Scribe: Chaitanya Garg

Puzzle: Spacecraft Malfunction in Dr. Ecco

When A accuses B of being faulty, either A is faulty or B is faulty. However, both A and B cannot be faulty. And if A says B is good, it doesn't mean that B is good because A could be faulty. One hint is to assume that only a few of the spots are bad. If they're all bad, there's really not much to do.

Prof. Shasha involved in a lawsuit as an expert witness. Company guaranteed 99.99999% uptime. 12 boxes. From a hardware point of view, that's fine. But from software, they all ran the same software. Their data structure was a variant of a binary tree called red and black tree (which is really hard to implement properly). Their delete method did not work.
This delete bug along with translation from phone number to SIM card led to a 17 hour downtime, and that led to a lawsuit.

What NASA does: 4 identical machines with identical software. A 5^{th} machine with different software done by a different company. This machine was on a space shuttle, and its job was to bring the shuttle back to earth. If a hardware error knocked out 1 or 2 of the first 4, that's fine. If it knocked out 3 or 4, the 5^{th} would kick into place.

Presentation by Justin on Glenn Reeves.

Presentation by Jillian by Louis Qualls.

Backtracking:

Try a solution, see it doesn't work, so go back and try a different solution. So you have to pretty much save every state of your solution. Data structure used is called a stack. You can "push" onto the stack or "pop" off the stack. You always work with the top of the stack. Think of it like a stack of books. You can look at the top, take the top book off, or put another book on top.
Not super-intelligent. If there's any square that there's no choice, fill it in. If no such square, find the square with the fewest number of choices. Save. Pick a choice. Repeat.

Genetic algorithms:

There's a value. This value is given to the genetic algorithm. A candidate solution is tested using that value. Arguments can still pop up when discussing what the value should be.
Candidate solutions also need to have independent factors. An "all others equal" type factor. For example, if a spacecraft has a nuclear engine, that can affect the shielding of the spacecraft. In order for the genetic algorithm to work, you have to have independent variables. If variables are dependent, you have to treat the dependent variables as one variable.

Python Functions:

```
# two ways to solve
# using a loop
# or using recursion

# 1 1 2 3 5 8 13 21
# number is the result of adding the previous two
# f(n) = f(n-1) + f(n-2)
# f(0) = 0
# f(1) = 1
# f(2) = 1
# even though fibonacci is defined as f here and starts really at f(1) = 1
# we're going to look at it from the point of view of arrays so start at f(0) = 1

# arrays start at location 0

def fibfast(num):
    # num is the location of the fibonnaci number you want
    fibarray = [1, 1]
    # fib array holds the value 1 at locaton 0, and the value 1 at location 1
    current_location = 2
    # start at current location 2 as 0 and 1 are already filled in
    while (current_location <= num):
        fibarray.append(fibarray[current_location - 2] + fibarray[current_location - 1])
        # go back 2 locations, get the value, go back 1 location, get the value
        # add those 2 values, and fill this sum into current location
        current_location += 1
        # and of course increment current location
    return fibarray[num]
    # return is what the function spits out, or an output

def fibslow(num):
    # same num as before

    # this is the base case
    # if num is 0 or 1, or num less than or equal to 1, return 1
    # as fib(0) = 1 and fib(1) = 1
    if (num <= 1):
        return 1
    # this will (slowly than before) find the fib value you are looking for
    # the program breaks down the problem into simpler ones, using itself as the solution
    # you can draw a "tree" diagram of how this works

    # however even though this method is the easiest to write, theres a way of optimizing
    # this recursion method using something called dynamic programming
    return fibslow(num - 1) + fibslow(num - 2)

# now we have the two functions defined, but if you run the program like this, there is
no output
# because nowhere does the program actually use the functions

# and now the program uses the functions
```

```
desired_fib_location = input("Please enter the desired location of the fibonacci
number.")
method_of_solution = raw_input("Please enter if you want to find this using a 'l'oop or
'r'ecursion.")
if (method_of_solution == "l"):
    print fibfast(desired_fib_location)
elif (method_of_solution == "r"):
    print fibslow(desired_fib_location)
else:
    print "No valid method of solution was entered."
```

Homework:
Solve spacecraft malfunction, checked by a variant on quiz
Read next 2 chapters in Natural Computing
Design an ‘and’ circuit