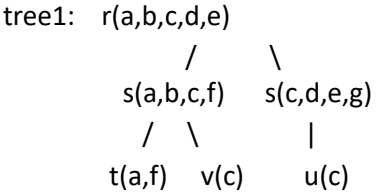
Un fapt are forma *fapt(c1,...cn)* unde c1,...,cn sunt constante. Scrieti o clasa **Fapt** care implementeaza structura respectiva de date si suprascrie functiile __str__ si __repr__ pentru a tipari un fapt. De asemenea implementati o metoda *getArguments(self)* care returneaza toate constantele care apar intr-un fapt.

O structura e o multime de fapte. Domeniul structurii e multimea tututor argumentelor faptelor care apar in structura. Scrieti clasa **Structura** impreuna cu functia *getDomain()*

Un arbore etichetat cu fapte este o structura de date de forma Tree(Fapt, [Tree(), Tree(),...]). Scrieti clasa **Tree**. Scrieti o metoda *getStructure(self)* pentru clasa **Tree** care returneaza structura care contine toate faptele care apar in vreo eticheta in arbore. De asemenea scrieti o functie *getSuccessors(self)* care returneaza lista arborilor succesori.

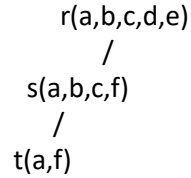
Un arbore T ca si cel de mai sus e un join tree pentru o structura S daca S=getStructure(T) si in plus urmatoarea conditie e indeplinita:

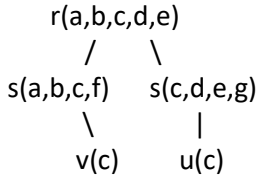
- pentru fiecare element x din domeniul lui S, daca consideram toate nodurile arborelui initiale in care x apare ca argument al faptului atasat cu nodul respectiv, nodurile respective formeaza un subarbore al arborelui initial (adica un set conectat; informal: x nu apare la un moment dat in arborele initial dupa care dispare si iar reapare). De exemplu:



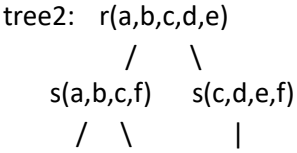
este un join tree pt {r(a,b,c,d,e), s(a,b,c,f), s(c,d,e,g), t(a,f), v(c), u(c)}

Daca consideram elementele a si c din domeniul structurii subarborii corespunzatori sunt:





Pe de alta parte,



t(a,f) v(c) u(c)

nu este un join tree pt {r(a,b,c,d,e), s(a,b,c,f), s(c,d,e,f), t(a,f), v(c), u(c)} deoarece pentru f avem:

s(a,b,c,f) s(c,d,e,f)
/
t(a,f)

adica doi subarbori care nu mai sunt conectati.

Scrieti o functie *isJoinTree(self)* pentru clasa Tree care verifica daca arborele curent este un join tree pentru structura returnata de getStructure()

5) Creati doua instante de arbori, tree1 si tree2. Pentru fiecare tipariti structura corespunzatoare si de asemenea apelati functia isJoinTree() si returnati rezultatul.

Tips pentru functia isJoinTree(): trebuie verificat ca pt fiecare element x din domeniul structurii, nodurile in care apar in arbore formeaza un arbore conectat. Pentru aceasta e util sa aveti o variabila globala *seen* care pt fiecare x e setata initial la False si e updatata la True cand x e intalnit prima data in parcurgerea arborelui.

In fiecare functie unde *seen* e updatata/folosita declarati prima data: *global seen*

Pentru fiecare x, initial se parcurge arborele (recursiv) pana se intalneste x prima data: *search(tree, x)*. Atunci se seteaza seen=True si se intra in alt mod de parcurgere, care verifica ca avem x pana dispare. Pt asta puteti folosi o functie pos_until_not(tree,x). In momentul in care se gaseste un nod unde x nu mai apare, se intra intr-un nou mod de parcurgere, check_negative(tree, x), care verifica ca din punctul respectiv si pana la final nu mai apare x.

Atentie la search(tree, x): i) daca x apare in eticheta lui tree, se intra in modul pos_until_not pentru succesori; ii) daca x nu apare in eticheta lui tree, dar a fost vazut intr-unul din succesorii lui tree; in urmatorii succesori, se intra in modul check_negative(tree, x). Nu vrem sa avem x pe alta ramura a lui tree. iii) daca x nu apare in eticheta lui tree, dar nici nu a fost vazut, se continua cu modul search pentru succesorii lui tree.

Puteti folosi urmatoarea implementare:

```
def search(tree, x):
    global seen
    if x in tree.get_value().get_args():
        seen=True
        for e in tree.get_succ():
            if not Tree.pos_until_not(e,x): return False
    else:
```

```
    for e in tree.get_succ():
        if seen:
            if not Tree.check_negative(e,x): return False
        elif not Tree.search(e,x): return False
return True
```