|  |
| --- |
| Searching & sorting |
| Onderzoek naar kortste pad algoritmes |
| Practicum 2 |
|  |
| **Alexander van der Herik & Joost Kingma** |
| **18-12-2014** |

|  |
| --- |
|  |

# Inhoudsopgave

Inhoud

[Proces 3](#_Toc408393770)

[Experimenten 4](#_Toc408393771)

[Conclusie 5](#_Toc408393772)

# Proces

We begonnen het practicum met een aantal dingen die voor ons waren gemaakt zoals de algoritme en de bijbehorende library. We begonnen met het kijken naar de al gemaakte code in de library om een beeld te krijgen hoe het in elkaar zitten en hoe we mogelijk later het aantal knoppen en zijden kunnen tellen.

Nadat we ons hadden georiënteerd op de algoritme begonnen we code toe te voegen om de benodigde informatie er uit te halen. Uiteindelijk hebben we in de constructor van Dijkstra.java, 2 integers mee laten tellen door die hier te plaatsen :  
  
ook hadden we de lengte van het pad nodig naast de totale waarden van het pad. We deden dit door 1 simpele regel toe te voegen aan “public Iterable<DirectedEdge> pathTo(int v)”, namelijk :  
hier was “path” een stack van het pad.

System.out.print(","+path.size()); //length of path

while (!pq.isEmpty()) {

int v = pq.delMin();

count++; //knots

for (DirectedEdge e : G.adj(v)) {

relax(e);

edges++; //edges

}

}

Op dit moment konden we in de main.java beginnen met het loopen door onze input bestanden waar ook een aantal van ons zelf tussen zaten , we hebben uit eindelijk er voor gekozen om elk plaatje 10 keer te doen en dan het gemiddelde daar van te nemen om een preciezer tijd te krijgen , ook kozen we er voor om de tijd in nano seconde te doen omdat er met seconde soms het geval was dat het maar 0 seconde duurde.

# Experimenten

Nadat we de experimenten hadden gedaan hebben we een hele hoop data gekregen en dit in Excel gezet , we hadden het even netjes in een tabel gezet en een grafiek gemaakt met de knoppen ten opzichte van de tijd.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| bitmap | knopen | lengte korste pad | kost korste pad | tijd | zijden |
| i5.png | 360 | 32 | 348 | 256 | 2826 |
| i12.png | 600 | no path | no path | 356 | 4592 |
| i6.png | 239 | no path | no path | 359 | 1832 |
| i8.png | 767 | 27 | 380 | 362 | 5936 |
| i7.png | 1200 | 54 | 982 | 419 | 9184 |
| i4.png | 1149 | 108 | 1596 | 423 | 8827 |
| i21.png | 629 | no path | no path | 427 | 4830 |
| i9.png | 1200 | 78 | 912 | 429 | 9184 |
| i17.png | 1146 | 178 | 1928 | 435 | 8826 |
| i10.png | 1190 | no path | no path | 445 | 9131 |
| i15.png | 1200 | 34 | 376 | 457 | 9184 |
| i14.png | 1200 | 22 | 320 | 459 | 9184 |
| i19.png | 1028 | 187 | 2486 | 497 | 7944 |
| i20.png | 832 | 53 | 792 | 498 | 6329 |
| i2.png | 596 | no path | no path | 522 | 4566 |
| i16.png | 1200 | 38 | 760 | 523 | 9184 |
| i11.png | 1200 | 52 | 600 | 558 | 9184 |
| j29.png | 1322 | no path | no path | 565 | 10281 |
| i3.png | 1200 | 40 | 656 | 571 | 9184 |
| i13.png | 1200 | 224 | 2292 | 596 | 9184 |
| i18.png | 1137 | 41 | 690 | 635 | 8722 |
| y23.png | 239 | no path | no path | 754 | 1832 |
| j30.png | 2700 | 52 | 572 | 812 | 20974 |
| i1.png | 1200 | 74 | 1246 | 842 | 9184 |
| j33.png | 2699 | 117 | 1376 | 1,235 | 20966 |
| y24.png | 2330 | no path | no path | 1,651 | 18228 |
| y25.png | 2924 | 54 | 772 | 1,781 | 23056 |
| y22.png | 4365 | 91 | 1202 | 2,243 | 34090 |
| k31.png | 10529 | 61 | 1082 | 4,535 | 83009 |
| k32.png | 9985 | 227 | 3728 | 4899 | 78666 |
| Z26.png | 12164 | 758 | 9946 | 8,199 | 96270 |
| Z27.png | 19124 | 165 | 2726 | 12324 | 151316 |
| Z28.png | 17556 | 152 | 3082 | 13,722 | 140448 |

# Conclusie

Doordat we in de constructor van Dijkstra.java 2 integers mee hebben laten tellen, een timer hadden toegevoegd in de main.java en de lengte en kostte van het kortste pad hadden gemeten, hebben we veel relevante data gekregen waarmee we de BigO kunnen aantonen van het algoritme van Dijkstra.

Als we naar de data kijken die we hebben gemeten en de grafiek die we hieruit hebben gemaakt dan zien we dat wanneer het aantal knopen verdubbel, de tijd ook verdubbeld en het vrij lineair is met uitzondering van enkele punten. Dit kan bijvoorbeeld zijn omdat er in sommige “werelden” veel stukken zijn waar het pad niet langs kan en dus veel omwegen moet zoeken, of het is juist heel makkelijk is om het pad te vinden.

Doordat alles , een beetje grof gezien, vrij lineair is, is de BigO is dus N.