Learning by Doing

Keuzethema Algoritmiek

Workshop op de i&i-conferentie 2022

Paul Bergervoet en Renske Weeda – Informatica-actief

10 november 2022

Programma voor deze workshop

Bespreking eindtermen in het kernprogramma en het keuzethema
Toelichting op de keuzes die gemaakt zijn in de module algoritmiek
- inhoud,
- wijze van formuleren/programmeren,
- niveaus in opdrachten, werkvormen

Walkthrough, met speciale aandacht voor
- ontwerpactiviteiten
- werkvormen in de opdrachten

Eindtermen keuzethema Algoritmiek

Subdomein G1: Complexiteit van algoritmen

31. De kandidaat kan
(havo:) van gegeven algoritmen de complexiteit vergelijken, en kan klassieke ‘moeilijke’ problemen herkennen en benoemen.
(vwo:) het verschil tussen exponentiële en polynomiale complexiteit uitleggen, kan algoritmen op basis hiervan onderscheiden, en kan klassieke ‘moeilijke’ problemen herkennen en benoemen.

Subdomein G2: Berekenbaarheid

32. De kandidaat kan berekeningen op verschillende abstractieniveaus karakteriseren en relateren, en kan klassieke *onberekenbare* problemen herkennen en benoemen.

Subdomein G3: Logica

33. De kandidaat kan eigenschappen van digitale artefacten uitdrukken in logische formules.

Algoritmen in kerndomein B

Eindtermen in het kerndomein B liggen al behoorlijk hoog, je kunt ze maar zeer beperkt doen in het basisprogramma.

> *14: De kandidaat kan een oplossingsrichting voor een probleem uitwerken tot een algoritme, daarbij standaardalgoritmen herkennen en gebruiken, en de correctheid en efficiëntie van digitale artefacten onderzoeken via de achterliggende algoritmen.*

Aspecten:

- Algoritmen beoordelen op efficiëntie
- Standaardalgoritmen (bv. lineair zoeken, min/max, filteren, sorteren)
- Correctheid

Afbakening kern en keuze

Eindterm 14 in kernprogramma gaat vooral over bedenken/ontwerpen
Eindtermen G1 en G2 gaan sterk over abstracte aspecten

Geen gezonde/levensvatbare inhoud als je niet op de ingeslagen weg in de kerndomeinen doorgaat en daar de eindtermen van het keuzethema aan ***toevoegt***.
Onze keuze: nog steeds ontwerpen, correctheid + formele efficiëntie, maar op een bredere en moeilijker klasse van algoritmen.

Welke algoritmen?

Efficiëntieklassen

Efficiëntieklasse	*Grote-O*	*Type probleem*	*Voorbeeldalgoritmen*
Logaritmisch	***O(lg(n))***	Binair zoeken	Binair zoeken in gesorteerde lijst, zoekbomen
Lineair	***O(n)***	Lijst of tekst doorlopen	String matching, Horspool, lineair zoeken
(geen naam)	***O(n*lg(n))***	Sorteren - verdeel&heers	Quicksort, Mergesort
Kwadratisch	***O(n^2)***	Sorteren - simpel, Combi's van twee	Bubble sort, Insertion sort
Exponentieel	***O(2^n)***	Kiezen uit verzameling	Brute kracht opspannende boom, bagage kiezen
Exponentieel, faculteit	***O(n!)***	Volgorde	Brute kracht handelsreizigerprobleem

Didactische aanpak

Niet gericht op het *leren* van (bestaande) algoritmes.

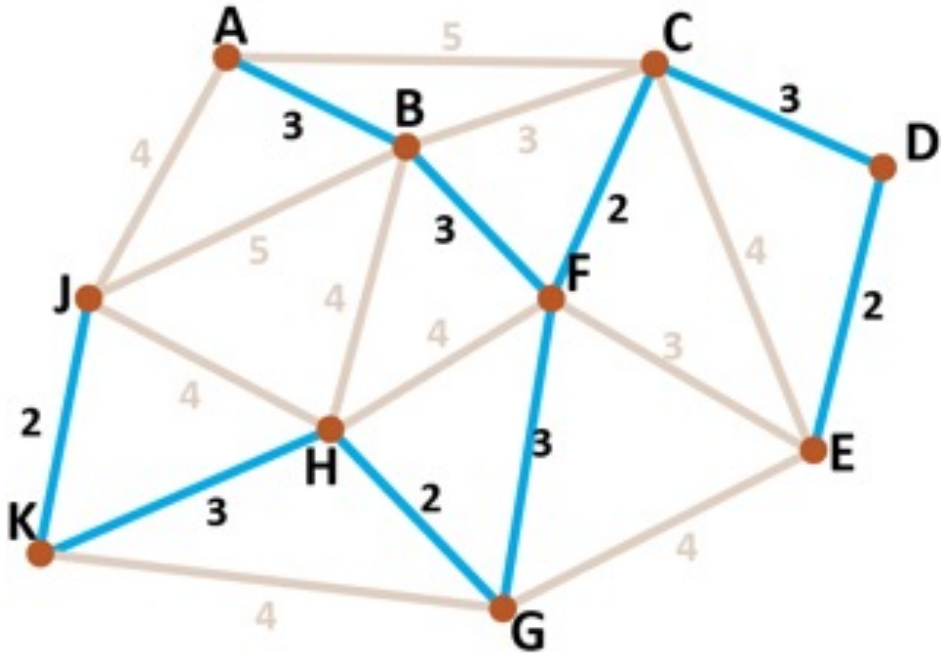
We willen juist dat leerlingen zelf (stukjes van) algoritmen bedenken en dan analyseren. Daarvoor is er een opbouw in activiteiten.

- Stellingen beoordelen bij een algoritme
- Presentatie laat de stappen van een algoritme zien, leerlingen formuleren het algoritme.
- Algoritmen verbeteren, leerlingen maken betere oplossing.
- Leerlingen bedenken zelf algoritme, voorbeelden geven hints

Problem solving

De lessen zijn gericht op *problem solving*. Een algoritme bedenk je om een bepaald probleem op te lossen.

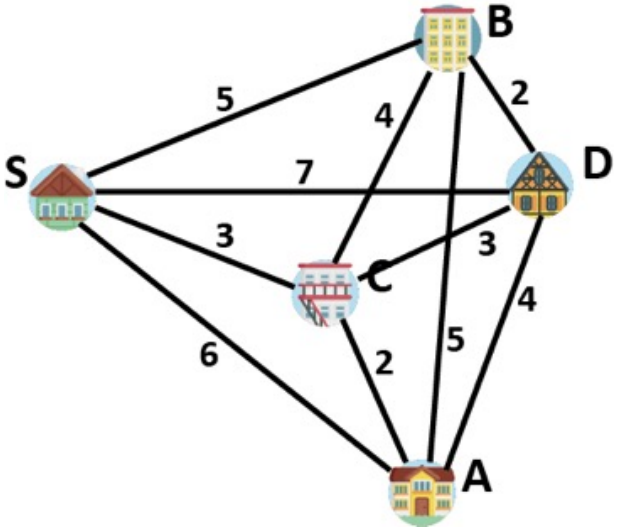
Daarbij komen de eindtermen vanzelf in beeld: een algoritme dat te lang rekent is geen oplossing van je probleem!



Coderen?

In het materiaal zijn geen opdrachten opgenomen waarin leerlingen een algoritme moeten uitprogrammeren.

Reden: te tijdrovend door alle overhead (90% van je programma is weergave, besturing, etc. Slechts een klein stukje is het algoritme)

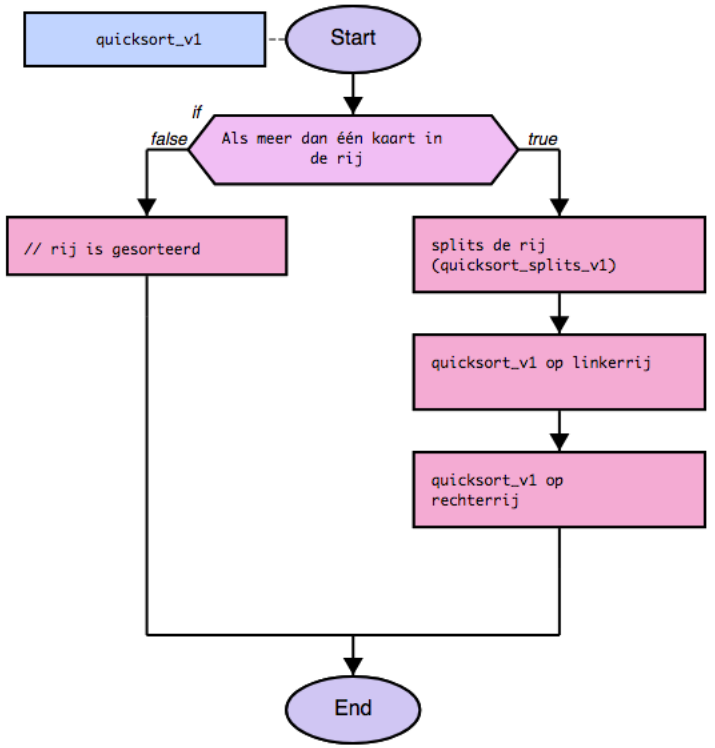


Coderen? (vervolg)

Wat we wel doen:

- natuurlijke taal,
- stroomdiagrammen

Dichtst bij programmeertaal staan de opdrachten waarin leerlingen een ‘precies’ stroomdiagram maken
(daarin precies een index aangeven: ‘lijst[0]’ ipv ‘voorste’)



Onderwerpen

1. **Inleiding**, met herhaling en intro-probleem (*string matching Horspool*)
2. **Zoeken**: *lineair* vs, *binair*, *zoekbomen*, (verdieping: *hash table*) Strategieën bij zoeken, *verdeel & heers*. Complexiteiten *log(n)* en *n*.
3. **Sorteren:** Verschillende algoritmen *vergelijken*, complexiteit n^2 en *n*lg(n)*. Niveaus in uitwerking.
4. **Grafen**: *minimale opspannende boom*: voorbeeld van *subset-probleem* (brute kracht is 2^n). *Gretige* (greedy) strategie voor goede oplossingen
5. **Handelsreizigersprobleem**: voorbeeld van *volgorde-probleem* (brute kracht is *n!*) Er is geen snelle exacte oplossing, strategieën bij *benaderen* van de beste oplossing.
6. **Algoritmen** die het **dagelijks leven** beïnvloeden: *gezichtsherkenning*, *ranking*, *SyRI*.

Complexiteit

1. **Inleiding**, met herhaling en intro-probleem (*string matching Horspool*)
2. **Zoeken**: *lineair* vs, *binair*, *zoekbomen*, (verdieping: *hash table*) Strategieën bij zoeken, *verdeel & heers*. Complexiteiten *log(n)* en *n*.
3. **Sorteren:** Verschillende algoritmen *vergelijken*, complexiteit n^2 en $n*lg(n)$. Niveaus in uitwerking.
4. **Grafen**: *minimale opspannende boom*: voorbeeld van *subset-probleem* (brute kracht is 2^n). *Gretige* (greedy) strategie voor goede oplossingen
5. **Handelsreizigersprobleem**: voorbeeld van *volgorde-probleem* (brute kracht is *n!*) Er is geen snelle exacte oplossing, strategieën bij *benaderen* van de beste oplossing.
6. **Algoritmen** die het **dagelijks leven** beïnvloeden: *gezichtsherkenning*, *ranking*, *SyRI*.

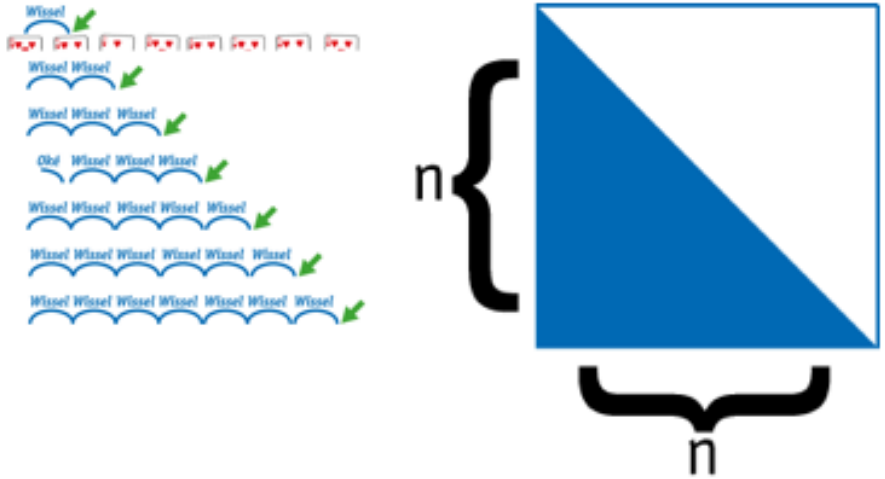
Probleemaanpak

1. **Inleiding**, met herhaling en intro-probleem (*string matching Horspool*)
2. **Zoeken**: *lineair* vs, *binair*, *zoekbomen*, (verdieping: *hash table*) Strategieën bij zoeken, *verdeel & heers*. Complexiteiten *log(n)* en *n*.
3. **Sorteren:** Verschillende algoritmen *vergelijken*, complexiteit n^2 en *n*lg(n)*. Niveaus in uitwerking.
4. **Grafen**: *minimale opspannende boom*: voorbeeld van *subset-probleem* (*brute kracht* is 2^n). *Gretige* (*greedy*) strategie voor goede oplossingen
5. **Handelsreizigersprobleem**: voorbeeld van *volgorde-probleem* (brute kracht is *n!*) Er is geen snelle exacte oplossing, strategieën bij *benaderen* van de beste oplossing.
6. **Algoritmen** die het **dagelijks leven** beïnvloeden: *gezichtsherkenning*, *ranking*, *SyRI*.

Tour

We doorlopen het materiaal in sneltreinvaart, zodat er ruimte is voor het daadwerkelijk doen van vier workshopopdrachten.

... noemen



Toetsing van domein B: Grondslagen

- Voorbeeld toetsvragen bekijken
- Varieren met vragen:
  - makkelijker/moeilijker (bv. havo/vwo, toetsvoorbereiding, leerachterstanden)
  - vorm (open, gesloten)
  - vaardigheden (onderzoeken, aanpassen, creeëren)
  - context met zelfde moeilijkheidsgraad (bv. voor herkansing)

Algoritmiek (domein B: kern, domein G: verdieping)

- Nieuw domein
- Inhoud pittig
- Lastig toetsen

Handreiking incl. voorbeeld toetsvragen

Het team bestaat uit (op alfabetische volgorde):

- Paul Bergevoet (vakdidacticus Universiteit Utrecht, auteur Informatica Actief)
- Martin Bruggink (vakdidacticus TU Delft)
- Adriaan Gijssen (docent vo, auteur Instruct)
- Jacco Gnodde (vakdidacticus VU, docent vo)
- Chris Hendriks (docent HBO)
- Johan Hoekstra (docent vo)
- Renske Weeda (onderzoeker Radboud Universiteit, docent vo)

https://www.slo.nl/zoeken/@19156/toetsing-domein-grondslagen/

Standaardalgoritmen: leerdoelen

Eindterm: Subdomein B1: Algoritmen

- De kandidaat kan een oplossingsrichting voor een probleem uitwerken tot een algoritme, daarbij **standaardalgoritmen herkennen en gebruiken**, en de correctheid en efficiëntie van digitale artefacten onderzoeken via de achterliggende algoritmen

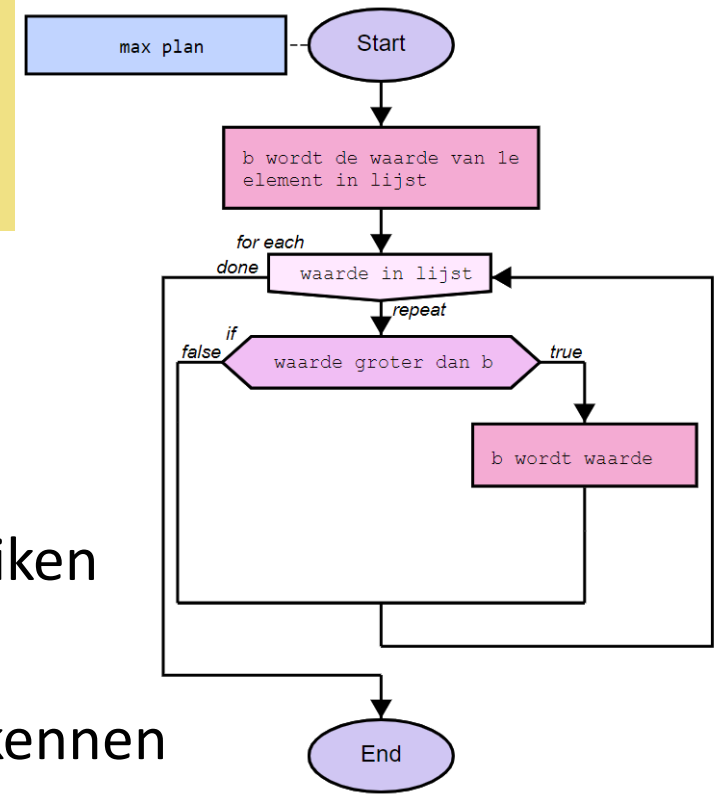
Leerdoel: Een aantal standaardalgoritmen herkennen en gebruiken.

- Welke standaardalgoritmen?

lineair zoeken, min/max, tellen, som, gemiddelde, filteren, sorteren...

Learning by Doing

Verschillende soorten vragen



Leerdoel: Een aantal standaardalgoritmen herkennen en gebruiken

Hoe kan een leerling aantonen dat ze standaardagoritmen herkennen en kunnen gebruiken?

OPDRACHT in tweetallen (2min):
Bedenk een aantal verschillende vragen

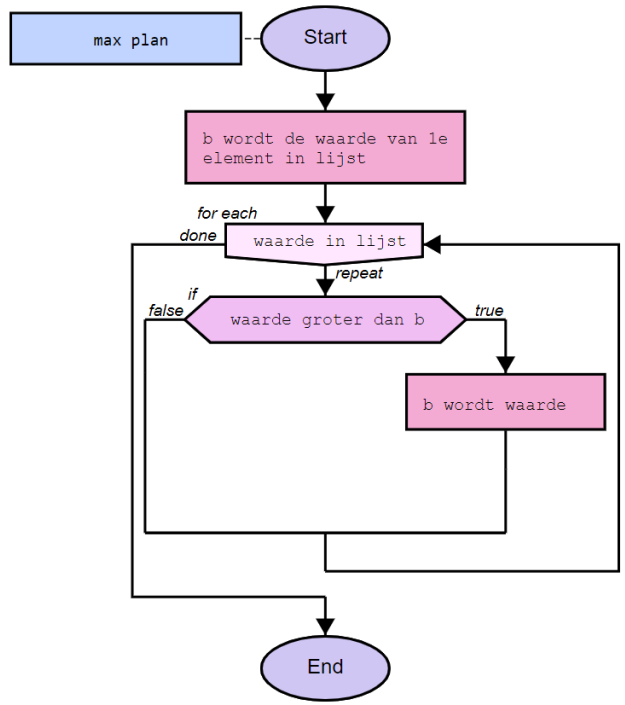
Merk op:

- maak het niet te moeilijk: houd het klein, geef evt iets voor
- ook zwakkere leerlingen moeten de kans krijgen te laten zien wat ze **wel** kunnen: subvragen

Moeilijkheid van vraagsoorten

inzicht

Traceren: Bepaal het resultaat gegeven een invoer
Parsons puzzle: gegeven alle onderdelen, deze op de juiste plek zetten (evt met afleiders)
Skelet: vul de ontbrekende onderdelen aan
Vind de fout
Corrigeer de fout
Omschrijf de rol van een variabele t.o.v. het algoritme als geheel
Gegeven een algoritme, vat het doel in eigen woorden samen
Geef aan bij welk invoer een algoritme wel/niet werkt
Breidt het algoritme uit zodat ... (telt hoe de max vaak voorkomt)
Pas aan zodat ... (b.v. voor het minimum werkt)
Verbeter de efficiëntie van het algoritme



Waarom vragen aanpassen

Zelfde moeilijkheid:
- herkansing
- tweede variant toets

Variëren in moeilijkheid:
- havo/vwo
- onderscheid sterk/zwakke leerlingen

Vragen aanpassen

Antwoordsoort:
- Meerkeuze: de leerling tussen verschillende algoritmes laten kiezen (makkelijker)
- Open vraag (moeilijker)

Inhoudelijk uitbreiden:
- Aangepaste *invoerwaarden*: lege lijst, negatieve waarden filteren/foutmelding (moeilijker)
- Aangepaste *invoertype*: invoerwaarden met kommagetallen eerst afronden (moeilijker)
- Aangepaste *uitvoertype*: waarde opleveren vs afdrukken (moeilijker)

Gebruik contexten:
- Voor min/max: Highscore, temperatuur, regenval, cijfer, lengte, datagebruik
- Gecombineerde context met min/max:
  - tel aantal voorkomens van max highscore/regenval/temperatuur/cijfer/datagebruik
  - filter min/max eruit (bv. laagste toetscijfer telt niet mee, of uitschieter datagebruik door wifi storing)

Vanuit één algoritme variëren in niveau

Voorbeeld algoritme: *bepaal of een woord een palindroom is*.

Voorbeelden van elk niveau op volgende dia's

moeilijkheid

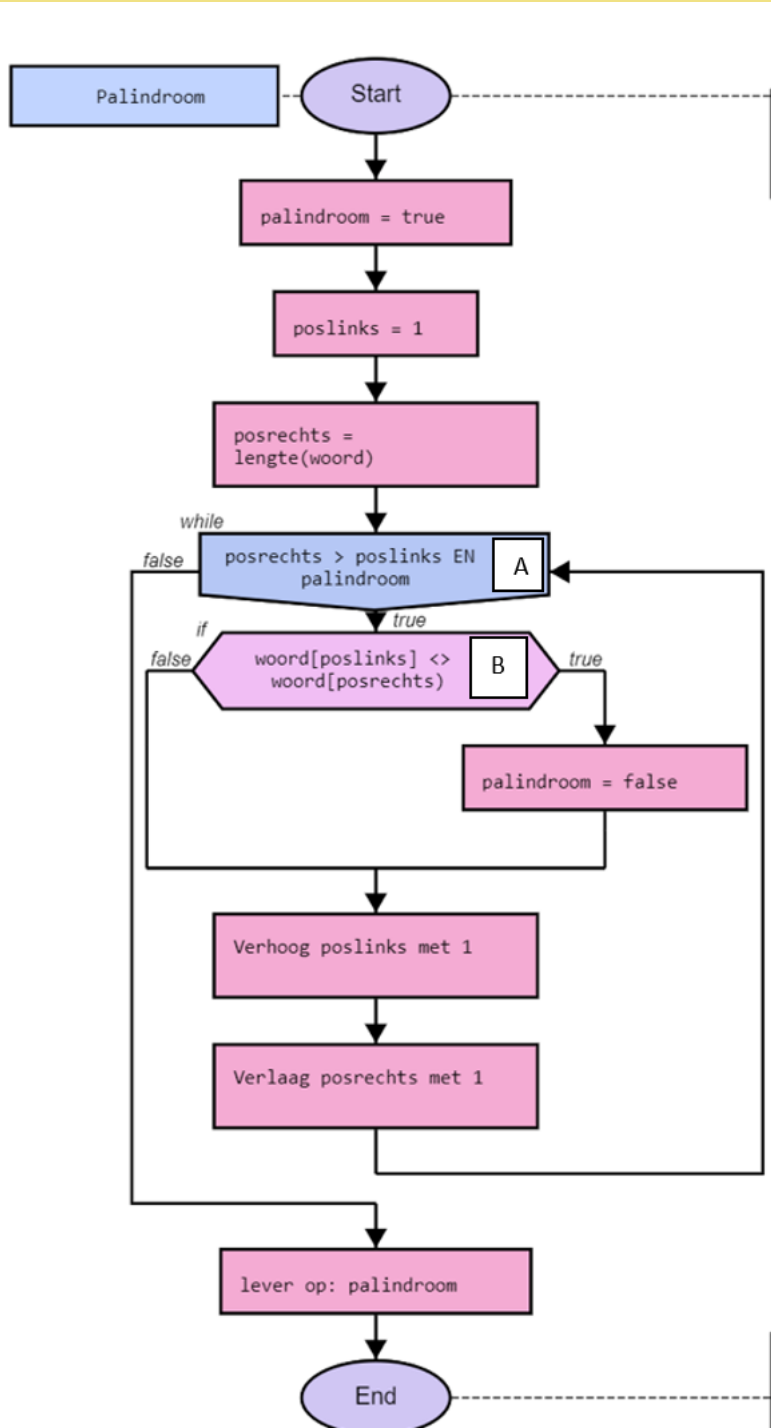
Lezen en uitvoeren van het algoritme
Debuggen van het algoritme
Algoritme opstellen door drag&drop
Stellingen over een algoritme
Zelf een algoritme opstellen op basis van gegeven oplossingsrichting
Zelf een algoritme opstellen op basis van probleem
Recursief algoritme opstellen

Vanuit één algoritme variëren in niveau: *palindroom*

Type 1: lezen en uitvoeren van het algoritme

traceren

	Waarde posrechts	Waarde poslinks	Waarde palindroom	Waarde A	Waarde B
Iteratie 1	9	1	true	true	false
Iteratie 2					
Iteratie 3					
…					



Vanuit één algoritme variëren in niveau: *palindroom*

Type 2: debuggen van het algoritme

analyseren

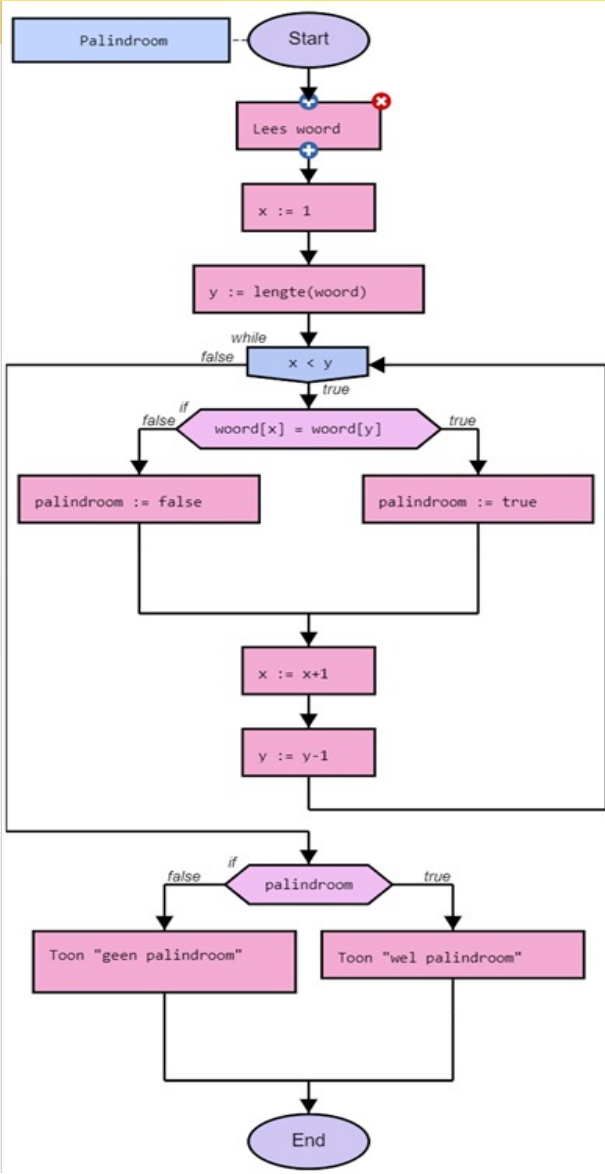
aanpassen

verbeteren (efficiëntie)

a) Helaas zit er nog een fout in dit stroomdiagram. Geef een voorbeeld van een woord dat geen palindroom is, maar volgens dit stroomdiagram wel een palindroom is. Het hoeft geen bestaand woord te zijn.

b) De fout kan worden hersteld door één blokje te verplaatsen. Geef aan welk blokje en waar het blokje naar verplaatst moet worden.

c) Nadat je het foutje er hebt uitgehaald werkt het programma op zich prima. Wanneer je er nog eens kritisch naar kijkt kom je erachter dat het programma in een aantal gevallen, waarbij het woord geen palindroom is, onnodig veel opdrachten uitvoert, bijvoorbeeld bij KANDELAAR. Hoe zou je het programma met een kleine aanpassing kunnen verbeteren zodat het minder opdrachten uitvoert?

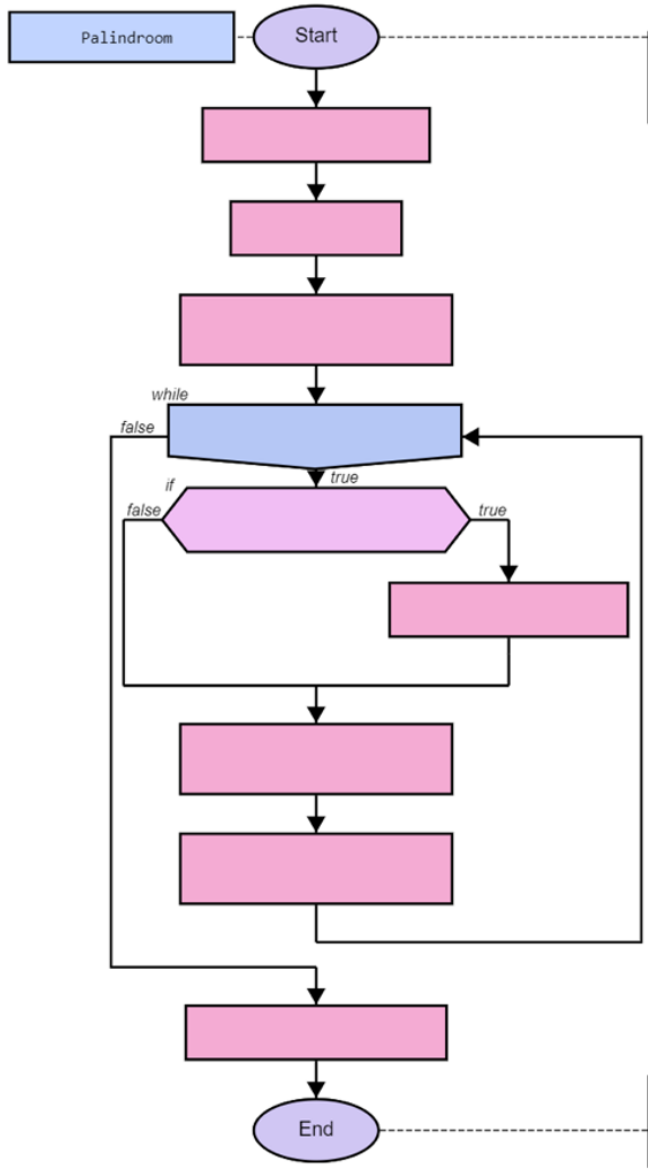


Vanuit één algoritme variëren in niveau: *palindroom*

Type 3: algoritme opstellen door drag&drop

evt afleiders toevoegen (moeilijker)

A	lever op: palindroom
B	palindroom = false
C	verhoog poslinks met 1
D	posrechts > poslinks EN palindroom
E	verlaag posrechts met 1
F	palindroom = true
G	poslinks = 1
H	posrechts = lengte(woord)
I	woord[poslinks] <> woord[posrechts]



Learning by Doing

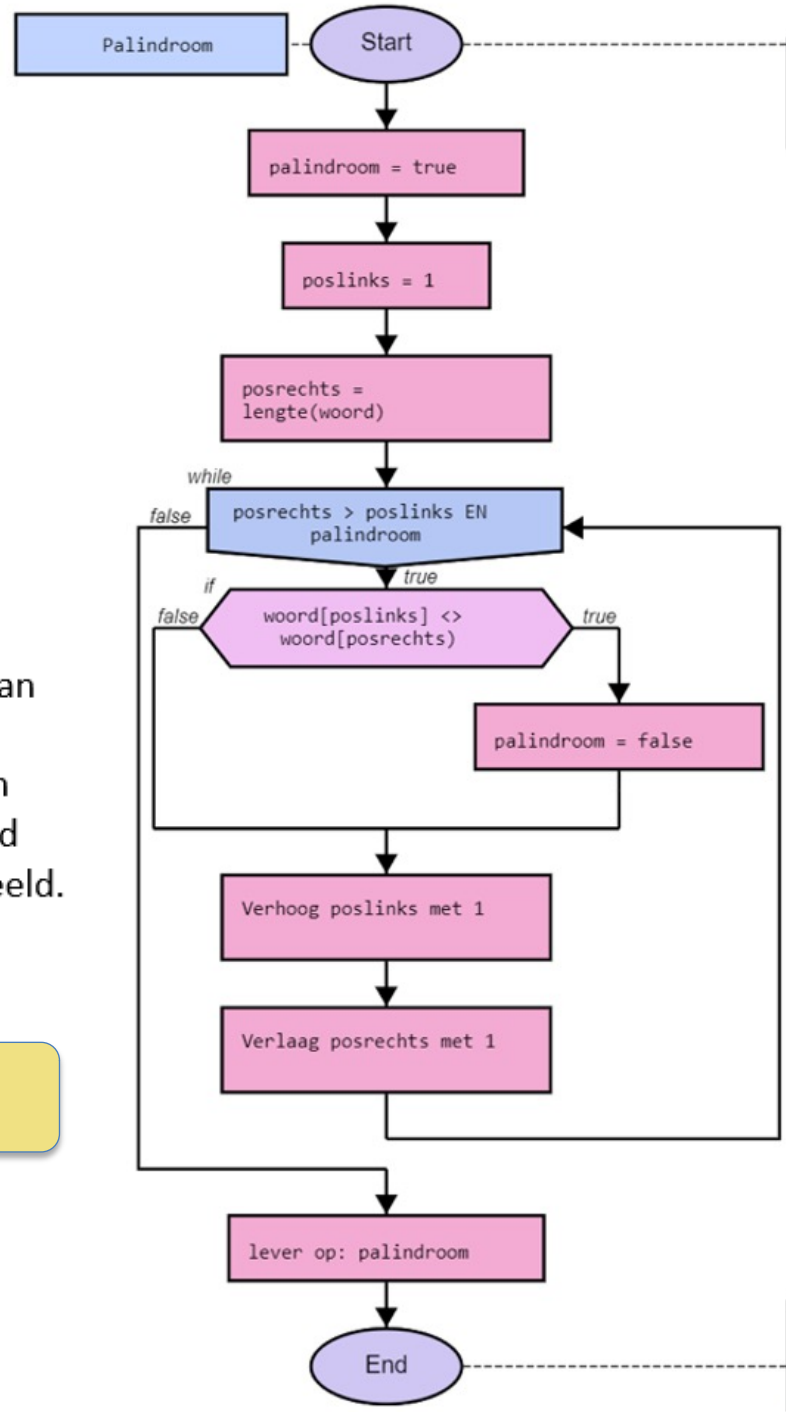
Vanuit één algoritme variëren in niveau: *palindroom*

Type 4: stellingen over een algoritme

a) Iemand beweert dat bij een woord van 1 letter, het resultaat altijd *true* is. Klopt dat? Zo ja, geef aan waarom. Zo nee, geef een tegenvoorbeeld.

b) In het algoritme zit een herhaling (een while-loop). Het doorlopen van zo'n herhalingslus heet een iteratie. Iemand beweert: het aantal iteraties is gelijk aan het aantal letters van het woord gedeeld door 2, afgerond naar beneden. Klopt dat? Zo ja, geef aan waarom. Zo nee, geef een tegenvoorbeeld.

evalueren



Vanuit één algoritme variëren in niveau: *palindroom*

Type 5: zelf een algoritme opstellen op basis van gegeven oplossingsrichting

De opdracht

Een palindroom is een woord dat van achter naar voren gelezen precies hetzelfde is als van voor naar achter. RAAR, LEPEL en PARTERRETRAP zijn palindromen, ROER niet.

Er is een algoritme dat test of een woord een palindroom is. Je geeft een woord als input, en het algoritme geeft 'JA!' terug als het een palindroom is en 'NEE' als het geen palindroom is. Hieronder zie je het algoritme toegepast op twee woorden: 'LEPEL' en 'LETSEL'.

Stel een stroomdiagram op van de werking van dit algoritme.

algoritme beschrijven

(1) LEPEL

======= Stap 1 ========

L E P E L

↑ ↑ ok

======= Stap 2 ========

L E P E L

↑ ↑ ok

======= Stap 3 ========

L E P E L

↗↖ ok Resultaat: JA

==========

(2) LETSEL

======= Stap 1 ========

L E T S E L

↑ ↑ ok

======= Stap 2 ========

L E T S E L

↑ ↑ ok

======= Stap 3 ========

L E T S E L

↑↑ niet ok Resultaat: NEE

Vanuit één algoritme variëren in niveau: *palindroom*

Type 6: zelf een algoritme opstellen op basis van probleem

De opdracht

Een palindroom is een woord dat van achter naar voren gelezen precies hetzelfde is als van voor naar achter. RAAR, LEPEL en PARTERRETRAP zijn palindromen, ROER niet. Stel een stroomdiagram op dat aangeeft of een woord een algoritme is. Het algoritme krijgt als invoer een woord bestaande uit 1 of meer letters. Het algoritme geeft 'JA!' terug als het een palindroom is en 'NEE' als het geen palindroom is.

algoritme bedenken

Vanuit één algoritme variëren in niveau: *palindroom*

Type 7: recursief algoritme opstellen (alleen vwo)

Stel een recursief algoritme op dat aangeeft of een woord een algoritme is. Het algoritme krijgt als invoer een woord bestaande uit 1 of meer letters. Het algoritme geeft *true* terug als het een palindroom is en *false* als het geen palindroom is. Je mag alle instructies in natuurlijke taal formuleren. Doe dit op basis van de volgende vragen.

a. Wat is de stap waarmee je begint bij de bepaling of een woord een palindroom is?
b. Wat is de recursieve aanroep? Geef aan hoe je probleem hier kleiner geworden is.

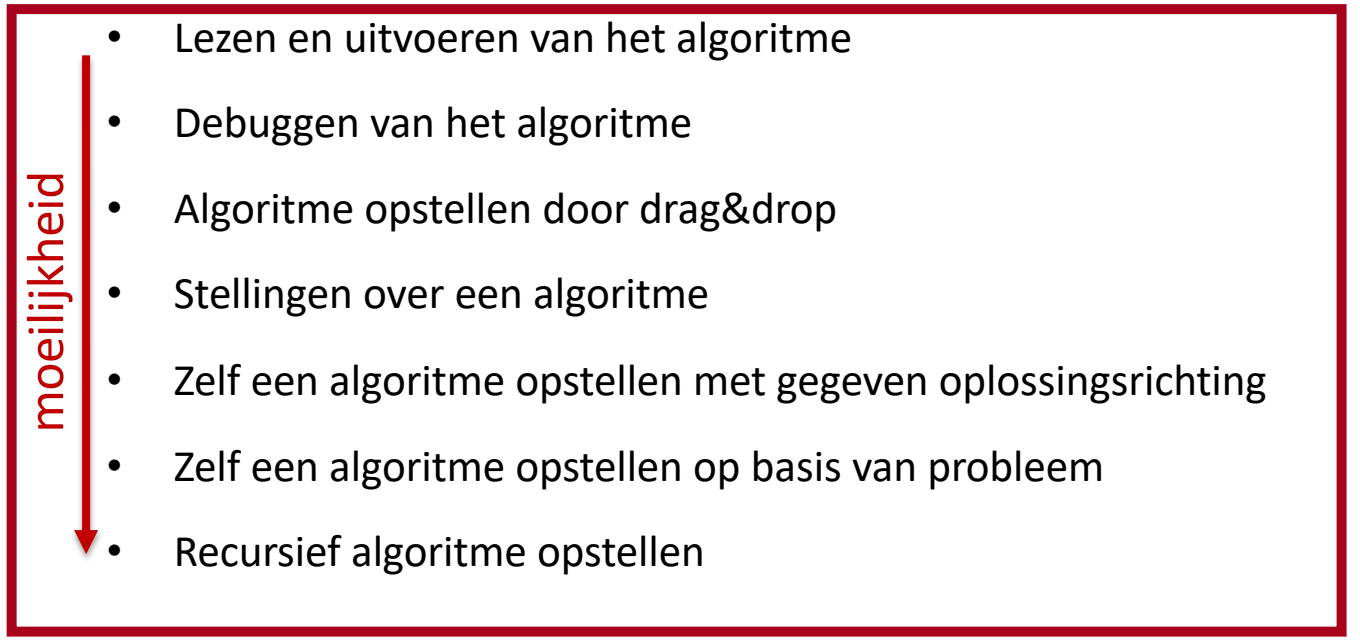
Een recursief algoritme kan oneindig doorgaan omdat het zichzelf steeds aanroept. Daarom moet je voorwaarden inbouwen om het algoritme te laten stoppen.

c. Welke voorwaarde kun je bij de stap aangeven om het algoritme te stoppen?
   Geef ook aan wat dan de uitkomst is (*true*/*false*)
d. Welke voorwaarde kun je bij de recursieve aanroep aangeven om het algoritme te stoppen?
   Geef ook aan wat dan de uitkomst is (*true*/*false*)
e. Maak nu het algoritme in pseudocode.

Zelfde 7 vragen met andere algoritmen

Voorbeelden van alternatieve algoritmen:

- Anagram
- Winstmaximalisatie met Bitcoins
- …



Learning by Doing

Vragen? Opmerkingen? Aanvullingen?

paulbergervoet@informatica-actief.nl

renskeweeda@informatica-actief.nl

https://www.slo.nl/zoeken/@19156/toetsing-domein-grondslagen/

Pro

xx