Algoritmer i Java

Tillæg til "Algorithms in C++" af R. Sedgewick

Keld Helsgaun
E-mail: keld@ruc.dk

Datalogi
Roskilde Universitetscenter
DK-4000 Roskilde, Danmark

Forord

Disse noter henvender sig til studerende, der følger kursusdelen "Algoritmik" på datalogiuddannelsens første modul. Noterne indholder omskrivninger til programmmeringssproget Java af al C++-kode i den anvendte lærebog "Algorithms in C++" af Robert Sedgewick (Addison Wesley 1992). Noterne tænkes læst sideløbende med lærebogen.

Samtlige Java-programmer er blevet testet, hvilket har afsløret fejl i nogle af lærebogens programmer. Disse fejl er omtalt i noterne. Desuden er der knyttet kommentarer til de enkelte Java-programmer, f.eks. om deres korrespondance med C++-programmerne.

Med Java er det ikke så simpelt at foretage terminalorienteret indlæsning og udskrivning som med C++. Java muliggør udvikling af avancerede grafiske brugergrænseflader, men simpel indlæsning fra tastatur og udskrivning på skærm er ikke simpel. For at afbøde denne ulempe har jeg konstrueret en Java-klasse, kaldet IO. Klassen benyttes i størsteparten af de programmer i noterne, der læser ind og/eller skriver ud. En kort beskrivelse af klassen samt dens kildetekst findes i Appendiks.

Kapitel 2. C++ (and C)

Side 8:

```
import IO.*;

public class Euclid {
    static int gcd(int u, int v) {
        while (u > 0) {
            if (u < v) { int t = u; u = v; v = t; }
            u = u-v;
        }
       return v;
    }

    public static void main(String args[]) {
        while (!IO.eof()) {
           int x = IO.readInt();
           int y = IO.readInt();
           IO.println(x + " "+ y + " " + gcd(x,y));
       }
    }
}
```

Kommentar:

Java giver ikke, som C og C++, mulighed for simpel indlæsning fra tastaturet. Programmøren skal eksplicit håndtere fejl under indlæsningen og opfange et eventuelt signal om, at indlæsningen skal stoppes (EOF). Til dette formål bruges Javas undtagelser (exceptions).

For at forsimple indlæsningen og udskrivningen benytter ovenstående program sig af den specialkonstruerede klasse `IO` (se Appendiks).

Kapitel 3. Elementary Data Structures

Side 16:

```
public class Eratosthenes {
    final static int N = 1000;

    public static void main(String args[]) {
        int i, j;
        boolean a[] = new boolean[N+1];
        for (a[1] = false, i = 2; i <= N; i++)
            a[i] = true;
        for (i = 2; i <= N/2; i++)
            for (j = 2; j <= N/i; j++)
                a[i*j] = false;
        for (i = 1; i <= N; i++)
            if (a[i]) IO.print(i + " ");
                IO.println();
    }
}
```

Side 20:

```
class Node {
    int key;
    Node next;
}

Node head = new Node();
Node z = new Node();
head.next = z; z.next = z;
```

Side 21:

```
import IO.*;

class Node {
    int key;
    Node next;
}

public class Program {
    public static void main(String[] args) {
        Node t, x; int i;
        int N = IO.readInt();
        int M = IO.readInt();
        t = new Node(); t.key = 1; x = t;
        for (i = 2; i <= N; i++) {
            t.next = new Node();
            t = t.next; t.key = i;
        }
        t.next = x;
        while (t != t.next) {
            for (i = 1; i < M; i++)
                t = t.next;
            IO.print(t.next.key + " ");
            x = t.next; t.next = x.next;
        }
        IO.println(t.key);
    }
}
```

Kommentar:

Frigivelse af pladsen for de objekter, der ikke længere er i brug, er her overladt til Javas spildopsamler (garbage collector).

Side 26:

```
public class Stack {
    private int stack[];
    private int p;

    Stack(int max) { stack = new int[max]; p = 0; }
    Stack() { this(100); }

    public void push(int v) { stack[p++] = v; }
    public int pop() { return stack[--p]; }
    public boolean empty() { return p == 0; }
}
```

Kommentar:

Muligheden for som i C/C++ at udskyde specifikationen af stakelementernes type (`itemType`) findes ikke på samme måde i Java. Klassen Object kunne være brugt til formålet, men det ville vanskeliggøre brugen af class `Stack` i de tilfælde, hvor elementernes type er simpel, for eksempel som her, hvor de er af typen `int`.

Side 27:

```
public class Program {
    static char get() throws IOException
        { return (char) System.in.read(); }

    public static void main(String[] args)
                        throws IOException {
       char c; Stack acc = new Stack(50); int x;
       while ((c = get()) != (char) (-1)) {
           x = 0;
           while (c == ' ') c = get();
           if (c == '+') x = acc.pop() + acc.pop();
           if (c == '*') x = acc.pop() * acc.pop();
           while (c >= '0' && c <= '9')
               { x = 10*x + (c-'0'); c = get(); }
           acc.push(x);
        }
        System.out.println(acc.pop());
    }
}
```

Kommentarer:

(1) Typen af stakelementer er `int`.

(2) Programmet kræver, at der i inddata er mellemrum imellem operatorer og operander.

Side 28:

```
public class Program {
    static char get() throws IOException
        { return (char) System.in.read(); }
    static void put(char c)
        { System.out.print(c); }

    public static void main(String[] args)
                        throws IOException {
        char c; Stack save = new Stack(50);
        while ((c = get()) != (char) (-1)) {
            if (c == ')') put(save.pop());
            if (c == '+') save.push(c);
            if (c == '*') save.push(c);
            while (c >= '0' && c <= '9')
                { put(c); c = get(); }
            if (c != '(') put(' ');
        }
        put('\n');
    }
}
```

Kommentarer:

(1) Typen af stakelementer er `char`.

(2) Programmet kræver, at der i inddata er mellemrum imellem operatorer og operander.

(3) Udtrykket, der indlæses, skal være fuldt forsynet med parenteser.

(4) En alternativ implementering, der benyttede Javas indbyggede klasse Stack, ses på næste side.

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

public class Program {
    static char get() throws IOException
        { return (char) System.in.read(); }
    static void put(char c)
        { System.out.println(c); }

    public static void main(String[] args)
                    throws IOException, EmptyStackException {
        char c; Stack save = new Stack();
        while ((c = get()) != (char) (-1)) {
            if (c == ')')
                put(((Character) save.pop()).charValue());
            if (c == '+') save.push(new Character(c));
            if (c == '*') save.push(new Character(c));
            while (Character.isDigit(c))
                { put(c); c = get(); }
            if (c != '(') put(' ');
        }
        put('\n');
    }
}
```

Side 29-30:

```
class Node {
    char key;
    Node next;
}

public class Stack {
    Stack(int max) {
        head = new Node(); z = new Node();
        head.next = z; z.next = head;
    }

    public void push(char v) {
        Node t = new Node();
        t.key = v; t.next = head.next;
        head.next = t;
    }

    public char pop() {
        Node t = head.next;
        head.next = t.next;
        return t.key;
    }

    public boolean empty() { return head.next == z; }

    private Node head, z;
}
```

Kommentar:

Her er stakelementerne af typen `char`.

Side 31:

```
public class Queue {
    private char queue[];
    private int head, tail, size;

    Queue(int max) {
        size = max;
        queue = new char[max + 1];
        head = tail = 0;
    }

    Queue() { this(100); }

    public void put(char v) {
        queue[tail++] = v;
        if (tail > size) tail = 0;
    }

    public char get() {
        char t = queue[head++];
        if (head > size) head = 0;
        return t;
    }

    public boolean empty() { return head == tail; }
}
```

Kommentar:

Alle køelementer er af typen `char`.

Kapitel 4. Trees

Side 41:

```
import java.io.*;

class Node {
    char info;
    Node l, r;
}

class Stack {
    private Node stack[], p;
    public Stack(int max) { stack = new Node[max]; p = 0; }
    public Stack() { this(100); }

    public void push(Node v) { stack[p++] = v; }
    public Node pop() { return stack[--p]; }
    public boolean empty() { return p == 0; }
}

public class Program {
    static char get() throws IOException
        { return (char) System.in.read(); }

    static Node z;
    static Stack stack = new Stack(50);

    public static void main(String[] args)
                        throws IOException {
        Node x;
        char c;

        z = new Node(); z.l = z; z.r = z;
        while ((c = get()) != (char) (-1)) {
            while (c == ' ') c = get();
            x = new Node();
            x.info = c; x.l = z; x.r = z;
            if (c == '+' || c == '*')
                { x.r = stack.pop(); x.l = stack.pop(); }
            stack.push(x);
        }
    }
}
```

Kommentar:

Stakelementerne er her af typen `Node`.

Side 45:

```
void traverse(Node t) {
    stack.push(t);
    while (!stack.empty()) {
        t = stack.pop(); visit(t);
        if (t.r != z) stack.push(t.r);
        if (t.l != z) stack.push(t.l);
    }
}
```

Side 48:

```
void traverse(Node t) {
    queue.put(t);
    while (!queue.empty()) {
        t = queue.get(); visit(t);
        if (t.r != z) queue.put(t.r);
        if (t.l != z) queue.put(t.l);
    }
}
```

Kapitel 5. Recursion

Side 52:

```
int factorial(int N) {
    if (N == 0) return 1;
    return N * factorial(N-1);
}
```

```
int fibonacci(int N) {
    if (N <= 1) return 1;
    return fibonacci(N-1) + fibonacci(N-2);
}
```

Side 53:

```
final int max = 25;

int fibonacci(int N) {
    int F[] = new int[max];
    F[0] = 1; F[1] = 1;
    for (int i = 2; i < max; i++)
        F[i] = F[i-1] + F[i-2];
    return F[N];
}
```

Kommentar:

Der opstår indeksfejl i Sedgewicks C++-udgave! Fejlen er rettet i Java-versionen (`i <= max` er erstattet med i < max).

Side 54:

```
void rule(int l, int r, int h) {
    int m = (l+r)/2;
    if (h > 0) {
        mark(m,h);
        rule(l,m,h-1);
        rule(m,r,h-1);
    }
}
```

Kommentar:

Nedenfor ses en komplet Java-applet til udtegning af linealen i figur 5.1.

```
import java.awt.*;
import java.applet.Applet;

public class App extends Applet {
    final int W = 64*10, H = 8*10;

    public void init() {
        resize(W,H);
        repaint();
    }

    void mark(int x, int h) {
        Graphics g = getGraphics();
        g.drawLine(x,H,x,H-h*10);
    }

    void rule(int l, int r, int h) {
        int m = (l+r)/2;
        if (h > 0) {
            mark(m,h);
            rule(l,m,h-1);
            rule(m,r,h-1);
        }
    }

    public void paint(Graphics g) {
        rule(0,W,8);
    }
}
```

Side 57:

```
void rule(int l, int r, int h) {
    for (int i = 1, j = 1; i <= h; i++, j += j)
        for (int t = 0; t <= (l+r)/j; t++)
            mark(l+j+t*(j+j),i);
}
```

Side 59:

```
void star(int x, int y, int r) {
    if (r > 0) {
        star(x-r,y+r,r/2);
        star(x+r,y+r,r/2);
        star(x-r,y-r,r/2);
        star(x+r,y-r,r/2);
        box(x,y,r);
    }
}
```

Kommentar:

En komplet Java-applet til udtegning af den højre del af figur 5.6 er vist på næste side.

```
import java.awt.*;
import java.applet.Applet;

public class App extends Applet {
    final int W = 64*10;

    public void init() { resize(W,W); repaint(); }

    void box(int x, int y, int r) {
        Graphics g = getGraphics();
        g.drawRect(x*10,y*10,2*r*10,2*r*10);
    }

    void star(int x, int y, int r) {
        if (r > 0) {
            star(x-r,y+r,r/2);
            star(x+r,y+r,r/2);
            star(x-r,y-r,r/2);
            star(x+r,y-r,r/2);
            box(x,y,r);
        }
    }

    public void paint(Graphics g) { star(32,32,32); }
}
```

Side 60:

```
void traverse(Node t) {
    if (t != z) {
        traverse(t.l);
        visit(t);
        traverse(t.r);
    }
}
```

Side 61:

```
class Node {
    int x, y;
    Node l, r;
}

void visit(Node t) { t.x = ++x; t.y = y; }

void traverse(Node t) {
    y++;
    if (t != z) {
        traverse(t.l);
        visit(t);
        traverse(t.r);
    }
    y--;
}
```

Side 62:

```
void traverse(Node t) {
    if (t != z) {
        visit(t);
        traverse(t.l);
        traverse(t.r);
    }
}
```

Side 63:

```
void traverse(Node t) {
    while (true) {
        while (t != z) {
            visit(t);
            stack.push(t.r); t = t.l;
        }
        if (stack.empty()) break;
        t = stack.pop();
    }
}
```

Kommentar:

Halerekursionen kan ikke elimineres ved brug af goto-sætninger, da Java ikke tillader brug af goto.

Side 64:

```
void traverse(Node t) {
    stack.push(t);
    while (!stack.empty()) {
        t = stack.pop();
        while (t != z) {
            visit(t);
            stack.push(t.r);
            t = t.l;
        }
    }
}
```

```
void traverse(Node t) {
    stack.push(t);
    while (!stack.empty()) {
        t = stack.pop();
        if (t != z) {
            visit(t);
            stack.push(t.r);
            stack.push(t.l);
        }
    }
}
```

Side 65:

```
void traverse(Node t) {
    stack.push(t);
    while (!stack.empty()) {
        t = stack.pop();
        if (t != z) {
            visit(t);
            if (t.r != z) stack.push(t.r);
            if (t.l != z) stack.push(t.l);
        }
    }
}
```

Kapitel 8. Elementary Sorting Methods

Side 95:

```
import IO.*;

public class Program {
    static void swap(int a[], int i, int j)
        { int t = a[i]; a[i] = a[j]; a[j] = t; }

    static void sort3(int a[], int N) {
        if (a[1] > a[2]) swap(a, 1, 2);
        if (a[1] > a[3]) swap(a, 1, 3);
        if (a[2] > a[3]) swap(a, 2, 3);
    }

    static final int maxN = 100;

    public static void main(String[] args) {
        int a[] = new int[maxN+1];
        int N = 0;
        while (!IO.eof())
            { int v = IO.readInt(); a[++N] = v; }
        a[0] = 0;
        sort3(a ,N);
        for (int i = 1; i <= N; i++)
            IO.print(a[i] + " ");
        IO.println();
    }
}
```

Kommentar:

Den konkrete specifikation af posttypen (`itemType`) kan ikke udskydes på samme måde som i C/C++.

En mulighed er dog at benytte sig af Javas klassebegreb. På næste side ses to klasser, `Record` og `Table`, der abstraherer begreberne “en post “ og “en tabel af poster, der kan sorteres”. Ideen er at et klientprogram definerer dem konkrete posttype i en underklasse af class `Record`. Med til specifikationen af posttypen hører specifikation af metoden `compare()`. Metoden skal udformes således, at et kald `R1.compare(R2)` returnerer en positiv heltalsværdi, nul, eller en negative heltalsværdi, hvis R1's nøgleværdi er henholdsvis større end, lig med, eller mindre end R2's nøgleværdi. Efter indsættelse af poster i arrayet `a` i et `Table`-objekt kan posterne sorteres med kald af `Table`-metoden `sort()`.

```
abstract class Record {
    abstract int compare(Record r);
}

public class Table {
    public Record a[];

    Table(int n) { a = new Record[n]; }

    public void swap(int i, int j)
        { Record t = a[i]; a[i] = a[j]; a[j] = t; }

    public void sort() {
        // kan overskrives med anden sorteringsmetode
        if (a[1].compare(a[2]) > 0) swap(1, 2);
        if (a[1].compare(a[3]) > 0) swap(1, 3);
        if (a[2].compare(a[3]) > 0) swap(2, 3);
    }
}
```

Nedenfor ses et eksempel på brugen af de to klasser (svarer til programmet på forrige side).

```
class Item extends Record {
    int key;

    Item(int key) { this.key = key; }

    int compare(Record r) { return key - ((Item) r).key; }
}

public class Program {
    static final int maxN = 100;

    public static void main(String args[] {
        Table table = new Table(maxN + 1);
        int N = 0;
        while (!IO.eof())
            table.a[++N] = new Item(IO.readInt());
        table.a[0] = new Item(0);
        table.sort();
        for (int i = 1; i <= N; i++)
           IO.print(((Item) table.a[i]).key + " ");
        IO.println();
    }
}
```

Side 98:

```
void selection(int a[], int N) {
    int i, j, min;
    for (i = 1; i < N; i++) {
        min = i;
        for (j = i+1; j <= N; j++)
            if (a[j] < a[min]) min = j;
        swap(a, min, i);
    }
}
```

Side 100:

```
void insertion(int a[], int N) {
    int i, j; int v;
    for (i = 2; i <= N; i++) {
        v = a[i]; j = i;
        while (a[j-1] > v)
            { a[j] = a[j-1]; j--; }
        a[j] = v;
    }
}
```

Side 101:

```
void bubble(int a[], int N) {
    for (int i = N; i >= 1; i--)
        for (int j = 2; j <= i; j++)
            if (a[j-1] > a[j]) swap(a, j-1, j);
}
```

Side 105:

```
void insertion(int a[], int p[], int N) {
    int i, j, v;
    for (i = 0; i <= N; i++) p[i] = i;
    for (i = 2; i <= N; i++) {
        v = p[i]; j = i;
        while (a[p[j-1]] > a[v])
            { p[j] = p[j-1]; j--; }
        p[j] = v;
    }
}
```

Kommentar:

Der er fejl i Sedgewicks udgave! Variablen `v` skal *ikke* være af typen `ItemType`, men af typen `int`.

Side 106:

```
void insitu(int a[], int p[], int N) {
    int i, j, k; int t;
    for (i = 1; i <= N; i++)
        if (p[i] != i) {
            t = a[i]; k = i;
            do {
                j = k; a[j] = a[p[j]];
                k = p[j]; p[j] = j;
            } while (k != i);
            a[j] = t;
        }
}
```

Side 107:

Kommentar:

Programmet kan ikke implementeres i Java, da direkte brug af maskinadresser ikke er tilladt i sproget.

Side 109:

```
void shellsort(int a[], int N) {
    int i, j, h; int v;
    for (h = 1; h <= N/9; h = 3*h+1) ;
    for ( ; h > 0; h /= 3)
        for (i = h+1; i <= N; i++) {
            v = a[i]; j = i;
            while (j > h && a[j-h] > v)
                { a[j] = a[j-h]; j -= h; }
            a[j] = v;
        }
}
```

Side 112:

```
void distribution(int a[], int N, int M) {
    int i, j;
    int count[] = new int[M];
    int b[] = new int[N+1];

    for (j = 0; j < M; j++) count[j] = 0;
    for (i = 1; i <= N; i++) count[a[i]]++;
    for (j = 1; j < M; j++) count[j] += count[j-1];
    for (i = N; i >= 1; i--) b[count[a[i]]--] = a[i];
    for (i = 1; i <= N; i++) a[i] = b[i];
}
```

Kapitel 9. Quicksort

Side 117:

```
void quicksort(int a[], int l, int r) {
    if (r > 1) {
        int i = partition(a, l, r);
        quicksort(a, l, i-1);
        quicksort(a, i+1, r);
    }
}
```

Side 118:

```
void quicksort(int a[], int l, int r) {
    int i, j; int v;

    if (r > l) {
        v = a[r]; i = l-1; j = r;
        for (;;) {
            while (a[++i] < v) ;
            while (a[--j] > v) ;
            if (i >= j) break;
            swap(a, i, j);
        }
        swap(a, i, r);
        quicksort(a, l, i-1);
        quicksort(a, i+1, r);
   }
}
```

Kommentar:

Metoden kan også implementeres ved brug af en hjælpeprocedure, `partition`.

```
int partition(int a[], int l, int r) {
    int i = l - 1, j = r; int v = a[r];
    for (;;) {
        while (a[++i] < v) ;
        while (a[--j] > v) ;
        if (i >= j) break;
        swap(a, i, j);
    }
    swap(a, i, r);
    return i;
}

void quicksort(int a[], int l, int r) {
    int i, j; int v;
    if (r > l) {
        i = partition(a, l, r);
        quicksort(a, l, i-1);
        quicksort(a, i+1, r);
    }
}
```

Side 122:

```
import IO.*;

public class Stack {
    private int stack[], p;

    public Stack(int max) { stack = new int[max]; p = 0; }
    public Stack() { this(100); }
    public void push(int v) { stack[p++] = v; }
    public int pop() { return stack[--p]; }
    public boolean empty() { return p == 0; }
}

public class Program {
    static void swap(int a[], int i, int j)
        { int t = a[i]; a[i] = a[j]; a[j] = t; }

    static int partition(int a[], int l, int r) {
        int i = l - 1, j = r; int v = a[r];
        for (;;) {
            while (a[++i] < v) ;
            while (a[--j] > v) ;
            if (i >= j) break;
            swap(a, i, j);
        }
        swap(a, i, r);
        return i;
    }

    static void quicksort(int a[], int l, int r) {
        int i; Stack sf = new Stack(50);
        for (;;) {
            while (r > l) {
                i = partition(a, l, r);
                if (i-1 > r-i)
                    { sf.push(l); sf.push(i-1); l = i+1; }
                else
                    { sf.push(i+1); sf.push(r); r = i-1; }
            }
            if (sf.empty()) break;
            r = sf.pop(); l = sf.pop();
        }
    }

    static final int maxN = 100;
```

```
    public static void main(String[] args ) {
        int v, a[] = new int[maxN+1], p[] = new int[maxN+1];
        int N = 0; while (!IO.eof()) a[++N] = IO.readInt();
        a[0] = 0; quicksort(a,1,N);
        for (int i = 1; i <= N; i++) IO.print(a[i] + " ");
        IO.println();
    }
}
```

Side 127:

```
void select(int a[], int l, int r, int k) {
    int i;
    if (r > l) {
        i = partition(a, l, r);
        if (i > l+k-1) select(a, l, i-1, k);
        if (i < l+k-1) select(a, i+1, r, l+k-1-i);
    }
}
```

Kommentar:

Der er fejl i Sedgewicks udgave! Argumentet `k-i` til det sidste rekursive kald af `select` skal erstattes med `l+k-1-i`.

Side 128:

```
void select(int a[], int N, int k) {
    int l = 1, r = N;
    while (r > l) {
        itemType v = a[r];
        int i = l-1, j = r;
        for (;;) {
            while (a[++i] < v) ;
            while (a[--j] > v) ;
            if (i >= j) break;
            swap(a, i, j);
        }
        swap(a, i, r);
        if (i >= k) r = i-1;
        if (i <= k) l = i+1;
    }
}
```

Kapitel 10. Radix Sorting

Side 134:

```
public class Bitskey {
    private int x;

    Bitskey(int i) { x = i; }

    public int get() { return x; }

    public int bits(int k, int j)
        { return (x >> k) & ~(~0 << j); }
}
```

Kommentar:

Metoden `get()` er tilføjet for at gøre det muligt at hente tabelværdierne, f.eks. efter en sortering.

Side 135:

```
void radixexchange(Bitskey a[], int l, int r, int b) {
    int i, j; Bitskey t;
    if (r > l && b >= 0) {
        i = l; j = r;
        while (j != i) {
            while (a[i].bits(b,1) == 0 && i < j) i++;
            while (a[j].bits(b,1) == 1 && j > i) j--;
            swap(a,i,j);
        }
        if (a[r].bits(b,1) == 0) j++;
        radixexchange(a, l, j-1, b-1);
        radixexchange(a, j, r, b-1);
    }
}
```

Kommentar:

På næste side ses `radixexchange` anvendt i et drivprogram.

```
public class Program {
    static void swap(Bitskey a[], int i, int j)
        { Bitskey t = a[i]; a[i] = a[j]; a[j] = t; }

    static void radixexchange(Bitskey a[],
                              int l, int r,int b) {
        ...
    }

    static final int maxN = 100;

    public static void main(String[] args)
                        throws IOException {
        Bitskey a[] = new Bitskey[maxN+1];
        int N = 0;
        while (!IO.eof()) a[++N] = IO.readInt();
        radixexchange(a, 1, N, 30);
        for (int i = 1; i <= N; i++)
            IO.print(a[i].get() + " ");
        IO.println();
    }
}
```

Side 140:

```
void straightradix(Bitskey a[], Bitskey b[], int N) {
    int i, j, pass;
    int count[] = new int[M];
    for (pass = 0; pass < w/m; pass++) {
        for (j = 0; j < M; j++) count[j] = 0;
        for (i = 1; i <= N; i++)
            count[a[i].bits(pass*m, m)]++;
        for (j = 1; j < M; j++)
            count[j] += count[j-1];
        for (i = N; i >= 1; i--)
            b[count[a[i].bits(pass*m, m)]--] = a[i];
        for (i = 1; i <= N; i++) a[i] = b[i];
    }
}
```

Kommentarer:

(1) Der er fejl i Sedgewicks udgave! Tabelerklæringen `count[M-1]` skal erstattes af `count[M]`.

(2) En mulig initialisering af konstanterne `w`, `m` og `M` kunne ske som følger:

```
static final int w = 32, m = 8;
static final int M = (int) Math.pow(2, m);
```

Kapitel 11. Priority Queues

Side 147:

```
public class PQ {
    private int a[];
    private int N;

    PQ(int max) { a = new int[max+1]; N = 0; }

    public void insert(int v) { a[++N] = v; }

    public int remove() {
        int j, max = 1;
        for (j = 2; j <= N; j++)
            if (a[j] > a[max]) max = j;
        swap(a, max, N);
        return a[N--];
    }
}
```

Kommentar:

Tabellen a oprettes med `max`+1 elementer, startende med index 0. Ellers virker den efterfølgende hobsortering ikke, da tabelindeks her starter med 1.

Side 150:

```
public void upheap(int k) {
    int v = a[k];
    a[0] = Integer.MAX_VALUE;
    while (a[k/2] <= v)
        { a[k] = a[k/2]; k /= 2; }
    a[k] = v;
}

public void insert(int v) { a[++N] = v; upheap(N); }
```

Side 152:

```
public void downheap(int k) {
    int j; int v = a[k];
    while (k <= N/2) {
        j = k+k;
        if (j < N && a[j] < a[j+1]) j++;
        if (v >= a[j]) break;
        a[k] = a[j]; k = j;
    }
    a[k] = v;
}
```

Side 153:

```
public int replace(int v) {
    a[0] = v;
    downheap(0);
    return a[0];
}
```

```
public int remove() {
    int v = a[1];
    a[1] = a[N--];
    downheap(1);
    return v;
}
```

Side 156:

```
void heapsort(int a[], int N) {
    for (int k = N/2; k >= 1; k--)
        downheap(a, N, k);
    while (N > 1)
        { swap(a, 1, N); downheap(a, --N, 1); }
}
```

Side 160:

```
public class PQ {
    private int a[], p[], info[];
    private int N;

    private void swap(int a[], int i, int j)
        { int t = a[i]; a[i] = a[j]; a[j] = t; }

    PQ(int size) {
        a = new int[size];
        p = new int[size];
        info = new int[size];
        N = 0;
    }

    public void insert(int x, int v)
        { a[++N] = v; p[x] = N; info[N] = x; }

    public void change(int x, int v) { a[p[x]] = v; }

    public int remove() {
        int j, min = 1;
        for (j = 2; j <= N; j++)
            if (a[j] < a[min]) min = j;
        swap(a, min, N); swap(info, min, N);
        p[info[min]] = min;
        return info[N--];
    }

    public boolean empty() { return N <= 0; }
}
```

Kommentar:

Metoden `swap` er her tilføjet som en privat metode i `PQ`.

Kapitel 12. Mergesort

Side 164:

```
i = 1; j = 1;
a[M+1] = Integer.MAX_VALUE; b[N+1] = Integer.MAX_VALUE;
for (k = 1; k <= M+N; k++)
    c[k] = (a[i] < b[j]) ? a[i++] : b[j++];
```

Kommentar:

Tabelværdierne antages at være at typen `int`.

Side 165:

```
class Node {
    int key;
    Node next;
}

public class Program {
    static Node z;

    static Node merge(Node a, Node b) {
        Node c = z;
        do
            if (a.key <= b.key)
                { c.next = a; c = a; a = a.next; }
            else
                { c.next = b; c = b; b = b.next; }
       while (c != z);
       c = z.next; z.next = z;
       return c;
    }

    public static void main(String[] args) {
        z = new Node();
        z.next = z; z.key = Integer.MAX_VALUE;
       ...
    }
}
```

Side 166:

```
void mergesort(int a[], int l, int r) {
    int i, j, k;
    if (r > l) {
        int m = (r+l)/2;
        mergesort(a, l, m);
        mergesort(a, m+1, r);
        for (i = m+1; i > l; i--) b[i-1] = a[i-1];
        for (j = m; j < r; j++) b[r+m-j] = a[j+1];
        for (k = l; k <= r; k++)
            a[k] = (b[i] < b[j]) ? b[i++] : b[j--];
    }
}
```

Side 168:

```
Node mergesort(Node c) {
    Node a, b;
    if (c.next != z) {
        a = c; b = c.next.next.next;
        while (b != z)
            { c = c.next; b = b.next.next; }
        b = c.next; c.next = z;
        return merge(mergesort(a), mergesort(b));
    }
    return c;
}
```

Side 170:

```
Node mergesort(Node c) {
    int i, N;
    Node a, b, head, todo, t;
    head = new Node();
    head.next = c; a = z;
    for (N = 1; a != head.next; N = N+N) {
        todo = head.next; c = head;
        while (todo != z) {
            t = todo; a = t;
            for (i = 1; i < N; i++) t = t.next;
            b = t.next; t.next = z; t = b;
            for (i = 1; i < N; i++) t = t.next;
            todo = t.next; t.next = z;
            c.next = merge(a,b);
            for (i = 1; i <= N+N; i++) c = c.next;
        }
    }
    return head.next;
}
```

Kapitel 14. Elementary Searching Methods

Side 195:

```
class Node {
    int key;
    String info;
}

class Dict {
    private Node a[];
    private int N;

    Dict(int max) {
        a = new Node[max];
        for (int i = 0; i < max; i++) a[i] = new Node();
        N = 0;
    }

    public String search(int v) {        // Sequential
        int x = N+1;
        a[0].key = v; a[0].info = null;
        while (v != a[--x].key) ;
        return a[x].info;
    }

    public void insert(int v, String info)
       { a[++N].key = v; a[N].info = info; }
}
```

Kommentarer:

(1) Nøgletypen (`itemType`) er valgt til at være `int`, mens informationen (`info`) er af typen `String`. Java muliggør ikke at fastsættelsen af disse typer udskydes, som det er tilfældet i C og C++. En mulighed ville være at specificere dem som værende af typen `Object`, men det besværliggør brugen af class `Dict`, når typerne er simple, f.eks. `int` og `char`. En udgave, der bruger `Object`, ser ud som følger:

```
class Node {
    Object key, info;
}

class Dict {
    private Node a[];
    private int N;

    Dict(int max) {
        a = new Node[max];
        for (int i = 0; i < max; i++) a[i] = new Node();
        N = 0;
    }

    public Object search(Object v) {
        int x = N+1;
        a[0].key = v; a[0].info = null;
        while (!v.equals(a[--x].key)) ;
        return a[x].info;
    }

    public void insert(Object v, Object info)
        { a[++N].key = v; a[N].info = info; }
}
```

(2) Bemærk at Java ikke som C og C++ tillader oprettelse af en tabel af objekter, men kun en tabel af referencer til objekter. Derfor oprettes `Node`-objekterne eksplicit af konstruktøren i `Dict`. Alternativt kunne `Node`-objekterne oprettes i metoden `insert`:

```
public void insert(int v, String info)
    { a[++N] new Node(); a[N].key = v; a[N].info = info; }
```

Side 196-197:

```
class Node {
    int key;
    String info;
    Node next;

    Node(int k, String i, Node n)
        { key = k; info = i; next = n; }
}

class Dict {
    private Node a[];
    private int N;
    private Node head, z;

    Dict(int max) {
        z = new Node(Integer.MAX_VALUE,null,null);
        head = new Node(0,null,z);
    }

    public String search(int v) {        // Sorted list
        Node t = head;
        while (v > t.key) t = t.next;
        return (v == t.key) ? t.info : z.info;
    }

    public void insert(int v, String info) {
        Node x, t = head;
        while (v > t.next.key) t = t.next;
        x = new Node(v,info,t.next);
        t.next = x;
    }
}
```

Kommentar:

Der er en fejl i Sedgewicks C++-udgave af metoden `search`! Udtrykket `v = t.key` sidst i `search` skal erstattes med `v == t.key`.

Side 198:

```
public String search(int v) {       // Binary search
    int l = 1, r = N, x;
    while (r >= l) {
        x = (l+r)/2;
        if (v == a[x].key) return a[x].info;
        if (v < a[x].key) r = x-1; else l = x+1;
    }
    return null;
}
```

Kommentar:

Metoden `insert` kunne programmeres således:

```
public void insert(int v, String info) {
    int i;
    Node t;
    a[0].key = a[N+1].key = v; a[N+1].info = info;
    for (i = N; v < (t = a[i]).key; i--) {
        a[i] = a[i+1];
        a[i+1] = t;
    }
    N++;
}
```

Side 204:

```
class Node {
    int key;
    String info;
    Node l, r;

    Node(int k, String i, Node ll, Node rr)
        { key = k; info = i; l = ll; r = rr; }
}

class Dict {
    private Node head, z;
    private int N;

    Dict(int max) {
        z = new Node(0,null,null,null);
        head = new Node(Integer.MIN_VALUE,null,null,z);
    }

    public String search(int v) {        // Binary search
        Node x = head.r;
        z.key = v;
        while (v != x.key)
            x = (v < x.key) ? x.l : x.r;
        return x.info;
    }
}
```

Side 206:

```
public void insert(int v, String info) {
    Node p = head, x = head.r;
    while (x != z)
        { p = x; x = (v < x.key) ? x.l : x.r; }
    x = new Node(v, info, z, z);
    if (v < p.key) p.l = x; else p.r = x;
}
```

Side 210:

```
public  void delete(int v) {
    Node c, p, x, t;
    z.key = v;
    p = head; x = head.r;
    while (v != x.key)
        { p = x; x = (v < x.key) ? x.l : x.r; }
    t = x;
    if (t.r == z) x = x.l;
    else if (t.r.l == z) { x = x.r; x.l = t.l; }
    else {
        c = x.r;
        while (c.l.l != z) c = c.l;
        x = c.l; c.l = x.r;
        x.l = t.l; x.r = t.r;
    }
    if (v < p.key) p.l = x; else p.r = x;
}
```

Kapitel 15. Balanced Trees

Side 221:

```
public void insert(int v, String info) {
    x = head; p = head; g = head;
    while (x != z) {
        gg = g; g = p; p = x;
        x = (v < x.key) ? x.l : x.r;
        if (x.l.b == red && x.r.b == red) split(v);
    }
    x = new Node(v, info, red, z, z);
    if (v < p.key) p.l = x; else p.r = x;
    split(v); head.r.b = black;
}
```

Kommentarer:

(1) Bemærk at `x`, `p`, `g` og `gg` skal være erklæret i klassen `Dict` (og helst være private).

(2) Konstanterne `black` og `red` tænkes erklæret som statiske `boolean`-felter i class `Dict`:

```
private static final boolean black = false, red = true;
```

Side 225:

```
private Node rotate(int v, Node y) {
    Node c, gc;
    c = (v < y.key) ? y.l : y.r;
    if (v < c.key)
        { gc = c.l; c.l = gc.r; gc.r = c; }
    else
        { gc = c.r; c.r = gc.l; gc.l = c; }
    if (v < y.key) y.l = gc; else y.r = gc;
    return gc;
}
```

Side 226:

```
private void split(int v) {
    x.b = red; x.l.b = x.r.b = black;
    if (p.b == red) {
        g.b = red;
        if (v<g.key != v<p.key) p = rotate(v,g);
        x = rotate(v,gg);
        x.b = black;
    }
}
```

```
Dict(int max) {
    z = new Node(0,null,black,null,null);
    z.l = z; z.r = z;
    head = new Node(Integer.MIN_VALUE,null,black,null,z);
}
```

Kapitel 16. Hashing

Side 233:

```
class Stringkey {
    String v;

    Stringkey(String v) { this.v = v; }

    int hash(int M) {
        int h = 0;
        for (int i = 0; i < v.length(); i++)
            h = (64*h + v.charAt(i)) % M;
        return h;
    }
}
```

Kommentarer:

(1) I modsætning til C og C++ tillader Java ikke adressearitmetik. Derfor hentes de enkelte tegn i strengen ved hjælp af metoden `charAt`;

(2) I Java er typen char repræsenteret i 2 bytes. Derfor burde konstanten `64` erstattes af `2^32 = 65536`; eller `256`, hvis tegnene i strengen kun kan være normale ASCII-tegn (7-bits kode).

(3) Java har en metode til hashkodning, `hashCode`, knyttet til klassen `String` (ja faktisk til ethvert objekt). Metoden `hash` overfor kunne derfor alternativt programmeres således:

```
int hash(int M) { return v.hashCode() % M; }
```

Side 234:

```
Dict(int sz) {
    M = sz;
    z = new Node(); z.next = z; z.info = null;
    heads = new Node[M];
    for (int i = 0; i < M; i++)
        { heads[i] = new Node(); heads[i].next = z; }
}
```

Side 237:

```
class Stringkey {
    String v;

    Stringkey(String v) { this.v = v; }

    boolean equals(Stringkey k) { return v.equals(k.v); }

    int hash(int M) {
        int h = 0;
        for (int i = 0; i < v.length(); i++)
            h = (64*h + v.charAt(i)) % M;
        return h;
    }
}

class Node {
    Stringkey key;
    String info;
    Node next;

    Node() { key = new Stringkey(" "); info = null; }
}

class Dict {
    private int M;
    private Node a[];

    Dict(int sz) {
        M = sz;
        a = new Node[M];
        for (int i = 0; i < M; i++)
            a[i] = new Node();
    }

    public String search(Stringkey v) {
        int x = v.hash(M);
        while (a[x].info != null && !v.equals(a[x].key))
            x = (x+1) % M;
            return a[x].info;
        }

    public void insert(Stringkey v, String info) {
        int x = v.hash(M);
        while (a[x].info != null) x = (x+1) % M;
        a[x].key = v; a[x].info = info;
    }
}
```

Kommentar:

Bemærk brugen af `equals` (i class `Stringkey` og `Dict.search`). I modsætning til C++ tillader Java ikke overlæsning af operatorer.

Kapitel 17. Radix Searching

Side 246:

```
public String search(Bitskey v) {
    Node x = head;
    int b = Bitskey.maxb;
    z.key = v;
    while (!v.equals(x.key))
        x = (v.bits(b--,1) != 0) ? x.r : x.l;
    return x.info;
}
```

Kommentarer:

(1) Bemærk brugen af metoden `equals`, som forventes erklæret i class `Bitskey`:

```
public class Bitskey {
    private int x;
    public static final int maxb = 31;
    // negative heltal kan ikke indsættes

    Bitskey(int i) { x = i; }

    public boolean equals(Bitskey b)
        { return x == b.x; }

    public int get() { return x; }

    public int bits(int k, int j)
        { return (x >> k) & ~(~0 << j); }
}
```

(2) Klasserne `Node` og `Dict` kan programmeres således:

```
class Node {
    Bitskey key;
    String info;
    Node l, r;
}

class Dict {
    private Node z, head;

    Dict() {
        z = new Node();
        z.key = new Bitskey(Integer.MAX_VALUE);
        head = new Node(); head.key = new Bitskey(0);
        head.l = head.r = z;
    }

    public String search(Bitskey v) { ... }

    public void insert(Bitskey v, String info) { ... }
}
```

Side 247:

```
public void insert(Bitskey v, String info) {
    Node p = null, x = head;
    int b = Bitskey.maxb;
    while (x != z) {
        p = x;
        x = (v.bits(b--,1) == 1) ? x.r : x.l;
    }
    x = new Node();
    x.key = v; x.info = info; x.l = z; x.r = z;
    if (v.bits(b+1,1) == 1) p.r = x; else p.l = x;
}
```

Kommentar:

Java kræver at `Node`-referencen `p` initialiseres. Det ville være bedre at erstatte `while`-løkken med en `do`-løkke.

Side 254:

```
public String search(Bitskey v) {
    Node p, x;
    p = head; x = head.l;
    while (p.b > x.b) {
        p = x;
        x = (v.bits(x.b,1) == 1) ? x.r : x.l;
    }
    if (!v.equals(x.key)) return null;
    return x.info;
}
```

Kommentarer:

(1) Metoden benytter sig af class `Bitskey`. Derfor skrives `v.bits(x.b,1)`, og ikke `bits(v,x,b,1)` som i C++-koden.

(2) Bemærk brugen af metoden `Bitskey.equals` til nøglesammenligning.

Side 256:

```
public void insert(Bitskey v, String info) {
    Node p, t, x;
    int i = Bitskey.maxb;
    p = head; t = head.l;
    while (p.b > t.b)
        { p = t; t = (v.bits(t.b,1) == 1) ? t.r : t.l; }
    if (v.equals(t.key)) return;
    while (t.key.bits(i,1) == v.bits(i,1)) i--;
    p = head; x = head.l;
    while (p.b > x.b && x.b > i)
        { p = x; x = (v.bits(x.b,1) == 1) ? x.r : x.l; }
    t = new Node();
    t.key = v; t.info = info; t.b = i;
    t.l = (v.bits(t.b,1) == 1) ? x : t;
    t.r = (v.bits(t.b,1) == 1) ? t : x;
    if (v.bits(p.b,1) == 1) p.r = t; else p.l = t;
}
```

Kommentar:

Nedenfor ses skeletter af klasserne `Node` og `Dict`.

```
class Node {
    Bitskey key;
    String info;
    Node l, r;
    int b;
}

class Dict {
    private Node head;

    Dict() {
        head = new Node();
        head.key = new Bitskey(0);
        head.l = head.r = head;
        head.b = Bitskey.maxb;
    }

    public String search(Bitskey v) { ... }

    public void insert(Bitskey v, String info) { ... }
}
```

Kapitel 19. String Searching

Side 279:

```
int brutesearch(char p[], char a[]) {
    int i, j, M = p.length, N = a.length;
    for (i = 0, j = 0; j < M && i < N; i++, j++)
        if (a[i] != p[j]) { i -= j; j = -1; }
    if (j == M) return i-M; else return i;
}
```

Kommentar:

Der er fejl i Sedgewicks udgave! Sætningen `i -= j - 1` skal erstattes med `i -= j`. Aternativt kan `if`-sætningen erstattes med

```
while (a[i] != p[j]) { i -= j-1; j = 0; }
```

I lighed med C++-udgaven benyttes `char`-tabeller som parametre. Metoden `String.toCharArray` kan benyttes til at konvertere en `String` til en `char`-tabel.

En alternativ udgave af `brutesearch`, der benytter `String`-parametre, ser således ud:

```
int brutesearch(String p, String a) {
    int i, j, M = p.length(), N = a.length();
    for (i = 0, j = 0; j < M && i < N; i++, j++)
        if (a.charAt(i) != p.charAt(j)) { i -= j; j = -1; }
    if (j == M) return i-M; else return i;
}
```

Side 282:

```
int kmpsearch(char p[], char a[]) {
    int i, j, M = p.length, N = a.length;
    initnext(p);
    for (i = 0, j = 0; j < M && i < N; i++, j++)
        while (j >= 0 && (a[i] != p[j])) j = next[j];
    if (j == M) return i-M; else return i;
}
```

Side 283:

```
void initnext(char p[]) {
    int i, j, M = p.length;
    next[0] = -1;
    for (i = 0, j = -1; i < M; i++, j++, next[i] = j)
        while (j >= 0 && p[i] != p[j]) j = next[j];
}
```

```
int kmpsearch(char a[]) {
    int i = -1, s = -1;
    for (;;) {
        switch(s) {
        case -1: i++;
        case  0: if (a[i] != '1') {s = -1; continue;}
                 else i++;
        case  1: if (a[i] != '0') {s =  0; continue;}
                 else i++;
        case  2: if (a[i] != '1') {s =  0; continue;}
                 else i++;
        case  3: if (a[i] != '0') {s =  1; continue;}
                 else i++;
        case  4: if (a[i] != '0') {s =  2; continue;}
                 else i++;
        case  5: if (a[i] != '1') {s =  0; continue;}
                 else i++;
        case  6: if (a[i] != '1') {s =  1; continue;}
                 else i++;
        case  7: if (a[i] != '1') {s =  1; continue;}
                 else i++;
        }
        return i-8;
    }
}
```

Kommentar:

Java indeholder ikke en `goto`-sætning, der tillader vilkårlige programhop. I stedet benyttes Javas `switch`-sætning i en løkke. Programkontrollen styres af tilstandsvariablen `s`.

Side 285:

```
next[i] = (p[i] == p[j]) ? next[j] : j;
```

Kommentar:

Koden giver indeksfejl, med mindre p-tabellen er dimensioneret med plads til et ekstra tegn. Problemet kan også løses ved at benytte følgende sætning:

```
next[i] = (i < M && p[i] == p[j]) ? next[j] : j;
```

Side 288:

```
int mischarsearch(char p[], char a[]) {
    int i, j, t, M = p.length, N = a.length;
    initskip(p);
    for (i = M-1, j = M-1; j >= 0; i--, j--)
        while (a[i] != p[j]) {
            t = skip[index(a[i])];
            i += (M-j > t) ? M-j : t;
            if (i >= N) return N;
            j = M-1;
    }
    return i+1;
}
```

Kommentarer:

(1) Der er fejl i lærebogen! Betingelsen `j > 0` i den yderste løkke skal erstattes med `j >= 0`, og sætningen `return i` skal erstattes med `return i+1`.

(2) Antages at de tilladte bogstaver er bogstaverne fra `A` til `Z`, så kan metoderne `index` og `initskip` programmeres som følger:

```
int index(char c) {
    if (c == ' ')
        return 0;
    return c - 'A' + 1;
}

void initskip(char p[]) {
    int i, j, M = p.length;
    for (i = 0; i < 27; i++)
        skip[i] = M;
    for (j = 0; j < M; j++)
        skip[index(p[j])] = M-j-1;
}
```

idet tabellen `skip` er erklæret således:

```
int skip[] = new int[27];     // 1 + ('Z' - 'A' + 1) = 27
```

Side 290:

```
int rksearch(char p[], char a[]) {
    int i, dM = 1, h1 = 0, h2 = 0;
    int M = p.length, N = a.length;
    for (i = 1; i < M; i++) dM = (d*dM) % q;
    for (i = 0; i < M; i++) {
        h1 = (h1*d+index(p[i])) % q;
        h2 = (h2*d+index(a[i])) % q;
    }
    for (i = 0; h1 != h2; i++) {
        h2 = (h2+d*q-index(a[i])*dM) % q;
        h2 = (h2*d+index(a[i+M])) % q;
        if (i > N-M) return N;
    }
    return i;
}
```

Kapitel 20. Pattern Matching

Side 302:

```
int match(char a[]) {
    int n1, n2; Deque dq = new Deque(100);
    int j = 0, N = a.length, state = next1[0];
    dq.put(scan);
    while (state != 0) {
        if (state == scan) { j++; dq.put(scan); }
        else if (ch[state] == a[j])
            dq.put(next1[state]);
        else if (ch[state] == ' ') {
            n1 = next1[state];
            n2 = next2[state];
            dq.push(n1);
            if (n1 != n2) dq.push(n2);
       }
       if (dq.empty() || j == N) return 0;
       state = dq.pop();
    }
    return j;
}
```

Kommentar:

Klassen `Deque` kan implementeres således ved brug af en tabel (`deque`):

```
public class Deque {
    private int deque[], head, tail, size;

    Deque(int max) {
        size = max; head = tail = 0;
        deque = new int[max + 1];
    }
    Deque() { this(100); }

    public void put(int v) {
        deque[tail++] = v;
        if (tail >= size) tail = 0;
    }

    public void push(int v) {
        if (--head < 0) head = size-1;
        deque[head] = v;
    }

    public int pop() {
        int t = deque[head++];
        if (head >= size) head = 0;
        return t;
    }

    public boolean empty() { return head == tail; }
}
```

Initialisering kan ske således:

```
char ch[] = " A B  ACD ".toCharArray();
int next1[] = {5,2,3,4,8,6,7,8,9,0};
int next2[] = {5,2,1,4,8,2,7,8,9,0};
final int scan = -1;
```

En match med eksempelstrengen "AAABD" kan undersøges med kaldet

```
match("AAABD.".toCharArray()))
```

Bemærk det afsluttende ekstra tegn, her '.', i inputstrengen, som ifølge Sedgewick (s. 301) er påkrævet af algoritmen.

Kapitel 21. Parsing

Side 308:

```
void expression() {
    term();
    if (p[j] == '+')
        { j++; expression(); }
}
```

Kommentar:

Heltalsvariablen `j` og `char`-tabellen `p` skal være globale iforhold til metoden `expression`.

Side 309:

```
void term() {
    factor();
    if ((p[j] == '(') || Character.isLetter(p[j]))
        term();
}

void factor() {
    if (p[j] == '(') {
        j++; expression();
        if (p[j] == ')') j++; else error();
    }
    else if (Character.isLetter(p[j])) j++;
    else error();
    if (p[j] == '*') j++;
}
```

Kommentar:

En simpel udgave af metoden `error` er følgende:

```
void error() {
    IO.println("Syntax error");
    System.exit(0);
}
```

Side 311:

```
void badexpression() {
    if (Character.isLetter(p[j])) j++; else {
        badexpression();
        if (p[j] == '+') { j++; term(); }
    }
}
```

Kommentar:

Der er en syntaksfejl i C++-udgaven! Semikolonet efter metodehovedet, `badexpression()`, bør slettes.

Side 313:

```
int expression() {
    int t1, t2, r;
    t1 = term(); r = t1;
    if (p[j] == '+') {
        j++; state++;
        t2 = state; r = t2; state++;
        setstate(t2, ' ', expression(), t1);
        setstate(t2-1, ' ', state, state);
    }
    return r;
}
```

Kommentar:

Metoden `setstate` programmeres således:

```
void setstate(int i, char c, int n1, int n2) {
    ch[i] = c; next1[i] = n1; next2[i] = n2;
}
```

Side 314:

```
int term() {
    int r;
    r = factor();
    if ((p[j] == '(') || Character.isLetter(p[j])) term();
    return r;
}
```

```
int factor() {
    int t1 = state, t2, r;
    if (p[j] == '(') {
        j++; t2 = expression();
        if (p[j] == ')') j++; else error();
    }
    else if (Character.isLetter(p[j])) {
        setstate(state, p[j], state+1, state+1);
        t2 = state; j++; state++;
    } else { t2 = 0; error(); }
    if (p[j] != '*') r = t2; else {
        setstate(state, ' ', state+1, t2);
        r = state; next1[t1-1] = state;
        j++; state++;
    }
    return r;
}
```

Kommentar:

Java forlanger, at `t2` i metoden `factor` tildeles en værdi i den første `if`-sætning, eftersom variablens værdi benyttes i den efterfølgende sætning. Derfor sættes `t2` til `0` inden kaldet af `error`, også selv om denne metode terminerer programmet.

Side 315:

```
void matchall(char a[]) {
    j = 0; state = 1;
    next1[0] = expression();
    setstate(0, ' ', next1[0], next1[0]);
    setstate(state, ' ', 0, 0);
    for (int i = 0; i < a.length; i++)
        IO.print(match(String.valueOf(
                a, i, a.length-i).toCharArray()) + " ");
    IO.println();
}
```

Kommentar:

Java tillader ikke adressearitmetik. Derfor opnås parameteren til metoden `match` ved hjælp af `String`-metoderne `valueOf` og `toCharArray`.

Kapitel 22. File Compression

Side 324:

```
for (i = 0; i <= 26; i++) count[i] = 0;
for (i = 0; i < M; i++) count[index(a[i])]++;
```

Kommentarer:

(1) M == a.length.

(2) Der er fejl i bogen på side 324! Tabellen `count` skal være dimensioneret med `count[27]`, og ikke `count[26]`. I Java oprettes tabellen således:

```
int count[] = new int[27];
```

Side 328:

```
for (i = 0; i <= 26; i++)
    if (count[i] != 0) pq.insert(i, count[i]);
for ( ; !pq.empty(); i++) {
    t1 = pq.remove(); t2 = pq.remove();
    dad[i] = 0; dad[t1] = i; dad[t2] = -i;
    count[i] = count[t1] + count[t2];
    if (!pq.empty()) pq.insert(i, count[i]);
}
```

Kommentar:

Der er fejl i bogen! Ved kald af `insert`-metoden i klassen `PQ` (i kapitel 9) skal parametrene byttes om, dvs. `pq.insert(count[i],i)` skal erstattes med `pq.insert(i,count[i])`.

En kørsel med det rettede program gav en anden udskrift med hensyn til `dad`-tabellen end nederst på side 327 (men dog stadigvæk repræsenterende et optimalt Huffman-træ):

```
  0   1   2  3  4  5   6   7   9  12  13  14 15 16 18 19  20  21
 11   3   3  1  2  5   1   2   6   2   4   5  3  1  2  4   3   2
-40 -31 -32 27 29 37 -27 -29 -37 -30 -34 -36 32 28 30 34 -33 -28


27 28  29 30 31  32 33  34 35 36 37  38 39  40 41  42 43
 2  3   4  4  5   6  6   8  8 10 11  12 16  21 23  37 60
31 33 -35 35 36 -38 38 -39 39 40 41 -41 42 -42 43 -43  0
```

Side 329:

```
for (k = 0; k <= 26; k++) {
    i = 0; x = 0; j = 1;
    if (count[k] != 0)
        for (t = dad[k]; t != 0; t = dad[t], j += j, i++)
            if (t < 0) { x += j; t = -t; }
    code[k] = x; len[k] = i;
}
```

Kommentar:

Med den afvigende `dad`-tabel bliver udskriften end anden end den nederst på side 328, nemlig

```
0  1 2  3  4 5  6  7 8  9 10 11 12 13 14 15 16 17
7 25 7 48 18 0 49 19 1 17 11 13  6  8 16 10  5  9
3  5 4  6  5 3  6  5 3  5  4  4  4  5  5  4  4  5
```

```
for (j = 0; j < M; j++)
    for (i = len[index(a[j])]; i > 0; i--)
        IO.print((code[index(a[j])] >> i-1) & 1);
```

Kommentar:

Udskriften af koden blev

```
11001111101000110110100010001000111101001011000000
11101100111110101011011101110001110001101110000011
01001000010010111010011010001110110011111110011111
01100111010011011110011000111111010100110110111000
10000111011011000111101110010101011010
```

som er af samme længde som den kode, der præsenteres i bogen (nemlig 236 bit).

Kapitel 24. Elementary Geometric Methods

Side 348:

```
class Point { int x, y, char c; }

class Line { Point p1, p2; }

Point polygon[] = new Point[Nmax];
```

Kommentar:

Det vil være en god ide at tilføje konstruktører til de to klasser:

```
class Point {
    int x, y; char c;

    Point(int x, int y, char c)
        { this.x = x; this.y = y; this.c = c; }
}

class Line {
    Point p1, p2;

    Line(Point p1, Point p2)
        { this.p1 = p1; this.p2 = p2; }
}
```

Punkterne nederst på side 349 kan så skabes således:

```
A = new Point(3,9,'A');   B = new Point(11,1,'B');
C = new Point(6,8,'C');   D = new Point(4,3,'D');
E = new Point(5,15,'E');  F = new Point(8,11,'F');
G = new Point(1,6,'G');   H = new Point(7,4,'H');
I = new Point(9,7,'I');   J = new Point(14,5,'J');
K = new Point(10,13,'K'); L = new Point(16,14,'L');
M = new Point(15,2,'M');  N = new Point(13,16,'N');
O = new Point(3,12,'O');  P = new Point(12,10,'P');
```

Side 350:

```
int ccw(Point p0, Point p1, Point p2) {
    int dx1, dx2, dy1, dy2;
    dx1 = p1.x - p0.x; dy1 = p1.y - p0.y;
    dx2 = p2.x - p0.x; dy2 = p2.y - p0.y;
    if (dx1*dy2 > dy1*dx2) return +1;
    if (dx1*dy2 < dy1*dx2) return -1;
    if (dx1*dx2 < 0 || dy1*dy2 < 0) return -1;
    if ((dx1*dx1+dy1*dy1) < (dx2*dx2+dy2*dy2)) return +1;
    return 0;
}
```

Side 351:

```
boolean intersect(Line l1, Line l2) {
    return ((ccw(l1.p1, l1.p2, l2.p1)
             *ccw(l1.p1, l1.p2, l2.p2)) <= 0)
        && ((ccw(l2.p1, l2.p2, l1.p1)
             *ccw(l1.p1, l1.p2, l1.p2)) <= 0);
}
```

Kommentar:

Metoden returnerer en boolsk værdi (og *ikke* et heltal som i C++-udgaven).

Side 353:

```
float theta(Point p1, Point p2) {
    int dx, dy, ax, ay;
    float t;
    dx = p2.x - p1.x; ax = Math.abs(dx);
    dy = p2.y - p1.y; ay = Math.abs(dy);
    t = (ax+ay) == 0 ? 0 : (float) dy/(ax+ay);
    if (dx < 0) t = 2-t; else if (dy < 0) t = 4+t;
    return (float) t*90.0;
}
```

Kommentar:

Begge konverteringer til typen `float` er nødvendige.

Side 354:

```
boolean inside(Point t, Point p[], int N) {
    int i, count = 0, j = 0;
    Line lt = new Line(), lp = new Line();
    p[0] = p[N]; p[N+1] = p[1];
    lt.p1 = t; lt.p2 = t; lt.p2.x = Integer.MAX_VALUE;
    for (i = 1; i <= N; i++) {
        lp.p1 = p[i]; lp.p2 = p[i];
        if (!intersect(lp,lt)) {
            lp.p2 = p[j]; j = i;
            if (intersect(lp,lt)) count++;
        }
    }
    return (count & 1) == 1;
}
```

Kommentar:

Metoden returnerer en boolsk værdi (og *ikke* et heltal som i C++-udgaven).

Kapitel 25. Finding the Convex Hull

Side 364:

```
int wrap(Point p[], int N) {
    int i, min, M;
    float th, v;
    for (min = 0, i = 1; i < N; i++)
        if (p[i].y < p[min].y) min = i;
    p[N] = p[min]; th = 0.0F; th = 0.0F;
    for (M = 0; M < N; M++) {
        swap(p, M, min);
        min = N; v = th; th = 360.0F;
        for (i = M+1; i <= N; i++)
           if (theta(p[M],p[i]) > v)
               if (theta(p[M],p[i]) < th)
                  { min = i; th = theta(p[M],p[min]); }
           if (min == N) return M;
    }
    return -1;
}
```

Kommentarer:

(1) Java kræver at metoden altid returner en værdi. Java kan ikke bevise, at metoden altid vil returne ved sætningen `return M`. Sætningen `return -1` tager højde for dette problem.

(2) Metoden `swap` kan programmeres således:

```
void swap(Point p[], int i, int j)
    { Point t = p[i]; p[i] = p[j]; p[j] = t; }
```

(3) Det er misvisende, når der over figuren på side 363 står tallene 1 til 16. Det skulle have været tallene fra 0 til 15. Et kald af `wrap` kan se ud som følger:

```
p[0] = A;    p[1] = B;  p[2] = C;  p[3] = D;
p[4] = E;    p[5] = F;  p[6] = G;  p[7] = H;
p[8] = I;    p[9] = J; p[10] = K; p[11] = L;
p[12] = M;  p[13] = N; p[14] = O; p[15] = P;
M = wrap(p,16);
```

Side 368:

```
int grahamscan(Point p[], int N) {
    int i, min, M;
    for (min = 1, i = 2; i <= N; i++)
        if (p[i].y < p[min].y) min = i;
    for (i = 1; i <= N; i++)
        if (p[i].y == p[min].y)
           if (p[i].x > p[min].x) min = i;
    swap(p, 1, min);
    shellsort(p, N);
    p[0] = p[N];
    for (M = 3, i = 4; i <= N; i++) {
        while (ccw(p[M], p[M-1], p[i]) >= 0) M--;
            M++; swap(p, i, M);
    }
    return M;
}
```

Kommentar:

Metoden `shellsort` omskrives til at kunne sortere punkter ved hjælp af en sammelingningsfunktion, `greater`:

```
boolean greater(Point P, Point Q, Point p1) {
     return theta(P,p1) > theta(Q,p1);
}

void shellsort(Point a[], int N) {
    int i, j, h; Point v;
    for (h = 1; h <= N/9; h = 3*h+1) ;
    for ( ; h > 0; h /= 3)
        for (i = h+1; i <= N; i++) {
           v = a[i]; j = i;
           while (j > h && greater(a[j-h],v,a[1]))
               { a[j] = a[j-h]; j -= h; }
           a[j] = v;
       }
}
```

Kapitel 26. Range Searching

Side 374:

```
int range(int v1, int v2)
    { return ranger(head.r,v1,v2); }

int ranger(Node t, int v1, int v2) {
    boolean tx1, tx2;
    int count = 0;
    if (t == z) return 0;
    tx1 = (t.key >= v1);
    tx2 = (t.key <= v2);
    if (tx1) count += ranger(t.l, v1, v2);
    if (tx1 && tx2) count++;
    if (tx2) count += ranger(t.r, v1, v2);
    return count;
}
```

Kommentar:

Variablerne `tx1` og `tx2` er af typen boolean.

Side 378:

```
class Node {
    Point p;
    Node next;
}

public class Range {
    private int maxG, size;
    private Node grid[][];
    private Node z;

    Range(int maxG, int size) {
        this.maxG = maxG; this.size = size;
        grid = new Node[maxG][maxG];
        int i, j;
        z = new Node();
        for (i = 0; i <= maxG; i++)
            for (j = 0; j <= maxG; j++)
                grid[i][j] = z;
    }

    public void insert(Point p) {
        Node t = new Node();
        t.p = p; t.next = grid[p.x/size][p.y/size];
        grid[p.x/size][p.y/size] = t;
    }
}
```

Kommentar:

Størrelserne `maxG` og `size` er her parametre til konstruktøren.

Side 379:

```
public int search(Rect range) {
    Node t;
    int i, j, count = 0;
    for (i = range.x1/size; i <= range.x2/size; i++)
        for (j = range.y1/size; j <= range.y2/size; j++)
           for (t = grid[i][j]; t != z; t = t.next)
               if (insiderect(t.p, range)) count++;
    return count++;
}
```

Kommentar:

Nedenfor ses en udgave af klassen `Rect` og metoden `insiderect`:

```
class Rect {
    int x1, x2, y1, y2;
}

boolean insiderect(Point P, Rect r) {
    return P.x >= r.x1 && P.x <= r.x2 &&
           P.y >= r.y1 && P.y <= r.y2;
}
```

Side 382:

```
class Node {
    Point p;
    Node l, r;
}

public class Range {
    private Node z, head;
    private Node dummy;

    public void insert(Point p) {
        Node f = null, t = null;
        boolean d, td = false;
        for (d = false, t = head; t != z; d = !d) {
            td = d ? (p.x < t.p.x) : (p.y < t.p.y);
            f = t; t = td ? t.l : t.r;
        }
        t = new Node(); t.p = p; t.l = z; t.r = z;
        if (td) f.l = t; else f.r = t;
    }
}
```

Kommentarer:

(1) Variablerne `d` og `td` er af typen boolean.

(2) Der er fejl i Sedgewicks kode! Udtrykket `d != d` bør erstattes med `d = !d`, eller tilsvarende (f.eks. `d ^= 1`).

Side 383:

```
public int search(Rect range)
    { return searchr(head.r, range, true); }

private int searchr(Node t, Rect range, boolean d) {
    boolean t1, t2, tx1, tx2, ty1, ty2;
    int count = 0;
    if (t == z) return 0;
    tx1 = range.x1 < t.p.x; tx2 = t.p.x <= range.x2;
    ty1 = range.y1 < t.p.y; ty2 = t.p.y <= range.y2;
    t1 = d ? tx1 : ty1; t2 = d ? tx2 : ty2;
    if (t1) count += searchr(t.l, range, !d);
    if (insiderect(t.p, range)) count++;
    if (t2) count += searchr(t.r, range, !d);
    return count;
}
```

Kapitel 27. Geometric Intersection

Side 394:

```
class Globals {
    static final int Nmax = 1000;
    static Dict Xtree = new Dict(Nmax),
                Ytree = new Dict(Nmax);
    static Line lines[] = new Line[Nmax];
    static int count = 0;
}

int intersections(){
    int x1, y1, x2, y2, N; Line l;
    for (N = 1; !IO.eof(); N++) {
        x1 = IO.readInt(); y1 = IO.readInt();
        x2 = IO.readInt(); y2 = IO.readInt();
        l = Globals.lines[N] = new Line();
        l.p1 = new Point(); l.p2 = new Point();
        l.p1.x = x1; l.p1.y = y1;
        l.p2.x = x2; l.p2.y = y2;
        Globals.Ytree.insert(y1,N);
        if (y2 != y1) Globals.Ytree.insert(y2,N);
    }
    Globals.Ytree.traverse();
    return Globals.count;
}
```

Kommentar:

Java giver ikke mulighed for brug af lokale variable, med mindre de er placeret i en klasse, og tilgås ved priknotation. I tilfældet her er de globale variable placeret som klasse-atributter i klassen `Globals`.

Side 395:

```
private void visit(int v, int info) {
    int t, x1, x2, y1, y2; Line l = Globals.lines[info];
    x1 = l.p1.x; y1 = l.p1.y;
    x2 = l.p2.x; y2 = l.p2.y;
    if (x2 < x1) { t = x2; x2 = x1; x1 = t; }
    if (y2 < y1) { t = y2; y2 = y1; y1 = t; }
    if (v == y1)
        Globals.Xtree.insert(x1,info);
    if (v == y2) {
        Globals.Xtree.remove(x1,info);
        Globals.count += Globals.Xtree.range(x1,x2);
    }
}
```

Kommentar:

Nedenfor er angivet en færdig udgave af class `Dict`. Metoden `remove` er blevet indført, da den ikke fandtes i forvejen.

```
class Dict {
    private Node head, z;

    Dict(int max) {
       z = new Node(0,0,null,null);
       head = new Node(Integer.MIN_VALUE,0,null,z);
    }

    public int search(int v) {
        Node x = head.r;
        z.key = v;
        while (v != x.key)
        x = (v < x.key) ? x.l : x.r;
        return x.info;
    }

    public void insert(int v, int info) {
        Node p = head, x = head.r;
        while (x != z)
            { p = x; x = (v < x.key) ? x.l : x.r; }
        x = new Node(v, info, z, z);
        if (v < p.key) p.l = x; else p.r = x;
    }
```

```
public void delete(int v) {
    Node c, p, x, t;
    z.key = v;
    p = head; x = head.r;
    while (v != x.key)
        { p = x; x = (v < x.key) ? x.l : x.r; }
    t = x;
    if (t.r == z) x = x.l;
    else if (t.r.l == z) { x = x.r; x.l = t.l;
    else {
         c = x.r; while (x.l.l != z) c = c.l;
         x = c.l; c.l = x.r;
         x.l = t.l; x.r = t.r;
    }
    if (v < p.key) p.l = x; else p.r = x;
}

public void remove(int v, int info) {
    Node c, p, x, t;
    z.key = v; z.info = info;
    p = head; x = head.r;
    while (v != x.key && info != x.info)
        { p = x; x = (v < x.key) ? x.l : x.r; }
    t = x;
    if (t.r == z) x = x.l;
    else if (t.r.l == z) { x = x.r; x.l = t.l; }
    else {
        c = x.r; while (x.l.l != z) c = c.l;
        x = c.l; c.l = x.r;
        x.l = t.l; x.r = t.r;
    }
    if (v < p.key) p.l = x; else p.r = x;
}

private void inorder(Node t) {
    if (t != z) {
       inorder(t.l);
       visit(t.key, t.info);
       inorder(t.r);
   }
}

public int range(int v1, int v2)
    { return ranger(head.r,v1,v2); }
```

```
    private int ranger(Node t, int v1, int v2) {
        boolean tx1, tx2;
        int count = 0;
        if (t == z) return 0;
        tx1 = (t.key >= v1);
        tx2 = (t.key <= v2);
        if (tx1) count += ranger(t.l, v1, v2);
        if (tx1 && tx2) count++;
        if (tx2) count += ranger(t.r, v1, v2);
        return count;
    }

    private void visit(int v, int info) { ... } // se ovenfor

    public void traverse() { inorder(head.r); }
}
```

Kapitel 28. Closest-Point Problems

Side 404:

```
int comp(Node t)
    { return (pass == 1) ? t.p.x : t.p.y; }

Node merge(Node a, Node b) {
    Node c;
    c = z;
    do
        if (comp(a) < comp(b))
           { c.next = a; c = a; a = a.next; }
        else
           { c.next = b; c = b; b = b.next; }
    while (c != z);
    c = z.next; z.next = z;
    return c;
}
```

```
void check(Point p1, Point p2) {
    float dist;
    if (p1.y != z.p.y && p2.y != z.p.y) {
       dist = (float) Math.sqrt((p1.x-p2.x)*(p1.x-p2.x)
                              + (p1.y-p2.y)*(p1.y-p2.y));
       if (dist < min)
           { min = dist; cp1 = p1; cp2 = p2; }
    }
}
```

Side 405:

```
Node sort(Node c, int N) {
    int i;
    Node a, b;
    float middle = 0;
    Point p1, p2, p3, p4;
    if (c.next == z) return c;
    a = c;
    for (i = 2; i <= N/2; i++) c = c.next;
    b = c.next; c.next = z;
    if (pass == 2) middle = b.p.x;
    c = merge(sort(a, N/2), sort(b, N-(N/2)));
    if (pass == 2) {
        p1 = p2 = p3 = p4 = z.p;
        for (a = c; a != z; a = a.next)
            if (Math.abs(a.p.x-middle) < min) {
                check(a.p,p1);
                check(a.p,p2);
                check(a.p,p3);
                check(a.p,p4);
                p1 = p2; p2 = p3; p3 = p4; p4 = a.p;
            }
    }
    return c;
}
```

Kommentar:

Hvis punktkoordinaterne som her alle er af typen `int`, kunne `middle` også have typen `int`.

Side 406:

```
z = new Node(); z.p.x = z.p.y = max; z.next = z;
h = new Node(); h.next = readlist();
min = max;
pass = 1; h.next = sort(h.next, N);
pass = 2; h.next = sort(h.next, N);
```

Kommentar:

Hvis klassen `Node` er defineret som følger

```
class Node {
    Point p;
    Node next;

    Node() {}
        Node(Point p, Node next)
           { this.p = p; this.next = next; }
}
```

kunne en initialisering med bogens punkter `A`, `B`, …, `P` ske således:

```
int max = Integer.MAX_VALUE;
z = new Node(); z.p.x = z.p.y = max; z.next = z;
Node n = z;
n = new Node(P,n); n = new Node(O,n); n = new Node(N,n);
n = new Node(M,n); n = new Node(L,n); n = new Node(K,n);
n = new Node(J,n); n = new Node(I,n); n = new Node(H,n);
n = new Node(G,n); n = new Node(F,n); n = new Node(E,n);
n = new Node(D,n); n = new Node(C,n); n = new Node(B,n);
n = new Node(A,n);
Node h = new Node(null,n);
```

Kapitel 29. Elementary Graph Algorithms

Side 420:

```
int V, E;
int a[][] = new int[maxV][maxV];
char v1, v2;

int index(char c) { return c - 'A' + 1; }

void adjmatrix() throws IOException {
    int j, x, y;
    V = IO.readInt();
    E = IO.readInt();
    for (x = 1; x <= V; x++)
        for (y = 1; y <= V; y++) a[x][y] = 0;
    for (x = 1; x <= V; x++) a[x][x] = 1;
    for (j = 1; j <= E; j++) {
        String xy = IO.readToken();
        v1 = xy.charAt(0); v2 = xy.charAt(1);
        x = index(v1); y = index(v2);
        a[x][y] = a[y][x] = 1;
    }
}
```

Kommentarer:

(1) Metoden `index` er implementeret som foreslået i bogen (s. 420).

(2) En mere kompakt repræsentation af matricen ville være at lade elementerne være af typen `boolean`.

(3) Udskrivning af matricen kan foretages med nedenstående metode:

```
void printadjmatrix() {
    for (int x = 1; x <= V; x++) {
        for (int y = 1; y <= V; y++)
           IO.print(a[x][y] + " ");
        IO.println();
    }
}
```

Side 421:

```
void adjlist() throws IOException {
    int j, x, y; Node t;
    V = IO.readInt();
    E = IO.readInt();
    z = new Node(); z.next = z;
    for (j = 1; j <= V; j++) adj[j] = z;
    for (j = 1; j <= E; j++) {
        String xy = IO.readToken();
        v1 = xy.charAt(0);
        v2 = xy.charAt(1);
        x = index(v1); y = index(v2);
        t = new Node();
        t.v = x; t.next = adj[y]; adj[y] = t;
        t = new Node();
        t.v = y; t.next = adj[x]; adj[x] = t;
    }
}
```

Kommentar:

Udskrivning af nabolisterne kan foretages med nedenstående metode:

```
void printadjlist() {
    for (int i = 1; i <= V; i++) {
        IO.print(i + ": ");
        for (Node t = adj[i]; t != z; t = t.next)
           IO.print(t.v + " ");
        IO.println();
    }
}
```

Side 424:

```
void search() {
    int k;
    for (k = 1; k <= V; k++) val[k] = unseen;
    for (k = 1; k <= V; k++)
        if (val[k] == unseen) visit(k);
}

void visit(int k) { // DFS, adjacency lists
    val[k] = ++id;
    for (Node t = adj[k]; t != z; t = t.next)
        if (val[t.v] == unseen) visit(t.v);
}
```

Kommentar:

Variablerne `unseen` og `id` kan erklæres således:

```
final int unseen = 0;
int id = 0;
```

Side 427:

```
void visit(int k) { // DFS, adjacency matrix
    val[k] = ++id;
    for (int t = 1; t <= V; t++)
        if (a[k][t] != 0)
           if (val[t] == unseen) visit(t);
}
```

Side 428:

```
Stack stack = new Stack(maxV);

void visit(int k) { // DFS, adjacency lists
    stack.push(k);
    while (!stack.empty()) {
        k = stack.pop(); val[k] = ++id;
        for (Node t = adj[k]; t != z; t = t.next)
            if (val[t.v] == unseen)
                { stack.push(t.v); val[t.v] = -1; }
    }
}
```

Side 431:

```
Queue queue = new Queue(maxV);

void visit(int k) { // DFS, adjacency lists
    queue.put(k);
    while (!queue.empty()) {
        k = queue.get(); val[k] = ++id;
        for (Node t = adj[k]; t != z; t = t.next)
            if (val[t.v] == unseen)
                { queue.put(t.v); val[t.v] = -1; }
    }
}
```

Kapitel 30. Connectivity

Side 440:

```
int visit(int k) { // DFS to find articulation points
    Node t;
    int m, min;
    val[k] = ++id;
    min = id;
    for (t = adj[k]; t != z; t = t.next)
        if (val[t.v] == unseen) {
            m = visit(t.v);
            if (m < min) min = m;
            if (m >= val[k]) IO.println(name[k]);
        }
        else if (val[t.v] < min) min = val[t.v];
    return min;
}
```

Kommentar:

Navnetabellen `name` kan oprettes således:

```
char name[] =
   {' ','A','B','C','D','E','F','G','H','I','J','K','L','M'};
```

Side 444:

```
public class EQ {
    private int dad[];

    EQ(int size) { dad = new int[size]; }

    public boolean find(int x, int y, boolean doit) {
        int i = x, j = y;
        while (dad[i] > 0) i = dad[i];
        while (dad[j] > 0) j = dad[j];
        if (doit && i != j) dad[j] = i;
        return (i != j);
    }
}
```

Side 447:

```
public boolean find(int x, int y, boolean doit) {
    int t, i = x, j = y;
    while (dad[i] > 0) i = dad[i];
    while (dad[j] > 0) j = dad[j];
    while (dad[x] > 0)
        { t = x; x = dad[x]; dad[t] = i; }
    while (dad[y] > 0)
        { t = y; y = dad[y]; dad[t] = j; }
    if (doit && i != j)
        if (dad[j] < dad[i])
            { dad[j] += dad[i] -1; dad[i] = j; }
        else
            { dad[i] += dad[j] - 1; dad[j] = i; }
    return (i != j);
}
```

Kapitel 31. Weighted Graphs

Side 455:

```
void visit(int k) { // PFS, adjacency lists
    Node t;
    if (pq.update(k, -unseen)) dad[k] = 0;
    while (!pq.empty()) {
        id++; k = pq.remove(); val[k] = -val[k];
        if (val[k] == -unseen) val[k] = 0;
        for (t = adj[k]; t != z; t = t.next)
            if (val[t.v] < 0)
                if (pq.update(t.v, priority)) {
                    val[t.v] = -(priority);
                    dad[t.v] = k;
                }
    }
}
```

Kommentar:

En mulig implementation af metode `update` i klassen `PQ` er

```
public boolean update(int x, int v) {
    if (p[x] == 0) {
        insert(x, v);
        return true;
    }
    else if (a[p[x]] > v) {
        change(x, v);
        return true;
    }
    return false;
}
```

En forudsætning er dog at følgende programlinje

```
p[info[N]] = 0;
```

indsættes før returhoppet i metoden `remove`.

Side 460:

```
class Edge {
    char v1, v2;
    int w;
}

static void kruskal() throws IOException {
    int i, m, V, E;
    Edge e[] = new Edge[maxE];
    PQ pq = new PQ(maxE); EQ eq = new EQ(maxE);
    V = IO.readInt();
    E = IO.readInt();
    for (i = 1; i <= E; i++) {
        String S = IO.readToken();
        e[i].v1 = S.charAt(0);
        e[i].v2 = S.charAt(1);
        e[i].w = IO.readInt();
    }
    for (i = 1; i <= E; i++)
        pq.insert(i, e[i].w);
    for (i = 0; i < V-1; ) {
        if (pq.empty()) break;
        m = pq.remove();
        if (eq.find(index(e[m].v1),index(e[m].v2),true))
            { IO.print(e[m].v1 + "," + e[m].v2 + " "); i++; }
    }
    IO.println();
}
```

Kommentar:

(1) Det er forkert, når der om C++-udgaven står, at en procedure `edgefound` kaldes, hver gang en kant i det minimale udspændende træ findes! I programmet *udskrives* disse kanter.

(2) Kanterne indtastes på formen `A B W`, hvor `A` og `B` angiver kantens to knuder, og `W` angiver dens vægt (et heltal).

Side 466:

```
void search() { // PFS, adjacency matrix
    int k, t, min = 0;
    for (k = 1; k <= V; k++)
       { val[k] = unseen; dad[k] = 0; }
    val[0] = unseen;
    for (k = 1; k != 0; k = min, min = 0) {
        val[k] = -val[k];
        if (val[k] == -unseen) val[k] = 0;
        for (t = 1; t <= V; t++)
            if (val[t] < 0) {
                if (a[k][t] > 0 && (val[t] < -priority))
                    { val[t] = -priority; dad[t] = k; }
                if (val[t] > val[min]) min = t;
            }
    }
}
```

Kapitel 32. Directed Graphs

Side 474:

```
for (k = 1; k <= V; k++) {
    id = 0;
    for (j = 1; j <= V; j++) val[j] = 0;
    visit(k);
    IO.println();
}
```

Side 475:

```
for (y = 1; y <= V; y++)
    for (x = 1; x <= V; x++)
        if (a[x][y])
           for (j = 1; j <= V; j++)
               if (a[y][j]) a[x][j] = true;
```

Side 477:

```
for (y = 1; y <= V; y++)
    for (x = 1; x <= V; x++)
        if (a[x][y] > 0)
            for (j = 1; j <= V; j++)
                if (a[y][j] > 0)
                    if (a[x][j] == 0 ||
                        a[x][y] + a[y][j] < a[x][j])
                        a[x][j] = a[x][y] + a[y][j];
```

Side 482:

```
Stack stack = new Stack(maxV);

int visit(int k) { // DFS to find strong components
    Node t; int m, min;
    val[k] = ++id; min = id;
    stack.push(k);
    for (t = adj[k]; t != z; t = t.next) {
        m = val[t.v] == 0 ? visit(t.v) : val[t.v];
        if (m < min) min = m;
    }
    if (min == val[k]) {
        do {
            m = stack.pop();
            IO.print(m + " ");
            val[m] = V+1;
        } while (m != k);
        IO.println();
    }
    return min;
}
```

Kapitel 33. Network Flow

Side 490:

```
priority = -flow[k][t];
if (size[k][t] > 0) priority += size[k][t];
if (priority > val[k]) priority = val[k];
```

Side 491:

```
for (;;) {
    if (!search(1,V)) break;
    y = V; x = dad[V];
    while (x != 0) {
        flow[x][y] = flow[x][y]+val[V];
        flow[y][x] = -flow[x][y];
        y = x; x = dad[y];
    }
}
```

Kapitel 34. Matching

Side 503:

```
for (m = 1; m <= N; m++) {
    for (s = m; s != 0; ) {
        next[s]++; w = prefer[s][next[s]];
        if (rank[w][s] < rank[w][fiancee[w]])
            { t = fiancee[w]; fiancee[w] = s; s = t; }
    }
}
```

Kapitel 35. Random Numbers

Side 511:

```
a[0] = seed;
for (i = 1; i <= N; i++)
    a[i] = (a[i-1]*b+1) % m;
```

Side 513:

```
import IO.*;

class Random {
    private int a;
    private static final int m = 100000000;
    private static final int m1 = 10000;
    private static final int b = 31415821;

    public Random(int seed) { a = seed; }

    public int next() {
        a = (mult(a,b) + 1) % m;
        return a;
    }

    private int mult(int p, int q) {
        int p1, p0, q1, q0;
        p1 = p/m1; p0 = p%m1;
        q1 = q/m1; q0 = q%m1;
        return (((p0*q1+p1*q0) % m1)*m1+p0*q0) % m;
    }
}

public class Program {
    public static void main(String args[]) {
    Random x = new Random(1234567);
    int N = IO.readInt();
    for (int i = 1; i <= N; i++)
        IO.print(x.next() + " ");
        IO.println();
    }
}
```

Kommentarer:

(1) Da Java ikke som C++ tillader globale variable, konstanter og procedurer, er disse flyttet ind i klassen `Random`.

(2) Nedenfor ses et tilsvarende hovedprogram, der benytter sig af Javas *indbyggede* klasse `Random`.

```
import java.util.*;
import IO.*;

public class Program {
    public static void main(String args[]) {
        int i, N;
        Random x = new Random(1234567);
        N = IO.readInt();
        for (i = 1; i <= N; i++)
            IO.println(x.nextInt() + " ");
        IO.println();
    }
}
```

En kørsel med `N` = 5 gav følgende udskrift:

```
1042961893 -1571432423 -1065072994 1922548996 1868514696
```

Side 514:

```
public int badnext(int r) {
    int b = 31415821;
    a = (mult(a, b) + 1) % m;
    return a % r;
}
```

```
public int next(int r) {
    int b = 31415821;
    a = (mult(a,b) + 1) % m;
    return ((a/m1)*r)/m1;
}
```

Side 516-517:

```
class Random {
    private static final int m = 100000000;
    private static final int m1 = 10000;
    private int a[] = new int[55], j;

    public Random(int seed) {
        int b = 3145821;
        a[0] = seed;
        for (j = 1; j <= 54; j++)
            a[j] = (mult(a[j-1], b) + 1) % m;
    }

    public int next(int r) {
        j = (j+1) % 55;
        a[j] = (a[(j+23) % 55]+a[(j+54) % 55]) % m;
        return ((a[j]/m1)*r)/m1;
    }

    private int mult(int p, int q) {
        int p1, p0, q1, q0;
        p1 = p/m1; p0 = p%m1;
        q1 = q/m1; q0 = q%m1;
        return (((p0*q1+p1*q0) % m1)*m1+p0*q0) % m;
    }
}
```

Side 518:

```
float chisquare(int N, int r) {
    int i, t, f[] = new int[rmax];
    Random x = new Random(1234567);
    for (i = 0; i < r; i++) f[i] = 0;
    for (i = 0; i < N; i++) f[x.next(r)]++;
    for (i = 0, t = 0; i < r; i++) t += f[i]*f[i];
    return (float) r*t/N - N;
}
```

Kommentar:

I C++-udgaven er parenteserne i det udtryk, der returneres, fejlplaceret! De bør fjernes, hvis resultatet skal kunne blive et kommatal.

Kapitel 36. Arithmetic

Side 522:

```
for (i = 0; i < 2*N-1; i++) r[i] = 0;
for (i = 0; i < N; i++)
    for (j = 0; j < N; j++)
        r[i+j] += p[i]*q[j];
```

Side 523:

```
Node add(Node p, Node q) {
    Node t;
    t = z; z.c = 0;
    while ((p != z) && (q != z)) {
        t.next = new Node();
        t = t.next; t.c = p.c + q.c;
        p = p.next; q = q.next;
    }
    t.next = z; t = z.next; z.next = z;
    return t;
}
```

Side 524:

```
class Node {
    int c, j;
    Node next;
}

Node insert(Node t, int c, int j) {
    t.next = new Node();
    t = t.next; t.c = c; t.j = j;
    return t;
}
```

```
Node add(Node p, Node q) {
    Node t;
    t = z; z.c = 0; z.j = maxN;
    while (p != z && q != z) {
        if (p.j == q.j && (p.c + q.c) != 0) {
            t = insert(t, p.c+q.c, p.j);
            p = p.next; q = q.next;
        }
        else
        if (p.j < q.j)
            { t = insert(t, p.c, p.j); p = p.next; }
        else if (q.j < p.j)
            { t = insert(t, q.c, q.j); q = q.next; }
    }
    t.next = z; t = z.next; z.next = z;
    return t;
}
```

Side 529:

```
float[] mult(float p[], float q[], int N) {
    float p1[] = new float[N/2], q1[] = new float[N/2],
          ph[] = new float[N/2], qh[] = new float[N/2],
          t1[] = new float[N/2], t2[] = new float[N/2];
    float r[] = new float[2*N-1], rl[], rm[], rh[];
    int i ,N2;
    if (N == 1)
        { r[0] = p[0]*q[0]; return r; }
    for (i = 0; i < N/2; i++)
        { p1[i] = p[i]; q1[i] = q[i]; }
    for (i = N/2; i < N; i++)
        { ph[i-N/2] = p[i]; qh[i-N/2] = q[i]; }
    for (i = 0; i < N/2; i++) t1[i] = p1[i]+ph[i];
    for (i = 0; i < N/2; i++) t2[i] = q1[i]+qh[i];
    rm = mult(t1, t2, N/2);
    rl = mult(p1, q1, N/2);
    rh = mult(ph, qh, N/2);
    for (i = 0; i < N-1; i++) r[i] = rl[i];
    r[N-1] = 0;
    for (i = 0; i < N-1; i++) r[N+i] = rh[i];
    for (i = 0; i < N-1; i++)
        r[N/2+i] += rm[i] - (rl[i]+rh[i]);
    return r;
}
```

Kommentarer:

(1) Dette program er et lovligt Java-program, hvorimod koden i bogen ikke er et lovligt C++-program.

(2) Nedenstående Java-kode multiplicerer de to polynomier i bogen (side 528).

```
float q[] = {1,1,3,-4};
float p[] = {1,2,-5,-3};
float r[] = mult(p,q,4);
```

Side 532:

```
for (i = 0; i < N; i++)
    for (j = 0; j < N; j++)
        for (k = 0, r[i][j] = 0; k < N; k++)
            r[i][j] += p[i][k]*q[k][j];
```

Kapitel 37. Gaussian Elimination

Side 538:

```
for (i = 1; i <= N; i++)
    for (j = i+1; j <= N; j++)
        for (k = N+1; k >= i; k--)
            a[j][k] -= a[i][k]*a[j][i]/a[i][i];
```

Side 539:

```
void eliminate() {
    int i, j, k, max;
    float t;
    for (i = 1; i <= N; i++) {
        max = i;
        for (j = i+1; j <= N; j++)
            if (Math.abs(a[j][i]) > Math.abs(a[max][i]))
                max = j;
        for (k = i; k <= N+1; k++) {
            t = a[i][k];
            a[i][k] = a[max][k];
            a[max][k] = t;
        }
        for (j = i+1; j <= N; j++)
            for (k = N+1; k >= i; k--)
                a[j][k] -= a[i][k]*a[j][i]/a[i][i];
    }
}
```

Side 540:

```
void substitute() {
    int j, k;
    float t;
    for (j = N; j >= 1; j--) {
        t = 0.0F;
        for (k = j+1; k <= N; k++) t += a[j][k]*x[k];
        x[j] = (a[j][N+1]-t)/a[j][j];
    }
}
```

Side 543:

```
for (i = 1; i < N; i++) {
    a[i+1][N+1] -= a[i][N+1]*a[i+1][i]/a[i][i];
    a[i+1][i+1] -= a[i][i+1]*a[i+1][i]/a[i][i];
}
for (j = N; j >= 1; j--)
    x[j] = (a[j][N+1]-a[j][j+1]*x[j+1])/a[j][j];
```

Kapitel 38. Curve Fitting

Side 549:

```
void makespline(float x[], float y[], int N) {
    int i;
    for (i = 2; i < N; i++) d[i] = 2*(x[i+1]-x[i-1]);
    for (i = 1; i < N; i++) u[i] = x[i+1]-x[i];
    for (i = 2; i < N; i++)
        w[i] = 6.0F*((y[i+1]-y[i])/u[i]
                    -(y[i]-y[i-1])/u[i-1]);
    p[1] = 0.0F; p[N] = 0.0F;
    for (i = 2; i < N-1; i++) {
        w[i+1] -= w[i]*u[i]/d[i];
        d[i+1] -= u[i]*u[i]/d[i];
    }
    for (i = N-1; i > 1; i--)
        p[i] = (w[i]-u[i]*p[i+1])/d[i];
}
```

Side 550:

```
float f(float x)
    { return x*x*x - x; }

float eval(float v) {
    float t; int i = 1;
    while (v > x[i+1]) i++;
    t = (v-x[i])/u[i];
    return t*y[i+1]+(1-t)*y[i] +
       u[i]*u[i]*(f(t)*p[i+1]+f(1-t)*p[i])/6.0F;
}
```

Side 553:

```
for (i = 1; i <= M; i++)
    for (j = 1; j <= M+1; j++) {
        t = 0.0;
        for (k = 1; k <= N; k++)
            t += f[i][k]*f[j][k];
        a[i][j] = t;
    }
```

Kapitel 39. Integration

Side 557:

```
double intrect(double a, double b, int N) {
    int i; double r = 0; double w = (b-a)/N;
    for (i = 1; i <= N; i++) r += w*f(a-w/2+i*w);
    return r;
}
```

Side 559:

```
double inttrap(double a, double b, int N) {
    int i; double t = 0; double w = (b-a)/N;
    for (i = 1; i <= N; i++)
        t += w*(f(a+(i-1)*w)+f(a+i*w))/2;
    return t;
}
```

Side 561:

```
double intsimp(double a, double b, int N) {
    int i; double s = 0; double w = (b-a)/N;
    for (i = 1; i <= N; i++)
        s += w*(f(a+(i-1)*w) +
                4*f(a-w/2+i*w) +
                f(a+i*w))/2;
    return s;
}
```

Side 563:

```
double adapt(double a, double b) {
    double x = intsimp(a, b, 10);
    if (Math.abs(x - intsimp(a, b, 5)) > tolerance)
        return adapt(a, (a+b)/2) + adapt((a+b)/2, b);
    return x;
}
```

Kapitel 41. The Fast Fourier Transform

Side 590:

```
eval(p, outN, 0);
eval(q, outN, 0);
for (i = 0; i <= outN; i++) r[i] = p[i].times(q[i]);
eval(r, outN, 0);
for (i = 1; i <= N; i++)
    { Complex t = r[i]; r[i] = r[outN+1-i]; r[outN+1-i] = t; }
for (i = 0; i <= outN; i++) r[i] = r[i].div(outN+1);
```

Kommentar:

Java tillader ikke overlæsning af operatorer. Dermed kan implementationen ikke blive så elegant som i C++.

En mulig løsning i Java er at definere en klasse, `Complex`, med metoder svarende til regenoperationerne på komplekse tal. På næste side ses et forslag til denne klasse.

```
class Complex {
    double re, im;

    Complex(double r, double i) { re = r; im = i; }
    Complex(double r) { re = r; im = 0; }

    Complex plus(Complex c)
        { return new Complex(re+c.re, im+c.im); }
    Complex plus(double d)
        { return new Complex(re+d, im); }

    Complex minus(Complex c)
       { return new Complex(re-c.re, im-c.im); }
    Complex minus(double d)
       { return new Complex(re-d, im); }

    Complex times(Complex c) {
        return new Complex(re*c.re-im*c.im, re*c.im+im*c.re);
    }
    Complex times(double d)
       { return new Complex(re*d, im*d); }

    Complex div(Complex c) {
        double d = c.re*c.re+c.im*c.im;
        return new Complex((re*c.re+im*c.im)/d,
                           (im*c.re-re*c.im)/d);
    }
    Complex div(double d) { return new Complex(re/d, im/d); }

    void print() {IO.print(re + " + " + im +"i"); }
}
```

Som det ses, har resultatet af hver af regneoperationerne form af et nyt `Complex`-objekt. Denne objektgenerering (og tilsvarende spildopsamling) kan tidsmæssigt være en dyr affære, men i det mindste virker beregningerne som ønsket.

Side 591:

```
void eval(Complex p[], int N, int k) {
    int  i, j;
    if (N == 1) {
        Complex p0 = p[k], p1 = p[k+1];
        p[k] = p0.plus(p1); p[k+1] = p0.minus(p1);
    }
    else {
        for (i = 0; i <= N/2; i++) {
            j = k+2*i;
            t[i] = p[j]; t[i+1+N/2] = p[j+1];
      }
      for (i = 0; i <= N; i++) p[k+i] = t[i];
      eval(p, N/2, k);
      eval(p, N/2, k+1+N/2);
      j = (outN+1)/(N+1);
      for (i = 0; i <= N/2; i++) {
          Complex p0 = w[i*j].times(p[k+(N/2)+1+i]);
          t[i] = p[k+i].plus(p0);
          t[i+(N/2)+1] = p[k+i].minus(p0);
      }
      for (i = 0; i <= N; i++) p[k+i] = t[i];
    }
}
```

Kommentar:

Der er fejl i Sedgewicks implementation! Kaldet `eval(p,N/2,(k+1+N)/2)` i Sedgewicks udgave skal erstattes med `eval(p,N/2,k+1+N/2)`. I ovenstående udgave, som udnytter klassen `Complex` (se kommentar til side 590), er fejlen rettet.

Multiplikationen af to polynomier, `p` og `q`, kan for eksempel varetages af en metode `multiply`, som vist på næste side.

```
Complex[] multiply(Complex p[], Complex q[]) {
    int i, j;
    int N = p.length/2;
    outN = 2*N-1;
    w = new Complex[N];
    for (j = 0; j < N; j++)
        w[j] = new Complex(Math.cos(2*Math.PI*j/(outN+1)),
                           Math.sin(2*Math.PI*j/(outN+1)));
    r = new Complex[2*N];
    t = new Complex[2*N];
    eval(p, outN, 0);
    eval(q, outN, 0);
    for (i = 0; i <= outN; i++) r[i] = p[i].times(q[i]);
    eval(r, outN, 0);
    for (i = 1; i <= N; i++) {
        Complex t = r[i];
        r[i] = r[outN+1-i];
        r[outN+1-i] = t;
    }
    for (i = 0; i <= outN; i++) r[i] = r[i].div(outN+1);
    return r;
}
```

Et program, der multiplicerer $p(x) = 2x^7 + x^6 - x^5 + x^4 + x^2 + x + 1$ med $q(x) = 3x^7 - x^6 + x^5 + x^3 + x^2 - x + 2$ ved hjælp af `multiply`, er givet nedenfor. Programmet udskriver koefficienterne til det resulterende produktpolynomium.

```
public static void main(String args[]) {
    int N = 7+1;
    Complex p[] = new Complex[2*N];
    Complex q[] = new Complex[2*N];

    p[0] = new Complex(1); p[1] = new Complex(1);
    p[2] = new Complex(1); p[3] = new Complex(3);
    p[4] = new Complex(1); p[5] = new Complex(-1);
    p[6] = new Complex(1); p[7] = new Complex(2);
    for (int i = N; i < 2*N; i++) p[i] = new Complex(0);

    q[0] = new Complex(2); q[1] = new Complex(-1);
    q[2] = new Complex(1); q[3] = new Complex(1);
    q[4] = new Complex(0); q[5] = new Complex(1);
    q[6] = new Complex(-1); q[7] = new Complex(3);
    for (int i = N; i < 2*N; i++) q[i] = new Complex(0);

    Complex r[] = multiply(p, q);
    for (int i = 0; i < 2*N-1; i++) IO.print(r[i].re + " ");
        IO.println();
}
```

Kapitel 42. Dynamic Programming

Side 596:

```
for (j = 1; j <= N; j++) {
    for (i = 1; i <= M; i++)
        if (i >= size[j])
            if (cost[i] < cost[i-size[j]]+val[j]) {
                cost[i] = cost[i-size[j]]+val[j];
                best[i] = j;
            }
}
```

Side 600:

```
for (i = 1; i <= N; i++)
    for (j = i+1; j <= N; j++)
        cost[i][j] = Integer.MAX_VALUE;
for (i = 1; i <= N; i++) cost[i][i] = 0;
for (j = 1; j < N; j++) {
    for (i = 1; i <= N-j; i++)
        for (k = i+1; k <= i+j; k++) {
            t = cost[i][k-1]+cost[k][i+j]
                            +r[i]*r[k]*r[i+j+1];
            if (t < cost[i][i+j])
                { cost[i][i+j] = t; best[i][i+j] = k; }
        }
}
```

Side 601:

```
void order(int i, int j) {
    if (i == j) IO.print(name(i));
    else {
        IO.print("(");
        order(i, best[i][j]-1);
        order(best[i][j], j);
        IO.print(")");
    }
}
```

Side 603:

```
for (i = 1; i <= N; i++)
    for (j = i+1; j <= N+1; j++)
        cost[i][j] = Integer.MAX_VALUE;
for (i = 1; i <= N; i++) cost[i][i] = f[i];
for (i = 1; i <= N+1; i++) cost[i][i-1] = 0;
for (j = 1; j <= N-1; j++) {
    for (i = 1; i <= N-j; i++) {
        for (k = i; k <= i+j; k++) {
            t = cost[i][k-1]+cost[k+1][i+j];
            if (t < cost[i][i+j])
                { cost[i][i+j] = t; best[i][i+j] = k; }
        }
        for (k = i; k <= i+j; cost[i][i+j] += f[k++]) ;
    }
}
```

Kapitel 43. Linear Programming

Side 617:

```
void pivot(int p, int q) {
    int j, k;
    for (j = 0; j <= N; j++)
        for (k = M+1; k >= 1; k--)
           if (j != p && k != q)
               a[j][k] -= a[p][k]*a[j][q]/a[p][q];
    for (j = 0; j <= N; j++)
        if (j != p) a[j][q] = 0;
    for (k = 1; k <= M+1; k++)
        if (k != q) a[p][k] /= a[p][q];
    a[p][q] = 1;
}
```

Side 619:

```
for (;;) {
    for (q = 0; (q <= M+1) && (a[0][q] >= 0); q++) ;
    if (q > M) break;
    for (p = 0; (p <= N+1) && (a[p][q] <= 0); p++) ;
    if (p > N) break;
    for (i = p+1; i <= N; i++)
        if (a[i][q] > 0)
           if (a[i][M+1]/a[i][q] < a[p][M+1]/a[p][q])
               p = i;
    pivot(p, q);
}
```

Kommentar:

Ved direkte oversættelse af Sedgewicks C/C++-udgave til Java, opstår der indeksfejl, såfremt `q` bliver større end `M`. Derfor beskyttes der i Java-udgaven mod denne fejl ved hjælp af sætningen: `if (q > M) break`.

Kapitel 44. Exhaustive Search

Side 623:

```
void visit(int k) {
    int t;
    val[k] = ++id;
    for (t = 1; t <= V; t++)
        if (a[k][t])
            if (val[t] == 0) visit(t);
    id--; val[k] = 0;
}
```

Side 629:

```
void visit(int k) {
    int t;
    val[k] = ++id;
    if (id == V) writeperm();
    for (t = 1; t <= V; t++)
        if (val[t] == 0) visit(t);
    id--; val[k] = 0;
}
```

Appendiks. Klassen IO

Class `IO` er en Java-klasse til simpel terminalorienteret indlæsning og udskrivning. Formålet med klassen er at give en Java-programmør let adgang til indlæsning fra tastatur og formateret udskrivning på skærm.

Her er et eksempelprogram, der illustrerer brugen af class `IO`. Programmet indlæser en række heltal fra tastaturet og udskriver til slut deres sum.

```
import IO.*;

public class Adder {
    public static void main(String args[]) {
        int Sum = 0;
        while (!IO.eof())
           Sum += IO.readInt();
        IO.println(Sum);
    }
}
```

Et tilsvarende program, der benytter sig af Javas indbyggede klasser, kan for eksempel se således ud:

```
import java.io.*;

public class  Adder{
    public static void main(String args[]) {
        int Sum = 0;
       StreamTokenizer input = new StreamTokenizer(System.in);
       try {
           while (input.nextToken() != StreamTokenizer.TT_EOF)
               Sum += input.nval;
       }
       catch (IOException e) {}
       System.out.println(Sum);
 }
```

Udviklingen af dette sidste program kræver godt kendskab til Javas undtagelsesbegreb (exceptions) og til klassen `StreamTokenizer`. Det første program er derimod er let at skrive.

Et andet problem, som class IO løser, er Javas mangel på faciliteter til formateret udskrivning. Antag for eksempel, at der ønskes udskrevet en tabel over den naturlige logaritmefunktion, ln(x), for x = 0.25, 0.5, 1, ... , 2.5. Et fragment af et Java-program, der løser denne opgave, ville være følgende:

```
for (int i = 1; i <= 10; i++) {
    double x = i/4.0;
    System.out.println(x + " " + Math.log(x));
}
```

Ved en kørsel fås imidlertid følgende noget uoverskuelige udskrift:

```
0.25 -1.38629
0.5 -0.693147
0.75 -0.287682
1 0
1.25 0.223144
1.5 0.4054
1.75 0.559616
2 0.693147
2.25 0.81093
2.5 0.916291
```

En bedre opstilling af tabellen fås ved anvendelse af udskrivningsmetoden `print` i class `IO`. Et kald `print(x,n,w)` bevirker, at tallet `x` udskrives højrejusteret med `n` decimaler i et tekstfelt på `w` tegn. Nedenfor er tabeludskrivningen foretaget ved hjælp af denne metode.

```
for (int i = 1; i <= 10; i++) {
    double x = i/4.0;
    IO.print(x,2,6);
    IO.println(Math.log(x),5,10);
}
```

Udskriften blev følgende:

```
  0.25  -1.38629
  0.50   0.69315
  0.75   0.28768
  1.00   0.00000
  1.25   0.22314
  1.50   0.40547
  1.75   0.55962
  2.00   0.69315
  2.25   0.81093
  2.50   0.91629
```

Her følger en kort oversigt over metoderne i class `IO`.

Metoder til brug ved indlæsning:

`eof()`:

Returnerer `true`, når der ikke kan læses flere symboler fra standard input (`System.in`). Ellers `false`.

`readBoolean()`,
`readDouble()`,
`readFloat()`,
`readInt()`,
`readLong()`,
`readToken()`:

Indlæser det næste symbol (`Boolean`, `Double`, `Float`, `Int`, `Long` eller `Token`) fra standard input og returnerer den tilhørende værdi som resultat. Hvis der ikke kan læses flere symboler, udskrives en fejlmeddelelse, og programmet terminerer.

Metoder til brug ved udskrivning:

`print(x)`:

Et kald svarer til `System.out.print(x)`.

`print(int x, int w)`,
`print(long x, int w)`:

Udskriver heltallet `x` højrejusteret i et tekstfelt af bredde `w` tegn. Såfremt feltet ikke kan rumme tallet, udskrives det i et felt bestående af det nødvendige antal tegn.

`print(float x, int n, int w)`,
`print(double x, int n, int w)`:

Udskriver det flydende tal `x` med `n` decimaler højrejusteret i et tekstfelt af bredde `w` tegn. Såfremt feltet ikke kan rumme tallet, udskrives det i et felt bestående af det nødvendige antal tegn.

`printReal(float x, int n, int w)`,
`printReal(double x, int n, int w)`:

Udskriver det flydende tal `x` i "videnskabelig notation" med `n` betydende cifre og højrejusteret i et tekstfelt af bredde `w` tegn. Såfremt feltet ikke kan rumme tallet, udskrives det i et felt bestående af det nødvendige antal tegn.

Tallet udskrives med et ciffer mellem 1 og 9 (inklusive) foran punktummet efterfulgt af `n`-1 decimaler og en eksponentdel. For tal af typen `float` indeholder eksponentdelen 2 cifre, mens den for tal af typen `double` indeholder 3 cifre.

Eksempel. Hvis tabeludskrivningsprogrammet ændres til

```
IO.printReal(x,4,12);
IO.printlnReal(Math.log(x),6,20);
```

fås følgende udskrift

```
        2.500e-001      -1.38629e+000
        5.000e-001      -6.93147e-001
        7.500e-001      -2.87682e-001
        1.000e+000       0.00000e-001
        1.250e+000       2.23144e-001
        1.500e+000       4.54650e-001
        1.750e+000       5.59616e-001
        2.000e+000       6.93147e-001
        2.250e+000       8.10930e-001
        2.500e+000       9.16291e-001
```

`println(...)`, `printlnReal(...)`:

Udskriver som `print()` og `printReal()`, idet der dog afsluttes med et linjeskift.

`flush()`:

Har samme virkning som `System.out.flush()`.

Kildeteksten til IO

```
import java.io.*;

public class IO {
    static StreamTokenizer in;

    static {
        in = new StreamTokenizer(
                 new BufferedReader(
                     new InputStreamReader(System.in)));
        in.resetSyntax();
        in.wordChars('a', 'z');
        in.wordChars('A', 'Z');
        in.wordChars('0', '9');
        in.wordChars(128 + 32, 255);
        in.whitespaceChars(0, ' '); ();
        in.wordChars('+', '+');
        in.wordChars('-', '-');
        in.wordChars('.', '.');
    }

    private static String formatDouble(double d, int n, int w) {
        if (n < 0)
            n = 0;
        double wholePart = d >= 0 ? Math.floor(d) : Math.ceil(d);
        StringBuffer s1 = new StringBuffer(
                                formatLong((long) wholePart, w-n-1));
        double decimalPart = (int) Math.round(
                                    (d - wholePart)* Math.pow(10, n));
        if (decimalPart < 0) decimalPart = -decimalPart;
        StringBuffer s2 = new StringBuffer(
                                 Long.toString((long) decimalPart));
        if (s2.length() > n)
            s2.setLength(n);
        else
            while (s2.length() < n)
                s2.append('0');
            return s1.toString() + "." + s2.toString();
    }
```

```
private static String formatReal(double d, int n, int w, int e) {
    if (n <= 1)
        n = 1;
    double p = d == 0 ? 0 :
                  (int) (Math.log(Math.abs(d))/Math.log(10));
    d /= Math.pow(10, p);
    if (Math.abs(d) < 1) { d *= 10; p--; }
    double wholePart = d >= 0 ? Math.floor(d) : Math.ceil(d);
    StringBuffer s1 = new StringBuffer(
                        formatLong((long) wholePart, w-n-3-e));
    double decimalPart = (int) Math.round(
                            (d - wholePart)*Math.pow(10, n-1));
    if (decimalPart < 0) decimalPart = -decimalPart;
    StringBuffer s2 = new StringBuffer(
                           Long.toString((long) decimalPart));
    if (s2.length() > n-1)
        s2.setLength(n);
    else
        while (s2.length() < n-1)
            s2.append('0');
   StringBuffer s3 = new StringBuffer(
                           Long.toString((long) Math.abs(p)));
   while (s3.length() < e)
        s3.insert(0, '0');
   s3.insert(0, p >= 0 ? '+' : '-');
   return s1.toString() + "." + s2.toString() + "e" + s3 ;
}

public static boolean eof() {
    try {
        in.nextToken();
        in.pushBack();
        return in.ttype == in.TT_EOF;
    }
    catch (EOFException e) { return true; }
    catch (IOException e) { return true; }
}

public static boolean readBoolean() {
    try {
        in.nextToken();
        if (in.ttype == in.TT_EOF)
            throw new EOFException();
    }
    catch (EOFException e) { eofError(); }
    catch (IOException e) { readError("readBoolean"); }
    return Boolean.valueOf(in.sval).booleanValue();
}
```

```
public static double readDouble() {
    try {
        in.nextToken();
        if (in.ttype == in.TT_EOF)
            throw new EOFException();
    }
    catch (EOFException e) { eofError(); }
    catch (IOException e) { readError("readDouble"); }
    return Double.valueOf(in.sval).doubleValue();
}

public static float readFloat() {
    try {
        in.nextToken();
        if (in.ttype == in.TT_EOF)
            throw new EOFException();
    }
    catch (EOFException e) { eofError(); }
    catch (IOException e) { readError("readFloat"); }
    return Double.valueOf(in.sval).floatValue();
}

public static int readInt() {
    try {
        in.nextToken();
        if (in.ttype == in.TT_EOF)
            throw new EOFException();
    }
    catch (EOFException e) { eofError(); }
    catch (IOException e) { readError("readInt"); }
    return Integer.valueOf(in.sval).intValue();
}

public final static String readLine()  {
    try {
        return new BufferedReader(
                    new InputStreamReader(
                        new DataInputStream(System.in))).
                            readLine();
    }
    catch (EOFException e) { eofError(); }
    catch (IOException e) { readError("readInt"); }
    return "@";
}

public static long readLong() {
    try {
        in.nextToken();
        if (in.ttype == in.TT_EOF)
            throw new EOFException();
    }
    catch (EOFException e) { eofError(); }
    catch (IOException e) { readError("readLong"); }
    return Long.valueOf(in.sval).longValue();
}
```

```
    public static String readToken() {
        try {
            in.nextToken();
            if (in.ttype == in.TT_EOF)
                throw new EOFException();
        }
        catch (EOFException e) { eofError(); }
        catch (IOException e) { readError("readToken"); }
        return in.sval;
    }

    public static void print(Object obj) { System.out.print(obj); }
    public static void print(String s) { System.out.print(s); }
    public static void print(char[] s) { System.out.print(s); }
    public static void print(char c) { System.out.print(c); }
    public static void print(int i) { System.out.print(i); }
    public static void print(long l) { System.out.print(l); }
    public static void print(float f) { System.out.print(f); }
    public static void print(double d) { System.out.print(d); }
    public static void print(boolean b) { System.out.print(b); }
    public static void println() { System.out.println(); }
    public static void println(Object obj) { System.out.println(obj); }
    public static void println(String s) { System.out.println(s); }
    public static void println(char[] s) { System.out.println(s); }
    public static void println(char c) { System.out.println(c); }
    public static void println(int i) { System.out.println(i); }
    public static void println(long l) { System.out.println(l); }
    public static void println(float f) { System.out.println(f); }
    public static void println(double d) { System.out.println(d); }
    public static void println(boolean b) { System.out.println(b); }

    public static void print(int i, int w)
        { System.out.print(formatLong((long) i, w)); }
    public static void print(long l, int w)
        { System.out.print(formatLong(l, w)); }
    public static void print(float f, int n, int w)
        { System.out.print(formatDouble((float) f, n, w)); }
    public static void print(double d, int n, int w)
        { System.out.print(formatDouble(d, n, w)); }
    public static void println(int i, int w)
        { System.out.println(formatLong((long) i, w)); }
    public static void println(long l, int w)
        { System.out.println(formatLong(l, w)); }
    public static void println(float f, int n, int w)
        { System.out.println(formatDouble((float) f, n, w)); }
    public static void println(double d, int n, int w)
        { System.out.println(formatDouble(d, n, w)); }
    public static void printReal(float f, int n, int w)
        { System.out.print(formatReal((double) f, n, w, 2)); }
    public static void printReal(double d, int n, int w)
        { System.out.print(formatReal(d, n, w, 3)); }
    public static void printlnReal(float f, int n, int w)
        { System.out.println(formatReal((double) f, n, w, 2)); }
    public static void printlnReal(double d, int n, int w)
        { System.out.println(formatReal(d, n, w, 3)); }
```

```
    public static void flush() { System.out.flush(); }

    private static void eofError()
        { throw new RuntimeException(
                            "Attempt to read past end of file"); }

    private static void readError(String method)
        { throw new RuntimeException(method + ": read error"); }

    private static String formatLong(long l, int w) {
        StringBuffer s = new StringBuffer(Long.toString(l));
        while (s.length() < w)
            s.insert(0, ' ');
        return s.toString();
    }
}
```

Indhold