МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»
Кафедра САПР

ЗВІТ

до лабораторної роботи № 11

на тему:

ВИВЧЕННЯ БІБЛІОТЕКИ ПРИКЛАДНИХ ПРОГРАМ NLTK, ДЛЯ ОПРАЦЮВАННЯ ТЕКСТІВ ПРИРОДНОЮ МОВОЮ.

АВТОМАТИЧНИЙ СИНТАКСИЧНИЙ АНАЛІЗ (частина 1)

з дисципліни “Комп’ютерна лінгвістика”

Виконала:

Студентка групи ПРЛм-12

Рибчак Х. В.

Перевірив:

Асистент кафедри САПР

Дупак Б. П.

Львів 2015

МЕТА РОБОТИ

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

КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІ

Лінгвістичні дані та необмежені можливості

При виконанні попередніх лабораторних робіт значна увага приділялася обробці та аналізу текстових корпусів текстів та наголошувалось на проблемах обробки природної мови в зв’язку з лавиноподібною кількість лінгвістичних даних. Припустимо, що можна побудувати корпус, який буде містити все що було сказано чи написано англійською (чи будь-якою іншою мовою) за останні 50 років. Чи справедливо назвати такий корпус – корпусом сучасної англійської мови? Очевидно, що легко знайти багато причин щоб дати негативну відповідь. Якщо здійснити пошук словосполучення «the of» то виявляється можна знайти дуже багато прикладів його вживання.

Спеціалісти, які володіють англійською мовою назвуть такі приклади помилкою, і скажуть що ці приклади не належать англійській мові. Відповідно, не можна вважати «сучасною англійською» велику кількість послідовностей слів з нашого уявного корпуса. Носії мови можуть розглядати такі послідовності і відкидати деякі з них як такі що не є граматичними (не відповідають граматиці природної мови). Звичайно, можна побудувати нове речення і знайти носіїв мови, які скажуть що це речення належить мові.

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

В лабораторній роботі розглядається формальне представлення породжуючої граматики, згідно якої мова представляється, як множина всіх граматично вірних речень, а граматика це формальна система , яка може бути використана для генерації елементів цієї множини.

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.

В цій лабораторній роботі розглядаються два прості алгоритми синтаксичного аналізу, алгоритм рекурсивного спуску, який належить до стратегії зверху-вниз та алгоритм переміщення-згортання який належить до стратегії знизу-вверх синтаксичного аналізу. Також буде розглядатися більш складний алгоритм аналізу, який передбачає здійснення аналізу зверху-вниз з фільтруванням знизу вверх - алгоритм left-corner parser.

ТЕКСТИ ПРОГРАМ НА МОВІ PYTHON

ВАРІАНТ №8

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

```
>>> 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 Tree.
 |
 |  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
```

Рис. 1. Фрагмент файлу допомоги Tree.

```
  |  draw(self)
  |      Open a new window containing a graphical diagram of this tree.


  |  height(self)
  |      Return the height of the tree.
  |
  |          >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP (D the)
(N cat))))")
  |          >>> t.height()
  |          5
  |          >>> print t[0,0]
  |          (D the)
  |          >>> t[0,0].height()
  |          2
  |
  |      :return: The height of this tree.  The height of a tree
  |          containing no children is 1; the height of a tree
  |          containing only leaves is 2; and the height of any other
  |          tree is one plus the maximum of its children's
  |          heights.
  |      :rtype: int
  |

  |  productions(self)
  |      Generate the productions that correspond to the non-terminal nodes of
the tree.
  |      For each subtree of the form (P: C1 C2 ... Cn) this produces a produc
tion of the
  |      form P -> C1 C2 ... Cn.
  |
  |          >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP (D the)
(N cat))))")
  |          >>> t.productions()
  |          [S -> NP VP, NP -> D N, D -> 'the', N -> 'dog', VP -> V NP, V ->
'chased',
  |          NP -> D N, D -> 'the', N -> 'cat']
  |
  |      :rtype: list(Production)
```

Рис. 2. Основні методи класу Tree.

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

File Edit Format Run Options Windows Help

```
import nltk
from nltk import Tree
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):
    print tree
    tree.draw()
```

Рис. 3. Текст програми №4.

```
>>>
(S
  (NP I)
  (VP
    (V shot)
    (NP (Det an) (N elephant) (PP (P in) (NP (Det my) (N pajamas))))))
```

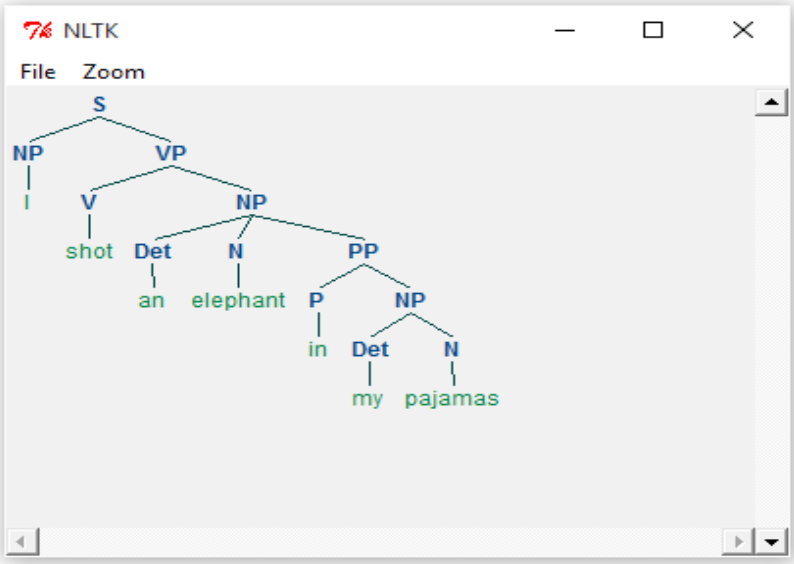


Рис. 4. Реалізація задачі з методичних вказівок.

```
>>>
(S
  (NP I)
  (VP
    (V shot)
    (NP (Det an) (N elephant) (PP (P in) (NP (Det my) (N pajamas))))))
```

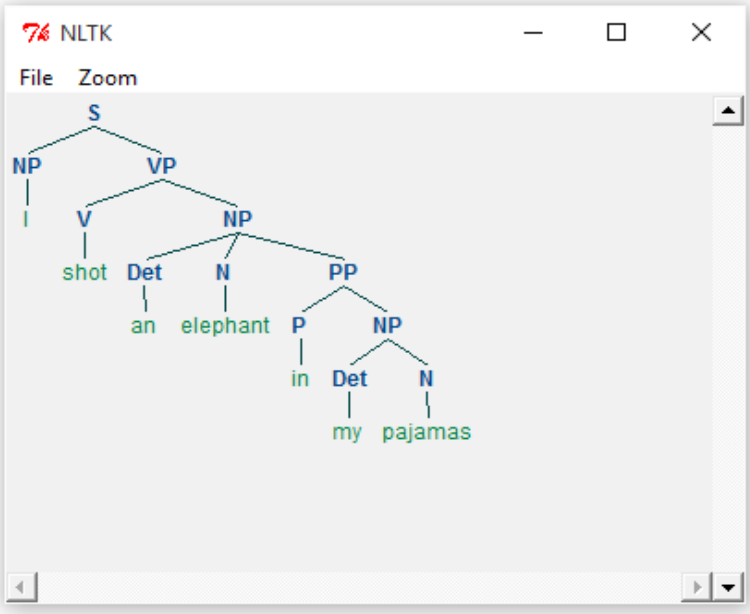


Рис. 5. Реалізація задачі з методичних вказівок.

```
>>>
(S (NP Mary) (VP (V saw) (NP Bob)))
```

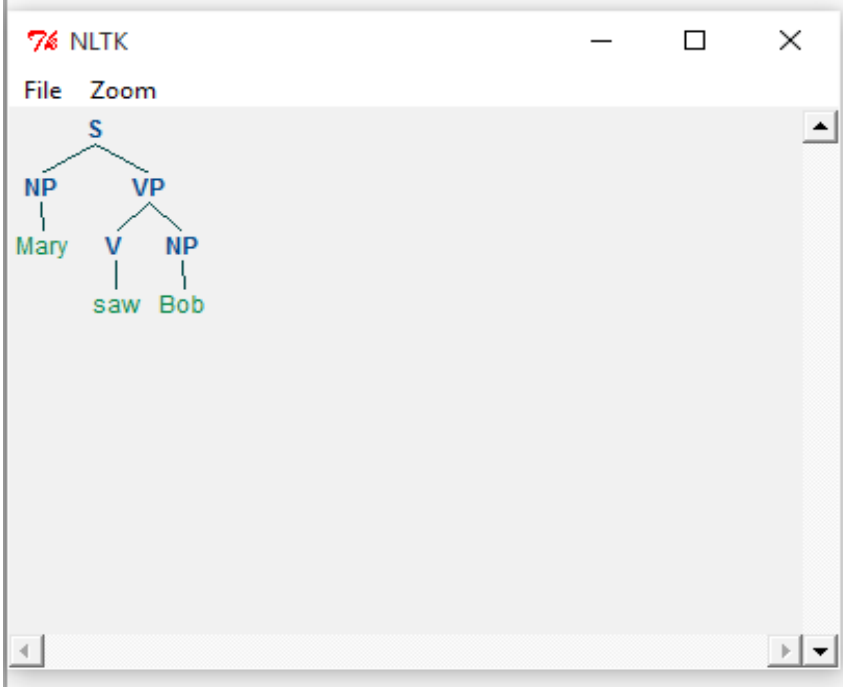


Рис. 6. Реалізація задачі з методичних вказівок.

```
>>>
(S
  (NP (Det the) (N dog))
  (VP
    (V saw)
    (NP (Det a) (N man) (PP (P in) (NP (Det the) (N park))))))
```

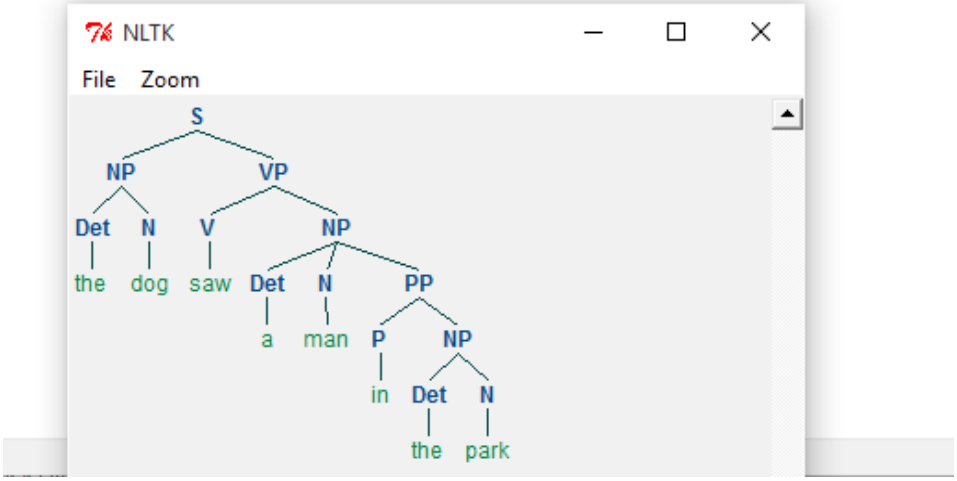
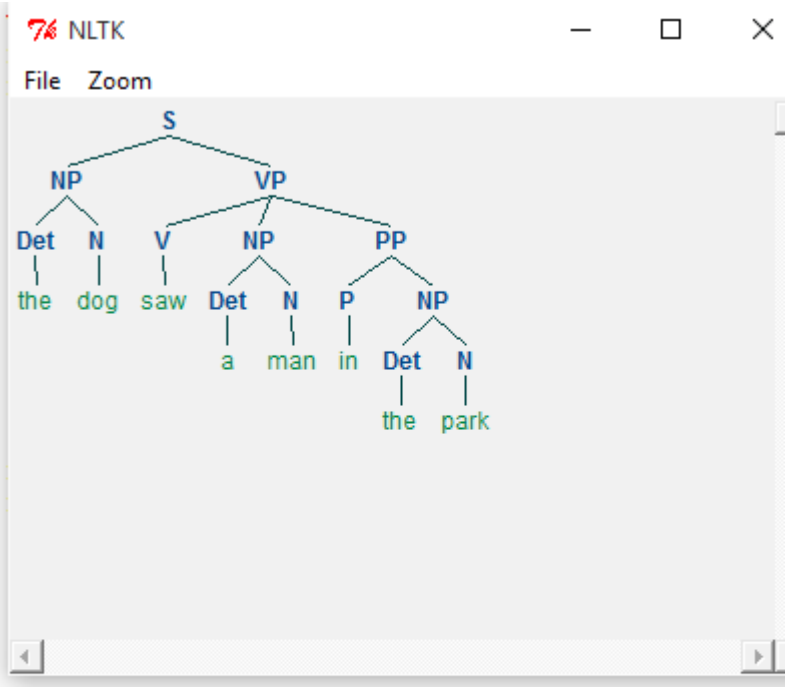


Рис. 7. Реалізація задачі з методичних вказівок.




```
(S
  (NP (Det the) (N dog))
  (VP
    (V saw)
    (NP (Det a) (N man))
    (PP (P in) (NP (Det the) (N park)))))
```

Рис. 8. Реалізація задачі з методичних вказівок.

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?

```
import nltk
from nltk import re
Winnie_the_Pooh = '''In after-years he liked to think that he had been in Very G
abstract = re.split('but|when|which|that|until', Winnie_the_Pooh)
for w in abstract:
    Winnie_the_Pooh = re.sub(w, ' S ', Winnie_the_Pooh)
print Winnie_the_Pooh
```

Ln: 8 Col:

```
>>>
 S that S but S when S until S until S which S which S when S when S
>>>
```

Рис. 9. Текст програми №6.

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 = "Mary saw Bob in the park John walked with a cat in".split()
parser = nltk.ChartParser(grammar1)
trees = parser.nbest_parse(sent)
for tree in trees:
    print tree
    tree.draw()
rd_parser = nltk.RecursiveDescentParser(grammar1)
for t in rd_parser.nbest_parse(sent):
    print t
    t.draw()
sd_parser = nltk.ShiftReduceParser(grammar1)
for s in sd_parser.nbest_parse(sent):
    print s
    s.draw()
```

Ln: 26 Col:

```
>>> ================================ RESTART ================================
>>>
>>>
```

Рис. 10. Текст програми №8.

10. Здійснити аналіз послідовності слів: 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'
""")
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:
    print tree
    tree.draw()
sent2 = "buffalo buffalo buffalo buffalo buffalo buffalo buffalo buffalo".split()
parser2 = nltk.ChartParser(buffalo_grammar)
trees2 = parser2.nbest_parse(sent2)
for tree in trees2:
    print tree
    tree.draw()
```

Рис. 11. Текст програми №10.

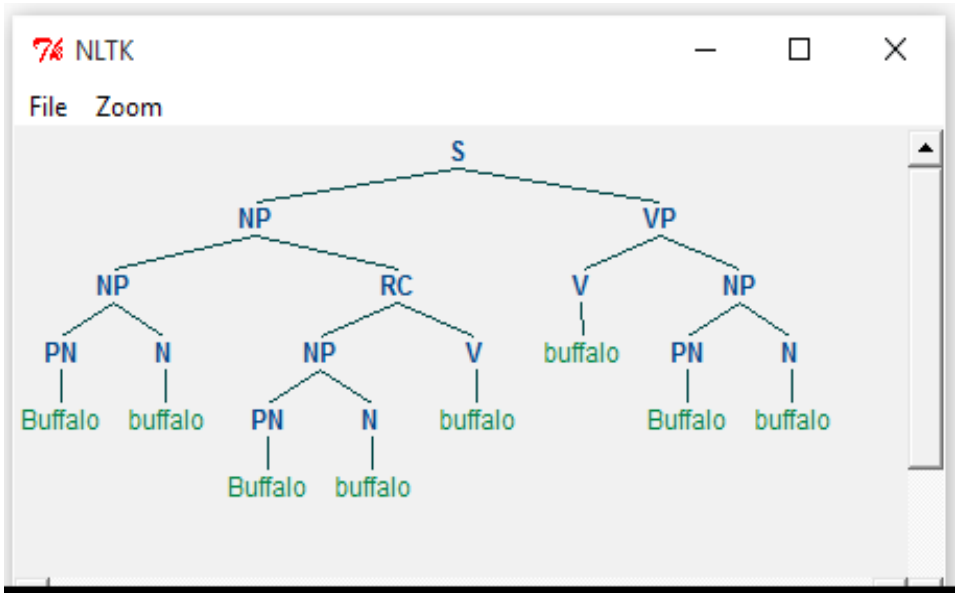
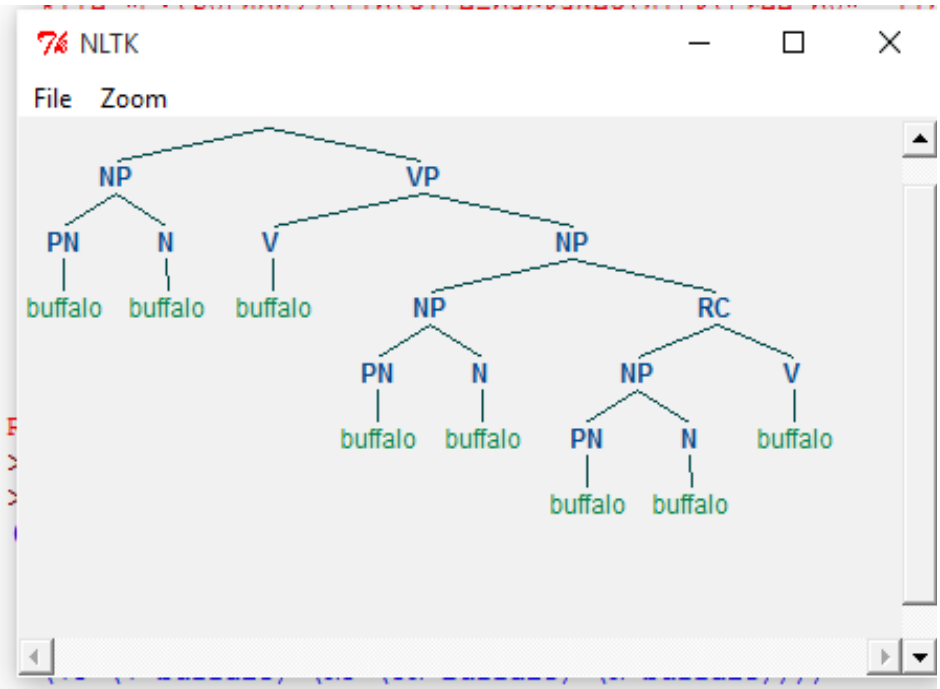
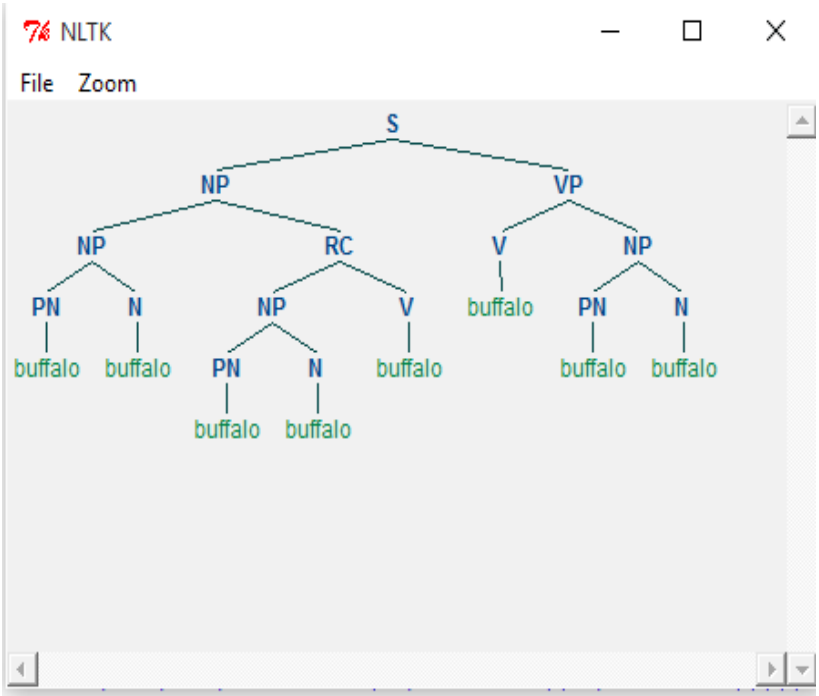


Рис. 12. Результат виконання програми №10.




```
(S
  (NP (PN buffalo) (N buffalo))
  (VP
    (V buffalo)
    (NP
      (NP (PN buffalo) (N buffalo))
      (RC (NP (PN buffalo) (N buffalo)) (V buffalo)))))
```

Рис. 13. Результат виконання програми №10.




```
(S
  (NP
    (NP (PN buffalo) (N buffalo))
    (RC (NP (PN buffalo) (N buffalo)) (V buffalo)))
  (VP (V buffalo) (NP (PN buffalo) (N buffalo))))
```

Рис. 14. Результат виконання програми №10.

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
def timeit(s ,parser):
    import time
    start = time.clock()
    print parser.parse(s)
    return time.clock()-start
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'
""")
parser = nltk.ChartParser(grammar2)
rd_parser = nltk.RecursiveDescentParser(grammar2)
sr_parser = nltk.ShiftReduceParser(grammar2)
sent = "the angry bear chased the frightened little squirrel".split()
ChartParser = timeit(sent, parser)
print 'ChartParser =', ChartParser
print
RecursiveDescentParser = timeit(sent, rd_parser)
print 'RecursiveDescentParser =', RecursiveDescentParser
print
ShiftReduceParser = timeit(sent, sr_parser)
print 'ShiftReduceParser =', ShiftReduceParser
```

Рис. 15. Текст програми №12.

```
>>>
(S
  (NP (Det the) (Nom (Adj angry) (Nom (N bear))))
  (VP
    (V chased)
    (NP
      (Det the)
      (Nom (Adj frightened) (Nom (Adj little) (Nom (N squirrel)))))))
ChartParser = 0.00676654301537

(S
  (NP (Det the) (Nom (Adj angry) (Nom (N bear))))
  (VP
    (V chased)
    (NP
      (Det the)
      (Nom (Adj frightened) (Nom (Adj little) (Nom (N squirrel)))))))
RecursiveDescentParser = 0.0279879314653

(S
  (NP (Det the) (Nom (Adj angry) (Nom (N bear))))
  (VP
    (V chased)
    (NP
      (Det the)
      (Nom (Adj frightened) (Nom (Adj little) (Nom (N squirrel)))))))
ShiftReduceParser = 0.00656561130807
>>>
```

Рис. 16. Результат виконання програми №12.

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" - це граматично правильні речення, які починаються в такий спосіб, що в читачів тлумачення буде неправильним; вони здійснюють неправильний граматичний аналіз, який заводить їх в тупік. Ці речення використовуються в психолінгвістиці."Garden path" відноситься до висловлювання - "управляти садовою доріжкою" тобто "вводити в оману". Згідно з існуючою теорією в психолінгвістиці, коли читач читає "garden path sentences" то він будує структуру значення одного слова в даний момент часу. В певний момент читачеві стає очевидно, що наступне слово чи фраза не може бути вставлено в структуру збудовану до цього часу: це не сумісно з доріжкою якою він управляє. Ці речення є поширені в аналітичній мові, де порядок в значній мірі залежить від встановлення граматичного відмінку в реченні.

Приклади "garden path sentence":

1) The author wrote the novel was likely to be a best-seller.

The author composed the novel...

The author wrote that the novel was likely to be a best-seller.

2) The man returned to his house was happy.

he man came back to his house...

Returned to his house, the man was happy.

3) The government plans to raise taxes were defeated.

The government is planning to raise taxes...

The government's plans to raise taxes were defeated.

Синтаксичний аналіз визначає як діляться фрази в "garden path sentences". Синтаксичний аналіз є загальним терміном, який використовується в психолінгвістиці описуючи розуміння мови. Людина скоріше(правильніше), ніж комп'ютер аналізує речення (визначає частини мови, синтаксичні зв'язки і т.д.)

Синтаксичний аналізатор, так як і людина визначає як поділити речення , але до одного і того ж речення можливо приписувати різні граматичні структури. Наприклад "He ate the cookies on the couch," це може означати, що він їсть печиво яке є на кушетці, або, що він сидить на кушетці в той час коли їсть печиво. Отже, як і людина, так і машина може припуститися помилки, хоча людина правильніше аналізує речення.

ВИСНОВОК

У цій лабораторній роботі я вивчила основи структурного програмування мовою Python та знайомилася з автоматичним синтаксичним аналізом в NLTK.