# Lab 9: Memoization

## CMPT 145

# Laboratory 9 Overview

# Section 1

# Laboratory Activities

# Fibonacci numbers

The Fibonacci numbers can be computed by a recursive function:

```
1  def fibonacci(n):
2      if n==0:
3          return 0
4      elif n==1:
5          return 1
6      else:
7          f1 = fibonacci(n-1)
8          f2 = fibonacci(n-2)
9          return f1 + f2
```

(The reason for separating the recursive case into steps will be clear later)

# Fibonacci, again

- This function is very simple, and obviously correct, but inefficient!
  - The inefficiency comes from having 2 recursive calls in the recursive case.
- In this lab, we'll explore just how bad it is.
- We'll also see a technique to make it more efficient.
- This technique can be applied to make many recursive functions more efficient.

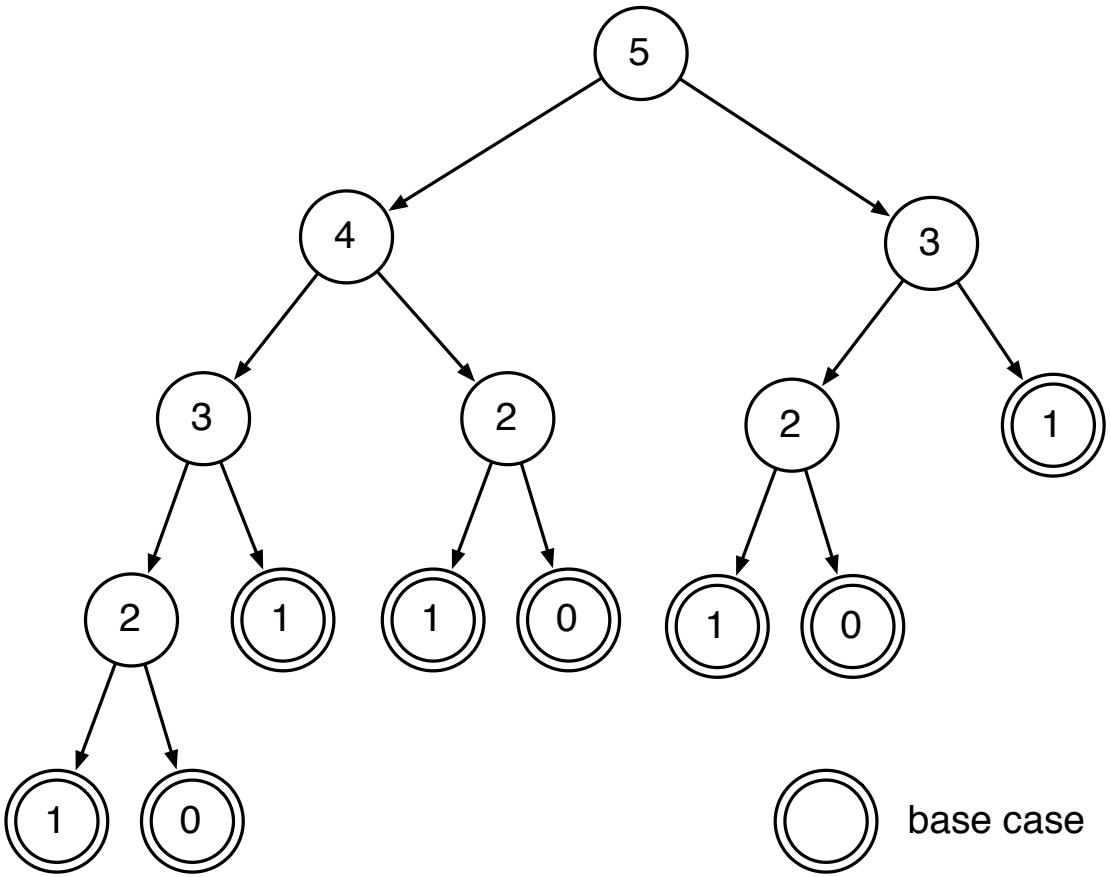
# Definition: Fibonacci trees

A Fibonacci tree for $n$ is a tree defined recursively as follows:

- A Fibonacci tree for 0 is a leaf whose data value is $0$.
- A Fibonacci tree for 1 is a leaf whose data value is $1$.
- A Fibonacci tree for $n > 1$ has a root whose data value is $n$, with:
  - A Fibonacci tree for $n - 2$ as its left subtree
  - A Fibonacci tree for $n - 1$ as its right subtree.

Draw a few Fibonacci trees ($n = 5$ is useful) to be sure you understand the definition. See next slide for an example.

# A Fibonacci tree for $n = 5$



# Activity: Counting nodes

- In the Fibonacci tree for $n = 5$, how many times do you see:
  - The value 2?
  - The value 3?
- In the Fibonacci tree for $n = 7$, how many times do you see:
  - The value 3?
  - The value 4?
- Put your answers to these questions in your `lab9-responses.txt` file.

# Fibonacci trees represent computation

- The Fibonacci tree for $n$ represents the computation of Fibonacci numbers by the simple (but inefficient) recursive function given on Slide 4.
- Each leaf node represents one of the base cases.
- All other nodes in the Fibonacci tree represent the recursive calls.
- The repeated values (non-leaf) in the Fibonacci tree represent repeated computations.
- The size of of the tree represents its run-time costs.

# Space-time trade-off

- Repeatedly calculating the same data over and over makes this recursive Fibonacci function very inefficient.
- Principle: Never repeat a calculation except if it is very simple.
- If you need the value of a calculation in more than one expression, calculate it once, and store it somewhere for re-use.
- By using more memory, we can usually save lots of time.
- This principle is known as a space-time trade-off.

# Memoization

- Not a typo! Memoization means to add a memo to a function that makes an inefficient recursion much more efficient.
- A general technique you can apply to many problems.
- A memo is a place to store data values when they've been computed once already.
- In Python, a dictionary is a perfect tool to create a memo.

# Memoization Strategy

- The strategy is as follows:
  1. Start with a recursive function that works.
  2. Add a new base case to the normal recursive function.
  3. The new base case checks the memo to see if the result of the requested calculation is already known.
  4. If so, use the already calculated value.
  5. If not, calculate the value recursively, as normal.
  6. After calculating the value with the recursive call, save the value in the memo.

# Fibonacci, memoized

```
1  def fibm(n):
2      memo = {}
3      def memoized_fib(n):
4          if n==0:
5              return 0
6          elif n==1:
7              return 1
8          elif n in memo:
9              return memo[n]
10         else:
11             f1 = memoized_fib(n-1)
12             f2 = memoized_fib(n-2)
13             memo[n] = f1+f2
14             return f1+f2
15
16     return memoized_fib(n)
```

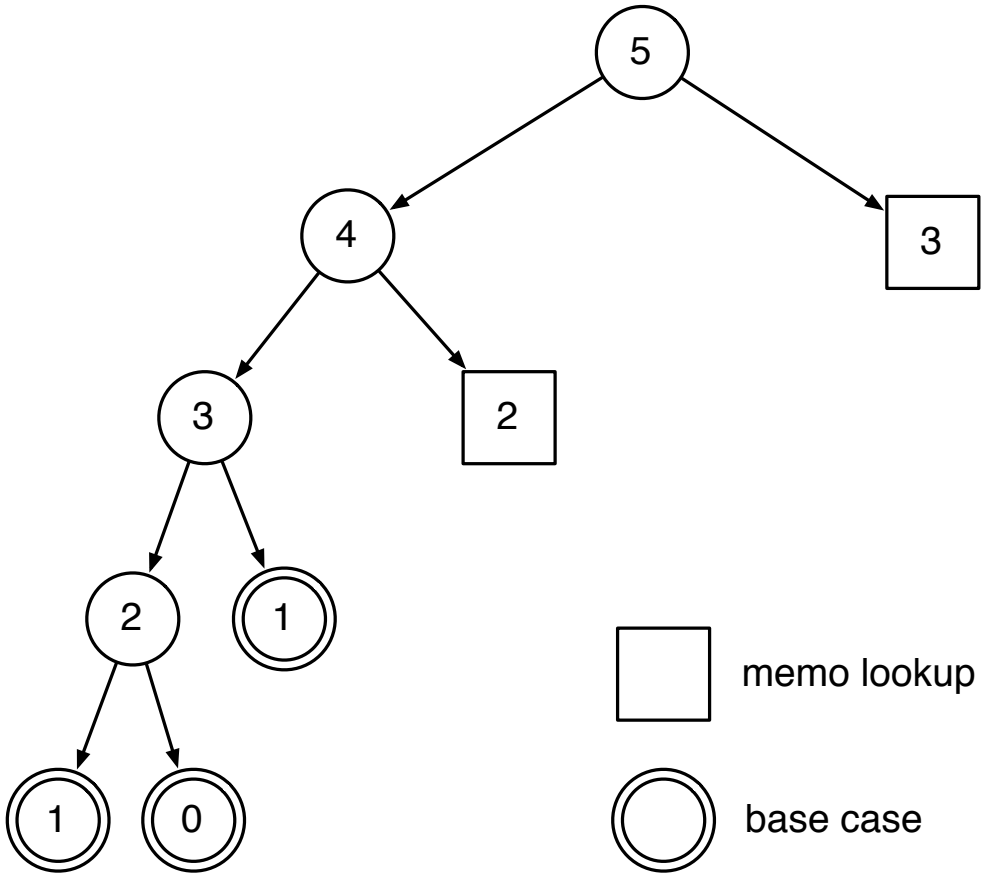
# Notes on the code

- The internal function (lines 3-14) does all the work. It resembles the original function most closely.
- The external function defines the dictionary to be used as a memo.
- The dictionary, `memo` is defined inside `fibm` (line 2), but is accessible by the internal function.
- A new base case is added, checking if the memo has a value for $n$ (lines 8-9).
- Saving the calculated value (line 13) before returning is key!

# Showing memoization in a diagram

- We can draw Fibonacci trees using 2 kinds of nodes:
  1. When `memoized_fib()` is called, a circle.
  2. When a value is retrieved from `memo`, a square.
- See next slide for an example!

# A Fibonacci tree showing memoization for $n = 5$



# Activity: Counting nodes, again

Consider the diagram on Slide 16:

- How many times do you see:
  - The value 2 circled? Squared?
  - The value 3 circled? Squared?
- Draw a Fibonacci tree for $n = 7$ showing memoization when it occurs. How many times do you see:
  - The value 3 circled? Squared?
  - The value 4 circled? Squared?
  - How many nodes in total, both circled and squared?
- Compare these answers to the ones from Slide 8.
- Put your answers to these questions in your `lab9-responses.txt` file.

# Activity: Fibonacci vs fibm

- Find a value for $n$ that causes the original function (Slide 4) to run for a few seconds.
    1. Something in the range 30-50, depending on your computer.
- Run the memoized version (Slide 13) on that value of $n$.
- Run the memoized version on larger $n$.
- For your choice of $n$, how much time does:
    1. The original function take?
    2. The memoized function take?
- Put your answers to these questions in your `lab9-responses.txt` file.

# Activity: Moosonacci numbers

- Recall the Moosonacci problem, from Assignment 7.
- Like Fibonacci, it's very inefficient.
- Apply the memoization technique to the function `moosonacci`.
- Put your implementation in `lab9.py`.
- Compare the two versions by timing them.
- Put your comparison in your `lab9-responses.txt` file.

# ACTIVITY: Counting nodes, one last time

Consider the diagram on Slide 16:

- The `fibm()` function uses a dictionary.
- For the Fibonacci tree for $n = 7$, how many times does `fibm()` obtain a value from the dictionary?
- If you study the diagram carefully, you will realize that you only need to store two pieces of data as a memo.
  - But they change as Fibonacci is calculated.
- If you are feeling like a challenge: Rewrite `fibm()` so that it uses only $O(1)$ space!

# Section 2

## Hand In

# What To Hand In

Hand in your `lab9-responses.txt` file showing:

- Activity on Slide 8
- Activity on Slide 17
- Activity on Slide 18

Also hand in your `lab9.py` implementation of memoization for `moosonacci`.