Universiteit Leiden
Opleiding Informatica

Mario

(of hoe je een programmeerwedstrijd kunt winnen)

Naam:	Richard Huybers
Datum:	26/08/2016
Begeleider:	Rudy van Vliet
Tweede lezer:	Hendrik Jan Hoogeboom

BACHELOR SCRIPTIE

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Mario

(of hoe je een programmeerwedstrijd kunt winnen)

Richard Huybers

Abstract

Mario is een opgave uit de *Benelux Algorithm Programming Contest 2015*. In dit probleem moet Super Mario een rivier zien te overbruggen met behulp van een aantal boten dat op de rivier vaart. Om de overkant te bereiken moet hij meestal meerdere overstappen maken tussen de verschillende boten. Het Mario probleem is in zekere zin een variatie op het kortste paden probleem, waarbij de afstanden tussen de knopen uit de graaf elke seconde veranderen. In dit onderzoek zullen we het Mario probleem en twee van zijn oplossingen bestuderen. We zijn hierbij vooral geïnteresseerd in de correctheid, verbeterpunten en tijdscomplexiteit van deze oplossingen. Het doel van dit onderzoek is het bepalen van de beste oplossing voor het Mario probleem.

Inhoudsopgave

Inleiding

Mario is een probleem uit de *Benelux Algorithm Programming Contest 2015* [web15], waarin Super Mario tracht een rivier over te steken. Hierbij kan hij gebruik maken van een aantal boten die op de rivier varen. Door heen en weer te springen tussen de verschillende boten kan Mario zich steeds verder richting de overzijde bewegen. De deelnemers van de programmeerwedstrijd werd gevraagd om voor dit probleem een programma te schrijven dat Mario zo snel mogelijk naar de overkant van de rivier kan navigeren.
In deze scriptie onderzoeken we twee algoritmes waarmee het Mario probleem kan worden opgelost. Deze algoritmes zijn in 2015 bedacht door juryleden van BAPC 2015. We zijn vooral geïnteresseerd in de correctheid, verbeterpunten en tijdscomplexiteit van de algoritmes. We willen met dit onderzoek bepalen welk van de twee algoritmes de beste oplossing is voor het Mario probleem.

Voordat we de twee bestudeerde algoritmes bespreken, zullen we in Hoofdstuk 1 het Mario probleem eerst in meer detail beschrijven. Ook worden in dit hoofdstuk een aantal definities geïntroduceerd. Deze zullen in de latere hoofdstukken gebruikt worden bij de analyse van het probleem en de algoritmes.
Hoofdstuk 2 behandelt een variant van *Dijkstra's kortste pad algoritme* waarmee het Mario probleem kan worden opgelost. In dit hoofdstuk wordt vooral aandacht besteed aan de verbeterpunten van het algoritme.
In Hoofdstuk 3 analyseren we een algoritme dat het Mario probleem oplost door de vaartrajecten van alle boten tegelijkertijd te *simuleren*. De focus van dit hoofdstuk ligt vooral op de worst case van dit algoritme.
De beste oplossing voor het Mario probleem wordt in Hoofdstuk 4 bepaald aan de hand van een aantal experimenten. Ook bekijken we de effecten van de gevonden verbeteringen op de rekentijd van het algoritme van Dijkstra.
Tot slot worden in Hoofdstuk 5 de conclusies van dit onderzoek besproken.

Hoofdstuk 1

Probleembeschrijving

Om het eindpunt van een zeer moeilijk level te bereiken moet Super Mario een wilde rivier zien te overbruggen. Deze rivier heeft een breedte $1 \leq W \leq 500$ en beslaat $x = [0, W]$. Transport over het water van de rivier is alleen mogelijk met boten. Het aantal boten dat op de rivier vaart is $0 \leq N \leq 100$. Elk van deze boten heeft een lengte en breedte 0, en vaart periodiek heen en weer tussen een linkergrens L en rechtergrens R met een constante snelheid van één afstandseenheid per seconde. De vaarrichting van de boten is hierbij loodrecht op de oevers van de rivier. Op tijdstip $t = 0$ bevinden alle boten zich op hun linkergrens en bevindt Mario zich op $x = 0$ (de linkeroever). Als Mario dezelfde x-coördinaat heeft als een boot, mag hij naar deze boot overstappen. Zodoende kan Mario zich naar verschillende boten verplaatsen en zich steeds verder richting de overkant van de rivier bewegen. De vraag van het probleem is nu om te bepalen wat de minimale tijd is waarin Mario $x = W$ (de rechteroever) kan bereiken, of om aan te tonen dat $x = W$ onbereikbaar is.

Voorbeeld 1.1. Mario wil een rivier met breedte $W = 7$ oversteken. Hij heeft hiervoor $N = 3$ boten tot zijn beschikking: $b_1 = [0, 4]$, $b_2 = [4, 7]$ en $b_3 = [2, 4]$. In Tabel 1.1 staan de x-coördinaten van deze boten op tijdstippen $t = 0, 1, \ldots, 9$. Tabel 1.1 wordt ook wel de *tijdstabel* van de boten genoemd. De snelste route van $x = 0$ naar $x = 7$ is rood gekleurd in de tijdstabel. Op $t = 0$ bevindt Mario zich op b_1, omdat dit de enige boot is met een linkergrens op $x = 0$. Na drie seconden hebben b_1 en b_3 dezelfde x-coördinaat. Mario kan daarom op $t = 3$ overstappen naar b_3. Op $t = 6$ is een overstap van b_3 naar b_2 mogelijk. Boot b_2 heeft vanaf $t = 6$ nog drie seconden nodig om $x = 7$ te bereiken. De tijd die Mario nodig heeft om van $x = 0$ naar $x = 7$ te varen, bedraagt daarom negen seconden. △

(Het symbool △ wordt gebruikt om het einde van een Voorbeeld aan te geven.)

Om het Mario probleem goed te kunnen analyseren, zullen een aantal aspecten uit het probleem formeel gedefinieerd moeten worden. We introduceren daarom nu een aantal definities. Deze zullen in de komende

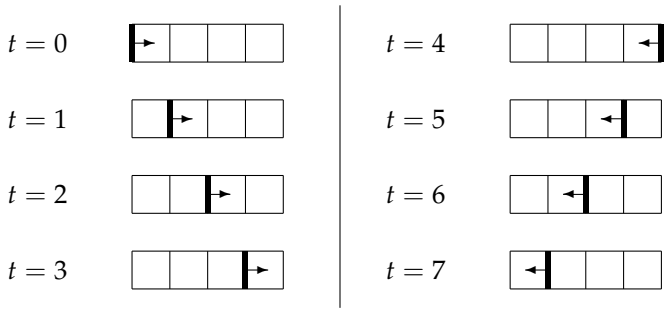
Tabel 1.1: De tijdstabel van $b_1 = [0,4]$, $b_2 = [4,7]$ en $b_3 = [2,4]$.

t	0	1	2	3	4	5	6	7	8	9
b_1	0	1	2	3	4	3	2	1	0	1
b_2	4	5	6	7	6	5	4	5	6	7
b_3	2	3	4	3	2	3	4	3	2	4

hoofdstukken gebruikt worden in de stellingen, lemma's en andere definities over het probleem.

Definitie 1.2. Een *boot* is een tweetal $[L, R]$, met $L, R \in \mathbb{N}$ en $0 \leq L < R \leq W$. Op het tijdstip $t = 0$ bevindt elke boot zich op positie $x = L$, waarna het periodiek heen en weer vaart tussen $x = L$ en $x = R$ met een constante snelheid van één afstandseenheid per seconde. De boot legt hierbij een continue reis af.

De *periode* van een boot is de tijd die de boot nodig heeft om zijn gehele bereik rond te varen. De periode van boot $b = [0,4]$ is bijvoorbeeld $2 \cdot (R - L) = 2 \cdot (4 - 0) = 8$ seconden, zoals te zien in Figuur 1.1. Het bereik van b wordt afgebeeld door een rechthoek, waarin de x-coördinaat van de boot wordt aangeduid door een dikke zwarte lijn. De pijl naast deze lijn geeft de vaarrichting van de boot aan.



Figuur 1.1: De periode van $b = [0,4]$.

Het belangrijkste gegeven van een boot is zijn x-coördinaat. Uit de x-coördinaten van twee boten kan namelijk worden opgemaakt of Mario tussen de boten kan overstappen. De x-coördinaat van een boot wordt ook wel zijn *positie* genoemd. De positie van een boot is als volgt gedefinieerd:

Definitie 1.3. Zij b een boot en t een tijdstip. De *positie* van b op t is de x-coördinaat van boot b op tijdstip t en wordt gegeven door de functie $pos(b, t)$.

De functiewaarde $pos(b, t)$ kan in constante tijd berekend worden met behulp van modulo rekenen. Deze berekening is weergegeven in Algoritme 1. Het quotiënt van de geheeltallige deling $\frac{t}{R-L}$, waarbij L en R de grenzen van b zijn, geeft aan hoe vaak b zijn halve periode $R - L$ heeft afgelegd op tijdstip t. Als dit quotiënt even is, dan heeft b na het begintijdstip $t = 0$ een even aantal keer zijn halve periode afgelegd. Omdat b op $x = L$ begint, betekent dit dat de boot op t richting zijn rechtergrens R vaart. De rest van de deling $\frac{t}{R-L}$ geeft aan hoelang b al in deze richting vaart, nadat de boot voor het laatst zijn halve periode heeft afgelegd. Omdat

b met een snelheid van één afstandseenheid per seconde vaart, geldt nu dat $pos(b,t) = L + t \mod (R-L)$. Analoog kunnen we beredeneren dat b zich op positie $x = R - t \mod (R-L)$ bevindt, als het quotiënt van $\frac{t}{R-L}$ oneven is.

Een preconditie van *pos* is $t \geq 0$, omdat het begintijdstip van het Mario probleem $t = 0$ is. Voor een negatief tijdstip is de positie van een boot niet gedefinieerd.

Algoritme 1 pos(b,t)

preconditie: $t \geq 0$
  $quotient := \lfloor t/(R-L) \rfloor$
  if $quotient \mod 2 = 0$ **then**
    return $L + t \mod (R-L)$
  else
    return $R - t \mod (R-L)$

Als twee boten op een tijdstip t dezelfde positie hebben, dan kan Mario op t een overstap maken tussen de boten. De boten liggen dan immers op dezelfde x-coördinaat. Met behulp van Definitie 1.3 kunnen we daarom ook een overstap formeel definiëren.

Definitie 1.4. Zij b_1 en b_2 twee boten en t een tijdstip. Een *overstap* is een drietal (b_1, t, b_2), waarbij $pos(b_1,t) = pos(b_2,t)$. Mario kan van b_1 naar b_2 *overstappen* op t als hij zich op tijdstip t op boot b_1 bevindt.

De vaarroute die Mario aflegt in zijn poging om de overkant van de rivier te bereiken wordt volledig gedefinieerd door de reeks overstappen die hij maakt. Deze reeks laat immers precies zien wanneer Mario welke boot bereikt en hoelang Mario op elke boot meevaart. In het algemeen ziet zo'n reeks er als volgt uit: $(b_{i_0}, t_1, b_{i_1}), (b_{i_1}, t_2, b_{i_2}), \ldots, (b_{i_{k-1}}, t_k, b_{i_k})$, waarbij $t_1 < t_2 < \ldots < t_k$. De reeks overstappen van Voorbeeld 1.1 is bijvoorbeeld $(b_1, 3, b_3), (b_3, 6, b_2)$. Merk op dat als Mario een boot $b = [0, W]$ tot zijn beschikking heeft, hij de overkant van de rivier kan bereiken zonder over te stappen. De reeks overstappen is in dit geval leeg. We zullen dit soort triviale instanties buiten beschouwing laten in ons onderzoek.

Het bepalen van de mogelijke overstappen die Mario kan maken is een belangrijk onderdeel bij de analyse van het Mario probleem. Een overstapmogelijkheid tussen twee boten b_1 en b_2 is met name interessant als deze de eerste overstapmogelijkheid is sinds een tijdstip t waarop $pos(b_1,t) \neq pos(b_2,t)$. We geven dit soort overstappen daarom een speciale naam.

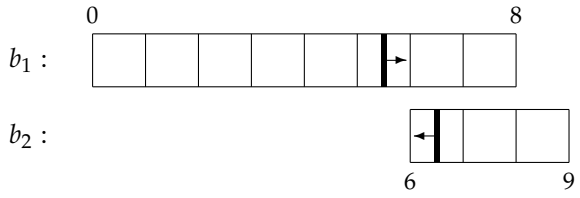
Definitie 1.5. Zij b_1 en b_2 twee boten en t een tijdstip. Een *kruising* is een overstap (b_1, t, b_2), waarbij $pos(b_1, t - \frac{1}{2}) \neq pos(b_2, t - \frac{1}{2})$. We zeggen dat b_1 en b_2 elkaar op t *kruisen*.

Definitie 1.5 gebruikt een tijdstap van een halve seconde, omdat dit de kleinst mogelijke nuttige tijdstap is. Op een tijdstip $t \in \mathbb{N}$ geldt namelijk dat elke boot zich op een positie $x \in \mathbb{N}$ moet bevinden. De grenzen van een boot zijn immers gehele getallen en elke boot vaart met een snelheid van één afstandseenheid per seconde. Twee boten kunnen elkaar daarom alleen kruisen op een gehele positie of precies tussen twee ge-

Figuur 1.2: De boten links kruisen elkaar op x_1, met $x_1 \in \mathbb{N}$. De boten rechts kruisen elkaar op $\frac{1}{2} \cdot x_2$, met $x_2 \in \mathbb{N}$.

hele posities in. Dit wordt geïllustreerd in Figuur 1.2. Bij tijdstappen van een halve seconde worden alle mogelijke kruisingen waargenomen en dus is dit de kleinst mogelijke nuttige tijdstap. In het vervolg zullen we er daarom alleen tijdstippen t bekijken die een veelvoud van $\frac{1}{2}$ zijn.
We eisen bij een kruising dat $pos(b_1, t - \frac{1}{2}) \neq pos(b_2, t - \frac{1}{2})$, omdat er ook overstappen bestaan waarbij dit niet het geval is (zie Voorbeeld 1.6. Tussen $t - \frac{1}{2}$ en t kan zich onmogelijk een kruising voordoen, waardoor deze eis garandeert dat een kruising de eerste overstapmogelijkheid tussen twee boten is sinds een tijdstip waarop de boten zich niet op dezelfde positie bevinden.

Voorbeeld 1.6. Zij $b_1 = [0, 8]$ en $b_2 = [6, 9]$ twee boten. In Figuur 1.3 zijn de posities van deze boten weergegeven op tijdstip $t = 5\frac{1}{2}$. Op $t = 5\frac{1}{2}$ geldt dat $pos(b_1, 5\frac{1}{2}) \neq pos(b_2, 5\frac{1}{2})$ en op $t = 6$ zal gelden dat $pos(b_1, 6) = pos(b_2, 6)$. De boten kruisen elkaar dus op $t = 6$.
Op $t = 6$ bereikt b_2 zijn linkergrens en keert de boot om. Daarom geldt ook op tijdstippen $t = 6\frac{1}{2}$, 7, $7\frac{1}{2}$, 8 dat $pos(b_1, t) = pos(b_2, t)$. Op deze tijdstippen kruisen b_1 en b_2 elkaar echter niet. Voor elk van deze tijdstippen is er immers ook een overstap tussen b_1 en b_2 mogelijk op $t - \frac{1}{2}$. △



Figuur 1.3: De posities van $b_1 = [0, 8]$ en $b_2 = [6, 9]$ op tijdstip $t = 5\frac{1}{2}$.

Hoofdstuk 2

Het Algoritme van Dijkstra

Het Mario probleem is in zekere zin een variatie op het kortste pad probleem. Het bepalen van de minimale tijd waarin de overkant van de rivier kan worden bereikt, komt namelijk neer op het vinden van het kortste pad tussen $x = 0$ en $x = W$. Het enige verschil met een standaard kortste pad probleem is dat de punten tussen het begin- en eindpunt van het pad continu van positie veranderen, wat het Mario probleem ietwat complexer maakt. Toch kan het wel degelijk worden opgelost met het kortste pad algoritme van Dijkstra [Dij59].
In de eerste paragraaf van dit hoofdstuk beschrijven we hoe het algoritme van Dijkstra kan worden toegepast om het Mario probleem op te lossen. De correctheid van deze implementatie zal worden bewezen in paragraaf 2.2. Tenslotte gaan paragrafen 2.3 en 2.4 in op de gevonden verbeteringen van dit algoritme.

2.1 Werking algoritme

Om het algoritme van Dijkstra uit te kunnen voeren op een instantie van het Mario probleem, zullen we deze eerst moeten omzetten naar een gewogen graaf. Bij een instantie met N boten bestaat deze graaf uit $N + 1$ knopen. De beginknoop van de graaf representeert de startpositie $x = 0$. De overige N knopen representeren de boten uit de instantie. Er bestaat een pijl tussen twee knopen als het mogelijk is om van de boot die gerepresenteerd wordt door de beginknoop van de pijl over te stappen naar de boot die gerepresenteerd wordt door de eindknoop van de pijl. Het gewicht van de pijl geeft aan hoeveel seconden nodig zijn voor deze overstap. Het is a priori niet duidelijk tussen welke boten een overstap mogelijk is. In paragraaf 2.3 komen we hierop terug.
Voordat het algoritme van Dijkstra wordt toegepast, is de beginknoop de enige knoop in de graaf met uitgaande pijlen. Mario kan zich immers vanaf zijn startpositie naar elke boot met een linkergrens $L = 0$

verplaatsen. Deze pijlen hebben een gewicht 0. Voor de overige knopen kunnen de uitgaande pijlen bepaald worden nadat ze door het algoritme zijn bezocht. Dan is namelijk pas bekend op welk tijdstip de door de knoop gerepresenteerde boot voor het eerst bereikt kan worden. Dit tijdstip is nodig om de gewichten van de uitgaande pijlen te berekenen. Als het algoritme alle knopen in de graaf (= boten uit de instantie) heeft bezocht, kan het kortste pad naar $x = W$ met een eenvoudige for-loop bepaald worden.

Het algoritme dat we nu zullen bespreken is geschreven door een jurylid van BAPC 2015. De invoer van het algoritme bestaat uit de rivierbreedte W, het aantal boten N en twee arrays L en R met de linker- en rechtergrenzen van de boten. Het algoritme neemt hierbij aan dat $0 \leq L[i] < R[i] \leq W$ geldt voor elke boot b_i. De uitvoer van het algoritme is het eerste tijdstip waarop Mario positie $x = W$ kan bereiken. Als Mario onmogelijk naar de overkant van de rivier kan varen, geeft het algoritme -1 terug.
Bij het doorlopen van de graaf houdt het algoritme twee arrays bij: *visited*, een array dat voor elke knoop aangeeft of het kortste pad naar deze knoop al bepaald is, en T, een array dat voor elke knoop de lengte bevat van het tot dan toe berekende kortste pad naar de knoop. Voordat het algoritme de graaf doorloopt, geldt dat $visited[i] = false$ voor alle boten b_i. Als b_i een linkergrens $L[i] = 0$ heeft, dan geldt $T[i] = 0$. Voor alle andere boten wordt de waarde van T geïnitialiseerd op 10^9. Dit is een willekeurig gekozen groot getal. In Hoofdstuk 3 zal een betere bovengrens worden bepaald.

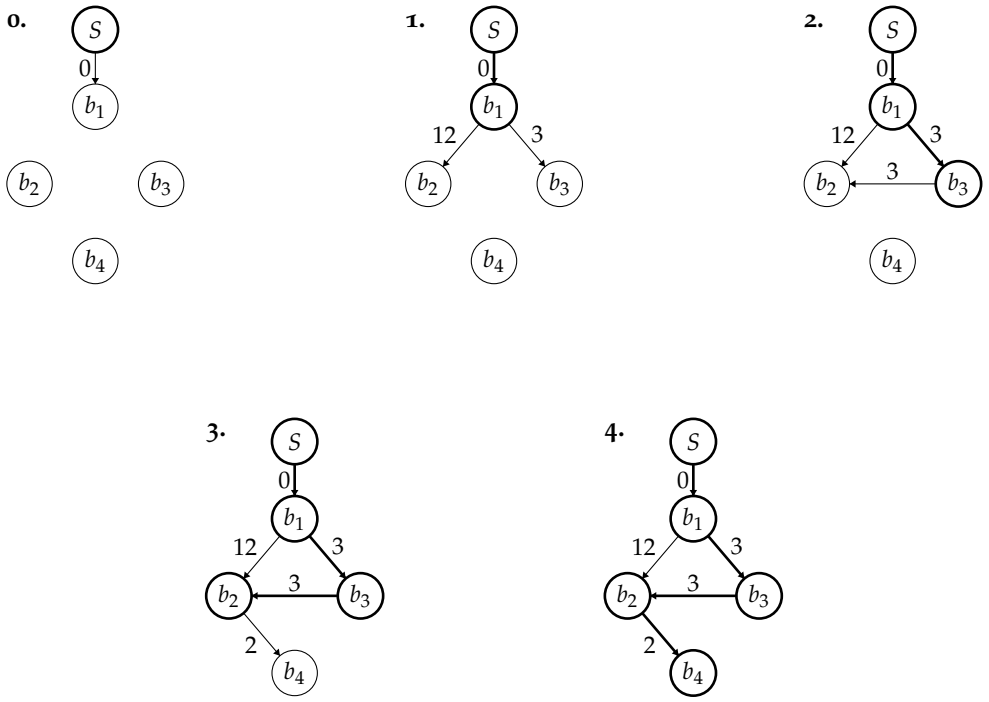
Algoritme 2 dijkstra(W,N,L,R)

```
// Stap 1: doorloop de graaf
for iter = 1 to N do
    i := −1
    for k = 1 to N do
        if visited[k] = false and (i = −1 or T[k] < T[i]) then
            i := k
    if T[i] = 10^9 then
        break
    visited[i] := true
    for j = 1 to N do
        if pos(b_j, T[i]) ≤ pos(b_i, T[i]) then
            continue
        t := QuickSim(b_i, b_j, T[i])
        if t ≠ −1 then
            T[j] := min(T[j], t)
// Stap 2: bepaal de lengte van het kortste pad naar x = W
ans := 10^9
for k = 1 to N do
    if R[k] = W then
        ans := min(ans, T[k] + pos(b_k, T[k]) − L[k] + R[k] − L[k])
if ans = 10^9 then
    return −1
else
    return ans
```

In stap 1 van Algoritme 2 is de for-loop te zien waarmee het algoritme een graaf doorloopt die overeenkomt met een instantie van het Mario probleem. Bij elke iteratie van deze loop wordt eerst een nog niet bezochte

knoop i met de kleinste T-waarde geselecteerd. Als $T[i] = 10^9$, dan zijn alle bereikbare knopen bezocht en stapt het algoritme uit de for-loop. Als $T[i] \neq 10^9$, dan wordt $visited[i]$ op *true* gezet. In de hierop volgende for-loop worden de uitgaande pijlen van knoop i bepaald. Hierbij worden alle boten b_j bekeken die op tijdstip $T[i]$ rechts van boot b_i liggen. De reden hiervoor zal in de volgende paragraaf worden uitgelegd. Het tijdstip van de eerst volgende kruising tussen b_i en b_j sinds $T[i]$ wordt berekend door de functie QuickSim. QuickSim geeft -1 terug als de boten elkaar nooit kruisen. De werking van QuickSim wordt besproken in paragraaf 2.4. Als het tijdstip van de gevonden kruising eerder is dan de tot nu toe berekende waarde van $T[j]$, dan wordt de waarde van $T[j]$ aangepast.

Nadat het algoritme alle bereikbare knopen uit de graaf heeft bezocht, gaat het verder met stap 2. Hierin wordt de lengte van het kortste pad berekend en opgeslagen in de variabele *ans*. Met een for-loop worden hiervoor alle boten b_k met $R[k] = W$ langsgegaan. Als deze boten voor het eerst bereikt worden, zijn ze onderweg naar of liggen ze op hun linkergrens $L[k]$. Ook dit zal worden verklaard in de volgende paragraaf. Het duurt vanaf $T[k]$ daarom nog $pos(b_k, T[k]) - L[k] + R[k] - L[k]$ seconden voordat $x = W$ bereikt wordt. Er geldt dus dat $ans = \min\{T[k] + pos(b_k, T[k]) - L[k] + R[k] - L[k] \mid R[k] = W \wedge 1 \leq k \leq N\}$.



Figuur 2.1: De graaf bestaande uit $b_1 = [0, 4]$, $b_2 = [4, 7]$, $b_3 = [2, 4]$ en $b_4 = [6, 10]$ wordt in vier stappen doorlopen.

Voorbeeld 2.1. Mario heeft in zijn poging om de overkant van een rivier met breedte $W = 10$ te bereiken $N = 4$ boten tot zijn beschikking: $b_1 = [0, 4]$, $b_2 = [4, 7]$, $b_3 = [2, 4]$ en $b_4 = [6, 10]$. Om de lengte van het kortste pad van $x = 0$ naar $x = W$ te berekenen, doorloopt het algoritme van Dijkstra een graaf met vijf knopen (zoals afgebeeld in Figuur 2.1). Als eerste stap bezoekt het algoritme de knoop van b_1. Dit is immers

de enige boot met een linkergrens op $x = 0$. Met QuickSim wordt vervolgens berekend hoe lang Mario op b_1 moet varen, voordat hij kan overstappen naar de andere boten. Voor b_2 is dit 12 seconden en voor b_3 is dit 3 seconden. Boot b_4 blijkt onbereikbaar vanaf b_1. Het algoritme bezoekt in stap 2 de knoop van b_3 en berekent zijn uitgaande pijlen. Boot b_2 kan na 3 seconden bereikt worden. Boot b_4 is ook vanaf b_3 onbereikbaar. De snelste manier om b_2 te bereiken is via een overstap vanaf b_3. In stap 3 wordt de knoop van b_2 bezocht en worden ook zijn uitgaande pijlen bepaald. Boot b_4 kan vanaf b_2 in 2 seconden worden bereikt. Het eerste tijdstip waarop Mario zich op b_4 kan bevinden is daarmee $t = 3 + 3 + 2 = 8$. Op $t = 8$ bevindt b_4 zich op zijn linkergrens $L_4 = 6$, waardoor Mario op $t = 8 + R_4 - L_4 = 8 + 10 - 6 = 12$ voor het eerst $x = W$ kan bereiken. △

2.2 Correctheid algoritme

Het algoritme van Dijkstra maakt bij het berekenen van het kortste pad van $x = 0$ naar $x = W$ twee aannames. Eén van deze aannames gaat over de richting van een boot bij een overstap. We hebben echter nog niet duidelijk gedefinieerd wanneer een boot precies naar links of naar rechts vaart.

Definitie 2.2. Zij b een boot en t een tijdstip, met t een veelvoud van $\frac{1}{2}$. We zeggen dat b op t naar links vaart als $pos(b,t) > pos(b,t+\frac{1}{2})$ en dat b op t naar rechts vaart als $pos(b,t) < pos(b,t+\frac{1}{2})$. Als b op t met een andere boot kruist, dan zeggen we dat b naar links vaart bij de kruising als $pos(b,t) < pos(b,t-\frac{1}{2})$, en dat b naar rechts vaart bij de kruising als $pos(b,t) > pos(b,t-\frac{1}{2})$.

Het lijkt misschien verwarrend dat de richting van een boot op een tijdstip gedefinieerd wordt met behulp van zijn eerstvolgende positie, terwijl deze bij een kruising gedefinieerd wordt middels zijn vorige positie. Een boot die zich op zijn linkergrens bevindt vaart dus naar rechts. Een boot die op zijn linkergrens kruist met een andere boot, vaart bij deze kruising naar links. Er is een goede reden voor dit verschil.
We kunnen de richting van een boot als volgt beschrijven: een boot vaart naar links/rechts als hij onderweg is naar zijn linker-/rechtergrens. Als we deze beschrijving hanteren bij kruisingen, kan dit echter onduidelijkheid veroorzaken. Bij een kruising tussen twee boten moet altijd één boot naar links en één boot naar rechts varen, anders zal de afstand tussen de boten immers nooit nul worden. Stel nu dat twee boten b_1 en b_2 elkaar kruisen op een grens van b_1. Dan keert b_1 zich om op het tijdstip van de kruising en varen beide boten in dezelfde richting (ervan uitgaande dat b_2 niet ook toevallig omkeert bij de kruising). De boten zijn nu niet meer te onderscheiden op basis van hun richting. We definiëren de richting van een boot bij een kruising daarom als de richting van de boot vlak voor de kruising. De kruising ontstond immers doordat de boot in deze richting voer. Ook bij een overstap definiëren we de richting van een boot als zijn richting een halve seconde voor de overstap.

De aannames die het algoritme van Dijkstra maakt zijn:

1. Er bestaat altijd een kortste pad van $x = 0$ naar $x = W$ waarbij alleen wordt overgestapt naar boten die bij de overstap naar links varen.

2. Voor elke boot b_j met $pos(b_j, T[i]) \leq pos(b_i, T[i])$ geldt dat $T[j] \leq T[i]$.

Het algoritme neemt de eerste aanname aan, omdat het de function QuickSim gebruikt bij het berekenen van het tijdstip van de eerstvolgende kruising tussen de boten b_i en b_j. QuickSim gaat er bij deze berekening van uit dat b_j naar links vaart bij deze kruising.
Om de correctheid van het algoritme van Dijkstra te bewijzen, zullen we in deze paragraaf aantonen dat beide aannames gegrond zijn.

2.2.1 Bewijs aanname 1

De richtingen van twee boten bij de eerstvolgende kruising tussen de boten wordt bepaald door de onderlinge positie van de boten, zoals het volgende lemma laat zien:

Lemma 2.3. *Zij b_1 en b_2 twee boten en t_1 een tijdstip, zodat $pos(b_1, t_1) < pos(b_2, t_1)$. Zij verder t_2 het eerste tijdstip na t_1 waarop een overstap tussen b_1 en b_2 mogelijk is. Bij de overstap op t_2 zal b_1 naar rechts en b_2 naar links varen.*

Bewijs. Omdat $pos(b_1, t_1) < pos(b_2, t_1)$ zal t_2 ook het eerste tijdstip na t_1 zijn waarop de boten elkaar kruisen. Aangezien beide boten een continue reis afleggen, zal b_1 zich tussen t_1 en t_2 altijd links van b_2 bevinden. Bij een kruising geldt volgens Definitie 1.5 dat $pos(b_1, t_2 - \frac{1}{2}) \neq pos(b_2, t_2 - \frac{1}{2})$ en $pos(b_1, t_2) = pos(b_2, t_2)$. Omdat b_1 op $t_2 - \frac{1}{2}$ links van b_2 ligt, moet wel gelden dat $pos(b_1, t_2 - \frac{1}{2}) = pos(b_1, t_2) - \frac{1}{2}$ en dat $pos(b_2, t_2 - \frac{1}{2}) = pos(b_2, t_2) + \frac{1}{2}$. Hieruit volgt dat b_1 naar rechts en b_2 naar links vaart bij de kruising op t_2. □

Het algoritme van Dijkstra berekent het kortste pad van $x = 0$ naar $x = W$ door eerst voor elke boot b_i het kortste pad te berekenen van $x = 0$ naar b_i. Het algoritme gaat er dus vanuit dat, als $x = W$ bereikbaar is, er altijd een kortste pad van $x = 0$ naar $x = W$ bestaat waarop elke boot op het eerst mogelijke tijdstip wordt bereikt. Het volgende lemma laat zien dat dit inderdaad klopt.

Lemma 2.4. *Zij $T[i]$ het eerste tijdstip waarop een overstap naar boot b_i mogelijk is. Als $x = W$ bereikbaar is, dan bestaat er een kortste pad van $x = 0$ naar $x = W$ van de vorm:*

$$(b_0, T[1], b_1), (b_1, T[2], b_2), \ldots, (b_{n-1}, T[n], b_n)$$

Bewijs. Merk op dat $n > 0$, omdat we geen instanties beschouwen die een boot $[0, W]$ bevatten. Als $x = W$ bereikbaar is, dan bestaat er ook een kortste pad van $x = 0$ naar $x = W$. We laten zien hoe zo'n kortste pad achterstevoren geconstrueerd kan worden. Stel nu dat b_k de laatste boot is op zo'n kortste pad. We kiezen b_k als onze laatste boot b_n. Als b_{n-1} kiezen we de boot b_{k-1} die de overstap $(b_{k-1}, T[k], b_k)$ mogelijk maakt. Als b_{n-2} kiezen we de boot b_{k-2} die de overstap $(b_{k-2}, T[k-1], b_{k-1})$ mogelijk maakt, etc. Op deze manier kunnen nieuwe boten aan het pad worden toegevoegd, totdat we een boot b_0 met linkergrens $L_0 = 0$ tegenkomen. Het geconstrueerde pad is nu van de gewenste vorm en, omdat $b_n = b_k$, is het pad ook een kortste pad van $x = 0$ naar $x = W$. Boot b_k wordt immers minstens zo vroeg bereikt als in het originele kortste pad. □

Het eerst mogelijke tijdstip $T[i]$ waarop een boot b_i bereikt kan worden, is in zekere zin ook het beste tijdstip om b_i te bereiken. Als Mario b_i op $T[i]$ bereikt kan hij immers geen enkele overstapmogelijkheid vanaf b_i mislopen.
Nu we weten dat er altijd een kortste pad bestaat waarbij elke boot b_i met $0 < i \leq n$ wordt bereikt op tijdstip $T[i]$, kan aanname 1 bewezen worden door aan te tonen dat b_i bij de overstap $(b_{i-1}, T[i], b_i)$ naar links vaart.

Lemma 2.5. *Zij b_i een boot met linkergrens $L_i \neq 0$ en $T[i]$ het eerst mogelijke tijdstip waarop b_i bereikt kan worden. Bij de overstap naar b_i op $T[i]$ zal de boot altijd naar links varen.*

Bewijs. Op tijdstip $t = 0$ bevindt Mario zich links van b_i, omdat $x = 0 < L_i$. Merk op dat Mario een continue reis aflegt. Hij kan zich immers alleen verplaatsen door op een boot te varen. Mario kan om deze reden onmogelijk rechts van b_i komen zonder de boot te kruisen. Zij nu b_{i-1} de boot die de overstap $(b_{i-1}, T[i], b_i)$ mogelijk maakt. Op het tijdstip $T[i-1]$ waarop b_{i-1} voor het eerst bereikt kan worden, geldt dat $pos(b_{i-1}, T[i-1]) < pos(b_i, T[i-1])$, omdat $T[i-1] < T[i]$ moet zijn en Mario zich tot $T[i]$ links van b_i bevindt. Uit Lemma 2.3 volgt nu dat b_i bij de overstap $(b_{i-1}, T[i], b_i)$ naar links vaart. □

Het bewijs van aanname 1 volgt direct uit Lemma's 2.4 en 2.5.

Stelling 2.6. *Als $x = W$ bereikbaar is, dan bestaat er altijd een kortste pad van $x = 0$ naar $x = W$ waarbij alleen wordt overgestapt op boten die bij de overstap naar links varen.*

2.2.2 Bewijs aanname 2

Om aanname 2 te bewijzen moeten we iets kunnen zeggen over de mogelijke overstappunten tussen boten in een bepaald tijdsinterval. Een bruikbaar gegeven hierbij is dat boten een continue reis maken over de rivier. Dit heeft als gevolg dat twee boten elkaar niet kunnen passeren zonder dat er een overstapmogelijkheid

plaatsvindt.

Lemma 2.7. *Zij b_1 en b_2 twee boten, en t_1 en t_2 twee tijdstippen met $t_1 \leq t_2$. Als $pos(b_1, t_1) \leq pos(b_2, t_1)$ en $pos(b_1, t_2) \geq pos(b_2, t_2)$, dan bestaat er een tijdstip t' met $t_1 \leq t' \leq t_2$, zodat $pos(b_1, t') = pos(b_2, t')$.*

Omdat ook Mario een continue reis aflegt, kan Lemma 2.7 gemakkelijk worden omgezet naar een stelling over Mario.

Lemma 2.8. *Zij b een boot, en t_1 en t_2 twee tijdstippen met $t_1 \leq t_2$. Als Mario zich op t_1 op een positie $x_1 \leq pos(b, t_1)$ bevindt en zich op t_2 op een positie $x_2 \geq pos(b, t_2)$ bevindt, dan bestaat er een tijdstip t' met $t_1 \leq t' \leq t_2$ waarop Mario en b dezelfde positie hebben.*

Aanname 2 is nu eenvoudig te bewijzen met behulp van Lemma 2.8:

Stelling 2.9. *Zij $T[i]$ het eerst mogelijke tijdstip waarop boot b_i bereikt kan worden, en zij b_1 en b_2 twee boten. Als $pos(b_1, T[2]) \leq pos(b_2, T[2])$, dan geldt ook dat $T[1] \leq T[2]$.*

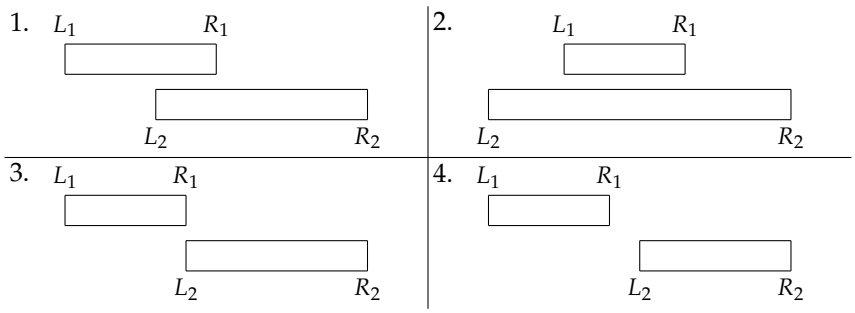
Bewijs. Op tijdstip $t = 0$ bevindt Mario zich op positie $x = 0 \leq pos(b_1, 0)$. Op tijdstip $T[2]$ kan Mario zich op $x = pos(b_2, T[2]) \geq pos(b_1, T[2])$ bevinden. Volgens Lemma 2.8 moet nu wel gelden dat $0 \leq T[1] \leq T[2]$. □

2.3 Tussen welke boten is een overstap mogelijk?

Elke keer dat het algoritme van Dijkstra een nieuwe knoop bezoekt, moet het algoritme de uitgaande pijlen van die knoop bepalen. Dit doet het algoritme door het tijdstip van de eerstvolgende overstapmogelijkheid tussen twee boten te berekenen met QuickSim. Er bestaan echter veel boten die elkaar nooit zullen kruisen. In deze paragraaf introduceren we een simpele methode waarmee bepaald kan worden of een overstap tussen twee boten überhaupt mogelijk is. Het doelloos zoeken naar het tijdstip van de eerstvolgende overstap wordt hiermee voorkomen, wat de rekensnelheid van het algoritme van Dijkstra zal verhogen.
Merk overigens op dat als twee boten elkaar ooit kruisen, ze elkaar oneindig vaak zullen kruisen. Als p_1 en p_2 de periodes van de twee boten zijn, dan herhaalt de beweging van de boten zich namelijk altijd na maximaal $p_1 \cdot p_2$ seconden.

Definitie 2.10. Zij $b_1 = [L_1, R_1]$ en $b_2 = [L_2, R_2]$ twee boten. De functie $overlap(b_1, b_2) = min(R_1, R_2) - max(L_1, L_2)$ geeft de overlap van de bereiken van b_1 en b_2 in afstandseenheden. We onderscheiden vier verschillende vormen van overlap, zoals afgebeeld in Figuur 2.2:

1. Als $0 < overlap(b_1, b_2) < \min(R_1 - L_1, R_2 - L_2)$, dan zeggen we dat b_1 en b_2 *deels overlappend* zijn.



Figuur 2.2: De vier vormen van overlap tussen twee boten $b_1 = [L_1, R_1]$ en $b_2 = [L_2, R_2]$: 1. deels overlappend; 2. volledig overlappend; 3. minimaal overlappend; 4. niet overlappend.

2. Als $overlap(b_1, b_2) = \min(R_1 - L_1, R_2 - L_2)$, dan zeggen we dat b_1 en b_2 *volledig overlappend* zijn.

3. Als $overlap(b_1, b_2) = 0$, dan zeggen we dat b_1 en b_2 *minimaal overlappend* zijn.

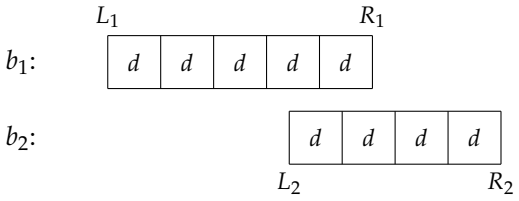
4. Als $overlap(b_1, b_2) < 0$, dan zeggen we dat b_1 en b_2 *niet overlappend* zijn.

Twee niet overlappende boten zullen elkaar nooit kruisen. De bereiken van zulke boten zijn immers disjunct. Daarentegen zullen twee volledig overlappende boten elkaar altijd kruisen. Als de boot met het grootste bereik zijn periode aflegt, zal deze immers het volledige bereik van de andere boot passeren.
In het vervolg van deze paragraaf zullen we deze twee triviale gevallen buiten beschouwing laten. We nemen dus aan dat elk paar boten deels of minimaal overlappend is. Voor elk paar niet-triviale boten b_1 en b_2 zullen we bewijzen dat aan de hand van de grootste gemene deler van de halve periodes en de grootte van de overlap van de boten te zien is of de boten elkaar ooit kruisen.

Stelling 2.11. *Zij $b_1 = [L_1, R_1]$ en $b_2 = [L_2, R_2]$ twee deels overlappende boten, zodat $L_1 < L_2 < R_1 < R_2$. Zij verder $l_1 = R_1 - L_1$ en $l_2 = R_2 - L_2$ de halve periodes van b_1 en b_2. Laat d de grootste gemene deler van l_1 en l_2 zijn. Als $d \leq overlap(b_1, b_2)$ dan zullen b_1 en b_2 elkaar ooit kruisen.*

Merk op dat Stelling 2.11 niet toepasbaar is op minimaal overlappende boten, omdat de overlap van zulke boten 0 is en een grootste gemene deler nooit ≤ 0 kan zijn.

Bewijs. De in de stelling beschreven situatie wordt geïllustreerd door Figuur 2.3. Volgens de identiteit van Bézout bestaan er twee gehele getallen a en b ongelijk aan 0, zodat $a \cdot l_1 + b \cdot l_2 = d$. In paragraaf 33.2 van [CLR90] wordt uitgelegd hoe het uitgebreide algoritme van Euclides gebruikt kan worden om oneindig veel oplossingen voor deze vergelijking te vinden, waarbij altijd geldt dat a óf b negatief is. Laat $a > 0$, $b < 0$ en $b' = -b$. We kunnen de identiteit van Bézout nu omschrijven naar $a \cdot 2l_1 = b' \cdot 2l_2 + 2d$. Deze vergelijking stelt dat steeds als b_1 zijn periode a keer aflegt, b_2 zijn periode b' keer aflegt plus $2d$ extra afstandseenheden. Als we tijdstappen van $a \cdot 2l_1$ seconden nemen, zal b_1 daarom stil blijven liggen terwijl b_2 steeds $2d$ afstandseenheden verder vaart.
We bekijken nu de positie van b_2 als b_1 zich op R_1 bevindt, en doen hierbij tijdstappen van $a \cdot 2l_1$ seconden. Als



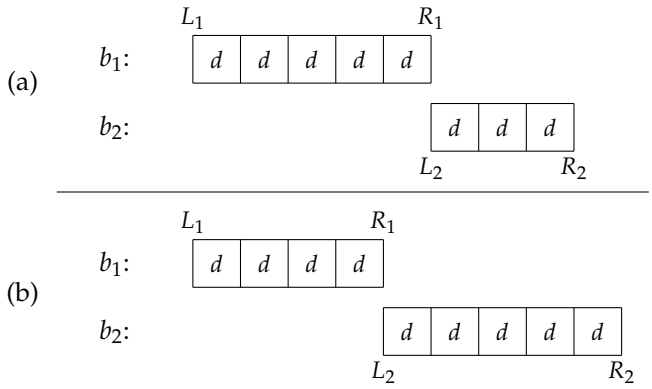
Figuur 2.3: Twee boten met een overlap die minstens zo groot is als de grootste gemene deler d van hun halve periodes.

$overlap(b_1, b_2) \geq d$ kost het b_2 op zijn minst $2d$ seconden om het overlappende deel van zijn periode af te leggen. Er moet daarom een tijdstip t bestaan waarop b_2 het overlappende deel van zijn bereik is binnengevaren. Elke tijdstap vaart b_2 immers slechts $2d$ afstandseenheden verder. Op t geldt nu dat $pos(b_1, t) \geq pos(b_2, t)$, omdat b_1 zich op R_1 bevindt. Verder geldt dat $pos(b_1, 0) < pos(b_2, 0)$, omdat $L_1 < L_2$. Volgens Lemma 2.7 moeten b_1 en b_2 elkaar daarom wel kruisen. □

Stelling 2.11 vertelt ons nog niets over boten b_1 en b_2 met $ggd(l_1, l_2) > overlap(b_1, b_2) > 0$. Wel weten we dat sommige boten met deze eigenschap elkaar wel kruisen en sommige niet. Zo kruisen boten $[0, 2]$ en $[2, 3]$ elkaar al na twee seconden, maar kruisen boten $[0, 1]$ en $[1, 3]$ elkaar nooit. We bekijken nu eerst minimaal overlappende boten en laten daarna zien welke boten met een grotere overlap elkaar kruisen.

Lemma 2.12. *Zij $b_1 = [L_1, R_1]$ en $b_2 = [L_2, R_2]$ twee minimaal overlappende boten, met $L_1 < R_1 = L_2 < R_2$. Zij verder $l_1 = R_1 - L_1$ en $l_2 = R_2 - L_2$ de halve periodes van b_1 en b_2. Laat d de grootste gemene deler zijn van l_1 en l_2, zodat $l_1 = a \cdot d$ en $l_2 = b \cdot d$. De boten zullen elkaar kruisen dan en slechts dan als a even is.*

Bewijs. Een kruising tussen b_1 en b_2 kan alleen plaatsvinden op $x = R_1 = L_2$, omdat dit het enige punt is waar de bereiken van de boten overlappen.
⇒ Stel dat a oneven is. Deze situatie wordt geïllustreerd door Figuur 2.4a. Om R_1 te bereiken moet b_1 een oneven aantal keer l_1 afleggen. Laat x een oneven getal zijn. Boot b_1 bevindt zich op het tijdstip $x \cdot l_1 = x \cdot a \cdot d$ op positie R_1. Merk op dat $x \cdot a$ oneven is, omdat x en a beide oneven zijn. Om L_2 te bereiken moet b_2 een even aantal keer l_2 afleggen. Laat y een even getal zijn. Boot b_2 bevindt zich op het tijdstip $y \cdot l_2 = y \cdot b \cdot d$ op positie L_2. Merk op dat $y \cdot b$ even is, omdat y even is.
Er kan geen tijdstip t bestaan, waarop geldt dat $pos(b_1, t) = R_1 = L_2 = pos(b_2, t)$. Om R_1 te bereiken moet b_1 immers een oneven aantal keer d afleggen, terwijl b_2 een even aantal keer d moet afleggen om op L_2 te komen. Als b_1 en b_2 elkaar kruisen, moet a dus wel even zijn.
⇐ Stel dat a even is, dan moet b wel oneven zijn, omdat $ggd(a, b) = 1$. Deze situatie wordt geïllustreerd door Figuur 2.4b. Op tijdstip $t = a \cdot b \cdot d = l_1 \cdot b = l_2 \cdot a$ geldt $pos(b_1, t) = R_1 = L_2 = pos(b_2, t)$, omdat b_1 een oneven aantal keer l_1 heeft afgelegd en b_2 een even aantal keer l_2 heeft afgelegd. Als a even is, moeten b_1 en b_2 elkaar dus wel kruisen. □



Figuur 2.4: (a) Een voorbeeldsituatie als a oneven is. (b) Een voorbeeldsituatie als a even is.

Nu we weten welke minimaal overlappende boten elkaar kruisen, kunnen we ook bepalen welke deels overlappende boten b_1 en b_2 met $ggd(l_1, l_2) > overlap(b_1, b_2)$ elkaar kruisen. Voor elk paar van zulke boten bestaat er namelijk een minimaal overlappend paar boten b_1' en b_2', zodat b_1 en b_2 elkaar kruisen dan en slechts dan als b_1' en b_2' elkaar ook kruisen.

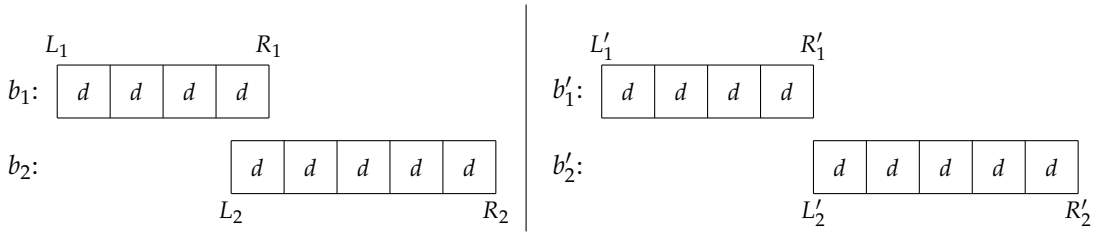
Stelling 2.13. *Zij $b_1 = [L_1, R_1]$ en $b_2 = [L_2, R_2]$ twee deels of minimaal overlappende boten, zodat $L_1 < L_2 \leq R_1 < R_2$. Zij verder $l_1 = R_1 - L_1$ en $l_2 = R_2 - L_2$ de halve periodes van b_1 en b_2. Laat d de grootste gemene deler zijn van l_1 en l_2, zodat $l_1 = a \cdot d$ en $l_2 = b \cdot d$. Als $d > overlap(b_1, b_2)$, dan kruisen de boten dan en slechts dan als a even is.*

Bewijs. Als b_1 en b_2 minimaal overlappen, dus als $L_2 = R_1$, dan volgt de bewering uit Lemma 2.12. Neem nu aan dat $L_2 < R_1$. We definiëren twee boten b_1' en b_2' als volgt:

$$b_1' = [L_1', R_1'] = [L_1, R_1]$$
$$b_2' = [L_2', R_2'], \text{ met } L_2' = R_1' = R_1 \text{ en } R_2' = L_2' + (R_2 - L_2)$$

Boot b_1' is dus gelijk aan b_1, met dezelfde halve periode l_1. Boot b_2' is een 'verschoven versie' van b_2, met dezelfde halve periode l_2. Beide paren boten zijn afgebeeld in Figuur 2.5. We bewijzen de stelling door aan te tonen dat b_1 en b_2 elkaar kruisen dan en slechts dan als b_1' en b_2' elkaar kruisen.
$\Rightarrow$ Laat $t_i = i \cdot d$ met $i \in \mathbb{N}$ een tijdstip zijn. Op dit tijdstip geldt dat $pos(b_1, t_i) = L_1 + x \cdot d$ met $x \in \{0, 1, \ldots, a\}$ en $pos(b_2, t_i) = L_2 + y \cdot d$ met $y \in \{0, 1, \ldots, b\}$, omdat l_1 en l_2 deelbaar zijn door d. Stel nu dat t_i het laatste tijdstip is dat deelbaar is door d, voordat b_1 en b_2 elkaar voor het eerst kruisen. Deze kruising moet plaatsvinden in het overlappende deel van de bereiken van b_1 en b_2. Verder volgt uit Lemma 2.3 dat b_1 naar rechts en b_2 naar links vaart bij deze kruising. De posities van b_1 en b_2 op t_i zijn daarom $pos(b_1, t_i) = L_1 + (a-1) \cdot d$ en $pos(b_2, t_i) = L_2 + 1 \cdot d$. Nu geldt op tijdstip t_{i+1} dat $pos(b_1, t_{i+1}) = L_1 + a \cdot d = R_1$ en $pos(b_2, t_{i+1}) = L_2 + 0 \cdot d = L_2$. Omdat b_1' en b_2' dezelfde halve periodes l_1 en l_2 hebben, geldt dan ook $pos(b_1', t_{i+1}) = R_1' = L_2' = pos(b_2', t_{i+1})$, en dus kruisen b_1' en b_2' elkaar.

$\Leftarrow$ Stel dat b'_1 en b'_2 elkaar kruisen op tijdstip t. Op t geldt dan dat $pos(b'_1, t) = R'_1 = L'_2 = pos(b'_2, t)$. Omdat b'_1 en b'_2 dezelfde halve periodes l_1 en l_2 hebben, geldt ook dat $pos(b_1, t) = R_1 \geq L_2 = pos(b_2, t)$. Volgens Lemma 2.7 moeten b_1 en b_2 elkaar nu wel kruisen, omdat $pos(b_1, 0) = L_1 < L_2 = pos(b_2, 0)$.
Lemma 2.12 stelt nu dat b_1 en b_2 elkaar kruisen dan en slechts dan als b'_1 en b'_2 elkaar kruisen dan en slechts dan als a is even. □



Figuur 2.5: Een voorbeeld van b'_1 en b'_2 bij twee boten b_1 en b_2 met overlap kleiner dan de grootste gemene deler d van hun halve periodes.

Stellingen 2.11 en 2.13 dekken samen alle mogelijke niet-triviale gevallen af. Merk op dat een niet-triviaal paar boten b_1 en b_2 met gelijke periodes (en dus ook gelijke halve periodes l_1 en l_2) elkaar vanwege Stelling 2.13 nooit zullen kruisen. Er geldt voor zulke boten namelijk dat $ggd(b_1, b_2) = l_1 = l_2$, zodat $a = b = 1$ oneven is. Dit is niet verrassend. Omdat de boten dezelfde periode afleggen, zal de afstand tussen b_1 en b_2 namelijk altijd gelijk blijven. In zekere zin is een paar boten met gelijke periodes dus ook een triviaal geval.

Algoritme 3 (JunctionPossible) is met behulp van Stellingen 2.11 en 2.13 opgesteld. JunctionPossible krijgt twee boten als invoer en geeft als uitvoer aan of deze boten elkaar ooit zullen kruisen. In het algoritme wordt eerst gecontroleerd of de gegeven boten een triviaal geval vormen. Als dit niet het geval is worden de tests uit Stellingen 2.11 en 2.13 toegepast. De worst case complexiteit van JunctionPossible is gelijk aan die van de functie ggd (logaritmisch), omdat het algoritme verder geen loops bevat. In Algoritme 2 kunnen we JunctionPossible aanroepen voordat we met QuickSim het gewicht van een pijl bepalen. QuickSim hoeft zo nooit het tijdstip van de eerstvolgende kruising tussen twee boten te berekenen die elkaar niet kruisen.

2.4 Het tijdstip van de eerstvolgende kruising

Het berekenen van de gewichten op de pijlen uit de graaf is de meest voorkomende stap in het algoritme van Dijkstra. Het is daarom belangrijk dat deze berekening zo snel mogelijk verloopt. In deze paragraaf bespreken we de huidige functie waarmee het algoritme van Dijkstra de gewichten bepaald, waarna we een alternatieve methode introduceren.

Algoritme 3 JunctionPossible(b_1,b_2)

$l_1 := R_1 - L_1$
$l_2 := R_2 - L_2$
if $overlap(b_1, b_2) < 0$ **then**
    return false
else if $overlap(b_1, b_2) = l_1$ **or** $overlap(b_1, b_2) = l_2$ **then**
    return true
else
    $d := ggd(l_1, l_2)$
    if $d \leq overlap(b_1, b_2)$ **then**
        return true
    if $L_1 < L_2$ **then**
        $a := l_1/d$
    else
        $a := l_2/d$
    if $a \mod 2 = 0$ **then**
        return true
return false

2.4.1 QuickSim

Voor twee boten b_1 en b_2 kan het tijdstip van een eerstvolgende kruising eenvoudig berekend worden met een brute-force algoritme. In dit algoritme worden de toekomstige posities van b_1 en b_2 per halve seconde gesimuleerd, totdat de boten elkaar kruisen, of totdat de beginsituatie zich herhaalt. In het laatste geval kruisen de boten elkaar nooit. Stel nu dat l_1 en l_2 de halve periodes zijn van b_1 en b_2. Het getal $2 \cdot l_1 \cdot l_2$ is dan deelbaar door zowel de periode van b_1 als de periode van b_2. De beginsituatie herhaalt zich daarom altijd na $2 \cdot l_1 \cdot l_2$ seconden. Dit leidt tot een worst case complexiteit van $O(l_1 \cdot l_2)$ voor dit algoritme.
Het tijdstip van de eerstvolgende kruising kan echter ook sneller berekend worden. Het algoritme van Dijkstra maakt hiervoor gebruik van Algoritme 4 (QuickSim). QuickSim is een verbeterde versie van het zojuist beschreven brute-force algoritme, waarin grotere tijdstappen worden gemaakt tijdens de simulatie. De invoer van QuickSim bestaat uit twee boten $b_1 = [L_1, R_1]$ en $b_2 = [L_2, R_2]$ en een begintijdstip t_0, waarbij wordt aangenomen dat $pos(b_1, t_0) < pos(b_2, t_0)$. De uitvoer van het algoritme is het eerste tijdstip na t_0 waarop b_1 en b_2 elkaar kruisen. Als de b_1 en b_2 elkaar nooit kruisen, dan geeft het algoritme -1 terug.

In de while-loop van QuickSim worden de toekomstige posities van b_1 en b_2 gesimuleerd. Hierbij wordt een teller t gebruikt om de tijd bij te houden. Omdat $pos(b_1, t_0) < pos(b_2, t_0)$, stelt Lemma 2.3 dat b_1 naar rechts en b_2 naar links moet varen bij de eerstvolgende kruising na t_0. Tijdstippen waarop b_1 of b_2 niet in de juiste richting varen hoeven dus niet gesimuleerd te worden. Als b_1 naar links vaart wordt de waarde van t daarom opgehoogd naar het tijdstip waarop de boot omkeert. Ditzelfde gebeurt als b_2 naar rechts vaart.
Zodra beide boten in de juiste richting varen, wordt er gecontroleerd of de boten elkaar kruisen voordat één van de boten weer van richting verandert. Hierbij geldt dat de boten elkaar kruisen als de afstand tussen de boten (dt) kleiner of gelijk is aan tweemaal de tijd totdat één van de boten omkeert ($2 \cdot dtmax$). Op het

Algoritme 4 QuickSim(b_1,b_2,t_0)

preconditie: $pos(b_1, t_0) < pos(b_2, t_0)$
  $t := t_0$
  while $t < t_0 + 2 \cdot (R_1 - L_1) \cdot (R_2 - L_2)$ **do**
    if $pos(b_1, t) > pos(b_1, t + \frac{1}{2})$ **then** // b_1 vaart naar links
      $t := t + pos(b_1, t) - L_1$
    else if $pos(b_2, t) < pos(b_2, t + \frac{1}{2})$ **then** // b_2 vaart naar rechts
      $t := t + R_2 - pos(b_2, t)$
    else
      $dtmax := \min(R_1 - pos(b_1, t), pos(b_2, t) - L_2)$
      $dt := pos(b_2, t) - pos(b_1, t)$
      if $dt \leq 2 \cdot dtmax$ **then**
        return $t + dt/2$
      else
        $t := t + dtmax$
  return -1

moment dat beide boten naar elkaar toe varen, neemt de afstand tussen de boten namelijk elke seconde met twee afstandseenheden af. Als $dt \leq 2 \cdot dtmax$, dan passeren de boten elkaar daarom na $\frac{dt}{2}$ seconden en geeft het algoritme $t + \frac{dt}{2}$ terug. Als $dt > 2 \cdot dtmax$, dan wordt t met $dtmax$ verhoogd.
Net als bij het brute-force algoritme stopt QuickSim zodra de beginsituatie zich herhaalt. Als $t \geq t_0 + 2 \cdot (R_1 - L_1) \cdot (R_2 - L_2)$, stapt het algoritme daarom uit de while-loop. In dit geval wordt -1 teruggegeven om aan te geven dat het algoritme geen kruising heeft kunnen vinden.

Om de complexiteit van QuickSim te meten, tellen we het aantal iteraties van de while-loop uit het algoritme. In de best case kan het tijdstip van de eerstvolgende kruising tussen b_1 en b_2 in constante tijd berekend worden. De boten varen in dit geval al in de goede richting en er geldt dat $dt \leq 2 \cdot dtmax$. Het algoritme gaat nu maar één keer de while-loop binnen. Een voorbeeld van een best case is $b_1 = [0, 2]$, $b_2 = [2, 3]$ en $t_0 = 1$.
Bij het uitvoeren van QuickSim wordt t elke loop iteratie (op de laatste iteratie na) één keer opgehoogd. Een bovengrens voor de worst case complexiteit van QuickSim kan daarom gevonden worden door het maximale aantal ophogingen van t te bepalen. We bekijken hiervoor de deels of minimaal overlappende boten b_1 en b_2 met halve periodes l_1 en l_2, waarbij $l_1 < l_2$. De analyse voor het geval dat $l_1 > l_2$ gaat analoog. De grens van een deels of minimaal overlappende boot die in het overlappende deel van zijn bereik ligt, noemen we ook wel de *overlappende grens* van de boot. Er zijn voor b_1 en b_2 vier verschillende situaties waarin t wordt opgehoogd:

1. Boot b_1 vaart in de verkeerde richting. De waarde van t wordt opgehoogd, zodat b_1 op zijn niet-overlappende grens komt.

2. Boot b_2 vaart in de verkeerde richting. De waarde van t wordt opgehoogd, zodat b_2 op zijn niet-overlappende grens komt.

3. Boot b_1 vaart in de juiste richting, maar keert om voordat de boten elkaar kruisen. De waarde van t

wordt opgehoogd, zodat b_1 op zijn overlappende grens komt.

4. Boot b_2 vaart in de juiste richting, maar keert om voordat de boten elkaar kruisen. De waarde van t wordt opgehoogd, zodat b_2 op zijn overlappende grens komt.

Tussen elk tweetal momenten waarop b_1 op zijn niet-overlappende grens wordt gezet, zitten ten minste $2 \cdot l_1$ seconden. In een totale tijd van $2 \cdot l_1 \cdot l_2$ seconden kan de ophoging uit situatie 1 daarom hoogstens l_2 keer voorkomen. Om dezelfde reden kan ook de ophoging uit situatie 3 hoogstens l_2 keer voorkomen, en kunnen de ophogingen uit situaties 2 en 4 elk hoogstens l_1 keer voorkomen. Het totale aantal ophogingen van t is daarom kleiner dan of gelijk aan $2 \cdot l_1 + 2 \cdot l_2$. In het algemeen leidt dit tot een bovengrens van $O(l_1 + l_2)$ voor de worst case complexiteit van QuickSim.

Deze bovengrens kan ook gehaald worden. Zij $b_1 = [0, l_1]$ en $b_2 = [l_1 - 1, 2 \cdot l_1]$ twee boten, met l_1 een even getal ≥ 10. De halve periode van b_1 is l_1 en de halve periode van b_2 is $l_2 = 2 \cdot l_1 - (l_1 - 1) = l_1 + 1$. Omdat l_1 even en l_2 oneven is, bevinden beide boten zich op het tijdstip $l_1 \cdot l_2$ op hun overlappende grens. De boten kruisen elkaar daarom op tijdstippen $l_1 \cdot l_2 - \frac{l_1 - (l_1 - 1)}{2} = l_1 \cdot l_2 - \frac{1}{2}$ en $l_1 \cdot l_2 + \frac{l_1 - (l_1 - 1)}{2} = l_1 \cdot l_2 + \frac{1}{2}$.
Bij het uitvoeren van QuickSim op de instantie $b_1 = [0, l_1]$, $b_2 = [l_1 - 1, 2 \cdot l_1]$ en $t_0 = l_1 \cdot l_2 + 1$ (direct na de tweede kruising), worden drie stappen steeds herhaald (zie ook Voorbeeld 2.14).

1. t wordt met l_1 opgehoogd, waardoor b_1 weer in de juiste richting vaart. Na stap 3 bevindt b_1 zich namelijk op positie $x = l_1$. Op t_0 bevindt b_1 zich op positie $x = l_1 - 1$ en vaart de boot naar links; t wordt daarom in dit geval opgehoogd met $l_1 - 1$.

2. t wordt met $1 + 2 \cdot k_1$ opgehoogd (na k_1 iteraties van deze stappen), waardoor b_2 weer in de juiste richting vaart. Op tijdstip $t_0 + l_1 - 1$ bevindt b_2 zich op positie $x = 2 \cdot l_1 - 1$ en vaart de boot naar rechts. Omdat $2 \cdot l_2 - 2 \cdot l_1 = 2$, gaat b_2 na elke iteratie twee afstandseenheden terug in zijn periode.

3. t wordt met $dtmax = l_1 - (1 + 2 \cdot k_1)$ opgehoogd, omdat b_1 omkeert voordat de boten elkaar kruisen. Boot b_1 bevindt zich op dit moment elke iteratie twee afstandseenheden verder in zijn periode, waardoor $dtmax$ kleiner wordt.

Tijdens iteratie $\frac{l_1}{2}$ bevindt b_1 zich na stap 2 op positie $x = 1 + 2 \cdot (\frac{l_1}{2} - 1) = l_1 - 1$, waarbij de boot naar rechts vaart. Als b_1 tijdens stap 2 van iteratie $\frac{l_1}{2} + 1$ nog eens twee afstandseenheden verder in zijn periode wordt gezet, zal de situatie voor stap 3 veranderen. Boot b_1 is dan namelijk over zijn overlappende grens heen gezet, waardoor de boot nu naar links vaart. Vanaf dit moment zullen drie andere stappen zich steeds herhalen:

1. t wordt met $l_1 - (1 + 2 \cdot k_2)$ opgehoogd (na k_2 iteraties van deze stappen), waardoor b_1 weer in de juiste richting vaart. Omdat $2 \cdot l_2 - 2 \cdot l_1 = 2$, raakt b_1 na elke iteratie twee afstandseenheden verder in zijn periode.

2. t wordt met $dtmax = 2 + 2 \cdot k_2$ opgehoogd, omdat b_2 omkeert voordat de boten elkaar kruisen. Tijdens

de eerste iteratie bevindt b_2 zich na stap 1 op positie $x = l_1 + 1$ en vaart de boot naar links. Boot b_2 bevindt zich op dit moment na elke iteratie twee afstandseenheden minder ver in zijn periode.

3. t wordt met $l_1 + 1$ opgehoogd, waardoor b_2 weer in de juiste richting vaart. Na stap 2 bevindt b_2 zich namelijk op positie $x = l_1 - 1$.

De positie van b_1 is na stap 1 altijd $x = 0$. Tijdens iteratie $\frac{l_1}{2}$ bevindt b_2 zich na stap 1 op positie $x = l_1 - 1 + 2 + 2 \cdot (\frac{l_1}{2} - 1) = 2 \cdot l_1 - 1$. Zowel b_1 als b_2 bereikt vanaf dit moment na l_1 seconden zijn overlappende grens. Er geldt daarom dat $dt = 2 \cdot l_1 - 1 \leq 2 \cdot l_1 = 2 \cdot dtmax$ en dus stopt het algoritme.
Het eerste drietal stappen verhoogt de waarde van t in totaal $3 \cdot \frac{l_1}{2} + 2$ keer (na stap 2 van iteratie $\frac{l_1}{2} + 1$ verandert de situatie). Het tweede drietal stappen verhoogt de waarde van t in totaal $3 \cdot (\frac{l_1}{2} - 1) + 1$ keer (de boten kruisen elkaar na stap 1 van iteratie $\frac{l_1}{2}$). Het totale aantal ophogingen komt daarmee uit op $3 \cdot \frac{l_1}{2} + 2 + 3 \cdot (\frac{l_1}{2} - 1) + 1 = 3 \cdot l_1$. Het algoritme gaat, vanwege de laatste iteratie, daarom in totaal $3 \cdot l_1 + 1 = 2 \cdot l_1 + l_2$ keer de while-loop binnen. De bovengrens van $O(l_1 + l_2)$ is hiermee gehaald.

Voorbeeld 2.14. Zij $b_1 = [0, 10]$ en $b_2 = [9, 20]$ twee boten. Dit paar boten vormt een worst case van QuickSim, waarbij $l_1 = 10$. Vanaf het tijdstip $t_0 = l_1 \cdot l_2 + 1 = 10 \cdot 11 + 1 = 111$ zal t daarom $3 \cdot l_1 = 3 \cdot 10 = 30$ keer worden opgehoogd. In Tabel 2.1 staat het aantal seconden waarmee t wordt opgehoogd tijdens het eerste drietal stappen en in Tabel 2.2 staat dit voor het tweede drietal stappen. △

Tabel 2.1: Het aantal seconden waarmee t wordt opgehoogd tijdens het eerste drietal stappen.

→ iteratie	1	2	3	4	5	6
stap 1	9	10	10	10	10	10
stap 2	1	3	5	7	9	11
stap 3	9	7	5	3	1	

Tabel 2.2: Het aantal seconden waarmee t wordt opgehoogd tijdens het tweede drietal stappen.

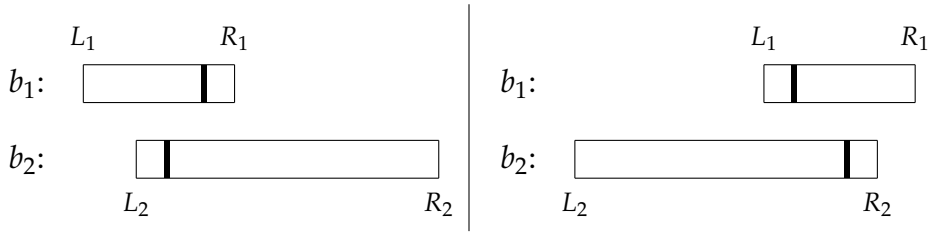
→ iteratie	1	2	3	4	5
stap 1	9	7	5	3	1
stap 2	2	4	6	8	
stap 3	11	11	11	11	

2.4.2 NextJunction

Omdat de berekening van het tijdstip van de eerstvolgende kruising tussen twee boten de meest voorkomende stap is in het algoritme van Dijkstra, resulteert een kleine verbetering in deze stap mogelijk in een relatief grote versnelling van de totale rekentijd van het algoritme. Om deze reden is er gezocht naar algoritmes die sneller werken dan QuickSim. Dankzij de bevindingen uit paragraaf 2.3 mocht hierbij worden aangenomen dat de boten elkaar ooit zullen kruisen.

NextJunctionA

Als eerste hebben we gekeken naar instanties waarbij de tijd tot de eerstvolgende kruising in constante tijd berekend kan worden. Zij $b_1 = [L_1, R_1]$ en $b_2 = [L_2, R_2]$ twee volledig overlappende boten met halve periodes



Figuur 2.6: Instanties waarbij de volgende kruising binnen $2 \cdot l_2$ seconden plaatsvindt.

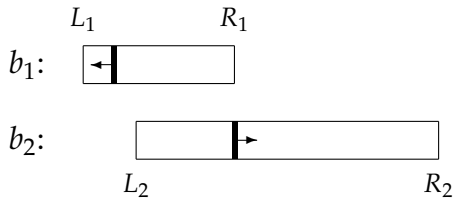
l_1 en l_2, zodat $overlap(b_1, b_2) = l_1$. De eerstvolgende kruising tussen b_1 en b_2 zal altijd binnen $2 \cdot l_2$ seconden plaatsvinden, omdat b_2 in deze tijd het volledig bereik van b_1 passeert.
Zij t een tijdstip. Stel nu dat $0 < overlap(b_1, b_2) < l_1$ en $l_1 < l_2$. Als $L_1 < L_2$ en $pos(b_1, t) > pos(b_2, t)$ of als $R_1 > R_2$ en $pos(b_1, t) < pos(b_2, t)$, zoals afgebeeld in Figuur 2.6, dan zal de eerstvolgende kruising tussen b_1 en b_2 binnen $2 \cdot l_1$ seconden plaatsvinden. Boot b_1 moet b_2 immers passeren om zijn periode af te leggen.

Het tijdstip van de eerstvolgende kruising kan voor de zojuist beschreven instanties in constante tijd berekend worden met Algoritme 5 (NextJunctionA). De invoer van dit algoritme bestaat uit twee boten $b_1 = [L_1, R_1]$ en $b_2 = [L_2, R_2]$ en een begintijdstip t, waarbij wordt aangenomen dat $pos(b_1, t) \neq pos(b_2, t)$ en b_1 volledig wordt overlapt door b_2, of dat b_1 en b_2 een instantie uit Figuur 2.6 vormen. De uitvoer van het algoritme is het eerste tijdstip na t waarop b_1 en b_2 elkaar kruisen.

Algoritme 5 NextJunctionA(b_1,b_2,t)

preconditie: ($pos(b_1, t) \neq pos(b_2, t)$ en $overlap(b_1, b_2) = R_1 - L_1$) of b_1 en b_2 vormen een instantie uit Figuur 2.6
if $pos(b_1, t) > pos(b_2, t)$ **then**
  if $pos(b_2, t) < pos(b_2, t + \frac{1}{2})$ **then** // b_2 vaart naar rechts
    $t := t + R_1 - pos(b_2, t)$
  else
    $t := t + pos(b_2, t) - L_2 + R_1 - L_2$
  if $pos(b_1, t) < pos(b_1, t + \frac{1}{2})$ **or** $pos(b_1, t) = R_1$ **then**
    $t := t - (pos(b_1, t) - L_1)$
  $t := t - (pos(b_2, t) - pos(b_1, t))/2$
else
  if $pos(b_2, t) > pos(b_2, t + \frac{1}{2})$ **then** // b_2 vaart naar links
    $t := t + pos(b_2, t) - L_1$
  else
    $t := t + R_2 - pos(b_2, t) + R_2 - L_1$
  if $pos(b_1, t) > pos(b_1, t + \frac{1}{2})$ **or** $pos(b_1, t) = L_1$ **then**
    $t := t - (R_1 - pos(b_1, t))$
  $t := t - (pos(b_1, t) - pos(b_2, t))/2$
return t

Voor een instantie met gedeeltelijke overlap waarbij $pos(b_1, t) > pos(b_2, t)$, zoals te zien in het linker plaatje van Figuur 2.6, berekent het algoritme eerst het aantal seconden dat b_2 nodig heeft om R_1 te bereiken, en telt dit vervolgens bij t op. Dit aantal is $R_1 - pos(b_2, t)$ als b_2 naar rechts vaart en $pos(b_2, t) - L_2 + R_1 - L_2$ als



Figuur 2.7: De posities van b_1 en b_2 op t, waarbij b_1 naar links vaart, en b_2 naar rechts vaart en zich op $x = R_1$ bevindt. De boten zijn elkaar precies in het midden van hun huidige posities gekruist.

b_2 naar links vaart. Op de nieuwe waarde van t bevindt b_2 zich op de rechtergrens van b_1, terwijl b_2 op het begintijdstip nog links van b_1 lag. Volgens Lemma 2.7 zijn de boten elkaar nu gekruist. Om het tijdstip van deze kruising te vinden moet t worden verlaagd. Als b_1 op het zojuist berekende tijdstip t naar links vaart, zijn de boten elkaar gekruist op tijdstip $t - \frac{pos(b_2,t)-pos(b_1,t)}{2}$ (zie Figuur 2.7). Mocht b_1 op t echter naar rechts varen, dan verlagen we t eerst met $pos(b_1, t) - L_1$, zodat b_1 op zijn linkergrens komt te liggen.
Als b_1 volledig wordt overlapt door b_2, dan gaan we in principe op precies dezelfde wijze te werk, op een speciaal geval na. Het kan dan namelijk voorkomen dat b_1 en b_2 samen R_1 bereiken. De boten kruisen elkaar in dit geval al op positie L_1 en zijn vervolgens samen naar R_1 gevaren. Door t te verlagen met $pos(b_1, t) - L_1$ wordt het tijdstip waarop b_1 zijn linkergrens bereikt bepaald. Hier wordt voor gezorgd door de test '**or** $pos(b_1, t) = R_1$' in Algoritme 5. Merk op dat de stap '$t := t - (pos(b_2, t) - pos(b_1, t))/2$' de waarde van t in dit geval onveranderd laat, omdat $pos(b_2, t) = pos(b_1, t)$.
De werking van het algoritme voor een invoer waarbij geldt dat $pos(b_1, t) < pos(b_2, t)$ is analoog. De berekeningstijd van NextJunctionA is constant voor elke invoer, omdat er geen loops voorkomen in het algoritme.

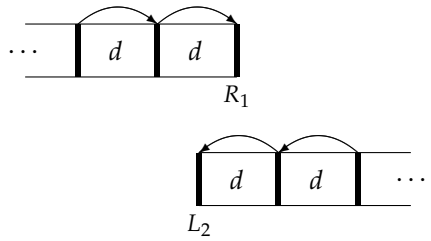
NextJunctionB

Bij het berekenen van het tijdstip van de eerst volgende kruising tussen twee boten, wordt in feite ook gezocht naar de positie van deze kruising. Uit het tijdstip van een kruising kan immers ook de positie van de kruising worden afgeleid met de *pos* functie. Bij een gegeven positie x kan echter ook met een simpele berekening bepaald worden op welke tijdstippen een boot zich op x bevindt. Als twee boten elkaar op slechts één positie kruisen, zoals bijvoorbeeld het geval is bij minimaal overlappende boten, kunnen we daarom de tijdstippen van alle mogelijke kruisingen bepalen. Ook sommige deels overlappende boten kruisen elkaar op maar één positie.

Lemma 2.15. *Zij $b_1 = [L_1, R_1]$ en $b_2 = [L_2, R_2]$ twee deels of minimaal overlappende boten met $L_1 < L_2 \leq R_1 < R_2$. Zij verder $l_1 = R_1 - L_1$ en $l_2 = R_2 - L_2$ de halve periodes van b_1 en b_2. Laat d de grootste gemene deler van l_1 en l_2 zijn, zodat $l_1 = a \cdot d$ en $l_2 = b \cdot d$. Als $d > overlap(b_1, b_2)$, dan kruisen b_1 en b_2 elkaar alleen op positie $x = R_1 - \frac{R_1-L_2}{2} = L_2 + \frac{R_1-L_2}{2}$.*

Merk overigens op dat volgens Stelling 2.13 a even en b oneven moet zijn.

Bewijs. Net als in het bewijs van Stelling 2.13 definiëren we t_i als het laatste door d deelbare tijdstip voor de eerste kruising tussen b_1 en b_2. We stellen ook nu vast dat $pos(b_1, t_i) = L_1 + (a-1) \cdot d = R_1 - d$ en $pos(b_1, t_i) = L_2 + 1 \cdot d$, en dat $pos(b_1, t_{i+1}) = R_1 \geq L_2 = pos(b_2, t_{i+1})$. Volgens Lemma 2.7 zijn de boten elkaar tussen t_i en t_{i+1} gekruist. Dit wordt afgebeeld in Figuur 2.8.
De afstand tussen de boten op t_i is $L_2 + d - (R_1 - d) = 2 \cdot d + L_2 - R_1$. Omdat de boten met een snelheid van één afstandseenheid per seconde naar elkaar toe varen, kruisen de boten elkaar op positie $x = R_1 - d + \frac{2 \cdot d + L_2 - R_1}{2} = R_1 + \frac{L_2 - R1}{2} = R_1 - \frac{R_1 - L_2}{2} = L_2 + \frac{R_1 - L_2}{2}$. Op t_{i+1} keren beide boten om, waardoor op t_{i+2} weer geldt dat $pos(b_1, t_{i+2}) = R_1 - d$ en $pos(b_2, t_{i+2}) = L_2 + d$. De boten zijn elkaar tussen t_{i+1} en t_{i+2} dus nogmaals gekruist. Op t_{i+1} is de afstand tussen de boten $R_1 - L_2$. De boten kruisen elkaar daarom ook deze tweede keer op positie $x = R_1 - \frac{R_1 - L_2}{2} = L_2 + \frac{R_1 - L_2}{2}$.
Bij de eerstvolgende kruising tussen b_1 en b_2 na t_{i+2} bestaat er weer een tijdstip $t_j = j \cdot d$, zodat t_j het laatste door d deelbare tijdstip is voor de kruising. De situatie op t_j is een herhaling van de situatie op t_i, omdat b_1 links van b_2 ligt na t_{i+2}. Alle kruisingen tussen b_1 en b_2 moeten daarom wel plaatsvinden op positie $x = R_1 - \frac{R_1 - L_2}{2} = L_2 + \frac{R_1 - L_2}{2}$. □



Figuur 2.8: b_1 en b_2 kruisen elkaar op $x = R_1 - \frac{R_1 - L_2}{2} = L_2 + \frac{R_1 - L_2}{2}$.

Omdat b_1 en b_2 elkaar op slechts één positie kruisen, kunnen alle tijdstippen waarop de boten elkaar kruisen worden vastgesteld.

Stelling 2.16. *Zij $b_1 = [L_1, R_1]$ en $b_2 = [L_2, R_2]$ twee deels of minimaal overlappende boten met $L_1 < L_2 \leq R_1 < R_2$. Zij verder $l_1 = R_1 - L_1$ en $l_2 = R_2 - L_2$ de halve periodes van b_1 en b_2. Laat d de grootste gemene deler van l_1 en l_2 zijn, zodat $l_1 = a \cdot d$ en $l_2 = b \cdot d$, en laat v het kleinste gemene veelvoud van $2 \cdot l_1$ en $2 \cdot l_2$ zijn. Als $d > overlap(b_1, b_2)$, dan kruisen boten b_1 en b_2 elkaar op een tijdstip t dan en slechts dan als a even en b oneven is en $t = a \cdot b \cdot d + k \cdot v \pm \frac{R_1 - L_2}{2}$ met $k \in \mathbb{N}$.*

Bewijs. $\Rightarrow$ Uit het bewijs van Lemma 2.15 volgt dat b_1 en b_2 elkaar alleen $\frac{R_1 - L_2}{2}$ seconden voor en na een tijdstip t kruisen waarop $pos(b_1, t) = R_1$ en $pos(b_2, t) = L_2$. We moeten dus laten zien dat $t_k = a \cdot b \cdot d + k \cdot v$ voor elk tijdstip t_k waarop $pos(b_1, t_k) = R_1$ en $pos(b_2, t_k) = L_2$.

Om vanaf tijdstip $t = 0$ positie R_1 te bereiken, moet b_1 een oneven aantal keer l_1 afleggen. Boot b_2 moet vanaf $t = 0$ een even aantal keer l_2 afleggen om L_2 te bereiken. Om dit op hetzelfde tijdstip t_k te laten gebeuren hebben we een oneven getal m en even getal n nodig, zodat $t_k = m \cdot l_1 = n \cdot l_2$. Er moet dus gelden dat $m \cdot a \cdot d = n \cdot b \cdot d$, oftewel $\frac{a}{b} = \frac{n}{m}$.

Omdat a en b relatief priem zijn, hebben ze geen gezamenlijke factoren. De kleinste waarden voor m en n waarmee de bovenstaande vergelijking kan worden opgelost zijn daarom $m = b$ en $n = a$. Hieruit volgt dat a even en b oneven is, en dat $t_0 = m \cdot l_1 = b \cdot l_1 = b \cdot a \cdot d = a \cdot b \cdot d + 0 \cdot v$.

Omdat v het kleinste getal is dat deelbaar is door zowel $2 \cdot l_1$ als $2 \cdot l_2$, geldt voor elk tijdstip t, waarbij $pos(b_1, t) = R_1$ en $pos(b_2, t) = L_2$, dat $t = t_k = t_0 + k \cdot v = a \cdot b \cdot d + k \cdot v$.

$\Leftarrow$ Omdat $pos(b_1, 0) < pos(b_2, 0)$ en $pos(b_1, a \cdot b \cdot d) = R_1 \geq L_2 = pos(b_2, a \cdot b \cdot d)$ als a even en b oneven is, zijn de boten elkaar volgens Lemma 2.7 gekruist tussen $t = 0$ en $t = a \cdot b \cdot d$. Uit het bewijs van Lemma 2.15 volgt dat deze kruising heeft plaatsgevonden op $t = a \cdot b \cdot d - \frac{R_1 - L_2}{2}$. Verder volgt uit dit bewijs dat de boten elkaar weer kruisen op $t = a \cdot b \cdot d + \frac{R_1 - L_2}{2}$. Omdat een tijdstap van v seconden de posities van beide boten onveranderd laat, moet voor elk tijdstip $t = a \cdot b \cdot d + k \cdot v \pm \frac{R_1 - L_2}{2}$ met $k \in \mathbb{N}$ wel gelden dat de boten elkaar op dit tijdstip kruisen. □

Als op een tijdstip t geldt dat $pos(b_1, t) > pos(b_2, t)$, dan kan het tijdstip van de eerstvolgende kruising na t tussen twee boten b_1 en b_2, zoals beschreven in Stelling 2.16, gevonden worden met Algoritme 5. Als $l_1 < l_2$ bevinden de boten zich namelijk in de linker situatie van Figuur 2.6, en als $l_1 > l_2$ bevinden de boten zich in de rechter situatie van Figuur 2.6 (waarbij de namen van de boten zijn verwisseld).

Als op t geldt dat $pos(b_1, t) < pos(b_2, t)$ kan het tijdstip van de eerstvolgende kruising na t in logaritmische tijd berekend worden met behulp van Stelling 2.16. Omdat $pos(b_1, t) < pos(b_2, t)$, kruisen de boten elkaar op een tijdstip $a \cdot b \cdot d + k \cdot v - \frac{R_1 - L_2}{2}$. We hoeven nu alleen de kleinst mogelijke waarde van k te vinden, waarvoor geldt dat $t < a \cdot b \cdot d + k \cdot v$. Dit kan door $t - a \cdot b \cdot d$ door v te delen en het resultaat hiervan naar boven af te ronden.

Algoritme 6 NextJunctionB(b_1,b_2,t)

preconditie: $pos(b_1, t) < pos(b_2, t)$, $ggd(l_1, l_2) > overlap(b_1, b_2)$, $L_1 < L_2 \leq R_1 < R_2$ en de boten zullen elkaar ooit kruisen.

$l_1 := R_1 - L_1$
$l_2 := R_2 - L_2$
$d := ggd(l_1, l_2)$
$v := kgv(2 \cdot l_1, 2 \cdot l_2)$
$a := l_1 / d$
$b := l_2 / d$
$k := \lceil (t - a \cdot b \cdot d)/v \rceil$
$t := a \cdot b \cdot d + k \cdot v$
return $t - (R_1 - L_2)/2$

Algoritme 6 (NextJunctionB) maakt gebruik van de zojuist beschreven berekening om het tijdstip van de

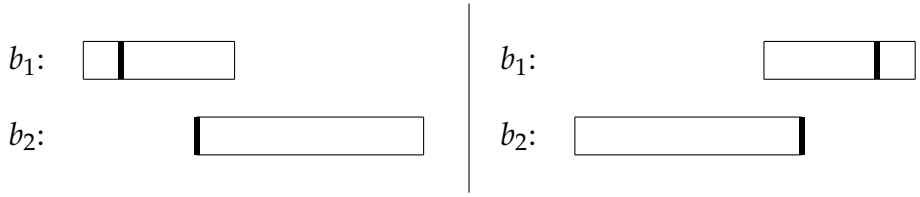
eerstvolgende kruising tussen twee boten te bepalen. Het algoritme heeft twee boten $b_1 = [L_1, R_1]$ en $b_2 = [L_2, R_2]$ met halve periodes l_1 en l_2, en een begintijdstip t als invoer, en neemt het volgende aan: $pos(b_1, t) < pos(b_2, t)$, $ggd(l_1, l_2) > overlap(b_1, b_2)$, $L_1 < L_2 \leq R_1 < R_2$, en b_1 en b_2 zullen elkaar ooit kruisen. De uitvoer van het algoritme is het eerste tijdstip na t waarop b_1 en b_2 elkaar kruisen.

NextJunctionC

Voor alle instanties waarvoor het tijdstip van de eerstvolgende kruising niet berekend kan worden met Algoritme 5 of Algoritme 6, hebben we een ander algoritme bedacht dat dit wel kan. Dit algoritme maakt gebruik van een techniek waarbij tijdstappen ter grootte van de grootste periode worden genomen. We noemen deze techniek daarom *period skipping*. Voordat het gevonden algoritme wordt geïntroduceerd, bespreken we eerst deze techniek.

Zij $b_1 = [L_1, R_1]$ en $b_2 = [L_2, R_2]$ twee deels overlappende boten met periodes p_1 en p_2, zodat $p_1 < p_2$. Omdat de periode van b_1 kleiner is dan de periode van b_2, zal b_1 in p_2 seconden zijn eigen periode afleggen plus een extra $p_2 - p_1$ afstandseenheden. Bij een tijdstap van p_2 seconden zal b_1 daarom $(p_2 - p_1) \mod p_1$ afstandseenheden verder in zijn periode worden geschoven. Boot b_2 blijft bij zulke tijdstappen echter stilliggen, omdat de boot precies zijn volledige periode aflegt.

Zij t_1 een tijdstip waarop b_1 in het niet-overlappende deel van zijn bereik ligt en waarop b_2 zich op zijn overlappende grens bevindt, zoals afgebeeld in Figuur 2.9. Door tijdstappen van p_2 seconden te nemen kunnen we b_1 verder in zijn periode schuiven, totdat de boot op een tijdstip t_2 in het overlappende deel van zijn bereik komt te liggen. De boten zijn elkaar dan volgens Lemma 2.7 gekruist. Met behulp van de nu volgende stelling kan worden aangetoond dat deze kruising de enige kruising is tussen t_1 en t_2.



Figuur 2.9: De beginsituaties waarvoor period skipping kan worden toegepast.

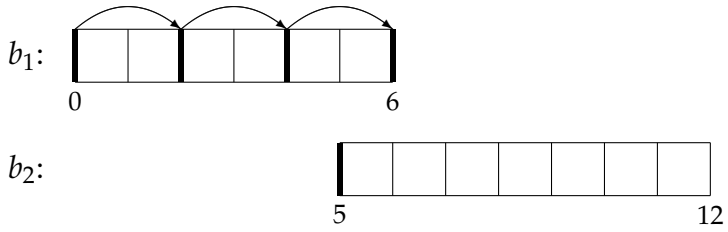
Stelling 2.17. *Zij b_1 en b_2 twee deels overlappende boten met periodes p_1 en p_2, zodat $p_1 < p_2$. Zij verder t een tijdstip waarop b_1 in het niet-overlappende deel van zijn bereik ligt en waar b_2 zich op zijn overlappende grens bevindt. Als b_1 zich op tijdstip $t + p_2$ in het overlappende deel van zijn bereik bevindt, dan zijn de boten elkaar precies één keer gekruist na t. Als b_1 op tijdstip $t + p_2$ in het niet-overlappende deel van zijn bereik ligt, dan zijn de boten elkaar niet gekruist na t.*

Bewijs. Tussen t en $t + \frac{p_2}{2}$ vaart b_2 naar zijn niet-overlappende grens. Om b_1 te kunnen kruisen moet b_2 echter

in de andere richting varen. De boten kunnen elkaar daarom niet kruisen in dit tijdsbestek. Van $t + \frac{p_2}{2}$ tot $t + p_2$ vaart b_2 terug naar zijn overlappende grens. De boot keert pas om op het tijdstip $t + p_2$, waardoor b_1 en b_2 elkaar maximaal één keer kunnen kruisen.
Als b_1 op tijdstip $t + p_2$ in het overlappende deel van zijn bereik ligt, geldt $pos(b_1, t) < pos(b_2, t)$ en $pos(b_1, t + p_2) \geq pos(b_2, t + p_2)$. Lemma 2.7 stelt nu dat de boten elkaar gekruist moeten zijn. Als b_1 op tijdstip $t + p_2$ in het niet-overlappende deel van zijn bereikt ligt, geldt $pos(b_1, t) < pos(b_2, t)$ en $pos(b_1, t + p_2) < pos(b_2, t + p_2)$. Omdat de boten elkaar maximaal één keer kunnen kruisen, impliceert dit dat de boten elkaar niet gekruist zijn. □

Omdat b_1 pas op tijdstip t_2 in het overlappende deel van zijn periode komt te liggen, volgt uit Stelling 2.17 dat er maar één kruising mogelijk is tussen t_1 en t_2. Period skipping kan dus gebruikt worden om een tijdstip te vinden waarop twee boten elkaar precies één keer gekruist zijn. Vanaf dit tijdstip kan vervolgens het tijdstip van de werkelijke kruising worden bepaald.

Voorbeeld 2.18. Zij $b_1 = [0, 6]$ en $b_2 = [5, 12]$ twee boten. Bij tijdstappen ter grootte van $p_2 = 2 \cdot (12 - 5) = 14$ seconden zal b_2 stil blijven liggen, terwijl b_1 steeds $(p_2 - p_1) \mod p_1 = (14 - 12) \mod 12 = 2$ afstandseenheden verder raakt in zijn periode. Deze situatie wordt voor een begintijdstip $t = 0$ geïllustreerd in Figuur 2.10. Hierin is te zien dat b_1 zich na drie tijdstappen van p_2 seconden in het overlappende deel van zijn periode bevindt. De boten zijn elkaar daarom na $3 \cdot p_2 = 3 \cdot 14 = 42$ seconden precies één keer gekruist. △



Figuur 2.10: Boot b_1 wordt elke p_2 seconden twee afstandseenheden verder geschoven.

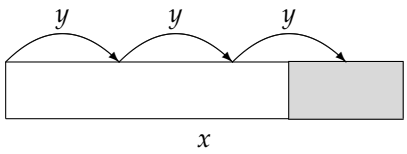
In het vorige voorbeeld werd het aantal tijdstappen tot de kruising van b_1 en b_2 handmatig geteld. We kunnen dit aantal echter ook bepalen met een simpele berekening. Hiervoor moet eerst de afstand x tussen de huidige positie van b_1 en het einde van het overlappende deel van zijn periode berekend worden. Dit is het aantal seconden dat b_1 nodig heeft om zijn overlappende grens te bereiken plus de overlap van b_1 en b_2. Ook het aantal afstandseenheden y dat b_1 wordt verder geschoven in zijn periode bij een tijdstap van p_2 seconden moet berekend worden. Voor een voorbeeld van de berekeningen van x en y, zie Voorbeeld 2.19.
De rest van de deling $\frac{x}{y}$ is het aantal seconden dat b_1 nodig heeft om het einde van het overlappende deel van zijn periode te bereiken, nadat de boot $\lfloor \frac{x}{y} \rfloor$ keer y afstandseenheden is doorgeschoven. Indien nu geldt

dat $x \mod y \leq 2 \cdot overlap(b_1, b_2)$, dan zal b_1 op dit moment in het overlappende deel van zijn periode liggen. Immers, b_1 heeft nog $x \mod y$ seconden nodig om het einde van het overlappende deel van zijn periode te bereiken, en zou $2 \cdot overlap(b_1, b_2)$ seconden nodig hebben om dit overlappende deel volledig door te varen. Een tijdstip waarop de boten elkaar precies één keer zijn gekruist, kan dan zijn $t = \lfloor \frac{x}{y} \rfloor \cdot p_2$.

Voorbeeld 2.19. Zij $b_1 = [0, 6]$ en $b_2 = [5, 12]$ twee boten, net zoals in Voorbeeld 2.18. Om een tijdstip te vinden waarop de boten elkaar precies één keer zijn gekruist na $t = 0$, berekenen we eerst de waarden van x en y:

$$x = R_1 - pos(b_1, 0) + overlap(b_1, b_2) = 6 + 1 = 7$$
$$y = (p_2 - p_1) \mod p_1 = (14 - 12) \mod 12 = 2$$

Nu geldt dat $7 \mod 2 = 1 \leq 2 \cdot 1$, en dus zijn de boten op tijdstip $t = \lfloor \frac{x}{y} \rfloor \cdot p_2 = \lfloor \frac{7}{2} \rfloor \cdot 14 = 3 \cdot 14 = 42$ elkaar precies één keer gekruist. Deze berekening wordt geïllustreerd in Figuur 2.11. △

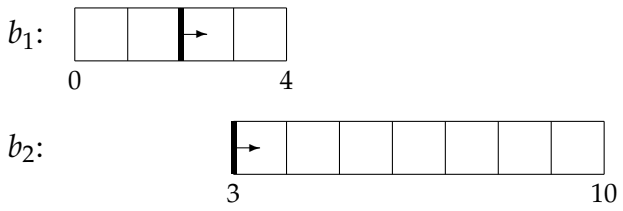


Figuur 2.11: Na $\lfloor \frac{x}{y} \rfloor = 3$ keer y afstandseenheden te zijn doorgeschoven, ligt b_1 in het (in grijs aangegeven) overlappende deel van zijn periode.

Het kan echter zo zijn dat $y > 2 \cdot overlap(b_1, b_2)$, waardoor het mogelijk is dat $x \mod y \not\leq 2 \cdot overlap(b_1, b_2)$. Als dit het geval is, dan kruisen de boten elkaar niet in $\lfloor \frac{x}{y} \rfloor \cdot p_2$ seconden. Boot b_1 wordt namelijk over het overlappende deel van zijn periode heen geschoven. Een tijdstip waarop de boten elkaar precies één keer zijn gekruist kan nu gevonden worden door x te verhogen met p_1 en nogmaals $x \mod y \leq 2 \cdot overlap(b_1, b_2)$ te testen. Door x te verhogen met p_1 kan gecontroleerd worden of b_1 wel in het overlappende deel van zijn periode belandt, als de boot zijn volledige periode is rond geschoven en nogmaals langs het overlappende deel van zijn periode komt. Als de boten elkaar weer niet kruisen, wordt x wederom opgehoogd met p_1. De waarde van x zal op deze wijze worden opgehoogd totdat de boten elkaar kruisen. Dit proces stopt altijd vanwege onze aanname dat elk paar boten elkaar ooit zal kruisen.

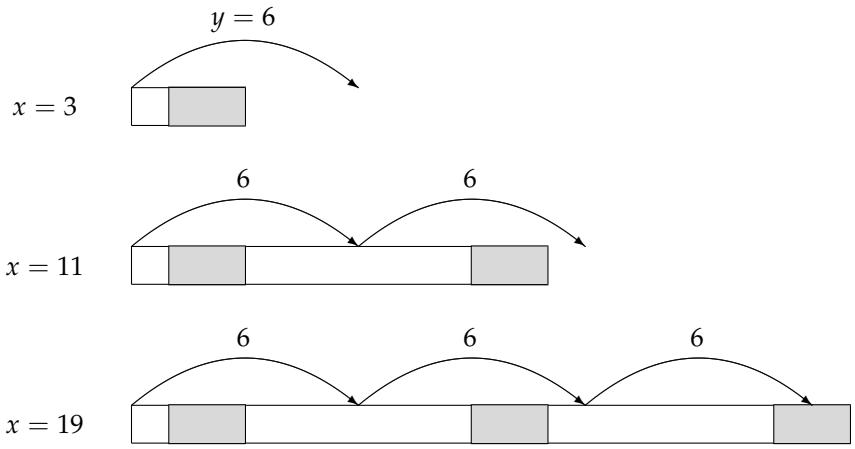
Voorbeeld 2.20. Zij $b_1 = [0, 4]$ en $b_2 = [3, 10]$ twee boten en $t = 42$ een begintijdstip, zoals afgebeeld in Figuur 2.12. Om een tijdstip te vinden waarop de boten elkaar precies één keer zijn gekruist, berekenen we eerst de waarden van x en y:

$$x = R_1 - pos(b_1, t) + overlap(b_1, b_2) = 4 - 2 + 1 = 3$$
$$y = (p_2 - p_1) \mod p_1 = (14 - 8) \mod 8 = 6$$



Figuur 2.12: De posities van $b_1 = [0,4]$ en $b_2 = [3,10]$ op tijdstip $t = 42$.

Uit $x \mod y = 3 \mod 6 = 3 \not\leq 2 \cdot 1$ blijkt dat b_1 over het overlappende deel van zijn periode heen wordt geschoven. De waarde van x wordt daarom verhoogd met $p_1 = 8$. Boot b_1 wordt nu echter alweer over het overlappende deel van zijn periode geschoven, aangezien $x \mod y = 11 \mod 6 = 5 \not\leq 2$. De waarde van x wordt nogmaals met 8 verhoogd, waarna de boten elkaar wel kruisen: $x \mod y = 19 \mod 6 = 1 \leq 2$. Dit proces is afgebeeld in Figuur 2.13. △



Figuur 2.13: Na x tweemaal met $p_1 = 8$ te verhogen komt b_1 wel in het (in grijs aangegeven) overlappende deel van zijn periode.

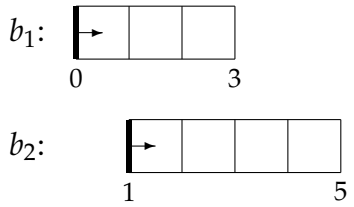
Als $y \leq 2 \cdot overlap(b_1, b_2)$ kan zich een ander probleem voordoen. De test $x \mod y \leq 2 \cdot overlap(b_1, b_2)$ controleert of b_1 na $\lfloor \frac{x}{y} \rfloor$ keer te zijn doorgeschoven in het overlappende deel van zijn bereik ligt. Als $y \leq 2 \cdot overlap(b_1, b_2)$ kan het echter zo zijn dat b_1 ook al in het overlappende deel van zijn bereik lag na minder dan $\lfloor \frac{x}{y} \rfloor$ keer te zijn doorgeschoven. Hierdoor is het mogelijk dat niet een tijdstip na precies één kruising gevonden wordt, maar een tijdstip na meerdere kruisingen.

Voorbeeld 2.21. Zij $b_1 = [0,3]$ en $b_2 = [1,5]$ twee boten en $t = 0$ een begintijdstip, zoals afgebeeld in Figuur 2.14. Om een tijdstip te vinden waarop de boten elkaar precies één keer zijn gekruist, berekenen we eerst de waarden van x en y:

$$x = R_1 - pos(b_1, t) + overlap(b_1, b_2) = 3 - 0 + 2 = 5$$

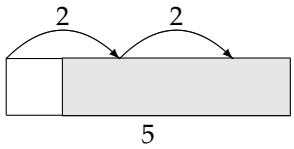
$$y = (p_2 - p_1) \mod p_1 = (8 - 6) \mod 6 = 2$$

Uit $x \mod y = 5 \mod 2 = 1 \leq 4$ blijkt dat de boten op het tijdstip $t = \lfloor \frac{x}{y} \rfloor \cdot p_2 = \lfloor \frac{5}{2} \rfloor \cdot 8 = 2 \cdot 8 = 16$ elkaar precies één keer zijn gekruist sinds $t = 0$. Dit klopt echter niet, want de boten zijn elkaar dan al drie keer gekruist; namelijk op $t = 7\frac{1}{2}$, $t = 9\frac{1}{2}$ en $t = 14\frac{1}{2}$. Omdat $y \leq 2 \cdot overlap(b_1, b_2)$ kan b_1 te vaak worden doorgeschoven, zoals geïllustreerd in Figuur 2.15, waardoor het berekende tijdstip incorrect is. △



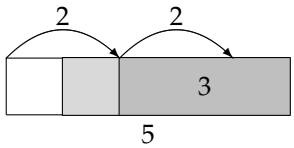
Figuur 2.14: De posities van $b_1 = [0, 3]$ en $b_2 = [1, 5]$ op tijdstip $t = 0$.

Het probleem uit Voorbeeld 2.21 kan worden opgelost door $2 \cdot overlap(b_1, b_2) - (y - 1)$ van x af te trekken, als $y \leq 2 \cdot overlap(b_1, b_2)$. Het laatste overlappende deel van x wordt hierdoor $y - 1$ afstandseenheden lang. Omdat b_1 telkens y afstandseenheden verder in zijn periode komt, kan de boot nu niet meer te ver worden doorgeschoven.



Figuur 2.15: Boot b_1 wordt één keer te ver doorgeschoven.

Voorbeeld 2.22. Zij $b_1 = [0, 3]$ en $b_2 = [1, 5]$ twee boten en $t = 0$ een begintijdstip, net zoals in Voorbeeld 2.21. We weten dat $x = 5$ en $x \mod y \leq 2 \cdot overlap(b_1, b_2)$. We verlagen x nu met $2 \cdot overlap(b_1, b_2) - (y - 1) = 2 \cdot 2 - (2 - 1) = 3$, zoals te zien in Figuur 2.16. Een tijdstip waarop de boten elkaar precies één keer kruisen is nu $t = \lfloor \frac{x}{y} \rfloor \cdot p_2 = \lfloor \frac{2}{2} \rfloor \cdot 8 = 1 \cdot 8 = 8$. △



Figuur 2.16: Het donkergrijs gekleurde deel wordt van x afgetrokken, waardoor het overlappende deel van x nog maar $y - 1 = 1$ lang is.

Het algoritme dat het tijdstip van de eerstvolgende kruising tussen twee boten berekent met behulp van period skipping, noemen we NextJunctionC. De invoer van dit algoritme bestaat uit twee boten b_1 en b_2 en een begintijdstip t, waarbij wordt aangenomen dat $pos(b_1, t) < pos(b_2, t)$, dat de invoer geen geldige instantie van $NextJunctionA$ betreft, en dat de boten elkaar ooit kruisen. De invoer mag overigens wel een geldige invoer van $NextJunctionB$ zijn. De uitvoer van het algoritme is het eerste tijdstip sinds t waarop b_1 en b_2 elkaar kruisen.

Algoritme 7 NextJunctionC(b_1,b_2,t)

preconditie: $pos(b_1,t) < pos(b_2,t)$, b_1 en b_2 vormen geen geldige instantie van NextJunctionA, b_1 en b_2 kruisen elkaar ooit en $p_1 < p_2$.
$l_1 := R_1 - L_1$
$l_2 := R_2 - L_2$
// Stap 1: bereken het tijdstip waarop b_2 zijn overlappende grens bereikt.
if $pos(b_2,t) \neq L_2$ **then**
  if $pos(b_2,t) > pos(b_2,t+\frac{1}{2})$ **then** // b_2 vaart naar links.
    $t := t + pos(b_2,t) - L_2$
  else
    $t := t + l_2 + R_2 - pos(b_2,t)$
// Stap 2: bereken een tijdstip waarop de boten elkaar precies één keer zijn gekruist na t.
if $pos(b_1,t) < pos(b_2,t)$ **then**
  if $pos(b_1,t) < pos(b_1,t+\frac{1}{2})$ **then** // b_1 vaart naar rechts.
    $x := overlap(b_1,b_2) + R_1 - pos(b_1,t)$
  else
    $x := overlap(b_1,b_2) + l_1 + pos(b_1,t) - L_1$
  $y := (2 \cdot (l_2 - l_1)) \mod (2 \cdot l_1)$
  if $y \leq 2 \cdot overlap(b_1,b_2)$ **then**
    $x := x - (2 \cdot overlap(b_1,b_2) - (y-1))$
  while $x \mod y > 2 \cdot overlap(b_1,b_2)$ **do**
    $x := x + 2 \cdot l_1$
  $t := t + \lfloor x/y \rfloor \cdot 2 \cdot l_2$
// Stap 3: bereken het tijdstip van de kruising.
if $pos(b_1,t) > pos(b_1,t+\frac{1}{2})$ **then** // b_1 vaart naar links.
  $t := t - (R_1 - pos(b_1,t))$
$t := t - (pos(b_1,t) - pos(b_2,t))/2$
return t

Zij p_1 en p_2 de periodes van b_1 en b_2. NextJunctionC onderscheidt twee soorten invoerinstanties: instanties waarbij $p_1 < p_2$ en instanties waarbij $p_1 > p_2$. In Algoritme 7 staat de pseudocode van NextJunctionC weergegeven voor instanties waarbij $p_1 < p_2$. De werking van NextJunctionC voor instanties waarbij $p_1 > p_2$ is analoog.

Algoritme 7 is verdeeld in drie stappen. In stap 1 wordt het tijdstip berekend waarop b_2 zijn overlappende grens bereikt. We kunnen daarna namelijk period skipping toepassen. Dit tijdstip is $t + pos(b_2,t) - L_2$ als b_2 naar links vaart, en $t + l_2 + R_2 - pos(b_2,t)$ als b_2 naar rechts vaart. Als $pos(b_1,t) \geq pos(b_2,t)$ op de nieuwe waarde van t, dan zijn de boten elkaar precies één keer gekruist. De reden hiervoor wordt gegeven in het bewijs van Stelling 2.17. In dit geval wordt stap 2 overgeslagen.
Als $pos(b_1,t) < pos(b_2,t)$ op de nieuwe waarde van t, dan bevinden de boten zich in de linker situatie van Figuur 2.9. Stap 2 past period skipping toe op deze instanties. Hiervoor worden eerst de waarden van x en y berekend. Als $y \leq 2 \cdot overlap(b_1,b_2)$, dan wordt $2 \cdot overlap(b_1,b_2) - (y-1)$ van x afgetrokken om te voorkomen dat b_1 te ver wordt doorgeschoven. Hierna volgt een while-loop waarin x wordt opgehoogd met p_1, totdat $x \mod y \leq 2 \cdot overlap(b_1,b_2)$. Een tijdstip waarop de boten elkaar precies één keer zijn gekruist sinds het begintijdstip is nu $t + \lfloor \frac{x}{y} \rfloor \cdot 2 \cdot l_2$.
Stap 3 bepaalt het tijdstip van de kruising door de nieuwe waarde van t te verlagen. Als b_1 op t naar links

vaart, dan wordt t verlaagd zodat de boot op zijn rechtergrens komt te liggen. Nu geldt dat b_1 op t naar rechts vaart of zich op zijn overlappende grens R_1 bevindt, en dat b_2 op t naar links vaart of zich op zijn overlappende grens L_2 bevindt. De positie van de kruising ligt nu precies in het midden van de boten. Het tijdstip van de kruising is daarom $t - \frac{pos(b_1,t)-pos(b_2,t)}{2}$.

Het aantal keer dat de waarde van x wordt opgehoogd is een goede maat om de complexiteit van NextJunctionC te meten. Dit is immers de enige stap uit het algoritme die zich in een loop bevindt. In de best case wordt stap 2 overgeslagen of wordt x niet opgehoogd. Het tijdstip van de eerstvolgende kruising kan dan in constante tijd berekend wordt. Een voorbeeld van een best case waarbij stap 2 word overgeslagen is: $b_1 = [0, 2]$, $b_2 = [1, 4]$ en $t = 1$. Na stap 2 geldt namelijk dat $t = 6$, en dat $pos(b_1, t) = 2 > 1 = pos(b_2, t)$.
In de worst case wordt x zo vaak mogelijk opgehoogd. Stel nu dat $l_1 < l_2$. Steeds als boot b_1 over het overlappende deel van zijn periode heen wordt geschoven, zal x met $2 \cdot l_1$ worden opgehoogd. Na een ophoging van x kan b_1 minstens één keer extra worden opgeschoven, omdat $y = (p_2 - p_1) \mod p_1 < 2 \cdot l_1$. Het aantal keer dat x wordt opgehoogd is daarom hoogstens het aantal keer dat b_1 wordt opgeschoven. Bij het opschuiven van b_1 wordt een tijdstap van $2 \cdot l_2$ seconden gemaakt. Omdat de gezamenlijke periode van b_1 en b_2 altijd hooguit $2 \cdot l_1 \cdot l_2$ seconden duurt, kan b_1 maximaal $\frac{2 \cdot l_1 \cdot l_2}{2 \cdot l_2} = l_1$ keer worden opgeschoven totdat de beginsituatie zich herhaalt. Een bovengrens voor de worst case complexiteit is daarom $O(l_1)$ als $l_1 < l_2$, of $O(\min(l_1, l_2))$ in het algemeen.
Deze bovengrens kan ook gehaald worden. Zij $b_1 = [0, l_1]$ en $b_2 = [l_1 - 1, 3 \cdot l_1 - 2]$ twee boten, waarbij l_1 even is. Als begintijdstip kiezen we een tijdstip vlak na een kruising tussen b_1 en b_2. We bepalen allereerst een tijdstip t waarop $pos(b_1, t) = l_1$ en $pos(b_2, t) = l_1 - 1$. Op t heeft b_1 een oneven aantal keer zijn halve periode l_1 afgelegd, en heeft b_2 een even aantal keer zijn halve periode $l_2 = 3 \cdot l_1 - 2 - (l_1 - 1) = 2 \cdot l_1 - 1$ afgelegd. Dit leidt tot de volgende vergelijking: $(2a - 1) \cdot l_1 = 2b \cdot (2 \cdot l_1 - 1)$, met $a, b \in \mathbb{N}^+$. Deze vergelijking kan worden opgelost door $a = l_1$ en $b = \frac{l_1}{2}$ te nemen, zodat $t = l_1 \cdot (l_1 - 1)$. Omdat de boten op t allebei op hun overlappende grens liggen, en omdat de afstand tussen deze twee grenzen slechts één afstandseenheid is, kruisen de boten elkaar op $t - \frac{1}{2}$ en $t + \frac{1}{2}$. We kiezen daarom $t + 1$ als begintijdstip.
Op het begintijdstip t geldt dat $pos(b_1, t) = l_1 - 1$ en $pos(b_2, t) = l_1$, waarbij b_1 naar links en b_2 naar rechts vaart. Bij het uitvoeren van de tweede stap van het algoritme wordt t verhoogd met $l_2 + R_2 - pos(b_2, t) = 2 \cdot l_1 - 1 + 3 \cdot l_1 - 2 - l_1 = 4 \cdot l_1 - 3$, zodat b_2 op zijn linkergrens komt te liggen. Na stap 2 ligt b_1 op positie $l_1 - 2$, waarbij de boot naar rechts vaart. De beginwaarde van x is daarom $l_1 - (l_1 - 2) + 1 = 2 + 1 = 3$. De waarde van y is $(2 \cdot (2 \cdot l_1 - 1) - 2 \cdot l_1) \mod 2 \cdot l_1 = 2 \cdot l_1 - 2$. Bij een period skip wordt b_1 dus $2 \cdot l_1 - 2$ posities verder in zijn periode geschoven, wat hetzelfde is als b_1 twee posities terugzetten in zijn periode. Merk op dat b_1 zich nu altijd op een even positie zal bevinden bij period skipping. De enige even positie in het overlappende deel van het bereik van b_1 is $x = l_1$. Om b_1 in het overlappende deel van zijn periode te schuiven, moet b_1 eerst van $x = l_1 - 2$ naar $x = 0$ worden geschoven en vervolgens van $x = 0$ naar $x = l_1$. Hiervoor zijn in totaal $\frac{l_1 - 2}{2} + \frac{l_1}{2} = l_1 - 1$ period skips nodig.

Boot b_1 legt bij het period skippen $(l_1 - 1) \cdot y = (l_1 - 1) \cdot (2l_1 - 2)$ afstandseenheden af. Op positie l_1 bevindt b_1 zich één seconde van het einde van het overlappende deel van zijn periode. De waarde van x moet dus in totaal $(l_1 - 1) \cdot (2 \cdot l_1 - 2) + 1$ worden. Bij elke iteratie van de while-loop wordt x met $2 \cdot l_1$ opgehoogd. Vanaf zijn startwaarde 3 moet x daarom $\frac{(l_1-1)\cdot(2\cdot l_1-2)+1-3}{2\cdot l_1} = \frac{2\cdot l_1^2-4\cdot l_1+2-2}{2\cdot l_1} = l_1 - 2$ keer worden opgehoogd. Dit komt overeen met de vastgestelde bovengrens $O(\min(l_1, l_2))$ van de worst case complexiteit. Een voorbeeld van een worst case is: $b_1 = [0, 166]$, $b_2 = [165, 496]$ en $t = 54947$. De waarde van x wordt voor deze instantie 164 keer opgehoogd.

Algoritmes 5, 6 en 7 vormen samen Algoritme 8 (NextJunction). NextJunction krijgt twee boten b_1 en b_2, en een tijdstip t als invoer. NextJunction kiest voor elk paar boten het geschikte algoritme om het tijdstip van de eerstvolgende kruising tussen b_1 en b_2 na t te berekenen. Hierbij krijgt NextJunctionA voorrang op NextJunctionB, en krijgt NextJunctionB voorrang op NextJunctionC. Dit omdat de rekentijd van NextJunctionA constant is, wat sneller is dan de logaritmische rekentijd van NextJunctionB, wat weer sneller is dan de lineaire rekentijd van NextJunctionC.

Algoritme 8 NextJunction(b_1,b_2,t)

preconditie: $pos(b_1, t) < pos(b_2, t)$ en b_1 en b_2 kruisen elkaar ooit
  $l_1 := R_1 - L_1$
  $l_2 := R_2 - L_2$
  if $l_1 < l_2$ **and** $(overlap(b_1, b_2) = l_1$ **or** $R_1 > R_2)$ **then**
    return $NextJunctionA(b_1, b_2, t)$
  else if $overlap(b_1, b_2) = l_2$ **or** $L_1 > L_2$ **then**
    return $NextJunctionA(b_2, b_1, t)$
  else if $ggd(l_1, l_2) > overlap(b_1, b_2)$ **then**
    return $NextJunctionB(b_1, b_2, t)$
  else
    return $NextJunctionC(b_1, b_2, t)$

De worst case complexiteit van NextJunction is gelijk aan die van NextJunctionC $O(\min(l_1, l_2))$, en is daarmee iets beter dan die van QuickSim $O(l_1 + l_2)$. De average case complexiteit is voor beide algoritmen echter moeilijk te bepalen. Om toch te testen welk algoritme gemiddeld het snelst is, is een experiment uitgevoerd. In dit experiment moesten beide algoritmen voor alle mogelijke paren boten de tijdstippen van alle mogelijke kruisingen in het tijdsinterval $[0, v)$ berekenen (met v het kleinste gemene veelvoud van de periodes van de boten). Hierbij werden alleen paren gebruikt waarbij de boten elkaar ook daadwerkelijk kruisen, omdat dit een aanname van NextJunction is. De pseudocode van dit experiment is voor NextJunction te zien in Algoritme 9.

Het resultaat van het experiment is dat QuickSim 85 minuten nodig had om alle kruisingen te berekenen, terwijl NextJunction dit in 59 minuten kon. NextJunction is gemiddeld daarom ongeveer 30% sneller dan QuickSim. Hiermee is NextJunction een kleine verbetering op QuickSim.

Algoritme 9 NextJunction Snelheidstest

$L_1 := 0$
for $R_1 = 1$ **to** W **do**
  for $L_2 = 1$ **to** R_1 **do**
    for $R_2 = L_2 + 1$ **to** W **do**
      if $JunctionPossible(b_1, b_2) =$ **false then**
        continue
      $t := 0$
      $v := (4 \cdot R_1 \cdot (R_2 - L_2))/ggd(2 \cdot R_1, 2 \cdot (R_2 - L_2))$
      while $t < v$ **do**
        while $pos(b_1, t) = pos(b_2, t)$ **do**
          $t := t + \frac{1}{2}$
        if $t \geq v$ **then**
          break
        if $pos(b_1, t) < pos(b_2, t)$ **then**
          $NextJunction(b_1, b_2, t)$
        else
          $NextJunction(b_2, b_1, t)$
return

Hoofdstuk 3

Het Simulatie Algoritme

In dit hoofdstuk bespreken we het tweede algoritme dat voor dit onderzoek is geanalyseerd. Ook dit algoritme is bedacht door één van de juryleden van BAPC 2015. Het algoritme berekent het eerste tijdstip waarop Mario de overkant van de rivier kan bereiken door de koers van elke boot gelijktijdig te simuleren. Hierbij houdt het algoritme een lijst van boten waar Mario zich op kan bevinden bij, waardoor op elk tijdstip alle mogelijke posities van Mario bekend zijn. Zodra één van deze mogelijke posities $x = W$ is, stopt het algoritme.
In de eerste paragraaf van dit hoofdstuk wordt de werking van dit simulatie algoritme in meer detail besproken, waarna in de tweede paragraaf wordt ingegaan op de complexiteit.

3.1 Werking algoritme

De invoer van het simulatie algoritme bestaat uit de rivierbreedte W, het aantal boten N en twee arrays L en R met de linker- en rechtergrenzen van de boten. Het algoritme doet hierbij de aanname dat $0 \leq L[i] < R[i] \leq W$ geldt voor elke boot b_i. De uitvoer van het algoritme is het eerste tijdstip waarop Mario positie $x = W$ kan bereiken. Als dit niet mogelijk is, geeft het algoritme -1 terug.
Om de toekomstige posities van de boten te simuleren, houdt het algoritme vijf arrays bij:

1. *pos*, de positie van elke boot.
2. *dir*, de richting van elke boot. De waarde van $dir[i]$ is $\frac{1}{2}$ als b_i naar rechts vaart en $-\frac{1}{2}$ als b_i naar links vaart.
3. *reached*, een enkelverbonden lijst met boten die Mario tot nu toe bereikt heeft.
4. *notReached*, een enkelverbonden lijst met boten die Mario tot nu toe nog niet bereikt heeft.

5. *timeAt*, voor elke positie het laatste tijdstip waarop Mario zich op deze positie bevond.

De twee lijsten *reached* en *notReached* zijn dus geïmplementeerd met arrays. Voordat het algoritme begint met de simulatie moeten alle arrays geïnitialiseerd worden voor het begintijdstip $t = 0$. Voor elke boot b_i geldt dat $pos[i] = L[i]$ en $dir[i] = \frac{1}{2}$ op het begintijdstip, omdat alle boten starten op hun linkergrens en daarbij naar rechts varen. Mario start zelf op positie $x = 0$. Alle boten b_i waarvoor geldt dat $L[i] = 0$ worden derhalve aan de lijst *reached* toegevoegd. De overige boten worden in *notReached* geplaatst. De waarde van *timeAt* wordt voor alle posities geïnitialiseerd op -1, om aan te geven dat Mario deze posities nog niet bereikt heeft. Als er een boot b_i bestaat met $L[i] = 0$, dan krijgt $timeAt[0]$ de waarde $t = 0$.

De simulatie van de toekomstige posities van de boten vindt plaats in een while-loop, zoals weergegeven in Algoritme 10. De variabele t staat voor het huidige tijdstip en wordt elke iteratie met een halve seconde opgehoogd. Het algoritme stapt pas uit de while-loop als Mario de overkant heeft bereikt, of als $t \geq 10 \cdot W^2$. Deze bovengrens van t is een willekeurig gekozen groot getal. In de volgende paragraaf zullen we proberen hier een betere waarde voor te bepalen.
Binnen de while-loop bevinden zich twee andere while-loops. In de eerste while-loop worden de volgende posities van alle boten uit *reached* berekend. Omdat t elke iteratie met een halve seconde wordt opgehoogd, is de volgende positie van elke boot b_i gelijk aan $pos[i] + dir[i]$. Het laatste tijdstip waarop Mario deze nieuwe waarde van $pos[i]$ kan bereiken is nu t, en dus geldt $timeAt[pos[i]] = t$. Mocht de nieuwe positie van b_i één van zijn grenzen zijn, dan wordt de richting van de boot aangepast.
Nadat alle boten uit *reached* zijn gesimuleerd, worden in de tweede while-loop de volgende posities van alle boten uit *notReached* berekend. Als $timeAt[pos[i]] = t$, voor een zojuist berekende positie $pos[i]$, dan kan Mario overstappen op boot b_i. Zowel b_i als één van de boten uit *reached* bevinden zich immers op positie $pos[i]$ op het huidige tijdstip. Boot b_i wordt nu uit *notReached* verwijderd en in *reached* geplaatst. Nadat ook alle boten uit *notReached* zijn gesimuleerd, eindigt de iteratie van de buitenste while-loop.
Zodra een boot uit *reached* positie $x = W$ bereikt, stopt het algoritme. Het algoritme geeft in dit geval $timeAt[W]$ terug. Als t zijn bovengrens bereikt, gaat het algoritme ervan uit dat het niet mogelijk is om de overkant te bereiken. Dit is echter slechts een aanname. Mogelijk bestaat er dus wel een instantie waarin Mario meer dan $10 \cdot W^2$ seconden nodig heeft om de overkant te bereiken. We kunnen daarom nog niet met zekerheid zeggen dat het simulatie algoritme voor elke invoer correct werkt.

3.2 Complexiteit algoritme

De basisoperatie van het simulatie algoritme is het berekenen van de toekomstige positie van een boot. De complexiteit van het algoritme is $\Theta(N \cdot T)$, waarbij T de waarde van t is op het moment dat de while-loop wordt verlaten. Het algoritme moet immers $2 \cdot T$ posities bepalen voor N verschillende boten. Als Mario de

Algoritme 10 simulate(W,N,L,R)

```
while timeAt[W] = −1 and t < 10 · W^2 do
    t := t + 1/2
    i := reached[0]
    while i ≠ −1 do
        pos[i] := pos[i] + dir[i]
        timeAt[pos[i]] := t
        if pos[i] = L[i] or pos[i] = R[i] then
            dir[i] := −dir[i]
        i := reached[i]
    i := notReached[0]
    prev := 0
    while i ≠ −1 do
        pos[i] := pos[i] + dir[i]
        if timeAt[pos[i]] = t then
            notReached[prev] := notReached[i]
            reached[i] := reached[0]
            reached[0] := i
        else
            prev := i
        if pos[i] = L[i] or pos[i] = R[i] then
            dir[i] := −dir[i]
        i := notRreached[i]
return timeAt[W]
```

overkant van de rivier niet kan bereiken, geldt dat $T = 10 \cdot W^2$. Zoals al eerder opgemerkt, is deze waarde willekeurig gekozen, waardoor het algoritme wellicht niet altijd correct werkt.

Een goede bovengrens voor T bij een instantie met N boten en een rivierbreedte W, is de langst mogelijk tijd die Mario nodig heeft om zo snel mogelijk de overkant te bereiken. Als we deze tijd kunnen uitdrukken in termen van N en/of W, dan kan voor elke invoer een simpele bovengrens gedefinieerd worden waarbij de simulatie nooit te vroeg stopt.

Om deze worst case tijden te bepalen, is een aantal experimenten uitgevoerd. In deze experimenten worden de worst case instanties met N boten berekend voor verschillende waardes van W. Dit gebeurt met een brute-force algoritme dat de lengte van het kortste pad naar de overkant berekent voor elke mogelijke invoer met N boten. De invoer die de langste tijd naar de overkant oplevert wordt hierbij opgeslagen. We hopen met deze experimenten een N te vinden, waarbij de worst case tijden gelijk zijn aan de worst case tijden van $N - 1$. Het is aannemelijk dat het toevoegen van extra boten vanaf deze N de worst case tijden niet verder zal verhogen. Er kan dan dus een algemene bovengrens voor T worden opgesteld. We komen hier aan het eind van deze paragraaf op terug.

We bespreken eerst de resultaten van het experiment voor instanties met $N = 2$. Als er slechts twee boten op de rivier varen, zal er maar één overstap plaatsvinden op het pad naar de overkant. In de worst case zal deze overstap zo lang mogelijk worden uitgesteld. Het valt daarom te verwachten dat de worst case instanties met $N = 2$ gelijk zijn aan die van het QuickSim algoritme. Dit algoritme simuleerde immers ook twee boten tot

een volgende kruising. Een worst case van QuickSim is $b_1 = [0, l_1]$ en $b_2 = [l_1 - 1, 2 \cdot l_1]$, waarbij l_1 even en ≥ 10 is.

Algoritme 11 experiment $N = 2$

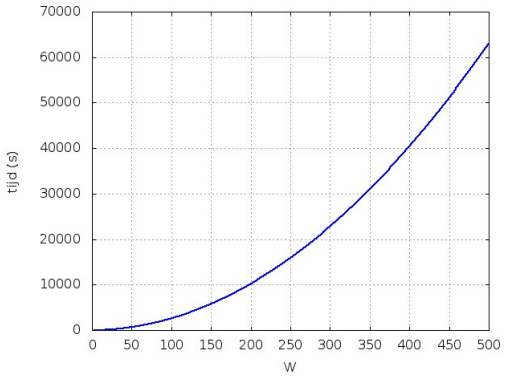
```
maxTime := 0
L[1] := 0
R[2] := W
for R[1] = 1 to W − 1 do
    for L[2] = 1 to R[1] do
        T := simulate(W, 2, L, R)
        if T > maxTime then
            maxTime := T
return maxTime
```

Algoritme 11 is gebruikt om de worst case instanties van alle rivierbreedtes $1 \leq W \leq 500$ te berekenen. Merk op dat de aanroep van *simulate* -1 oplevert als Mario de overkant van de rivier niet kan bereiken. Hoewel deze instanties in feite worst cases zijn van het simulatie algoritme, wordt de waarde van T dus niet in *maxTime* opgeslagen. Op deze manier wordt *maxTime* inderdaad de langst mogelijke tijd die Mario nodig heeft om zo snel mogelijk daadwerkelijk de overkant te halen. In Tabel 3.1 staan de resultaten van het experiment met $N = 2$ voor enkele waardes van W.

Tabel 3.1: De worst cases bij $N = 2$.

W	b_1	b_2	t
10	$[0, 4]$	$[3, 10]$	35
25	$[0, 13]$	$[11, 25]$	182
50	$[0, 26]$	$[23, 50]$	675
100	$[0, 50]$	$[49, 100]$	2601
250	$[0, 124]$	$[123, 250]$	15875
500	$[0, 250]$	$[249, 500]$	63001

Voor een worst case van QuickSim moet de rechtergrens van b_2 gelijk zijn aan $2 \cdot l_1$ (een veelvoud van vier). Hierdoor zijn alleen de instanties waar W een veelvoud van vier is, gelijk aan de worst case instanties van QuickSim. Voor de overige waardes van W vertonen de worst cases sterke overeenkomsten met de worst case van QuickSim. Zo hebben de boten van deze instanties ook een kleine overlap en is de periode van b_2 iets langer dan die van b_1. De halve periodes van b_1 en b_2 verschillen echter meer dan één afstandseenheid of de overlap is > 1, waardoor de worst case tijd relatief kort is ten opzichte van instanties waar $\frac{W}{2}$ even is.
Zoals uitgelegd in de beschrijving van de worst case van Quicksim, kruisen de boten $b_1 = [0, l_1]$ en $b_2 = [l_1 - 1, 2 \cdot l_1]$, met halve periode l_1 en l_2, elkaar op het tijdstip $l_1 \cdot l_2 - \frac{1}{2}$. Dit is ook de eerst mogelijke kruising tussen de boten, waardoor de worst case tijd $l_1 \cdot l_2 + l_2$ seconden is. Een goede bovengrens voor T bij $N = 2$ is daarom $l_1 \cdot l_2 + l_2 + 1 = \frac{W}{2} \cdot (\frac{W}{2} + 1) + \frac{W}{2} + 1 + 1 = \frac{W^2}{4} + W + 2$. In Figuur 3.1 is een grafiek afgebeeld waarin de worst case tijden voor alle waardes van W zijn geplot. Hierin is te zien dat de groei van T inderdaad kwadratisch is.



Figuur 3.1: De worst case tijd bij twee boten voor $1 \leq W \leq 500$.

We bekijken nu de worst case tijden van instanties met $N = 3$ of $N = 4$. Algoritme 12 is gebruikt om deze tijden te berekenen voor de instanties met $N = 3$. Het algoritme voor instanties met $N = 4$ werkt analoog. Vanwege de langere rekentijd zijn de experimenten voor $N = 3$ en $N = 4$ niet voor alle waardes van W uitgevoerd.

Algoritme 12 experiment $N = 3$

```
maxTime := 0
L[1] := 0
R[3] := W
for R[1] = 1 to W − 1 do
    for L[2] = 0 to W − 2 do
        for R[2] = L[2] + 1 to W − 1 do
            for L[3] = 1 to W − 1 do
                T := simulate(W, 3, L, R)
                if T > maxTime then
                    maxTime := T
return maxTime
```

Bij drie of vier boten zijn twee of drie overstappen nodig om de overkant te bereiken. Ook hier lijkt het logisch om elke overstap zo lang mogelijk uit te stellen. De verwachting is daarom opnieuw dat de perioden van de boten ongeveer even lang zijn en dat hun bereiken weinig overlappen. In Tabellen 3.2 en 3.3 is echter te zien dat dit niet het geval is.

Tabel 3.2: De worst cases bij $N = 3$.

W	b_1	b_2	b_3	t
10	$[0, 2]$	$[1, 8]$	$[6, 10]$	52
25	$[0, 2]$	$[1, 18]$	$[16, 25]$	297
50	$[0, 2]$	$[1, 34]$	$[33, 50]$	1105
100	$[0, 4]$	$[3, 42]$	$[41, 100]$	4543
250	$[0, 4]$	$[4, 145]$	$[144, 250]$	29574
500	$[0, 6]$	$[5, 276]$	$[274, 500]$	121362

Wat opvalt in Tabel 3.2 voor $N = 3$ is dat b_1 een zeer korte periode heeft. De overstap van b_1 naar b_2 vindt dan ook vrij snel plaats. Bij $W = 50$ gebeurt dit bijvoorbeeld al na 66 seconden. Boot b_2 wordt hierdoor echter pas bereikt vlak nadat de boot met b_3 is gekruist.

We definiëren het *kruisingstraject* van twee boten als het vaartraject van de boten tussen twee kruisingen in. Omdat b_2 en b_3 elkaar net gekruist zijn als b_2 wordt bereikt, moet bijna het volledige kruisingstraject worden afgelegd voordat de boten elkaar nogmaals kruisen. Bij $N = 2$ starten de boten al halverwege hun kruisingstraject, omdat beide boten op hun linkergrens beginnen. Dit verklaart waarom de worst case tijden bij $N = 3$ ongeveer verdubbelen ten opzichte van de tijden bij $N = 2$. Een goede bovengrens voor T bij drie boten lijkt daarom $2 \cdot (\frac{W^2}{4} + W + 2)$.

Tabel 3.3: De worst cases bij $N = 4$.

W	b_1	b_2	b_3	b_4	t
10	$[0, 4]$	$[1, 5]$	$[3, 8]$	$[5, 10]$	65
25	$[0, 12]$	$[2, 14]$	$[11, 22]$	$[14, 25]$	407
50	$[0, 24]$	$[1, 25]$	$[23, 48]$	$[25, 50]$	1825
100	$[0, 50]$	$[1, 51]$	$[49, 98]$	$[51, 100]$	7399
250	$[0, 124]$	$[1, 125]$	$[123, 248]$	$[125, 250]$	46625
500	$[0, 250]$	$[1, 251]$	$[249, 498]$	$[251, 500]$	186999

In Tabel 3.3 voor $N = 4$ valt op dat zowel b_1 en b_2 als b_3 en b_4 een grote overlap hebben. Dit leidt echter niet tot een snelle overstap, omdat de perioden van de boten gelijk zijn en de boten elkaar dus nooit kruisen. De eerste overstap vindt plaats tussen b_1 en b_3 op een tijd vergelijkbaar met de worst case van $N = 2$. De tweede overstap vindt vrijwel direct hierna plaats tussen b_3 en b_2. Boot b_2 loopt nu echter net b_4 mis, waardoor bijna het volledige kruisingstraject van b_2 en b_4 moet worden afgelegd. Als gevolg hiervan is de worst case tijd bijna drie keer zo groot als bij instanties met $N = 2$. Een goede bovengrens voor T bij vier boten lijkt daarom $3 \cdot (\frac{W^2}{4} + W + 2)$.

Vanwege het grote aantal instanties is het experiment voor $N = 4$ in eerste instantie alleen uitgevoerd voor $W \leq 50$. Toen uit de resultaten bleek dat de periodes van b_1 en b_2 en de periodes van b_3 en b_4 altijd gelijk zijn, is het experiment nogmaals uitgevoerd voor hogere waardes van W. Hiervoor is Algoritme 13 gebruikt. Door $R_2 = L_2 + (R_1 - L_1) = L_2 + R_1$ en $R_3 = L_3 + (R_4 - L_4) = L_3 + (W - L_4)$ te nemen kon het experiment nu wel in redelijke tijd (binnen enkele dagen) worden uitgevoerd.

Een brute-force algoritme voor algemene N analoog aan Algoritme 12 test $(W - 1)^N \cdot (\frac{W}{2})^{N-2}$ verschillende instanties. Voor de rechtergrens van boot b_1 en de linkergrenzen van boten b_2 tot b_N kunnen namelijk $W - 1$ waardes worden gekozen, waarna er voor de rechtergrenzen van boten b_2 tot b_{N-1} gemiddeld $\frac{W}{2}$ mogelijke waardes zijn. Het koste Algoritme 12 ongeveer drie dagen om de worst case van $N = 3$ bij $W = 500$ te berekenen. Hierbij werden ongeveer $(500 - 1)^3 \cdot (\frac{500}{2})^{3-2} \approx 3,1 \cdot 10^{10}$ instanties bekeken.

Bij $N = 5$ is het aantal instanties bij $W = 30$ al $(30 - 1)^5 \cdot (\frac{30}{2})^{5-2} \approx 6,9 \cdot 10^{10}$. Het leek daarom niet mogelijk om het experiment voor vijf boten uit te voeren. Na het zien van de resultaten uit Tabel 3.3 vermoedden we echter dat de worst cases van $N = 5$ wellicht ook een bepaalde structuur hebben. Om dit vermoeden te testen zijn toch de worst cases met vijf boten van $W = 10, 11, \ldots, 25$ berekend. Hieruit bleek dat als W even is, de

Algoritme 13 tweede experiment $N = 4$

```
maxTime := 0
L[1] := 0
R[4] := W
for R[1] = 1 to W − 1 do
    for L[2] = 1 to W − 2 do
        R[2] = L[2] + R[1]
        if R[2] ≥ W then
            continue
        for L[3] = 1 to W − 2 do
            for L[4] = 1 to W − 1 do
                R[3] = L[3] + (W − L[4])
                if R[3] ≥ W then
                    continue
                T := simulate(W, 4, L, R)
                if T > maxTime then
                    maxTime := T
return maxTime
```

worst case van $N = 5$ gelijk is aan de worst case van $N = 4$. Als W echter oneven en ≥ 13 is, dan is de worst case tijd van $N = 5$ iets langer dan die van $N = 4$. Ook deze worst cases hadden een bepaalde structuur: de periodes van b_1 en b_2 en van b_3 en b_4 bleken opnieuw gelijk. Het experiment voor $N = 5$ en oneven W is daarom ook voor hogere waardes van W uitgevoerd met Algoritme 14. De resultaten hiervan zijn te zien in Tabel 3.4.

Algoritme 14 experiment $N = 5$

```
maxTime := 0
L[1] := 0
R[5] := W
for R[1] = 1 to W − 1 do
    for L[2] = 1 to W − 2 do
        R[2] = L[2] + R[1]
        if R[2] ≥ W then
            continue
        for L[3] = 1 to W − 2 do
            for R[3] = L[3] + 1 to W − 1 do
                for L[4] = 1 to W − 2 do
                    R[4] = L[4] + (R[3] − L[3])
                    if R[4] ≥ W then
                        continue
                    for L[5] = 0 to W − 1 do
                        T := simulate(W, 5, L, R)
                        if T > maxTime then
                            maxTime := T
return maxTime
```

De worst cases van $N = 5$ lijken sterk op de worst cases van $N = 4$. Bovendien geldt voor alle gevonden worst cases van $N = 5$ dat de worst case tijd maximaal slechts vier seconden langer is dan de overeenkomende worst case van $N = 4$. Het toevoegen van b_5 heeft dus weinig effect op de worst case tijd. We kunnen hier echter niet uit concluderen dat het toevoegen van nog meer boten geen groot effect zal hebben op de worst

Tabel 3.4: De worst cases bij $N = 5$.

W	b_1	b_2	b_3	b_4	b_5	t
35	$[0, 16]$	$[1, 17]$	$[15, 32]$	$[17, 34]$	$[32, 35]$	837
37	$[0, 18]$	$[1, 19]$	$[17, 34]$	$[19, 36]$	$[34, 37]$	939
39	$[0, 18]$	$[1, 19]$	$[17, 36]$	$[19, 38]$	$[36, 39]$	1047
41	$[0, 20]$	$[1, 21]$	$[19, 38]$	$[21, 40]$	$[39, 41]$	1162
43	$[0, 20]$	$[1, 21]$	$[19, 40]$	$[21, 42]$	$[41, 43]$	1282
45	$[0, 22]$	$[1, 23]$	$[21, 42]$	$[23, 44]$	$[43, 45]$	1410

case tijd. Bij $N = 6$ vaart er namelijk een even aantal boten op de rivier, wat de situatie weer anders maakt. Gegeven dat we geen worst case tijd gevonden hebben die (veel) slechter is dan $\frac{3}{4}W^2$, vermoeden we dat de bovengrens $10 \cdot W^2$ goed genoeg is. Als deze bovengrens niet te laag is, dan is de worst case complexiteit van het simulatie algoritme $O(N \cdot W^2)$.

De bovengrens $10 \cdot W^2$ kan ook gebruikt worden om de bovengrens 10^9 uit het algoritme van Dijkstra te vervangen.

Hoofdstuk 4

Evaluatie

Bij het doorlopen van de graaf bezoekt het algoritme van Dijkstra vanaf de startknoop N knopen. Voor elke bezochte knoop worden telkens maximaal N kandidaat-knopen overwogen. Verder berekent het algoritme voor elke kandidaat-knoop het gewicht van de pijl naar de knoop. Bij twee boten met halve periodes l_1 en l_2 is de worst case complexiteit van deze berekening $O(l_1 + l_2)$ met QuickSim en $O(\min(l_1, l_2))$ met NextJunction. De waardes van l_1 en l_2 kunnen maximaal net zo groot zijn als de rivierbreedte W. De worst case complexiteit van het algoritme van Dijkstra bij een instantie met N boten en een rivierbreedte W is daarom $O(N^2 \cdot W)$.
In paragraaf 3.2 is vastgesteld dat de worst case complexiteit van het simulatie algoritme $\Omega(N \cdot W^2)$ is. Het beste algoritme voor een instantie van het Mario probleem lijkt daarom afhankelijk van de waardes van N en W. Omdat in de beschrijving van het probleem vermeld staat dat $0 \leq N \leq 100$ en $1 \leq W \leq 500$, vermoeden we dat het algoritme van Dijkstra in de meeste gevallen de beste oplossing is.
Om de rekentijd van de algoritmes te testen zijn vier experimenten uitgevoerd. Hierin moesten de algoritmes een aantal instanties van het Mario probleem oplossen. Voor het algoritme van Dijkstra is elk experiment drie keer uitgevoerd. Eén keer voor het algoritme zonder verbeteringen, één keer voor het algoritme met JunctionPossible en één keer voor het algoritme met JunctionPossible en NextJunction. Zo konden de effecten van deze twee verbeteringen goed in beeld worden gebracht.

Als eerste experiment hebben de algoritmes de oplossingen van duizend pseudo-willekeurige instanties berekend. De instanties uit dit experiment hadden de volgende eigenschappen:

- $N = 100$ en $W = 500$.
- Elke gegenereerde boot krijgt als halve periode een willekeurig getal l met $\lceil \frac{W}{2 \cdot N} \rceil \leq l \leq \min(8 \cdot \lceil \frac{W}{N} \rceil, W)$, en vervolgens een willekeurige linkergrens tussen 0 en $W - l$.
- Het is mogelijk om de overkant van de rivier te bereiken.

Met dit experiment willen we de gemiddelde rekentijd van elk algoritme bij een willekeurige instantie toetsen.

Voor het tweede experiment werd de rekentijd getest op een instantie van honderd boten die bestaat uit vijfentwintig kopieën van de worst case van het simulatie algoritme voor $N = 4$ en $W = 500$ ($[0, 250]$, $[1, 251]$, $[249, 498]$ en $[251, 500]$). Deze instantie is vanwege het grote aantal boten ook een worst case voor het algoritme van Dijkstra. We hopen met dit experiment te zien hoe de worst cases zich tot elkaar verhouden.

De op te lossen instantie uit het derde experiment bevat vijf kopieën van de instantie uit experiment 2, zodat het aantal boten is verhoogd naar 500. Omdat nu geldt dat $N = W$, is dit experiment eerlijker dan experiment 2.

Bij het laatste experiment is de rekentijd getest voor honderd pseudo-willekeurige instanties met bijna dezelfde eigenschappen als de instanties uit experiment 1. Het enige verschil met de instanties uit experiment 1 is dat de overkant van de rivier nu onbereikbaar is. We willen hiermee testen hoeveel tijd de algoritmes nodig hebben om te bepalen dat Mario de overkant niet kan bereiken. Merk op dat dit voor het simulatie algoritme een worst case is.

De resultaten van de vier experimenten zijn te zien in Tabel 4.1.

Tabel 4.1: De rekentijd in seconden van de algoritmes bij de vier experimenten.

	simulatie	Dijkstra	Dijkstra + JP	Dijkstra + JP & NJ
experiment 1	2.70	39.12	1.40	1.17
experiment 2	0.30	0.22	0.10	0.004
experiment 3	1.37	5.24	2.25	0.05
experiment 4	796.57	2.29	0.07	0.06

Uit de resultaten van experiment 1 blijkt dat het algoritme van Dijkstra zonder verbeteringen gemiddeld veel langzamer is dan het simulatie algoritme voor een willekeurige instantie met $N = 100$ en $W = 500$ waarbij de overkant bereikbaar is. Het aantal seconden dat nodig is om de overkant te bereiken blijkt over het algemeen relatief klein te zijn. Zo kost het Mario bij experiment 1 in het slechtste geval 4473 seconden om de overkant te bereiken. Ter vergelijking: bij de worst case instantie van experiment 2 zijn hiervoor 186999 seconden nodig. Het simulatie algoritme is daarom voor een willekeurige instantie met $N = 100$ en $W = 500$ relatief snel. Een tweede reden voor het grote verschil in de rekentijd van de algoritmes is het gebruik van QuickSim. Zonder de toevoeging van JunctionPossible, zal QuickSim ook de eerstvolgende kruising tussen niet-kruisende boten berekenen. Niet-kruisende boten vormen altijd een worst case voor QuickSim, omdat het algoritme de posities van de boten zal blijven simuleren tot de bovengrens van t bereikt is. De toevoeging van JunctionPossible versnelt de rekensnelheid van het algoritme van Dijkstra dan ook aanzienlijk.

Bij experiment 2 was het algoritme van Dijkstra iets sneller. Uit experiment 3 blijkt echter dat het simulatie algoritme zelfs in de worst case sneller is dan het algoritme van Dijkstra als geldt dat $N = W$. Bij deze twee experimenten zorgt het gebruik van NextJunction voor een duidelijke versnelling van het algoritme van Dijkstra. Dit is niet vreemd, want bij deze experimenten waren de instanties opgebouwd uit kopieën van de

worst case van het simulatie algoritme. De boten die hierin voorkomen, lijken erg op de worst case instanties van QuickSim. De rekensnelheid van het algoritme van Dijkstra met JunctionPossible en QuickSim is daardoor vrij laag. Voor deze boten werkt het period skippen in NextJunctionC juist snel, omdat y in Algoritme 7 gelijk wordt aan $(2 \cdot (250 - 249)) \mod 498 = 2$.

Als de bereikbaarheid van de overkant bepaald moet worden, is het algoritme van Dijkstra de duidelijke winnaar. De rekentijd van het simulatie algoritme bij dit experiment is zo hoog, omdat het algoritme de toekomstige posities van boten blijft simuleren tot de bovengrens van $10 \cdot W^2$ seconden bereikt wordt.

Hoofdstuk 5

Conclusies

In deze scriptie hebben we het probleem *Mario* uit de *Benelux Algorithm Programming Contest 2015* bestudeerd. Anders dan de titel van deze scriptie doet vermoeden, kon men de programmeerwedstrijd niet winnen door de beste oplossing voor dit probleem te vinden. Wel was *Mario* onverwachts het enige probleem uit de wedstrijd dat door slechts één team is opgelost. Dit team kreeg hiervoor een speciale prijs. Het doel van dit onderzoek was het bepalen van de beste oplossing voor *Mario*. Om de beste oplossing te bepalen, zijn twee algoritmes geanalyseerd die door juryleden van BAPC 2015 zijn geschreven.

Het eerste algoritme dat is onderzocht, lost het Mario probleem op met behulp van het kortste paden algoritme van Dijkstra. Na de correctheid van het algoritme te hebben aangetoond, is gezocht naar verbeterpunten van dit algoritme. Allereerst is het algoritme JunctionPossible gevonden dat in logaritmische tijd kan bepalen of twee boten elkaar ooit kruisen. Hierna is het drietal algoritmes NextJunction A, B en C bedacht dat samen in lineaire tijd het tijdstip van de eerstvolgende kruising tussen twee boten kan berekenen. De rekentijd van dit nieuwe algoritme is 30% sneller dan de rekentijd van het oude algoritme QuickSim. Een ander algoritme, dat niet beschreven staat in deze scriptie maar wel is onderzocht, maakt gebruik van de Chinese reststelling om het tijdstip van de eerstvolgende kruising te berekenen. Dit algoritme kan echter alleen kruisingen op posities x met x een geheel getal berekenen. Een bovengrens voor de worst case complexiteit van het algoritme van Dijkstra is $O(N^2 \cdot W)$, bij een instantie van het Mario probleem met N boten en een rivierbreedte W.

Het tweede algoritme dat we geanalyseerd hebben, simuleert de toekomstige posities van alle boten tegelijkertijd om zo de snelste tijd naar de overkant van de rivier te bepalen. We hebben met behulp van een aantal experimenten geprobeerd een bovengrens voor de worst case complexiteit van dit algoritme vast te stellen. Helaas is alleen de ondergrens $\Omega(N \cdot W^2)$ gevonden. Omdat we de correctheid van het algoritme nu niet kunnen garanderen, hopen we dat een betere bovengrens in een toekomstig onderzoek kan worden bepaald.

Om onze onderzoeksvraag te beantwoorden, zijn tot slot vier experimenten uitgevoerd. Uit de resultaten van deze experimenten blijkt dat het simulatie algoritme over het algemeen sneller is dan het onverbeterde

algoritme van Dijkstra, als de overkant van de rivier bereikbaar is. Bij instanties waarbij $x = W$ onbereikbaar is, is het algoritme van Dijkstra de betere keuze. Als het algoritme van Dijkstra gebruik maakt van de twee geïntroduceerde verbeteringen, dan is het algoritme van Dijkstra duidelijk de beste oplossing van het Mario probleem.

Bibliografie

[CLR90] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. The MIT Press and McGraw-Hill, 1990.

[Dij59] E.W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1:269–271, 1959.

[web15] BAPC 2015 · Benelux Algorithm Programming Contest. `http://2015.bapc.eu/`, november 2015. Bezocht op 16 augustus 2016.