STICHTING
MATHEMATISCH CENTRUM
2e BOERHAAVESTRAAT 49
AMSTERDAM
AFDELING ZUIVERE WISKUNDE

ZW 1968-009

Voordracht in de serie
"Elementaire onderwerpen vanuit hoger standpunt belicht"
door
Prof.Dr. F.E.J. Kruseman Aretz.

"Waarom langer reizen dan nodig is?"

1. Formulering van het te behandelen probleem

Gegeven n steden, en bij elk geordend paar steden de afstand. Gevraagd: wat is de minimale lengte van een rondreis, waarbij elke stad precies één maal bezocht wordt? Geef een voorbeeld van een zo kort mogelijke route.

Bij bovenstaande opgave wordt natuurlijk niet gevraagd naar het antwoord van één specifiek geval, maar naar een methode, een algorithme om bij elk probleem van bovenstaande vorm op systematische wijze een oplossing te vinden.

Voor dit z.g. handelsreizigersprobleem is direct een algorithme aan te geven: nummer de steden met de getallen 1 t/m n, dan zijn er evenveel routes, die in stad 1 beginnen en eindigen, als er permutaties zijn van de getallen 2 t/m n. Schrijf al deze (n-1)! permutaties op, en bepaal voor elk de lengte van de corresponderende route. Hieruit volgen op eenvoudige wijze de minimale lengte en een route die deze lengte heeft.

Afgezien van het feit, dat we nog niet aangegeven hebben, hoe we op systematische wijze die (n-1)! permutaties genereren, is dit algorithme praktisch onbruikbaar, daar voor n = 20 reeds $1.2 * 10^{17}$ routes bekeken moeten worden.

Tot de efficientste processen, die tot nu toe gevonden zijn, behoort de methode van Little, Murty, Sweeney en Karel (Operations Research 11 (1963) 972).

2. Het algorithme van Little e.a.

Dit zullen we schetsen aan de hand van een voorbeeld van 5 steden met de volgende afstandsmatrix A:

x	27	11	19	15
27	x	18	16	16
11	17	x	13	6
19	16	13	x	21
15	16	6	21	x

$s = 0$ (1)

Hierin geeft A_{ij} de afstand van stad i naar stad j. Merk op dat $A_{23} \neq A_{32}$.

Elke rondreis zal n trajecten bevatten, en wel uit iedere rij en kolom precies één. Immers: we moeten elke stad één maal aan doen, d.w.z., er één maal naar toe reizen, en er één maal uit vertrekken. Hieruit volgt, dat als we elk element uit een willekeurige rij of kolom met hetzelfde bedrag wijzigen, de lengte van elke route met datzelfde bedrag verandert. Hierbij blijft iedere optimale route dus optimaal. Twee problemen waarvan de afstandsschema's uit elkaar verkregen kunnen worden door (herhaalde) kolom - of rij-veranderingen zullen we equivalent noemen.
We zullen nu (1) herleiden tot een equivalent probleem, door eerst de rijen, en daarna de kolommen te reduceren, d.w.z.: eerst in elke rij alle elementen te verminderen met het kleinste element uit die rij:

x	16	0	8	4
11	x	2	0	0
5	11	x	7	0
6	3	0	x	8
9	10	0	15	x

s = 52 (2)

en vervolgens in elke kolom alle elementen te verminderen met het kleinste element uit die kolom:

x	13	0	8	4	4
6	x	2	[0]	0	0
0	8	x	7	0	0
1	0	0	x	8	0
4	7	0	15	x	4
1	7	0	7	0	

s = 60 (3)

De waarde van s geeft hierbij aan, hoeveel in totaal alle routes van het nieuwe, equivalente probleem korter zijn dan die van het oorspronkelijke probleem met afstandsmatrix (1).
Aangezien in (3) uitsluitend niet-negatieve elementen voorkomen, is elke route uit (1) dus tenminste 60 eenheden lang. We hebben dus al een ondergrens gevonden voor de kleinste lengte.
We zullen nu, na deze reduktie, een traject i → j gaan uitkiezen, waarvan het voordelig lijkt het in een route op te nemen om de lengte klein te houden. We laten de keus vallen op een traject i → j met lengte 0 in (3), zodanig, dat de som van het op een na kleinste element in rij i en het op een na kleinste element in kolom j zo groot mogelijk is. Als we dan n.l. achteraf onderzoeken, wat de implicaties zijn van het niet-opnemen van traject i → j, zal de boete, die we voor dit weren van het traject in kwestie moeten betalen zo groot mogelijk zijn. Het is daartoe handig, achter elke rij en onder iedere kolom van (13) dit op één na kleinste element te noteren. Als we de matrix rijgewijs afzoeken, valt onze keuze op het omcirkelde element. We krijgen een vertakking: we moeten

nu van twee groepen routes een optimale gaan bepalen, en deze onderling vergelijken. De groepen zijn:

1) alle routes die niet traject 2 → 4 bevatten; minimale lengte $\geq 60 + 7 = 67$,

2) alle routes die wel traject 2 → 4 bevatten; minimale lengte ≥ 60.

We beginnen met het onderzoek van groep 2.

Aangezien we de elementen uit rij 2 en kolom 4 van (3) niet meer nodig hebbenm verwisselen we de 2e en 5e rij, en de 4e en 5e kolom van (3). Dit leidt tot de matrix:

	1	2	3	5	4
1	x	13	0	4	8
5	4	7	0	x	15
3	0	8	x	0	7
4	1	100	0	8	x
2	6	x	2	0	0

s = 60 (4)

Om te verhinderen, dat we in onze route ook nog het traject 4 → 2 op nemen (waardoor we een deelreis 2 → 4 → 2 zouden krijgen) hebben we element (4,2) door iets groots te vervangen. Het is hier voldoende om er 100 bij op te tellen. Om te weten, waar dat element nu staat, hebben we voor elke rij en boven elke kolom het juiste plaatsnummer geschreven.

We hoeven nu nog slechts 4 trajecten te kiezen uit het linker bovendeel van de matrix. In dit gedeelte is geen nieuwe rij-reductie mogelijk, maar wel een kolom-reductie. We trekken van kolom 2 een 7 af, maar zullen dat doen in de hele matrix (4) ten einde een met (1) equivalente matrix te houden (op element (4,2) na!).

Zo krijgen we:

	1	2	3	5	4	
1	x	6	[0]	4	8	4
5	4	0	0	x	15	0
3	0	1	x	0	7	0
4	1	93	0	8	x	1
2	6	x	2	0	0	
	1	1	0	4		

s = 67 (5)

We kunnen nu uit de deelmatrix een traject gaan kiezen op dezelfde manier als zo straks. De keuze valt dan op traject 1 → 3, en we krijgen een nieuwe vertakking. Te onderzoeken zijn de groepen routes:

1) alle routes die niet traject 2 → 4 bevatten; minimale lengte ≥ 67,

21) alle routes die wel 2 → 4, maar niet 1 → 3 bevatten; minimale lengte ≥ 71,

22) alle routes die zowel 2 → 4 als 1 → 3 bevatten; minimale lengte ≥ 67.

We gaan met groep 22 verder. Nu hoeven we de elementen in rij 1 en kolom 3 niet langer te beschouwen. Daarom verwisselen we rij 1 en 4, en kolom 3 en 5. Dit leidt tot de matrix:

	1	2	5	3	4
4	1	93	8	0	x
5	4	0	x	0	15
3	100	1	0	x	7
1	x	6	4	0	8
2	6	x	0	2	0

s = 67 (6)

waarin we element 3 → 1 met 100 verhoogd hebben om een keuze van traject 3 → 1 te voorkomen (dit zou n.l. leiden tot de deelreis 1 → 3 → 1). We kunnen nu een rij-reductie uitvoeren in de 3 x 3 links-boven-deelmatrix, en komen tot:

	1	2	5	3	4	
4	[0]	92	7	-1	x	7
5	4	0	x	0	15	4
3	100	1	0	x	7	1
1	x	6	4	0	8	
2	6	x	0	2	0	
	4	1	7			

s = 68 (7)

Het negatieve element 4 → 3 is geen bezwaar aangezien we nog slechts trajecten mogen kiezen uit het 3 x 3 deel. De keuze valt dan op traject 4 → 1.

De opgave vertakt zich opnieuw: we kennen nu al 4 groepen, n.l.,

1) alle routes die niet traject 2 → 4 bevatten; minimale lengte $\geq$ 67,

21) alle routes die wel 2 → 4 maar niet 1 → 3 bevatten; minimale lengte $\geq$ 71,

221) alle routes die wel 2 → 4 en 1 → 3 bevatten, maar niet 4 → 1; minimale lengte $\geq$ 79,

222) alle routes die 2 → 4, 1 → 3 en 4 → 1 bevatten; minimale lengte $\geq$ 68.

We gaan met groep 222 verder.
We verwisselen rij 4 en 3 en kolom 1 en 5, en verhogen element 3 → 2 om de deelreis 1→3→2→4→1 uit te sluiten. We krijgen zo:

	5	2	1	3	4	
3	[0]	101	100	x	7	101
5	x	0	4	0	15	x
4	7	92	0	-1	x	
1	4	6	x	0	8	
2	0	x	6	2	0	
	x	101				

s = 68 (8)

Een verdere reductie blijkt niet mogelijk, en, hoewel onze keuzestrategie de keus zou doen vallen op traject 3 → 5, hebben we in feite geen keuze: we moeten in dit gereduceerde 2 x 2 probleem wel de trajecten 3 → 5 en 5 → 2 kiezen.

We hebben nu een route aangewezen, n.l.:
1 → 3 → 5 → 2 → 4 → 1 met lengte 68. Het is de beste uit groep 222) (die in feite maar 1 route bevatte!) en dankzij de gevolgde strategie kunnen we hopen, dat deze route niet tot de langere routes zal behoren. Het zal blijken meteen de beste te zijn! In ieder geval is 68 een bovengrens **voor** de kleinste lengte.

We gaan nu eindelijk de vruchten plukken van al het werk, dat we tot nu toe al geinvesteerd hebben, en dit is de pointe van de hele methode:

We hoeven de routes in groep 221 niet verder te beschouwen, aangezien ze alle langer zijn (n.l. minstens 79) dan de route, die we in groep 222 gevonden hebben.
Evenmin behoeven we de routes in groep 21 te onderzoeken: ook deze zijn alle kansloos. Wel moeten we nog groep 1 gaan behandelen.
Hierbij kunnen we uitgaan van de matrix (8), mits we daarin ongedaan maken de verhogingen van de elementen $3 \to 2$, $3 \to 1$, en $4 \to 2$, en bovendien element $2 \to 4$ passend verhogen om een keuze van traject $2 \to 4$ in groep 1 onmogelijk te maken. We krijgen dan:

	5	2	1	3	4
3	0	1	0	x	7
5	x	0	4	0	15
4	7	-8	0	-1	x
1	4	6	x	0	8
2	0	x	6	2	100

$s = 68$ (9)

De negatieve elementen verdwijnen vanzelf;
rijreductie geeft:

	5	2	1	3	4
3	0	1	0	x	7
5	x	0	4	0	15
4	15	0	8	7	x
1	4	6	x	0	8
2	0	x	6	2	100

$s = 60$ (10)

kolomreductie vervolgens:

	5	2	1	3	4	
3	0	1	0	x	0	0
5	x	0	4	0	8	0
4	15	[0]	8	7	x	7
1	4	6	x	0	1	1
2	0	x	6	2	93	2
	0	0	4	0	1	

$s = 67$ (11)

De keuze valt op traject 4 → 2. Dit is niet toevallig: aangezien de oorspronkelijke tabel bijna symmetrisch is, zal de tegengestelde reis van 1 → 3 → 5 → 2 → 4 → 1 ook wel heel goed zijn.
We krijgen een vertakking in de groepen:

11) alle routes die noch 2 → 4 noch 4 → 2 bevatten; minimale lengte $\geq$ 74,

12) alle routes die niet 2 → 4 maar wel 4 → 2 bevatten; minimale lengte $\geq$ 67.

We gaan verder met groep 12). We verwisselen rij 4 en 2 en kolom 2 en 4, en verhogen element 2 → 4 nog eens extra.
We krijgen:

	5	4	1	3	2	
3	0	0	0	x	1	0
5	x	8	4	0	0	4
2	0	193	6	2	x	2
1	4	1	x	0	6	1
4	15	x	8	7	0	
	0	1	4	0		

s = 67 (12)

Te reduceren valt er niet; we kiezen traject 3 → 1 en vertakken.
We hebben dan de groepen:

11) alle routes die noch 2 → 4 noch 4 → 2 bevatten; minimale lengte $\geq$ 74,

121) alle routes die niet 2 → 4, wel 4 → 2, en niet 3 → 1 bevatten; min. lengte $\geq$ 71,

122) alle routes die niet 2 → 4 maar wel 4 → 2 en 3 → 1 bevatten; min. lengte $\geq$ 67.

We gaan verder met groep 122). We verwisselen rij 3 en 1 en kolom 1 en 3. Verder verhogen we element 1 → 3 om de deelroute 1 → 3 → 1 uit te sluiten. We verkrijgen:

	5	4	3	1	2
1	4	1	100	x	6
5	x	8	0	4	0
2	0	193	2	6	x
3	0	0	x	0	1
4	15	x	7	8	0

s = 67 (13)

In 13 is rij 1 te reduceren. Dit leidt tot s = 68
voor groep 122). Hieruit volgt, dat we het onderzoek van 122) kunnen staken, want geen route < 68 zal er in gevonden kunnen worden.

Maar ook groep 121) en groep 11) kunnen geen kortere routes dan 68 meer opleveren, en deze groepen behoeven dus geen verdere analyse. Aangezien we daarmee alle groepen behandeld hebben, weten we nu definitief, dat 1 → 3 → 5 → 2 → 4 → 1 een optimale route is.

3. Een programma voor het algorithme van Little e.a.

De in de vorige paragraaf gegeven beschrijving van het algorithme van Little e.a. is allerminst volledig: zo wordt bijvoorbeeld niet aangegeven, welk(e) element(en) van de matrix we ophogen om de keuze van het bijbehorende traject te voorkomen, en op welk moment we die ophogingen weer ongedaan maken. Ook is de volgorde, waarin we de groepen onderzoeken, niet wel omschreven.

Onze dagelijkse omgangstaal leent zich er niet zo goed toe om al zulke details uitputtend weer te geven; en zoals een wiskundige zijn toevlucht neemt tot formules om zijn structuren aan te duiden, zal een beschrijver van algorithmen zich uitdrukken in een formele taal, speciaal geschapen om rekenmethodes streng te formuleren. In paragraaf 4 is voor het onderhavige algorithme gebruik gemaakt van ALGOL 60. De beschrijving is zo compleet en gedetailleerd, dat rekenautomaten er minder moeite mee hebben dan de meeste lezers.

In deze paragraaf willen we dat programma toelichten, en aantonen, dat het correct is. Allereerst dient te worden opgemerkt, dat het hier gegeven programma niet het efficientst denkbare is:

veel aandacht is besteed aan de begrijpelijkheid.

De kern van het programma wordt gevormd door de procedure "optimize". Het hierdoor beschreven proces heeft de volgende externe specificaties:

"optimize" kent, en werkt met, een aantal grootheden, die buiten "optimize" gedefinieerd worden. Hiertoe behoort m, het aantal steden, en de m x m - matrix "A". Aan het begin van een uitvoering van het proces bevat "A" de afstandstabel van een met het oorspronkelijke probleem equivalent probleem, behoudens dat:

a) rijen en kolommen gepermuteerd mogen zijn,

b) sommige elementen opgehoogd kunnen zijn om de keuze van de corresponderende trajecten te voorkomen.

Bij het einde van die uitvoering van het proces voldoet "A" aan dezelfde voorwaarden. In het bijzonder zal voor elk traject, waarvan het element tijdens het proces opgehoogd is, deze ophoging weer ongedaan gemaakt worden.

Welke permutaties op rijen en kolommen zijn toegepast sinds het begin van het gehele programma, is te vinden in de m-vectoren "row", "col", "invrow", "invcol". Hierbij geeft bijvoorbeeld "row [i]" aan, welke stad bij de i-de rij van "A" hoort, terwijl "invrow [i]" de rij van "A" aanwijst, die met de i-de stad correspondeert; "row" en "invrow" zijn dus inverse permutaties.

Aan het begin van een uitvoering van "optimize" zijn reeds (m-n) trajecten uitgekozen. Welke dit zijn is af te lezen uit de laatste (m-n) elementen van de vectoren "row" en "col". Zo is de recentste keuze $p \rightarrow q$ geweest met p = row [n+1], q = col [n+1]; als gevolg van die keuze is juist een verwisseling uitgevoerd om de elementen corresponderende met trajecten $p \rightarrow x$ naar de n + 1-ste rij, en die corresponderende met trajecten $y \rightarrow q$ naar de n + 1-ste kolom te verhuizen.

Na die (m-n) reeds gedane keuzes mag "optimize" nog n trajecten kiezen. De elementen, die hiermee corresponderen bevinden zich juist in het n x n linker-boven gedeelte van "A" (de elementen "A[i, j]" met $i \leq n$, $j \leq n$). Het is de taak van "optimize" om

a) na te gaan of er, gegeven de (m-n) reeds gedane keuzes, en eventueel gegeven een aantal trajecten die niet gekozen mogen worden, een route te vinden is, waarvan de lengte kleiner is dan de grootheid "length",
b) in dat geval een bij de gegeven restricties optimale route vast te leggen in de vector "route", en de lengte van die route vast te leggen in de grootheid "length".
In de loop van de uitvoering van het programma zal "length" een monotoon dalende grootheid zijn in die zin, dat iedere wijziging een verlaging betekent. De initiële waarde van "length" zal groter dan elke route moeten zijn.

Het gehele proces wordt op gang gebracht door de activering van "optimize" die te vinden is in de op-drie-na-laatste regel van de declaratie van het proces "shortest route". De procedure "shortest route" heeft o.a. als taak de grootheden "s", "length", "row", "col", "invrow", "invcol", alsmede enkele andere hulpgrootheden te introduceren. Verder worden de vectoren "row", "col", "invrow" en "invcol" passend geinitialiseerd met de identieke permutatie, en de trajecten p → p verboden door bij de diagonaal-elementen van "A" een groot getal, "giant" genoemd, op te tellen. Tevens wordt aan "length" datzelfde grote getal toegekend. Vervolgens wordt "optimize" geactiveerd met als parameter m; er zijn nog geen trajecten gekozen, en er zijn (buiten de oneigenlijke trajecten p → p) nog geen trajecten verboden. Aan alle beginvoorwaarden voor een aanroep van "optimize" is voldaan; volgens bovenstaande specificaties zal na voltooiing van deze aanroep de vector "route" een optimale route aangeven, met lengte "length". En aangezien er geen restricties waren ten aanzien van de te onderzoeken routes, en vooraf "length" een voldoend grote waarde had, is daarmee de gewenste oplossing gevonden.

We moeten nog controleren, dat de procedure "optimize" inderdaad aan de specificaties voldoet. Laten we nagaan wat er achtereenvolgens gebeurt.
In de eerste drie alinea's, beginnend met "for i:= ", "for j: = " en "for i: = " wordt de matrix "A" gereduceerd, zodanig dat in het n x n

linker-boven gedeelte slechts elementen ≥ 0 voorkomen, terwijl hierin in iedere (deel-)rij of (deel-)kolom ten minste één nul voorkomt. Tevens wordt de waarde van het een-na-kleinste element van iedere (deel-)rij of (deel-)kolom in de vectoren "rowmin" en "colmin" genoteerd. Als tijdens de kolom-reductie blijkt, dat de waarde van "s" na de reductie tenminste "length" zal bedragen, wordt het proces verlaten en is "optimize" klaar: bij de gegeven restricties is geen route korter dan "length" mogelijk.

In de vierde alinea, beginnend met "1: = 1", wordt een rij-index "ii" en een kolom-index "jj" gevonden, zodanig dat $1 \leq ii \leq n$, $1 \leq jj \leq n$, $A[i,j] = 0$, $1 = \text{rowmin}[ii] + \text{colmin}[jj]$ maximaal. De boete op het niet kiezen van het met $A[i,j]$ corresponderende traject is juist 1.
In de nu volgende alinea wordt dat traject $u \rightarrow v$ genoemd; door rij- en/of kolomverwisselingen wordt bereikt, dat de n-de rij met "u", de n-de kolom met "v" correspondeert.

In de zesde alinea hangt de behandeling af van de waarde van n. Als n = 2 weten we zeker, dat we een kortere route gevonden hebben. Door de reductie van een 2 x 2 matrix ontstaan n.l. steeds twee nullen; door de verwisselingen staan deze nu gegarandeerd op de hoofd-diagonaal. De keuze van de trajecten row [1] → col [1] en row [2] → col [2] leidt tot een route met totale lengte "s", die, aangezien "optimize" niet afgebroken is, kleiner dan "length" moet zijn. We nemen de gevonden route over in "route" en de lengte ervan in "length". Als n > 2 gaan we vertakken. Eerst onderzoeken we alle routes die traject $u \rightarrow v$ bevatten. Daartoe bepalen we eerst het beginpunt "ii" en eindpunt "jj" van de langste rij op elkaar aansluitende, reeds gekozen trajecten die $u \rightarrow v$ bevat, en hogen het met $jj \rightarrow ii$ corresponderende element tijdelijk op om kortsluitingen te voorkomen. We willen nu weten, of, gegeven de m - n + 1 gekozen trajecten en de uitgesloten trajecten (inclusief $jj \rightarrow ii$), uit de overblijvende (n-1) x (n-1) matrix nog (n-1) trajecten gekozen kunnen worden op een zodanige wijze dat we een rondreis met kleinere lengte dan "length" vinden. Het proces dat dit precies doet heet "optimize" , dat we dan ook activeren met parameter "n-1". Na terugkeer weten we, dat als er

betere routes in deze tak aanwezig waren, ook zeker een beste hieruit gekozen is.
We maken nu de ophoging van het met traject "jj → ii" corresponderende element ongedaan en onderzoeken vervolgens, of de andere tak nog een kans heeft. De ondergrens van deze groep was al tijdig in de locale grootheid "l" vastgelegd. Als het nog zin heeft om de tak te onderzoeken die juist u → v weert, maken we het met u → v corresponderende element door ophoging onverkiesbaar, en zoeken, of, gegeven de m - n gekozen trajecten en de uitgesloten trajecten (inclusief u → v) uit de overblijvende n x n matrix nog n trajecten gekozen kunnen worden op zodanige wijze dat we een kortere rondreis vinden dan de tot nu toe beste. Het proces dat dit doet heet "optimize", dat we dan ook activeren met parameter "n". Na terugkeer weten we, dat als er betere routes in deze tak aanwezig waren, ook zeker een beste hieruit gekozen is.
Na de verhoging van het met u → v corresponderende element ongedaan gemaakt te hebben, zijn we klaar met het proces "optimize". We hebben, als er met de restricties die bij de aanroep golden, een betere oplossing mogelijk was, deze zeker gevonden.
Het moge lijken, alsof we de juistheid van de procedure "optimize" bewezen hebben door van de juistheid van "optimize" gebruik te maken. Toch is de redenering niet circulair, maar recursief: we maken gebruik van de juistheid van "optimize" voor een kleiner probleem, een probleem, waarin in ieder geval de keuzevrijheid van trajecten beperkt is, hetzij doordat "n", het aantal nog te kiezen trajecten, kleiner is, hetzij doordat meer trajecten verboden zijn dan tevoren. In beide richtingen heeft de recursie een einde: voor n = 2 doordat een speciale behandeling gegeven wordt, die geen verder beroep op "optimize" doet en voor gelijkblijvende n doordat, zodra alle n elementen uit een kolom opgehoogd zijn, een zo grote reductie van de matrix mogelijk is dat zeker "s" > "length", waardoor het proces wordt afgebroken.

De rest van het programma bevat de instructies, nodig om de afstandstabel in te lezen, de tijdsduur te meten, en de resultaten af te drukken, terwijl ook de procedure "optimize" nog enkele tussenresultaten afdrukt die het mogelijk maken de gekozen vertakkingen te volgen.

Het hier beschreven algorithme is een typisch voorbeeld van de zogenaamde "branch and bound" - techniek. Afgezien van de efficientie ervan, die vooral te danken is aan de goede strategie waarmee trajecten uitverkozen worden, is de elegance van het algorithme groot. Dit blijkt duidelijk uit de mogelijkheid, het te beschrijven in een zo kort programma van een zo doorzichtige structuur.

4. De ALGOL 60 - tekst

```
begin comment handelsreiziger-probleem, FKA, R1609;
      integer n;
      n:= read;
      begin integer i, j;
            real t;
            integer array route[1 : n];
            integer array A[1 : n, 1 : n];

            integer procedure shortest route (A, m, route);
            value m; integer m; integer array A, route;
            begin integer i, j, rowi, rowmini, s, t, length, giant;
                  integer array row, col, invrow, invcol[1 : m];
                  integer array rowmin, colmin[1 : m];

                  procedure optimize (n); value n; integer n;
                  begin integer l, ii, jj, u, v;
                        NLCR; SPACE ((m - n) × 5); ABSFIXT (2, 0, n);
                        for i:= 1 step 1 until n do
                        begin u:= A[i, 1];
                              for j:= 2 step 1 until n do
                              begin t:= A[i, j];
                                    if t < u then u:= t
                              end;
                              if u ≠ 0 then
                              begin s:= s + u;
                                    for j:= 1 step 1 until m do
                                    A[i, j]:= A[i, j] - u
                              end
                        end;

                        for j:= 1 step 1 until n do
                        begin u:= A[1, j]; v:= A[2, j];
                              if u > v then begin t:=u; u:=v; v:=t end;
                              for i:= 3 step 1 until n do
                              begin t:= A[i, j];
                                    if t < u then begin v:=u; u:=t end
                                             else if t < v then v:= t
                              end;
                              if s + u ≥ length then goto end;
                              if u ≠ 0 then
                              begin s:= s + u;
                                    for i:= 1 step 1 until m do
                                    A[i, j]:= A[i, j] - u
                              end;
                              colmin[j]:= v - u
                        end;
```

```
for i:= 1 step 1 until n do
begin u:= A[i, 1]; v:= A[i, 2];
      if u > v then begin t:=u; u:=v; v:=t end;
      for j:= 3 step 1 until n do
      begin t:= A[i, j];
            if t < u then begin v:=u; u:=t end
                     else if t < v then v:= t
      end;
      rowmin[i]:= v
end;

l:= - 1;
for i:= 1 step 1 until n do
begin rowmini:= rowmin[i];
      for j:= 1 step 1 until n do
      if A[i, j] = 0 then
      begin t:= rowmini + colmin[j];
            if t > l then
            begin l:= t; ii:= i; jj:= j end
      end
end;

u:= row[ii]; v:= col[jj];
if ii ≠ n then
begin t:= row[n]; row[ii]:= t; row[n]:= u;
      invrow[u]:= n; invrow[t]:= ii;
      for j:= 1 step 1 until m do
      begin t:= A[ii, j]; A[ii, j]:= A[n, j];
            A[n, j]:= t
      end
end;
if jj ≠ n then
begin t:= col[n]; col[jj]:= t; col[n]:= v;
      invcol[v]:= n; invcol[t]:= jj;
      for i:= 1 step 1 until m do
      begin t:= A[i, jj]; A[i, jj]:= A[i, n];
            A[i, n]:= t
      end
end;
```

```
                    if n = 2 then
                    begin length:= s; ABSFIXT (3, 0, length);
                          for t:= 1 step 1 until m do
                          route[row[t]]:= col[t]
                    end n = 2
                    else
                    begin ABSFIXT (3, 0, 1); ABSFIXT (3, 0, s);
                          PRINTTEXT (⟨via:⟩);
                          ABSFIXT (2, 0, u); ABSFIXT (2, 0, v);
                          ii:= u; jj:= v; l:= l + s;
                          for t:=invcol[ii] while t>n do ii:=row[t];
                          for t:=invrow[jj] while t>n do jj:=col[t];
                          i:= invcol[ii]; j:= invrow[jj];
                          A[j, i]:= A[j, i] + giant;
                          optimize (n - 1);
                          i:= invcol[ii]; j:= invrow[jj];
                          A[j, i]:= A[j, i] - giant;
                          if l < length then
                          begin A[n, n]:= A[n, n] + giant;
                                optimize (n);
                                i:= invrow[u]; j:= invcol[v];
                                A[i, j]:= A[i, j] - giant
                          end
                    end n > 2;
              end:  NLCR; SPACE ((m - n) × 5); PRINTTEXT (⟨exit⟩)
              end optimize;

              giant:= length:= 2 ↑ 24; s:= 0;
              for i:= 1 step 1 until m do
              begin row[i]:= col[i]:= invrow[i]:= invcol[i]:= i;
                    A[i, i]:= A[i, i] + giant
              end;
              optimize (m);
              NLCR; NLCR; NLCR;
              shortest route:= length
        end shortest route;


        for i:= 1 step 1 until n do
        for j:= 1 step 1 until n do A[i, j]:= read;
        t:= time;
        ABSFIXT (6, 0, shortest route (A, n, route));
        t:= time - t;
        PRINTTEXT (⟨via⟩); ABSFIXT (2, 0, 1); i:= 1;
        for i:= route[i] while i ≠ 1, 1 do
        begin PRSYM (65); ABSFIXT (2, 0, i) end;
        PRINTTEXT (⟨in⟩); ABSFIXT (6, 2, t); PRINTTEXT (⟨sec⟩)
    end
end
```