Algoritmen en Datastructuren 2: Union-Find

Youri Coppens
Youri.Coppens@vub.be

Opmerking

Voor de WPO's maken we gebruik van R7RS en heb je de code van de cursus nodig: download de `a-d` folder op Canvas. Installeer de `a-d` folder als package via de DrRacket Package Manager of voeg deze folder toe aan DrRacket's Collections Path. Verdere instructies vind je in het bestand `r7rs.rkt` op Canvas. Een beschrijving van R7RS vind je eveneens terug op Canvas. Relevante bestanden voor dit WPO vind je in `a-d/disjoint-sets`.

1 Enkel rangen van leiders bijhouden

In de geöptimaliseerde up-tree implementatie van het `disjoint-sets` ADT maken we gebruik van 2 vectoren: één om de up-tree bij te houden en één om de rangen bij te houden. Echter zijn we enkel geïnteresseerd in de rangen van de leider (**motiveer waarom!**). Daardoor zouden we de vector met rangen kunnen weggooien en de rang van de leider bijhouden in de up-tree zelf. Deze bevat namelijk een "nutteloze"pointer naar zichzelf op de plaats van de leider. Hoe kan je er dan voor zorgen dat `find!` nog steeds een leider kan herkennen? Maak een kopie van `optimized.rkt` en wijzig het ADT zodat deze slechts 1 vector hoeft bij te houden.

2 Rangen vervangen door gewichten

In de geöptimaliseerde up-tree implementatie van het `disjoint-sets` ADT maken we gebruik van *rangen* om de hoogte van een up-tree bij te houden. Bij `union!` wordt telkens een boom met een lagere rang onder een boom met een hogere rang gehangen. Een alternatieve manier is om het *gewicht* van een boom bij te houden. Het gewicht van een boom is het totale aantal nodes in die boom.

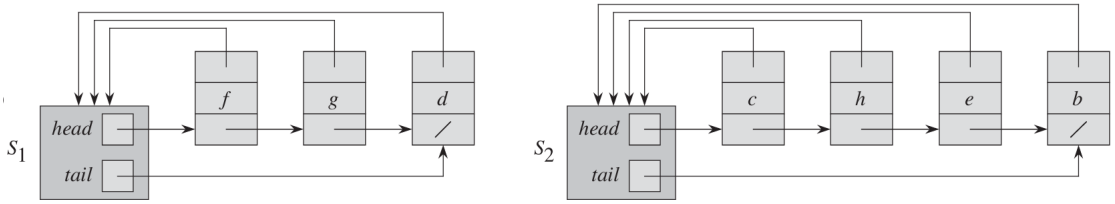
- Pas het `disjoint-sets` ADT aan zodat elke node zijn gewicht bijhoudt. Maak hiervoor een kopie van het originele bestand, `optimized.rkt`.

- Pas de implementatie van `union!` aan zodat de gewichten correct worden aangepast.
- Pas de implementatie van `find` aan zodat de gewichten correct worden aangepast tijdens de padcompressie.
- Bij de originele implementatie met rangen hebben we nooit de rangen aangepast tijdens de padcompressie. Is dit een probleem? Motiveer je antwoord.

3 Disjoint Sets met gelinkte lijsten

De geziene implementatie van het `disjoint-sets` ADT onderhoudt een bos van uptrees in een vector. Er bestaat echter ook een variant die gebruikt maakt van gelinkte lijsten. De idee is om elke set voor te stellen als een enkelgelinkte lijst die een referentie bijhoudt naar zijn laatste element. Bovendien zal elke node in deze lijst ook een referentie bijhouden naar het eerste element van de lijst. Het eerste element van de lijst kan dus telkens gevonden worden in $O(1)$. Hierdoor kunnen we het eerste element van de lijst beschouwen als de leider van de set.

- Beschrijf de implementatie van `union!` die gebruik maakt van deze datastructuur. Toon aan dat deze implementatie in $O(n)$ is. Wat zal de performantie van `find` zijn?
- Vervolledig de code van de meegeleverde bestanden, `uf-linked-list.rkt` en `oefening3.rkt`. Welk voordeel haal je uit de gelinkte versie die je niet in de vorige implementaties had?



Figuur 1: Voorbeeld van 2 sets als gelinkte lijsten. Overgenomen uit *Introduction to algorithms* (3e editie, p. 565), door T. H. Cormen, C. E. Leiserson, R. L. Rivest en C. Stein, 2009, Cambridge, MA: MIT Press. Copyright 2009 door Massachusetts Institution of Technology.