Övning 3 - Tillämpad datalogi 2012

```
0  # coding: latin
```

Summering

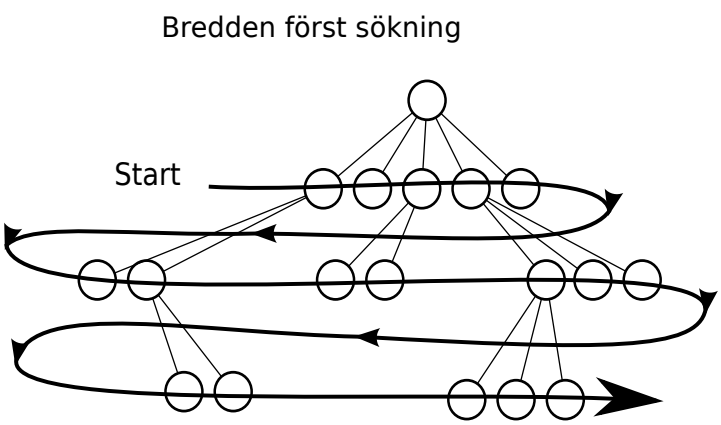
Vi gick igenom **problemträd**, sökning i problem träd med **bredden först**, och **djupet först**. Vi exemplifierade det genom att lösa två uppgifter **strykord**, med djupet först, och **sjuor i rad**, med bredden först.

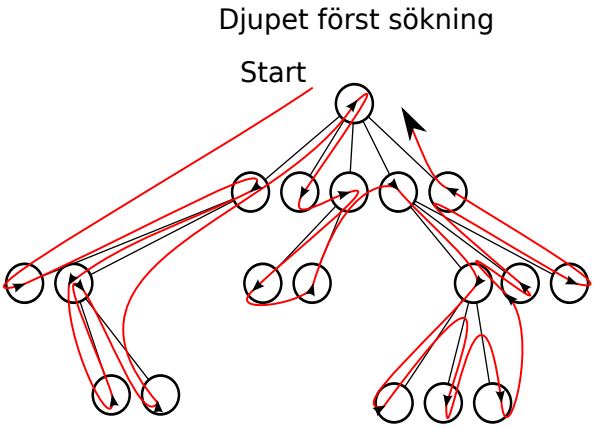
- Some imports, ignore

```
20 import sys, os
21 sys.path.append(os.getcwd())
```

Problemträd, breddenförst, djupetförst

- Ett problemräd kan ses som en enkelriktad graf med en start nod som sen ger upphov till hela trädet
- Bredden först söker av problemträdet nivå för nivå
- Djupet först söker igenom problemträden genom att gå på djupet från vänster till höger
- Djupet först sökning påminner om trädutskrifen **Preordning:** (noden) (vänster träd) (höger träd), med skillnaden att vi nu kan ha fler kanter än två till en nod. Regeln blir då istället (noden) (träd längst till vänster),...,(träd längst till höger träd)





1. Strykord

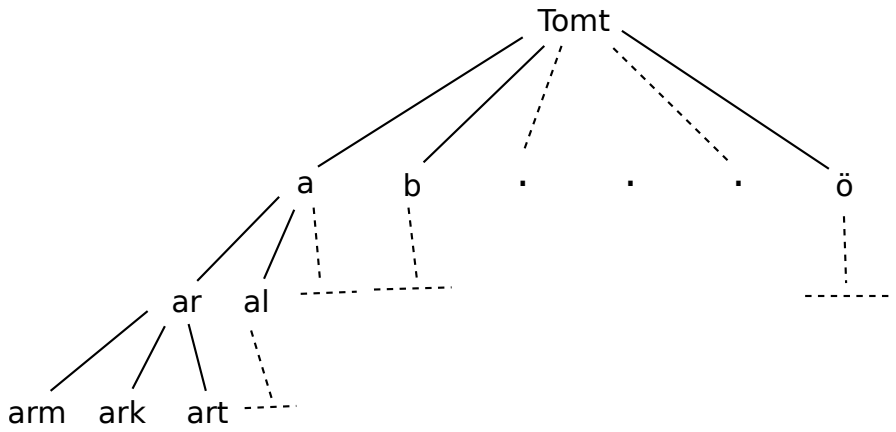
Ordet orkan är ett strykord eftersom man kan stryka sista bokstaven om och om igen och bara få riktiga ord.

orkan - orka - ork - or - o

Uppgiften är att finna det längsta strykordet i svenska språket. Ordlistan finns på fil. Rita problemträdet och jämför olika tänkbara algoritmer.

Börja med tomt ord, skapa sedan barn genom att lägga på en bokstav och kolla ifall det ordet finns. Vi vill ha uttömmande sökning, så breddenförst är inte nödvändigt. Vi använder därför en *rekursiv djupetförst*-sökning. Orden läggs i en *dict* - kan använda binärträd eller något smartare som sparar minne.

Problemträdet vi får ser ut på följande sätt:




```
59
60 # coding:iso-8859-1
61
62 def langsta_strykord(ord, ordlista):
63     """ Rekursiv djupetförst som hittar längsta strykordet givet orden
64     i ordlista """
65
66     alfabet="abcdefghijklmnopqrstuvwxyzåäö"
67     global rekord
68     if len(ord) > len(rekord):
69        rekord = ord
70     for tkn in alfabet:
71       if ordlista.has_key(ord+tkn):
72           langsta_strykord(ord+tkn, ordlista)
73
74 rekord=""
75 svenska={} #Dictionary
76
77 svenskfil = open("ss100.txt")
78 for rad in svenskfil.readlines():
79     svenska[rad.strip().lower()]=True #Ta bort returtecknet och gör om till
       upper
80
81
82 langsta_strykord("", svenska)
83 print "Längsta strykord: None"
```

```
Längsta strykord: None
```

```
82 s=''
83 for b in rekord:
84     s+=b
85     print '->', s,
```

```
-> s -> sk -> ska -> skar -> skarp -> skarpa -> skarpas -> skarpast ->
   skarpaste -> skarpastes
```

2. Sjuor till hundra

Det gäller att skriva talet 100 med enbart sjuor och dom fyra räknesätten, till exempel så här:

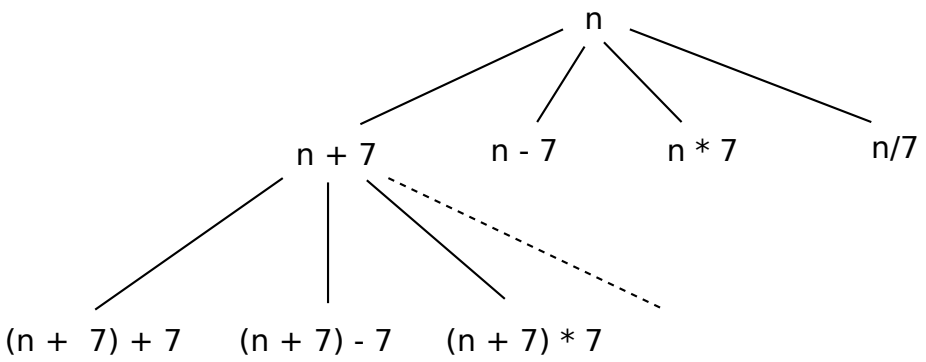
*7 +7 /7 *7 *7 =98*

Det var ett gott försök som inte nådde ända fram. För att man ska få använda / måste divisio- nen gå jämnt ut. Rita problemträdet och diskutera bästa algoritm för att avgöra OM problemet är lösbart.

Om man dessutom vill veta hur lösningen ser ut krävs en mer komplicerad datastruktur. Be- skriv den och

skissa ett program.

Vi vill hitta kortaste sättet att komma fram till hundra. Som gjort för bredden- först, där man använder en kö med heltal för att stoppa i alla tal man får från n (division omm. n = 0 (mod 7)):




```
110
111
112 from queue import Queue
113
114 q = Queue()
115 def makesons(tal):
116     global q
117     if tal==100:
118         print "Hundra!",
119         return False
120     q.put(tal+7)
121     q.put(tal-7)
122     q.put(tal*7)
123     if(tal%7==0):
124         q.put(tal/7)
125     return True
126
127 m=True
128 q.put(0)
129 while (not q.isempty()) and m:
130     m=makesons(q.get())
```

```
Hundra!
```

Notera dock att vi kommer att testa samma tal många ggr:

0 ⇒ [7, -7, 0, 0]
7 ⇒ [14, 0, 49, 1]
-7 ⇒ [0, -14, -49, ...]

Notera att vi får ett stort span av tal med många dubletter:

Totalt antal: ⇒ 19568
Antal unika: ⇒ 809
Med många stora och små tal ⇒
[−823543. − 235298. − 134456.]...[−1.0.1.2.]...[252105.352947.823543.]

För att få ut kedjan måste varje nod minnas sin "far", dvs varifrån den kom- mer. Det lättaste är att returnera den noden istf. False från makesons (istf. True returnerar man None), men det går även att använda ett Exception.

```
152
153 class Node:
154     def __init__(self, tal = 0, far = None, op='?'):
155         self.tal = tal
156         self.far = far
```

```
157         self.op = op
158
159 from queue import Queue
160 from sys import exit
161
162 q=Queue()
163 N=7
164
165
166 def makesons(far):
167     global N, q
168     tal = far.tal
169     if tal == 100:
170         raise StopIteration(far)
171     q.put(Node(tal+N, far,'+'))
172     q.put(Node(tal-N, far,'-'))
173     q.put(Node(tal*N, far,'*'))
174     if(tal%N==0):
175         q.put(Node(tal/N, far,'/'))
176
177 def writechain(p):
178     if p != None:
179         writechain(p.far)
180         print p.op, N, '->', p.tal
181
182 q.put(Node())
183
184 try:
185     while not q.isempty():
186         makesons(q.get())
187 except StopIteration as si:
188     writechain(si.args[0])
```

```
? 7 -> 0
+ 7 -> 7
+ 7 -> 14
* 7 -> 98
* 7 -> 686
+ 7 -> 693
+ 7 -> 700
/ 7 -> 100
```

Misc

Graph

- A “graph” in this this context is a collection of “vertices” or “nodes” and a collection of edges that connect pairs of vertices.
- Complexity bredden först, O(V)+O(E)=O(V+E) där V är antalet noder(vertex) och E antalet kopplingar (edges).
- Complexity djupet först, O(V)+O(E)=O(V+E) där V är antalet noder(vertex) och E antalet kopplingar (edges).

Searching

Even though a binary search is generally better than a sequential search, it is important to note that for small values of n, the additional cost of sorting is probably not worth it. In fact, we should always consider whether it is cost effective to take on the extra work of sorting to gain searching benefits. If we can sort once and then search many times, the cost of the sort is not so significant. However, for large lists, sorting even once

can be so expensive that simply performing a sequential search from the start may be the best choice. [ref http://interactivepython.org/courselib/static/pythonds/SortSearch/searching.html#lst-binarysearchpy]

Hash table

A hash table is a collection of items which are stored in such a way as to make it easy to find them later. Each position of the hash table, often called a slot, can hold an item and is named by an integer value starting at 0. For example, we will have a slot named 0, a slot named 1, a slot named 2, and so on. Initially, the hash table contains no items so every slot is empty.

- Load factor lambda=numberofitems/tablesize

Creating hashfunctions

- Folding, divide and add. For 436-555-4601, divide digits into groups of 2 (43,65,55,46,01) and add 43+65+55+46+01, giving the hash number 210 Collision resolution
- linear probing,
  - rehash(pos)= (pos+1)% sizeoftable
  - rehash(pos)= (pos+3)% sizeoftable
  - rehash(pos)= (pos+skip)% sizeoftable
- quadratic probing
  - rehash(pos)= (pos+skip)% sizeoftable, skip =1,3,5,7