Data Structures and Algorithms
Data Structures

- static arrays
  - if sorted, supports binary search (\(O(\log n)\))
- linked lists - dynamic size
- doubly-linked lists - easier deletion
- queues - FIFO
- stacks - LIFO
- deques - double-ended queue - push/pop either end
- maps - \(<key,value>\)
  - with hash tables, \(O(1)\) search, insert, delete
Binary Search Trees

- each node holds a value, and has two subtrees
- all values in the left subtree are smaller than the parent's value
- all values in the right subtree are larger than the parent's value
- Search is $O(\log_2 n)$ if tree balanced
AVL Tree

- binary search tree
- height of subtrees of any node differ by at most one (balanced)
- rebalance on insert/delete
Binary search tree == 1-D spatial index

• consider the number line

• each tree node partitions the number line in two

• leaves correspond to convex regions (connected line segments)
Searching a Binary Search Tree

def search(Tree, x):
    """returns leaf containing x"""
    if Tree.isLeaf():
        return Tree
    elif x < Tree.value:
        return search(Tree.leftChild, x)
    else:
        return search(Tree.rightChild, x)
Binary Space Partitioning Tree

- A BSP tree is a binary search tree for N-D space
- Uses (N-1)-D linear splitting elements
  - 2-D: lines
    - line equation:
      - \( ax + by + c = 0 \)
      - \( Ax + c = 0 \)
  - 3-D: planes
    - plane equation:
      - \( ax + by + cz + d = 0 \)
      - \( \mathbf{N}x + d = 0 \) \textbf{(boldface used for vectors)}
      - \( \mathbf{N} = \text{normal to plane} \)
      - \( d = \text{distance from origin} \)
      - if \( \mathbf{N}x + d < 0, x \) is "behind" the plane
      - if \( \mathbf{N}x + d > 0, x \) is "in front of" the plane
def buildBSP ( faces ):
    if len(faces) > 0:
        splitter = chooseSplittingPlane(faces)
        backFaces, inFaces, frontFaces = split(faces, splitter)
        node = BSPTreeNode()
        node.plane = splitter
        node.polys = inFaces
        node.leftChild = buildBSP(backFaces)
        node.rightChild = buildBSP(frontFaces)
        return node
    else:
        return BSPLefNode()
Building a BSP Tree

1, 2, 3, 4, 5
Building a BSP Tree
Building a BSP Tree
Building a BSP Tree
Building a BSP Tree
Building a BSP Tree

[Diagram of a BSP tree with labels and arrows indicating the structure of the tree and the division of space with bounding boxes and lines.]
Building a BSP Tree
Searching a BSP Tree

- Test if point x is "inside" or "outside" of the shape represented by a BSP Tree:

```python
def searchBSP ( Tree, x ):
    if Tree.isLeaf:
        return Tree.label
    dist = Tree.planeEquationAt(x)
    if dist > 0:
        return searchBSP ( Tree.rightChild, x)
    else:
        return searchBSP ( Tree.leftChild, x)
```
Rendering the Polygons in a BSP Tree

- draw the in-plane polygons in back-to-front (or front-to-back) order with respect to viewpoint
- a form of **in-order traversal**, where the order of visiting the subtrees depends on the viewpoint's position with respect to the node's plane

```python
def renderBSP ( Tree, viewpoint ):
    if Tree.isLeaf:
        return
    dist = Tree.planeEquationAt(viewpoint)
    if dist > 0:
        nearSubtree = Tree.rightChild
        farSubtree = Tree.leftChild
    else:
        nearSubtree = Tree.leftChild
        farSubtree = Tree.rightChild
    renderBSP ( farSubtree )
    drawPolygons ( Tree.polys )
    renderBSP ( nearSubtree )
```
BSP Tree Summary

- BSP Trees can classify regions of space (e.g., as solid or empty)
- Speeds up collision tests
- Provide front-to-back or back-to-front orderings
- Are "global" - any change requires rebuilding tree
- Best for static elements
Quadtrees

• each level of the quadtree subdivides space into 4 square regions

• each region is recursively subdivided, as needed

• leaf nodes of quadtree correspond to regions

• terminate subdivision when a region is "simple enough" (eg, has < n objects)
Quadtrees

no. items: 3 1 0 0 3 0 0 0 1 2 2
http://groups.csail.mit.edu/graphics/classes/6.837/F98/talecture/
Octree

- "3-D quadtree"
- each non-leaf node has 8 children

http://groups.csail.mit.edu/graphics/classes/6.837/F98/talecture/
Spatial Subdivision

- regular subdivision
  - squares, cubes
  - may have many empty regions
  - easy to find in which region a point lies
- adaptive subdivision
  - BSP Trees, quadtrees, octrees
  - can be less verbose than regular subdivisions
  - can cost more to search than regular subdivisions
Others

- tries - branching based on representation
- priority queue - find "largest" element in $O(1)$ time
- graphs
  - $G = \{ N, E \}$
  - $N = \text{nodes}$
  - $E = \text{edges (n1, n2)}$