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. 0 1 2 3 4 5 6 7 8 9

Vaší úlohou bylo pro zadanou gramatiku a slovo rozhodnout, zda gramatika generuje dané slovo. Jelikož je vstupní gramatika regulární, máme garantováno, že každé pravidlo bude jednoho z tvarů $S \to \varepsilon$, $A \to a$, nebo $A \to aB$ a navíc pravidlo tvaru $S \to \varepsilon$ můžeme použít jen pro vygenerování prázdného slova. Můžeme tedy postupně procházet vstupní slovo znak po znaku a zároveň se přesouvat mezi neterminály gramatiky podle toho, kterým pravidlem lze vygenerovat aktuální znak ve slově. Může se nicméně stát, že písmeno na dané pozici lze vygenerovat více způsoby, tj. lze pokračovat různými neterminály. Tento problém můžeme vyřešit například tak, že si budeme udržovat ne jeden neterminál, ale množinu neterminálů, ze kterých můžeme vycházet a které mohly již zpracovaný prefix slova vygenerovat.

V Pythonu to může vypadat například následovně:

```
1  def generates(g : Grammar, w : Word) -> bool:
2      # extract parts of the grammar
3      nonterms, terms, rules, start = g
4
5      # if the input word is empty, check it directly: in that case, there has to
6      # be a rule generating \epsilon from the starting nonterminal
7      if len(w) == 0:
8          for t, n in rules[start]:
9              if n is None and len(t) == 0:
10                 return True
11         return False
12
13     starts = set([start]) # a set of nonterminals to try to generate from
14
15     # now process the non-empty words, one letter at the time
16     for i in range(len(w)):
17         next_starts = set()
18
19         # for each rule of each of the 'starts' nonterminals
20         for start in starts:
21             for t, n in rules[start]:
22                 # if this rule can be used to generate the current letter
23                 if t == w[i]:
24                     # if the rule is a terminal rule and this is the last letter,
25                     # we are done
26                     if n is None and i == len(w) - 1:
27                         return True
28
29                     # if the rule is not a terminal rule, add the target
30                     # nonterminal of the rule to the set of nonterminals for the
31                     # next iteration
32                     if n is not None:
33                         next_starts.add(n)
34
35         # if there were no rules to be used, exit
```

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

```
36          if len(next_starts) == 0:
37              return False
38
39          # go to the next iteration with collected nonterminals
40          starts = next_starts
41      return False
```

Pro zajímavost se můžeme zamyslet nad časovou složitostí našeho algoritmu (terminace plyne triviálně z toho, že všechny cykly mají pevné ohraničení shora). Pro prázdné slovo se provede pouze průchod přes všechna pravidla počátečního neterminálu, počet iterací je v tomto případě tedy shora omezen velikostí gramatiky. Pro neprázdná slova se provádí celkem tři vnořené cykly. Cyklus na řádku 16 má nejvýše $|w|$ iterací. Cyklus na řádku 20 může procházet nejvýše přes všechny neterminály gramatiky a je tedy lineárně omezen velikostí gramatiky. Cyklus na řádku 21 prochází přes pravidla daného neterminálu a je tedy opět lineárně omezen velikostí gramatiky. Dohromady však cykly na řádcích 20 a 21 mohou projít každé pravidlo v gramatice nejvýše jednou a tedy mají dohromady lineární počet iterací vzhledem k velikosti gramatiky.

Celkově tedy dostáváme složitost $\mathcal{O}(|w| \cdot |G|)$, tedy lineární k délce slova a velikosti gramatiky, avšak neuvažujíc složitost operací v jazyce Python. Jediná nekonstantní operace je však přidávání do množiny `next_starts`. Vložení do množiny má v Pythonu konstatní složitost v průměrném případě, avšak lineární v nejhorším. Reálná složitost našeho algoritmu v nejhorším případě je tedy $\mathcal{O}(|w| \cdot |G|^2)$.

Všimněte si také, že kdybychom namísto množiny `next_starts` používali seznam a nekontrolovali v něm duplikáty, mohla by délka tohoto seznamu růst exponenciálně vzhledem k velikosti gramatiky (například pro gramatiku s pravidly $A \to aA \mid aB \mid a, B \to bA \mid bB \mid b$).