CSCI 136
Data Structures & Advanced Programming

Bill Jannen

Lecture 31

May 1, 2017

Lab 9?


one line, less waiting
one line, less waiting, the ♀ starts here.
lane closed

Administrative Details

- This Week: Hexapawn Lab
  - Bailey 313
  - Implement a game and 3 types of players
    - Human, AI, Random
  - Again, you *may* choose to work with a partner
    - **EACH INDIVIDUAL** must come to lab with a design doc for GameTree class
      - We will check at start of lab, -2 points if you come empty-handed

Last Time

- Splay tree demo
- Implemented balance() for BSTs
- Said farewell to Morgan ☹

Today's Outline

- Binary search tree remove()
- Discuss game trees

Recap

- Previously, we looked at several BST methods:
  - Constructor(s)
  - protected BT<E> locate(BT<E> root, E value)
  - public boolean contains(E value)
  - public E get(E value)
  - public void add(E value)
  - protected BT predecessor(BT root)
    - What is the intuition for predecessor?

Removal

- Removing the root is the hardest
- If we figure out how to remove the root, we can remove any element in BST in same way
  - Why?
  - How?
- We need to implement:
  - protected BT<E> removeTop(BT<E> top)
  - public E remove(E value)

removeTop(BT<E> topNode)

Remember the BST requirements:

1. All nodes in the left subtree are <= root
2. All nodes in the right subtree are >= root

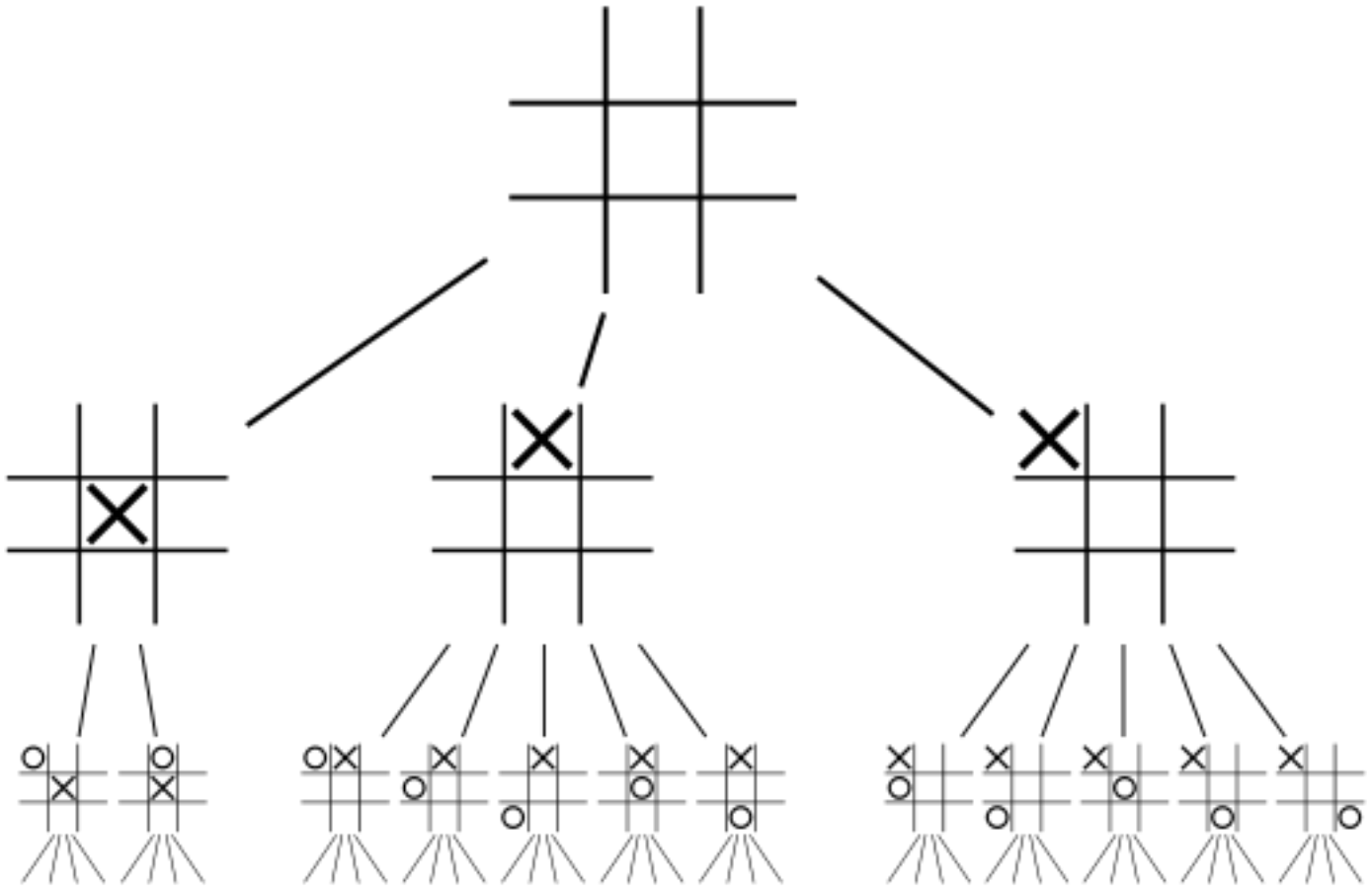
removeTop() cases:

- Case 1: No left BinaryTree
- Case 2: No right BinaryTree
- Case 3: Left node has no right subtree
- Case 4: Everything else (general case)

remove(E value)

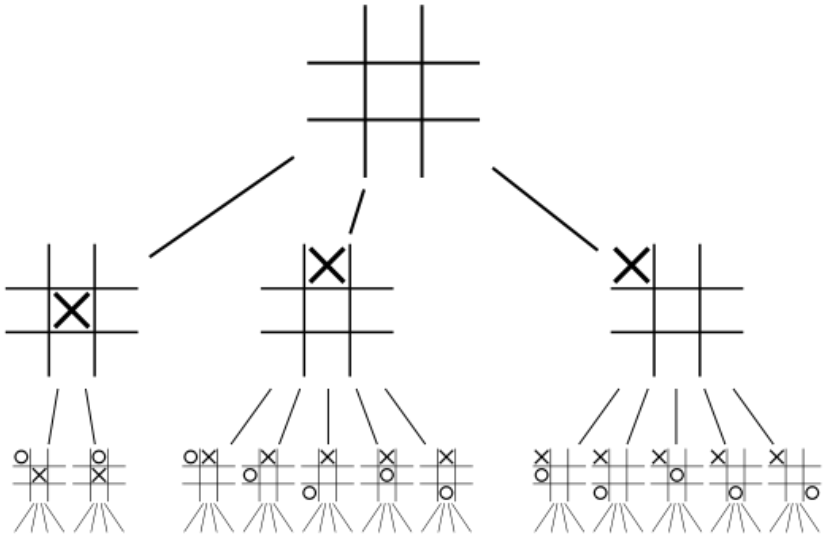
- Locate the target node with the target value
- removeTop() the target node
- Adjust target node's old parent to point to the root of the new subtree

Game Trees



Game Trees

- Nodes are positions in a game (game state)
- Edges are moves (transition from one game state to another)
  - All nodes at a given level represent moves by the same player
- Leaf nodes represent ending board states (winner or tie)
  - # of leaf nodes = # of ways a game can be played

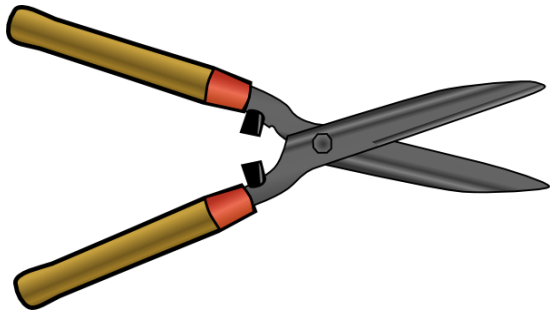


Game Trees

- In AI, often search the game tree and use an algorithm like **minimax** to choose the next "best move"
  - Chess, checkers, Go, etc.
  - Why not real-time games

Game Trees

- The **complete game tree**: the root is the initial game state and the tree contains all possible moves from each position
  - You will build complete Hexapawn game trees
  - But your computer player will "prune" the losing branches



Backwards Induction (from Wikipedia)

- Pick 3 colors: player 1 win (P1W), player 2 win (P2W), and tie (T).
- Color leaves (height 0) of the game tree so that:
  - all wins for player 1 are colored P1W,
  - all wins for player 2 are colored P2W,
  - all ties are T.
- Look at height 1 nodes. For each node:
  - If any child is colored for the current player's opponent, color this for the current player's opponent
  - If all children are colored for the current player, color this node for the current player
  - Otherwise, color this node for a tie
- Repeat for each level, moving upwards, until all nodes are colored.
- The color of the root node is the outcome of optimal play.

Backwards Induction (from Wikipedia)