Laboratorio di Programmazione a.a. 2023-24

Progetto "Longest Increasing Subsequence" – II

8 Maggio 2024

Premessa

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*).

Per una descrizione del problema e dell'algoritmo ricorsivo, accompagnata da alcuni esempi, fai riferimento all'esercitazione precedente, relativa al progetto "Longest Increasing Subsequence" – parte I (2/05/2024).

```
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) );
  }
}
```

1. Applicazione della tecnica di programmazione dinamica bottom-up

La struttura di supporto appropriata per applicare la tecnica di memoization seguendo l'impostazione proposta al punto 2 dell'esercitazione precedente è una matrice quadrata $(n+1) \times (n+1)$.

Una matrice di questo tipo per la sequenza $s = < 2,\ 7,\ 5,\ 7,\ 4 >$ è illustrata nella figura della pagina seguente ($n=5$) attraverso una griglia (tratti orizzontali e verticali blu) i cui nodi corrispondono agli elementi della matrice.

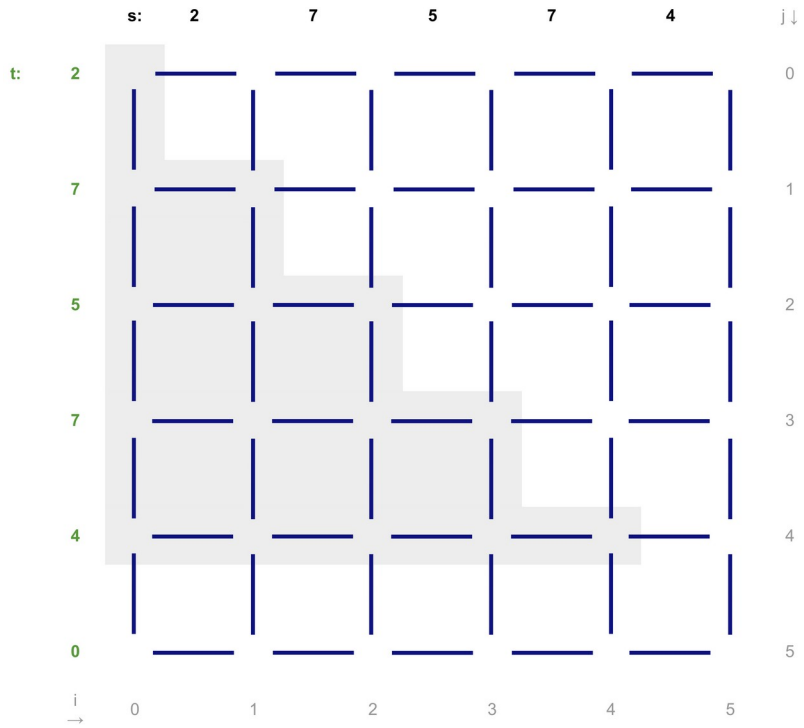
Ciascuna colonna è associata a una "coda" della sequenza s a partire dall'indice i, e tale porzione della sequenza si legge in alto, guardando gli elementi in neretto a destra della colonna di indice i. In particolare, l'indice 0 corrisponde all'intera sequenza s (tutti gli elementi sono a destra), mentre n corrisponde a una coda di s vuota (non ci sono elementi a destra della colonna n).

Ciascuna riga è invece associata a un valore del parametro t, in accordo con l'interpretazione $t = s[j]$ se $0 \le j < n$ oppure $t = 0$ se $j = n$, dove j è l'indice di riga della matrice. Il valore di t, in verde nella figura, si legge a sinistra in corrispondenza alla riga j.

Gli indici di colonna (i) e di riga (j) sono riportati in grigio, rispettivamente sotto la griglia e a destra della griglia.

Questa stessa struttura può essere utilizzata per applicare una tecnica di programmazione dinamica *bottom-up*, basata su costrutti iterativi anziché sulla ricorsione, in cui il programmatore controlla esplicitamente ogni passo del processo di calcolo, in particolare l'ordine delle operazioni da effettuare per determinare i valori da assegnare ai nodi della griglia, cioè agli elementi della matrice. Si può anche osservare che i valori dei nodi compresi nell'area a fondo grigio non sono rilevanti al fine di calcolare la lunghezza della sottosequenza crescente più lunga di s in quanto, considerando i possibili argomenti delle invocazioni di `llisRec`, $t \neq 0$ è sempre il valore di un elemento di s la cui posizione in s precede strettamente i e pertanto non può mai essere associato a un indice $j \ge i$. (Ciò significa che ci si può occupare o meno di quei nodi, a seconda di cosa sembra più semplice; presta però attenzione al fatto che, pur essendo $n \ge i$, n non è indice di un elemento della sequenza, ma corrisponde a $t = 0$, e quindi i nodi della riga più in basso sono tutti significativi.)

Le schede `llis_bottom_up.pdf`, consultabili in forma di presentazione, illustrano il processo di elaborazione per assegnare valori agli elementi della matrice nell'esempio considerato. Gli archi rossi orientati che hanno origine in un nodo riflettono le ricorsioni di `llisRec` (una o due) per i valori dei parametri corrispondenti ai nodi coinvolti.



Completa il programma preimpostato nel file `BottomUpLIS.java`, senza modificare le parti già codificate, per realizzare la procedura `llisDP` (metodo statico) applicando una tecnica di programmazione dinamica *bottom-up* in accordo con le indicazioni fornite sopra. Verifica quindi che i risultati ottenuti siano coerenti con i valori calcolati dal programma ricorsivo originale.

2. Ricostruzione di una sottosequenza crescente più lunga seguendo un percorso attraverso la matrice

Analogamente a quanto visto a lezione per il problema della sottosequenza comune più lunga (LCS), a partire dalla matrice risultante alla fine dell'elaborazione oggetto del punto precedente è possibile ricostruire una sottosequenza crescente più lunga (LIS). A tale proposito è sufficiente identificare un opportuno cammino attraverso la matrice che percorre i nodi corrispondenti alle ricorsioni di `llisRec` che hanno contribuito al risultato finale.

Le schede `lis_percorso.pdf`, consultabili in forma di presentazione, illustrano il processo di elaborazione per identificare un percorso utile nell'esempio considerato sopra. Gli archi verdi riflettono le ricorsioni di `llisRec` che hanno fornito i valori utilizzati per calcolare il risultato: in base alla codifica adottata per egli argomenti di `llisRec`, archi orizzontali corrispondono a elementi di *s* non selezionati; archi inclinati corrispondono a elementi presi a far parte della sottosequenza (evidenziati in questo caso dal fondo verde chiaro).

Completa il programma preimpostato nel file `BottomUpLIS.java`, senza modificare le parti già codificate, per realizzare la procedura `lisDP` (metodo statico) applicando una tecnica di programmazione dinamica *bottom-up* completata da un cammino attraverso la matrice in accordo con le indicazioni fornite. Verifica quindi che i risultati ottenuti siano coerenti con quanto ci si dovrebbe attendere.

Materiali disponibili online: schede illustrative `llis_bottom_up.pdf` e `lis_percorso.pdf`; programma preimpostato `BottomUpLIS.java`.