UPPSALA UNIVERSITET

Sammanfattning av modul MA1

UPPSALA UNIVERSITET

Rekursion

- Dela upp problemet i (ett eller flera) delproblem av samma slag men mindre.
- Lös delproblemen.
- Kombinera lösningarna av delproblemen till en lösning av ursprungsproblemet.

Det måste finnas ett eller flera basfall.

Det blir vanligtvis bäst kod om man använder de enklaste basfallen (t.ex. 0 i stället för 1 i Hanois torn eller en tom lista i stället för en lista med ett element).

UPPSALA UNIVERSITET

Varför rekursion

- Generell teknik för problemlösning.
- Lätt att hitta lösningen.
- Lätt att hitta effektiva lösningar.
- Naturligt i många sammanhang.
- Särskilt bra vid rekursivt definierade strukturer.

Men

- Lätt att producera hopplöst långsamma program.
- Problem med rekursionsdjup.

UPPSALA UNIVERSITET

Vad är avgörande?

Avgörande är *antalet* delproblem och *storleken* på delproblemen.

- Om vi har två eller fler delproblem med *nästan samma storlek* som ursprungsproblemet så har vi en exponentiell tillväxt.
  Exempel: Hanois torn, Fibonacci.

- Oftast bättre med *två* delproblem som har *halva storleken* är *ett* delproblem som är *nästan lika stort* som ursprungsproblemet.
  Exempel: mergesort – insticksortering

- I regel bra att *balansera* storleken på delproblemen.

- Undvik rekursion över långa strukturer – problem med stackdjupet.

UPPSALA UNIVERSITET

Asymptotisk notation

Ett sätt att beskriva hur tiden för en algoritm växer med problemstorleken oberoende av dator, programmeringsspråk etc.

När använder man $\mathcal{O}, \Theta,$ repektive Ω ?

- Θ ger mest information.
- $\mathcal{O}$ är en *övre* gräns (ej nödvändigtvis ”tät”).
- Ω är en *undre* gräns.

UPPSALA UNIVERSITET

Ordo, Omega eller Theta?

- Om du har en ”bra” $\mathcal{O}$-funktion kan den användas för att säga att en algoritm är bra.
  Exempel: Om du hittat på en sorteringsalgoritm är det bra att kunna säga att den är $\mathcal{O}(n \log n)$ men meningslöst att säga att den är $\mathcal{O}(n^2)$.

- Om du vill säga att en algoritm är dålig kan du använda Ω.
  Exempel: Om någon kommer med en sorteringsalgoritm kan du säga att den är inte så bra om den är $\Omega(n^2)$ och att den är usel om den är $\Omega(n^3)$. Det är dock meningslöst att säga att den är $\Omega(n \log n)$.

- Θ är mest informativ. Använd om möjligt.

UPPSALA UNIVERSITET

Vad är bra och vad är dåligt?

- $\Theta(a^n)$ är *dåligt* om $a > 1$.
- $\Theta(\log n)$ är *mycket bättre* än $\Theta(n)$.
- $\Theta(n \log n)$ är *mycket bättre* än $\Theta(n^2)$.
- $\Theta(n)$ är *inte så mycket bättre* än $\Theta(n \log n)$.

UPPSALA UNIVERSITET

Tiduppskattningar

Om vi vet att en algoritm är $\Theta(f(n))$ så kan vi uppskatta tidsåtgången $t(n)$ för stora problem med uttrycket

$$t(n) = c \cdot f(n)$$

och uppskatta konstanten genom att mäta tiden för något n.

Man bör verifiera modellen genom att mäta tiden för några olika n.

Använd inte för små n – det blir säkrare uppskattningar ju större värden man mäter för.

UPPSALA UNIVERSITET

Exempel

För att verifiera komplexiteten för en viss algoritm är det bra att ha strategi för hur n ska varieras.

Att dubbla värdet passar bra om man har (tror sig ha) polynomial komplexitet:

- För en $\Theta(n)$-algoritm så bör tiden dubblas om n dubblas.
- För en $\Theta(n^2)$-algoritm så gäller $\frac{t(2n)}{t(n)} = \frac{c \cdot (2n)^2}{c \cdot n^2} = 4$
- För en $\Theta(n^3)$-algoritm så gäller $\frac{t(2n)}{t(n)} = \frac{c \cdot (2n)^3}{c \cdot n^3} = 8$

UPPSALA UNIVERSITET

Exempel

- För en $\Theta(\log n)$-algoritm så är det bra att kvadrera: $\frac{\log n^2}{\log n} = 2$

- För en $\Theta(n \log n)$-algoritm är en dubblering användbar:

$$\frac{t(2n)}{t(n)} = \frac{c \cdot 2n \log 2n}{c \cdot n \log n} = 2 \cdot \frac{\log 2 + \log n}{\log n} = 2 + \frac{1}{\log n} \approx 2 \text{ för stora } n.$$

- För en exponentiell komplexitet, dvs $\Theta(a^n)$ så passar n och $n+1$:

$$\frac{c \cdot a^{n+1}}{c \cdot a^n} = a$$

UPPSALA UNIVERSITET

Behöver vi bry oss?

Ja! Det finns fortfarande många problem där datorkraften begränsar oss.

Ett axplock:

- fysik och teknik: aerordynamik, väderprognoser, …
- biologi: bioinfomatik, genomsekvensering, …
- realtidssystem: robotar, självkörande bilar, …
- animation: datorspel, filmindustri, …
- infomationssökning: google, …

UPPSALA UNIVERSITET

Men

Citat från ”*The elements of programming style*” av Kernighan och Ritchie:

- Correctness is much more important than speed!
- Do not sacrifice clarity for small efficiency gains!
- Do not sacrifice simplicity for small efficiency gains!
- Do not sacrifice modifiability for small efficiency gains!
- If the program is too slow: Find a better algorithm!

UPPSALA UNIVERSITET

Några nya Python-detaljer

Man kan deklarera funktioner inuti funktioner.

Första exemplet:

```
def power(x, n):

    def sqr(x):
        return x*x

    if n<0:
        return 1./power(x, -n)
    elif n==0:
        return 1
    elif n%2==0:
        return sqr(power(x, n//2))
    else:
        return x*sqr(power(x, (n-1)//2))
```

UPPSALA UNIVERSITET

Nya Python-detaljer

Andra exemplet:

```
def fib(n):
    memory = {0:0, 1:1}

    def _fib(n):
        if n not in memory:
            memory[n] = _fib(n-1) + _fib(n-2)
        return memory[n]

    return _fib(n)
```

Nya Python-detaljer

Funktioner är *objekt* som kan lagras i variabler, listor, lexikon, …

```
sort_functions = [ins_sort_iter, merge_sort, psort, sorted]

for sort in sort_functions[1:]:
    print(f'\n ***{sort.__name__}***')
    for n in [100000, 200000, 400000, 800000]:
        lst = []
        for i in range(n):
            lst.append(random.random())
        tstart = time.perf_counter()
        lst = sort(lst)
        tstop = time.perf_counter()
        print(f" Time for {n}\t : {tstop - tstart:4.2f}")
```

UPPSALA UNIVERSITET

The end

.