**МІНІСТЕРСТВО ОСВІТИ І НАУКИ, молоді та спорту УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”**

**іНСТИТУТ КОМП’ютерних НАУК та ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**

### *Кафедра “Системи автоматизованого проектування”*

**Лабораторна робота №12**

**з теми**

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

***АВТОМАТИЧНИЙ СИНТАКСИЧНИЙ АНАЛІЗ (частина2).***

***.***

Виконала:

Ст. гр. ПРЛм-12

Василишин Вікторія

Прийняв:

Асистент кафедри САПР

Дупак Б. П.

Львів 2011

**МЕТА РОБОТА**

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

**КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІ**

### Well-Formed Substring Tables

Прості синтаксичні аналізатори, які розглядалися в попередній лабораторній роботі мають ряд недоліків, які накладають значні обмеження як на ефективність так і взагалі на можливість отримання результатів синтаксичного аналізу. Для вирішення цих проблем використовуються алгоритми що базуються на динамічному програмуванні. Динамічне програмування передбачає збереження проміжних результатів та їх використання при необхідності що дозволяє значно підвищити ефективність роботи різноманітних алгоритмів. Застосування динамічного програмування для синтаксичного аналізу дозволить зберегти часткові рішення при аналізі та використовувати ці рішення для одержання загальних (повних, завершених) результатів синтаксичного аналізу. Одним з алгоритмів що базується на динамічному програмуванні є алгоритм аналізу за допомогою схем (chartparser).

Динамічне програмування дозволяє при синтаксичному аналізі речення речення Ishotanelephantinmypajamas, будувати PPinmypajamas тільки один раз. Як тільки цей складник побудовано він зберігається в таблиці і ним можна скористатися при потребі як безпосереднім складником при побудові NPабоVP. Таку таблицю називають таблицею закономірностей підстрічок (well-formedsubstringtable WFST). Підстрічкою вважається послідовність слів в межах одного речення. Розглянемо побудову такої таблиці на основі стратегії знизу-вверх в якій будуть послідовно збережені всі можливі безпосередні складники.

Розглянемо речення I shot an elephant in my pajamas як вхідні дані. Дане речення можна представити у вигляді, який показано на рис.1 і така структура називається ChartDataStructure.



Рис. 1.Структура речення: словами речення промарковані дуги графа.

В таблицюWFST, записуються позиції слів шляхом заповнення комірок трикутної матриці в якій по вертикалі записуються початкові позиції підстрічок а по горизонталі кінцеві позиції (таким чином слову shotбуде відповідати комірка з координатами (1, 2)). Для спрощення такого представлення, припускається що кожному слову відповідає тільки одна унікальна лексична категорія (тег морфологічних характеристик) і саме вона зберігається в комірці матриці(в комірці (1, 2) міститьсяV). Більш загально, якщо вхідної стрічкою є стрічка a1a2 ... an, і граматика містить правило вигляду A → ai, тоді Aдодається до коміркиl (i-1, i).

Який тег відповідає слову shot з спискуtext, на основі граматики:

>>>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'

... """)

можна знайти наступним чином:.

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | |  | **>>>text = ['I', 'shot', 'an', 'elephant', 'in', 'my', 'pajamas']**  **>>>groucho\_grammar.productions(rhs=text[1])**  **[V -> 'shot']** | |

Для побудовиWFST, створюється матриця розміром(n-1)на(n-1). В Python така матриця за допомогою функції init\_wfst()будується як список списків. Спочатку в цій матриці заповнена тільки центральна діагональ, куди записані теги всіх слів речення.В наступному прикладі показано реалізацію цієї функції та функції display()яка допоможе відображати на екрані результати побудови WFST.

Потрібно звернути увагу на спосіб пошуку правила (продукції) за його правою частиною. У функції complete\_wfstпроіндексовано всю граматику (змінна index) що дозволяє здійснювати пошуку за правою частиною без перебору всіх правил граматики.

Процес роботи алгоритму завершується якщо для вхідної стрічки в комірці з координатами (0, 7) отримано S, що вказує на те що, знайдено синтаксичну структуру ,яка відповідає вхідній послідовності. Кінцевий стан WFSTвідповідає зображенню на рис.2.

Програма побудови WFST це практично простий chartparser, який має декілька недоліків. Перший, таблиця яку ми отримали це не є окреме дерево розбору, а скоріше метод розпізнавання чи речення породжується (допускається) граматикою, а не програма його синтаксичного аналізу. Другий, програма потребує щоб правила граматики, де в правій частині знаходяться не термінали були бінарними. Звичайно, будь-яку контекстно-вільну граматику можна перетворити у таку форму (нормальну форму Хомського), але зручніше працювати без таких додаткових обмежень. Третій, підхід знизу вверх, відзначається великою надлишковістю, коли будує складники (пропонує розмістити не термінальні символи в комірках), які не передбачені граматикою .

**ВАРІАНТ №2**

1.Написати рекурсивну функцію для перегляду дерева, яка визначає його глибину. Дерево з одного вузла має глибину рівну нулю. (глибина піддерева це максимальна глибина його дітей плюс один)

**>>> t = nltk.Tree('(S (NP Alice) (VP chased (NP the rabbit)))')**

**>>> def traverse(t):**

**try:**

**t.node**

**except AttributeError:**

**print t,**

**else:**

**print '(', t.node,**

**for child in t:**

**traverse(child)**

**print ')',**

**>>> t.height()**

**4**

4.Розширити граматику grammar2 з попередньої лабораторної роботи правилами які розділяють прийменники як перехідні, неперехідні та такі що вимагають PP доповнення. На основі цих правил побудуйте дерево розбору для речення Lee ran away home, використовуючи аналізатор рекурсивного спуску.

**>>> grammar2 = nltk.parse\_cfg("""**

**S -> NP VP**

**NP -> Det Nom | PropN|N**

**Nom -> Adj Nom | N**

**VP -> V Adj | V NP | V S | V NP PP|V PP**

**PP -> P NP**

**PropN -> 'Buster' | 'Chatterer' | 'Joe'**

**Det -> 'the' | 'a'**

**N -> 'bear' | 'squirrel' | 'tree' | 'fish' | 'log'|'Lee'|'home'**

**Adj -> 'angry' | 'frightened' | 'little' | 'tall'**

**V -> 'ran'| 'chased' | 'saw' | 'said' | 'thought' | 'was' | 'put'|’give’|’lead’**

**P -> 'on'|'away'**

**""")**

**>>> rd\_parser = nltk.RecursiveDescentParser(grammar2)**

**>>> for tree in rd\_parser.nbest\_parse(sent):**

**print tree**

**(S (NP (N Lee)) (VP (V ran) (PP (P away) (NP (N home)))))**

**>>>**

**grammar2 = nltk.parse\_cfg("""**

**S -> NP VP**

**NP -> Det Nom | PropN|N**

**TP -> PRO NP**

**IP -> PRO ADJ|PRO|PRO ADV**

**PC -> PRO PP**

**DP-> NP PRO NP**

**Nom -> Adj Nom | N**

**VP -> V Adj | V NP | V S | V NP PP|V PP**

**PP -> P NP**

**PropN -> 'Buster' | 'Chatterer' | 'Joe'**

**Det -> 'the' | 'a'**

**N -> 'bear' | 'squirrel' | 'tree' | 'fish' | 'log'|'Lee'|'home'**

**Adj -> 'angry' | 'frightened' | 'little' | 'tall'|'full'**

**ADV -> 'heavily'|'soon'|'already'**

**V -> 'ran'| 'chased' | 'saw' | 'said' | 'thought' | 'was' | 'put'|’give’|’lead’**

**P -> 'on'|'away'|'up with'|'up'**

**""")**

**>>> rd\_parser = nltk.RecursiveDescentParser(grammar2)**

**>>> rd\_parser = nltk.RecursiveDescentParser(grammar2)**

**>>> for tree in rd\_parser.nbest\_parse(sent):**

**print tree**

**(S (NP (N Lee)) (VP (V ran) (PP (P away) (NP (N home)))))**

**>>>**

5. Вибрати декілька (2) загальних дієслова та напишіть програми для вирішення наступних задач:

Пошук дієслів в корпусіPrepositionalPhraseAttachmentCorpusnltk.corpus.ppattach.

Пошук всіх випадків вживання дієслова з двома різними РР в яких перший іменник, або другий іменник або прийменник залишаються незмінними

**>>> entries = nltk.corpus.ppattach.attachments('training')**

**>>> table = nltk.defaultdict(lambda: nltk.defaultdict(set))**

**>>> for entry in entries:**

**key = entry.noun1 + '-' + entry.prep + '-' + entry.noun2**

**table[key][entry.attachment].add(entry.verb)**

**>>> for key in sorted(table):**

**if len(table[key]) > 1:**

**print key, 'N:', sorted(table[key]['N']), 'V:',**

**sorted(table[key]['V'])**

**%-below-level N: ['left'] V:**

**['be']**

**%-from-year N: ['was'] V:**

**['declined', 'dropped', 'fell', 'grew', 'increased', 'plunged', 'rose', 'was']**

**%-in-August N: ['was'] V:**

**['climbed', 'fell', 'leaping', 'rising', 'rose']**

**%-in-September N: ['increased'] V:**

**['climbed', 'declined', 'dropped', 'edged', 'fell', 'grew', 'plunged', 'rose', 'slipped']**

**%-in-week N: ['declined'] V:**

**['was']**

**%-to-% N: ['add', 'added', 'backed', 'be', 'cut', 'go', 'grow', 'increased', 'increasing', 'is', 'offer', 'plummet', 'reduce', 'rejected', 'rise', 'risen', 'shaved', 'wants', 'yield', 'zapping'] V:**

**['fell', 'rise', 'slipped']**

**%-to-million N: ['declining'] V:**

**['advanced', 'climbed', 'cutting', 'declined', 'declining', 'dived', 'dropped', 'edged', 'fell', 'gained', 'grew', 'increased', 'jump', 'jumped', 'plunged', 'rising', 'rose', 'slid', 'slipped', 'soared', 'tumbled']**

**1-to-21 N: ['dropped'] V:**

**['dropped']**

**1-to-33 N: ['gained'] V:**

**['dropped', 'fell', 'jumped']**

**1-to-4 N: ['added'] V:**

**['gained']**

**1-to-47 N: ['jumped'] V:**

**['added', 'rose']**

**1-to-point N: ['ended'] V:**

**['fell', 'rose']**

**3-to-17 N: ['lost'] V:**

**['lost']**

**offer-from-group N: ['rejected'] V:**

**['received']**

8.Здійснити аналіз корпуса Prepositional Phrase Attachment Corpus та спробувати знайти фактори, які впливають на місце приєднання PP.

**>>> import nltk**

**>>> entries = nltk.corpus.ppattach.attachments('training')**

**>>> table = nltk.defaultdict(lambda: nltk.defaultdict(set))**

**>>> for entry in entries:**

**key = entry.verb + '-' + entry.prep + '-' + entry.noun1**

**table[key][entry.attachment].add(entry.verb)**

**>>> for key in sorted(table):**

**if len(table[key]) > 1:**

**print key, 'V:', sorted(table[key]['N']), 'V:',**

**sorted(table[key]['V'])**

**'s-for-one V: ["'s"] V:**

**["'s"]**

**'s-for-place V: ["'s"] V:**

**["'s"]**

**'s-in-nothing V: ["'s"] V:**

**["'s"]**

**Appointed-in-editor V: ['Appointed'] V:**

**['Appointed']**

**accumulating-in-stake V: ['accumulating'] V:**

**['accumulating']**

**acquire-in-stake V: ['acquire'] V:**

**['acquire']**

**add-to-% V: ['add'] V:**

**['add']**

**added-to-% V: ['added'] V:**

**['added']**

**added-to-1 V: ['added'] V:**

**['added']**

**answer-from-questions V: ['answer'] V:**

**['answer']**

**approved-in-million V: ['approved'] V:**

**['approved']**

**are-in-the V: ['are'] V:**

**['are']**

**are-of-a V: ['are'] V:**

**['are']**

**asked-in-court V: ['asked'] V:**

**['asked']**

**attract-in-investment V: ['attract'] V:**

**['attract']……..**

12.Розробити програму обробки дерев корпуса Treebank nltk.corpus.treebank, яка вилучить всі правила з кожного з дерев за допомогою Tree.productions(). Правилами, які зустрічаються тільки один раз можна знехтувати. Правила з однаковими лівими частинами та подібними правими частинами об’єднати для отримання еквівалентного але більш компактного набору правил.

**>>> from nltk.corpus import treebank**

**>>> t = treebank.parsed\_sents('wsj\_0002.mrg')[0]**

**>>> print t**

**(S**

**(NP-SBJ-1**

**(NP (NNP Rudolph) (NNP Agnew))**

**(, ,)**

**(UCP**

**(ADJP (NP (CD 55) (NNS years)) (JJ old))**

**(CC and)**

**(NP**

**(NP (JJ former) (NN chairman))**

**(PP**

**(IN of)**

**(NP (NNP Consolidated) (NNP Gold) (NNP Fields) (NNP PLC)))))**

**(, ,))**

**(VP**

**(VBD was)**

**(VP**

**(VBN named)**

**(S**

**(NP-SBJ (-NONE- \*-1))**

**(NP-PRD**

**(NP (DT a) (JJ nonexecutive) (NN director))**

**(PP**

**(IN of)**

**(NP**

**(DT this)**

**(JJ British)**

**(JJ industrial)**

**(NN conglomerate)))))))**

**(. .))**

**>>> t.productions()**

**[S -> NP-SBJ-1 VP ., NP-SBJ-1 -> NP , UCP ,, NP -> NNP NNP, NNP -> 'Rudolph', NNP -> 'Agnew', , -> ',', UCP -> ADJP CC NP, ADJP -> NP JJ, NP -> CD NNS, CD -> '55', NNS -> 'years', JJ -> 'old', CC -> 'and', NP -> NP PP, NP -> JJ NN, JJ -> 'former', NN -> 'chairman', PP -> IN NP, IN -> 'of', NP -> NNP NNP NNP NNP, NNP -> 'Consolidated', NNP -> 'Gold', NNP -> 'Fields', NNP -> 'PLC', , -> ',', VP -> VBD VP, VBD -> 'was', VP -> VBN S, VBN -> 'named', S -> NP-SBJ NP-PRD, NP-SBJ -> -NONE-, -NONE- -> '\*-1', NP-PRD -> NP PP, NP -> DT JJ NN, DT -> 'a', JJ -> 'nonexecutive', NN -> 'director', PP -> IN NP, IN -> 'of', NP -> DT JJ JJ NN, DT -> 'this', JJ -> 'British', JJ -> 'industrial', NN -> 'conglomerate', . -> '.']**

**Висновок:**

Під час виконання лабораторної роботи я ознайомилася з автоматичним синтаксичним аналізом в NLTK, а саме написала рекурсивну функцію для перегляду дерева, яка визначає його глибину, розширювала граматику grammar2 з попередньої лабораторної роботи правилами які розділяють прийменники як перехідні, неперехідні та такі що вимагають PP доповнення і т.д.