KTH, NADA, Henrik Eriksson

2D1320, Tilda, Tentamenslösning 10 januari 2002

1. *Prydligt stackade prydnader*

Rekursiv tanke: En stack är ordnad om dom två första orden ligger i rätt ordning och den stack som återstår när ett ord poppats är ordnad.
. . . men stackar med högst ett element är alltid ordnade.

För att tanken ska kunna omsättas direkt i kod måste den formuleras så här:

- Om `isEmpty` returneras `true`.
- Poppa elementet `a`. Om nu `isEmpty` pushas `a` och returneras `true`.
- Om inte `isOrdered` pushas `a` och returneras `false`.
- Poppa elementet `b` och kolla om `a` och `b` låg i rätt ordning.
- Pusha tillbaka `b` och `a` och returnera kollens resultat.

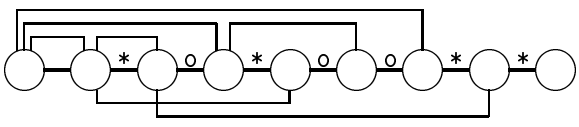
2. *Prydnadskö*

Först lägger man vaktposten `öööööööö` sist i kön. Sedan plockar man ut dom två första orden ur kön. Nu kan bubbelsorteringen börja: Jämför orden, lägg det minsta sist ikön och ta ut ett nytt ord. Håll på tills vaktposten visar sej. Upprepa alltihop ända tills en hel omgång gått igenom välordnad. Kasta då bort vaktposten.

3. *Prydnadssortering*

Eftersom det finns så många tänkbara värden är distributionsräkning dålig (tar lång tid och kräver mycket minne). Urvalssortering tar också lång tid och bloomfilter är oanvändbart för sortering. Mergesort och quicksort är lika bra. Visserligen tar mergesort dubbelt så mycket minne, men det lär inte spela någon roll här. Vad som spelar roll är att man inte anropar hashtabellen mer än en gång för varje prydnad. Man bör spara värdet tillsammans med prydnadens namn under sorteringen. När sorteringen är klar skriver man förstås bara prydnadernas namn på filen.

4. *Tillståndsautomat*



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

Tillståndsföljd: `123012345341212345672345678`

5. *Prydnadspåsar*

Sortera in prydnaderna i en vektor med största värdet först. Summera värdena och dividera med två för att få idealvärdet för påsen.

Problemträdets stamfar är en tom påse. Man skapar söner genom att lägga i en prydnad till i faderspåsen och den ska vara billigare än tidigare prydnader. En sonpost behöver bara innehålla tre fält: summavärdet, sista prydnadens index i vektorn och en faderspekare. Metoden `makeSons` tillverkar bara söner med summa högst lika med idealvärdet. Rekordsonen (bäst hittills) sparas och när hela problemträdet har gåtts igenom kan lösningen skrivas ut med `writeChain(rekordson)`.

Djupetförst är bäst eftersom det inte är kortaste lösningen vi är ute efter.

6. *Prydlig molekyl*

Rekursiv tanke: Avslöjas inte – ingår i labb sju!

7. *Syntax för prydnadsjämförelser*

```
<mening>     ::= Prydnad <lista> är värdefullare än prydnad <lista>.
<lista>      ::= <tal> | <talföljd> och <tal>
<talföljd>   ::= <tal> | <tal>, <talföljd>
<tal>        ::= <1..9> | <tal> <1..9> | <tal> 0
<1..9>       ::= 1|2|3|4|5|6|7|8|9
```

8. *Abstrakta prydnader*

Man vill kunna ändra implementeringen utan att ändra programmen som gäller prydnader.

En abstrakt `Prydnad` kan vara en klass med till exempel följande metoder.

```
Prydnad(String namn, int värde); //Konstruktor
int compareTo(Prydnad prydnad2);
String getNamn();
int getVärde();
```

Kan implementeras med två privata datafält i objektet eller med ett privat namnfält och en hashtabell gemensam för alla prydnader.