10-607 Computational Foundations for Machine Learning
Machine Learning Department
School of Computer Science
Carnegie Mellon University

Dynamic Programming
+
Data Structures

Matt Gormley
Lecture 8
Nov. 14, 2018

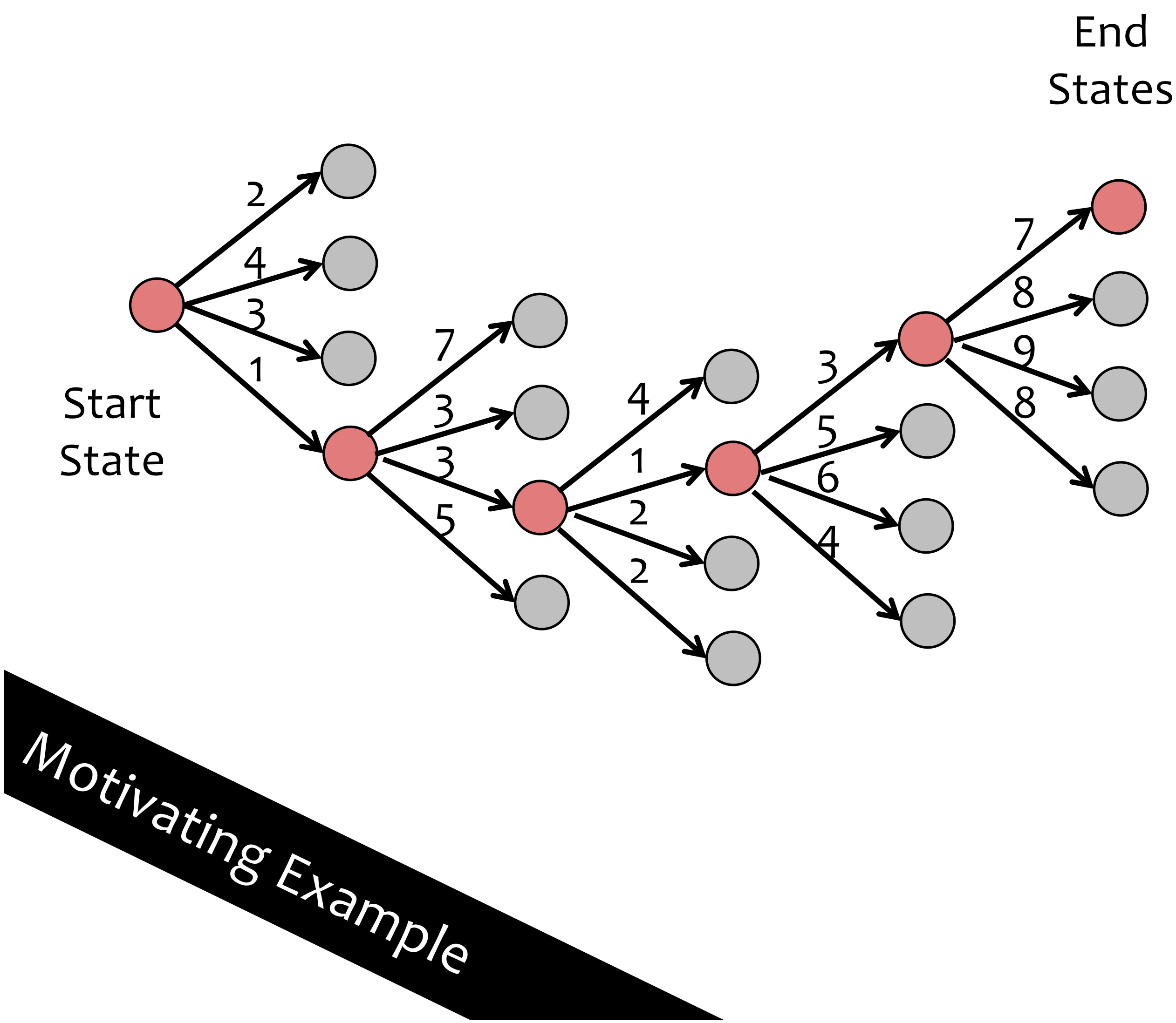
Reminders

- Homework B: Complexity & Recursion
  - Out: Thu, Nov. 8
  - Due: Tue, Nov. 20 at 11:59pm
- Quiz 1: Logic & Proofs; Computation
  - Mon, Nov. 19, in-class
  - Covers Lectures 1 – 6

Q&A

RECURSION

Example: Greedy Search



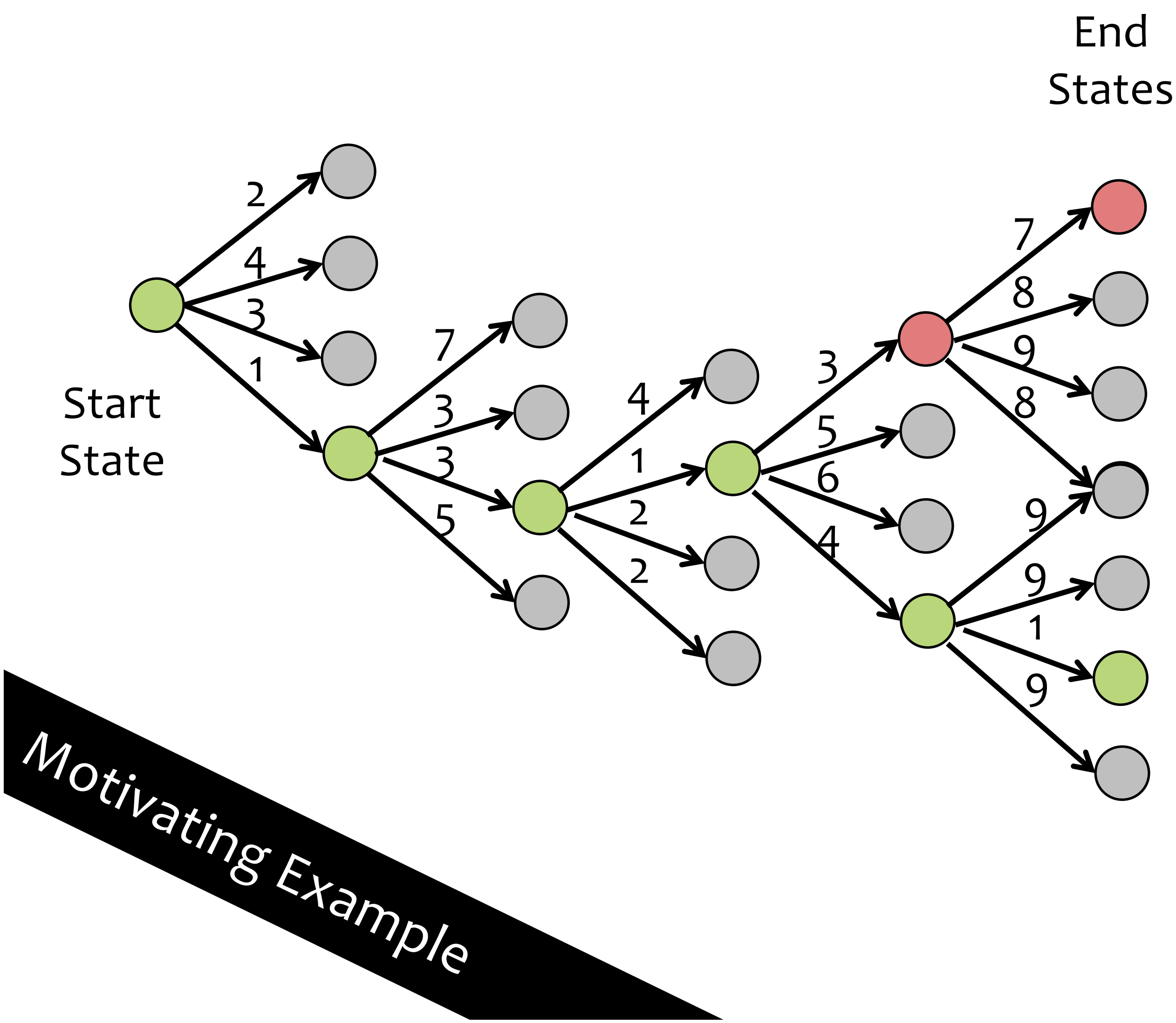
Goal:

- Search space consists of nodes and weighted edges
- Goal is to find the lowest (total) weight path from root to a leaf

Greedy Search:

- At each node, selects the edge with lowest (immediate) weight
- Heuristic method of search (i.e. does *not* necessarily find the best path)

Example: Greedy Search



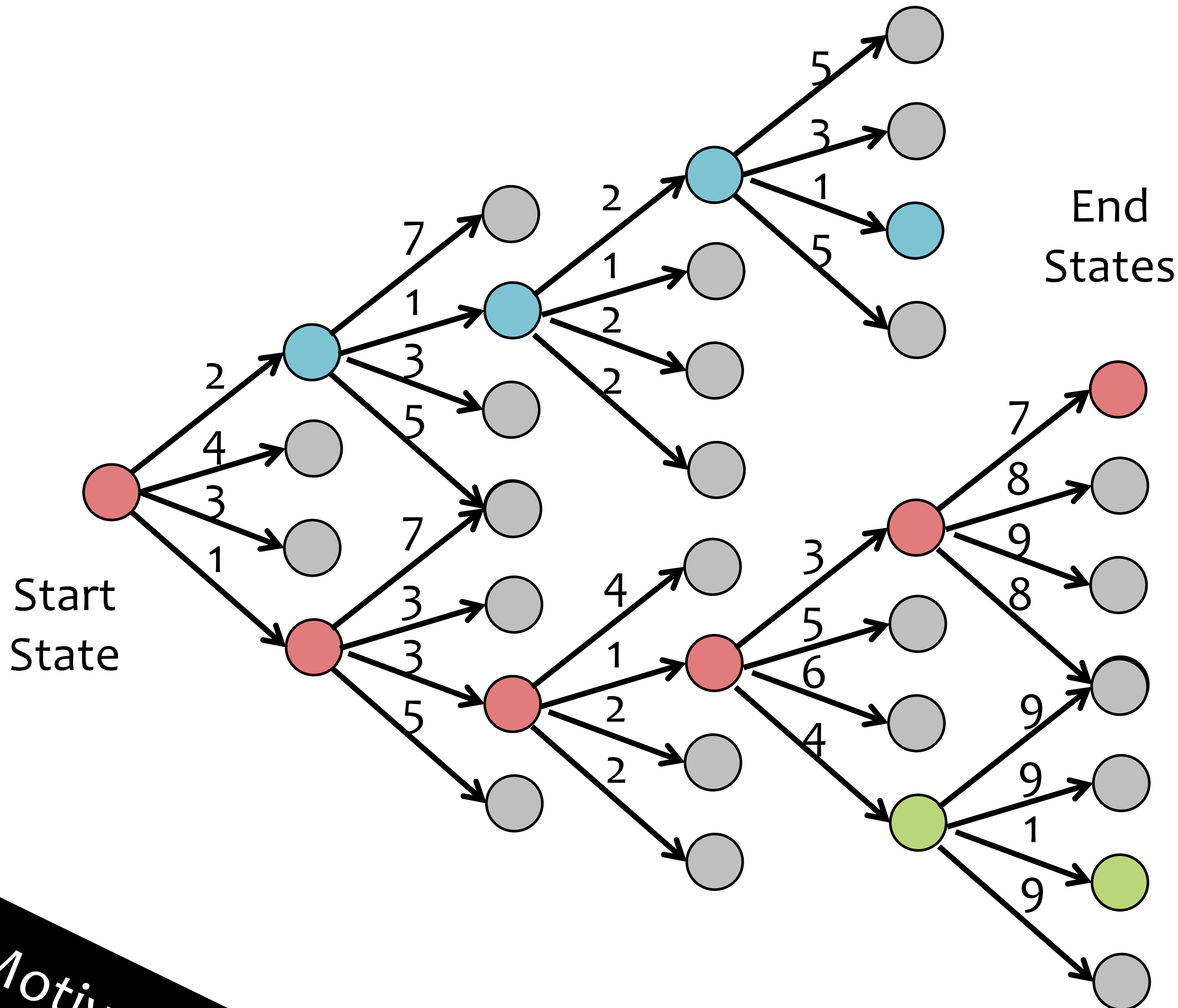
Goal:

- Search space consists of nodes and weighted edges
- Goal is to find the lowest (total) weight path from root to a leaf

Greedy Search:

- At each node, selects the edge with lowest (immediate) weight
- Heuristic method of search (i.e. does *not* necessarily find the best path)

Example: Greedy Search



Goal:

- Search space consists of nodes and weighted edges
- Goal is to find the lowest (total) weight path from root to a leaf

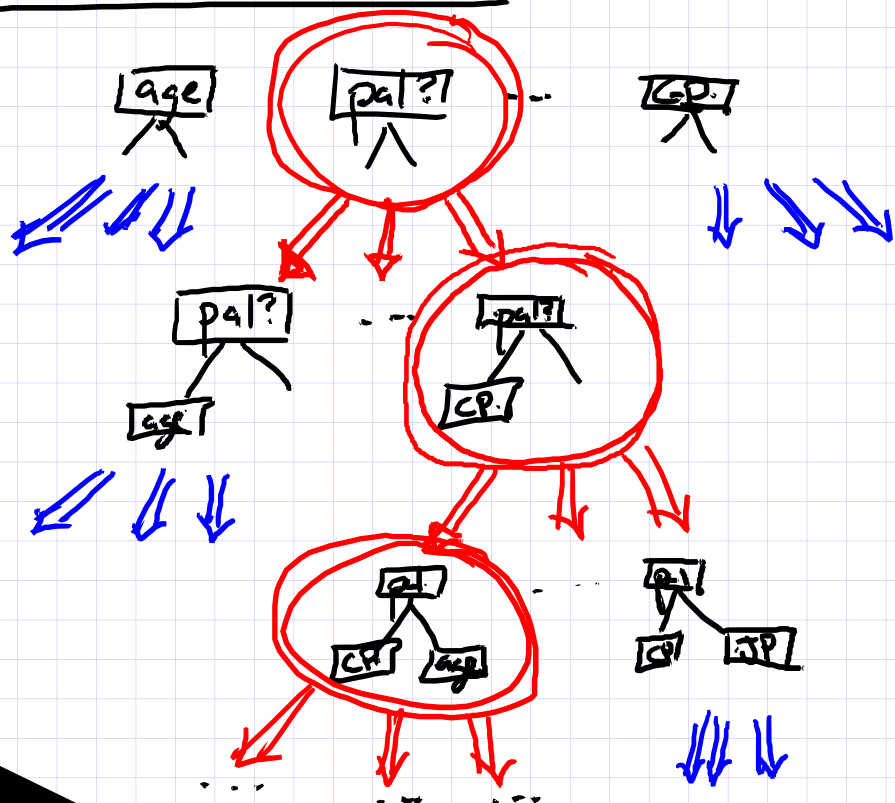
Greedy Search:

- At each node, selects the edge with lowest (immediate) weight
- Heuristic method of search (i.e. does *not* necessarily find the best path)

Motivating Example

Example: Decision Trees

ID3 as Search



Search space: all possible trees

ID3: greedy search, maximizing info. gain at each split

searches for smallest tree consistent with the training data

"inductive bias" of ID3

★ Occam's Razor: prefers the simplest hypothesis that explains the data. (i.e. 1300s smallest expl. is best)

Motivating Example

Proof by Induction

Chalkboard:

- Weak Induction
  - basis case
  - inductive hypothesis
  - inductive step
- Example: sum of powers of two
- Why does proof by induction work?
  - propositional logic interpretation

Proof by Induction

In-Class Exercise

Prove the following statement by induction.

$$\sum_{i=1}^{n} i = n(n+1)/2$$

Answer Here:

Recursion

Chalkboard:

- Example: Factorial (iterative implementation)
- Example: Factorial (recursive implementation)
- Strong Induction
  - multiple basis cases
  - complete assumption
- Proof of recursive factorial correctness

Recursion

Chalkboard:

– Definition: Sorted Array
– Example: Insertion Sort (iterative implementation)
– Example: Insertion Sort (recursive implementation)
– Big Idea: Divide and Conquer
– Example: Merge Sort

Insertion Sort

```
def swap(a, i, j):
    '''Swap the values in a[i] and a[j].'''
    assert 0 <= i and i < len(a)
    assert 0 <= j and j < len(a)
    tmp = a[i]
    a[i] = a[j]
    a[j] = tmp
```

```
def insertion_sort(a):
    '''Sort an array in place via insertion sort.'''
    for i in range(0, len(a)):
        for j in range(i, 0, -1):
            if a[j-1] < a[j]:
                break
            swap(a, j, j-1)
    return
```

```
def recursive_insertion_sort(a, n=None):
    '''Sort an array in place via insertion sort up to its
       n'th element.'''
    if n == None:
        n = len(a)
    if n == 1:
        return
    recursive_insertion_sort(a, n-1)
    for j in range(n-1, 0, -1):
        if a[j-1] < a[j]:
            break
        swap(a, j, j-1)
    return
```

Divide and Conquer

Key Idea: Divide a large problem into independent subproblems and solve each subproblem separately

Merge Sort

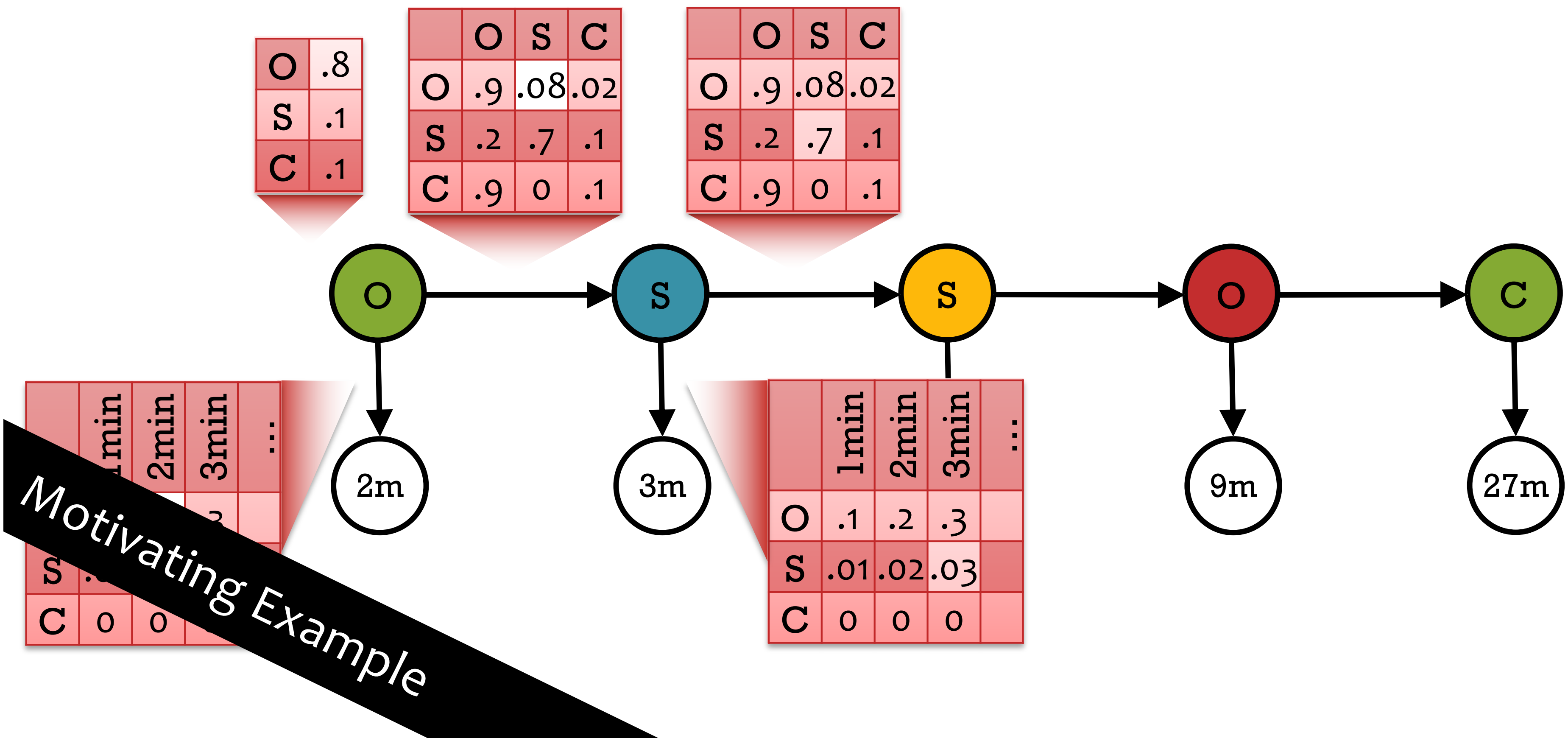
```
def merge_sort(a):
    '''Sort the array a in place via merge sort.'''
    if len(a) <= 1:
        return

    # Split into two halves
    mid = int(len(a)/2)
    left = a[:mid]
    right = a[mid:]

    # Sort each half
    merge_sort(left)
    merge_sort(right)

    # Merge sorted halves back into original
    i = 0
    j = 0
    for k in range(0, len(a)):
        if i >= len(left):
            a[k] = right[j]
            j += 1
        elif j >= len(right):
            a[k] = left[i]
            i += 1
        elif left[i] < right[j]:
            a[k] = left[i]
            i += 1
        else:
            a[k] = right[j]
            j += 1
    return
```

DYNAMIC PROGRAMMING

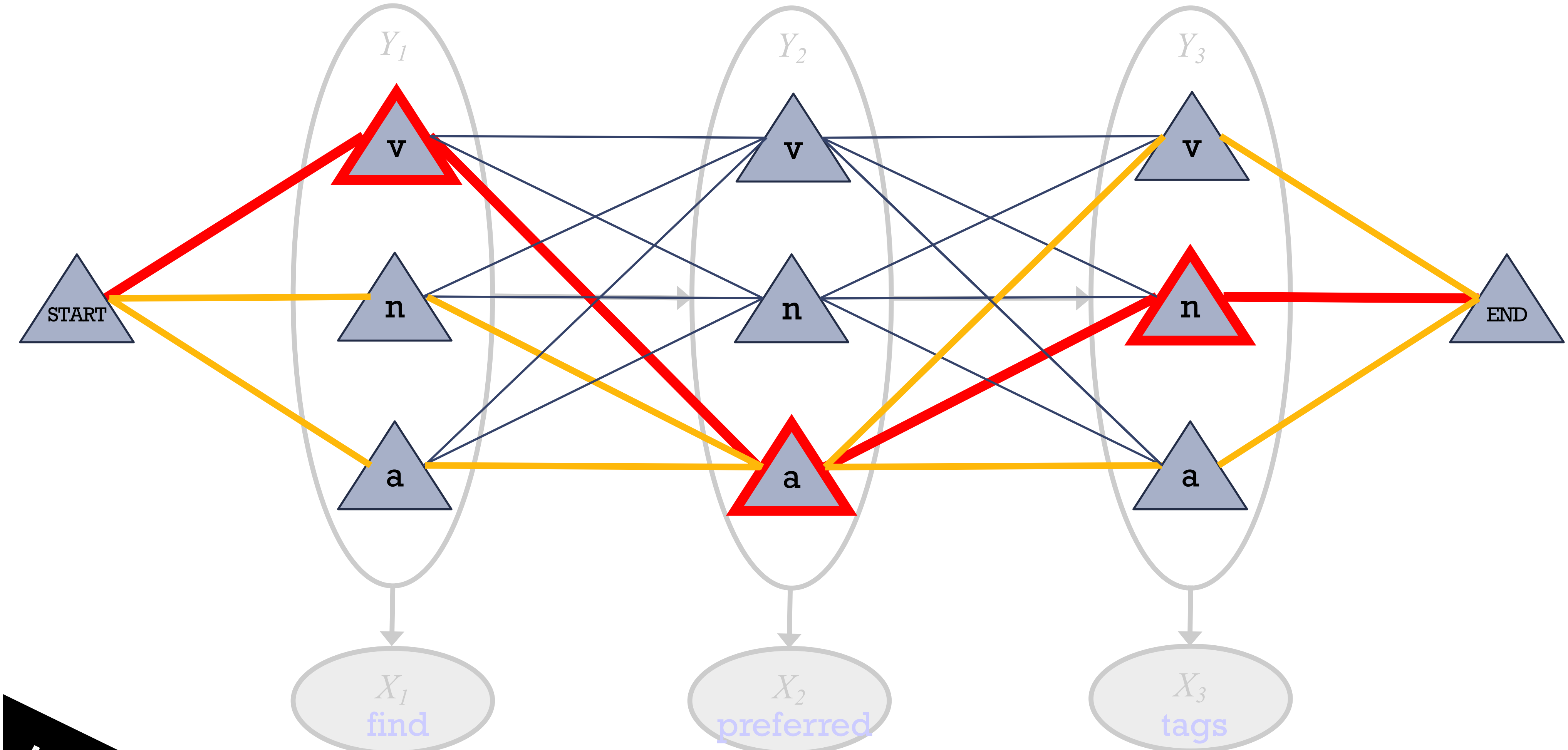
Hidden Markov Model

A Hidden Markov Model (HMM) provides a joint distribution over the the tunnel states / travel times with an assumption of dependence between adjacent tunnel states.

$$p(\mathrm{O, S, S, O, C, 2m, 3m, 18m, 9m, 27m}) = (.8 * .08 * .2 * .7 * .03 * \ldots)$$

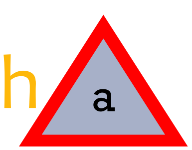


Forward-Backward Algorithm: Finds Marginals



Motivating Example

[redacted]$p(\mathbf{v\ a\ n}) = (1/Z)$ * product weight of one path

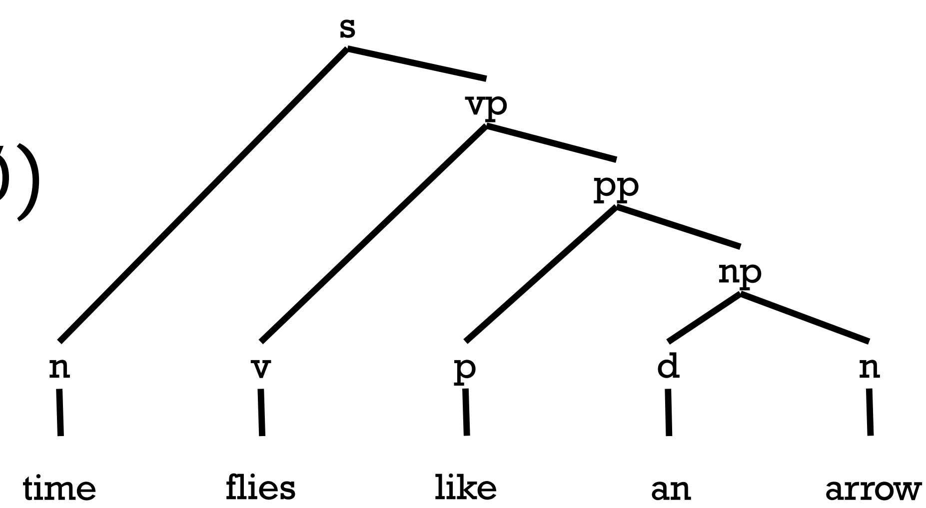
- M[redacted]l probability $p(Y_2 = \mathrm{a})$
  = ([redacted]* total weight of *all* paths through 

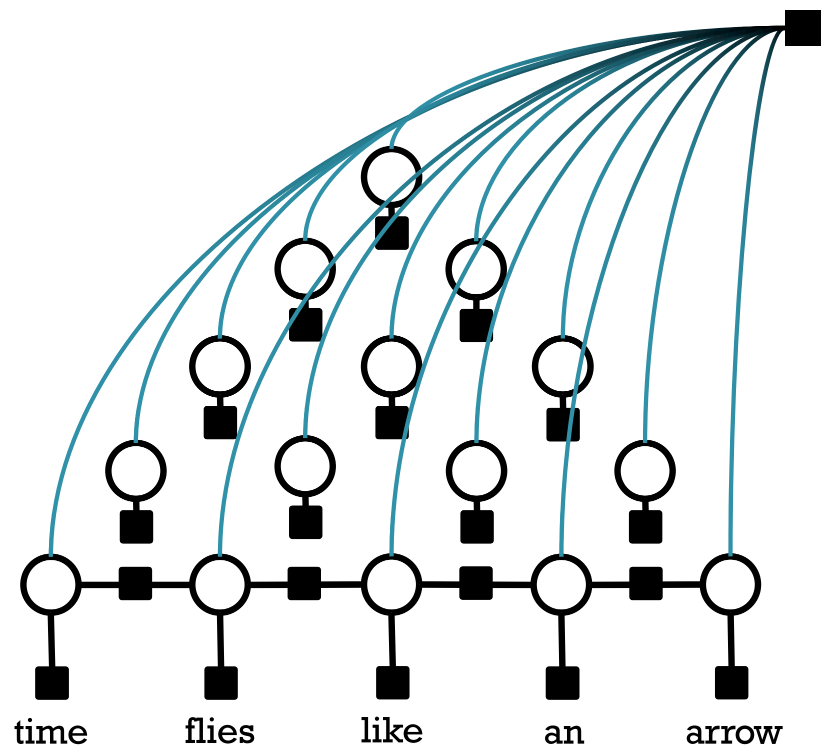
Example Task:

Constituency Parsing

- **Variables:**
  - Constituent type (or Ø) for each of $O(n^2)$ substrings
- **Interactions:**
  - Constituents must describe a binary tree
  - Tag bigrams
  - Nonterminal triples (parent, left-child, right-child)

[these factors not shown]





Motivating Example

(Naradowsky, Vi… …h, 2012)

Dynamic Programming

Key Idea: Divide a large problem into reusable subproblems and solve each subproblem, storing the result of each for later reuse

"Let's take a word that has an absolutely precise meaning, namely dynamic, in the classical physical sense. It also has a very interesting property as an adjective, and that is it's impossible to use the word, dynamic, in a pejorative sense. Try thinking of some combination that will possibly give it a pejorative meaning. [. . .] Thus, I thought **dynamic programming** was a good name. It was something not even a Congressman could object to. So I used it as an umbrella for my activities."

Richard Bellman, Autobiography (1984)

Dynamic Programming

Chalkboard:

- Big Idea: Dynamic Programming
- Example: Fibonacci with and without dynamic programming
  - Recursive Fibonacci's computational complexity
  - Dynamic programming Fibonacci's computational complexity
- Types of Dynamic Programming
  - Tabulation (bottom-up)
  - Memoization (top-down)
- Example: Matrix Product Parenthesization

DATA STRUCTURES FOR ML

Abstractions vs. Data Structures

Abstractions

- List
- Set
- Map
- Queue (FIFO)
- Stack (LIFO)
- Graph
- Priority Queue

Data Structures

- Array (fixed size)
- Array (variable size)
- Linked List
- Doubly-Linked List
- Multidimensional Array
- Tensor
- Hash Map
- Binary Search Tree
- Balanced Tree
- Trie
- Stack
- Heap
- Graph
- Bipartite Graph
- Sparse Vector
- Sparse Matrix

Data Structures for ML

- Data:
  - Dense feature vector (array)
  - Sparse feature vector (sparse vector)
  - Design matrix (multidimensional array)
- Models:
  - Decision Trees (tree)
  - Bayesian Network (directed acyclic graph)
  - Factor Graph (bipartite graph)
- Algorithms:
  - Greedy Search (weighted graph)
  - A* Search (priority queue/heap)
  - Forward-backward for HMM (trellis)

Trees

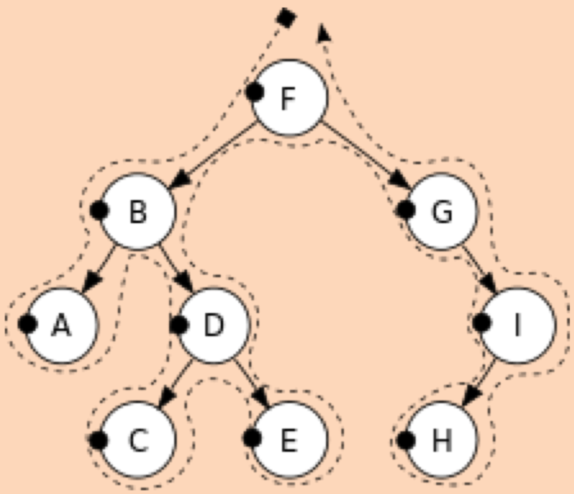
Chalkboard:

- Binary Tree
  - Representation
  - Depth First Search
    - pre-order traversal
    - in-order traversal
    - post-order traversal
  - Breadth First Search
- Decision Tree
  - Representation

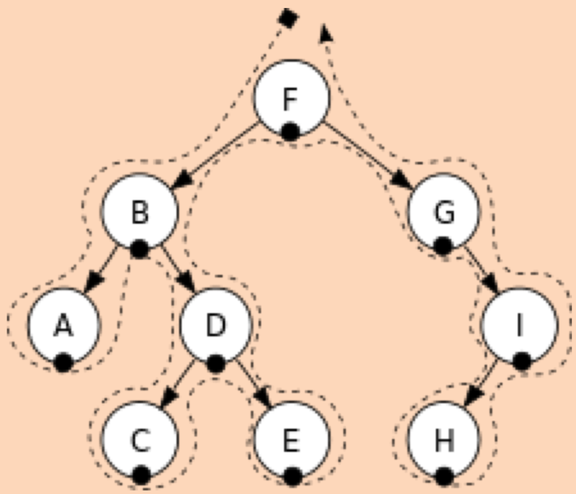
Tree Traversals

Depth First Search

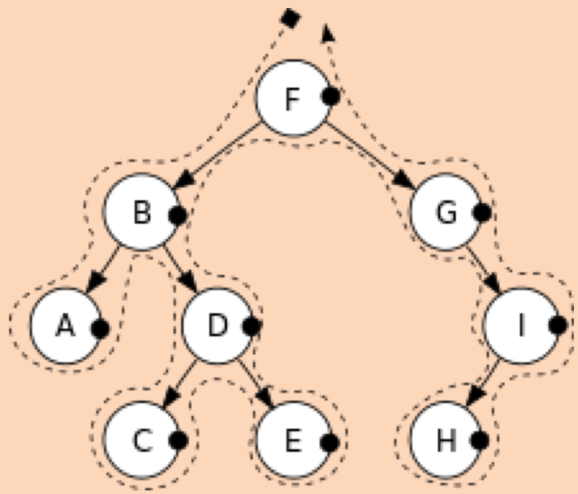
Pre-order



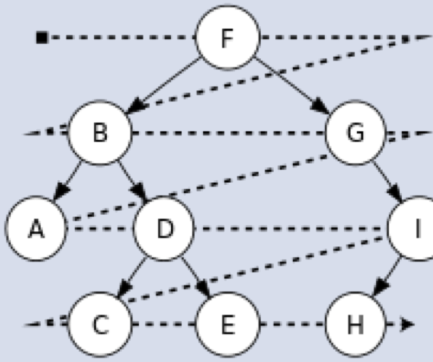
In-order



Post-order



Breadth First Search



Figures from Wikipedia

Sparse Vectors

Chalkboard:

- Sparse Vector
  - Representation
  - Sparse Dot Product
  - Addition of dense vector and sparse vector

Data Structures & Algorithms

Chalkboard:

- Weighted Directed Acyclic Graph
  - Representation
  - Greedy Search
  - Dijkstra's Algorithm
  - A* Search
- Binary Search Tree
  - Representation
  - Average vs. Worst Case Time Complexity
  - Search
  - Insertion
  - Deletion

Efficiency

- CPython vs. PyPy
- Example: Python's Tuple
  - https://stackoverflow.com/questions/14135542/how-is-tuple-implemented-in-cpython
  - https://bitbucket.org/python_mirrors/cpython/src/d81d4b3059e4e5dca67515315c2ada6dfe1c52a4/Objects/tupleobject.c?at=default&fileviewer=file-view-default