Lezione 20 aprile

Monday, April 20, 2020 11:11 AM

Strutture dati dinamiche

- heap
- dizionario < tabelle hash
  < alberi ABR

array di dimensione variabile:

a array

d dimensione effettiva

$n \leq d$ numero di elementi effettivamente presenti in a

Inserzione se $n+1 \leq d$ ok

else raddoppiamo l'array portando la dimensione a $2d$ allocando una nuova array b in cui si copiano gli n el e si fa l'inserzione.

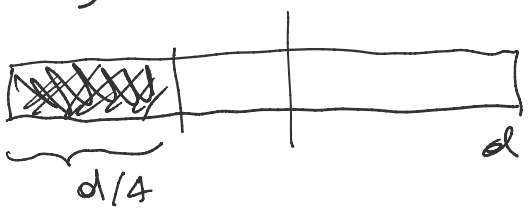
```
a = b;
```
$\Theta(n)$

Cancellazione: $n = n-1$; se $n = d/4$ si dimezza l'array.

```
d = d/2;
si alloca b di size d;
si copiano gli n elementi in b;
a = b;
```
$\Theta(n)$

$a = b$; $\quad 0 \quad \Theta(n)$

```
VerificaRaddoppio()
if (n+1 == d) {
  b = nuovo array (2*d);
  for (i=1; i<=n; i++) b[i] = a[i
  a = b;
  d = d*2;
  }   prima dell'inserzione
Verifica_dimezzamento(); dopo cancella.
if (d>1) && (n == d/4) {
   b = nuovoarray (d/2);
   for (i=1; i<=n; i++) b[i] = a[i];
   b = a;
   d = d/2;                    Θ(n)
}
```



Ins e canc. complessità $\leq$ $\Theta(\log n)$ hash
$\Theta(1)$ hash

complessità è $\Theta(n)$ caso pessimo

ma avviene una volta tanto

complessità ammortizzata

Teo: n operazioni di inserzione o cancellazione

complessiva ammortizzata

Teo: n operazioni di inserzione o cancellazione in un array dinamico richiedono $\Theta(n)$.

complessità ammortizzata è $\Theta(1)$

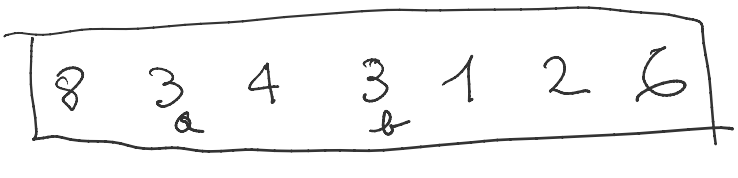
Prova: Occorrono $\Omega(n)$ op. per dimezzare o raddoppiare, a costo $O(n)$.

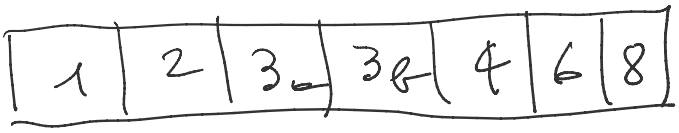
cfr. Grossi: 1.1.3

Ordinamento

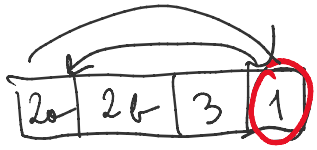
Algoritmo	caso pessimo	caso medio	caso ottimo	spazio	stabilità
Selection Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	cost	SI
Ins. Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	cost	SI
Merge Sort	$\Theta(n\log n)$	$\Theta(n\log n)$	$\Theta(n\log n)$	$\Theta(n)$	SI
QuickSort	$\Theta(n^2)$	$\Theta(n\log n)$	$\Theta(n\log n)$	? $\log n$ caso medio	NO
HeapSort	$\Theta(n\log n)$	$\Theta(n\log n)$	$\Theta(n\log n)$	$\Theta(\log n)$	NO

Un alg. di ordinamento è stabile se conserva (non modifica) l'ordine iniziale degli elementi di uguale valore

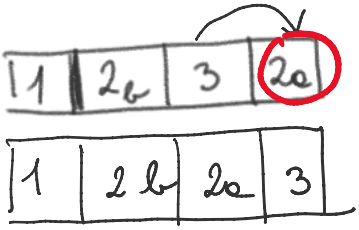




Quick Sort : algoritmo comune

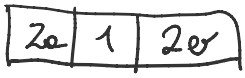
A : 

alla fine di Partition scambio del pivot col primo dei maggiori

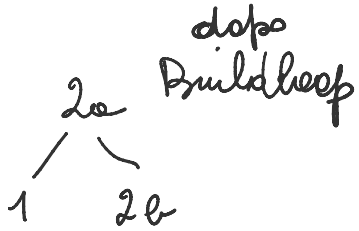


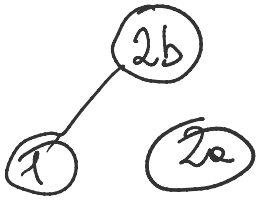
partition

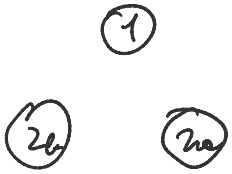
Heapsort

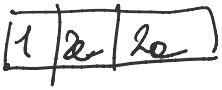
A 

input









Ogni alg. di ordinamento non stabile può essere reso stabile

(elemento, posizione)

si ordina su elemento, se posizione.

si ordina su elemento, sua posizione.

spazio: n elementi $\log n$ bit $\forall$ el.

$S(n) = \Theta(n \log n)$ tempo invariato

Sorting per confronti $\Omega(n \log n)$

- restrizioni sull'input } idee
- usiamo op. diverse } per migliorare

Ricerca $\Omega(\log n)$ ricerca per confronti

tabelle hash $\Theta(1)$ in media

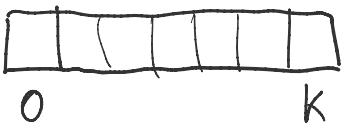
funzioni hash per trovare la posizione
+ confronti

SORTING IN TEMPO LINEARE

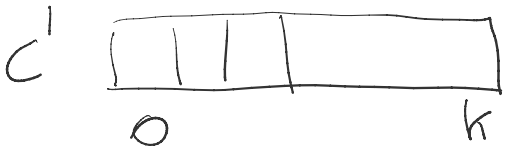
Algoritmo COUNTING_SORT

ipotesi sull'input:

$0 \leq |A[i]| \leq K$

C 

per ogni $0 \leq j \leq K$ quanti elementi $= j$ ci sono in A



numero = j ci sono in A

per ogni $0 \le j < K$

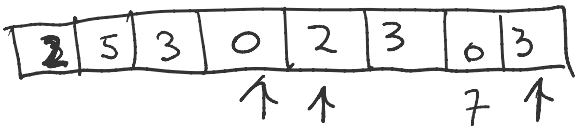
```
COUNTING_SORT (A, B, K):
"sia C[0..K] nuovo array";
Θ(K)   for (i = 0, i ≤ K, i++) C[i] = 0;
Θ(n    for (j = 1, j ≤ n, j++) C[A[j]]++;
```

```
Θ(K) for  i = 1; i ≤ K; i++)  C[i] = C[i] + C[i-1];
Θ(n) for (j = n, j ≥ 1, j--) {
        B[C[A[j]]] = A[j];
        C[A[j]]--;
     }
```

complessità: $\Theta(n) + \Theta(K) = \Theta(n+K)$

COUNTING_SORT è **STABILE**

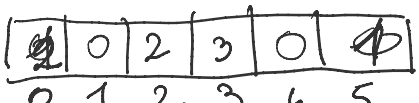
A



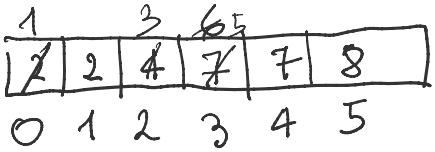
$|A[i]| \le 5$

$K = 5$

C



C'



B

0	0	2	2	3	3	3	5
1	2	3	4	5	6	7	8

$j = 8$ $A[j] = 3$ $C[A[j]] = C[3] = 7$ $B[7] = 3$

$j = 7$ $A[7] = 0$ $C[0] = 2$ $B[2] = 0$ $C[0] = 1$

$j = 6$ $A[6] = 3$ $C[3] = 6$ $B[6] = 3$ $C[3] = 5$

$j = 5$ $A[5] = 2$ $C[2] = 4$ $B[4]$

$C[2] = 3$

$j = 4$ $A[4] = 0$ $C[0] = 1$ $B[1] = 0$ $C[0] = 0$

$j = 3$ $A[3] =$ $C[3] = 5$ $B[5] = 3$

ipotesi: $|A[i]| = O(n)$, $1 \le i \le n$.

COUNTING SORT è $\Theta(n)$ tempo

conviene se k è $O(n \log n)$

se $k = O(n \log n)$ meglio Heapsort

→ spazio $= \Theta(n + k)$

COUNTING SORT è STABILE nella implementazione Cormen, non fa confronti tra elementi

RADIX_SORT

ordinamento cifra per cifra : d cifre

1) 329	720	720	329
2) 457	355	329	355
3) 657	436	436	436
4) 839 →	457 →	839 →	457
5) 436	657	355	657
6) 720	329	457	720
7) 355	839	657	839

si ordina a partire dalla cifra meno significativa

```
RADIX_SORT (A, d):
for(i=1 ; i ≤ d · i++ {

"Ordina A sulla cifra
i con ordinamento stabile"
}
```

ogni cifra decimale $0 \leq d_i \leq 9$

COUNTING SORT sulle singole cifre

$d \cdot \Theta(n+g)$

$\Theta(dn + \underbrace{gn}_{\uparrow}) = \Theta(dn)$

se $d = \log n$ non ci guadagniamo.