La Ricorsione

Salvatore Venticinque

Università degli Studi della Campania *Luigi Vanvitelli*

Agenda

- Approccio alla Progettazione ricorsiva
- La Torre di Hanoi
- Tipi di Ricorsione
  - Ricorsione Lineare
  - Ricorsione Multipla
  - Ricorsione Mutua
  - Ricorsione Annidata
- Esercizi

Un algoritmo viene detto ricorsivo quando nel suo corpo richiama se stesso, direttamente o indirettamente.

- **Ricorsione Diretta**: la procedura invoca direttamente nel suo corpo se stessa

  ```
  int f(int x) { …. f(x) …. }
  ```

- **Ricorsione indiretta:** la procedura invoca un’altra procedura che a sua volta chiama la procedura originaria

  ```
  int f(int x) { …. g(x) …. }
  int g(int x) { …. f(x) …. }
  ```

Sia ***P*** una proprietà (espressa da una frase o una formula che contiene la variabile ***n*** che varia sui numeri naturali). Supponiamo che:

- ***P(k)*** sia vera per ***k*** = 1, (***Base dell'induzione***),
- che ***P*** sia vera per un valore generico ***n*** (***Ipotesi Induttiva***),

se a partire dalla verità di ***P(n)*** riusciamo a dimostrare la verita di ***P(n + 1)***, allora ***P(k)*** è vera per qualsiasi valore di ***k***

Approccio alla Progettazione ricorsiva

- **Divide:** a partire dal problema da risolvere **P** sui dati di ingresso **d** si individuano ***k*** problemi del medesimo tipo di quelli originario, ma aventi dimensioni più piccole, immaginando delle opportune divisioni di **d**. La suddivisione va ripetuta fino a problemi tali da conoscerne la soluzione; tali casi vengono detti casi base.
- **Impera**: si ritiene, per ipotesi, di essere in grado di risolvere ciascuno dei sotto problemi in cui si è decomposto **P** correttamente e di conoscerne la soluzione.
- **Combina**: a partire dalle soluzioni, ritenute corrette, dei sotto problemi in cui si è diviso **P**, si costruisce la soluzione corretta al problema **P** stesso.

- **Ricorsione Lineare:** Un algoritmo si dice ricorsivo lineare se nel suo corpo e presente una sola chiamata a se stesso.

- **Ricorsione in coda:** Un algoritmo ricorsivo si dice che e ricorsivo in coda se la chiamata ricorsiva e l'ultima istruzione dell'algoritmo stesso.

- Domanda:

```
long factorial(int n) {
    /* Caso base */
    if (n == 0)
        return 1;

    else
    /* Fasi di divide, impera e combina */
        return factorial(n - 1) * n;
}
```

- E’ ricorsiva in coda o no?

- **Divide**: Il vettore di ingresso V [first::last] viene diviso considerando separatamente l'elemento che occupa la prima posizione V [first] ed il vettore V [first+1::last] costituito dai rimanenti elementi.

- **Caso base**: Quando si giunge, per divisioni induttive, ad un vettore che contiene un solo elemento, il problema ammette una soluzione banale, essendo il minimo eguale all'unico elemento presente nel vettore stesso.

- **Impera**: Si ritiene, per ipotesi induttiva, che si sappia risolvere correttamente il problema della ricerca del minimo nel vettore V [first + 1::last]; si indichi con Minfirst+1;last la relativa soluzione, ovvero il minimo calcolato in V [first +1::last].
- **Combina**: Sulla base del valore del primo elemento del vettore V [first] e del minimo calcolato sugli elementi dal secondo in poi, ovvero Minfirst+1;last, si calcola il minimo sull'intero vettore V , dato dal minimo tra i due, ovvero:
  - Minfirst;last = minimo(V [first];Minfirst+1;last)

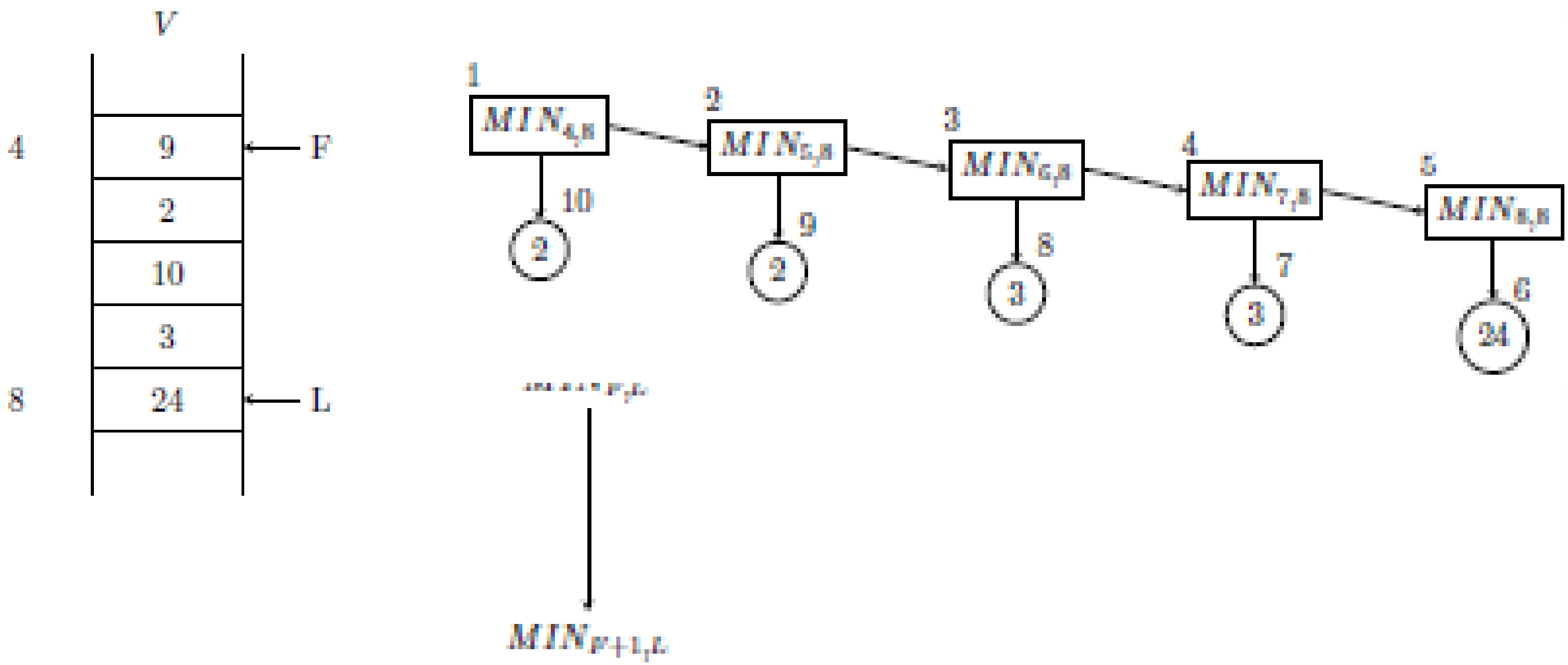
```c
int min_search_rec(int v[], int first, int last) {
    int ris;

    /* Caso Base */
    if (first == last)
        return (first);

    /* Divide e Impera */
    ris = min_search_rec(v, first + 1, last);

    /* Combina */
    if (v[ris] < v[first])
        return ris;

    else
        return first;
}
```



$$MIN_{F,L} = \min(V[F], MIN_{F+1,L})$$

- **Ricorsione Multipla:** Un algoritmo si dice a ricorsione multipla se nel suo corpo sono presenti piu chiamate a se stesso.

- Un caso semplice e molto frequente di ricorsione multipla e quella detta binaria che si presenta quando sono presenti due sole chiamate ricorsive.

Serie di Fibonacci

$$Fib(n) = \begin{cases} 0 & \text{per } n = 0 \\ 1 & \text{per } n = 1 \\ Fib(n-2) + Fib(n-1) & \text{per } n \geq 2 \end{cases}$$

```c
long fibonacci(long n) {
    /* Casi base */
    if (n == 0 || n == 1)
        return n;

    else
    /* Fasi di divide, impera e combina */
        return fibonacci(n - 1) + fibonacci(n - 2);
}
```

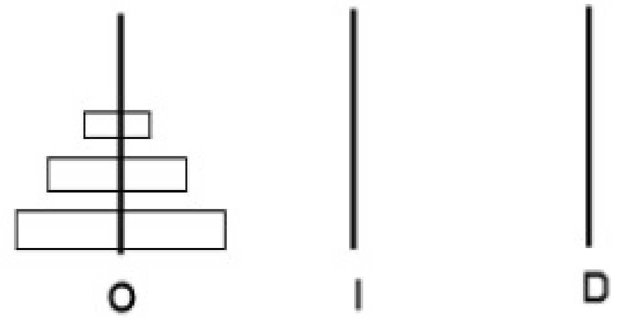
Il gioco si compone di tre pioli:

O (per Origine),

D (per Destinazione)

I (sta per Intermedio),

su cui sono disposti n dischi di diametro diverso.



I dischi sono inizialmente disposti sul piolo O con ordine crescente di diametro dall'alto verso il basso.

Mossa Lecita: Muovere un solo disco alla volta, dalla cima di un piolo e depositarlo su un piolo vuoto o contenente un disco di diametro superiore.

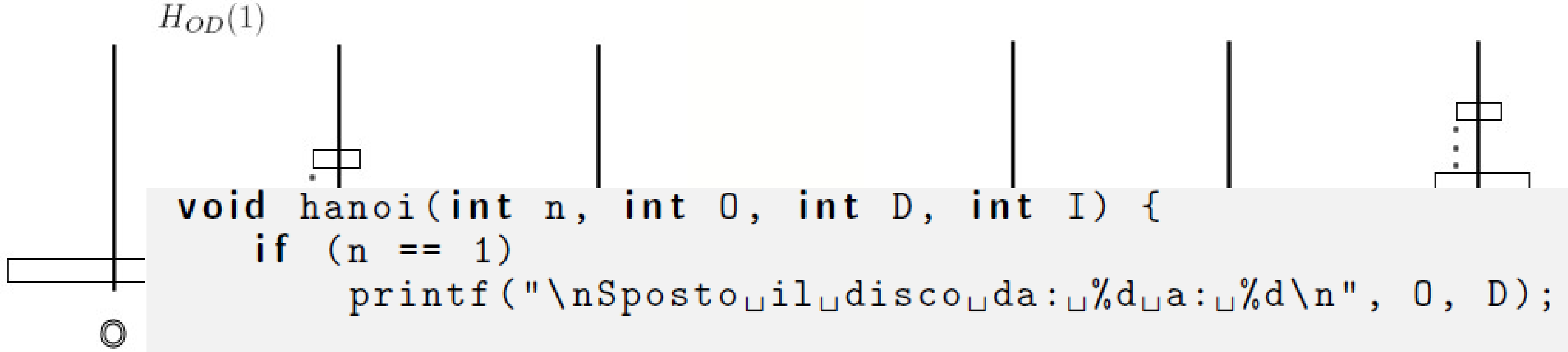
Obiettivo: portare nello stesso ordine, tutti i dischi nel piolo D.

- **Divide**: si riconduce il generico problema di Hanoi H(n) a problemi di Hanoi con n-1 Questa divisione permette, tra l'altro, di giungere per successive riduzioni al caso base rappresentato dal problema con un solo disco H(1), facilmente risolvibile in quanto prevede un'unica mossa.

- **Caso base**: il problema banale di Hanoi con un solo disco HX;Y (1), tra due generici pioli X ed Y , si risolve muovendo il disco da X a Y .

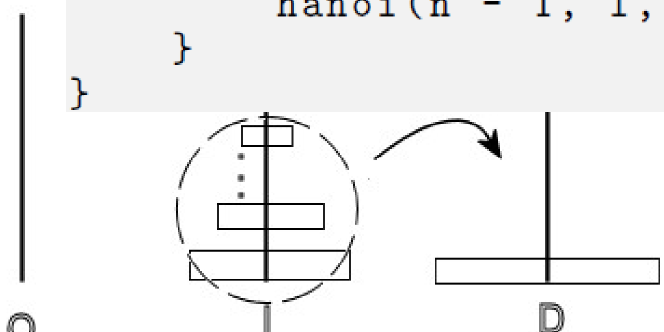
- **Impera**: si ritiene, per ipotesi, di saper risolvere correttamente il medesimo problema H(n-1) con n-1 dischi, e di conoscere quindi la sequenza corretta di mosse.

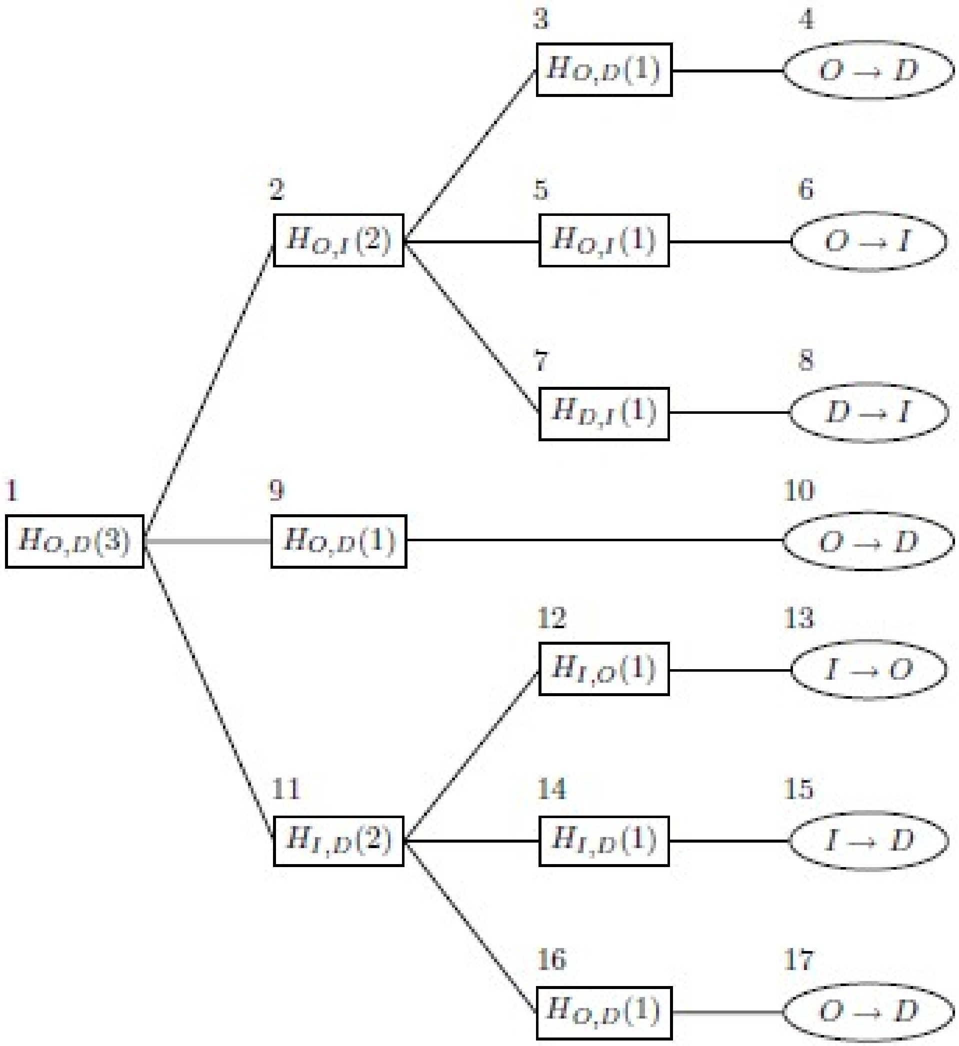
- **Combina**: e facile verificare che nella ipotesi di saper risolvere $H_{X;Y}$ (n-1), la soluzione ad $H_{O;D}$(n) e data dalla sequenza $H_{O;I}$ (n-1) $H_{O;D}$(1), $H_{I;D}$(n-1)




```c
void hanoi(int n, int O, int D, int I) {
    if (n == 1)
        printf("\nSposto␣il␣disco␣da:␣%d␣a:␣%d\n", O, D);

    else {
        hanoi(n - 1, O, I, D);
        hanoi(1, O, D, I);
        hanoi(n - 1, I, D, O);
    }
}
```





1
$H_{O,D}(3)$
2
$H_{O,I}(2)$
3
$H_{O,D}(1)$
4
$O \to D$
5
$H_{O,I}(1)$
6
$O \to I$
7
$H_{D,I}(1)$
8
$D \to I$
9
$H_{O,D}(1)$
10
$O \to D$
11
$H_{I,D}(2)$
12
$H_{I,O}(1)$
13
$I \to O$
14
$H_{I,D}(1)$
15
$I \to D$
16
$H_{O,D}(1)$
17
$O \to D$

- **Mutua Ricorsione:** Un algoritmo si dice a ricorsione mutua se e composto da una prima funzione che al suo interno ne chiama una seconda che a sua volta richiama la prima.

- **Esercizio:** Sviluppare una funzione che dato un intero stabilisce se questo è pari o dispari utilizzando un approccio ricorsivo.

- **Divide**: Un numero è dispari se il decremento a uno del numero è pari e viceversa.
- **Caso base**: il numero 0 è pari per convenzione
- **Impera**: assumiamo per ipotesi induttiva che sappiamo verificare se un numero n è pari
- **Combina**: se chiedi se un numero è pari allora verifica che il decremento è dispari. Se chiedi se un numero è dispari verifica che NON sia pari.

- **Esercizio:** Sviluppare una funzione che dato un intero stabilisce se questo è pari o dispari utilizzando un approccio ricorsivo.

```
int is_even(unsigned int n) {
    if (n == 0)
        return 1;
    else
        return (is_odd(n - 1));
}
```

```
int is_odd(unsigned int n) {
    return (!is_even(n));
}
```

- **Ricorsione Annidata:** Un algoritmo si dice a ricorsione annidata se e composto da una funzione che ha come argomento una chiamata alla funzione stessa

```
int ackermann(int m, int n) {
  if (m < 0 || n < 0)
     return -1; /* la funzione non e' definita per interi negativi! */

  if (m == 0)
     return n+1;
  else
     if (n == 0)
        return ackermann(m-1,1);
     else
        return
           ackermann(m-1,ackermann(m,n-1));
}
```

ESERCIZI

- Dato un vettore di interi v[First..Last], scrivere una ***funzione ricorsiva lineare*** che inverta il vettore v. La funzione non restituisce alcun valore.

- Prototipo:

```
void Inversione ( int v[], int First , int Last );
```

- **Divide**: il vettore di ingresso v[rst..last] viene diviso in tre parti: il l'elemento di posizione rst, l'elemento di posizione last ed il vettore v[rst+1..last-1].
- **Caso base**: se il vettore ha un solo elemento, oppure nessun elemento.
- **Impera**: si ritiene per ipotesi induttiva che si sappia risolvere correttamente il problema dell'inversione del vettore v[rst+1..last-1].
- **Combina**: si scambia l'elemento di posizione rst con quello di posizione last perche la parte centrale del vettore e già invertita per ipotesi

Inversione di un vettore
Ricorsivo

```
/* Inverte un vettore v dato in ingresso
 * VALORE DI RITORNO: il vettore invertito
*/
     void reverse(int v[], int First, int Last) {
    int tmp;
    if (First < Last)
         {

        /* Divide e Impera */
        reverse(v, First + 1, Last - 1);

        /*  Combina */
        tmp = v[Last];
        v[Last] = v[First];
        v[First] = tmp;
        }
}
```

Calcolo Maiuscole Ricorsivo

- Data una stringa (un vettore di char) stringa[First..Last] scrivere una funzione ricorsiva lineare che calcoli il numero di lettere maiuscole nella stringa. La funzione restituisce numero di lettere maiuscole
- Prototipo:

```
int conta_maiuscole ( char str[], int First , int Last );
```

- Si utilizzi la seguente funzione che ritorna il valore Vero, se il carattere fornito èmaiuscolo:

```
isupper ( char c);
```

- **Divide**: la stringa di ingresso str[rst..last] viene diviso nell'elemento di testa (str[rst]) e la coda della stringa (str[rst+1..last]).
- **Caso base**: se la stringa ha un unico carattere la soluzione e banale, se il singolo carattere e maiuscolo si restituisce 1, altrimenti si restituisce 0. Allo stesso modo se la stringa e vuota la soluzione e altrettanto banale e pari a 0.

- **Impera**: si ritiene per ipotesi induttiva che si sappia risolvere correttamente il problema piu semplice del calcolo delle occorrenze delle lettere maiuscole nella stringa str[rst+1..last].
- **Combina**: si somma il numero di occorrenze della sottostringa str[rst+1..last] con 1 se la prima lettera della stringa e maiuscola, con 0 se minuscola.

Calcolo Maiuscole Ricorsivo

```
int conta_maiuscole(char str[], int first, int last) {
    int occ;
    /*Caso base*/
    if (first==last)
        if (isupper(str[first])==1)
            return 1;
        else
            return 0;

    /*Caso base*/
    if (first>last)
        return 0;

    /*Fasi di divide et impera*/
    occ = conta_maiuscole(str, first+1, last);

    /*Fase di combina*/
    if (isupper(str[first])==1)
        return 1+occ;
    else
        return occ;
}
```

- Scrivere una funzione ricorsiva che calcola il massimo comune divisore (MCD).
- Si ricorda al lettore che il MCD tra due numeri è l'intero più grande che li divide entrambi senza resto.

Teorema di Euclide:

“ogni divisore comune di a e b è divisore di a, b e del resto r della divisione tra a e b (a *mod* b), se questo non è nullo”

- Prototipo:

```
int mcd( int m, int n );
```

- **Divide**: Effettuiamo l'operazione r= a *mod* b
- **Caso base**: se r=0 allor b è MCD di a
- **Impera**: assumiamo di saper calcolare il mcd tra b ed il resto r
- **Combina**: poiché per il teorema di euclide mcd(a,b)=mcd(b,r) restituisci il risultato della operazione effettuata

Inversione di un vettore
Ricorsivo

```
int mcd(int m, int n) {
    int r;
    if (m < n)
        return mcd(n, m);
    r = m % n;

    if (r == 0)
        return (n);

    else
        return (mcd(n, r));
}
```