TDP002 - Algoritmer, datatyper och funktionsobjekt

Pontus Haglund

Department of Computer and information science

LINKÖPING UNIVERSITY

1 Rekursion
2 Algoritmer
3 Representation av tal
4 Polymorfism
5 Funktionsobjekt
6 Högra ordningens funktioner
7 Avslutning

LINKÖPING UNIVERSITY

1 Rekursion
2 Algoritmer
3 Representation av tal
4 Polymorfism
5 Funktionsobjekt
6 Högra ordningens funktioner
7 Avslutning

LINKÖPING UNIVERSITY

Rekursion

Ett annat sätt att göra repetition

Räkna

Räkna från i upp till 10

Iterativ

```
def count_print(i):
    for j in range(i,10):
        print(i)
```

Rekursiv

```
def count_print(i):
    print(i)
    if i < 10:
        count_print(i+1)
```

Factorial

Iterativ

```
def factorial(n):
    summan = 1
    for i in range(1, n+1):
        summan *= i
    return summan
```

Rekursiv

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
```

Summera listor

Iterativ

```
def sum_list(l):
    summa = 0
    for e in l:
        summa += e
    return summa
```

Rekursiv

```
def sum_list(l):
    if not l:
        return 0
    else:
        return l[0] + sum_list(l[1:])
```

Summera listor i listor...

Iterativ - är detta lika bra?

```
def sum_arb_list(l):
    tot = 0
    for x in l:
        if isinstance(x, list):
            for y in x:
                tot += y
        else:
            tot += x
    return tot
```

Rekursiv

```
def sum_arb_list(l):
    tot = 0
    for x in l:
        if not isinstance(x, list):
            tot += x
        else:
            tot += sum_arb_list(x)
    return tot
```

Towers of hanoi

```
A = [4, 3, 2, 1]
B = []
C = []

def move(n, source, target, auxiliary):
    if n > 0:
        # move n - 1 disks from source to aux, so they are out of the way
        move(n - 1, source, auxiliary, target)

        # move the nth disk from source to target
        target.append(source.pop())

        # Display our progress
        print(A, B, C, '###############', sep = '\n')

        # move the n - 1 disks that we left on auxiliary onto target
        move(n - 1, auxiliary, target, source)

# initiate call from source A to target C with auxiliary B
move(len(A), A, C, B)
```

1 Rekursion
2 Algoritmer
3 Representation av tal
4 Polymorfism
5 Funktionsobjekt
6 Högra ordningens funktioner
7 Avslutning

LINKÖPING UNIVERSITY

Algoritmer

Sortering, sökning och att göra egna

Linjär sökning i lista

```
# Hitta var siffran 8 är i följande lista:
li = [16, 7, 8, 32, 1]
```

```
def linear_search(x, seq):
    for index in range(len(seq)):
        if x == seq[index]:
            return index
    return -1
```

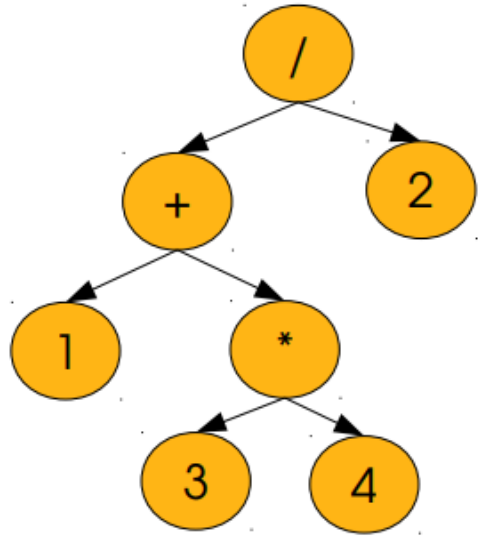
I värsta fallet måste vi söka igenom hela listan

Träd

Uttryck/term: (1 + (3 * 4)) / 2

- Graf
- Består av noder och bågar
- En rot
- Riktade pilar
- inga loopar eller flätor

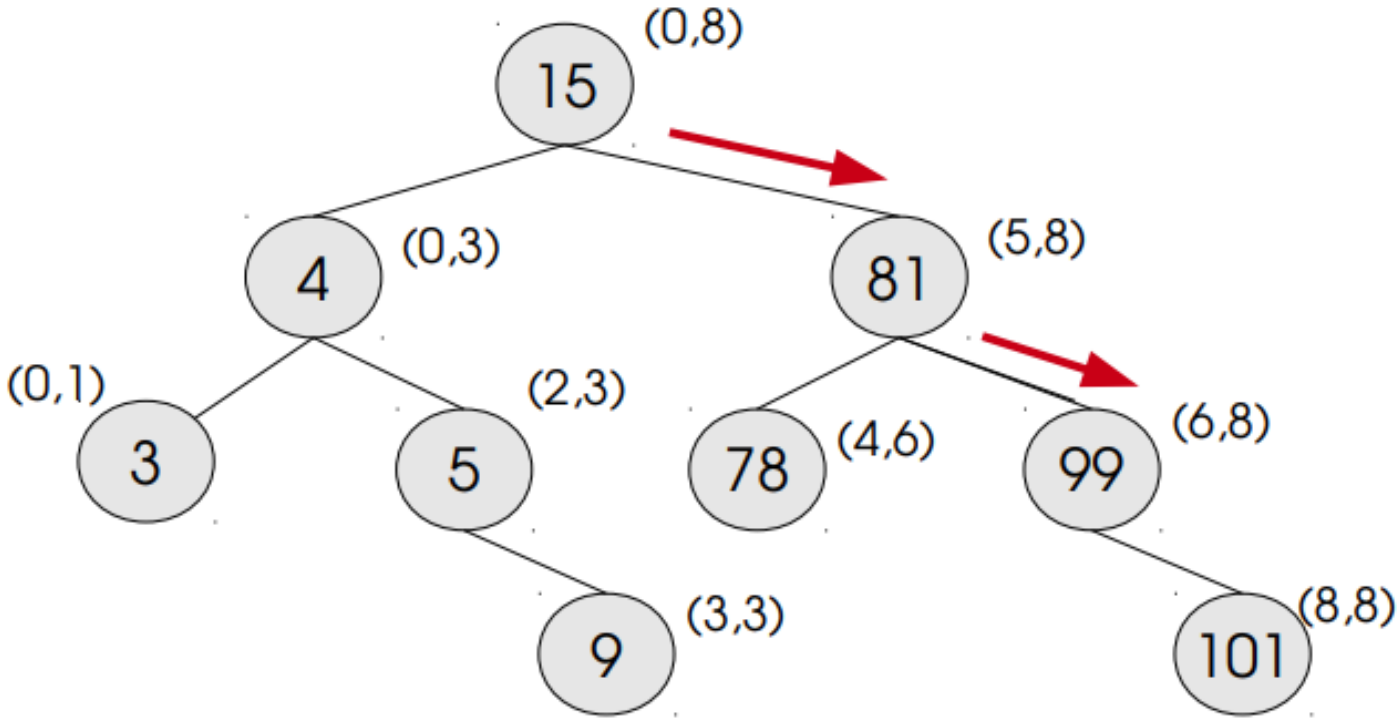
Trädstruktur:



Kan exempelvis skrivas som en lista av listor:

Binär sökning i ordnad lista

Hitta 99? [3, 4, 5, 9, 15, 78, 81, 99, 101]



Implementera binary search

```
def bin_search(x, seq):
    low = 0
    high = len(seq) - 1
    while low <= high :
        mid = (low + high) // 2
        if seq[mid] < x:
            low = mid + 1
        elif seq[mid] > x:
            high = mid - 1
        else:
            return mid
    return -1
```

Sortering

- Ofta bra att hålla data sorterad
- Bra att göra det ’in place’
- Görs ofta genom att byta plats på element i listan
- Utmaningar?

Exempel bubble sort

https://en.wikipedia.org/wiki/Bubble_sort

Exempel selection sort

https://en.wikipedia.org/wiki/Selection_sort

Exempel quick sort

https://en.wikipedia.org/wiki/Quicksort

1 Rekursion
2 Algoritmer
3 Representation av tal
4 Polymorfism
5 Funktionsobjekt
6 Högra ordningens funktioner
7 Avslutning

LINKÖPING UNIVERSITY

Hur sparar datorn tal?

All data representeras av datorn med hjälp av binära tal. Hur representeras bas-10 talet 26?

```
Dec: 26
Bin: 00011010
```

Räkna med olika talbaser

```
För alla talbaser gäller:
x*y**z
x är siffran
y är talbasen
z är positionen från höger med start 0

Binära talet 101 är:
1*2**2 + 0*2**1 + 1*2**0 = 4+0+1 = 5 (dec)

Oktala talet 275:
2*8**2 + 7*8**1 + 5*8**0 = 128 + 56 + 5 = 189 (dec)

Decimala talet 235:
2*10**2 + 3*10**1 + 3*10**0 = 200 + 30 + 5 = 236 (dec)
```

Literaler och fysisk representation

```
>>> 0o352
234
>>> 0xEA
234
>>> 0xEA + 0o352
468
```

- 0o352 är en literalen av en oktalt tal
- Den fysiska representationen av detta tal är binärt

Skriva tecken i python

```
>>> '£' # ange tecken: att föredra
£
>>> '\u00A3' # anger unicode tecknet
£
>>> '\N{POUND SIGN}' # samma som ovan
£
>>> '\xa3' # UTF kodningen direkt, platforms beroende
£
```

1 Rekursion
2 Algoritmer
3 Representation av tal
4 Polymorfism
5 Funktionsobjekt
6 Högra ordningens funktioner
7 Avslutning

LINKÖPING UNIVERSITY

Polymorfism

- Generella funktioner som ger oss återanvändbar kod.
- Exempelvis:
  - print
  - len
  - in
  - +
  - ...

Exempel - intersect

```
def intersect(list_a, list_b):
    results=[]
    for element in list_a:
        if element in list_b:
            results.append(element)
    return results
```

```
intersect([1,2,3,4],[0,2,4,6])        ==> [2, 4]
intersect([1,2,3,4],(0,2,4,6))        ==> [2, 4]
intersect(['a','b','c','d'],'Acdfgh') ==> ['c', 'd']
```

1 Rekursion
2 Algoritmer
3 Representation av tal
4 Polymorfism
5 Funktionsobjekt
6 Högra ordningens funktioner
7 Avslutning

LINKÖPING UNIVERSITY

Funktioner är objekt

- Funktioner är källkodsblock vi namngivit
- ret_one innehåller den koden
- Evalueras med call-funktionen
- Kan tilldelas till andra variabler
- Kan skickas som en parameter

```
def ret_one():
    return 1

print(ret_one)   ==> <function ret_one..>

print(ret_one()) ==> 1
```

Exempel - tilldelning

```
def ret_one():
    return 1

x = ret_one()
print(x())       # ==> 1
print(ret_one()) # ==> 1
```

```
 1
 1
```

1 Rekursion
2 Algoritmer
3 Representation av tal
4 Polymorfism
5 Funktionsobjekt
6 Högra ordningens funktioner
7 Avslutning

LINKÖPING UNIVERSITY

Högra ordningens funktioner

- Funktioner som hanterar funktionsobjekt..
  1. Funktionsobjekt som parametrar till funktion
  2. Funktionsobjekt som returneras av en funktion
- Python har stöd för detta
  - Referenser till funktionsobjekt via namn
  - Lambda
  - Inbyggda funktioner (map, filter..)

Funktioner som parameterar

```
def add_all(li):
    result = 0
    for i in li:
        result = result + i
    return result
```

```
def sub_all(li):
    result = 0
    for i in li:
        result = result - i
    return result
```

```
li = [1,2,3]
print( add_all(li) ) ==> 6
print( sub_all(li) ) ==> -6
```

Vad hette den där regeln igen?

Funktioner som parameterar

- DRY - Don’t Repeat Yourself

```
def sum_all(li, fun):
    result = 0
    for i in li:
        result = fun(result, i)
    return result
```

forts.

```
def sum_all(li, fun):
    result = 0
    for i in li:
        result = fun(result, i)
    return result

def add(x, y):
    return x+y

def sub(x, y):
    return x-y

li = [1, 2, 3]
print(sum_all(li, add)) #==> 6
print(sum_all(li, sub)) #==> -6
```

Hur lägger vi till funktionalitet för att multiplicera alla talen?

Lambda - anonyma funktioner

- Men Pontus, det är jobbigt att skriva de där funktionerna!
- Det är därför vi har möjlighet att skapa anonyma funktioner i Python, så kallade Lambda-funktioner
- De skiljer sig från vanliga funktioner eftersom de är uttryck istället för satsblock

```
#Skrivs på formatet:
lambda <param-1>, <param -2>...<param-n>: <uttryck >
lambda x, y: x + y
```

Lambda istället för add och sub

- Används och fungerar i princip som vanliga funktionsobjekt
- Resultatet av uttrycket returneras automatiskt

```
add = lambda x, y: x+y
li = [1, 2, 3]
print(sum_all(li, add)) ===> 6
print(sum_all(li, lambda x, y: x-y)) ===> -6
```

Returnera funktionsobjekt

Vi kan skicka funktioner som parameterar. Det är också möjligt att få skicka dem som returvärden.

```
def inc(number):
    return lambda x: x + number

inc_one = inc(1)
inc_two = inc(2)

print( inc_one(4) ) ==> 5
print( inc_two(4) ) ==> 6
print( inc(4)(3) ) ==> ?
```

Inbyggda funktioner

- Vanligt att inbyggda funktioner i python tar funktionsobjekt som parametrar
- Map
- Filter
- Sort

Exempel på map

```
[el_1, el_2, ... el_n]
```

```
[f(el_1), f(el_2), ... f(el_n)]
```

```
add_one = lambda x: x + 1
li = [1, 2, 3, 4]
list(map(add_one , li))
# [2, 3, 4, 5]

li_1 = [1, 2, 3, 4]
li_2 = [2, 3, 4, 5]
add_pair = lamda x, y: x + y
list(map(add_pair , li_1, li_2))
# [3, 5, 7, 9]
```

- Applicerar funktionen på varje element i listan
- Kan köra mot flera listor parallellt
- Antal argument till funktionen är antal listor
- Listorna måste vara lika långa
- Ger tillbaka ett map-objekt

1 Rekursion
2 Algoritmer
3 Representation av tal
4 Polymorfism
5 Funktionsobjekt
6 Högra ordningens funktioner
7 Avslutning

LINKÖPING UNIVERSITY

Innan vi slutar...

- TDP001
- TDP002
- ’sista föreläsningen’

www.liu.se

LINKÖPING UNIVERSITY