Progetto e analisi di algoritmi

Roberto Cordone

DTI - Università degli Studi di Milano

Polo Didattico e di Ricerca di Crema

Tel. 0373 / 898**089**

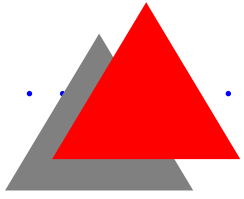
E-mail: **cordone@dti.unimi.it**

Ricevimento: **su appuntamento**

Web page: **http://www.dti.unimi.it/~cordone**

Lezioni: **Martedì dalle 11.00 alle 13.00**

Giovedì dalle 11.00 alle 13.00



Complessità di un problema

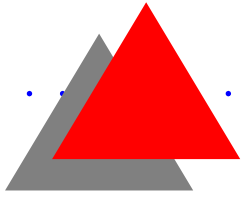
Abbiamo definito quanto costa risolvere un problema con un dato algoritmo

Quanto è difficile in sé un problema?

Quanto costa risolverlo col miglior algoritmo possibile?

Complessità di un problema P è la
complessità del miglior algoritmo per risolvere P
(noto o ignoto che sia)

Ma chi sa qual è il miglior algoritmo possibile?



Esiste un algoritmo ottimo?

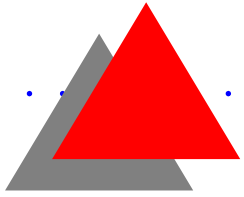
Come si fa a stimare una grandezza che non si conosce, come la complessità del miglior algoritmo possibile?

Si costruiscono

- stima per eccesso (*l'algoritmo ottimo è meglio di...*):
  è il numero di passi di qualsiasi algoritmo noto
- stima per difetto (*l'algoritmo ottimo è peggio di...*):
  è il numero dei passi strettamente necessari a risolvere il problema (qualunque algoritmo si usi)

Gap algoritmico: differenza fra le due stime di complessità

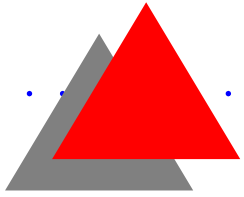
Problema chiuso: problema con gap algoritmico nullo



Somma di numeri binari

$$\begin{array}{r} 1\ 0\ 1\ 1\ 0\ 1\ + \\ 1\ 1\ 1\ 1\ 0\ = \\ \hline \end{array}$$

Sommare due o tre bit richiede un tempo costante

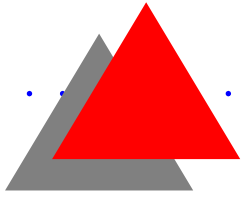


Somma di numeri binari

$$\begin{array}{rrrrrrr} 1 & 0 & 1 & 1 & 0 & 1 & + \\ & 1 & 1 & 1 & 1 & 0 & = \\ \hline & & & & & & \\ & & & & & 1 & \end{array}$$

Sommare due o tre bit richiede un tempo costante

$$T(n) = \Theta(1) + \ldots$$

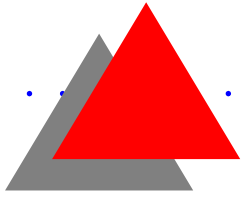


Somma di numeri binari

$$\begin{array}{r} 1\ 0\ 1\ 1\ 0\ 1\ + \\ 1\ 1\ 1\ 1\ 0\ = \\ \hline 1\ 1 \end{array}$$

Sommare due o tre bit richiede un tempo costante

$$T(n) = \Theta(1) + \Theta(1) + \ldots$$

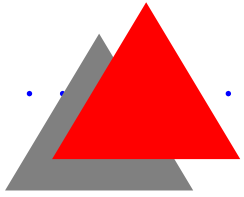


Somma di numeri binari

$$
\begin{array}{r}
+ \\
1\ 0\ 1\ 1\ 0\ 1\ + \\
1\ 1\ 1\ 1\ 0\ = \\
\hline
0\ 1\ 1
\end{array}
$$

Sommare due o tre bit richiede un tempo costante

$$T(n) = \Theta(1) + \Theta(1) + \Theta(1) + \ldots$$

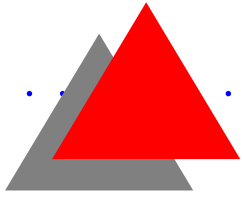


Somma di numeri binari

$$\begin{array}{cccccccc} & & + & + & & & & \\ & 1 & 0 & 1 & 1 & 0 & 1 & + \\ & & 1 & 1 & 1 & 1 & 0 & = \\ \hline & & & 1 & 0 & 1 & 1 & \end{array}$$

Sommare due o tre bit richiede un tempo costante

$$T(n) = \Theta(1) + \Theta(1) + \Theta(1) + \ldots$$

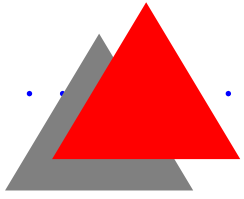


Somma di numeri binari

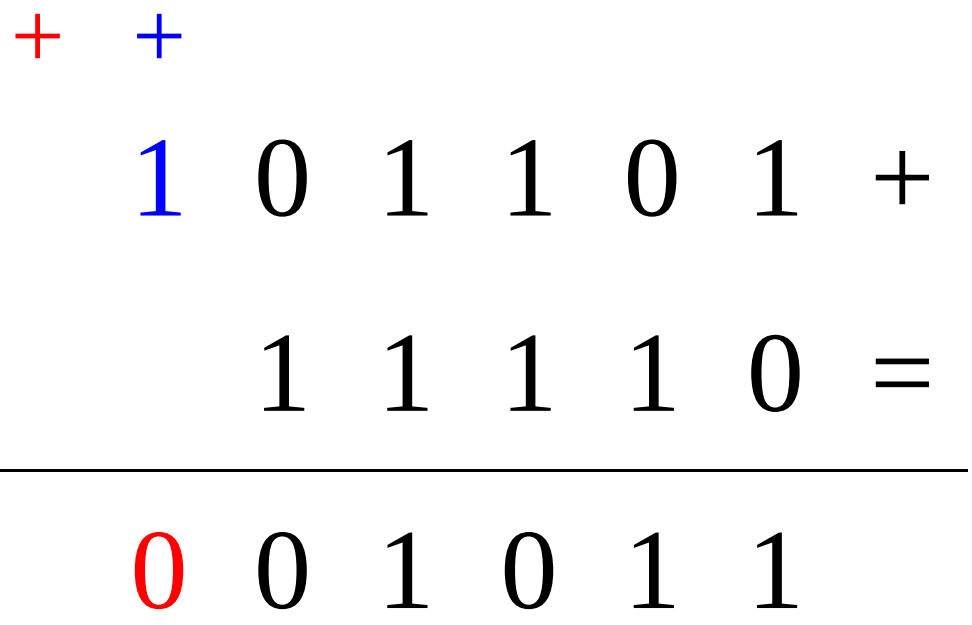
$$\begin{array}{cccccccc} & + & + & & & & & \\ 1 & 0 & 1 & 1 & 0 & 1 & + & \\ & 1 & 1 & 1 & 1 & 0 & = & \\ \hline & 0 & 1 & 0 & 1 & 1 & & \end{array}$$

Sommare due o tre bit richiede un tempo costante

$$T(n) = \Theta(1) + \Theta(1) + \Theta(1) + \ldots$$



Somma di numeri binari



Sommare due o tre bit richiede un tempo costante

$$T(n) = \Theta(1) + \Theta(1) + \Theta(1) + \ldots$$

Somma di numeri binari

$$
\begin{array}{rcccccc}
 & 1 & 0 & 1 & 1 & 0 & 1 \; + \\
 & & 1 & 1 & 1 & 1 & 0 \; = \\
\hline
1 & 0 & 0 & 1 & 0 & 1 & 1
\end{array}
$$

Sommare due o tre bit richiede un tempo costante

$$T(n) = \Theta(1) + \Theta(1) + \Theta(1) + \ldots$$

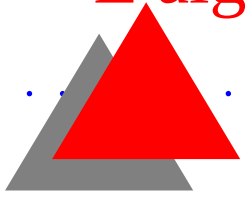
Somma di numeri binari

$$\begin{array}{r} 1\ 0\ 1\ 1\ 0\ 1\ + \\ 1\ 1\ 1\ 1\ 0\ = \\ \hline 1\ 0\ 0\ 1\ 0\ 1\ 1 \end{array}$$

Sommare due o tre bit richiede un tempo costante

$$T(n) = \Theta(1) + \Theta(1) + \Theta(1) + \ldots$$

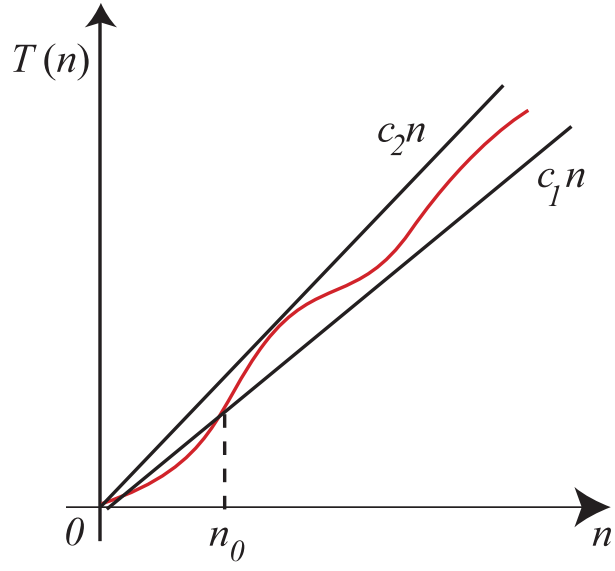
L'algoritmo classico di somma è lineare: $T(n) = \Theta(n)$



Somma di numeri binari

L'algoritmo classico di somma è lineare,
cioè racchiuso "a sandwich" fra due rette

$$\exists n_0 \in \mathbb{N}, c_1, c_2 > 0 : \quad c_1 n \leq T(n) \leq c_2 n \quad \forall n \geq n_0$$



Che complessità ha il problema della somma?

Somma di numeri binari

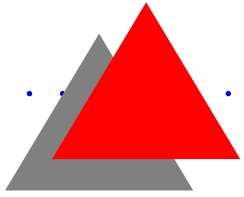
Vogliamo sapere se esiste un algoritmo A sublineare (dunque migliore di quello classico)

Supponiamo per assurdo che A esista e risolva l'istanza I

- A non legge interamente I (sarebbe lineare. . .)
- Eseguiamo A sull'istanza I', che differisce da I solo per uno dei bit non letti: A dà la stessa soluzione

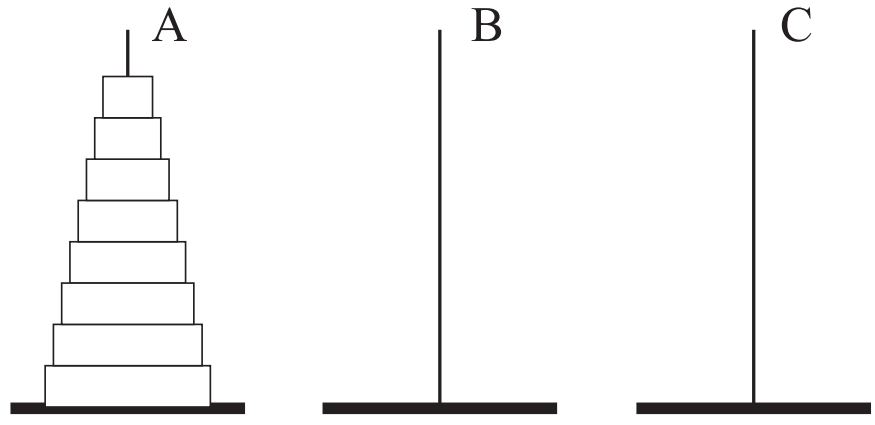
Quindi A non risolve correttamente I'

L'algoritmo classico per l'addizione è ottimo!



Le torri di Hanoi

Mossa: spostare un pezzo dalla cima di una pila alla cima di un'altra; nessun pezzo può stare su un pezzo più stretto



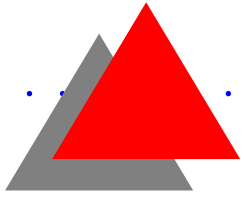
Problema: Con quali mosse si sposta la pila di pezzi in B?

Narra la leggenda che il mondo avrà termine nel momento in cui qualcuno terminerà di spostare una pila di 64 *pezzi d'oro*

Un approccio

Divide et impera

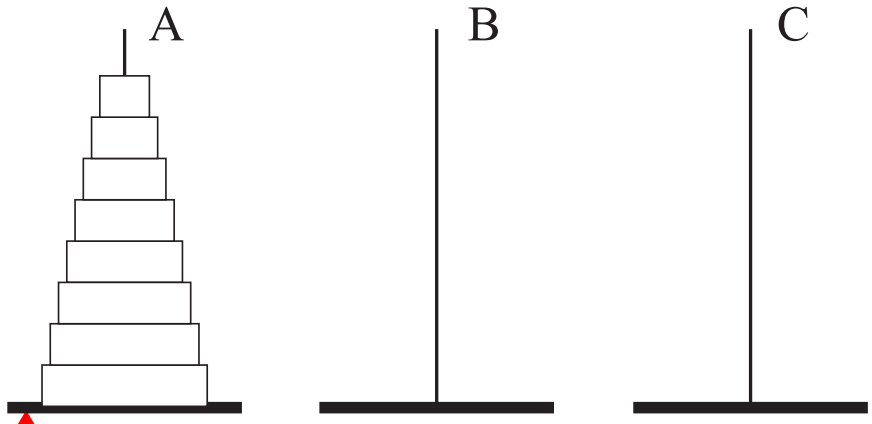
- Se il problema è molto piccolo,
  lo risolvo con la forza bruta
- Altrimenti
  - lo scompongo in problemi più piccoli
  - affronto i problemi più piccoli con lo stesso approccio
  - ricompongo le soluzioni dei problemi più piccoli nella soluzione globale



Un algoritmo

Se c'è un solo pezzo su A, spostalo su B
Altrimenti

1. sposta tutti i pezzi tranne il più largo da A a C
2. sposta il pezzo più largo da A a B
3. sposta l'intera pila da C a B



MoveStack(n, r, s) {

If $n = 1$ then *MoveDisk*(r, s)

else *MoveStack*$(n-1, r, 6-r-s)$

MoveDisk(r, s)

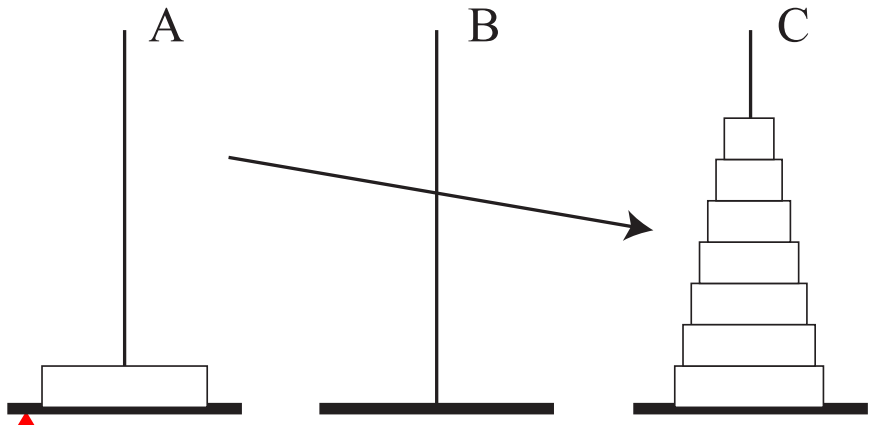
MoveStack$(n-1, 6-r-s, s)$

}

Un algoritmo

Se c'è un solo pezzo su A, spostalo su B
Altrimenti

1. sposta tutti i pezzi tranne il più largo da A a C
2. sposta il pezzo più largo da A a B
3. sposta l'intera pila da C a B



MoveStack(n, r, s) {

If $n = 1$ then *MoveDisk*(r, s)

else *MoveStack*$(n - 1, r, 6 - r - s)$

MoveDisk(r, s)

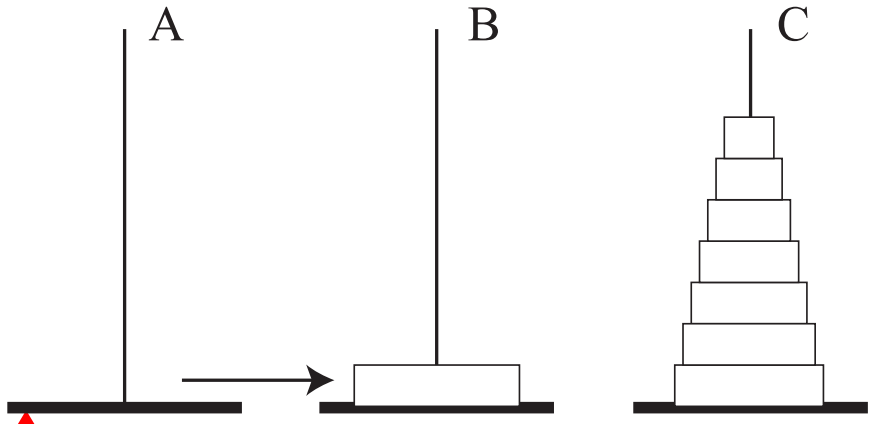
MoveStack$(n - 1, 6 - r - s, s)$

}

Un algoritmo

Se c'è un solo pezzo su A, spostalo su B
Altrimenti

1. sposta tutti i pezzi tranne il più largo da A a C
2. sposta il pezzo più largo da A a B
3. sposta l'intera pila da C a B



MoveStack(n, r, s) {

If $n = 1$ then *MoveDisk*(r, s)

else *MoveStack*$(n-1, r, 6-r-s)$

MoveDisk(r, s)

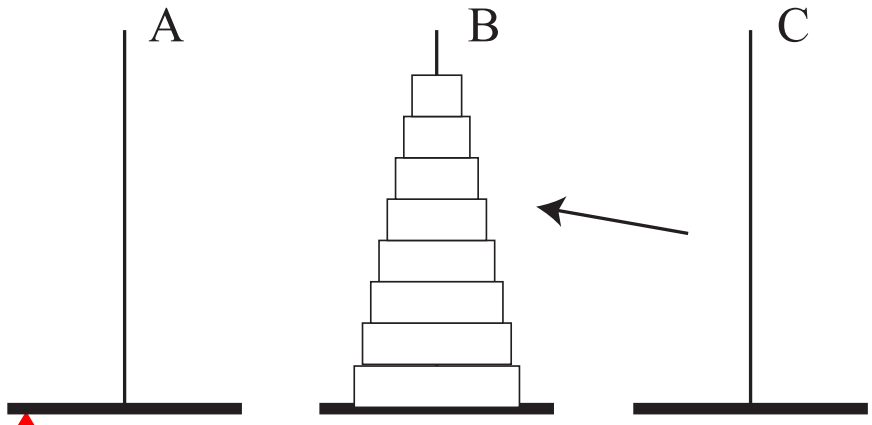
MoveStack$(n-1, 6-r-s, s)$

}

Un algoritmo

Se c'è un solo pezzo su A, spostalo su B
Altrimenti

1. sposta tutti i pezzi tranne il più largo da A a C
2. sposta il pezzo più largo da A a B
3. sposta l'intera pila da C a B



MoveStack(n, r, s) {

If $n = 1$ then *MoveDisk*(r, s)

else *MoveStack*$(n-1, r, 6-r-s)$

MoveDisk(r, s)

MoveStack$(n-1, 6-r-s, s)$

}

Hanoi: stima per eccesso

Quante mosse ci vogliono?

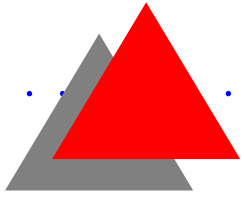
$$T_A(n) = \begin{cases} 1 & \text{se } n = 1 \\ T_A(n-1) + 1 + T_A(n-1) & \text{se } n > 1 \end{cases}$$

$$\Rightarrow T_A(n) = 2^n - 1$$

Quindi, se l'operazione elementare è lo spostamento di un pezzo, l'algoritmo A ha complessità $T_A(n) = O(2^n)$

Per $n = 64$ e una mossa al secondo, $T_A(64) = 2^{64} - 1$ secondi, cioè quasi 600 miliardi di anni.

L'universo finirà prima...



Hanoi: stima per difetto

Sia $T^*(n, A, B)$ il numero minimo possibile di mosse

- la mossa t-esima sposta il pezzo n da A a B
- subito prima, A contiene solo il pezzo n, B è vuoto e C contiene gli altri pezzi in pila ordinata
- le prime mosse spostano $n-1$ dischi da A a C
- le ultime mosse spostano $n-1$ dischi da C a B

$$T^*(n, A, B) \geq T^*(n-1, A, C) + 1 + T^*(n-1, C, B)$$

$\Rightarrow T^*(n, A, B) \geq 2^n - 1$. Poiché $T^*(n, A, B) \leq T(n, A, B)$, risulta che $T^*(n, A, B) = T_A(n, A, B)$ e A è l'algoritmo ottimo!

Nota: da studiare il caso in cui si sposta n da A a B passando per C...

Ricerca in sequenze ordinate

Istanza: una chiave x e un insieme V

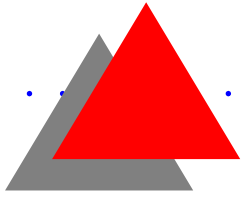
Soluzione

- $i \in \{1, \ldots, |V|\}$ se $x = V[i]$
- *Falso* se $x \notin V$

Consideriamo due stime per eccesso (algoritmi)

1. ricerca sequenziale
2. ricerca binaria

e una stima per difetto



Ricerca: stima per eccesso (1)

RicercaSequenziale(V, x)

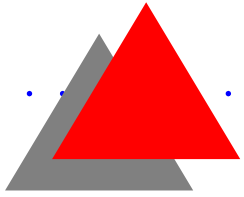
$i := 1$;

While $i \leq |V|$ do

If $V[i] = x$ Return i;

$i := i + 1$;

Return *Falso*;

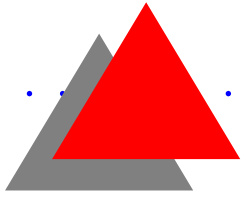


Ricerca: stima per eccesso (1)

- Se ogni elemento è codificato come stringa di α simboli, la dimensione di un'istanza è $|I| = \alpha n + \alpha$
- Supponiamo che la macchina RAM compia un numero di passi $\leq C$ per ogni operazione (l'assegnamento $i := 1$, l'incremento $i := i + 1$, i confronti $i \leq n$ e $V[i] = x$)
- La complessità dipende fortemente dall'istanza: il numero di iterazioni dipende dalla presenza (e posizione) di x in V

Stima nel caso peggiore ($x \notin V$): si esegue il ciclo n volte e il numero totale di passi è $T(|I|) \leq C + 3Cn = C + 3C\dfrac{(|I| - \alpha)}{\alpha}$

$$\Rightarrow T(|I|) \in O(|I|)$$



Ricerca: stima per eccesso (2)

RicercaBinaria(V, x)

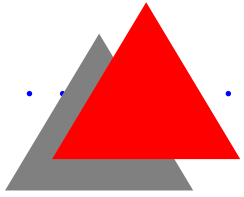
$i := 1$; $j := |V|$;

While $i < j$ do

$$m := \left\lfloor \frac{i+j}{2} \right\rfloor;$$

If $x \leq V[m]$ then $j := m$ else $i := m + 1$;

If $x = V[i]$ Return i else Return *Falso*;

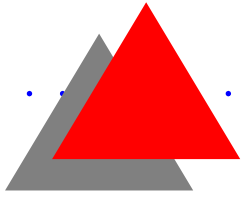


Ricerca: stima per eccesso (2)

- La dimensione dell'istanza è sempre $|I| = \alpha n$
- La macchina RAM compie un numero di passi $\leq C$ per ogni operazione
- Ogni ciclo dimezza (almeno) il vettore; l'algoritmo termina quando $n/2^h \leq 1$: si eseguono $h = \lceil \log_2 n \rceil$ cicli
- La complessità dipende dalla dimensione dell'istanza

Stima nel caso peggiore: $T(|I|) \leq 2C + 6C \left\lceil \log_2 \frac{|I| - \alpha}{\alpha} \right\rceil$

$$\Rightarrow T(|I|) \in O(\log_2 |I|)$$



Ricerca: stima per difetto

Qualsiasi algoritmo per questo problema deve distinguere $|V| + 1$ casi:

- la chiave x è in posizione i (con $i = 1, \ldots, |V|$)
- la chiave x non è in V

Ogni confronto fra due numeri a e b distingue al più 3 casi:

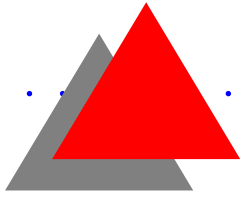
$$a < b \qquad a = b \qquad a > b$$

Un algoritmo che fa t confronti distingue al più 3^t casi

Per risolvere il problema della ricerca deve rendere

$$3^t \geq |V| + 1 \Rightarrow t \geq \log_3 (|V| + 1)$$

Quindi, il problema della ricerca è almeno logaritmico!



Complessità di un algoritmo

Come si dimostra che un algoritmo ha complessità...

- $T(n) \in O(f(n))$?

  Per tutte le istanze di dimensione n

  l'algoritmo richiede al più $c_2 f(n)$ passi

- $T(n) \in \Omega(f(n))$?

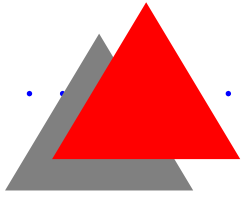
  Per almeno un'istanza di dimensione n

  l'algoritmo richiede almeno $c_1 f(n)$ passi

- $T(n) \in \Theta(f(n))$?

  Valgono entrambe le proprietà

La prospettiva è quella del caso peggiore



Complessità di un problema

Come si dimostra che un problema ha complessità...

- $T(n) \in O(f(n))$?

  Per almeno un algoritmo si richiedono al più $c_2 f(n)$ passi

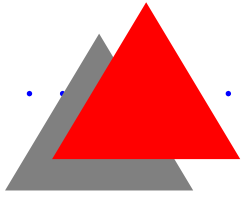
- $T(n) \in \Omega(f(n))$?

  Per tutti gli algoritmi si richiedono almeno $c_1 f(n)$ passi

- $T(n) \in \Theta(f(n))$?

  Valgono entrambe le proprietà

La prospettiva è quella del caso migliore

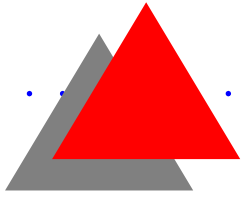


Classi di complessità

Classe di complessità è l'insieme di tutti i problemi che hanno complessità temporale $T(n)$ o spaziale $S(n)$ appartenente a una data classe di funzioni

$\mathcal{P}$ è la classe dei problemi di esistenza la cui complessità temporale $T(n)$ è un polinomio

Se la classe di funzioni è abbastanza ampia, si può assegnarle un problema senza conoscerne esattamente la complessità: ad es., una stima per eccesso (algoritmo) polinomiale garantisce complessità polinomiale (anche se sconosciuta)

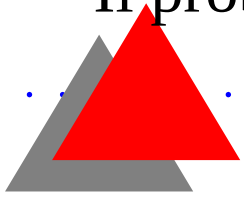


Un problema polinomiale (1)

Il problema (di decisione) dell'albero ricoprente minimo

- Istanza I:
  - un grafo non orientato $G = (V, E)$ semplice e connesso
  - una funzione costo definita sui lati ($c : E \rightarrow \mathbb{Z}$)
  - un numero $K \in \mathbb{Z}$
- Soluzione S:
  - *Vero* se il grafo $G = (V, E)$ contiene un albero ricoprente di costo totale $\leq K$
  - *Falso* altrimenti

Il problema ammette diversi algoritmi polinomiali: $MST \in \mathcal{P}$



Un problema polinomiale (2)

Il problema è aperto:

- la dimensione $|I|$ delle istanze è proporzionale ad $|E|$
- per risolverlo occorre considerare i costi di tutti i lati (stima per difetto lineare rispetto ad $|E|$, e quindi ad $|I|$)
- i migliori algoritmi noti danno una stima per eccesso superlineare rispetto ad $|E|$, e quindi ad $|I|$

Ma l'algoritmo di Prim ha complessità $O(n^2)$ e su grafi densi $|E| \in \Theta(n^2)$. Quindi $|I| \in \Theta(n^2)$ e Prim è lineare in $|I|$

Quindi, il sottoproblema relativo ai grafi densi è chiuso