Introduction to Natural Language Processing
Phase 2: Question Answering
Baseline System

- Due Wed night May 20th.
- "Easy" questions
  - Don't expect to get close to 100% precision and recall
  - Get something working that returns a reasonable answer for all the questions
- We'll release HW7 on Thursday along with the first development set (2 more fables & 2 more blogs, though may change)
- HW7 will include MED questions
- We’re preparing an “advanced dev set” that has more examples of MED and hard questions, not the ones that are used in the heldout test sets.
Recall vs. Precision

- How would you get perfect precision?

- How would you get perfect recall?
Generic QA Architecture

Architecture of Typical Q/A System

Question

Question Typing

entity type(s)

Answer Identification

Text(s)

Document/Passage Retrieval

relevant text(s)

Named Entity Tagging

tagged text
HW7: Phase 2 QA System

Next Steps

- Classify questions into types
- Build simple parsers for each question type:
  - Who, what, when, where, etc.
  - E.g., identify type of expected answer
- Take word order into account
- Stem/lemmatize words
- Look for named entities of the proper type near keywords
- Rephrase question

- Use the parse trees and dependency graphs
# Review the PTB Parts of Speech Tags

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>CC</td>
</tr>
<tr>
<td>2.</td>
<td>CD</td>
</tr>
<tr>
<td>3.</td>
<td>DT</td>
</tr>
<tr>
<td>4.</td>
<td>EX</td>
</tr>
<tr>
<td>5.</td>
<td>FW</td>
</tr>
<tr>
<td>6.</td>
<td>IN</td>
</tr>
<tr>
<td>7.</td>
<td>JJ</td>
</tr>
<tr>
<td>8.</td>
<td>JJR</td>
</tr>
<tr>
<td>9.</td>
<td>JJS</td>
</tr>
<tr>
<td>10.</td>
<td>LS</td>
</tr>
<tr>
<td>11.</td>
<td>MD</td>
</tr>
<tr>
<td>12.</td>
<td>NN</td>
</tr>
<tr>
<td>13.</td>
<td>NNS</td>
</tr>
<tr>
<td>14.</td>
<td>NNP</td>
</tr>
<tr>
<td>15.</td>
<td>NNPS</td>
</tr>
<tr>
<td>16.</td>
<td>PDT</td>
</tr>
<tr>
<td>17.</td>
<td>POS</td>
</tr>
<tr>
<td>18.</td>
<td>PRP</td>
</tr>
<tr>
<td>19.</td>
<td>PP$</td>
</tr>
<tr>
<td>20.</td>
<td>RB</td>
</tr>
<tr>
<td>21.</td>
<td>RBR</td>
</tr>
<tr>
<td>22.</td>
<td>RBS</td>
</tr>
<tr>
<td>23.</td>
<td>RP</td>
</tr>
<tr>
<td>24.</td>
<td>SYM</td>
</tr>
<tr>
<td>25.</td>
<td>TO</td>
</tr>
<tr>
<td>26.</td>
<td>UH</td>
</tr>
<tr>
<td>27.</td>
<td>VB</td>
</tr>
<tr>
<td>28.</td>
<td>VBD</td>
</tr>
<tr>
<td>29.</td>
<td>VBG</td>
</tr>
<tr>
<td>30.</td>
<td>VBN</td>
</tr>
<tr>
<td>31.</td>
<td>VBP</td>
</tr>
<tr>
<td>32.</td>
<td>VBZ</td>
</tr>
<tr>
<td>33.</td>
<td>WDT</td>
</tr>
<tr>
<td>34.</td>
<td>WP</td>
</tr>
<tr>
<td>35.</td>
<td>WP$</td>
</tr>
<tr>
<td>36.</td>
<td>WRB</td>
</tr>
<tr>
<td>37.</td>
<td>#</td>
</tr>
<tr>
<td>38.</td>
<td>$</td>
</tr>
<tr>
<td>39.</td>
<td>.</td>
</tr>
<tr>
<td>40.</td>
<td>,</td>
</tr>
<tr>
<td>41.</td>
<td>:</td>
</tr>
<tr>
<td>42.</td>
<td>(</td>
</tr>
<tr>
<td>43.</td>
<td>)</td>
</tr>
<tr>
<td>44.</td>
<td>&quot;</td>
</tr>
<tr>
<td>45.</td>
<td>‘</td>
</tr>
<tr>
<td>46.</td>
<td>”</td>
</tr>
<tr>
<td>47.</td>
<td>’</td>
</tr>
<tr>
<td>48.</td>
<td>”</td>
</tr>
</tbody>
</table>
### Review the PTB Syntactic Tags

**Table 1.2. The Penn Treebank syntactic tagset**

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADJP</td>
<td>Adjective phrase</td>
</tr>
<tr>
<td>ADVP</td>
<td>Adverb phrase</td>
</tr>
<tr>
<td>NP</td>
<td>Noun phrase</td>
</tr>
<tr>
<td>PP</td>
<td>Prepositional phrase</td>
</tr>
<tr>
<td>S</td>
<td>Simple declarative clause</td>
</tr>
<tr>
<td>SBAR</td>
<td>Subordinate clause</td>
</tr>
<tr>
<td>SBARQ</td>
<td>Direct question introduced by <em>wh</em>-element</td>
</tr>
<tr>
<td>SINV</td>
<td>Declarative sentence with subject-aux inversion</td>
</tr>
<tr>
<td>SQ</td>
<td>Yes/no questions and subconstituent of SBARQ excluding <em>wh</em>-element</td>
</tr>
<tr>
<td>VP</td>
<td>Verb phrase</td>
</tr>
<tr>
<td>WHADVP</td>
<td>Wh-adverb phrase</td>
</tr>
<tr>
<td>WHNP</td>
<td>Wh-noun phrase</td>
</tr>
<tr>
<td>WHPP</td>
<td>Wh-prepositional phrase</td>
</tr>
<tr>
<td>X</td>
<td>Constituent of unknown or uncertain category</td>
</tr>
<tr>
<td>*</td>
<td>“Understood” subject of infinitive or imperative</td>
</tr>
<tr>
<td>0</td>
<td>Zero variant of <em>that</em> in subordinate clauses</td>
</tr>
<tr>
<td>T</td>
<td>Trace of <em>wh</em>-Constituent</td>
</tr>
</tbody>
</table>
Review the use of Regular Expressions

<table>
<thead>
<tr>
<th>Operator</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>Wildcard, matches any character</td>
</tr>
<tr>
<td>^abc</td>
<td>Matches some pattern ( abc ) at the start of a string</td>
</tr>
<tr>
<td>abc$</td>
<td>Matches some pattern ( abc ) at the end of a string</td>
</tr>
<tr>
<td>[abc]</td>
<td>Matches one of a set of characters</td>
</tr>
<tr>
<td>[A-Za-z]</td>
<td>Matches one of a range of characters</td>
</tr>
<tr>
<td>editing</td>
<td>Matches one of the specified strings (disjunction)</td>
</tr>
<tr>
<td>*</td>
<td>Zero or more of previous item, e.g. ( a^* ), ([a-z]^* ) (also known as Kleene Closure)</td>
</tr>
<tr>
<td>+</td>
<td>One or more of previous item, e.g. ( a^+ ), ([a-z]^+ )</td>
</tr>
<tr>
<td>?</td>
<td>Zero or one of the previous item (i.e. optional), e.g. ( a? ), ([a-z]? )</td>
</tr>
<tr>
<td>{n}</td>
<td>Exactly ( n ) repeats where ( n ) is a non-negative integer</td>
</tr>
<tr>
<td>{n,}</td>
<td>At least ( n ) repeats</td>
</tr>
<tr>
<td>{,n}</td>
<td>No more than ( n ) repeats</td>
</tr>
<tr>
<td>{m,n}</td>
<td>At least ( m ) and no more than ( n ) repeats</td>
</tr>
<tr>
<td>a(b</td>
<td>c)+</td>
</tr>
</tbody>
</table>
There once was a crow. The crow was sitting on a branch of a tree. Some cheese was in the beak of the crow. A fox observed the crow and tried to discover how to get the cheese. The fox came and stood under the tree. The fox looked toward the crow and said that he saw a noble bird who was above him. The fox said that the beauty of the bird was incomparable. The fox said that the hue of the plumage of the bird was exquisite. The fox said that -- if the sweetness of the voice of the bird is equal to the fairness of the appearance of the bird -- the bird would be undoubtedly the queen of every bird. The crow felt that the fox had flattered her and cawed loudly in order for she to show him that she was able to sing. The cheese fell. The fox snatched the cheese, said that the crow was able to sing and the fox said that the crow needed wits.
The Fox & The Crow

Where was the crow sitting?

There once was a crow. The crow was sitting on a branch of a tree. Some cheese was in the beak of the crow. A fox observed the crow and tried to discover how to get the cheese. The fox came and stood under the tree. The fox looked toward the crow and said that he saw a noble bird who was above him. The fox said that the beauty of the bird was incomparable. The fox said that the hue of the plumage of the bird was exquisite. The fox said that -- if the sweetness of the voice of the bird is equal to the fairness of the appearance of the bird -- the bird would be undoubtedly the queen of every bird. The crow felt that the fox had flattered her and cawed loudly in order for she to show him that she was able to sing. The cheese fell. The fox snatched the cheese, said that the crow was able to sing and the fox said that the crow needed wits.
Syntactic Representations
Constituency Parses and Dependency Parses
Higher Precision Solution

- Chunk the PPs & look for locative preps
- Start using the parse trees

(S
  (NP The/DT (N crow/NN))
  (VP (V was/VBD))
  (VP
   (V sitting/VBG)
   (PP on/IN
    (NP a/DT (N branch/NN)))
   (PP of/IN
    (NP a/DT (N tree/NN))))
  ./.)

natural_language_and_dialogue_systems

http://nlds.soe.ucsc.edu
Stanford Dependencies, Constituents

- LEARN TO USE THE DEMO PAGE. VERY USEFUL.
A fox observed the crow and tried to discover how to get the cheese.
Looking at Stanford’s outputs

Your query

A fox observed the crow and tried to discover how to get the cheese.

Tagging

A/DT fox/NN observed/VBD the/DT crow/NN and/CC tried/VBD

Parse

(ROOT
   (S
      (NP (DT A) (NN fox))
      (VP
         (VP (VBD observed)
            (NP (DT the) (NN crow)))
         (CC and)
         (VP (VBD tried)
            (S
               (VP (TO to)
                  (VP (VB discover)
                     (SBAR
                        (WHADVP (WRB how))
                        (S
                           (VP (TO to)
                              (VP (VB get)
                                 (NP (DT the) (NN cheese))))))))
            (.. .))))

Typed dependencies

det(fox-2, A-1)
nsubj( observed-3, fox-2)
root(ROOT-0, observed-3)
det( crow-5, the-4)
dobj( observed-3, crow-5)
cc( observed-3, and-6)
conj( observed-3, tried-7)
advmod( get-12, how-10)
advmod( get-12, to-11)
ccomp( discover-9, get-12)
det( cheese-14, the-13)
dobj( get-12, cheese-14)
What is syntax good for?

- Increasing Precision

- Some kinds of questions hard to answer from the string because of long distance syntactic dependencies.

  “The people rebelled and created a riot”

- Who created a riot?

- What did the people create?
Two Types of Parse Trees

- **Constituency Parses**
  - Use `nltk.Tree`
  - This is a basic tree
    - N-ary instead of binary

- **Dependency Parses**
  - Use `nltk.DependencyGraph`

- Both have methods for parsing string input
Sample Constituency Tree

```
ROOT
  S
    NP
      DT The
      NN crow
    VBD was
    VP
      VBG sitting
      IN on
      PP
        NP
          DT a
          NN branch
          IN of
            NP
              DT a
              NN tree
```
Stanford Dependencies

- Lots of dependency relations
- Full list and description available at:
  - Should download and USE IT!
- Several different types
  - Basic
    - Form a tree structure
    - Each word is the dependent of another
  - Collapsed
    - More direct relationships between content words
    - Might form a cycle
**root: root**
The root grammatical relation points to the root of the sentence. A fake node “ROOT” is used as the governor. The ROOT node is indexed with “0”, since the indexation of real words in the sentence starts at 1.

```
“I love French fries.”
“Bill is an honest man”
```

**nsbj: nominal subject**
A nominal subject is a noun phrase which is the syntactic subject of a clause. The governor of this relation might not always be a verb: when the verb is a copular verb, the root of the clause is the complement of the copular verb, which can be an adjective or noun.

```
“Clinton defeated Dole”
“The baby is cute”
```

**dobj: direct object**
The direct object of a VP is the noun phrase which is the (accusative) object of the verb.

```
“She gave me a raise”
“They win the lottery”
```

*This is categorized differently in the Stanford Dependency Manual*
REVIEW: Arguments

arg - argument
  agent - agent
  comp - complement
    acomp - adjectival complement
    ccomp - clausal complement with internal subject
    xcomp - clausal complement with external subject
  obj - object
    dobj - direct object
    iobj - indirect object
    pobj - object of preposition
  subj - subject
    nsubj - nominal subject
    nsubjpass - passive nominal subject
    csubj - clausal subject
    csubjpass - passive clausal subject
**REVIEW: Dependencies (Modifiers)**

*aomp: adjectival complement*
An adjectival complement of a verb is an adjectival phrase which functions as the complement (like an object of the verb).

```
She looks very beautiful
```

**amod: adjectival modifier**
An adjectival modifier of an NP is any adjectival phrase that serves to modify the meaning of the NP.

```
“Sam eats red meat”
“Sam took out a 3 million dollar loan”
“Sam took out a $3 million loan”
```

```
amod(meat, red)
amod(loan, dollar)
amod(loan, $)
```

**advmod: adverb modifier**
An adverb modifier of a word is a (non-clausal) adverb or adverb-headed phrase that serves to modify the meaning of the word.

```
“Genetically modified food”
“less often”
```

```
advmod(modified, genetically)
advmod(often, less)
```
Looking at Stanford’s outputs

Your query

A fox observed the crow and tried to discover how to get the cheese.

Tagging

A/DT fox/NN observed/VBD the/DT crow/NN and/CC tried/VBD

Parse

(ROOT
  (S
    (NP (DT A) (NN fox))
    (VP
      (VP (VBD observed)
        (NP (DT the) (NN crow)))
      (CC and)
      (VP (VBD tried)
        (S
          (VP (TO to)
            (VP (VB discover)
              (SBAR
                (WHADVP (WRB how))
              (S
                (VP (TO to)
                  (VP (VB get)
                    (NP (DT the) (NN cheese)))))))))))
  (. .)))

Typed dependencies

det(fox-2, A-1)
nsubj(looked-3, fox-2)
root(ROOT-0, looked-3)
det(crow-5, the-4)
dobj(looked-3, crow-5)
cc(looked-3, and-6)
conj(looked-3, tried-7)
aux(discover-9, to-8)
xcomp(tried-7, discover-9)
advmod(get-12, how-10)
aux(get-12, to-11)
ccomp(discover-9, get-12)
det(cheese-14, the-13)
dobj(get-12, cheese-14)
How do we use the parser output

- **http://nlp.stanford.edu:8080/parser/index.jsp**

How do we use the parser output

- **http://nlp.stanford.edu:8080/parser/index.jsp**

Who tried to discover how to get the cheese?

How do we figure out the subject of “tried to discover”

- S = NP VP

- VP = VP CC VP
Fox observed and tried to discover

- Who tried to discover how to get the cheese?
- How do we figure out the subject of “tried to discover”
- NSUBJ (observed, fox)
- Where is the NSUBJ for “tried”
- CONJ (observed, tried)
Story: Fox observed and set his wits

http://nlp.stanford.edu:8080/parser/index.jsp
Story: Fox observed and set his wits

(NP (DT a) (NN piece))
(PP (IN of))
(NP (NN cheese))
(PP (IN in)
  (NP (PRP$ her) (NN beak))))))))))

(SBAR
  (WHADVP (WRB when))
(S
  (NP (DT a) (NNP Fox))
  (VP
    (VP (VBD observed)
      (NP (PRP her)))
    (CC and)
    (VP (VBD set)
      (NP (PRP$ his) (NNS wits))
    )
  )
(S
  (VP (TO to)
    (VP (VB work))
  )
)
Fox observed and set his wits
Eagle Knocked and Spilled.

Heated with his exertions, the man was about to slake his thirst with a draught from the horn, when the Eagle knocked it out of his hand, and spilled its contents upon the ground.
The constituent Parse: eagle knocked and spilled
The Dependency Tree

det(Eagle-23, the-22)
nsubj(knocked-24, Eagle-23)
rcmod(horn-19, knocked-24)
dobj(knocked-24, it-25)
prt(knocked-24, out-26)
prep(knocked-24, of-27)
poss(hand-29, his-28)
pobj(of-27, hand-29)
cc(knocked-24, and-31)
conj(knocked-24, spilled-32)
poss(contents-34, its-33)
dobj(spilled-32, contents-34)
prep(spilled-32, upon-35)
det(ground-37, the-36)
SCH. A young man long ago crashed the motorbike of the young man on the front yard of a narrator and broke the neck of the young man. The narrator stayed with the young man and didn't aid him because the young man had broken the neck of the young man. The young man died on the spot of the yard of the narrator. The narrator later went back to the spot of the yard of the narrator and decided to talk to the young man because it wanted the young man to know the narrator regretting that it had not aided him. The narrator saw some bright flash in a group of trees that was above the narrator. The narrator thought for the brother of the narrator to use the flashlight of the brother of the narrator. The narrator entered the house of the narrator and heard that the asleep family of the narrator was asleep. The narrator began to wonder that the flash was an orb
More coordination

Who broke the neck of the young man?
Who broke the neck of the young man?
SCH. A summit meeting named G20 summit started on eventful today. G20 summit happened annually. A world and many leader came and talked about it running a government. A people protested because it disagreed about a view. The people protested peacefully on a street. The people rebelled and created riot. The people burned a police car and threw a thing at a police. The police alleviated the people of riot. The police fired a tear gas at the people and fired a bullet at the people, and the people smashed a window.

- Who created a riot?
- Who fired a bullet at the people?
SCH. A summit meeting named G20 summit started on eventful today. G20 summit happened annually. A world and many leader came and talked about it running a government. A people protested because it disagreed about a view. The people protested peacefully on a street.

The people rebelled and created riot. The people burned a police car and threw a thing at a police. The police alleviated the people of riot. The police fired a tear gas at the people and fired a bullet at the people, and the people smashed a window.

- What did the people create?
What did the people create? OBJ | VP NP

Your query

The people rebelled and created riot.

Tagging

The/DT people/NNS rebelled/VBD and/CC

Parse

(Root
 (S
  (NP (DT The) (NNS people))
  (VP (VBD rebelled))
  (CC and)
  (VP (VBD created)
   (NP (NN riot)))
  (.)
)

Typed dependencies

det(people-2, The-1)

nsubj(rebelled-3, people-2)

root(ROOT-0, rebelled-3)

cc(rebelled-3, and-4)

conj(rebelled-3, created-5)

dobj(created-5, riot-6)
Adverbials: When did the young man die?

"years" can answer a when question
When adverbials can answer a when question
Data Structures for Parse Trees
Two Types of Parse Trees

- Constituency Parses
  - Use nltk.Tree
  - This is a basic tree
    - N-ary instead of binary

- Dependency Parses
  - Use nltk.DependencyGraph

- Both have methods for parsing string input
Additional resources for assignment

- assignment6-stub.py (3 KB; May 9, 2015 11:51 am)
- baseline-stub.py (2 KB; May 9, 2015 11:51 am)
- chunk-demo.py (4 KB; May 9, 2015 11:51 am)
- parse-demo.py (3 KB; May 9, 2015 11:51 am)
- constituency-demo-stub.py (3 KB; May 9, 2015 11:51 am)
- dependency-demo-stub.py (3 KB; May 9, 2015 11:51 am)
- score-answers.pl (13 KB; May 9, 2015 11:52 am)
- fables-01_my_answers.txt (1 KB; May 9, 2015 11:52 am)
- hw6_dataset.zip (20 KB; May 9, 2015 11:53 am)
- hw6-v3.pdf (125 KB; May 9, 2015 11:53 am)
Constituent Parses .par files

- USE the TREE READER to convert

```plaintext
QuestionId: blogs-01-1
(ROOT (SBARQ (WHADYP (WRB When))) (SQ (VBD did) (NP (DT the) (NN G20) (NN summit)) (VP (VB start))) (. ?)))

QuestionId: blogs-01-10
(ROOT (SBARQ (WHNP (WP What))) (SQ (VP (VBD happened) (PP (TO to) (NP (DT the) (NN police) (NNS cars))))) (. ?)))

QuestionId: blogs-01-11
(ROOT (SBARQ (WHNP (WP What))) (SQ (VBD was) (VP (VBN fired))) (. ?)))
```
Constituency Tree

- On disk it looks like this:

(ROOT (S (NP (DT The) (NN crow)) (VP (VBD was) (VP (VBG sitting) (PP (IN on) (NP (NP (DT a) (NN branch)) (PP (IN of) (NP (DT a) (NN tree))))))))) (. .))

- In our dataset it's all on one line

- It doesn't have to be but it makes reading it in easier
Reading in Constituency Trees

- Easy to read
- Each parse is on a single line
Reading in Constituency Trees

```python
# Read the constituency parse from the line and construct the Tree
def read_con_pares(parfile):
    fh = open(parfile, 'r')
    lines = fh.readlines()
    fh.close()
    return [Tree.fromstring(line) for line in lines]
```

- Easy to read
- Each parse is on a single line
- Voila!
# Read the constituency parse from the line and construct the Tree

def read_con_parses(parfile):
    fh = open(parfile, 'r')
    lines = fh.readlines()
    fh.close()
    return [Tree.fromstring(line) for line in lines]

- Easy to read
- Each parse is on a single line
- Voila!
Manipulating a Constituency Tree

- **Basic Operations**
  - `[]` – access the children
  - `subtrees(filter)` – get all subtrees optionally filtering only ones that meet a criteria

- **Parented Tree Operations**
  - `parent()`,
  - `parent_index()`,
  - `left_sibling()`,
  - `right_sibling()`,
  - `root()`,
  - `treeposition()`
Manipulating a Constituency Tree
Manipulating a Constituency Tree

```
ROOT
  S
  VP
    NP
      DT The
      NN crow
      VBD was
    VP
      VBG sitting
      IN on
    PP
      NP
        DT a
        NN branch
        IN of
      NP
        DT a
        NN tree
```
Manipulating a Constituency Tree

The diagram shows a constituency tree with the root node labeled as 'ROOT'. The tree structure is as follows:

- The root node 'S' has two children: 'NP' and 'VP'.
- The 'NP' node has children 'DT', 'NN', and 'VBD' labeled as 'The', 'crow', and 'was', respectively.
- The 'VP' node has children 'VBG', 'IN', and 'PP'.
  - 'VBG' is highlighted, indicating it is the target for manipulation.
  - The 'IN' node is labeled 'sitting', and the 'PP' node is labeled 'on'.
- The 'PP' node has children 'NP' and 'IN'.
  - The 'NP' node is labeled 'a branch of', and the 'IN' node is labeled 'a tree'.

The term 'right_sibling' is marked next to the highlighted 'VBG' node, indicating that it can be manipulated as part of its right sibling relation.
Manipulating a Constituency Tree

None

left_sibling
Manipulating a Constituency Tree

The diagram represents a constituent tree with the root labeled as "VBG". The tree structure breaks down the constituent parts of the sentence "The crow was sitting on a branch of a tree."
Dependency Parses .dep files

QuestionId: blogs-01-1
When  WRB 6  advmod
did   VBD 6  aux
det
the  DT 5  det
g20nn 6  npn
summitNN 6  nsubj
startVB 0  root

QuestionId: blogs-01-10
What WP 2 nsubj
happenedVBD 0 root
toTO 2 prep
det
det
theDT 6  det
carsNN 6  nn
cars

QuestionId: blogs-01-11
What WP 3 nsubjpass
wasVBD 3 auxpass
firedVBN 0 root
Reading in Dependency Graphs

- Each DependencyGraph consists of a list of nodes
  - `nodes.values()`

- Each node is a dict with the following keys
  - `head`: index of the parent (the root doesn't have a head)
  - `word`: the lexical item
  - `rel`: the grammatical relation between the item and the head
  - `tag`: the part of speech tag of the node
  - `deps`: the list of dependent nodes
  - `address`: index of the item in the sentence (starting from 1)
  - `lemma`: the lemma of the lexical item
  - `feats`: (new to NLTK 3.0 – don’t need to use)
  - `ctag`: (new to NLTK 3.0 – don’t need to use)
# Read the lines of an individual dependency parse

def read_dep(fh):
    dep_lines = []
    for line in fh:
        line = line.strip()
        if len(line) == 0:
            return '\n'.join(dep_lines)
        dep_lines.append(line)

    return '\n'.join(dep_lines) if len(dep_lines) > 0 else None
Read a Dependency Parse

```python
{0: {'address': 0,
     'ctag': 'TOP',
     'deps': defaultdict(<class 'list'>, {'ROOT': [4]}),
     'feats': None,
     'lemma': None,
     'rel': 'TOP',
     'tag': 'TOP',
     'word': None},
1: {'address': 1,
    'ctag': 'DT',
    'deps': defaultdict(<class 'list'>, {}),
    'feats': '',
    'head': 2,
    'lemma': 'The',
    'rel': 'det',
    'tag': 'DT',
    'word': 'The'},
2: {'address': 2,
    'ctag': 'NN',
    'deps': defaultdict(<class 'list'>, {'det': [1]}),
    'feats': '',
    'head': 4,
    'lemma': 'crow',
    'rel': 'nsubj',
    'tag': 'NN',
    'word': 'crow'}, ....
```
Parse Tree Demo
Simple Subtree Match

- Given our tree
  - Does it contain the subtree?
  - \((VP \ (\ast)\ (PP))\)
  - \(\ast\) is a wildcard (can be anything)
Matching Algorithm

def matches(pattern, root):
    if root is None and pattern is None: return root
    elif pattern is None: return root
    elif root is None: return None

    plabel = pattern if isinstance(pattern, str) else pattern.label()
    rlabel = root if isinstance(root, str) else root.label()

    if plabel == "*":
        return root
    elif plabel == rlabel:
        for pchild, rchild in zip(pattern, root):
            match = matches(pchild, rchild)
            if match is None:
                return None
        return root

    return None