Een oplossing voor het handelsreizigersprobleem

Door Stefan Veurink

11-9-2013

Voorwoord

Ik ben maar een amateur. Dat gegeven zal dan ook als rode draad door dit document lopen. Ik beheers vrijwel geen wiskundige notaties en/of terminologie, en heb er verder ook geen achtergrond in. Om heel eerlijk te zijn weet ik niet eens of ik de hele P=NP verhaal wel begrijp, ik vertrouw hierbij vooral op de uitleg en verklaringen van anderen. Wellicht blijkt het zo te zijn dat met een algoritme n^6 helemaal geen P=NP geldt. Of dat er met 'in een polynoom te vatten' heel wat anders wordt bedoeld. In dat geval is dit document vrij nutteloos.

Dit amateurisme heeft mij ook in mijn mogelijkheden beperkt. Het algoritme zoals in dit document beschreven is dan ook niet op specifieke voorbeelden getest. Simpelweg omdat ik niet kan programmeren. Het algoritme zoals beschreven is onttrokken uit een ander algoritme, waar het oorspronkelijk allemaal mee begonnen is. Dit andere algoritme is weliswaar vele malen uitgebreider, en bevat veel meer verschillende controleslagen en specifieke behandelingen, maar werkt in praktijk met de hand nog vrij snel, voornamelijk omdat eventuele fouten er doorgaans vanzelf mee boven komen drijven, waardoor je over het algemeen met een kleine gok wagen heel wat berekeningen achterwege kan laten. Het is te veel werk om dat hele algoritme uit te werken naar een goede omschrijving, ook door de ingewikkeldheid ervan. Daarom is het algoritme sterk veralgemeniseerd en aangepast op zo'n manier dat je 100% zeker geen fouten meer kunt maken. Dit betekent dan dus wel dat je er veel meer berekeningen mee uit moet voeren. Zoveel dat het eigenlijk niet meer mogelijk is deze met de hand uit te voeren en/of enigszins te controleren.

De opzet en bedoeling van dit document is dan ook vooral indirect aan te tonen dat P=NP, en niet zozeer om op de snelst mogelijke manier het handelsreizigersprobleem op te lossen. Het algoritme zoals in dit document beschreven valt dan ook overduidelijk nog te optimaliseren. Desalniettemin blijkt uit het document wel dat het mogelijk is binnen een tijdspanne n^x het probleem op te lossen.

Waarschijnlijk is het algoritme zoals in dit document beschreven niet perfect. Met andere woorden: wie goed zoekt, zal waarschijnlijk best een fout kunnen vinden in de snelheidsberekening, of bij het algoritme een verkeerde uitleg van woorden eraan kunnen geven. Dat doet echter niets af aan het algoritme zoals ik het probeer te beschrijven. Wie iets verder kijkt dan zijn neus lang is, en niet direct struikelt over de eerste de beste spelfout, moet aan de hand van dit document kunnen begrijpen hoe het handelsreizigersprobleem binnen n^x kan worden opgelost..

Voor mij persoonlijk denk ik, en dat vind ik best jammer, dat met het droppen van dit document mijn werk er wel op zit. Ervanuitgaand dat het algoritme klopt, zie ik geen manier waarop ik me verder nog nuttig kan maken. Een ieder is dan ook uitgenodigd ermee te doen wat hij wil, en het naar eigen inzicht aan te passen of uit te breiden. Ik vertrouw bij het plaatsen van dit document op de antwoorden die ik op dit forum heb gekregen, dank daarvoor, het lijkt me dan ook wel zo eerlijk het gewoon hier op dit forum (www.wetenschapsforum.nl) te plaatsen.

Indien u dus oprechte interesse heeft in hoe het handelsreizigersprobleem kan worden opgelost op een manier dat er geldt P = NP is het zinvol dit artikel te lezen. Wanneer u binnen 5 minuten de snelste route hoopt te vinden voor een praktische situatie kunt u beter een benaderingsmethode kiezen of zsm aan het rekenen slaan.

Legenda & Terminologie

Legenda

Omwille van de duidelijkheid heb ik ervoor gekozen de teksten van verschillende strekking in bijbehorende verschillende kleuren weer te geven. Wanneer ik wiskundige notaties en termen volledig zou beheersen was dit waarschijnlijk niet nodig geweest, maar zo is het dus niet. Het is misschien even wennen, maar op deze manier is het document denk ik toch stuk makkelijker te lezen en bovendien overzichtelijker. Grijze tekst kunt u bijvoorbeeld zien als de begeleidende beschrijving bij het algoritme. Blauw is gekozen voor extra informatie. Het daadwerkelijke algoritme zelf wordt in de zwarte tekst weergegeven. In totaal leidt dit tot de volgende kleuren:

+Persoonlijke tekst, die ik zelf graag wilde vermelden.
+Begeleidende tekst, welke de structuur en samenhang weergeeft bij de verschillende onderdelen
+Harde tekst, deze betreft puur het uit te voeren algoritme.
+Beschrijvende tekst, deze tekst geeft extra informatie over het algoritme welke misschien niet noodzakelijk is, maar wel als behulpzaam of verhelderend kan worden gezien.
+Snelheidsberekeningen. De berekening van de tijd die het kost het algoritme uit te voeren is weergegeven in het groen.
+ Een verwijzing naar een ander hoofdstuk of gedeelte binnen dit document.

Terminologie

+ kortste weg en kortste route
Wanneer in het document wordt gesproken over de weg tussen twee punten, gaat het over de afstand van de directe verbinding tussen deze 2 punten of rijen. Dit is dus een afstand die als input voor het algoritme gebruikt wordt.
Wanneer in het document wordt gesproken over de route tussen twee punten, of specifieker de kortste route tussen 2 punten, gaat het over de verbinding tussen die 2 punten, eventueel via andere steden of rijen. Het kan hier dus meer dan 1 lijn betreffen, een route kan bestaan uit meerdere afstanden die als input van het algoritme gebruikt zijn.

+ gekleurde / gevulde vakjes, verbindingen.
Het algoritme gebruikt als basis een afstandentabel. In deze tabel stelt iedere afstand een vakje voor. Wanneer we een verbinding kiezen doen we dit door een vakje in de tabel te vullen, te kleuren, of te markeren, of hoe je het ook wil noemen.

+ rij(x).
De afstandentabel bestaat uit een aantal rijen en uit een aantal kolomen. Wanneer er in dit document achter de term ‘rij’ of ‘kolom’ een getal tussen haakjes staat, staat dit getal voor het aantal gevulde vakjes in de desbetreffende rij.

Het algortime

Het algoritme zoekt de kortste route langs alle n punten, waarbij je uiteindelijk weer terugkomt bij het startpunt. Dit probleem kan worden omgevormd tot het officiële handelsreizigersprobleem, waarbij je niet meer terug hoeft naar het startpunt. Dat omvormen gebeurt door een punt P aan n toe te voegen, welke naar elk andere punt dezelfde afstand heeft. Wanneer men het algoritme begrijpt, zal men ook begrijpen dat het verstandig is die afstand relatief groot te maken. Grootste verschil tussen de twee problemen is dat er bij het probleem zoals in dit document besproken sprake is van een gesloten lijn, en bij het handelsreizigersprobleem niet. Dit betekent dat bij het handelsreizigersprobleem 2 punten aanwezig zijn met slechts 1 verbinding, terwijl bij het probleem zoals hier besproken alle punten 2 verbindingen naar andere hebben.

Bovendien is het algoritme (in eerste instantie) ontworpen voor een situatie waarbij de heenweg bij twee punten altijd even groot is als de terugweg bij die punten. Er geldt dus A -> B = B -> A. Overigens is er volgens mij geen reden te bedenken waarom het algoritme niet zou werken wanneer A -> B = B -> A niet altijd het geval is.

De situatie met zijn n punten en ∑n onderlinge afstanden valt weer te geven in een tabel, waar je op zowel de x als y as alle steden uitzet, en binnen de tabel de onderlinge afstanden invult. Wanneer elk punt met 2 andere punten verbonden moet zijn, betekent dit dus voor de tabel dat zowel elke kolom als elke rij 2 verbindingen (gevulde vakjes) moet bevatten. Dat die verbindingen (gevulde vakjes) in de tabel gespiegeld moeten zijn langs de diagonaal spreekt voor zich. Wanneer je ook nog eens de kortste route wilt, betekent dit dat je precies die combinatie van verbindingen moet hebben waarbij de totale som van alle verbindingen het laagste is. In eerste instantie is het aantal mogelijke combinaties ongeveer gelijk aan n^n. Echter zijn valt dit aantal aanzienlijk te beperken, dit is wat het algoritme dan ook zal doen. Wanneer je dan ook nog wil dat alle routes deel uitmaken van 1 en dezelfde route zul je moeten berekenen op welke manier verschillende routes met elkaar kunt verbinden, wederom op zo'n manier dat er zo min mogelijk afstand bij komt.

De situatie samengevat: het algoritme dat in dit document is beschreven geeft de kortste route langs alle n punten, waarbij:
- de route eindigt bij het punt waar je begonnen bent en heeft in feite dus geen start- en eindpunt.
- De onderlinge afstand tussen punt A en punt B is net zo groot als tussen punt B en punt A.

OPMERKING

Doordat in de tabel ieder punt altijd 1 keer met zichzelf kruist, en doordat je niet 1 maar 2 vakjes per zowel kolom als rij wil, is het zo dat niet altijd elke verplaatsingen van gevulde vakjes ook daadwerkelijk valt uit te voeren. Dit blijkt in het algoritme dan echter gelijk al bij het noteren van de afstand, omdat je direct ziet dat die afstand niet mogelijk is. In principe wordt dat probleem dus ondervangen. De enige situatie waar het eventueel niet meer ondervangen wordt is bij het verbinden van losse routes. Hier zorgt een verandering door het kiezen van de kortste weg ervoor dat er een nieuwe optie open komt. Hoewel deze optie bij het berekenen van andere rijen in principe alsnog boven komt drijven is voor alle zekerheid bij '[verbinden losse routes]' het punt 1b toegevoegd.

Het algoritme

Stap 1
BESCHRIJVING: je hebt een verzameling van n punten, met hun onderlinge afstanden.

- zet de punten en hun onderlinge afstanden uit in een tabel (op zowel de x als y as)

- geef in elke kolom de twee laagste vakjes een kleurtje (als 3 de laagste zijn maakt het niet uit welke je pakt). De vakjes zijn nu dus 'gevuld'.

- zet vervolgens bij iedere rij een cijfer met hoeveel vakjes er in die rij gekleurd zijn. Een rij wordt in dit document vanaf nu weergegeven als rij(x) met x het aantal gevulde vakjes in de rij.

BESCHRIJVING: grafisch gezien heb je nu dus elk punt verbonden met zijn 2 dichtstbijzijnden.

Stap 2:
In elke kolom staan nu 2 gevulde vakjes. Per rij kan dit echter verschillen. De situatie waar je naar toe wil is dat zowel in elke kolom als in elke rij 2 vakjes gevuld zijn.

- ga de rijen(< 2) volmaken door rijen(> 2) leeg te maken, of andersom, volgens 'De manier'

Doe dit dus totdat je er in elke rij 2 hebt.

Zodra in elke kolom en in elke rij 2 vakjes gevuld zijn en je hebt het goed uitgevoerd, zou de tabel ook gespiegeld moeten zijn langs de diagonaal. Dit kun je als controle zien.

BESCHRIJVING: grafisch gezien maakt elk punt nu deel uit van een 'route'. Ieder punt is (op de kortst mogelijke manier) met precies 2 andere punten verbonden, waarbij je altijd een aantal punten aflegt, en uiteindelijk weer terugkomt bij je beginplek.

Stap 3:
BESCHRIJVING: bij stap 3 is het raadzaam om de situatie zoals die na stap 2 is grafisch af te beelden. Op die manier zijn de verschillende routes veel makkelijker inzichtelijk te maken.

- Deel de rijen in groepen, waarbij elke groep staat voor een afzonderlijke route.

- Bepaal volgens 'Verbinden losse routes' de efficiëntste manier om 2 afzonderlijke routes te verbinden.

- verbindt deze twee routes waarmee er dus het minste afstand bij komt.

Herhaal stap 3 totdat alle routes verbonden zijn.

De Manier

Het algoritme

1) Je neemt een rij(0) of rij(1)
2) Noteer voor elk lege vakje de afstand vanaf elk gevulde vakje in zijn kolom (maximaal 2), maar alleen wanneer het gevulde vakje in een overvolle rij zit.
- noteer de laagste waarde en schrap de rest
3a) Bekijk bij elke rij met 2 gevulde vakjes voor elk gevulde vakje de afstand naar de rij waar je mee bezig bent. Noteer vervolgens de laagste van de 2 afstanden.
3b op1) Bekijk bij elke rij met 2 gevulde vakjes voor elk lege vakje naar de 2 gevulde vakjes in zijn kolom. Noteer de laagste afstand (als) die vanaf een overvolle rij gemaakt wordt. (dus noteer alleen een afstand als het gevulde vakje in een overvolle rij zit, en de laagste is).
3b op2) Noteer voor elk gevulde vakje in een overvolle rij zijn afstand naar elke rij van 2. Neem vervolgens voor een rij alle bijbehorende afstanden en noteer de laagste.
3.1) Combineer voor elke rij van 2 de afstand naar de rij waar je mee bezig bent (3a) met zijn laagste stap vanaf een overvolle (3b). Tel voor elke rij van 2 deze 2 waarden bij elkaar op.
- noteer de laagste waarde en schrap de rest
4) Voer voor alle rijen van 2 het algoritme van Dijkstra uit zoals beschreven bij Dijkstra
Noteer voor elke rij van 2 de hierbij kortst berekende route naar elke andere rij van 2
4a) Bekijk voor elke rij van 2 voor beide vakjes hoeveel het kost hem te verplaatsen naar de rij waar je mee bezig bent. Noteer (?alleen de laagste of ze allebei?)
4b) Bekijk voor elke rij van 2 voor elk lege vakje naar de 2 gevulde vakjes in zijn kolom. Noteer de laagste afstand (als) die vanaf een overvolle rij gemaakt wordt.
4.1a) Neem voor een rij(2) zijn afstand naar de rij waar je mee bezig bent. (van 4a)
4.1b) Neem voor elke andere rij(2) zijn kortste afstand vanaf een overvolle rij (van 4b), en tel deze op bij de kortste afstand naar de rij(2) uit (4.1a). (van 4)

4.1c) Neem de afstanden berekend bij 4.1b en tel bij elke afstand de afstand uit 4.1a op.
- noteer de laagste waarde en schrap de rest

4.2) voer 4.1a-c uit voor elke rij(2)
- noteer de laagste waarde en schrap de rest

5) je hebt bij stap 2, 3 en 4 een aantal waarden genoteerd. Neem de laagste waarde, en schrap de rest. Je hebt nu voor een rij(0) of een rij(1) de bewerking met laagste bijkomende afstand bepaald om hem te vullen. Voer deze manier dus uit voor elke rij(0) en rij(1). Bepaal vervolgens de laagste afstand en voer de bijbehorende bewerking uit.

Berekening snelheid 'De manier'

1) 1	1
2) 2n	1 + 2n
3a) 2n	1 + 2n + 2n
3bop2) $2n^2$	1 + 2n + $(2n + 2n^2)$
3.1) n	1 + 2n + $(2n + 2n^2 + n)$
4) D = $1 + n + 5n^2 + n^4$	D (voor D zie hoofdstuk 'Dijkstra')
4a) 2n	D + 2n
4b) $2n^2$	D + 2n + $2n^2$
4.1a) 1	D + 2n + $2n^2$ + (1)
4.1b) n	D + 2n + $2n^2$ + (1 + n)
4.1c) n	D + 2n + $2n^2$ + (1 + n + n)
-	
4.2) n	D + 2n + $2n^2$ + n(1 + n + n)
	D + 3n + $4n^2$

3 + 4)	$1 + 2n + (2n + 2n^2 + n)$	+	$1 + n + 5n^2 + n^4$	+	$3n + 4n^2$
3+ 4)	$1 + 5n + 2n^2$	+	$1 + n + 5n^2 + n^4$	+	$3n + 4n^2$
3+ 4)	$2 + 9n + 11n^2 + n^4$				

5) n	$n(2 + 9n + 11n^2 + n^4)$
	$2n + 9n^2 + 11n^3 + n^5$

We hebben nu dus de snelheid bepaald om voor 1 bewerking te berekenen wat de laagste optie is. Hiermee kunnen we dus 1 bewerking uitvoeren, 1 vakje goed zetten, in een tijdspanne die gegeven wordt door de formule

$M = 2n + 9n^2 + 11n^3 + n^5$

Het algoritme van Dijkstra

Het Algoritme van Dijkstra wordt gebruikt om de bewerkingen in kaart te brengen waarbij er meerdere verbindingen veranderd moeten worden, oftewel wanneer er sprake is van 'doortikken'. Met dit doortikken wordt bedoeld dat, wanneer je een bepaalde rij(2) bekijkt, het gekleurde vakje over de rij verplaatst lijkt te worden. In werkelijkheid wordt het vakje niet over de rij verplaatst. Het vakje dat gevuld wordt is binnen zijn kolom verplaatst naar de betreffende rij, het vakje dat geleegd is binnen zijn kolom juist bij de betreffende rij vandaan verplaatst.

Het is zinvol eerst dieper in te gaan op het algoritme van Dijkstra zelf. Met dit algoritme kan bij een situatie met n punten en bijbehorende afstanden voor een punt de kortste weg naar een willekeurig ander punt, maar net zo goed naar elk ander punt bepaald worden[1]. Dit gebeurt met een snelheid die evenredig is aan het aantal punten in het kwadraat[2]. Logischerwijs kun je dus wanneer je dit voor ieder punt uitvoert voor elk punt de kortste weg naar elk andere punt berekenen. Deze kortste weg is van belang bij het bepalen van de bewerking waarbij er zo min mogelijk afstand bijkomt. Het is namelijk mogelijk dat door onderling doortikken een gunstigere overvolle rij en te lege rij gebruikt kunnen worden dan wanneer het vakje uit de overvolle rij direct naar een te lege rij verplaatst wordt. Door gebruik te maken van het algoritme van Dijkstra hoeven hierdoor in een later stadium niet alle $n(2)^{n(2)}$ onderlinge bewerkingen bekeken te worden, maar slecht $n(2)^2$ opties.

Om verwarring te voorkomen dient direct al vermeld te worden dat het algoritme van Dijkstra dus niet gebruikt zal worden om de kortste route tussen verschillende punten te berekenen. De situatie zoals deze voor een bepaalde bewerking is wordt ingevoerd in het algoritme van Dijkstra om hiermee de bewerking te bepalen waarbij er zo min mogelijk afstand bij komt. Dit is een belangrijk verschil en verdient dan ook genoemd te worden.

Het algoritme van Dijkstra hoeft niet per se grafisch uitgebeeld te worden, en kan ook prima in tabelvorm geschieden. Om te zien hoe het algoritme van Dijkstra in tabelvorm in z'n werk gaat, zie bijvoorbeeld [3].

Het omwerken naar en uitvoeren van het algoritme van Dijkstra.

Als eerste is het van belang de situatie zoals die voor een bepaalde bewerking is, om te werken naar een juiste invoer voor het algoritme van Dijkstra.

1) Men neemt de tabel zoals die is.
2) Schrap alle rijen die niet rij(2) zijn.
3) Nu ga je voor elke rij(2) de kortste directe weg naar elke andere rij(2) bepalen:
3a1) Bekijk rij(2)1 en rij(2)2. In rij(2)1 zitten twee gevulde vakjes. Kijk voor elk gevulde vakje in rij(2)1 naar de waarde in de kolom van deze gevulde vakjes in rij(2)2. Bepaal voor beide hokjes hoeveel afstand erbij komt wanneer ze naar rij(2)2 verplaatst worden. Noteer de laagste waarde als kenmerkende 'afstand' van rij(2)1 naar rij(2)2.
3a2) Bepaal ook gelijk voor de andere rij(2), dus voor rij(2)2 de afstand naar rij(2)1 op dezelfde manier als 3a1. Hiermee kan de berekening waarschijnlijk iets versneld worden.

3b1) Bekijk vervolgens rij(2)1 tegenover rij(2)3. En rij(2)1 tegenover rij(2)4. En rij(2)1 tegenover rij(2)5. Voer deel 3a1-3a2 voor elke combinatie uit. Hiermee heb je dus de afstand van 1 rij(2) naar elke andere rij(2) bepaald

3b2) Doe dit vervolgens voor elke rij. Dus rij(2)2 naar rij(2)1. En rij(2)2 naar rij(2)3. En rij(2)2 naar rij(2)4. En rij(2) naar rij(2)5. En dan dus ook rij(2)3 naar rij (2)1. En rij(2)3 naar rij(2)2. Enzenzenz.

4) Vul telkens de laagste afstand(en) die je vindt in in de tabel.

5) Met de verkregen tabel kan vervolgens het algoritme van Dijkstra worden uitgevoerd. Zie hiervoor[3]

Bij de tabel zoals die gebruikt moet worden bij het algoritme van Dijkstra is er dus geen sprake meer van A -> B = B -> A.

Berekening snelheid omrekenen + uitvoeren algoritme Dijkstra

De tijdsduur die het kost als functie van n punten wordt gegeven door:

1) 1		
2) maximaal n		$1 + n$
3a1) 2		$1 + n + (2)$
3a2) 2		$1 + n + (2 + 2)$
3b1) n		$1 + n + n(2+2)$
3b2) n		$1 + n + 4n^2$
4) $\sum n$	$\sum(n) = 1/2n^2$	$1 + n + 5n^2$

5)Als ik het goed begrijp is het zo dat het algoritme van Dijkstra 1 keer uitvoeren in een tijdsduur n2 ertoe leidt dat je van elk punt de kortste weg naar 1 vast beginpunt weet. Wanneer je dit dan voor elke rij doet, betekent dat dus dat je $n * n^2 = n^3$ kan nemen als snelheid.
Het kan echter zijn dat ik fout zit, en dat het slechts zo is dat in n^2 de kortste route naar slechts 1 ander punt gevonden kan worden. In dat geval zal ja vanaf dat punt n keer Dijkstra uit moeten voeren om alle kortste wegen naar dat punt te bepalen. Dan zit je dus op $n * n^2 = n^3$. Vervolgens moet je dit dan nog voor elke rij uitvoeren, waarmee je snelheid komt op $n^3 * n = n^4$.
omdat het verschil tussen n^3 en n^4 geen invloed heeft op de vraag of de oplossingsnelheid in een polynoom gevat kan worden, zullen we verder gaan met n^4. Het kan dan achteraf alleen maar positief uitpakken.

De uiteindelijke tijd die het hele stukje 'Dijkstra' te berekenen wordt dan dus gegeven door:
$D = 1 + n + 5n^2 + n^4$

Verbinden losse routes

Wanneer je zover bent dat zowel in elke kolom als in elke rij 2 gevulde vakjes ofwel verbindingen zijn, betekent dit dus dat ieder punt nu echt met precies 2 andere punten verbonden is door middel van een lijn. Elke punt maakt dus deel uit van precies 1 route. Je zou misschien denken dat we er dan zijn, maar dat is nog net iets te vroeg gejuicht. Niet alle punten hoeven namelijk per definitie deel uit te maken van dezelfde route.

In feite is dit direct uit de tabel af te lezen. Wanneer je de tabel kunt splitsen in 2 tabellen zonder dat er vakjes veranderen, is er namelijk sprake van meerdere routes. Het is ook mogelijk de situatie zoals die na stap 2. is grafisch uit te beelden. In 1 oog opslag is dan te zien of alle punten op 1 en dezelfde route liggen.

Mocht blijken dat het algoritme in dit document toch niet altijd de kortste route geeft, durf ik vrijwel zeker te zeggen dat de fout in dit hoofdstuk 'verbinden van losse routes' zit. Ten eerste omdat het geen twijfel leidt dat de eerste 2 stappen correct zijn, ten tweede omdat ik niet helemaal zeker ben over stap 3. Hoewel in principe alles te beredeneren en te onderbouwen valt, blijft mijn gevoel toch twijfelachtig. Vragen die opspelen zijn bijvoorbeeld:
- waarom moet je de directe weg heen nemen, en de kortste route terug, en niet andersom?
Deze vraag is te beantwoorden met dat je het hieronder beschreven algoritme voor elke rij moet doen, en in feite dus wel degelijk beide opties uitrekent.
- waarom moet je niet zowel heen als terug de kortste route nemen?
Deze vraag is te beantwoorden met het feit dat een kortste route heen en een kortste route terug ook te schrijven is als een directe weg, plus een kortste route, maar dan dus wederom wel vanaf een andere rij
- kan een vrijgekomen vakje in een kolom plaats vrij maken voor een ander vakje, waardoor je een switchcontrole zou moeten uitvoeren.
Je zou zeggen: als het andere vakje (hogere) gevulde vakje in dezelfde groep zit is de weg vanaf dat andere vakje naar de betreffende rij buiten de groep net zo lang als de weg van het lagere vakje + de switch. Dat is niet helemaal waar omdat bij de switch er ook nog wat afgaat dankzij het andere bij de switchbetrokkenzijnde vakje, maar dat vakje wordt zonder de switchcontrole meegenomen bij het bepalen van de kortste route terug.
als het andere vakje (hogere) gevulde vakje in een andere groep zit kun je deze niet zomaar wegdenken, omdat je dan geen groepen met elkaar in verband brengt en er dus ook geen reden is nog een bewerking uit te voeren. ECHTER zit dat andere gevulde vakje per definitie in dezelfde groep. Dan lijkt het er toch wel op dat je geen switchcontrole hoeft uit te voeren.

Het algoritme 'verbinden van losse routes'

0) Vorm de tabel zoals die is om naar de tabel zoals die voor Dijkstra gebruikt kan worden.

1a) bepaal voor een rij binnen de groep zijn kortste weg naar een rij buiten de groep. Controleer hiervoor dus de twee gevulde vakjes in de betreffende rij, en trek deze af van de vakjes in dezelfde kolom van de desbetreffende andere rij buiten de groep. Let op: deze stap zet je dus niet in de tabel zoals die voor dijkstra gebruikt wordt, maar in de oorspronkelijke tabel. Je kan de waarde wel eventueel opzoeken in de tabel zoals die wel voor dijkstra gebruikt wordt.

1b) pas eventueel de afstand in de bij 0 verkregen tabel aan indien het andere gevulde vakje in de kolom waar de kortste weg genomen wordt hoger is, en indien hiermee de kortste weg van die andere rij naar de rij waar je mee bezig bent kleiner wordt. Volgens mij is 1b niet strikt noodzakelijk, immers zou die bij punt '3)' alsnog naar voren moeten komen, maar voor de zekerheid wil ik hem toch vermelden.

1c) bepaal voor de rij buiten de groep via Dijkstra zijn kortste route terug naar de rij binnen de groep
Deze stap voer je dus wel uit in de tabel zoals die voor Dijkstra verkegen is.

2) Voer 1a-1c uit voor elke rij buiten de groep naar de rij van 1 binnen de groep.

3) voer 2 uit voor elke rij.

Berekenen snelheid algoritme 'verbinden van losse routes'

0) $1 + n + 5n^2$ (zie Dijkstra)

1a) 2
1b) 1
1c) n^2

1) $2 + 1 + n^2$

2) $(3+n^2)*n$
3) $((3+n^2)*n*n)$

0 + 3) = $1 + n + 8n^2 + n^4$

Zoveel tijd kost het om voor 1 situatie de meest gunstige bewerking te bepalen. De meest ongunstige situatie is het wanneer elke route precies 3 punten aflegt, en je dus grafisch gezien allemaal kleine driehoekjes hebt. In dat geval moet je dus 1/3n routes samenvoegen, wat in de orde van grootte n ligt. Al met al kun je zeggen dat heel stap 3 dus

stap 3: $n + n^2 + 8n^3 + n^5$

Berekening snelheid algoritme

Stap 1:

$n^2 + 2n^2 + n^2 = 4n^2$

Stap 2:

$M = 2n + 9n^2 + 11n^3 + n^5$ i

M is de snelheid waarmee 1 optimale bewerking berekend kan worden

In totaal moet je dit maximaal 2n keer doen, dit betekent dus dat je maximaal 2n bewerkingen uit moet voeren. De meest ongunstige beginsituatie is immers diegene waarbij waar mogelijk alle laagste vakjes in dezelfde rij zitten ,waarmee je in feite dus 2 extreem overvolle rijen hebt, die helemaal goed gezet moeten worden. Je zult dan dus vrijwel alle 2n gevulde vakjes 1 keer moeten bewerken.

$2n * M = 2n(2n + 9n^2 + 11n^3 + n^5)$
$= 4n^2 + 18n^3 + 22n^4 + 2n^6$

Stap 1 + Stap 2 $= 8n^2 + 18n^3 + 22n^4 + 2n^6$

Stap 3 $= n + n^2 + 8n^3 + n^5$

stap 1 + stap 2 + stap 3 $= 8n^2 + 18n^3 + 22n^4 + 2n^6 + n + n^2 + 8n^3 + n^5$
$= n + 9n^2 + 26n^3 + 22n^4 + n^5 + 2n^6$

Aantal acties dat moet worden uitgevoerd om de kortste route langs alle punten te bepalen is dus:

$n + 9n^2 + 26n^3 + 22n^4 + n^5 + 2n^6$

Optimalisaties.
Het algoritme zoals in dit document beschreven kan sterk worden geoptimaliseerd. Punten waar makkelijk verbetering valt te halen zijn bijvoorbeeld:
- bij 'de manier' combineren van 3a en 4a, 3b en 4b
- niet telkens de hele Dijkstra tabel opnieuw bepalen, maar slechts enkele afstanden aanpassen aan je laatst gedane bewerking.

Anderzijds zou het net zo goed zou het kunnen dat je bij het bepalen en noteren van de laagste waarde binnen een verzameling ook n^x acties moet schrijven, en dat dat hier niet gebeurd is, hoewel ik daar geen voorbeeld van kan vinden. In dat geval wordt het algoritme weer iets langzamer. Desalniettemin zijn dit niet de verschillen die daadwerkelijk een noemenswaardig verschil maken in de vraag of de snelheid in een polynoom te vatten is, er zal altijd sprake blijven van een n^x, de n zal niet naar de x verschuiven.

Samengevat

De daadwerkelijke winst ten opzichte van de huidige efficiëntste methode, namelijk alle opties berekenen en vervolgens de laatste kiezen, zit hem in het gebruik maken van het algoritme van Dijkstra. Hierdoor valt een probleem waar in principe n^n opties mogelijk zijn, te beperken tot slechts n^x opties.

Wat het algoritme in feite doet is de situatie splitsen in 2 situaties, namelijk in de x-as en de y-as. Achterliggend weet je dat elke lijn wordt gedefinieerd door 2 punten (net zoals elk punt gedefinieerd wordt door 2 lijnen), en dus een startpunt en een eindpunt heeft. In feite kun je zeggen dat de x-as de beginpunten van de lijnen zijn, en de y-as de eindpunten van de lijnen, of andersom. In wezen is er echter helemaal geen verschil tussen start en eindpunt, omdat je de route, die als 2d-lijn gezien kan worden, in 2 richtingen kunt afleggen. Je houdt jezelf voor de gek door 2 situaties te gebruiken, terwijl eigenlijk beide situaties precies hetzelfde voorstellen, alleen vanuit een ander oogpunt. Je bekijkt namelijk de route in een andere richting.

Het splitsen van die 2 situaties gebeurt door het weer te geven in een tabel.

Bij het probleem zoals in dit document van toepassing zijn er aan die tabel dus 3 eisen gesteld:
- elke kolom 2 gevulde vakjes
- elke rij 2 gevulde vakjes
- laagste combinatie (waarbij 2 andere eisen gelden)

We zijn op zoek naar de situatie waar alle 3 de eisen gelden. Door alle kolommen de 2 laagste verbindingen te geven zet je 2 van deze 3 eisen goed. Vervolgens ga je door te sleutelen aan 1 eis, de laagste combinatie, zorgen dat de tabel ook aan de laatste eis voldoet. Elke keer bereken je voordat je zo'n foutje recht zet wat de meest efficiente manier hiervoor is, oftewel op welke manier er het minste afstand bij komt. Op die manier zet je de rijen steeds goed op de meest efficiënte manier, maar nooit ten koste van de kolommen.

Wanneer elke rij ook goed staat heeft zowel elke rij als elke kolom dus 2 verbindingen. Dit betekent dat elk punt 2 startpunten en 2 eindpunten van lijnen bezit. Omdat je zoals gezegd de 2d-route in 2 richtingen kunt aflopen en er ook geen wezenlijk maar slechts een subjectief verschil is tussen de x en y-as, overlappen deze 2 startpunten met de 2 eindpunten, waarmee elk punt dus in feite 1 start en 1 eindpunt heeft, en daarmee deel uitmaakt van een route. De situatie is dus dat elk punt deel uit maakt van een route, en wel op de manier waarop de totale afstand het laagste is. De laatste stap is dan de eventueel verschillende losse routes te verbinden op wederom de meest efficiënte manier.

Bij het verbinden van die losse routes ga je dus eerst geforceerd de twee routes met elkaar verbinden ten koste van 1 van die 2 routes. Je brengt als het ware weer een foutje aan in de tabel. Deze fout heeft een bepaalde waarde. Vervolgens bekijk je via Dijkstra wat de meest efficiënte manier is om dat foutje weer op te lossen. Die manier heeft eveneens een waarde, samen opgeteld zijn ze kenmerkend voor het betreffende foutje. Door vervolgens alle mogelijke 'foutjes' te controleren en na te rekenen bepaal uiteindelijk elke keer opnieuw voor een gegeven situatie bij welk foutje er het minste afstand bij komt.

De uiteindelijke route die je krijg is het kortste, omdat eerst ieder punt met de 2 dichtstbijzijnde punten verbindt, en je dus uitgaat van de meest ideale beginsituatie. Echter is er nog geen sprake van een route, daarvoor moeten er fouten worden rechtgezet, maximaal 2n om precies te zijn. Bij elke bewerking waarmee zo'n fout wordt rechtgezet, wordt berekend wat de meest gunstige manier hiervoor is, dus wat de manier is waarop er zo min mogelijk afstand bij komt. De laatste stap is de maximaal 1/3n foutjes die er dan nog inzitten door 'losse routes' eveneens steeds op de meest efficiënte manier op te lossen.

Bronverwijzingen

---

http://en.m.wikipedia.org/wiki/Dijkstra's_algortihm

beiden uit inleidende beschrijving:
[1] "Dijkstra's original algorithm does not use a min-priority queue and runs in O ($|V|^2$) (where |V| is the number of vertices)

[2] "For a given vertex (node) in the graph, the algortihm finds the path with lowest cost between that vertex and every other vertex"

---

[3] http://m.youtube.com/watch?v=_8jr5DVxGiA&desktop_uri=%2Fwatch%3Fv%3D_8jr5DVxGia

Bijlage toegevoegd ter info:

Omdat ik er niet aan kon ontkomen toch even de switchcontrole te noemen, is dit deel toegevoegd. De switchcontrole hoeft dus niet te worden uitgevoerd bij het algoritme zoals beschreven in dit document. Het was wel nodig bij de werkversie ervan die dus zeer veel verschillende handelingen bevatte, maar uiteindelijk is er voor gekozen deze versie niet in het document op te nemen, omdat dat de leesbaarheid en gebruiksvriendelijkheid en implementeerbaarheid geen goed doet.

SWITCHCONTROLE:

Bij elke berekening voor een bewerking die je doet moet je een controleslag doen.

Er komt namelijk altijd 1 (en in geval van doortikken meer dan 1) vakje vrij, oftewel het gaat van gevuld naar niet gevuld.

Het kan zo zijn dat het andere gevulde vakje in de kolom deze plaats in wil nemen. dat is zo als je bij de bewerking met het laagste van de 2 vakjes bezig was.

Daarom moet je dus 'switchen' controleren

je neemt daarvoor het nietbewerkte vakje

en je neemt de gevulde vakjes in die rij van het vakje dat wel bewerkt is.

en je controleert of die gevulde vakjes naar die rij kunnen van het niet bewerkte vakje, met of zonder doortikken

~~Dit moet je dus doen voor elk vakje dat vrijkomt, dus maximaal n keer per bewerking.~~

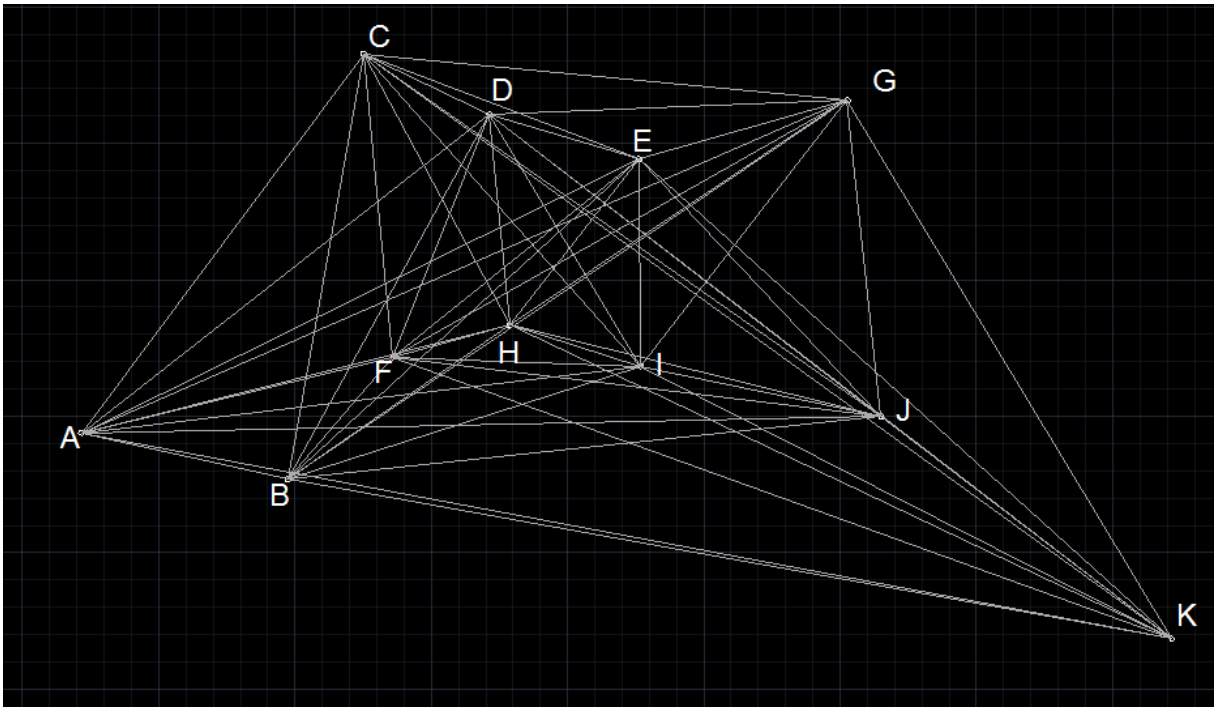
~~De vraag is dan moet je dit alleen doen bij de bewerking, of moet je het al bij het uitzoeken van de beste bewerking, en dus voor elke mogelijke bewerking.~~

Voorbeeld

Dit voorbeeld heb ik pas op het laatste moment pas gemaakt en toegevoegd, omdat ik bij een laatste keer nalezen toch niet zeker was of het grote plaatje nu wel duidelijk werd. Daarom hier wat grafisch materiaal aan de hand van een SIMPEL voorbeeld, met als een doel een beeld te krijgen van de termen en omschrijvingen in het document:

Je hebt een situatie:

Met bijbehorende verbindingen:



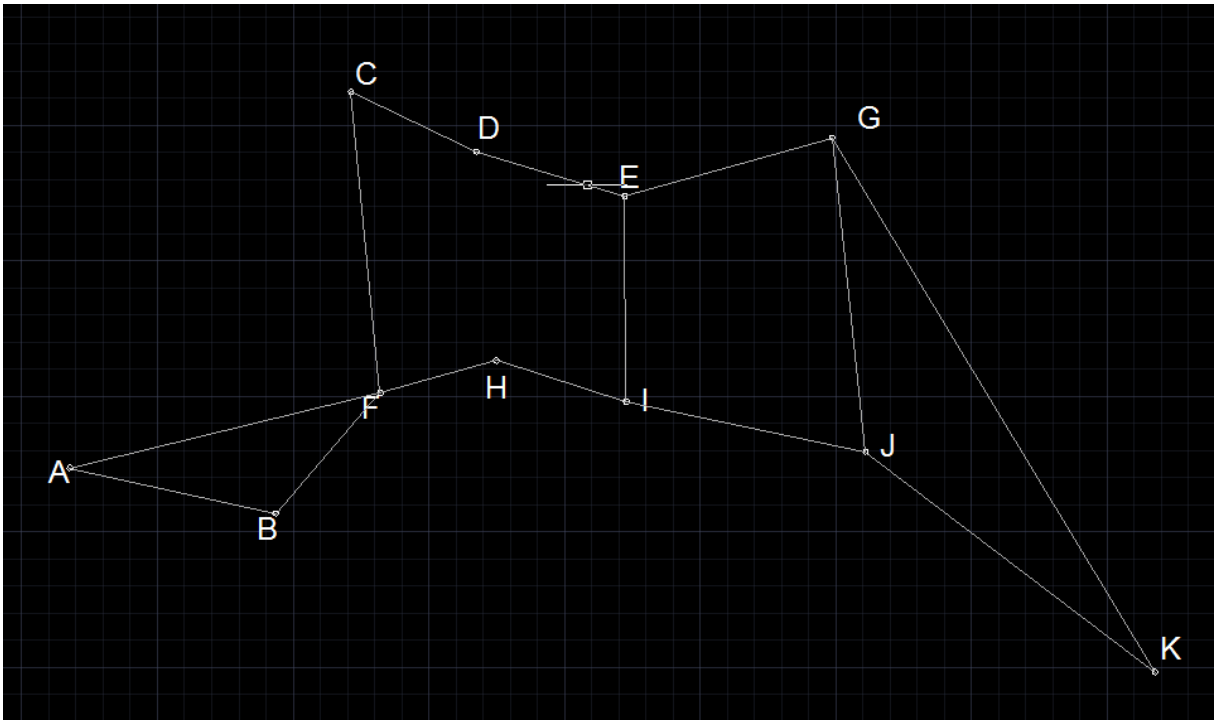
Deze verbindingen zet je uit in een tabel:

	A	B	C	D	E	F	G	H	I	J	K
A	X	78	173	190	228	118	306	162	207	293	407
B	78	XXX	158	153	174	59	248	99	136	219	329
C	173	158	XXX	51	108	112	178	113	153	232	366
D	190	153	51	XXX	57	96	131	77	107	181	315
E	228	174	108	57	XXX	116	79	77	76	129	263
F	118	59	112	96	116	XXX	191	44	91	180	304
G	306	248	178	131	79	191	XXX	148	123	116	230
H	162	99	113	77	77	44	148	XXX	50	140	269
I	207	136	153	107	76	91	123	50	XXX	90	219
J	293	219	232	181	129	180	116	140	90	XXX	134
K	407	329	366	315	263	304	230	269	219	134	XXX

Vervolgens voer je stap 1 helemaal uit:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1		A	B	C	D	E	F	G	H	I	J	K		
2	A	X	78	173	190	228	118	306	162	207	293	407		1
3	B	78	XXX	158	153	174	59	248	99	136	219	329		2
4	C	173	158	XXX	51	108	112	178	113	153	232	366		1
5	D	190	153	51	XXX	57	96	131	77	107	181	315		2
6	E	228	174	108	57	XXX	116	79	77	76	129	263		3
7	F	118	59	112	96	116	XXX	191	44	91	180	304		4
8	G	306	248	178	131	79	191	XXX	148	123	116	230		2
9	H	162	99	113	77	77	44	148	XXX	50	140	269		2
10	I	207	136	153	107	76	91	123	50	XXX	90	219		3
11	J	293	219	232	181	129	180	116	140	90	XXX	134		2
12	K	407	329	366	315	263	304	230	269	219	134	XXX		0

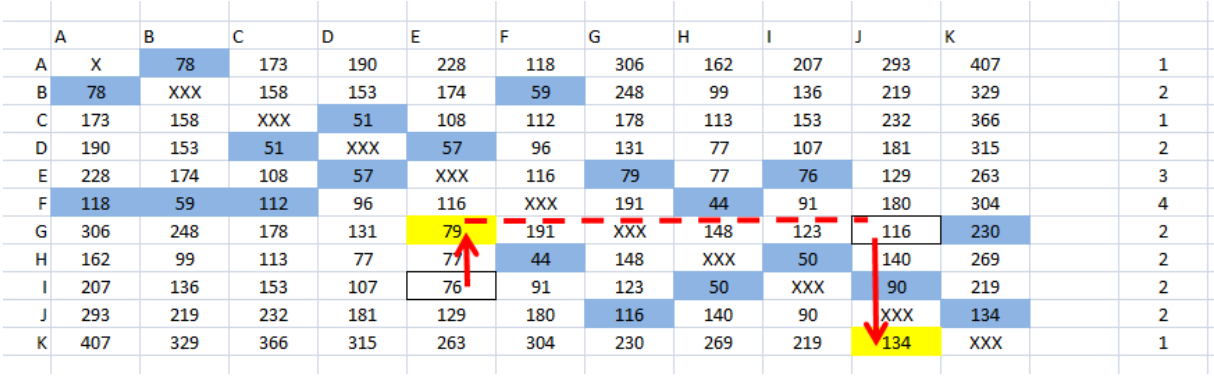
Je hebt nu grafisch gezien de volgende situatie:



Het maakt natuurlijk niet uit of je bovenstaande situatie hebt opgenomen vanuit de x-as of vanuit de y-as, omdat er geen wezenlijk verschil is tussen het begin en eindpunt van een lijn.

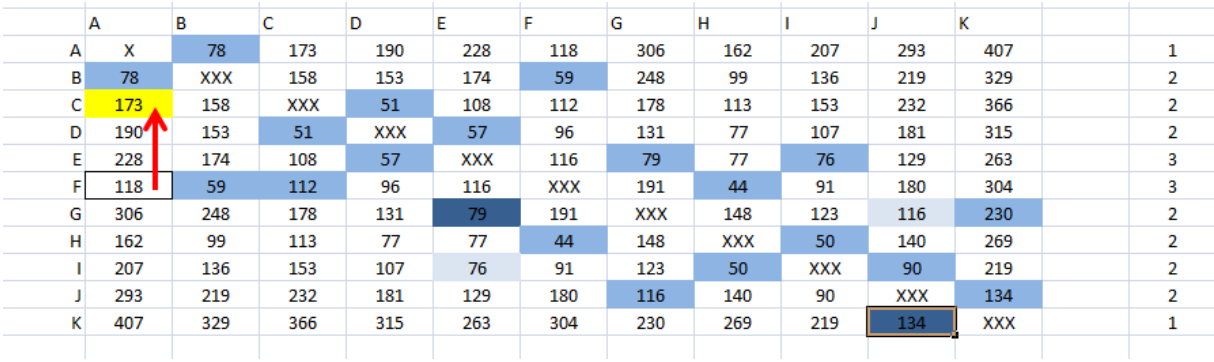
Stap 2:

Wat we nu dus gaan doen is te lege rijen vullen door de overvolle rijen te legen. Hiervoor worden telkens alle mogelijke bewerkingen berekenen volgens ‘de manier’. Vervolgens voeren we steeds de laagst mogelijke uit. Dit leidt achtereen volgens tot de volgende uit te voeren bewerkingen:



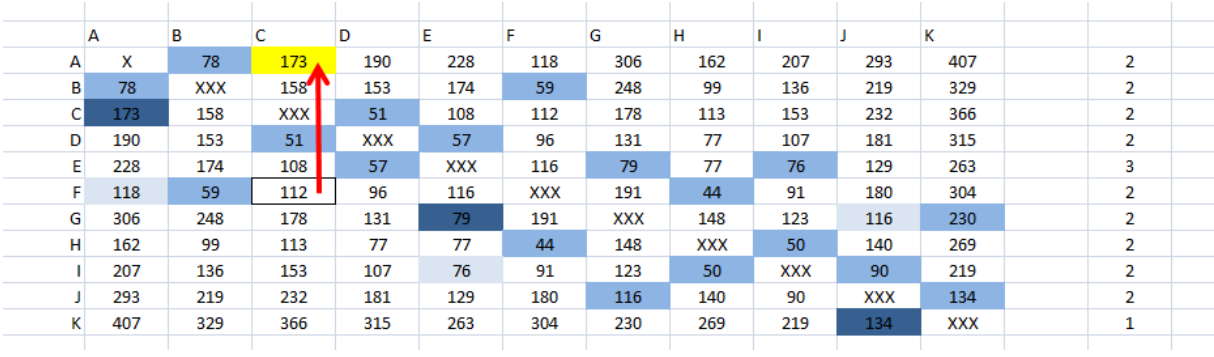
	A	B	C	D	E	F	G	H	I	J	K	
A	X	78	173	190	228	118	306	162	207	293	407	1
B	78	XXX	158	153	174	59	248	99	136	219	329	2
C	173	158	XXX	51	108	112	178	113	153	232	366	1
D	190	153	51	XXX	57	96	131	77	107	181	315	2
E	228	174	108	57	XXX	116	79	77	76	129	263	3
F	118	59	112	96	116	XXX	191	44	91	180	304	4
G	306	248	178	131	79	191	XXX	148	123	116	230	2
H	162	99	113	77	77	44	148	XXX	50	140	269	2
I	207	136	153	107	76	91	123	50	XXX	90	219	2
J	293	219	232	181	129	180	116	140	90	XXX	134	2
K	407	329	366	315	263	304	230	269	219	134	XXX	1

Eerst een bewerking van 3+18=21. Vervolgens berekenen we opnieuw volgens de manier alle mogelijk uit te voeren bewerkingen, wat leidt tot een stap van 55:



	A	B	C	D	E	F	G	H	I	J	K	
A	X	78	173	190	228	118	306	162	207	293	407	1
B	78	XXX	158	153	174	59	248	99	136	219	329	2
C	173	158	XXX	51	108	112	178	113	153	232	366	2
D	190	153	51	XXX	57	96	131	77	107	181	315	2
E	228	174	108	57	XXX	116	79	77	76	129	263	3
F	118	59	112	96	116	XXX	191	44	91	180	304	3
G	306	248	178	131	79	191	XXX	148	123	116	230	2
H	162	99	113	77	77	44	148	XXX	50	140	269	2
I	207	136	153	107	76	91	123	50	XXX	90	219	2
J	293	219	232	181	129	180	116	140	90	XXX	134	2
K	407	329	366	315	263	304	230	269	219	134	XXX	1

Vervolgens een stap van 61:



	A	B	C	D	E	F	G	H	I	J	K	
A	X	78	173	190	228	118	306	162	207	293	407	2
B	78	XXX	158	153	174	59	248	99	136	219	329	2
C	173	158	XXX	51	108	112	178	113	153	232	366	2
D	190	153	51	XXX	57	96	131	77	107	181	315	2
E	228	174	108	57	XXX	116	79	77	76	129	263	3
F	118	59	112	96	116	XXX	191	44	91	180	304	2
G	306	248	178	131	79	191	XXX	148	123	116	230	2
H	162	99	113	77	77	44	148	XXX	50	140	269	2
I	207	136	153	107	76	91	123	50	XXX	90	219	2
J	293	219	232	181	129	180	116	140	90	XXX	134	2
K	407	329	366	315	263	304	230	269	219	134	XXX	1

En tot slot een stap van 128:

	A	B	C	D	E	F	G	H	I	J	K	
A	X	78	173	190	228	118	306	162	207	293	407	2
B	78	XXX	158	153	174	59	248	99	136	219	329	2
C	173	158	XXX	51	108	112	178	113	153	232	366	2
D	190	153	51	XXX	57	96	131	77	107	181	315	2
E	228	174	108	57	XXX	116	79	77	76	129	263	2
F	118	59	112	96	116	XXX	191	44	91	180	304	2
G	306	248	178	131	79	191	XXX	148	123	116	230	2
H	162	99	113	77	77	44	148	XXX	50	140	269	2
I	207	136	153	107	76	91	123	50	XXX	90	219	2
J	293	219	232	181	129	180	116	140	90	XXX	134	2
K	407	329	366	315	263	304	230	269	219	134	XXX	2

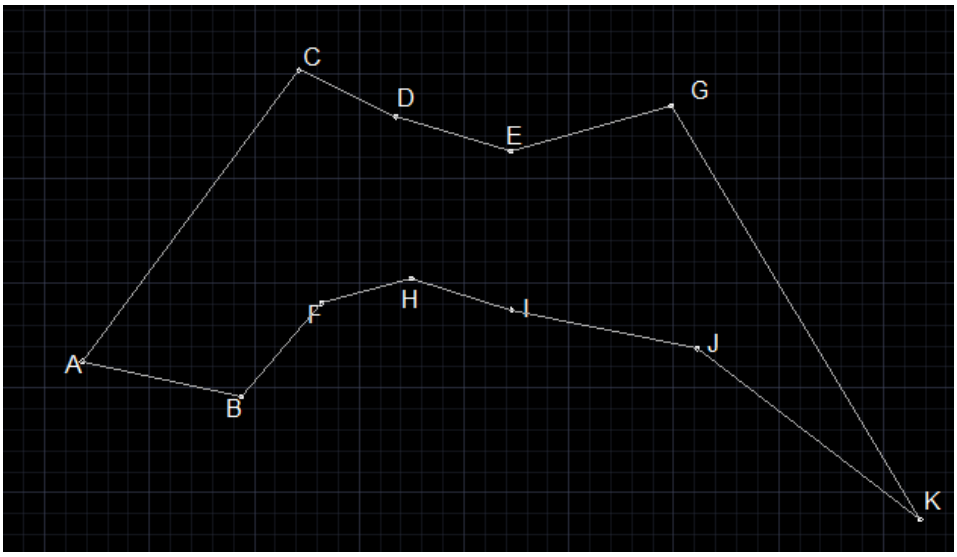
Nu zijn we waar we willen zijn. Elke rij heeft 2 gevulde vakjes, elke kolom heeft twee gevulde vakjes. De tabel ziet er nu dus als volgt uit:

	A	B	C	D	E	F	G	H	I	J	K	
A	X	78	173	190	228	118	306	162	207	293	407	2
B	78	XXX	158	153	174	59	248	99	136	219	329	2
C	173	158	XXX	51	108	112	178	113	153	232	366	2
D	190	153	51	XXX	57	96	131	77	107	181	315	2
E	228	174	108	57	XXX	116	79	77	76	129	263	2
F	118	59	112	96	116	XXX	191	44	91	180	304	2
G	306	248	178	131	79	191	XXX	148	123	116	230	2
H	162	99	113	77	77	44	148	XXX	50	140	269	2
I	207	136	153	107	76	91	123	50	XXX	90	219	2
J	293	219	232	181	129	180	116	140	90	XXX	134	2
K	407	329	366	315	263	304	230	269	219	134	XXX	2

We kunnen nog even controleren of gevulde vakjes gespiegeld zijn langs de diagonaal, wat inderdaad het geval is. Stap 2 is dus nu voltooid.

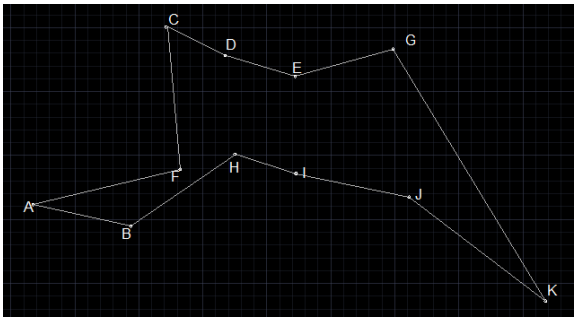
<u>Stap 3</u>

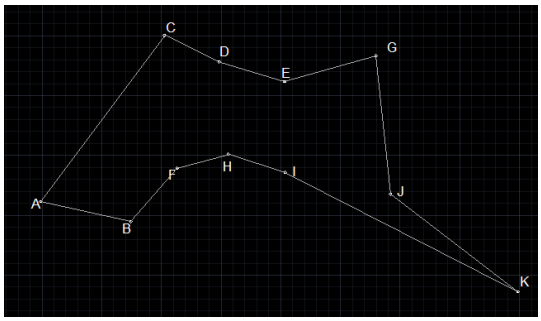
We implementeren de situatie zoals die in de tabel staat weer naar een grafische voorstelling, waardoor in 1 opslag te zien is of er nog sprake is van verschillende routes:



Blijkbaar is dat bij dit specifieke voorbeeld niet het geval. We zijn dus klaar. Hierboven staat de kortste route langs alle punten, waarmee je weer uitkomt bij het startpunt. De route blijkt 1047 ‘meter’ lang.

Voor alle zekerheid controleer ik nog 2 andere potentiële kanshebbers:





Respectievelijk zijn ze 1097 en 1061 ‘meter’. Langer dus.

Overigens zie ik dat er bij stap 1 een klein foutje gemaakt is, maar deze heeft verder geen noemenswaardig effect, u mag zelf zoeken waar die zit.