Laboratorio di Programmazione a.a. 2019-20

Progetto "Longest Increasing Subsequence"

5 Maggio 2020

Applicazione della tecnica top-down di memoization

Parte I

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*). La procedura ricorsiva `llisRec` risolve un problema correlato per una "coda" della sequenza *s* che inizia con l'elemento di posizione *i*, imponendo l'ulteriore vincolo che gli elementi della sottosequenza debbano essere strettamente maggiori del valore di un terzo parametro *t*. I casi base 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 e ci si riconduce direttamente al corrispondente problema per una coda più corta, a partire dalla posizione $i+1$. Altrimenti *x* può far parte o meno della sottosequenza più lunga: se vi fa parte, i successivi elementi (da cercare nella coda che inizia in posizione $i+1$) devono essere maggiori di *x*; se invece non vi fa parte il vincolo aggiuntivo resta determinato da *t*; si sceglierà poi l'opzione più favorevole in base ai risultati ottenuti ricorsivamente.

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

public static int llisRec( int[] s, int i, int t ) {
  final int n = s.length;
  if ( i == n ) {
    return 0;
  } else if ( s[i] <= t ) {
    return llisRec( s, i+1, t );
  } else {
    return Math.max( 1+llisRec(s,i+1,s[i]), llisRec(s,i+1,t) );
  }
}
```

Scrivi un programma equivalente che applica la tecnica top-down di *memoization* per realizzare una versione più efficiente del programma, registrando i risultati di volta in volta calcolati ai fini di un eventuale riutilizzo. Verifica quindi che i risultati ottenuti siano coerenti con i valori calcolati dal programma riportato sopra.

Suggerimento.

- Poiché il terzo argomento di `llisRec` può variare in un ampio intervallo, non è opportuno utilizzare i valori di *t* direttamente come indici di array. 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.
- Per trattare anche il caso $t = 0$ uniformemente rispetto agli altri, è possibile aggiungere alla sequenza (cioè ricostruire l'array con) una componente artificiale, per esempio in posizione *n* (= lunghezza della sequenza originale), il cui valore è *zero*; in alternativa si può semplicemente attribuire alla soglia il valore zero quando l'indice che potrebbe rappresentarne la posizione è *n*, evitando così di ricostruire l'array.
- In conclusione si può rappresentare lo stato dell'elaborazione attraverso una matrice quadrata, i cui elementi sono accessibili attraverso due indici compresi nell'intervallo *[0,n]*: il primo che corrisponde direttamente al parametro *i*; il secondo, *j*, che permette di determinare *t* leggendo l'elemento in posizione *j* nella sequenza estesa (con un elemento in più di posizione *n* e valore *0*).

(Continua nella pagina successiva.)

Esempi:

```
llis( new int[] {5, 4, 3, 2, 1} )                          ⟶  1

llis( new int[] {47, 38, 39, 25, 44} )                     ⟶  3

llis( new int[] {27, 90, 7, 29, 49, 8, 53, 1, 28, 6} )     ⟶  4

llis( new int[] {9, 46, 54, 71, 60, 47, 0, 32, 25, 61} )   ⟶  5

llis( new int[] {54, 52, 42, 33, 14, 40, 37, 61, 53, 1} )  ⟶  3
```

Parte II

Sul modello del programma che applica la tecnica top-down di memoization al calcolo della funzione `llis`, sviluppa un nuovo programma per determinare *una sottosequenza crescente più lunga*, restituendola attraverso un'istanza di `IntSList`. L'intestazione del metodo statico pubblico sarà quindi:

```
public static IntSList lis( int[] s )
```

Controlla quindi le soluzioni ottenute, verificando anche che la lunghezza delle liste corrisponda al risultato di `llis`.

Parte III

Realizza, infine, una versione modificata del programma per calcolare `llis` attraverso la memoization che ti consenta di determinare per quante componenti dell'*array* di supporto sono stati effettivamente registrati i risultati delle invocazioni ricorsive di `llis` — e quante componenti sono invece rimaste indefinite. Riesci a trovare una spiegazione per l'esito riscontrato?