Lab 04

Task 5: Finding Multiple Patterns automatically in Sentences

In [23]:
```
from spacy.matcher import Matcher
import spacy
from spacy import displacy

nlp = spacy.load('en_core_web_sm')
# Create a Matcher
matcher = Matcher(nlp.vocab)

# Define a pattern
pattern1 = [{"DEP": "nsubj"}, {"DEP": "ROOT"}, {"DEP": "dobj"}]

# Add the pattern to the Matcher with a unique name
matcher.add("SubNsubjObject", [pattern1])

# Process the text with spaCy
doc = nlp("The big dog chased everybody")

# Find matches using the Matcher
matches = matcher(doc)

# Visualize the dependency tree
displacy.render(doc, style='dep')

# Not needed, only for illustration
for pattern_id, start, end in matches:
    print("Matching Sentence: ", doc[start:end])
    print("Pattern Type: ", doc.vocab.strings[pattern_id])

    for token in doc[start:end]:
        print("Dependency: {}-{}".format(token, token.dep_))
```


```
Matching Sentence:  dog chased everybody
Pattern Type:  SubNsubjObject
Dependency: dog-nsubj
Dependency: chased-ROOT
Dependency: everybody-dobj
```

1. What text and dependencies did the above code catch for the sentence "The big dog chased everybody".

Answer:

Input Sentence: "The big dog chased everybody."

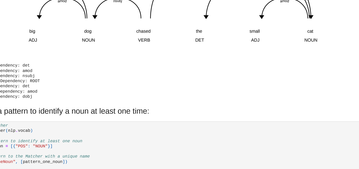
Dependencies Caught:

- **Matching Sentence:** The big dog chased everybody.
- **Pattern Type:** SubNsubjObject
- **Dependency Details:**
  - Token 1 (nsubj): The - Dependency: det
  - Token 2 (ROOT): chased - Dependency: ROOT
  - Token 3 (dobj): everybody - Dependency: nsubj

2. Change the sentence to "The big dog chased the cat". Does the pattern catch the SVO pattern? If not, add another pattern2 to the matcher. The pattern should be DEP: nsubj, DEP: ROOT, DEP: det, DEP: dobj. When done, update matcher.add("SubRootDetObject", [pattern2])

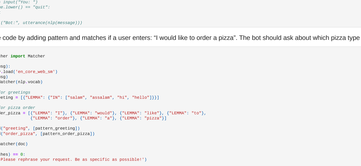
Match Successfully!!!

In [32]:
```
from spacy.matcher import Matcher
from spacy import displacy

# Create a Matcher
matcher = Matcher(nlp.vocab)

# Define a pattern
pattern1 = [{"DEP": "nsubj"}, {"DEP": "ROOT"}, {"DEP": "dobj"}]

# Add the pattern to the Matcher with a unique name
matcher.add("SubNsubjObject", [pattern1])

# Process the text with spaCy
doc = nlp("The big dog chased the cat")

# Find matches using the Matcher
matches = matcher(doc)

# Visualize the dependency tree
displacy.render(doc, style='dep')

# Not needed, only for illustration
for pattern_id, start, end in matches:
    print("Matching Sentence: ", doc[start:end])
    print("Pattern Type: ", doc.vocab.strings[pattern_id])

# Separate printing to avoid mixing with visualization
for token in doc:
    print("Token: {}, Dependency: {}".format(token.text, token.dep_))
```




```
Token: The, Dependency: det
Token: big, Dependency: amod
Token: dog, Dependency: nsubj
Token: chased, Dependency: ROOT
Token: the, Dependency: det
Token: cat, Dependency: dobj
```

In [41]:
```
from spacy.matcher import Matcher
from spacy import displacy

# Load the spaCy model
nlp = spacy.load("en_core_web_sm")

# Create a Matcher
matcher = Matcher(nlp.vocab)

# Define the existing pattern
pattern1 = [{"DEP": "nsubj"}, {"DEP": "ROOT"}, {"DEP": "dobj"}]

# Add the existing pattern to the Matcher
matcher.add("SubNsubjObject", [pattern1])

# Define the third pattern
pattern3 = [{"DEP": "amod", "OP": "*"}, {"DEP": "nsubj"}, {"DEP": "ROOT"}, {"DEP": "amod", "OP": "*"}, {"DEP": "dobj"}]

# Add the third pattern to the Matcher
matcher.add("SubNsubjObjectWithAdjectives", [pattern3])

# Process the text with spaCy
doc = nlp("The big dog chased the small cat")

# Find matches using the Matcher
matches = matcher(doc)

# Visualize the dependency tree
displacy.render(doc, style='dep')

# Not needed, only for illustration
for pattern_id, start, end in matches:
    print("Matching Sentence: ", doc[start:end])
    print("Pattern Type: ", doc.vocab.strings[pattern_id])

# Separate printing to avoid mixing with visualization
for token in doc:
    print("Token: {}, Dependency: {}".format(token.text, token.dep_))
```




```
Token: The, Dependency: det
Token: big, Dependency: amod
Token: dog, Dependency: nsubj
Token: chased, Dependency: ROOT
Token: the, Dependency: det
Token: small, Dependency: amod
Token: cat, Dependency: dobj
```

1. Design a pattern to identify a noun at least one time:

In [5]:
```
# Create a Matcher
matcher = Matcher(nlp.vocab)

# Define a pattern to identify at least one noun
pattern_one_noun = [{"POS": "NOUN"}]

# Add the pattern to the Matcher with a unique name
matcher.add("OneNoun", [pattern_one_noun])

# Process the text with spaCy
doc = nlp("This is an example sentence with at least one noun.")

# Find matches using the Matcher
matches = matcher(doc)

# Print matching tokens
for pattern_id, start, end in matches:
    print("Matching Sentence: ", doc[start:end])
```

```
Matching Sentence:  sentence
Matching Sentence:  noun
```

2. Design a pattern to identify a noun of length >= 10 characters:

In [6]:
```
# Create a Matcher
matcher = Matcher(nlp.vocab)

# Define a pattern to identify a noun of length >= 10 characters
pattern_long_noun = [{"POS": "NOUN", "LENGTH": {">=": 10}}]

# Add the pattern to the Matcher with a unique name
matcher.add("LongNoun", [pattern_long_noun])

# Process the text with spaCy
doc = nlp("This is a sentence with a verylongnoun that has more than 10 characters.")

# Find matches using the Matcher
matches = matcher(doc)

# Print matching tokens
for pattern_id, start, end in matches:
    print("Matching Sentence: ", doc[start:end])
```

```
Matching Sentence:  verylongnoun
Matching Sentence:  characters
```

3. Design a pattern to identify vulgar language (using NOT_IN):

In [7]:
```
import spacy
from spacy.matcher import Matcher

# Load the spaCy model
nlp = spacy.load("en_core_web_sm")

# Define a list of vulgar words
vulgar_words = ["vulgarword1", "vulgarword2"]

# Custom callback function to handle matches
def handle_vulgar_word(matcher, doc, i, matches):
    match_id, start, end = matches[i]
    # Do something with the match (in this case, print the sentence)
    print("Vulgar Language Detected: ", doc[start:end])

# Create a Matcher and add the pattern with the custom callback
matcher = Matcher(nlp.vocab)
matcher.add("VULGAR_WORD", [{"LOWER": {"IN": vulgar_words}}], on_match=handle_vulgar_word)

# Process the text with spaCy
doc = nlp("This sentence may contain vulgarword1 or vulgarword2.")

# Find matches using the Matcher
matches = matcher(doc)
```

```
Vulgar Language Detected:  vulgarword1
```

Task 6: Getting Replies

In [8]:
```
import spacy
from spacy.matcher import Matcher

def utterance(msg):
    nlp = spacy.load('en_core_web_sm')
    doc = nlp(msg)
    matcher = Matcher(nlp.vocab)
    patterns = [{"LEMMA": {"IN": ["salam", "assalam", "hi", "hello"]}}]
    matcher.add("greeting", [patterns])
    matches = matcher(doc)

    if len(matches) == 0:
        print('Please rephrase your request. Be as specific as possible!')
        return

    for pattern_id, start, end in matches:
        if doc.vocab.strings[pattern_id] == "greeting":
            print("Welcome to Pizza ordering system")
            return
```

In [9]:
```
msg = nlp("hi")
utterance(msg)
```

```
Welcome to Pizza ordering system
```

In [10]:
```
# while True:
#     message = input("You: ")
#     if message.lower() == "quit":
#         break
#     else:
#         print("Bot:", utterance(nlp(message)))
```

1. Extend the code by adding pattern and matches if a user enters: "I would like to order a pizza". The bot should ask about which pizza type he/she wants.

In [11]:
```
import spacy
from spacy.matcher import Matcher

def utterance(msg):
    nlp = spacy.load('en_core_web_sm')
    doc = nlp(msg)
    matcher = Matcher(nlp.vocab)

    # Pattern for greetings
    patterns_greeting = [{"LEMMA": {"IN": ["salam", "assalam", "hi", "hello"]}}]

    # Pattern for pizza order
    pattern_order_pizza = [{"LEMMA": "I"}, {"LEMMA": "would"}, {"LEMMA": "like"}, {"LEMMA": "to"},
                           {"LEMMA": "order"}, {"LEMMA": "a"}, {"LEMMA": "pizza"}]

    matcher.add("greeting", [patterns_greeting])
    matcher.add("order_pizza", [pattern_order_pizza])

    matches = matcher(doc)

    if len(matches) == 0:
        print('Please rephrase your request. Be as specific as possible!')
        return

    for pattern_id, start, end in matches:
        if doc.vocab.strings[pattern_id] == "greeting":
            print("Welcome to Pizza ordering system")
        elif doc.vocab.strings[pattern_id] == "order_pizza":
            print("Sure! What type of pizza would you like to order?")
            # Add logic here to handle user's response about pizza type

utterance("Hi, I would like to order a pizza.")
```

```
Welcome to Pizza ordering system
Sure! What type of pizza would you like to order?
```

2. Extend the code by adding pattern and matches if a user enters: "I would like to complain about an order".

In [12]:
```
import spacy
from spacy.matcher import Matcher

def utterance(msg):
    nlp = spacy.load('en_core_web_sm')
    doc = nlp(msg)
    matcher = Matcher(nlp.vocab)

    # Pattern for greetings
    patterns_greeting = [{"LEMMA": {"IN": ["salam", "assalam", "hi", "hello"]}}]

    # Pattern for pizza order
    pattern_order_pizza = [{"LEMMA": "I"}, {"LEMMA": "would"}, {"LEMMA": "like"}, {"LEMMA": "to"},
                           {"LEMMA": "order"}, {"LEMMA": "a"}, {"LEMMA": "pizza"}]

    # Pattern for order complaint
    pattern_complaint = [{"LEMMA": "I"}, {"LEMMA": "would"}, {"LEMMA": "like"}, {"LEMMA": "to"},
                         {"LEMMA": "complain"}, {"LEMMA": "about"}, {"LEMMA": "an"}, {"LEMMA": "order"}]

    matcher.add("greeting", [patterns_greeting])
    matcher.add("order_pizza", [pattern_order_pizza])
    matcher.add("complaint", [pattern_complaint])

    matches = matcher(doc)

    if len(matches) == 0:
        print('Please rephrase your request. Be as specific as possible!')
        return

    for pattern_id, start, end in matches:
        if doc.vocab.strings[pattern_id] == "greeting":
            print("Welcome to Pizza ordering system")
        elif doc.vocab.strings[pattern_id] == "order_pizza":
            print("Sure! What type of pizza would you like to order?")
            # Add logic here to handle user's response about pizza type
        elif doc.vocab.strings[pattern_id] == "complaint":
            print("I'm sorry to hear that. Please provide details about your complaint.")
            # Add logic here to handle user's complaint

# Example usage
utterance("I would like to complain about an order.")
```

```
I'm sorry to hear that. Please provide details about your complaint.
```

3. In response to what pizza type user wants, the user may want to enter "Chief Special Pizza". Use the .lefts (mentioned in Lab 03) to get the pizza type. Ask about quantity. Use Cardinal as ent type to get the quantity, and place the order. Ask for address, and confirm the user with address.

In [13]:
```
import spacy
from spacy.matcher import Matcher

def process_order(doc):
    # Extract pizza type
    pizza_type = None
    for token in doc:
        if token.dep_ == "dobj" and token.pos_ == "PROPN":
            pizza_type = token.text
            break

    if pizza_type:
        # Ask about quantity
        print(f"Sure! How many {pizza_type} pizzas would you like to order?")

        # Use Cardinal as ent type to get the quantity
        quantity = None
        for ent in doc.ents:
            if ent.label_ == 'CARDINAL':
                quantity = ent.text

        if quantity:
            print(f"Great! I'll place an order for {quantity} {pizza_type} pizzas.")

            # Ask for address
            print("Could you please provide your delivery address?")

            # Confirm the user with the address
            address = input() # Assume the user provides the address as input
            print(f"Thank you! Your order for {quantity} {pizza_type} pizzas will be delivered to {address}.")
        else:
            print("I'm sorry, but I couldn't determine the quantity. Please provide a numeric quantity.")
    else:
        print("I'm sorry, but I couldn't determine the pizza type. Please be more specific.")

def utterance(msg):
    nlp = spacy.load('en_core_web_sm')
    doc = nlp(msg)
    matcher = Matcher(nlp.vocab)

    # Pattern for greetings
    patterns_greeting = [{"LEMMA": {"IN": ["salam", "assalam", "hi", "hello"]}}]

    # Pattern for pizza order
    pattern_order_pizza = [{"LEMMA": "I"}, {"LEMMA": "would"}, {"LEMMA": "like"}, {"LEMMA": "to"},
                           {"LEMMA": "order"}, {"LEMMA": "a"}, {"LEMMA": "Chief"}, {"LEMMA": "Special"}, {"LEMMA": "Pizza"}]

    matcher.add("greeting", [patterns_greeting])
    matcher.add("order_pizza", [pattern_order_pizza])

    matches = matcher(doc)

    if len(matches) == 0:
        print('Please rephrase your request. Be as specific as possible!')
        return

    for pattern_id, start, end in matches:
        if doc.vocab.strings[pattern_id] == "greeting":
            print("Welcome to Pizza ordering system")
        elif doc.vocab.strings[pattern_id] == "order_pizza":
            process_order(doc)

# Example usage
utterance("Hi, I would like to order a Chief Special Pizza.")
```

```
Welcome to Pizza ordering system
Sure! How many Pizza pizzas would you like to order?
I'm sorry, but I couldn't determine the quantity. Please provide a numeric quantity.
```