Limite asintotico inferiore

Lezione 2

Notazione Ω (omega grande) (limite asintotico inferiore)

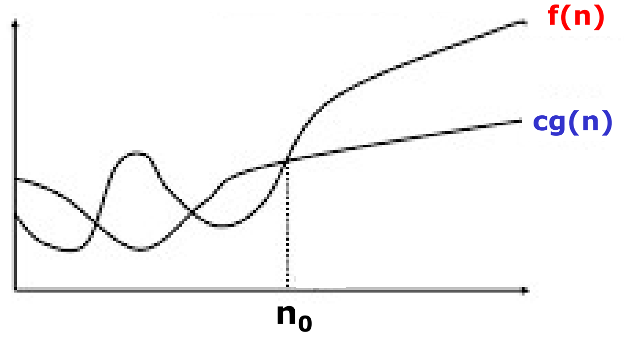
f(n) è Ω (g(n)) se esistono

un intero n_0 ed una costante $c>0$ tali che

per ogni $n \geq n_0$: $f(n) >= c\, g(n)$

Notazione Ω grande

f(n) è Ω(g(n))



Esempi

- $2n^2 + 3n + 2$ è $\Omega(n)$ n0=1, c=1
- $2n^2 + 3n + 2$ è $\Omega(n^2)$ n0=1, c=1
- n^2 è $\Omega(2n^2)$ n0=1, c=1/2
- n^2 è $\Omega(100n)$ n0=101, c=1 (oppure n0=1, c=1/100)

Notazione Θ (theta grande) (limite asintotico stretto)

f(n) è Θ(g(n)) se

f(n) è O(g(n)) e f(n) è Ω(g(n))

in altre parole

f(n) è Θ (g(n)) se esistono

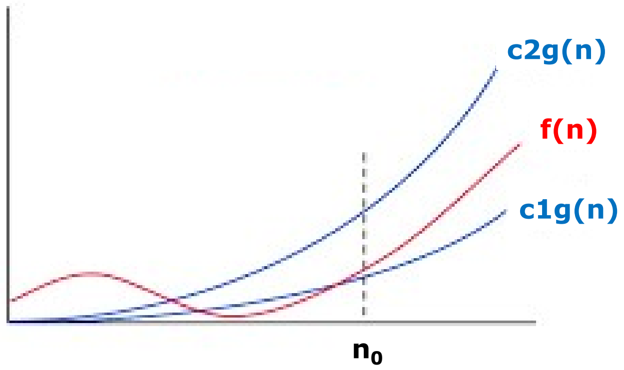
un intero n_0 e due costante c1, c2>0 tali che

per ogni $n \geq n_0$:

c1 g(n) <= f(n) <= c2 g(n)

Notazione Θ

f(n) è Θ (g(n))



Anche per Ω e Θ valgono le regole dei fattori costanti, del prodotto e della somma

Esempi

- $2n^2 + 3n + 2$ è $\Omega(n)$ ma non è $O(n)$, quindi non è $\Theta(n)$

- $2n^2 + 3n + 2$ è $O(n^2)$ [n0=1 , c=7]
- $2n^2 + 3n + 2$ è $\Omega(n^2)$ [n0=1 , c=1]
- $2n^2 + 3n + 2$ è $\Theta(n^2)$

- **un polinomio di grado m è $\Theta(n^m)$**

alcuni risultati generali

1. f(n) è $O(g(n))$ se e solo se g(n) è $\Omega(f(n))$
2. se f(n) è $\Theta(g(n))$ allora g(n) è $\Theta(f(n))$
3. f(n) è $\Theta(g(n))$ quando f e g hanno lo stesso ordine di complessità

Complessità dei programmi iterativi

Programmi iterativi

C: costrutti del linguaggio -> Classi di complessità

V: costante, I: variabile, E: espressione, C: comando

C [V] = C [I] = O(1)

C [E1 op E2] = C [E1] + C [E2]

C [I[E]] = C [E] variabile indicizzata

C [I=E;] = C [E]

C [I[E1] =E2;] = C [E1] + C [E2]

Programmi iterativi

C [return E;] = C [E]

C [if (E) C]] = C [E] + C [C]

C [if (E) C1 else C2] =

C [E] + C [C1] + C [C2]

Programmi iterativi

C [for (E1; E2; E3) C] =

C [E1] + C [E2] +

(C [C] + C [E2] + C [E3]) O(g(n))

g(n): numero di iterazioni

C [while (E) C] =

C [E] + (C [C] + C [E]) O(g(n))

C [{ C1 … Cn }] = C [C1] + … +C [Cn]

Selection sort

```
void exchange( int& x, int& y) {
O(1)  int temp = x;
O(1)  x = y;
O(1)  y = temp;
 }

void selectionSort(int A[ ], int n) {
 O(n2)   for (int i=0; i< n-1; i++) {
  O(1)   int min= i;
  O(n)   for (int j=i+1; j< n; j++)
  O(1)       if (A[ j ] < A[min]) min=j;
  O(1)   exchange(A[i], A[min]);
  }
 }
```

$O(n^2)$

Bubblesort

$O(n^2)$

```
void bubbleSort(int A[], int n) {
O(n2) for (int i=0; i < n-1; i++)
O(n)       for (int j=n-1; j >= i+1; j--)
O(1)          if (A[j] < A[j-1]) exchange(A[j], A[j-1]);
}
```

numero di scambi = $O(n^2)$

con selectionSort numero di scambi = $O(n)$

Esempio (I)

```
int f (int x){
      return  x;
}
```

risultato: O(n)

complessità: O(1)

```
int h (int x){
      return  x*x;
}
```

risultato: $O(n^2)$

complessità: O(1)

```
int k (int x) {
   int a=0;
   for (int i=1; i<=x; i++)
      a++;
   return a;
}
```

risultato: O(n)

complessità: O(n)

Esempio (II)

```
void g (int n){     // n>=0
   for (int i=1; i<= f(n); i++)
      cout << f(n);
}
```

complessità: O(n)

```
void g (int n){
   for (int i=1; i<= h(n); i++)
      cout << h(n);
}
```

complessità: $O(n^2)$

```
void g (int n){
   for (int i=1; i<= k(n); i++)
      cout << k(n);
}
```

complessità: $O(n^2)$

Esempio (III)

```
void p (int n){
    int b=f(n);
    for (int i=1; i<=b; i++)
        cout << b;
}
```

complessità: O(n)

```
void p (int n){
    int b=h(n);
    for (int i=1; i<=b; i++)
        cout << b;
}
```

complessità: O(n^2)

```
void p (int n){
    int b=k(n);
    for (int i=1; i<=b; i++)
        cout << b;
}
```

complessità: O(n)

Moltiplicazione fra matrici

```
void matrixMult  (int A[N] [N], int B[N] [N], int C [N] [N]) {

 O(n³) for (int i=0; i < N; i++)
 O(n²)      for (int j=0; j < N; j++) {
 O(1)            C[ i ] [ j ]=0;
 O(n)            for (int k=0; k < N; k++)
 O(1)                C[ i ] [ j ]+=A[ i ] [ k ] * B[ k ] [ j ];
          }
 }
```

O(n^3)

Ricerca lineare e div2

```
int linearSearch (int A [], int n, int x) {
  int b=0;
  for (int i=0; !b & (i<n);  i++)
       if (A[ i ] == x) b=1;
  return b;
}
```

O(n)

```
int div2(int n) {
   int i=0;
   while (n > 1) {
     n=n/2;
     i++;
   }
   return i;
 }
```

O(logn)

Esercizio 5.a

a) Date le seguenti dichiarazioni di funzione, calcolare la complessità in funzione di n>0 della chiamata P(F(n),y).

```
int F(int n) {int b; int a=0;
    for (int i=1; i<=n; i++)
         for (int j=1; j<=n;j++) a++;
    b= a; return 2*b;}   // *


void P (int m, int &x) { for (int i=1; i<=m*m; i++) x+=3;}
```

C[F(n)] = $O(n^2)$,   Risultato[F(n)] $\in O(n^2)$,
C[P(m,x)] = $O(m^2)$

C[P(F(n),y)] = C[F(n)]+ C[P(n^2,y)] = $O(n^2) + O(n^4)=O(n^4)$

Esercizio 5.b

Date le seguenti dichiarazioni di funzione, calcolare la complessità in funzione di n>0 della chiamata P(F(n),y).

```
int F(int n) {int b; int a=0;
     for (int i=1; i<=n; i++)
            for (int j=1; j<=n;j++) a++;
      b= a/n; return 2*b;}   // *

void P (int m, int &x) { for (int i=1; i<=m*m; i++) x+=3;}
```

$C[F(n)] = O(n^2)$, Risultato$[F(n)] \in O(n)$
$C[P(m,x)] = O(m^2)$

$C[\,P(F(n),y)] = C[F(n)] + C[P(n,y)] = O(n^2) + O(n^2) = O(n^2)$

Complessità dei programmi ricorsivi

Programmi ricorsivi : definizioni iterative e induttive

Fattoriale di un numero naturale : n!

0!=1
n! = 1 x 2 x … n se n>0 **definizione iterativa**

0!=1
n!=n*(n-1)! se n>0 **definizione induttiva (o ricorsiva)**

Fattoriale: algoritmo iterativo

0!=1
n! = 1 x 2 x … n

```
int fact(int n) {
  if (n == 0) return 1;
  int a=1;
  for (int i=1; i<=n; i++) a=a*i;
  return a;
}
```

Fattoriale: algoritmo ricorsivo

0!=1
n!=n*(n-1)! se n>0

```
int fact(int x) {
  if (x == 0) return 1;
  else return x*fact(x-1);
}
```


Algoritmo ricorsivo per la moltiplicazione

mult (0, y) = 0
mult (n,y)= y + mult (n-1,y)   se n>0

```
int mult(int x, int y) {
   if (x == 0) return 0;
   return y + mult(x-1,y);
}
```

Algoritmi ricorsivi per riconoscere i numeri pari e calcolare il massimo comun divisore

```
int pari(int x) {
   if (x == 0) return 1;
   if (x == 1) return 0;
   return pari(x-2);
}
```

Algoritmo di Euclide

```
int MCD (int x, int y) {
   if (x == y)  return x;
   if (x < y) return MCD (x, y-x);
   return MCD (x-y, y);
}
```


Scrivere un algoritmo ricorsivo corretto

Regola 1: casi base in cui la funzione è definita immediatamente

Regola 2: ad ogni chiamata ricorsiva l'insieme dei dati è più piccolo

Regola 3: ogni sequenza di chiamate porta ad uno dei casi base

Applicazioni sbagliate

```
int pari_errata(int x) {
   if (x == 0) return 1;
   return pari_errata(x-2);
}
```

```
int MCD _errata(int x, int y) {
   if (x == y)  return x;
   if (x < y) return MCD _errata(x, y-x);
   return MCD _errata(x, y);
}
```


Induzione naturale

Sia P una proprietà sui naturali.

Base. P vale per 0

Passo induttivo. Per ogni naturale n è vero che:

se P vale per n allora P vale per (n+1)

P vale per tutti i naturali

Somma dei primi n numeri naturali

Dimostrare con il principio di induzione naturale che la somma dei primi n numeri è n(n+1)/2

$$\Sigma\_0..n = n(n+1)/2$$

Base: $\Sigma\_0..0$ = (0*1)/2= 0

Passo induttivo:

Ip: $\Sigma\_0..n$ = n(n+1)/2

Tesi: $\Sigma\_0..n+1$ = [(n+1)(n+2)]/2

Dim:

$\Sigma\_0..n+1 = \Sigma\_0..n + (n+1)$ def

= n(n+1)/2 + (n+1) ip

=[n(n+1)+2(n+1)]/2

= [(n+1)(n+2)]/2

Induzione completa

Sia P una proprietà sui naturali.

Base. P vale per 0

Passo induttivo. Per ogni naturale n è vero che:

se P vale per ogni m <= n allora P vale per (n+1)

P vale per tutti i naturali

Induzione ben fondata

Sia P una proprietà di un insieme ordinato S

Base. P vale per i minimali di S

Passo induttivo. Per ogni elemento E di S è vero che:

se P vale per ogni elemento minore di E allora P vale per E

P vale per S

Calcolo della complessità dei programmi ricorsivi

```
int fact(int x) {
  if (x == 0) return 1;
  else return x*fact(x-1);
}
```

T (0) = a
T (n) = b + T(n-1)

Relazione di ricorrenza

soluzione

T (0) = a
T (n) = b + T(n-1)

T (0) = a

T (1) = b + a

T (2) = b + b+ a = 2 b + a

T (3) = b+2 b + a =3 b + a

.

.

T (n) = n b + a

T(n) è O(n)

selection sort ricorsiva

```
void r_selectionSort (int* A, int m, int i=0) {
        if (i == m -1)  return;
        int min= i;
        for (int j=i+1; j <m; j++)
                if (A[j] < A[min]) min=j;
        exchange(A[i],A[min]);
         r_selectionSort (A, m, i+1)
    }
}
```

T (1) = a

T (n) = bn + T(n-1)

soluzione

T (1) = a
T (n) = bn + T(n-1)

T (1) = a

T (2) = 2b + a

T (3) = 3b + 2b + a

.

.

T (n) = (n + n-1 + n-2 + … 2) b + a
= (n(n+1)/2-1)b+ a

T(n) è $O(n^2)$