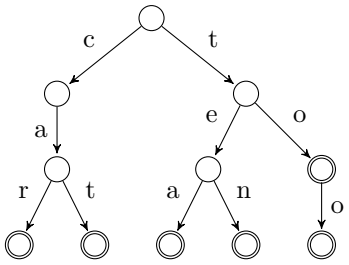
Speciální domácí úkol 2018: Trie

V tomto domácím úkolu budeme pracovat s datovou strukturou *trie*. Trie (někdy též prefixový strom) je datová struktura, která se používá pro uchovávání množin řetězců. Řetězce jsou v trie uloženy na cestách od kořene do vybraných uzlů – těmto vybraným uzlům říkáme *akceptující*.

Příklad trie vidíte na obrázku níže, akceptující uzly jsou vyznačeny dvojitou čarou. Zobrazená trie obsahuje řetězce „cat“, „car“, „tea“, „ten“, „to“ a „too“.



V tomto testu se omezíme na řetězce, které obsahují pouze malá písmena anglické abecedy. Datová struktura trie je tedy pro nás 26-ární strom, kde každý uzel může mít až 26 následníků, přičemž následníci jsou očíslováni od 0 do 25.

```
class Node:
    def __init__(self):
        self.succs = [None] * 26  # pole nasledniku
        self.succ_count = 0       # pocet nasledniku
        self.accepting = False    # je toto akceptujici uzel?

class Trie:
    def __init__(self):
        self.root = None
```

Používejte pouze ty atributy datových struktur `Node` a `Trie`, které jsou inicializovány v metodách `__init__`. V tomto úkolu je zakázáno přidávat si vlastní atributy.

Korektní trie je taková, že v každém uzlu platí, že `succ_count` je přesně počet následníků (ze seznamu `succs`), kteří nejsou `None`, a navíc splňuje tu vlastnost, že listy (uzly bez následníků) jsou vždy akceptující (tj. `accepting` je `True`). Uvědomte si, že to mimo jiné znamená, že korektní prázdná trie (neobsahující žádná slova) nesmí mít žádné uzly, tj. její `root` je `None`.

Do nultého následníka (`node.succs[0]`), pokud existuje, vede hrana s písmenem *a*, do 25. následníka (`node.succs[25]`) vede hrana s písmenem *z*. Pokud tedy například `node` označuje kořen trie na obrázku výše, pak pouze `node.succs[2]` a `node.succs[19]` jsou uzly typu `Node`, ostatní hodnoty v seznamu `node.succs` jsou `None`. Atribut `succ_count` bude tedy v našem příkladě 2.

Pro snadné přepočítání písmen na čísla od 0 do 25 máte k dispozici funkci `get_id`:

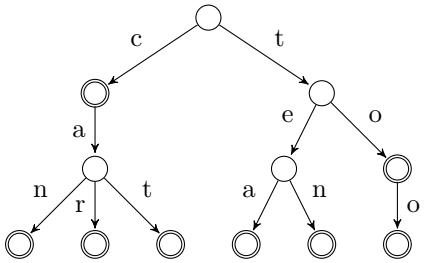
```
def get_id(x):
    return ord(x) - ord('a')
```

Úkol 1. Count (1 bod)

Napište funkci `count(trie)`, která vrátí počet řetězců, které se v zadané trie nacházejí, tj. počet akceptujících uzlů. Pro trie na obrázku výše by tedy vrátila číslo 6. Požadovaná složitost funkce `count` je $\mathcal{O}(n)$, kde n je počet uzlů v trie. Předpokládejte, že `trie` je korektní trie.

Úkol 2. Insert (1 bod)

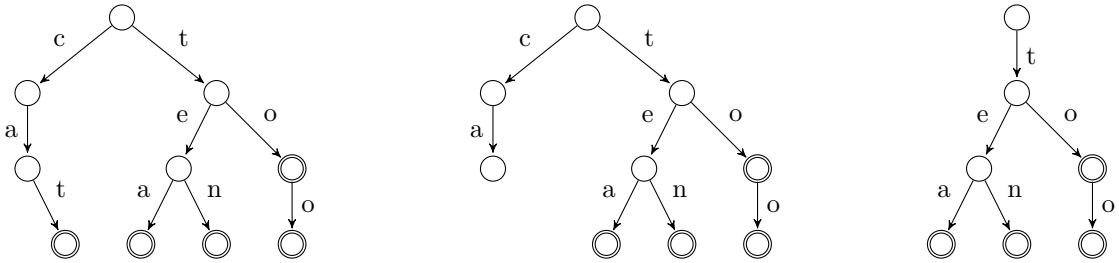
Implementujte funkci `insert(trie, word)`, která do zadané trie vloží zadané slovo. Pokud trie už toto slovo obsahovala, nezmění se. Příklad: Pokud `trie` je trie na obrázku výše, pak po volání `insert(trie, "can")` následovaném voláním `insert(trie, "c")` má být výsledkem následující trie. Všimněte si zejména, že se změnila vlastnost jednoho z následníků kořene – toho, ve kterém končí řetězec `"c"`. Původně nebyl akceptující, ale nově je. Přidání slova do trie tedy může znamenat jak přidání nových uzlů, tak i změnu atributu `accepting`.



Při vkládání rovněž nezapomeňte správně upravit atribut `succ_count` při přidávání nových následníků. Požadovaná složitost funkce `insert` je $\mathcal{O}(d)$, kde d je délka slova `word`. Předpokládejte, že `trie` je korektní trie a `word` je řetězec obsahující pouze malá písmena anglické abecedy. Funkce `insert` nic nevrací, modifikuje rovnou zadanou trie. Výsledná trie musí být korektní.

Úkol 3. Delete (1 bod)

Implementujte funkci `delete(trie, word)`, která ze zadané trie smaže zadané slovo. Pokud trie slovo neobsahovala, nezmění se. Vezměme si opět jako příklad trie z prvního obrázku; zavoláme nyní `delete(trie, "car")` a následně `delete(trie, "cat")`.



Obrázek vlevo ukazuje stav trie po smazání řetězce `"car"`. Obrázek uprostřed ukazuje situaci, do které dojdeme v průběhu mazání řetězce `"cat"`. Zde je nutné si uvědomit, že ještě nesmíme skončit. Odstranění uzlu, do kterého vedl řetězec `"cat"` totiž vede k tomu, že se v trie objevil list, který není akceptující. Takový vrchol je nutno opět odstranit a **rekurzivně** pak pokračovat směrem ke kořeni. Výsledný stav trie je potom na obrázku vpravo. *Návod k rekurzivnímu řešení je v souboru se zadáním.*

Požadovaná složitost funkce `delete` je opět $\mathcal{O}(d)$, kde d je délka slova `word`. Opět předpokládejte, že `trie` je korektní trie a `word` je řetězec obsahující pouze malá písmena anglické abecedy. Funkce `delete` nic nevrací, modifikuje rovnou zadanou trie. Výsledná trie musí být korektní.