Tentamen – extra del

Förklaringar

Den här delen av tentamen gör möjligt att få betyg högre än E

Utöver den obligatoriska delen, kan studenten göra även den här delen av tentamen. Denna del har sin betydelse bara i fall att studenten har klarat den obligatoriska delen. I så fall kan studenten samla tillräckligt antal poäng på den här delen, och uppnå ett betyg som är högre än E.

Antalet poäng och betyg

Totalt: 20 poäng

För betyget D räcker med: 4 poäng
För betyget C räcker med: 8 poäng
För betyget B räcker med: 14 poäng
För betyget A räcker med: 17 poäng

Uppgifter

Uppgift 1 (4 poäng + 1 poäng)

ETT PROBLEM

Det finns i ett trafiksystem en startstation, en destinationsstation, och n mellanstationer. Mellanstationerna är numrerade från och med 1 till och med n, där n är ett positivt heltal.

Det går att ta sig från startstationen till destinationsstationen genom vilken som helst av mellanstationerna. Längden av vägen mellan startstationen och mellanstationen nummer i ($i = 1, 2, ..., n$) är a_i. Längden av vägen mellan mellanstationen nummer i ($i = 1, 2, ..., n$) och destinationsstationen är b_i.

En mellanstation ska väljas, så att vägen mellan startstationen och destinationsstationen blir så kort som möjligt.

EN ALGORITM

a) Hitta en minneseffektiv algoritm som löser det givna problemet. Beskriv denna algoritm med pseudokod.

b) Hur kan man få en algoritm, som löser givna problemet, vars minneskomplexitet är $\theta(n)$?

Uppgift 2 (5 poäng)

Man använder klasserna `Worker` och `Supporter`:

```
Worker    worker = new Worker (10);
Worker.Supporter    supporter = worker.new Supporter (0.5);
worker.work (8);
supporter.support ("actively");
```

När det här kodavsnittet utförs, erhålls följande utskrift:

```
working 10 x 8 hours
supporting 0.5 x 10 days actively
```

En annan utskrift erhålls när följande kodavsnitt utförs.

```
Worker    worker = new Worker (60);
Worker.Supporter    supporter = worker.new Supporter (0.75);
worker.work (7);
supporter.support ("passively");
```

I det här fallet blir utskriften:

```
working 60 x 7 hours
supporting 0.75 x 60 days passively
```

Skapa klasserna `Worker` och `Supporter`.

Uppgift 3 (1 poäng + 2 poäng + 2 poäng)

Ett gränssnitt, `StringCollection`, definierar en samling teckensträngar.

En klass, `NodeStringCollection`, representerar en samling teckensträngar. En sådan samling är definierad i gränssnittet `StringCollection`. Element i samlingen lagras i en sekvens av noder.

```
class NodeStringCollection implements StringCollection
{
    private static class Node
    {
        public String    element;
        public Node      nextNode;

        public Node (String element)
        {
            this.element = element;
            this.nextNode = null;
        }
    }

    private Node    firstNode;

    public NodeStringCollection ()
    {
        firstNode = null;
    }

    // metoden size här

    // metoden add här

    public String toString ()
    {
        String    s = "";
        Node    currentNode = firstNode;
        while (currentNode != null)
        {
            s = s + currentNode.element + " ";
            currentNode = currentNode.nextNode;
        }

        return s;
    }
}
```

Gränssnittet `StringCollection` och klassen `NodeStringCollection` kan användas så här:

```
StringCollection    sc = new NodeStringCollection ();
sc.add (new String ("A"));
sc.add (new String ("B"));
sc.add (new String ("C"));
sc.add (new String ("D"));
System.out.println (sc);

int    countStrings = sc.size ();
System.out.println (countStrings);
```

När den här kodsekvensen exekveras, erhålls följande utskrift:

```
D C B A
4
```

a) Skapa gränssnittet `StringCollection`.

b) Skapa metoden `size`.

c) Skapa metoden `add`.

Uppgift 4 (2 poäng + 1 poäng + 2 poäng)

Det finns en sekvens med n $(n \in N, n > 0)$ element. En metod, `sort`, sorterar element i den sekvensen.

```
public static void sort (int[] elements)
{
    int    last = elements.length - 1;
    int    current = 0;

    int    t = 0;
    while (current < last)
    {
        for (int pos = last; pos > current; pos--)
        {
            if (elements[pos] < elements[pos - 1])
            {
                t = elements[pos - 1];
                elements[pos - 1] = elements[pos];
                elements[pos] = t;
            }
        }

        current++;
    }
}
```

a) En elementjämförelse kan betraktas som en elementär operation i den algoritm som används i metoden `sort`. Bestäm i så fall algoritmens tidskomplexitet – bestäm motsvarande komplexitetsfunktion. Det ska framgå hur komplexitetsfunktionen erhålls.

b) Till vilken Θ–mängd tillhör den erhållna komplexitetsfunktionen. Varför?

c) Även ett elementutbyte kan betraktas som en elementär operation i algoritmen. Bestäm då algoritmens tidskomplexitet – i bästa fall och i värsta fall.