Oplossingenbundel Coma 2009

PRIME, 12 oktober 2009

1 Stamnummers

a: 25	b: 1101	c: 48514	d: 196165

Hier kon je bijvoorbeeld voor elke modulus m gaan kijken of er twee stamnummers gelijk waren modulo m. Sneller dan daartoe alle $n(n-1)$ koppels vergelijken is een lijst bijhouden met alle restklassen modulo n en deze aanvullen totdat een restklasse twee elementen zou tellen.

We lopen daartoe alle moduli af, in stijgende volgorde, startend van $m = n$ (voor $m < n$ zullen er immers altijd twee studenten in dezelfde restklasse modulo m zitten.

Dit idee is uitgewerkt in onderstaande java-code.

```
package coma2009;
import java.io.*;

public class Modulo {
    int n; //grootte studentengroep
    int[] ids; //de huidige ID's
    int m; //de ideale modulus

    public Modulo (String file, int n) {
        this.n=n;
        ids = new int[n];
        try {
            BufferedReader reader = new BufferedReader(new FileReader(file));
            for (int i=0; i<n; i++) ids[i] = Integer.parseInt(reader.readLine());
        } catch (IOException e) {
            System.out.println(e);
        }
        m=n-1;
        boolean gevonden = false;
        while(!gevonden) {
            boolean dubbels=false;
            boolean[] algebruikt = new boolean[++m];
            for (int i=0; i<n && !dubbels; i++)
                if (algebruikt[ids[i]%m]) dubbels=true;
                else algebruikt[ids[i]%m]=true;
            if (!dubbels) gevonden=true;
      }
    }

    public static void main (String[] args) {
        Modulo test;
        test = new Modulo("C://stam10.txt",10);
        System.out.println(test.m);
        test = new Modulo("C://stam100.txt",100);
        System.out.println(test.m);
        test = new Modulo("C://stam1000.txt",1000);
        System.out.println(test.m);
        test = new Modulo("C://stam2009.txt",2009);
        System.out.println(test.m);
    }

}
```

2 Cosinalië

a: 28	b: 536	c: 120006	d: 18666676

Noteren we $T_n\colon \cos(nx) \mapsto \cos x$ dan geldt dat

$$\cos((n+1)\alpha) + \cos((n-1)\alpha) = 2\cos(\alpha)\cos(n\alpha),$$

dus $T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$. De betere googlegebruiker had wellicht al ontdekt dat dit de naam Chebyshevpolynoom draagt.

Vermits we enkel de coëfficiënt van x^9 nodig hebben, volstaat het een rijtje aan te maken met de coëfficiënten van $x^0, x^1, \ldots, x^9$ en hierop bovenstaande iteratieformule toe te passen, telkens modulo het jaartal. Dit idee is uitgewerkt in onderstaande java-code.

```
package coma2009;

public class Chebyshev {
    int result;

    public int mod(int a, int p) { //zorgt dat mod(a,p)>=0
        return ((a%p)+p)%p;
    }

    public Chebyshev(int n) {
        int[] T_0 = new int[10];
        int[] T_1 = new int[10];
        T_0[0]=1;
        T_1[1]=1;
        for (int i=1; i<n; i++) {
            int[] temp = new int[10];
            temp[0]=mod(-T_0[0],n+1);
            for (int j=1; j<10; j++)
                temp[j]=mod(2*T_1[j-1]-T_0[j],n+1);
            T_0=T_1;
            T_1=temp;
        }
        result = T_1[9];
    }

    public static void main (String[] args) {
        Chebyshev cheb;
        cheb = new Chebyshev(29);
        System.out.println(cheb.result);
        cheb = new Chebyshev(2009);
        System.out.println(cheb.result);
        cheb = new Chebyshev(200009);
        System.out.println(cheb.result);
        cheb = new Chebyshev(20000009);
        System.out.println(cheb.result);
    }
}
```

3 Pixarkaarten

a: 2	b: 160	c: 15428	d: 1542602

De truc hier is om langs achter te beginnen: als je weet wat de maximale winst is vanuit punten k+1,k+2,..., dan kun je hieruit snel de maximale winst in punt k berekenen door eenmalig alle keuzes af te lopen (dit zijn er meestal maar enkele). Als je langs voor zou beginnen geeft elke keuze aanleiding tot een hoop nieuwe keuzes, zodat het aantal hier héél snel stijgt.

```
package coma2009;
import java.io.*;
public class Pixar {
    public int n;
    public int[] waarden; //1-dim daar enkel som rol speelt
    public int[] maxwinstvanaf; //jouw max winst vanaf hier
    public int[] aantalB; //hoeveel kaarten pakt B vanaf hier
    public int[] somB; //de totaalsom van die kaarten
    int threshold = 2009;
    int compstop = 1830; //B doet enkel verder while <1830

    public Pixar(String file, int n) {
        this.n=n;
        this.waarden = new int[n];
        try {
            BufferedReader reader = new BufferedReader(new FileReader(file));
            for (int i=0; i<n; i++) {
                String[] s = reader.readLine().split(" ");
                waarden[i] = Integer.parseInt(s[0])+Integer.parseInt(s[1])+Integer.parseInt(s[2])+Integer.parseInt(s[3]);
            }
        } catch (IOException e) {
            System.out.println(e);
        }
        maxwinstvanaf = new int[n+1];
        maxwinstvanaf[n]=0; maxwinstvanaf[n-1]=2;
        aantalB = new int[n+1];
        somB = new int[n+1];

        //bereken nu aantal kaarten dat tegenstander die beurt zal trekken en de totale som in dat geval
        for (int i=0; i<n; i++) {
            int som=0;
            int aantal=0;
            while (som<compstop && i+aantal<n) {
                som+=waarden[i+aantal];
                aantal++;
            }
            aantalB[i]=aantal;
            somB[i]=som;
        }

        //berekent max winst in ieder startpunt
        for (int i=n-1; i>=0; i--) {
            int som=0; int aantal=0;
            do {
                int dezewinst = maxwinstvanaf[i+aantal+aantalB[i+aantal]]+win(som,somB[i+aantal]);
                maxwinstvanaf[i]=Math.max(maxwinstvanaf[i],dezewinst);
                som+=waarden[i+aantal];
                aantal++;
            } while (som<=threshold && i+aantal<n);
            int dezewinst = maxwinstvanaf[i+aantal+aantalB[i+aantal]] +win(som,somB[i+aantal]);
            maxwinstvanaf[i]=Math.max(maxwinstvanaf[i],dezewinst);
        }
    }

    public int win(int score1, int score2) {
        int tot=0;
        if (score1>threshold) tot-=2;
        if (score2>threshold) tot+=2;
        if (score1<=threshold && score2<=threshold) {
            if (score1>score2) tot+=2;
            if (score1<score2) tot-=2;
        }
        return tot;
    }

    public static void main (String[] args) {
        Pixar spel;
        spel = new Pixar("C://pixar10.txt",10); System.out.println("Max winst: "+spel.maxwinstvanaf[0]);
        spel = new Pixar("C://pixar1000.txt",1000); System.out.println("Max winst: "+spel.maxwinstvanaf[0]);
        spel = new Pixar("C://pixar100000.txt",100000); System.out.println("Max winst: "+spel.maxwinstvanaf[0]);
        spel = new Pixar("C://pixar10000000.txt",10000000); System.out.println("Max winst: "+spel.maxwinstvanaf[0]);
    }
}
```

4 Tunneltje graven

a: 58	b: 557	c: 5632	d: 11136

Een mogelijke oplosssing bestond erin een matrix aan te maken die van ieder punt de snelste tijd ernaartoe vanuit de linkerbovenhoek bijhoudt. Iedere keer als deze afstand verlaagt, wordt gekeken of dat ook voor de buren een nieuw minimum meebrengt. De snelste implementatie hiervan is met een wachtrij, al zijn ook tragere maar meer elementaire oplossingen (bijvoorbeeld een boolean matrix) mogelijk. Dit idee is uitgewerkt in onderstaande java-code.

```
package coma2009;
import java.io.*;
import java.util.*;

public class Tunnel {
    public int[][] bestepad;
    public int[][] gewichten;
    Queue<int[]> wachtrij;
    int n; //n=m
    int minkost;

    public Tunnel(String file, int n) {
        this.n=n;
        gewichten = new int[n][n];
        try {
            BufferedReader reader = new BufferedReader(new FileReader(file));
            for (int i=0; i<n; i++) {
                String[] s = reader.readLine().split(" ");
                for (int j=0; j<s.length; j++) {
                    gewichten[i][j]=Integer.parseInt(s[j]);
                }
            }
        } catch (IOException e) {
            System.out.println(e);
        }

        bestepad = new int[n][n];
        for (int i=0; i<n; i++) for (int j=0; j<n; j++) bestepad[i][j]=9999999;
        bestepad[0][0] = gewichten[0][0];
        wachtrij = new LinkedList<int[]>();
        wachtrij.add(new int[]{0,0});
        while (!wachtrij.isEmpty()) process(wachtrij.poll());
        minkost = bestepad[n-1][n-1];
    }

    public void process(int[] coor) {
        int x = coor[0]; int y = coor[1];
        if (x!=0) if (bestepad[x-1][y]>gewichten[x-1][y]+bestepad[x][y]){
            bestepad[x-1][y]=bestepad[x][y]+gewichten[x-1][y];
            wachtrij.add(new int[]{x-1,y});
        }
        if (y!=0) if (bestepad[x][y-1]>gewichten[x][y-1]+bestepad[x][y]){
            bestepad[x][y-1]=bestepad[x][y]+gewichten[x][y-1];
            wachtrij.add(new int[]{x,y-1});
        }
        if (x!=n-1) if (bestepad[x+1][y]>gewichten[x+1][y]+bestepad[x][y]){
            bestepad[x+1][y]=bestepad[x][y]+gewichten[x+1][y];
            wachtrij.add(new int[]{x+1,y});
        }
        if (y!=n-1) if (bestepad[x][y+1]>gewichten[x][y+1]+bestepad[x][y]){
            bestepad[x][y+1]=bestepad[x][y]+gewichten[x][y+1];
            wachtrij.add(new int[]{x,y+1});
        }
    }

    public static void main (String[] args) {
        Tunnel t;
        t = new Tunnel("C://tunnel10.txt",10); System.out.println(t.minkost);
        t = new Tunnel("C://tunnel100.txt",100); System.out.println(t.minkost);
        t = new Tunnel("C://tunnel1000.txt",1000); System.out.println(t.minkost);
        t = new Tunnel("C://tunnel2009.txt",2009); System.out.println(t.minkost);
    }
}
```

Ook andere aanpakken (zoals de tunnel bezien als graaf en dan het algoritme van Dijkstra toepassen) waren hier mogelijk, al bleek deze toch net iets sneller te gaan. Ook optimalisaties zijn nog mogelijk, maar voor de eenvoudigheid hebben we ze niet aangebracht.