Welcome to CS 225

Data Structures

Learning Objectives

- Conceptualize the use of dictionaries for memoization
- Practice working with dictionaries

Memoization

In computing, memoization is an optimization technique used primarily to speed up computer programs by storing the results of expensive function calls and returning the cached result when the same inputs occur again.

-- wikipedia

Factorial is a deterministic function.

The formal (recursive) definition is:

$$Fac(0) = 0! = 1$$

$$Fac(n) = n! = n * (n - 1)!$$

Let’s compute 6! and 7!:

$$6! = 6 \times (5 \times (4 \times (3 \times (2 \times (1))))) = 720$$

$$7! = 7 \times 6 \times (5 \times (4 \times (3 \times (2 \times (1))))) = 5040$$

Let’s compute 6! and 7!:

subproblem

$$6! = 6 \times (5 \times (4 \times (3 \times (2 \times (1))))) = 720$$

$$7! = 7 \times 6 \times (5 \times (4 \times (3 \times (2 \times (1))))) = 5040$$

or $7! = 7 \times 6! = 7 \times 720 = 5040$

$$F(0)=0$$
$$F(1)=1$$

$$F(n)=F(n-1)+F(n-2)$$

Task #1 - Fibonacci

Implement both the normal and memoized version of the fib function in fib.cpp.
After you do this you can race them with the fib_generator executable:

```
make fib_generator
 ./fib_generator 45
```

To use the memoized version, pass the -m flag:

```
 ./fib_generator 45 -m
```

Recursive Fibonacci

Base Cases:

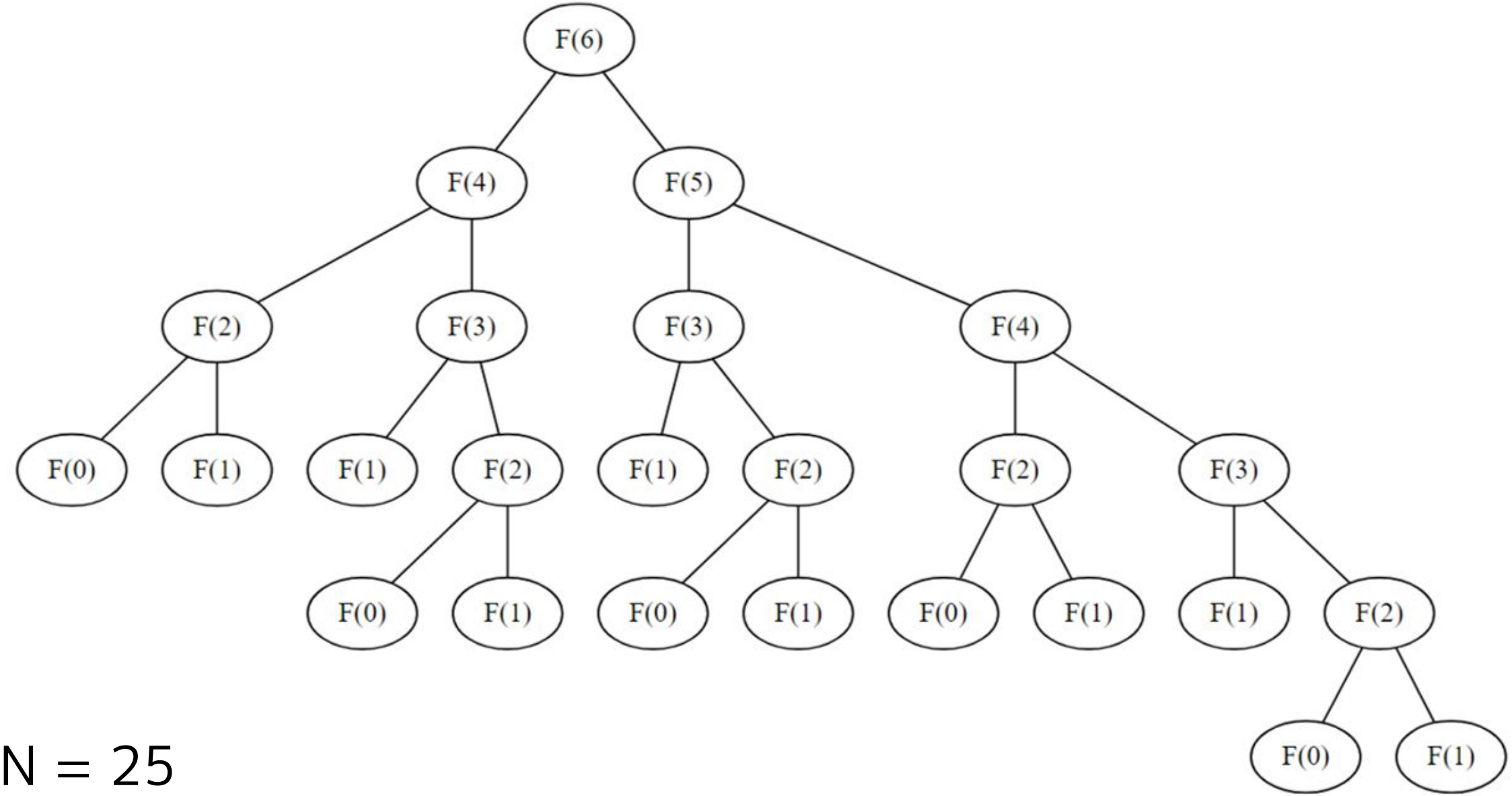
- F(0) = 0
- F(1) = 1

```
1 int fibonacci(int n) :=
2     if (n == 0) return 0
3     if (n == 1) return 1
4     return fibonacci(n - 1) + fibonacci(n - 2)
```

Recursive Case:

- F(n) = F(n - 1) + F(n - 2)

Computation Tree for F(6)



N = 25

Dictionary Fibonacci

Store:

- dictionary (n -> F(n))

Base Cases:

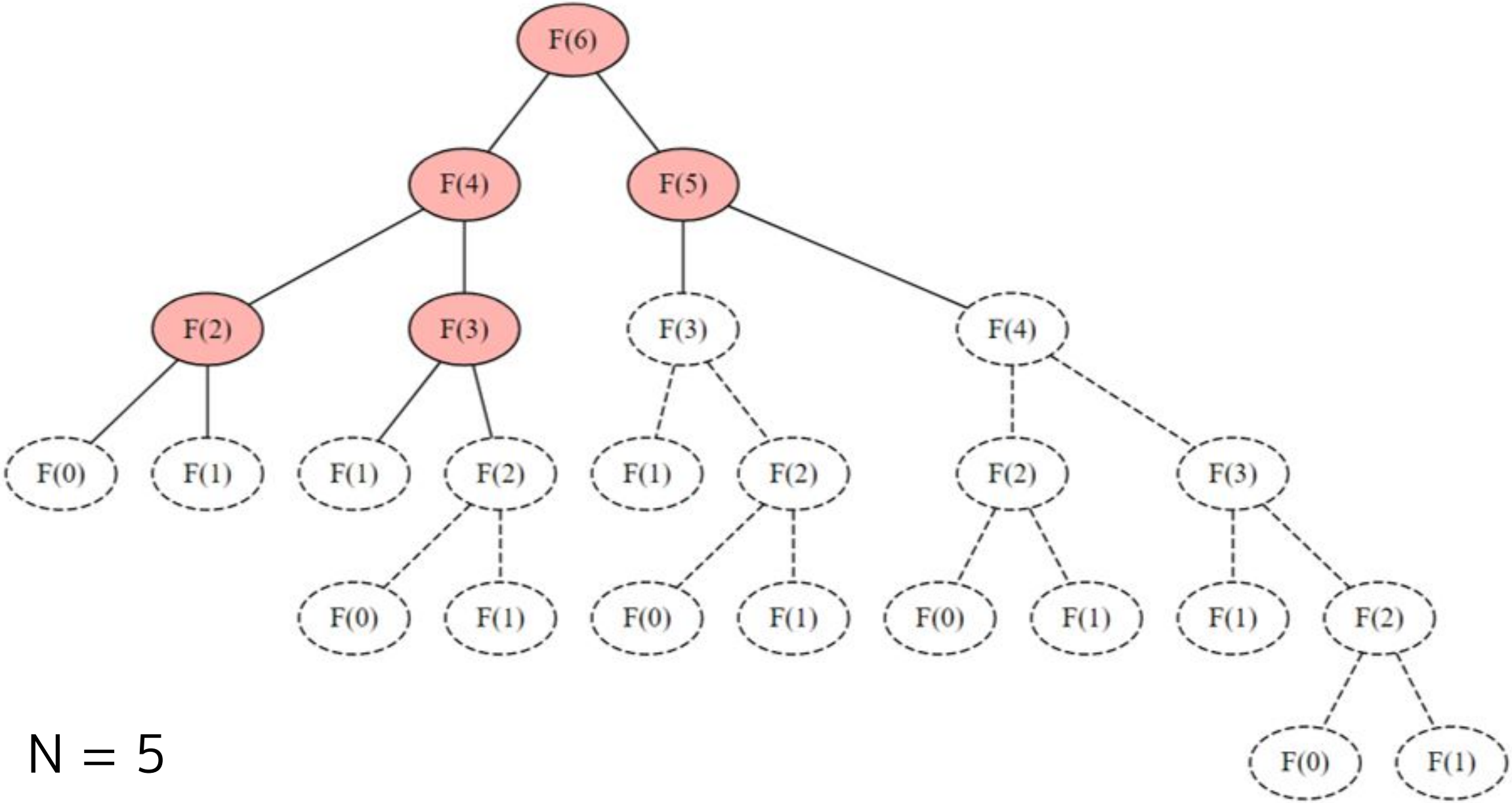
- F(0) = 0
- F(1) = 1
- F(n), if F(n) in dictionary

Recursive Case:

- F(n) = F(n - 1) + F(n - 2)

```
1  int fibonacci(int n) :=
2      if (n == 0) return 0
3      if (n == 1) return 1
4      if (n in memoized.keys()) :=
5          return memoized[n]
6      int ret = fibonacci(n - 1) + fibonacci(n - 2)
7      memoized[n] = ret
8      return ret
```

The tree after memoization



N = 5

https://en.wikipedia.org/wiki/Associative_array

Dictionary (map) Operations

1. **Add** a new (key, value) pair to the collection
2. **Remove** a pair from the collection using its key
3. **Modify** the value associated with an existing key
4. **Lookup** the value given a specific key

Are key and value unique?

We'll be using dictionary types that have already been defined for us.

In C++ there are two "built-in" dictionary types: **std::map** and **std::unordered_map.**

Both exist in the Standard Template Library (STL).

	Lookup		Insert		Delete		Sorted?
	Average	Worst Case	Average	Worst Case	Average	Worst Case	
Map (BBST)							Yes
Unordered_map (Hash)							No

	Lookup		Insert		Delete		Sorted?
	Average	Worst Case	Average	Worst Case	Average	Worst Case	
Map (BBST)	O(log n)	O(log n)	O(log n)	O(log n)	O(log n)	O(log n)	Yes
Unordered_map (Hash)	O(1)	O(n)	O(1)	O(n)	O(1)	O(n)	No

Helpful Coding Tips

[] operator

a. accesses an element in a map
b. also adds an element to the map if it doesn’t exist
c. therefore, it won’t work with a const map

Helpful Coding Tips

```
unsigned long memoized_fac(unsigned long n)
{
    /* Fancy initialization of the static map with an initializer list Maps 0->1 */
    static map<unsigned long, unsigned long> memo = {
        {0, 1},
    };

    map<unsigned long, unsigned long>::iterator lookup = memo.find(n);
    if (lookup != memo.end()) {
        return lookup->second;
    } else {
        unsigned long result = n * memoized_fac(n - 1);
        memo[n] = result;
        return result;
    }
}
```

Helpful Coding Tips

```
unsigned long memoized_fac(unsigned long n)
{
    /* Fancy initialization of the static map with an initializer list Maps 0->1 */
    static map<unsigned long, unsigned long> memo = {
        {0, 1},
    };
    map<unsigned long, unsigned long>::iterator lookup = memo.find(n);
    if (lookup != memo.end()) {
        return lookup->second;
    } else {
        unsigned long result = n * memoized_fac(n - 1);
        memo[n] = result;
        return result;
    }
}
```

Initializes global variable accessible only to the function

Helpful Coding Tips

```
unsigned long memoized_fac(unsigned long n)
{
    /* Fancy initialization of the static map with an initializer list Maps 0->1 */
    static map<unsigned long, unsigned long> memo = {
        {0, 1},
    };

    map<unsigned long, unsigned long>::iterator lookup = memo.find(n);
    if (lookup != memo.end()) {
        return lookup->second;
    } else {
        unsigned long result = n * memoized_fac(n - 1);
        memo[n] = result;
        return result;
    }
}
```

Returns memo.end if not in map