Programarea Algoritmilor

Broscoteanu Daria-Mihaela

Grupa 143

1) a)

```
def litere(*param):
    d=dict()
    for x in param:
        dd=dict()
        for i in range(0,26):
            c=chr(i+ord('a'))
            nr=x.count(c)
            if nr>0:
                dd[c]=nr
        d[x]=dd
    return d
```

b)

```
numere=[x for x in range(10,100) if x%2==0 and x%6!=0]
```

c) $f(L, 0, n-1)$

$T(1) = 1$

$$T(n) = 2\,T\left(\frac{n}{2}\right) + \left(\frac{n}{2}\right)^2 + O(1)$$

$$T(n) = 2\left[2\,T\left(\frac{n}{2^2}\right) + \left(\frac{n}{2^2}\right)^2 + O(1)\right] + \left(\frac{n}{2}\right)^2 + O(1)$$

$$T(n) = 2^2\,T\left(\frac{n}{2^2}\right) + 2\cdot\left(\frac{n}{2^2}\right)^2 + \left(\frac{n}{2}\right)^2 + O(1)$$

$$T(n) = 2^2\left[2T\left(\frac{n}{2^3}\right) + \left(\frac{n}{2^3}\right)^2 + O(1)\right] + 2\left(\frac{n}{2^2}\right)^2 + \left(\frac{n}{2}\right)^2 + O(1)$$

$$T(n) = 2^3\,T\left(\frac{n}{2^3}\right) + 2^2\cdot\left(\frac{n}{2^3}\right)^2 + 2\left(\frac{n}{2^2}\right)^2 + \left(\frac{n}{2}\right)^2 + O(1)$$

$$T(n) = 2^3\,T\left(\frac{n}{2^3}\right) + \frac{n^2}{2^4} + \frac{n^2}{2^3} + \frac{n^2}{2^2} + O(1)$$

$\vdots$ — — — .

$$T(n) = 2^k\cdot T\left(\frac{n}{2^k}\right) + \frac{n^2}{2^{k+1}} + \ldots + \frac{n^2}{2^2} + O(1)$$

$k = \log_2 n$

$$T\left(\frac{n}{2^{\log_2 n}}\right) = T\left(\frac{n}{n}\right) = 1$$

$\log_2 n + 1 = \log_2(2n)$

$$\Rightarrow T(n) = n\cdot 1 + \frac{n^2}{2n} + \frac{n^2}{n} + \ldots + \frac{n^2}{2^2} + O(1)$$

$$T(n) = 2^k\cdot T\left(\frac{n}{2^k}\right) + n^2\sum_{p=2}^{k+1}\frac{1}{2^p}$$

$$\sum_{p=2}^{k+1}\frac{1}{2^p} = \frac{1}{2^2} + \ldots + \frac{1}{2^{k+1}} = \frac{1}{2^2}\cdot\frac{\frac{1}{2^{k+1}} - 1}{-1 + \frac{1}{2}} = \frac{1}{4}(-2)\cdot\left(\frac{1}{n} - 1\right)$$

$$= \frac{1}{2}\cdot\frac{n-1}{n}$$

c)

$$T(n) = n \cdot 1 + n^{\frac{1}{2}} \cdot \frac{1}{2} \cdot \frac{n-1}{n}$$

$$T(n) = n + n(n-1) \cdot \frac{1}{2}$$

$$\Rightarrow T(n) = O(n^2)$$

Subiectul 3 - PD

```
n = int(input())
v = [x for x in input().split()]
v = [""] + v
l = [0 for i in range(n+1)] # pentru a determina cel mai lung subsir
por = [0 for i in range(n+1)]
l[n] = 1
por[n] = -1

def verifica(a, b):
    c1 = a[len(a)-1]
    c2 = b[0]
    if c1 == 'a':
        if c2 == 'b':
            return 1
        else:
            return 0
    else:
        if c1 == 't':
            if c2 == 'y':
                return 1
            else:
                return 0
        else:
            c3 = chr(ord(c1)+1)
            c4 = chr(ord(c1)-1)
            if c2 == c3 or c2 == c4:
                return 1
            return 0
```

```
for i in range(n-1,0,-1):
        ma=0
        p=-1
        for j in range (i+1, n+1):
            if l[j]>ma and verifica(v[i], v[j]):
                ma= l[j]
                p=j
        l[i]=ma+1
        poz[i] = p

ma = max(l)
nr = l.count(ma)
p = l.index(ma)
ok =[False for i in range(n+1)]
while p!=-1:
        ok[p] = True
        p = poz[p]
for i in range (1,n+1):
        if ok[i] == False:
            print(v[i], end=" ")
print()
if nr > 1:
    print("solutia nu este unica")
else
    print("solutia este unica")
```

Descrierea algoritmului:

Pentru a determina cuvintele care trebuie eliminate, folosesc un algoritm de subsir de lungime maxima pentru a sti cuvintele care trebuie pastrate.

Pentru a verifica proprietatea, apelez la functia verifica, in care tratez 3 cazuri:

- daca ultimul caracter din primul sir este 'a' si se "inlocuieste" cu 'b' in celalalt cuvant
- daca ultimul caracter din primul sir este 'z' si se "inlocuieste" cu 'y'
- si daca unul din cele doua: c4 (care este cel precedent lui c1) si c3 (care este cel aflat dupa c1) este egal cu c2

Algoritmul parcurge cu un for cele n cuvinte si pt fiecare dintre acestea determina subsirul maxim la care poate sa se alipeasca.

Dupa determinarea subsirurilor, se determina maximul care reprezinta lungimea subsirului care trebuie pastrat si numarul lor pentru a vedea unicitatea. Se marcheaza elem. din subsirul pastrat si se afiseaza cele nemarcate.

Avem for cu max. n pasi intr-un for cu maxim n pasi

=> $O(n^2)$.

Subiectul 4 - Backtracking

a)
```
nrf = int(input())
nrb = int(input())
n = int(input())
x = [0 for i in range(n+1)]
n = nrf + nrb
fete = 0
baieti = 0
def back(k):
    global fete, baieti, ok
    if k == n+1:
        if fete == baieti:
            print(x[1:k], sep=" ")
            ok = 1
    else:
        for i in range(x[k-1], n+1):
            x[k] = i
            if i <= nrf:
                fete += 1
            else:
                baieti += 1
            if x[k] not in x[:k] and fete <= n//2 and baieti <= n//2:
#aici inserez   back(k+1)
            if i <= nrf:
                fete -= 1
            else:
                baieti -= 1
back(1)
if ok == 0:
    print("imposibil")
```

b)

```
if k>=1 and k<=n//2 and (1 in X):
        back(k+1)
else:
    if k>=n//2+1 and (1 in x) and (nxf+1 in x):
        back(k+1)
```