|  |  |  |
| --- | --- | --- |
| **Guus Portegies** | **IS203** | **500691990** |
| **Koen Karsten** | **IS203** | **500695520** |

Sorting & Searching Practicum 3

Inhoudsopgave

Sorting & Searching Practicum 3 1

1. Inhoudsopgave 2

2. Inleiding 3

3. Opdracht 4

3.1 Java project en bibliotheken importeren 4

4. Programma 5

4.1 Testen Dijkstra’s algoritme 5

4.2 Dijkstra klasse 6

5. Resultaten 7

Inleiding

In dit rapport behandelen wij de uitwerking van opdracht 3 voor het vak Searching & Sorting. De opdracht gaat met name over ‘substring search’ en ‘regular expressions’.

In hoofdstuk 3 hebben wij de drie verschillende opdrachten uitgewerkt. Deze staan beschreven in de onderliggende paragrafen. In paragraaf 3.1 vergelijken wij de substring methode van Java met een zelfde soort methode uit een andere programmeertaal. Vervolgens vergelijken wij in paragraaf 3.2 de algoritmes Knuth-Morris-Pratt en Boyer-Moore met elkaar. In de laatste paragraaf behandelen wij een aantal reguliere expressies om onder andere een telefoonnummer te controleren.

Opdracht

Implementatie van substring in programmeertalen

Voor deze opdracht hebben wij moeten kijken naar de implementatie van de substring methode in verschillende programmeertalen. De eerste programmeertaal waarvan we de substring methode bekijken is Java. Als tweede programmeertaal hebben wij gekozen

Om te beginnen hebben we voor dit practicum een nieuw Java project aangemaakt in Netbeans. Tevens hebben we een private repository aangemaakt op GitHub om samenwerking en versiebeheer te bevorderen. Vervolgens hebben wij de PO2Library geïmporteerd in onze applicatie.

Tevens hebben wij in de root folder van onze applicatie de mappen: input en output aangemaakt. Dit hebben wij gedaan omdat de EdgeWeightedDigraph klasse gebruikt maakt van deze mappen. In de input map bevinden zich de 21 afbeeldingen die wij gekregen hebben bij aanvang van deze opdracht. De output folder wordt gebruikt om afbeeldingen in op te slaan wanneer hier het kortste pad in is getekend.

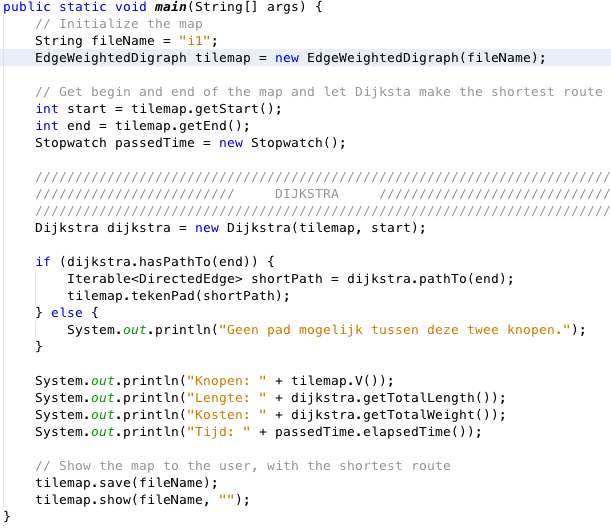
Vergelijken van algoritmes

Reguliere expressies

Programma

Testen Dijkstra’s algoritme

Om te testen wat de uitkomsten van het Dijkstra’s algoritme zijn hebben we de volgende code geschreven in de ‘main’ van onze applicatie.



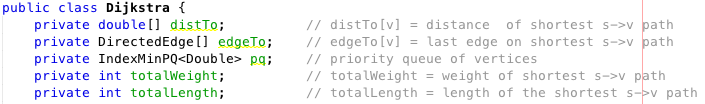
Zoals u hierboven kunt zien maken we eerst van de betreffende afbeelding een EdgeWeightedDigraph. Dit is een graaf waar iedere verbinding tussen knopen een gewicht of kosten hebben. Vervolgens definiëren we het begin en eindpunt uit de afbeelding(dit zijn de rode en de groene tegel in de afbeelding). Nadat we dit gedaan hebben starten we een stopwatch om de tijd te meten die het Dijkstra algoritme er over doet om het kortste pad te berekenen.

We maken een nieuwe Dijkstra object aan en geven de betreffende kaart en het beginpunt(rode tegel) mee. Nadat we dit gedaan hebben kijken we eerst of er wel een pad bestaat naar het eind. Indien dit pad bestaat halen we Dijkstra’s kortste pad op en tekenen we die op het kaartje. Als er geen enkel pad mogelijk is tonen wij een bericht met “Geen pad mogelijk tussen deze twee knopen”.

Als laatste tonen wij de resultaten, waaronder: het aantal knopen dat doorzocht is, de totale lengte van het kortste pad, de totale kosten van het kortste pad en de tijd die Dijkstra’s algoritme er over gedaan heeft om dit te doen. Vervolgens slaan wij de afbeelding op in de output folder en tonen wij deze aan de gebruiker.

Dijkstra klasse

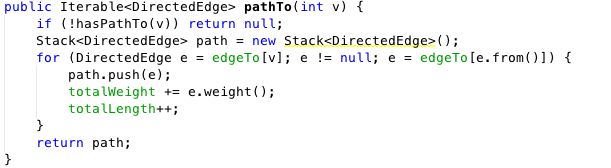
Aan de Dijkstra klasse hebben we een twee attributen toegevoegd om de totale kosten en de totale lengte van het kortste pad te kunnen tonen. Dit zijn de attributen ‘totalWeight’ en ‘totalLength’.



Vervolgens hebben we er voor gezorgd dat wanneer er een nieuw Dijkstra object wordt aangemaakt, deze beide attributen een waarde van 0 hebben. Dit hebben we gedaan in de constructor van de Dijkstra klasse.

Samsung SSD 840 Pro:Users:guus_portegies:Desktop:Schermafbeelding 2015-01-14 om 16.58.51.png

Om uiteindelijk de juiste waarden van het kortste pad in de twee attributen te krijgen hebben we ook code aan de pathTo methode toegevoegd. Voor iedere DirectedEdge die aan het kortste pad wordt toegevoegd tellen wij 1 bij de totale lengte op. Voor het totale gewicht tellen wij het vorige totale gewicht plus het gewicht van de DirectedEdge die wordt toegevoegd bij elkaar op.



Om deze waardes op te vragen hebben we voor beide attributen een get methode aangemaakt die kan worden aangeroepen.

