Aldea Alexia Elena

grupa 144

Examen la programarea algoritmilor

Subiectul 1

a)
```
def aparitii (*numere):
    dict = {}
    c = -1
    l = []
    for element in numere:
        while c<10:
            c+=1
            f=0
            for i in element
    while c<10:
        c+=1
        f=0
        for i in numere:
            if i==c:
                f+=1
        if f>0:
            t=[c,f]
            tt=tuple(t)
            l.append(tt)
    dict[element]=l
return dict
```

b) `numere = [[[m[i][j] ** 2 for i in range(len(m)) for j in range(len(m)) if i==j]`

Aldea Alexia Elena

grupa 144

c) def f(lista, p, u):

if u-p <= 2: ⟶ $\Theta(1)$

return sum(lista[p : u+1]) ⟶ $\Theta(n)$

K = (u-p+1) // 3 ⟶ $\Theta(1)$

aux_1 = sum(lista[p : p+K])

aux_2 = f(lista, p+K+1, p+2*K-1) ⟶ $3T(n/3)+2$

aux_3 = sum(lista[p+2*K+1 : u+1])

return sum([aux_1, aux_2, aux_3]) ⟶ $\Theta(3)$

$\Rightarrow 2 \cdot T(1) + T(n) + T(3) + 3T(n/3+2) \Rightarrow T(n) + \Theta(n/3+2)$

$\Rightarrow \Theta(n) + \Theta(n/3+2)$

Aldea Alexia Elena
grupa 144

Subiectul 2

```
n = int(input("n="))
t = int(input("t="))
l = []
for i in range(n):
    s = input("el=")
    t = [float(f) for f in s.split()]
    t = tuple(t)
    l.append(t) # formăm o listă de tupluri pt. fiecare cantitate și timp.sa
l.sort(key = lambda t: (t[0])) # sortăm lista în funcție de cantitate
sol = []
sol.append(l[0][0]) # reținem prima cantitate în lista de soluții finale
s = l[0][0] * l[0][1] # o adăugăm la sumă înmulțită cu timp.sa
cant = l[0][0] # o adaugăm la sumă
for i in range(1, n):
    if s/cant == t:  # dacă formula din enunț este egală cu t, afișăm
        print(format(sol, "3f"))
        i = n # și ieșim din for pentru a salva timp
    else:
        s += l[i][0] * l[i][1]
        cant += l[i][0]
        x = l[i][0] # vrem să reținem în lista de soluții fiecare cantitate bagată la sumă
        if s/cant > t: # dacă depășim temperatura cerută, scădem până
            while s/cant > t:                    când ajungem la aceasta
                s -= 1
                cant -= 1
                x -= 1
                sol.append(x)
```

Vom salva într-o listă, sub formă de tupluri, toate perechile (x_i, y_i) primite de la tastatură $\Rightarrow$ ~~cost~~ $O(n)$. Apoi, vom sorta lista, ~~pe~~ în ordine crescătoare $\Rightarrow O(\log_2 n)$. După aceea, reținem prima ~~soluție~~ cantitate și o ~~prima~~ pornim la sume, deoarece aceasta va fi mereu bună.

Parcurgem lista $\Rightarrow O(n)$, și, dacă găsim înainte de ~~sfârșit~~ a parcurge toată lista, temperatura dorită, ne vom opri din parcurs și vom afișa soluțiile. $\Rightarrow O(n)$

În caz contrar, vom adăuga elementul curent la sume, îl vom reține într-un x și vom verifica dacă ~~nu~~ am depășit temp. cerută. Dacă da, scădem câte 1 până când vom obține ceea ce ne dorim.

Complexitatea algoritmului este ~~$O(n) + O(n \log_2 n) = O(n \log_2 n)$~~

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

Demonstrație:

Presupunem că soluția prezentată nu este optimă. Fp. că am ordonat descrescător în listă, după cantitate.

$\Rightarrow M = \{(c_j, t_j), (c_{j-1}, t_{j-1}), \ldots, (c_1, t_1)\}$ (temperatură / cantitate)

~~Dacă~~

Conform metodei Greedy, vom adăuga primul element din lista sortată în lista de soluții. Elementul fiind cel mai mare ca și cantitate și temperatură, există ~~și~~ riscul să ca la ~~un~~ sumele ce vor urma, să ~~l~~ depășim temperatura căutată și nici nu folosim un număr maxim de găleți cu apă.

Aldea Alexia Elena
grupa 144

Subiectul 3

```
def afis (k): # ~~subprogram~~ funcție de afișare ~~a unei linii~~
    for i in range ~~(k)~~ : (1, k):
        for j in range (sol[i]):
            print (i, sep=" ")
    print('\n')

def pmarginit (k, p): # funcție de verif. dc. un nr. este p-mărginit
    for i in range (1,k):
        for j in range (i+1, sol[i]+1):
            if i-j >= p or j-i>p
                return 0

    return 1
def back (k)
    if ~~s~~ s == c and pmarginit (k,p)! = 0: # dacă este p-mărginit și suma ver.,
                                              este c, afișăm soluția
        afis (k)

    ~~else~~
    ~~elif~~
    else:
        if k == c+1 : # ieșim din ~~de~~ funcție
            return

        else:
            for i in range (~~c-k+1~~ 0, c-k, -1):
                if s+i*k <= c:
                    sol[k] = i
                    s = s+i*k ~~, back(k+1); s = s - i*k~~
```

```
        back(k+1)
        s = s - i * k
p = int(input("p="))
c = int(input("c="))
back(1)
sol = [0 for i in range(p)]
s = 0
u = 0
back(1)
```

Tehnica for. este backtracking deoarece soluțiile generate se completează dem. cu elem.

b)
```
def back(k):
    if(s==c and prmaxim(k,p)!=0 and sol[1]==sol[k]
```