Algoritmer og Datastrukturer 2

Aflevering 5

Kristian Gausel[1], Lasse Alm[2], and Steffan Sølvsten[3]

[1]201509079@post.au.dk
[2]201507435@post.au.dk
[3]201505832@post.au.dk

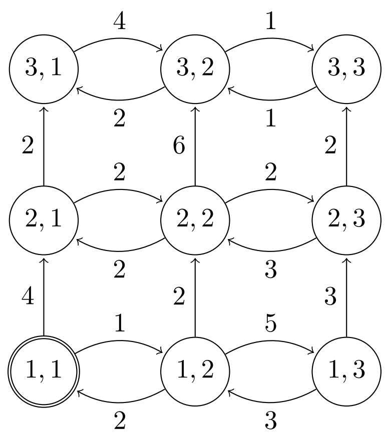
22. december 2016

1 Opgave 6

En *gitter-graf* er en orienteret graf med k rækker hver indeholdende k knuder for $k \geq 0$. For alle knuderne $v_{i,j}$ er kanterne defineret som følgende.

$$\begin{aligned} V =& \{v_{i,j} | 1 \leq i \leq k \vee 1 \leq j \leq k\} \\ E =& \{(v_{i,j}, v_{i,j+1}) | 1 \leq i \leq k \wedge 1 \leq j < k\} \\ & \{(v_{i,j}, v_{i,j-1}) | 1 \leq i \leq k \wedge 1 < j \leq k\} \\ & \{(v_{i,j}, v_{i+1,j}) | 1 \leq i < k \wedge 1 \leq j \leq k\} \end{aligned}$$

I resten af opgaven antages det, at alle kanterne har en ikke-negativ vægt. Hermed vil et eksempel på en graf for $k = 3$ være som i figur 1.



Figur 1: En vægtet gittergraf.

1.1 n og m udtrykt ved k

For n værende antallet af knuder og m antallet af kanter i en gitter-graf skal n og m udtrykkes som funktion af k.

En gitter-graf består af k rækker hver indeholdende k knuder, hvorfor antallet af knuder n er

$$n = k \cdot k = k^2 \tag{1}$$

Fra definitionen af kanterne E i afsnit 1 vides det, at der for alle k rækker er $k-1$ kanter både fra $v_{i,j}$ til $v_{i,j+1}$ og fra $v_{i,j}$ til $v_{i,j-1}$. Samtidig er der for alle rækker undtagen den k'te række k kanter fra $v_{i,j}$ til $v_{i+1,j}$.

$$m = k \cdot 2 \cdot (k-1) + k \cdot (k-1) = 3k^2 - 3k \tag{2}$$

1.2 Udførselstid af Dijkstra's algoritme som funktion af k

Udførselstiden, for at finde længden af den korteste vej fra $s = v_{1,1}$ til alle øvrige knuder i en gitter-graf ved brug af Dijkstra's algoritme, skal bestemmes som funktion af k.

Dijkstra's algoritme kører i $O((n+m)\lg n)$ tid ved brug af en heap og i $O(n^2+m)$ tid ved brug af et array. Indsættes n fra ligning 1 og m fra ligning 2 fra afsnit 1.1, så bliver dette

$$\begin{aligned}
\text{Heap}: \quad & O((n+m)\lg n) \\
= \quad & O((k^2 + (3k^2 - 3k))\lg k^2) \\
= \quad & O((4k^2 - 3k)\lg k^2) \\
= \quad & O(k^2 \lg k^2)
\end{aligned}$$

$$\begin{aligned}
\text{Array}: \quad & O(n^2 + m) \\
= \quad & O(k^4 + (3k^2 - 3k)) \\
= \quad & O(k^4 + 3k^2 - 3k) \\
= \quad & O(k^4)
\end{aligned}$$

1.3 Algoritme til korteste veje i O(m) tid

En algoritme, der finder længden af de korteste veje fra $s = v_{1,1}$ til alle øvrige knuder i en gitter-graf, skal bestemmes. Algoritmen skal køre i $O(m)$ tid.

1.3.1 Algoritme

Til bestemmelse af korteste veje, så benyttes algortimerne *Initialize-Single-Source* og *Relax*. [1, s. 648 - 649] Algoritmen gennemløber en række fra venstre til højre og tilbage igen, hvor først kun de højre kanter $(v_{i,j}, v_{i,j+1})$ relaxes og på tilbageløbet relaxes de venstre $(v_{i,j}, v_{i,j-1})$ og opadgående $(v_{i,j}, v_{i+1,j})$ kanter. Dette gentages for hver række startende i rækken $i = 1$. Pseudokode til algoritmen kan findes i kode 1.

Kode 1: Algoritme til at bestemme korteste veje i en gitter-graf

```
FindShortestPathInGG(G)
    //Get s (v) and the weightfunction w
    v = G[1,1]
    w = G.weightfunction

    Initialize-Single-Source(G, v)   //See: [1, p. 648]

    do  //For every row in G
        do  //Run through the whole row relaxing the right edges
            v_right = G[v.i, v.j+1]
            Relax(v, v_right, w)        //See: [1, p. 649]

            v = v_right
        while G[v.i, v.j+1] ≠ NIL

        do  //Run all the way back, relaxing the left edges and up
            v_left = G[v.i, v.j-1]
            Relax(v, v_left, w)

            v_up = G[v.i+1, v.j]
            if v_up ≠ NIL
                Relax(v, v_up, w)

            v = v_left
        while G[v.i, v.j-1] ≠ NIL

        //The last edge upwards is missed in the second loop
        v_up = G[v.i+1, v.j]
        if v_up ≠ NIL
            Relax(v, v_up, w)

        v = v_up
    while v ≠ NIL
```

1.3.2 Korrekthed

For korrekthed benyttes Lemma 1. [1, s. 650]

Lemma 1 *Hvis $p = \langle v_0, v_1, ..., v_k \rangle$ er den korteste vej fra $s = v_0$ til v_k, og kanterne i p relaxes i rækkefølgen $(v_0, v_1), (v_1, v_2), ..., (v_{k-1}, v_k)$, så vil $v_k.d = \delta(s, v_k)$. Denne egenskab holder ligegyldig at andre relaxing operationer udføres, også hvis disse udføres indbyrdes med relaxing af kanterne i p.*

I algoritmen relaxes kanterne indenfor en række i fra en ende til den anden, det vil sige først $(v_{i,j}, v_{i,j+1})$. Hermed relaxes med hensyn til alle mulige stier gennem rækken i, såfremt at kanterne $(v_{i-1,j}, v_{i,j})$ er relaxet korrekt. Kanterne op til række $i+1$, dvs $(v_{i,j}, v_{i+1,j})$ bør først relaxes efter at kanterne indenfor den $i$te række er relaxet. Derfor relaxes disse først ved tilbageløbet sammen med kanterne $(v_{i,j}, v_{i,j-1})$.

Hermed er der relaxet i en rækkefølge, der tager hensyn til alle mulige stier gennem grafen G, hvormed lemma 1 er opfyldt, hvorfor algoritmen er korrekt.

1.3.3 Tidskompleksitet

Lad $m = |E| \in G$. Algoritmen følger ikke alle m kanter gennem grafen, da kun kanten mellem to rækker i og $i+1$ følges i langs kanten $(v_{i,1}, v_{i+1,1})$. I samme gennemløb bliver alle m kanter relaxet præcis en gang. Da alle disse operationer tager konstant tid, så er tidskompleksiteten $O(m)$.

Litteratur

[1] Cormen, Thomas H. mfl.: *Introduction to Algorithms*, 3rd ed.

[2] Gausel, Kristian mfl.: Python implementation af algoritme
`https://github.com/yurippe/School/tree/master/dADS/Opgave%206%20-%20Gitter%20Graf`