Esercitazione9_Latex

October 25, 2023

1 Esercitazione 9

1.1 Es 1. Supersequenza di lunghezza minima

Testo

Date due sequenze X = (x1,…,xn) e Y = (y1,…,ym) una supersequenza di X e Y è una qualsiasi sequenza Z tale che sia X che Y sono sottosequenze di Z. Ad esempio, per le sequenze di lettere alberi e libri le seguenti sono supersequenze: alberilibri, albelibri, lialberi, a liberi. - Dare lo pseudo-codice di un algoritmo che, date due sequenze X e Y, di lunghezze n ed m, calcola la lunghezza minima di una supersequenza di X e Y in O(nm). - Dare poi lo pseudo-codice di un algoritmo che ritorna una supersequenza di lunghezza minima di X e Y.

Idea

Soluzione

[ ]:
```
def superSeqRecAux(x,y,i,j):
        m = len(x)
        n = len(y)
        if i==m and j==n:
                return 0, [[]]
        if i==m:
                return n-j, [[y[j:]]]
        if j==n:
                return m-i, [[x[i:]]]
        if x[i]==y[j]:
                l, seqs = superSeqRecAux(x, y, i+1, j+1)
                return l+1, [[x[i]]+z for z in seqs]
        l1, seqs1 = superSeqRecAux(x, y, i+1, j)
        l2, seqs2 = superSeqRecAux(x, y, i, j+1)
        if l1 < l2:
                return l1+1, [[x[i]]+z for z in seqs1]
        if l2 < l1:
                return l2+1, [[y[j]]+z for z in seqs2]
        return l1+1, [[x[i]]+z for z in seqs1] + [[y[j]]+z for z in seqs2]
```

[ ]:
```
def superSeqRec(x,y):
# versione top-down ricorsiva che calcola la lunghezza minima supersequenza
# e tutte le supersequenze che realizzano tale minimo.
```

```
# E' inerentemente esponenziale nel caso pessimo (sequenze con caratteri␣
↪disgiunti)
# perche' in tal caso il numero di sequenze da generare e' nell'ordine di␣
↪2^(m+n)
        return superSeqRecAux(x, y, 0, 0)
```

[ ]:
```
def superSeqRecAuxDP(x,y,i,j,T):
        m = len(x)
        n = len(y)
        if T[i][j] < 0:
                if i==m and j==n:
                        T[i][j] = 0
                elif i==m:
                        T[i][j] = 1 + superSeqRecAuxDP(x, y, i, j+1,␣
↪T)
                elif j==n:
                        T[i][j] = 1 + superSeqRecAuxDP(x, y, i+1, j,␣
↪T)
                elif x[i]==y[j]:
                        T[i][j] = 1 + superSeqRecAuxDP(x, y, i+1, j+1, T)
                else:
                        T[i][j] = 1 + min(superSeqRecAuxDP(x, y, i+1, j, T),␣
↪superSeqRecAuxDP(x, y, i, j+1, T))
        return T[i][j]
```

[ ]:
```
def superSeqRecDP(x,y):
# versione top-down ricorsiva con matrice per evitare di ricadere
# negli stessi casi che dipendono solo dagli indici i e j e sono
# quindi m x n
        m = len(x)
        n = len(y)
        T = [[-1 for _ in range(n+1)] for _ in range(m+1)]
        #T[m][n] = 0
        l = superSeqRecAuxDP(x, y, 0, 0, T)
        return T
```

[ ]:
```
def superSeq(x,y):
# versione bottom-up iterativa
# costruisce una matrice T di dimensione n+1 x m+1 in cui
# T[i][j] e' la lunghezza della minima supersequenza
# tra x[i:] e y[j:]
        m = len(x)
        n = len(y)
        T = [[-1 for _ in range(n+1)] for _ in range(m+1)]
        T[m][n] = 0
        d = m+n-1
```

```
        while (d>=0):
                if (d>=n):
                        j = n
                        i = d - j
                else:
                        j = d
                        i = 0
                while i<=m and j>=0:
                        if j<=n and i+1<=m:
                                m1 = T[i+1][j]
                        else: m1=m+n
                        if i<=m and j+1<=n:
                                m2 = T[i][j+1]
                        else: m2=m+n
                        if i+1 <= m and j+1 <= n and x[i]==y[j]:
                                m3 = T[i+1][j+1]
                        else: m3 = m+n
                        T[i][j] = 1 + min(m1, m2, m3)
                        i, j = i+1, j-1
                d = d-1
        return T
```

[ ]:
```
def ricostruisci(T, x, y):
# da T posso ricostruire una soluzione ottima:
# scendere di riga significa prendere il prossimo carattere di x
# scendere di colonna significa prendere il prossimo carattere di y
# devo scendere in diagonale solo se x[i]==y[j]
        m = len(T)
        n = len(T[1])
        z = []
        i, j = 0, 0
        while i<m-1 or j<n-1:
                if i < m-1 and j < n-1 and x[i] == y[j]:
                        z.append(x[i])
                        i, j = i+1, j+1
                elif i<m and j<n-1 and T[i][j] == T[i][j+1] + 1:
                        z.append(y[j])
                        j += 1
                else:
                        z.append(x[i])
                        i += 1
        return z
```

Esecuzione

[ ]:
```
x = "alberi"
y = "libri"
```

```
T = superSeq(x, y)
print(ricostruisci(T, x, y))
print(superSeqRec(x, y))
T = superSeqRecDP(x, y)
print(ricostruisci(T, x, y))

x = "aaaa"
y = "bbbbbb"
T = superSeq(x, y)
print(ricostruisci(T, x, y))
T = superSeqRecDP(x, y)
print(ricostruisci(T, x, y))

print(superSeqRec("alberi", "libri"))
T = superSeq("aaa", "bbbb")
print(T)
print(ricostruisci(T, "aaa", "bbbb"))
```

```
['a', 'l', 'i', 'b', 'e', 'r', 'i']
(7, [['a', 'l', 'i', 'b', 'e', 'r', 'i']])
['a', 'l', 'i', 'b', 'e', 'r', 'i']
['b', 'b', 'b', 'b', 'b', 'b', 'a', 'a', 'a', 'a']
['b', 'b', 'b', 'b', 'b', 'b', 'a', 'a', 'a', 'a']
(7, [['a', 'l', 'i', 'b', 'e', 'r', 'i']])
[[7, 6, 5, 4, 3], [6, 5, 4, 3, 2], [5, 4, 3, 2, 1], [4, 3, 2, 1, 0]]
['b', 'b', 'b', 'b', 'a', 'a', 'a']
```

1.2 Es 2. Percorso crescente più lungo in matrice

Testo

Data una matrice di dimensione n x n le cui celle sono numerate con numeri distinti che vanno da 1 a n2, vogliamo trovare la massima lunghezza possibile per cammini che toccano celle con numerazione crescente e incremento di 1.
I cammini possono partire da una qualunque cella e, nel corso del cammino, dalla generica cella (i, j) ci si puo' spostare in una qualunque cella adiacente in orizzontale o verticale (vale a dire in una delle celle (i, j+1),(i+1, j),(i, j1),(i1, j)). La lunghezza di un cammino e' data dal numero di nodi toccati dal cammino.
Progettare un algoritmo che risolve il problema in tempo O(n^2)
Ad esempio: per la matrice A la risposta e' 1 mentre per la matrice B, grazie al cammino 2 -> 3 -> 4 -> 5 -> 6 -> 7, la risposta e' 6

Idea

Allocare una matrice T con i percorsi piu' lunghi gia' computati inizialmente -1 (non computato) poi si computano tutti con la programmazione dinamica

Soluzione

```
[ ]: def camminoMaxAux(M,i,j,T,v):
        n = len(M)
        # mi fermo se l'elemento corrente non e' il successivo del precedente
        if M[i][j] != v and M[i][j]!=v+1 :
                return 0
        # calcolo il massimo tra tutti i percorsi (alto, sinistra, basso,␣
 ↪destra)
        # se il valore non e' gia' disponibile in T[i][j]
        if T[i][j]<0:
                m1, m2, m3, m4 = 0,0,0,0
                if i-1 >= 0:
                        m1 = camminoMaxAux(M, i-1, j, T, M[i][j])
                if j-1 >=0:
                        m2 = camminoMaxAux(M, i, j-1, T, M[i][j])
                if i+1 < n:
                        m3 = camminoMaxAux(M, i+1, j, T, M[i][j])
                if j+1 < n:
                        m4 = camminoMaxAux(M, i, j+1, T, M[i][j])
                # aggiorno T[i][j]
                T[i][j]        = 1 + max(m1,m2,m3,m4)
        return T[i][j]

def camminoMax(M):
        n = len(M)
        T = [[-1 for _ in range(n)] for _ in range(n)]
        for i in range(n):
                for j in range(n):
                # calcolo quelli non ancora computati dalla funzione ausiliaria␣
 ↪ricorsiva
                        if T[i][j]<0:
                                T[i][j] = camminoMaxAux(M, i, j, T, M[i][j])
        return T
```

Esecuzione

```
[ ]: A =[[3,6,2],[7,1,9],[4,8,5]]
B =[[9,7,6],[8,2,5],[1,3,4]]
T = camminoMax(A)
print(T)
print(max(max(T)))
T = camminoMax(B)
print(T)
print(max(max(T)))
```

```
[[1, 1, 1], [1, 1, 1], [1, 1, 1]]
1
[[1, 1, 2], [2, 6, 3], [1, 5, 4]]
6
```

1.3 Es 3. Sequenza valida di somma minima

Testo

Abbiamo una sequenza S = (s1, s2,…,sn) di interi positivi. Una sottosequenza S' di S si definisce valida se per ogni coppia di elementi consecutivi di S almeno un elemento della coppia compare in S'.
Il valore di una sottosequenza valida e' la somma dei suoi elementi.
Ad esempio: per S = (1, 2, 3, 5, 4, 6, 7), la sottosequenza S' = (1, 3, 6) non e' valida, mentre la sottosequenza S' = (2, 5, 4, 7) e' valida ed ha valore valore 18 e la sottosequenza S'' = (2, 3, 4, 6) e' valida ed ha valore 15. - Descrivere un algoritmo che, data la sequenza S, calcola il valore minimo di una sottosequenza valida in tempo O(n). - Descrivere poi un algoritmo che trova una sottosequenza valida di valore minimo.

Idea

Soluzione

[ ]:
```
def ricostruisciSV(TP, TU, s):
        # TP e TU sono sufficienti a ricostruire la soluzione ottima
        # res[i] sara' s[i] se s[i] e' nella soluzione ottima
        # e None altrimenti
        res = [None for _ in s]
        n = len(s)
        # l'ultimo elemento sta nella soluzione ottima se TU[n-1]<TP[n-1]
        if TU[n-1]<TP[n-1]:
                res[n-1]=s[n-1]
        for i in reversed(range(n-1)):
                # l'elemento s[i] e' nella soluzione ottima se:
                #         * s[i+1] non e' nella soluzione ottima
                #    * oppure TU[i+1] viene ottenuto da TU[i]
                if res[i+1] == None or TU[i+1]==TU[i]+s[i+1]:
                        res[i]=s[i]
        return res

def seqValida(s):
        n = len(s)
        TU = [-1 for _ in s] #somma della seq minima in cui ho preso l'ultimo
        TP = [-1 for _ in s] #somma della seq minima in cui non ho preso␣
 ↪l'ultimo
        TU[0], TP[0] = s[0],0
        for i in range(1,n):
                TU[i] = min(TU[i-1]+s[i], TP[i-1]+s[i])
                TP[i] = TU[i-1]
        # il valore della sequenza ottima e' il minimo tra il valore della␣
 ↪sequenza ottima contenente l'ultimo
        # e quello della sequenza ottima NON contenente l'ultimo
        return TP, TU, min(TP[n-1], TU[n-1])
```

Esecuzione

[ ]:
```python
s = [1,2,3,5,4,6,7]
TP, TU, v = seqValida(s)
print(v, ricostruisciSV(TP, TU, s))

s = [1,5,1,1,5,1,1,5,1]
TP, TU, v = seqValida(s)
print(v, ricostruisciSV(TP, TU, s))
```

```
13 [None, 2, None, 5, None, 6, None]
6 [1, None, 1, 1, None, 1, 1, None, 1]
```