Trees

Assignment #4

Tuesday, February 16: Read Rosen and Write Essay

Sections 9.6, 9.7, 9.8 -- Pages 647-672

Thursday, February 18: Read Rosen and Write Essay

Section 10.1, 10.2, 10.3 -- Pages 683-722

Assignment #4

Homework Exercises

Section 9.1: Problem 31

Section 9.2: Problems 18, 28

Section 9.3: Problems 30, 31

Section 9.4: Problems 54, 55

Section 9.5: Problems 10, 26, 27, 46, 54

Extra Credit Problem: See Comp 280 web page

Motivation for Trees

Many, Many Applications

Fundamental Data Structure

Neat Algorithms

Simpler than Graphs

Examples and Animations

http://oneweb.utc.edu/~Christopher-Mawata/petersen/

<u>Definitions</u>

Tree

- Connected graph with no simple circuits.
- Graph with a unique path between any two vertices.

Rooted Tree

- Explicit Definition
  -- A tree with one vertex called the *root*
- Recursive Definition
  -- A single vertex r is a rooted tree with root r.
  -- If $T_1, \ldots, T_n$ are rooted trees with roots $r_1, \ldots, r_n$ and r is a new vertex, then the graph with edges joining r to $r_1, \ldots, r_n$ is a rooted tree with root r.

Counting Theorems

Theorem 1: $e = v - 1$

Proof: By induction on the number of vertices.
(Inductive Step: Remove a leaf.)

Theorem 2: $\# \, leaves \leq m^{height}$ *(m-ary trees)*

Proof: By induction on the height of the tree.
(Inductive Step: Remove the root.)

Theorem 3: $height \geq \log_m(\# \, leaves)$

Proof: This result is just a restatement of Theorem 2.

Standard Terminology

Nodes	*Trees*
Ancestor	Level
Parent	Height
Child	Balanced
Sibling	N-ary
Descendent	Binary
Leaf	-- Left Subtree / Child
Internal Vertex (Node)	-- Right Subtree / Child

Examples

Family Trees

Organization Charts

Computer File Systems

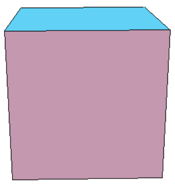
Recursive Calls
-- Neville's Algorithm
-- Bezier Subdivision

Parallel Processors

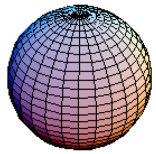
Computer Graphics
-- Constructive Solid Geometry (CSG-Trees)
-- Binary Space Partition Trees (BSP-Trees)

Constructive Solid Geometry

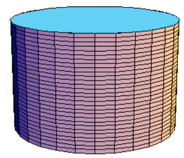
Primitive Solids



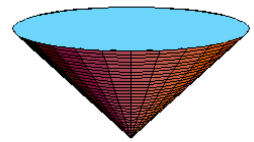
BOX



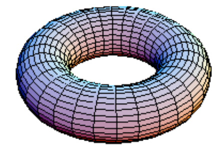
SPHERE



CYLINDER



CONE



TORUS

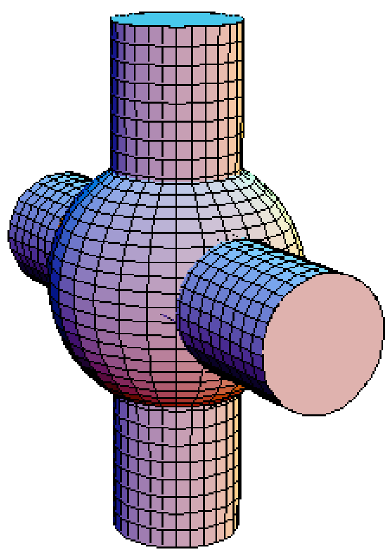
Boolean Operations

- Union
- Intersection
- Difference

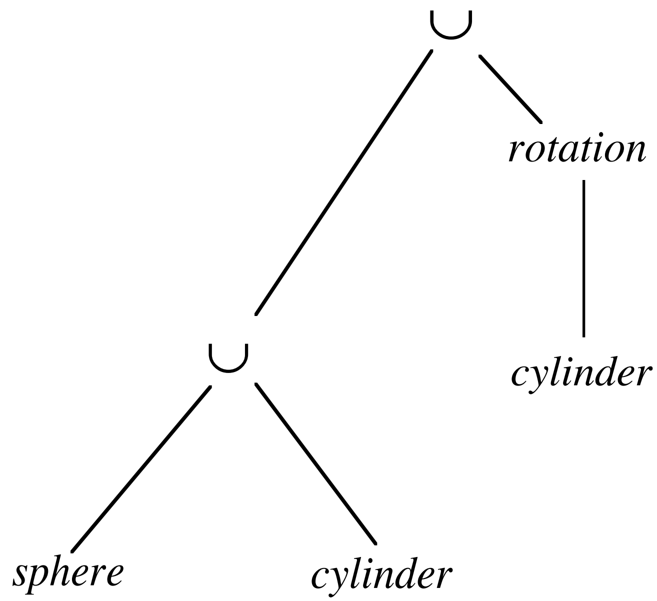
CSG-Tree

- Leaves = Primitive Solids
- Internal Nodes = Boolean Ops or Transformations
- Root = Solid

CSG-Tree

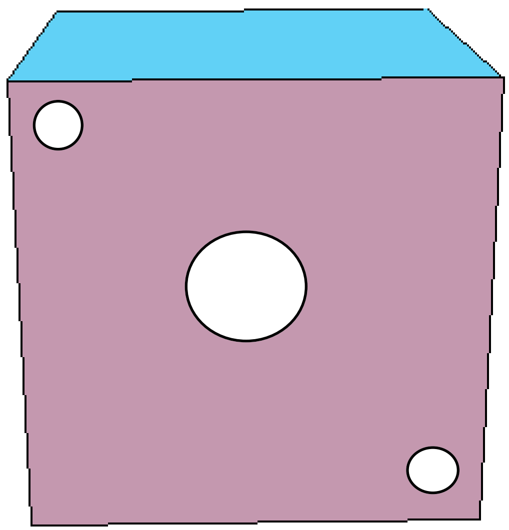


Spherical tank with two cylindrical pipes

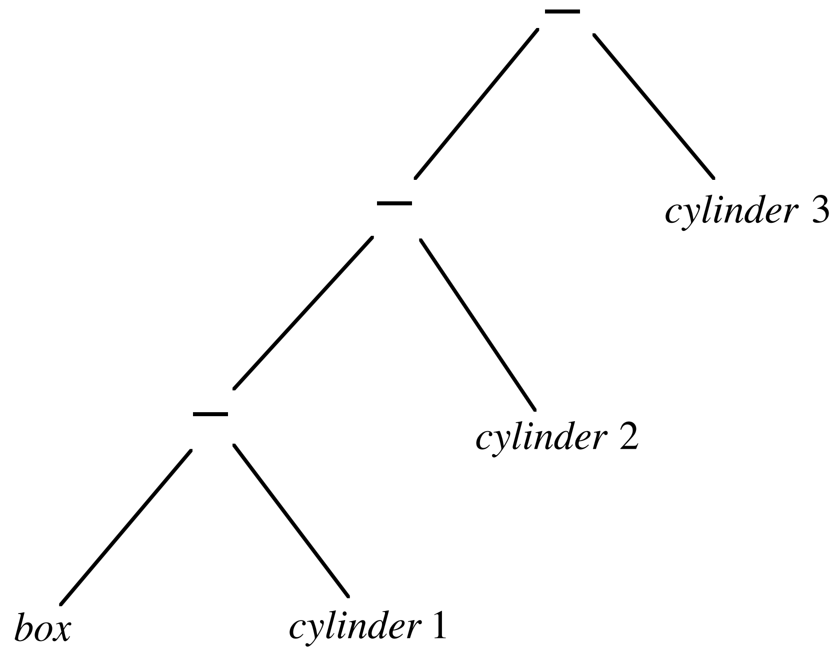


CSG tree: union of 1 sphere and 2 cylinders

CSG-Tree



Solid block with three cylindrical holes,



CSG tree: subtract three cylinders from box

Applications

Minimal Spanning Trees

-- Minimizing Network Cost

Searching and Sorting

-- BSP-Trees (Computer Graphics)

Searching Arbitrary Trees

-- Breadth First Search

-- Depth First Search

  -- Back Tracking

  -- Graph Coloring Algorithm

Data Compression (Efficient Coding)

-- Prefix Coding -- Horner's Method

-- Huffman Coding

Game Trees

-- Min-Max Strategy

Binary Search Trees

Applications

- Fast Searching and Sorting any Ordered Collection

  -- Dictionaries

  -- Telephone Books

- Computer Graphics / Computer Games

  -- Hidden Surface Algorithms

  -- Fast Polygon Shading (BSP Trees)

Examples and Animations

http://www.cosc.canterbury.ac.nz/mukundan/dsal/BSTNew.html

Algorithms for Binary Search Trees

Sorting

If $v < root$, Insert into Left Subtree

Else $v > root$, Insert into Right Subtree

Searching

If $v = root$, FOUND

Else if $v < root$, Search Left Subtree

Else $v > root$, Search Right Subtree

<u>Binary Space Partitioning Trees (BSP-Trees)</u>

<u>Algorithm for Generating a BSP–Tree</u>

- Select any polygon (plane) in the scene for the root.
- Partition all the other polygons in the scene to the back (left subtree) or the front (right subtree).
- Split any polygons lying on both sides of the root.
- Build the left and right subtrees recursively.

<u>BSP-Tree Rendering Algorithm</u> (In Order Tree Traversal)

- If the eye is in front of the root, then
  - -- Display the left subtree (behind)
  - -- Display the root
  - -- Display the right subtree (front)
- If the eye is in back of the root, then
  - -- Display the right subtree (front)
  - -- Display the root
  - -- Display the left subtree (back)

Binary Space Partitioning Trees (continued)

Advantages

- Can use the same BSP–tree for different positions of the eye.
- When we want to move around in a scene, the BSP-tree is the preferred approach to detecting hidden surfaces.

Binary Space Partitioning Trees

http://maven.smith.edu/~mcharley/bsp/createbsptree.html

Ordered Tree Traversal

Pre Order

- Visit the Root
- Pre Order Traverse the Children $T_1,\ldots,T_n$

In Order

- In Order Traverse T_1
- Visit the Root
- In Order Traverse the Children $T_2,\ldots,T_n$

Post Order

- Post Order Traverse the Children $T_1,\ldots,T_n$
- Visit the Root

Applications

CSG Tree Evaluation Algorithm

- In Order Tree Traversal

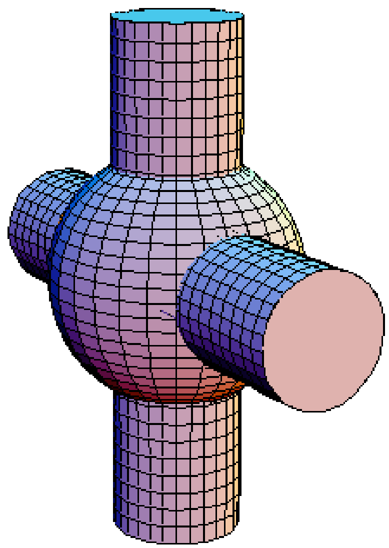
Arithmetic and Logical Expressions

- Infix Form
- Prefix Form (Polish Notation)
- Postfix Form (Reverse Polish Notation)

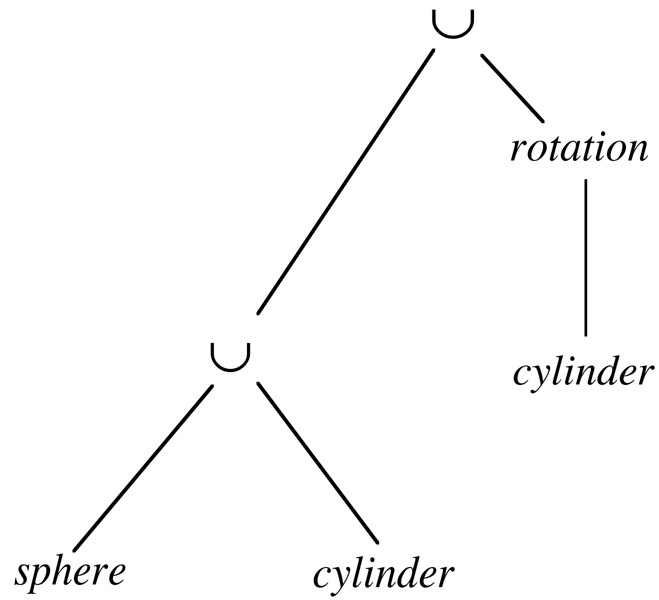
BSP-Tree Rendering Algorithm

- Back to Front -- In Order
- From to Back -- Reverse In Order

<u>CSG Tree Evaluation Algorithm -- In Order Tree Traversal</u>



Spherical tank with two cylindrical pipes



CSG tree: union of 1 sphere and 2 cylinders

BSP-Tree Rendering Algorithm

- If the eye is in front of the root (In Order Tree Traversal)
  - -- Display the left subtree (behind)
  - -- Display the root
  - -- Display the right subtree (front)
- If the eye is behind the root (Reverse Order Tree Traversal)
  - -- Display the right subtree (front)
  - -- Display the root
  - -- Display the left subtree (back)

Tree Searching Algorithms

Depth First Search (Back Tracking)

- Base Case: Search the root.
- Recursion: For each vertex adjacent to the root, perform Depth First Search.
- STOP When object is found or all vertices have been searched.

Breadth First Search

- Level 0: Search the root.
- Level 1: Search all the children of the root.

$\vdots$ $\vdots$

- Level n: Search all the children incident to parents on level $n-1$.
- STOP When object is found or all vertices have been searched.

Complexity

- $O(n^2) = O(e)$

Tree Searching Algorithms

Depth First Search (Back Tracking)

http://www.rci.rutgers.edu/~cfs/472_html/AI_SEARCH/SearchAnimations.html

Breadth First Search

http://www.rci.rutgers.edu/~cfs/472_html/AI_SEARCH/SearchAnimations.html

<u>**Applications**</u>

Depth First Search -- Backtracking

n Queen Problem

http://www.apl.jhu.edu/~hall/NQueens.html

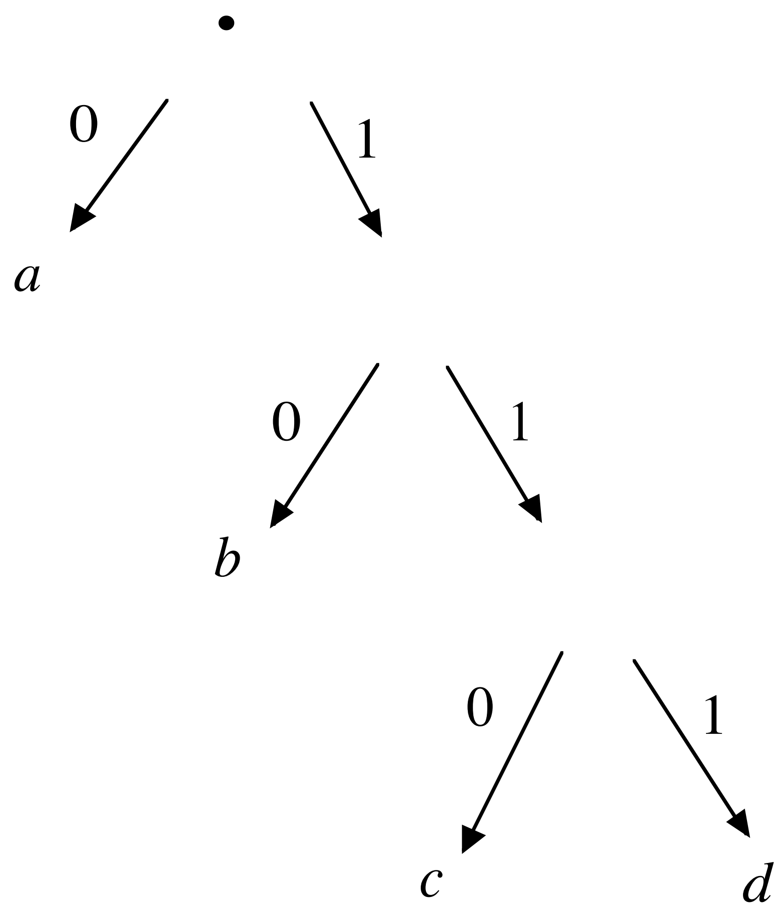
http://www.animatedrecursion.com/advanced/the_eight_queens_pr
oblem.html

Applications

- Graph Coloring
  http://oneweb.utc.edu/~Christopher-Mawata/petersen/lesson8.htm

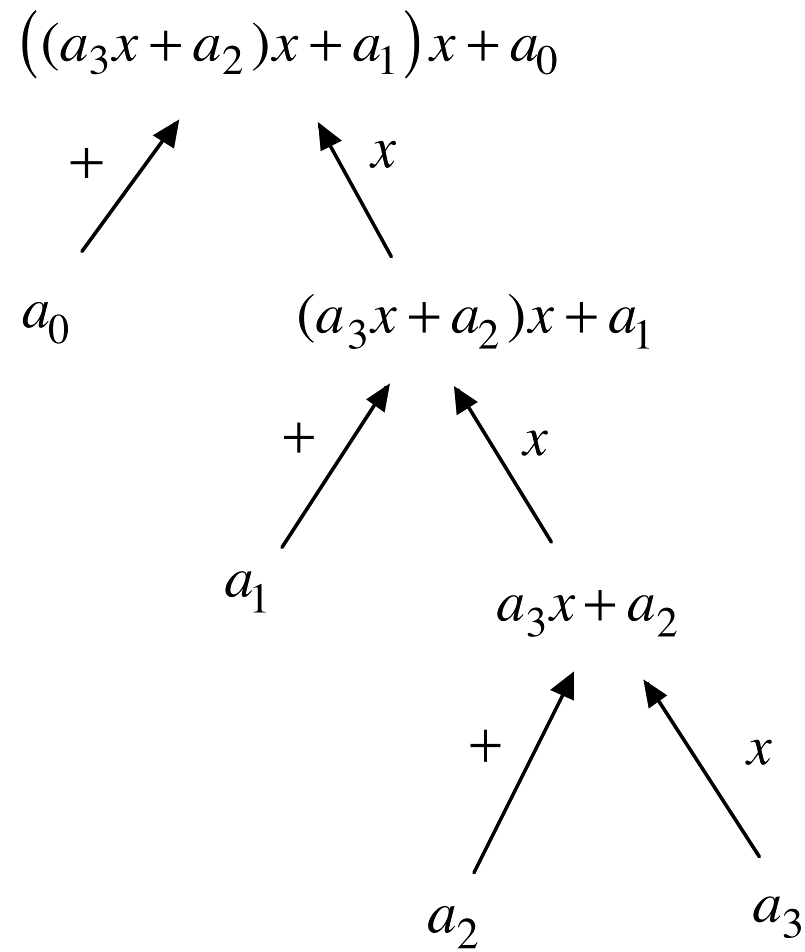
- Web Spiders

Prefix Coding



Efficient, Non-Redundant Coding

Horner's Method



Fast Polynomial Evaluation -- $O(n)$ Multiplications

Huffman Coding

Coding Algorithm

- Assign probability to each symbol
- Combine trees (and their probabilities) with smallest probabilities
  -- Smaller probability to right $\rightarrow$ 1
  -- Larger probability to left $\rightarrow$ 0
- Symbol code = unique path of 0's and 1's from root

Proof of Optimality

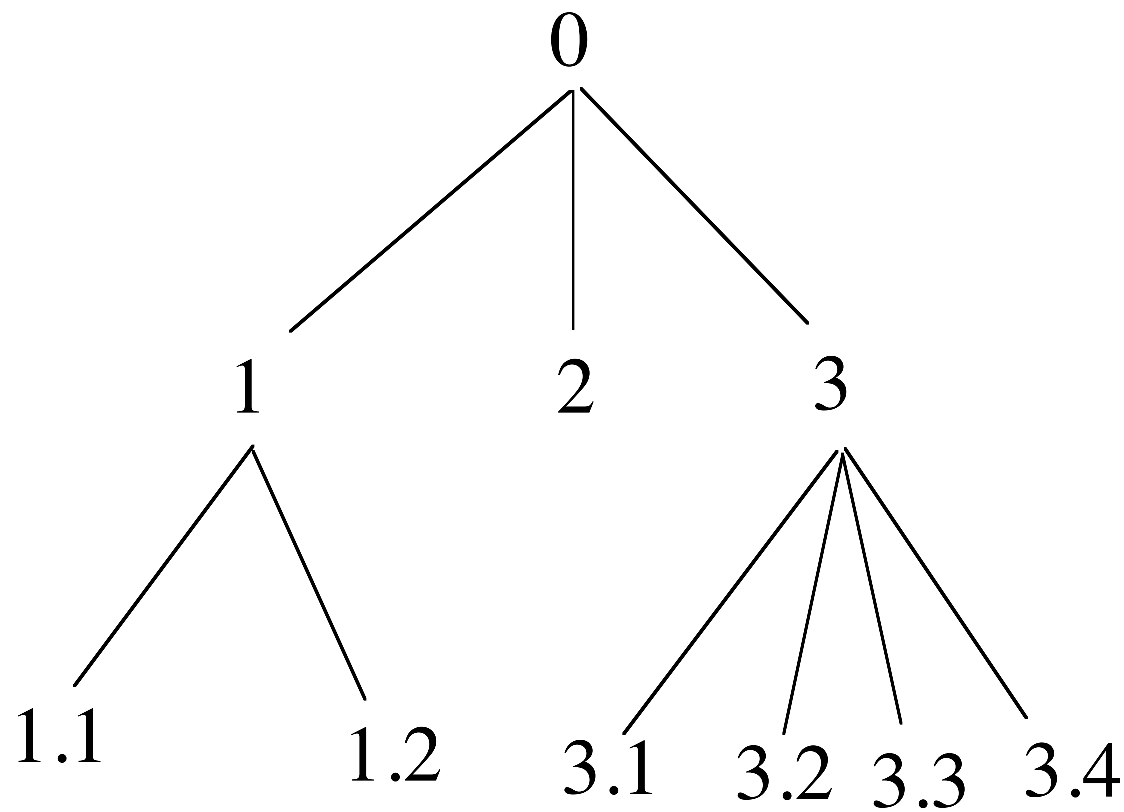
- Homework

Huffman Coding

http://www.cs.duke.edu/csed/poop/huff/info/

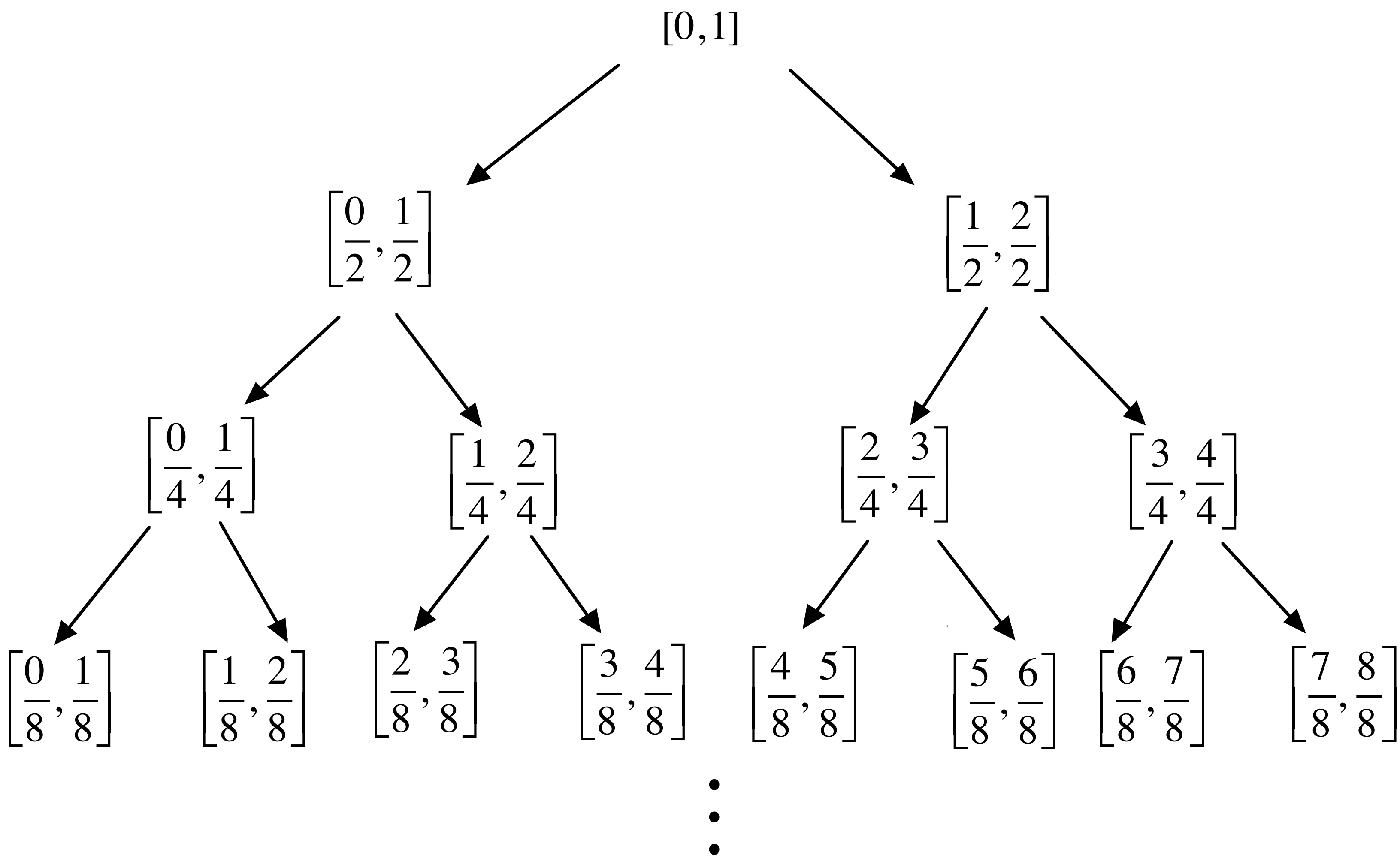
http://www.maths.abdn.ac.uk/~igc/tch/mx4002/notes/node59.html

Universal Address System

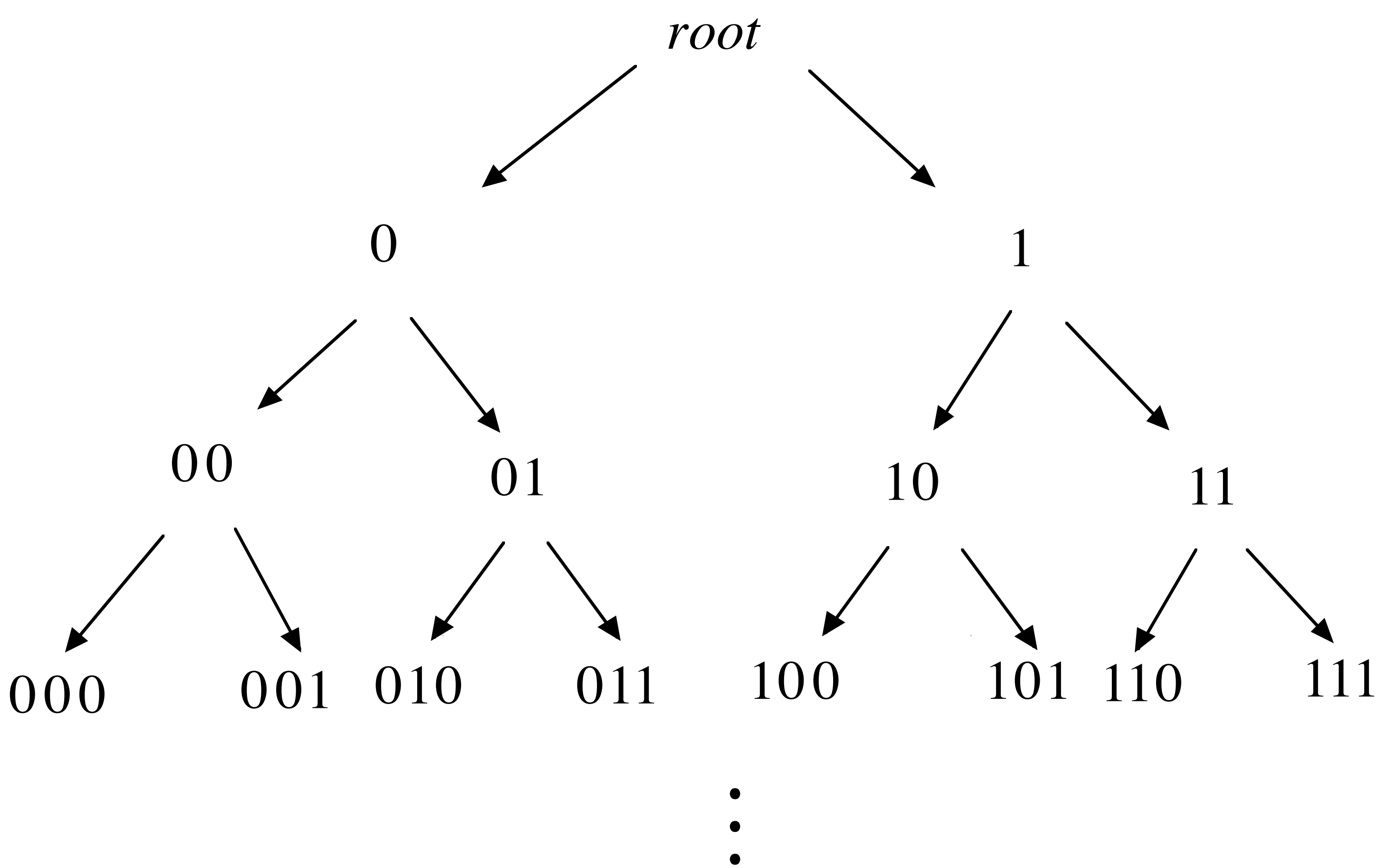


Lexicographic Order (Depth First)

Interval Subdivision

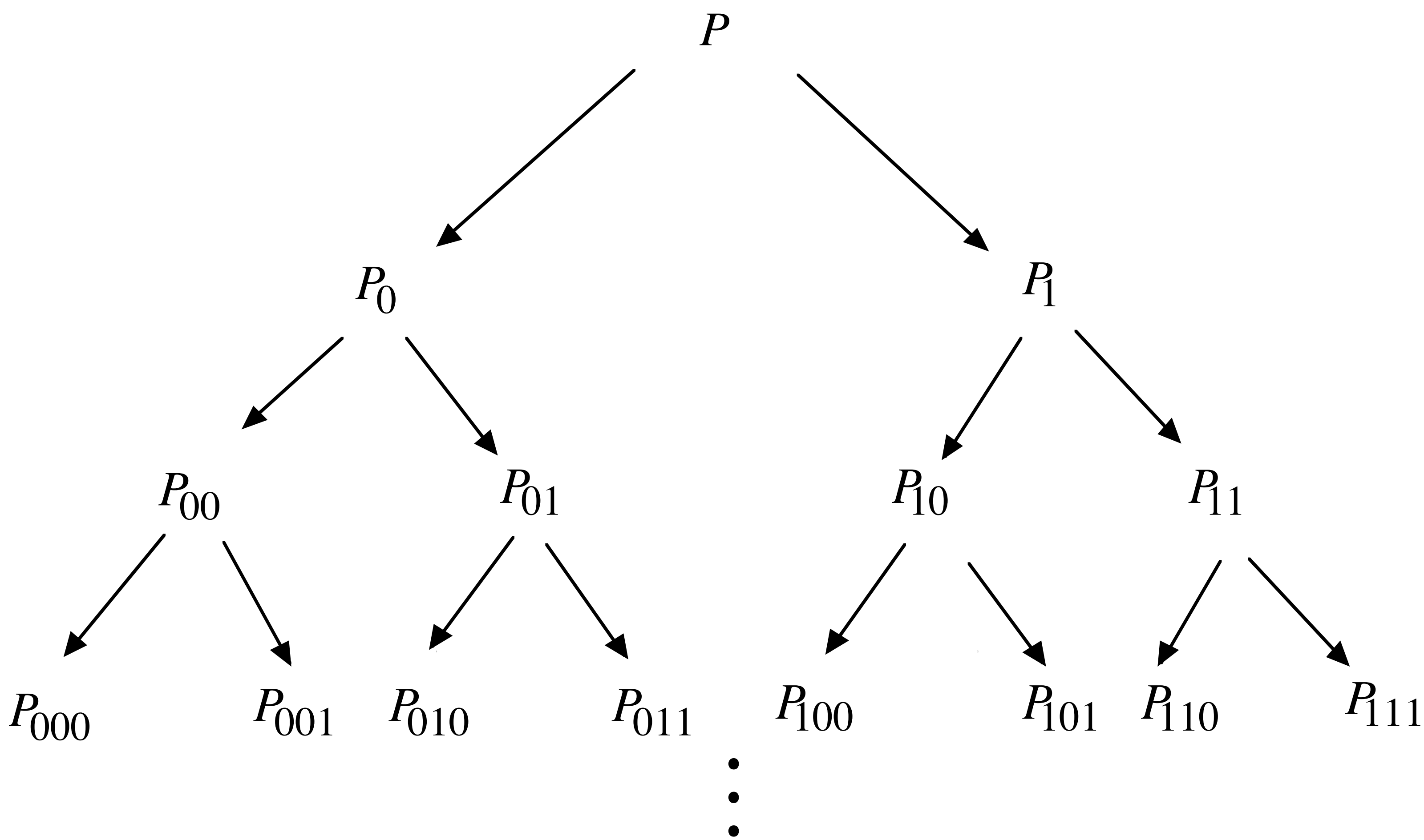


Interval Subdivision



$$b_1 \cdots b_n \leftrightarrow [.b_1 \cdots b_n,\ .b_1 \cdots b_n + 2^{-n}]$$

Bezier Subdivision



$$P_{b_1 \cdots b_n} \leftrightarrow \textit{Control Points for the Interval } [.b_1 \cdots b_n,\ .b_1 \cdots b_n + 2^{-n}]$$

Game Trees

- Vertices $\leftrightarrow$ Positions
- Edges $\leftrightarrow$ Legal Moves
- Leaves $\leftrightarrow$ Final Positions

  -- Win = 1 (First Player)

  -- Draw = 0 (First Player)

  -- Lose = –1 (First Player)

Game Trees

http://www.youtube.com/watch?v=SO-oXQgvJt4

http://www.youtube.com/watch?v=Unh51VnD-hA

Min-Max Strategy

Payoff -- Recursive Definition

- $payoff(leaf) = value\ at\ leaf$
- $payoff(node\ at\ even\ level) = \max(payoff\ to\ children)$
- $payoff(node\ at\ odd\ level) = \min(payoff\ to\ children)$

Min-Max Strategy

- First Player -- Moves to Child with <u>Maximum</u> Payoff
- Second Player -- Moves to Child with <u>Minimum</u> Payoff

Min-Max Strategy (continued)

Theorem: The Min-Max Strategy is Optimal for both Players

Payoff at each vertex represents payoff to first player if game starts in this vertex and both players play min-max strategy.

Proof: By induction on level.