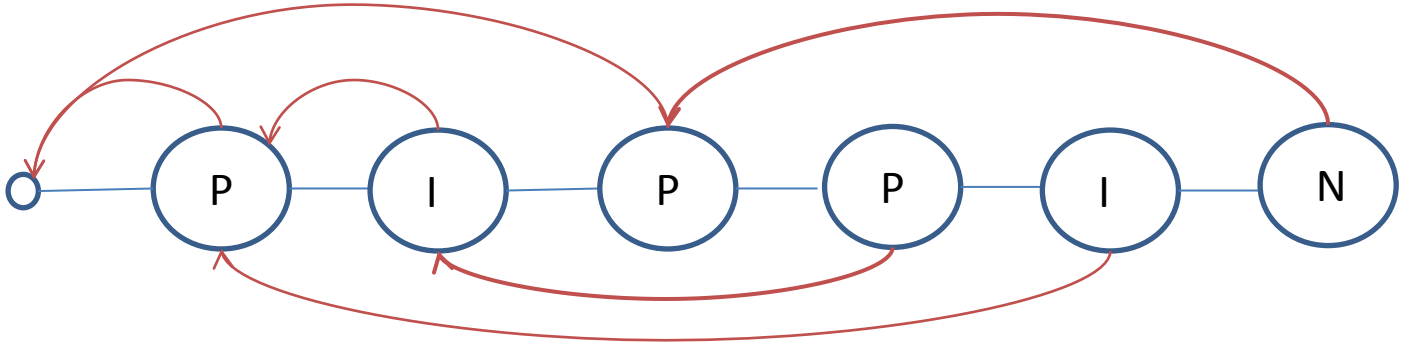
DD1320 Tillämpad datalogi

Lösning (skiss) till tenta 20 okt 2011

1 KMP



i	1	2	3	4	5	6
Next[i]	0	1	0	2	1	3

2 Huffmankodning: Algoritmen

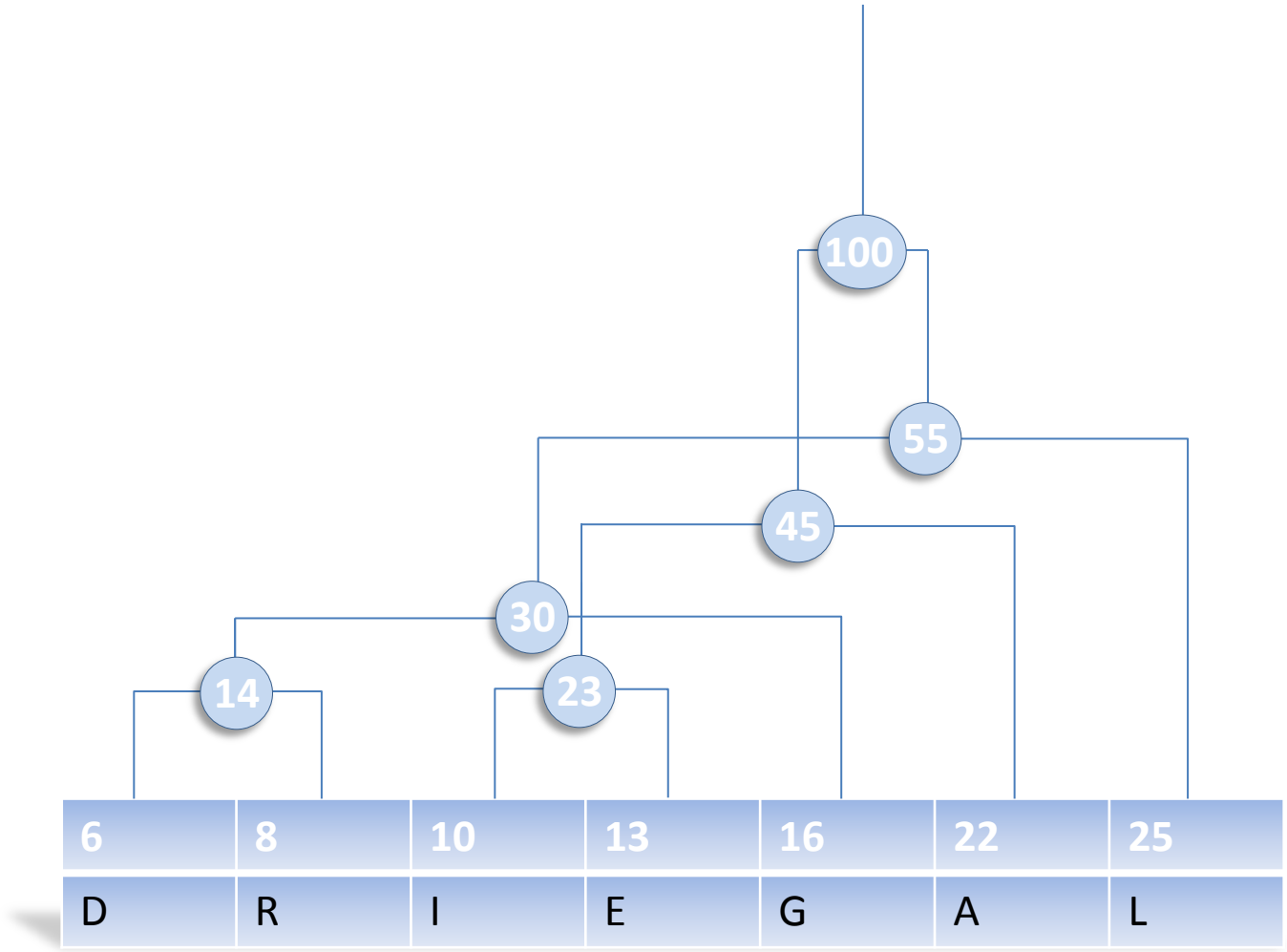
1. Sortera tecknen som ska kodas i stigande förekomstordning.

2. Rita grenar från de två tecken som är ovanligast och låtsas att vi har ett nytt tecken med summan av procentandelen. Numrera ena grenen med 0 och andra med 1.

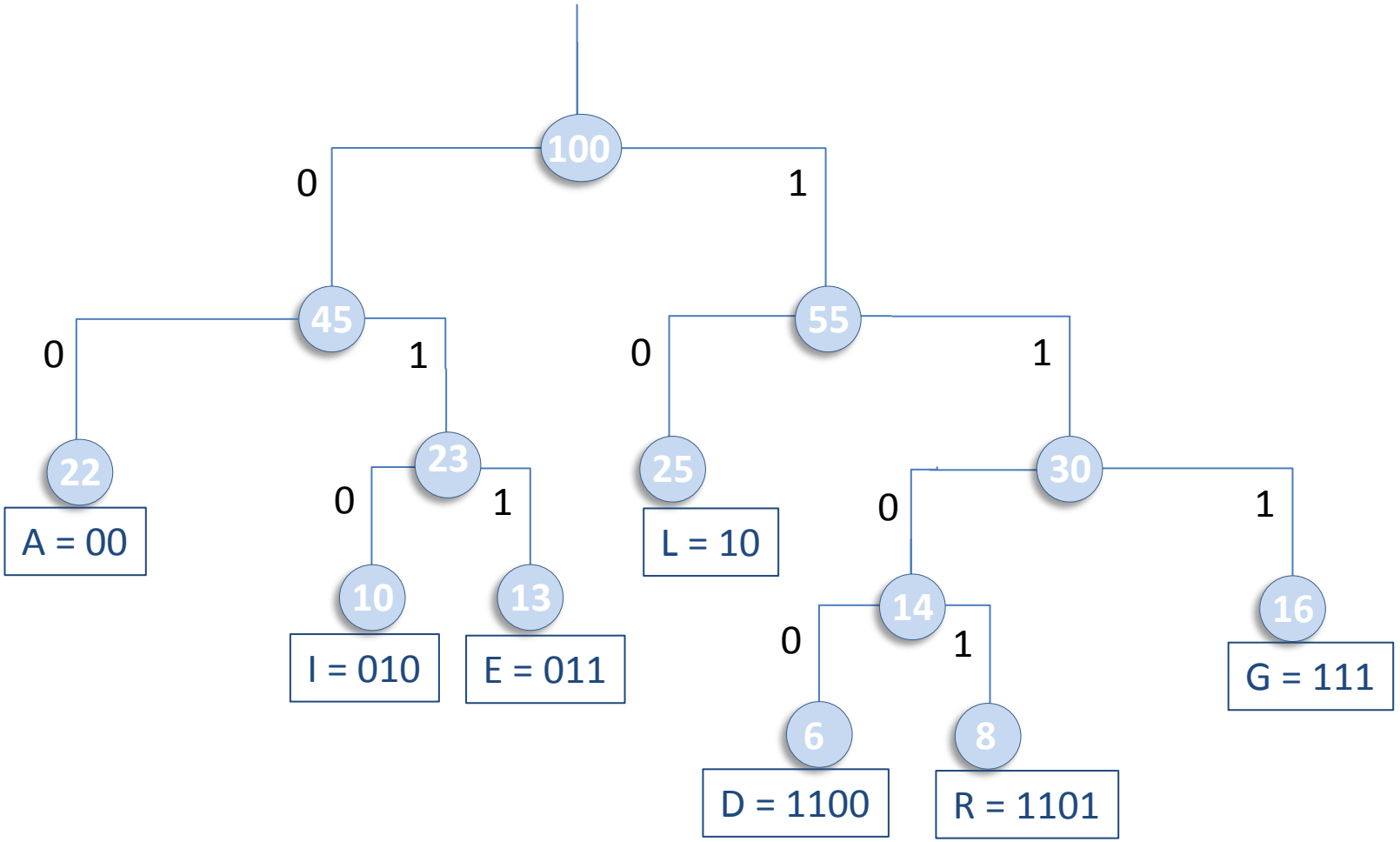
3. Upprepa punkt 2 tills alla tecken kommit med. Roten bör bli 100%.

4. Börja från roten och följ grenarna ut till ett löv. Samla nollor och ettor på vägen - dessa ger koden för lövets tecken.

2 Huffmankodning:Trädet



2 Flytta om noderna: mindre värden till vänster




```
111 00 10 00 1100 1101 010 011
G   A  L  A  D    R    I   E
```

3 Algoritm - binära tal

Numrera vinflaskorna med binära tal:

flaska nr	A	B	C	D	E	F	G	H	I	J	
0	0	0	0	0	0	0	0	0	0	0	ingen dricker
1	0	0	0	0	0	0	0	0	0	1	J dricker ur flaska 1
2	0	0	0	0	0	0	0	0	1	0	I dricker ur flaska 2
3	0	0	0	0	0	0	0	0	1	1	J och I dricker ur flaska 3
…	…										
999	1	1	1	1	1	0	0	1	1	1	A,B,C,D,E,H,I,J ur flaska 999

Om ingen dör vet man att det var flaska 0.
Om bara Jonatan dör vet man att det var flaska 1.
Om både Jonatan och Ingela dör var det flaska 2, osv

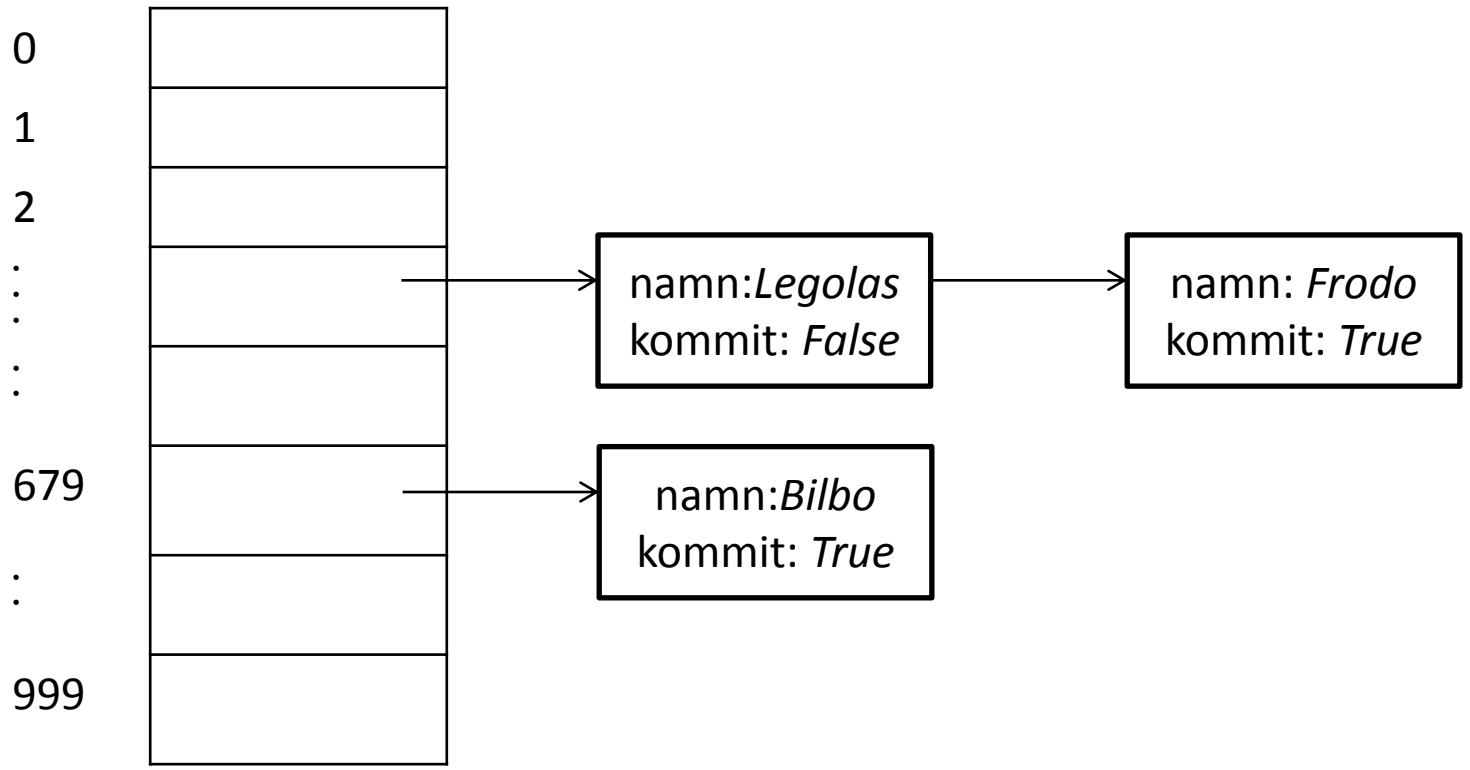
4a Hashning

- **Nyckeln** är gästens namn, t ex "BILBO"
- **Noderna** som lagras i tabellen innehåller gästens namn (behövs för krockhanteringen) samt *kommit* (True/False) för att hålla reda på om gästen kommit.
- **Krockhantering -** krocklistor
- **Hashtabllens storlek n =** 1000 (beror av krockhanteringen)
- **Hashfunktion** f(gästens namn)

Slå ihop ASCII-koderna (ord-funktionen i Python) för varje bokstav till ett tal och ta det modulo n (hashtabellens storlek).

Exempel: f("BILBO") = 6673766679 % 1000 = 679

4 a


0
1
2
679
999
namn:*Legolas*
kommit: *False*
namn: *Frodo*
kommit: *True*
namn:*Bilbo*
kommit: *True*

4 b & c

b) I en min-heap är föräldern mindre än bägge barnen. På alla nivåer!
   b) $1 < 3$ och $1 < 2$
   c) $3 < 5$ och $3 < 4$
   d) $2 < 6$

c) Nod 1 plockas ut, nod 6 fyller igen hålet, jämför med bägge barnen (3 och 2), 2 är minst så nod 2 och nod 6 byter plats. Klart!

5

a) Räknesortering är bra om de **m** kategorierna är mycket färre än de **n** objekten som ska sorteras, alltså **m << n**.

b) För en abstrakt datatyp anger man operationerna men säger inte hur dom ska implementeras. Exempel: En kö har operationerna put (lägg sist i kön), get (ta första från kön) och isEmpty (kolla om kön är tom)

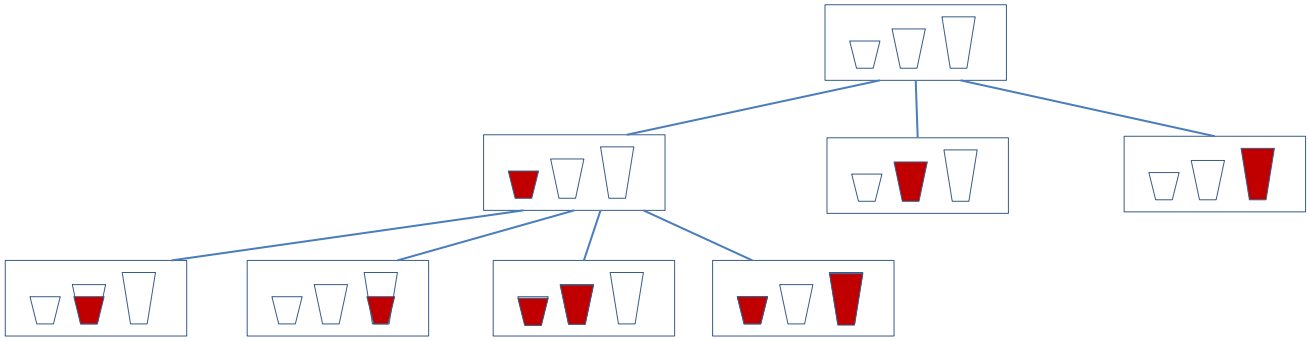
5

c) Det tar O(nlogn) att lägga in i ett balanserat träd (bästa fallet) och O(n) om trädet är som en lista (värsta fallet). Visa med bild!

d) Att en algoritm går i polynomisk tid innebär att den är O(n^k) för något k>0. Tumregel: problem som inte går att lösa i polynomisk tid är för svåra för att lösa med dagen datorer.

e) I ett Bloomfilter lagras bara True/False, så krockar kan inte lösas genom nyckeljämförelse. Flera hashfunktioner minskar sannolikheten för fel pga krock.

6 Problemträdet (första åtta noderna)



I varje nod lagras de tre bägarnas innehåll, samt en referens till föräldernoden.

Breddenförst ger kortaste lösningen.

6 Algoritm: breddenförst

1. Läs in önskad bägarvolym B
2. Skapa en tom startnod och lägg den i kön
3. Plocka ut en nod N ur kön och hitta alla dess barn genom
   - att fylla på nodens tre bägare (en i taget)
   - att hälla från varje bägare till varje annan bägare
   - att tömma nodens tre bägare (en i taget)
4. Skapa nya noder för alla rimliga barn, med referens till N
5. Om någon av noderna har B i en bägare bryter vi och skriver ut lösningen
6. De nya noder som inte är dumbarn läggs in i kön, och även i dumbarnsstrukturen
7. Upprepa från punkt 3.
8. Om kön blir tom finns ingen lösning för B

6 Datastrukturer

- Datastrukturer:
  - Noder med attribut:
    - bägare (t ex lista med heltal)
    - bägarnas maxvolym (klassattribut)
    - förälder (referens till nod)
- Kö
- Hashtabell för dumbarnen (kan t ex innehålla bägarvärden som tupler)

6 Funktioner

- Metoder i Nodklassen: utskrift, rimlighetskontroll
- En funktion som skapar alla barn till en nod
- Funktioner för att fylla, hälla och tömma bägare
- En utskriftsfunktion som skriver ut den följd av bägarnoder som ger önskad bägarvolym

7

a) Caesarchiffer

ARAGORN

BSBHPSO

b) Transpositionschiffer

DERNHELM

DE

RN

HE

LM

DRHLENEM

7

c) Problemet här är att hon måste kunna berätta för de övriga hur chiffreringen ska gå till. Med RSA kan hon publicera sin offentliga nyckel på torgets anslagstavla där alla kan läsa den. Men det är bara hon själv som kan kryptera med hjälp av den privata nyckeln.

8 a) i text

- Basfall1: Om stacken är tom är den sorterad
- Poppa ett element (a)
- Basfall 2: Om stacken bara innehöll ett element är den sorterad
- Rekursiv tanke:
  - Poppa ett element till (b)
  - Och pusha det på stacken igen
  - Om a och b är i rätt ordning och resten av stacken (med b som översta element) är sorterad så är hela stacken sorterad.
- Pusha tillbaka a

8 a) i Python

```
def ordnad(s):
    if s.isEmpty():
        return True
    a = s.pop()
    if s.isEmpty():
        return True
    else:
        b = s.pop()
        OK = a > b
        s.push(b)
        OK = OK and ordnad(s)
    s.push(a)
    return OK
```

8 b)

- Vi jämför varje element med det intill vilket ger n-1 jämförelser om stacken har n element.
- I varje anrop gör vi också (i värsta fallet) två anrop till isEmpty, två anrop till push, och två anrop till pop, alltså 6*n anrop.

Men oavsett vilken operation man räknar blir resultatet O(n)