

# Delta Debugging

## Lecture 8

# Debugging sans Debuggers

---

- Debugging is more than debuggers
- In fact, debuggers are often the last resort
- Two other common problems:
  - Figuring out which program change caused a bug
  - Reducing a test case to a minimal example

# A Generic Algorithm

---

- How do people solve these problems?
- Binary search
  - Cut the test case in half
  - Iterate
- Brilliant idea: *Why not automate this?*

# Delta Debugging

---

- Find set of changes that cause a program to fail a test case
- Want to find a minimal set of changes that cause failure

# Example

---

- Printing the following file causes Mozilla to crash:

```
<td align=left valign=top>
<SELECT NAME="op sys" MULTIPLE SIZE=7>
<OPTION VALUE="All">All<OPTION VALUE="Windows 3.1">Windows
3.1<OPTION VALUE="Windows 95">Windows 95<OPTION VALUE="Windows
98">Windows 98<OPTION VALUE="Windows ME">Windows ME<OPTION
VALUE="Windows 2000">Windows 2000<OPTION VALUE="Windows
NT">Windows NT<OPTION VALUE="Mac System 7">Mac System 7<OPTION
VALUE="Mac System 7.5">Mac System 7.5<OPTION VALUE="Mac
System 7.6.1">Mac System 7.6.1<OPTION VALUE="Mac System 8.0">Mac
System 8.0<OPTION VALUE="Mac System 8.5">Mac System
8.5<OPTION VALUE="Mac System 8.6">Mac System 8.6<OPTION VALUE="Mac
System 9.x">Mac System 9.x<OPTION VALUE="MacOS X">MacOS
X<OPTION VALUE="Linux">Linux<OPTION VALUE="BSDI">BSDI<OPTION
VALUE="FreeBSD">FreeBSD<OPTION VALUE="NetBSD">NetBSD<OPTION
VALUE="OpenBSD">OpenBSD<OPTION VALUE="AIX">AIX<OPTION
```

---

```

VALUE="BeOS">BeOS<OPTION VALUE="HP-UX">HP-UX<OPTION
VALUE="IRIX">IRIX<OPTION VALUE="Neutrino">Neutrino<OPTION
VALUE="OpenVMS">OpenVMS<OPTION VALUE="OS/2">OS/2<OPTION
VALUE="OSF/1">OSF/1<OPTION VALUE="Solaris">Solaris<OPTION
VALUE="SunOS">SunOS<OPTION VALUE="other">other</SELECT></td>
<td align=left valign=top>
<SELECT NAME="priority" MULTIPLE SIZE=7>
<OPTION VALUE="--">--<OPTION VALUE="P1">P1<OPTION
VALUE="P2">P2<OPTION VALUE="P3">P3<OPTION VALUE="P4">P4<OPTION
VALUE="P5">P5</SELECT>
</td>
<td align=left valign=top>
<SELECT NAME="bug severity" MULTIPLE SIZE=7>
<OPTION VALUE="blocker">blocker<OPTION
VALUE="critical">critical<OPTION VALUE="major">major<OPTION
VALUE="normal">normal<OPTION VALUE="minor">minor<OPTION
VALUE="trivial">trivial<OPTION
VALUE="enhancement">enhancement</SELECT>
</tr>
</table>

```

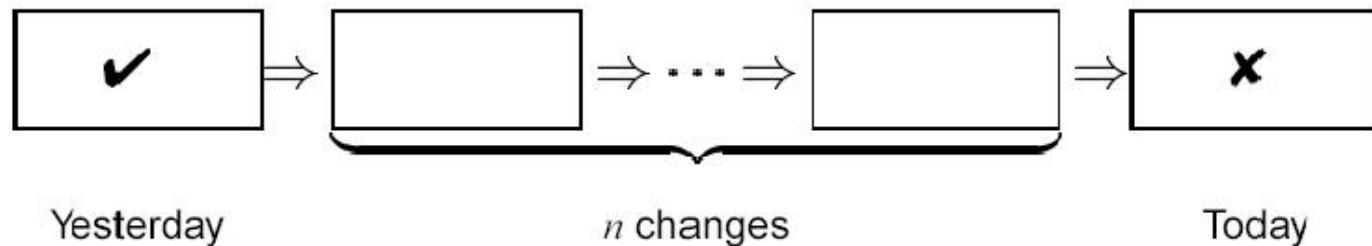
# Example

---

- Now looking at that file it is hard to figure out what the real cause of the failure is
- It would be very helpful in finding the error if we can simplify the input file and still generate the same failure

# Worked Yesterday, Not Today

---



- Yesterday, my program worked. Today, it does not. Why?
  - The new release 4.17 of GDB changed 178,000 lines
  - it no longer integrated properly with DDD (a graphical front-end)
  - How to isolate the change that caused the failure.

# GCC-2.95.2 Crash

---

```
double bug(double z[], int n)
{
    int i , j ;
    i = 0;
    for (j = 0; j < n; j++) {
        i = i + j + 1;
        z[i] = z[i] *(z[0]+1.0);
    }
    return z[n];
}
```

```
$ gcc-2.95.2 -O bug.c
gcc: Internal error:
program ccl got fatal
signal 11
$ _
```

- What's *causes* for GCC crashing?

# Delta Debugging Version I

---

- Assume
  - There is a set of changes  $C$
  - There is a single change that caused failure
  - Every set of changes is possible
    - Any subset produces a test case that either passes ✓ or fails ✗

# Algorithm for Version I

---

/\* invariant: P with changes  $c_1, \dots, c_n$  fails \*/

$DD(P, \{c_1, \dots, c_n\}) =$   
if  $n = 1$  return  $\{c_1\}$   
let  $P_1 = P \oplus \{c_1 \dots c_{n/2}\}$   
let  $P_2 = P \oplus \{c_{n/2+1} \dots c_n\}$   
if  $P_1 = \checkmark$   
    then  $DD(P, \{c_{n/2+1} \dots c_n\})$   
    else  $DD(P, \{c_1 \dots c_{n/2}\})$

*This is just binary search . . .*

# Extensions

---

- Let's get fancy. Assume:
- Any subset of changes may cause the bug
  - But no undetermined (?) tests, yet

- And the world is

- *Monotonic:*

$$P \oplus C = \times \quad \dagger \quad P \oplus (C \& C') \neq \checkmark$$

- *Unambiguous:*

$$P \oplus C = \times \quad \& \quad P \oplus C' = \times \quad \dagger \quad P \oplus (C \text{ } C') \neq \checkmark$$

- *Consistent*

$$P \oplus C \neq ?$$

# Scenarios

---

Try binary search:

- Divide changes  $C$  into  $C_1$  and  $C_2$
- If  $P \oplus C_1 = \times$ , recurse with  $C_1$
- If  $P \oplus C_2 = \times$ , recurse with  $C_2$

• Notes:

- At most one case can apply, by unambiguity
- By consistency, only other possibility is

$$P \oplus C_1 = \checkmark \text{ and } P \oplus C_2 = \checkmark$$

- What happens in this case?

# Interference

---

By monotonicity, if  $P \oplus C_1 = \checkmark$   $\wedge$   $P \oplus C_2 = \checkmark$

then no subset of  $C_1$  or  $C_2$  causes failure

So the failure must be a combination of elements from  
 $C_1$  and  $C_2$

This is called *interference*

# Handling Interference

---

- The cute trick:
  - Consider  $P \oplus C_1$ 
    - Find minimal  $D_2$  '  $C_2$  s.t.  $P \oplus C_1 \& D_2 = \times$
  - Consider  $P \oplus C_2$ 
    - Find minimal  $D_1$  '  $C_1$  s.t.  $P \oplus C_2 \& D_1 = \times$
  - Then by unambiguity
$$P \oplus ((C_1 \& D_2) \exists (C_2 \& D_1)) = P \oplus (D_1 \& D_2)$$
  - This is also minimal

# Example: 3 & 6 (of 8) Cause Failure

---

<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>7</i>	<i>8</i>	<i>result</i>
1	2	3	4					✓
				5	6	7	8	✓
1	2			5	6	7	8	✓
		3	4	5	6	7	8	x
		3		5	6	7	8	x
1	2	3	4	5	6			x
•	2	3	4	5				✓
1	2	3	4		6			x

*interference*

# Algorithm

---

/\* invariant: P with changes  $c_1, \dots, c_n$  fails \*/

DD( $P, \{c_1, \dots, c_n\}$ ) =

if  $n = 1$  return  $\{c_1\}$

$P_1 * P \oplus \{c_1 \dots c_{n/2}\}$

$P_2 * P \oplus \{c_{n/2+1} \dots c_n\}$

if  $P_1 = \times$  then DD( $P, \dots$ )

elseif  $P_2 = \times$  then DD( $P, \dots$ )

else DD( $P_2, \dots$ ) & DD( $P_1, \dots$ )

# Algorithm

---

/\* invariant: P with changes  $c_1, \dots, c_n$  fails \*/

DD( $P, \{c_1, \dots, c_n\}$ ) =

if  $n = 1$  return  $\{c_1\}$

$P_1 * P \oplus \{c_1 \dots c_{n/2}\}$

$P_2 * P \oplus \{c_{n/2+1} \dots c_n\}$

if  $P_1 = \times$  then DD( $P, \{c_1 \dots c_{n/2}\}$ )

elseif  $P_2 = \times$  then DD( $P, \{c_{n/2+1} \dots c_n\}$ )

else DD( $P_2, \{c_1 \dots c_{n/2}\}$ ) & DD( $P_1, \{c_{n/2+1} \dots c_n\}$ )

# Complexity

---

- If a single change induces the failure, then logarithmic
  - Why?
- Otherwise, linear
  - Assumes constant time per invocation
  - Is this realistic?

# Example

---

- Assume that we know that when Mozilla tries to print the following HTML input it crashes:

**<SELECT NAME="priority" MULTIPLE SIZE=7>**

- How can we go about simplifying this input?
  - Remove parts of the input and see if it still causes the program to crash
- For the above example assume that we remove characters from the input file

Bold parts remain in the input, the rest is removed



---

1	<b>&lt;SELECT NAME="priority" MULTIPLE SIZE=7&gt;</b>	F
2	<SELECT NAME="priority" <b>MULTIPLE SIZE=7</b> >	P
3	<b>&lt;SELECT NAME="priority"</b> MULTIPLE SIZE=7>	P
4	<SELECT NAME=" <b>priority"</b> <b>MULTIPLE SIZE=7</b> >	P
5	<b>&lt;SELECT NAME="priority" MULTIPLE SIZE=7&gt;</b>	F
6	<b>&lt;SELECT NAME="priority"</b> MULTIPLE <b>SIZE=7</b> >	F
7	<b>&lt;SELECT NAME="priority"</b> MULTIPLE SIZE=7>	P
8	<SELECT <b>NAME="priority" MULTIPLE SIZE=7</b> >	P
9	<b>&lt;SELECT NAME="priority"</b> MULTIPLE <b>SIZE=7</b> >	P
10	<b>&lt;SELECT NAME="priority"</b> MULTIPLE SIZE=7>	F
11	<b>&lt;SELECT NAME="priority"</b> MULTIPLE SIZE=7>	P
12	<SELECT <b>NAME="priority" MULTIPLE SIZE=7</b> >	P
13	<b>&lt;SELECT NAME="priority"</b> MULTIPLE <b>SIZE=7</b> >	P

F means input caused failure  
P means input did not cause failure (input passed)

---

14	<SELECT NAME="priority" MULTIPLE SIZE=7>	P
15	<SELECT NAME="priority" MULTIPLE SIZE=7>	P
16	<SELECT NAME="priority" MULTIPLE SIZE=7>	F
17	<SELECT NAME="priority" MULTIPLE SIZE=7>	F
18	<SELECT NAME="priority" MULTIPLE SIZE=7>	F
19	<SELECT NAME="priority" MULTIPLE SIZE=7>	P
20	<SELECT NAME="priority" MULTIPLE SIZE=7>	P
21	<SELECT NAME="priority" MULTIPLE SIZE=7>	P
22	<SELECT NAME="priority" MULTIPLE SIZE=7>	P
23	<SELECT NAME="priority" MULTIPLE SIZE=7>	P
24	<SELECT NAME="priority" MULTIPLE SIZE=7>	P
25	<SELECT NAME="priority" MULTIPLE SIZE=7>	P
26	<SELECT NAME="priority" MULTIPLE SIZE=7>	F

# Example

---

- After 26 tries we found that printing an HTML file which consists of:

**<SELECT>**

causes Mozilla to crash

- Delta debugging technique automates this approach of repeated trials for reducing the input

# Delta Debugging ++: Revisit the Assumptions

---

- All three assumptions are suspect

- *Monotonic:*

$$P \oplus C = \times \quad \ddagger \quad P \oplus (C \& C') \neq \checkmark$$

- *Unambiguous:*

$$P \oplus C = \times \quad \text{K} \quad P \oplus C' = \times \quad \ddagger \quad P \oplus (C \text{ } C') \neq \checkmark$$

- *Consistent*

$$P \oplus C \neq ?$$

# Delta Debugging ++

---

- Drop all of the assumptions
- What can we do?
- Problem formulation
  - Find a set of changes that cause the problem, but removing any change causes the problem to go away*
- This is *1-minimality*

# Model

---

- A test either
  - Passes ✓
  - Fails ✗
  - Is unresolved ?

# Naïve Algorithm

---

- To find a 1-minimal subset of  $C$ , simply
- Remove one element  $c$  from  $C$
- If  $C - \{c\} = x$ , recurse with smaller set
- If  $C - \{c\} \neq x$ ,  $C$  is 1-minimal

# Analysis

---

- In the worst case,
  - We remove one element from the set per iteration
  - After trying every other element
- Work is potentially
$$N + (N-1) + (N-2) + \dots$$
- This is  $O(N^2)$

# Work Smarter, Not Harder

---

- We can often do better
- Silly to start out removing 1 element at a time
  - Try dividing change set in 2 initially
  - Increase # of subsets if we can't make progress
  - If we get lucky, search will converge quickly

# Algorithm

---

DD(P, C) =

split C into  $C_1 \sqcup \dots \sqcup C_n$  (initially  $n=2$ )

if  $P \oplus C_i = \times$  then DD(P,  $C_i$ )

if  $P \oplus \sqcup C_i = \times$  then DD(P,  $C_1 \& \dots \& C_{i-1} \& C_{i+1} \& \dots \& C_n$ )

else double n and try again

# Analysis

---

- Worst case is still quadratic
- Subdivide until each set is of size 1
  - Reduced to the naïve algorithm
- Good news
  - For single, monotone failure, converges in  $\log N$
  - Binary search again

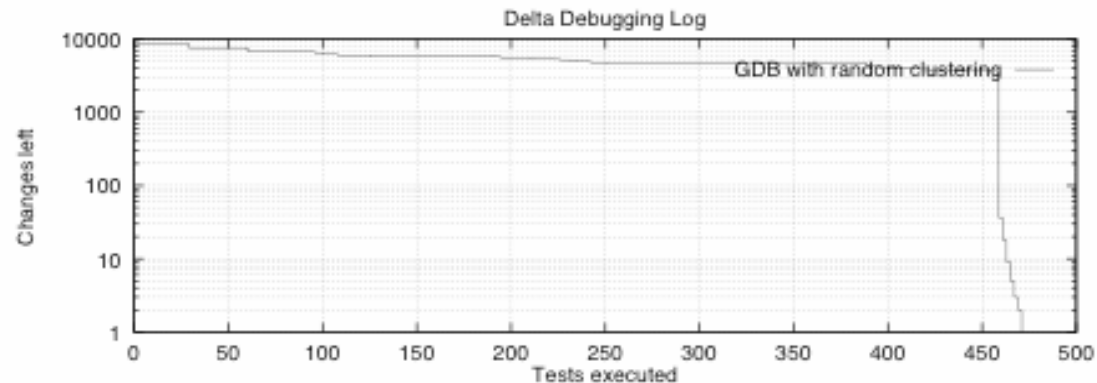
# Case Studies

---

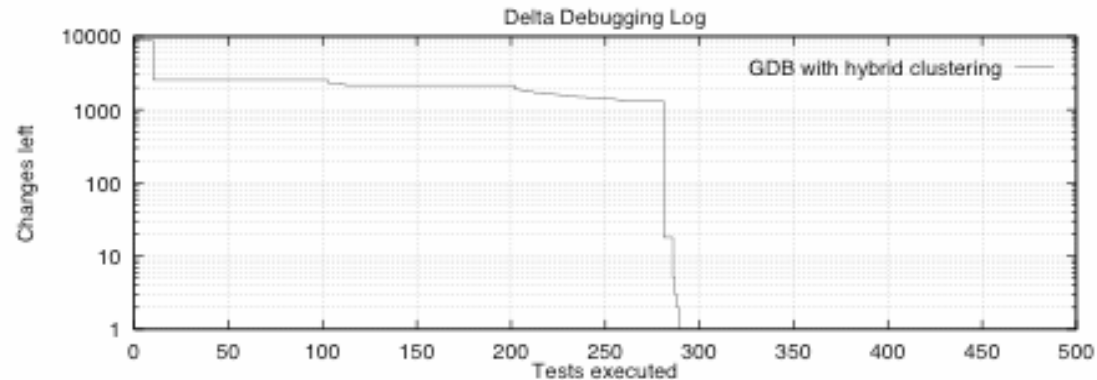
- Many in the papers
    - And convincing, too
  - Isolating failure in modified gdb
    - 178,000 modified source lines
    - Symptom was that program simply crashed
    - What was the bug? Changing
      - "Set arguments to give ..."
- to
- "Set argument **list** to give ..."

# Second Case Study: GDB 4.17 does not work with DDD

(8721 changes from a 178,000-line DIFF output; 370s/test)



with random clustering



with file clustering + failure resolution loop

# Failure Inducing Differences: Example

---

- Changing the input program for *GCC* from the one on the left to the one on the right removes the failure

This input causes failure

```
#define SIZE 20
double mult(double z[], int n)
{
    int i , j ;
    i = 0;
    for (j = 0; j < n; j++) {
        i = i + j + 1;
        z[i] = z[i] *(z[0]+1.0);
    }
    return z[n];
}
```

This input does not cause failure

```
#define SIZE 20
double mult(double z[], int n)
{
    int i , j ;
    i = 0;
    for (j = 0; j < n; j++) {
        i + j + 1;
        z[i] = z[i] *(z[0]+1.0);
    }
    return z[n];
}
```

Modified statement is shown in box

# The Importance of Changes

---

- Basic to delta debugging is a *change*
  - We must be able to express the difference between the good and bad examples as a set of changes
- But notion of change is semantic
  - Not easy to capture in a general way in a tool
- And notion of change is algorithmic
  - Poor notion of change  $\dagger$  many unresolved tests
  - Performance goes from linear (or sub-linear) to quadratic

# Notion of Change

---

- We can see this in the experiments
  - Some gdb experiments took 48 hours
  - Improvements came from improving notion of changes
- Also important to exploit correlations between changes
  - Some subsets of changes require other changes
  - Again, can affect asymptotic performance

# Opinion

---

- Delta Debugging is a technique, not a tool
- Bad News:
  - Probably must be reimplemented for each significant system
  - To exploit knowledge of changes
- Good News:
  - Relatively simple algorithm, significant payoff
  - It's worth reimplementing