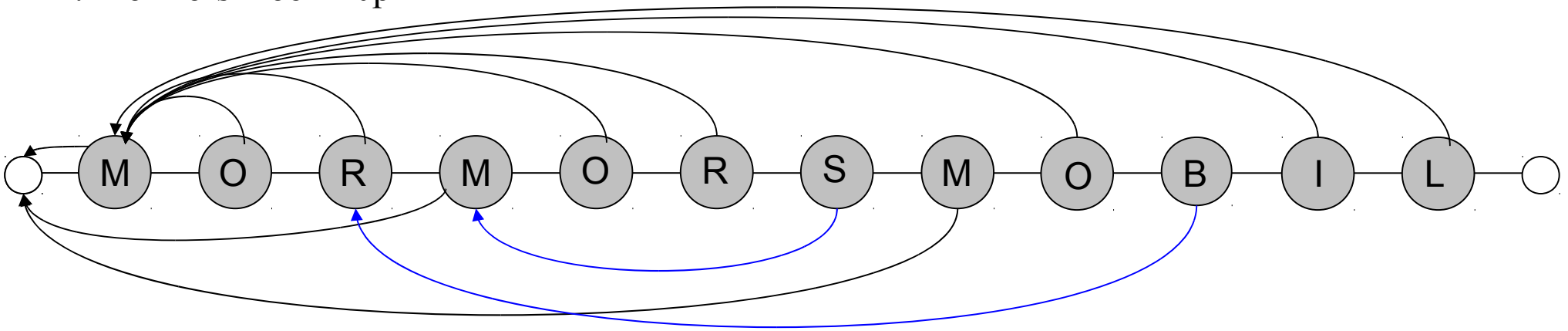
DD1320 Tillämpad datalogi

Lösnings-skiss till tentamen 2010-10-18

1. Mormors mobil 10p

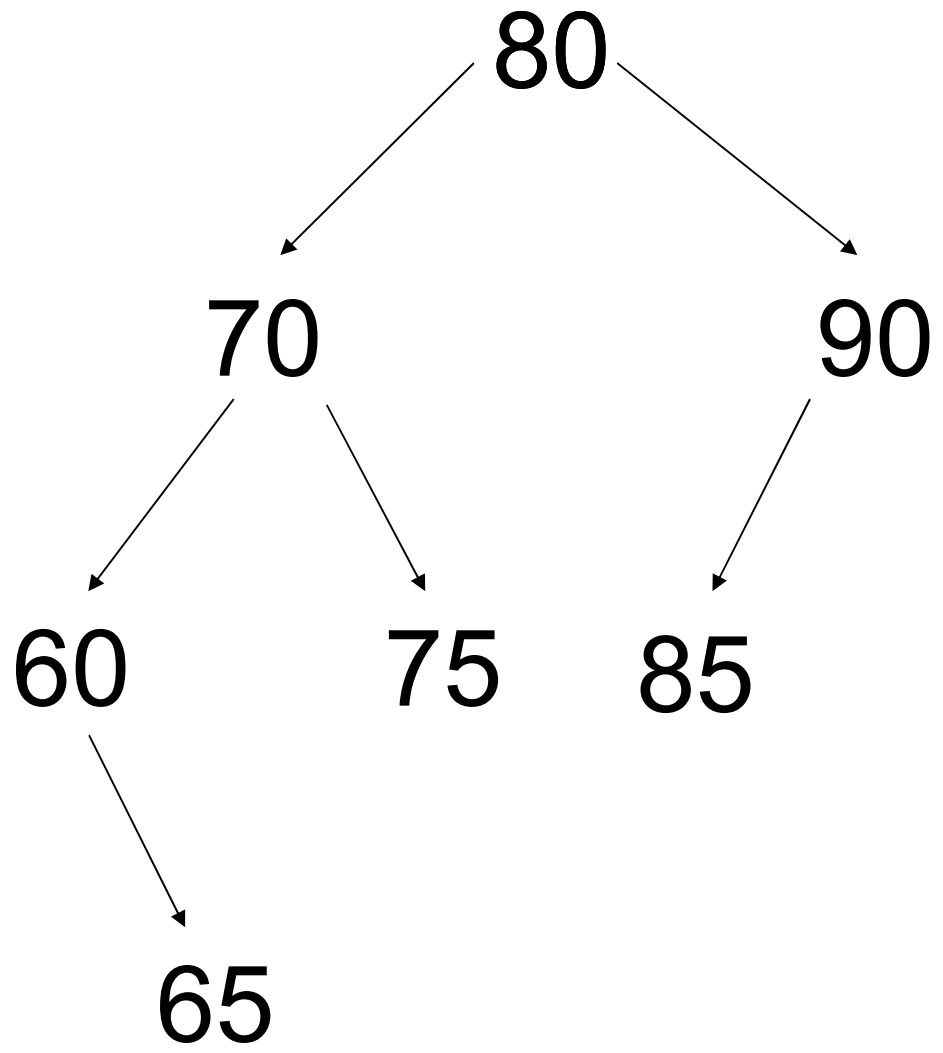


	M	O	R	M	O	R	S	M	O	B	I	L
i	1	2	3	4	5	6	7	8	9	10	11	12
next[i]	**0**	**1**	**1**	**0**	**1**	**1**	**4**	**0**	**1**	**3**	**1**	**1**

Bakåtpilarna/next-värde för i=7 och i=10 är värda mest poäng vid rättningen.

2. Mobiler av mobiler

a) 4p



b) 4p

Preorder: roten, vänster delträd, höger delträd
80 70 60 65 75 90 85

Inorder: vänster delträd, roten, höger delträd
60 65 70 75 80 85 90

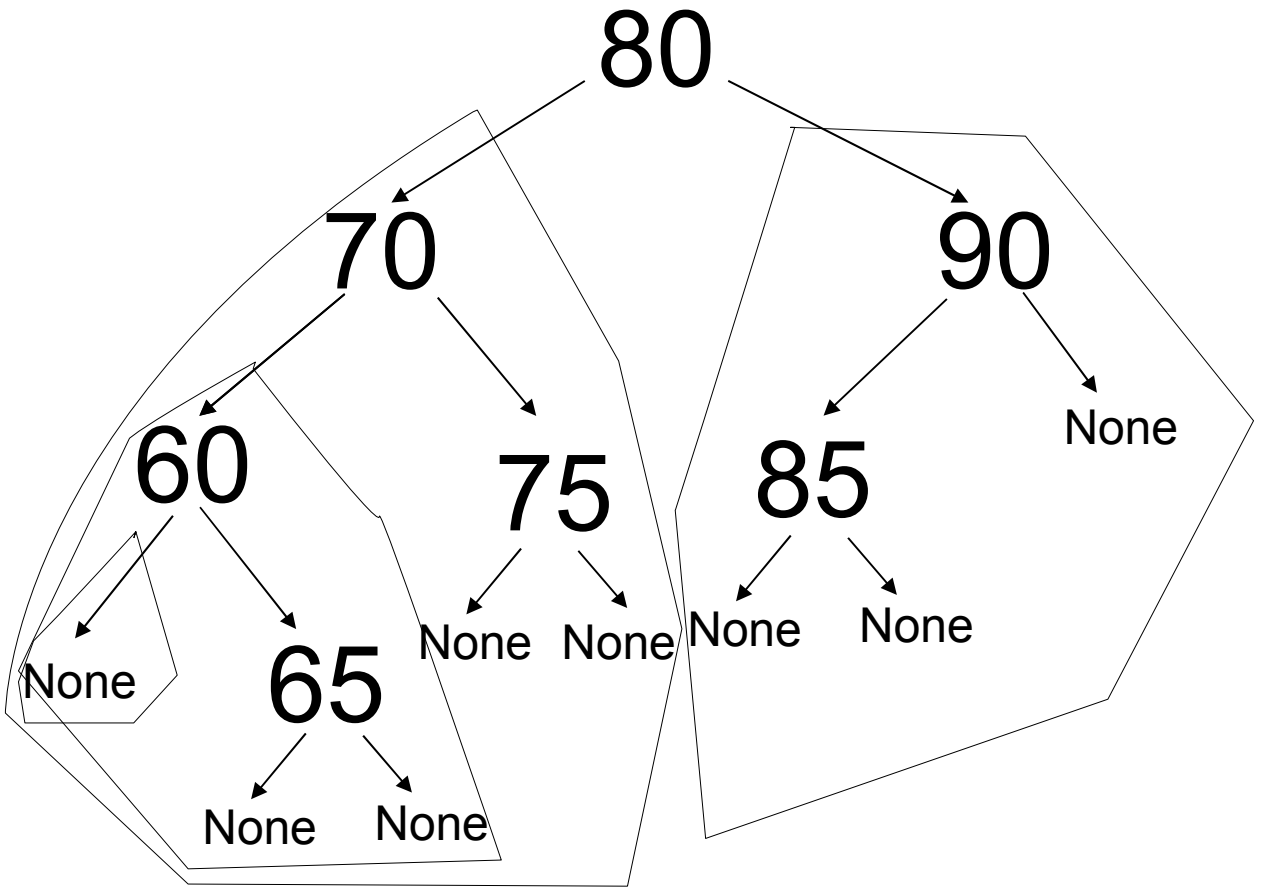
Postorder: vänster delträd, höger delträd, roten
65 60 75 70 85 90 80

Inorder ger 2p (1p om man gjort fel men upptäckt det), preorder och postorder 1p vardera.

2. Mobiler av mobiler
c) 4p

Basfall:
Ett tomt träd (inga noder) har vikten 0

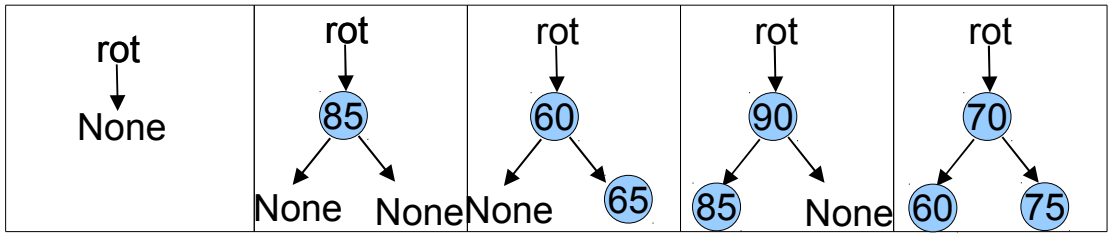
Rekursiv tanke:
Totala vikten =
Totala vikten av vänster delträd + vikten i rotnoden + Totala vikten av höger delträd



En lösning utan rekursiv tanke ger 0 poäng.

2. Mobiler av mobiler
d) 4p

I ett binärt sökträd är:
- alla noder i vänster delträd mindre än roten
- alla noder i höger delträd större än roten
- vänster och höger delträd också sökträd



Basfall:
Ett tomt träd är ett sökträd

Rekursiv tanke:
Ett träd är ett sökträd om
vänsterträdet är tomt eller är ett sökträd vars rotvärde är mindre än trädets rotvärde
och
högerträdet är tomt eller är ett sökträd vars rotvärde är större än trädets rotvärde

Även här ger en lösning utan rekursiv tanke 0 poäng.

3. Mobilkod

a) 4p

* Snabb sökning, det går lika snabbt oavsett hur många nummer man har i registret
* Det är lätt att skapa registret.
* Det är enkelt att lägga till nya nummer.
* Jämför med alternativen:
  Linjärsökning är för långsamt, lika många jämförelser som antal nummer.
  Binärsökning är snabbare, men inte lika snabb (över 20 jämförelser för
  sex miljoner nummer), och innebär extra arbete för att hålla datastrukturen i ordning.

Här gäller det att ha övertygande argument.
De tekniska detaljerna är mindre intressanta – det kommer i nästa deluppgift.

3. Mobilkod

b) 6p

Hashfunktion:
Hashfunktionen är en funktion av nyckeln, som är IMEI-koden
T ex f = IMEI modulo n, där n är hashtabellens storlek

Krockhantering:
Vi använder kvadratisk probning: vid krock på plats *i* tittar vi vidare
på plats $i+1$, $i+2^2$, $i+3^2$, $i+4^2$ osv.

Hashtabellens storlek:
Vid kvadratisk probning måste hashtabellen vara dubbelt så stor (50% luft)
och ett primtal (för att vi inte ska hamna i en oändlig slinga vid probning).
Här har vi sex miljoner abonnenter, så tabellens storlek ska vara ett primtal
större än 12 000 000.

Skapa tabellen:
Innan hashtabellen kan användas måste alla abonnenter hashas in med IMEI-
koden som nyckel.

Sökning:
För att söka i registret anger man en IMEI-kod. Hashfunktionen värde
anger det index där abonnentens data ligger.

Bara teori duger inte - här måste man ge konkreta förslag med hänsyn till just detta problem.

3. Mobilkod

d) 4p

* Ett Bloomfilter innehåller bara True/False men det är OK eftersom vi bara vill veta om telefonen är svartlistad eller inte. Tabellen kan innehålla False på alla platser till att börja med, och True får ange att telefonen är svartlistad
* Varje nummer hashas in i tabellen så här: Vi använder flera olika hashfunktioner och stoppar in True på alla index vi får fram.
* Vid sökning kollar vi samma hashfunktioner igen – om det står True på alla platser är numret svartlistat.
* Ingen krockhantering. Det innebär att ett nummer kan råka se svartlistat ut trots att det inte är det, vilket är en katastrof!
* Om en telefon har blivit svartlistad av misstag går det inte att ta bort den, man kan inte återställa till False eftersom man inte vet vilka andra nummer som påverkas. Detta är också katastrofalt.

Slutsats: Det går inte att använda Blommfilter till svartlistningen.

4. Mobilfråga
a) (5p)

Dekryptera AIEBNDRIDESARLYNOLM
Kan det vara Caesarchiffer?
Prova att skifta vanligaste bokstaven till E

2A, 1B, 2D, 2E, 2I, 2L, 1M, 2N, 1O, 2R, 1S, 1Y
Det finns sju vanliga bokstäver (två av varje) och fem lite mindre vanliga.
Prova A->E (flytta fyra steg framåt)
EMIFRHVMHIWEVPCRSPQ
Det var uppenbarligen inte rätt. Innan vi provar övriga sex testar vi

Transpositionschiffer:
Prova att lägga in radbyten efter 2, 3, … bokstäver och läs kolumnvis:

AI
EB
ND
RI
DE
SA
RL
YN
OL
M

AIE
BND
RID
ESA
RLY
NOL
M

ABRERNMINISLOEDDAYL

AIEB
NDRI
DESA
RLYN
OLM

ANDROIDELLERSYMBIAN

AENRDSRYOMIBDIEALNL

Strukturerade knäckningsförsök kan ge delpoäng även om man inte når målet!

4. b (3 p)
Transpositionschiffer

RSA är bättre för att:

* Det är i praktiken oknäckbart – för att knäcka det måste vi faktorisera ett stort tal (produkten av två stora primtal) och det tar för lång tid (går inte att göra i polynomisk tid).

* Systemet med en publik och en privat nyckel gör det lätt att visa för andra hur dom ska kryptera (man behöver inte vara rädd för att nån tjuvlyssnar).

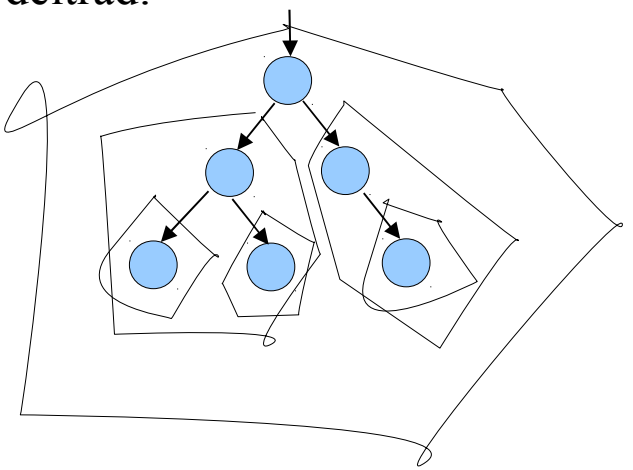
5. Teori (4p per deluppgift)

a) Räknesortering lönar sig när ett stort antal element ska sorteras med avseende på ett mycket mindre antal nyckelvärden. Exempel: Sortera Sveriges befolkning (9 miljoner) på födelseår (110 olika).

b) Ja, man måste lagra även nyckeln i en hashtabell, för att man ska kunna hitta rätt bland krockande element (oavsett krockhanteringsmetod).

c) Frekvenstabeller kan användas:
1. Inom kryptering för att knäcka Caesarchiffrerad text genom att ta reda på vilka bokstäver som är vanligast i den chiffrerade texten och jämföra med frekvenserna i naturligt språk.
2. Inom komprimering genom att ge vanliga tecken kortare koder, t ex i Huffmankodning.

d) Ett binärträd är rekursivt uppbyggt i vänster delträd, rot och höger delträd:



e) Med abstraktion döljer vi implementationen – datatypen används via ett gränssnitt.
Det gör det enklare att använda datatypen: man behöver inte förstå implementationen och ev ändringar av implementationen påverkar inte användningen.

6. Mobilladdning

a) (4p)

Anta att vi har m mobiler och n laddare, där n >= m

Om vi för varje mobil provar igenom laddare får vi i värsta fallet:

mobil nr	antal jämförelser
1	n-1
2	n-2
...	...
m	n-m

dvs n-1 + n-2 + … + n-m = n*m – (1+2+... +m) = m(2n-m-1)/2

Om vi för varje laddare provar igenom mobilerna får vi istället:

laddare nr	antal jämförelser
1	m-1
2	m-2
...	...
n	m-n

dvs m-1 + m-2 + … + m-n = m*n – (1+2+...+n) = n(2m-n-1)/2

Samma formel => bägge varianterna är lika bra.

I a)-uppgiften räcker det egentligen med att visa med några exempel!

6. b (4p)

Med m=8 mobiler och n=8 laddare tar det
m(2n-m-1)/2 = 8(2*8-8-1)/2 = 28 jämförelser
och 28*10 sekunder = 280 sekunder, dvs knappt 5 minuter.

6. c (6p)

Hoppsan, det här gjorde jag visst redan i uppgift a; se ovan!

7. Syntax för SMS (10p)

sms:+46070432215?body=var är du
sms:+46070974293,+46070974295?body=på Q

frågetecken saknas, komma mellan flera nummer saknas,
siffor blandas med +, det ska vara elva siffror i ett nummer,
letter innehåller bara gemena

Förbättrad version:

<sms-uri> ::= <scheme> ":" <sms-hier-part> <sms-field>
<scheme> ::= "sms"
<sms-hier-part> ::= <sms-recipient> | <sms-recipient>, <sms-hier-part>
<sms-recipient> ::= "+" <number>"Ö"
<number> ::= <n><n><n><n><n><n><n><n><n><n><n>
<n> ::= "0"..."9"
<sms-field> ::= "body=" <?body>
<body> ::= <letter> | <letter> <body>
<letter> ::= "a" ... "z" | "å" | "ä" | "ö"|"A" ... "Z" | "Å" | "Ä" | "Ö"

2p per fixat fel, blott 1p per fel som hittats men inte fixats!

8. Mobilappsabstraktion (10p)

Mobiltelefonen har ett ett gränssnitt till operativsystemet, med funktioner som man kan anropa för att t ex kolla gps-koordinater, spela musik etc. För app-utvecklaren fungerar det här gränssnittet som en abstraktion av mobilen, man slipper försöka lista ut hur mobilens hårdvara egentligen fungerar så det blir enklare att programmera. Och när operativsystemet uppdateras ska gränssnittet bevaras (det är bara innehållet i funktionerna som ändrats) så appen går fortfarande att köra. Samma sak gäller om vi vill flytta appen till en annan mobilmodell – så länge den har samma typ av operativsystem går det bra.
Exempel: Min mobil är en Nokia med Symbian. Det finns ett gränssnitt för Python så jag kan t ex skriva en Python-app för att hitta gympapass. Då kan jag anropa funktioner för att läsa från en webbsida (med info om dagens pass), funktioner för att kolla vad klockan är (vilket pass hinner jag till?) och funktioner för att kolla gps-koordinater (vilken gympa-lokal ligger närmast?).
Min app kan sen användas av andra som har Symbian-mobiler, och kommer fortfarande att fungera efter uppdateringar.

Här måste man anknyta till mobiler, allmänt prat om abstraktion duger inte!