Recursion[*]

Nabil M. Al-Rousan, nabil@ece.ubc.ca

February 19, 2020

Objectives

- be able to decompose a recursive problem into recursive steps and base cases
- understand the advantages and disadvantages of recursion vs. iteration

Definition

A recursive function is defined in terms of base cases and recursive steps.

- In a base case, we compute the result immediately given the inputs to the function call.
- In a recursive step, we compute the result with the help of one or more recursive calls to this same function, but with the inputs somehow reduced in size or complexity, closer to a base case.

Examples

Consider writing a function to compute factorial. We can define factorial in two different ways:

$$n! = n \times (n-1) \times ... \times 2 \times 1 = \prod_{k=1}^{n} k$$

or can be written as recurrence relation:

$$n! = \begin{cases} 1 & \text{if (n = 0)} \\ (n-1)! \times n & \text{if } (n > 0) \end{cases}$$

which leads to two different implementations:

Listing 1: Factorial Recursive Implementation

```java
public static long factorial(int n) {
  if (n == 0) {
    return 1;
  } else {
    return n * factorial(n-1);
  }
}
```

Listing 2: Factorial Iterative Implementation

```java
public static long factorial(int n) {
  long fact = 1;
  for (int i = 1; i ≤ n; i++) {
    fact = fact * i;
  }
  return fact;
}
```

In the recursive implementation:

- the base case is $n = 0$, where we compute and return the result immediately: 0! is defined to be 1.
- The recursive step is $n > 0$, where we compute the result with the help of a recursive call to obtain $(n-1)!$, then complete the computation by multiplying by n.

[*]MIT 6.005 Software Construction Course

Another common example is the Fibonacci series: $0, 1, 1, 2, 3, 5, 8, ..$

Listing 3: Fibonacci Recursive Implementation

```
/**
 * @param n ≥ 0
 * @return the nth Fibonacci number
 */
public static int fibonacci(int n) {
    if (n == 0 || n == 1) {
        return n; // base cases
    } else {
        return fibonacci(n−1) + ...
            ↪ fibonacci(n−2); // ...
            ↪ recursive step
    }
}
```

Listing 4: Fibonacci Iterative Implementation

```
/**
 * @param n ≥ 0
 * @return the nth Fibonacci number
 */
public static int fibonacci(int n) {
    /* Declare an array to store ...
        ↪ Fibonacci numbers. */
    int f[] = new int[n+1]; // 1 extra ...
        ↪ to handle case, n = 0
    int i;
    /* 0th and 1st number of the series ...
        ↪ are 0 and 1*/
    f[0] = 0;
    f[1] = 1;
    for (i = 2; i ≤ n; i++)
    {
       /* Add the previous 2 numbers in ...
           ↪ the series
         and store it */
        f[i] = f[i−1] + f[i−2];
    }

    return f[n];
    }
}
```

Fibonacci is interesting because it has multiple base cases: n=0 and n=1. What happens to the function stack when the series get longer?

When to Use Recursion Rather Than Iteration

- The problem is naturally recursive (e.g. Fibonacci)
- The data is naturally recursive (e.g. filesystem)
- Take more advantage of immutability

Common Mistakes in Recursive Implementations[1]

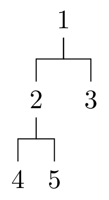
- The base case is missing entirely, or the problem needs more than one base case but not all the base cases are covered.
- The recursive step doesn't reduce to a smaller subproblem, so the recursion doesn't converge.

[1]http://blog.moertel.com/posts/2013-05-11-recursive-to-iterative.html

Application (Tree Traversal) [2]

Depth First Traversals:
(a) Inorder (Left, Root, Right) : 4 2 5 1 3
(b) Preorder (Root, Left, Right) : 1 2 4 5 3
(c) Postorder (Left, Right, Root) : 4 5 2 3 1

Breadth First or Level Order Traversal : 1 2 3 4 5



```java
// Java program for different tree traversals

/* Class containing left and right child of current
node and key value*/
class Node
{
    int key;
    Node left, right;

    public Node(int item)
    {
        key = item;
        left = right = null;
    }
}

class BinaryTree
{
    // Root of Binary Tree
    Node root;

    BinaryTree()
    {
        root = null;
    }

    /* Given a binary tree, print its nodes according to the
    "bottom-up" postorder traversal. */
    void printPostorder(Node node)
    {
        if (node == null)
            return;

        // first recur on left subtree
        printPostorder(node.left);

        // then recur on right subtree
        printPostorder(node.right);

        // now deal with the node
        System.out.print(node.key + " ");
    }

    /* Given a binary tree, print its nodes in inorder*/
    void printInorder(Node node)
    {
        if (node == null)
            return;

        /* first recur on left child */
        printInorder(node.left);

        /* then print the data of node */
        System.out.print(node.key + " ");

        /* now recur on right child */
        printInorder(node.right);
    }

    /* Given a binary tree, print its nodes in preorder*/
    void printPreorder(Node node)
    {
        if (node == null)
            return;

        /* first print data of node */
        System.out.print(node.key + " ");

        /* then recur on left sutree */
        printPreorder(node.left);

        /* now recur on right subtree */
        printPreorder(node.right);
    }

    // Wrappers over above recursive functions
    void printPostorder() {  printPostorder(root); }
    void printInorder() {    printInorder(root); }
    void printPreorder() {   printPreorder(root); }

    // Driver method
    public static void main(String[] args)
    {
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(5);

        System.out.println("Preorder traversal of binary tree is ");
        tree.printPreorder();

        System.out.println("\nInorder traversal of binary tree is ");
        tree.printInorder();

        System.out.println("\nPostorder traversal of binary tree is ");
        tree.printPostorder();
    }
}
```

Listing 5: Tree traversal - Recursive Implementation

[2]https://www.geeksforgeeks.org/tree-traversals-inorder-preorder-and-postorder/