Algoritmen en Datastructuren 2

Project Dynamisch Programmeren

Maximale palindromische paden in gerichte grafen

Ruben Wambacq

Universiteit Gent

25 November, 2018

Inhoud

Algoritme in pseudocode

De pseudocode van het algoritme dat ik gebruik is bijgevoegd in appendix A.

Theoretische vragen

Vraag 1

Bewijs dat jouw algoritme voor acyclische grafen correct is.

Doordat elk mogelijk pad binnen de graaf volledig wordt afgelopen, zullen alle mogelijke combinaties van begin- en eindnodes (gevonden door het volgen van de gerichte bogen) zullen worden nagekeken in het algoritme, waardoor elk mogelijk "woord" in graaf zal worden gecheckt.

Daardoor wordt het bewijs gereduceerd tot het bewijzen dat, indien een woord door het algoritme wordt beoordeeld en het een palindroom is, het zal worden opgeslagen (in de 2D-array).

Dit kan gemakkelijk worden bewezen door het beschouwen van een recursieve definitie van een palindroom:
Een woord is een palindroom indien het teken aan het begin en het einde van het woord gelijk is en ook het woord zonder deze twee tekens (indien dat er is) voldoet aan deze definitie van een palindroom.

Het algoritme dat ik gebruik zal nakijken voor elke combinatie van begin- en eindnodes of deze twee gelijk zijn en dan ook of het tussenliggende woord een palindroom is. Deze tweede voorwaarde wordt op dynamische wijze nagekeken en uit de 2D-array gehaald, waardoor dit niet telkens opnieuw zal moeten worden berekend.

Doordat de woorden recursief worden nagekeken en in het rooster worden gestopt (dus indien bij het onderzoeken van een woord met lengte n het woord dat er tussen zit met lengte $<$ n nog niet in het rooster zit, zal dit recursief worden ingevuld) zal bij het beschouwen van een woord met lengte $>$ n steeds al in het rooster ingevuld staan of het woord van lengte n een palindroom is.

Vraag 2

Bepaal en bewijs de complexiteit van jouw algoritme voor acyclische grafen.

a. In functie van het aantal toppen n

De ondergrens van de uitvoeringstijd zal op n liggen, dit is namelijk het geval waar elke node verschillend is, dan zal in de evalueer functie niet worden verder gegaan bij elke node die wordt nagekeken, waardoor enkel de strings van lengte 1 worden ingevuld in het rooster (dat is dus de hoofddiagonaal), wat neerkomt op n stappen.

De bovengrens zal worden gevonden in een graaf met de maximale hoeveelheid links tussen toppen. Daar hebben we n checks voor strings van lengte 1, (n-1) checks voor strings van lengte 2, dit gaat zo verder tot 1 check voor een string van lengte n, namelijk de string die alle nodes bevat.

Dit gaat zorgen voor een complexiteit van
n + (n-1) + … + 2 + 1 = (n * (n+1))/2
wat neerkomt op een kwadratische ($O(n^2)$) complexiteit.

b. In functie van het aantal toppen n en het aantal bogen m

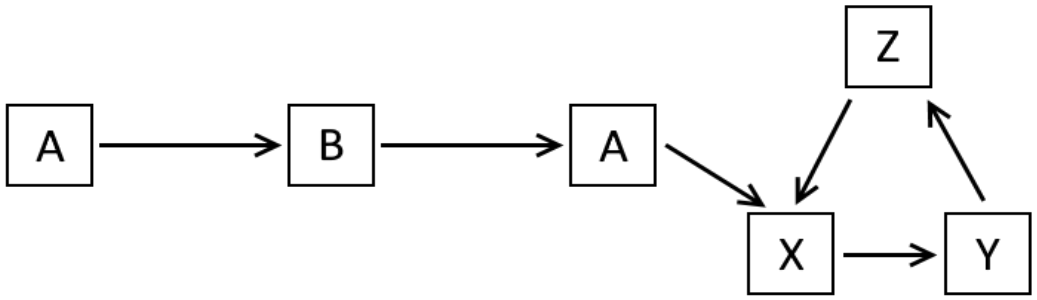
Voor de complexiteit met de toppen en de bogen is er weinig verschil. Dit zal ook neerkomen op een kwadratische complexiteit, er worden namelijk even veel checks gedaan, maar doordat het in functie van m+n moet worden uitgedrukt, zal het uiteindelijke resultaat minder groot zijn, het wordt namelijk nog gedeeld door een getal tussen 1 en 2 , doordat het aantal bogen tussen 0 en n-1 ligt. Het uiteindelijke resultaat moet dus worden uitgedrukt a.d.h.v. een waarde die tussen n en 2n ligt.

Vraag 3

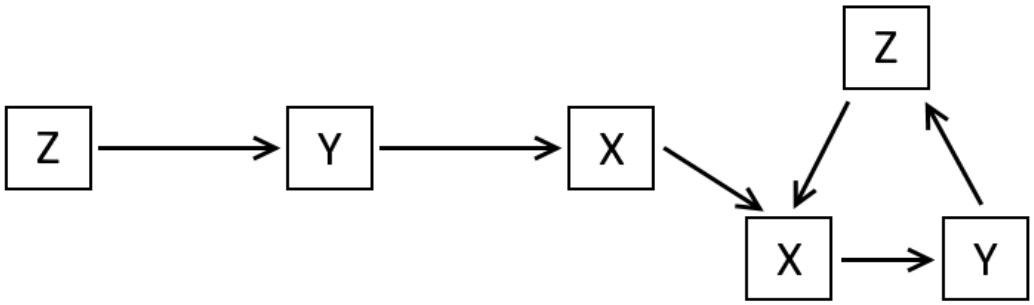
Geef een voorbeeld van de volgende situaties en leg het uit, of bewijs dat het onmogelijk is.

a. Een acyclische graaf die palindromisch onbegrensd is:
Dit zal niet bestaan. Het langst mogelijke palindroom in een acyclische graaf met n toppen is er een van lengte n, indien alle toppen van de graaf deel uitmaken van het langste palindroom. Er zal geen palindroom van lengte groter dan n bestaan in deze graaf. Aangezien dat juist de definitie is van palindromische onbegrensdheid, wil dit dus zeggen dat een acyclische graaf onmogelijk palindromisch onbegrensd kan zijn.

b. Een gerichte graaf met cykel die palindromisch begrensd is:
Deze gerichte graaf bevat een cykel, namelijk bij de nodes met de labels X, Y en Z, maar is palindromisch begrensd, aangezien het grootste palindroom in deze boom van lengte 3 is, meer bepaald ABA. Er zal geen groter palindroom gevonden kunnen worden doordat de letters in de cykel in de cykel niet gelijk zijn en dus nooit een palindroom zullen vormen.

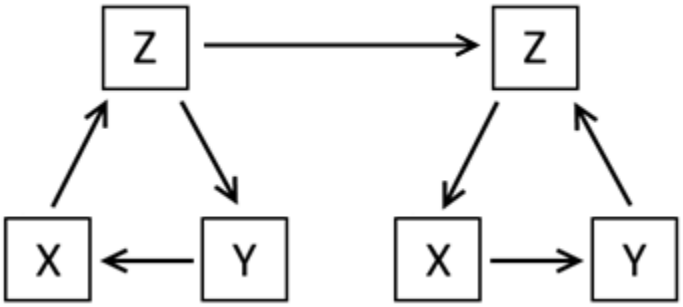


c. Een maximaal palindromisch pad dat een cykel bevat:
Het maximaal palindromisch pad in deze gerichte graaf met cykel is ZYXXYZ, dit pad bevat de nodes uit de cykel. Maar aangezien de labels van de nodes in deze cykel niet gelijk zijn, zal de graaf palindromisch begrensd zijn en zullen er dus geen langere



palindromen kunnen gevonden worden.

d. Een palindromisch onbegrensde graaf zonder palindromische cykels:
Dit soort graaf kan bestaan. Indien een graaf bestaat uit 2 niet-palindromische cykels die elkaar opvolgen (mogelijk verbonden door een palindromisch pad) waarbij de cykels dezelfde reeks karakters bevatten (in dezelfde ordening), maar in omgekeerde volgorde worden afgelopen (en het karakter waarmee ze aan elkaar verbonden zijn is hetzelfde), kan eerst de eerste cykel n keer worden afgelopen en daarna de 2^e cykel ook n keer worden afgelopen, waardoor een palindroom bestaat van lengte 2n (+ mogelijks de lengte van het palindroom waarmee ze verbonden zijn). Doordat we n zo groot kunnen laten worden als we maar willen, zal deze graaf palindromisch onbegrensd zijn.



Vraag 4

Het maximale aantal verschillende paden in een gerichte graaf is exponentieel in functie van het aantal toppen. De maximaal palindromische paden zijn daar natuurlijk maar een deel van. Is het maximale aantal maximaal palindromische paden exponentieel?

Ja, ook het maximale aantal maximaal palindromische paden is exponentieel tegenover het aantal nodes in de graaf. Dit komt door wat dat hierboven ook werd gezegd, namelijk dat het maximale aantal maximaal palindromische paden een fractie is van het maximale aantal verschillende paden. Stel dan dat het maximale aantal verschillende paden een aantal x is. In dat geval heeft x een grootte van $\Omega(2^n)$. Laat ons nu zeggen dat het maximale aantal maximaal palindromische paden x/y is, een fractie van x. Dan zal x/y een grootte hebben van $\Omega(\frac{2n}{y})$, wat nog steeds neerkomt op $\Omega(2^n)$, wat dus wil zeggen dat ook het maximale aantal maximaal palindromische paden exponentieel is tegenover het aantal nodes n in de graaf.

Vraag 5

Geef en bewijs een polynomiale bovengrens voor de lengte van een maximaal palindromisch pad in functie van het aantal toppen. Dit hoeft geen scherpe bovengrens te zijn.

Voor deze vraag ga ik enkel palindromisch begrensde grafen bekijken, aangezien de bovengrens van de lengte van een palindromisch pad bij een palindromisch onbegrensde graaf volgens definitie steeds oneindig zal zijn.

Deze zal een bovengrens hebben van $n^2 - n$, dit is namelijk de lengte van een pad binnen een complete graaf. Een palindromisch pad zal nooit een complete graaf kunnen afgaan, aangezien dan zeker een palindromische cykel zal ontstaan.

Vraag 6

Bewijs dat jouw algoritme voor algemene gerichte grafen correct is.

In vraag 1 is reeds bewezen dat het algoritme klopt indien er geen cykels in het algoritme zitten. Daardoor kan het bewijs voor algemene gerichte grafen worden aangepast naar het bewijzen dat een palindromische cykel zal worden gedetecteerd en opgevangen indien er zich 1 in het pad bevindt.

Ook zal de methode om cykels uit te voeren gedaan worden voordat de rest van het algoritme wordt uitgevoerd, dus zal dit niets veranderen aan het uitvoeren van het algoritme dat in vraag 1 beschreven is.
De methode die ik gebruik om dat te doen bestaat uit 2 delen.

Het eerste dat ik doe is voor elke node nagaan of er een palindroom bestaat die begint en start in die node. Dit zal sowieso bestaan, namelijk dat van lengte 1, aangezien 1 enkel karakter ook een palindroom is. Maar het zou kunnen dat er een palindroom gevonden wordt dat langer is dan 1, in dat geval zal deze node deel uitmaken van een palindromische cykel. Op dat moment is de palindromische cykel dus correct gedetecteerd en zal er met de rest van het algoritme gestopt worden.

Er zit in het uitvoeren van dit algoritme wel nog 1 probleem.
Bij het invullen van de 2D-array zou het in sommige gevallen (bv. bij een cykel die zelf ook nog deel uitmaakt van een cykel) kunnen dat het algoritme na enkele recursieve stappen terug bij dezelfde node uitkomt, waardoor het in een oneindige lus komt.

Dit is waar ik het 2^e deel van mijn cykeldetectie gebruik.
Bij het onderzoeken van een nieuw node-paar zal ik steeds dit paar in een lijst steken. Indien de node volledig verwerkt is en er dus iets is ingevuld in de 2D-array hiervoor zal dit node-paar terug uit de lijst worden weg gehaald.
Wanneer bij het toevoegen van een node-paar blijkt dat dit reeds aanwezig is in de lijst, wil dit zeggen dat er een cykel in de boom zit die zorgt voor een oneindige lus. Op dat moment zal ook worden gestopt met het algoritme.

Met deze 2 methodes samen zullen alle palindromische cykels in de graaf worden opgevangen.
Indien er geen cykels gedetecteerd zijn tijdens deze eerste fase zal het algoritme gewoon verder gaan en op normale wijze het langste palindroom zoeken, waarvan al bewezen is in vraag 1 dat dit een correct resultaat zal opleveren.

Vraag 7

Bepaal en bewijs de complexiteit van jouw algoritme voor algemene gerichte grafen.

In vraag 2 is reeds de complexiteit van het algoritme zonder cykels bewezen. Doordat deze algemene implementatie enkel voor de hoofdlus extra werk zal verrichten, volstaat het om de complexiteit die we in vraag 2 gevonden hebben, nl. $O(n^2)$ op te tellen met de complexiteit van het extra deel vooraan.

In dit extra deel zal voor elke n nodes in de graaf 1 extra check worden gedaan, namelijk of er tussen deze nodes een palindroom zal zitten dat langer is dan 1. In deze extra checks zullen recursief ook andere delen van het rooster worden opgevuld, maar dat is werk dat toch zou gedaan worden in het algemene deel van het algoritme. Daarom zullen deze recursieve oproepen dus niet voor een grotere complexiteit zorgen, deze zaten al bevat in de $O(n^2)$ complexiteit uit vraag 2.

Dit wil dus zeggen dat er n extra stappen zullen worden uitgevoerd, vergeleken met de $O(n^2)$ complexiteit uit vraag 2. Dat zal dan zorgen voor een totale tijdscomplexiteit van $O(n^2 + n)$, wat nog steeds neerkomt op een complexiteit van $O(n^2)$.

```
DAG:
    main:
        while(true):
            Tree t = boom(input)
            t.langstePalindroomAcyclisch()
DG:
    main:
        while(true):
            Tree t = boom(input)
            t.langstePalindroomCyclisch()

Tree:
    int[][][][] matrix;
    // 2D rooster waarbij op indices i,j de langste palindromen worden bijgehouden van index i naar index j
    // De 3e en 4e dimensie van de matrix zijn omdat er meerdere palindromen kunnen zijn en een palindroom ook bestaat uit meerdere indices

    langstePalindroomAcyclisch():
        for i in [0..#nodes]:
            matrix[i][i] = palindroom lengte 1
        vindMaxPalindroom // Vul de rest van de matrix

    langstePalindroomCyclisch():
        HashMap<int, int[]> researching
        // Een HashMap waar de key een startindex in de graaf voorstelt en een element in de lijst een eindindex in de graaf
        // Als bij het recursief opvullen van de matrix blijkt dat een bepaalde index reeds wordt onderzocht, volgt hieruit dat er een oneindige palindromische cykel in het pad zit

        while i = 0 < #nodes && researching bevat geen dubbels:
            vulPalindroomMatrix(i, i)
            i++;
        if ( dubbel gevonden in researching ):
            stop alles, return lege palindroom
        else:
            for i in [0..#nodes]:
                if ( matrix[i][i] == palindroom met lengte > 1 ):
                    stop alles, return lege palindroom
            vindMaxPalindroom() // Vul de rest van de matrix

    vindMaxPalindroom():
        if ( size boom == 0 ):
            stop alles, return lege palindroom
        else:
            for (i, j) in rooster:
                if ( i != j ):
                    vulPalindroomMatrix(i, j)
            result = langste palindromen in matrix
            d = doorsnede( string(result) )
            print( result[0].size + d + result[0] )

    vulPalindroomMatrix(int i, int j):
    // Hierbij is i de index van de startnode in de boom en j de index van de eindnode in de boom
    // Samen stellen ze dus een pad voor waarvan we onderzoeken of het palindromisch is

        voeg (i, j) toe aan researching HashMap
        if (matrix[i][j] bevat nog geen palindromen):
            if ( beginkarakter == eindkarakter ):
                if ( eindnode zit in buren startnode ):
                    l = []
                    for ( b = alle buren startnode behalve de eindnode):
                        for( v = alle voorgangers eindnode behalve de startnode ):
                            if ( matrix[b][v] leeg ):
                                vulPalindroomMatrix(i, j)
                            if ( palindromen in matrix[b][v] > palindromen in l):
                                l = matrix[b][v]
```

```
                            else if ( palindromen in matrix[b][v] == palindromen in l):
                                l += matrix[b][v]
                    if ( l niet leeg ):
                        matrix[i][j] = i + l + j
                    else:
                        matrix[i][j] = i + j
                else:
                    l = []
                    for ( b = alle buren startnode ):
                        for( v = alle voorgangers eindnode ):
                            if ( matrix[b][v] leeg ):
                                vulPalindroomMatrix(i, j)
                            if ( palindromen in matrix[b][v] > palindromen in l):
                                l = matrix[b][v]
                            else if ( palindromen in matrix[b][v] == palindromen in l):
                                l += matrix[b][v]
                    if ( l niet leeg ):
                        matrix[i][j] = i + l + j
                    else:
                        matrix[i][j] = -1 // Dit wil zeggen dat er geen palindroom zit
            else:
                matrix[i][j] = -1 // Dit wil zeggen dat er geen palindroom zit

    doorsnede( char[][] palindromen ):
        if ( palindromen.size > 0 ):
            long samen = stringNaarLong( palindromen[0] )
            for ( i = 1..size palindromen ):
                samen &= stringNaarLong( palindromen[i] )
            l = [];
            for ( i = 0..64 ):
                if ( samen & bit(i) != 0 ):
                    l += i
            toReturn = []
            for ( i = 0..#l ):
                toReturn += intNaarChar(l[i])

    charNaarInt (char c):
        ascii = (int)c
        if ( 48 <= c <= 57 ):
            return ascii - 48
        else if ( 65 <= c <= 90 ):
            return ascii - 55
        else if ( 97 <= c <= 122 ):
            return ascii - 61
        else:
            return -1

    intNaarChar (int i):
        if ( 0 <= c <= 9 ):
            return (char) ascii + 48
        else if ( 10 <= c <= 35 ):
            return (char) ascii + 55
        else if ( 36 <= c <= 61 ):
            return (char) ascii + 61

    stringNaarLong ( char[] s ):
        long toReturn = 0
        for ( char c in s ):
            toReturn |= bit(charNaarInt(c))
        return toReturn;
```