Università degli Studi di Milano
Corso di Laurea in Informatica, A.A. 2018-2019

Procedure ricorsive

Turno A
Nicola Basilico
Dipartimento di Informatica
Via Comelico 39/41 - 20135 Milano (MI)
Ufficio S242
nicola.basilico@unimi.it
+39 02.503.16294

Turno B
Marco Tarini
Dipartimento di Informatica
Via Celoria 18 - 20133 Milano (MI)
Ufficio 4010
marco.tarini@unimi.it
+39 02.503.16217

Ricorsione

- La risoluzione di un problema P è costruita sulla base della risoluzione di un sottoproblema di P
- Esempio classico: il fattoriale di n

$$n! = \prod_{k=1}^{n} k = n \prod_{k=1}^{n-1} k = n \times (n-1)!$$

- il fattoriale di n è uguale a n moltiplicato per il fattoriale di $n-1$ a meno che $n = 0$, in questo caso il fattoriale vale 1
- Definizione ricorsiva del fattoriale:

$$n! = \begin{cases} n \times (n-1)! & \text{if } n > 0 \\ 1 & \text{if } n = 0. \end{cases}$$

Ricorsione

- Applico la regola in cascata

$$n! = \begin{cases} n \times (n-1)! & \text{if } n > 0 \\ 1 & \text{if } n = 0. \end{cases}$$

$$4! = 4 \times \underbrace{(3)!}$$
$$3! = 3 \times \underbrace{(2)!}$$
$$2! = 2 \times \underbrace{(1)!}$$
$$1! = 1 \times \underbrace{(0)!}$$
$$0! = 1$$

Procedure ricorsiva: una procedura che invoca se stessa

- Tipico uso: approcci «divide et impera»:
- Cioè:
  per risolvere un dato problema P,
  una procedura ricorsiva invoca se stessa su un sotto-problema di P
  (più semplice di P)
- Per evitare una «ricorsione infinita», la procedura ricorsiva deve però EVITARE di invocare se stessa per quelle istanze di problema che possono essere risolte direttamente (il «caso base»)
- Per esempio:
  - per calcolare il fattoriale di 6 serve di calcolare il fattoriale 5 (caso ricorsivo)
  - tuttavia, per calcolare il fattoriale di 0 non devo calcolare alcun altro fattoriale: è semplicemente 1 (caso base)
- Senza un caso base, una funzione ricorsiva è condannata a invocare se stessa all'infinito, o meglio, fino a esaurimento dello stack
  (si verifica uno «stack overflow»)

Procedure ricorsive, oppure cicli?

- La teoria ci garantisce che
  - ogni algoritmo iterativo (con cicli) si può riformulare senza cicli, usando funzioni ricorsive,
  - ogni algoritmo che usa funzioni ricorsive può essere riformulato senza funzioni ricorsive, usando cicli
- Cioè: ogni problema che possa essere risolto con un algoritmo, può essere risolto
  - sia usando solo cicli (senza ricorsione)
  - sia usando solo ricorsione (senza cicli)
- Tuttavia, alcuni problemi sono molto più intuitivi da risolvere usando uno o l'altro (oppure entrambi) i sistemi
- A parità di altre considerazioni, i programmi che usano ricorsione tendono ad essere un poco meno efficienti, a causa dei costi insiti nelle chiamate di funzione

Procedure ricorsive: a basso livello

Una procedura ricorsiva non è una procedura foglia perché al suo interno invoca almeno una funzione: se stessa (eccettuando il caso base)

Nello scrivere una procedura ricorsiva dobbiamo stare attenti, perché la procedura è al contempo:

- il chiamante:
  dunque non può fare affidamento sulla preservazione, prima e dopo la chiamata ricorsiva dei registri: $t0…$t9, $a0…$a3, $v0…$v1
- il chiamato:
  dunque deve salvare quelli che modifica dei registri: $s0…$s7, $fp e $ra (quest'ultimo sempre, perché non è una funzione foglia!) prima del ritorno deve ripristinare questi valori

Procedure ricorsive: schema di implementazione (non l'unico)

1. Punto di ingresso (etichetta)
2. Push sullo stack dei registri da salvare
3. Branch: saltare al caso base?
4. Caso ricorsivo:
   - settare $v0,$v1
     usando una (o più) invocazioni della funzione
   - goto 6
5. Caso base:
   - settare $v0,$v1 (senza invocazione della funzione)
6. Ripristino dei registri salvati
7. Ritorno al chiamante: jr $ra

Esempio 1: fattoriale

$$n! = \begin{cases} n \times (n-1)! & \text{if } n > 0 \\ 1 & \text{if } n = 0. \end{cases}$$

- In pseudocodice:

```
func fattoriale(n int) int
{
   if (n<=1) return 1;                    // caso base
   else return n * fattoriale( n - 1 ); // caso ricorsivo
}
```

- Codice MIPS: vedi sul sito

Numeri troppo grandi!

- nota: la nostra soluzione di questo esercizio ha un grosso limite: il fattoriale di un numero cresce così velocemente che anche per numeri relativamente piccoli (per es 30) il fattoriale eccede i numeri rappresentabili con 32 bit
- Il problema verrebbe rimandato solo di poco persino usando rappresentazioni a 64 bit
  - infatti 30! è anche maggiore di 2^{64}
- risolvere questo problema esula dai contenuti di questa lezione
- limitiamoci ad ammettere che il nostro programma calcola solo fattoriali di numeri piccoli (a una cifra o quasi)

Esempio 2: numeri di Fibonacci

1 , 1 , 2 , 3 , 5 , 8 , 13 , 21 , 34 , 55 , …

$$\text{Fibo}(n) = \begin{cases} 1 & altrimenti \\ \text{Fibo}(n-1) + \text{Fibo}(n-2) & se\ n \geq 2 \end{cases}$$

F_i	**1**	**1**	**2**	**3**	**5**	**8**	**13**	**21**	**34**	**55**	**…**
i	0	1	2	3	4	5	6	7	8	9	10

In pseudocodice:

```
func fibo(n int) int
{
   if (n<2) return 1; // caso base
   else return fibo( n - 1 ) + fibo( n - 2 ); // caso ricorsivo
}
```

Codice in MIPS: vedi sul sito

Esplosione del numero di invocazioni!

- nota: la nostra soluzione di questo esercizio ha un grosso limite: dato che ogni invocazione di Fibo ne causa altre 2, anche per parametri piccoli il numero totale di invocazioni totali diventa insensatamente grande, e così anche il tempo di esecuzione
- risolvere questo problema esula dai contenuti di questa lezione
- limitiamoci ad ammettere che il nostro algoritmo è adatto a calcolare solo i primi pochi numeri della successione di Fibonacci

Esempio 3: sommatoria di un array

Sommatoria di un array A è..

- se l'array è vuoto: (caso base) 0
- altrimenti: (caso ricorsivo)
  il primo elemento +
  sommatoria dal secondo elemento in poi

Codice MIPS: vedi sul sito