Shay Walker Project 4: Recursion

Professor Sabal

Box: 613

R-5.8: Describe (not code) a recursive algorithm for converting a string of digits into the integer it represents. For example, '13531' represents the integer 13,531.

I assume that the string length is n and the string is stored in an array a. So, a[n-1] stores the

most significant digit and a[0] stores the least significant digit. The algorithm used would then be stringToInt(inta[],int i). The input would be array a, which is storing the string as well as integer i. The output would be the integer of the string stored in a[n-1],a[n-2],... ,a[n-i]. Therefore, if i=1, I want the return statement to be a[n-1] -’0’; and stringToInt(a,i-1)\*10+a[n-i] - ‘0’. This depending on what is in your array, should return the converted digits into the integer it represents.

//Also sound this example for own personal reference if needed at a later date

//<http://site.iugaza.edu.ps/ehabib/files/Chapter-3.pdf>

int convert(String s){

if(s.length()==1)

return s.charAt(0)-48;

else{

int c = s.charAt(s.length()-1)-48;

return c + 10\*convert(s.substring(0,s.length()-1));

}

//Also this example,<https://webapps.cse.unsw.edu.au/webcms2/course/showfile.php?cid=2407&color=sky&addr=Tutorials/hw3-sol.pdf>

Q2. Describe a recursive algorithm for converting a string of digits into the integer it represents. For

example, "12520" represents the integer 12,520.

Solution: Assume that the string length is n and the string is stored in an array a, where a[n-1] stores the

most significant digit and a[0] stores the least significant digit.

Algorithm stringToInt( int a[], int i)

Input: a: the array storing the string, and integer i.

Output: The integer of the string stored in a[n-1] a[n-2], …, a[n-i].

{

if i=1

return a[n-1] - '0';

return stringToInt(a, i-1)\*10+a[n-i] - '0'

}

P-5.30: Write a program in either Java or Python that can solve instances of the Tower of Hanoi problem (described all over the Internet, and in exercise C-5.16).

Had to look up the following for Java.

* <https://www.cs.utexas.edu/users/ndale/Scanner.html>

The Scanner class is a class in java.util, which allows the user to read values of various types. There are far more methods in class Scanner than you will need in this course. We only cover a small useful subset, ones that allow us to read in numeric values from either the keyboard or file without having to convert them from strings and determine if there are more values to be read.

* I used/edited this one:

import java.util.Scanner;

public class TowersOfHanoi {

//n is the number of discs in the puzzle

//start,auxiliary, end are the names of the three poles which will be used for showing the solution

public void move(int n, String start, String auxiliary, String end) {

//Checks to see if the number of poles is equal to one(if so the base case solution will be used)

//If not the recursive solution is used (two recursive calls)

if (n == 1) {

System.out.println(start + "->"+end);

}else {

//Written this way due to returning two recursions (allows the poles to trade names if needed)

move(n-1, start, end, auxiliary);

//moves the largest disc as the bottom form the start peg to the end peg

System.out.println(start + "->" + end);

//Need to move n-1 discs form the start pole o the auxiliary pole(making it the end pole)

//The auxiliary peg becomes the start peg and the start peg becomes the auxiliary peg

move(n-1, auxiliary, start, end);

}

}

@SuppressWarnings("resource")

public static void main(String[] args) {

TowersOfHanoi towersOfHanoi = new TowersOfHanoi();

//Allows for the number of discs to be changed

System.out.print("Enter number of discs: ");

Scanner scanner = new Scanner(System.in);

int discs = scanner.nextInt();

towersOfHanoi.move(discs, "A", "B", "C");

}

}

* Java Stdin and Stdout I. Most HackerRank challenges require you to read input from stdin (standard input) and write output to stdout (standard output). In this challenge, you must read integers from stdin and then print them to stdout. Each integer must be printed on a new line <https://www.hackerrank.com/challenges/java-stdin-and-stdout-1/problem>
* <http://simpledeveloper.com/towers-of-hanoi/> (move method)

move( n – 1, origin, temp, dest)

“Move disk ” + n + ” from ” + origin + ” to ” + dest + “\n”

move( n – 1, temp, dest, origin)

/\*\*

\* Determines the steps needed to move disks from an origin

\* to a destination. The worstTme(n) is 0(2 raised to n),

\* where n is the number of disks to be moved

\*

\* @param n the number of disks to be moved

\* @param orig the pole where the disks are originally

\* @param dest the destination pole

\* @param temp the pole used for temporary storage

\*

\* @return a String representation of the moves needed

\*

\* @throws IllegalArgumentException if n is < = 0 \*/ public static String move(int n, char orig, char dest, char temp){ final String DIRECT\_MOVE = "Move disk " + n + " from " + orig + " to " + dest + "\n"; if( n <= 0 ){ throw new IllegalArgumentException(); } if( n == 1 ){ return DIRECT\_MOVE; } String result = move( n - 1, orig, temp, dest ); result += DIRECT\_MOVE; result += move( n - 1, temp, dest, origin ); return result; } //end method move [/java]

* https://introcs.cs.princeton.edu/java/23recursion/TowersOfHanoi.java.html

//https://introcs.cs.princeton.edu/java/23recursion/TowersOfHanoi.java.html

//Solves the Towers of Hanoi problem on n discs. THe discs are labeled in increasing order of size from 1 to n.

public class TowersOfHanoi {

// the left (if left is true) or right (if left is false)

public static void moves(int n, boolean left) {

if (n == 0)return;

moves(n-1, !left);

if(left) StdOut.println(n + "left");

else StdOut.println(n + "right");

moves(n-1, !left);

}

public static void main(String[] args) {

int n = Integer.parseInt(arg[0]);

moves(n, true);

}

}

For Python:

* http://interactivepython.org/runestone/static/pythonds/Recursion/TowerofHanoi.html

def moveTower(height,fromPole, toPole, withPole):

if height >= 1:

moveTower(height-1,fromPole,withPole,toPole)

moveDisk(fromPole,toPole)

moveTower(height-1,withPole,toPole,fromPole)

def moveDisk(fp,tp):

print("moving disk from",fp,"to",tp)

moveTower(3,"A","B","C")

* https://www.python-course.eu/towers\_of\_hanoi.php

def hanoi(n, source, helper, target):  
 if n > 0:  
 # move tower of size n - 1 to helper:  
 hanoi(n - 1, source, target, helper)  
 # move disk from source peg to target peg  
 if source:  
 target.append(source.pop())  
 # move tower of size n-1 from helper to target  
 hanoi(n - 1, helper, source, target)  
   
source = [4,3,2,1]  
target = []  
helper = []  
hanoi(len(source),source,helper,target)  
  
print source, helper, target

This function is implementing, what we have explained in the previous subchapter. First we move a tower of size n-1 from the peg source to the helper peg. We do this by calling

hanoi(n - 1, source, target, helper)

After this, there will be the largest disk left on the peg source. We move it to the empty peg target by the statement

if source:  
 target.append(source.pop())

After this, we have to move the tower from "helper" to "target", i.e. on top of the largest disk:

hanoi(n - 1, helper, source, target)

If you want to check, what's going on, while the recursion is running, we suggest the following Python program. We have slightly changed the data structure. Instead of passing just the stacks of disks to the function, we pass tuples to the function. Each tuple consists of the stack and the function of the stack:

def hanoi(n, source, helper, target):  
 print "hanoi( ", n, source, helper, target, " called"  
 if n > 0:  
 # move tower of size n - 1 to helper:  
 hanoi(n - 1, source, target, helper)  
 # move disk from source peg to target peg  
 if source[0]:  
 disk = source[0].pop()  
 print "moving " + str(disk) + " from " + source[1] + " to " + target[1]  
 target[0].append(disk)  
 # move tower of size n-1 from helper to target  
 hanoi(n - 1, helper, source, target)  
   
source = ([4,3,2,1], "source")  
target = ([], "target")  
helper = ([], "helper")  
hanoi(len(source[0]),source,helper,target)  
  
print source, helper, target

* <https://www.geeksforgeeks.org/iterative-tower-of-hanoi/>

// Java program for iterative

// Tower of Hanoi

public class TOH

{

// A structure to represent a stack

class Stack

{

int capacity;

int top;

int array[];

}

// function to create a stack of given capacity.

Stack createStack(int capacity)

{

Stack stack=new Stack();

stack.capacity = capacity;

stack.top = -1;

stack.array = new int[capacity];

return stack;

}

// Stack is full when top is equal to the last index

boolean isFull(Stack stack)

{

return (stack.top == stack.capacity - 1);

}

// Stack is empty when top is equal to -1

boolean isEmpty(Stack stack)

{

return (stack.top == -1);

}

// Function to add an item to stack. It increases

// top by 1

void push(Stack stack,int item)

{

if(isFull(stack))

return;

stack.array[++stack.top] = item;

}

// Function to remove an item from stack. It

// decreases top by 1

int pop(Stack stack)

{

if(isEmpty(stack))

return Integer.MIN\_VALUE;

return stack.array[stack.top--];

}

// Function to implement legal movement between

// two poles

void moveDisksBetweenTwoPoles(Stack src, Stack dest,

char s, char d)

{

int pole1TopDisk = pop(src);

int pole2TopDisk = pop(dest);

// When pole 1 is empty

if (pole1TopDisk == Integer.MIN\_VALUE)

{

push(src, pole2TopDisk);

moveDisk(d, s, pole2TopDisk);

}

// When pole2 pole is empty

else if (pole2TopDisk == Integer.MIN\_VALUE)

{

push(dest, pole1TopDisk);

moveDisk(s, d, pole1TopDisk);

}

// When top disk of pole1 > top disk of pole2

else if (pole1TopDisk > pole2TopDisk)

{

push(src, pole1TopDisk);

push(src, pole2TopDisk);

moveDisk(d, s, pole2TopDisk);

}

// When top disk of pole1 < top disk of pole2

else

{

push(dest, pole2TopDisk);

push(dest, pole1TopDisk);

moveDisk(s, d, pole1TopDisk);

}

}

// Function to show the movement of disks

void moveDisk(char fromPeg, char toPeg, int disk)

{

System.out.println("Move the disk "+disk +

" from "+fromPeg+" to "+toPeg);

}

// Function to implement TOH puzzle

void tohIterative(int num\_of\_disks, Stack

src, Stack aux, Stack dest)

{

int i, total\_num\_of\_moves;

char s = 'S', d = 'D', a = 'A';

// If number of disks is even, then interchange

// destination pole and auxiliary pole

if (num\_of\_disks % 2 == 0)

{

char temp = d;

d = a;

a = temp;

}

total\_num\_of\_moves = (int) (Math.pow(2, num\_of\_disks) - 1);

// Larger disks will be pushed first

for (i = num\_of\_disks; i >= 1; i--)

push(src, i);

for (i = 1; i <= total\_num\_of\_moves; i++)

{

if (i % 3 == 1)

moveDisksBetweenTwoPoles(src, dest, s, d);

else if (i % 3 == 2)

moveDisksBetweenTwoPoles(src, aux, s, a);

else if (i % 3 == 0)

moveDisksBetweenTwoPoles(aux, dest, a, d);

}

}

// Driver Program to test above functions

public static void main(String[] args)

{

// Input: number of disks

int num\_of\_disks = 3;

TOH ob = new TOH();

Stack src, dest, aux;

// Create three stacks of size 'num\_of\_disks'

// to hold the disks

src = ob.createStack(num\_of\_disks);

dest = ob.createStack(num\_of\_disks);

aux = ob.createStack(num\_of\_disks);

ob.tohIterative(num\_of\_disks, src, aux, dest);

}

}

// This code is Contibuted by Sumit Ghosh