FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Gevorderde algoritmen

Bert De Saffel

Master in de Industriële Wetenschappen: Informatica Academiejaar 2018–2019

Gecompileerd op 10 december 2019

UNIVERSITEIT
GENT

Inhoudsopgave

Deel I

Grafen II

Hoofdstuk 1

Kortste afstanden II

- Het traditioneel algoritme om de kortste afstand tussen twee knopen te bepalen is het algoritme van Dijkstra.
- Probleem:
  - Dijkstra gebruikt het feit dat indien een pad naar A $\rightarrow$ C bestaat met kost $K_{A,C}$, er geen korter pad A $\rightarrow$ B $\rightarrow$ C kan zijn met kost $K_{A,B} + K_{A,C}$, daarom is het algoritme performant, maar er mogen dus geen negatieve verbindingen zijn. Indien $K_{A,B}$ negatief zou zijn dan klopt Dijkstra niet want dan

$$K_{A,B} + K_{A,C} > K_{A,C}$$

- Volgende algoritmen hebben enkel betrekking tot **gerichte grafen** met mogelijks **negatieve gewichten**.

1.1 Kortste afstanden vanuit één knoop

1.1.1 Algoritme van Bellman-Ford

- Werkt voor negatieve verbindingen.
- Geen globale kennis nodig van heel het netwerk, zoals bij Dijkstra, maar slechts enkel de buren van een bepaalde knoop. Daarom gebruiken routers Bellman-Ford (distance vector protocol).

! Zal niet stoppen indien er een negatieve lus in de graaf zit, aangezien het pad dan zal blijven dalen tot $-\infty$.

Het algoritme berust op een belangrijke eigenschap: Indien een graaf geen negatieve lussen heeft, zullen de kortste wegen evenmin lussen hebben en hoogstens $n-1$ verbindingen bevatten. Hieruit kan een recursief verband opgesteld worden tussen de kortste wegen met maximaal k verbindingen en de kortste wegen met maximaal $k-1$ verbindingen.

$$d_i(k) = \min(d_i(k-1), \min_{j \in V}(d_j(k-1) + g_{ji}))$$

met

- $d_i(k)$ het gewicht van de kortste weg met maximaal k verbindingen vanuit de startknoop naar knoop i,

- g_{ji} het gewicht van de verbinding (j, i),
- $j \in V$ is elke knoop j.

Er bestaan twee goede implementaties:

1.
   - Niet nodig om in elke iteratie alle verbindingen te onderzoeken. Als een iteratie de voorlopige kortste afstand tot een knoop niet aanpast, is het zinloos om bij de volgende iteratie de verbindingen vanuit die knoop te onderzoeken.
   - Plaats enkel de knopen waarvan de afstand door de huidige iteratie gewijzigd werd in een wachtrij.
   - Enkel de buren van deze knopen worden in de volgende iteratie getest.
   - Elke buur moet, indien hij getest wordt en nog niet in de wachtrij zit, ook in de wachtrij gezet worden.
2.
   - Gebruik een deque in plaats van een wachtrij.
   - Als de afstand van een knoop wordt aangepast, en als die knoop reeds vroeger in de deque zat, dant voegt men vooraan toe, anders achteraan.
   - Kan in bepaalde gevallen zeer inefficiënt uitvallen.

1.2 Kortste afstanden tussen alle knopenparen

- Voor dichte grafen → Floyd-Warshall (Algoritmen I).
- Voor ijle grafen → Johnson.

1.2.1 Het algoritme van Johnson

- Maakt gebruik van Bellman-Ford en Dijkstra.
- Omdat we Dijkstra gebruiken, moet elk gewicht positief worden.
  1. Breidt de graaf uit met een nieuwe knoop s, die verbindingen van gewicht nul krijgt met elke andere knoop.
  2. Voer Bellman-Ford uit op de nieuwe graaf om vanuit s de kortste afstand d_i te bepalen tot elke originele knoop i.
  3. Het nieuwe gewicht $\hat{g}_{ij}$ van een oorspronkelijke verbinding g_{ij} wordt gegeven door:

$$\hat{g}_{ij} = g_{ij} + d_i - d_j$$

- Het algoritme van Dijkstra kan nu worden toegepast op elke originele knoop, die alle kortste wegen zullen vinden. Om de kortste afstanden te bepalen moeten de originele gewichten opgeteld worden op deze wegen.
- Dit algoritme is $O(n(n+m)\lg n)$ want:
  - Graaf uitbreiden is $\Theta(n)$.
  - Bellman-Ford is $O(nm)$.
  - De gewichten aanpassen is $\Theta(m)$.
  - n maal Dijkstra is $O(n(n+m)\lg n)$. Dit is de belangrijkste term, al de andere termen mogen verwaarloosd worden.

1.3 Transitieve sluiting

Sluiting = algemene methode om één of meerdere verzamelingen op te bouwen. ('als een verzameling deze gegevens bevat, dan moet ze ook de volgende gegevens bevatten').

- **Fixed point**: Een sluiting wordt fixed point genoemd omdat op een bepaald moment verdere toepassing niets meer verandert, $f(x) = x$.
- **Least fixed point**: De kleinste x zoeken zodat $f(x) = x$ voldaan wordt.

Transitieve sluiting = 'Als (a, b) en (b, c) aanwezig zijn dan moet ook (a, c) aanwezig zijn.'

- Transitieve sluiting van een gerichte graaf is opnieuw een gerichte graaf, maar:
  - er wordt een nieuwe verbinding van i naar j toegevoegd indien er een weg bestaat van i naar j in de oorspronkelijke graaf.
- 3 algoritmen:
  1. **Diepte-of breedte-eerst zoeken**:
     - Spoor alle knopen op die vanuit een startknoop bereikbaar zijn en herhaal dit met elke knoop.
     - Voor ijle grafen $\rightarrow \Theta(n(n+m))$.
     - Voor dichte grafen $\rightarrow \Theta(n^3)$.
  2. **Met de componentengraaf**:
     - Interessant wanneer men verwacht dat de transitieve sluiting een dichte graaf zal zijn, want dan zijn veel knopen onderling bereikbaar, zodat er een beperkt aantal sterk samenhangende componenten zijn. Die kunnen in $\Theta(n+m)$ bepaald worden.
     - Maak dan de componentengraaf (kan in $O(n+m)$).
     - Als nu blijkt dat component j beschikbaar is vanuit component i, dan zijn alle knopen van j beruikbaar vanuit knopen van i.
  3. **Het algoritme van Warshall**:
     - Maak een reeks opeenvolgende $n \times n$ matrices $T^{(0)}, T^{(1)}, ..., T^{(n)}$ die logische waarden bevatten.
     - Element $t_{ij}^{(k)}$ duidt aan of er een weg tussen i en j met mogelijke intermediaire knopen $1, 2, ..., k$ bestaat.
     - Bepalen opeenvolgende matrices:

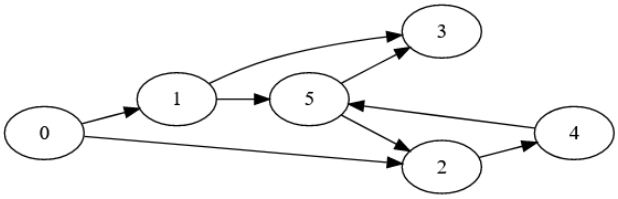
$$t_{ij}^{(0)} = \begin{cases} \text{onwaar} & \text{als } i \neq j \text{ en } g_{ij} = \infty \\ \text{waar} & \text{als } i = j \text{ of } g_{ij} < \infty \end{cases}$$

       en

$$t_{ij}^{(k)} = t_{ij}^{(k-1)} \text{ OF } \left(t_{ik}^{(k-1)} \text{ EN } t_{kj}^{(k-1)}\right) \qquad \text{voor } 1 \leq k \leq n$$

     - $T^{(n)}$ is de gezochte burenmatrix.
     - Alle berekeningen kunnen in dezelfde tabel T gebeuren. Er moet geen plaats voorzien zijn voor andere tabellen.
     - **Voorbeeld**
       - De initieële tabel $T^{(0)}$ is gewoon een kopie van de burenlijst van de graaf.

$$T^{(0)} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$



Figuur 1.1: Een gerichte graaf met 6 knopen.

◇ De tabel $T^{(1)}$ geeft een uitbreiding van $T^{(0)}$, waarbij knoop 1 een intermediaire knoop mag zijn in een weg naar knopen knopen. Het is logisch dat enkel knopen die 1 als buur hebben een nieuwe weg kunnen vinden.

$$T^{(1)} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

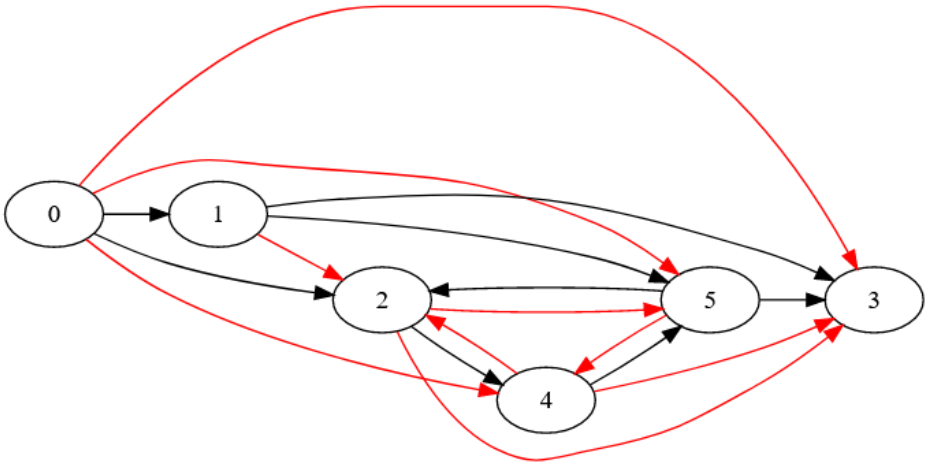
◇ De tabel $T^{(2)}$ geeft een uitbreiding van $T^{(1)}$, waarbij knoop 2 een intermediaire knoop mag zijn in een weg naar twee knopen. Het is logisch dat enkel knopen die 2 als buur hebben (in de nieuwe matrix $T^{(1)}$) een nieuwe weg kunnen vinden.

$$T^{(2)} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

◇ Via diezelfde redenering wordt uiteindelijk $T^{(5)}$ bekomen, die de burenmatrix voorstelt van de transitieve sluiting.

$$T^{(5)} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

◇ Figuur 1.2 toont de transitieve sluiting.



Figuur 1.2: De transitieve sluiting van de graaf op figuur 1.1. De transitieve sluiting bevat dezelfde verbindingen als de graaf (zwarte verbindingen) en ook de nieuwe verbindingen die de transitieve eigenschap vastleggen (rode verbindingen).