Strutture Dati

Lezione 17
Ricerca e ordinamento

Oggi parleremo di …

- Ricerca
  - sequenziale
  - binaria
- Ordinamento
  - selezione
  - inserzione
  - quicksort

La ricerca

- Un insieme di informazioni è memorizzato all'interno della memoria come lista
  - ciascun oggetto, detto record, è suddiviso in più unità elementari, chiamate campi
  - il campo particolare su cui si effettua una ricerca è detto chiave
  - la scelta della chiave dipende dalla particolare applicazione.
- L'efficienza di un metodo di ricerca dipende dalle ipotesi che si fanno sull'organizzazione dei record
  - per liste ordinate le ricerche sono molto efficienti (ricerca binaria)
  - per liste random bisogna ricorrere ad una ricerca sequenziale.

La ricerca sequenziale

- La ricerca inizia ad una estremità della lista.
- Si esamina ogni record sino a trovare il campo chiave richiesto o si raggiunge l'altra estremità della lista.

```
int ricseq(elemento lista[], int numric, int n)
{
 /* Ricerca numric in un array lista di n
    numeri.
    Ritorna i, se lista[i]=numric.
    Ritorna -1 se numric non e' nella lista
 */

 int i;

 lista[n] = numric;
 for(i = 0; lista[i].chiave!= numric;  i++)
        ;
 return((i < n) ? i : -1);
}
```

$O(n)$

La ricerca binaria

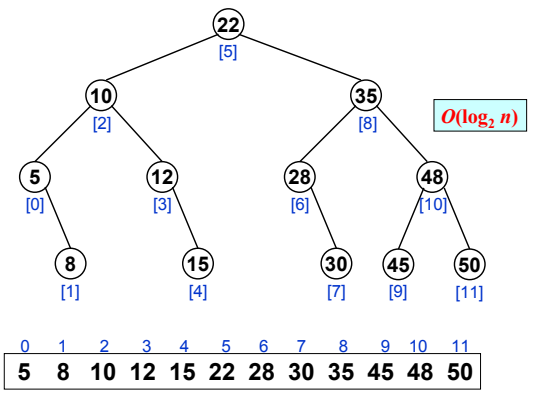
- La ricerca binaria ipotizza che la lista sia ordinata in base al campo chiave in modo che

lista[0].*chiave* ≤*lista*[1].*chiave* ≤…≤*lista*[n-1].*chiave*

```
int ricbin(elemento lista[], int numric, int n)
{
 int primo=0, ultimo=n-1, mezzo;
 while(primo<=ultimo) {
   mezzo = (primo + ultimo)/2;
   switch(CONFRONTA(lista[mezzo].chiave, numric)){
        case -1:  primo = mezzo + 1;
                  break;
        case 0:    return mezzo;
        case 1:    ultimo = mezzo -1;
      }
 }
 return -1;
}
```

$O(\log_2 n)$

Albero decisionale



Algoritmi di ordinamento

- Algoritmi semplici
  - selezione
  - inserzione
- Algoritmi complessi
  - veloce (quicksort)
  - per fusione (mergesort)
  - mediante heap (heapsort)
  - con radice (radixsort).

I dati da ordinare sono numeri interi.

Ordinamento per selezione

- Per $i = 0,\ldots, n-2$
  - si esamina la *lista* da i a $n-1$ per trovare il numero intero più piccolo, sia *lista*[*min*];
  - si scambia *lista*[i] con *lista*[*min*].

```
void selezione(int lista[], int n)
{
 int i,j,min;

 for(i=0; i<n-1; i++)
  {
    min = i;
    for(j=i+1; j<n; j++)
       if(lista[j] < lista[min])
                  min = j;
    SWAP(&lista[i], &lista[min]);
  }
}
```

$O(n^2)$

Ordinamento per inserzione

- E' il metodo usato dai giocatori per ordinare le carte.
- Si considera un elemento alla volta da sinistra a destra
  - si inserisce ciascun elemento al proprio posto tra quelli già considerati, mantenendoli ordinati.
- L'elemento considerato viene inserito nel posto rimasto vacante in seguito allo spostamento di un posto a destra degli elementi più grandi.

Ordinamento per inserzione

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

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

Ordinamento per inserzione

```
void inserzione(int lista[], int n)
{
  int i,j;
  int prossimo;

  for(i = 0; i < n; i++)
    {
     prossimo = lista[i];
     for(j = i-1; j >= 0  &&  prossimo < lista[j];  j--)
                 lista[j+1] = lista[j];
    lista[j+1] = prossimo;
    }
}
```

Complessità nel caso peggiore: $O(n^2)$

Complessità nel caso migliore: $O(n)$

Ordinamento veloce (quicksort)

- Vantaggi
  - richiede in media $O(n\log_2 n)$ operazioni
  - facile da implementare.
- Svantaggi
  - richiede $O(n^2)$ operazioni nel caso peggiore
  - ricorsivo.

Ordinamento veloce (quicksort)

- È basato sulla metodologia Divide et Impera:
- Dividi: L'array A[p…u] viene "partizionato" (tramite spostamenti di elementi) in un elemento A[q] e due sottoarray A[p…q-1] e A[q+1…u] in cui
  - ogni elemento di A[p…q-1] è minore di A[q]
  - ogni elemento di A[q+1…u] è maggiore di A[q].
- Conquista: i due sottoarray A[p…q-1] e A[q+1…u] vengono ordinati ricorsivamente con quicksort.
- Combina: i sottoarray vengono ordinati anche reciprocamente, quindi non è necessaria alcuna combinazione. A[p…u] è già ordinato.

Ordinamento veloce (quicksort)

- L'array A[p…u] viene "suddiviso" intorno ad un elemento perno in due sottoarray A[p…q-1] e A[q+1…u] in cui ogni elemento di A[p…q-1] è minore di ogni elemento di A[q+1…u] :
  - l'algoritmo sceglie un valore dell'array che fungerà da elemento "spartiacque" tra i due sottoarray, detto valore pivot.
  - sposta i valori maggiori del pivot verso l'estremità destra dell'array e i valori minori verso quella sinistra.
- q dipenderà dal valore del pivot: sarà l'indice della posizione in cui si troverà alla fine l'elemento pivot.
- Solitamente si sceglie come pivot il primo elemento dell'array.

Ordinamento veloce (quicksort)

```
Quicksort(A,p,u)
 IF p < u
  THEN q = Partiziona(A,p,u)
       Quicksort(A,p,q - 1)
       QuicksSort(A,q + 1,u)
```

L'elemento scelto come pivot è il primo elemento dell'array, cioè A[p].

La partizione sull'array A[p,…,u] produce la permutazione seguente:

A[p … q-1], A[q], A[q+1 … u]

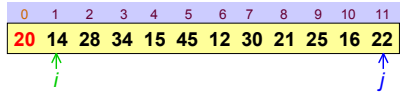
< pivot | pivot | > pivot

Ricorsivamente si ordinano gli elementi da p a q-1 e da q+1 ad u.

Chiave dell'algoritmo è la partizione dell'array intorno al pivot!

Ordinamento veloce (quicksort)

Partizione

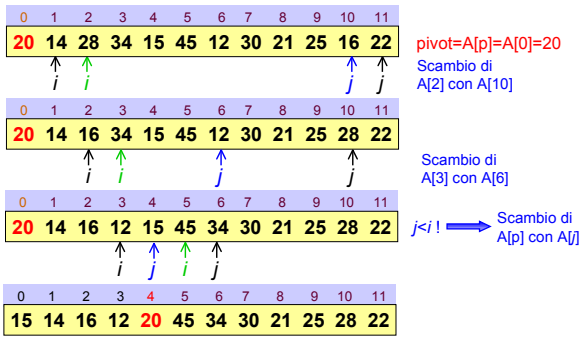


p=0, u= n-1 = 11
pivot=A[p]=A[0]=20

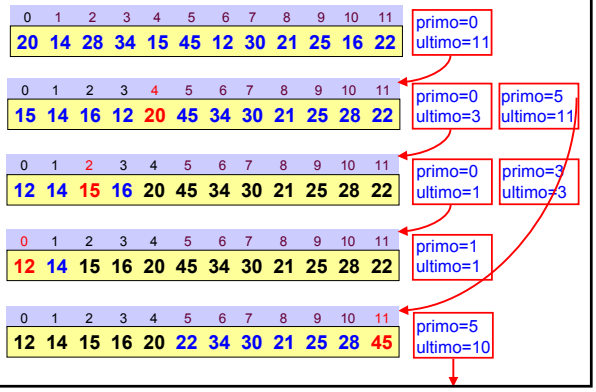
- Si scorre l'array con due indici *i* e *j*
  - *i* punta all'elemento dopo il perno e si sposta verso destra
  - *j* punta all'ultimo elemento e si sposta verso sinistra.
- Il movimento dei due cursori si ferma non appena A[p]<A[*i*] e A[p]>A[*j*].
- Si scambiano di posto A[*i*] e A[*j*] e si riprende il moto dei cursori.
- Si itera il procedimento finché i due cursori si incontrano in una posizione intermedia.
- Si scambia A[p] con l'elemento minore più a destra, A[*j*].

Ordinamento veloce (quicksort)

Partizione



Ordinamento veloce (quicksort)



Ordinamento veloce (quicksort)