Graph-Based Dependency Parsing

Natalie Parde
UIC CS 421

Graph-based Dependency Parsing

- Search through the space of possible trees for a given sentence, attempting to maximize some score
- This score is generally a function of the scores of individual subtrees within the overall tree
- **Edge-factored approaches** determine scores based on the scores of the edges that comprise the tree
  - overall_score(t) = $\sum_{e \in t} score(e)$
  - Letting *t* be a tree for a given sentence, and *e* be its edges

Why use graph-based methods for dependency parsing?

- Transition-based methods tend to have high accuracy on shorter dependency relations, but that accuracy declines as the distance between the two words increases
- This is largely due to the fact that transition-based methods are greedy …they can be fooled by seemingly-optimal local solutions
- Graph-based methods score entire trees, thereby avoiding that issue

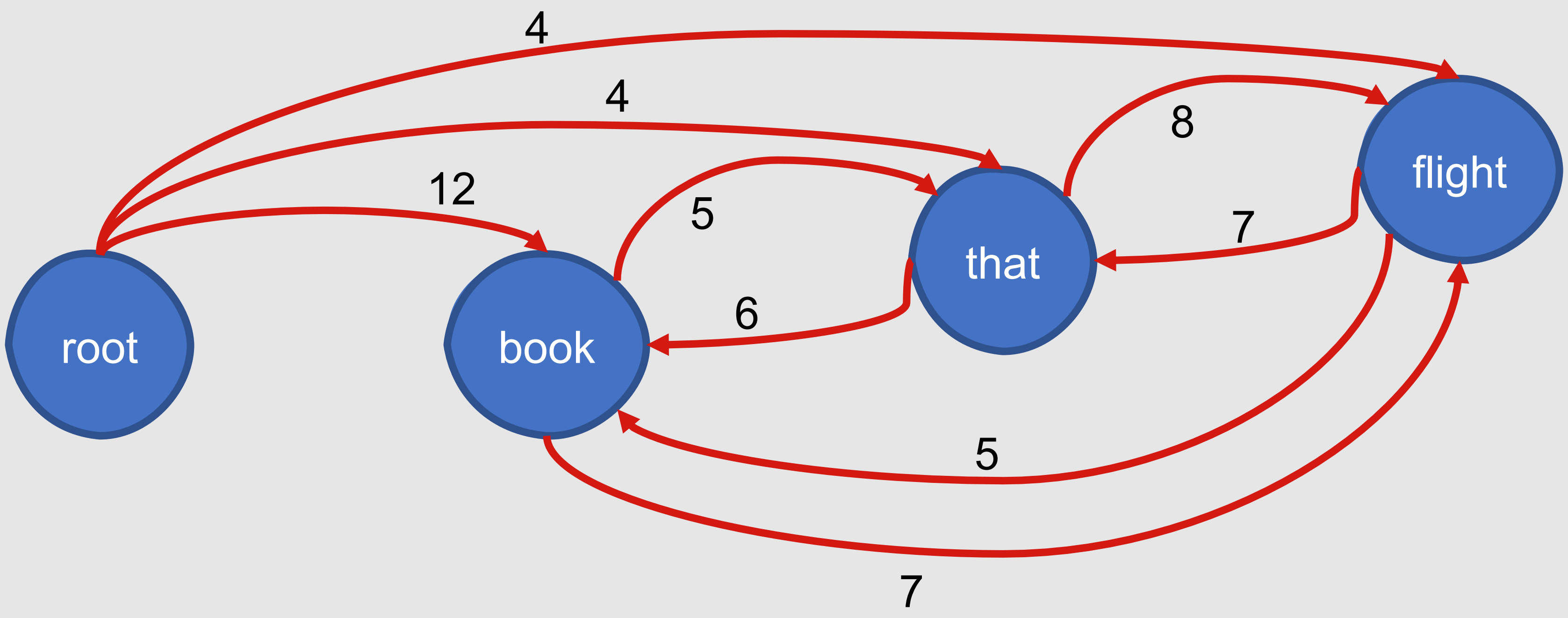
Maximum Spanning Tree

Given an input sentence, construct a fully-connected, weighted, directed graph

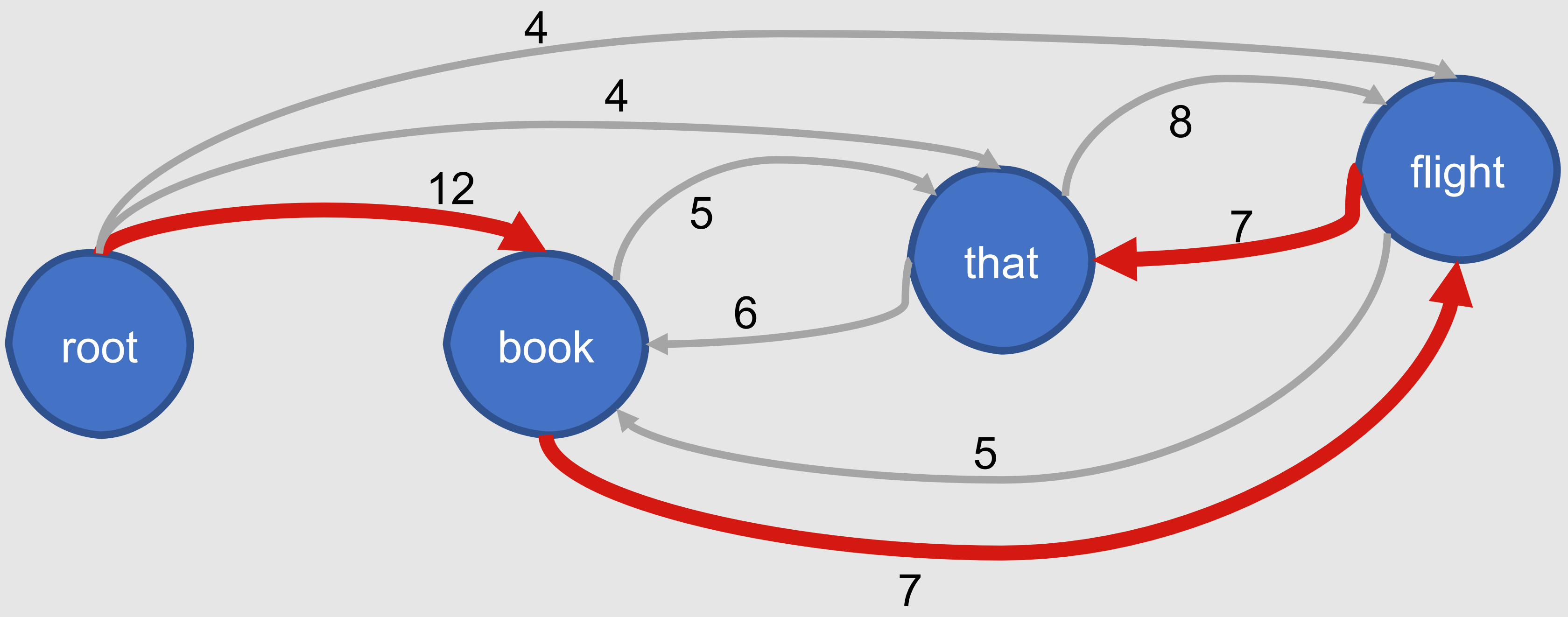
- Vertices are input words
- Directed edges represent all possible head-dependent assignments
- Weights reflect the scores for each possible head-dependent assignment, predicted by a supervised machine learning model

A maximum spanning tree represents the preferred dependency parse for the sentence, as determined by the weights

Maximum Spanning Tree: Example

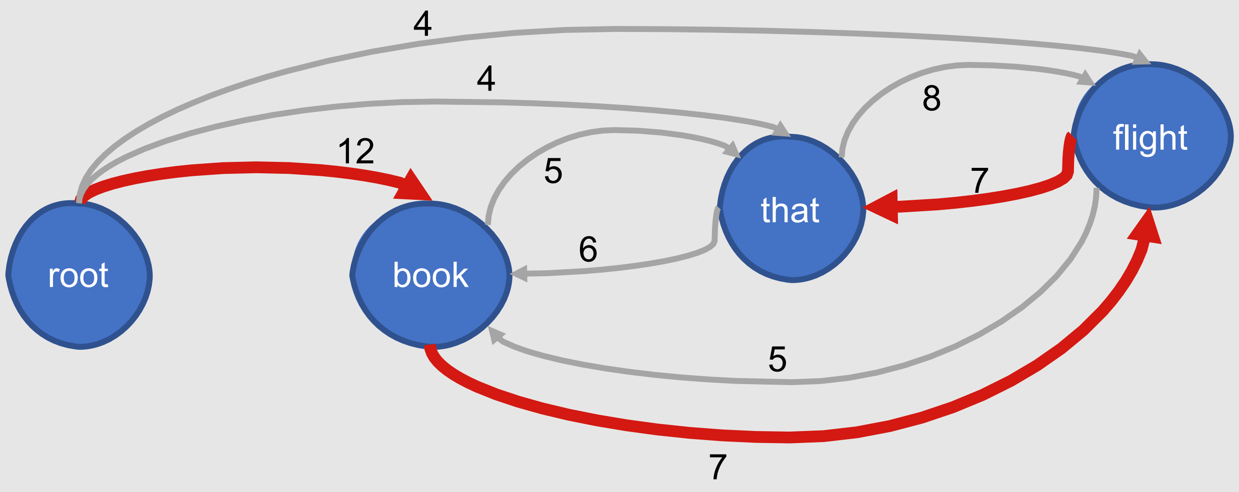


Maximum Spanning Tree: Example



Two things to keep in mind….

- Every vertex in a spanning tree has exactly one incoming edge
- Absolute values of the edge scores are not critical
  - Relative weights of the edges entering a vertex are what matter!



How do we know that we have a spanning tree?

- Given a fully-connected graph G = (V, E), a subgraph T = (V, F) is a spanning tree if:
  - It has no cycles
  - Each vertex (except the root) has exactly one edge entering it

How do we know that we have a maximum spanning tree?

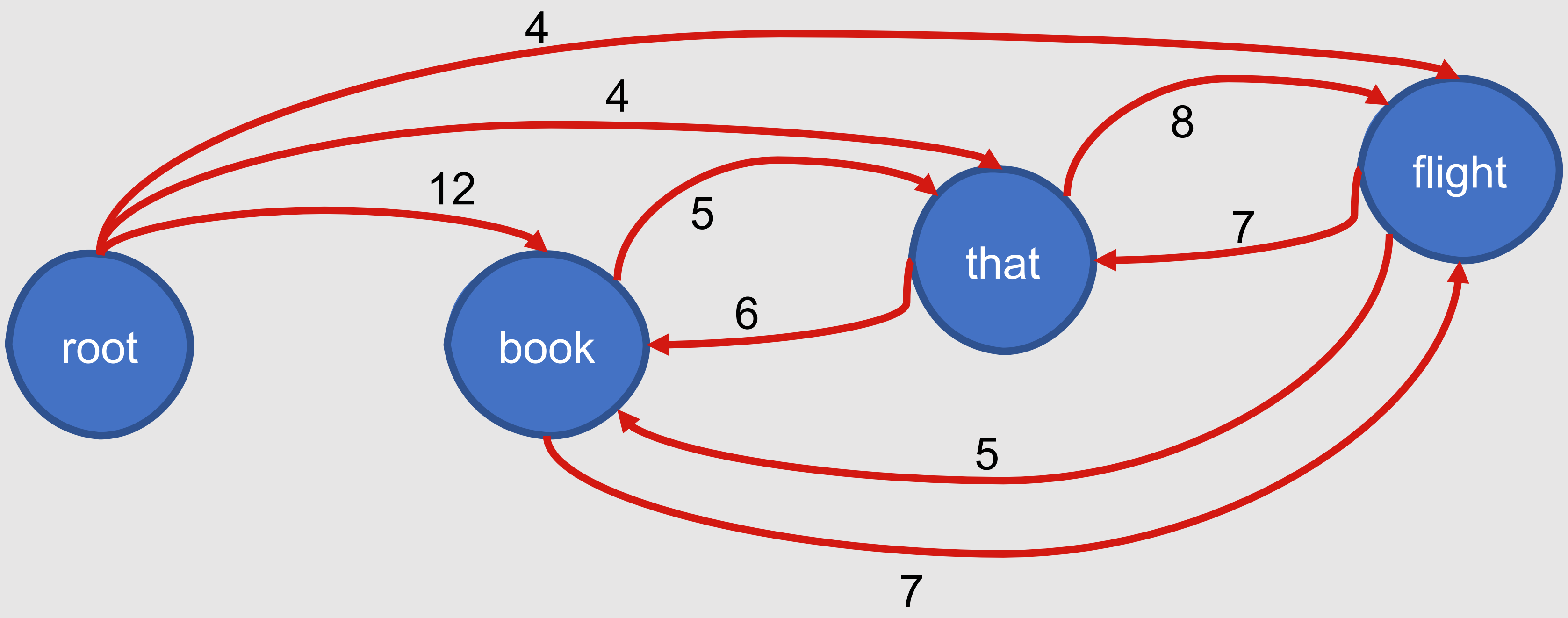
- **If the greedy selection process produces a spanning tree, then that tree is the maximum spanning tree**
- However, the greedy selection process may select edges that result in cycles
- If this happens, an alternate graph can be created that collapses cycles into new nodes, with edges that previously entered or exited the cycle now entering or exiting the new node
- The greedy selection process is then recursively applied to the new graph until a (maximum) spanning tree is found

Formal Algorithm

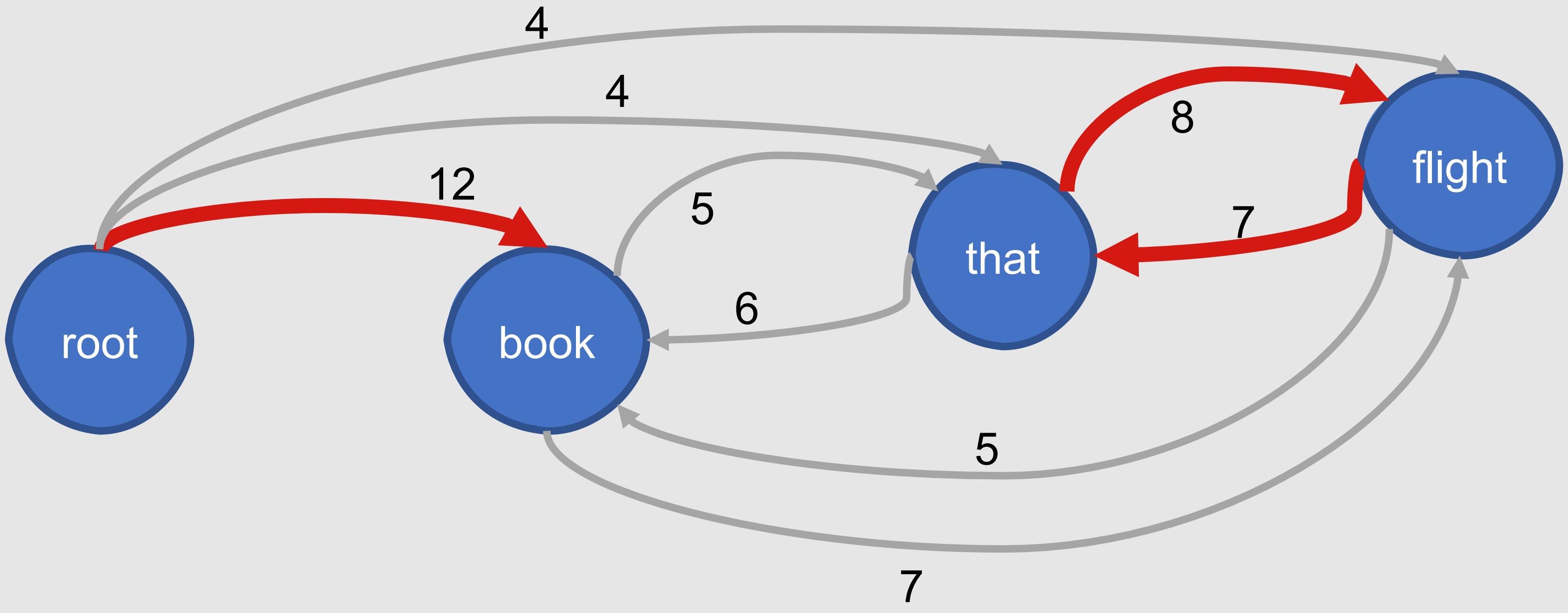
```
F ← []
T ← []
score' ← []
for each v in V do:
        bestInEdge ← argmax_{e=(u,v)∈E} score[e]
        F ← F ∪ bestInEdge
        for each e = (u,v) ∈ E do:
                score'[e] ← score[e] - score[bestInEdge]

        if T=(V,F) is a spanning tree:
                return T
        else:
                C ← a cycle in F
                G' ← collapse(G, C)
                T' ← maxspanningtree(G', root, score') # Recursively call the current function
                T ← expand(T', C)
                return T
```

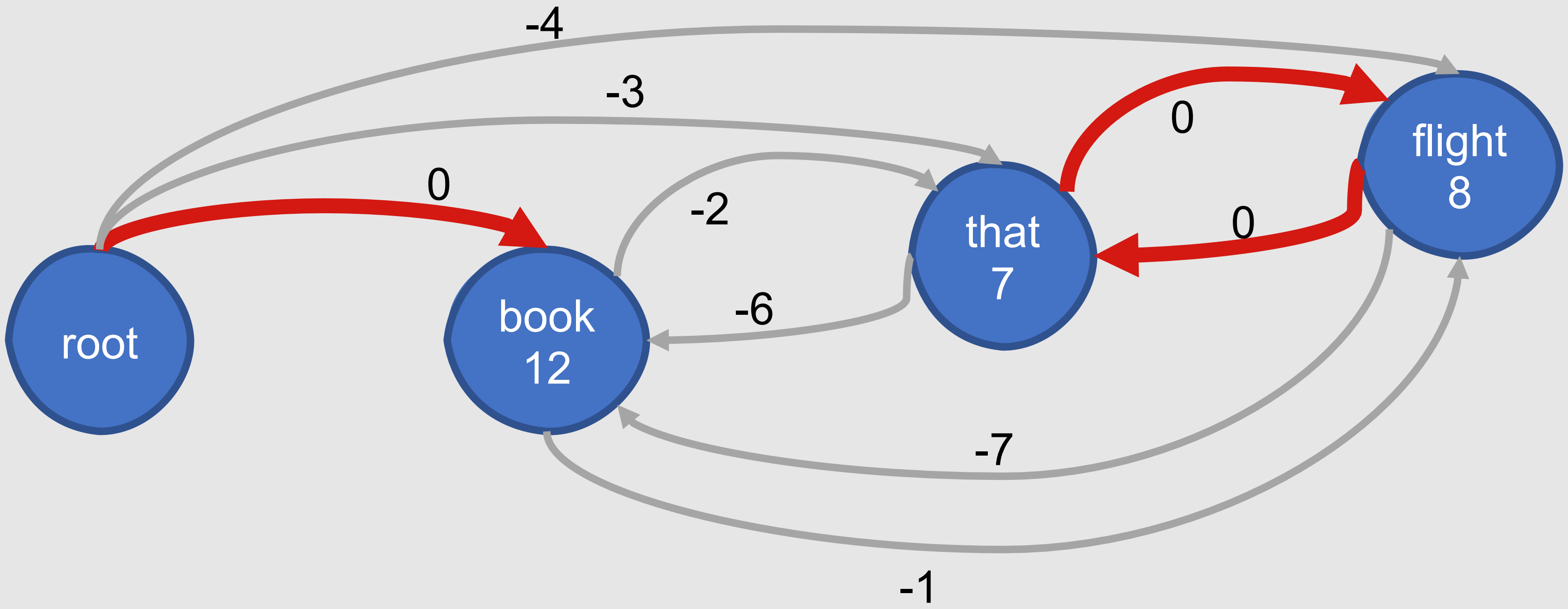
Maximum Spanning Tree: Updated Example



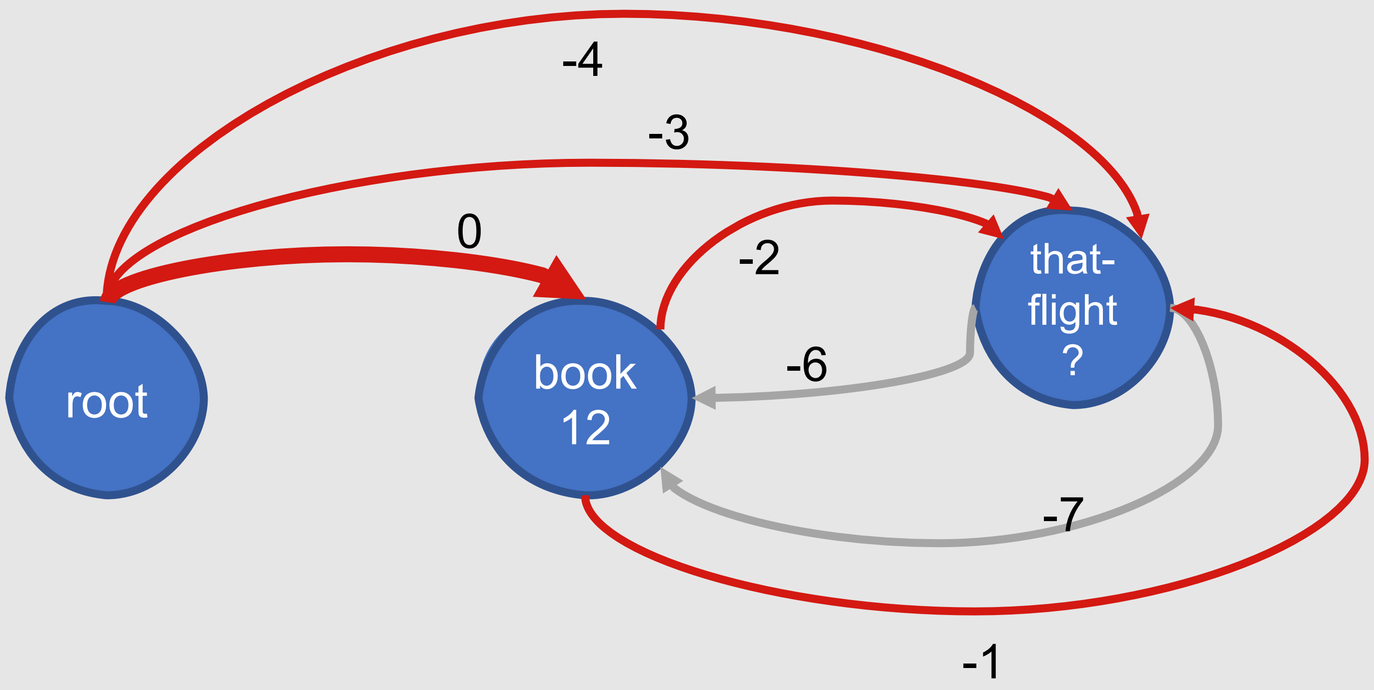
Maximum Spanning Tree: Updated Example



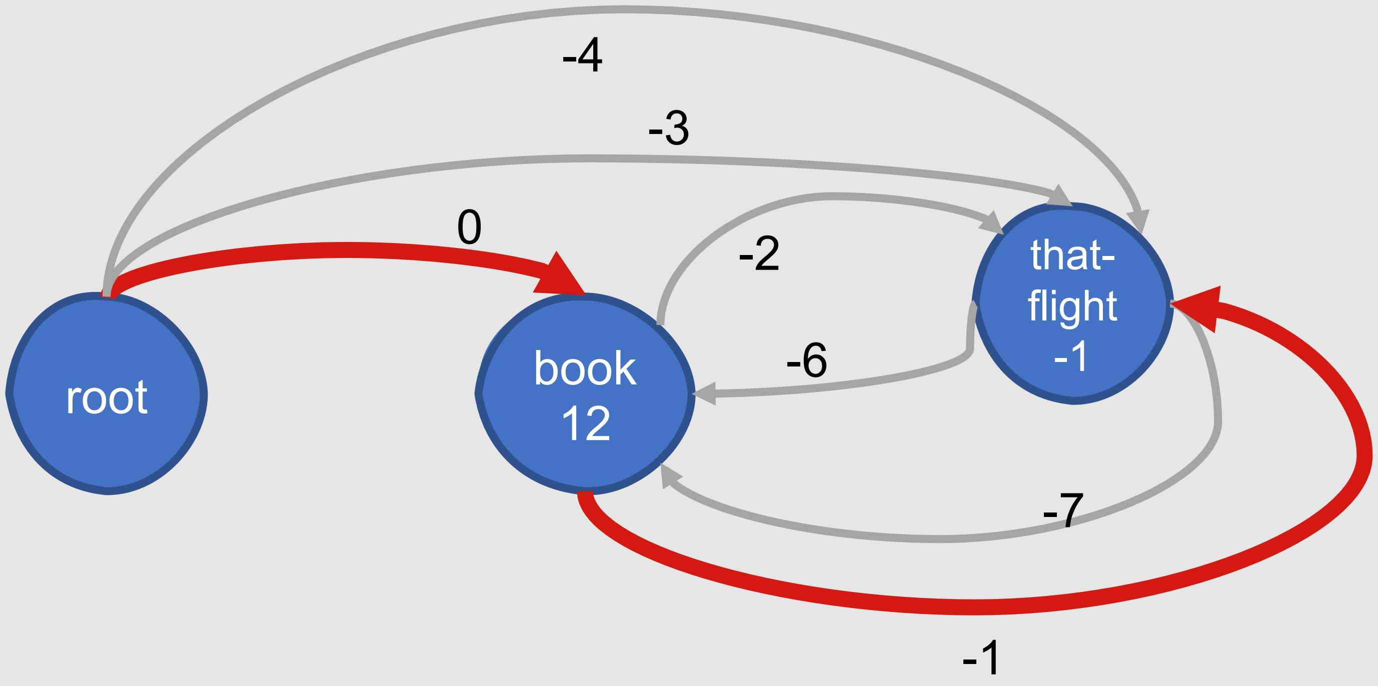
Maximum Spanning Tree: Updated Example



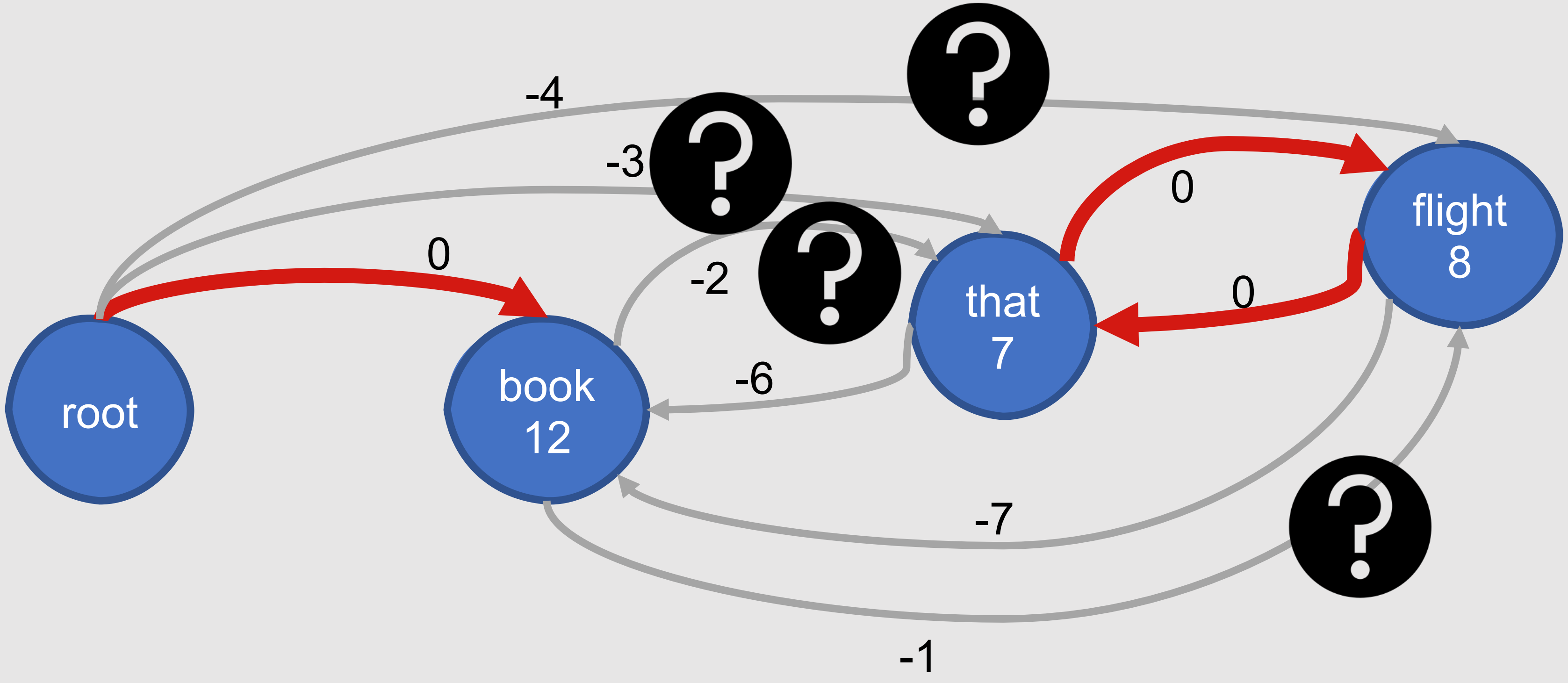
Maximum Spanning Tree: Updated Example



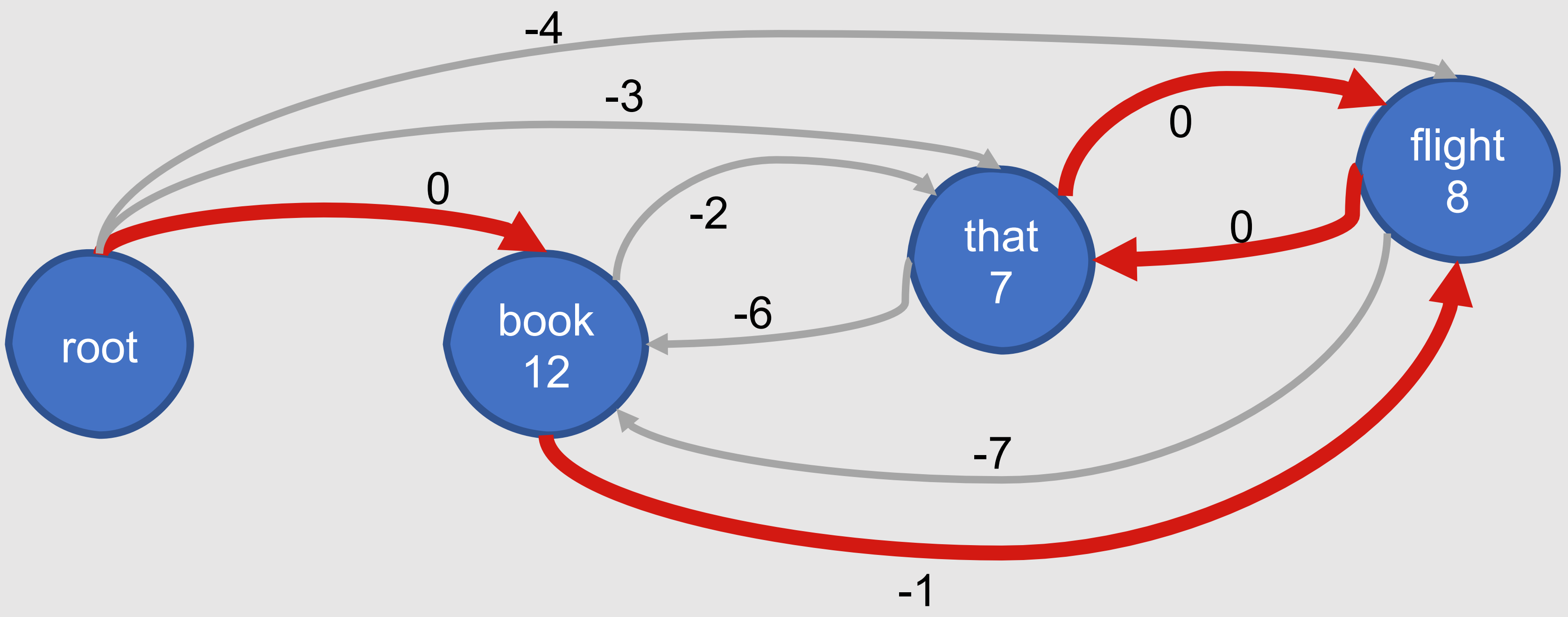
Maximum Spanning Tree: Updated Example



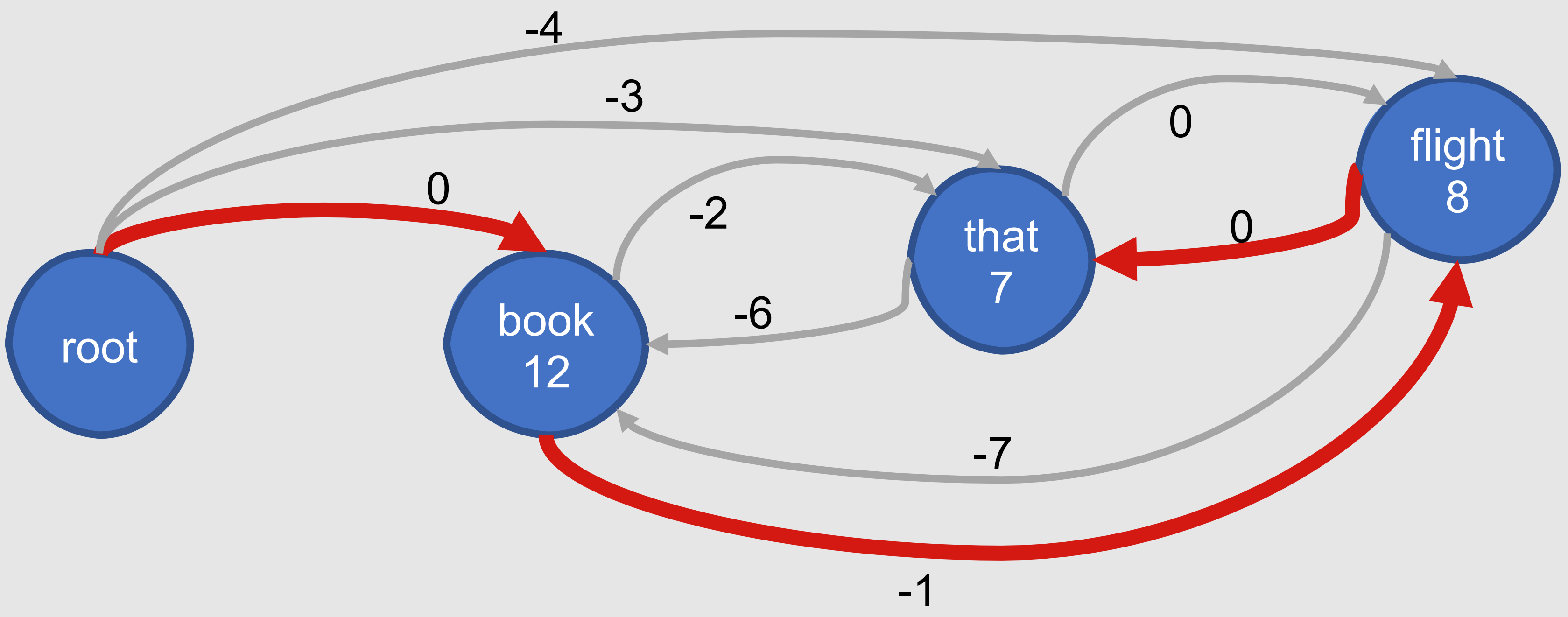
Maximum Spanning Tree: Updated Example



Maximum Spanning Tree: Updated Example



Maximum Spanning Tree: Updated Example



How do we train our model to predict edge weights?

- Similar approach to training the oracle in a transition-based parser
- Common features can include:
  - Words, lemmas, parts of speech
  - Corresponding features from contexts before and after words
  - Word embeddings
  - Dependency relation type
  - Dependency relation direction
  - Distance from head to dependent