POLITECNICO
DI TORINO

Dipartimento
di Automatica e Informatica

Gli Algoritmi di Ordinamento

Paolo Camurati

Definizione del problema

Ordinamento:

- Input: simboli $<a_1, a_2, \ldots, a_n>$ di un insieme con relazione d'ordine totale $\leq$
- Output: permutazione $<a'_1, a'_2, \ldots, a'_n>$ dell'input per cui vale la relazione d'ordine $a'_1 \leq a'_2 \leq \ldots \leq a'_n$.

Relazione d'ordine $\leq$

È una relazione binaria tra elementi di un insieme A che soddisfa le seguenti proprietà:

- riflessiva $\forall\, x \in A\ x \leq x$
- antisimmetrica $\forall\, x, y \in A\ \ x \leq y \wedge y \leq x \Rightarrow x = y$
- transitiva $\forall\, x, y, z \in A\ \ x \leq y \wedge y \leq z \Rightarrow x \leq z$

A è un insieme parzialmente ordinato (poset). Se la relazione $\leq$ vale $\forall\, x, y \in A$, A si dice totalmente ordinato

Esempi di relazione d’ordine $\leq$:

- relazione (totale) $\leq$ su numeri naturali, relativi, razionali e reali N, Z, Q, R, ordine alfabetico di stringhe, ordine cronologico di date
- relazione (parziale) di divisibilità su naturali escluso 0

L'importanza dell'ordinamento

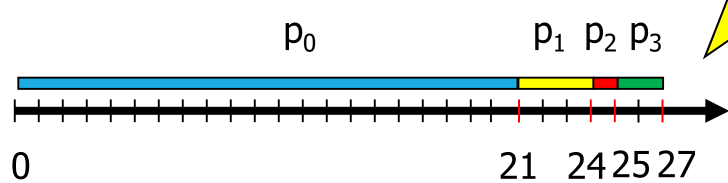
- Il 30% dei tempi di CPU è per ordinare dati
- CPU scheduling: come selezionare tra i processi pronti quello da eseguire sulla CPU.
  - soluzione semplice: coda, quindi politica *First-come First-Served*: la prima richiesta che perviene è la prima ad essere soddisfatta
  - problema: *minimizzare il tempo di attesa medio*: diventa un problema di ottimizzazione.

Esempio: processi p_i con durata:

p_0 21, p_1 3, p_2 1, p_3 2

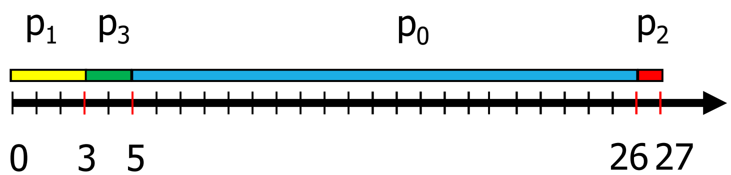
- scheduling p_0-p_1-p_2-p_3

diagramma di Gantt



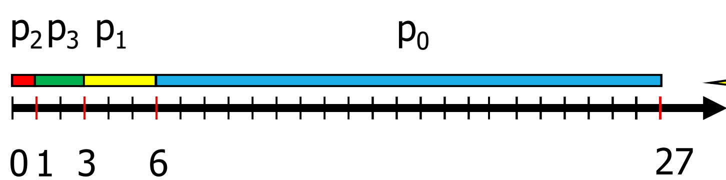
tempo medio di attesa (0+21+24 +25)/4 =17,5

- scheduling p_1-p_3-p_0-p_2



tempo medio di attesa (0+3+5 +26)/4 =8,5

- scheduling (ordinamento per durata crescente) p_2-p_3-p_1-p_0



scheduling shortest job first

tempo medio di attesa (0+1+3 +6)/4 =2,5

Applicazioni dell'ordinamento

- ordinamento di una lista di nomi
- organizzazione di un libreria MP3
- visualizzazione dei risultati di Google PageRank
- …

applicazioni ovvie

- trovare la mediana
- ricerca binaria in un database
- trovare i duplicati in una mailing list
- …

problemi semplici se i dati sono ordinati

- compressione dei dati
- computer graphics (es. inviluppo complesso)
- biologia computazionale
- …

applicazioni non banali

Classificazione: interni/esterni

- Ordinamento interno
  - dati in memoria centrale
  - accesso diretto agli elementi
- Ordinamento esterno
  - dati in memoria di massa
  - accesso sequenziale agli elementi

Classificazione: in loco / stabili

- Ordinamento in loco
  vettore di n dati + locazioni di memoria ausiliarie in numero fisso

- Ordinamento stabile
  immutato l’ordinamento relativo di dati con ugual valore della chiave (l’ordine in uscita di dati con la stessa chiave è lo stesso dell’ordine in ingresso)o)

Esempio

Struct con 2 chiavi: nome (la chiave è la prima lettera) e gruppo (la chiave è un intero)

Primo ordinamento per prima lettera:

Andrea	3
Barbara	4
Chiara	3
Fabio	3
Franco	1
Giada	4
Lucia	3
Roberto	2

Secondo ordinamento per gruppo:

Algoritmo
NON stabile

Franco	1
Roberto	2
Chiara	3
Fabio	3
Andrea	3
Lucia	3
Giada	4
Barbara	4

Algoritmo
stabile

Franco	1
Roberto	2
Andrea	3
Chiara	3
Fabio	3
Lucia	3
Barbara	4
Giada	4

Classificazione: complessità

- $O(n^2)$:
  - semplici, iterativi, basati sul confronto
  - Insertion sort, Selection sort, Exchange/Bubble sort
- $O(n^x)$ con $x \leq 2$
  - evoluzione di quelli semplici, iterativi, basati sul confronto
  - Shell sort, $O(n^2)$, $O(n^{3/2})$, $O(n^{4/3})$ in funzione della scelta di una certa sequenza
- $O(n \log n)$:
  - più complessi, ricorsivi, basati sul confronto. Si vedranno nel Corso del II anno
  - Merge sort, Quick sort, Heap sort
- $O(n)$:
  - applicabili solo con ipotesi restrittive sui dati, basati sul calcolo
  - Counting sort, Radix sort, Bin/Bucket sort

È possibile un'analisi più raffinata, in cui si distinguono le operazioni di:

- confronto
- scambio.

Quando infatti il dato da ordinare occupa molta memoria, lo spostamento di blocchi di memoria (non quindi di soli puntatori) può essere costoso.
La complessità asintotica comunque non cambia, in quanto misurata sui soli confronti.

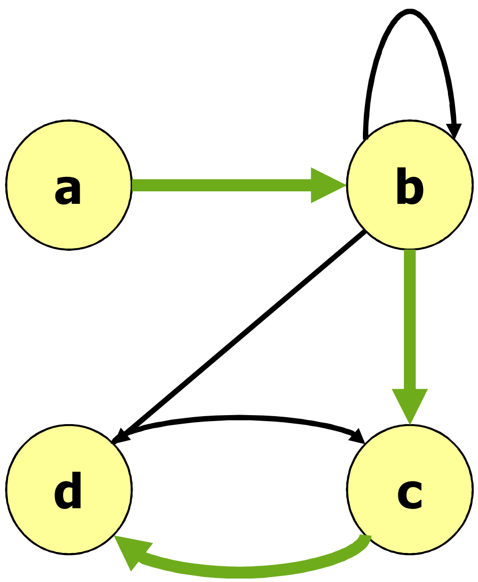
Grafi: i Cammini

In un grafo G = (V, E)

Cammino p: $u \rightarrow_p u'$:
$\exists(v_0,v_1,v_2,\ldots,v_k) \mid u=v_0,\ u'=v_k,\ \forall i = 1,2,\ldots,k\ (v_{i-1},v_i) \in E$

- k = lunghezza del cammino
- u' è raggiungibile da u $\Leftrightarrow\ \exists$ p: $u \rightarrow_p u'$. In un grafo non orientato vale anche che u è raggiungibile da u' $\Leftrightarrow\ \exists$ p: $u' \rightarrow_p u$
- cammino p semplice $\Leftrightarrow (v_0,v_1,v_2,\ldots,v_k) \in$ p distinti

Esempio



$G = (V, E)$

p: $a \rightarrow_p d$: (a, b), (b, c), (c, d)

$k = 3$

d è raggiungibile da a (non necessariamente viceversa)

p semplice.

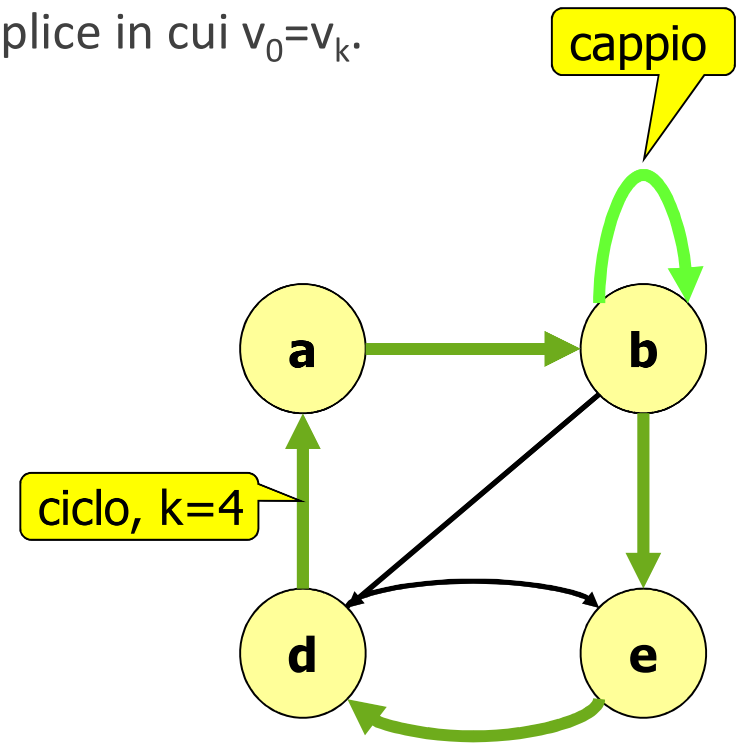
Grafi: i Cicli

Ciclo = cammino in cui $v_0=v_k$.

Ciclo semplice = cammino semplice in cui $v_0=v_k$.

Cappio = ciclo di lunghezza 1.

Un grafo senza cicli = aciclico.



Grafi non orientati: Connessione

Un grafo non orientato G = (V, E) si dice connesso se e solo se:

$\forall v_i, v_j \in V \qquad \exists p \qquad v_i \rightarrow_p v_j$

Ci

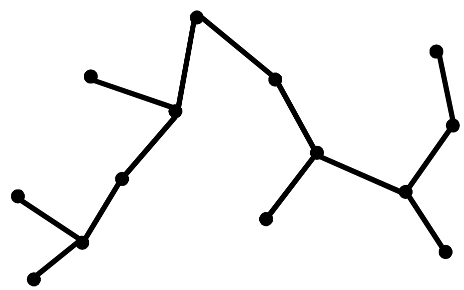
oè ogni coppia di vertici è connessa da un cammino.

Componente connessa: sottografo connesso massimale (= $\nexists$ sottoinsiemi per cui vale la proprietà che lo includono).

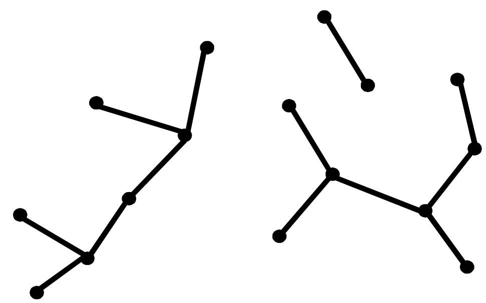
Grafo non orientato connesso: è presente una sola componente connessa (tutto il grafo).

Alberi non radicati (o liberi)

Albero non radicato (o libero) = grafo non orientato, connesso, aciclico



Foresta = grafo non orientato, aciclico



Proprietà

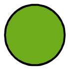
$G = (V, E)$ grafo non orientato $|E|$ archi, $|V|$ nodi:

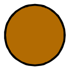
- G = albero non radicato
- ogni coppia di nodi è connessa da uno e un solo cammino semplice
- G connesso, la rimozione di un arco lo sconnette
- G connesso e $|E| = |V| - 1$
- G aciclico e $|E| = |V| - 1$
- G aciclico, l'aggiunta di un arco introduce un ciclo.

Alberi radicati

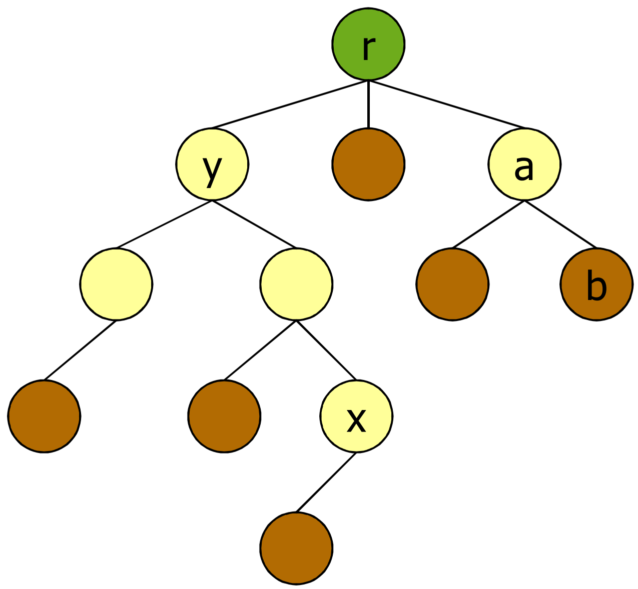
∃ nodo r detto radice che induce una relazione di parentela tra nodi:

- y antenato di x se y appartiene al cammino da r a x. x discendente di y
- antenato proprio se $x \neq y$
- padre/figlio: nodi adiacenti

La radice non ha padre 

Le foglie non hanno figli 

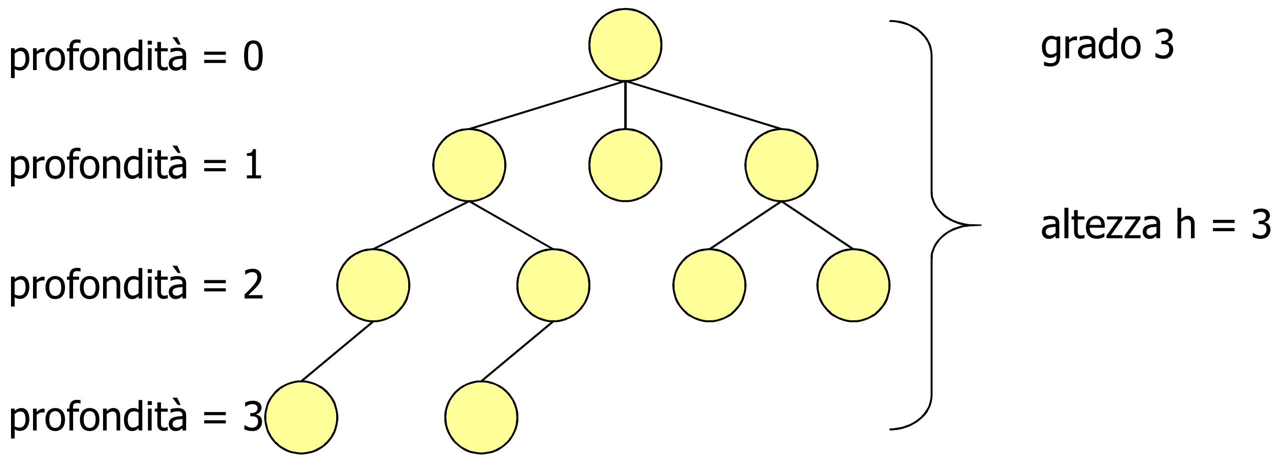
Esempio



r radice
y antenato proprio di x
x discendente proprio di y
a padre di b
b figlio di a

Proprietà di un albero T

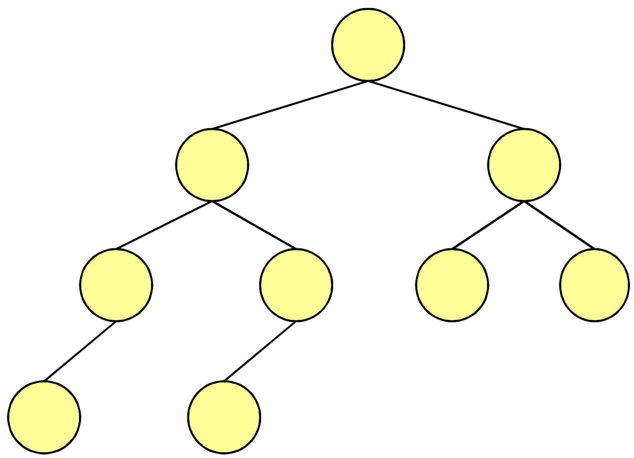
- grado(T) = numero max di figli
- profondità(x) = lunghezza del cammino da r a x
- altezza(T) = profondità massima.



Albero binario

Definizione:

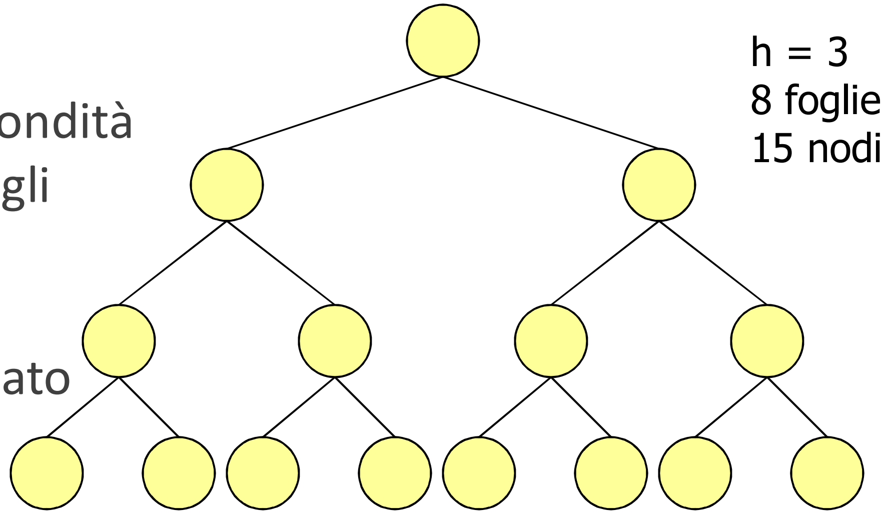
- Albero di grado 2: ogni nodo ha 0, 1 o 2 figli
- Possibile anche una definizione ricorsiva (la si vedrà nel corso del II anno)



Albero binario completamente bilanciato (pieno)

Due condizioni:

- tutte le foglie hanno la stessa profondità
- ogni nodo o è una foglia o ha 2 figli



Albero binario completamente bilanciato (pieno) di altezza h:

- numero di foglie 2^h
- numero di nodi = $\Sigma_{0 \leq i \leq h} 2^i = 2^{h+1} - 1$

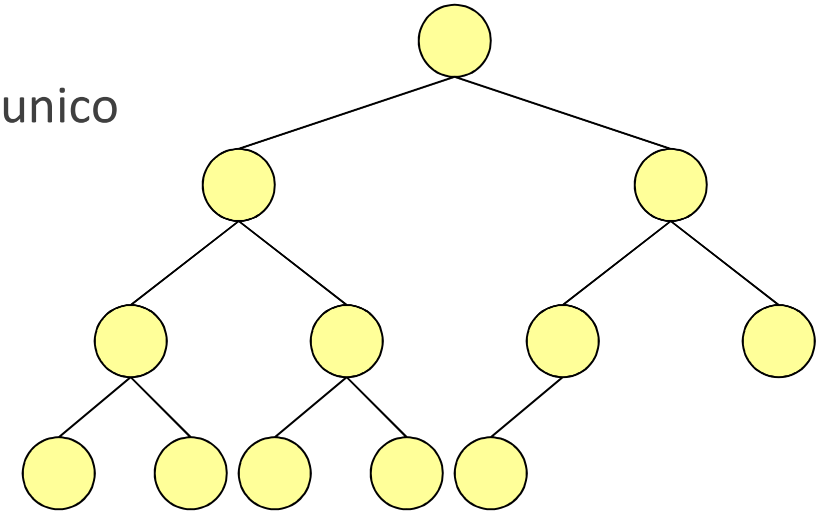
progressione geometrica finita di ragione 2


Hyphaene Compressa - Doum Palm
© Shlomit Pinter

Albero binario completo (a sinistra)

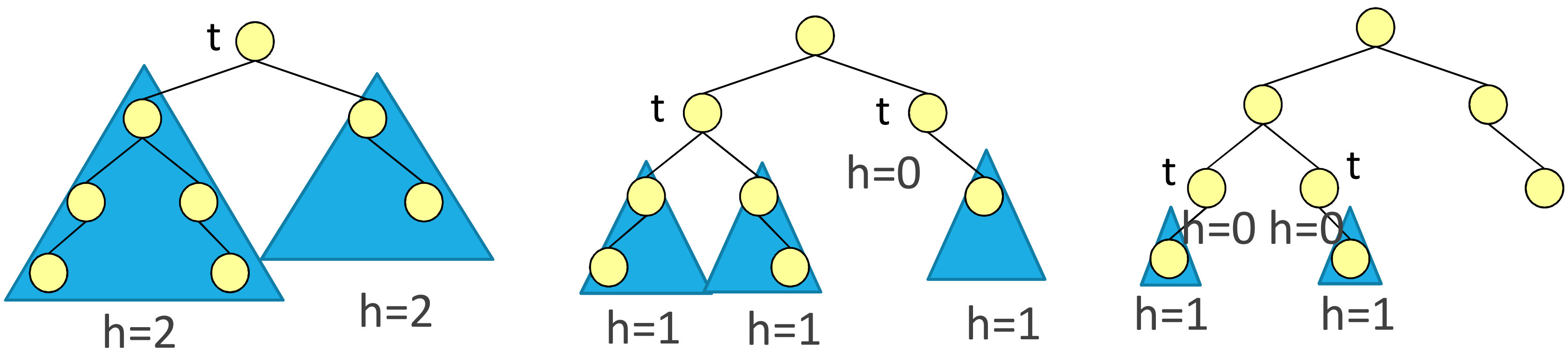
Tutti i livelli sono completi (hanno tutti i nodi) eccetto l’ultimo che è riempito da sinistra a destra.

Dato un numero di nodi n esiste ed è unico l’albero completo (a sinistra).

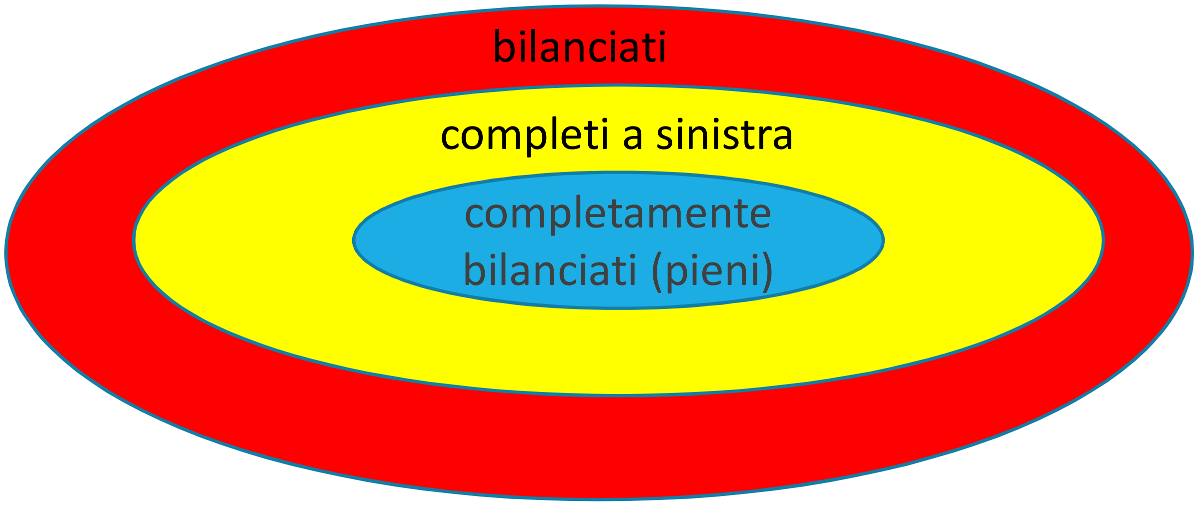


Albero binario bilanciato in altezza

Un albero è bilanciato in **altezza** se e solo se, per ogni sottoalbero t radicato in un suo nodo, l'altezza del sottoalbero sinistro di t differisce di al più di 1 dall'altezza del sottoalbero destro di t.

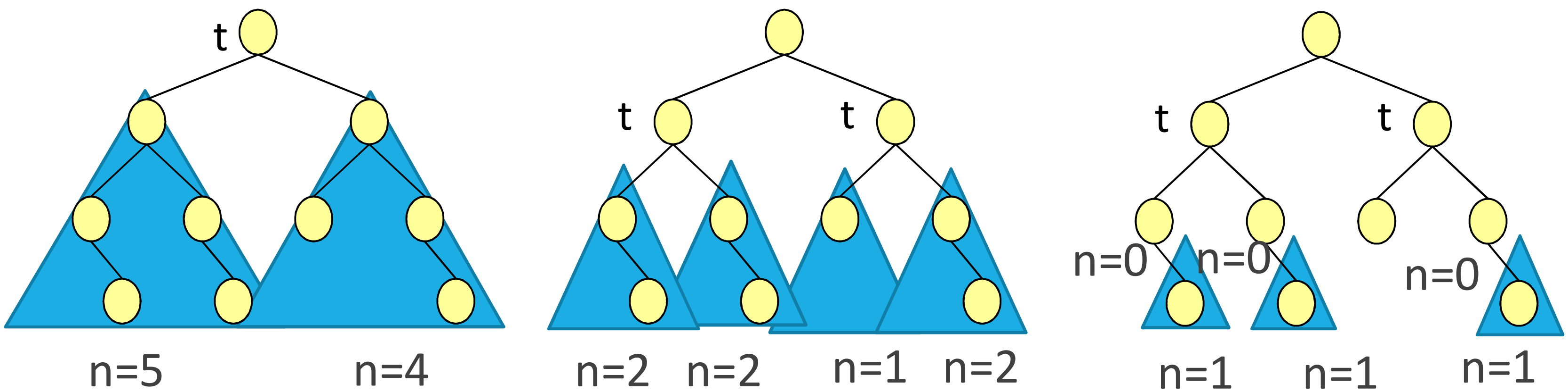


Gli alberi completamente bilanciati (pieni) sono un sottoinsieme proprio degli alberi completi (a sinistra) che a loro volta sono un sottoinsieme proprio degli alberi bilanciati.



Albero binario bilanciato in nodi

Un albero è bilanciato in nodi se e solo se, per ogni sottoalbero t radicato in un suo nodo, il numero di nodi del sottoalbero sinistro di t differisce di al più di 1 dal numero di nodi del sottoalbero destro di t.



Limite inferiore (Ω)

Scopo: determinare un limite inferiore alla complessità asintotica di caso peggiore per TUTTI gli algoritmi di ordinamento basati sul confronto.

La dimostrazione è INDIPENDENTE dall'algoritmo.

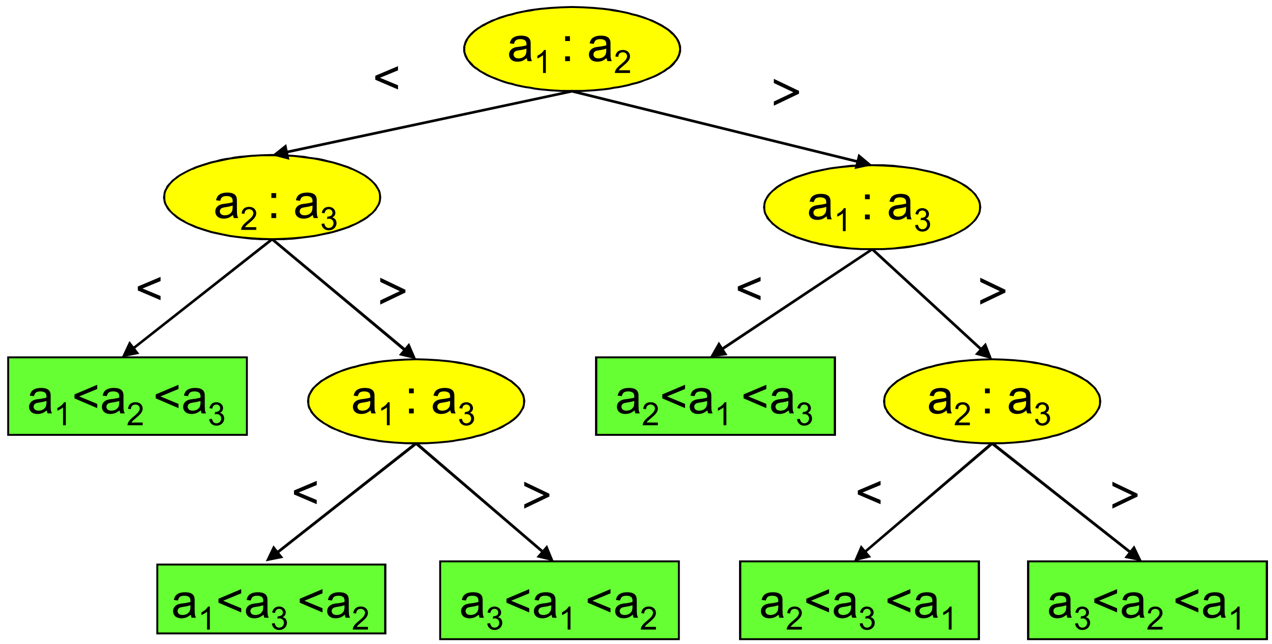
Operazione elementare: confronto tra 2 elementi $a_i : a_j$

Esito: decisione ($a_i > a_j$ o $a_i \leq a_j$), riportata su un albero binario detto albero delle decisioni.

Esempio

Analizzare la complessità di ordinare il vettore A di 3 elementi distinti a_1, a_2 , a_3.

Si crea un albero delle decisioni dove il nodo è etichettato con il confronto corrente ($a_i : a_j$) e i 2 archi con l'esito (> o <). Si prosegue con altri confronti fino ad arrivare ad una soluzione (foglia).


a1 : a2
<
>
a2 : a3
a1 : a3
<
>
<
>
a1<a2 <a3
a1 : a3
a2<a1 <a3
a2 : a3
<
>
<
>
a1<a3 <a2
a3<a1 <a2
a2<a3 <a1
a3<a2 <a1

La complessità è legata al numero di confronti.

Quale è il minimo numero di confronti da fare nel caso peggiore? 3

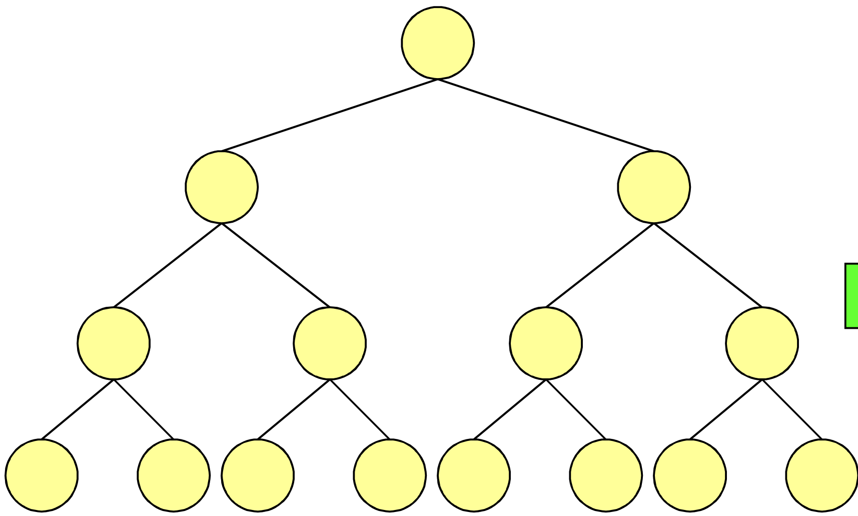
L'albero delle decisioni ha altezza h=3. Il numero minimo di confronti da eseguire nel caso peggiore è pari all'altezza h.

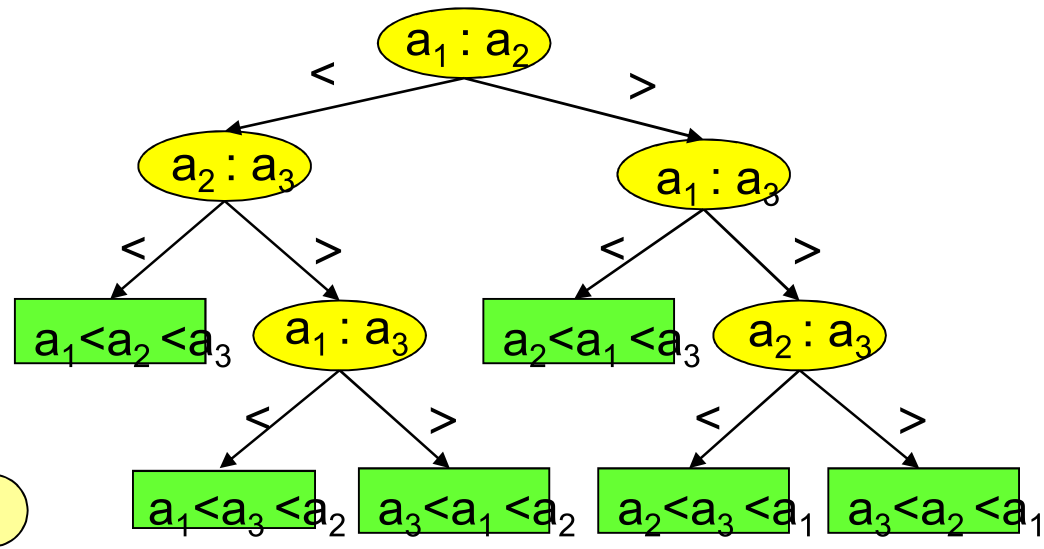
La complessità minima di caso peggiore è O(h). La complessità è espressa in funzione dell'altezza h, non del numero di dati n.

Quale altezza ha un albero delle decisioni in grado di contenere le decisioni per un vettore di n dati?

Un albero binario completamente bilanciato (pieno) di altezza h ha:

- 2^h foglie
- $\Sigma_{0 \leq i \leq h} 2^i = 2^{h+1} - 1$ nodi





Per n dati distinti: il numero di ordinamenti è pari al numero di permutazioni, quindi è n!

Gli ordinamenti stanno nelle foglie dell'albero, quindi ci devono essere almeno tante foglie quanti sono gli ordinamenti

$$2^h \geq n!$$

Usando l'approssimazione di Stirling $n! > (n/e)^n$ si ricava

$$2^h \geq n! > (n/e)^n$$

Prendendo il logaritmo di entrambi i membri si ottiene

$$h > \lg(n/e)^n = n \lg n - n \lg e = \Omega(n \lg n)$$

Non esistono algoritmi di ordinamento basati sul confronto la cui complessità asintotica di caso peggiore sia migliore di quella linearitmica.

Gli algoritmi di ordinamento basati sul confronto che sono $\Omega(n \lg n)$ sono **OTTIMI**.