from nltk import CFG

from nltk.parse import ChartParser

from nltk.tokenize import word\_tokenize

nltk.download('punkt')

grammar = CFG.fromstring("""

S -> NP VP

VP -> V NP

NP -> NAME

NP -> ART N

NAME -> 'John'

V -> 'ate'

ART -> 'the'

N -> 'cat'

""")

parser = ChartParser(grammar)

sentence = "John ate the cat."

tokens = word\_tokenize(sentence)[:-1] # Removing the period

print("Original Sentence:")

print(sentence)

print("\nTOP-DOWN PARSING: ")

top\_down\_steps = ["S"] # Start with the root node

np\_count = 0 # Count NP expansions to differentiate them

print(\*top\_down\_steps)

while True:

expanded = top\_down\_steps[-1].split()

for i, symbol in enumerate(expanded):

if symbol == "S":

expanded[i:i+1] = ["NP", "VP"]

np\_count += 1

break

elif symbol == "VP":

expanded[i:i+1] = ["V", "NP"]

np\_count += 1

break

elif symbol == "NP":

if np\_count == 1: # First NP → NAME

expanded[i:i+1] = ["NAME"]

else: # Second NP → ART N

expanded[i:i+1] = ["ART", "N"]

break

elif symbol == "NAME":

expanded[i] = "John"

break

elif symbol == "V":

expanded[i] = "ate"

break

elif symbol == "ART":

expanded[i] = "the"

break

elif symbol == "N":

expanded[i] = "cat"

break

step\_sentence = ' '.join(expanded)

if step\_sentence == top\_down\_steps[-1]: # Stop if no more expansion

break

top\_down\_steps.append(step\_sentence)

print(step\_sentence)

if top\_down\_steps[-1] != ' '.join(tokens):

top\_down\_steps.append(' '.join(tokens))

print(top\_down\_steps[-1])

print("\nBOTTOM-UP PARSING: ")

bottom\_up\_steps = []

current = tokens[:]

while len(current) > 1:

if current[0] == "John":

current[0] = "NAME"

elif current[1] == "ate":

current[1] = "V"

elif current[2] == "the":

current[2] = "ART"

elif len(current) > 3 and current[3] == "cat":

current[3] = "N"

elif len(current) > 3 and current[2] == "ART" and current[3] == "N":

current = current[:2] + ["NP"]

elif len(current) > 2 and current[1] == "V" and current[2] == "NP":

current = ["S"]

bottom\_up\_steps.append(' '.join(current))

print(bottom\_up\_steps[-1])

if current == ["S"]:

break # Stop when the sentence is fully parsed

print("\nTop-Down Parsing Tree:")

for tree in parser.parse(tokens):

print(tree)

tree.pretty\_print()): print(tree) tree.pretty\_print()