KTH, NADA, Linda Kann

2D1320, Tilda, lösningsskiss 20 okt 2003

1. *Soppträd*

(5p) Om trädet inte är tomt:
Om rotens datum är efter 1969: skriv ut vänsterträdet.
Om rotens datum ligger mellan 1970 och 1979: skriv ut soppan i roten.
Om rotens datum ligger före 1980: skriv ut högerträdet.

(2p) När nya soppor sorteras in efter datum hamnar alla längst till höger i trädet, vilket ger linjär sökning för datum efter år 2000.

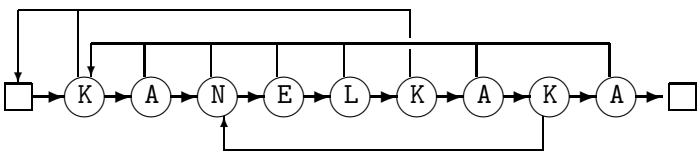
(3p) Om detta utspelar sig i början av år 2060 och om vi bortser från framtida recept så finns det ca 110*12=1320 soppor att hasha in. Det räcker med 1320 platser om vi väljer hashfunktionen:

```
(år - 1950)*12 + månad
```

2. *Stacka plättar*

(3p) Poppa en plätt i taget och pusha på andra stacken tills första stacken är tom. Poppa en plätt från andra stacken, kolla om den är acceptabel och ät isåfall upp den, lägg den annars på första stacken. Fortsätt så tills andra stacken är tom. Då bör alla goda plättar vara uppätna och tallriken kan ställas ner på golvet till hunden.

(4p) **3.** *Baka kaka*



i	next[i]
1	0
2	1
3	1
4	1
5	1
6	0
7	1
8	3
9	1

4. *Restaurangsortering och -sökning*

(2p) Välj till exempel urvalssortering eftersom den är lätt att implementera. Att den inte är så snabb gör inget eftersom sorteringen inte görs varje dag.

(2p) För $n = 1000$ är tidsåtgången $k_1 \cdot 1000$ för linjärsökning och $k_2 \cdot log1000$ för binärsökning. Att binärsökning går 100 gånger snabbare innebär att $\frac{k_1 \cdot 1000}{k_2 \cdot log1000} = 100$.

Om antalet restauranger är tusen gånger större får vi: $\frac{k_1 \cdot 1000 \cdot 1000}{k_2 \cdot log(1000 \cdot 1000)} = \frac{100 \cdot 1000}{2} =$ femtiotusen gånger snabbare.

5. *Teori*

(10p)

a. Ja, att utnyttja redundans innebär att man tar bort onödig information och det gör man vid textkomprimering.

b. Nej, ett Bloomfilter är olämpligt att använda till ett adressregister eftersom man bara kan få veta om en person finns med, inte själva adressen.

c. Ja, när hashtabellen väl är upplagd kräver en sökning bara drygt en jämförelse, medan binärsökningen kräver logn.

d. Nej, det är inte lämpligt eftersom varupriserna varierar så mycket.

e. Javisst, heapsort sorterar med en prioritetskö.

6. *Smörgåssyntax*

(4p)

* Mackan kan bara ha ett pålägg, inte två.
* Rekursionen för mackan saknas, så den kan inte bli hur stor som helst.
* <bröd> används som symbol men finns inte definierat i något vänsterled.
* Pålägg är definierat som en lång sträng, istället för enstaka bokstäver.

(2p) Ett program som undersöker om mackor följer syntaxen har en metod för varje symbol i vänsterledet (t ex macka(), bröd(), pålägg()). Symboler i högerled i syntaxen ger metodanrop, och rekursiva högerled ger rekursiva anrop. Brott mot syntaxen ger lämpligen upphov till särfall (som fortplantas genom anropskedjan).

7. *Chokladuppdelning*

(8p) Använd djupetförstsökning för att prova alla kombinationer av praliner. Läs in pralinvikterna till en vektor. Börja med en tom presentask och skapa döttrar genom att fylla på med en pralin som har index större än den förra som las i, men se till att totala vikten inte överstiger 250 gram. Om vi har turen att få exakt 250 gram i en dotter kan vi avbryta direkt, annars måste vi gå igenom alla kombinationer och fortlöpande hålla reda på den hittills bästa dottern.

När alla kombinationer gåtts igenom anropar vi en metod som skrver ut bästa dotterns vikt samt alla anmödrars vikter. Samtidigt tar vi reda på vilka index (praliner) som utnyttjats så att vi sen kan skriva ut pralinerna som motsvarar övriga index (för den andra asken), med undantag av den minsta som vi äter upp.

Datstrukturer: En vektor för pralinvikterna, problemträdet, med objekt som ska innehålla totala vikten, index för den senast ilagda pralinen samt en moderspekare. Metoder:läsInVikter, skapaDotter, skrivAsk1, skrivAsk2 Klasser:Huvudklass och en klass för döttrarna.

8. *Generell sortering*

(4p) Skriv en sorteringsmetod som använder gränssnittet Comparable (eller ett egendefinierat gränssnitt med en jämförelsemetod). Sen gör du en klass för maträtter. Och till sist en klass för varje sorteringsnyckel, som implementerar Comparable, innehåller en skräddarsydd jämförelsemetod, och en referens till en maträtt. För varje ny nyckel behöver man nu bara skriva en ny jämförelsemetod.