Představení úlohy – Souborový systém

V této úloze budeme pracovat se stromovou strukturou představující souborový systém. Soubory i adresáře budou v naší struktuře reprezentovat tzv. *uzly* (nodes). To, jestli jde ve skutečnosti o běžný soubor nebo adresář budeme rozlišovat pomocí atributu `is_dir` (viz níže).

Část 1 – Reprezentace a vytvoření souborového systému (2 body)

Do připravené třídy `Node`, která reprezentuje uzel (soubor nebo adresář) v souborovém systému, doplňte inicializační metodu `__init__`. Objekty této třídy musí mít následující atributy:

- `nid`: identifikační číslo uzlu – toto číslo je v celém souborovém systému unikátní, tj. nebudou existovat dva uzly se stejným `nid`;
- `name`: jméno uzlu (řetězec);
- `owner`: jméno vlastníka (řetězec);
- `is_dir`: `True`, pokud je tento uzel adresářem, `False`, pokud jde o běžný soubor;
- `size`: velikost souboru (nezáporné celé číslo), přičemž velikost adresáře je vždy `0`;
- `parent`: adresář (uzel), ve kterém se tento uzel nachází nebo `None`, pokud je aktuální uzel kořenovým adresářem;
- `children`: seznam potomků uzlu, přičemž běžné soubory nemají nikdy žádné potomky.

Přesná podoba inicializační metody (počet a typy parametrů) je na vás. Kromě uvedených atributů si můžete přidat libovolné vlastní.

Dále implementuje čistou funkci

```
def build_fs(metadata: Dict[int, Tuple[str, str]],
             file_sizes: Dict[int, int],
             dir_content: Dict[int, List[int]]) -> Optional[Node]
```

která vytvoří souborový systém podle zadaných údajů. Parametry této funkce jsou:

- `metadata` je slovník, který `nid` uzlu přiřazuje dvojici (jméno uzlu, vlastník uzlu).

- `file_sizes` je slovník, který `nid` souboru přiřazuje jeho velikost. Smíte přitom předpokládat, že hodnoty slovníku jsou vždy nezáporná celá čísla.
- `dir_content` je slovník, který `nid` adresáře přiřazuje seznam `nid` jeho potomků. Smíte přitom předpokládat, že tento slovník neobsahuje žádné cyklické vztahy, ani se jedno `nid` nevyskytuje jako potomek dvou různých adresářů. Pokud se `nid` adresáře v `dir_content` nevyskytuje jako klíč, znamená to, že je daný adresář prázdný.

Řešení za 1 bod smí dále předpokládat, že vstup splňuje následující podmínky:

- `metadata` obsahují každé `nid`, které se vyskytuje ve `file_sizes` nebo `dir_content`;
- `file sizes` a `dir content` nemají žádný společný klíč;
- existuje vždy právě jeden *kořen* (uzel, který nemá žádného rodiče);

a vrátí kořen souborového systému. Řešení za 2 body navíc v případech porušení jedné z těchto podmínek vrátí `None`.

Nezapomeňte při vytváření souborového systému vždy správně nastavit všechny požadované atributy objektů typu `Node` (zejména `parent` a `children`).

Část 2 – Validace souborového systému (1 bod)

Implementujte následující metodu-predikát třídy `Node`:

```
    def is_valid(self) -> bool
```

Tato metoda zkontroluje, jestli *celý souborový systém*, který obsahuje aktuální uzel, splňuje následující podmínky:

- jméno žádného uzlu *kromě kořenového* není prázdné;
- jméno žádného uzlu neobsahuje znak lomítko (`/`);
- vlastník žádného uzlu není prázdný;
- žádný adresář neobsahuje dva uzly stejného jména.

V případě, že jsou tyto podmínky splněny, vrací `True`, jinak vrací `False`.

Ve všech dalších částech smíte předpokládat, že aktuální uzel je součástí souborového systému, který vznikl funkcí `build_fs` a splňuje predikát `is_valid`.

Část 3 – Vykreslení souborového systému (1 bod)

Implementuje následující metodu, která textově vykreslí souborový systém podle níže uvedeného vzoru:

```
    def draw(self) -> None
```

Vykresluje se pouze směrem „dolů“ od aktuálního uzlu, tj. aktuální uzel, jeho potomci, potomci jeho potomků, atd., ignorují se rodiče aktuálního uzlu. Metoda draw nemodifikuje aktuální objekt.

Příklad výstupu (funkci použitou k vytvoření tohoto souborového systému najdete v kostře řešení) po zavolání `draw` na kořenovém uzlu.

```
-- MY_FS
   |-- bin
   |   |-- ls
   |   |-- python
   |   \-- bash
   |-- usr
   |   \-- bin
   |       \-- env
   |-- home
   |   \-- user
   |       |-- ib111
   |       |   \-- reviews
   |       |       |-- review1.txt
   |       |       |-- review2.txt
   |       |       |-- review3.txt
   |       |       \-- .timestamp
   |       \-- pv264
   |           \-- projects
   \-- tmp
```

Příklad výstupu po zavolání `draw` na uzlu se jménem `home`:

```
-- home
   \-- user
       |-- ib111
       |   \-- reviews
       |       |-- review1.txt
       |       |-- review2.txt
       |       |-- review3.txt
       |       \-- .timestamp
       \-- pv264
           \-- projects
```

Všimněte si zejména toho, že

- svislá čára (tvořená znaky `'|'`) sahá vždy jen tak daleko, jak je třeba,
- u posledního uzlu v adresáři se svislá čára mění ve zpětné lomítko (`'\'`),
- výpis každého uzlu má formát dva znaky `-`, mezera, jméno uzlu,
- každá další úroveň začíná o čtyři znaky více vpravo než předchozí.

Část 4 – Zjišťování vlastností souborového systému (2 body)

Implementuje následující čisté metody. Příklady výstupu najdete v kostře řešení.

- `full_path(self) -> str` vrátí plnou cestu k aktuálnímu uzlu ve tvaru `/adresář/adresář/...`. Cesta má končit znakem lomítka právě tehdy, je-li aktuální uzel adresářem. Jméno kořenového uzlu ignorujte (jeho plná cesta bude vždy jen `/`).
- `disk_usage(self) -> Tuple[int, int]` vrátí dvojici (počet souborů, součet velikostí souborů) pro podstrom aktuálního uzlu.
- `all_owners(self) -> Set[str]` vrátí množinu všech vlastníků uzlů v podstromu aktuálního uzlu. (Zahrnujeme zde i vlastníka aktuálního uzlu.)
- `empty_files(self) -> List['Node']` vrátí seznam všech prázdných souborů v podstromu aktuálního uzlu. Na pořadí prvků v seznamu nezáleží.

Část 5 – Modifikace souborového systému (1 bod)

Implementujte následující metody, které modifikují souborový systém.

- `prepend_owner_name(self) -> None` změní jméno každého souboru v podstromu aktuálního uzlu tak, že mu předřadí jméno vlastníka a podtržítko.
- `add_keep_files(self, start: int) -> None` vloží do každého prázdného adresáře v podstromu aktuálního uzlu nový prázdný soubor se jménem `'.keep'`, přičemž `nid` těchto nově vytvořených souborů budou začínat číslem `start`, tedy budou to `start`, `start + 1`, `start + 2`, atd. Smíte předpokládat, že nedojde ke kolizi s `nid` existujícího souboru. Vlastníkem každého takto vytvořeného souboru bude vlastník adresáře, do nějž byl soubor vložen.

Část 6 – Prořezávání souborového systému (1 bod)

Implementujte následující metody, které odřezávají některé uzly souborového systému. Odříznutím uzlu zde myslíme odstranění ze seznamu potomků jeho rodiče (tím se ze souborového systému odstraní i celý podstrom daného uzlu, což je očekávané chování). Pro obě metody platí, že zpracováváte pouze podstrom aktuálního uzlu (`self`) bez tohoto uzlu samotného – ten nikdy neodřezávejte.

- `remove_empty_dirs(self) -> None` odstraní z podstromu aktuálního uzlu všechny prázdné adresáře (tj. takové, že nikde ve svém podstromu neobsahují žádný soubor).
- `remove all foreign(self, user: str) -> None` odstraní z podstromu aktuálního uzlu všechny uzly, jejichž vlastníkem je někdo jiný, než zadaný uživatel.

Výše uvedený příklad po zavolání `remove_empty_dirs()` na kořenovém uzlu:

```
-- MY_FS
   |-- bin
   |   |-- ls
   |   |-- python
   |   \-- bash
   |-- usr
   |   \-- bin
   |       \-- env
   \-- home
       \-- user
           \-- ib111
               \-- reviews
                   |-- review1.txt
                   |-- review2.txt
                   |-- review3.txt
                   \-- .timestamp
```

Tentýž příklad po zavolání `remove_all_foreign('root')` na kořenovém uzlu:

```
-- MY_FS
   |-- bin
   |   |-- ls
   |   |-- python
   |   \-- bash
   |-- usr
   |   \-- bin
   |       \-- env
   \-- home
```

Tentýž příklad po zavolání `remove_all_foreign('nobody')` na kořenovém uzlu:

```
-- MY_FS
   \-- tmp
```