Per Sedholm
DD1320 (tilda13)
2013-09-18

Övning 3

Problemträd, breddenförst, djupetförst, hashning

Person (selektorer, kodformatering)

Skriv ett program som läser personers namn och födelsedag från fil.

Vi väljer att göra en klass för att representera personer. Huvudprogrammet kan då göra

```
f = open("ovn3-E-01a.txt", "r")
lst = []
for line in f:
    (date, name) = line.strip().split(None, 1)
    lst.append(Person(name, date))
f.close()

for p in lst:
    print(p) ## assume that Person has a __str__ method
```

Vilka metoder behöver klassen ha? De självklara är

Speciella metoder `__init__` (konstruktor), `__str__` (sträng-representation) mfl. Notera skillnaden mellan

str – `__str__` ger en *läsbar sträng-version* av denna Person. Anropas av *str()*.

repr – `__repr__` ger en *entydig sträng-representation* av denna Person. Anropas av *repr()*; *str(lista)* kommer att anropa *repr* för varje element.

Selektorer för att plocka ut egenskaper för Person:en

Annat? metoder som har bieffekter, ex. skriver ut (ej returnerar) en sträng.

```
class Person(object):
    """A record for a person"""
    def __init__(self, name, bday):
        """Represent a person with the given name and birthday"""
        self.__name = name
        self.__bday = bday

    def name(self):
        """Return this person's name"""
        return self.__name

    def bday(self):
        """Return this person's birthday"""
        return self.__bday

    def __str__(self):
        """Return a string version of this Person"""
        return "<" + self.name() + " (" + self.bday() + ")>"

    def showname(self):
        """Print this person's name"""
        print(self.name())
```

Lägg märke till att

- Klassen har en kommentar
- Varje metod (inklusive *init*-metoden) har en kommentar
- Dessa är inte kod-kommentarer.
- Privata attribut (och metoder) börjar med "__".

Funktionskommentarer skrivs (helst) med *docstring*-konventioner, så att man kan auto-generera dokumentation. Kodkommentarer är till för den som läser koden, se exempel i slutversionen nedan.

Indata; resultat:

```
8/1 Graham Chapman            <Graham Chapman (8/1)>
29/3 Eric Idle                <Eric Idle (29/3)>
11/22 Terry Gilliam           <Terry Gilliam (11/22)>
1/2 Terry Jones               <Terry Jones (1/2)>
27/10 John Cleese             <John Cleese (27/10)>
5/5 Michael Palin             <Michael Palin (5/5)>
```

Men nu blev det fel. Terry Gilliam (som är amerikan) skriver ju *MM/DD*, medan de övriga (britter) skriver *DD/MM*. Det är nog bättre att byta format i filen – men ger det fortfarande samma utskrifter? (Gilliam undantaget)?

Ja, vi skrev ju selektorer! Byt till

```
1941-01-08 Graham Chapman
1943-03-29 Eric Idle
1940-11-22 Terry Gilliam
1942-02-01 Terry Jones
1939-10-27 John Cleese
1943-05-05 Michael Palin
```

```
    def bday(self):
        """Return this person's birthday"""
        (y, m, d) = self.__bday.split("-")
        m = m.lstrip("0")
        d = d.lstrip("0")
        return d + "/" + m
```

Metoden (selektorn) uppfyller *samma yttre gränssnitt*, så huvudprogrammet kan använda *exakt samma kod*, och få samma resultat.

Det finns dock ett par saker man kan vilja ändra. *Best practice* är att skriva *Efternamn, Förnamn* eftersom det är lättare att plocka isär. Har man flera olika fält vill man oftast ha annat än mellanslag för att skilja dem – t.ex. `:` (kolon).

```
1941-01-08:Chapman, Graham Arthur
1943-03-29:Idle, Eric
1940-11-22:Gilliam, Terence Vance "Terry"
1942-02-01:Jones, Terence Graham Parry "Terry"
1939-10-27:Cleese, John Marwood
1943-05-05:Palin, Michael Edward
```

Det gör att huvudprogrammet måste använda

```
    (date, name) = line.strip().split(":")
```

Vi kan sedan *split*:a namn-delen för att hitta för- resp. efternamn, och även plocka fram tilltalsnamnet.

Ska man hålla på och räkna med datum så vill man använda färdiga bibliotek, som inte bara tar hänsyn till skottår på rätt sätt, eller alla bysantinska regler för tidszoner, utan även till egenheter som:

- Följs 2:a sept 1752 av 3:e sept, eller 14:e sept? (Storbrittannien bytte från juliansk till gregoriansk kalender)
- Vad hände 30:e feb 1712? (Sverige bytte fram och tillbaka mellan juliansk och gregoriansk kalender. Det blev *mycket krångligt*. . . .)

Vi använder därför *date.datetime*.

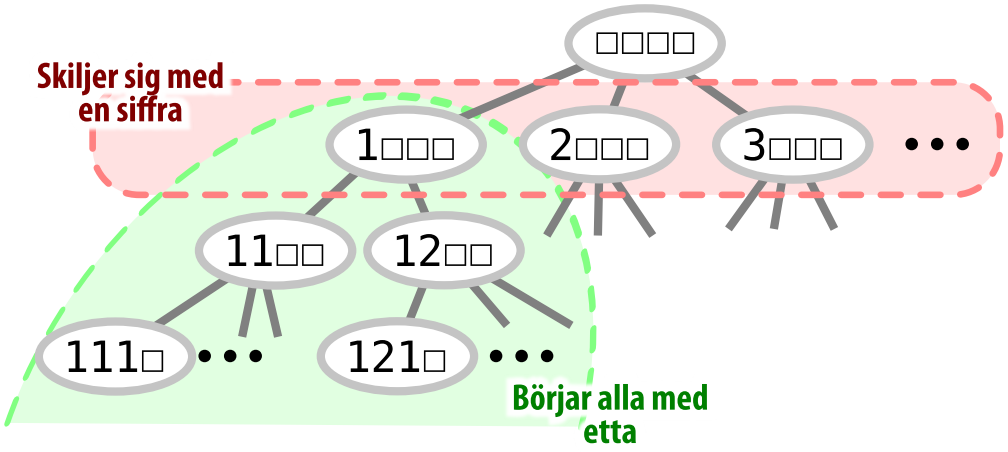
```
from datetime import date

class Person(object):
    """A record for a person"""
    def __init__(self, name, bday):
        """Represent a person with the given name and birthday"""
        (last, first) = name.split(", ")
        if '"' in first:
            called = first[first.find('"')+1:first.rfind('"')]
        elif " " in first:
            ## This is not often correct, but works here
            called = first[0:first.find(" ")]
        else:
            called = first

        self.__first = first
        self.__last = last
        self.__name = called + " " + last

        (y, m, d) = bday.split("-")
        self.__bday = date(int(y), int(m), int(d))

    def name(self):
        """Return this person's name"""
        return self.__name

    def bday(self):
        """Return this person's birthday"""
        return self.__bday.strftime("%d/%m")

    def this_years_birthday(self):
        """Return the weekday of this person's birthday in the
        current year"""
        today = date.today()
        bd = date(today.year, self.__bday.month, self.__bday.day)
        ## date.weekday() is a number, this gives the full name
        return bd.strftime("%A")

    def __str__(self):
        """Return a string version of this Person"""
        return "<" + self.name() + " (" + self.bday() + ")>"

f = open("ovn3-E-01c.txt", "r")
lst = []
for line in f:
    (d, n) = line.strip().split(":")
    lst.append(Person(n, d))
f.close()

#for p in lst:
#    print(p.name() + ":", p.this_years_birthday())

print("\n".join(map(str, lst)))

print("\n".join(map(lambda p: p.name() + ": " + p.this_years_birthday(),
                    lst)))
```

4. Taltävling

Under sommaren har det gått en tävling i radio där det gäller att så snabbt som möjligt hitta ett tal som uppfyller två villkor. Första villkoret, som gäller hela tävlingen, är att siffrorna i talet måste vara strikt stigande (23578 är OK men inte 235578 eller 23587). Andra villkoret är nytt för varje dag och kan t ex vara att talet ska vara ett primtal, jämnt delbart med 599 eller en tvåpotens. Beskriv utförligt en breddenförst- eller djupetförstsökning (effektivare metod ger fler poäng) som hittar det minsta talet som uppfyller villkoren. Vilka klasser och metoder behövs?

Antag att vi endast har fyrsiffriga tal. Hur söker man igenom dem?

1. Utgå från startläge, elementet e
2. Skapa ”barn-element”, $e_1, e_2, e_3, \ldots$, men filtrera bort ”dum-barn”.
3. Lägg varje barn i en kö (BFS) eller stack (DFS).
4. Plocka nästa element ifrån kön/stacken, återgå till (1).



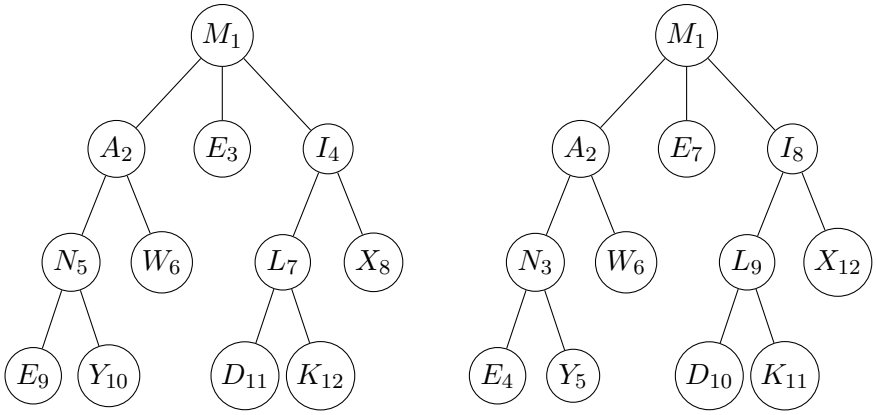
Djupetförst söker igenom ”alla tal som börjar med 1”. Med godtyckligt många siffror i talen blir det ganska många…

För att lösa uppgiften vill man varje gång man går från noden n till dess barn filtrera bort dåliga barn – de som inte uppfyller villkoren.

Att se siffror som strängar är oeffektivt (måste konvertera för att se ifall de är delbara etc), men annars är det svårt att veta att siffrorna är strikt stigande.

Breddenförst, djupetförst

Påverkar bredden- eller djupetförst vilken ordning man får ut elementen? Ja! Se på trädet nedan:




```
Wordlist = [ "MA", "ME", "MI", "MAN", "MAW", "MIL", "MIX", "MANE",
             "MANY", "MILD", "MILK" ]
Alphabet = "".join([chr(c) for c in range(ord('A'), ord('Z'))])

def recurseQueue(word, s):
    for c in Alphabet:
        child = word + c
        if child in Wordlist:
            s.put(child)

print("Queue (breadth-first)")
s = Queue()
s.put("M")
while not s.isEmpty():
    e = s.get()
    print(e, end=' ')
    recurseQueue(e, s)
print(".")

# Queue (breadth-first)
# M MA ME MI MAN MAW MIL MIX MANE MANY MILD MILK .

def recurseStack(word, s):
    for c in Alphabet[::-1]:
        child = word + c
        if child in Wordlist:
            s.push(child)

print("Stack (depth-first)")
s = Stack()
s.push("M")
while not s.isEmpty():
    e = s.pop()
    print(e, end=' ')
    recurseStack(e, s)
print(".")

# Stack (depth-first)
# M MA MAN MANE MANY MAW ME MI MIL MILD MILK MIX .
```

1. Strykord

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

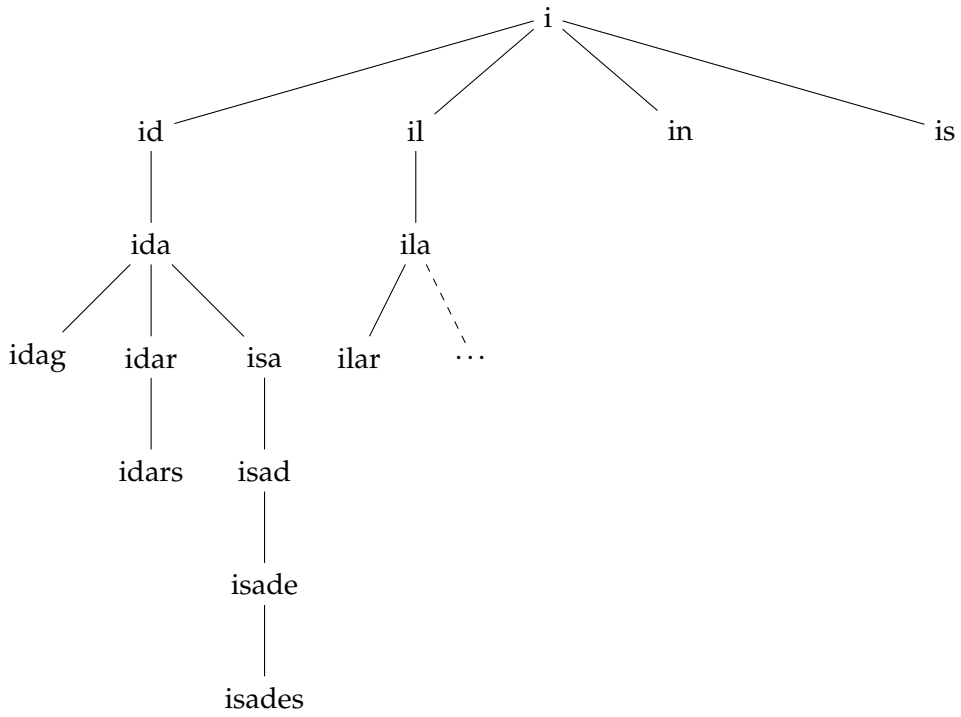
orkan — orka — ork — or — o

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.

I Ubuntus svenska ordlista är enda enbokstavs-orden ”i”, ”å” och ”ö”; enligt SAOL är varje bokstav ett ord. Vi kommer dock att hitta:



där det längsta ordet kommer att ändras som:

```
i -> id -> ida -> idag
               == idar -> idars
                       == idens
                       == idets
                       == ilars
                       == ilats
                       == isade -> isades ...
```

```
class Swedish(object):
    __words = "/usr/share/dict/swedish"
    def __init__(self):
        self.__swe = {}
        words = open(Swedish.__words, mode='r', encoding='iso-8859-1')
        for w in words:
            wu = w.strip().lower()
            self.__swe[wu] = True
        words.close()
    def exists(self, w):
        return w in self.__swe

longest = ''
swedish = Swedish()
# Note: we convert all known words to lower-case
alphabet = "".join([chr(c) for c in range(ord('a'), ord('z'))]) + "ääö"

def find_removable(word):
    """Recursive depth-first search"""
    global longest
    if len(word) > len(longest):
        longest = word
    for c in alphabet:
        if swedish.exists(word + c):
            find_removable(word + c)

find_removable("")
print("Longest removable word:", longest)
# Longest removable word: åtalades
```

Fotnot: Detta program kommer inte att gå från `armé` → `armen`, eftersom det inte ser `é` och `e` som samma bokstav. (Vi bygger aldrig på é, è, ë, ê, ...)

Egentligen skulle vi vilja ta hänsyn till bokstäver med diakritiska tecken som trema (*umlaut*), tilde, etc. Men i Unicode så kan bokstaven ”Å” representeras som:

U+00C5 (Latin capital letter A with ring above)
U+0041 U+030A (Latin capital letter A, Combining ring above)*
som dessutom ser ut precis som
U+212B (Angstrom sign)

så en lösning skulle behöva kunna hantera alla fall.†

*används t.ex. i Mac OS X:s filnamn

†Finns metoder för detta, modulen *unicodedata* innehåller funktioner för att *normalisera* genom *canonical decomposition* och *canonical composition*.

2. Sjuor till hundra

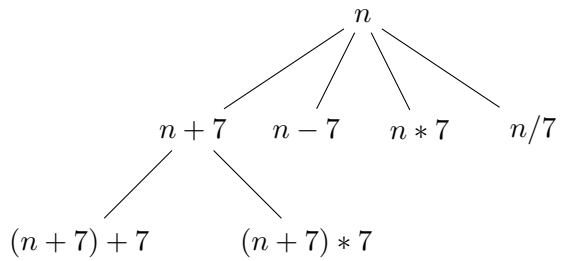
Det gäller att skriva talet 100 *med enbart sjuor och dom fyra räknesätten, till exempel så här:*

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

Det var ett gott försök som inte nådde ända fram. För att man ska få använda / måste divisionen 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. Beskriv 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. $n \equiv 0 \pmod 7$):




```
q = Queue()
N = 7
def makesons(num):
    global N
    if num == 100:
        print("Target reached")
        return False
    q.put(num + N)
    q.put(num - N)
    q.put(num * N) # NB: 0*7 == 0...
    if num % 7 == 0:
        q.put(int(num / N))
    return True


m = True
q.put(0)
while not q.isempty() and m:
    e = q.get()
    m = makesons(e)
```

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

$$\begin{aligned} 0 &\Rightarrow \{7, -7, \mathbf{0}, \mathbf{0}\} \\ 7 &\Rightarrow \{14, \mathbf{0}, 49, 1\} \\ -7 &\Rightarrow \{\mathbf{0}, -14, -49, \ldots\} \end{aligned}$$

Vi kommer även att ha med många *mycket* stora/små tal:

```
$ python ovn3-02.py > ovn3-02.out
$ wc -l ovn3-02.out
5188
$ uniq ovn3-02.out | wc -l
4869    ## 319 duplicates
$ sort -nu ovn3-02.out
-117649, -33614, -19208, ..., 0..24, 27, 28, ..., 33614, 36015, 50421, 117649
```

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

```
class Node(object):
    def __init__(self, num = 0, father = None, op = '?'):
        self.num = num
        self.father = father
        self.op = op

q = Queue()
N = 7

def makesons(father):
    global N
    if father.num == 100:
        print("Target reached")
        raise StopIteration(father)
    q.put(Node(father.num + N, father, '+'))
    q.put(Node(father.num - N, father, '-'))
    q.put(Node(father.num * N, father, '*')) # NB: 0*7 == 0...
    if father.num % 7 == 0:
        q.put(Node(father.num / N, father, '/'))

def writechain(p):
    if p != None:
        writechain(p.father)
        print(p.op, N, '->', p.num)

try:
    q.put(Node())
    while not q.isempty():
        e = q.get()
        #print(e)
        makesons(e)
except StopIteration as si:
    print("«breaking iterations»")
    writechain(si.args[0])
# ? 7 -> 0
# + 7 -> 7
# + 7 -> 14
# * 7 -> 98
# * 7 -> 686
# + 7 -> 693
# + 7 -> 700
# / 7 -> 100
```