```
Alp3 Übung 4 - Musterlösung von Jochen Brunhorn und Anne Driemel, 17.02.04

Aufgabe 23

public class Hash{
        // Klasse berechnet bei Eingabe der Anzahl als Parameter
        // eine Reihe von Zahlen, deren paarweise Summen
        // alle unterschiedlich sind und in einem optimalen Intervall liegen

        //Konstanten
        int ersterWertimIntervall = 0;

        //Felder
        Summand Summanden[];    // Summanden mit Summenarrays
        int anzahl;             // Anzahl der Summanden
        int Intervallgroesse;

        //Konstruktor
        Hash(int n){
                anzahl = n;
                Summanden = new Summand[n];
                for (int i=0; i<n; i++) Summanden[i] = new Summand(i);
        }

        void init(){

                // nur, um den ersten Hash zu initialisieren
                // und mit einer möglichst kleine maxSum am anfang
                // den Suchbereich einzuschränken

                Summanden[0].wert = ersterWertimIntervall;
                System.out.println(Summanden[0].wert);

                // restliche berechnen und ausgeben
                for(int i=1; i<anzahl; i++){
                        boolean gefunden = false;               // richtiger Wert noch nicht
                                                                // gefunden
                        for(int j = 1; !gefunden ; j++){        // solange der nicht gefunden,
                                                                // wird gesucht
                                boolean abbruch = false;        // es wurde nicht vorzeitig
                                                                // abgebrochen
                                int aktuellerWert = Summanden[i-1].wert + j;

                                // Summen-array füllen und prüfen
                                // also alle bis dahin geschriebenen Summe
                                for (int k=0; k < i && !abbruch; k++){
                                        int aktuelleSumme = aktuellerWert + Summanden[k].wert;
                                        Summanden[i].Summen[k] = aktuelleSumme;

                                        //die aktuelle Summe mit allen anderen Summen vergleichen
                                        // bei Übereinstimmung ->abbruch
                                        for (int l = 1; l < i && !abbruch; l++)
                                                for (int m = 0; m < l && !abbruch ; m++){
                                                if (aktuelleSumme == Summanden[l].Summen[m])
                                                                abbruch = true;
                                                }
                                }
                                //wenn nicht vorzeitig abgebrochen wurde, erfüllt der wert die
                                // Bedingungen
                                if (!abbruch){
                                        gefunden = true;
                                        System.out.println(aktuellerWert);
                                        Summanden[i].wert = aktuellerWert;
                                }
                        }
                }
                this.Intervallgroesse();        //arbeit auf dem objekt -->globale
                                                // Intervallgroesse aktualisieren
        }

        Hash optimiere(){

                Hash best = this;
                int Intervall = this.Intervallgroesse();

                System.out.println("Ausgangspunkt: kleinstes Summenintervall <= " +
                                        this.Summanden[this.anzahl-1].Summen[this.anzahl-2]);
                System.out.println("---> systematisch suchen ");

                for (int i = this.anzahl - 1; this.relevantesZ(i, Intervall); i++){
                        Hash temp = new Hash(this.anzahl);
                        if((temp = this.createBetterHashFor(i, Intervall)) != null)
                                if (temp.Intervallgroesse() < best.Intervallgroesse()) {
                                        best = temp;
                                        Intervall = best.Intervallgroesse;
                                }
                        }
                        System.out.println("\n---------------------------\n");

                        System.out.println("..ein Hash, optimal fuer "+this.anzahl+" Stellen");
                        best.show();
                        System.out.println();
                        return best;
                }

                Hash createBetterHashFor(int z, int x){
                        // wird von  der methode optimiere auf einem Hashobjekt aufgerufen, bei dem nur
                        // der erste und der letzte summand feststehen
                        // z ist der letzte Summand
                        // geht alle Möglichen Kombinationen von Summanden in diesem Bereich durch und
                        // testet, ob hash-fähig
                        // wobei bei jeder gefundenen Lösung der Suchbereich eingeschränkt wird
                        // durch grenzwertY, y soll der vorletzte Summand sein

                        System.out.print("\nVielleicht ein besserer Hash in [0.. "+z+"]?");

                        Hash best = this;
                        boolean gefunden = false;
                        int Intervall = x;

                        //zb. 1110 für 3 Zahlen zwischen [0..5]
                        long größteRelevanteBinaerzahl = (long) (Math.pow(2,z-1) -
                                                                Math.pow(2,z-this.anzahl)+1);
                        //nur für die Punkte
                        int i=0;

                        System.out.println("  ( " +
                                        this.moeglichkeiten(z-1,this.anzahl-2)+" Moeglichkeiten )");

                        // erste Möglichkeit generieren
                        Binaer binaer = new Binaer(this.anzahl - 2);

                        // ..  und testen
                        while(binaer.wert <= größteRelevanteBinaerzahl){

                                //Fortschritt (intuitiv)
                                if(++i % 100000 == 0) System.out.print(".");

                                Hash temp = new Hash(this.anzahl);
                                temp.init(binaer, z);

                                //wenn besser als die vorigen, übernehme und gib aus
                                if (temp.Intervallgroesse() < Intervall){

                                        //probiere, ob Hash...
                                        if(temp.isHash()) {
                                                System.out.println("\n \n  Hash gefunden:");
                                                gefunden = true;
                                                best = temp;
                                                best.show();
                                                Intervall = best.Intervallgroesse();
                                                System.out.println("\n.. und weitersuchen");
                                        }
                                }
                                //alle weiteren relevanten Möglichkeiten generieren
                                binaer.naechste();
                        }
                        if(!gefunden) System.out.println("..keinen gefunden");
                        return best;
                }

                void init(Binaer binaer, int z){
                        // bei dieser initialisierung werden noch keine Summen zugeteilt

                        //anfang und ende:
```

```
        this.Summanden[0].wert = ersterWertimIntervall;
        this.Summanden[this.anzahl-1].wert = z;

        //rest:
        long  t = binaer.wert;
        int indexBin = 1;
        int indexSumm = 1;

        while(t!=0){
                //wenn t ungerade ist, steht in der binärdarstellung an dieser Stelle
                // eine '1'
                if (t%2 == 1) this.Summanden[indexSumm++].wert = indexBin;
                t/=2;
                indexBin++;
        }
}

boolean isHash(){
        // testet ein Hashobjekt, in dessen Summanden alle feststehen
        // indem sie von hinten angefangen der reihe nach alle paarweisen Summen
        // in die Summenarrays der summanden schreibt und gleichzeitig testet
        // ob die Summe doppelt vorkommt

        if (this == null) return false;
        else{
                boolean abbruch = false;

                for(int j1 = this.anzahl-1; j1 >0 && !abbruch; j1--){

                        // alle bis dahin entstehenden Summen schreiben (von hinten)
                        for (int i1 = j1 - 1; i1 >= 0 &&  !abbruch; i1--){
                                int aktuelleSumme = this.Summanden[j1].wert +
                                                        this.Summanden[i1].wert;
                                this.Summanden[j1].Summen[i1] = aktuelleSumme;

                                //..und mit allen schon geschriebenen Summen vergleichen
                                // bei Übereinstimmung ->abbruch
                                for (int j2 = this.anzahl - 1; j2 > j1 && !abbruch; j2--)
                                        for (int i2 = j2 - 1; i2 >= 0 && !abbruch; i2--){
                                        if (aktuelleSumme ==
                                                        this.Summanden[j2].Summen[i2])
                                                                abbruch = true;
                                        }
                        }
                }
                return !abbruch;
        }
}

boolean relevantesZ(int z, int x){
        return (z + this.anzahl - 2) < x;
}

int Intervallgroesse(){
        return this.Intervallgroesse = this.Summanden[this.anzahl-1].wert +
                                        this.Summanden[this.anzahl-2].wert -
                                        this.Summanden[1].wert +
                                        Summanden[0].wert + 1;
}

//Ausgabefunktion
void show(){
        System.out.println();
        System.out.print("   [");
        for (int i=0; i<this.anzahl; i++) System.out.print(" "+Summanden[i].wert+" ");
        System.out.println("]");
        System.out.println("\n  Intervall: "+this.Summanden[1].Summen[0]+".. "+
                        this.Summanden[this.anzahl-1].Summen[this.anzahl-2]);
        System.out.println("  Groesse:   "+this.Intervallgroesse());
}

int moeglichkeiten(int n, int k){
        // Errechnet die Binomialkoeffizienten
        long KFak = 1;
        long NfallendeFakK = 1;

        long i = 1;
        long j = n;

                while(i <= k) {NfallendeFakK*=j--; i++;}

                i = 1;
                while(i <= k) {KFak*=i; i++;}

                return  (int) (NfallendeFakK/KFak);
        }

        public static void main(String[] args){
                int n = Integer.parseInt(args[0]);
                Hash test = new Hash(n);
                test.init();
                Hash loesung = test.optimiere();
        }

}

class Summand{
        int wert, Summen[];

        // in dem Summenarray jedes Summanden sollen nur
        // die Summen mit seinen Vorgängern stehen
        Summand(int n){
                Summen = new int[n];
        }
}


class Binaer{

                // initialisiert ein Objekt Binaer mit der Anzahl
                // der gesuchten Summanden als Parameter z , dass zuerst
                // die kleinste Binaerzahl mit z einsen und mit jedem
                // Aufruf von .naechste() alle weiteren Binaerzahlen
                // mit z einsen generiert


        long wert;
        int einsen;             // anzahl der einsen, nur für initialisierung

        Binaer(int n){
                this.einsen = n;
                this.wert = (long) Math.pow(2,n) - 1;
        }

        void naechste(){                // setzt sich selbst auf die nächstgrößere Zahl mit
                                        // der selben Anzahl von einsen in der Binärdarstellung

                //ermittlt Größe des ersten 1-er Blocks und des ersten 0-er Blocks (von hinten)
                int anzNullen = 0;
                int anzEinsen = 0;
                boolean ersteEinsGefunden = false;
                boolean letzteEinsGefunden = false;

                long t = this.wert;

                while(t > 0 && !letzteEinsGefunden){
                        if (t % 2 == 0){
                                if(!ersteEinsGefunden)  anzNullen++;
                                else letzteEinsGefunden = true;
                        }
                        else {
                                anzEinsen++;
                                ersteEinsGefunden = true;       //immer wieder auf true setzen
                        }
                        t/=2;
                }

                //neuen Wert errechnen:
                long alterWert = this.wert;
                this.wert = alterWert+(int)Math.pow(2,anzEinsen-1)+ (int)Math.pow(2,anzNullen)-1;
        }
}

Ergebnisse:

   [ 0  1  2  4  7 ]
```

```
  Intervall: 1.. 11
  Groesse:   11


   [ 0  1  2  4  7  12 ]
  Intervall: 1.. 19
  Groesse:   19

   [ 0  5  7  9  10  13  21 ]
  Intervall: 5.. 34
  Groesse:   30

   [ 0  1  2  4  8  14  19  24 ]
  Intervall: 1.. 43
  Groesse:   43
```

bei 9 und 10 Stellen braucht das Programm etwas länger..

Aufgabe 24 b

Wenn man die Elemente durchnummeriert
 und sie im array so anordnet:

```
 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ...
2,1|3,2|3,1|4,3|4,2|4,1|5,4|5,3|5,2|5,1| ...
```

bekommt man den index des Paares i,j
 durch die Formel:

index(i,j) = j*(j - 1)/2 - i , für i < j

Somit wird kein Speicherplatz verschwendet
und man kann mit Hilfe der geschlossene Formel
in konstanter Zeit auf das gesuchte Paar
zugreifen.

Aufgabe 27

t(n) = t(n-1) + t(n-2)
t(1) = 1 ist der erste Summand
t(2) = 2 ist der zweite Summand

daraus folgt t(3)= 3, t(4)= 5, t(6)= 8 ...

die Länge des Summenintervalls für einen Hash
der Größe n ist t(n)+t(n-1)

Aufgabe 28

```
void zugroß(int i){     // a[i] ist m¨oglicherweise gr¨oßer als seine Nachfolger.
        int kleinsterNachfolger;
        while(2*i <= n){
                if (2*i+1 <= n) {
                        if (a[2*i]<a[2*i+1]) kleinsterNachfolger = 2*i;
                        else kleinsterNachfolger = 2*i+1;
                }
                else kleinsterNachfolger = 2*i;
                if ((a[i] > a[kleinsterNachfolger])) {
                        vertausche(i, kleinsterNachfolger);
                        zugroß(kleinsterNachfolger);
                }
        }
}
```