Data Structures and Algorithms

Session 26. April 29, 2009

Instructor: Bert Huang

http://www.cs.columbia.edu/~bert/courses/3137

Announcements

- Homework 6 due before last class: May 4th
- Final Review May 4th
- Exam Wednesday May 13th 1:10-4:00 PM, 633

Review

- Finish Quicksort discussion,
  - worst case, average case
- Quickselect
  - worst case, average case
- External Sorting

Today's Plan

- Examples of Data Structures used in Artificial Intelligence and Machine Learning
  - Game trees: minimax, search
  - Bayesian Graphs
  - kd-trees

Artificial Intelligence

- Sub-field of Computer Science concerned with algorithms that behave *intelligently*
  - or if we're truly ambitious, **optimally.**
- An AI program is commonly called an **agent**
  - which makes decisions based on its **percepts**

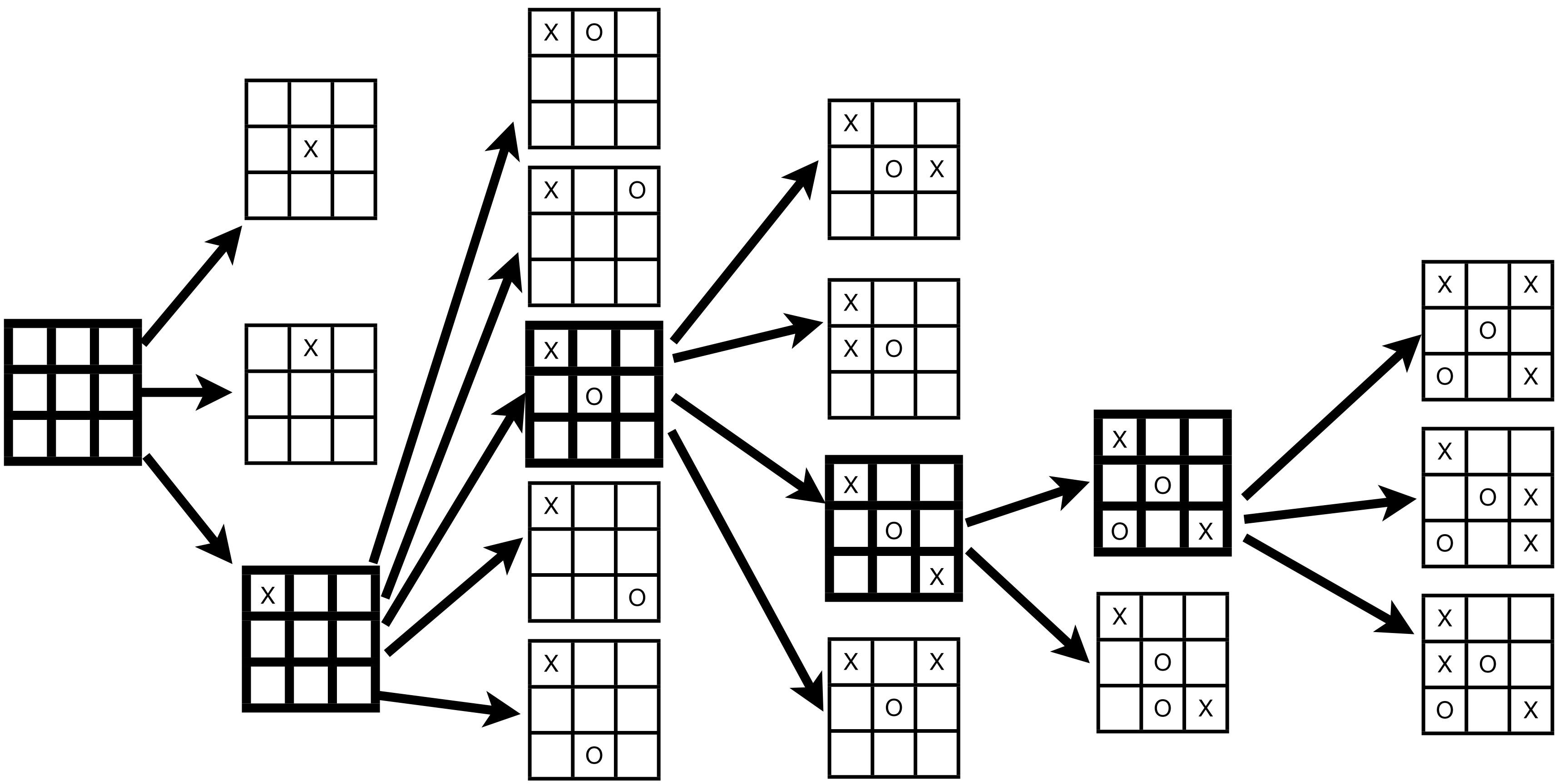
A.I. in Games

- AI still needs to simplify the environment for its agents, so games are a nice starting point
- Many board games are turn-based, so we can take some time to compute a good decision at each turn
- Deterministic turn-based games can be represented as **game trees**

Game Trees

- The root node is the starting state of the game
- Children correspond to possible moves
- If 2-player, every other level is the computer's turn
- The other levels are the adversary's turns
- In a simple game, we can consider/store the whole tree, make decisions based on the subtrees

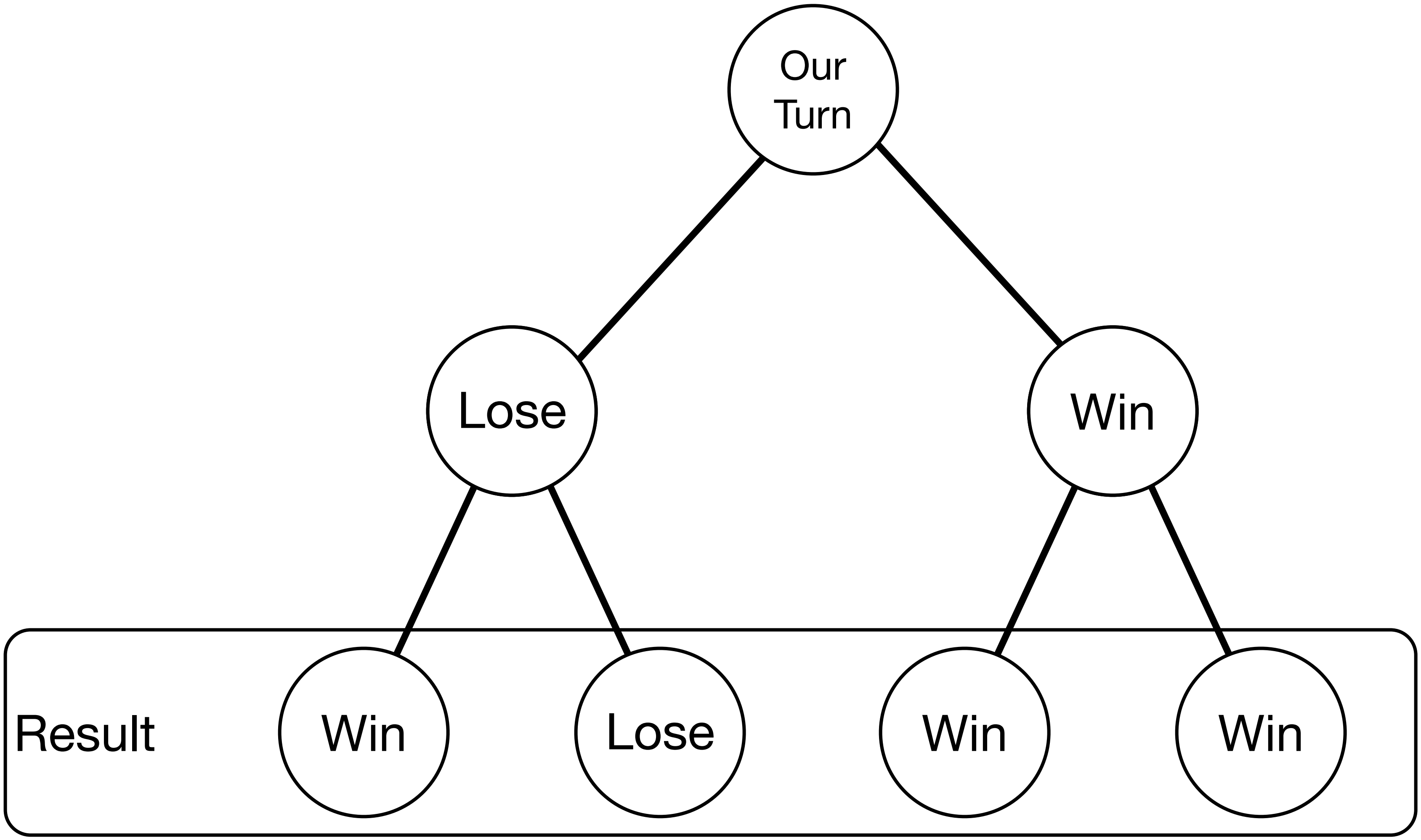
Partial Tic-Tac-Toe Game Tree



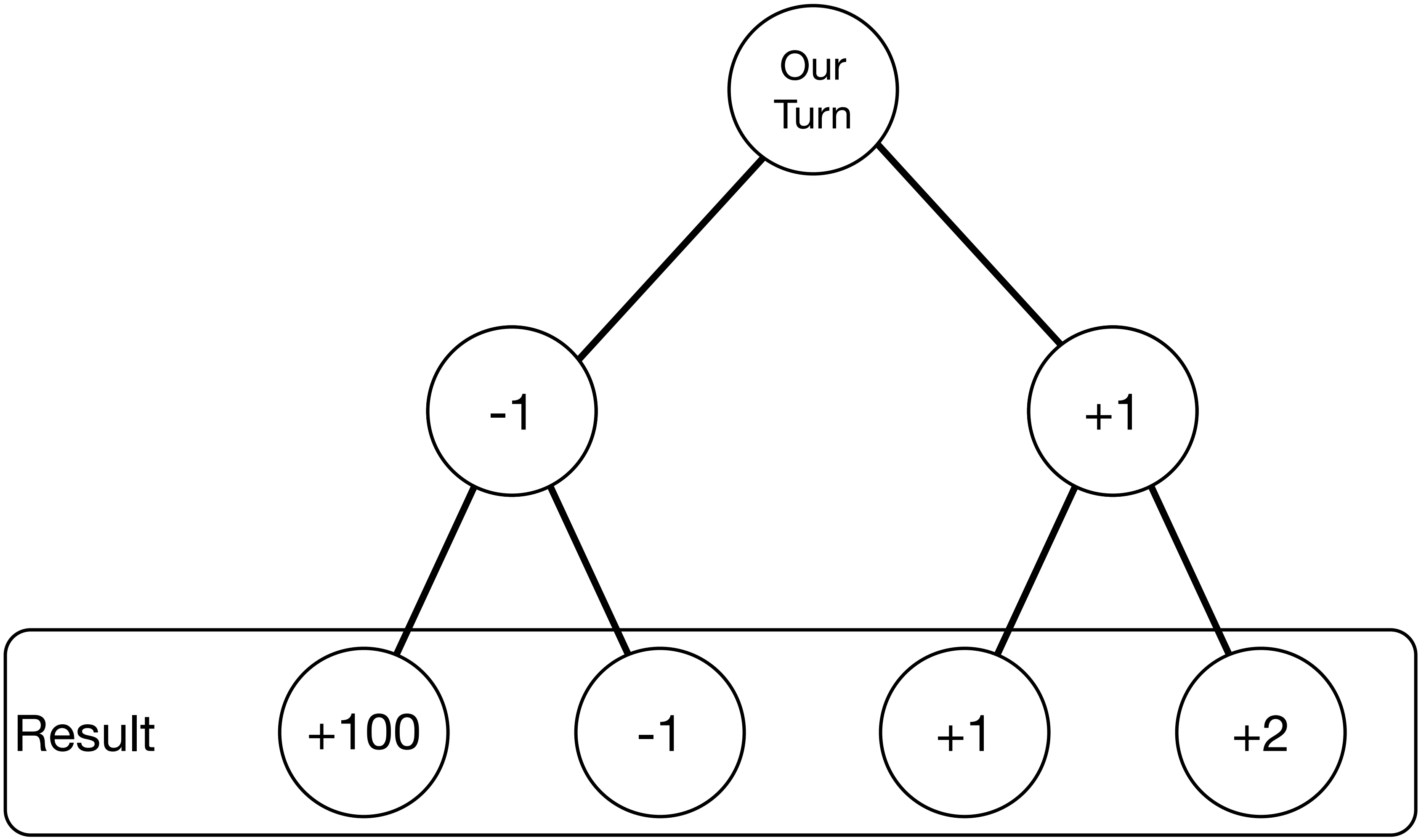
Tree Strategy

- Thinking about the game as a tree helps organize computational strategy
- If adversary plays optimally, we can define the optimal strategy via the **minimax** algorithm
- Assume the adversary will play the optimal move at the next level. Use that result to decide which move is optimal at current level.

Simple Tree



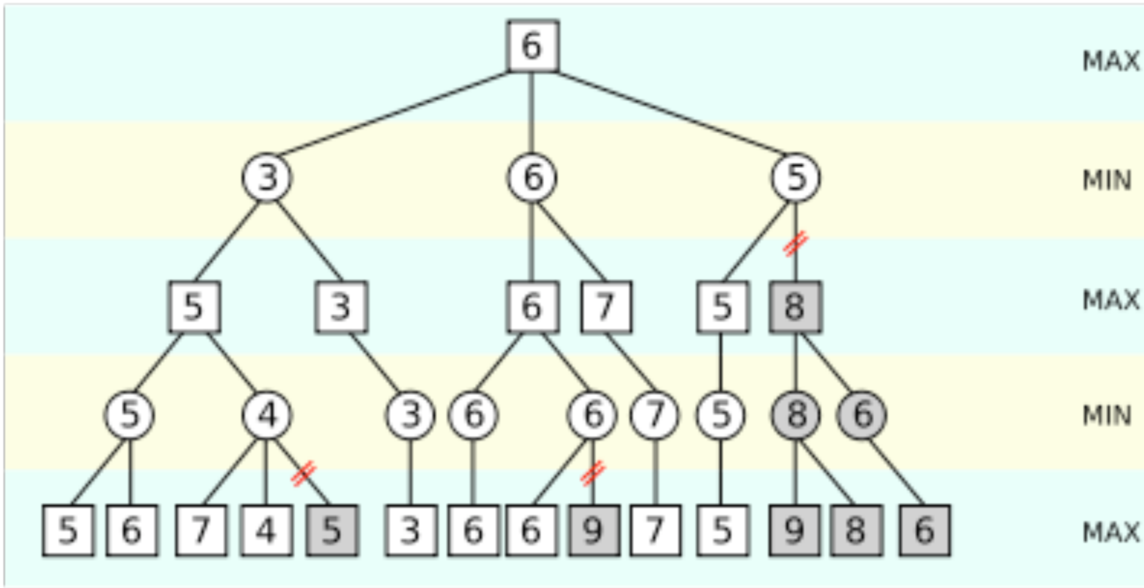
Numerical Rewards



Minimax Details

- Depth first search (postorder) to find leaves; propagate information up
- Adversary also assume you will play optimally
- Impossible to store full tree for most games, use heuristic measures
  - e.g., Chess piece values, # controlled squares
- Cut off after a certain level

Pruning



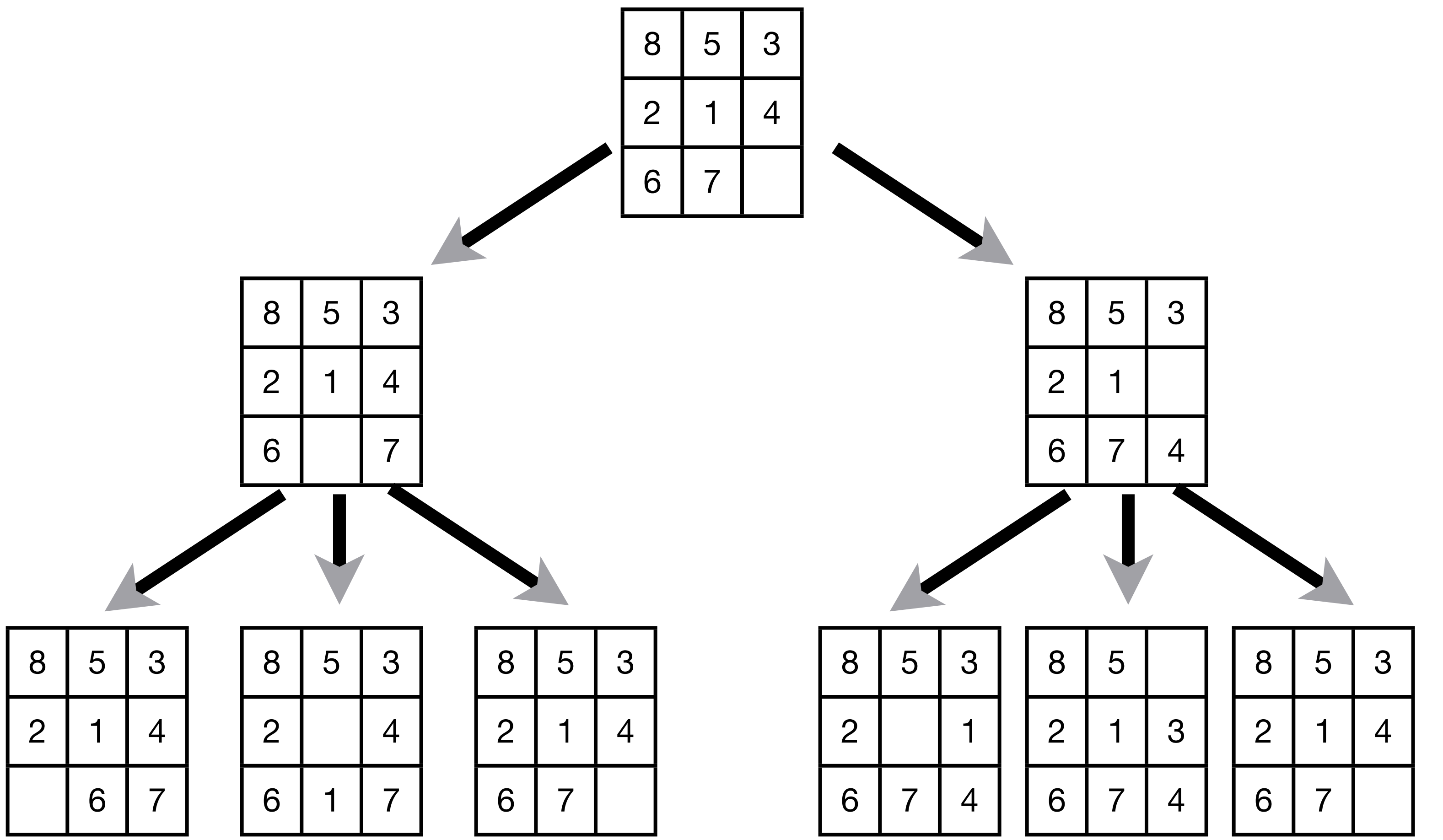
- We can also ignore parts of the tree if we see a subtree that can't possibly be better than one we saw earlier
- This is called **alpha-beta** pruning

- Figure from wikipedia article on alpha-beta pruning

Search

- Some puzzles can be thought of as trees too
- 15-puzzle, Rubik's Cube, Sudoku
- Discrete moves move from current state to children states
- A.I. wants to find the solution state efficiently

8-puzzle



8 5 3
2 1 4
6 7
8 5 3
2 1 4
6 7
8 5 3
2 1
6 7 4
8 5 3
2 1 4
6 7
8 5 3
2 4
6 1 7
8 5 3
2 1 4
6 7
8 5 3
2 1
6 7 4
8 5
2 1 3
6 7 4
8 5 3
2 1 4
6 7

Simple Idea

- Breadth first search; level-order
  - Try every move from current state
  - Try 2 moves from current state
  - Try 3 moves from current state
  - ...

Another Idea

- Depth first search
    - Try a move
    - Try another move...
    - If we get stuck, backtrack

Heuristic Search

- The main problem is without any knowledge, we are guessing arbitrarily
- Instead, design a heuristic and choose the next state to try according to heuristic
  - e.g., # of tiles in the correct location, distance from maze goal

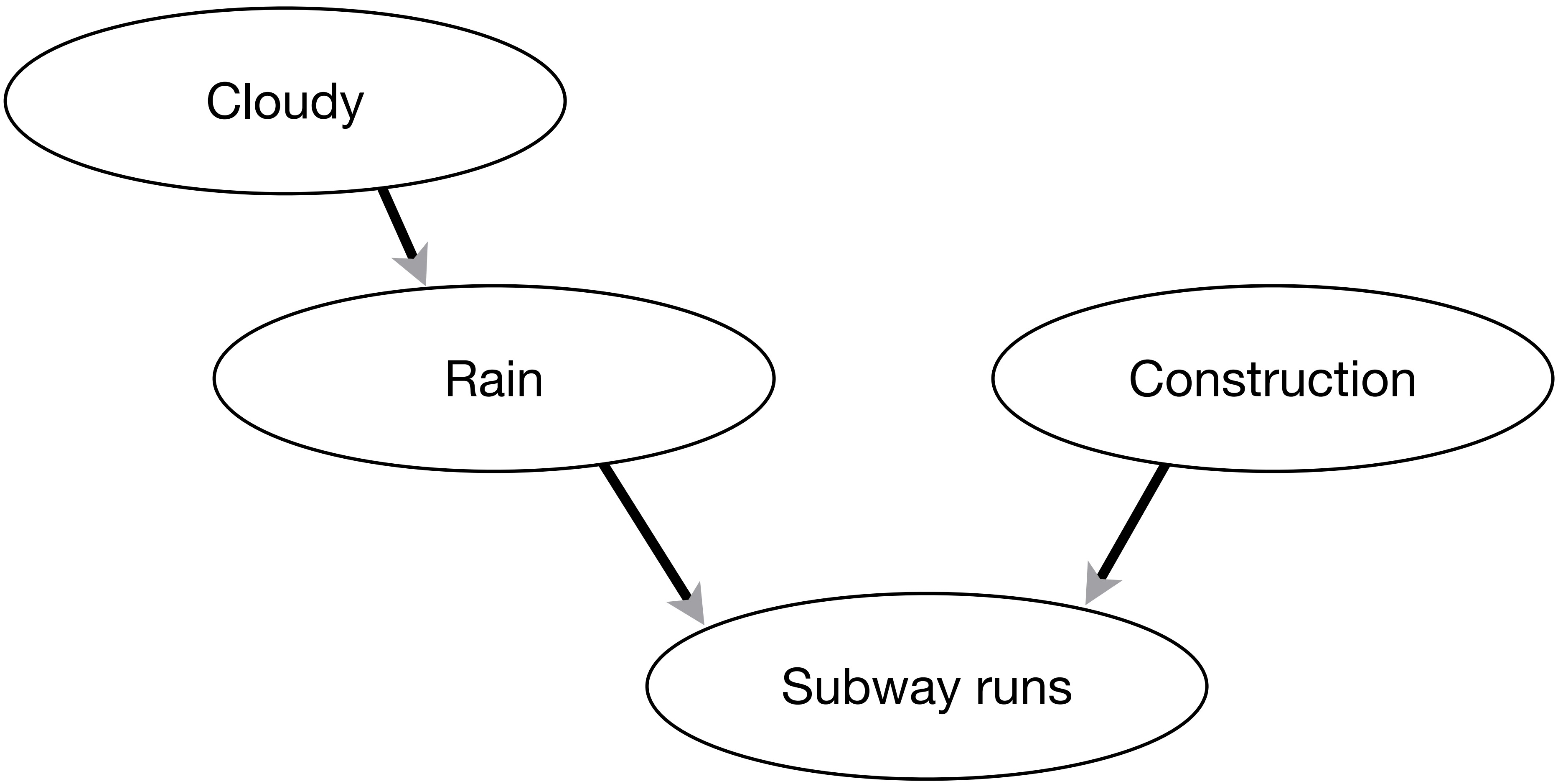
Probabilistic Inference

- Some of these decisions are too hard to compute exactly, and often there is insufficient information to make an exact decision
- Instead, model uncertainty via probability
- An important application for graph theory is using graphs to represent **probabilistic independence**

Independent Coins

- 1. Suppose I flip coin twice, what is the probability of both flips landing heads?
- 2. Compare to if we flip a coin, and if it lands heads, we buy 2 lottery tickets. If tails, we buy 1 lottery ticket. What is the probability we will win the lottery?
- In Scenario 1, we reason with less computation by taking advantage of independence

A Simple Bayesian Network



Basic Rules of Thumb

- Trees and DAGs are easier to reason
    - We can use similar strategy to Topological sort:
    - Only do computation once all incoming neighbors have been computed
- Cyclic graphs are difficult; NP-hard in some settings

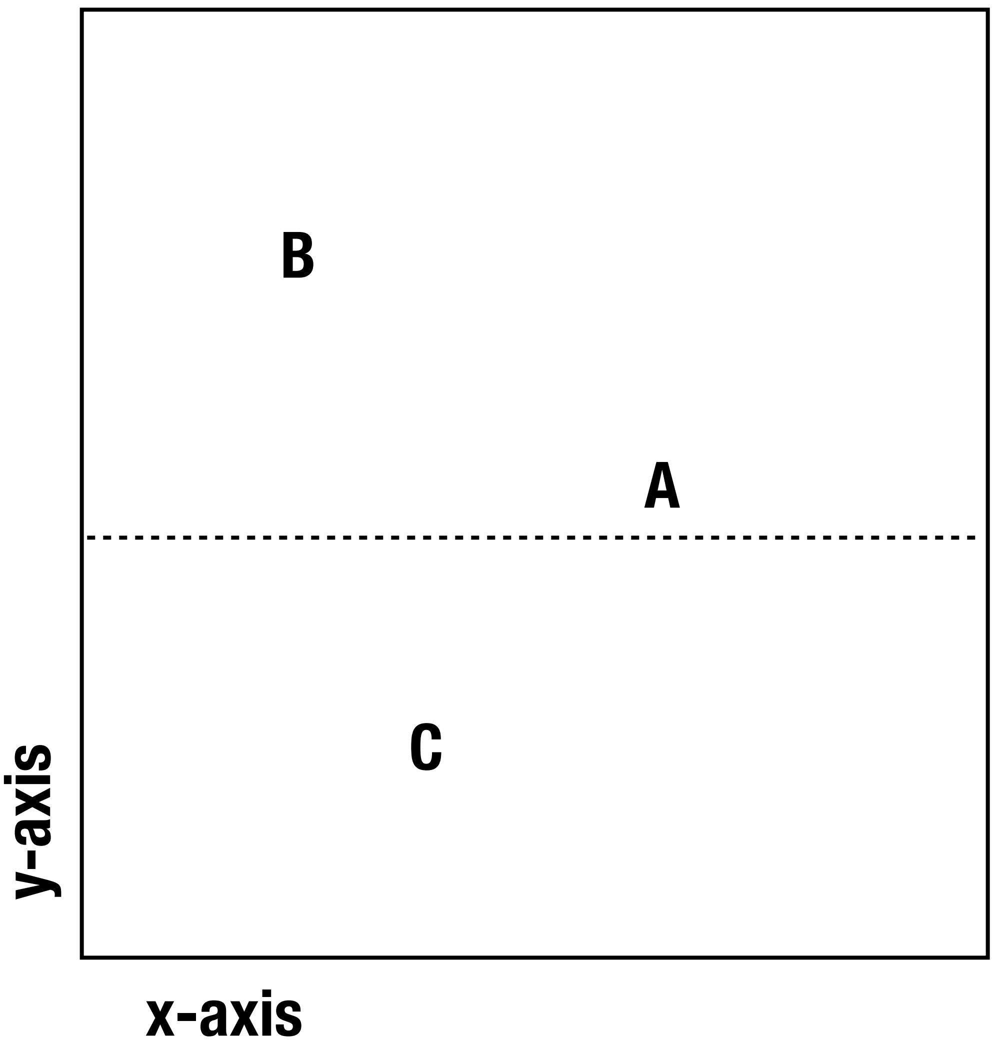
Machine Learning

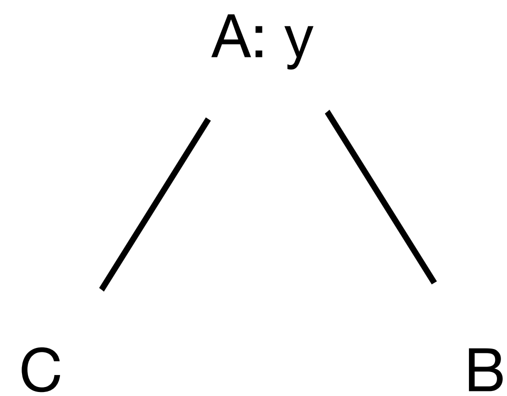
- Another related field, Machine Learning, handles making intelligent decisions after looking at data
  - e.g., a list of surveyed voters, their demographic information, answers to questions, location, etc.
- We typically think of each of these data points as a high-dimensional vector
- We need smart data structures to allow efficient spatial reasoning (e.g., finding nearest neighbors)

kd-trees

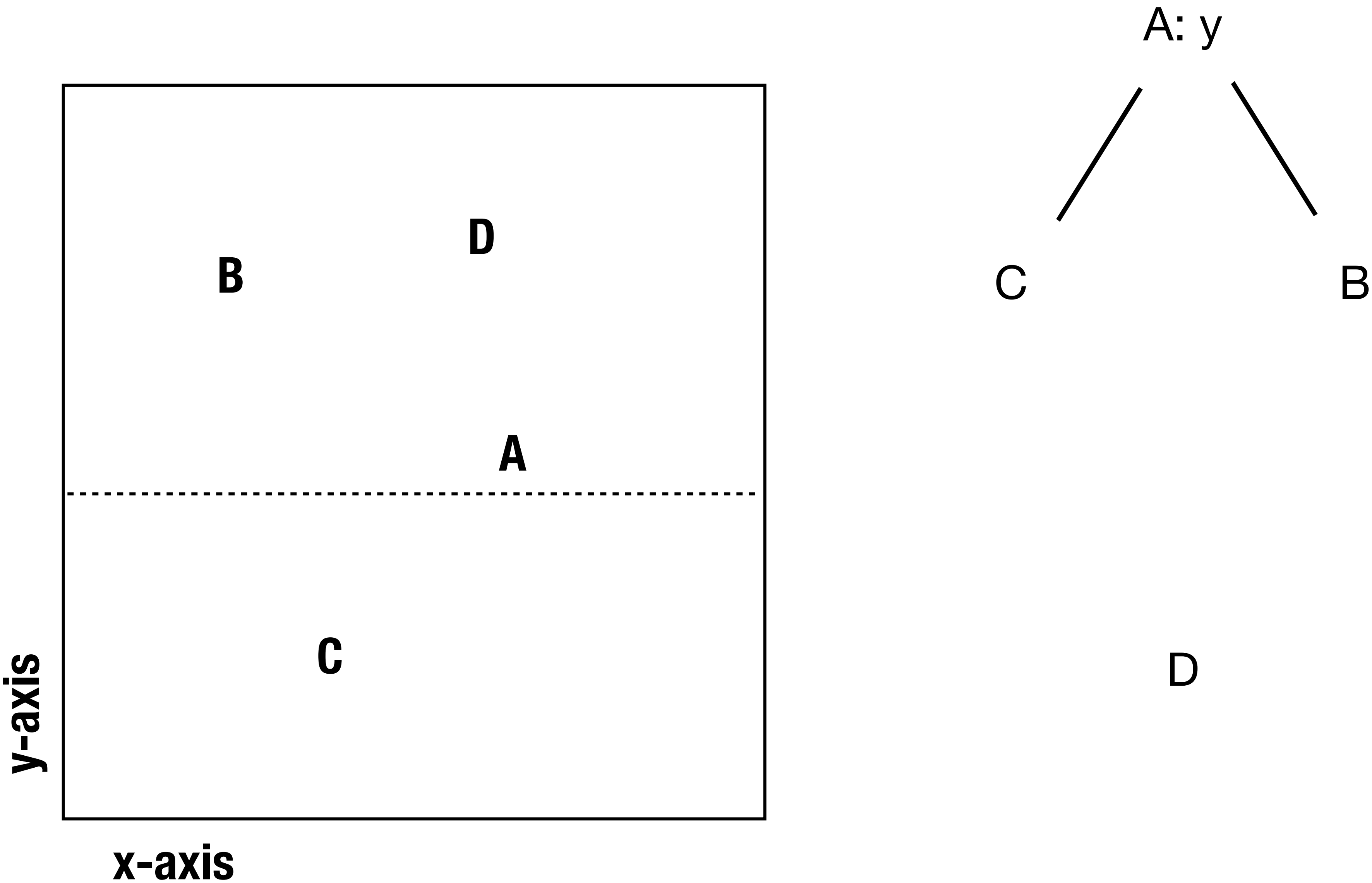
- A kd-tree is a multidimensional binary search tree
  - a BST that partitions in k-dimensions
- Each node specifies a dimension (x, y or z).
  - Left subtree is less than node in that dimension
  - Right subtree is greater than node in that dimension

2d-tree

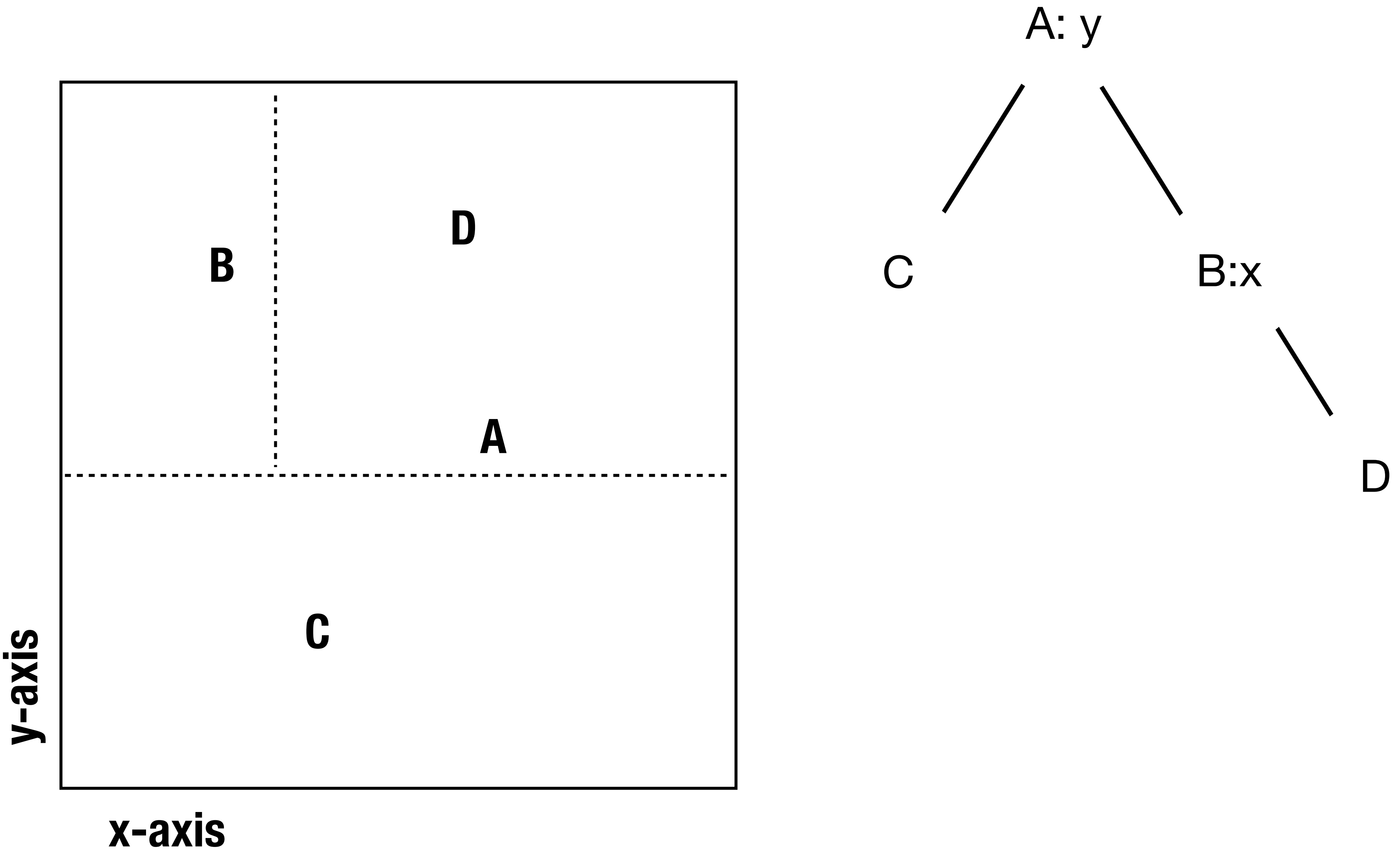




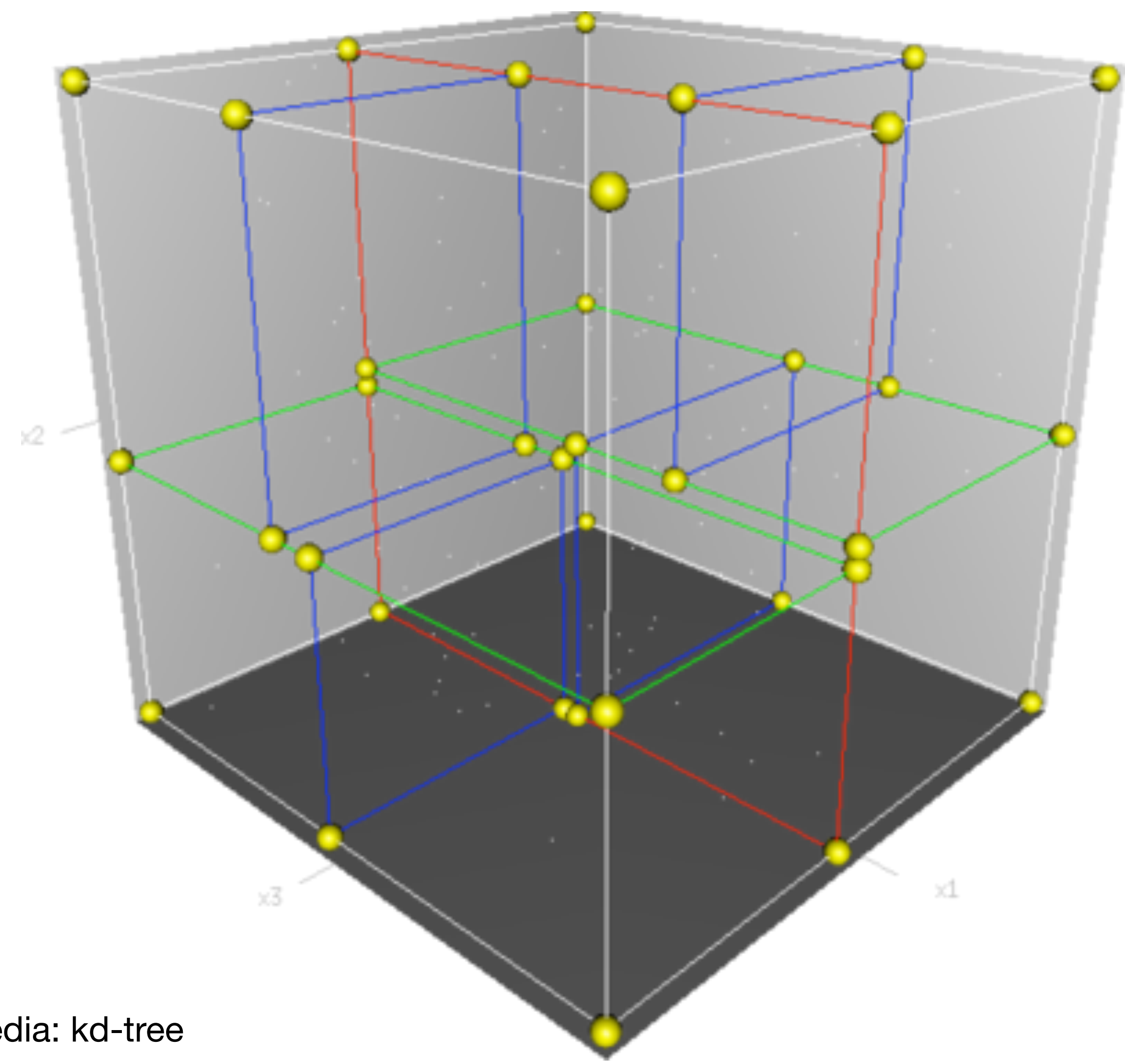
2d-tree



2d-tree



3d-tree



- From Wikipedia: kd-tree

Benefits of kd-trees

- Finding the nearest neighbor of a point is more efficient
- We don't have to compute the distance between all other points
- Only siblings and *some* more distant relatives
- Reduces cost from O(Nk) to O(log N) if balanced, but worst case $O(N^{1-\frac{1}{k}}k)$

Summary

- Three unrelated examples of Data Structures in A.I. and Machine Learning
  - Tree logic useful in analyzing games in A.I.
  - Graph theory useful in probabilistic reasoning
  - kd-trees allow fast computation for handling machine learning data

Final Topics Overview

- Big-Oh definitions (Omega, little o)
- Arraylists/Linked Lists
- Stacks/Queues
- Binary Search Trees: AVL, Splay
- Tries
- Heaps
- Huffman Coding Trees
- Hash Tables: Separate Chaining, Probing
- Graphs: Shortest Path, Max-Flow, Min Spanning Tree, Euler
- Complexity Classes
- Disjoint Sets
- Sorting: Insertion Sort, Shell Sort, Merge Sort, Quick Sort, Radix Sort, Quick Select