Opgave 2. De rivier

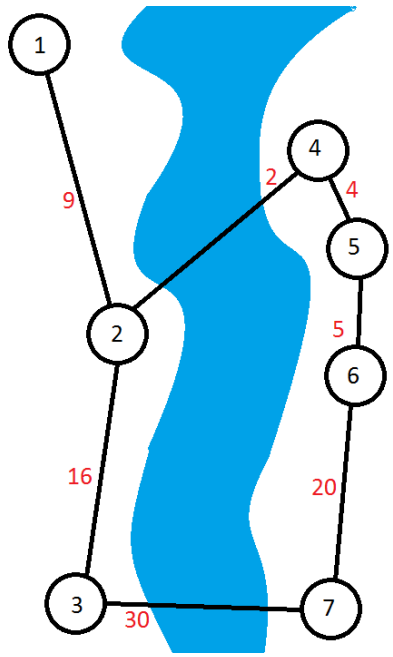
In deze opgave gaat het om verbindingen tussen plaatsen die aan weerszijden van een rivier liggen. De plaatsen aan beide kanten van de rivier zijn verbonden met één doorlopende weg. Er zijn enkele bruggen. De geschatte tijdsduur van de verbinding tussen twee aangrenzende plaatsen is bekend.

Je programma's lezen de informatie over de rivier van standard input.

- Op de eerste regel staat een getal L (0 < L < 100) dat aangeeft hoeveel plaatsen er links van de rivier liggen. Deze plaatsen zijn genummerd 1 tot en met L.
- Op de volgende L-1 regels staan de afstanden van de opvolgende plaatsen, eerst die tussen 1 en 2, dan die tussen 2 en 3, enzovoorts.
- Op de volgende regel staat een getal R (0 < R < 100) dat aangeeft hoeveel plaatsen er rechts van de rivier liggen. Deze plaatsen zijn genummerd L+1 tot en met L+R.
- Op de volgende R-1 regels staan de afstanden van de opvolgende plaatsen, eerst die tussen L+1 en L+2, enzovoorts.
- Op de volgende regel staat een getal B (0 < B < 20) dat aangeeft hoeveel bruggen er zijn tussen beide oevers.
- Op de laatste B regels staan achtereenvolgens het nummer van een plaats links van de rivier, het nummer van een plaats rechts van de rivier en de afstand bij deze verbinding, gescheiden door spaties. De bruggen worden op volgorde gegeven, beginnend bij de laagst genummerde plaatsen.

Bruggen snijden of kruisen elkaar niet. Ze kunnen wel bij dezelfde plaats aankomen of vertrekken; tussen twee plaatsen aan weerszijden van de rivier is maximaal één brug.

Alle afstanden in de opgave zijn onderling verschillend; de afstanden zijn positief en geen enkele afstand is groter dan 1000.



Voorbeeld invoer bij het plaatje hiernaast:

3	3 plaatsen links
9	Afstand 1-2
16	Afstand 2-3
4	4 plaatsen rechts
4	Afstand 4-5
5	Afstand 5-6
20	Afstand 6-7
2	2 bruggen
2 4 2	Verbinding van 2 naar 4
3 7 30	Verbinding van 3 naar 7

Dit voorbeeld wordt bij alle opgaven als voorbeeldinvoer gebruikt.

Overzicht:

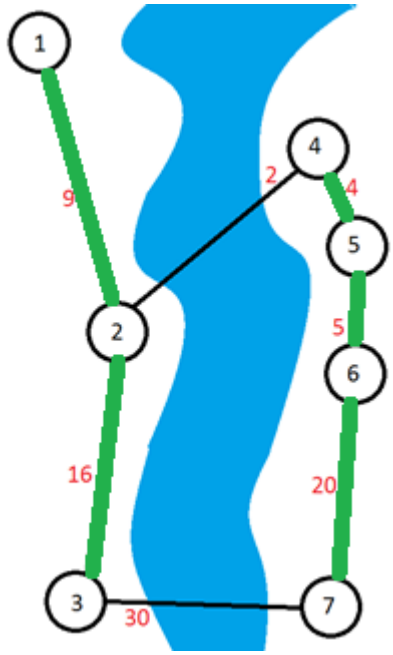
Opgave	Tijdlimiet	Testen	Per test	Totaal
2A	1	10	1	10
2B	1	10	2	20
2C	2	10	3	30
2D	2	10	4	40
2E	2	10	5	50
2F	2	10	5	50

2A: Langs de oevers

Schrijf een programma dat een rivier inleest van standard input. Het programma schrijft de lengte van de weg langs de linkeroever en de lengte van de weg langs de rechteroever naar één regel van standard output, gescheiden door een spatie.

Uitvoer bij het voorbeeld:

```
25 29
```



Oplossing van Erwin van Hunnik in Python 2 (commentaar is weggelaten):

```
OL=int(0)
L=int(input(""))
L=int(L-1)
while int(L)!=0:
    L=int(L-1)
    P=int(input(""))
    OL=int(OL)+int(P)
OR=int(0)
R=int(input(""))
R=int(R-1)
while int(R)!=0:
    R=int(R-1)
    P=int(input(""))
    OR=int(OR)+int(P)
print(str(OL)+" "+str(OR))
```

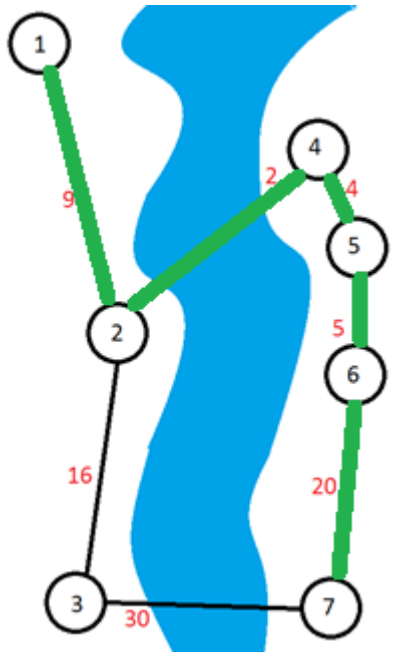
55 van de 68 deelnemers hadden hier de volledige score.

2B: Van linksboven naar rechtsonder

Schrijf een programma dat een rivier inleest van standard input. Het programma schrijft de lengte van de kortste verbinding van de eerste plaats aan de rechteroever naar de laatste plaats aan de rechteroever, waarbij de rivier maar eenmaal wordt overgestoken, naar één regel van standard output.

Uitvoer bij het voorbeeld:

```
40
```



Oplossing van Hugo Arkestein in C:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main ()
{
      int i,j;
      int minafstand=100000000, linkseind, rechtsstart, afstand;
      int afstandlinks=0,afstandrechts=0;
      int test, n;

      int hvll,hvlr;
      scanf("%d",&hvll);
      int afstanden[201];
      for (i=0;i<hvll-1;i++)
      {
            scanf("%d",&afstanden[i]);
      }
      scanf("%d",&hvlr);
      n=hvlr+hvll;
      i++;
```

```
      for (;i<n-1;i++)
      {
            scanf("%d",&afstanden[i]);
      }

      int aantalbruggen;
      scanf("%d",&aantalbruggen);
      int bruglinks[aantalbruggen];
      int brugrechts[aantalbruggen];

      int bruggenlengte[aantalbruggen];
      for (i=0;i<aantalbruggen;i++)
      {
            scanf("%d %d
%d",&bruglinks[i],&brugrechts[i],&bruggenlengte[i]);
      }

      for (j=0;j<aantalbruggen;j++)
      {
            afstandlinks=0;
            afstandrechts=0;
            linkseind=bruglinks[j];
            rechtsstart=brugrechts[j];
            for (i=0;i<linkseind-1;i++)
            {
                  afstandlinks+=afstanden[i];
            }
            for (i=rechtsstart-1;i<n-1;i++)
            {
                  afstandrechts+=afstanden[i];
            }
            afstand=afstandrechts+afstandlinks+bruggenlengte[j];
            if (afstand<minafstand) minafstand=afstand;
      }
      printf("%d",minafstand);

/*    for (i=0;i< ;i++)
      {
      }
      for (i=0;i< ;i++)
      {
            for (j=0;j< ;j++)
            {
            }
      } */
      return 0;
}
```

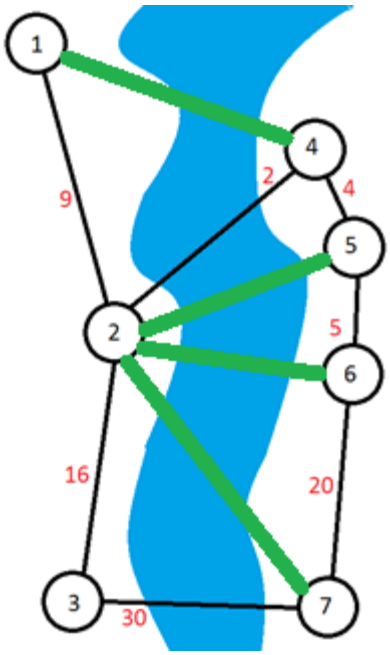
25 deelnemers hadden bij deze opgave de volledige score.

2C: Nieuwe bruggen

Schrijf een programma dat een rivier inleest van standard input. Het programma zoekt uit hoeveel extra bruggen kunnen worden gemaakt. Daarbij mag er tussen de beide plaatsen aan weerszijden van de brug nog geen andere brug lopen. Een nieuwe brug mag geen bestaande brug snijden of kruisen. Het programma telt het aantal mogelijke nieuwe bruggen en schrijft dat aantal naar één regel van standard output.

Uitvoer bij het voorbeeld:

```
4
```



Toelichting: Er zijn ook andere manieren denkbaar om de bruggen aan te leggen, maar meer dan vier bruggen bijbouwen zal nooit lukken.

Oplossing:

Je kunt bewijzen dat het aantal bruggen dat je maximaal kunt hebben tussen N en M plaatsen op beide oevers altijd N+M-1 is. Het antwoord op 2C is dan ook N+M-B-1.

Voor de liefhebbers: Dat kun je laten zien met de methode van volledige inductie. Zie voor een toelichting daarop bijvoorbeeld http://www.wiskundeleraar.nl/page3.asp?nummer=12272

Stap 1. Het maximale aantal bruggen tussen 1 plaats links en M plaatsen rechts is M.

Dat ligt voor de hand; iedere plaats rechts kun je verbinden met de ene plaats links, dus je hebt M bruggen.

Stap 2. Neem aan dat het maximale aantal bruggen tussen N plaatsen links en M plaatsen rechts N+M-1 is.

Dit noemen we de inductieveronderstelling. Je gaat dadelijk verder redeneren in de veronderstelling dat dit klopt.

Stap 3. Bekijk nu de situatie dat je 1 plaats toevoegt aan de linkeroever, terwijl er al een maximaal aantal bruggen ligt.

Die ene plaats ligt bijvoorbeeld aan het einde van de linkeroever. Dan kun je altijd een brug toevoegen naar de laatste plaats aan de rechteroever. Meer mogelijkheden zijn er niet, want als er bij die laatste plaats nog geen brug aan zou komen, was niet het maximaal aantal bruggen gebruikt. Je kunt dus één brug toevoegen.
Als de ene plaats extra links tussen twee bestaande plaatsen komt, dan gaan de bruggen van de beide buurplaatsen naar dezelfde plaats aan de overkant. Als dat niet zo zou zijn, had er nog een brug bij gekund, maar het maximale aantal bruggen was al gebruikt. Dus kan ik precies één brug toevoegen, die naar dezelfde plaats aan de overkant gaat als de bruggen van de buren.

Conclusie:

De bewering dat het aantal bruggen van N plaatsen links naar M plaatsen rechts maximaal N+M-1 is is waar. In stap 1 hebben we gezien dat het waar is voor N=0. De combinatie van stap 1 en de aanname van stap 2 geeft via stap 3 dat het dan ook waar is voor N=1. Daarna kun je stap 2 en 3 blijven combineren, zodat je het voor iedere N kun bewijzen.

Er waren 11 deelnemers met een volledige score.

De eerste die met een oplossing kwam was Siebe Verheijen. Hier zijn programma in Java:

```
import java.util.ArrayList;
import java.util.Scanner;

public class eenalpha {
    public static void main (String[] args)
    /**
     * Array = Object[] = Deze haakjes
     * ArrayList = List<Object> = list.get(nummer vanaf 0)
     */
    {
        Scanner s = new Scanner(System.in);
        int L = Integer.parseInt(s.nextLine());
        for (int i=1; i<=L-1; i=i+1) {
            int t = Integer.parseInt(s.nextLine());
        }
        int R = Integer.parseInt(s.nextLine());

        for (int i=1; i<=R-1; i=i+1) {
            int t = Integer.parseInt(s.nextLine());
        }

        int B = Integer.parseInt(s.nextLine());

        System.out.println(R+L-B-1);



    }
}
```

2D: Niemands naaste buur

De **naaste buur** van een plaats is de plaats die er het dichtst bij ligt. In het voorbeeld: de plaatsen 2 en 4 zijn elkaars naaste buur, 2 is ook de naaste buur van 1, maar 1 is niet de naaste buur van 2 (dat is 4 namelijk al).

Schrijf een programma dat een rivier inleest van standard input. Het programma schrijft naar standard output eerst een regel met een getal N, dat aangeeft dat er N plaatsen zijn die van niemand de naaste buur zijn. Vervolgens komen er N regels met de plaatsen die van niemand naaste buur zijn, oplopend geordend.

Uitvoer bij het voorbeeld:

```
3
1
3
7
```

Toelichting:

Plaats 1 heeft als naaste buur plaats 2 met afstand 9
Plaats 2 heeft als naaste buur plaats 4 met afstand 2
Plaats 3 heeft als naaste buur plaats 2 met afstand 16
Plaats 4 heeft als naaste buur plaats 2 met afstand 2
Plaats 5 heeft als naaste buur plaats 4 met afstand 4
Plaats 6 heeft als naaste buur plaats 5 met afstand 5
Plaats 7 heeft als naaste buur plaats 6 met afstand 20

Geen enkele plaats heeft plaats 1, 3 of 7 als naaste buur.

Oplossing van Tom Udding in PHP:

```
<?php
$plaatsenLinks = $plaatsenRechts = $aantalBruggen = $afstand = 0;
$plaatsLinks = $plaatsRechts = 0;
$afstandenLinks = $afstandenRechts = array();

$totaal = array();
$bruggen = array();
$buren = array();

fscanf(STDIN, "%d\n", $plaatsenLinks);

for ($i = 0; $i < ($plaatsenLinks - 1); $i++) {
    fscanf(STDIN, "%d\n", $afstand);
    array_push($afstandenLinks, array("van" => $i+1, "naar" => $i+2,
"afstand" => $afstand));
    array_push($totaal, array("van" => $i+1, "naar" => $i+2,
"afstand" => $afstand));
}
```

```
fscanf(STDIN, "%d\n", $plaatsenRechts);

for ($i = 0; $i < ($plaatsenRechts - 1); $i++) {
    fscanf(STDIN, "%d\n", $afstand);
    array_push($afstandenRechts, array("van" => $i+$plaatsenLinks+1,
"naar" => $i+$plaatsenLinks+2, "afstand" => $afstand));
    array_push($totaal, array("van" => $i+$plaatsenLinks+1, "naar"
=> $i+$plaatsenLinks+2, "afstand" => $afstand));
}

fscanf(STDIN, "%d\n", $aantalBruggen);

for ($i = 0; $i < $aantalBruggen; $i++) {
    fscanf(STDIN, "%d %d %d\n", $plaatsLinks, $plaatsRechts,
$afstand);
    array_push($bruggen, array("van" => $plaatsLinks, "naar" =>
$plaatsRechts, "afstand" => $afstand));
    array_push($totaal, array("van" => $plaatsLinks, "naar" =>
$plaatsRechts, "afstand" => $afstand));
}

//print_r($afstandenLinks);
//print_r($afstandenRechts);
//print_r($bruggen);
//print_r($totaal);

$dorpen = array();
$dorpen2 = array();

for ($i = 0; $i < ($plaatsenLinks + $plaatsenRechts); $i++) {
    $temp = array();
    foreach ($totaal as $plaats) {
        if ($plaats['van'] == ($i+1)) {
            $temp[$plaats['naar']] = $plaats['afstand'];
        } else if ($plaats['naar'] == ($i+1)) {
            $temp[$plaats['van']] = $plaats['afstand'];
        }
    }
    $afstand = min($temp);
    $dorpen[] = array_search($afstand, $temp);
    //echo "Plaats ".($i+1)." heeft als naaste buur {$dorp} met
afstand {$afstand}\n";
}

for ($i = 1; $i <= ($plaatsenLinks + $plaatsenRechts); $i++) {
    $dorpen2[] = $i;
}

$output = array();

for ($i = 1; $i <= ($plaatsenLinks + $plaatsenRechts); $i++) {
    if (!in_array($i, $dorpen)) {
        $output[] = $i;
    }
}
```

```
echo count($output)."\n";
foreach ($output as $out) {
    echo $out."\n";
}
?>
```

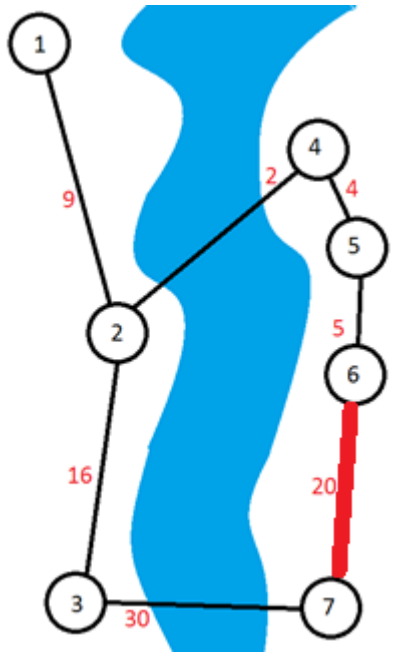
Er waren 21 deelnemers met een volledige oplossing.

2E: Grootste wegafsluiting

Schrijf een programma dat een rivier inleest van standard input. Het programma zoekt naar de wegen die langs beide oevers kunnen worden afgesloten voor onderhoud, zodat alle plaatsen wel onderling verbonden blijven. De bruggen worden niet afgesloten. Je programma berekent de som van de verbindingstijden langs de afgesloten wegdelen en schrijft die som naar standard output.

Uitvoer bij het voorbeeld:

```
20
```



Toelichting: Als je de weg tussen 6 en 7 weglaat kun je nog steeds overal komen. Als je nog een weg schrapt is er minstens één plaats onbereikbaar geworden vanuit andere plaatsen.

Oplossing van Andor Michels in C++:

```
#include<iostream>
using namespace std;

int L;
int* afstandenL;
int R;
int* afstandenR;
int B;
int* bruggen;
int* bruggenL;
int* bruggenR;

int main()
{
	cin >> L;
	afstandenL = new int[L-1];
	for (int i = 0; i < L-1; i++)
```

```
	{
		cin >> afstandenL[i];
	}
	cin >> R;
	afstandenR = new int[R-1];
	for (int i = 0; i < R-1; i++)
	{
		cin >> afstandenR[i];
	}

	cin >> B;
	bruggen = new int[B];
	bruggenL = new int[B];
	bruggenR = new int[B];
	for (int i = 0; i < B; i++)
	{
		cin >> bruggenL[i] >> bruggenR[i] >> bruggen[i];
	}

	//logic
	int maxA = 0;
	for (int i = 1; i < B; i++)
	{
		int maxWeg = 0;
		for (int j = bruggenL[i-1]-1; j < bruggenL[i]-1; j++)
		{
			maxWeg = max(maxWeg, afstandenL[j]);
		}
		for (int j = bruggenR[i-1]-1-L; j < bruggenR[i]-1-L; j++)
		{
			maxWeg = max(maxWeg, afstandenR[j]);
		}
		maxA += maxWeg;
	}
	cout << maxA;

	return 0;
}
```

Christel van Diepen, Arend Mellendijk, Reijer van Harten, Ernest van Wijland, Wietze Koops en Andor Michels hadden een correcte oplossing.

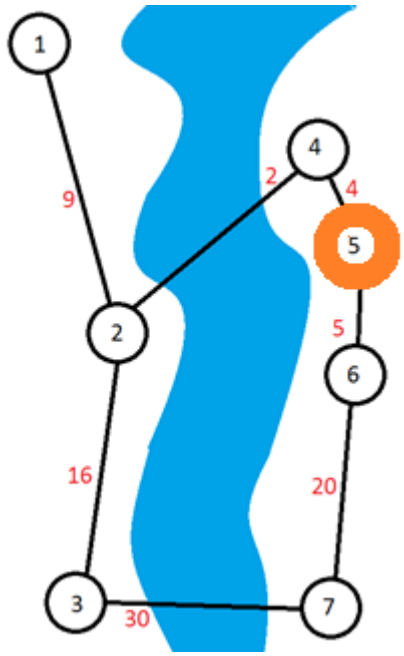
Een mooie manier om dit probleem op te lossen is het algoritme van Kruskal. Zie bijvoorbeeld https://nl.wikipedia.org/wiki/Kruskals_algoritme

2F: Centrum

Schrijf een programma dat een rivier inleest van standard input. Het programma schrijft het nummer van de plaats die de kleinste maximale afstand heeft naar alle andere plaatsen naar één regel van standard output. Als dat kleinste maximum voor verschillende plaatsen hetzelfde is, voert je programma het laagste plaatsnummer uit.

Uitvoer bij het voorbeeld:

```
5
```



Toelichting:

Vanaf plaats 1 is plaats 7 het verst weg, afstand 40
Vanaf plaats 2 is plaats 7 het verst weg, afstand 31
Vanaf plaats 3 is plaats 7 het verst weg, afstand 30
Vanaf plaats 4 is plaats 7 het verst weg, afstand 29
Vanaf plaats 5 is plaats 7 het verst weg, afstand 25
Vanaf plaats 6 is plaats 3 het verst weg, afstand 27
Vanaf plaats 7 is plaats 1 het verst weg, afstand 40

Oplossing van Christel van Diepen in C++:

```
#include <iostream>
#include <string>

using namespace std;

int L, R, B;
int dist[200][200];
int minDis[200];
bool naast[200];
bool justDone[200];
```

```
int dis[200];

void rdInput()
{
    int i = 1;
    cin >> L;
    for (i = 1; i < L; i++) {
        cin >> dist[i][i+1];
        dist[i+1][i] = dist[i][i+1];
    }
    cin >> R;
    for (i = L + 1 ; i < R + L; i++) {
        cin >> dist[i][i+1];
        dist[i+1][i] = dist[i][i+1];
    }
    cin >> B;
    for (int j = 0; j < B; j++) {
        int x, y;
        cin >> x >> y;
        cin >> dist[x][y];
        dist[y][x] = dist[x][y];
    }
}

void A()
{
    long l = 0, r = 0;
    for (int i = 1; i < L; i++) {
        l += dist[i][i+1];
    }
    for (int i = L + 1; i < R + L; i++) {
        r += dist[i][i+1];
    }

    cout << l << ' ' << r;
}

void Bee()
{
    for (int i = 1; i <= L + R; i++) {
        for (int j = i + 1; j <= L + R; j++) {
            if (dist[i][j] != 0) {
                if (minDis[j] == 0) minDis[j] = minDis[i] +
dist[i][j];
                else minDis[j] = min(minDis[j], minDis[i] +
dist[i][j]);
            }
        }
    }

    cout << minDis[L+R];
}

int nst(int n)
{
    int d = 1001;
```

```
    int dI = 0;
    for (int i = 1; i <= L + R; i++) {
        if (dist[i][n] != 0 && dist[i][n] < d) {
            d = dist[i][n];
            dI = i;
        }
    }

    return dI;
}

void D()
{
    for (int i = 1; i <= L + R; i++) {
        naast[nst(i)] = true;
    }
    int c = 0;
    for (int i = 1; i <= L + R; i++) {
        if (!naast[i]) c++;
    }
    cout << c << endl;
    for (int i = 1; i <= L + R; i++) {
        if (!naast[i]) {
            cout << i << endl;
        }
    }
}

int maxDis(int n)
{
    for (int i = 0; i <= L + R; i++) dis[i] = 0;
    justDone[n] = true;
    int maxD = 0;
    int jd = 0;
    do {
        jd = 0;
        for (int i = 1; i <= L + R; i++) {
            if (justDone[i]) {
                for (int j = 1; j <= L + R; j++) {
                    //if (n == 2) cerr << i << ' ' << j << endl;
                    if (dist[i][j] != 0 && j != n && (dis[j] == 0 ||
(dist[i][j] + dis[i] < dis[j]))) {
                        dis[j] = dist[i][j] + dis[i];
                        justDone[j] = true;
                        jd++;
                    }
                }
                justDone[i] = false;
            }
        }
    } while (jd > 0);
    for (int i = 1; i <= L + R; i++) {
        //if (n == 2) cerr << i << ' ' << dis[i] << endl;
        maxD = max(maxD, dis[i]);
    }
    return maxD;
```

```
}

void F()
{
    int c = 1;
    int m = maxDis(1);
    for (int i = 2; i <= L + R; i++) {
        //cerr << c << ' ' << m << endl;
        int ma = maxDis(i);
        //cerr << i << ' ' << ma << endl;
        if (ma < m) {
            //cerr << i << ' ' << ma << endl;
            c = i;
            m = ma;
        }
    }

    cout << c;
}

int main()
{
    rdInput();

    F();

    return 0;
}
```

Christel van Diepen, Jelmer Hinssen, Ernest van Wijland en Alex Keizer hebben de volledige score behaald voor deze deelopgave.

Deze opgave kun je oplossen door gebruikt te maken van het algoritme van Dijkstra, een beroemde Nederlandse bijdrage aan de informatica. Zie bijvoorbeeld de beschrijving op http://www.wiskundemeisjes.nl/20090420/ode-aan-dijkstra/