Hoofdstuk 7
Kortste Afstanden II

	Negatieve gewichten?	Start	Performantie
Dijkstra	Nee	1 knoop	O((m+n) lg n)
Floyd - Warshall	Ja	Vanuit alle knopen	$O(n^3)$
Bellman - Ford	Ja	1 knoop	O(nm)
Johnson	Ja	Vanuit alle knopen	O(nm lg n)

Voor de eenvoud veronderstellen we dat alle grafen gericht zijn.

7.1 Kortste Afstanden vanuit één Knoop

Als gewichten negatief kunnen zijn, dan werkt Dijkstra niet meer. Het algoritme moet dan op zijn beslissing kunnen terugkomen, wat ons bij dynamisch programmeren brengt.

7.1.1 Bellman-Ford

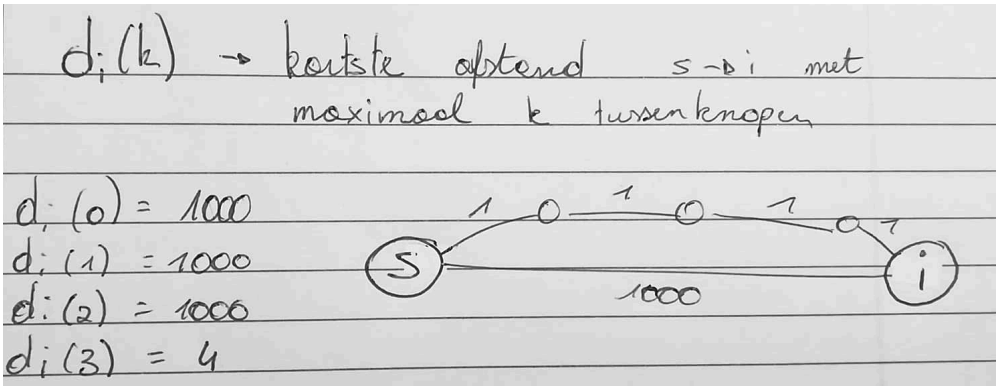
Dit algoritme kan gezien worden als een **combinatie van BEZ in een niet gewogen graaf met kortste afstanden in een gewogen graaf.**

De methode is gebaseerd op het feit dat als een graaf geen negatieve lussen heeft, de kortste wegen evenmin lussen hebben, en dus maximaal *n-1* verbindingen bevatten.
Er bestaat dus een recursief verband tussen kortste wegen met maximaal *k* verbindingen en kortste wegen met maximaal *k-1* verbindingen.

Als $d_i(k)$ het gewicht is van de kortste weg met maximaal *k* verbindingen vanuit de startknoop naar knoop *i*, en g_{ij} het gewicht van de verbindingen *(j,i)* dan geldt:

$$d_i(k) = min\ (\ d_i(k\text{-}1)\ ,\ d_j(k\text{-}1) + g_{ij}\ )$$

Daarbij zijn de gezochte kortste afstanden de waarden $d_i(n\text{-}1)$ voor elke knoop *i*.



Om *di(k)* te bepalen hebben we telkens *dj(k-1)* nodig voor verschillende waarden van *j*. De deeloplossingen zijn onafhankelijk en overlappend en er is een optimale deelstructuur.

Er zijn *n-1* iteraties vereist, waarbij de kortste wegen telkens één verbinding langer mogen worden. Voor elke knoop moet onderzocht worden of zijn kortste afstand verbeterd kan worden via zijn buren. Daarvoor worden alle voorlopige afstanden in een tabel bijgehouden. Om kortste wegen te reconstrueren volstaat het de voorloperknoop op elke voorlopig kortste weg op te slaan.

De performantie is O(nm), want elke iteratie moet in principe *m* verbindingen testen.

Twee goede implementaties:

- Meestal is het niet nodig om in elke iteratie alle verbindingen te onderzoeken. Als een iteratie de voorlopig kortste afstand tot een knoop *v* niet aangepast heeft, is het zinloos om bij de volgende iteratie de verbindingen vanuit die knoop *v* te onderzoeken.
  Daarom zet men alle knopen waarvan de voorlopige kortste afstand aangepast is op een *queue*, en worden enkel hun buren bij de volgende iteratie getest (~ BEZ).
  Wanneer een knoop uit de queue gehaald wordt en de afstand van één van zijn buren wordt aangepast, moet die buur op de queue.
  Het kan voorkomen dat er een negatieve lus is, en dan zou het algoritme blijven doorgaan. Daarom wordt vaak bijgehouden hoeveel iteraties de queue niet leeg is. Na n iteraties heeft de graaf een negatieve lus.
- De implementatie van **Pape** blijkt praktisch zeer snel, maar kan heel inefficiënt uitvallen. Ze lijkt sterk op de vorige, maar in plaats van een queue wordt een *deque* gebruikt. Een knoop wordt vooraan weggenomen om te onderzoeken, maar toevoegen gebeurt vooraan én achteraan. Als de afstand van een knoop wordt aangepast, en de knoop zat vroeger al in de deque wordt vooraan toegevoegd, in het andere geval wordt steeds achteraan teogevoegd.

Het algoritme van Bellman-Ford laat negatieve gewichten toe, en is ook veel flexibeler en meer gedecentraliseerd dan Dijkstra. Als de voorlopige kortste afstand tot een knoop wijzigt kan die knoop zijn buren verwittigen, dat is beter dan Dijkstra, die een globale kennis van het netwerk vereist.

7.2 Kortste Afstanden tussen alle Knopenparen

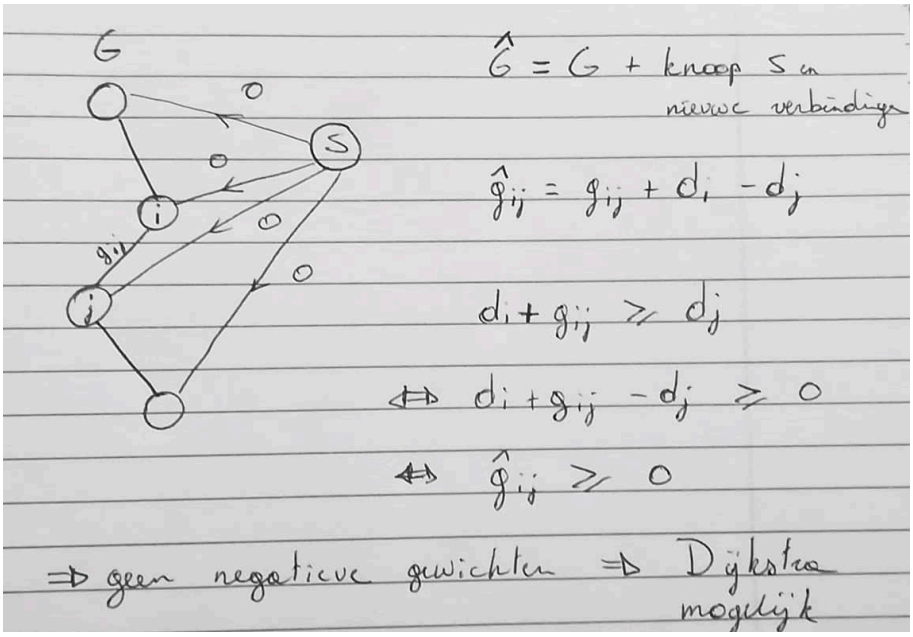
7.2.1 Het algoritme van Johnson

Het algoritme van Johnson maakt gebruik van de algoritmen van Bellman-Ford en van Dijkstra.

Om Dijkstra te kunnen gebruiken krijgen alle verbindingen een positief gewicht. De kortste wegen moeten echter ook met de nieuwe gewichten de kortste wegen blijven.

Daarom breidt men de graaf eerst uit met een nieuwe knoop *s* die verbindingen van gewicht 0 krijgt naar alle andere knopen. Als de oorspronkelijke graaf geen negatieve lussen had, dan geldt dit nog steeds, want knoop *s* heeft enkel uitgaande verbindingen.

We kunnen dus Bellman-Ford toepassen op de nieuwe graaf om vanuit *s* de kortste afstand d_i te bepalen tot elke originele knoop i.



Het algoritme van Dijkstra kan dus toegepast worden op elke originele knoop in de nieuwe graaf en zal de correcte kortste wegen tussen alle knopenparen vinden. Natuurlijk mag geen enkele weg daarbij langs *s* gaan. Voor de kortste afstanden moeten natuurlijk de oorpronkelijke gewichten gebruikt worden.

7.3 Transitieve Sluiting

Een **sluiting** is een **algemene methode om één of meerder verzamelingen op te bouwen**. Daarbij past men herhaaldelijk één of meerdere regels toe van de vorm "*als een verzameling deze gegevens bevat, moet ze ook de volgende gegevens bevatten.*" Dit tot er niets meer aan de verzameling(en) kan toegevoegd worden.

Een **transitieve sluiting** is een speciaal geval, met regels van de vorm "*als (a,b) en (b,c) aanwezig zijn, moet (a,c) ook aanwezig zijn.*"

Toegepast op een gerichte graaf is dit een nieuwe gerichte graaf met dezelfde knopen en een verbinding van *i* naar *j* als er in de oorspronkelijke graaf een weg bestaat van *i* naar *j*.

De enige verzameling die opgebouwd moet worden is die met de verbindingen van de nieuwe graaf. Hiervoor zijn drie manieren:

- De eenvoudigste manier gebruikt DEZ of BEZ om alle knopen op te sporen die vanuit de startknoop bereikbaar zijn. Dit wordt herhaald met elke knoop als startknoop. Voor ijle grafen krijgen we $\Theta(n\ (n+m))$, voor dichte grafen $\Theta(n^3)$.
- Als men verwacht dat de transitieve sluiting een dichte graaf zal zijn, zijn veel knopen onderling bereikbaar. De graaf bestaat dan uit een klein aantal sterk samenhangende componenten. Het bepalen van die componenten is $\Theta(n+m)$. Daarna bepaalt men de *componentengraaf*, die heel wat kleiner zal zijn dan de oorspronkelijke (opstelling is

O(n+m)). Nu kan men eenvoudigweg nagaan of component *j* bereikbaar is vanuit component *i*. Is dit het geval, dan zijn alle knopen van *j* bereikbaar vanuit alle knopen van *i*.

- Voor dichte grafen is het algoritme van Warshall aangewezen, die sterk op Floyd-Warshall lijkt. Er worden een reesk opeenvolgende *n x n* matrices $T^{(0)}$, ... , $T^{(n)}$ bepaald die nu *logische waarden* bevatten. Element $T^{(k)}_{ij}$ duidt aan of er een weg bestaat met als mogelijke intermediaire knopen *1..k*.
  Als $T^{(n)}_{ij}$ *"true" is* dan behoort verbinding (i,j) tot de transitieve sluiting.