LogicVis

By Candice Miao, Leo Gao, Jed Chen, Glenn Zhang, Andrew Liu

Introduction

People have trouble understanding code in many situations

- Learning to code
- Reading others' code

Recursion is one of the topics that are difficult to understand

Use visualization to help understanding code, especially recursion

Outline

- **The Problem**
  - Our Solution
  - Research
  - Challenges and Risks

Problem: Understanding Recursion

Study shows that only 13% of the students are able to completely understand recursion.

Reasons

1. Students are not using abstractions well enough
2. Lack of a proper methodology to represent a recursive solution

Scholtz, Tamarisk Lurlyn, and Ian Sanders. “Mental Models of Recursion.” *Proceedings of the Fifteenth Annual Conference on Innovation and Technology in Computer Science Education - ITiCSE '10*, 2010, doi:10.1145/1822090.1822120.

Understanding Recursion is important

- Make Programming Easier
  - A simple but essential approach for certain tasks
- An approach that can be used in most programming languages
- Truncate massive amount of code

Motivation

- Make understanding foreign code easier
- Help programmer debug
- Facilitate the learning experience for new learners in Computer Science

Previous Approaches

Verbal Explanation
- Not strong enough

Code Visualization Tools
- Eclipse CFG Generator
    - No indication of recursion

Recursion Trackers
- VisuAlgo
    - Limited information presented

Outline

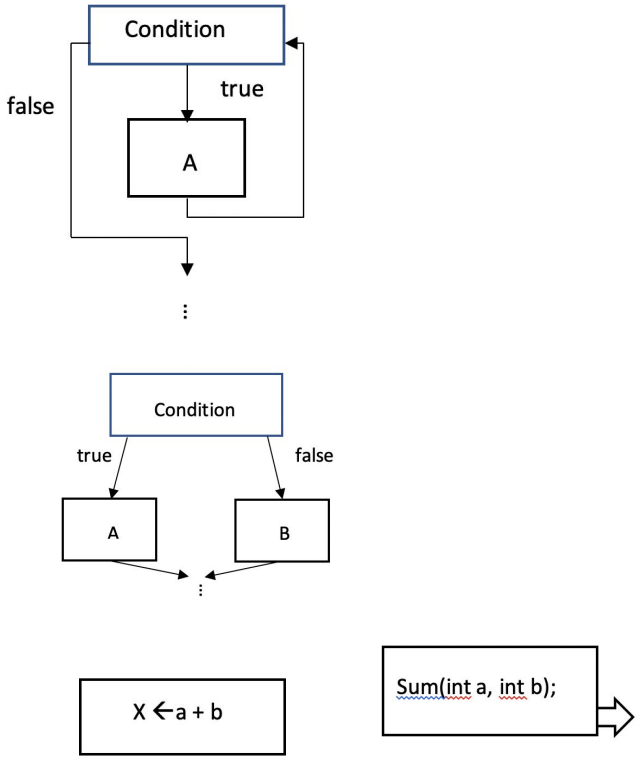
- The Problem
- **Our Solution**
- Research
- Challenges and Risks

Approach

Building an application to translate a Java method into a graph.

- Easier for people to track the control flow of the code.
- In the case of recursion, allows people to track the logic within the method instead of just the value of parameters

Integration with JavaFX for UI.



Why it is useful

- Visualized Logic is easier to understand
- Our solution not only indicates the function call but also visualizes the specific call iteration and instruction pointer
- Our solution provides more information than just the parameters of each function call
    - Visualizes the execution

Example

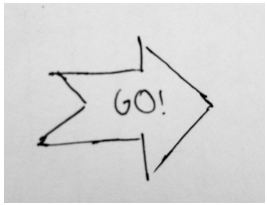
```
public static void mystery(int x) {
    mystery(x, 2);
}

private static void mystery(int x, int n) {
    if (n > x / n) {
        System.out.println(x);
    } else if (x % n == 0) {
        System.out.print(n + " ");
        mystery(x / n, n);
    } else {
        mystery(x, n + 1);
    }
}
```

How would we trace **`mystery(20)`**?

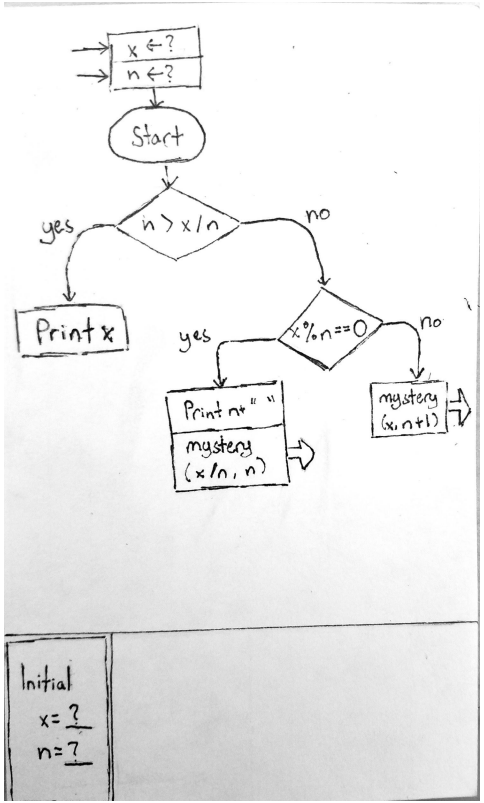
Example

```
public static void mystery(int x, int n) {
    if (n > x/n) {
        System.out.println(x);
    else if (x%n == 0){
        System.out.print(n+" ");
        mystery(x/n, n);
    } else {
        mystery(x, n+1);
    }
}
```

Input



Set input

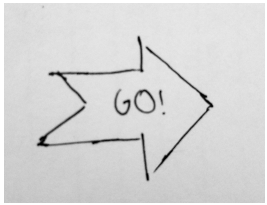


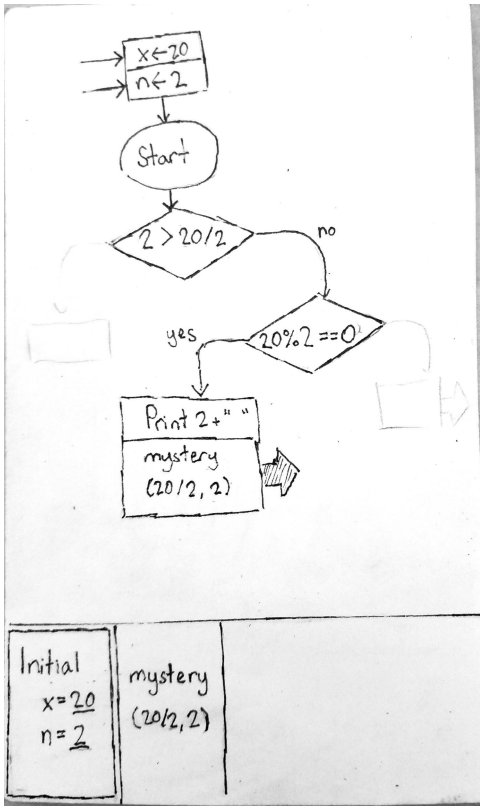
Output:

Example

```
public static void mystery(int x, int n) {
    if (n > x/n) {
        System.out.println(x);
    else if (x % n == 0){
        System.out.print(n+" ");
        mystery(x/n, n);
    } else {
        mystery(x, n+1);
    }
}
```

Input



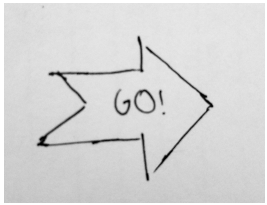


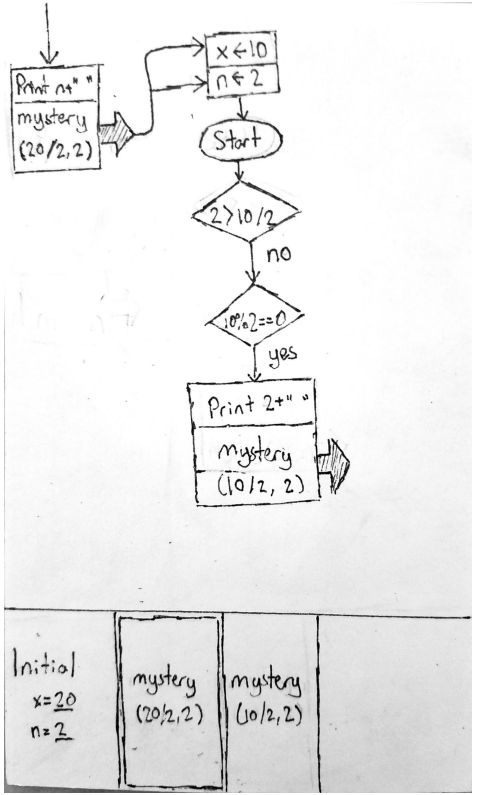
Output:

Example

```
public static void mystery(int x, int n) {
    if (n > x/n) {
        System.out.println(x);
    else if (x%n == 0){
        System.out.print(n+" ");
        mystery(x/n, n);
    } else {
        mystery(x, n+1);
    }
}
```

Input



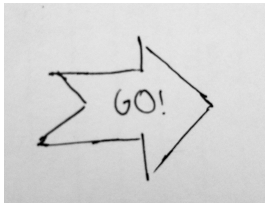


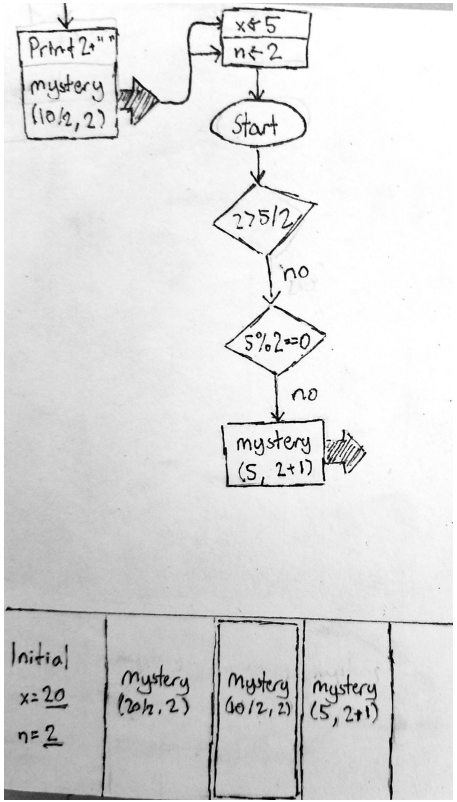
Output:
2

Example

```
public static void mystery(int x, int n) {
    if (n > x/n) {
        System.out.println(x);
    else if (x % n == 0){
        System.out.print (n+ " ");
        mystery (x/n, n);
    } else {
        mystery(x, n+1);
    }
}
```

Input



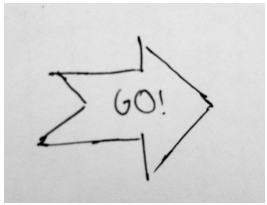


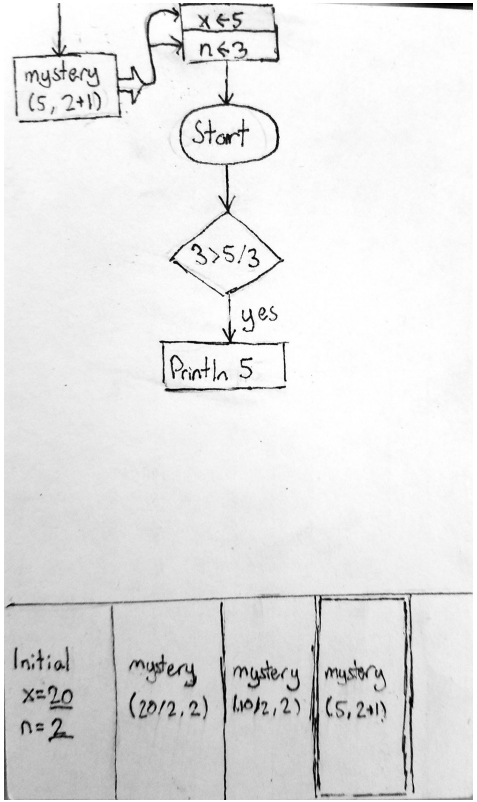
Output:
2 2

Example

```
public static void mystery(int x, int n) {
    if (n > x/n) {
        System.out.println(x);
    else if (x % n == 0){
        System.out.print(n+" ");
        mystery(x/n, n);
    } else {
        mystery(x, n+1);
    }
}
```

Input



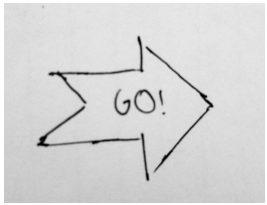


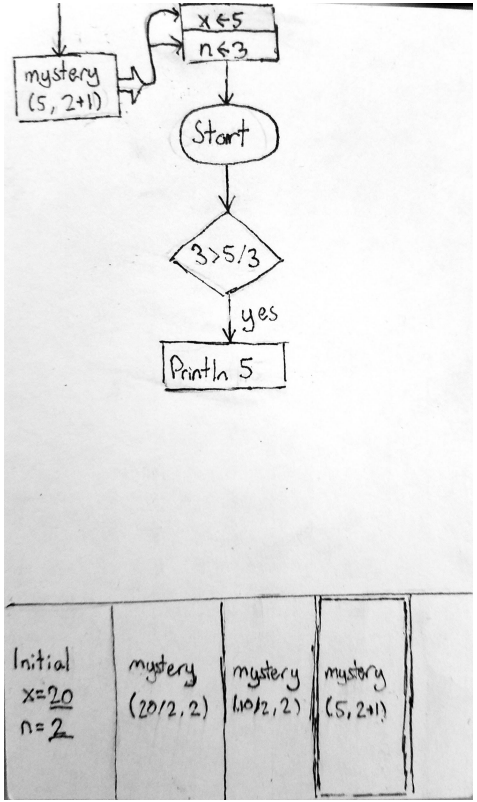
Output:
2 2

Example

```
public static void mystery(int x, int n) {
    if (n > x/n) {
        System.out.println(x);
    else if (x%n == 0){
        System.out.print(n+" ");
        mystery(x/n, n);
    } else {
        mystery(x, n+1);
    }
}
```

Input





Output:
2 2 5

Prime Factorization!

Outline

- The Problem
- Our Solution
- **Research**
- Challenges and Risks

Hypothesis

- By expanding the condensed form recursion usually takes, we can also unpack the difficulty of understanding the abstraction
- A tool to expand recursive code can help students understand the impact of every line they write

Metrics

- Focus on user studies
- Paper Prototyping set functions
- Measure Time and Understanding
- The actual effect of the tool on helping students write code is difficult to observe without the final product

Related Work

ChiQat-Tutor system, a system that helps visualize the recursive calls.

Statistics from such studies on this system shows a huge improvement on students understanding of recursion by using such a system.

Limitations: Pre-designed recursive cases visualized for teaching purposes, not able to generalize to random recursive code.

AlZoubi, Omar & Fossati, Davide & Di Eugenio, Barbara & Green, Nick & Alizadeh, Mehrdad & Harsley, Rachel. (2015). A Hybrid Model for Teaching Recursion. 10.1145/2808006.2808030.

Preliminary Results

- Learned to specify (wish we had known a while ago)
- Specific Challenges: Difference in users, Making UI intuitive

Outline

- The Problem
- Our Solution
- Research
- **Challenges and Risks**

Challenges and Risks

1. Making the recursion visualization intuitive for everyone is difficult.
2. The scope of the project is not able to be decided due to the limit of time
    a. Integration with UI Frameworks
    b. Tracking of values during execution
    c. UI design

Conclusion

- The pursuit to teach Recursion better is not a novel idea
- Aim at the root of the problem in order to solve it