МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Лабораторна робота № 11

ВИВЧЕННЯ БІБЛІОТЕКИ ПРИКЛАДНИХ ПРОГРАМ NLTK, ДЛЯ ОПРАЦЮВАННЯ ТЕКСТІВ ПРИРОДНОЮ МОВОЮ. АВТОМАТИЧНИЙ СИНТАКСИЧНИЙ АНАЛІЗ (частина1).

Виконала:
студентка групи ПРЛм-12
Іваськів М.Є.

Прийняв:
Дупак Б.П.

Львів 2015

МЕТА РОБОТИ

- Вивчення основ програмування на мові *Python*.
- Ознайомлення з автоматичним синтаксичним аналізом в NLTK.

КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІ

Речення, як граматичної конструкції, побудованої з одного чи кількох слів певної мови, яка становить окрему, відносно незалежну думку; це значеннєве, граматичне і інтонаційне ціле, що виражає якусь думку в відношенні її до дійсності (предикативність, створена категоріями модальності, часу й особи) одним словом чи сполукою слів.

Мові властиві конструкції, які дозволяють, здається, безмежно розширювати речення. Також вражає те що ми можемо зрозуміти речення довільної довжини, які раніше ніколи не чули. Досить легко придумати зовсім нове речення, яке ніколи раніше ніде в даній природній мові не зустрічалося а всі носії мови його зрозуміють.

Мета граматики – дати явний опис природної мови. Щоб описати мову потрібно визначитись що вважати природною мовою та вивчити основні підходи до її представлення.

Beyond n-grams

Програма генерує цілковито прийнятний текст з невеликої кількості слів але зі збільшенням слів текст перетворюється на несинітницю.

Структура координації: якщо v_1 та v_2 це вирази граматичної категорії X, тоді v_1 *and* v_2 це також вираз категорії X.

Поняття структури складників базується на основі того що слова можуть поєднуватися з іншими словами в окремі самостійні одиниці. Визначити, що послідовність слів виступає окремою одиницею можна на основі ознаки – заміщення, коли послідовність слів в «правильному» реченні може бути замінена більш короткою послідовністю і це не вплине на «правильність» речення.

Кожен вузол дерева (враховуючи слова) називається складником. Оскільки речення можуть бути довільної довжини, то і дерево , яке представляє фразову структуру може мати довільну глибину (кількість рівнів складників).

Проста граматика

Розглянемо просту (найпростішу) контекстно-вільну граматику. Згідно означення, першим символом зліва в першому правилі граматики є спеціальний початковий символ S, і всі дерева повинні мати цей символ , як корінь. В NLTK, контекстно-вільна граматика визначається в модулі nltk.grammar .

Правило подібне до VP -> V NP | V NP PP містить оператор диз'юнкції в правій частині |, і це є скорочений запис двох правил VP -> V NP та VP -> V NP PP.

Оскільки граматика дозволяє побудувати два дерева для цього речення то це речення називають структурно неоднозначним. До неоднозначності приводить двозначність приєднання прийменникового виразу. Прийменниковий вираз PP *in the park* може бути приєднаний до двох місць в дереві: або це донька VP або донька NP. В залежності від місця приєднання прийменникового виразу міняється не тільки синтаксична структура речення але і його зміст.

Розробка власної граматики

При розробці власної контекстно-вільної граматики її правила можна записувати і редагувати у звичайному текстовому файлі, наприклад mygrammar.cfg і використовуючи NLTK завантажувати цю граматику та використовувати її для синтаксичного аналізу.

Потрібно пересвідчитись, що файл має розширення .cfg і відсутні пробіли в стрічці 'file:mygrammar.cfg'. Якщо після виконання оператора print tree на екран нічого не виводиться то це означає що граматика не передбачає (не допускає) побудову такого речення. В такому випадку можна запустити програму синтаксичного аналізу з параметром trace=2: rd_parser = nltk.RecursiveDescentParser(grammar1, trace=2), для відслідковування процесу роботи програми. Також можна перевірити (переглянути) правила граматики, вивівши їх на екран в циклі for p in grammar1.productions(): print p.

Рекурсія в синтаксичних структурах

Граматику називають рекурсивною, якщо категорії з лівої частини її правил також зустрічаються і їх правих частинах, як показано в наступному прикладі. Правило Nom -> Adj Nom містить пряму рекурсію категорії Nom, тоді як не пряма рекурсія S виникає з комбінації двох правил S -> NP VP та VP -> V S.

Синтаксичний аналіз на основі контексно-вільної граматики.

Синтаксичний аналізатор це програма, яка обробляє вхідне речення згідно правил граматики і будує одну або декілька синтаксичних структур, які відповідають (узгоджуються) цій граматиці. Граматика, це декларативна специфікація певних закономірностей - це насправді стрічка (набір стрічок) а не програма. Аналізатор здійснює процедурну інтерпретації граматики – шукає серед лісу дерев, які може породжувати граматика одне потрібне дерево, яке відповідає вхідному реченню.

Алгоритм рекурсивного спуску

Найпростіший тип аналізатора інтерпретує граматику як специфікацію (вказівки) для перетворення (поділу) елемента вищого рівня на декілька елементів нижчого рівня. Початкова задача знайти символ (елемент) S. Правило S → NP VP дозволяє аналізатору замінити (перетворити) цей елемент на два інших елементи: знайти спочатку NP а потім VP. Кожен з цих елементів може бути перетворений на інші елементи на основі правил граматики в яких та є в лівих частинах правил NP та VP. Такий процес буде продовжуватися до тих пір поки не буде поставлена задача знайти (замінити елемент на) слово, наприклад *telescope.* Тоді відбувається порівняння цього слова зі словом зі вхідної послідовності слів і якщо встановлюється відповідність між цими словами то алгоритм продовжує свою роботу. Якщо відповідність не встановлюється то аналізатор повертається на крок назад і пробує розглянути інші варіанти.

RecursiveDescentParser()може мати необов’язковий параметр trace. Якщо значення цього параметру більше нуля то аналізатор виводить на екран результати

Алгоритм переміщення-згортання

Найпростіший аналізатор знизу-вверх це аналізатор переміщення-згортання. Так як і інші аналізатори знизу-вверх цей аналізатор намагається (пробує) знайти послідовності слів і словосполучень (виразів), які відповідають правій частині правила граматики і замінює їх на ліву частину правила, до тих пір поки речення не «згорнеться» до символу S.

В NLTK реалізовано ShiftReduceParser(), як простий аналізатор переміщення - згортання. Ця програма синтаксичного аналізу не передбачає здійснення операцій повернення на крок назад, що не гарантує побудову дерева розбору для тексту, навіть якщо таке дерево існує

Аналізатор left-corner це поєднання аналізаторів знизу – вверх та зверху вниз.

ВИКОНАННЯ ЗАВДАНЬ

Завдання 6. Здійснити аналіз речення з твору A.A. Milne , замінюючи всі прості речення символом S. Намалювати дерево, яке відповідає цій структурі. Які основні синтаксичні конструкції було використано для побудови такого довгого речення?

для побудови цього речення автор використав складносурядні, складнопідрядні, безсполучникові конструкції.

```
sent='''[You can imagine Piglet's joy when at last the ship came in sight of him
abstract=sent.split()
print abstract
>>>
['[You', 'can', 'imagine', "Piglet's", 'joy', 'when', 'at', 'last', 'the', 'ship
', 'came', 'in', 'sight', 'of', 'him.]', 'In', 'after-years', 'he', 'liked', 'to
', 'think', 'that', 'he', 'had', 'been', 'in', 'Very', 'Great', 'Danger', 'durin
g', 'the', 'Terrible', 'Flood,', 'but', 'the', 'only', 'danger', 'he', 'had', 'r
eally', 'been', 'in', 'was', 'the', 'last', 'half-hour', 'of', 'his', 'imprisonm
ent,', 'when', 'Owl,', 'who', 'had', 'just', 'flown', 'up,', 'sat', 'on', 'a', '
branch', 'of', 'his', 'tree', 'to', 'comfort', 'him,', 'and', 'told', 'him', 'a'
, 'very', 'long', 'story', 'about', 'an', 'aunt', 'who', 'had', 'once', 'laid',
'a', "seagull's", 'egg', 'by', 'mistake,', 'and', 'the', 'story', 'went', 'on',
'and', 'on,', 'rather', 'like', 'this', 'sentence,', 'until', 'Piglet', 'who', '
was', 'listening', 'out', 'of', 'his', 'window', 'without', 'much', 'hope,', 'we
nt', 'to', 'sleep', 'quietly', 'and', 'naturally,', 'slipping', 'slowly', 'out',
 'of', 'the', 'window', 'towards', 'the', 'water', 'until', 'he', 'was', 'only',
 'hanging', 'on', 'by', 'his', 'toes,', 'at', 'which', 'moment,', 'luckily,', 'a
', 'sudden', 'loud', 'squawk', 'from', 'Owl,', 'which', 'was', 'really', 'part',
 'of', 'the', 'story,', 'being', 'what', 'his', 'aunt', 'said,', 'woke', 'the',
'Piglet', 'up', 'and', 'just', 'gave', 'him', 'time', 'to', 'jerk', 'himself', '
back', 'into', 'safety', 'and', 'say,', '"How', 'interesting,', 'and', 'did', 's
he?"', 'when', '-', 'well,', 'you', 'can', 'imagine', 'his', 'joy', 'when', 'at'
, 'last', 'he', 'saw', 'the', 'good', 'ship,', 'Brain', 'of', 'Pooh', '(Captain,
', 'C.', 'Robin;', '1st', 'Mate,', 'P.', 'Bear)', 'coming', 'over', 'the', 'sea'
, 'to', 'rescue', 'him...']
```

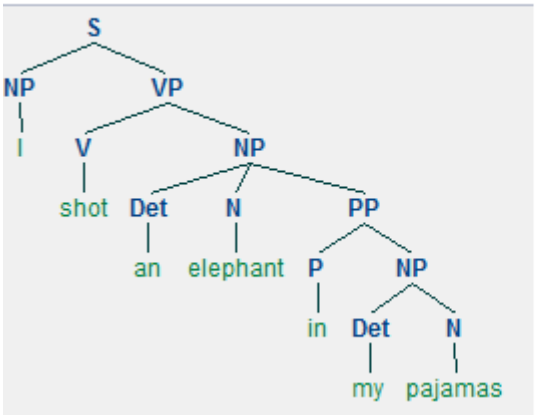
Завдання 2. В класі Tree реалізовано різноманітні корисні методи. Переглянути файл допомоги Tree з документації та описати основні з цих методів (import Tree, help(Tree).

Викликавши файл допомоги Tree, видно, що основним методом є chomsky_normal_form, оскільки його інтерпретації зустрічаються частіше, ніж інших методів. Файл допомоги пояснює принцип побудови дерев.

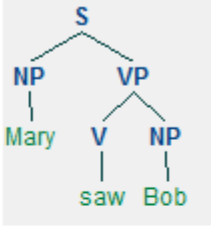
```
>>> import nltk
>>> from nltk import Tree
>>> help(Tree)
Help on class Tree in module nltk.tree:

class Tree(__builtin__.list)
 |  A Tree represents a hierarchical grouping of leaves and subtrees.
 |  For example, each constituent in a syntax tree is represented by a single Tr
ee.
 |
 |  A tree's children are encoded as a list of leaves and subtrees,
 |  where a leaf is a basic (non-tree) value; and a subtree is a
 |  nested Tree.
 |
 |      >>> from nltk.tree import Tree
 |      >>> print Tree(1, [2, Tree(3, [4]), 5])
 |      (1 2 (3 4) 5)
 |      >>> vp = Tree('VP', [Tree('V', ['saw']),
 |      ...                  Tree('NP', ['him'])])
 |      >>> s = Tree('S', [Tree('NP', ['I']), vp])
 |      >>> print s
 |      (S (NP I) (VP (V saw) (NP him)))
 |      >>> print s[1]
 |      (VP (V saw) (NP him))
 |      >>> print s[1,1]
 |      (NP him)
 |      >>> t = Tree("(S (NP I) (VP (V saw) (NP him)))")
 |      >>> s == t
 |      True
 |      >>> t[1][1].node = "X"
 |      >>> print t
 |      (S (NP I) (VP (V saw) (X him)))
 |      >>> t[0], t[1,1] = t[1,1], t[0]
 |      >>> print t
 |      (S (X him) (VP (V saw) (NP I)))
 |
 |  The length of a tree is the number of children it has.
 |
 |      >>> len(t)
 |      2
```

Завдання 4. Перетворити всі дерева , які зустрічаються в методичних вказівка і зображені за допомогою дужок використовуючи nltk.Tree() . Використовувати draw() для побудови графічного зображення дерева.

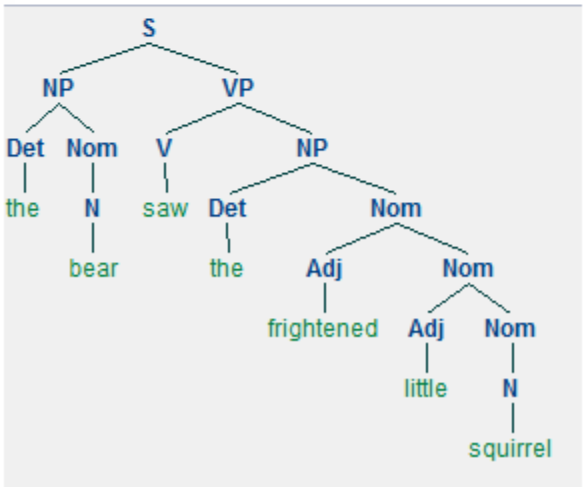
```
import nltk
groucho_grammar = nltk.parse_cfg("""
S -> NP VP
PP -> P NP
NP -> Det N | Det N PP | 'I'
VP -> V NP | VP PP
Det -> 'an' | 'my'
N -> 'elephant' | 'pajamas'
V -> 'shot'
P -> 'in'
""")
sent = ['I', 'shot', 'an', 'elephant', 'in', 'my', 'pajamas']
parser = nltk.ChartParser(groucho_grammar)
trees = parser.nbest_parse(sent)
for tree in trees:
    tree.draw()
```




```
import nltk
grammar1 = nltk.parse_cfg("""
S -> NP VP
VP -> V NP | V NP PP
PP -> P NP
V -> "saw" | "ate" | "walked"
NP -> "John" | "Mary" | "Bob" | Det N | Det N PP
Det -> "a" | "an" | "the" | "my"
N -> "man" | "dog" | "cat" | "telescope" | "park"
P -> "in" | "on" | "by" | "with"
""")
sent = "Mary saw Bob".split()
rd_parser = nltk.RecursiveDescentParser(grammar1)
for tree in rd_parser.nbest_parse(sent):
    tree.draw()
```



```
import nltk
grammar2 = nltk.parse_cfg("""
S  -> NP VP
NP -> Det Nom | PropN
Nom -> Adj Nom | N
VP -> V Adj | V NP | V S | V NP PP
PP -> P NP
PropN -> 'Buster' | 'Chatterer' | 'Joe'
Det -> 'the' | 'a'
N -> 'bear' | 'squirrel' | 'tree' | 'fish' | 'log'
Adj  -> 'angry' | 'frightened' |  'little' | 'tall'
V ->  'chased'  | 'saw' | 'said' | 'thought' | 'was' | 'put'
P -> 'on'
""")
sent = "the bear saw the frightened little squirrel ".split()
parser = nltk.ChartParser(grammar2)
trees = parser.nbest_parse(sent)
for tree in trees:
    tree.draw()
```



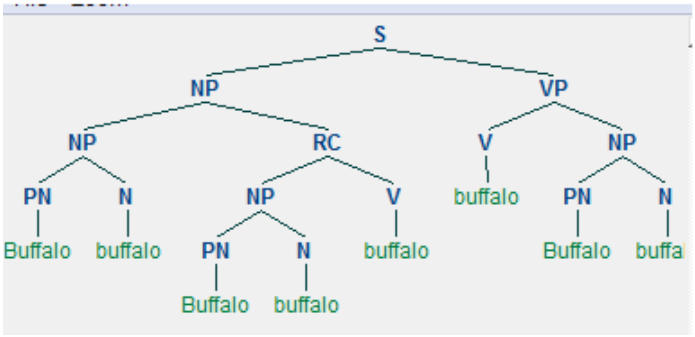
Завдання 8. Написати програму для пошуку відповіді на питання. Чи може grammar1 граматика використовуватися для опису речення довжиною більше ніж 20 слів?

```
import nltk
grammar1 = nltk.parse_cfg("""
S -> NP VP
VP -> V NP | V NP PP
PP -> P NP
V -> "saw" | "ate" | "walked"
NP -> "John" | "Mary" | "Bob" | Det N | Det N PP
Det -> "a" | "an" | "the" | "my"
N -> "man" | "dog" | "cat" | "telescope" | "park"
P -> "in" | "on" | "by" | "with"
""")
sent = "a man saw a dog in the park John walked a cat with a dog in the park Mar
sr_parse = nltk.ShiftReduceParser(grammar1, trace=2)
print sr_parse.parse(sent)
sr_parse = nltk.ShiftReduceParser(grammar1)
print sr_parse.parse(sent)
for p in grammar1.productions():
        print p
import nltk.draw.rdparser
nltk.draw.rdparser.demo()
nltk.draw.srparser.demo()
```

```
Parsing 'a man saw a dog in the park John walked a cat with a dog in the park Ma
ry ate man in the park with the cat on the telescope'
    [ * a man saw a dog in the park John walked a cat with a dog in the park Mar
y ate man in the park with the cat on the telescope]
  S [ 'a' * man saw a dog in the park John walked a cat with a dog in the park M
ary ate man in the park with the cat on the telescope]
  R [ Det * man saw a dog in the park John walked a cat with a dog in the park M
ary ate man in the park with the cat on the telescope]
  S [ Det 'man' * saw a dog in the park John walked a cat with a dog in the park
 Mary ate man in the park with the cat on the telescope]
  R [ Det N * saw a dog in the park John walked a cat with a dog in the park Mar
y ate man in the park with the cat on the telescope]
  R [ NP * saw a dog in the park John walked a cat with a dog in the park Mary a
te man in the park with the cat on the telescope]
  S [ NP 'saw' * a dog in the park John walked a cat with a dog in the park Mary
 ate man in the park with the cat on the telescope]
  R [ NP V * a dog in the park John walked a cat with a dog in the park Mary ate
 man in the park with the cat on the telescope]
  S [ NP V 'a' * dog in the park John walked a cat with a dog in the park Mary a
te man in the park with the cat on the telescope]
  R [ NP V Det * dog in the park John walked a cat with a dog in the park Mary a
te man in the park with the cat on the telescope]
  S [ NP V Det 'dog' * in the park John walked a cat with a dog in the park Mary
 ate man in the park with the cat on the telescope]
  R [ NP V Det N * in the park John walked a cat with a dog in the park Mary ate
 man in the park with the cat on the telescope]
  R [ NP V NP * in the park John walked a cat with a dog in the park Mary ate ma
n in the park with the cat on the telescope]
  R [ NP VP * in the park John walked a cat with a dog in the park Mary ate man
in the park with the cat on the telescope]
  R [ S * in the park John walked a cat with a dog in the park Mary ate man in t
he park with the cat on the telescope]
  S [ S 'in' * the park John walked a cat with a dog in the park Mary ate man in
 the park with the cat on the telescope]
  R [ S P * the park John walked a cat with a dog in the park Mary ate man in th
e park with the cat on the telescope]
  S [ S P 'the' * park John walked a cat with a dog in the park Mary ate man in
the park with the cat on the telescope]
  R [ S P Det * park John walked a cat with a dog in the park Mary ate man in th
e park with the cat on the telescope]
```

Завдання 10. Здійснити аналіз послідовності слів: Buffalo buffalo Buffalo buffalo buffalo buffalo buffalo Buffalo buffalo. Оскільки, згідно з http://en.wikipedia.org/wiki/Buffalo_buffalo_Buffalo_buffalo_buffalo_buffalo_Buffalo_buffalo це граматично правильне речення, напишіть контексно-вільну граматику на основі дерева наведеного на цій сторінці з Інтернету. Здійсніть нормалізацію слів (lowercase), для моделювання ситуації коли слухач сприймає це речення на слух. Скільки дерев розбору може мати це дерево в такому випадку?

```
import nltk
Buffalo_grammar = nltk.parse_cfg("""
S -> NP VP
NP -> NP RC | PN N
VP -> V NP
RC -> NP V
PN -> 'Buffalo'
N -> 'buffalo'
V -> 'buffalo'
""")
sent = "Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo".split()
parser = nltk.ChartParser(Buffalo_grammar)
trees = parser.nbest_parse(sent)
for tree in trees:
    tree.draw()
r="Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo"
print r.lower()
a=r.lower()
b=a.split()
for tree in rd_parser.nbest_parse(b):
        print tree
```



Завдання 12. Написати програму порівняння швидкодії всіх аналізаторів, які згадувалися в методичних. Використовувати timeit функцію для визначення часу синтаксичного аналізу одного і того самого речення різними аналізаторами.

```
import nltk
import timeit
t = timeit.Timer(setup = 'from nltk import ChartParser')
a = t.timeit()
print "ChartParser: ", a

t = timeit.Timer(setup='from nltk import RecursiveDescentParser')
b = t.timeit()
print "RecursiveDescentParser: ", b

t = timeit.Timer(setup='from nltk import ShiftReduceParser')
c = t.timeit()
print "ShiftReduceParser: ", c
>>>
ChartParser:  0.0328198352007
RecursiveDescentParser:  0.030806519601
ShiftReduceParser:  0.0331879485931
```

Завдання 13. Прочитати про "garden path" речення http://en.wikipedia.org/wiki/Garden_path_sentence. Оцінити обчислювальну складність аналізу таких речень в порівнянні з труднощами аналізу таких речень людиною?

Garden path sentences mostly appear in analytic languages, where word order is heavily relied upon to establish the grammatical case and function in a sentence.

Garden path sentences are less common in spoken communication because the prosodic qualities of speech (such as the stress and the tone of voice) often serve to resolve ambiguities in the written text.

ВИСНОВОК

Під час виконання лабораторної роботі я ознайомилася з автоматичним синтаксичним аналізатором NLTK, здійснила аналіз послідовності слів, вивчила контексно-вільну граматику на основі наведеного дерева.