Lecture 17: Binary Trees II

CS 62

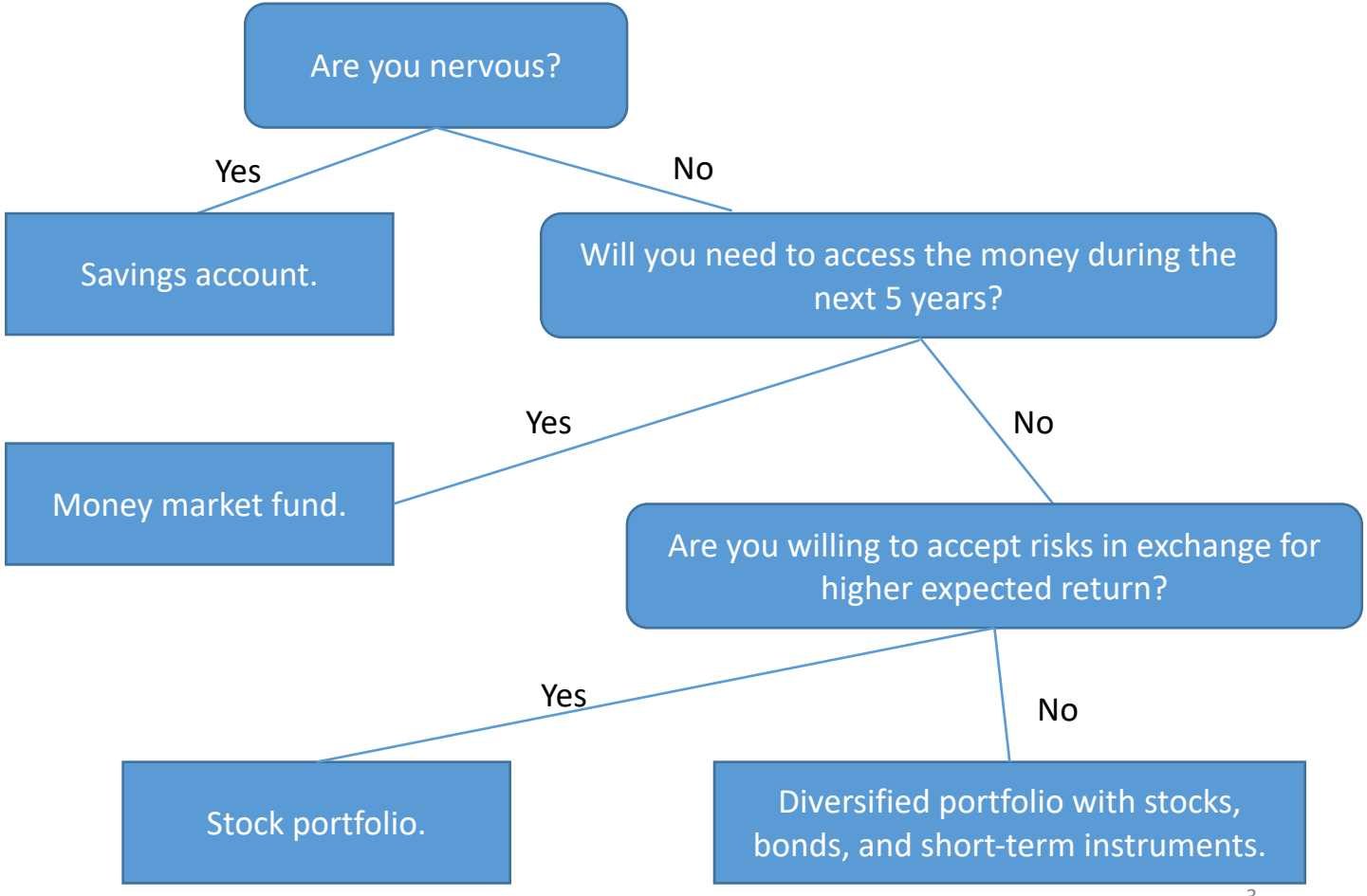
Fall 2018

Alexandra Papoutsaki & William Devanny

Decision Trees

- Binary trees that represent a decision process
  - Internal nodes: Questions with yes/no answers
  - External nodes: Decisions

Decision Tree for Investment Advice



Goodrich & Tamassia (2006)

Animals Game

Guess animal using only yes/no questions.

See demo program.

```
public class BinaryTree<E> {
	protected E val;
	protected BinaryTree<E> parent, left, right;
	public BinaryTree() {//creates an empty tree
		val = null;
		parent = null;
		left = right = this;
	}
	public BinaryTree(E value) {//creates a node with no children
		val = value; //check non-null value
		right = left = new BinaryTree<E>();
		setLeft(left);
		setRight(right);
	}
	public BinaryTree(E value, BinaryTree<E> left, BinaryTree<E> right) {
		//constructs node with two (potentially empty) children
		val = value; //check non-null value
		if (left == null) {
			left = new BinaryTree<E>();
		}
		setLeft(left);
		if (right == null) {
			right = new BinaryTree<E>();
		}
		setRight(right);
		}
	//left(), right(), parent(), getters
```

```
public void setLeft(BinaryTree<E> newLeft) {
	if(isEmpty()) return;
	if(left != null && left.parent() == this) left.setParent(null);
	left = newLeft;
	left.setParent(this);
}
public void setRight(BinaryTree<E> newRight) {
	if(isEmpty()) return;
	if(right != null && right.parent() == this) right.setParent(null);
	right = newRight;
	right.setParent(this);
}
public void setParent(BinaryTree<E> newParent) {
	if(!isEmpty()){
		parent = newParent;
	}
}
public boolean isEmpty(){
	return val == null;
}
public int size(){
	if(isEmpty()) return 0;
	return left().size() + right().size() + 1;
}
```

```
    public int height() {
          if (isEmpty()) return -1;
          return 1 + Math.max(left.height(),right.height());
    }
    public int depth() {
          if (parent() == null) return 0;
          return 1 + parent.depth();
    }

    public BinaryTree<E> root() {
          if (parent() == null) return this;
          else return parent().root();
    }
}
```

Write play(Scanner human, BinaryTree<String> database)

```
public static void main(String args[]) {
    Scanner human = new Scanner(System.in);
    // a simple database -- knows only about a horse being larger than a
    breadbox and an ant being smaller than a breadbox
    BinaryTree<String> database = new BinaryTree<String>(“Is it bigger
    than a breadbox?");
    database.setLeft(new BinaryTree<String>(“horse”));
    database.setRight(new BinaryTree<String>(“ant”));
    System.out.println("Do you want to play a game?");
    while (human.nextLine().equals("yes")) {
        System.out.println("Think of something...I'll guess it");
        play(human, database);
        System.out.println("Do you want to play again?");
    }
    System.out.println("Have a good day!");
}
```

```
public static void play(Scanner human, BinaryTree database) {
        if (!database.left().isEmpty()) {
                // further choices; must ask a question to distinguish them
                System.out.println(database.value());
                if (human.nextLine().equals("yes")) {
                        play(human,database.left());
                }
                else {
                        play(human,database.right());
                }
        } else {
                // must be a statement node
                System.out.println(”Were you thinking of a(n) "+database.value()+"?");
                if (human.nextLine().equals("yes")) {
                        System.out.println(”Ha! I knew I could guess it! What next?"); }
                else {
                        System.out.println("What animal were you thinking of?"); // learn!
                        BinaryTree newObject = new BinaryTree(human.nextLine());
                        BinaryTree oldObject = new BinaryTree(database.value());
                        database.setLeft(newObject);
                        database.setRight(oldObject);
                        System.out.println("What question would distinguish "+
                        newObject.value()+"from "+ oldObject.value()+"?");
                        database.setValue(human.nextLine());
                }
        }
}
```

```
class BTPreorderIterator<E> extends AbstractIterator<E>{
	protected BinaryTree<E> root; // root of tree to be traversed
	protected Stack<BinaryTree<E>> todo; // stack of unvisited nodes
	public BTPreorderIterator(BinaryTree<E> root){
		todo = new StackList<BinaryTree<E>>();
		this.root = root;
		reset();
	}
	public void reset() {
		todo.clear(); // stack is empty; push on root
		if (root != null)
			todo.push(root);
	}
	public boolean hasNext() {
		return !todo.isEmpty();
	}
	public E next(){
		BinaryTree<E> old = todo.pop();
		E result = old.value();
		if (!old.right().isEmpty())
			todo.push(old.right());
		if (!old.left().isEmpty())
			todo.push(old.left());
		return result;
	}
}
```