CS 106B
Lecture 12: Memoization and Structs
Friday, April 27, 2018

Programming Abstractions in C++, Chapter 10
Today's Topics

- Logistics
- Midterm Next Thursday
- Assignment 4 is out, although we will cover classes on Monday

- Memoization
- More on Structs
The Triangle Game

https://www.youtube.com/watch?v=kbKtFN71Lfs&feature=youtu.be

The whole video is great, but there is a real treat starting at 5:36
Tell me and I forget. Teach me and I rememoize.*

- Xun Kuang, 300 BCE

* Some poetic license used when translating quote
• Let's look at one of the most beautiful recursive definitions:

\[ F_n = F_{n-1} + F_{n-2} \]

where \( F_0 = 0 \), \( F_1 = 1 \)

• This definition leads to this:
Beautiful Recursion

- And this:
• And this:

Beautiful Recursion
• And this:

Beautiful Recursion
Beautiful Recursion

• And this:
Beautiful Recursion

- And this:
The Fibonacci Sequence

\[ F_n = F_{n-1} + F_{n-2} \]

where \( F_0 = 0 \), \( F_1 = 1 \)

This is particularly easy to code recursively!

```c
long plainRecursiveFib(int n) {
    if(n == 0) {
        // base case
        return 0;
    } else if(n == 1) {
        // base case
        return 1;
    } else {
        // recursive case
        return plainRecursiveFib(n - 1) + plainRecursiveFib(n - 2);
    }
}
```

Let's play!
The Fibonacci Sequence

What happened??

Recursive Fibonacci

<table>
<thead>
<tr>
<th>n</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>38</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
</tr>
</tbody>
</table>
The Fibonacci Sequence

What happened??

Recursive Fibonacci

\[ y = 3E-06e^{0.4852x} \]

\[ R^2 = 0.99986 \]

O\(a^n\)
The Fibonacci Sequence

What happened??

Recursive Fibonacci

https://www.youtube.com/watch?v=qXNqEURmKtA

y = 3E-06e^{0.4852x}
R^2 = 0.99986

O(a^n)
The Fibonacci Sequence

What happened??

https://www.youtube.com/watch?v=qXNqEURmKtA
The Fibonacci Sequence

By the way:

$$3 \times 10^{-6} e^{0.4852n} \approx O(1.62^n)$$

$O(1.62^n)$ is technically $O(2^n)$ because

$$O(1.62^n) < O(2^n)$$

We call this a "tighter bound," and we like round numbers, especially ones that are powers of two. :)

Recursive Fibonacci

The Fibonacci Sequence

By the way:

$$3 \times 10^{-6} e^{0.4852n} \approx O(1.62^n)$$

$O(1.62^n)$ is technically $O(2^n)$ because

$$O(1.62^n) < O(2^n)$$

We call this a "tighter bound," and we like round numbers, especially ones that are powers of two. :)
This is basically the reverse of binary search: we are splitting into two marginally smaller cases, not splitting into half of the problem size!
Fibonacci: There is hope!

Notice! A repeat!

$\text{fib}(3)$ is completely calculated twice
Fibonacci: There is hope!

more repeats!
Fibonacci: There is hope!

let's leverage all the repeats!
Fibonacci: There is hope!

If we store the result of the first time we calculate a particular fib(n), we don't have to re-do it!
Memoization: Store previous results so that in future executions, you don’t have to recalculate them.

aka

Remember what you have already done!
Memoization: Don't re-do unnecessary work!

Cache: <empty>
Memoization: Don't re-do unnecessary work!

Cache: <empty>
Memoization: Don't re-do unnecessary work!

Cache: <empty>
Memoization: Don't re-do unnecessary work!

Cache: <empty>
Memoization: Don't re-do unnecessary work!

Cache: <empty>
Memoization: Don't re-do unnecessary work!

Cache: fib(2) = 1
Memoization: Don't re-do unnecessary work!

Cache: $\text{fib}(2) = 1$, $\text{fib}(3) = 2$
Memoization: Don't re-do unnecessary work!

Don't recurse! Use the cache!

Cache: \( \text{fib}(2) = 1 \), \( \text{fib}(3) = 2 \)
Memoization: Don't re-do unnecessary work!

Cache: fib(2) = 1, fib(3) = 2
Memoization: Don't re-do unnecessary work!

Don't recurse! Use the cache!

Cache: \(\text{fib}(2) = 1, \text{fib}(3) = 2, \text{fib}(4) = 3\)
Memoization: Don't re-do unnecessary work!

Don't recurse! Use the cache!

Cache: fib(2) = 1, fib(3) = 2, fib(4) = 3
Memoization: Don't re-do unnecessary work!

Cache: \( \text{fib}(2) = 1, \text{fib}(3) = 2, \text{fib}(4) = 3, \text{fib}(5) = 5 \)
Memoization: Don't re-do unnecessary work!

Cache: \( \text{fib}(2) = 1, \text{fib}(3) = 2, \text{fib}(4) = 3, \text{fib}(5) = 5 \)
Memoization: Don't re-do unnecessary work!

```java
long memoizationFib(int n) {
    Map<int, long> cache;
    return memoizationFib(cache, n);
}
```

setup for helper function
long memoizationFib(int n) {
    Map<int, long> cache;
    return memoizationFib(cache, n);
}

long memoizationFib(Map<int, long>&cache, int n) {
    if(n == 0) {
        // base case #1
        return 0;
    } else if (n == 1) {
        // base case #2
        return 1;
    } else if(cache.containsKey(n)) {
        // base case #3
        return cache[n];
    }
    // recursive case
    long result = memoizationFib(cache, n-1) + memoizationFib(cache, n-2);
    cache[n] = result;
    return result;
}
Memoization: Don't re-do unnecessary work!

Complexity?

The recursive path only happens on the left...

\( O(n \log n) \) if using a map for the cache

\( O(n) \) if using a hashmap for the cache
There are actually many ways to write a fibonacci function.

This is a case where the plain old iterative function works fine:

```c
long iterativeFib(int n) {
    if(n == 0) {
        return 0;
    }
    long prev0 = 0;
    long prev1 = 1;
    for (int i=n; i >= 2; i--) {
        long temp = prev0 + prev1;
        prev0 = prev1;
        prev1 = temp;
    }
    return prev1;
}
```

Recursion is used often, but not always.
Another way to keep track of previously-computed values in fibonacci is through the use of a different helper function that simply passes along the previous values:

```c
long passValuesRecursiveFib(int n) {
    if (n == 0) {
        return 0;
    }
    return passValuesRecursiveFib(n, 0, 1);
}

long passValuesRecursiveFib(int n, long p0, long p1) {
    if (n == 1) {
        // base case
        return p1;
    }
    return passValuesRecursiveFib(n-1, p1, p0 + p1);
}
```
More on Structs

We have mentioned structs already -- they are useful for keeping track of related data as one type, which can get used like any other type. You can think of a struct as the Lunchable of the C++ world.

```cpp
struct Lunchable {
    string meat;
    string dessert;
    int numCrackers;
    bool hasCheese;
};

// Vector of Lunchables
Vector<Lunchable> lunchableOrder;
```
A Real Problem

Your cool picture from that trip to Europe doesn't fit on Instagram!
Bad Option #1: Crop

You got cropped out!
Bad Option #2: Resize

Stretchy castles look weird...
New Algorithm: Seam Carving!
New Algorithm: Seam Carving!

How can you change an image without changing its aspect ratio, but while retaining the important information?
New Algorithm: Seam Carving!

We could delete an entire column of pixels, but we could also weave our way through a path of 1-pixel wide image that removes the least amount of stuff.
How to represent the path

A struct!

```cpp
struct Coord {
    int row;
    int col;
};
```

A path is just a Vector of coordinates:

```cpp
int main() {
    Coord myCord;
    myCord.row = 5;
    myCord.col = 7;
    cout << myCord.row << endl;
    Vector<Coord> path;
    return 0;
}
```
New Algorithm: Seam Carving!

Important pixels are ones that are considerably different from their neighbors.
New Algorithm: Seam Carving!

Let's write a recursive algorithm that can find the seam that minimizes the sum of all the importances of the pixels.
New Algorithm: Seam Carving!

Vector<Coord> getSeam(Grid<double> &weight, Coord curr);
References and Advanced Reading

• References:
  • https://en.wikipedia.org/wiki/Fibonacci_number
  • https://en.wikipedia.org/wiki/Seam_carving