Esercitazione11_Latex

October 25, 2023

1 Esercitazione 10

1.1 Es 1. Massima sequenza di 'a' consecutivi pari

Testo

Dare lo pseudo-codice di un algoritmo che dato l'intero n, stampa tutte le stringhe lunghe n con simboli in {a, b} dove i blocchi di simboli 'a' di lunghezza massima che appaiono nella stringa hanno lunghezza pari.
Ad esempio per n = 1 viene stampata la sola stringa b mentre per n = 5, le stringhe da stampare sono:
bbbbb aabbb baabb bbaab bbbaa baaaa aabaa aaaab
L'algoritmo deve avere complessita' O(nS(n)) dove S(n) e' il numero di stringhe da stampare. Motivare la complessit'a del vostro algoritmo.

Idea

Soluzione

[ ]:
```
def maxKaCons(x, inf, sup, k, kmax):
# scrive tutte le sequenze con al massimo kmax 'a' consecutivi
        if inf == sup:
                print(''.join(x))
        else:
                x[inf] = 'b'
                maxKaCons(x, inf+1, sup, kmax, kmax)
                if k > 0:
                        x[inf] = 'a'
                        maxKaCons(x, inf+1, sup, k-1, kmax)

def pariA(x, inf, aCons, n):
        for j in range(inf, inf+aCons):
                x[j] = 'a'
        if inf + aCons < n:
                x[inf+aCons] = 'b'
                maxKaCons(x, inf+aCons+1, n, aCons, aCons)
        else:
                print(''.join(x))

def maxKaConsInit(x, inf, sup, k, kmax, acons, n):
```

```
# genera le sequenze con al piu' k a consecutivi,
# poi chiama la funzione che mette la sequenza 'b'.'a'^{2n}.'b'
        #print x, inf, sup, k, kmax, acons, n
        if inf == sup:
                if sup == 0 or x[sup-1] == 'b':
                        pariA(x, sup, acons, n)
        else:
                x[inf] = 'b'
                maxKaConsInit(x, inf+1, sup, kmax, kmax, acons, n)
                if k > 0:
                        x[inf] = 'a'
                        maxKaConsInit(x, inf+1, sup, k-1, kmax, acons, n)

def maxAConsPari(n):
        x = [None for _ in range(n)]
        maxKaCons(x, 0, n, 0, 0)
        for p in range(2,n+1,2):
                for k in range(n-p+1):
                        maxKaConsInit(x, 0, k, p-1, p-1, p, n)
```

Esecuzione

[ ]:
```
n = 3
k = 2
x = [None for _ in range(n)]
maxKaCons(x, 0, len(x), k, k)
maxAConsPari(n)
```

```
bbb
bba
bab
baa
abb
aba
aab
bbb
aab
baa
```

1.2 Es 2. Tutte le sequenze palindrome

Testo

Dare lo pseudo-codice di un algoritmo che dato un intero n, stampa tutte le stringhe palindromi lunghe 2n con valori in {a, b}.
Ad esempio per n = 2, le stringhe da stampare sono: aaaa abba baab bbbb
L'algoritmo deve avere complessita' O(n2^n). Motivare la complessita' del vostro algoritmo.

Idea

scrivere una funzione che genera tutte le sequenze lunghe n e poi ricopia le n posizioni rimanenti

Soluzione

[ ]:
```
def allStringsABAux(x, n, i):
        if i==n:
                print(''.join(x))
        else:
                x[i] = 'a'
                allStringsABAux(x, n, i+1)
                x[i] = 'b'
                allStringsABAux(x, n, i+1)

def allStringsAB(n):
        x = [None for _ in range(n)]
        allStringsABAux(x, n, 0)

#Esercizio vero e proprio: e' sufficiente modificare le funzioni viste prima
def allPalindromeABAux(x, n, i):
        if i==n:
                for j in range(n):
                        x[n+j] = x[n-j-1]
                print(''.join(x))
        else:
                x[i] = 'a'
                allPalindromeABAux(x, n, i+1)
                x[i] = 'b'
                allPalindromeABAux(x, n, i+1)

def allPalindromeAB(n):
        x = [None for _ in range(2*n)]
        allPalindromeABAux(x, n, 0)
```

Esecuzione

[ ]:
```
n = 2
allStringsAB(n)
allPalindromeAB(n)
```

```
aa
ab
ba
bb
aaaa
abba
baab
bbbb
```

1.3 Es 3. Sequenze in cui elementi consecutivi differiscono almeno di 2

Testo

Dare lo pseudo-codice di un algoritmo che dato l'intero n, stampa tutte le sequenze lunghe n formate da interi nell'insieme {1, 2, 3, 4} con la proprieta' che nella sequenza numeri adiacenti distano almeno 2.
Ad esempio per n = 3 delle 43 = 64 possibili sequenze ne vanno stampate solo 10, vale a dire:
131 141 142 241 242 313 314 413 414 424
L'algoritmo deve avere complessita' O(nS(n)) dove S(n) e' il numero di matrici da stampare. Motivare la complessita' del vostro algoritmo.

Idea

Soluzione

[ ]:
```
def allStringsAux(x, m, n, i, prec):
        if i==n:
                print(x)
        else:
                for j in range(1,prec-1):
                        x[i] = j
                        allStringsAux(x, m, n, i+1, j)
                for j in range(prec+2,m+1):
                        x[i] = j
                        allStringsAux(x, m, n, i+1, j)

def allStrings(m, n):
        x = [None for _ in range(n)]
        for j in range(1,m+1):
                x[0] = j
                allStringsAux(x, m, n, 1, j)
```

Esecuzione

[ ]:
```
allStrings(4,3)
```

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

1.4 Es 4. Stampa matrici non decrescenti su righe/colonne

Testo

Dare lo pseudo-codice di un algoritmo che dato l'intero n, stampa tutte le matrici ternarie n x n con la proprieta' che le righe e le colonne della matrice risultano ordinate in modo crescente.

Idea

Soluzione

[ ]:
```
def stampaMatriciAux(M, i, j, k):
        n = len(M)
        m = len(M[0])
        if i == n: #and j == m-1:
                print(M)
                return
        if i == 0 and j == 0: v = 0
        elif i == 0:
                v = M[i][j-1]
        elif j == 0:
                v = M[i-1][j]
        else:
                v = max(M[i-1][j],M[i][j-1])
        if j==m-1:
                ii = i+1
                jj = 0
        else:
                ii = i
                jj = j+1
        for x in range(v, k+1):
                M[i][j] = x
                stampaMatriciAux(M, ii, jj, k)


def stampaMatrici(n,k):
        M = [[None for _ in range(n)] for _ in range(n)]
        stampaMatriciAux(M, 0, 0, k)
```

Esecuzione

[ ]:
```
stampaMatrici(2, 2)
```

```
[[0, 0], [0, 0]]
[[0, 0], [0, 1]]
[[0, 0], [0, 2]]
[[0, 0], [1, 1]]
[[0, 0], [1, 2]]
[[0, 0], [2, 2]]
[[0, 1], [0, 1]]
[[0, 1], [0, 2]]
[[0, 1], [1, 1]]
[[0, 1], [1, 2]]
```

```
[[0, 1], [2, 2]]
[[0, 2], [0, 2]]
[[0, 2], [1, 2]]
[[0, 2], [2, 2]]
[[1, 1], [1, 1]]
[[1, 1], [1, 2]]
[[1, 1], [2, 2]]
[[1, 2], [1, 2]]
[[1, 2], [2, 2]]
[[2, 2], [2, 2]]
```