Ricorsivo

Faaputuga:

o è vuota

oppure

è un piatto

seguito da una faaputuga.

Una cosa ricorsiva è una funzione che richiama se stessa al suo interno fino a quando non si verifica una certa condizione che ne cambia il suo stato e la porta alla fine (senza loop infinito quindi che manderebbe tutto in stackoverflow).

Esercizio da eseguire, calcolare il valore fattoriale di un numero con un metodo ricorsivo.

5! = 5\*4\*3\*2\*1 = un numero.

# Fibonacci:

Definizione:

Fib(1) = 1  
Fib(2) = 1  
Fib(n+2) = Fib(n+1) + Fib(n)

n = Numero in ingresso

## Ragionamento iterativo:

Si dichiarano 3 variabili oltre a n ricevuto in input:

(Fibonacci) fib = 0;  
f1 = 1;  
f2 = 0;

Tramite un ciclo, che incomincia da 2 (La condizione vuole che -i- sia minore o uguale al numero in ingresso), tanto il numero se minore ad esso darà sempre 0 in output, e quindi così si risparmiano un paio di cicli.

Si ottiene il numero di Fibonacci (fib) tramite somma di f1 e f2, poi f2 assume il valore di f1 ed f1 quello di Fibonacci, il tutto continua fino alla fine del ciclo.

Questo è molto più veloce del metodo ricorsivo.

## Ragionamento ricorsivo:

Ci sono due condizioni, le quali se si verificano fanno ritornare il valore stesso, se il valore n vale 0 o 1, questo ritorna 0 o 1.

In caso contrario, viene richiamata la somma della funzione stessa con n-1 in input e n-2, il ciclo si ripete in base al numero ricevuto in input per un numero di volte quadratico alla dimensione del numero.

Questo significa che la prima condizione non si verifica subito per numeri grandi e che quindi la funzione sarà richiamata moltissime volte, e sua volta altrettante volte al suo interno, fino a raggiungere un valore di somma pari a 0 o 1.

Metodo inefficiente rispetto a quello iterativo, sia per uso di risorse e tempi di esecuzione.po

## trovare valore in vettore “non ordinato”.

1. Creare vettore di numeri ordinati.
2. Ordinare il vettore.

### Non ricorsivo:

Leggo tutti i numeri fino a trovare quello ricercato, in caso non venga trovato ritorno -1 e avviso l’utente.

### Ricorsivo:

Decremento di uno la dimensione del vettore e inizializzo una variabile che è la posizione in cui è stato provato a trovare il numero (e alla fine sarà la posizione in cui eventualmente si trova).

Verifico se il numero di numeri è maggiore o uguale a 0, nel caso non lo sia, significa che tutto il vettore è stato letto e che il numero non è stato trovato, il numero di numeri come detto prima decrementa di uno ogni volta che si richiama la funzione.

Se il numero letto è quello da trovare, ritorno la posizione in cui mi trovavo al momento della lettura.  
In caso non venga trovato, il valore di prova otterrà il valore della funzione che viene richiamata con le stesse variabili ma la dimensione decrementata di 1 all’inizio, e a sua volta questa potrebbe fare lo stesso al suo interno, fino eventualmente a ritornare il valore -1 (se non presente) oppure la posizione.

Il metodo usato può essere immaginato come una piramide con dei gradoni, o delle scale, in cui si inizia a cercare dalla cima a scendere se su ogni scalino è presente il numero che cerchiamo, in caso affermativo ci fermiamo, in caso negativo scendiamo ancora fino a trovarci a terra, nel tale caso (come detto diverse volte) il numero non è stato trovato.

Nota Bene che la prima lettura incomincia dal nNumeri – 1, questo perché nNumeri va in overflow già all’inizio se non viene decrementato di 1 (un vettore da 100 numeri, ha posizione massima vettore[99]).

## Trova valore in vettore ordinato (ricerca binaria):

### Non ricorsivo

Vado alla posizione diretta in cui si dovrebbe trovare il numero che cerco (il vettore ha 1000 numeri ad esempio e cerco 340, vado a 340 – 1 in cui si dovrebbe trovare), oppure (magari il numero di numeri non corrisponde o alcuni mancano), leggo tutto il vettore fino a trovare il numero giusto.

### ricorsivo

Chiedo posizione inizio e fine del vettore (massima – 1 per non andare in overflow) come parametri della funzione oltre al vettore stesso e al numero da trovare.

Verifico se l’inizio è minore o uguale alla fine, in caso non lo sia significa che il vettore è già stato completamente letto e il valore non è stato trovato.

Ottengo la posizione centrale del vettore facendo la somma dell’inizio + fine / 2, dal momento che il vettore letto sarà sempre lo stesso ma da posizioni diverse successivamente.

Verifico se il numero da cercare si trova nel centro appena calcolato, in caso affermativo ritorno la posizione e finisco la funzione.

In caso negativo, controllo se il numero letto al centro è maggiore o minore rispetto a quello da trovare, e avviene una delle due seguenti condizioni:

* Se il numero al centro è maggiore di quello da trovare, l’inizio del vettore rimane lo stesso ma leggo fino al massimo il centro precedente – 1 (escludo il valore appena letto).
* Se invece il numero al centro è minore di quello da trovare, faccio l’opposto e metto come inizio la posizione del centro + 1 (sempre per non rileggere di nuovo il valore appena verificato) e come posizione di fine mantengo la solita.

Questo viene ripetuto fino a quando eventualmente non si trovi il numero, oppure la fine diventa minore dell’inizio (numero non trovato, sto rischiando di rileggere quello già letto).

Il numero di cicli è minore della ricerca iterativa, è molto efficiente.

##### La complessità è logaritmica.

|  |  |
| --- | --- |
| Range Numeri: | Numero di tentativi: |
| 1 – 1.000.000 | 18 |
| 1-1.000 | 7 |
| 1-2.000.000 | 21 |
|  |  |

Più numeri su Classroom nel documento di classe.

# :

Algoritmo ricorsivo per sommare tutti gli elementi di un vettore.

La somma di un vettore è la lettura di ogni elemento di un vettore – 1 per ciclo fino a ritornare alla posizione iniziale.

Se si trova alla posizione iniziale di lettura, si ferma, al contrario continua.

La somma del vettore di dimensione n, è o la somma del vettore in posizione n, oppure è la somma di n – 1, e così via.

Risultati Ricorsivo:

|  |  |
| --- | --- |
| numeri: | tempo: |
| 1000 | 0 |
| 10000 | 0 |
| 20000 | 0 |
| 25000 | 0 |

Risultati Iterativo:

|  |  |
| --- | --- |
| numeri: | tempo: |
| 1000 | 0 |
| 10000 | 0 |
| 20000 | 0 |
| 1000000 | 0 |

# Compito per casa:

Stringhe ricorsivo:  
Data una stringa, visualizzarla carattere per carattere, in modo ricorsivo, dalla fine.  
ES: Input: Ciao -> Ciao / oaiC.

Ordinamento vettore (anche non ricorsivo per ora):  
Ordinare un vettore.  
Consiglio: Dividere in due il vettore e ordinarlo, poi dati i due vettori ordinati, fai qualcosa (fondili).

Per ragionare.  
Immagina un vettore grande, ordinato metà per metà, e poi dobbiamo unire queste due zone ordinate in una sola tramite una funzione fusione.

Questa parte della fusione non è necessaria che sia ricorsiva.  
Concentrarsi sulla fusione dei vettori, dal momento che per ordinare le due metà possiamo usare uno degli algoritmi già fatti.

La complessità è lineare.

Questo può funzionare con un metodo ricorsivo, continuamente dividendo in 2 il vettore, ordinandolo e rifondendolo dopo.

La fusione non è ricorsiva, l’ordinamento e divisione si.

## merge (era il compito):

merge(A,p,q,r)

i <- p; j <- q + 1; k <- 1

while(i <= q and j <= r)  
 if(A[i] < A[j]) B[k] <- A[i]; i <- i + 1  
 else B[k] <- A[j]; j <- j + 1  
 k <- k + 1

while(i <= q) B[k] <- A[i]; i <- i + 1; k <- k + 1  
while(j <= r) B[k] <- A[j]; j <- j + 1; k <- k + 1

A [p..r] <- B[1..k-1]

Merge-sort(A, p,q)  
if p < r  
 then q <- (p+r)/2  
 merge-sort(A,p,q)  
 merge-sort(A,q+1,r)  
 merge(A,p,q,r)  
  
Consigliato di effettuare un debug di 16 elementi e verificare cosa succede, scrivendolo sulla carta (anche qui), magari in uno schema.

L’algoritmo risulta particolarmente veloce e funzionante anche con valori non interi, in un ordinamento da 10 milioni di numeri sono stati necessari solamente 5 secondi.

### risultati debug

Tramite una serie di messaggi, ho ottenuto diversi risultati della divisione - fusione dei vettori, questi inizialmente vengono divisi in parti sempre più piccole e poi uniti e ordinati tramite la funzione “unisciVettore” (un nome più appropriato sarebbe stato fusione), in sintesi si continua a dividere in parti più piccole fin quando è possibile in modo ricorsivo, e poi solamente dopo esegue l’unione.  
  
Il risultato con l’ordinamento di 16 numeri da 1 a 100 e output tramite printf è il seguente:

Divisione: [ 42 68 35 1 70 25 79 59 ] [ 63 65 6 46 82 28 62 92 ]

Divisione: [ 42 68 35 1 ] [ 70 25 79 59 ]

Divisione: [ 42 68 ] [ 35 1 ]

Divisione: [ 42 ] [ 68 ]

Fusione: [ 42 68 ]

Divisione: [ 35 ] [ 1 ]

Fusione: [ 42 68 1 35 ]

Fusione: [ 1 35 42 68 ]

Divisione: [ 70 25 ] [ 79 59 ]

Divisione: [ 70 ] [ 25 ]

Fusione: [ 1 35 42 68 25 70 ]

Divisione: [ 79 ] [ 59 ]

Fusione: [ 1 35 42 68 25 70 59 79 ]

Fusione: [ 1 35 42 68 25 59 70 79 ]

Fusione: [ 1 25 35 42 59 68 70 79 ]

Divisione: [ 63 65 6 46 ] [ 82 28 62 92 ]

Divisione: [ 63 65 ] [ 6 46 ]

Divisione: [ 63 ] [ 65 ]

Fusione: [ 1 25 35 42 59 68 70 79 63 65 ]

Divisione: [ 6 ] [ 46 ]

Fusione: [ 1 25 35 42 59 68 70 79 63 65 6 46 ]

Fusione: [ 1 25 35 42 59 68 70 79 6 46 63 65 ]

Divisione: [ 82 28 ] [ 62 92 ]

Divisione: [ 82 ] [ 28 ]

Fusione: [ 1 25 35 42 59 68 70 79 6 46 63 65 28 82 ]

Divisione: [ 62 ] [ 92 ]

Fusione: [ 1 25 35 42 59 68 70 79 6 46 63 65 28 82 62 92 ]

Fusione: [ 1 25 35 42 59 68 70 79 6 46 63 65 28 62 82 92 ]

Fusione: [ 1 25 35 42 59 68 70 79 6 28 46 62 63 65 82 92 ]

Fusione: [ 1 6 25 28 35 42 46 59 62 63 65 68 70 79 82 92 ]

Da cui si può notare come avvenga la fusione solamente quando i valori nei vettori divisi sono 1 o 2 ordinati, in caso non lo siano vengono prima ordinati e solamente dopo eseguita la fusione in un vettore più grande.

Complessità merge -> n\*log(n)

# Dato un valore N visualizzare i numeri dispari precedenti in modo ricorsivo

Come da titolo.