

Bound asintotici per alcune funzioni di uso comune

Logaritmi

- $\log n = O(n)$.

Dim. Dimostriamo per induzione che $\log_2 n \leq n$ per ogni $n \geq 1$.

Base dell'induzione: Vero per $n=1$ in quanto $\log_2 1 = 0 < 1$

Passo Induttivo: Supponiamo $\log_2 n \leq n$ vera per $n \geq 1$.

Dimostriamo che è vera per $n+1$.

$$1. \quad \log_2(n+1) \leq \log_2(2n) = \log_2 2 + \log_2 n = 1 + \log_2 n$$

Per ipotesi induttiva $\log_2 n \leq n$ e quindi

$$2. \quad 1 + \log_2 n \leq 1 + n.$$

Dalla catena di disuguaglianze 1. e dalla disuguaglianza 2. si ha

$$\log_2(n+1) \leq n+1.$$

Un utile richiamo

Parte intera inferiore:

La parte intera inferiore di un numero x è denotata con $\lfloor x \rfloor$ ed è definita come quell'unico intero per cui vale che $x-1 < \lfloor x \rfloor \leq x$. In altre parole, $\lfloor x \rfloor$ è il più grande intero minore o uguale di x (infatti dalla definizione si ha $\lfloor x \rfloor \leq x$ ma $\lfloor x \rfloor + 1 > x$)

Esempio: $\lfloor 4.3 \rfloor = 4$, $\lfloor 6.9 \rfloor = 6$, $\lfloor 3 \rfloor = 3$

Proprietà 1: L'intero più piccolo strettamente maggiore di x è $\lfloor x \rfloor + 1$.

Dim. Siccome $\lfloor x \rfloor$ è il più grande intero minore o uguale di x allora qualsiasi intero più grande di $\lfloor x \rfloor$ è maggiore di x . La proprietà discende allora dal fatto che il più piccolo degli interi maggiori di $\lfloor x \rfloor$ è $\lfloor x \rfloor + 1$.

Proprietà 2: $\lfloor \lfloor a/b \rfloor / c \rfloor = \lfloor a / (bc) \rfloor$, per a, b e c interi con b e c maggiori o uguali di 1

Un utile richiamo

Parte intera superiore:

La parte intera superiore di un numero x è denotata con $\lceil x \rceil$ ed è definita come quell'unico intero per cui vale che $x \leq \lceil x \rceil < x+1$. In altre parole, $\lceil x \rceil$ è il più piccolo intero maggiore o uguale di x (infatti dalla definizione si ha $\lceil x \rceil \geq x$ ma $\lceil x \rceil - 1 < x$).

Esempio: $\lceil 4.3 \rceil = 5$, $\lceil 6.9 \rceil = 7$, $\lceil 3 \rceil = 3$

Proprietà 3: L'intero più grande strettamente minore di x è $\lceil x \rceil - 1$.

Dim. Siccome $\lceil x \rceil$ è il più piccolo intero maggiore o uguale di x allora qualsiasi intero più piccolo di $\lceil x \rceil$ è minore di x . La proprietà discende allora dal fatto che il più grande intero più piccolo di $\lceil x \rceil$ è $\lceil x \rceil - 1$.

Proprietà 4: $\lceil \lceil a/b \rceil / c \rceil = \lceil a/(bc) \rceil$ per a, b e c interi con b e c diversi da 0

Tempo logaritmico

Tipicamente si ha quando ogni passo riduce di un fattore costante il numero di passi che restano da fare

Per esercizio dimostriamo che il seguente for richiede tempo $\Theta(\log n)$

```
For (i=n; i ≥ 1; i=i/2)
  print(i)
```

Dimostrazione: Il for termina quando i diventa minore di 1.

Ad ogni iterazione il valore di i è minore o uguale della metà del valore che aveva in precedenza → dopo la k -esima iterazione $i = \lfloor \dots \lfloor \lfloor n/2 \rfloor / 2 \rfloor \dots / 2 \rfloor = \lfloor n/2^k \rfloor$ per la proprietà 2 della parte intera inferiore

k divisioni

Per sapere dopo quante iterazioni termina il for dobbiamo trovare il più piccolo k per cui $\lfloor n/2^k \rfloor < 1$. Vediamo per quali k è soddisfatta la disuguaglianza $\lfloor n/2^k \rfloor < 1$

$$\lfloor n/2^k \rfloor < 1 \iff n/2^k < 1 \iff 2^k > n \iff k > \log_2 n \quad (1)$$

Noi vogliamo il piccolo intero k per cui vale la (1), cioè il più piccolo intero strettamente maggiore di $\log_2 n$. Per la proprietà 1 della parte intera inferiore si ha $k = \lfloor \log_2 n \rfloor + 1$

N.B. Se invece di venire diviso per 2, il valore di i viene diviso per una generica costante $c > 1$ allora la base del log è c ma ai fini della valutazione asintotica non cambia niente.

Tempo logaritmico

```
For (i=1; i<= n; i=i*2)
  print(i)
```

Il for in alto richiede tempo $\Theta(\log n)$

Dimostrazione : Il for termina quando i diventa maggiore di n .

- Ad ogni iterazione il valore di i raddoppia \rightarrow dopo la k -esima iterazione $i = 2^k$.
- Per sapere dopo quante iterazioni termina il for dobbiamo trovare il **più piccolo k per cui $2^k > n$** . In altre parole vogliamo k tale che $2^k > n$
- La disequazione $2^k > n$ se e solo se $k > \log_2 n$
- Tra tutti gli interi $k > \log_2 n$, noi vogliamo quello più piccolo. Per la proprietà 1 della parte intera inferiore si ha $k = \lfloor \log_2 n \rfloor + 1$
- Dopo esattamente $k = \lfloor \log_2 n \rfloor + 1$ iterazioni 2^k diventa più grande di n e il for termina.
- \rightarrow Numero iterazioni è $\lfloor \log_2 n \rfloor + 1 = \Theta(\log n)$

N.B. Se invece di raddoppiare, il valore di i viene moltiplicato per una generica costante $c > 1$ allora la base del log è c ma ai fini della valutazione asintotica non cambia niente.

51

Tempo logaritmico: $O(\log n)$

Tipicamente si ha quando ogni passo riduce di un fattore costante il numero di passi che restano da fare

Ricerca binaria. Dato un array A ordinato di n numeri ed un numero x vogliamo determinare se x è in A

```
binarySearch(A, n, x)
  l = 0;
  r = n-1
  while l <= r
    c = (l+r)/ 2 //assumiamo troncamento
    if x = A[c]
      return true
    if x < A[c]
      r = c-1
    else l=c+1 //caso x>A[c]
  return false
```

Se la dimensione $r-l+1$ dell'intervallo $[l, r]$ è pari allora il sottointervallo di destra $[c+1, r]$ ha un elemento in più rispetto a quello di sinistra. In caso contrario i due sottointervalli hanno la stessa dimensione.

Caso $r-l+1$ pari: intervallo di sinistra ha $(r-l+1)/2 - 1 = \lfloor (r-l+1)/2 \rfloor - 1$ elementi e quello di destra $(r-l+1)/2 = \lfloor (r-l+1)/2 \rfloor$.

Caso $r-l+1$ dispari: entrambi gli intervalli hanno $\lfloor (r-l+1)/2 \rfloor$ elementi

52

Tempo logaritmico: $O(\log n)$

Analisi ricerca binaria.

Il while termina quando $l > r$, cioè quando il range $[l, r]$ vuoto.

- Inizialmente $[l, r] = [0, n-1]$ e quindi contiene n elementi
- Dopo la prima iterazione, $[l, r]$ contiene al più $\lfloor n/2 \rfloor$ elementi
- Dopo la seconda iterazione, $[l, r]$ contiene al più $\lfloor \lfloor n/2 \rfloor / 2 \rfloor = \lfloor n/4 \rfloor$ elementi
- Dopo la terza iterazione, $[l, r]$ contiene al più $\lfloor \lfloor \lfloor n/4 \rfloor / 2 \rfloor \rfloor = \lfloor n/8 \rfloor$ elementi
- ...
- Dopo la k -esima iterazione, $[l, r]$ contiene al più $\lfloor n/2^k \rfloor$ elementi
- Per sapere quando termina il ciclo di while dobbiamo trovare il più piccolo intero k per cui $\lfloor n/2^k \rfloor < 1$
- Abbiamo già dimostrato che questo k è $\Theta(\log n)$
- NB: per sbarazzarci delle parti intere inferiori annidate abbiamo usato la proprietà 2.

53

Espressione O	nome
$O(1)$	costante
$O(\log \log n)$	log log
$O(\log n)$	logaritmico
$O(\sqrt[c]{n}), c > 1$	sublineare
$O(n)$	lineare
$O(n \log n)$	$n \log n$
$O(n^2)$	quadratico
$O(n^3)$	cubico
$O(n^k) (k \geq 1)$	polinomiale
$O(a^n) (a > 1)$	esponenziale

54

Tempo $O(\sqrt{n})$

```

j=0;
i=0;
while(i<=n){
  j++;
  i=i+j;
}

```

Analisi :

Il while termina quando i diventa maggiore di n.

All'iterazione k al valore di i viene sommato $j=k$ per cui dopo aver iterato il while k volte il valore di i è $(1+2+3+\dots+k) = k(k+1)/2$.

Affinche' si interrompa il while e' sufficiente che $k(k+1)/2 > n$

Per semplicita' osserviamo che $k^2/2 \leq k(k+1)/2$ per cui se $k^2/2 > n$ allora $k(k+1)/2 > n$.

Risolviamo $k^2/2 > n$.

$$k^2/2 > n \iff k^2 > 2n \iff k > (2n)^{1/2} \text{ (le implicazioni utili sono } \leftarrow \text{)}$$

Dalla propriet  1, $\lfloor (2n)^{1/2} \rfloor + 1$ e' il pi  piccolo intero maggiore di $> (2n)^{1/2}$ per cui dopo al pi  $\lfloor (2n)^{1/2} \rfloor + 1 = O(\sqrt{n})$ iterazioni il while termina.

Progettazione di Algoritmi, a.a. 2021-22
A. De Bonis

55

55

Tempo $O(\sqrt{n})$

```

j=0;
i=0;
while(i<=n){
  j++;
  i=i+j;
}

```

Analisi :

Il while termina quando i diventa maggiore di n.

All'iterazione k al valore di i viene sommato $j=k$ per cui dopo aver iterato il while k volte il valore di i   $(1+2+3+\dots+k) = k(k+1)/2$.

Affinche' si interrompa il while e' sufficiente che $i=k(k+1)/2 > n$

Risolviamo la disequazione $k^2/2 + k/2 - n > 0$: le soluzioni di $k^2/2 + k/2 - n = 0$ sono

$k_1 = -1/2 - (1/4 + 4n/2)^{1/2}$ e $k_2 = -1/2 + (1/4 + 4n/2)^{1/2}$ e la disequazione   soddisfatta per $k < k_1$ e $k > k_2$. Siccome il nostro k   positivo puo' essere solo $k > k_2$.

Quindi dopo $k > -1/2 + (1/4 + 2n)^{1/2}$ iterazioni si ha $i = k(k+1)/2 > n$ e il while si interrompe.

Dalla propriet  1, il pi  piccolo intero strettamente maggiore di $-1/2 + (1/4 + 4n/2)^{1/2}$  

$\lfloor -1/2 + (1/4 + 2n)^{1/2} \rfloor + 1$. L'intero $\lfloor -1/2 + (1/4 + 2n)^{1/2} \rfloor + 1$   $\leq \lfloor (1/4 + 2n)^{1/2} \rfloor + 1 = \lfloor ((9/4)n)^{1/2} \rfloor + 1 \leq (\sqrt{9/4}) \sqrt{n} + 1$

ed   $\geq \lfloor -1 + (1/4 + 2n)^{1/2} \rfloor + 1 = \lfloor \sqrt{9/4} \rfloor \sqrt{n} \geq (\sqrt{9/4}) \sqrt{n} + 1$ per ogni n. Quindi dopo $\Theta(\sqrt{n})$ iterazioni il while termina.

Progettazione di Algoritmi, a.a. 2021-22
A. De Bonis

56

56

Logaritmi a confronto con polinomi e radici

Per ogni **costante** $x > 0$, $\log n = O(n^x)$. (N.B. x può essere < 1)

Dim.

Caso $x \geq 1$:

Se $x \geq 1$ si ha $n \leq n^x$ per ogni $n \geq 0 \rightarrow n = O(n^x)$.

Abbiamo già dimostrato che $\log n = O(n)$ per cui la proprietà transitiva $\rightarrow \log n = O(n^x)$

Caso $x < 1$:

Vogliamo trovare le costanti $c > 0$ e $n_0 \geq 0$ tale che $\log n \leq cn^x$ per ogni $n \geq n_0$

Siccome sappiamo che $\log_2 m < m$ per ogni $m \geq 1$ allora ponendo $m = n^x$ con $n \geq 1$, si ha $\log_2 n^x < n^x$ da cui $x \log_2 n < n^x$ e dividendo entrambi i membri per x si ha $\log_2 n < 1/x n^x$. Perchè la disequazione $\log_2 n \leq cn^x$ sia soddisfatta per ogni $n \geq n_0$, basta quindi prendere $c = 1/x$ ed $n_0 = 1$.

NB: abbiamo visto che nella notazione asintotica possiamo eliminare la base del log se questa è costante

Progettazione di Algoritmi, a.a. 2021-22
A. De Bonis

57

57

Potenze di logaritmi a confronto con polinomi e radici

Per ogni $x > 0$ e $b > 0$ **costanti**, $(\log n)^b = O(n^x)$.

Dim.

Vogliamo trovare le costanti $c > 0$ e $n_0 \geq 0$ tali che $(\log n)^b \leq cn^x$ per ogni $n \geq n_0$

Risolviamo la disequazione $(\log n)^b \leq cn^x$:

$(\log n)^b \leq cn^x \iff \log n \leq c^{1/b} n^{x/b}$ (\leftarrow vale perchè assumiamo $\log n > 0$)

Troviamo le costanti $c > 0$ ed $n_0 \geq 0$ tali che $\log n \leq c^{1/b} n^{x/b}$ per ogni $n \geq n_0$

Abbiamo già dimostrato nella slide precedente che $\log n = O(n^y)$ per ogni $y > 0$. Ciò vale anche se poniamo $y = x/b$. Quindi esistono due costanti $c' > 0$ e $n'_0 \geq 0$ tali che $\log n \leq c' n^{x/b}$ per ogni $n \geq n'_0$.

Di conseguenza basta imporre $c^{1/b} = c'$ ed $n_0 = n'_0$ da cui $c = (c')^b$ ed $n_0 = n'_0$

Progettazione di Algoritmi, a.a. 2021-22
A. De Bonis

58

58

Potenze di logaritmi a confronto con polinomi e radici

Dimostrare che per ogni $x > 0$, $a > 0$ e $b > 0$ costanti, $(\log n^a)^b = O(n^x)$.

La dimostrazione è molto semplice se si usa quanto visto nelle slide precedenti

Dim: Per quanto dimostrato nella slide precedente $(\log n)^b = O(n^x)$.

- Per la proprietà 3. dei logaritmi $\log n^a = a \log n$
 $\log n^a = a \log n \rightarrow (\log n^a)^b = a^b (\log n)^b$
- Per la prima regola della notazione asintotica (slide 42), moltiplicare una funzione per una costante non cambia la limitazione asintotica per cui
 $\log n^b = O(n^x) \rightarrow a^b (\log n)^b = O(n^x)$ da cui $(\log n^a)^b = O(n^x)$

Potenze di logaritmi a confronto con polinomi e radici

Per esercizio dimostriamo che per ogni $x > 0$ costante, $\log n$ non è $\Omega(n^x)$.

Per definizione di Ω : $\log n = \Omega(n^x) \rightarrow$ esistono due costanti $c > 0$ e $n_0 \geq 0$ tali che $\log n \geq cn^x$ per ogni $n \geq n_0$

risolviamo la disequazione $\log n \geq cn^x$
 $\log n \geq cn^x \leftrightarrow c \leq (\log n)/n^x$

quindi la costante c e la costante n_0 devono essere tali che $c \leq (\log n)/n^x$ per ogni $n \geq n_0$

il limite di $(\log n)/n^x$ al tendere di n all'infinito è 0 per cui comunque scegliamo piccola la costante c esisterà un n_c per cui $(\log n)/n^x < c$ per ogni $n \geq n_c$

quindi per qualsiasi costante c non è possibile trovare una costante n_0 per cui $\log n \geq cn^x$ per ogni $n \geq n_0$

Tempo $O(n \log n)$

Tempo $O(n \log n)$. Tipicamente viene fuori quando si esamina la complessità di algoritmi basati sulla tecnica del divide et impera

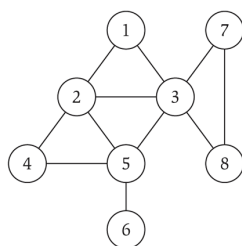
Ordinamento. Mergesort e heapsort sono algoritmi di ordinamento che effettuano $O(n \log n)$ confronti.

Il più grande intervallo vuoto. Dati n time-stamp x_1, \dots, x_n che indicano gli istanti in cui le copie di un file arrivano al server, vogliamo determinare qual è l'intervallo di tempo più grande in cui non arriva alcuna copia del file.

Soluzione $O(n \log n)$. Ordina in modo non decrescente i time stamp. Scandisci la lista ordinata dall'inizio computando la differenza tra ciascun istante e quello successivo. Prendi il massimo delle differenze calcolate. Tempo $O(n \log n + n) = O(n \log n)$

Grafo

- Esempio (vedremo meglio questo concetto nelle prossime lezioni)



$V = \{ 1, 2, 3, 4, 5, 6, 7, 8 \}$
 $E = \{ 1-2, 1-3, 2-3, 2-4, 2-5, 3-5, 3-7, 3-8, 4-5, 5-6 \}$
 $n = 8$
 $m = 11$

Tempo polinomiale $O(n^k)$

Insieme indipendente di dimensione k (k costante). Dato un grafo, esistono k nodi tali che nessuna coppia di nodi è connessa da un arco?

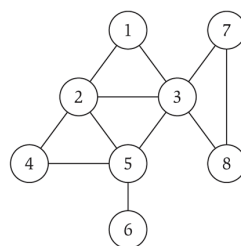
Soluzione $O(n^k)$. Enumerare tutti i sottoinsiemi di k nodi.

```
foreach sottoinsieme S di k nodi {
  controlla se S è un insieme indipendente
  if (S è un insieme indipendente)
    riporta che S è in insieme indipendente
}
```

- Controllare se S è un insieme indipendente = $O(k^2)$
- Numero di sottoinsiemi di k elementi = $\binom{n}{k} = \frac{n(n-1)(n-2)\dots(n-k+1)}{k(k-1)(k-2)\dots(2)(1)} \leq \frac{n^k}{k!}$
- Tempo totale $O(k^2 n^k / k!) = O(n^k)$

Insieme indipendente

- Esempio: per $k=3$ l'algoritmo riporta gli insiemi $\{1,4,6\}$, $\{1,4,7\}$, $\{1,4,8\}$, $\{1,5,7\}$, $\{1,5,8\}$, $\{1,6,7\}$, $\{1,6,8\}$, $\{2,6,7\}$, $\{2,6,8\}$, $\{3,4,6\}$, $\{4,6,7\}$, $\{4,6,8\}$



$V = \{ 1, 2, 3, 4, 5, 6, 7, 8 \}$
 $E = \{ 1-2, 1-3, 2-3, 2-4, 2-5, 3-5, 3-7, 3-8, 4-5, 5-6 \}$
 $n = 8$
 $m = 11$

Tempo esponenziale

Esempio:

Massimo insieme indipendente. Dato un grafo G , qual è la dimensione massima di un insieme indipendente di G ?

Def. insieme indipendente: un insieme indipendente di un grafo è un sottoinsieme di vertici a due a due non adiacenti

Soluzione $O(n^2 2^n)$. Esamina tutti i sottoinsiemi di vertici.

NB: Il numero totale di sottoinsiemi di un insieme di n elementi è 2^n

```

S* ← ∅
foreach sottoinsieme S di nodi {
  controlla se S è un insieme indipendente
  Se (S è il più grande insieme indipendente visto finora)
    aggiorna S* ← S
}

```

Progettazione di Algoritmi, a.a. 2021-22
A. De Bonis

65

65

Tempo esponenziale

Per esercizio proviamo che il tempo del nostro algoritmo per il massimo insieme indipendente è $\Theta(n^2 2^n)$

Dim: Assumiamo per semplicità n dispari

- Stimiamo il tempo per controllare l'indipendenza di tutti gli insiemi di dimensione maggiore o uguale a $\lceil n/2 \rceil$. Un limite inferiore a questo tempo è sicuramente un limite inferiore al tempo totale.
- 1. Il numero di insiemi di dimensione maggiore o uguale a $\lceil n/2 \rceil$ è 2^{n-1}
Dim. Indichiamo con S_1, S_2, S_3, \dots gli insiemi di dimensione $\geq \lceil n/2 \rceil$
- Per ogni insieme S_i di dimensione k , per $k = \lceil n/2 \rceil, \dots, n$, il suo complemento $S'_i = V - S_i$ ha dimensione $n - k < \lceil n/2 \rceil$.
- Ad ogni insieme corrisponde un unico complemento \rightarrow se metto da una parte tutti gli insiemi S_1, S_2, S_3, \dots e dall'altra gli insiemi S'_1, S'_2, S'_3, \dots avrò lo stesso numero di insiemi da una parte e dall'altra \rightarrow ho diviso 2^n insiemi in due metà: una contenente tutto gli insiemi di dimensione $\geq \lceil n/2 \rceil$ e l'altra tutti gli insiemi di dimensione $< \lceil n/2 \rceil$ \rightarrow numero insiemi di dimensione maggiore o uguale a $\lceil n/2 \rceil$ è numero totale insiemi / 2 = $2^n / 2 = 2^{n-1}$
- 2. Il tempo per controllare l'indipendenza di ciascuno di questi insiemi è $\Omega(n^2)$ perché dobbiamo controllare almeno $n/2(n/2-1)/2$ coppie di nodi nel caso pessimo.
- 1. e 2. \rightarrow Tempo totale per controllare insiemi di dimensione maggiore o uguale a $\lceil n/2 \rceil$ è $\Omega(n^2 2^{n-1}) = \Omega(n^2 2^n / 2) = \Omega(n^2 2^n) \rightarrow$ Algoritmo ha tempo $\Omega(n^2 2^n)$

Progettazione di Algoritmi, a.a. 2020-21
A. De Bonis

66

66

Tempo esponenziale

Siccome nelle due slide precedenti abbiamo dimostrato che vale sia $O(n^2 2^n)$ che $\Omega(n^2 2^n)$ allora abbiamo dimostrato che il tempo **dell'algoritmo** è $\Theta(n^2 2^n)$. E se n è pari?

67

Esercizio:

Ci chiediamo: 3^n è $O(2^n)$?

Sappiamo che 3^n è $O(2^n)$ se e solo se esistono due costanti $c > 0$ ed $n_0 \geq 0$ t.c. $3^n \leq c \cdot (2^n)$ per ogni $n \geq n_0$.

Proviamo a determinare tali costanti risolvendo la disequazione $3^n \leq c \cdot 2^n$ rispetto a c

$3^n \leq c \cdot 2^n \iff c \geq 3^n / 2^n = (3/2)^n$. Occorre quindi prendere $c \geq (3/2)^n$.

La funzione $(3/2)^n$ cresce al crescere di n e tende all'infinito al tendere di n all'infinito. Siccome $(3/2)^n$ tende all'infinito, qualsiasi valore scegliamo per la costante c , questo valore sarà superato da $(3/2)^n$ per n sufficientemente grande.

Ne deduciamo che **non** esistono $c > 0$ ed $n_0 \geq 0$ t.c. $3^n \leq c \cdot (2^n)$ per ogni $n \geq n_0$.

→ Abbiamo dimostrato che 3^n non è $O(2^n)$

68