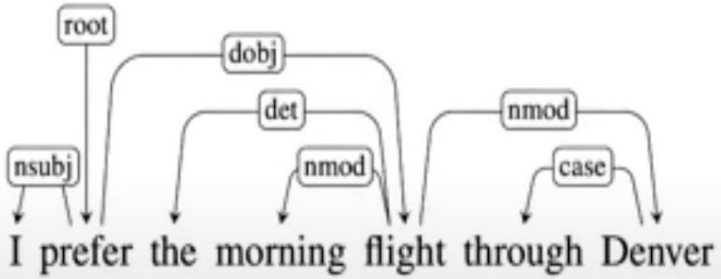
Dependency Parsing

Syntactical analysis

Dependency Parsing

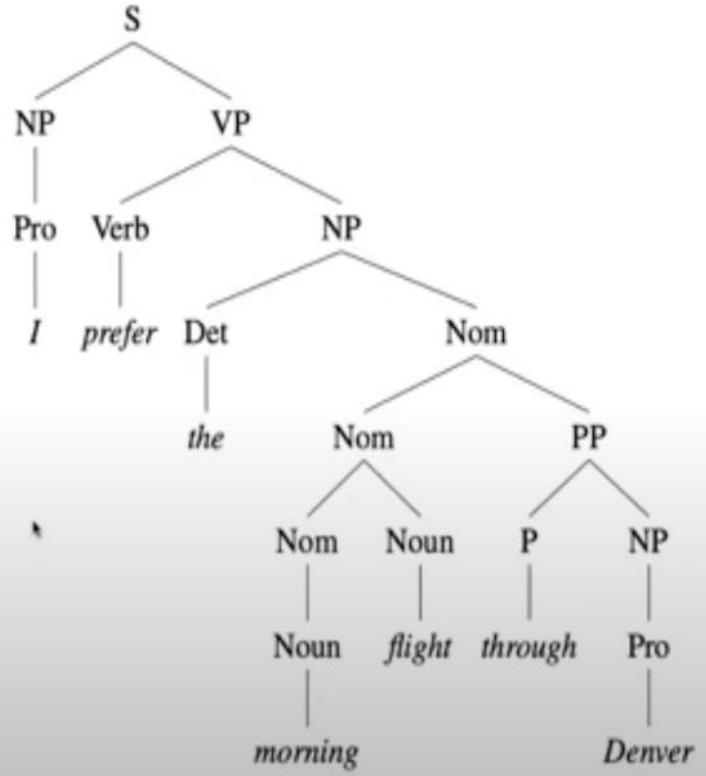
So far, we've thought about language statistically, or as a set of rules specified by a CFG; **Dependency grammars** provides a another representation of language as a graph: nodes are words, edges are dependencies.



heads and dependents

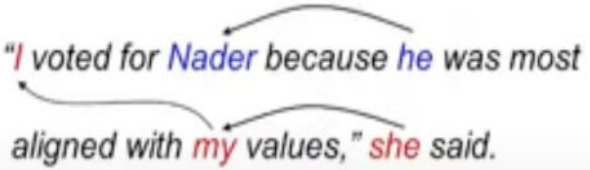
Dependency Parsing

In this way, a dependency grammar approach abstracts away from word-order information, representing only the information that is necessary for the parse.



Dependency Parsing

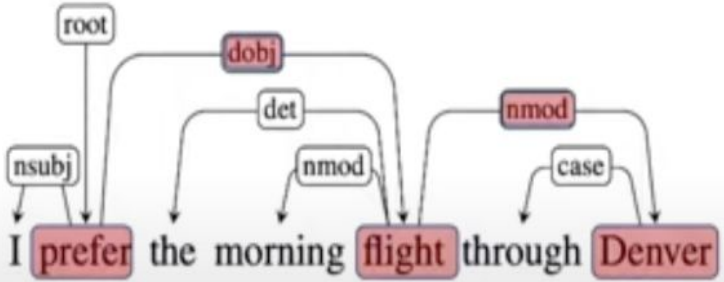
It is particularly important in contemporary speech and language processing systems for **coreference resolution**, question answering and information extraction.



Dependency Parsing

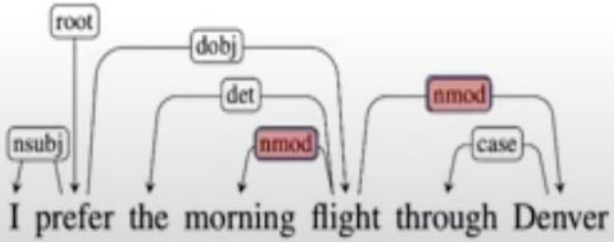
Dependency structures are represented by directed graphs that satisfy the following constraints:

1. There is a single designated root node that has no incoming arcs.
2. With the exception of the root node, each vertex has exactly one incoming arc.
3. There is a unique path from the root node to each vertex in V.



Dependency Parsing

Graphs may be represented by an adjacency **tensor** for analysis.



nmod	I	prefer	the	morning	flight	through	Denver
I							
prefer							
the							
morning							
flight				1			1
through							
Denver							

Dependency Parsing

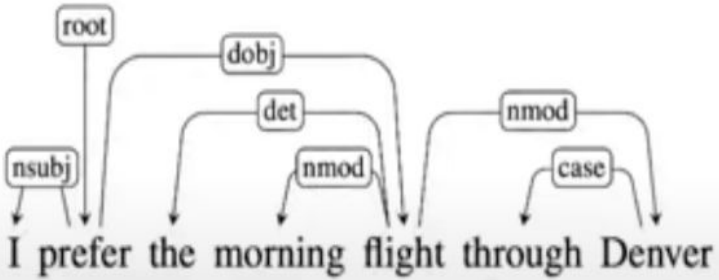
Peter Piper picked a peck of pickled peppers

Dependency tree?

Dependency Parsing

There are multiple algorithms that generate dependency trees from data:

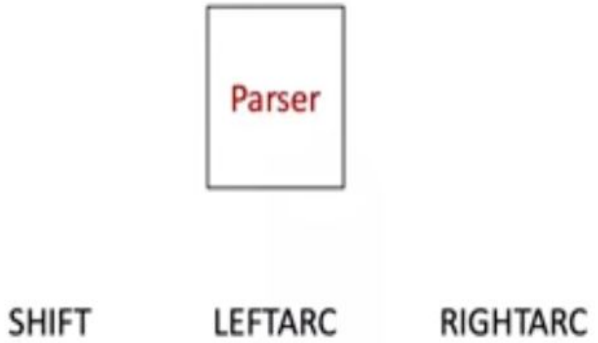
- **Transition-based Approaches**
  - Shift-reduce
- **Graph Algorithms**
  - Maximum Spanning Tree



Dependency Parsing

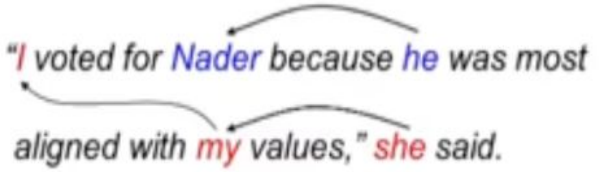
Arc-standard algo

State-of-the-art transition-based systems use supervised machine learning methods to train classifiers that perform the parsing. Given appropriate training data, these methods learn a function that maps from configurations to transition operators.



Dependency Parsing

The **Universal Dependencies project**, and other Treebanks provide an inventory of dependency relations generated by human annotators; these can be used as training data for models that automate graph generation.



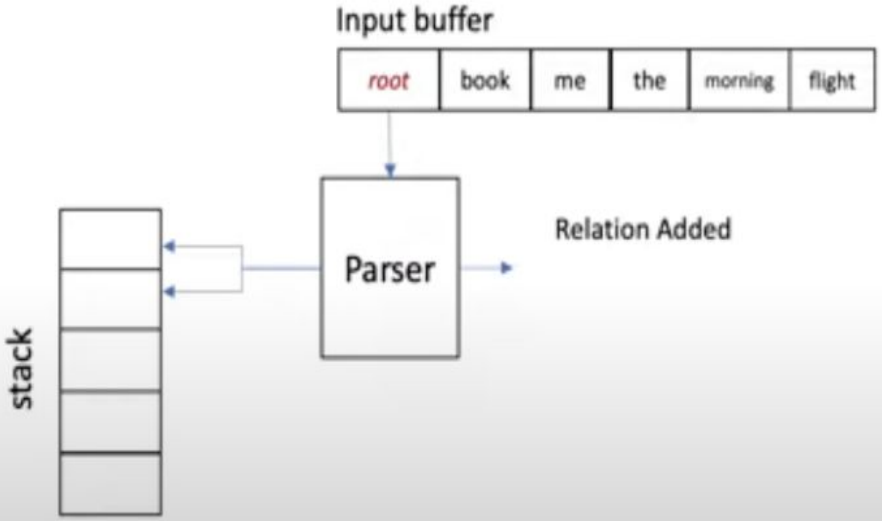
Dependency Parsing

Note that some algorithms are **projective**: the edges in the graph are not allowed to cross. While most English sentences are projective, they're not in all cases:

Dependency Parsing

In arc-standard approaches we step across the *n* words in the **Input buffer** from left to right; and for each for word w_i, our **parser** can:

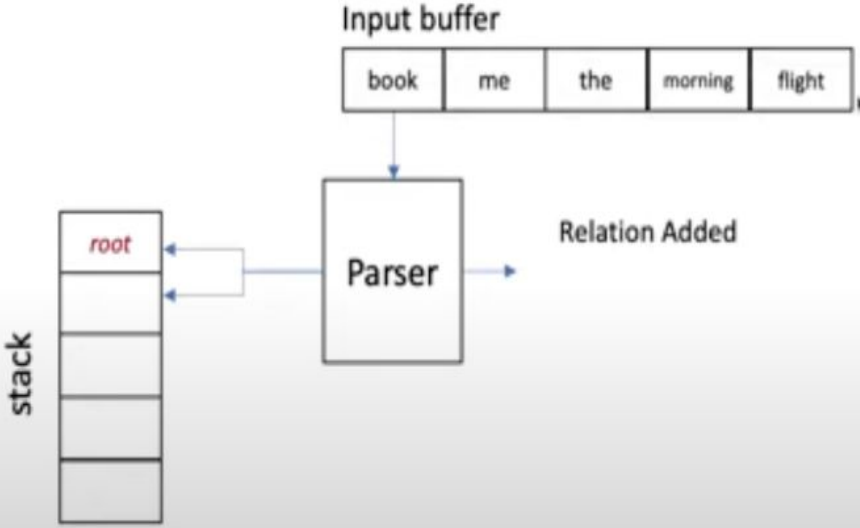
- SHIFT: Put a word in a **stack** for analysis
- LEFTARC: Assign the word in the top of the stack as head of the previous word in the stack
- RIGHTARC: Assign the previous word in the stack as head of the word at the top of the stack



Dependency Parsing

In arc-standard approaches we step across the *n* words in the **Input buffer** from left to right; and for each for word w_i, our **parser** can:

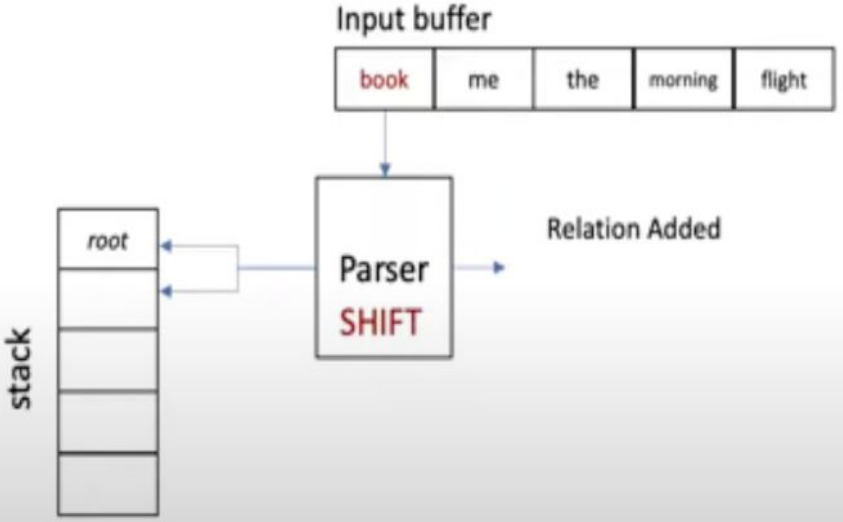
- SHIFT: Put a word in a **stack** for analysis
- LEFTARC: Assign the word in the top of the stack as head of the previous word in the stack
- RIGHTARC: Assign the previous word in the stack as head of the word at the top of the stack



Dependency Parsing

In arc-standard approaches we step across the *n* words in the **Input buffer** from left to right; and for each for word w_i, our **parser** can:

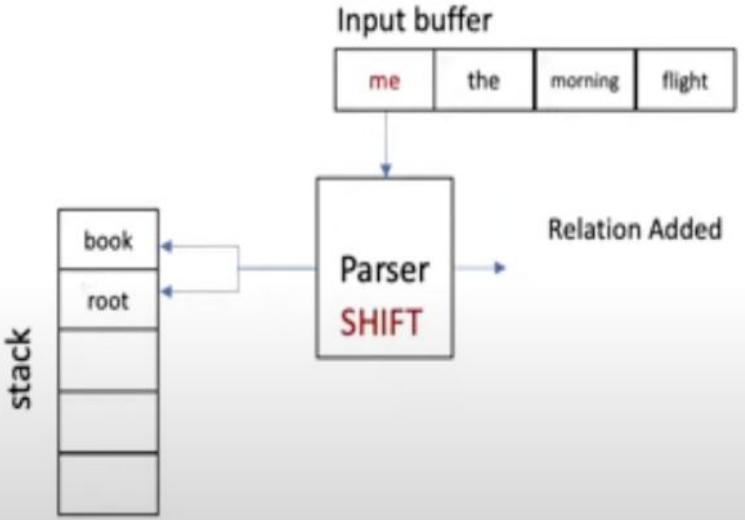
- SHIFT: Put a word in a **stack** for analysis
- LEFTARC: Assign the word in the top of the stack as head of the previous word in the stack
- RIGHTARC: Assign the previous word in the stack as head of the word at the top of the stack



Dependency Parsing

In arc-standard approaches we step across the *n* words in the **Input buffer** from left to right; and for each for word w_i, our **parser** can:

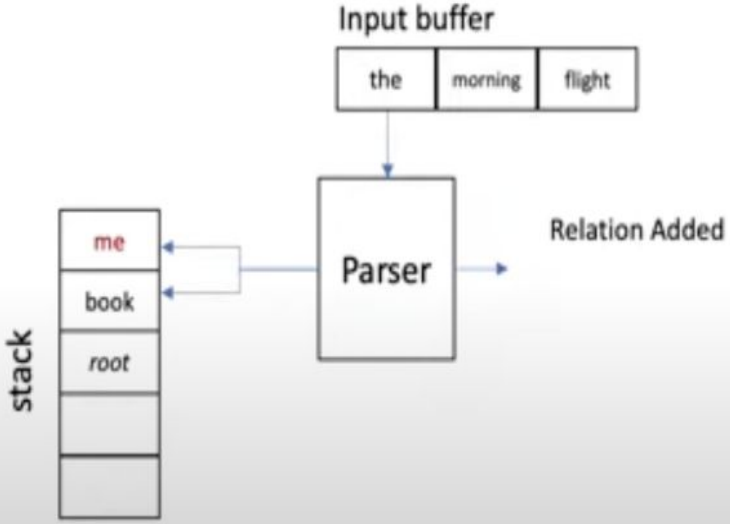
- SHIFT: Put a word in a **stack** for analysis
- LEFTARC: Assign the word in the top of the stack as head of the previous word in the stack
- RIGHTARC: Assign the previous word in the stack as head of the word at the top of the stack



Dependency Parsing

In arc-standard approaches we step across the *n* words in the **Input buffer** from left to right; and for each for word w_i, our **parser** can:

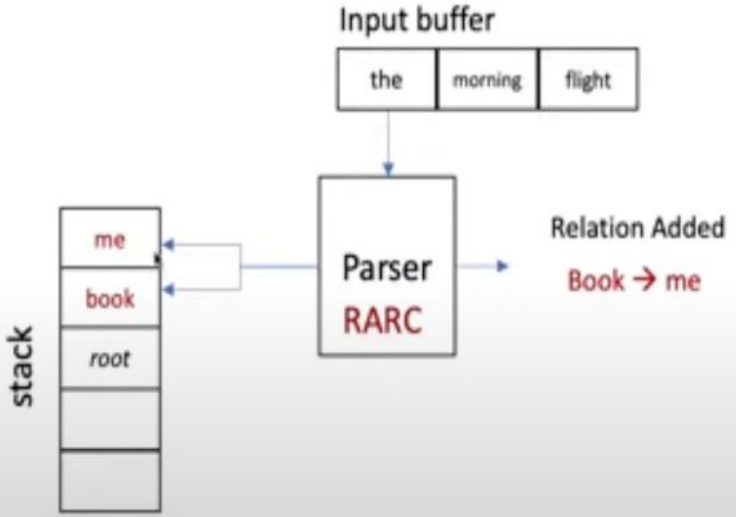
- SHIFT: Put a word in a **stack** for analysis
- LEFTARC: Assign the word in the top of the stack as head of the previous word in the stack
- RIGHTARC: Assign the previous word in the stack as head of the word at the top of the stack



Dependency Parsing

In arc-standard approaches we step across the *n* words in the **Input buffer** from left to right; and for each for word w_i, our **parser** can:

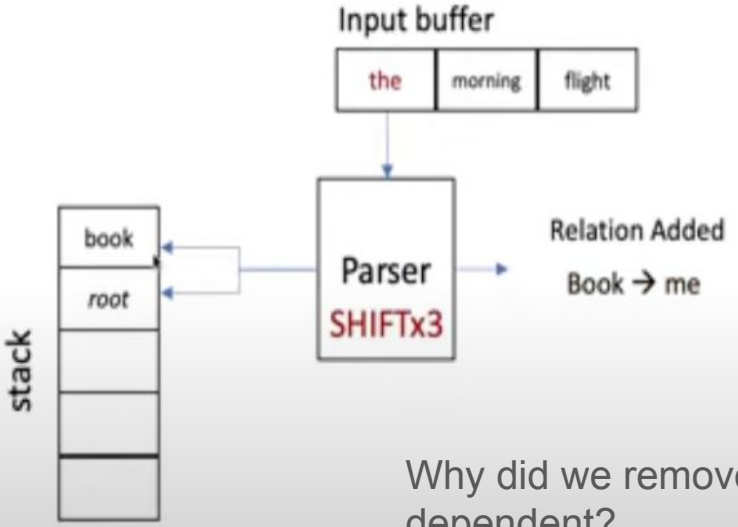
- SHIFT: Put a word in a **stack** for analysis
- LEFTARC: Assign the word in the top of the stack as head of the previous word in the stack
- RIGHTARC: Assign the previous word in the stack as head of the word at the top of the stack



Dependency Parsing

In arc-standard approaches we step across the *n* words in the **Input buffer** from left to right; and for each for word w_i, our **parser** can:

- SHIFT: Put a word in a **stack** for analysis
- LEFTARC: Assign the word in the top of the stack as head of the previous word in the stack
- RIGHTARC: Assign the previous word in the stack as head of the word at the top of the stack



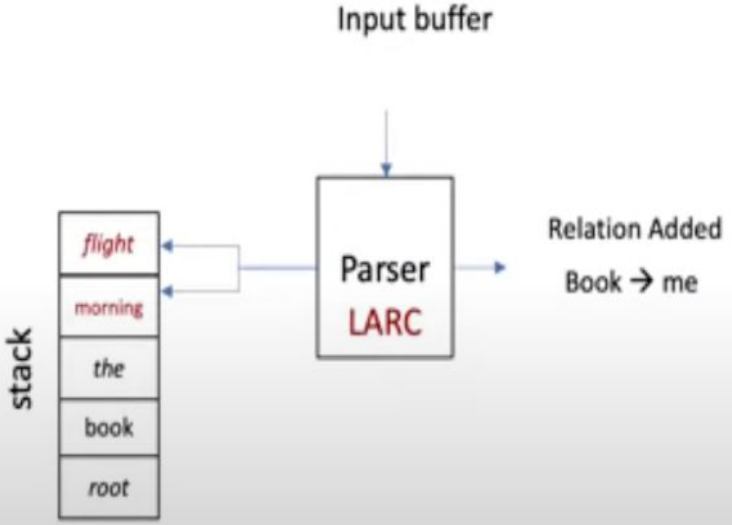
Why did we remove dependent?

Let's parse the entire string!

Dependency Parsing

In arc-standard approaches we step across the *n* words in the **Input buffer** from left to right; and for each for word w_i, our **parser** can:

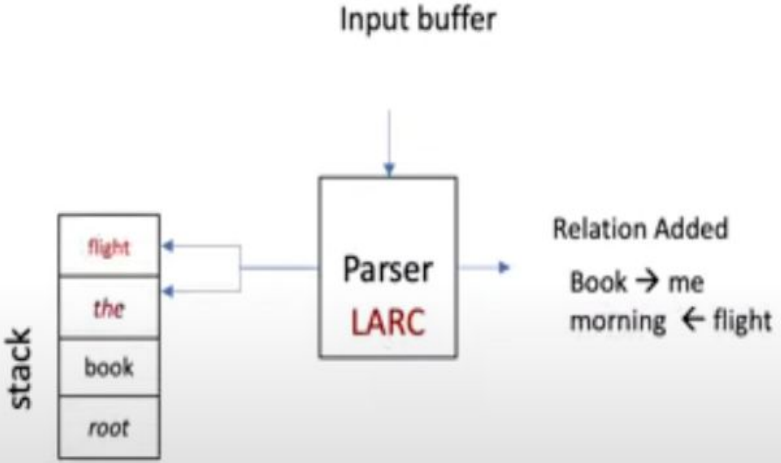
- SHIFT: Put a word in a **stack** for analysis
- LEFTARC: Assign the word in the top of the stack as head of the previous word in the stack
- RIGHTARC: Assign the previous word in the stack as head of the word at the top of the stack



Dependency Parsing

In arc-standard approaches we step across the *n* words in the **Input buffer** from left to right; and for each for word w_i, our **parser** can:

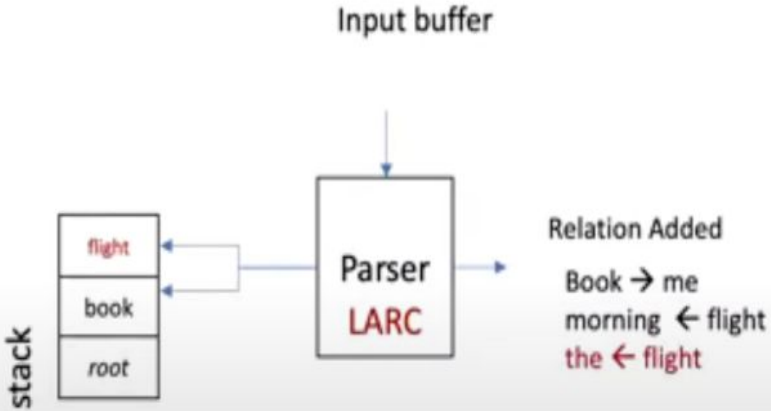
- SHIFT: Put a word in a **stack** for analysis
- LEFTARC: Assign the word in the top of the stack as head of the previous word in the stack
- RIGHTARC: Assign the previous word in the stack as head of the word at the top of the stack



Dependency Parsing

In arc-standard approaches we step across the *n* words in the **Input buffer** from left to right; and for each for word w_i, our **parser** can:

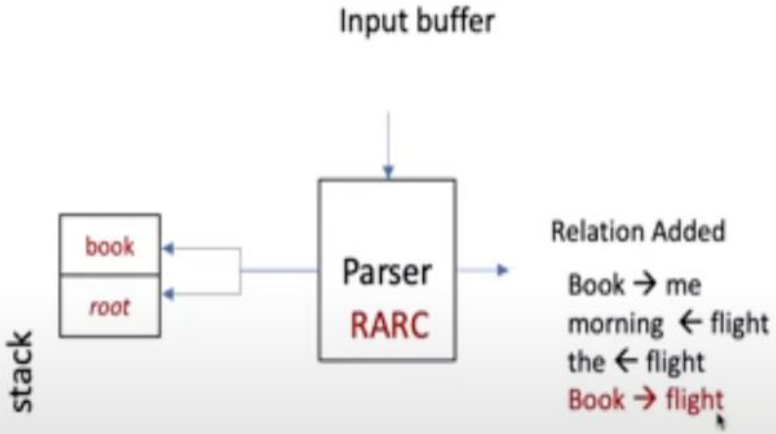
- SHIFT: Put a word in a **stack** for analysis
- LEFTARC: Assign the word in the top of the stack as head of the previous word in the stack
- RIGHTARC: Assign the previous word in the stack as head of the word at the top of the stack



Dependency Parsing

In arc-standard approaches we step across the *n* words in the **Input buffer** from left to right; and for each for word w_i, our **parser** can:

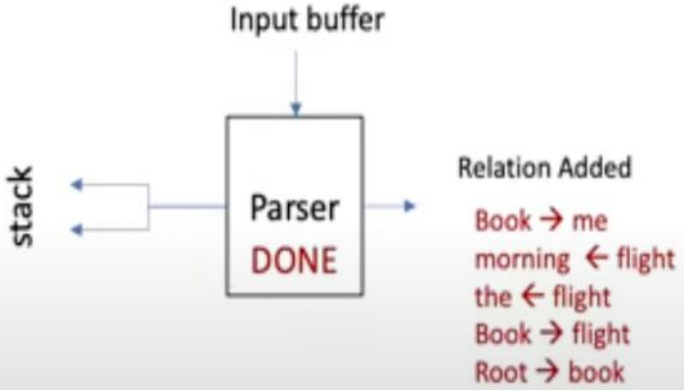
- SHIFT: Put a word in a **stack** for analysis
- LEFTARC: Assign the word in the top of the stack as head of the previous word in the stack
- RIGHTARC: Assign the previous word in the stack as head of the word at the top of the stack



Dependency Parsing

In arc-standard approaches we step across the *n* words in the **Input buffer** from left to right; and for each for word w_i, our **parser** can:

- SHIFT: Put a word in a **stack** for analysis
- LEFTARC: Assign the word in the top of the stack as head of the previous word in the stack
- RIGHTARC: Assign the previous word in the stack as head of the word at the top of the stack



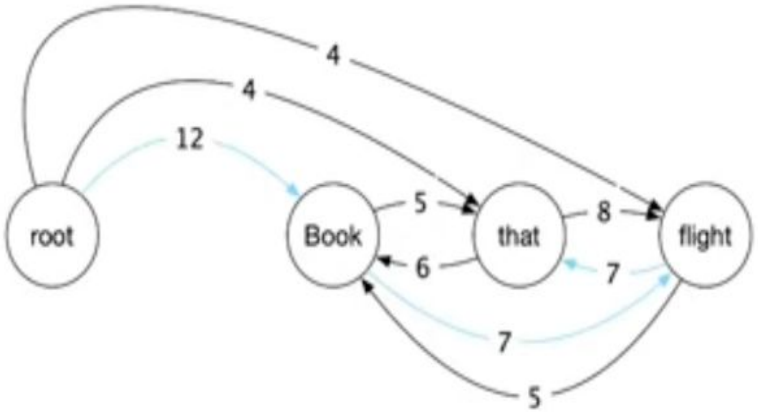
Dependency Parsing

Weights by ML algo

Do we have to maximize or mize the weights?

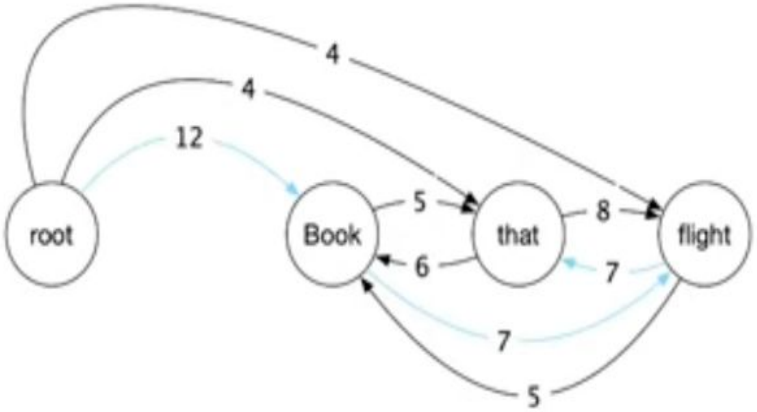
How to draw dependency tree?

How to remove ambiguity?



Dependency Parsing

Graph-based approaches to dependency parsing search through the space of possible trees for a given sentence for a tree (or trees) that maximize some score – this tree is called the **Maximum spanning tree**.



Focus on global optimal solution

Can capture non-projective dependencies (long dependencies)

Acknowledgements

Dr. Ghassemi at Michigan State University

Dr. Dan Jurafsky at Stanford University