1 Suffixbomen

Een meerwegstrie ligt voor de hand. De zoektijd wordt beperkt tot sleutellengte *p*, onafhankelijk van het aantal suffixen *t*. Suffixen één voor één toevoegen vereist $O(t^2)$. Een meerwegstrie neemt ook veel geheugen in. We gebruiken dus géén meerwegstrie.

We gebruiken wel een **suffix tree**. Deze is *verwant aan een meerwegs patriciatrie*. Bij Patricia worden alle knopen met één kind geëlimineerd, het aantal inwendige knopen wordt dan *O(t)*.

De wijzigingen ten opzichte van Patricia:

1. Bij Patricia werden strings bij de bladeren opgeslagen, nu kunnen we de tekst als referentie nemen. In plaats van de volledige $suff_i$ op te slaan, **houden we enkel beginindex *i* bij**.
2. Bij Patricia bewaarden we de testindex in elke knoop. Nu houden we een **begin- en een eindindex bij in elke knoop**. We weten dan meteen of verdergaan nog zin heeft.
3. In elke inwendige knoop kan een **staartpointer** opgenomen worden die de opbouw en sommige toepassingen veel sneller maakt.

De *staart* van een string is de string bekomen door het eerste karakter te verwijderen. We noteren de staart van string *s* als *staart(s)*.

Logischerwijs zit dit ook in de boom, de trie heeft een zeer "repeterend" karakter.

We weten dat een trie problemen oplevert als er een string is die een prefix is van een ander. Een afsluitteken gebruiken werkt hier niet. We starten immers met een lege suffixboom. Stel nu dat er een suffix *a* bestaat die een prefix is van een andere suffix *ab*, dan is de volgende suffix *staart(a)*, en die is op zijn beurt een prefix van *staart(ab)*, die ook in de suffixboom zit.

Na *k* iteraties is er dus een kleinste index *i*, zodat $suff_i$ een prefix is van een vorige suffix. Dan geldt:

1. Alle suffixen *suffj* waarbij $j < i$ geen prefix worden aangeduid door een blad.
2. Alle suffixen *suffj* waarbij $j \geq i$ prefix zijn van een andere string en dus niet zichtbaar zijn in de suffxiboom. $Suff_i$ noemen we dan de *actieve suffix*.

Als we nu *T[k]* toeveogen moeten we alle strings in de suffixboom verlengen met dit *k-de* karakter en de karakterstring *"T[k]"* toevoegen, tenzij dit karakter al eerder in de string zat.

1. Voor suffixen aangeduidt door een blad moeten we niks doen, in het blad staat de beginindex van de string in T, daardoor gaat de verlenging automatisch.
2. Dan gaan we de strings af die een prefix waren, in orde van dalende lengte. Als ze door dit extra karakter geen prefix meer zijn, moeten ze een blad krijgen, waarna we naar de volgende suffix gaan. Vinden we een suffix die met dat extra karakter terug een prefix is dan kunnen we stoppen, want alle kortere suffixen zijn dan ook een prefix.

Als het laatste karakter van T nergens anders voorkomt in de tekst (stopkarakter), dan hebben alle suffixen een blad gekregen en zitten ze expliciet in de suffixboom.

We moeten deze operatie (blad toevoegen en naar volgende impliciete suffix springen) *O(1)* houden. Daarvoor houden we een pointer bij naar de laatst toegevoegde inwendige knoop, en een pointer naar het *actieve punt*, de laatste expliciete inwendige knoop die we tegenkomen als we het actieve suffix zoeken in de suffixboom.

De actieve suffix behandelen gaat als volgt:

1. Onderzoek of de inwendige knoop waar de suffix eindigt een uitgang heeft voor T[j]. Indien ja, dan is het verlengde actieve suffix nog steeds een prefix en blijft het actieve suffix. Het is mogelijk dat het actieve punt wel verlegd moet worden.
2. Is rer geen uitgang, maak de impliciete knoop expliciet en hang er een blad aan. Daarna wordt het volgende suffix actief. Via de staartpointer van het actieve punt springen we naar dit volgende suffix.
3. We maakten impliciete knoop *I* expliciet, we moeten dan nog de staartpointer invullen. Die moet wijzen naar de knoop waar het nieuwe actieve suffix eindigt. Ofwel is die knoop al expliciet, ofwel een impliciete knoop *J*, maar dan heeft hij maar één uitgang, dezelfde als *I*. *I* had geen uitgang voor T[j], *J* dus ook niet. Ook *J* zal expliceit gemaakt worden bij het verwerken van de volgende actieve suffix.

Bij het overspringen naar de volgende actieve suffix moeten we gebruik maken van de staartpointer van de *laatste oude expliciete knoop* op het pad van de huidige suffix. Als dit de wortel is, blijven we in de wortel. Als we dus een nieuwe expliciete knoop aanmaken verleggen we het actieve punt niet. Als w ein een oude expliciete knoop komen, dan verleggen we het actieve punt wel, want de staartpointer is al ingevuld.

Toepassingen:

- **Klassieke deelstringprobleem**. Gevraagd: alle beginposities van P in T.
- **Langste gemeenschappelijke deelstring**. Verzameling S van *k* strings met totale lengte *t*. Gezocht: de langste gemeenschappelijke deelstring van al die strings.
  We gebruiken hier een *veralgemeende suffixboom*. Die bevat alle suffixen voor alle strings. Zijn bladeren bevatten zowel beginpositie, als de string waartoe hij behoort. Hetzelfde suffix kan tot meerdere strings behoren. De boom wordt overlopen om de lengte van al de prefixen en het aantal verschillende strings te bepalen waarin ze voorkomen.

TL;DR - Een suffixboom slaat alle suffixen van een tekst T op. De bladeren bevatten de beginindex (in de tekst) van een suffix. De inwendige knopen slaan een begin- en eindindex op. In elke inwendige knoop kan ook een staartpointer opgeslagen worden.

12.2 Suffixtabellen

Een suffixtabel is een *eenvoudiger* alternatief voor een suffixboom. Zelfde operaties, zelfde performantie, vereist minder geheugen.

Het is eigenlijk gewoon een tabel met gerangschikte suffixen van tekst T. De tabel bevat natuurlijk enkel de beginindices.

Een suffixtabel construeren is eenvoudig, tenzij men dat efficiënt wil doen (rangschikken neemt veel tijd in beslag). De klassieke oplossing construeert eerst de suffixboom en bekomt daarna de rangschikken door hem inorder te overlopen.

Er is een belangrijke **hulpstructuur** die nuttig is bij het gebruik van een suffixtabel. De **LGP-tabel**, langste gemeenschappelijke prefix. LGP[i] is de lengte van het langste gemeenschappelijke prefix van $suff_i$ met zijn opvolger in alfabetisch volgorde.

Al de rest is super ingewikkeld en komt wss niet voor op de test.

12.3 Tekstzoekmachines

Artikel, dus niet te kennen voor test.