CS 106X, Lecture 9
Fractals
reading:
*Programming Abstractions in C++, Chapters 7-8*
Plan For Today

• Announcements

• Recap: Runtime and Memoization

• Fractals
  – Cantor fractal
  – Snowflake fractal
  – Emblem fractal
Plan For Today

• Announcements

• Recap: Runtime and Memoization

• Fractals
  – Cantor fractal
  – Snowflake fractal
  – Emblem fractal
Announcements

• HW3 – Recursion – going out at 3PM today
  – Fractals
  – Grammar Generator
  – Human Pyramid
Plan For Today

• Announcements

• Recap: Runtime and Memoization

• Fractals
  – Cantor fractal
  – Snowflake fractal
  – Emblem fractal
void reverseLines(ifstream& input) {
    string line;
    if (getline(input, line)) {
        reverseLines(input);
        cout << line << endl;
    }
}

• What is the Big-O of the above function?
• (What is N?)

How many times is this function called in total? x What is the runtime of each individual function call?
• The runtime of a recursive function is the number of function calls times the work done in each function call.
• The number of calls for a branching recursive function is usually $O(b^d)$

where
  – $b$ is the worst-case branching factor (# recursive calls per function execution)
  – $d$ is the worst-case depth of the recursion (the longest path from the top of the recursive call tree to a base case).
Fibonacci: Big-O

Each recursive call makes 2 additional recursive calls.
The worst-case depth of the recursion is the index of the Fibonacci number we are trying to calculate (N).
Therefore, the number of total calls is $O(2^N)$.
Each individual function call does $O(1)$ work. Therefore, the total runtime is $O(2^N) \times O(1) = O(2^N)$.
Is there a way to remember what we already computed?
/ Returns the nth Fibonacci number (no error handling).
// This version uses memoization.
int fibonacci(int i, Map<int, int>& cache) {
    if (i < 2) {
        return i;
    } else if (cache.containsKey(i)) {
        return cache[i];
    } else {
        int result = fibonacci(i-1, cache) + fibonacci(i-2, cache);
        cache[i] = result;
        return result;
    }
}
Wrapper Functions

Map<int, int> cache;
int sixthFibonacci = fibonacci(5, cache);  // 5

• The above function signature isn’t ideal; it requires the client to know to pass in an (empty) map.

• In general, the parameters we need for our recursion will not always match those the client will want to pass.

• Is there a way we can remove that requirement, while still memoizing?

• YES! A “wrapper” function is a function that “wraps” around the first call to a recursive function to abstract away any additional parameters needed to perform the recursion.
That’s a Wrap(per)!

// “Wrapper” function that returns the nth Fibonacci number.
// This version calls the recursive version with an empty cache.
int fibonacci(int i) {
    Map<int, int> cache;
    return fibonacci(i, cache);
}

// Recursive function that returns the nth Fibonacci number.
// This version uses memoization.
int fibonacci(int i, Map<int, int>& cache) {
    if (i < 0) {
        throw "Illegal negative index";
    } else if (i < 2) {
        return i;
    } else if (cache.containsKey(i)) {
        return cache[i];
    } else {
        int result = fibonacci(i-1, cache) + fibonacci(i-2, cache);
        cache[i] = result;
        return result;
    }
}
Plan For Today

• Announcements

• **Recap:** Runtime and Memoization

• Fractals
  – Cantor fractal
  – Snowflake fractal
  – Emblem fractal
A **fractal** is a recurring graphical pattern. Smaller instances of the same shape or pattern occur within the pattern itself.
Fractals in Nature

Many natural phenomena generate fractal patterns:

1. earthquake fault lines
2. animal color patterns
3. clouds
4. mountain ranges
5. snowflakes
6. crystals
7. DNA
8. ...
Cantor Fractal
Parts of a cantor set image ... are Cantor set images
Cantor Fractal

Another cantor set

Another cantor set
Level 1 Cantor Fractal
1. Draw a line from start to finish.

2. Draw a Cantor of size $n-1$
GWindow w;
GPoint a(100, 100);
cout << a.getX() << endl;
GWindow w;
GPoint a(100, 100);
GPoint b(20, 20);
w.drawLine(a, b);
Cantor Fractal
Plan For Today

- Announcements
- **Recap:** Runtime and Memoization
- Fractals
  - Cantor fractal
  - **Snowflake fractal**
  - Emblem fractal
Snowflake
Snowflake
Snowflake
Snowflake
Snowflake
Snowflake
Plan For Today

- Announcements
- **Recap**: Runtime and Memoization
- Fractals
  - Cantor fractal
  - Snowflake fractal
  - Emblem fractal
Emblem Fractal
• We want to draw this figure at a given center and radius on-screen.

• An order-0 emblem is nothing

• An order-1 emblem is a circle of the specified size

• An order-n emblem is a circle of the specified size, containing 6 order n-1 emblems at increments of 60 degrees around the circle 2/3 away from the center, with 1/3 the radius.
• Fractals
  • Fractals are self-referential, and that makes for nice recursion problems!
  • Break the problem into a smaller, self-similar part, and don't forget your base case!
References and Advanced Reading

• References:
  • [http://www.cs.utah.edu/~germain/PPS/Topics/recursion.html](http://www.cs.utah.edu/~germain/PPS/Topics/recursion.html)
  • Why is iteration generally better than recursion? [http://stackoverflow.com/a/3093/561677](http://stackoverflow.com/a/3093/561677)

• Advanced Reading:
  • Tail recursion: [http://stackoverflow.com/questions/33923/what-is-tail-recursion](http://stackoverflow.com/questions/33923/what-is-tail-recursion)
  • Interesting story on the history of recursion in programming languages: [http://goo.gl/P6Einb](http://goo.gl/P6Einb)