Recursive Functions and Memoization

1. Recursive Functions
   - computing Fibonacci numbers
   - an exponential number of function calls

2. Memoization
   - storing results of function calls
   - using a dictionary

MCS 320 Lecture 20
Introduction to Symbolic Computation
Jan Verschelde, 3 July 2024

Recursive Functions and Memoization

1. Recursive Functions
   - computing Fibonacci numbers
   - an exponential number of function calls

2. Memoization
   - storing results of function calls
   - using a dictionary

computing Fibonacci numbers

The Fibonacci numbers $F(n)$ are defined as

$$F(0) = 0, \quad F(1) = 1, \quad n > 1 : F(n) = F(n-1) + F(n-2).$$

This definition leads to a natural recursive function:

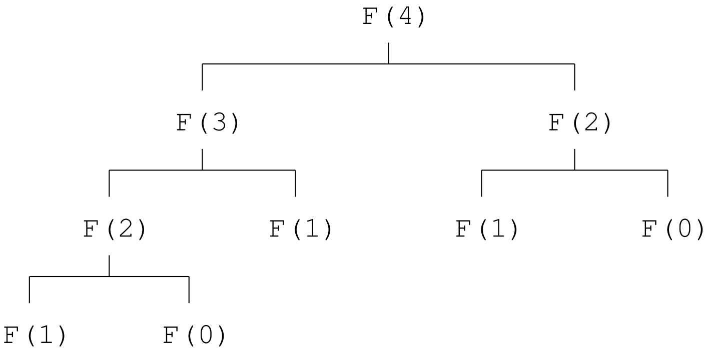
```
def F(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else
        return F(n-1) + F(n-2)
```

Recursive Functions and Memoization

1. Recursive Functions
   - computing Fibonacci numbers
   - an exponential number of function calls

2. Memoization
   - storing results of function calls
   - using a dictionary

an exponential number of function calls



Let c_n be the number of calls to compute $F(n)$.

- Consider some base cases: $c_2 = 2$, $c_3 = 4 = 2^2$, $c_4 = 8 = 2^3$.
- Recursion: $c_n = c_{n-1} + c_{n-2} + 2 = \cdots$ is $O(2^n)$.

Recursive Functions and Memoization

1. Recursive Functions
   - computing Fibonacci numbers
   - an exponential number of function calls

2. Memoization
   - storing results of function calls
   - using a dictionary

storing results of function calls

Definition (memoization)

Memoization is a technique to improve the performance of programs by storing the results of function calls.

Applied to the Fibonacci numbers, the list

```
L = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

stores the first 10 Fibonacci numbers, $F(n) = $ `L[n]`.

Two steps in a memoized computation of the n-th Fibonacci number:

1. Before computing $F(n)$, return `L[n]` if it exists.
2. After computing a new $F(n)$, store $F(n)$ as `L[n]`.

Recursive Functions and Memoization

1. Recursive Functions
   - computing Fibonacci numbers
   - an exponential number of function calls

2. Memoization
   - storing results of function calls
   - using a dictionary

using a dictionary

```
def memoizedF(n, D={}):
    """
    All calls made to memoizedF are stored in D.
    """
    if n in D:          # dictionary lookup
       return D[n]
    else:
       if n == 0:
           result = 0
       elif n == 1:
           result = 1
       else:
           result = memoizedF(n-1) + memoizedF(n-2)
       D[n] = result    # store the result in D
       return result
```