ANLP
Assignment 2

s1719048

s1755650

Task 1

Method 1: CKY. buildIndices

buildIndices(self, productions)

Postcondition: Initialise two dictionaries from the grammar2 rules. Their keys are children, values are parents. Reasons for using dictionary of list: children may have several parents in the grammar, and CKY is a bottom-up algorithm, so we need to know the parents when given children.

How: Starting from the first grammar rule, if the right hand has one child, then append the lhs(left hand side) to the value of unary dictionary with the key rhs(right hand side). If there are two children, then append the lhs to the value of binary dictionary with the key rhs.

:type productions: list(nltk.grammar.Production)
:param productions: the list of grammar rules(nltk.grammar.Production)

Method 2: CKY.unaryFill

unaryFill(self)

Postcondition: Fill the diagonal of the matrix, containing terminal or non-terminal, the non-terminal is the parent of that terminal directly or indirectly.

How: Starting from the first word(token) in the sentence, filling the cells from the upper-left diagonal. Firstly, it adds the terminal, then call the "unaryUpdate" to find all direct or indirect parents and fill in the cell.

Method 3: Cell.unaryUpdate

unaryUpdate(self,symbol,depth=0,recursive=False)

Postcondition: Update a cell in the CKY matrix, find the parents in a recursive way, which means given a child, find its parent and add it as a label. Then treat this parent as a child and find its parent.

How: If the symbol(child) is the key in unary dictionary, then treat its value(parent) as a new key and find the corresponding value(parent), add this value to the label. Then call unaryUpdate recursively to find all related parents.

:type symbol: a string (for terminals) or an nltk.grammar.Nonterminal
:param symbol: a terminal or non-terminal

Method 4: CKY.recognise

recognise(self,tokens,verbose=False)

Postcondition: Initialise a matrix from the sentence, then run the CKY algorithm over it and fill it. The matrix is a list, each item is a row(list) of that matrix. Each cell holds the related terminal or non-terminal.

How: Starting from the upper-left cell from left to right, only append cells to the upper-right part of the matrix. Then fill in each cell using the methods "unaryFill" and "binaryScan".

:type tokens: list(string)
:param tokens: the token list of the sentence
:type verbose: bool
:param verbose: show debugging output if True, defaults to False
:rtype: bool or int
:return: If the input is not recognised, return False. If recognised, return the number of successful analyses.

Method 5: CKY.maybeBuild

maybeBuild (self, start, mid, end)

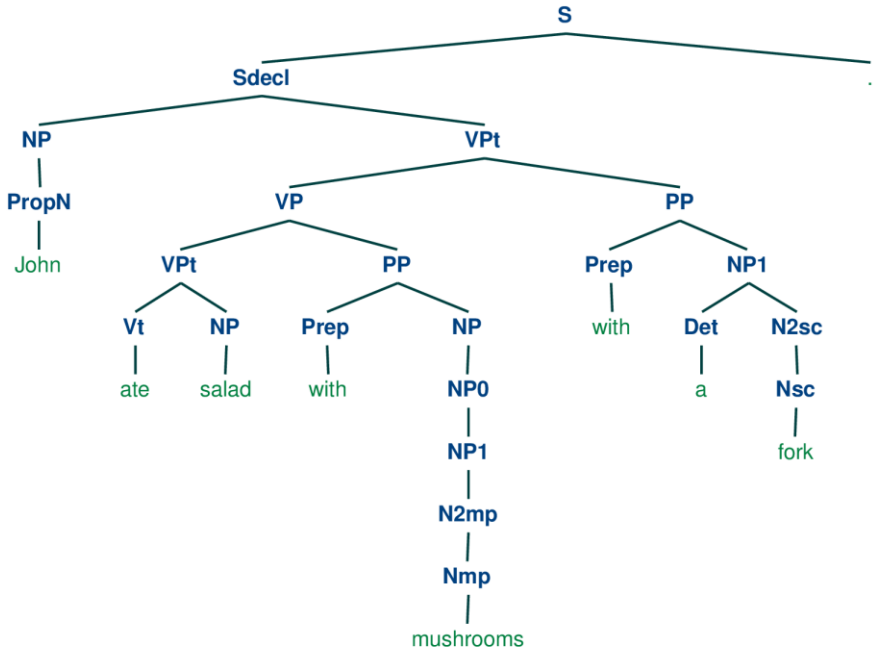
Postcondition: When given two positions: (start, mid) and (mid, end), try to fill in the position(start, end).

How: Starting from the first label of cell[start][mid] and cell[mid][end], try to find their children, if exists, add this child as a label of the new cell[start][end], then call "unaryUpdate" to find all direct and indirect labels.

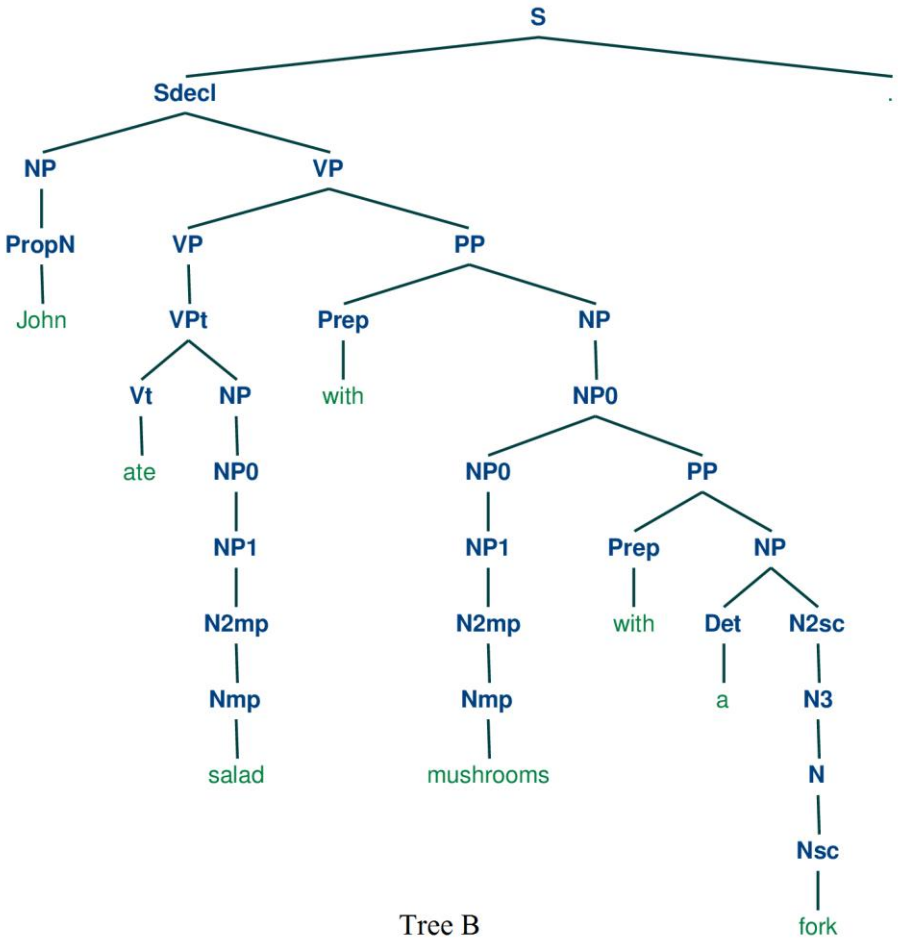
:type start: int
:param start: the row number of one of two parents and the child.
:type mid: int
:param mid: the column number of one parent, also the row number of another parent in the same binary rule.
:type end: int
:param end: the colume number of one of two parents and the child.

Task 2

a) The sentence we chose is: John ate salad with mushrooms with a fork. The drawings of this sentence are showed in Figure 1.



Tree A



Tree B

Figure 1: Figure above is tree A, below is tree B

b) Tree A attaches “a fork” to the end of “ate salad with mushrooms”, which means using a fork to eat salad and mushrooms. It is reasonable. However, Tree B treats “a fork” as a component of “mushrooms”, which means “a fork” is not a tool here, but a part of mushrooms. It is inappropriate.
   We could replace “with a fork” with “in the soup”, then the sentence is like: John ate salad with mushrooms in the soup. Now, Tree A means salad and mushrooms are in the soup, which is weird. While Tree B means eat salad and the mushrooms soup, which is more reasonable than Tree A.

Task 5

Background: CKY is a bottom-up, breadth-first dynamic programming parsing algorithm. According to the grammar rules, we get the children from previous steps and find their parents, then fill the corresponding cell step by step.

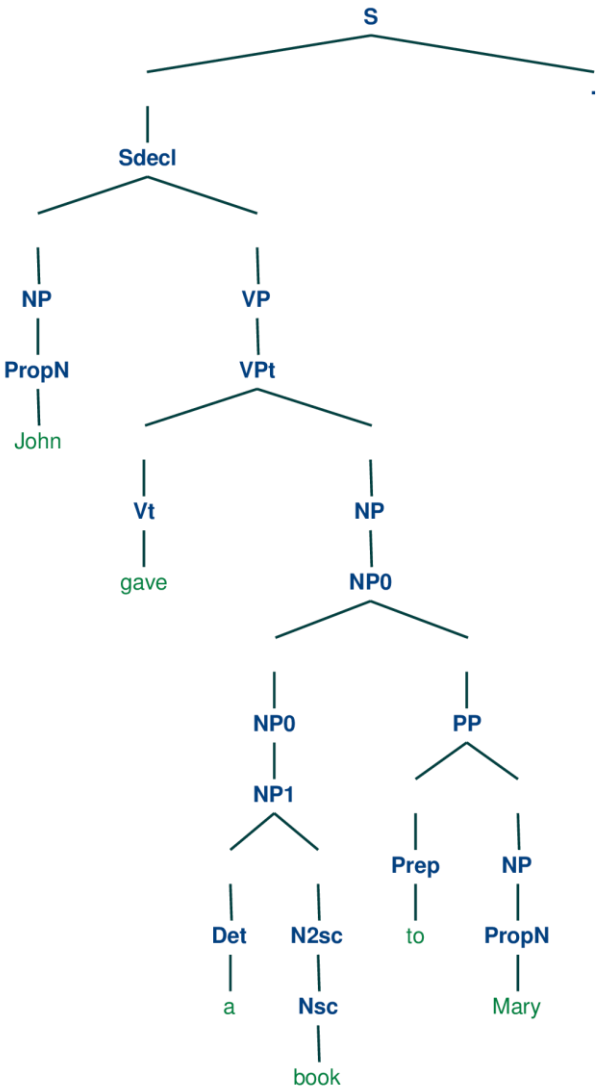
Implementation details:

In order to construct the tree, we need to know where each point is from, which means from the top to the bottom of the tree, we need to know each point’s children and children’s position. So we store the information of position and children in the Label class. Each label is stored in a matrix. Then, we can use the information in this matrix to construct NLTK tree recursively.

We have made several changes in this Task:

1) Edit the class Label, add attributes: children (Label or tuple of Labels) and location(tuple). “children” is the children of this label, “location” is where this label lies in the chart.
2) Edit method “unaryFill” and “maybeBuild” in CKY class, fill the chart with Label instead of string or Noneterminal.
3) Edit method “unaryUpdate” in Cell class, fill the chart with Label instead of string or Noneterminal.
4) Add method “firstTree” to construct a nltk tree.
5) Add method “createTree”, call itself recursively to form a tree string.

Below shows the tree of “John gave a book to Mary.”, which is drawn by firstTree().


S
.
Sdecl
NP
VP
PropN
VPt
John
Vt
NP
gave
NP0
NP0
PP
NP1
Prep
NP
Det
N2sc
to
PropN
a
Nsc
Mary
book