Praktische Opdracht Informatica Leerjaar 5

Routeplanner voor een Bezorgservice

Introductie

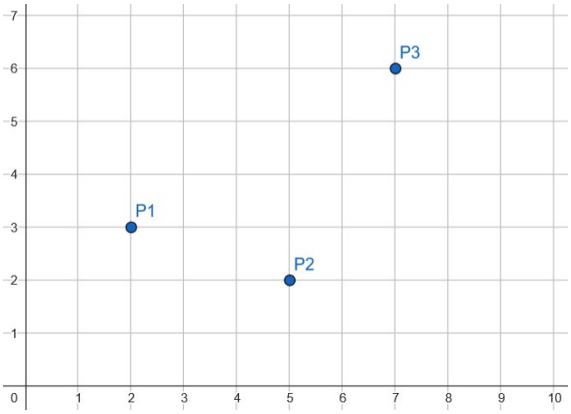
Stel je voor: je werkt bij een klein bezorgbedrijf dat pakketten aflevert in een stad. Je baas vraagt jou om een systeem te bouwen dat helpt om de beste route te plannen voor de bezorger. Hoe bepaal je in welke volgorde je de pakketten bezorgt? Hoe sla je de locaties en bestellingen slim op? En kun je met een algoritme een efficiëntere route vinden dan gewoon “de volgorde van binnenkomst”?

In deze opdracht ga je een eenvoudige webapplicatie bouwen met behulp van:

- PHP (om de logica en algoritmes te maken)
- SQL / phpMyAdmin (voor opslag van locaties en bestellingen)
- Algoritmiek (je vergelijkt manieren om routes te bepalen, zoals brute-force en greedy)

Je werkt aan meerdere deelopdrachten, en sluit af met een werkend prototype voorzien van helder commentaar bij de gebruikte algoritmes.

In deze opdracht werken we met de euclidische afstand. De berekening van de afstanden gaat dan als volgt:



Je ziet hierboven de punten op de kaart. De onderstaande formule berekent de **Euclidische afstand** tussen twee punten $P_1(x_1, y_1)$ en $P_2(x_2, y_2)$ in een vlak. Je neemt het verschil in x en in y, **kwadrateert** beide verschillen, **telt** ze op en neemt daarvan de **wortel**:

$$afstand = \sqrt{\{(x_2 - x_1)^2 + (y_2 - y_1)^2\}}$$

Dit is direct afgeleid van de stelling van Pythagoras; de wortel zorgt ervoor dat de uitkomst weer in dezelfde eenheid is als de coördinaten. (Tip: als je alleen afstanden wilt vergelijken, mag je de wortel weglaten, omdat dat hetzelfde ordeverschil geeft.)

Leerdoelen

Na deze opdracht kun je:

- Een eenvoudige database opzetten met twee gekoppelde tabellen
- Gegevens ophalen en verwerken met PHP en SQL
- Een route-algoritme schrijven en vergelijken qua efficiëntie
- Inzien hoe algoritmes toegepast worden in realistische situaties

Opdracht 1: De database opzetten

Doel van deze opdracht

Je maakt een eenvoudige database aan voor een bezorgservice. Deze bevat informatie over **adressen** en **bestellingen**. Je voert dit uit in **phpMyAdmin**.

Wat je gaat leren

- Hoe je een database aanmaakt in phpMyAdmin
- Hoe je tabellen aanmaakt met de juiste kolommen en gegevenstypen
- Hoe je een eenvoudige relatie legt tussen twee tabellen (via een foreign key)

Stap 1: Maak de database aan

1. Ga naar phpMyAdmin
2. Maak een nieuwe database aan met de naam:
3. routeplanner

Stap 2: Maak de tabellen aan

We maken 2 tabellen aan. Een tabel met adressen van de klanten (let op volgens de x, y coördinaten zoals bij de introductie aangegeven) en een tabel met de bestellingen per dag en naar welke locatie die moeten worden gebracht.

Tabel adressen

Veldnaam	Type	Extra

Tabel bestellingen

Veldnaam	Type	Extra

Maak een relatie tussen de tabellen.

Stap 3: Voeg testdata toe

Voeg handmatig in phpMyAdmin logische testdata in.

Wat is het Greedy-algoritme?

Het **greedy-algoritme** (in het Nederlands: het **gulzige algoritme**) maakt bij elke stap **de keuze die op dat moment het beste lijkt**, zonder verder vooruit te kijken. Het is eenvoudig, snel en vaak *goed genoeg*, maar **niet altijd optimaal**.

Greedy in de Routeplanner

Stel: je bezorger moet 5 pakketten afleveren op verschillende locaties in de stad. Hij start bij het depot (bijv. Centrum).

Hoe werkt het greedy-algoritme in deze situatie?

1. Begin bij de startlocatie (bijv. x=2, y=3).
2. Zoek van alle nog onbezochte locaties degene die het dichtstbij ligt (bijvoorbeeld via de afstand tussen de coördinaten).
3. Ga daarheen.
4. Herhaal stap 2 en 3 totdat alle locaties bezocht zijn.

Je maakt dus **bij elke stap de 'kortste sprong'**. Het algoritme kijkt niet naar de hele route, alleen naar wat nú het kortst is.

Voorbeeld

Je begint op (2, 3). De andere locaties zijn:

- (5, 2) → Station
- (7, 6) → Winkelgebied

Afstanden:

- Naar Station: $\sqrt{[(5-2)^2 + (2-3)^2]} = \sqrt{10} \approx 3.2$
- Naar Winkelgebied: $\sqrt{[(7-2)^2 + (6-3)^2]} = \sqrt{34} \approx 5.8$

Dus ga je eerst naar Station. Vanaf daar kijk je opnieuw: wat is nu het dichtstbij? Enzovoort.

Voordelen van greedy

- Simpel te begrijpen en te programmeren
- Snel: je hoeft niet alle mogelijke routes te berekenen
- Werkt vaak redelijk goed

Nadelen van greedy

- Niet altijd de optimale route
  Soms zorgt een "gulzige" keuze in het begin voor een slechte totaalroute.
- Geen garantie op de kortste weg

Bij het greedy-algoritme kies je telkens de dichtstbijzijnde volgende locatie. Je hoopt dat dat uiteindelijk de kortste route oplevert, maar dat is niet gegarandeerd.

Opdracht 2: Greedy algoritme

Situatie:

Een bezorger begint in **Centrum (2,3)** en moet pakketten afleveren op:

- Station (5,2)
- Winkelgebied (7,6)
- Park (3,7)

Stap 1: Teken alle punten op een coördinatenstelsel

Gebruik ruitjespapier of een leeg assenstelsel. Geef elk punt een naam en coördinaten.

Stap 2: Pas het greedy-algoritme toe

1. Begin bij Centrum
2. Bepaal welk punt het dichtstbij is
3. Ga daarheen
4. Herhaal tot alle punten bezocht zijn

Teken deze route en bereken alle afstanden met de afstandsformule:

Stap 3: Maak het algoritme in PHP

In deze stap mag je de data nog in een variabele zetten. Je hoeft dus nog niets met de database.

Stap 4: Gebruik de data uit de database

In deze stap moet je de data gebruiken uit de database.

Brute force

Het brute-force-algoritme is een aanpak waarbij je **alle mogelijke volgordes van locaties uitprobeert** en vervolgens de route kiest die het **kortst** is. In de context van een routeplanner betekent dit dat je vanaf een startpunt begint en dan voor alle mogelijke volgordes waarin je de bezorglocaties kunt bezoeken de totale afstand uitrekent. Je vergelijkt al die routes en kiest de route met de **kleinste totale afstand**.

Stel je voor dat een bezorger begint in het Centrum en pakketten moet afleveren op drie locaties: Station, Winkelgebied en Park. Dan zijn er 3! = 6 mogelijke volgordes waarin hij die drie locaties kan bezoeken. Voor elke volgorde bereken je de totale afstand van de route: vanaf het startpunt naar het eerste afleverpunt, van daar naar het volgende, enzovoort. De volgorde met de kortste afstand is dan de optimale route.

Het voordeel van brute-force is dat je altijd de beste oplossing vindt. Je weet zeker dat je geen kortere route over het hoofd ziet, omdat je alles uitprobeert. Het nadeel is dat het **snel onpraktisch wordt bij meer locaties**. Voor 4 locaties moet je al 24 routes proberen, voor 6 locaties zijn het er 720. Bij 10 locaties zijn het er meer dan 3,5 miljoen. Het algoritme is dus **niet schaalbaar**, en daarom vooral bruikbaar bij een klein aantal punten.

In deze opdracht kun je het brute-force-algoritme gebruiken als vergelijking met het greedy-algoritme. Greedy is snel en eenvoudig, maar niet altijd optimaal. Brute-force is traag, maar

levert altijd de kortste route op. Door de uitkomsten en tijden van beide methodes naast elkaar te zetten, krijg je inzicht in de **efficiëntie** van algoritmes.

Voorbeeld

Situatie:
De bezorger start in **Centrum (2,3)** en moet pakketten afleveren op drie locaties:

- Station (5,2)
- Winkelgebied (7,6)
- Park (3,7)

We zoeken de kortste route door **alle mogelijke volgordes van de drie afleverpunten** te berekenen (brute-force).

Stap 1: Alle permutaties van de afleverpunten

Omdat er 3 punten zijn, zijn er 3! = 6 mogelijke routes (zonder terug naar start). We noteren alleen het gedeelte ná het startpunt (we negeren even dat de bezorger ook weer terug moet):

1. Centrum → Station → Winkelgebied → Park
2. Centrum → Station → Park → Winkelgebied
3. Centrum → Winkelgebied → Station → Park
4. Centrum → Winkelgebied → Park → Station
5. Centrum → Park → Station → Winkelgebied
6. Centrum → Park → Winkelgebied → Station

Stap 2: Coördinaten van de punten

- Centrum: (2,3)
- Station: (5,2)
- Winkelgebied: (7,6)
- Park: (3,7)

Stap 3: Afstanden berekenen met de afstandsformule

afstand=(x2−x1)2+(y2−y1)2\text{afstand} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}

We ronden af op 2 decimalen.

Route 1: Centrum → Station → Winkelgebied → Park

- Centrum → Station: $\sqrt{[(5-2)^2 + (2-3)^2]} = \sqrt{(9 + 1)} = \sqrt{10} \approx 3.16$
- Station → Winkelgebied: $\sqrt{[(7-5)^2 + (6-2)^2]} = \sqrt{(4 + 16)} = \sqrt{20} \approx 4.47$
- Winkelgebied → Park: $\sqrt{[(3-7)^2 + (7-6)^2]} = \sqrt{(16 + 1)} = \sqrt{17} \approx 4.12$
- **Totaal: 3.16 + 4.47 + 4.12 = 11.75**

Route 2: Centrum → Station → Park → Winkelgebied

- Centrum → Station: 3.16
- Station → Park: $\sqrt{[(3-5)^2 + (7-2)^2]} = \sqrt{(4 + 25)} = \sqrt{29} \approx 5.39$
- Park → Winkelgebied: $\sqrt{[(7-3)^2 + (6-7)^2]} = \sqrt{(16 + 1)} = \sqrt{17} \approx 4.12$
- **Totaal: 3.16 + 5.39 + 4.12 = 12.67**

Route 3: Centrum → Winkelgebied → Station → Park

- Centrum → Winkelgebied: $\sqrt{[(7-2)^2 + (6-3)^2]} = \sqrt{(25 + 9)} = \sqrt{34} \approx 5.83$
- Winkelgebied → Station: 4.47
- Station → Park: 5.39
- **Totaal: 5.83 + 4.47 + 5.39 = 15.69**

Route 4: Centrum → Winkelgebied → Park → Station

- Centrum → Winkelgebied: 5.83
- Winkelgebied → Park: 4.12
- Park → Station: 5.39
- **Totaal: 5.83 + 4.12 + 5.39 = 15.34**

Route 5: Centrum → Park → Station → Winkelgebied

- Centrum → Park: $\sqrt{[(3-2)^2 + (7-3)^2]} = \sqrt{(1 + 16)} = \sqrt{17} \approx 4.12$
- Park → Station: 5.39
- Station → Winkelgebied: 4.47
- **Totaal: 4.12 + 5.39 + 4.47 = 13.98**

Route 6: Centrum → Park → Winkelgebied → Station

- Centrum → Park: 4.12
- Park → Winkelgebied: 4.12
- Winkelgebied → Station: 4.47
- **Totaal: 4.12 + 4.12 + 4.47 = 12.71**

Stap 4: Conclusie

De kortste route is:
Centrum → Station → Winkelgebied → Park
met een totale afstand van **11.75**.
Deze route wordt dus gekozen door het brute-force-algoritme.

Samenvatting:
Het brute-force-algoritme heeft alle 6 mogelijke routes doorgerekend. Alleen op die manier weet je zeker dat je de kortste route kiest. Dit kost relatief veel rekentijd, maar is in dit voorbeeld nog goed te overzien.

Opdracht 3: Brute force algoritme

Gebruik dezelfde data als in opdracht 2 van het Greedy algoritme.

Stap 1: Maak het algoritme in PHP

In deze stap mag je de data nog in een variabele zetten. Je hoeft dus nog niets met de database.

Stap 2: Gebruik de data uit de database

In deze stap moet je de data gebruiken uit de database.

Opdracht 4: Applicatie afmaken

De applicatie moet nog verder worden afgemaakt. Hieronder staat een lijst met eisen waar de applicatie aan moet voldoen:

- Overzicht van alle adressen
- Overzicht van alle bestellingen, inclusief filter op een bepaalde dag
- Valide invoer van nieuwe adressen
- Valide invoer van nieuwe bestelling
- Uitvoeren Greedy algoritme voor een bepaalde dag
- Uitvoeren Brute Force voor een bepaalde dag
- Een éénvoudige maar bruikbare user interface
- De query's zijn beveiligd tegen SQL-injectie

Dit zou leuk zijn

- Bewerken van adressen en bestellingen
- Verwijderen van adressen en bestellingen
- Een goede usability
- Inzicht in de efficiëntie en effectiviteit van de 2 algoritmes

Uitgangspunten

- De opdracht wordt in tweetallen gemaakt. Je maakt het samen en bent in staat om individueel het ingeleverde product toe te lichten.
- De routeplanner heeft een startpunt, dit is tevens het eindpunt. Deze is elke dag gelijk. In het algoritme hoeft geen (mag wel) rekening met het eindpunt te worden gehouden. De route mag dus stoppen op het laatste adres.
- We werken met x, y coördinaten en dus niet met echte GPS gegevens.
- Je houdt je aan de regels voor informatica PO's (https://mijn-in.nl/algemeen/pages/poregels)

Reflectievragen (inleveren, kort). Deze vragen maak je individueel.

1. Waar faalt Greedy in jullie data en waarom?
2. Hoe schaalbaar is Brute Force? Zou je dit in productie willen gebruiken?
3. Welke ontwerpkeuze in de database bleek later handig of onhandig?
4. Wat zou je als volgende stap verbeteren en waarom?