TUTORIAT 11

Linkuri utile:

- https://www.python.org/
- https://docs.python.org/3/
- https://www.codecademy.com/learn/learn-python-3
- https://www.learnpython.org/
- https://stanfordpython.com/#/

Comentariu multiplu PyCharm: CTRL + /

Ce conține tutoriatul ?

Metoda BACKTRACKING

Backtracking ⟶ "to go back to an earlier point in a sequence"

Backtracking ⟶ căutare cu revenire

Generalități despre metoda BACKTRACKING

- utilizată pentru rezolvarea problemelor a căror **soluție** poate fi **reprezentată sub forma unui vector**, $S_v = x_1, x_2,...,x_n$; $x_1 \in A_1, x_2 \in A_2,..., x_n \in A_n$
  $A = A_1 \times A_2 \times \dots \times A_n$, A este **spațiul tuturor soluțiilor posibile**
- mulțimile $A_1, A_2, \dots, A_n$ sunt **finite**, cu elementele într-o relație de ordine bine stabilită
- mulțimile $A_1, A_2, \dots, A_n$ **pot fi identice**
- $x_1, x_2,...,x_n$ pot fi la rândul lor vectori
- mulțimea rezultatelor $R = \{ S_v \in A \mid S_v$ **îndeplinește anumite condiții interne** (relații între componentele $x_1, x_2,...,x_n$ pe care trebuie să le îndeplinească pentru a fi soluție)
- se pot cere toate soluțiile sau o soluție care optimizează o funcție dată
- în cazul în care se caută **o singură soluție**, algoritmul **se oprește forțat** pentru a economisi timp
- numărul de elemente ale soluției poate fi sau nu cunoscut
- vectorul S_v se completează **secvențial**, atribuind componentei x_k o valoare numai după ce au fost atribuite deja valori componentelor $x_1, x_2,...,x_{k-1}$; nu se va trece la componenta x_{k+1} decât dacă sunt verificate condițiile interne de continuare pentru $x_1, x_2,...,x_k$
- **condiţiile de continuare** sunt necesare pentru a ajunge la rezultat cu primele k alegeri făcute; dacă condiţiile de continuare nu sunt îndeplinite, atunci indiferent cum se aleg valori pentru următoarele componente $x_{k+1}, x_{k+2},...,x_n$, nu se va ajunge la rezultat, deci nu are sens continuarea completării vectorului S_v
- cu cât condiţiile de continuare sunt mai restrictive, cu atât algoritmul va fi mai eficient, deci o bună alegere a condiţiilor de continuare va duce la reducerea numărului de căutări
- dacă nu sunt îndeplinite condiţiile de continuare se alege altă valoare din mulţimea A_k, pentru componenta x_k, iar dacă în mulţimea A_k nu mai sunt alte valori (pentru că

mulţimile sunt finite), atunci se revine la componenta anterioară x_{k-1}, încercând o nouă alegere (din mulţimea A_{k-1} pentru componenta x_k-1); dacă s-a ajuns la x_1 şi nu se mai poate executa o revenire (k=0), atunci algoritmul se termină

PAȘI DE URMAT ÎN REZOLVAREA PROBLEMELOR

PAS 1: Se generează toţi candidaţii parţiali la "titlul" de soluţie .

PAS 2: Candidaţii la soluţie se construiesc pe vectori unidimensionali/bidimensionali.

PAS 3: Generarea candidaţiilor se face în paşi succesivi (înainte şi inapoi).

PAS 4: După fiecare pas se poate face o validare pentru reducerea numărului căutărilor în spaţiul soluţiilor.

PAS 5: Când s-a ajuns la o anumită dimensiune a vectorului, se verifică dacă candidatul parţial (vectorul) este sau nu o soluţie.

PAS 6: Se alege soluţia/soluţiile din candidaţii parţiali după criterii impuse de problem.

SUBALGORITM(recursiv)

```
subalgoritm back(X, k,n):
    pentru ∀ e ∈ Ak execută
        Xk = e
        dacă Cond_continuare(X,k) atunci
            dacă k = n atunci
                rezultate(X,n)
            altfel back(X,k+1,n)
```

Pentru completarea vectorului începând cu prima componentă apelul subalgoritmului este **back(X,1,n)**.

EXEMPLE

1.Se dau n numere pozitive $a_1,a_2,...,a_n$. Se cere determinarea subșirului de sumă dată T.

Rezolvare

Fie $x_i \in \{0, 1\}$ astfel: 0 – cazul în care a_i nu intră în sumă, 1 – cazul în care a_i intră în sumă

$A = \{0, 1\}^n$ este finită (**spațiul tuturor soluțiilor posibile)**

Condițiile interne: $x_1*a_1 + x_2*a_2 + ... + x_n*a_n = T$

Condițiile de continuare: $x_1*a_1 + x_2*a_2 + ... + x_k*a_k \leq T$ și

$x_1*a_1 + x_2*a_2 + ... + x_k*a_k + a_{k+1} + a_{k+2} + ... + a_n \geq T$.

Deci, $T - (a_{k+1} + a_{k+2} + ... + a_n) \leq x_1*a_1 + x_2*a_2 + ... + x_k*a_k \leq T$

Pentru k = n condițiile de continuare devin condiții interne.

Exemplu:

n = 10 v = {10, 20, 30, 40, 50, 100, 150, 1000, 5000, 10000} T = 11150

După pași succesivi se obține soluția: {150, 1000, 10 000}

2. Partiţiile unui număr natural

Se dă un număr natural n. Să se scrie un program care să afişeze toate partiţiile lui n.

Se numește partiţie a unui număr natural nenul n o mulţime de numere naturale nenule {p1, p2, …, pk} care îndeplinesc condiţia p1+p2+ …+pk = n.

Ex: pt n = 4 programul va afişa:

4 = 1+1+1+1

4 = 1+1+2

4 = 1+3

4 = 2+2

4 = 4

Observaţie: - lungimea vectorului soluţie X cel mult n;

!ATENȚIE! există posibilitatea ca soluţiile să se repete;

condiţia de final este îndeplinită atunci când suma elementelor vectorului soluţie este n

Rezolvare

Se utilizează doi parametri, unul pentru poziţia în vectorul soluţie şi un al doilea în care se memorează sumele parţiale la fiecare moment. Este găsită o soluţie atunci când valoarea celui de-al doilea parametru este egală cu n.

În această situaţie, la fiecare plasare a unei valori în vectorul X, valoarea celui de-al doilea parametru se mărește cu elementul ce se plasează în vectorul soluţie.

Apelul procedurii back din programul principal va fi back(1, 0). Există şi posibilitatea de a apela procedura back din programul principal back(1, n) şi valoarea celui de-al doilea parametru se

decrementează cu valoarea elementului ce se plasează în vectorul X, iar o soluţie este găsită când acest parametru este zero.

Indiferent care modalitate este aleasă, acest al doilea parametru permite să se optimizeze puţin programul, în sensul că se pot considera nişte condiţii de continuare mai strânse.

Model subiecte examen

Subiectul 1 – limbajul Python – 3 p.

a) Scrieţi o funcţie *litere* care primeşte un număr variabil de cuvinte formate doar din litere mici şi returnează un dicţionar care conţine, pentru fiecare cuvânt primit ca parametru, un dicţionar conţinând frecvenţele literelor distincte din cuvântul respectiv sub forma unei perechi *cuvânt*: {*litere*: *frecvenţe*}. De exemplu, pentru apelul *litere*("teste", "programare") funcţia trebuie să furnizeze dicţionarul {'teste': {'e': 2, 't': 2, 's': 1}, 'programare': {'g': 1, 'a': 2, 'm': 1, 'e': 1, 'o': 1, 'p': 1, 'r': 3}}. **(1.5 p.)**

b) Înlocuiţi punctele de suspensie din instrucţiunea *numere* = [...] cu o secvenţă de iniţializare (*list comprehension*) astfel încât, după executarea sa, lista să conţină numerele naturale formate din exact două cifre care sunt divizibile cu 2, dar nu sunt divizibile cu 6. **(0.5 p.)**

Să îndrăznim mai mult!

https://towardsdatascience.com/top-16-python-applications-in-real-world-a0404111ac23