Complexitat en temps

Tecnologia de la Programació

Sebastià Vila-Marta

Enginyeria de Sistemes TIC
Universitat Politècnica de Catalunya
http://www.epsem.upc.edu

29 d'abril de 2020

- En el tema anterior...
- Cost d'execució en temps
- Anàlisi empírica
- Anàlisi teòrica
- Per a la setmana vinent ...

En el tema anterior...

- Hem vist com la recursivitat pot ajudat a solucionar problemes difícils que, d'altra manera, serían complicats de plantejar.
- Hem conegut l'esquema de backtracking, que permet resoldre problemes de cerca de solucions en espais de cerca complicats.

Quan ràpid és un programa?

- Una pregunta habitual quan s'escriu un programa que fa càlculs sofisticats és: quan triga en executar-se?
- Podem mesurar-ho usant un «rellotge» amb el següent montatge:

```
from time import clock
start = clock()
for a in range(1000):
    # a mesurar
    l = [2] * 20000
    x = 0
    for i in l:
        x += i
mesura = rell() – start
print '{0:8.5f} ms'.format(mesura)
```

- Si l'executem diverses vegades tenim els valors:

  6.75 ms, 6.83 ms i 6.76 ms

  Els valors no són constants !

Quan ràpid és un programa? (II)

- Què estem mesurant realment?
  - El temps real?
    És el temps que marca el rellotge. Pot introduïr variacions segons la càrrega del computador.
  - El temps de CPU?
    És el temps que la CPU gasta en executar el programa. Té una variació menor, però no s'acosta al que percep l'usuari.

El cas de la cerca

- Cerca d'un número natural en una llista ordenada creixent de naturals.
- Per cada mida de llista, una única llista.
- Calcularem la mitja de 1000 observacions.
- Variarem els diversos factors que afecten al temps:
  - La mida de les dades.
  - La disposició de les dades per a una mateixa mida.
  - La potència del computador.
  - L'algoritme de càlcul.

Modifiquem la mida i la disposició

- Sempre és una cerca amb garantia d'èxit.
- Fem 1000 experiments amb cada configuració.
- Modifiquem la mida de les dades i ho provem amb 3 disposicions per cada mida.
- Els resultats per diverses mides de dades i disposicions són:

n	$T^1(n)$	$T^2(n)$	$T^3(n)$	$\bar{T}(n)$
200	0.02193 ms	0.02856 ms	0.01810 ms	0.023 ms
2000	0.15844 ms	0.00791 ms	0.14378 ms	0.103 ms
20000	3.98047 ms	3.55719 ms	0.83136 ms	2.790 ms
200000	13.00442 ms	19.90661 ms	6.68528 ms	13.199 ms
2000000	129.18255 ms	224.19447 ms	353.44418 ms	235.607 ms

- Conclusions:
  - A més mida, més temps. Quina relació deu haver-hi?
  - La disposició de les dades té una gran importància.

El factor disposició

- Com la cerca és lineal, el factor disposció determina si el natural que busquem és al principi o al final de la llista.
- Per eliminar el factor disposició ens podem fixar en:
  - El cas millor. En general és poc informatiu.
  - El cas mitjà. És dificil de calcular.
  - El cas pitjor. Acostuma a ser més pessimista del compte.
- Eliminem el factor disposició fent 1000 cerques diferents sobre les dades i calculant la mitja:

n	$T^{cl}(n)$
200	0.026014 ms
2000	0.235489 ms
20000	2.338265 ms
200000	23.478292 ms
2000000	235.846203 ms

- La relació és clarament lineal i, aproximadament: $T(n) \approx \frac{1}{1 \cdot 10^5} n$

Modifiquem el computador

- En les mateixes disposicions d'abans, usem un computador més ràpid.
- Els resultats per diverses mides de dades i disposicions són:

n	$T^{cr}(n)$	$T^{cl}(n)$
200	0.010333 ms	0.026014 ms
2000	0.095085 ms	0.235489 ms
20000	0.983780 ms	2.338265 ms
200000	9.734289 ms	23.478292 ms
2000000	97.380424 ms	235.846203 ms

- En el cas de la màquina ràpida, la relació continua essent lineal, tot i que ara el factor és menor: $T^r(n) \approx \frac{1}{2 \cdot 10^4} n$

Modifiquem l'algoritme

- Els veritables guanys venen quan s'obtenen algoritmes inherentment més ràpids.
- En les mateixes disposicions d'abans, canviem d'algoritme sobre el computador lent i ho comparem amb les dades anteriors.
- Els resultats per diverses mides de dades i disposicions són:

n	$T^{cr}(n)$	$T^{cl}(n)$	$T^{arcl}(n)$
200	0.010333 ms	0.026014 ms	0.005303 ms
2000	0.095085 ms	0.235489 ms	0.005356 ms
20000	0.983780 ms	2.338265 ms	0.005618 ms
200000	9.734289 ms	23.478292 ms	0.005628 ms
2000000	97.380424 ms	235.846203 ms	0.005726 ms

- Increïble!! Amb un algoritme diferent aconseguim temps, en el computador lent, aproximadament 10000 vegades menors que abans en el computador ràpid.
- La relació no sembla lineal. Aparentment és gairebé constant: el temps no depèn de la mida de les dades.

Complexitat temporal en cas pitjor

- Analitzar el cost en temps dels algoritmes és fonamental per poder-los comparar.
- Cal una forma homogènia de fer-ho.
- La forma més habitual és calcula el cost en cas pitjor. És a dir quan triga l'algoritme a calcular assumint la pitjor disposició possible de les dades.
- El resultat és una funció $T(n)$ on n és la mida de les dades.
- Això planteja un problema: com comparem dues funcions? Per exemple:

$$\begin{aligned} T_1(n) &= 2.3n^2 - 2n + 6 \\ T_2(n) &= 40n + 36 \end{aligned}$$

- Solució: ordre assimptòtic d'una funció.

Ordre assimptòtic d'una funció

Definition

Siguin f i g dues funcions definides $f, g : \mathbb{R} \to \mathbb{R}$. Direm que $f(x) \in O(g(x))$ si i solament si $\exists k, x_o \in \mathbb{R} : k > 0$ tals que $\forall x > x_0 : |f(x)| \leq k|g(x)|$.

- Per tota funció polinòmica $f(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_0$ es cumpleix que $f(x) \in O(x^n)$.
- Usant el concepte d'ordre assimptòtic podem classificar les funcions de cost en la següent jerarquia de classes:

$$\begin{aligned} O(1) \quad &\subset \\ &\subset \quad O(\log n) \subset O(n) \subset O(n \log n) \subset \\ &\subset \quad O(n^2) \subset O(n^3) \subset \cdots \subset O(n^k) \subset \\ &\subset \quad O(2^n) \subset O(n!) \end{aligned}$$

- Sempre classifiquem en la classe més «petita» possible.

Ordre assimptòtic d'una funció (II)

Algunes propietats interessants:

- Si $f_1 \in O(g_1)$ i $f_2 \in O(g_2)$ llavors $f_1 f_2 \in O(g_1 g_2)$.
- $f \cdot O(g) \subset O(fg)$.
- Si $f_1 \in O(g_1)$ i $f_2 \in O(g_2)$ llavors $f_1 + f_2 \in O(|g_1| + |g_2|)$.
- Particularment, si $f_1 \in O(g)$ i $f_2 \in O(g)$ llavors $f_1 + f_2 \in O(g)$
- Si $k \in \mathbb{R}$ i $k \neq 0$, llavors $O(kf) = O(f)$.
- Si $k \in \mathbb{R}$ i $f \in O(g)$, llavors $kf \in O(g)$.

El cas de la cerca lineal

```
# l = list()
trobat = False
for e in l:
    if e == d :
        trobat = True
        break
```

Assumim que:

- n longitud llista.
- t_c temps de la comparació.
- t_i temps d'una iteració del **for**.
- t_a temps d'una assignació.

Llavors, en cas pitjor:

$$\begin{aligned} T(n) &= t_a + n(t_i + t_c) + t_a \\ &= (t_i + t_c)n + 2t_a \end{aligned}$$

Per tant,

$$T(n) \in O(n)$$

El cas de la cerca binària

```
def bsearch(x, l, low, high):
    if low > high:
        return -1
    else:
        mid = (low + high) / 2
        item = l[mid]
        if item == x:
            return mid
        elif x < item:
            return bsearch(x, nums, low, mid-1)
        else:
            return bsearch(x, nums, mid+1, high)
```

Assumim que:

- n longitud llista.
- t_c temps de la comparació.
- t_a temps d'una assignació.

Llavors per $n > 0$, en cas pitjor:

$$\begin{aligned} T(n) &= 2t_a + 2t_c + T(n/2) \\ &= T(n/2) + c_0 \end{aligned}$$

Per tant:

$$T(n) = \begin{cases} c_1 & n = 0 \\ T(n/2) + c_0 & n > 0 \end{cases}$$

Això és una equació de recurrència
...i la solució és:

$$T(n) = \log_2 n$$

Resoldre l'equació de recurrència

- Les eqüacions de recurrència són complicades de resoldre.
- Una estratègia corrent passa per intuïr una solució i comprovar que ho sigui.
- Provem, en aquest cas, si $T(n) = \log_2 n$.

  Primer eliminem la constant. A tal efecte plantejem dues equacions pels valors de n i de $\frac{n}{2}$:

$$\begin{aligned} T(n) &= T(n/2) + c_0 \\ T(n/2) &= T(n/4) + c_0 \end{aligned}$$

  Si ara restem equacions tenim:

$$T(n) = 2T(n/2) - T(n/4)$$

  Ara subtituim $T(n) = \log_2 n$ i vegem que passa:

$$\begin{aligned} T(n) &= 2T(n/2) - T(n/4) \\ &= 2\log_2 \frac{n}{2} - \log_2 \frac{n}{4} \\ &= 2(\log_2 n - \log_2 2) - (\log_2 n - \log_2 4) \\ &= 2\log_2 n - 2 - \log_2 n + 2 \\ &= \log_2 n \end{aligned}$$

La dictadura de l'ordre

- Finalment, què implica que el cost d'un algoritme tingui un o altre ordre?
- Fixem-nos en aquesta taula de temps i analitzem-la:

$T(n)$	Màx. mida 10^3 s	Màx. mida 10^4 s	Ratio
$100 \log_2 n$	1024	1024^{10}	1024^9
$100n$	10	100	10.0
$5n^2$	14	45	3.2
$n^3/2$	12	27	2.3
2^n	10	13	1.3

- Els algoritmes, segons l'ordre...
  - Els algoritmes $O(\log_2 n)$ són extraordinaris.
  - Els algoritmes $O(n)$ són bons.
  - Els algoritmes $O(n^k)$ amb k petit, acceptables.
  - Els algoritmes $O(n^k)$ amb k gran, molt lents.
  - Els algoritmes $O(2^n)$, intractables.

Feina per aquesta setmana

1. Estudi de la teoria. A partir de les transparències i les notes que heu pres.
2. Incrementar el xuletari i el glossari del tema.
3. Solució dels problemes del tema.