Toepassingen van meetkunde in de informatica
Academiejaar 2017 - 2018

Project: Bepaling van het Dichtste Puntenpaar

Dit project wordt bij voorkeur met twee gemaakt !

Het doel van het project is een aantal algoritmen voor de bepaling van het dichtste puntenpaar van een verzameling punten te ontwikkelen, te vergelijken, uit te testen en de rekencomplexiteit ervan te onderzoeken. Het dichtste puntenpaar van een verzameling van N punten kan bepaald worden met o.m. (a) een eenvoudig algoritme met rekencomplexiteit $O(N^2)$, (b) een doorlooplijnalgoritme, waarbij verscheidene varianten met verschillende rekencomplexiteit kunnen ontwikkeld worden.
Bij het **eenvoudig algoritme** met rekencomplexiteit $O(N^2)$ wordt voor alle puntenparen de afstand tussen de punten berekend en het paar met de kleinste afstand is het dichtste puntenpaar.
Een **doorlooplijnalgoritme** maakt gebruik van een doorlooplijn die zich van links naar rechts door de puntenverzameling beweegt. Dit vereist een *overgangspuntenlijst* waarin de puntenverzameling volgens x-coördinaat gesorteerd is. Voor het doorlooplijnalgoritme onderzoek je **twee varianten** (zie bijgevoegde figuur):

Variant 1. Bij de behandeling van een punt p_i (d.w.z. de doorlooplijn gaat door p_i) wordt dit punt getest met de punten in de verticale strook V met breedte d, waarbij d de afstand is tussen de punten die het "voorlopig dichtste puntenpaar" vormen (zie figuur). Immers, enkel punten in deze strook komen in aanmerking om met p_i een dichtste puntenpaar te vormen.

Vermits we in de overgansgpuntenlijst de punten gesorteerd naar x-coördinaat bijhouden, moeten we bij de behandeling van p_i enkel testen met punten die in deze sortering vóór p_i liggen totdat we een punt tegenkomen met een "te kleine" x-coördinaat (d.w.z. buiten de strook V).

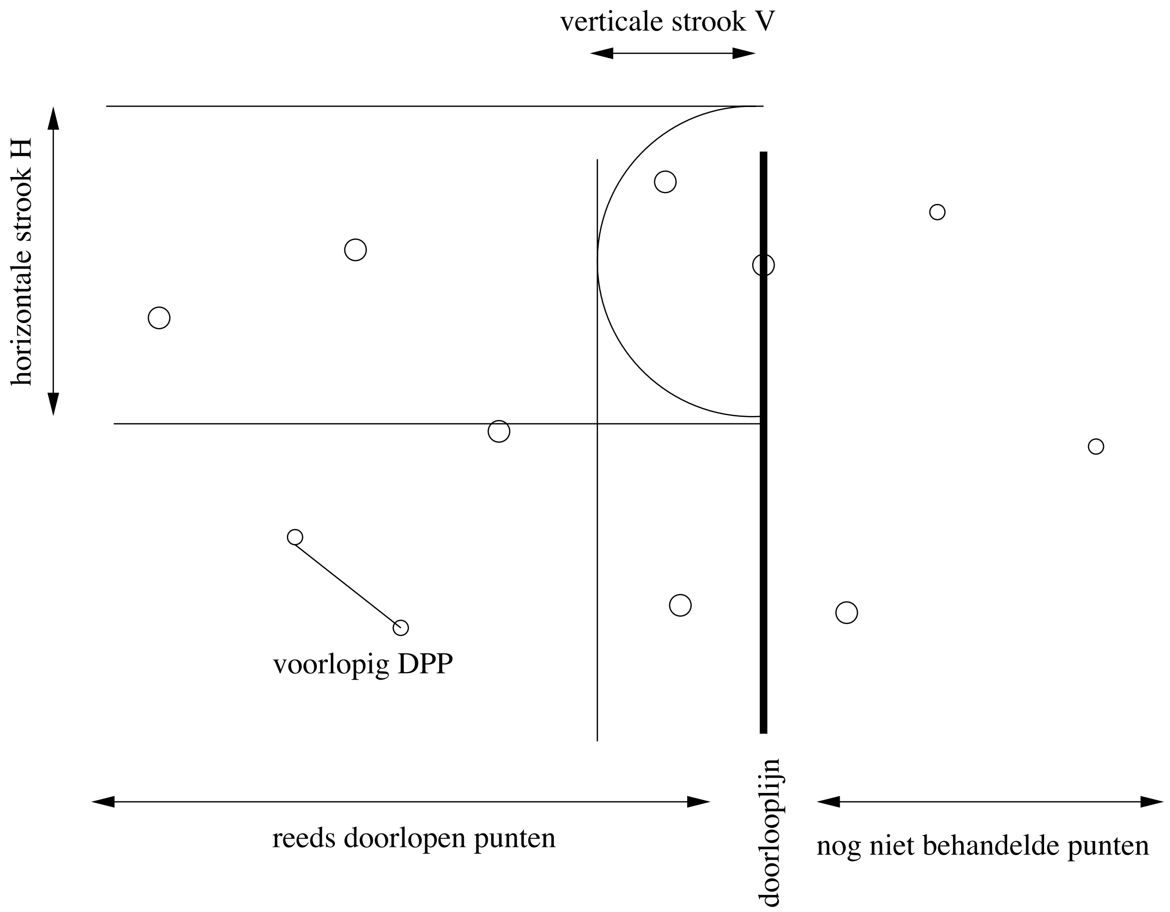
Ga na dat de rekencomplexiteit van dit algoritme (na sortering volgens x-coördinaat) $O(NK_{gem})$ is met K_{gem} =gemiddeld aantal punten die in de strook V liggen (uitgemiddeld over $i = 1,\ldots,N$).

Variant 2. De punten links van de doorlooplijn (de reeds behandelde punten) worden gesorteerd volgens y-coördinaat in een gegevensstructuur die we de *doorlooplijnstatus* t noemen. We kunnen dan bij de behandeling van een punt p_i het punt p_i toevoegen aan t en vervolgens de gesorteerde lijst van punten links van de doorlooplijn, doorlopen vanaf p_i naar boven en naar onder tot we een punt tegenkomen dat buiten de horizontale strook H met breedte $2d$ ligt. Immers, enkel punten in deze strook komen in aanmerking om met p_i een dichtste puntenpaar te vormen.

Indien de behandeling van punt p_i (gemiddeld) $O(\log N)$ bewerkingen vergt is de rekencomplexiteit van het hele algoritme $O(N \log N)$. Elk punt kan in (gemiddeld) $O(\log N)$ bewerkingen behandeld worden door de doorlooplijnstatus t bij te houden in een gebalanceerde zoekboom. Op zulke gegevensstructuur kunnen immers alle nodige bewerkingen (boven(t,p), onder(t,p), verwijder(t,p), voegtoe(t,p) – zie verder) in $O(\log K)$ elementaire operaties worden uitgevoerd, met K het aantal elementen in de zoekboom.

Opmerking: indien we tijdens de behandeling van p_i een punt tegenkomen dat buiten de verticale strook V ligt, kunnen we dit verwijderen uit de doorlooplijnstatus t. Heeft dit zin? Is dit nodig?

Voor meer informatie: zie `http://www.cs.mcgill.ca/~cs251/ClosestPair/ClosestPairPS.html`



1 Hoog-niveau beschrijving van verscheidene algoritmes

Opgave

Schrijf voor het eenvoudig algoritme en voor beide varianten van het doorlooplijnalgoritme een *hoog-niveau*[1] algoritme, gebruik makend van de bijgevoegde beschrijving van gegevensstructuren en bewerkingen.

Gegevensstructuur en bewerkingen voor het doorlooplijnalgoritme

- *Gegevensstructuur:*

rij	overgangspuntenlijst: array van punten, gesorteerd naar stijgende x-coördinaat (array van lengte N, $rij[i].x$ en $rij[i].y$ zijn de x- en y-coördinaat van $rij[i]$)
$dpp1$, $dpp2$	de twee punten die het dichtste puntenpaar vormen
d	afstand tussen het dichtste puntenpaar $\left(= \sqrt{(dpp1.x - dpp2.x)^2 + (dpp1.y - dpp2.y)^2}\right)$

[1] d.w.z. op de manier waarop algoritmes in de cursus en de oefenzittingen worden voorgesteld, onafhankelijk van een specifieke programmeertaal

- *Bijkomende gegevensstructuur voor variant 2:*

t	doorlooplijnstatus: gegevensstructuur waarin punten links van de doorlooplijn opgeslagen zijn, gesorteerd naar stijgende y-coördinaat
boven(t,p)	geeft als resultaat het punt dat in t boven het punt p ligt
onder(t,p)	geeft als resultaat het punt dat in t onder het punt p ligt
verwijder(t,p)	verwijdert het punt p uit t
voegtoe(t,p)	voegt het punt p toe aan t

2 Implementatie en testen

1. Implementeer het eenvoudig algoritme en de *eerste variant* van het doorlooplijnalgoritme om het dichtste puntenpaar te berekenen. Je mag deze implementaties maken in een programmeertaal naar keuze.
2. Gebruik vervolgens deze implementaties om experimenten uit te voeren op willekeurige en speciaal gekozen puntenverzamelingen (zie hieronder) en ga hierbij de rekentijd na.
3. Breid vervolgens je implementatie uit naar het berekenen van het dichtste puntenpaar van een verzameling van M-dimensionale punten ($M > 2$). Voor stijgende M gaat het voordeel van het doorlooplijnalgoritme geleidelijk aan verloren. Waarom? Voor welke waarde van M is er geen voordeel meer? Waarom?
4. *Facultatief:* Implementeer, test en evalueer de *tweede variant* van het doorlooplijnalgoritme voor het twee-dimensioneel geval ($M = 2$).
5. Een kritische en grondige *bespreking* van de resultaten van je experimenten is belangrijk – zie verder. Het is daarvoor essentieel dat je je experimenten goed kiest!

Stel willekeurige puntenverzamelingen op van verschillende grootten en vergelijk de rekentijden van beide implementaties.
De totale tijdscomplexiteit van de eerste variant van het doorlooplijnalgoritme is $O(N \log N + NK_{gem})$, waarbij K_{gem} het gemiddelde aantal elementen in de verticale strook V is (het $N \log N$ gedeelte komt van de sortering volgens x-coördinaat). We vermoeden dat voor het twee-dimensionale geval ($M = 2$) K_{gem} meestal zeer klein zal zijn in vergelijking met N zodat dit algoritme veel sneller zal zijn dan het $O(N^2)$-algoritme. Aan jou om te onderzoeken of dit vermoeden juist is!
Onderzoek ook hoe groot K_{gem} en K_{max} (= maximaal aantal punten in de verticale strook V bij de behandeling van een punt p_i) in de praktijk worden en of deze waarden afhangen van N en M.
Zoek ook een *worst-case* puntenverzameling voor de eerste variant van het doorlooplijnalgoritme voor $M = 2$. Hoe evolueert de rekentijd, K_{gem} en K_{max} in functie van N voor dit *worst-case* geval?

2.1 Uitvoerbaar bestand, invoer en uitvoer

Maak een *uitvoerbaar bestand* waarmee de verschillende algoritmen eenvoudig getest kunnen worden. De *invoer* wordt gegeven als een txt-bestand dat op de volgende manier is opgebouwd (zie Tabel 1):

- De eerste lijn bevat één getal: het nummer van het algoritme dat gebruikt moet worden: 1 (eenvoudige algoritme), 2 (eerste variante van het doorlooplijnalgoritme) of 3 (tweede variante van het doorlooplijnalgoritme);

- De tweede lijn bevat één getal: de dimensie van de punten $M \geq 2$;
- De derde lijn bevat één getal: het aantal punten N;
- De volgende N lijnen zijn een opeenvolging van M reële getallen, nl. de coördinaten $x_i,\ y_i, \ldots$ van een punt, van elkaar gescheiden door een spatie.

1		
3		
5		
0.200000000000000	0.200000000000000	0.100000000000000
0.500000000000000	0.400000000000000	0.200000000000000
0.700000000000000	0.600000000000000	0.300000000000000
0.350000000000000	0.600000000000000	0.100000000000000
0.200000000000000	0.800000000000000	0.100000000000000

Tabel 1: voorbeeld input.txt

De *uitvoer* van het gekozen algoritme moet weggeschreven worden naar een txt-bestand dat er als volgt moet uitzien (zie Tabel 2):

- Het bestand bevat slechts één regel: Dit algoritme is niet geïmplementeerd.

  OF

- Lijnen 1 en 2 bevatten telkens M reële getallen, van elkaar gescheiden door een spatie, nl. de coördinaten $x_i,\ y_i, \ldots$ van de punten die het dichtste puntenpaar vormen;
- Lijn 3 bevat één getal: de afstand tussen de twee dichtste punten;
- Lijn 4 bevat een getal dat de uitvoeringstijd weergeeft in milliseconden.

0.350000000000000	0.600000000000000	0.100000000000000
0.200000000000000	0.800000000000000	0.100000000000000
0.250000000000000		
13		

Tabel 2: voorbeeld output.txt

3 Bespreking van de resultaten en verslag

Bespreek je experimenten bondig maar kritische en grondig! Beschrijf enkel de resultaten van je werk (geen opgave herhalen, ...) en verklaar de rekentijden. Geef hierin zeker de volgende dingen weer:

1. de hoogniveau beschrijvingen van de algoritmen (zie §1)
2. Een beschrijving (eventueel programma) van hoe je je puntenverzamelingen hebt opgesteld, en hoe je een worst-case puntenverzameling kan opstellen.
3. Vergelijk de rekentijden van het eenvoudig en de eerste variant van het doorlooplijnalgoritme in 2D ($M = 2$). Zet deze uit in grafieken. Denk goed na over de schaal die je gebruikt.

4. Onderzoek hoe K_{gem} en K_{max} verlopen in functie van N voor het 2D-doorlooplijnalgoritme.

5. Vergelijk de rekentijden van het doorlooplijnalgoritme in 2D ($M = 2$) en in hogere dimensies ($M > 2$) voor hetzelfde aantal punten. Wat merk je op?

6. Wat is de evolutie van K_{gem} voor stijgend aantal dimensies?

7. *Facultatief:* Vergelijk de rekentijden van de beide varianten van het doorlooplijnalgoritme in 2D ($M = 2$) .

8. Aarzel niet om ook eventuele andere interessante bevindingen uit je experimenten in het verslag op te nemen.

4 Praktische tips

1. Willekeurige puntenverzamelingen: voor het doorlooplijnalgoritme voor $M = 2$ kan je perfect werken met punten in het eenheidsvlak (x- en y-coördinaten tussen 0 en 1). Je kan hierin makkelijk willekeurige punten genereren door gebruik te maken van een randomgenerator die reële getallen tussen 0 en 1 genereert. Java levert zulke generator met de methode `random()` in de klasse `java.lang.Math`. In Matlab kan je de functie `rand` gebruiken.

2. Meten van rekentijden: meet de rekentijd van het volledige programma, maar zonder het inlezen en/of genereren van de gegevens (puntenverzameling). Indien je in Java programmeert kan je de methode `currentTimeMillis()` uit de klasse `java.lang.System` gebruiken. In Matlab zijn hiervoor de functies `tic` en `toc` beschikbaar. Het volstaat om de "huidige" tijd juist voor en juist na de berekeningen te meten; het verschil tussen beide tijden geeft de uitvoeringstijd. Voor kleine N zal de uitvoeringstijd van uw programma zo klein zijn dat het moeilijk of zelfs onmogelijk is om een nauwkeurige tijdsmeting te doen. Dit kan opgevangen worden door de 'kern' van het algoritme een (groot aantal) keer na elkaar uit te voeren (redundante berekeningen met als enig doel de uitvoeringstijd te verhogen zodat de tijdsmeting voldoende nauwkeurig gebeurt. Het is van groot belang dat precies dezelfde bewerkingen herhaald worden (let bv. op met sorteringen ...).

3. Voor de implementatie van de doorlooplijnstatus bij de eerste variante van het doorlooplijnalgoritme kan je gebruik maken van een voorgedefinieerde binaire zoekboom (bijvoorbeeld voor Java `http://algs4.cs.princeton.edu/32bst/BST.java.html`).

4. Je hoeft geen rekening te houden met *speciale gevallen*, maar je moet wel aangeven in uw verslag voor welke speciale gevallen (randgevallen) uw algoritme niet werkt.

5 Project indienen

De deadline van het project is **vrijdag 25 mei 2018** om 14u.

5.1 Code

De code moet opgeladen worden op Toledo (door 1 persoon van het groepje indien je met twee gewerkt hebt). Zet al de code in één zip-bestand met jullie achternamen in de bestandsnaam,
bv. codeJanssensPeeters.zip. Het zip-bestand moet zeker bevatten:

- het uitvoerbaar bestand (zie §2.1) waarmee wij je code kunnen testen. Het programma moet uitvoerbaar zijn op de computers van de computerklas A00.124 (http://www.cs.kuleuven.be/cs/system/wegwijs/computerklas/).

- een invoerbestand `invoerpunten.txt` en het bijhorende uitvoerbestand `uitvoerpunten.txt`

- een bestand `leesmij.txt` waarin kort beschreven staat hoe jouw programma moet uitgevoerd worden

Het verslag hoort niet thuis in dit zip-bestand !

5.2 Verslag

Het verslag moet afgegeven worden in de studentenbrievenbus in 200A. Vergeet niet je naam, het vak en de bestemming (N. Scheerlinck) te vermelden! Indien je met twee hebt gewerkt, volstaat één verslag met beide namen op. Het verslag moet ook (apart) opgeladen worden op Toledo als pdf-bestand (door 1 persoon van het groepje indien je met twee gewerkt hebt).