Università di Pisa

Pietro Ducange

Algoritmi e strutture dati

Limiti Inferiori

a.a. 2020/2021

Si ringrazia la prof. Nicoletta De Francesco per aver messo a disposizione la maggior parte delle slide utilizzate nella presente lezione

Limiti inferiori per i problemi

Un problema è di ordine $\Omega(f(n))$ se non è possibile trovare un algoritmo che lo risolva con complessità minore di $f(n)$

Tutti gli algoritmi che risolvono il problema sono O(f(n))

Limiti inferiori: alberi di decisione

Si applica soltanto agli algoritmi

- **basati su confronti**

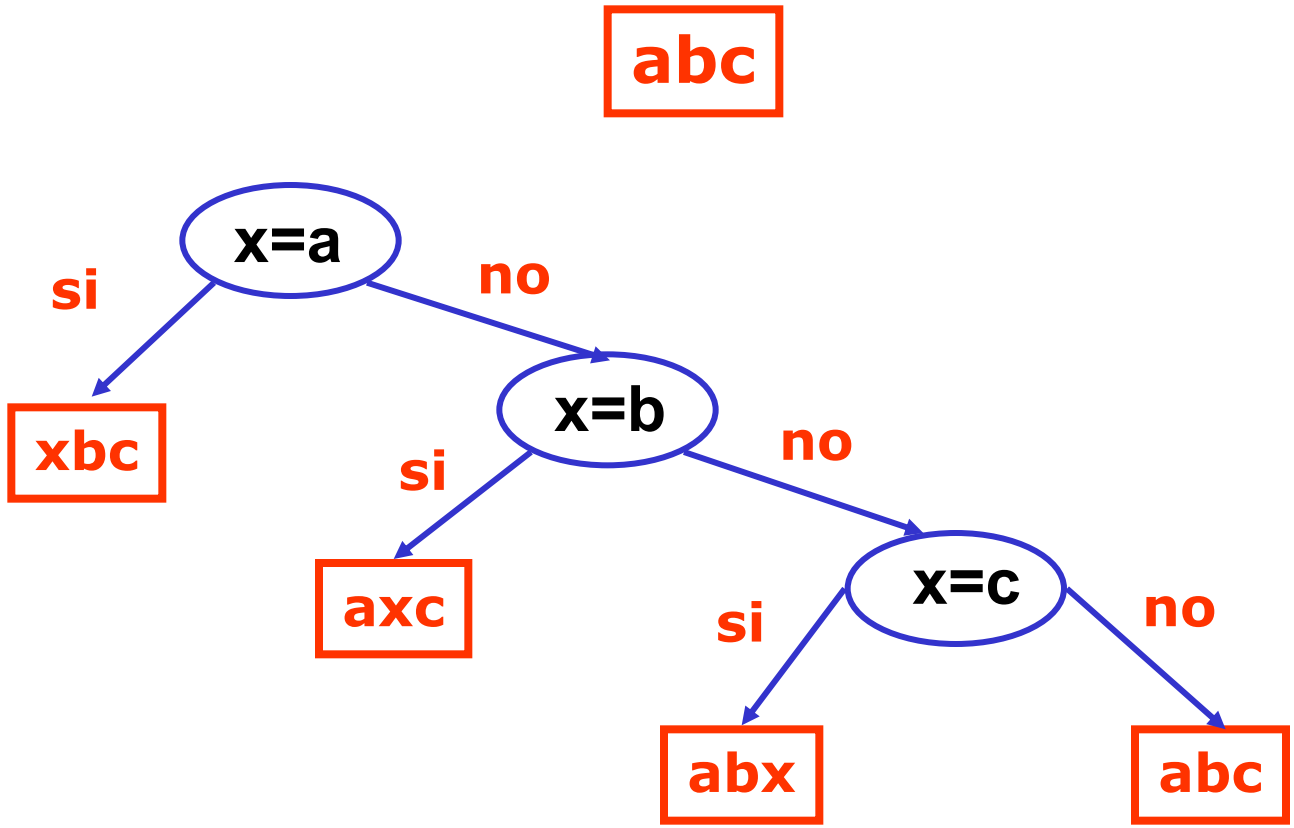
- **che hanno complessità proporzionale al numero di confronti che vengono effettuati durante l'esecuzione dell'algoritmo**

Limiti inferiori: alberi di decisione

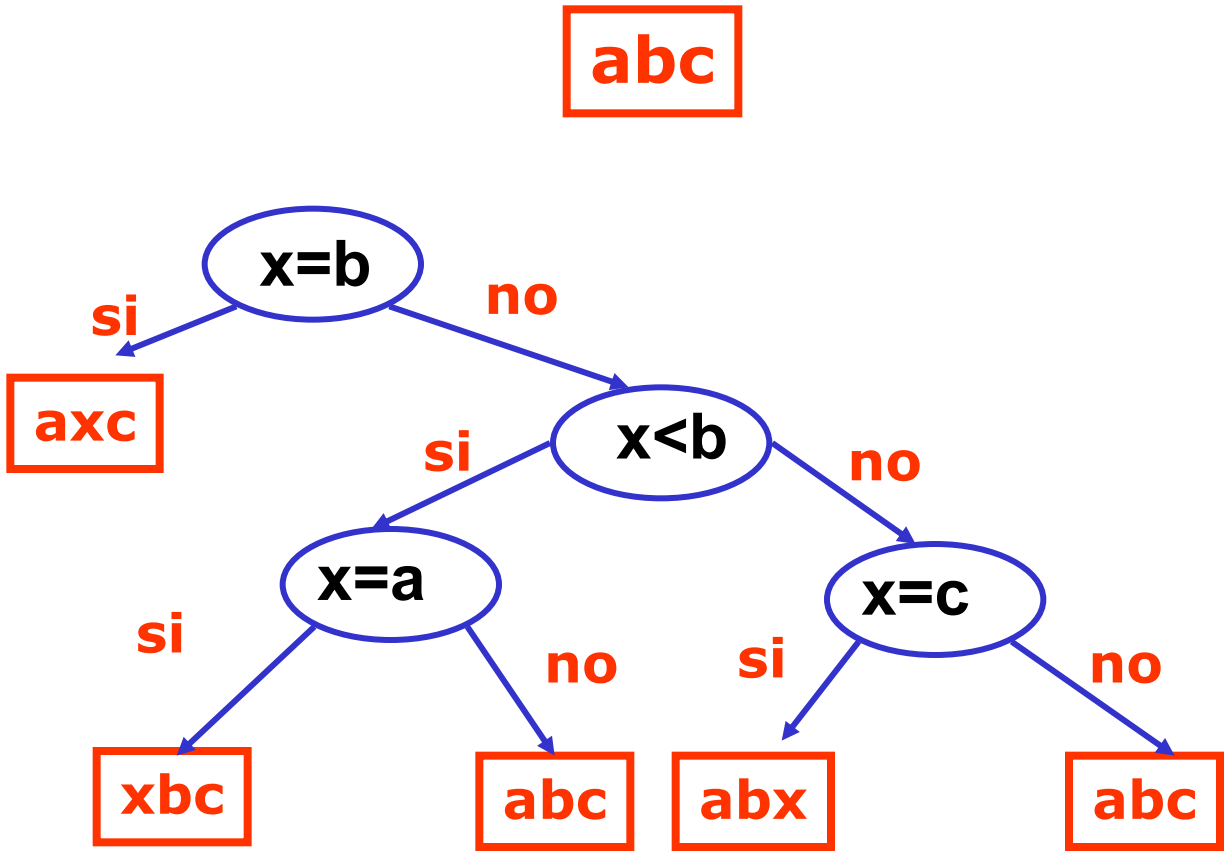
albero binario che corrisponde all'algoritmo:

- ogni foglia rappresenta una soluzione per un particolare assetto dei dati iniziali.
- ogni cammino dalla radice ad una foglia rappresenta una esecuzione dell'algoritmo (sequenza di di confronti) per giungere alla soluzione relativa alla foglia

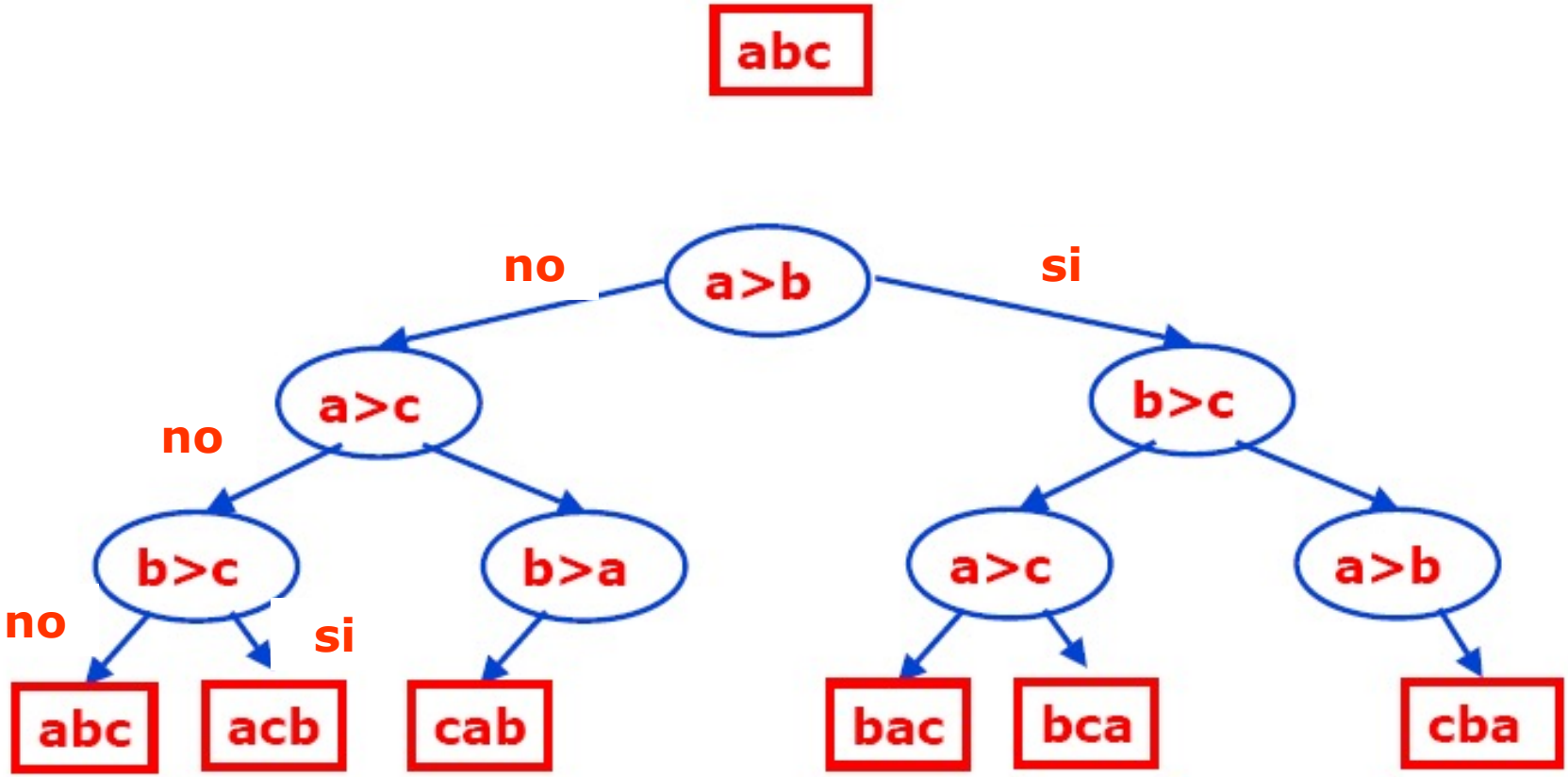
albero di decisione per la ricerca lineare



albero di decisione per la ricerca binaria



Albero del selection sort con 3 elementi



Limiti inferiori: alberi di decisione

Ogni algoritmo che risolve un problema che ha ***s*** soluzioni ha un albero di decisione corrispondente con almeno ***s*** foglie.

Fra tutti gli alberi di decisione per un particolare problema cha ***s*** soluzioni:

- l'albero di decisione che minimizza la ***lunghezza massima*** dei percorsi fornisce un ***limite inferiore*** al numero di confronti che un algoritmo che risolve il problema deve fare nel ***caso peggiore***.
- l'albero di decisione che minimizza la ***lunghezza media*** dei percorsi fornisce un limite inferiore al numero di confronti che un algoritmo che risolve il problema deve fare nel ***caso medio***

fatti

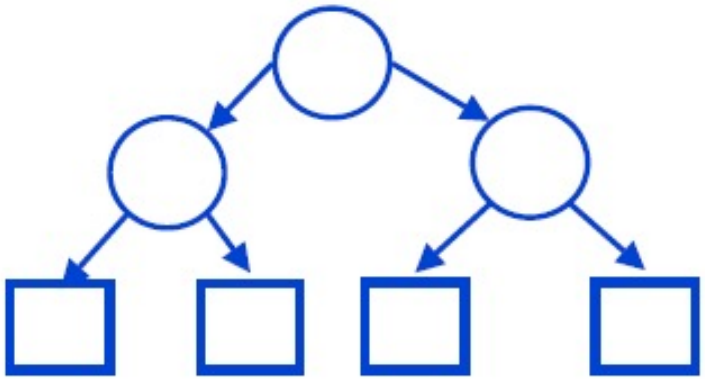
- Un albero binario con k livelli ha al massimo 2^k foglie

(ce l'ha quando è bilanciato)

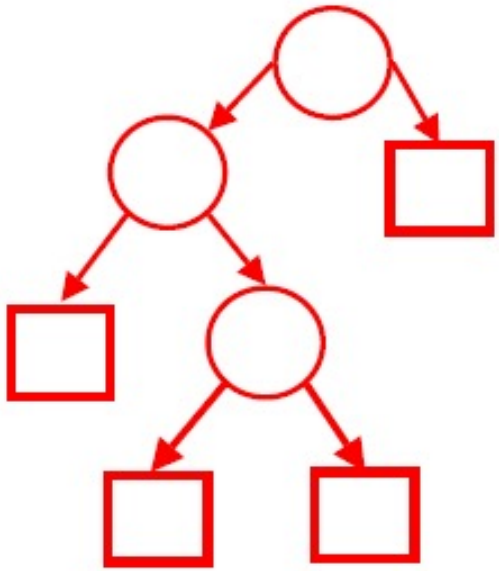
- Un albero binario con s foglie ha almeno $\log_2 s$ livelli

- Gli alberi binari bilanciati minimizzano sia il caso peggiore che quello medio: hanno log s(n) livelli.

Confronto fra algoritmi con 4 soluzioni



cammino max :2
cammino medio: 2



cammino max : 3
cammino medio: 2,25

(1+2+2*3)/4=9/4=2,25

algoritmi di ordinamento

Numero soluzioni : n!

$n! = (n/e)^n$

cammino medio e max: log (n!) ≅ nlogn

- Mergesort è ottimo
- Quicksort è ottimo nel caso medio
- Non sempre il limite è raggiungibile
  (la ricerca è Ω (logn))

Ordinamenti con complessità minore di O(nlogn)

- **counting sort**
- **radix sort**

counting sort

- **Ordina una sequenza di interi**
- **Si può usare quando si conoscono i valori minimo e massimo degli elementi da ordinare**
- **Per ogni valore presente nell'array, si contano gli elementi con quel valore utilizzando un array ausiliario avente come dimensione l'intervallo dei valori**
- **Successivamente si ordinano i valori tenendo conto dell'array ausiliario**

Esempio

A=

0	1	2	3	4	5	6	7	8	9	10	11	12	13
7	7	4	4	7	5	4	7	4	5	1	1	0	1

C=

0	1	2	3	4	5	6	7
1	3	0	0	4	2	0	4

n=14
minimo=0
massimo=7
possibili valori:8

0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	1	1	1	4	4	4	4	5	5	7	7	7	7

counting sort

```
void counting_sort(int A[], int k, int n)
  // 0 è il minimo, k il massimo, C array ausiliario
{   int i, j; int C[k+1];
   for (i=0; i<=k; i++) C[i] = 0;          // O(k)
   for (j=0; j<n; j++) C[A[j]] ++;         // O(n)
   j=0;
   for (i=0; i<=k; i++)                    // O(?)
       while (C[i]>0){
         A[j]=i;
         C[i]--;
         j++;
   }
```

counting sort

Non basato su confronti

Complessità O(n+k) (nel caso sopra citato O(n+8))

Conveniente quando k è O(n)

Necessaria memoria ausiliaria

radix sort

- **Ordina una sequenza di interi**
- **Si può usare quando si conosce la lunghezza massima (numero di cifre) d dei numeri da ordinare**
- **Si eseguono d passate ripartendo, in base alla d-esima cifra, i numeri in k contenitori, dove k sono i possibili valori di una cifra, e rileggendo il risultato con un determinato ordine**

Esempio di radix sort con cifre decimali

Numeri da ordinare (d=3, k=10): 190, 051, 054, 207, 088, 010

1° passata (O(n+k))

Si inseriscono i numeri nei contenitori in base al valore dell'ultima cifra (la meno significativa)

010									
190	051			054			207	088	
0	1	2	3	4	5	6	7	8	9

Si estraggono i numeri rileggendoli da sinistra a destra e dal basso verso l'alto:

190, 010, 051, 054, 207, 088

Esempio di Radix sort con cifre decimali

190, 010, 051, 054, 207, 088

2° passata (O(n+k))

Si inseriscono i numeri nei contenitori in base al valore della penultima cifra

					054				
207	010				051			088	190
0	1	2	3	4	5	6	7	8	9

Si estraggono i numeri rileggendoli da sinistra a destra e dal basso verso l'alto:

207, 010, 051, 054, 088, 190

Esempio di Radix sort con cifre decimali

207, 010, 051, 054, 088, 190

3° e ultima passata (O(n+k))

Si inseriscono i numeri nei contenitori in base al valore della prima cifra

088									
054									
051									
010	190	207							
0	1	2	3	4	5	6	7	8	9

Si estraggono i numeri rileggendoli da sinistra a destra e dal basso verso l'alto:

010, 051, 054, 088, 190, 207

Radix sort

- **Non basato su confronti**
- **E' fondamentale partire dalla cifra meno significativa**
- **La complessità è $O(d(n+k))$ dove d è la lunghezza delle sequenze e k è il numero dei possibili valori di ogni cifra (nel caso dell'esempio O(3(n+10)))**
- **Necessaria memoria ausiliaria**
- **Conveniente quando d è molto minore di n**
- **Si può usare per ordinare in ordine alfabetico sequenze di caratteri**

Pseudocodice Radix Sort

```
     procedura bucketSort(array A di n interi, interi b e t)
1.       sia Y un array di dimensione b
2.       for i = 1 to b do Y[i] ← lista vuota
3.       for i = 1 to n do
4.           c ← t-esima cifra di A[i] nella rappresentazione in base b
5.           appendi A[i] alla lista Y[c + 1]
6.       for i = 1 to b do
7.           copia ordinatamente in A gli elementi della lista Y[i]

     algoritmo radixSort(array A di n interi)
8.       t ← 0
9.       while (esiste un numero la cui t-esima cifra è ≠ 0)
10.          bucketSort(A, 10, t)
11.          t ← t + 1
```

Immagine estratta dal libro Demetrescu

Riferimenti Bibliografici

Demetrescu:

Capitolo 4

Cormen:

Capitolo 8

Esercizio

Dato un alfabeto di 5 lettere: a, b, c, d, e, ordinare con l'algoritmo radix sort le seguenti stringhe di 3 caratteri in ordine alfabetico (d=3, k=5, n=6), indicando i passi successivi eseguiti dall'algoritmo:

ace, ceb, bec, abc, eba, bba