University of Illinois at Urbana-Champaign
Dept. of Electrical and Computer Engineering

ECE 220: Computer Systems & Programming

Testing the Nonogram Code

Let's Talk About Testing the Nonogram Code

What about MP4, the nonogram solver?

Corner cases? (such as 1 0 0 0 1)

Zero and non-zero region combinations?

Regions that are
- all X's,
- part X's and part blanks, and
- all blanks?

Other cases?

What about paths through your code?

Example Nonogram Code Solution

Let's work through an example solution
- adapted from a real student's code (not a student at Illinois).
- The code earned 90% of the functionality points using my tester.

Let's
- start with code reading, then
- create tests to cover the code.

Nothing Surprising in the First Part

```
#include <stdio.h>

#include "mp4.h"

int32_t print_row
    (int32_t r1, int32_t r2,
     int32_t r3, int32_t r4,
     int32_t width)
{
```

Were you expecting to see comments?

Initial Code Illuminates Variable Usage

```
int i, j, a, num = 0;
int u[4];
u[0] = r1;
u[1] = r2;
u[2] = r3;
u[3] = r4;
for (i = 0; 4 > i; i++) {
    if (0 != u[i]) {
        num++;
    }
}
```

Uses an array to record region sizes.

num is the number of non-zero regions.

Bizarre Control Flow, But Return Values Seem Ok

```
    if (r1 + r2 + r3 + r4 + num - 1
        > width) {
        return 0;
    } else {
        // print
        // the row
    }
    printf ("\n");
    return 1;
}
```

Space needed for regions (left) and for gaps (right) must fit within **width**.

Control flow done strangely here, but it works.

Regions are Printed One by One (Using Array **u**)

```
// code to print the row
a = width – (r1 + r2 + r3 +
             r4 + num - 1);
```

a is the extra space in the row.

```
for (i = 0; 4 > i; i++) {
    // print one region
}
```

Print regions one at a time.

Start Each Region with Zero or More Blank Spaces

```
if (a > u[i]) {
    for (j = 0; u[i] > j; j++) {
        printf (".");
    }
} else {
    for (j = 0; a > j; j++) {
        printf (".");
    }
}
```

Start by printing **min (a, u[i])** blank spaces.

Print the Region and Maybe a Gap

```
for (j = 0; u[i] - a > j; j++) {
    printf ("X");
}
if ((num - 1) > i && 0 != u[i]) {
    printf (".");
}
```

Region appears with **a** fewer X's.

Print a gap if this region is not the last non-zero (left) and this region is non-zero (right).

After the Last Region, Print Extra Spaces

```
if (i == 3) {
    for (j = 0; a > j; j++) {
        printf(".");
    }
}
```

After end of last region, print **a** extra blanks (should be done unconditionally after the loop).

See Any Bugs?

Did you notice the bug(s)?

I saw one, but let's see if covering the code exposes something.

Let's walk through the code again and see what tests we need.

Need a Test with at Least One Non-Zero Region

```
int i, j, a, num = 0;
int u[4];
u[0] = r1;
u[1] = r2;
u[2] = r3;
u[3] = r4;
for (i = 0; 4 > i; i++) {
    if (0 != u[i]) {
        num++;
    }
}
```

We need at least one non-zero region to execute this line (so any valid input suffices).

Need Tests that Do and Do Not Fit in `width`

```
    if (r1 + r2 + r3 + r4 + num - 1
        > width) {
        return 0;
    } else {
        // print
        // the row
    }
    printf ("\n");
    return 1;
}
```

We need a test that doesn't fit in **`width`**.

And a test that does.

No Requirements for this Block of Code

```
// code to print the row
a = width – (r1 + r2 + r3 +
             r4 + num - 1);
```

All code on this slide always executes.

```
for (i = 0; 4 > i; i++) {
    // print one region
}
```


Two More Requirements for These Loops

```
if (a > u[i]) {
    for (j = 0; u[i] > j; j++) {
        printf (".");
    }
} else {
    for (j = 0; a > j; j++) {
        printf (".");
    }
}
```

We need a non-zero region smaller than **a**.

And a region as large as a non-zero value of **a**.

Print the Region and Maybe a Gap

```
for (j = 0; u[i] - a > j; j++) {
    printf ("X");
}
if ((num - 1) > i && 0 != u[i]) {
    printf (".");
}
```

Need a region larger than **a**.

First region (for which **i** has value 0) is always non-zero.
So we need >1 non-zero region.

No New Requirements for this Block of Code

```
if (i == 3) {
    for (j = 0; a > j; j++) {
        printf(".");
    }
}
```

No new requirements here, since we already need a non-zero value of **a**.

Summary of Tests Needed to Cover All Code

1. Regions that do not fit in **`width`**.
2. Regions that do fit in **`width`**.
3. A non-zero region smaller than extra space.
4. A region as least as large as non-zero extra space.
5. A region larger than extra space.
6. More than one non-zero region.

Notice that covering the code does not even require a zero region (so it's not really enough).

Use Corner Cases When Possible

Try to **use corner cases**. For example,
- for #1 (regions that do not fit in **`width`**),
- let's make the regions 1 too large.
- Say 1, 2, 3, and 4, which needs width 13,
- so we'll set **`width`** to 12.

Test #1: 1 2 3 4 12, which should fail.

Try to Minimize Human Work, Too

For requirement #2 (regions that fit),
- a corner case (an exact fit),
- means no empty space, precluding requirements #3 and #4,
- so let's try to reduce tests instead.

Let's choose extra space as 2.

One More Test Satisfies All Other Requirements!

Given an extra space of 2,
- requirement #3 means that one region should be 1, and
- a region of 3 satisfies requirements #4 and #5.

Together, the two regions above satisfy #6.

So we could try…

Test #2: 1 3 0 0 7, which should print ". . . . X . . \n".

Let's Try the Code on Our Coverage Tests

As you see, we need only two tests to cover all of the code.

Let's try them…

Test #1: 1 2 3 4 12, which should fail.

Test #2: 1 3 0 0 7, which should print ". . . . X . . \n".

The code passes both tests!

Can We Cover Code Blocks that Are Empty?

Let's be slightly more thorough.

When we see
- an **`if`** statement with an **`else`**,
- we cover both then and else blocks.

Did we cover else blocks that do nothing?

Let's take a look.

Need a Test with at Least One Zero Region

```
int i, j, a, num = 0;
int u[4];
u[0] = r1;
u[1] = r2;
u[2] = r3;
u[3] = r4;
for (i = 0; 4 > i; i++) {
    if (0 != u[i]) {
        num++;
    }
}
```

We need at least one zero region to execute the "else" (but we have zero regions in test #2).

Four Possibilities for Two Conditions

```
for (j = 0; u[i] - a > j; j++) {
    printf ("X");
}
if ((num - 1) > i && 0 != u[i]) {
    printf (".");
}
```

There are four possibilities for these two conditions.

How Many Cases Does Test #2 Cover?

Test #2: 1 3 0 0 7

num - 1 > i	0 != u[i]	Test #2?
false	false	regions 3 & 4
false	true	region 2
true	false	**not covered**
true	true	region 1

We Need to Add One More Test

7. A zero region with index (0, 1, 2, 3) less than the number of non-zero regions – 1.

We can't make the first region zero-length,
- so the region index must be at least 1,
- and the number of non-zero regions must be at least 3.

Let's make a tight fit (a corner case), too…

Test #3: 1 0 2 3 8, which should print "X.XX.XXX\n".

Let's Try the Code on Our Coverage Tests

Let's try the last test…

Test #1: 1 2 3 4 12, which should fail.

Test #2: 1 3 0 0 7, which should print "....X..\n".

Test #3: 1 0 2 3 8, which should print "X.XX.XXX\n".

The code fails the third test!

What is Wrong with the Code?

```
if ((num - 1) > i && 0 != u[i]) {
    printf (".");
}
```

What's wrong?

The programmer **confused the region index `i` with the index among non-zero regions**.

All but the last non-zero region should be followed by a gap, but `i` also counts zero regions.

How to Fix the Bug

To fix the bug quickly, we can
- **add a separate variable `non_zero`** to index non-zero regions,
- **initialize `non_zero` to 0** when `i` is set to 0,
- **increment `non_zero` only when we see a non-zero region**, and
- **compare `non_zero` to `(num - 1)`** to decide whether to print a gap.

Fixing the Bug

Here's how it might look
(except for declaration and initialization).

```
if (0 != u[i] &&
    (num - 1) > non_zero++) {
    printf (".");
}
```

With this change, the code passes all 6,391 of my tests as well.

Fixing the Bug

Alternatively,
- compress zero regions out at the start,
- making the false equivalence true.

```
for (i = 0; 4 > i; i++) {
    if (0 != u[i]) {
        u[num++] = u[i];
    }
}
for (i = num; 4 > i; i++) {
    u[i] = 0;
}
```

