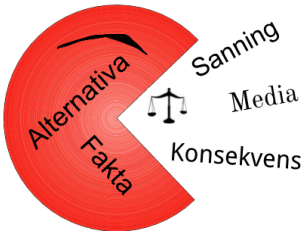
KTH CSC, Alexander Baltatzis

DD1320/1325 TENTAMEN I TILLÄMPAD DATALOGI

Måndag 13 mars 2017 kl 14–18



Hjälpmedel: Fem handskrivna formelblad. För betyg E krävs att alla E-uppgifter är godkända (upp till två E-uppgifter kan kompletteras). För betyg D krävs (utöver E-kraven) D på bägge C-uppgifterna, för C krävs minst ett C och ett D (en C-uppgift kan kompletteras till D). För betyg B respektive A krävs (utöver C-kraven) betyg B respektive A på A-uppgiften (A-uppgiften kan inte kompletteras). Lycka till!

1. *KMP*

Betyg E. En regeringschef anklagar sin företrädare för att ha tjuvspanat på denne, men fakta i ärendet saknas. En tildastudent erbjuder sig att söka efter eventuell kamerautrustning.

15 min Rita en KMP-automat för CAMCORDERCAMERA samt ange next-vektorn.

	C	A	M	C	O	R	D	E	R	C	A	M	E	R	A
Nextvektorn	0	1	1	0	2	1	1	1	1	0	1	1	4	1	1

2. *prioritetskö*

Betyg E. Fakta och alternativa fakta konkurrerar om mediautrymmet. Det är väldigt viktigt att prioritera rätt. En prioritetskö kan komma till användning.

En prioritetskö (maxheap) är internt representerad med en vektor som innehåller följande värden:

80, 50, 10, 40, 35, 5, 3, 1

15 min Hur ser heapvektorn ut när man sätter in först 20, därefter 4. Motivera kort.

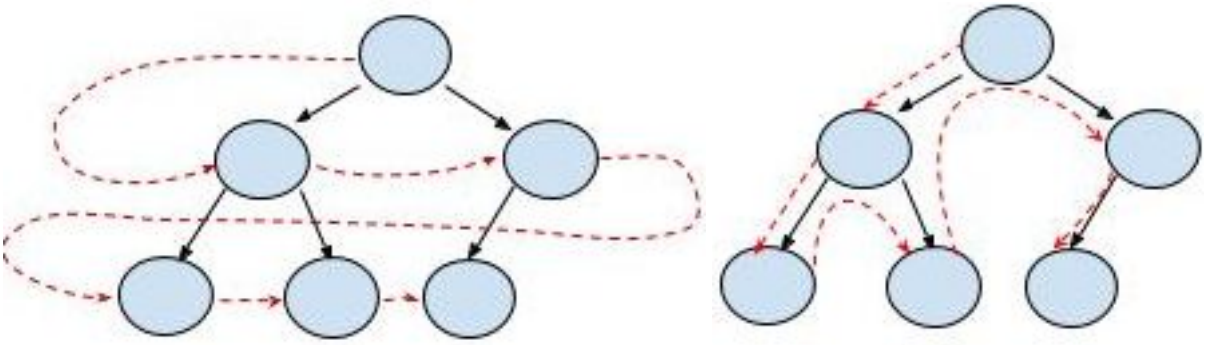
a) 80, 50, 10, 40, 35, 20, 3, 1, 5, 4
b) 80, 10, 50, 40, 35, 20, 3, 1, 5, 4
c) 35, 10, 50, 5, 20, 40, 80, 3, 1, 4
d) 35, 5, 50, 3, 10, 40, 80, 1, 4, 20
e) 80, 50, 10, 40, 35, 5, 3, 1, 4, 20
f) 80, 50, 10, 40, 35, 5, 3, 1, 20, 4

svar f) 80, 50, 10, 40, 35, 5, 3, 1, 20, 4 Talen läggs in på sista platsen i heapvektorn och ingen upptrappning sker eftersom de är mindre prioriterade än sin förälder.

3. *bredden-först vs djupet-först.*

Betyg E. För att skilja på fakta och alternativa fakta och reda ut vad som gäller är det bra med en bred allmänbildning och att kunna gå på djupet vid behov.

15 min Beskriv kortfattat vad skillnaden är mellan en breddenförstsökning och en djupetförstsökning. Illustrera med två figurer.



Man ska nämna att bredden-först ger kortaste vägen. Det är fel att påstå att bredden-först är snabbast. Det kan räcka med en figur om man ritat båda sätten i samma figur på bra sätt.

4. *bloomfilter*

Betyg E. För att komma ihåg vad som sagts och inte sagts så tänker man använda ett bloomfilter. Ett bloomfilter har gränssnittet `set` och `exists`. En tildastudent påpekar att `exists` inte fungerar 100-procentigt. Vad är det som menas?

a) `exists` kan returnera false fast man lagt in värdet man söker efter

b) `exists` kan returnera true även om man inte lagt in värdet man söker efter.

c) Både a och b menas

10 min Förklara och motivera! Rita gärna.

Svar b) Att man kan få "false positives" är rätt. Det krävs motivering och förklaring men ingen bild. a) kan inte vara rätt, när man lägger in ett ord så skriver man flera (14) ettor. Det finns inget som tar bort dem.

5. *publika och privata nycklar*

Betyg E. En stadig ström av ibland motsägelsefulla nyheter tickar fram i nyhetsterminalerna. I tider av alternativa fakta är det svårt att skilja på olika versioner av samma nyhet. Tildastudenter har dock örnkoll på versionshantering via terminaler.

För att använda github från terminalen så var du tvungen att skapa ett nyckelpar `id_rsa` och `id_rsa.pub`. Därefter sparade du `id_rsa.pub` på github.

15 min Vilket problem löses med detta förfarande? Förklara, gärna med illustrationer.

Det som löses är autentisering. När man ansluter till github så vet github att det är du och ingen annan som försöker ansluta sig. Om man ansluter med ssh via terminalen och inte gjort det tidigare så får man en fråga om man litar på adressen och gör man det så sparas ssh-nycklart. Krypteringen kommer att skötas i ssh-förbindelsen.

Det är viktigt att knyta an till uppgiften, t.ex. de namngivna filerna. Det är godkänt att knyta an till nyhetstemat.

15 min **6.** Vad är det för tidskomplexitet för följande operationer? Svara med ordo-notation. Motivera kort.

a) O(1) Lägga in ett nytt element först i en enkellänkad lista.

b) O(N) Lägga in ett nytt element först i en vektor (array/pythonslista)

c) O(log N) Lägga in ett nytt element i ett balanserat binärt sökträd.

d) O(log N) Lägga in ett nytt element i en prioritetskö (heap)

7. *stabil sortering*

Betyg C. En stabil sortering bibehåller inbördes ordning, exempel om man först sorterar på namn sedan på ålder.

```
Anna    21 år                Eric    20 år
Bertie  22 år                Fiona   20 år
Celia   22 år                Anna    21 år
David   21 år                David   21 år
Eric    20 år                Gina    21 år
Fiona   20 år                Bertie  22 år
Gina    21 år                Celia   22 år
```

Vilka av följande sorteringsmetoder är stabila?

1. Urvalssortering
2. Bubbelsortering
3. Quicksort

30 min Motivera, förklara implementationsdetaljer och visa med belysande exempel.

`Tips:` för att slippa skriva så mycket räcker det med namnens begynnelsebokstav istället för att skriva hela namnet.

`Tips:` det är tillåtet att ändra sorteringsmängden för att göra kunna göra ett bra belysande exempel.

Urvalssortering:

```
Anna    21 år * Eric    20 år    Eric    20 år
Bertie  22 år   Bertie  22 år  * Fiona   20 år
David   21 år   David   21 år    David   21 år
Eric    20 år * Anna    21 år    Anna    21 år
Fiona   20 år   Fiona   20 år  * Bertie  22 år
Gina    21 år   Gina    21 år    Gina    21 år
```

Man behöver bara påvisa ett byte där den inbördes ordningen går förlorad. I listan ovan byter Anna 21 plats med Eric 20 när man sorterar på ålder. Då hamnar Anna 21 efter David 21 och den inbördes ordningen förloras. tt vanligt fel var att bland ihop urvalssortering och insättningssortering. I insättningssortering så sätts Eric 20 in före Anna 21 och alla element flyttas ett steg.

Quicksort:

```
Anna    21 år     Anna    21 år * Gina    21 år
Bertie  22 år     Bertie  22 år   Bertie  22 år
David   21 år     David   21 år   David   21 år
Eric    20 år   * Gina    21 år * Anna    21 år
Fiona   20 år     Fiona   20 år   Fiona   20 år
Gina    21 år   * Eric    20 år   Eric    20 år
```

Antag att Anna 21 är pivot. Alla element strikt mindre än 21 ska placeras längst ner. Redan i första steget när Gina 21 och Eric 20 byter plats så

För bubbelsortering kan man visa att bytena sker mellan två bredvidliggande element. Bytena "hoppar" inte som i quicksort urvalssortering. Visa med exempel och jämför med de andra sätten.

Man måste göra en ansats till jämförelse. T.ex. quicksort är inte stabil därför att ... Fx om man gjort åtminstone ett belysande exempel och en slutsats. quicksort är inte stabil därför att ... D Om man gjort en helt korrekt slutsats med belysande exempel och en ansats på en till. C Om man gjort två helt korrekta slutsatser med belysande exempel och gjort ansats på en till

8. *alternativa syntaxer*

Betyg C. Studera följande två alternativa syntaxer och svara på frågorna a-e nedan

Syntax A

```
<Mening>      ::= <Vetenskaper> <AltFakta>.
<Vetenskap>   ::= <Ämne> är en vetenskap | <FleraÄmnen> är vetenskaper
<FleraÄmnen>  ::= <Ämne> och <Ämne> | <Ämne>, <FleraÄmnen>
<Ämne>        ::= fysik | meteorologi | historia

<AltFakta>    ::= men <Pseudo> är en inte vetenskap | men <FleraPseudon> är inte
                  vetenskaper
<FleraPseudon>::= <Pseudo> och <Pseudo> | <Pseudo>, <Pseudo>  och <Pseudo>
<Pseudo>      ::= intelligent design | kreationism | scientologi
```

Syntax B

```
<Mening>      ::= <Vetenskaper> <AltFakta>
<Vetenskap>   ::= <Ämne> är en vetenskap | <FleraÄmnen> är vetenskaper
<FleraÄmnen>  ::= <Ämne> och <Ämne> | <Ämne>, <Ämne> och <Ämne>
<Ämne>        ::= fysik | meteorologi | historia

<AltFakta>    ::= men <Pseudo> är en inte vetenskap | men <FleraPseudon> är inte
                  vetenskaper
<FleraPseudon>::= <Pseudo> och <Pseudo> | <Pseudo>, <FleraPseudon>
<Pseudo>      ::= intelligent design | kreationism | scientologi
```

Följande krav ställs på syntaxerna.

Syntaxen ska godkänna följande tre meningar:

1 *fysik är en vetenskap men kreationism, intelligent design och scientologi är inte vetenskaper*

2 *histora och fysik är vetenskaper men scientologi och kreationism är inte vetenskaper*

3 *fysik, meteorologi och historia är vetenskaper men intelligent design är inte en vetenskap*

Syntaxen ska underkänna grammatiskt felaktiga meningar med avseende på pluralform som t.ex. följande två meningar:

4 *fysik är en vetenskap men kreationism, intelligent design och scientologi är inte en vetenskap*

5 *fysik är vetenskaper men kreationism är inte en vetenskap*

Syntaxen ska använda `och` samt `,` när ämnen staplas och inte godkänna meningen:

6 *fysik, historia är vetenskaper men intelligent design och kreationism och scientologi är inte vetenskaper*

25 min

a) Vilka meningar (1-3) underkänner syntax A som inte borde underkännas?

b) Vilka meningar (4-6) godkänner syntax A som inte borde godkännas?

c) Vilka meningar (1-3) underkänner syntax B som inte borde underkännas?

d) Vilka meningar (4-6) godkänner syntax B som inte borde godkännas?

e) Jämför de två syntaxerna. Antag att det tillkommer fler ämnen som man vill uttala sig om d.v.s. man vill godkänna meningar med betydligt fler ämnen uppräknade. Resonera och reflektera och visa med exempel.

Det finns tryckfel i uppgiften. Några rättades under tentamen på tavlan. Om man bortser från tryckfelen kan man konstatera att båda syntaxerna godkänner de meningar de ska och underkänner de meningar de ska underkänna. Rätt svar blir då inga på a-d. Det är också rätt att ta hänsyn till tryckfelen, t.ex. det oavsiktliga bindestrecket i ett av exemplen.

Skillnaden i syntaxerna är att de har rekursivitet på olika ställen, den ena för <FleraÄmnen> och den andra för <FleraPseudon>. Det bästa vore att kombinera dessa om man vill räckna upp både vetenskapliga och ovetenskapliga ämnen. För C måste man uttala sig om båda i e). Man måste beskriva rekursivitet (ordet behövs inte) och får inte ha allvarligt fel a-d. För D får inte ha allvarligt fel a-d och uttalat sig om båda men inte beskrivit rekursivitet eller uttalat sig om endast en samt beskrivit rekursivitet. För Fx får inte ha allvarligt fel a-d samt man måste ha gjort en ansats på e)

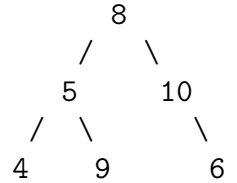
9. *binära sökträd*

Betyg A. Ett binärt sökträd byggs upp av noder, till exempel

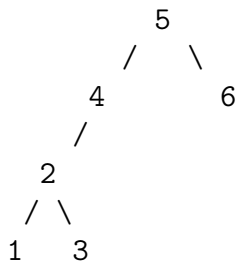
```
class Node:
    def __init__(self, d=None):
            self.data  = d
            self.right = None
            self.left  = None
```

Men det räcker inte att ha noder. Vi vill ha ett binärt sökträd som är korrekt och balanserat.

Nedan är ett exempel på ett felaktigt binärt sökträd. Siffrorna 9 och 6 är fel. Däremot är trädet balanserat.



Nedan är ett exempel på ett korrekt binärt sökträd som inte är balanserat



Man vill ha en effektiv funktion som givet en pekare till en root-nod i ett binärt sökträd returnerar `True` om det binära sökrädet är ett korrekt binärt sökträd samt att trädet är balanserat. Båda villkoren ska vara uppfyllda i annat fall ska `False` returneras.

40 min

a) Konstruera en effektiv algoritm för att lösa problemet. Beskriv utförligt eventuella extra datastrukturer och hjälpfunktioner som du använder.

b) Ange komplexitet för din algoritm.

c) Reflektera över din lösnings effektivitet.

För betyg A krävs att din algoritmbeskrivning är tydlig och välstrukturerad.

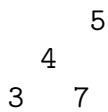
O(N) är krav B löst en uppgift rätt A två uppgifter rätt samt avbryter när det är fel eller reflekterar att det borde avbryta innan O(N) när det är fel

b) För att övertyga sig om att trädet är korrekt och balanserat måste man gå igenom alla noder d.v.s. O(N). För betyg A behöver man inse det. O(log N) är fel. O(N^2) är för oeffektivt.

a) Det finns flera olika lösningar. Man kan dela upp problemet ett i två delar. För att lösa om det är ett korrekt binärt träd kan man t.ex. tänka sig en rekursiv lösning där man skickar med ett intervall som noden måste hålla sig inom. Den rekursiva tanken är att jag kollar om min egen nod är inom intervallet.

I ett specifikt anrop så kollar jag om min nods värde är inom intervallet. Om inte kastar jag ett undantag. I annat fall så anropar jag rekursivt till vänster med minvärdet jag fått och min egen nods värde som maxvärde. På samma sätt anropar jag rekursivt till höger med min nod som minsta värde och skickar med maxvärdet jag fått. Är noden None gör jag ingenting.

Det räcker inte att kolla om min nod är korrekt i förhållande till mina barns värden vilket man kan se i det givna exemplet på tentatalet.



Ett annat alternativ till lösning är att gå igenom trädet inorder och ser om de påträffade värdena är sorterade. Det räcker med att spara och jämföra med det senast besökta värdet, man behöver inte spara alla värden i en lista som man sedan går igenom. Koden blir lite enklare och skulle kunna se ut ungefär så här

```
class Remember:
     def check(value):
         if value < self.previous_value:
             raise "WRONG TREE"
         self.previous_value = value
```

```
def isCorrect(p, remember)
    if p != None
       isCorrect(p.left)
       remember.check(p.value)
       isCorrect(p.right)
```

Att kolla om trädet är balanserat kan man också göra rekursivt men det är svårt att få det rätt. En naiv tanke är att att jämföra maxhöjden till vänster och till höger.

```
def checkBalance1(p):
   r = height(p.right, 1)
   l = height(p.left, 1)
   if abs ( l - r ) > 1 :
       raise "NOT BALANCED"

def height(p, h):
   if p == None:
       return h
   else
      return max(height(p.right, h+1), height(p.left, h+1))
```

Det blir dock fel svar. Ovanstående kommer att godkänna ett träd som ser ut som ett stort lambda.

```
      5
    3   6
  2       8
1           9
```

Man måste i varje delträd avgöra om det är balanserat. En kollosalt oeffektiv lösning är

```
def isBalanced2(p)
    if p != None:
    return isBalanced2(p.left) and
            isBalanced2(p.right) and
            abs(height(p.left) - height(p.right)) <= 1)
```

Där man traverserar trädet flera gånger.

Ett lite bättre alternativ som fortfarande har problem

```
def almostGoodEnough(p, height):
   if p == None:
       return height
   else:
        height += 1
        l_height = almostGoodEnough(p.left, height)
        r_height = almostGoodEnough(p.right, height)
        if abs ( l_height - r_height ) > 1 :
           raise "NOT BALANCED"
```

```
        return max( l_height, r_height )
```

```
          6
     4       9
   1   3    8
    2
```

Delträdet i 4 är balanserat och delträdet i 9 är också balanserat. Jämför man maxdjupen skulle de vara balanserade även i 6. Man kan fixa det med att returnera både min och max och ändra if-satsen typ:

```
if abs ( max (l_height.max, r_height.max) - min (l_height.min, r_height.min)) >
```

Ett lättare alternativ är att spara undan minsta och största djupet på trädet alltmedan man traverserar. Om man vid något tillfälle påträffar en nod som är mycket större än minsta eller mycket mindre än största så avbryter man sökandet. Går man igenom med bredden först så vet man att man kommer att stöta på det minsta djupet först och kan jämföra med detta minsta värde.

Det är fel att bara titta på löven, se exempel på tentan, eftersom man missar noder med ett barn. Det är lättare att titta på None. Så fort man träffar på ett None så gör man en koll mot störst_hittills och minst_hittills och eventuellt uppdaterar man dem.

```
def isCorrect(p, h, remember)
    h += 1
    if p != None
       isCorrect(p.left)
       remember.check(p.value)
       isCorrect(p.right)
    else:
       remember.checkheight(h) # kontrollerar höjden mot minsta
                               # och största djup och/eller uppdaterar
```

I kodexemplen ovan har jag konsekvent använt raise exception. Jag tänker mig att gränssnittet mot användaren är en funktion som fångar exception och i så fall returnerar false i annat fall true till användaren vilket implicit betyder att de anropade funktionerna inte behöver returnera någonting. Gränssnittsfunktionen kan också ta hand om speciallfall som tomma trädet, eller träd med endast en nod. def

```
def checkTree(root):
    if root == None: # Tomt träd ...
    if root.left == None and root.right == None: # En nod..
    try:
        remember = Remember(# initialize ..
        isCorrectandBalanced(root, remember)
        return True
    except:
        return False:
```

Man behöver inte använda undantag för att få godkänt på sin lösning. I djupa rekursioner där man vill avbryta kan det vara motiverat att använda goto i språk

som tillåter goto. Om man inte använder exceptions eller goto måste man ta hand om returvärdet i varje rekursion och själv returnera.

c) I teoretisk datalogi är det här problemet O(N) alldeles oavsett. Indata kan vara så elakt att man alltid måste köra till sista jämförelsen på ett felaktigt träd. Det här är en kurs i tillämpad datalogi och i praktiken är det viktigt att avbryta sökningen så fort man kan. Man vet ofta något om indatat, t.ex. att det är slumpmässigt fördelat och då lönar det sig att avbryta så fort man vet att trädet är fel. Helst med endast en trädgenomgång.