Jméno: **UČO:**

0007 *list* 1 *učo* *body*

Oblast strojově snímaných informací. Své učo a číslo listu vyplňte zleva dle vzoru číslic. Jinak do této oblasti nezasahujte. 0123456789

1. [0,7 bodu] Tento úkol bude vaší první příležitostí se algoritmus nad automaty nejen naučit, ale také si ho i implementovat. Konkrétněji, budeme požadovat, abyste implementovali dva různé algoritmy. Implementace musí být v jazyce Python a využívat kostru dostupnou v interaktivní osnově.

Motivace a představení zadání

Jednou ze základních funkcí při práci s textem je vyhledávání. Proto bychom chtěli, aby tato operace byla rychlá. Naivní řešení hledání slova w v textu t pro každou pozici zkouší, zda se na ní slovo nachází. Pokud se nějaké znaky neshodují, zahodí všechny informace a začne znovu na další pozici v textu. Složitost tohoto přístupu pro text délky n a vyhledávané slovo délky m je $\mathcal{O}(n \cdot m)$.

Jde to ale i lépe! Jednou z možností je *KMP* (Knuth-Morris-Pratt) algoritmus. Tento algoritmus dokáže vyhledávat slova v textu se složitostí $\mathcal{O}(n)$. K dosažení lineární složitosti využívá faktu, že když dojde ke konfliktu při porovnávaní symbolu w_i a t_j, od začátku porovnávaní w_0 a t_{j-i}, už víme jaké znaky jsou na předchozích pozicích $t_{j-i+1}, t_{j-i+2}, \ldots, t_j$.

Tradičně je tato myšlenka implementována tvorbou a držením tabulky přechodů v případě neshody mezi znaky. Vaším úkolem bude vytvořit stejně efektivní algoritmus na vyhledávaní pomocí automatů; konkrétně budete programovat dvě funkce. Jedna z nich vytvoří pro zadané slovo w nedeterministický automat akceptující všechna slova, která slovo w obsahují jako podřetězec. Druhá funkce bude implementovat standardní algoritmus na determinizaci nedeterministických konečných automatů. Tyto funkce bude pak využívat funkce `find`, která implementuje samotné vyhledávání. Tahle funkce je už připravena v kostře.

Reprezentace automatů

Automat bude reprezentovaný v jednodušší variantě, než jste v tomto předmětu zvyklí. Namísto držení si stavů a abecedy budeme uvažovat fixní abecedu (malá písmena anglické abecedy a mezeru) a množina stavů bude daná implicitně pomocí přechodové funkce. Přechodovou funkci budeme reprezentovat slovníkem (`dict`). Klíči budou dvojice $(stav, symbol)$ a hodnotami stavy u DFA a množiny stavů u NFA. Mimo to si třída drží množinu akceptujících stavů a iniciální stav.

Do definice tříd **není povoleno zasahovat**.

Implementace vyhledávání

Funkci `find`(*word*: `str`, *string*: `str`) → `int`, která realizuje samotné vyhledávání, už dostáváte jako součást kostry.

Funkce pro vyhledávané slovo *word* nejprve vytvoří nedeterministický automat, který pak determinizuje (tyto dvě funkce budete implementovat). Následně prochází zadaný text *string* a zároveň sleduje, ve kterém stavu determinizovaného automatu se nachází. Pokud se dostane do akceptujícího stavu, pak na dané pozici ve *string* končí vyhledávané slovo. Pokud se do akceptujícího stavu nikdy nedostane, pak *string* hledané slovo *word* neobsahuje.

Zadání funkcí

`to_nfa`(*word*: `str`) → `NFA`:

Funkce dané slovo *word* vrátí nedeterministický automat $\mathcal{A}$. Pro automat $\mathcal{A}$ platí

$$L(\mathcal{A}) = \{u \in \Sigma^* \mid u \text{ obsahuje slovo } word\}.$$

Jméno: **UČO:**

0007 *list* 2 *učo* *body*

Oblast strojově snímaných informací. Své učo a číslo listu vyplňte zleva dle vzoru číslic. Jinak do této oblasti nezasahujte. 0123456789

Příklad

Funkce pro vstup *ahoj* funkce může vrátit automat $\mathcal{A}_N$ zadaný následující tabulkou níže vlevo. Validních řešení je více, pro zadané slovo můžete vrátit libovolný automat akceptující jazyk uvedený výše. Napravo je částečný popis `NFA` $\mathtt{A}_N$ vytvořeného referenční implementací pro vstup *ahoj*. Vynechané jsou přechody pro znaky abecedy, které se nevyskytují v *ahoj*. Tyto znaky mají smyčku ve stavech nula a čtyři, v ostatních přechodová funkce vrací prázdnu množinu.

$\mathcal{A}$	a	h	o	j
$\rightarrow 0$	$\{0,1\}$	$\{0\}$	$\{0\}$	$\{0\}$
1	$\emptyset$	$\{2\}$	$\emptyset$	$\emptyset$
2	$\emptyset$	$\emptyset$	$\{3\}$	$\emptyset$
3	$\emptyset$	$\emptyset$	$\emptyset$	$\{4\}$
$\leftarrow 4$	$\{4\}$	$\{4\}$	$\{4\}$	$\{4\}$

```
A_N.init = 0
A_N.acc = {4}
A_N.transition_table = {
  (0, 'a'): {0,1}, (0, 'b'): {0}, ...
  (1, 'h'): {2}, (1, 'a'): set(), (1, 'b'): set() ...
  (2, 'o'): {3}, (2, 'a'): set(), (2, 'b'): set() ...
  (3, 'j'): {4}, (3, 'a'): set(), (3, 'b'): set() ...
  (4, 'a'): {4}, (4, 'b'): {4}, ...
}
```

`nfa_to_dfa`$(nfa$ `: NFA)` $\rightarrow$ `DFA:`

Funkce pro daný vstupní nedeterministický automat $\mathcal{A}_N$ funkce `nfa_to_dfa` vrátí ekvivalentní deterministický automat $\mathcal{A}_D$. K vytvoření $\mathcal{A}_D$ využijte **algoritmus 2.3** ze skript, se kterým se také znáte z cvičení.

Příklad

Pro vstupní automat $\mathtt{A}_N$ popsaný výše vrátí referenční implementace automat $\mathtt{A}_D$ částečně popsaný níže. Referenční implementace používá pro reprezentace stavů ntice (`tuple`), lze ovšem využít jakýkoli jiný typ, například `frozenset`.

```
A_D.init =  (0,) # ntice délky 1 obsahující 0
A_D.acc = {(0, 4)}
A_D.transition_table = {
    ((0,),   'a'): (0, 1), ((0,),   'b'): (0,), ..., (((0,),   'h'), (0,)),   ...
    ((0, 1), 'a'): (0, 1), ((0, 1), 'b'): (0,), ..., (((0, 1), 'h'), (0, 2)), ...
    ((0, 2), 'a'): (0, 1), ((0, 2), 'b'): (0,), ..., (((0, 2), 'h'), (0,)),   ...
    ...
}
```

Hodnocení

- K získaní nenulového hodnocení musíte implementovat obě funkce.
- **Není povolena** úprava kostry s výjimkou implementace funkcí `to_nfa` a `nfa_to_dfa` požadovaných zadáním. Konkrétně tedy nezasahujte do definice tříd `GenericAutomaton`, `NFA` a `DFA`, ani do implementace `find`. Neměňte ani hlavičky funkcí `to_nfa` a `nfa_to_dfa`, konkrétně nepřidávejte ani neubírejte argumenty.

Jméno: **UČO:**

0007 *list* 3 *učo* *body*

Oblast strojově snímaných informací. Své učo a číslo listu vyplňte zleva dle vzoru číslic. Jinak do této oblasti nezasahujte. 0123456789

- Můžete si implementovat pomocné funkce (vzorové řešení ale žádné pomocné funkce nepoužívá).
- Využívaní modulů s výjimkou `typing` a `collections` **není povoleno** bez explicitního schválení na diskusním fóru. Vzorové řešení si vystačí se zmíněnými moduly. V případě opodstatněného požadavku na nový modul skrz diskuzní fórum je možné povolit i ten.
- Řešení **nemusí** projít kontrolou `mypy`, i když silně doporučujeme, aby prošlo.
- Obě implementované funkce musí být čisté. Speciálně **není povoleno**, aby něco vypisovaly na standardní nebo standardní chybový výstup.

Odevzdávání

Vypracovaný úkol (jediný `.py` soubor s definicemi požadovaných funkcí) odevzdávejte do příslušné odevzdávárny v ISu.

Úkol se bude vyhodnocovat ve stejném režimu, jako textové domácí úkoly, tedy hromadně po konci odevzdání. Nebudete tedy mít k dispozici průběžné výsledky od vyhodnocovací služby (toto je bohužel důsledek technických omezení daných současnou organizací předmětu).

V kostře máte dodáno několik základních testů a jistě si snadno dokážete dopsat vlastní. Můžete také například využít testovací techniku *fuzzing*, při které vstupy pro funkci generujete (systematicky nebo manuálně) a výsledky testujete vůči referenční implementaci (můžete použít například naivní implementaci popsanou na začátku zadání). Své vlastní testy volejte z `if __name__ == "__main__"` bloku.

Řešení budeme testovat pomocí Pythonu verze 3.10.12.

Příklad musíte vypracovávat samostatně, bez sdílení kódu mezi sebou a bez přebírání kódu z internetu (myšlenku algoritmů přebírat můžete, implementaci však nikoli) a bez využití jazykových modelů.

Případné dotazy směřujte jako vždy do diskusního fóra.