Tudose Alexandru-Ștefan
131

Subiectul IV - Backtrack - Parole.

Cod:

a)
```
def back(k):
    global n, tipar, L, S.
    if tipar[k-1]=='l':
        for i in L:
            sol[k]=i
            if i not in sol[1:k]:
                if k==n:
                    print("".join([str(x) for x in sol[1:k+1]]))
                else:
                    back(k+1)

    else:
        for i in S:
            sol[k]=i
            if i not in sol[1:k]:
                if k==n:
                    print("".join([str(x) for x in sol[1:k+1]]))
                else:
                    back(k+1)

n=int(input("scrieți un n: "))
tipar=list(input("scrieți un tipar: "))
L= list(input("scrieți mulțimea L: "))
S= list(input("scrieți mulțimea S: "))
sol=[0]*(n+1)

back(1).
```

Tudose Alexandru-Ștefan Examen PA
131

Subiectul I

a)

```
def litere(*cuvinte):
    trecuteLitere={}
    for i in cuvinte:
        cuvcurent={}
        for j in sorted(list(i)):
            if j not in cuvcurent:
                cuvcurent[j]=1
            else:
                cuvcurent[j]+=1
        trecuteLitere[i]=cuvcurent
    return trecuteLitere
```

b) lista=[[i, cuvcurent[i]] for i in cuvcurent if cuvcurent[i]%2==0]

c) Definim următoarea relație de recurență asociată funcției:

$$T(n)=\begin{cases}2, & n\leq 2\\ 3\cdot T\left(\frac{n}{3}\right)+2, & n\geq 3\end{cases}$$

Parametrii pentru aplicarea teoremei master vor fi:

$a=3$; $b=3$; $f(n)=2=2\cdot n^0 \Rightarrow p=0$.

$\log_b a=\log_3 3=1$ (cazul a)

$p=0$ $\Rightarrow p<\log_b a \Rightarrow T(n)\in O(n^{\log_b a})=O(n)$.

Tudose Alexandru-Ștefan
131

Subiectul al III-lea: Programare Dinamică → Cățelușa Laika

Scurtă descriere: Algoritmul se folosește de două liste suplimentare costmin și ult, ambele indexate de la 1. Pentru fiecare indice i din [1, n], costmin[i] reține costul minim pe care cățelușa îl poate avea până la treapta i, alegând suma minimă dintre costul (minim al) treptelor precedente, costul săriturii dintre ele și cea curentă și taxa treptei curente. Se formează relația de recurență:

$$costmin[i] = \begin{cases} 0, & i = 0 \\ taxe[i] + costsar[i], & i = 1 \\ \min\{costmin[i-j] + costsar[j] + taxe[i] \mid j \in \{1,2,\dots i\}\}, & i \geq 2 \end{cases}$$

Algoritmul se încadrează în prog. dinamică de tipul „căutare în spate" deoarece soluția globală este construită pas cu pas din opțiuni (costul ultimei trepte) locale ale subproblemelor curente (treptele precedente). De asemenea, nu este Greedy deoarece se respectă condiția de suprapozabilitate (fiecare treaptă este folosită pentru calculul celorlalte), combinată de tehnica memorării (reținerea costului min pentru fiecare treaptă). Lista ult e folosită pentru afișarea drumului.

Complexitate: ~~Codul are com~~

Pentru fiecare treaptă curentă, codul le analizează pe toate cele dinaintea ei. Complexitatea va fi: $O(1+2+\dots+n-1) = O(\frac{n \cdot (n-1)}{2}) \approx O(n^2)$

Tudose
13

Tudor Alexandru-Ștefan
131

Subiectul al III-lea: Programare Dinamică → Cățelușa Laika

Cod:

```
nrtrepte = int(input("dați un număr de trepte: "))
taxe = input("dați taxele treptelor, pe o linie: ")
costsar = input("dați costurile săriturilor, pe o linie: ")
taxe = [int(x) for x in taxe.split()]
costsar = [int(x) for x in costsar.split()]
costsar.insert(0, 0)
taxe.insert(0, 0)
costmin = [10 000]*(nrtrepte + 1)
costmin[0]=0
costmin[1]=taxe[1]+costsar[1]
ult=[0]*(nrtrepte +1)
ult[0]=-1

for i in range(2, nrtrepte+1):
    for j in range(1, i+1):
        if costmin[i]>costmin[i-j]+costsar[j]+taxe[i]:
            costmin[i]=costmin[i-j]+costsar[j]+taxe[i]
            ult[i] = i-j

sol=[]
print(f"taxa totală {costmin[nrtrepte]} pentru traseul cu sărituri")
while nrtrepte != -1:
    sol.append(nrtrepte)
    nrtrepte = ult[nrtrepte]
sol.reverse
for i in sol:
    print(i, end=' ')
```

...dor
Alexandru
Ștefan
...137

Sub. IV - Backtrack - Parole

Descriere:

Algoritmul se reapelează recursiv pentru fiecare poziție din parola finală, încercând pe rând ~~tot~~ toate caracterele din lista corespunzătoare L sau S, în funcție de caracterul citit din tipar. Pentru a continua către caracterul următor din soluție, el verifică dacă cel curent apare sau nu în soluție. Soluția este afișată dacă sunt n caract înregistrate în lista soluției, altfel se reapelează.

Este backtrack deoarece generează recursiv toate combinațiile posibile, evitând brute-force search-ul prin aplicarea unui criteriu de selecție.

b) Pentru afișarea ~~vocal~~ doar a parolelor care încep cu vocală, la criteriul ~~de selecție cu if din for (analat~~ ~~de afișare~~ ~~if k~~ de selecție din for (if ; not in sol[1:k] se mai adaugă:

and str(sol[1]) in "aeiouAEIOU".