UPPSALA UNIVERSITET

MA1: L01

Tom Smedsaas

Introduktion till rekursion

UPPSALA UNIVERSITET

Algoritmer

Definition: En *algoritm* är en sekvens av väldefinierade instruktioner som löser ett problem eller utför en beräkning med ett ändligt antal steg.

Beståndsdelar:

- *sekvens* (en följd av instruktioner)
- *selektion* (`if`, `elif`, `else`)
- *iteration* (`for`, `while`)
- *abstraktion* (funktioner, metoder)

Exempel: Fakultetberäkning

Fakulteten kan definieras *iterativt*:

$$n! = \begin{cases} 1 & \text{om } n = 0 \\ 1 \cdot 2 \cdot 3 \cdot \ \ldots \ \cdot n & \text{om } n > 0 \end{cases}$$

vilket kan uttryckas i kod med någon form av loop:

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

Rekursiv fakultetsberäkning

Fakulteten kan också definieras *rekursivt*:

$$n! = \begin{cases} 1 & \text{om } n = 0 \\ n \cdot (n-1)! & \text{om } n > 0 \end{cases}$$

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

```
def fac(n):
    result = 1
    if n > 0:
        result =n*fac(n-1)
    return result
```

UPPSALA UNIVERSITET

Demonstration med hjälp av debuggern i Thonny

Exempel: Beräkna x^n

Operationen x^n där x reellt och n heltal:

Iterativt:
$$x^n = \begin{cases} 1 & \text{om } n = 0 \\ x \cdot x \cdot x \cdot \ \ldots \ \cdot x & \text{om } n > 0 \end{cases}$$

Rekursivt:
$$x^n = \begin{cases} 1 & \text{om } n = 0 \\ x \cdot x^{n-1} & \text{om } n > 0 \end{cases}$$

Om n får vara negativt?

$$x^n = \begin{cases} \dfrac{1}{x^{-n}} & \text{om } n < 0 \\ 1 & \text{om } n = 0 \\ x \cdot x^{n-1} & \text{om } n > 0 \end{cases}$$

```
def power(x, n):
    if n < 0:
        return 1./power(x, -n)
    elif n == 0:
        return 1
    else:
        return x*power(x, n-1)
```

UPPSALA UNIVERSITET

Varför rekursion?

- Kraftfullt sätt att *hitta* algoritmer.
- Ett kraftfullt sätt att hitta *effektiva* algoritmer.
- *Naturligt* i många problem.

Men också

- Kraftfullt sätt att konstruera ineffektiva algoritmer

UPPSALA UNIVERSITET

Rekursion generellt

1. Dela upp problemet i ett eller flera delproblem av samma typ.
2. Lös delproblemen (rekursivt)
3. Kombinera lösningarna till delproblemen till en lösning av ursprungsproblemet.

Det måste finnas minst ett rekursionsterminerande fall, så kallade basfall.

Fakultetsberäkningen $n!$:

- **Ett** delproblem: Beräknas $(n-1)!$.
- **Kombinera**: Multiplicera med n.
- **Basfall**: $n = 0$.

UPPSALA UNIVERSITET

Hur hittar man delproblemen?

I ovanstående exempel definieras problemstorleken av ett tal *n* som ges som parameter: $n!$ och x^n.

Annat exempel:

`number_of_digits(x)`

som ska returnera antalet decimala siffror i heltalet x:

`number_of_digits(125)` → `3`

`number_of_digits(2341562)` → `7`

```
def number_of_digits(x):
    if x < 10:
        return 1
    else:
        return 1 + number_of_digits(x//10)
```

Ibland kan delproblemen göras på olika sätt

Skriv funktionen `reverse(lst)` som returnerar en ny lista där elementen i listan `lst` kommer i omvänd ordning.

(Vi ignorerar alla inbyggda funktioner och metoder för att reversera.)

```
def reverse(lst):
    if len(lst) <= 1:
        return lst
    else:
        mid = len(lst)//2
        return reverse(lst[mid:]) + reverse(lst[:mid])
```

Fråga: Hur gör vi samma operation på en sträng?

UPPSALA UNIVERSITET

Ibland är rekursionen självklar

Antag att vi har följande *iterativa* funktion:

```
def reverse_list(lst):
    result = []
    for x in lst:
        result.insert(0, x)
    return result
```

In: `[1, [2, [3, 4]], [[5, 6], 7], [8, 9]]`

Ut: `[[8, 9], [[5, 6], 7], [2, [3, 4]], 1]`

Ibland är rekursionen självklar …

Antag nu att funktionen även ska vända på alla ingående dellistor dvs

In: `[1, [2, [3, 4]], [[5, 6], 7], [8, 9]]`

Ut: `[[9, 8], [7, [6, 5]], [[4, 3], 2], 1]`

Enkelt! Vi har ju en funktion som reverserar listor:

```
def reverse_list(lst):
    result = []
    for x in lst:
        result.insert(0, x)
    return result
```

```
def reverse_list(lst):
    result = []
    for x in lst:
        if type(x) is list:
            x = reverse_list(x)
        result.insert(0, x)
    return result
```

UPPSALA UNIVERSITET

Exempel med två basfall

En funktion som tar emot två listor med tal och returnerar en ny lista där elementen består av talen talen summerade parvis:

```
summa([1,2,3], [1,2,3,4,5])   →   [2,4,6,4,5]
summa([1,2,3], [5,7])         →   [5,9,3]
```

```
def summa(x, y):
    if len(x) == 0:
        return y
    elif len(y) == 0:
        return x
    else:
        return [x[0] + y[0]] + summa(x[1:], y[1:])
```

UPPSALA UNIVERSITET

The end

.