Universiteit van Amsterdam

Technisch rapport

Auto Complete

21 maart 2015

Docent:
Floris Kroon

Cursus:
Datastructuren

Vakcode:
5062DATA6Y

Student:
Tijmen Zwaan
10208917

1 Introductie

Het doel van deze opdracht is om een automatische aanvulfunctie te maken voor woorden, gebaseerd op een woordenboek dat wordt opgebouwd uit een gespecificeerde tekst. Het moet daarbij ook mogelijk zijn om woorden uit het woordenboek te verwijderen. Het de bedoeling dat een trie-datastructuur gebruikt wordt.

1.1 Definities

- Node - Een punt in een netwerk of diagram waar lijnen of paden kruisen of splitsen.
- Edge - De connectie tussen twee nodes.
- Child - Een node die een extensie is van een andere node.
- Parent - De tegenovergestelde relatie als een child node.
- Recursief proces - Een zichzelf herhalend proces.
- Sequentieel proces - Een proces dat een aantal stappen doorloopt in een vaste volgorde.

2 Methode

De 'trie' datastructuur lijkt op een normale 'tree' datastructuur. Echter wordt er geen data in de nodes opgeslagen, maar in de edges van de trie. Zo

zullen alle woorden beginnend met dezelfde letter de children zijn van slechts één node. De edge van die node heeft daarbij de waarde van de letter, en de node bevat alleen nieuwe edges naar de volgende letters van de verschillende woorden.

2.1 Algoritme

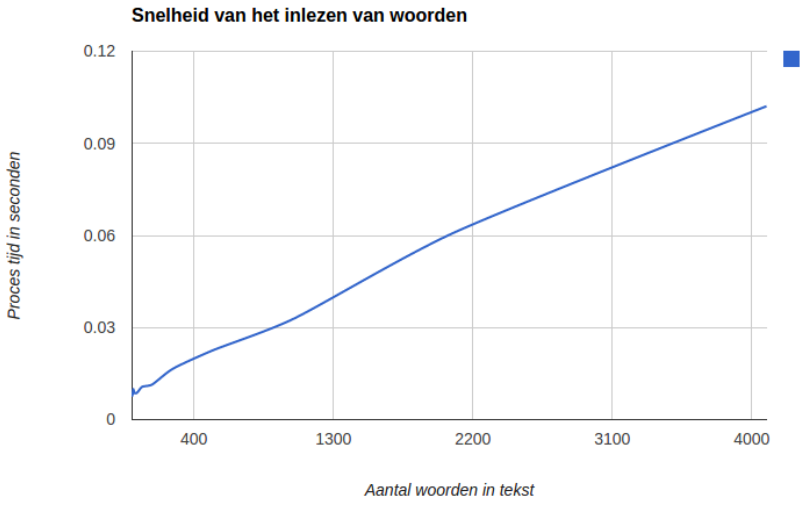
Omdat iedere node in de trie zich hetzelfde gedraagt als zowel zijn children als zijn parents zijn veel delen goed aan te pakken op een recursieve manier. Omdat echter na testen bleek dat een sequentiele aanpak efficienter was voor het toevoegen van woorden is dit dus niet overal het geval.

Bij het toevoegen van een woord wordt eerst de start-node de actieve node. Er wordt dan gekeken naar de eerste letter van het woord. Als deze al bestaat als edge in de actieve node, wordt die node de nieuwe actieve node en gaat het algoritme door naar de volgende letter. Als deze nog niet bestaat, wordt de node aangemaakt en toegevoegd aan de actieve node. Daarna wordt de nieuw aangemaakte node de actieve node.
Wanneer het eind van het woord bereikt is wordt in de actieve node de zogenaamde 'end-flag' op 'true' gezet. Wanneer later door de trie wordt gegaan zijn alle nodes waar deze 'end-flag' op 'true' staan woorden.

Binnen elke node bestaat er een lijst met de karakters die zijn toegevoegd als children van de node. Daarnaast is er ook een lijst met de bijbehorende child-nodes. Deze lijsten staan in dezelfde volgorde. Wanneer een specifiek child wordt gezocht, wordt er gezocht in de eerste lijst van karakters. Wanneer het juiste karakter is gevonden weet het programma dus ook op welke plek de node in de andere array staat.
Deze arrays worden vergroot wanneer ze vol zijn, en er moet een nieuw element bij.

3 Resultaten

Om te testen hoe efficient dit algoritme is, heb ik het programma op teksten van verschillende groottes losgelaten. Uit de grafiek hieronder is duidelijk zichtbaar dat de tijd dat het programma bezig is lineair evenredig is aan het aantal woorden in de input.
Als hier een zogenaamde linked-list datastructuur was gebruikt zoals in de vorige opgave, was dit exponentieel gebleken. Dit komt doordat bij een linked-list iedere keer door alle woorden gelopen moet worden tot het juiste woord is bereikt. Bij de trie structuur hangt de tijd per woord af van het aantal letters in het woord, niet van het aantal voorgaande woorden.



4 Discussie

In mijn eerste implementatie van dit probleem had ik besloten om in iedere node een array van 26 pointers neer te zetten. Eén pointer voor iedere letter van het alphabet. Hierdoor is een pointer 'NULL' wanneer de edge niet bestaat en anders wijst hij naar de juiste node. Dit zorgt ervoor dat er binnen een node niet gezocht hoeft te worden naar de edges. Echter limiteert dit het woordenboek tot slechts 26 tekens. Dit is in de huidige implementatie niet zo wat betekent dat alle karakters toegevoegd kunnen worden, maar er dus wel extra tijd nodig is om binnen een node te zoeken naar de correcte child-node.