Strutture Dati

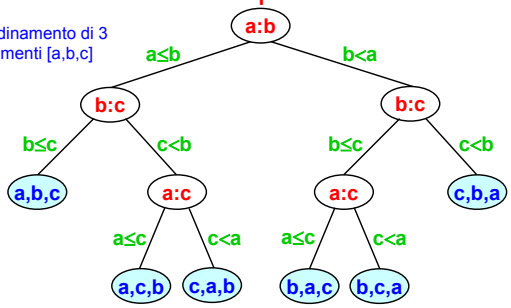
Lezione 19
Mergesort

Oggi parleremo di …

- Limite inferiore per l'ordinamento
- Mergesort
  - algoritmo
  - implementazione
    - iterativa
    - ricorsiva
  - analisi della complessità

Albero decisionale per l'ordinamento

Ordinamento di 3 elementi [a,b,c]



Il numero delle possibili soluzioni è 3!=6.
L'albero non è un albero binario completo di altezza 4: ha meno di $2^{4-1} = 8$ nodi terminali.
L'albero decisionale ha un numero di foglie sufficienti per ordinare 3 elementi.

Limite inferiore per l'ordinamento

- Ogni albero decisionale che ordina n elementi distinti ha un'altezza almeno pari a $\log_2(n!)+1$
  - ordinando n elementi, si hanno $n!$ possibili soluzioni
  - ogni albero decisionale deve avere almeno $n!$ foglie
  - se k è l'altezza dell'albero, poiché il numero massimo di foglie di un albero binario è 2^{k-1}, k deve essere tale che $2^{k-1} \geq n!$
  - l'altezza k deve essere maggiore o uguale a $\log_2(n!)+1$
- Qualsiasi algoritmo di ordinamento deve avere un tempo di $\Omega(n \log n)$
  - nell'albero decisionale esiste un percorso di lunghezza $\log_2(n!)$
  - $n! = n(n-1)(n-2) \ldots (3)(2)(1) \geq (n/2)^{(n/2)}$
  - $\log_2(n!) \geq (n/2) \log_2(n/2) \in O(n \log n)$

Ordinamento per fusione (mergesort)

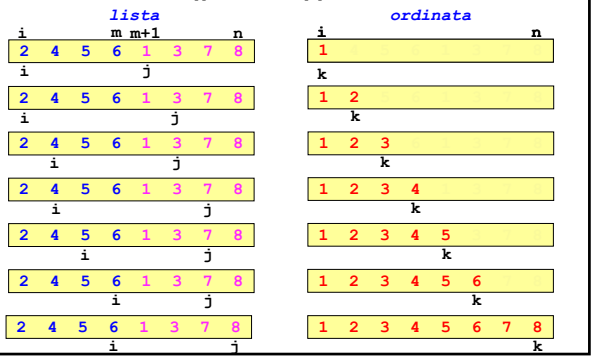
- Il limite inferiore al numero di confronti generato con l'albero di decisione è $n \log n$
  - il quicksort richiede $O(n \log n)$ confronti nel caso medio: è un algoritmo ottimo in questo caso
  - il quicksort richiede $O(n^2)$ confronti nel caso peggiore.
- La possibilità di raggiungere il limite inferiore dipende dalla partizione dell'insieme, bilanciata in ogni caso!
- L'ordinamento per fusione raggiunge tale limite, operando sempre su insiemi di uguale (o quasi) dimensione.

Ordinamento per fusione (mergesort)

- Suddividere l'insieme in piccoli gruppetti.
- Ordinare i gruppetti separatamente (con un metodo qualsiasi).
- Prendere il primo ed il secondo gruppetto e fonderli per formare un nuovo gruppo ordinato.
- Ripetere l'operazione con tutti gli altri gruppetti presi a coppie, fino ad ottenere un numero di gruppi ordinati pari a metà dei gruppetti originari.
- Fondere i nuovi gruppi a coppie e procedere similmente sino ad ottenere due gruppi ordinati contenenti ciascuno metà degli elementi, che vengono ora fusi per ottenere l'ordinamento.
- Il cuore del metodo di ordinamento è la fusione.

Mergesort: la fusione

Fusione di due liste ordinate *lista*[*i*], …, *lista*[*m*] e *lista*[*m*+1], …,*lista*[*n*] in una sola lista ordinata *ordinata*[*i*], …, *ordinata*[*n*]



Mergesort: la fusione

```
void merge(elemento lista[], elemento ordinata[], int i, int m, int n)
{
 /* fonde due file ordinati: lista[i],...,lista[m]
    e lista[m+1],...,lista[n], per ottenere una sola lista ordinata
    ordinata[i],...,ordinata[n]    */

 int j, k, t;

 j = m+1;        // indice per la seconda lista
 k = i;          // indice per la lista ordinata

 while(i<=m && j<=n)
   {
    if(lista[i]<=lista[j])  ordinata[k++] = lista[i++];
    else                    ordinata[k++] = lista[j++];
   }

 if(i>m)  for(t=j; t<=n; t++)   ordinata[k+t-j] = lista[t];
                  // ordinata[k],...,ordinata[n] = lista[j],...,lista[n]

 else     for(t=i; t<=m; t++)   ordinata[k+t-i] = lista[t];
                  // ordinata[k],...,ordinata[n] = lista[i],...,lista[m]
}
```

$O(n)$

Mergesort: versione iterativa

- Siano date n liste ordinate, ciascuna di lunghezza 1.
- Fondere le liste a coppie per ottenere $n/2$ liste ordinate di dimensione 2.
- Fondere le $n/2$ liste a coppie e così via sino ad ottenere un'unica lista.

Mergesort: versione iterativa

14 12 5 6 10 3 7 8 4 2	n=10
14 12 5 6 10 3 7 8 4 2	lungh = 1
12 14 5 6 3 10 7 8 2 4	
12 14 5 6 3 10 7 8 2 4	lungh = 2
5 6 12 14 3 7 8 10 2 4	
5 6 12 14 3 7 8 10 2 4	lungh = 4
3 5 6 7 8 10 12 14 2 4	
3 5 6 7 8 10 12 14 2 4	lungh = 8
2 3 4 5 6 7 8 10 12 14	

Mergesort: versione iterativa

Singolo passo di fusione: fusione di due file adiacenti di dimensione `lungh`

```
void passo_merge(elemento lista[], elemento ordinata[], int n, int lungh)
{
 /*      Svolge un passo dell'ordinamento per fusione.
         Fonde una coppia di sottofile adiacenti da lista a ordinata;
         n è il numero di elementi nella lista;
         lungh è la lunghezza del sottofile   */

   int i, j;

   for(i = 0; i <= n-2*lungh;  i += 2*lungh)
                merge(lista, ordinata, i, i+lungh-1, i+2*lungh-1);
   if(i+lungh < n)
                merge(lista, ordinata, i, i+lungh-1, n-1);
   else  for(j = i; j < n; j++)
                ordinata[j] = lista[j];
}
```

$O(n)$

Mergesort: versione iterativa

Algoritmo di ordinamento per fusione di una lista di interi rappresentata mediante array: mergesort(lista, n)

```
void mergesort(elemento lista[], int n)
{
 /* ordina la lista lista[0],...,lista[n-1] per fusione  */

 int lungh = 1;   // lunghezza corrente delle liste da fondere
 elemento extra[MAX];

 while(lungh < n)
  {
    passo_merge(lista, extra, n, lungh);
    lungh *= 2;
    passo_merge(extra, lista, n, lungh);
    lungh *= 2;
  }
}
```

Il numero totale di passi per ordinare n elementi è $\lceil \log n \rceil$.
La fusione richiede un tempo lineare.

La complessità di mergesort è $O(n \log n)$

Mergesort: versione ricorsiva

Algoritmo di ordinamento per fusione di una lista di interi rappresentata mediante array:
mergesort(lista, 0, n-1)

```
void mergesort(elemento lista[], int lower, int upper)
{
 /* ordina la lista lista[lower],...,lista[upper]   */

 int medium;

 if(lower < upper)
   {
    medium = (lower+upper)/2;
    mergesort(lista, lower, medium);
    mergesort(lista, medium+1, upper);
    merge(lista, lower, medium, upper);
   }
}
```

Mergesort: versione ricorsiva

- La complessità del mergesort è descritta dalla seguente relazione di ricorrenza:

$$T(n) \le cn + 2T(n/2) \qquad \text{con } T(0) = 0$$

La complessità di mergesort è $O(n \log n)$

↓

Il **mergesort** è un algoritmo di ordinamento **ottimo**

Mergesort: versione ricorsiva

Algoritmo di ordinamento per fusione di una lista di interi rappresentata mediante lista concatenata:

- il campo di collegamento è un numero intero e non un puntatore.
- il campo link in ogni record è posto inizialmente a –1.

```
typedef struct{
     int chiave;
     int link;
        } elemento;
```

	0	1	2	3	4	5	6	7	8	9	
Lista iniziale	14	12	5	6	10	3	7	8	4	2	chiave
	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	link

	0	1	2	3	4	5	6	7	8	9	
Lista ordinata	14	12	5	6	10	3	7	8	4	2	chiave
	-1	0	3	6	1	8	7	4	2	5	link

start=9

Mergesort: versione ricorsiva

Algoritmo di ordinamento per fusione di una lista rappresentata mediante concatenamenti:
start = rmerge(lista, 0, n-1);

```
int rmerge(elemento lista[], int lower, int upper)
{
 /* Ordina la lista lista[lower],...,lista[upper]
    Il campo link in ogni record è posto inzialmente a -1 */

 int medium;

 if(lower >= upper)  return lower;
 else
   {
     medium = (lower+upper)/2;
     return  mergelista(lista, rmerge(lista,lower,medium),
                                rmerge(lista,medium+1,upper));
   }
}
```

Mergesort: versione ricorsiva

Algoritmo di fusione di due liste rappresentate mediante concatenamenti.

```
int mergelista(elemento lista[], int prima, int seconda)
{
// fonde le liste puntate da prima e seconda

   int start = n;
   while (prima!=-1 && seconda!=-1)
          if(lista[prima].chiave <= lista[seconda].chiave)  {
               /*  chiave nella prima lista è inferiore allora collega questo elemento a
                   start e cambia start in modo da puntare a prima    */
               lista[start].link = prima;
               start = prima;
               prima = lista[prima].link;
               }
          else {
              /* chiave nella seconda lista è inferiore allora collega questo elemento a
                 start e cambia start in modo da puntare a seconda    */
               lista[start].link = seconda;
               start = seconda;
               seconda = lista[seconda].link;
              }
       /* sposta il resto */
   if(prima == -1)  lista[start].link = seconda;
   else             lista[start].link = prima;
   return lista[n].link;               // inizio della nuova lista
}
```