Ohlásená doba meškania sa môže zmeniť

Zadanie 1 – deadline: 21. 3. 2025

Určite to poznáte – stojíte na stanici, vlak nikde, a v reprákoch zaznie osudová veta: váš vlak bude meškať. Päť minút. Ohlásená doba meškania sa môže zmeniť. Desať. Pätnásť. Čas na tabuli sa mení, ale realita zostáva krutá – stále čakáte a rozmýšľate o tom, že určite musí existovať nejaký spôsob riešenia problému.
V tomto zadaní sa pozrieme na prvý krok k riešeniu optimalizácie liniek. K dispozícii budeme mať mesačné dáta cestovného poriadku a meškaní vybraných vlakov. Vašou úlohou bude načítať a spracovať tieto dáta, ako aj analyzovať meškania. Budete analyzovať spoľahlivosť spojov, a zistíte, ktoré linky by mali v cestovnom poriadku radšej uvádzať len „prídeme, keď prídeme“.
Pri riešení použijete základné jazykové konštrukty, precvičíte si prácu so súbormi, ako aj so základnými údajovými typmi v Pythone ako sú reťazce, zoznamy, n-tice a slovníky.

Poznámka: Uvedené funkcie viete implementovať nezávisle, avšak pre úplné pochopenie úlohy odporúčame riešiť úlohy v uvedenom poradí. Okrem uvedených funkcií môžete implementovať aj ďalšie pomocné funkcie, hodnotené však budú iba tie uvedené v tomto dokumente.

Úloha 1: `load_line_infos` (2 body)

Vašou úlohou je načítať a spracovať dáta o cestovných poriadkoch z CSV súboru. Súbor obsahuje informácie o viacerých linkách, a každá linka je reprezentovaná číslom a zoznamom zastávok. Implementujte funkciu `load_line_infos()`, ktorá načíta obsah súboru zo zadanej cesty (`path`), spracuje dáta a uloží ich do slovníka, kde kľúčom je číslo linky (ako `int`) a hodnotou je zoznam *tuple*, ktoré obsahujú informácie o zastávkach.

Každá linka má niekoľko zastávok, pričom informácie o prvej zastávke obsahujú názov mesta a zoznam všetkých časov odchodov. Časy sú reprezentované ako zoznam *tuple* (hodina, minúta). Informácie o ostatných zastávkach obsahujú názov mesta a dĺžku cesty (v minútach) z predchádzajúcej zastávky.

Aby sa vám jednoduchšie pracovalo s názvami miest, všetky majú iba trojpísmenový kód, príklady popisu liniek nájdete v súbore *timetable.csv*. Napríklad pre linku:

```
398;CNI,06:10,14:30,18:35;FEA,6;DKC,10
```

máte vygenerovať slovník:

```
{398: [("CNI", [(6, 10), (14, 30), (18, 35)]), ("FEA ", 6), ("DKC ", 10)]}
```

Funkcia reálne vráti jeden slovník s informáciami o všetkých linkách, ktoré sú popísané v zdrojovom súbore. Môžete rátať s tým, že všetky vstupné údaje budú platné a na konci súboru bude jeden prázdny riadok. V každom riadku máte oddelené bodkočiarkou číslo linky, a jednotlivé zastávky, pričom informácie pre jednu zastávku sú oddelené čiarkou. Dbajte na správny typ hodnôt vo finálnom slovníku – okrem názvu miest musíte všetky ostatné hodnoty prekonvertovať na celé čísla.

Úloha 2: `find_next_train` (1 bod)

Predstavte si, že čakáte na stanici a chcete zistiť, kedy príde najbližší vlak na konkrétnej linke. Vašou úlohou je implementovať funkciu `find_next_train()`, ktorá na základe cestovného poriadku nájde a vygeneruje presný rozpis príchodov a odchodov na jednotlivých zastávkach pre danú linku. Funkcia má nasledovné vstupné parametre:

- `line_infos` – slovník obsahujúci informácie o linkách (vygenerovaný `load_line_infos()`).
- `line` – číslo linky, pre ktorú chceme nájsť najbližší spoj (`int`).
- `hour` – hodina, od ktorej hľadáme spoj (`int`).
- `minute` – minúta, od ktorej hľadáme spoj (`int`, predvolená hodnota 0).

Funkcia vracia zoznam n-tíc, kde každá z nich popisuje informácie o príchode na danú stanicu. Každá dvojica teda obsahuje názov mesta (ako *string*), a čas príchodu (ako *tuple* vo formáte (hodina, minúta)). Ak daná linka neexistuje, funkcia vracia *None*. Ak po zadanom čase linka už nepremáva, vráťte rozpis pre prvý ranný spoj na danej linke (musíme cestovať až v ďalší deň). Hľadať začínajte od zadaného času vrátane, t.j. ak parametre reprezentujú čas 6:30 a v tom čase odchádza vlak, máte vrátiť informácie o tomto spoji.
Pri generovaní času príchodov zohľadnite, že vlak začína svoju cestu načas vo východiskovej stanici, následne ku každému mestu pridajte čas jazdy v minútach podľa cestovného poriadku, rátajte však s tým, že vlak na každej zastávke stojí presne 1 minútu.

Napríklad pre vstupný cestovný plán:

```
line_infos = {12: [("Bratislava", [(6, 30), (7, 45), (9, 0)]), ("Trnava", 35), ("Nitra", 50), ("Zvolen", 60)]}
```

Funkcia môže vygenerovať výstup (podľa zadaného času):

```
[
    ("Bratislava", (7, 45)),  # vlak odchádza o 7:45
    ("Trnava", (8, 20)),  # príchod do Trnavy (7:45 + 35)
    ("Nitra", (9, 11)),  # príchod do Nitry (8:20 + 50 + 1 min)
    ("Zvolen", (10, 12))  # príchod do Zvolena (9:11 + 60 + 1 min)
]
```

Úloha 3: `get_city_timetable` (2 body)

V tejto úlohe sa pozrieme na odchody vlakov z konkrétneho mesta a vygenerujeme prehľadný cestovný poriadok pre všetky linky, ktoré odchádzajú odtiaľ. Naprogramujte funkciu `get_city_timetable()`, ktorá na základe načítaného cestovného poriadku vygeneruje prehľad odchodov vlakov do všetkých možných konečných staníc z daného mesta. Funkcia má nasledovné vstupné parametre:

- `line_infos` – slovník obsahujúci informácie o linkách (vygenerovaný `load_line_infos()`).
- `city` – názov mesta, pre ktoré chceme generovať cestovný poriadok (`string`).

Funkcia vracia slovník, v ktorom kľúče sú možné cieľové stanice a hodnotami sú zoznamy n-tíc, pričom každá z nich reprezentuje čas odchodu vlaku v smere cieľovej stanice. Tieto časy musia byť zoradené chronologicky a bez ohľadu na konkrétnu linku.

Funkcia napríklad môže vygenerovať slovník (reálne sa môže jednať o odchody rôznych liniek):

```
{"Košice": [(6, 30), (8, 45), (12, 10)], "Zvolen": [(7, 15), (10, 20)]}
```

Poznámka: pre efektívnejšiu prácu odporúčame riešenie rozbiť na menšie časti (napríklad výpočet času odchodu zo zadanej stanice) a napísať pre ne pomocné funkcie.

Úloha 4: `load_arrival_times` (2 body)

Na to, aby sme sa mohli pozrieť na reálne meškania, najprv si musíme načítať konkrétne údaje, ktoré nájdete napríklad v súbore *arrivals.csv*. Jedná sa o CSV súbor, kde každý riadok predstavuje zaznamenaný príchod alebo odchod vlaku z jednej stanice vo formáte:

```
číslo linky,mesto,dátum a čas,typ udalosti (A pre príchod, D pre odchod)
142,VYO,2025-01-01 06:00:01,D
```

Tieto riadky sú zoradené podľa času bez ohľadu na linku alebo mesto. Pre počiatočnú stanicu nás bude zaujímať čas odchodu, pre ostatné stanice budeme spracovávať iba časy príchodov.

Vašou úlohou je implementovať funkciu `load_arrival_times()`, ktorá načíta tieto údaje a spracuje ich do použiteľnej formy, konkrétne do štruktúrovaného slovníka. Funkcia má dve vstupné hodnoty:

- `path` – cesta k CSV súboru, z ktorého máme načítať informácie
- `line_infos` – slovník obsahujúci informácie o linkách (vygenerovaný `load_line_infos()`).

Vo výslednom slovníku budú kľúčmi čísla liniek (`int`) a hodnotami budú ďalšie slovníky. Vo vnútornom slovníku ako kľúč použijeme plánovaný čas odchodu vlaku z počiatočnej stanice (*tuple* vo formáte (hodina, minúta)). Pod týmto kľúčom nájdeme ešte jeden slovník, v ktorom kľúčom bude dátum, a hodnotou štvorica (*tuple*): názov zastávky, plánovaný príchod (*tuple*), skutočný príchod (*tuple*) a meškanie v minútach (`int`, vždy zaokrúhľujte smerom nadol).

Funkcia môže vygenerovať nasledovný výstup (samozrejme tam budete mať viac liniek a viac dní):

```
{
    12: {
        (6, 30): {
            "2025-02-24": [
                ("Trnava", (7, 05), (7, 08), 3),
                ("Nitra", (8, 00), (8, 12), 12),
                ("Zvolen", (9, 15), (9, 45), 30)
            ]
        }
    }
}
```

Poznámka: pre efektívnejšiu prácu odporúčame riešenie rozbiť na menšie časti (napríklad výpočet plánovaného príchodu, identifikácia konkrétneho vlaku na základe času príchodu) a napísať pre ne pomocné funkcie.

Úloha 5: `get_most_delayed_line` (1 bod)

V tejto úlohe nájdete linku, ktorá najviac meškala v priemere na konečnej zastávke. Implementujte funkciu `get_most_delayed_line()`, ktorá na základe vstupných dát načítaných funkciou `load_arrival_times()` nájde najmenej spoľahlivú linku a vráti dvojicu hodnôt:

- číslo linky (`int`)
- priemerné meškanie na konečnej zastávke (`float`).

Možný výstup funkcie je: `(592, 8.516129032258064)`

Poznámka: keďže pri spracovaní meškaní môže dôjsť k rozdielom pri zaokrúhľovaní, získaná hodnota bude porovnaná s očakávanou s toleranciou 0,001 (zaokrúhlenie na tri desatinné miesta).

Úloha 6: `get_most_delayed_city` (1 bod)

V tejto úlohe nájdete mesto, v ktorom vlaky najviac meškajú v priemere. Implementujte funkciu `get_most_delayed_city()`, ktorá na základe vstupných dát načítaných funkciou `load_arrival_times()` nájde hľadané mesto a vráti dvojicu hodnôt:

- názov mesta (`string`)
- priemerné meškanie na danej zastávke (`float`).

Možný výstup funkcie je: `('REN', 3.9161290322580644)`

Poznámka: keďže pri spracovaní meškaní môže dôjsť k rozdielom pri zaokrúhľovaní, získaná hodnota bude porovnaná s očakávanou s toleranciou 0,001 (zaokrúhlenie na tri desatinné miesta).

Vaše riešenia môžete otestovať aj pomocou sady testov v súbore `assignment1_test.py`. Pri hodnotení vášho riešenia sa použijú podobné testy, avšak ich bude viac. Testy budú zverejnené dva týždne pred odovzdaním. Vaše riešenie môžete testovať aj vo funkcii `main`. V odovzdanom riešení nenechajte príkazy mimo funkcií.

Vzhľadom na neskoršie zverejnenie úlohy, v prípade, že dosiahnete aspoň 5 bodov, získate jeden bod navyše (maximum teda bude 9b + 1b).

Približná dĺžka riešenia: *cca. 250 riadkov formátovaného kódu s komentármi*.