COMPLESSITÀ COMPUTAZIONALE DEGLI ALGORITMI

Fondamenti di Informatica a.a.2006/07
Prof. V.L. Plantamura
Dott.ssa A. Angelini

Esempio 10

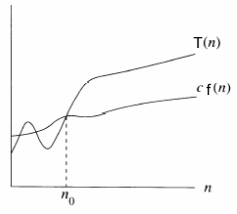
```
int potenza(int base, int esp);
        main () {               \* Genera le prime <n> potenze *\
            int b, n, espo;     \* di base <b> *\
            printf (“Scrivi la base”);
            scanf(“%d”, &b);
            printf (“Quante potenze?”);
            scanf(“%d”, &n);
            for (espo=1; espo <= n; espo++)
                    printf (“%d”, potenza(b,espo));
        } \* main *\
int potenza(int base, int esp) {
        int i,ris;
        ris=1;
        for (i=1; i <= esp; i++)
            ris = ris*base;
        return ris; }
```

Ω *(grande omega)*

- Diciamo che $T(n) = \Omega(f(n))$, *- leggiamo “T(n) ha complessità grande omega di f(n)” -* se esistono due costanti positive c ed n_0, tali che per ogni $n > n_0$ risulti $T(n) \geq cf(n)$
- Ovvero, a partire da una certa dimensione n_0 del dato di ingresso, la funzione $f(n)$ minora la funzione $T(n)$. Possiamo quindi anche dire che la $f(n)$ rappresenta un limite inferiore per la $T(n)$.

Ω (grande omega)

- $T(n) = \Omega(f(n))$



Esempio - Dimostrazione

- Sia $T(n) = 7n^2 + 6$,

  Sia $n^2 + (6/7) = T(n)/7$

  Si ha $n^2 + (6/7) \geq n^2$ per n ≥ 1, per cui

  Per cui $T(n) \geq 7\ n^2$; $T(n) = \Omega(n^2)$

e le costanti utilizzate sono quindi: c=7 e n_0=1

Ω (grande omega)

- Anche in questo caso il limite non è "stretto", e valgono sostanzialmente tutte le considerazioni fatte per la notazione O(n);
- È anche vero infatti che:
  $T(n) = \Omega(n \log n)$
  $T(n) = \Omega(n)$
  $T(n) = \Omega(1)$

Osservazione

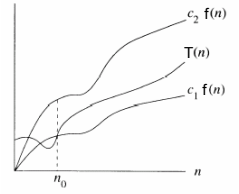
- Se una funzione non è nota ma sono note più delimitazioni asintotiche inferiori, va preferita la più grande tra esse.
- Ad esempio se sapessimo che:
  $T(n) = \Omega(n^2 \log n)$
  $T(n) = \Omega(n (\log n)^2)$
  $T(n) = \Omega(1)$
  sarebbe opportuno asserire che $T(n) = \Omega(n^2 \log n)$

Θ (grande theta)

- Diciamo che $T(n) = \Theta(f(n))$, - *leggiamo "T(n) ha complessità grande theta di f(n)"* - se esistono tre costanti positive *c, d* ed n_0, tali che per ogni $n > n_0$ risulti $cf(n) < T(n) < df(n)$
- Ovvero quando una funzione $T(n)$ è contemporaneamente $O(f(n))$ e $\Omega(f(n))$, allora diciamo che $f(n)$ rappresenta una delimitazione asintotica stretta per la $T(n)$.

Θ (grande theta)

- $T(n) = \Theta(f(n))$



Θ (grande theta)

- In altre parole un algoritmo ha complessità computazionale $\Theta(f(n))$ se la funzione che calcolo per l'algoritmo è $T(n)$ e si ha che:

$$\lim_{n\to\infty}\frac{f(n)}{T(n)} = k$$

dove k è una costante positiva.

Θ (grande theta)

- Nell'esempio di $T(n) = 2n^2 + 3n + 1$:

$$\lim_{n\to\infty}\frac{f(n)}{T(n)} = \frac{n^2}{2n^2+3n+1} = \frac{1}{2}$$

Per cui $T(n) = \Theta(n^2)$

Regola pratica

- Per tutte le funzioni polinomiali e poli-logaritmiche, cioè della forma generale: $T(n) = \sum c_i n^t (\log n)^k$
- La delimitazione inferiore e quella superiore coincidono, e sono rispettivamente: $O(n^h(\log n)^z)$ e $\Omega(n^h(\log n)^z)$

Dove h è il più grande esponente tra le t che compaiono nella somma, e z il più piccolo tra gli esponenti di *logn*.

Algebra degli O grandi

- Abbiamo visto esempi di calcolo della complessità in numero di passi base per programmi a blocchi.
- Definiamo ora, tramite l'algebra degli O grandi, un criterio per il calcolo della complessità asintotica di un programma strutturato.
- In un programma a blocchi si possono presentare le due seguenti situazioni:

Algebra degli O grandi

a) Blocchi in sequenza:

```
i=1;
while (i<=n) {
      stampastelle(i);
       i=i+1; }
for (i=1; i <= 2n; i++) {
      scanf("%d", &num);
}
```

Sia $g_1(n)$ la complessità del primo blocco e $g_2(n)$ la complessità del secondo. La complessità globale è:

$O(g_1(n)+g_2(n)) = O(max\{g_1(n),g_2(n)\})$

Algebra degli O grandi

b) Blocchi annidati:

```
for (i=1; i <= n; i++) {
      scanf("%d", &j); printf(("%d", j*j);
      do { scanf("%d", &num);
            j = j+1; } while (j<=n)
}
```

Sia $g_1(n)$ la complessità del blocco esterno e $g_2(n)$ la complessità di quello interno. La complessità globale è:

$O(g_1(n) * g_2(n)) = O(g_1(n))* O(g_2(n))$

La complessità di un blocco costituito da più blocchi annidati è data dal prodotto delle complessità dei blocchi componenti.

Esempio…

```
# define MAXN 10000; /* massimo numero di elementi */
int Int_Array[MAXN] /* dichiarazione array */
void CaricaArray(int L); /* Carica L valori in array (L<=MAXN deciso
                            dall'utente) */
int RicercaLineare(int L, int x);
/* restituisce la posizione di un elemento in un array
   Int_Array; limita la ricerca alle prime L posizioni;
   se l'elemento è presente restituisce la posizione, altrimenti
   restituisce -1 */
main () {
    int L, elemento, pos; /* richiesta delle dimensioni reali, L di Int_Array */
    do {  printf("\n Numero elementi: ");
          scanf("%d", &L); }
    while ((L >= MAXN) || (L < 0))
    CaricaArray(L);
    printf("\n elemento da cercare: ");
    scanf("%d", &elemento);
    pos = RicercaLineare (L, elemento);
```

…Esempio

```
    if (pos >= 0)
          printf("L'elemento cercato si trova in posizione", "%d",
    pos)
    else
          printf ("L'elemento cercato non è presente nell'array")
}
void CaricaArray(int Nelem) {
    int i;
    for (i=0; i <= Nelem; i++) {
          printf("Inserisci elemento %d:", i);
          scanf("%d", &Int_Array[i]);} };
int RicercaLineare(int Nelem, int x) {
    int k;
    k =0;
    while ((k <= Nelem) && (Int_Array[k] !=x)) ++k;
    if (k <= Nelem) return k
    else return -1; }
```

Calcolo Complessità

- Carica Array: $O(n) *[O(1)+O(1)] = O(n) * [O(max\{1,1\})] = O(n)*O(1) = O(n*1) = O(n)$
- Ricerca lineare (caso pessimo): $O(1)+ O(n)*O(1) + O(1)*O(1)=… \rightarrow O(n)$
- Ciclo do: $O(1)+ O(1)+ O(1)=… \rightarrow O(1)$
- *Main:* $O(1)+ O(n)+ O(1)+ O(1)+ O(1)+ O(n) + O(1)*O(1) = … \rightarrow O(n)$