

Algoritmen en Datastructuren 2: Ongerichte Graafalgoritmen

Youri Coppens
Youri.Coppens@vub.be

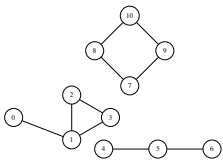
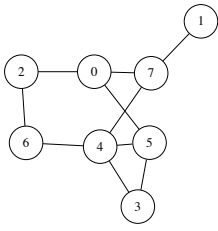
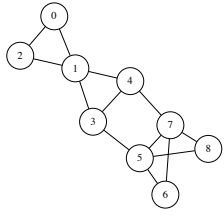
Opmerking

Tijdens dit WPO zullen we niet alle ongerichte graafalgoritmen uit de les bespreken. Deze hebben echter wel allemaal een implementatie in Scheme. Al deze algoritmen zijn terug te vinden in de folder `a-d/graph-algorithms/undirected/`. Hier volgt een kort overzicht van waar je alle algoritmen terug kan vinden:

- De file `dft-applications.rkt` bevat `cyclic?` en `exist-path?`. *Waarom zijn dit goede applicaties voor DFT?*
- De file `bft-applications.rkt` bevat `shortest-path` en `distance`. *Waarom zijn dit goede applicaties voor BFT?*
- De file `connectivity.rkt` bevat `connected-components/dft`, `connected-components/bft`, `edge-connected-components`, `biconnected-components`, `bipartite/dft` en `bipartite/bft`.

1 Boogsamenhangendheid

In de file `a-d/graph-algorithms/undirected/connectivity.rkt` vind je een **DFT-gebaseerd algoritme** voor het vinden van **bruggen** in een **ongerichte graaf**. Deze functie, `edge-connected-components`, geeft ons nu enkel de lijst van bruggen. Het zou echter ook nuttig zijn om het **aantal boogsamenhangende componenten** te kennen. Tevens kunnen we met deze implementatie niet achterhalen tot **welke** van die componenten individuele knopen behoren. De opdracht is dus om het algoritme zodanig aan te passen dat het ook het **aantal** boogsamenhangende componenten en het **lidmaatschap** van de knopen tot de componenten teruggeeft. Het gewenste resultaat voor de grafen `three-cc`, `connected` en `kite` is te zien in Tabel 1.

| | | | |
|---------------------------|---|--|---|
| Graaf |  |  |  |
| Bruggen | '((4 . 5) (5 . 6) (0 . 1)) | '((7 . 1)) | '() |
| Aantal Componenten | 6 | 2 | 1 |
| Lidmaatschap | #(1 2 2 2 3 5 4 6 6 6 6) | #(1 2 1 1 1 1 1 1) | #(1 1 1 1 1 1 1 1 1) |

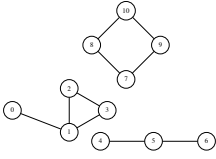
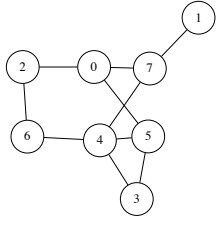
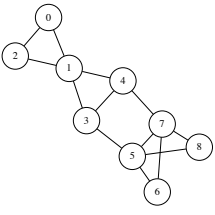
Tabel 1: Boogsamenhangende componenten in `three-cc`, `connected` en `kite`

Ga als volgt te werk:

1. Introduceer een teller om het aantal boogsamenhangende componenten bij te houden en verhoog deze teller op de juiste plaats.
2. Introduceer een knoop-geïndexeerde vector waarin voor elke knoop bijgehouden wordt tot welke boogsamenhangende component de knoop behoort.
3. Introduceer een stack (`a-d/stack/linked.rkt`) om bezochte knopen bij te houden. Waar moet je de knopen op de stack pushen?
4. Wanneer er een brug (en dus ook een nieuwe boogsamenhangende component) gedetecteerd wordt ledig je de stack tot en met één van de uiteinden van de brug (welke?) en ken je alle gepopte knopen toe aan de zojuist gedetecteerde boogsamenhangende component.
5. Zorg dat de procedure naast het aantal boogsamenhangende componenten en de lijst van bruggen nu ook de nieuwe knoop-geïndexeerde vector teruggeeft.
6. Gebruik de code in `Vraag1-Tests.rkt` om je oplossing te testen op de grafen `three-cc`, `connected` en `kite`.

2 Bigeconnecteerdheid

In de file `a-d/graph-algorithms/undirected/connectivity.rkt` vind je een **DFT-gebaseerd algoritme** voor het vinden van **articulatiepunten** (of scharnierpunten) in een **ongerichte graaf**. Deze functie, `biconnected-components`, geeft ons nu enkel de scharnierpunten terug (als een knoop-geïndexeerde vector met booleans). Het zou echter ook nuttig zijn om het **aantal bigeconnecteerde componenten** te kennen, alsook tot **welke** van die componenten individuele knopen behoren. De opdracht is dus opnieuw om het algoritme zodanig aan te passen dat het ook het **aantal bigeconnecteerde**

| Graaf |  |  |  |
|--------------------|---|--|---|
| Scharnierpunten | #(F T F F F T F F F F) | #(F F F F F F F T) | #(F T F F F F F F F) |
| Aantal Componenten | 5 | 2 | 2 |
| Lidmaatschap | #((2) (2 1) (1) (1) (4) (4 3) (3) (5) (5) (5) (5)) | #((2) (1) (2) (2) (2) (2) (2) (2 1)) | #((2) (2 1) (2) (1) (1) (1) (1) (1) (1)) |

Tabel 2: Bigeconnecteerde componenten in **three-cc**, **connected** en **kite**

componenten en het **lidmaatschap** van de knopen tot de componenten teruggeeft. Het gewenste resultaat voor de grafen **three-cc**, **connected** en **kite** is te zien in Tabel 2.

Ga als volgt te werk:

1. Introduceer een teller om het aantal bigeconnecteerde componenten bij te houden en verhoog deze teller op de juiste plaats.
2. Introduceer een knoop-geïndexeerde vector waarin voor elke knoop bijgehouden wordt tot welke bigeconnecteerde component(en) de knoop behoort.
3. Introduceer een stack (**a-d/stack/linked.rkt**) om bezochte knopen bij te houden. Waar moet je de knopen op de stack pushen?
4. Wanneer er een scharnierpunt (en dus ook een nieuwe bigeconnecteerde component) gedetecteerd wordt ledig je de stack tot wanneer het preorder nummer van de top van de stack $< / > / = / <= / >=$ (schrab wat niet past) wordt dan het preorder nummer van de **from** of **to** knoop (bepaal zelf). Ken alle gepopte knopen toe aan de zojuist gedetecteerde bigeconnecteerde component. Wijs tenslotte ook het articulatiepunt zelf toe aan de nieuwe bigeconnecteerde component.
5. Zorg dat de procedure naast het aantal bigeconnecteerde componenten en de vector van scharnierpunten nu ook de nieuwe knoop-geïndexeerde vector teruggeeft.
6. Gebruik de code in **Vraag2-Tests.rkt** om je oplossing te testen op de grafen **three-cc**, **connected** en **kite**.