Gegevensstructuren en algoritmen

Practicum 3: Image Compositing algoritme

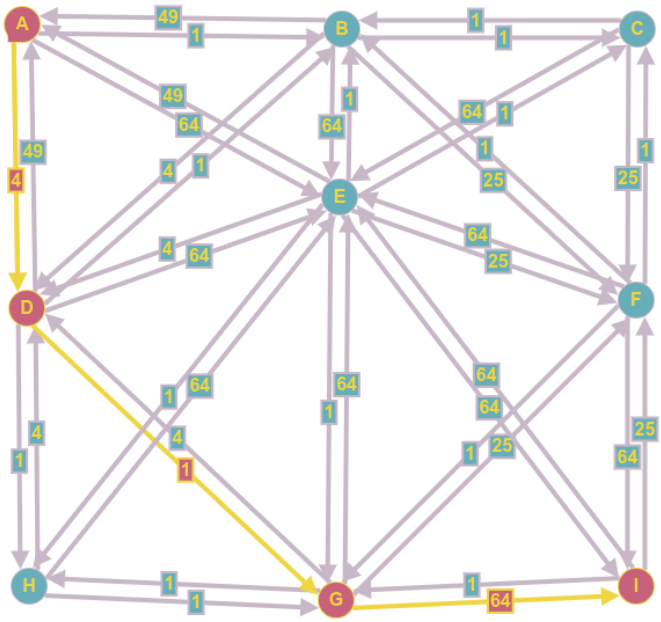
Author:
Stef Tweepenninckx, r0677232

18 mei 2018

Inleiding

Het derde en laatste practicum van Gegevensstructuren en Algoritmen gaat over de implementatie van een Image Compositing Algoritme. Dit algoritme voegt twee afbeeldingen samen. In deze variant is het belangrijk dat de grens tussen de twee afbeeldingen zo min mogelijk opvalt.
Om dit te verwezenlijken implementeren we *seam()*, *floodfill()* en *stitch()*. Seam bepaalt de grens tussen de twee afbeeldingen zodat deze zo min mogelijk opvalt. Floodfill gaat op basis van de gevonden grens de afbeeldingen aan elkaar hechten. Stitch combineert *seam()* en *floodfill()* tot een uitvoerbare method.

Kortste pad



Figuur 1: Grafe met aangeduid kortste pad

Het resulterende kortste pad, aangeduid in het geel, is:

$$A \implies D \implies G \implies I$$

met een totale kost van $49 + 4 + 1 + 64 = 118$.

Andere afstandsfunctie

Als we de nieuwe afstandsfunctie zouden gebruiken (gegeven in de opgave), houden we geen rekening meer met de blauwe component van een pixel. Aangezien we hierdoor minder berekeningen gaan doen, zal deze functie in de praktijk bij bijna iedere afbeelding een snellere uitvoeringstijd hebben.

Indien we gebruik maken van de oude afstandsfunctie, wordt eerst positie (0,1) gekozen. Daarna positie (1,0) en dan uiteindelijk eindpositie (1,1). Dijkstra heeft dus 3 stappen nodig om het einde te bereiken met de oude afstandsfunctie. Als we de nieuwe afstandsfunctie gebruiken zullen we rechtstreeks van (0,0) naar de eindpositite (1,1) gaan. In dit geval is er dus maar 1 stap nodig, wat duidelijk minder is en dus sneller.

(0,0,0) (0,0,0) (0,0,0) (0,0,0)	(0,0,0) (1,0,0) (1,1,0) (0,0,42)

Tabel 1: Twee images bij voorbeeld snellere uitvoeringstijd

Met de oude afstandsfunctie gaan we rechtstreeks van (0,0) naar eindpositie (1,1). De oude functie heeft dus 1 stap nodig om het einde te bereiken. Bij gebruik van de nieuwe afstandsfunctie zal Dijkstra eerst naar (0,1), dan naar (1,0) en dan pas naar eindpositie (1,1). In dit geval heeft de nieuwe functie 3 stappen nodig, meer dan de 1 stap van de oude functie en zal dus bijgevolg trager zijn dan de oude functie.

(0,0,0) (0,0,0) (0,0,0) (0,0,0)	(0,0,0) (0,0,45) (1,0,42) (1,1,0)

Tabel 2: Twee images bij voorbeeld trager uitvoeringstijd

Tijdscomplexiteit

Voorkomen van complexe vormen seam

Het voorkomen van complexe vormen van de seam kan gerealiseerd worden door middel van een kleine aanpassing. In het seam algoritme gebruik ik een methode om de buren van de huidige node te bekijken. Als we willen voorkomen dat de seam terug naar boven kan lopen, kunnen we de buren boven de huidige node gewoon uitsluiten uit deze methode. Als we willen voorkomen dat de seam terug naar links kan lopen, sluiten we de buren links van de huidig node uit.

Langste pad ipv kortste pad

Als we in plaats van het kortste pad, het langste, niet-cyclische pad zouden gebruiken, dan zal elke node deal uitmaken van de seam. Als elke node deel is van de seam, zal geen enkele pixel ingekleurd worden. We krijgen dus een "lege" afbeelding als resultaat.