

Ricerca, ordinamento e fusione

Antonella Santone

Anno Accademico 2008/2009

1

Oppure ...

ricerca	<==>	search
ordinamento	<==>	sort
fusione	<==>	merge

2

Ricerca di un elemento

Determinare se un valore è presente nell'array

Due casi:

1. Array non ordinato
2. Array ordinato

3

Ricerca di un elemento

... in un array NON ORDINATO

4

Ricerca in un array non ordinato

```
/* dichiarazioni, int vett[N]; */  
/* input riempimento ed array */  
printf("Elemento da cercare: ");  
scanf("%d", &elem);  
i=0;  
trovato=0;  
while(!trovato && i<N)  
    if (elem == vet[i]) trovato=1;  
    else i++;  
if (trovato)  
    printf("Elemento trovato in pos. %d\n", i);  
else  
    printf("Elemento non presente \n");
```

5

Ricerca di un elemento

... in un array ORDINATO

6

Due algoritmi: array ordinato

❖ Ricerca lineare

❖ Ricerca binaria o logaritmica

7

Ricerca lineare

Se l'array è ordinato in senso crescente, non è necessario arrivare alla fine dell'array per stabilire che l'elemento non è stato trovato ...

Ci si può fermare appena si trova un elemento maggiore (o uguale) di quello dato ...

maggiore → non trovato!
uguale → trovato!

Ovviamente, se l'elemento è maggiore di tutti quelli presenti nell'array, allora si visiterà l'intero array (caso peggiore) ...

8

Esempio:

[4 6 7 10 12 15 18 20 24 30]

La ricerca di 13 termina quando l'elemento corrente è 15
→ fallimento



La ricerca di 15 termina quando l'elemento corrente è 15
→ successo



La ricerca di 40 termina quando si sono visitati tutti gli elementi dell'array

→ fallimento



9

Programma ricerca lineare

```
...
i = 0;
trovato = 0;
while(i < N && !trovato)
    if (a[i] >= elem)
        trovato = 1;
    else i++;
if(trovato && (a[i] == elem))
    printf("elemento trovato in posizione %d", i);
else printf("elemento non trovato");
```

trovato diventa 1
quando trovo un
elemento >= elem

mi fa uscire dal
ciclo

valutazione pigra

10

Analisi

```
while(i < N && !trovato)
    if (a[i] >= elem) trovato = 1;
    else i++;
if(trovato && (a[i] == elem)) printf("elemento tr...");
else printf("elemento non trovato");
```

trovato = 1 quando $a[i] \geq elem$

affinché l'elemento ci sia, deve essere vera la seguente espressione:

$trovato \ \&\& \ (a[i] == elem)$

nel valutare $trovato \ \&\& \ (a[i] == elem)$ potrebbe succedere (nel caso peggiore quando gli elementi dell'array sono tutti più piccoli di elem) che $i == N$. In questo caso $trovato=0$; poiché il C usa la valutazione pigra, non si valuta $a[N]$

11

Due algoritmi: array ordinato

❖ Ricerca lineare

❖ Ricerca binaria o logaritmica

12

Ricerca binaria o logaritmica

Dividere l'array in due metà e confrontare l'elemento da cercare con l'elemento centrale dell'array

- ❖ uguali → trovato (... e ci si ferma)
- ❖ elemento dell'array maggiore → continuare la ricerca nella prima metà dell'array
- ❖ elemento dell'array minore → continuare la ricerca nella seconda metà dell'array

13

Esempio: cercare 15

4	6	7	10	12	15	18	20	24	30
0	1	2	3	4	5	6	7	8	9

15	18	20	24	30
5	6	7	8	9

15	18
5	6

14

Perché detta logaritmica?

Nel caso peggiore si visitano $\log_2 n$ elementi

Esempio: cercare 80

5	7	8	10	17	20	25	30	33	36	38	50	60	70	80	90
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

33	36	38	50	60	70	80	90
8	9	10	11	12	13	14	15

4 (i.e. $\log_2 16$) confronti

60	70	80	90
12	13	14	15

Con la ricerca lineare:
15 confronti

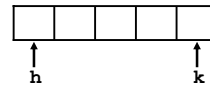
80	90
14	15

15

Algoritmo informale

Usiamo **elem**: elemento da cercare

Usiamo **h** e **k**: estremi dell'intervallo in cui cercare



Usiamo **trovato**: inizialmente è vale 0. Il suo valore sarà 1 se trovo elem

16

finché non trovo elem !trovato e ci sono ancora elementi (h <= k):

{
cerco l'elemento in posizione centrale $p = (h+k)/2$

4	6	7	10	12	15	18	20	24	30
↑0	1	2	3	4	5	6	7	8	9
h				p					k

se sono uguali (vett[p]==elem) per es.: elem=12 → successo
se è maggiore (vett[p]>elem) per es.: elem=7 → k = p-1

4	6	7	10
↑0	1	2	3
h			k

se è minore (vett[p]<elem) per es.: elem=20 → h = p+1

15	18	20	24	30
↑5	6	7	8	9
h				k

17

Programma

```

h = 0; k = N-1; // estremi dell'intervallo in cui
                // ricercare
trovato = 0; // inizialmente elem non è stato trovato
while (h <= k && !trovato)
{
    p = (h + k) / 2; // posizione centrale
    if (vet[p] == elem) trovato = 1;
    else if (vet[p] > elem)
        k = p-1; //la ricerca continua nella I metà
    else h = p+1; //la ricerca continua nella II metà
}
if (trovato)
    printf("Elemento trovato in posizione %d\n", p);
else
    printf("Elemento non trovato \n");
    
```

Cerchiamo 15

h k

4	6	7	10	12	15	18	20	24	30
0	1	2	3	4	5	6	7	8	9

h=0 → p=4 h=5 k=9

h=5 → p=7 h=5 k=6

h=5 → p=5 vett[p]=15 😊 successo

Numero di confronti = 3

Ricerca lineare
Numero di confronti = 6

19

Ordinamento

20

Ordinamento

Ordinare gli elementi di un vettore in ordine crescente o decrescente

Esistono molti algoritmi

Vediamo: **Selection sort**

21

Selection sort

Il vettore viene diviso in due parti da un indice (cursore) che scorre dalla prima alla penultima posizione. Ad ogni passo si cerca l'elemento minimo tra quelli della II parte e lo si mette nella posizione del cursore; quindi si fa avanzare il cursore di una posizione

22

Idea

Trovo il minimo nella parte NON ordinata 10

Scambio

23

Situazione iniziale

24

Algoritmo Selection Sort: informale

```
/* dichiarazioni int vett[N] */
/* input riempimento ed array */
for(i = 0; i < N-1; i++)
{
    Individua la posizione pmin dell'elemento minimo
    compreso tra le posizioni i e N-1 di vett

    Scambia gli elementi di posizione i e pmin
}

/* output array */
```

25

```
/* individua la posizione pmin del minimo */
```

```
min = vett[i];
pmin = i;
for (j = i+1; j < N; j++)
    if (vett[j] < min)
        { min = vett[j];
          pmin = j;
        }
```

26

```
/* scambia gli elementi di posizione i e pmin */
```

```
temp = vett[i];
vett[i] = vett[pmin];
vett[pmin] = temp;
```

27

```
#include<stdio.h>
#define N 7
main()
{ int vett[N] = {1,110,23,4,55,1,7};
  int i, j, min, pmin, temp;
  for(i = 0; i < N-1; i++)
  { min = vett[i];
    pmin = i;
    for (j = i+1; j < N; j++)
        if (vett[j] < min)
            { min = vett[j];
              pmin = j;
            }
    temp = vett[i];
    vett[i] = vett[pmin];
    vett[pmin] = temp;
  }
  // stampa il vettore ordinato
  for (i=0; i<N; i++) printf("%d\n", vett[i]);
}
```

Fusione

Fusione

Partendo da due array ordinati ne ricava un terzo anch'esso ordinato

La dimensione dei due array non necessariamente è la stessa

29

30

Idea

2	3	17	20	24	26	35
---	---	----	----	----	----	----

1	8	25
---	---	----

↑ i

↑ j

1	2	3							
---	---	---	--	--	--	--	--	--	--

↑ k

ecc.

31

```
...
int vet1[N], vet2[M], vet3[N+M];
int i=0, j=0, k=0;
do
{ if (vet1[i] <= vet2[j])
  {vet3[k] = vet1[i];
   i++;
   k++;
  }
  else {vet3[k] = vet2[j];
        j++;
        k++;
       }
} while(i < N && j < M);

if (i < N) for (; i < N; i++, k++) vet3[k] = vet1[i];
else for (; j < M; j++, k++) vet3[k] = vet2[j];
```