APCS Notes

Yicheng Wang

2014-2015

Contents

1 2015-02-05

1.1 Do Now

Figure out what the following code does.

```
public void printme(int n) {
    if (N > 0) {
        printme(n - 1);
        System.out.println(n);
    }
}
```

It should print out an increasing sequence of numbers from 1-N.

1.2 Stack and ROP

The stack on top is the current function, and each layer beneath that is the function that called the current function.

1.3 Recursion

Simple recursive problem: FACTORIAL
Hallmarks of a recursive solutioin:

- Base Case: thing that stops the program, simple case you know the answer of. In the case of factorials, factorial(0) = 1
- Reduction Case: You need to alternate the variable in some sort of way, for example, we should do n * factorial(n - 1)
- Recursion: function A need to eventually call A

Final code:

```
public int factorial(n) {
    if (n == 0) {
        return 1; // Base Case
    }
    else {
        return n * factorial(n - 1); // Reduction Step
    }
}
```

2 2015-02-06

2.1 Traditional Recursion Examples

2.1.1 Fibonacci Numbers

1, 1, 2, 3, 5, 8, 13 ...

Base Case: if n ¡ 2, return 1
Reduction Step: fib(n) = fib(n - 1) + fib(n - 2)
Example Code:

```java
public int fib(int n) {
    if (n < 2) {
        return 1;
    }

    else {
        return fib(n - 1) + fib(n - 2);
    }
}
```

2.1.2 List/String Manipulation

Example, finding the length of a substring.
Base case: ”” has length of 0
Reduction Step: 1 + ”cdr” of the string, i.e. s.substring(1);
Example Code:

```java
public int lenStr(String s) {
    if (s.equals("")) {
        return 0;
    }
    else {
        return 1 + lenStr(s.substring(1));
    }
}
```

3 2015-02-09

3.1 Getting out of a Maze

1. Maze vs. Laybrith: Laybrith may not have choices, mazes have choices.

2. Strategies in solving the maze:

    - Doesn’t work due to loops

3. Greek Way of Solving Mazes:

    - Invented by Odysseus, bring a thread, use process of elimination to loop through all possible intersection.
    - Works really well.

Maze solving in java

1. This is clearly a recursive solution.

2. Using a recursive solution we can easily trace back the stack to find the previous intersection

3. We represent our map as a char array of paths ('#')

4. Base case: location is a wall OR location of exit

5. Reduction step: call solve at a different (x,y) location, if it's a dead end, it'll be peeled off the stack due to the "return"

6. We will try to solve the maze systematically, x+/-1, y+/-1.

4 2015-02-11

4.1 Blind Search

Trying all possibilities until we find our solutions. Also called exhaustive search, linear search, recursive search, search with backtracking.

The maze algorithm we wrote is sometimes known as a depth first search. It's because you say in one path for as long as possible. Advantage is if the solution is deep, this works pretty well. However, if the solution is closer, breath first search goes n steps in all possible directions.

4.2 State Space Search

Search algorithm to solve a problem, searching for a "state space."

Idea is if you have a problem, you can describe said problem as a "state."

State is a configuration of the world, such as turtle in netLogo.

The world is made up of many states, and one can transition from one state to the next.

State Space Search: The series of states one has to go through to get to the desirable final state. (like exit in the maze thing)

4.3 Graph Theory

Graphs are collections of edges and nodes. Nodes represent the states, edges marks the transition between one node to the next.

5 2015-02-12

5.1 Space State Search

Examples of space state search:

- Maze path-finding

- 15 puzzle

- Cube
- Chess algorithm – More complex
  - two players!
  - high branching factor
  - harder base case
  - uses mini-max searching: best for me and worst for you
- Description of an entity

5.2 Implicit Data Structure

When we did the maze solver, we used an explicit data structure $\rightarrow$ the 2D array. However, the graph of the transition is also a data structure, it's implicit though, just running in the background. Whenever we call solve, it creates a node on the stack. However, if a call returns, it destroy that branch of the graph.

As the program runs, we can imagine it as a graph.

5.3 PROJECT

Do the diagonistic exam in the Barron's Review book, do as follows:

- First do it under test condition
- Go back and tried to fix your problem

Option 1. Knight Tour:

Given a $N \times N$ board, find a path such that the knight visits each square once without repetition. (Start with 5×5);

Option 2. N-Queen:

A way to place N mutually non-attacking queens on a $N \times N$ chessboard.