Depth-First Search, Tree Traversals and Min-Max AI

CS 151: Introduction to Computer Science I

Pseudo-Code for Depth-First Search

Input to the DFS algorithm is the adjacency list representation of a graph and a vertex

```
procedure DFS(G,v):
  label v as discovered
   for all vertices w in the adjacency list of v do
    if vertex w is not labeled as discovered then
        recursively call DFS(G,w)
```

Tree traversals

4 commons ways to traverse a tree

- Pre-order traversals
- In-order traversals
- Post-order traversals
- Level-order traversal

Pre-order traversals

- Visit the root.
- Traverse the left subtree.
- Traverse the right subtree.

Inorder traversals

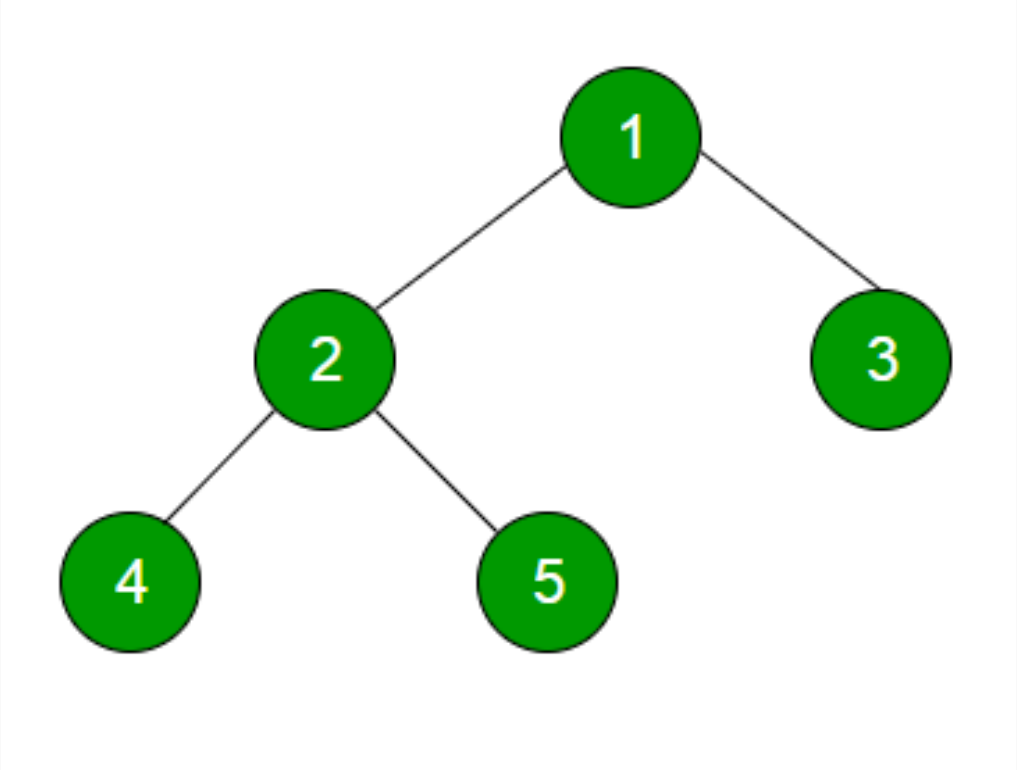
- Traverse the left subtree.
- Visit the root.
- Traverse the right subtree.

Post-order traversals

- Traverse the left subtree.
- Traverse the right subtree.
- Visit the root.

Level-order traversals

In level order traversal, we visit the nodes level by level from left to right.



The level order traversal of above tree is 1, 2, 3, 4 and 5.

A Simple Checkers AI

Recall our type for the game state of a checkers game:

```
(define-type Piece
  ([loc :  Loc]
   [color :  Color]))

(define-type Checkers
  ([pieces :  (Listof Piece)]
   [turn :  Color]))
```

A type for a simple AI:

```
(define-type Move
  ([start :  Loc]
   [end :  Loc]))
(:  simple-checkers-ai :  Checkers -> Move)
```

A Simple Checkers AI

```
(: simple-checkers-ai : Checkers -> Move)
```

Which move should we pick?

```
(: evaluate : Checkers -> Integer)
```

...then pick the move that leads to the highest evaluation

Example:

evaluate = # black pieces -# red pieces

Min-max Checkers AI

Looking one move ahead is a good start. Real pros have two good skills:

- A very good sense of which board positions are good or bad (an improved `evaluate` function)
- The ability to think many moves ahead

How should we incorporate boards that are two moves away?

Assume that your opponent will pick the move that minimizes `evaluate`.

Then simulate the game.

Min-max Checkers AI

Min-max is a method to *bootstrap* your `evaluate` function into a better one.

```
(:  min-max-eval :  (Checkers -> Integer) Integer
    Checkers -> Integer)
(define (min-max-evaluate eval lookahead board)
  (cond
    [(= 0 lookahead) (eval board)]
    [else (foldr (if (black-turn?  board) max min)
                 (if (black-turn?  board) -∞ +∞)
           (map
            (lambda ([g :  Checkers])
             (min-max-eval eval (- lookahead 1) g))
            possible-moves))]))
```

Min-max Checkers AI

Curried version:

```
(: cur-min-max-evaluate : (Checkers -> Integer)
Integer -> Checkers -> Integer)

(: improved-evaluate : Checkers -> Integer)
(define improved-evaluate
  (cur-min-max-evaluate evaluate 5))
```

Then the AI picks the move that maximizes its evaluation function, like before.

```
(: min-max-ai : Checkers -> Move)
```

What to know

- Depth-first search
- Tree Traversal
- Min-Max AI