TDA 545: Objektorienterad programmering

Föreläsning 12:
Exempel och problemlösning

Magnus Myréen

Chalmers, läsperiod 1, 2015-2016

Idag

Problemlösning, dvs hur man "ska tänka" för att hitta lösning

- int mängd/set (att skriva klass, arrays, testning)
- testning av Goldbachs hypotes (arrays)
- quicksort (arrays, rekursion)

Idag skriver vi koden interaktivt.

Kom med förslag. Fråga frågor!

Koden som skrivs kommer upp på websidan efter lektionen.

Nästa två föreläsningar: swing, grafik, händelser; läs kap 19

IntSet

```
/** This class creates *immutable* objects that
 * represent sets of integer numbers. */
public class IntSet {

    // ...

    /** Creates a new IntSet object containing the
     * integers that appear in the content array. */
    public IntSet(int[] content) {
        // ...
    }

    /** Returns true if n is a member of the set,
     * otherwise it returns false. */
    public boolean hasInt(int n) {
        // ...
    }

}
```

a) *implementera* klassens metoder [6 poäng]
b) *skriv* ett enkelt testprogram [3 poäng]
c) *förklara* hur man kan optimera koden [6 poäng]

IntSet, del (a)

```
public class IntSet {

    int[] a;  // always non-null

    public IntSet(int[] content) {
        if (content == null) {
            a = new int[0];
        } else {
            a = new int[content.length];
            for (int i=0; i < a.length; i++) {
                a[i] = content[i];
            }
        }
     }

    public boolean hasInt(int n) {
         for (int i=0; i < a.length; i++) {
            if (a[i] == n) {
                return true;
            }
        }
        return false;
     }

}
```

IntSet, del (b)

```
public static void main(String[] args) {
    int[] arr = {2,6,1,7,9,2,34,7};
    IntSet s = new IntSet(arr);
    System.out.println("s.hasInt(0) = " + s.hasInt(0));
    arr[5] = 0;
    System.out.println("s.hasInt(0) = " + s.hasInt(0));
    System.out.println("s.hasInt(2) = " + s.hasInt(2));
}
```

IntSet, del (c)

```
public class IntSet {

    int[] a; // always non-null and sorted

    public IntSet(int[] content) {
        if (content == null) {
            a = new int[0];
        } else {
            a = new int[content.length];
            for (int i=0; i < a.length; i++) {
                a[i] = content[i];
            }
            BubbleSort.sort(a);
        }
    }

    public boolean hasInt(int n) {
        return BinarySearch.lookup(n,a);
    }

}
```

Obs: uppgiften krävde inte konkret kod, men här ger jag koden.

Goldbachs hypotes

(från övningstenta 2006)

Hypotesen:

"Alla jämna heltal (större än 2) kan skrivas som en summa av två primtal."

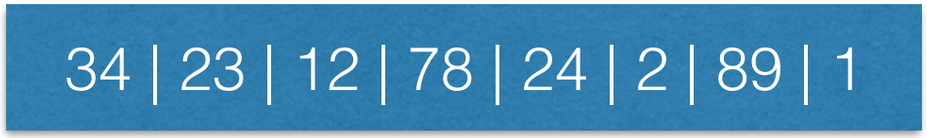
Skriv ett program som kan användas för att testa denna hypotes.

```
public class Goldbach {

    public static boolean isPrime(int n) {
        for (int i=2;i<n;i++) {
            if (n % i == 0) {
                return false;
            }
        }
        return true;
    }

    public static void findSum(int n) {
        if (n % 2 == 1) { return; }
        for (int p = (n/4*2)+1; p > 0; p--) {
            if (isPrime(p) && isPrime(n-p)) {
                System.out.println(n + " = " + p + " + " + (n-p));
                return;
            }
        }
        System.out.println("Goldbach was wrong!");
     }

     public static void main(String[] args) {
        findSum(8);
        findSum(50);
        findSum(500);
    }

}
```

Implementera Quicksort

Så här fungerar Quicksort:

steg 1: Välj ett element i arrayn, t.ex. 34.



steg 2: Dela arrayn i två: en del med elementen mindre än 34, och en del med de som är större än 34.

steg 3: Kör Quicksort rekursivt på de två delarna av arrayn.

Video: https://www.youtube.com/watch?v=kPRAoW1kECg&t=39

```
public class Quicksort {

    private static void swap(int i, int j, int[] arr) {
        int ti = arr[i];
        int tj = arr[j];
        arr[i] = tj;
        arr[j] = ti;
    }

    private static int partition(int b, int e, int[] arr) {
        int k = arr[b];
        int i = b+1;
        int j = e;
        while (i < j) {
            if (arr[i] < k) {
                i = i+1;
            } else {
                swap(i,j-1,arr);
                j = j-1;
            }
        }
        swap(b,i-1,arr);
        return i-1;
    }

    private static void qsortAux(int b, int e, int[] arr) {
        if (b+1 < e) {
            int k = partition(b,e,arr);
            qsortAux(b,k,arr);
            qsortAux(k+1,e,arr);
        }
    }

    public static void qsort(int[] arr) {
        qsortAux(0,arr.length,arr);
    }

    public static void main(String[] args) {
        int[] arr = { 3, 1, 5, 4, 6, 2, 8, -3, 15, 67 };
        System.out.println("\nBefore qsort:");
        for (int k=0;k<arr.length;k++) {
            System.out.println("  arr["+k+"] = " + arr[k]);
        }
        qsort(arr);
        System.out.println("\nAfter qsort:");
        for (int k=0;k<arr.length;k++) {
            System.out.println("  arr["+k+"] = " + arr[k]);
        }
        System.out.println("");
    }

}
```

Quicksort
utan debug kod

```
public class QuicksortOO {

    private static void swap(int i, int j, Comparable[] arr) {
        Comparable ti = arr[i];
        Comparable tj = arr[j];
        arr[i] = tj;
        arr[j] = ti;
    }

    private static int partition(int b, int e, Comparable[] arr) {
        Comparable k = arr[b];
        int i = b+1;
        int j = e;
        while (i < j) {
            if (arr[i].compareTo(k) < 0) {
                i = i+1;
            } else {
                swap(i,j-1,arr);
                j = j-1;
            }
        }
        swap(b,i-1,arr);
        return i-1;
    }

    private static void qsortAux(int b, int e, Comparable[] arr) {
        if (b+1 < e) {
            int k = partition(b,e,arr);
            qsortAux(b,k,arr);
            qsortAux(k+1,e,arr);
        }
    }

    public static void qsort(Comparable[] arr) {
        qsortAux(0,arr.length,arr);
    }

    public static void main(String[] args) {
        Circle[] arr = {
           new Circle(3), new Circle(1), new Circle(5),
           new Circle(4), new Circle(6), new Circle(67),
           new Circle(8), new Circle(15), new Circle(2) };
        System.out.println("\nBefore qsort:");
        for (int k=0;k<arr.length;k++) {
            System.out.println("  arr["+k+"] = " + arr[k]);
        }
        qsort(arr);
        System.out.println("\nAfter qsort:");
        for (int k=0;k<arr.length;k++) {
            System.out.println("  arr["+k+"] = " + arr[k]);
        }
        System.out.println("");
    }

}
```

Quicksort som sorterar Objekt

```
public class Circle implements Comparable {
    private int radius;
    public Circle(int radius) {
        this.radius = radius;
    }
    public String toString() {
        return "<< circle with radius " + radius + " >>";
    }
    public int compareTo(Object o) {
        if (o instanceof Circle) {
            Circle other = (Circle)o;
            int otherRadius = other.radius;
            return this.radius - other.radius;
        } else {
            return -1;
        }
    }
}
```

Utskrift:

```
Before qsort:
  arr[0] = << circle with radius 3 >>
  arr[1] = << circle with radius 1 >>
  arr[2] = << circle with radius 5 >>
  arr[3] = << circle with radius 4 >>
  arr[4] = << circle with radius 6 >>
  arr[5] = << circle with radius 67 >>
  arr[6] = << circle with radius 8 >>
  arr[7] = << circle with radius 15 >>
  arr[8] = << circle with radius 2 >>

After qsort:
  arr[0] = << circle with radius 1 >>
  arr[1] = << circle with radius 2 >>
  arr[2] = << circle with radius 3 >>
  arr[3] = << circle with radius 4 >>
  arr[4] = << circle with radius 5 >>
  arr[5] = << circle with radius 6 >>
  arr[6] = << circle with radius 8 >>
  arr[7] = << circle with radius 15 >>
  arr[8] = << circle with radius 67 >>
```