UNIVERSITÀ
DEGLI STUDI
FIRENZE

Dipartimento di Ingegneria Informatica

Laboratorio di Algoritmi e Strutture Dati

Autore:
Francesco Vici

N° Matricola:
7047233

Corso principale:
Algoritmi e Strutture Dati

Docente corso:
Simone Marinai

Contents

1 Introduzione tecnica

1.1 Esperimento assegnato

L'esperimento assegnato consiste nel confrontare vari modi per calcolare la LCS (Longest Common Subsequence) tra due stringhe:

- versione che utilizza l'algoritmo "forza-bruta"
- versione ricorsiva
- versione ricorsiva con memoization
- versione bottom-up

1.2 Specifiche tecniche del dispositivo usato

Poiché l'esperimento si incentrerà sui tempi di esecuzione degli algoritmi appena elencati, è indispensabile la descrizione delle specifiche hardware del computer usato per effettuare i test. Di seguito le specifiche:

CPU : Intel(R) Core(TM) i5-6400 CPU @ 2.70GHz,

RAM :

Manufacturer	Capacity (GB)	Speed (MHz)
Kingston	4	2133
Kingston	4	2133

SSD :

Manufacturer	Model	Size (GB)
Kingston	SUV400S37240G	240
Kingston	SA400S37960G	960

Mentre gli strumenti software utilizzati nella realizzazione del progetto e della relazione sono:

Linguaggio di programmazione : Python 3.13.x,

IDE : Visual Studio Code (latest release),

Editor LaTex : TeXstudio (latest release),

Compilatore LaTex : TeX Live 2023.

1.3 Come affrontare l'esperimento

La relazione dell'esperimento è costituita da 4 parti fondamentali:

- **Breve spiegazione teorica del problema**: oltre al testo dell'esercizio viene fornito un sommario delle principali caratteristiche teoriche del problema in esame, partendo dal materiale fornito durante il corso di Algoritmi e Strutture Dati ed ampliandolo con materiale esterno.
- **Documentazione del codice**: sono riportati i frammenti più importanti del codice Python insieme con uno schema UML di tutte le classi e una breve spiegazione delle struttura del progetto.
- **Descrizione dell'esperimento**: viene ripercorso lo svolgimento dell'esperimento con l'esposizione dei risultati ottenuti.
- **Analisi dei risultati e conclusioni**: i dati ottenuti nel corso dell'esperimento vengono messi in relazione con i risultati teorici attesi ed esposizione di una tesi conclusiva.

2 Panoramica Teorica

2.1 Introduzione al problema della Longest Common Subsequence (LCS)

L'algoritmo LCS, che sta per Longest Common Subsequence, consiste nel trovare la sottosequenza comune più lunga tra due stringhe qualsiasi, X e Y, date in input. Definiamo una sottosequenza di X come una stringa ottenuta eliminando 0 o più caratteri da X senza modificarne l'ordine. È importante sottolineare che l'ordine dei caratteri nelle sequenze di partenza è fondamentale; diversamente, si parlerebbe di sottoinsiemi e non di sottosequenze.

2.2 Diverse implementazioni della LCS

Esistono numerosi metodi per implementare l'algoritmo LCS. Quelli utilizzati in questo esperimento sono i seguenti:

Versione con algoritmo 'Brute Force' : Genera tutte le possibili sottosequenze (di qualsiasi lunghezza) di una delle due stringhe date in input e verifica la loro presenza nell'altra stringa.

Versione ricorsiva : Scompone il problema iniziale in sottoproblemi più semplici, fino ad arrivare al caso base, ovvero quando una delle due stringhe ha lunghezza pari a zero.

Versione con memoization : Segue la stessa logica della versione ricorsiva, ma memorizza in una matrice i risultati dei sottoproblemi già risolti, evitando di ripetere calcoli già effettuati. Se per un dato input è già stata trovata una soluzione, questa viene riutilizzata senza ricalcolarla.

Versione bottom-up : Risolve il problema partendo dai casi più semplici (dimensione minima) e aumentando progressivamente la dimensione fino a risolvere il problema completo. La risoluzione avviene in maniera inversa rispetto alle altre versioni.

2.3 Costi delle diverse implementazioni

Ogni implementazione dell'algoritmo LCS ha un costo computazionale diverso, a seconda della strategia utilizzata per risolvere il problema. Qui di seguito vengono analizzati i principali costi associati alle diverse versioni dell'algoritmo:

Versione con algoritmo 'Brute Force' : Poiché vengono generate tutte le possibili sottosequenze di una stringa e verificate nella seconda stringa, il tempo di esecuzione è esponenziale. In particolare, se le due stringhe X e Y sono lunghe rispettivamente n e m, si dovranno controllare tutti i caratteri di X per ogni sottosequenza di Y, quindi il tempo di esecuzione complessivo sarà nell'ordine di $\Theta(n) \cdot \Theta(2^m) = \Theta(n \cdot 2^m)$.

Versione ricorsiva : La versione ricorsiva, mantiene il costo quadratico, infatti ha un costo computazionale di $\Theta(2^{\min(n,m)})$, dove n e m sono le lunghezze delle due stringhe X e Y. Questo algoritmo infatti è abbastanza efficiente su stringhe di dimensioni molto diverse tra loro, mentre causa una notevole inefficienza se gli sono sottoposte stringhe di lunghezza simile, entrambe molto lunghe.

Versione con memoization : L'uso della memoization migliora significativamente le prestazioni, riducendo il costo computazionale a $\Theta(n \cdot m)$, dove n e m sono le lunghezze delle due stringhe X e Y. La memoization evita di risolvere più volte gli stessi sottoproblemi, memorizzando i risultati in una matrice, il che riduce drasticamente il numero di calcoli necessari rispetto alla versione ricorsiva senza memoization.

Versione bottom-up : La versione bottom-up ha anch'essa un costo di $\Theta(n \cdot m)$, dove n e m sono le lunghezze delle due stringhe X e Y. La differenza principale rispetto alla versione con memoization è che in questa versione si costruisce la soluzione a partire dal caso base, riempiendo progressivamente una matrice con le soluzioni parziali. Questo approccio è generalmente più efficiente rispetto alla versione ricorsiva con memoization in termini di utilizzo della memoria.

In generale, le versioni con memoization e bottom-up sono le più efficienti in termini di tempo, mentre la versione brute force è altamente inefficiente e viene solitamente utilizzata solo per scopi didattici o con stringhe di piccole dimensioni.

Di seguito una tabella riassuntiva con costi di tempo e memoria dei diversi algoritmi:

Algoritmo	**Costo (tempo)**	**Costo (memoria)**
Brute force	$\Theta(n \cdot 2^m)$	medium
Recursive	$\Theta(2^{\min(n,m)})$	low
Memoization	$\Theta(n \cdot m)$	high
Bottom-up	$\Theta(n \cdot m)$	low

Table 1: Costi dei diversi algoritmi LCS

3 Documentazione del codice

3.1 Struttura generale

Per svolgere l'esperimento richiesto è necessario implementare gli algoritmi coinvolti, poterli eseguire un numero qualsiasi di volte e infine salvare i tempi di esecuzione delle varie iterazioni.
In questo caso ho deciso di definire 4 classi con la stessa struttura per ogni diversa implementazione della LCS.

3.2 UML delle classi

3.3 Descrizione dei metodi più importanti