2 - Recursion

Joseph Afework
CS 241

Agenda

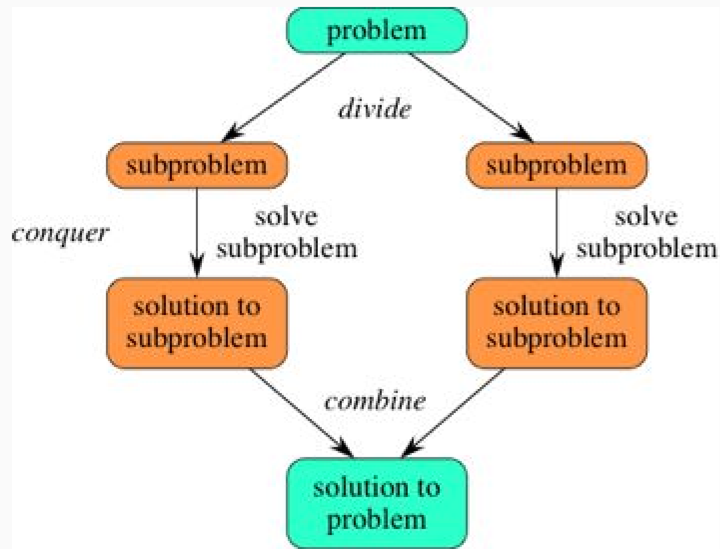
- Intro
- Recursion
- Examples
- Types of Recursion
- Recursion vs. Iteration

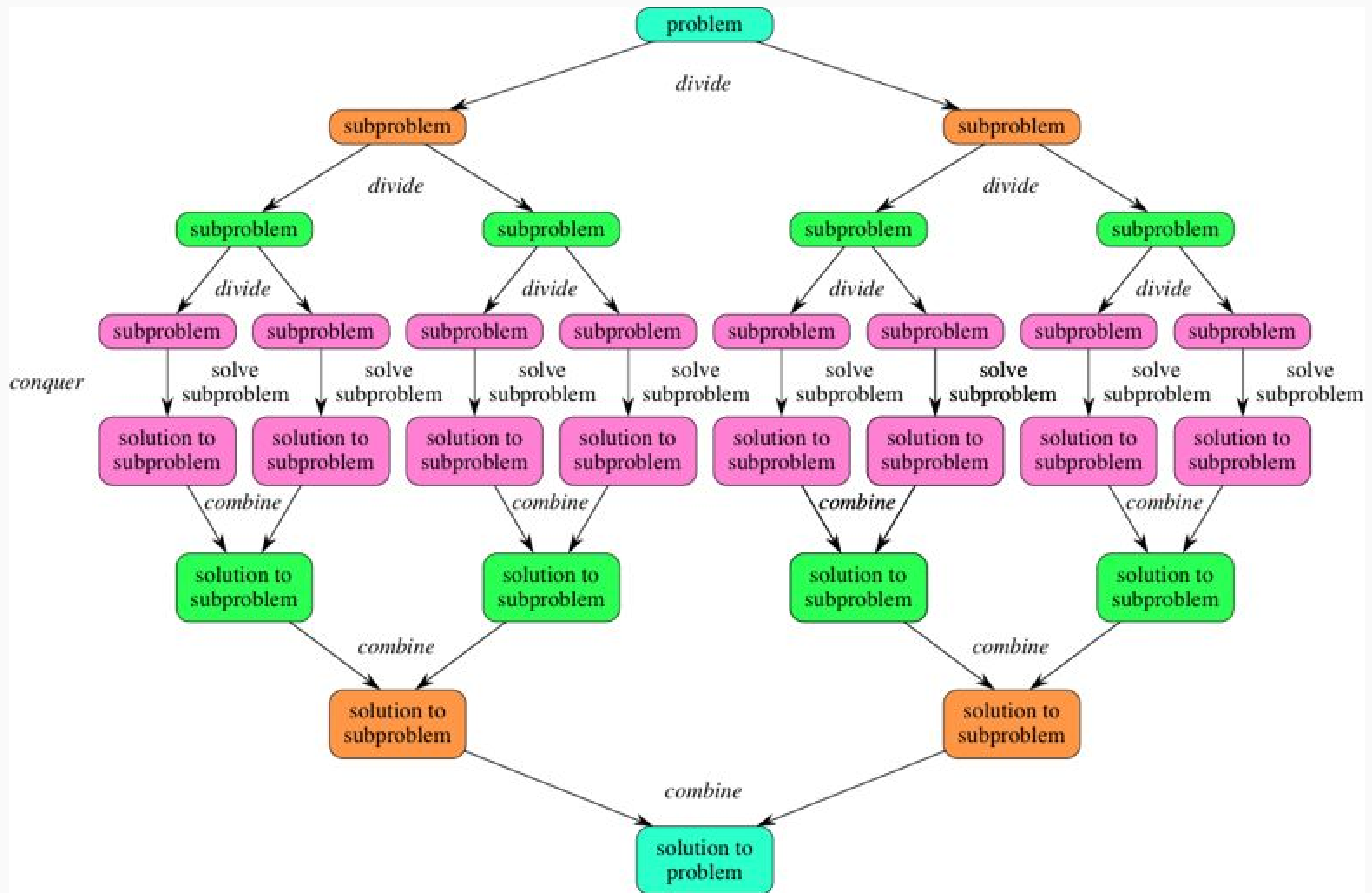
Reading Assignment

- Read Chapter 7 - Recursion
  - Chapter 7 All Sections
  - Read Chapter Summary Section (end of chapter)
  - Read Programming Tips Section (end of chapter)

Intro

- Imagine you start with a problem:
  - Problem is too big to solve
  - Problem is easier to understand in smaller pieces
  - Solution is known for a small iteration of the problem


problem
divide
subproblem
subproblem
conquer
solve
subproblem
solve
subproblem
solution to
subproblem
solution to
subproblem
combine
solution to
problem


problem
divide
subproblem
subproblem
divide
divide
subproblem
subproblem
subproblem
subproblem
divide
divide
divide
divide
subproblem
subproblem
subproblem
subproblem
subproblem
subproblem
subproblem
subproblem
conquer
solve subproblem
solve subproblem
solve subproblem
solve subproblem
solve subproblem
solve subproblem
solve subproblem
solve subproblem
solution to subproblem
solution to subproblem
solution to subproblem
solution to subproblem
solution to subproblem
solution to subproblem
solution to subproblem
solution to subproblem
combine
combine
combine
combine
solution to subproblem
solution to subproblem
solution to subproblem
solution to subproblem
combine
combine
solution to subproblem
solution to subproblem
combine
solution to problem

Recursion

- **Recursion** is a technique that can be used to break a larger problem or task into identical yet smaller sub pieces.

- A ***recursive method*** is a method that calls itself

Ex. Recursion

```
Public void countDown(int integer)
{
	if(integer >= 1)
	{
		System.out.println(integer);
		countDown(integer-1);
	}
}
```

Designing a recursive function

1. What part of the solution do you know?

2. What solution do you know for a sub problem of the whole problem?

3. **When does the recursion end?**

ICE 2.1

- What part of the solution do you know?
- What solution do you know for a sub problem of the whole problem?
- When does the recursion end?

```
Public void countDown(int integer)
{
	if(integer >= 1)
	{
		System.out.println(integer);
		countDown(integer-1);
	}
}
```

Base Case

A **base case** (stopping case) is the termination case for ending a recursive function.

WARNING: No base case = infinite execution = infinite recursion

```
Public void countDown(int integer)
{
	if(integer >= 1)
	{
		System.out.println(integer);
		countDown(integer-1);
	}
}
```

Limits to Recursion

- In Java there is a call stack that keep tracks of return addresses + metadata for each function call.
- Each new function call gets pushed on the stack, each return statement will pop data of the stack.
- The more recursive a function, the quicker the call stack fills up.
- Once you smash the stack (no more room) - **STACK OVERFLOW Exception or JVM memory exception**

ICE 2.2 Linked List

Given a Linked List, print the nodes of the linked list in reverse order:

```
public class Node
{
      Node next;
      String data;
}

Tail:
node.next == null

… Node list = … // head node
```

ICE 2.2 Solution

You probably...

- Iterated through the list pushing data into a stack
- popped all the values off the stack and printed the nodes

Is there a better way?

ICE 2.2 Recursive Solution

// Let’s Start With A recursive function

```
Private void displayData(Node node)
{
     displayData(null);
}
```

ICE 2.2 Recursive Solution

// Let’s Loop over the nodes

```
Private void displayData(Node node)
{
     displayData(node.next);
}
```

ICE 2.2 Recursive Solution

// Let’s Add A Base Case

```
Private void displayData(Node node)
{
      if(node != null)
      {
            displayData(node.next);
      }
}
```

ICE 2.2 Recursive Solution

// Let’s log the node

```
Private void displayData(Node node)
{
	if(node != null)
	{
		displayData(node.next);
		System.out.println(node.data);
	}
}
```

Tail Recursion

- Last operation in a method is a recursive call
- Replaceable with iteration (loop structure)

```
Public void countDown(int integer)
{
    if(integer >= 1)
    {
        System.out.println(integer);
        countDown(integer-1);
    }
}
```

Note

- Some programming languages/compilers may change code that is using tail recursion to iteration.
- Recursion usually incurs a higher memory cost.
- Compiler optimization
- ***It may be possible for an optimized method to never terminate… Don’t write code trying to catch stack overflows**

ICE 2.3 Tail Recursion

Change the following recursive function to use iteration (loop) instead:

```
Public void countDown(int integer)
{
     if(integer >= 1)
     {
          System.out.println(integer);
          countDown(integer-1);
     }
}
```

Indirect Recursion

- Indirect recursive calls
  - Ex. A calls B….. Then B calls C…. Then C calls A
- Can be difficult to trace

```
static bool IsOdd(int number) {
    if(number == 0)
    {
        return false;
    }
    else
        return IsEven(Math.Abs(number) - 1);
}
```

```
static bool IsEven(int number) {
    if(number == 0)
    {
        return true;
    }
    else
        return IsOdd(Math.Abs(number) - 1);
}
```

Mutual Recursion

- Special case of Indirect Recursion
- Only Two functions that have mutual dependency on each other.

```
static bool IsOdd(int number) {
      if(number == 0)
      {
            return false;
      }
      else
            return IsEven(Math.Abs(number) - 1);
}
```

```
static bool IsEven(int number) {
      if(number == 0)
      {
            return false;
      }
      else
            return IsOdd(Math.Abs(number) - 1);
}
```

HW 1

Instructions: Write an infinitely recursive function in Java.

1. Find out how many recursive calls you can make on your java environment before your program crashes.
2. Does the number change when you use tail recursion?
3. The Java compiler has optimizations enabled by default, that can affect the results you found above. Try running/compiling the above scenarios with compiler optimizations disabled. What are the results? Did they change?

Learning Outcomes

- Proficiency in recursion

References

https://www.safaribooksonline.com/library/view/functional-programming-in/9780470971109/xhtml/sec55.html

http://introcs.cs.princeton.edu/java/23recursion/