Övning 2 - Tillämpad datalogi 2013

```
0  # coding: latin
```

Summering

Vi gick igenom **listor**, **pekare**, **binära träd** , **rekursion** och **komplexitet**. Detta exemplifierades genom att vi skrev **remove_second** och **gcd** (greatest common divisor) och . Vi gick även igenom hur binära träd byggs, utag från binärträd med in, pre och postordning samt vi räknade på programtidsåtgång givet en viss programkomplexitet. Övningsanteckningarna innehåller även ett ytterligare ett par program, **remove_every_second**, **fib** (fibonnaccis tal), **summa** (trädsumma) och **height** (trädhöjd) som det kan vara bra att kolla på.

- Some imports, ignore

```
23 import sys, os
24 sys.path.append(os.getcwd())
```

Listor noder

1. Andra noden bort

a) ***Skriv en sats som tar bort andra noden ur en länkad lista.***

```
36
37 from MyUtils import LinkedList
38
39 def remove_second(ll):
40     n=ll.first
41     try:
42         n.next=n.next.next
43         return ll
44     except:
45         raise IndexError('List does not have two elements')
46
47 ll = LinkedList()
48 ll.append('Goose IPA')
49 ll.append('Fat Tire')
50 print ll
```

```
Goose IPA
Fat Tire
```

```
50 remove_second(ll)
51 print ll
```

```
Goose IPA
```

```
51 #remove_second(ll)
```

b) ***Nu ska du ta bort andra noden ur en stack. Lös uppgiften abstrakt, alltså med anrop till push och pop!***

```
59 from MyUtils import Stack
60
61 s = Stack()
62 s.push('San Fransico')
63 s.push('Chicago')
64 s.push('Las Vegas')
65 print(s)
```

```
Las Vegas
Chicago
San Fransico
```

```
66 e = s.pop()
67 s.pop()
68 s.push(e)
69 print(s)
```

```
Las Vegas
San Fransico
```

2. Ta bort varannan nod

OBS Tänk igenom och rita bilder för att visa att alla dessa fungerar för alla listor:

- tom lista
- lista med ett element
- lista med ojämnt antal element
- lista med jämnt antal element

a) ***genom att meka med pekare***

- Vi ska ha kvar det aktuella elementet n
- Ifall den har en efterföljare, så ska n.next sättas n.next.next för att hoppa över efterföljaren.

```
96
97 def remove_every_second(ll):
98     n = ll.first
99     while n != None and n.next != None:
100         n.next = n.next.next
101         n = n.next
102 '''
103 #Test kod:
104 ll = LinkedList()
105 for i in range(10):
106     ll.append(i)
107 print ll
108 remove_every_second(ll)
109 print ll
110 '''
```

```
    ....................................
    b) *abstrakt, med hjälp av en stack*
    ....................................

     Ifall vi har två element, lägg undan ett och släng det andra.
     Varför fungerar det när vi har ett element?
     Hamnar allt i rätt ordning
    Ide: Ta bort från listan med pop. Lägg varannan nod på en stack. Skapa en
    ny lista från stacken.
```

```
127
128 def remove_every_second(ll):
129     stack = Stack()
130     while len(ll) >= 2: # Så länge vår ursprungliga lista är längre än två
131
132         a = ll.pop() # Tar bort första listelementet
133
```

```
134         stack.push(a) #Lägger det på en stack
135
136         ll.pop() # Slänger bort nästa
137
138
139     # Vänd på stacken, som i lab 1.
140     new_stack=Stack()
141     while not stack.isempty():
142         new_stack.push(stack.pop())
143
144     # Läg in värdena i en lista
145     while not new_stack.isempty():
146         el = new_stack.pop()
147         ll.append(el)
148
149     return ll
150 '''
151 #Test kod:
152 ll = LinkedList()
153 for i in range(10):
154     ll.append(i)
155 print ll
156 remove_every_second(ll)
157 print ll
158 '''
```

c) ***abstrakt, med hjälp av rekursion***

- Varje anrop tar in listan, och returnerar listan till “ovanstående” rekur- sionsnivå
- En stack med mindre än två element är klar.
- Annars, ta bort två element, gör rekursivt anrop, stoppa tillbaka.

```
162
163 def remove_every_second(ll):
164     if len(ll) < 2:
165         return ll
166     else:
167         a = ll.pop()
168         b = ll.pop()
169         ll = remove_every_second(ll)
170         ll.insert_as_first(a)
171     return ll
172 '''
173 #Test kod:
174 ll = LinkedList()
175 for i in range(10):
176     ll.append(i)
177 print ll
178 remove_every_second(ll)
179 print ll
180 '''
```

Laws of recursion:

- A recursive algorithm must have a base case.
- A recursive algorithm must change its state and move toward the base case.
- A recursive algorithm must call itself, recursively.

[http://interactivepython.org/courselib/static/pythonds/Recursion/recursionsimple.html]

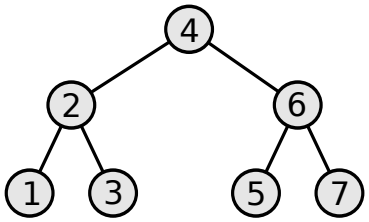
Binära sökträd

1. Trädbygge

Hur ser det binärträd ut som skapas om man puttar in talen 4 2 1 6 3 7 5 i denna ordning? Och hur ser det ut om man sätter in dom i omvänd ordning, alltså 5 7 3 6 1 4 2? Är båda träden binära sökträd? Är de balanserade?

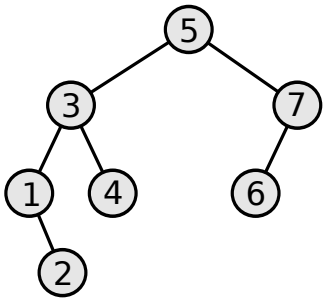
- Ett binärt sökträd har varje nod två löv samt en regel för att kunna avgöra storlek på objekt
- I ett *balanserat* träd är höjdskillnaden i alla grenar <= 1.
- **Inordning:** (vänster träd) (noden) (höger träd)
- **Preordning:** (noden) (vänster träd) (höger träd)
- **Postordning:** (vänster träd) (höger träd) (noden)

Insättning av 4 2 1 6 3 7 5



Binärt sökträd: Ja, Balanserat: Ja

Insättning av 5 7 3 6 1 2 4



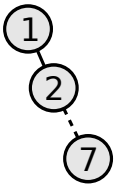
Binärt sökträd: Ja, Balanserat: Nej

2. In, pre och postorderträd

Skriv ut något av träden i inordning och bygg upp ett nytt träd från denna talföljd. Skriv sedan ut dom i preordning och bygg upp nya träd från dessa talföljder. Skriv slutligen ut dom i postorder och bygg upp nya träd från dessa talföljder.

Inordning (v,n,h)

Båda träden blir 1...7 med inordning.
Trädet blir helt obalancerat med en lång sekvens 1-7

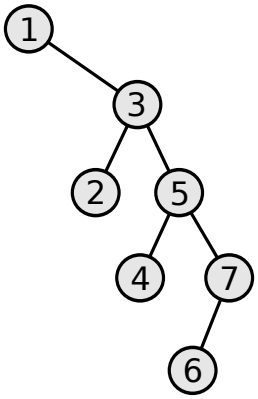


Preorder (n,v,h)

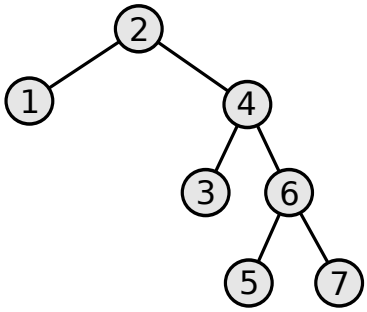
Första: 4, 2, 1, 3, 6, 5, 7 => urspungsträdet
Andra: 5, 3, 1, 2, 4, 7, 6 => ursprungsträdet
Preorder ändrar alltså inte trädstrukturen

Postordning (v,h,n)

Första 1, 3, 2, 5, 7, 6, 4



Andra 2, 1, 4, 3, 6, 7, 5



Rekursion

1. Euklides algoritm för att beräkna greates common divisor (GCD)

För att beräkna största faktorn som två tal m och n har gemensamt kan vi använda Euklides algoritm:

- Om m är jämnt delbar med n så är n den största gemensamma faktorn. Annars är GCD(m, n) = GCD(n, m % n).

Skriv en rekursiv funktion!

```
222 #1 Vad är **m % n**? Med % beräknar man resten vid division mellan m med n.
    Detta görs genom att
```

räkna ut hur k, dvs hur gånger som n går i m. T.ex. m=11, n=3. n (3) går 3 gånger i m (11) så k=3. Föratt få ut resten tar vi r=m-n*k=11-3*3=2.

```
236
237 def gcd(m,n):
238     s="GCD(" + str(m) + ", " + str(n) + ") -> "
239
240     if m % n == 0:
241         print(s+str(n))
242         return n
243     else:
244         print( s+"GCD(" + str(n) + ", " + str(m % n) + ")")
245         gcd(n, m % n)
246
247 gcd(867, 1989)
```

```
GCD(867, 1989) -> GCD(1989, 867)
GCD(1989, 867) -> GCD(867, 255)
GCD(867, 255) -> GCD(255, 102)
GCD(255, 102) -> GCD(102, 51)
GCD(102, 51) -> 51
```

2. Fibonaccital

Leonardo Fibonacci skrev år 1225 en bok där han beskrev denna intressanta talföljd: Sista december föds en kaninpojke och en kaninflicka. Vid två månaders ålder och varje månad därefter producerar varje kaninpar ett nytt kaninpar. Vi kan skriva en rekursiv formel för antal kaninpar:

- $f_0 = 0$
- $f_1 = 1$
- $f_n = f_{n-1} + f_{n-2}$

Skriv en rekursiv funktion för att beräkna Fibonaccital. Visa vilka rekursiva anrop den ger upphov till vid beräkningen av fib(5). Är det här det effektivaste sättet att beräkna Fibonaccitalen?

```
267
268
269 def fib(n):
270     s=n*' '+"fib(" + str(n)+ ") -> "
271     if n <= 1:
272         print(s+str(n))
273         return n
274     else:
275         print(s+"fib(" + str(n-1)+ ") + "+ "fib(" + str(n-2)+ "))")
276         return fib(n-1) + fib(n-2)
277
278 print fib(5)
```

```
fib(5) -> fib(4) + fib(3))
 fib(4) -> fib(3) + fib(2))
  fib(3) -> fib(2) + fib(1))
```

```
    fib(2) -> fib(1) + fib(0))
  fib(1) -> 1
 fib(0) -> 0
  fib(1) -> 1
   fib(2) -> fib(1) + fib(0))
  fib(1) -> 1
 fib(0) -> 0
    fib(3) -> fib(2) + fib(1))
   fib(2) -> fib(1) + fib(0))
  fib(1) -> 1
 fib(0) -> 0
  fib(1) -> 1
5
```

Man kan visa att antalet anrop faktiskt växer snabbare än själva Fibonaccitalen! (För n = 40 är Fibonaccitalet ca hundra miljoner men antalet rekursiva anrop är över 300 miljoner.) Bättre vore här att använda en for-slinga:

```
282
283 def fib(n):
284     f0 = 0
285     f1 = 1
286     for i in range(n-1):
287         f2 = f1 + f0
288         f0 = f1
289         f1 = f2
290     return f1
```

3. Rekursiv trädsumma

Ge en rekursiv tanke för summan av alla talen i trädet och programmera den så att anropet tree.sum() ger rätt svar.

- Anta att trädet är uppbyggt av noder som har tre attribut två som vi kallar pekare, **left** och **right**, och en med nodens värde, **value**.
- **Rekursiv tanke:** Summan är lika med summan av vänster träd, plus höger träd plus nodens värde... men vid tomt träd är summan noll

Som program:

```
304 def summa(p):
305     if p == None:
306         return 0
307     else:
308         return int(p.value) + summa(p.left) + summa(p.right)
```

4. Rekursiv trädhöjd

Ge en rekursiv tanke för höjden av ett träd. Höjden är den maximala nivån som nån av trädets noder befinner sig på. Ett träd med bara en rotnod har höjden 0, och ett tomt träd har höjden -1.

- Rekursiv tanke: Höjden är 1+max(höjd vänster, höjd höger) ... men tomt träd har höjden -1.

```
323
324 def height(p):
325     if p == None:
326         return -1
327     else:
328         h1 = height(p.left)
329         h2 = height(p.right)
330     return max(h1, h2) + 1
```

Komplexitet

1. Körtider

En viss algoritm tar 0.5 ms när antal indata är 100. Hur lång kommer körtiden att bli när antal indata är 500 om man vet att körtiden är:

- Linjär, dvs O(n)
- n log n dvs $O(n \cdot log(n))$
- kvadratisk, dvs $O(n^2)$
- kubisk, dvs $O(n^3)$

Linjär: t $\approx k * n$

$$\left.\begin{array}{l} n = 500 \ -> \ k * 500 = x \\ n = 100 \ -> \ k * 100 = 1/2 \end{array}\right\} x \approx \frac{500}{100} = 2.5 \quad \text{(dvs 5 ggr längre)}$$

Logaritmisk linjär: $t \approx k * n * log(n)$

$$\left.\begin{array}{l} n = 500 \ -> \ k * 500 * log(500) = x \\ n = 100 \ -> \ k * 100 * log(100) = 1/2 \end{array}\right\} x \approx \frac{6.2 * 500}{24.6 * 100} = 3.4 \quad \text{(dvs 6.8 ggr längre)}$$

Kvadratisk: $t \approx k * n^2$

$$\left.\begin{array}{l} n = 500 \ -> \ k * 500^2 = x \\ n = 100 \ -> \ k * 100^2 = 1/2 \end{array}\right\} x = \frac{5 * 5}{2} = 12.5 \quad \text{(dvs 25 ggr längre)}$$

Kubisk: $t \approx k * n^3$

$$\left.\begin{array}{l} n = 500 \ -> \ k * 500^3 = x \\ n = 100 \ -> \ k * 100^3 = 1/2 \end{array}\right\} x = \frac{5 * 5 * 5}{2} = 62.5 \quad \text{(dvs 125 ggr längre)}$$