Recursion

2020 Spring: AP Computer Science A

February 5th, 2020

Today

- **Recursion**
- **Complexity**
- **Memoization**

Recursion

- ***recursion*: The definition of an operation in terms of itself**
  - Solving a problem using recursion depends on solving smaller occurrences of the same problem

- ***recursive programming*: Writing methods that call themselves to solve problems recursively**
  - An equally powerful substitute for iteration
  - Particularly well-suited to solving certain types of problems

Recursive Programming

- **Every recursive algorithm involves at least 2 cases**

- ***base case*****: A simple occurrence that can be answered directly**
- ***recursive case*****: A more complex occurrence of the problem that cannot be directly answered, but can instead be described in terms of smaller occurrences of the same problem**

  - Some recursive algorithms have more than one base or recursive case, but all have at least one of each

- **Always identify these cases!**

Example

- **Let's try to write this method recursively**
  - The objective is to **print n stars**
  - Notice that for each iteration, the problem gets smaller
    - After 1 iteration, now the objective is to **print $n - 1$ stars**
  - Other than the problem size, it is identical to the original problem

```
public static void printStars(int n) {
    for(int i = 0; i < n; ++i)
        System.out.print("*");
    System.out.println();
}
```

Base Case

- **Base case should...**
  - Be relatively easy to compute/handle
  - Provide a termination condition for the recursive calls

```java
public static void printStars(int n) {
    if (n == 1) {
        System.out.println("*");
    } else {
        // ...
    }
}
```

Recursive Case

- **Call the smaller version of the same problem**

```
public static void printStars(int n) {
    if (n == 1) {
        System.out.println("*");
    } else {
        System.out.print("*");
        printStars(n - 1);
    }
}
```

Exercise

- **Guess the value of `mystery(648)`**

```java
public static int mystery(int n) {
    if(n < 10) {
        return n;
    } else {
        int a = n / 10;
        int b = n % 10;
        return mystery(a + b);
    }
}
```

Exercise

- **Guess the value of `mystery(648)`**

```java
public static int mystery(int n) {
    if(n < 10) {
        return n;
    } else {
        int a = n / 10;
        int b = n % 10;
        return mystery(a + b);
    }
}
```

- **`mystery(648)`**
  - `a = 64, b = 8`
  - `return` **`mystery(72)`**
    - `a = 7, b = 2`
    - `return` **`mystery(9)`**
      - `return 9`

Relation to Sequences

- **What are factorials?**
  - Let $f(n) = n!$
- $$\boldsymbol{f}(\boldsymbol{n}) = \begin{cases} 1 & (n = 0) \\ n \times \boldsymbol{f}(\boldsymbol{n-1}) & (n > 0) \end{cases}$$
  - Factorials can be written recursively!

- **#10872 팩토리얼**

Relation to Sequences

- **The famous *Fibonacci sequence***
  - Let f_n denote the n-th Fibonacci number
- $$f_n = \begin{cases} 0 & (n = 0) \\ 1 & (n = 1) \\ f_{n-1} + f_{n-2} & (n \geq 2) \end{cases}$$
- **#10870 피보나치 수 5**

Tower of Hanoi

- **#11729 하노이의 탑 이동 순서**

- **Problem:** ***Move n disks from post 1 to post 3 !***

- **Base case:** moving a single disk
- ***Recursive case***
  - Move $n-1$ disks from post 1 to post 2
  - Move a single disk from post 1 to post 3
  - Move the $n-1$ disks from post 2 to post 3

Tower of Hanoi

- **Observation**
  - Number of disks matters, obviously
  - The posts that we use also matter

- **Define $f(n, s, t)$ as the sequence of operations to**
  - Move n disks from post s to post t
  - Define the leftover post as u

- **Base case:** moving a single disk from post s to post t $\quad f(1, s, t)$

- ***Recursive case***
  - Move $n-1$ disks from post s to post u $\quad f(n-1, s, u)$
  - Move a single disk from post s to post t $\quad f(1, s, t)$
  - Move the $n-1$ disks from post u to post s $\quad f(n-1, u, s)$

Complexity

- ***Running time is proportional to the number of operations (roughly)***

- **How many operations does this take?**

```
int x = 1;
int y = 3;
int z = x + y;
System.out.println(z);
```

- **How many operations does this take?**

```
int[] arr = new int[10];
for (int i = 0; i < arr.length; ++i) {
    arr[i] = i * i;
}
```

Complexity

- If there is an input for a program, its running time may depend on the input
  - Ex. Input is an array of size n, whose length can vary

- ***We ignore the details and try to find the part of code that dominates the overall running time!***

- **Big-O notation:** Suppose our program takes an input of size n. We say that *"This program has complexity $\mathcal{O}(f(n))$"* if the *running time of the program is proportional to $f(n)$.*
  - Mathematically, if the running time our program is $T(n)$,
    - $T(n)$ is $\mathcal{O}(f(n))$ if and only if $\lim_{n\to\infty} \frac{T(n)}{f(n)} = c < \infty$
    - And we write $T(n) = \mathcal{O}(f(n))$

Complexity

- **What are the complexities of these operations?**
  - `(ArrayList) add(int idx, E e)`
  - `(Stack) push(E e)`
  - `(Queue) pop()`
  - `(IntQueue) pop()`
  - `(Deq) pushFront(E e)`
  - `(ArrayList) remove(int idx)`
  - `(ArrayList) get(int idx)`

Complexity

- **Comparisons of complexities**

Complexity	Feel	Meaning
$\mathcal{O}(1)$	😎	Speed doesn't depend on dataset
$\mathcal{O}(\log n)$	😃	10x data means 2x more time
$\mathcal{O}(n)$	🙂	10x data means 10x more time
$\mathcal{O}(n \log n)$	😐	10x data means 20x more time
$\mathcal{O}(n^2)$	☹️	10x data means 100x more time
$\mathcal{O}(2^n)$	😰	10x data means 1024x more time
$\mathcal{O}(n!)$	😱	10x data means 3628800x more time

Analysis of Recursive Programs

- **If the running time is $T(n)$**

- **#10872 팩토리얼**
  - $T(n) = 1 + T(n - 1), T(0) = 1$
  - $T(n) = n + 1$
  - $T(n) = \mathcal{O}(n)$

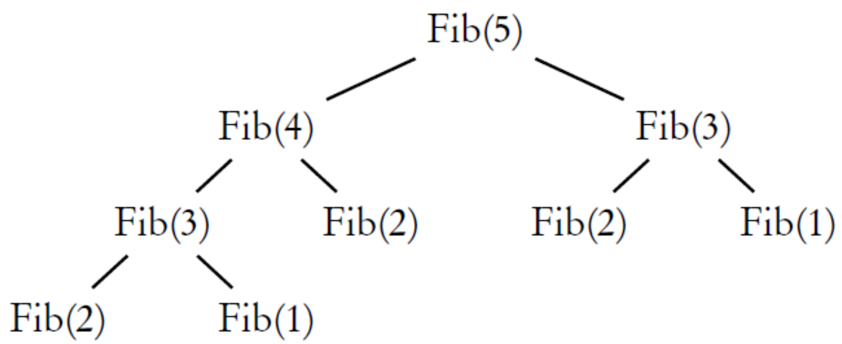
- **#10870 피보나치 수 5**
  - $T(n) = T(n-1) + T(n-2) + 1, T(0) = T(1) = 1$
  - $T(n) = \frac{2}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^{n+1} - \left(\frac{1-\sqrt{5}}{2}\right)^{n+1}\right) - 1$
  - $T(n) = \mathcal{O}\left(\left(\frac{1+\sqrt{5}}{2}\right)^{n}\right)$ (Terrible)

Improving Fibonacci

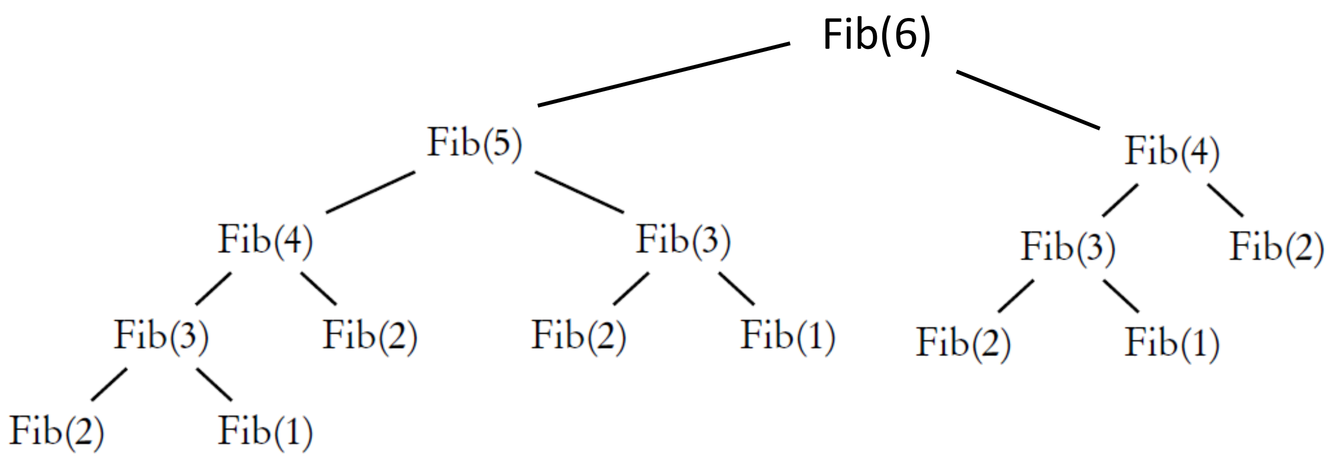
- **Why is Fibonacci so inefficient?**

Improving Fibonacci

- **Same function with same parameter is called multiple times!**



- **Does this seem okay?**



Improving Fibonacci

- **What if we saved the values after computing them?**
  - If there is a saved value, use the value
  - If there isn't, compute it

- ***memoization*: saving computed values for future reference**

- **Usually the results are stored in an array**

Improving Fibonacci

- **#2748 피보나치 수 2**

- **Create an integer array of size 100 to store the results**

`Pow(x, n)`

- **Write a function to compute x^n, recursively**

- **Base case?**
- **Recursive case?**

`Pow(x, n)`

- **Can we do better?**
- **Find a better recursive case!**
- **Aim for complexity** $\mathcal{O}(\log n)$

Palindrome

- **#10988 팰린드롬인지 확인하기**

- **Write a recursive program to check if a given string is a palindrome**

- **palindrome: A string that is same as itself when reversed**
  - The string is read the same forwards and backwards

2D Memoization

- **#11051 이항 계수 2**

- **#2167 2차원 배열의 합**

Euclidean Algorithm

- Greatest common divisor
- #2609 최대공약수와 최소공배수
- #1850 최대공약수
- #17504 제리와 톰 2