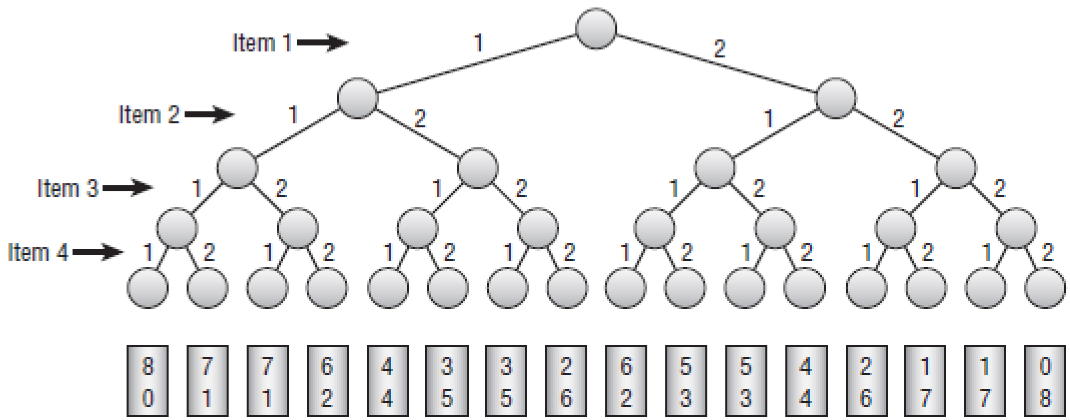
Intro to Decision Trees



Agenda

Decision Trees

- Each branch represents a single choice
- A leaf node represents a complete set of decisions that produces a final solution
- The goal is to find the best possible set of choices or the best leaf node in the tree
- Types:
  - Game trees
  - Optimization decision trees

Searching Game Trees

- Game trees grow extremely quickly
- A 40 move chess game (each player moves 20 times) and has an average of about 30 possible moves per turn gives a game tree with roughly $30^{40} \approx 1.2 \times 10^{59}$ possible paths
- Searching 1 billion paths per second would take roughly 2.3×10^{44} years

Tic-tac-toe Game Trees

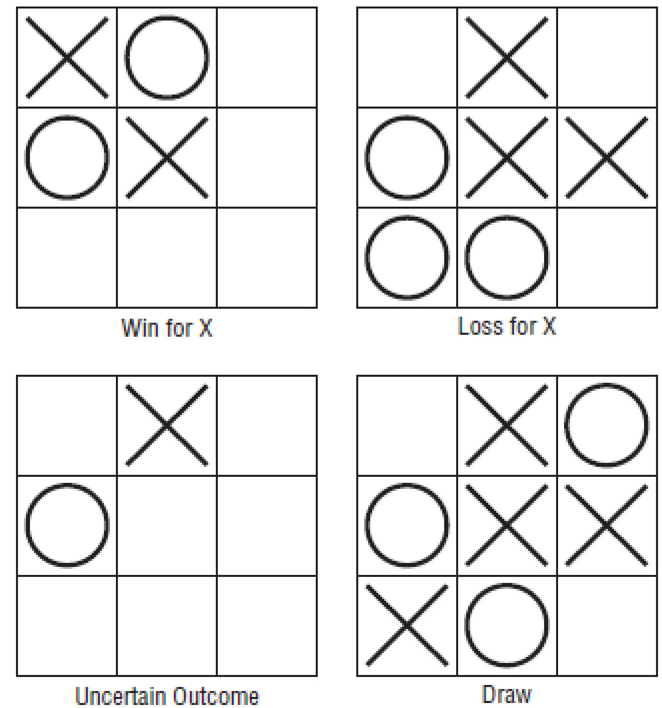
- Simpler than chess but still a big game tree
- $9 \times 8 \times 7 \times \ldots x\ 1 = 9! = 362{,}880$ possible paths

Minimax

- Recursively follow the path that minimizes the maximum value your opponent can get
- Search to a maximum depth of recursion

Minimax Board Values

- 4 A win for this player
- 3 Outcome unclear
- 2 Draw
- 1 A loss for this player



Minimax Code, Part 1

```
// Find the best move for player1.
Minimax(Board: board_position, Move: best_move, Value: best_value,
  Player: player1, Player: player2, Integer: depth, Integer: max_depth)
    // See if we have exceeded our allowed depth of recursion.
    If (depth > max_depth) Then
        // We have exceeded the maximum allowed depth of recursion.
        // The outcome for this board position is unknown.
        best_value = Unknown
        Return
    End If

    // Find the move that gives player2 the lowest value.
    Value: lowest_value = Infinity
    Move: lowest_move
    For Each <possible test move>
        ... continued ...
```

Minimax Code, Part 2

```
        <Update board_position to make the test move>
        // Evaluate this board position.
        If <this is a win, loss, or draw> Then
            <Set lowest_value and lowest_move appropriately>
        Else
            // Recursively try other future moves.
            Value: test_value
            Move: test_move
            Minimax(board_position, test_move, test_value,
                player2, player1, depth, max_depth)

            // See if we found a worse move for player2.
            If (test_value < lowest_value) Then
                // This is an improvement. Save it.
                lowest_value = test_value
                lowest_move = test_move
            End If
        End If
<Restore board_position to unmake the test move>
Next <possible test move>
// Save the best move
```

Minimax Code, Part 3

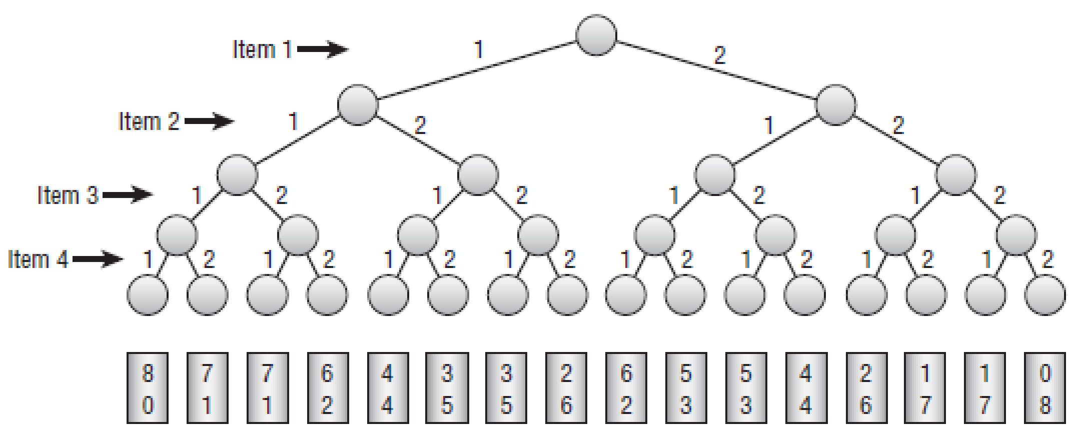
```
        <Restore board_position to unmake the test move>
    Next <possible test move>

    // Save the best move.
    best_move = lowest_move

    // Convert board values for player2 into values for player 1.
    If (lowest_value == Win)
        best_value = Loss
    Else If (lowest_value == Loss)
        best_value = Win
    Else
        ... <Convert other board values> ...
    End If
End Minimax
```

Game Tree Improvements

- Pre-computed initial moves and responses
- Game tree heuristics
  - Patterns in board position (long series of trades, castling, queen promotion, discovered check, fork, en passant, etc.)
  - Board square values

Searching General Decision Trees

- Partition problem
  - Divide a set of objects with values into two sets with the same total value
- Left and right branches represent putting an object into set 1 or set 2
- Leaf nodes correspond to complete solutions



Reporting and Optimization

- Reporting: Is a solution possible?
- Optimization: What is the best possible solution?

Exhaustive Search

- Search the entire decision tree
- See which leaf nodes give improved solutions
- Always works but time consuming

Exhaustive Search Code

```
StartExhaustiveSearch()
    <Initialize best solution so it is replaced by the first test solution>
    ExhaustiveSearch(0)
End StartExhaustiveSearch

ExhaustiveSearch(Integer: next_index)
    // See if we are done.
    If <next_index > max_index>
        // We have assigned all items, so we are at a leaf node.
        <If the test solution is better than the best solution found so far, save it>
    Else
        // We have not assigned all items, so we are not at a leaf node.
        <Assign item next_index to group 0>
        ExhaustiveSearch(next_index + 1)
        <Unassign item next_index to group 0>
        <Assign item at next_index to group 1>
        ExhaustiveSearch(next_index + 1)
        <Unassign item next_index to group 1>
    End If
End ExhaustiveSearch
```

Improving Exhaustive Search

- If you detect an optimal solution, stop early
- Example: 20 item partition problem
  - Full tree: 2,097,150 nodes
  - In one test, a perfect division was found after checking only 4,098 nodes
  - Results vary greatly depending on the item values

Branch and Bound

- After moving down a branch:
  - Calculate the best possible outcome. If that doesn't improve the current best solution, abandon that path.
- Still finds an exact solution

Branch and Bound Code

```
StartBranchAndBound()
    <Initialize best solution so it is replaced by the first test solution>
    BranchAndBound(0)
End StartBranchAndBound
```

Branch and Bound Code (continued)

```
BranchAndBound(Integer: next_index)
    // See if we are done.
    If <next_index > max_index>
        // We have assigned all items, so we are at a leaf node.
        <If the test solution is better than the best solution, save it>
    Else
        // We have not assigned all items, so we are not at a leaf node.
        If <the test solution cannot be improved enough
            to beat the current best solution> Then Return

        <Assign item next_index to group 0>
        BranchAndBound(next_index + 1)
        <Unassign item next_index to group 0>
        <Assign item next_index to group 1>
        BranchAndBound(next_index + 1)
        <Unassign item next_index to group 1>
    End If
End BranchAndBound
```

Branch and Bound Performance

- Recall: In a binary tree, roughly half of the nodes are leaf nodes so skipping the bottom of the tree removes around half of the nodes
- After finding a solution, branch and bound often trims higher-level branches and that removes many nodes lower down
- Example: 20 item partition problem
  - Full tree: 2,097,150 nodes
  - Exhaustive search with early stopping: 4,098 nodes
  - Branch and bound: 298 nodes

Summary

- Decision Trees
- Searching Game Trees
  - Minimax
- Searching General Decision Trees
  - Reporting and Optimization
  - Exhaustive Search
  - Branch and Bound

Exercises

- Chapter 12 Exercises 1 – 5.
- Bonus: Chapter 12 Exercise 6.
- Read *Essential Algorithms, 2e* Chapter 12 pages 381 – 402. (The rest of Chapter 12.)