



## Progetto “Longest Increasing Subsequence” – I

28 Aprile 2021

### Introduzione

Data una sequenza  $s$  di  $n$  interi positivi, rappresentata da un array, il seguente programma in Java calcola la lunghezza della più lunga sottosequenza di  $s$  strettamente crescente (llis: *length of the longest increasing subsequence*).

Una sottosequenza strettamente crescente di  $s$  è formata da una selezione dei suoi elementi che, considerati esattamente nell'ordine in cui compaiono in  $s$ , determinano una sequenza di interi crescente in senso stretto. Per esempio, in relazione alla sequenza  $s = \langle 2, 7, 5, 7, 4 \rangle$ , le sottosequenze  $\langle 2, 4 \rangle$  e  $\langle 5, 7 \rangle$  sono strettamente crescenti, ma non della lunghezza massima possibile,  $\langle 2, 5, 7 \rangle$  è una sottosequenza crescente più lunga (di fatto l'unica nel caso considerato), mentre  $\langle 2, 7, 7 \rangle$  non è *strettamente* crescente e  $\langle 2, 4, 5, 7 \rangle$  non è una sottosequenza di  $s$  perché non è rispettato l'ordine degli elementi.

Il programma per llis si basa essenzialmente sulla procedura ricorsiva llisRec, che risolve un problema correlato per una “coda” della sequenza originale  $s$ . Più precisamente, llisRec calcola la lunghezza della sottosequenza crescente più lunga relativa agli elementi di  $s$  a partire dalla posizione  $i$ , corrispondenti cioè all'intervallo di indici compresi fra  $i$  e  $n-1$ , imponendo l'ulteriore vincolo che gli elementi della sottosequenza debbano essere strettamente maggiori del valore di un parametro aggiuntivo  $t$  che funge da soglia. I casi base di llisRec corrispondono a una coda vuota ( $i = n$ ). Se l'elemento  $x$  nella posizione iniziale  $i$  non soddisfa il vincolo  $x > t$ , allora non può far parte della sottosequenza misurata da llisRec e ci si riconduce direttamente al corrispondente problema per una coda più corta che inizia in posizione  $i+1$ . Altrimenti  $x$  può far parte o meno della sottosequenza più lunga: se vi fa parte, l'elemento va contato ai fini della lunghezza (1+) e i successivi elementi, da cercare a partire da  $i+1$ , devono essere maggiori di  $x$ ; se invece non vi fa parte il vincolo aggiuntivo resta determinato da  $t$  anche quando si restringe la ricerca alla coda che inizia in posizione  $i+1$ ; si sceglierà poi l'opzione più favorevole in base ai risultati ottenuti ricorsivamente, confrontando le lunghezze nei due casi.

Per quanto riguarda la prima invocazione di llisRec da parte di llis,  $i = 0$  identifica l'intera sequenza e  $t = 0$  non vincola la scelta del primo elemento della sottosequenza poiché tutti gli interi di  $s$  sono positivi e quindi maggiori di 0. Quindi, llisRec( $s, 0, 0$ ) calcola precisamente la lunghezza della più lunga sottosequenza crescente di  $s$ .

```
public static int llis( int[] s ) { // s[i] > 0 per i in [0,n-1], dove n = s.length
    return llisRec( s, 0, 0 );
}

private static int llisRec( int[] s, int i, int t ) {
    if ( i == s.length ) { // i = n : coda di s vuota
        return 0;
    } else if ( s[i] <= t ) { // x = s[i] ≤ t : x non può essere scelto
        return llisRec( s, i+1, t );
    } else { // x > t : x può essere scelto o meno
        return Math.max( 1+llisRec(s,i+1,s[i]), llisRec(s,i+1,t) );
    }
}
```

Nella pagina successiva sono riportati i risultati della valutazione del metodo statico llis in alcuni casi esemplificativi; quindi vengono proposti due problemi di programmazione che prevedono l'applicazione della tecnica top-down di *memoization*.

## Esempi:

<code>llis( new int[] {5, 4, 3, 2, 1} )</code>	<code>→</code>	<code>1</code>
<code>llis( new int[] {2, 7, 5, 7, 4} )</code>	<code>→</code>	<code>3</code>
<code>llis( new int[] {47, 38, 39, 25, 44} )</code>	<code>→</code>	<code>3</code>
<code>llis( new int[] {27, 90, 7, 29, 49, 8, 53, 1, 28, 6} )</code>	<code>→</code>	<code>4</code>
<code>llis( new int[] {9, 46, 54, 71, 60, 47, 1, 32, 25, 61} )</code>	<code>→</code>	<code>5</code>
<code>llis( new int[] {54, 52, 42, 33, 14, 40, 37, 61, 53, 1} )</code>	<code>→</code>	<code>3</code>
<code>llis( new int[] {10, 8, 9, 5, 6, 7, 1, 2, 3, 4} )</code>	<code>→</code>	<code>4</code>
<code>llis( new int[] {10, 11, 12, 6, 7, 8, 9, 1, 2, 3, 4, 5} )</code>	<code>→</code>	<code>5</code>
<code>llis( new int[] {7, 8, 9, 10, 4, 5, 6, 2, 3, 1} )</code>	<code>→</code>	<code>4</code>
<code>llis( new int[] {8, 9, 10, 11, 12, 4, 5, 6, 7, 1, 2, 3} )</code>	<code>→</code>	<code>5</code>
<code>llis( new int[] {6, 1, 7, 2, 8, 3, 9, 4, 10, 5} )</code>	<code>→</code>	<code>5</code>
<code>llis( new int[] {6, 1, 7, 2, 8, 3, 9, 4, 10, 5, 6} )</code>	<code>→</code>	<code>6</code>

### 1. Applicazione della tecnica top-down di memoization in situazioni semplificate

Assumi provvisoriamente che tutti gli elementi di una sequenza di lunghezza  $n$  siano minori o uguali a  $n$  e scrivi un programma equivalente che applica la tecnica top-down di *memoization* per realizzare una versione più efficiente di `llis`, registrando i risultati delle invocazioni ricorsive di `llisRec` ai fini di un potenziale riutilizzo. Verifica quindi che i risultati ottenuti siano coerenti con i valori calcolati dal programma riportato sopra (a tal fine, una parte degli esempi non possono essere usati per la verifica poiché incompatibili con l'assunzione fatta).

### 2. Applicazione della tecnica top-down di memoization in casi più generali

Estendi ora il programma che applica la tecnica top-down di *memoization* ai casi più generali in cui gli elementi della sequenza possano assumere qualunque valore intero (di tipo `int`) rappresentabile e verifica che i risultati ottenuti siano sempre coerenti con i valori calcolati dal programma riportato sopra.

*Suggerimento.*

- Poiché in generale il terzo argomento di `llisRec` può assumere valori molto grandi, non è ragionevole utilizzare i valori di  $t$  direttamente come indici di array, cosa che richiederebbe di allocare uno spazio in memoria di estensione esorbitante. Tuttavia è evidente che  $t$  è zero oppure è il valore di un elemento della sequenza  $s$ , nel qual caso lo si può rappresentare *indirettamente* attraverso la posizione di quella componente.
- Inoltre, il caso  $t = 0$  può essere a sua volta rappresentato *indirettamente* utilizzando un intero diverso dagli indici della sequenza, per esempio  $n$  ( $\notin [0, n-1]$ ).
- Alla luce delle osservazioni precedenti, al fine di rielaborare il programma attraverso una tecnica di *memoization*, si può sostituire il parametro della procedura ricorsiva corrispondente a  $t$  (rappresentato direttamente) con un indice  $j$  compreso nell'intervallo  $[0, n]$  (che rappresenta  $t$  indirettamente). A partire dall'indice  $j$  sarà comunque possibile risalire facilmente al valore della soglia  $t$ , sulla base dell'interpretazione  $t = s[j]$  se  $0 \leq j < n$ , oppure  $t = 0$  se  $j = n$ .

### 3. Caricamento del codice prodotto attraverso la piattaforma e-learning

Accedi con le tue credenziali alla sezione dedicata al laboratorio del corso di Programmazione attraverso la piattaforma *uniud e-learning*: <https://elearning.uniud.it>

Seguendo lo schema indicato nel *quiz moodle* predisposto per questa esercitazione, inserisci i programmi realizzati per risolvere i problemi proposti nei due punti precedenti.

Le soluzioni corrette caricate entro la scadenza del quiz moodle (vedi termini indicati in corrispondenza al quiz stesso) saranno considerati elementi positivi ai fini della valutazione della prova di Laboratorio di Programmazione.