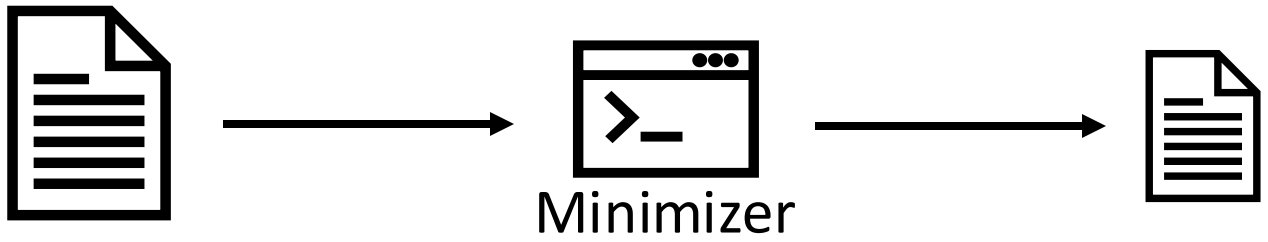
WDD: Weighted Delta Debugging

Xintong Zhou, Zhenyang Xu, Mengxiao Zhang, Yongqiang Tian, Chengnian Sun

x27zhou@uwaterloo.ca
School of Computer Science
University of Waterloo

Background: Test Input Minimization



Test input I
triggers a bug

Minimized test input I_{min}
triggers the **same** bug

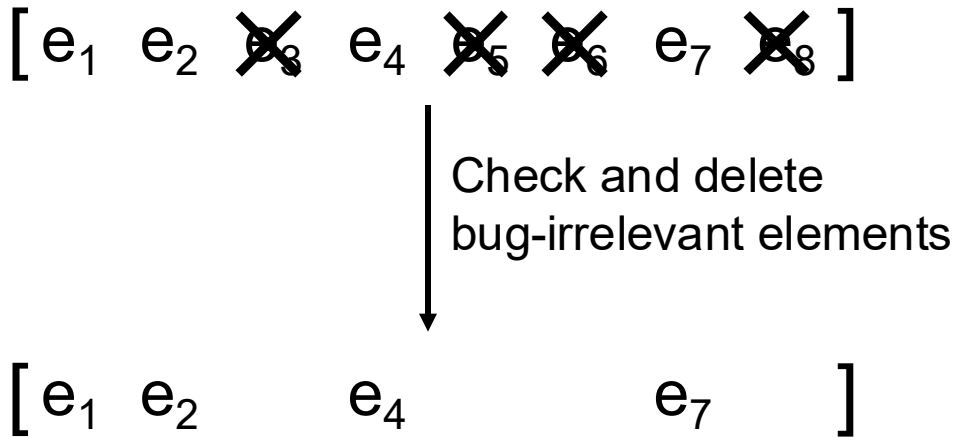
Facilitate software debugging

Background: Minimization Approaches

List-based: Delta Debugging

Represent the test input in a list of elements

- e.g. tokens, lines

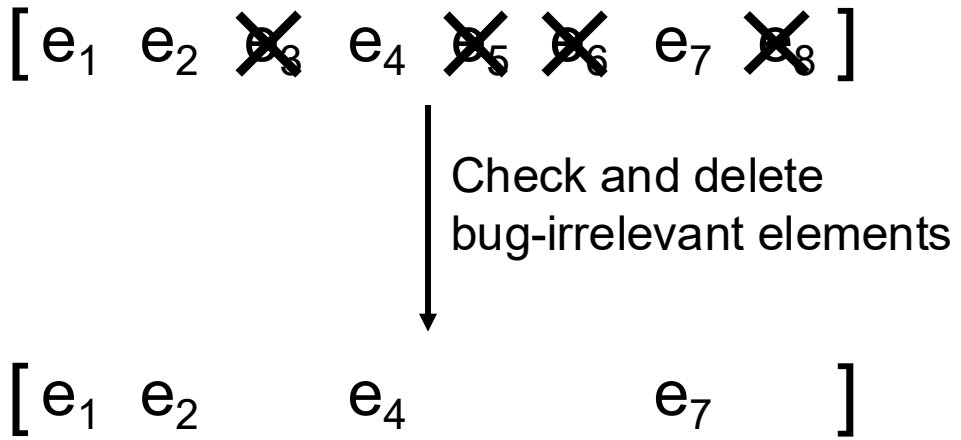


Background: Minimization Approaches

List-based: Delta Debugging

Represent the test input in a list of elements

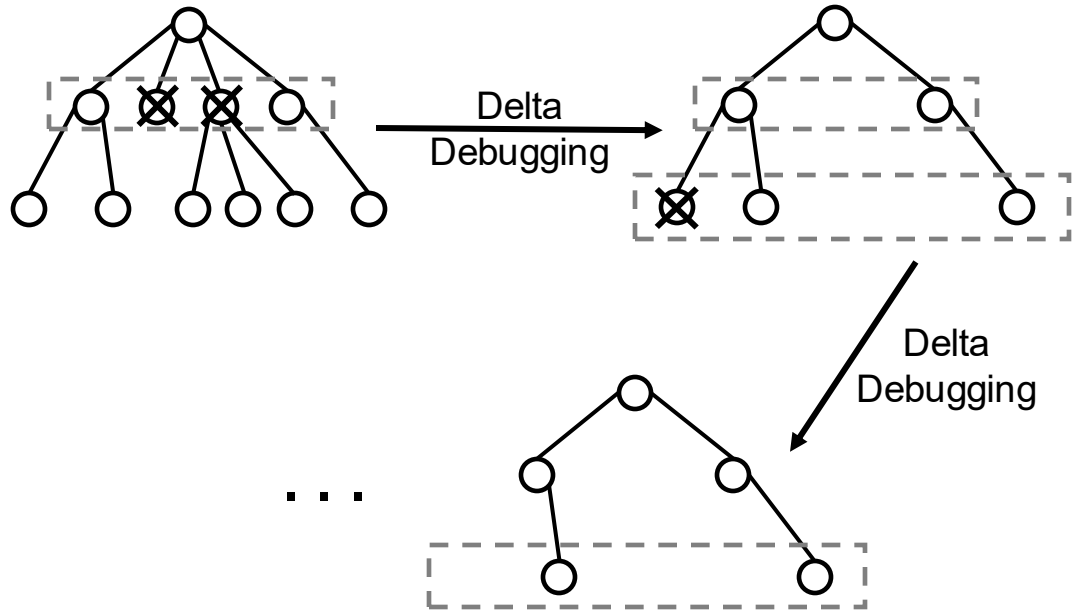
- e.g. tokens, lines



Tree-based: HDD, Perses …

Represent the input as a tree (e.g. parse tree or AST)

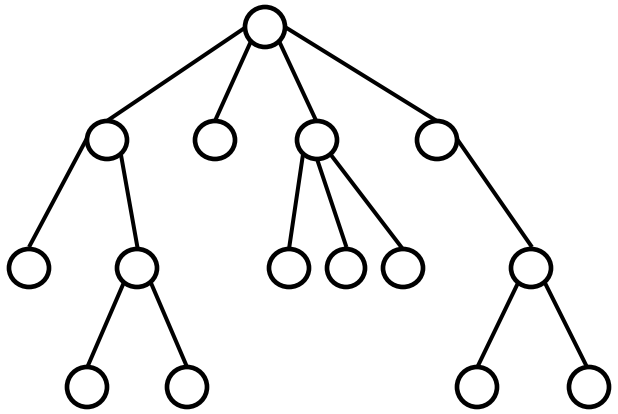
Use Delta Debugging to prune the list of tree nodes on the same level



Limitations of Delta Debugging

In practice:

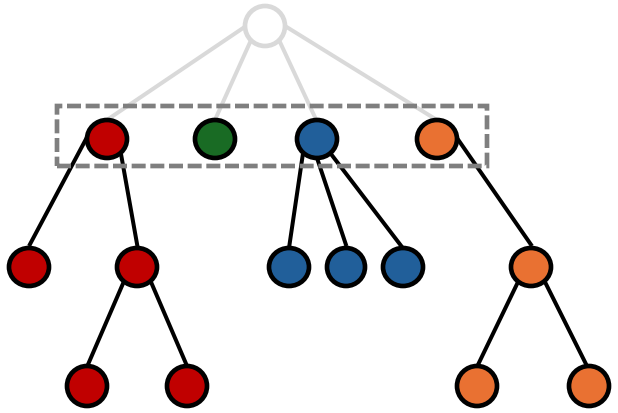
- Each element represent a different fragment of original input
- Elements are different in size
- Larger elements are more likely to contain the bug-inducing code thus less likely to be deleted



Limitations of Delta Debugging

In practice:

- Each element represent a different fragment of original input
- Elements are different in size
- Larger elements are more likely to contain the bug-inducing code thus less likely to be deleted



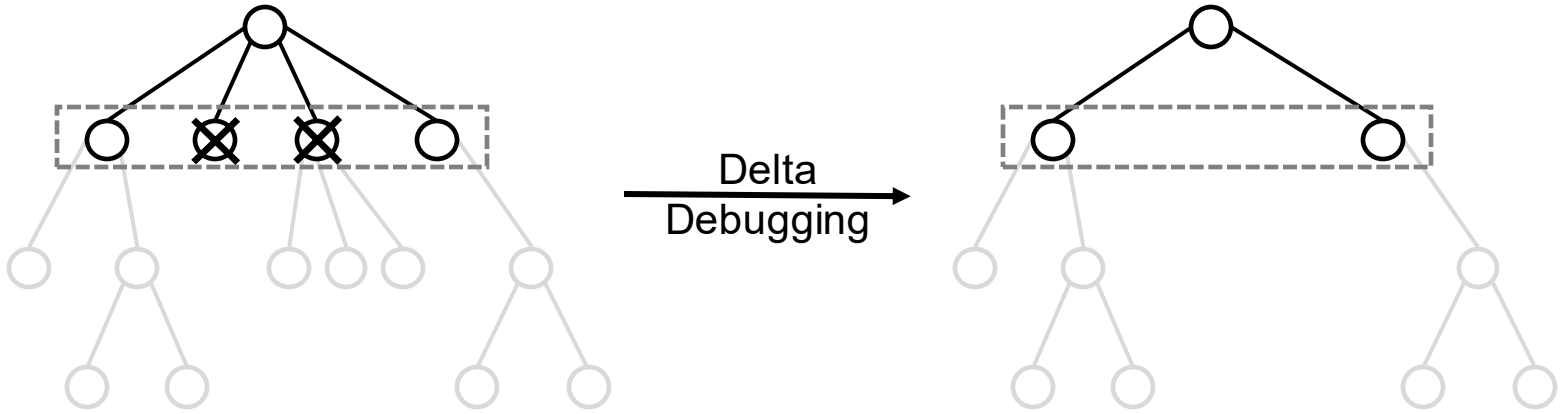
Limitations of Delta Debugging

In practice:

- Each element represent a different fragment of original input
- Elements are different in size
- Larger elements are more likely to contain the bug-inducing code thus less likely to be deleted

Delta Debugging:

- Overlook the different representations of elements
- Treat all elements uniformly
- Suboptimal efficiency, even effectiveness



Example

GCC bug 71626[1]:

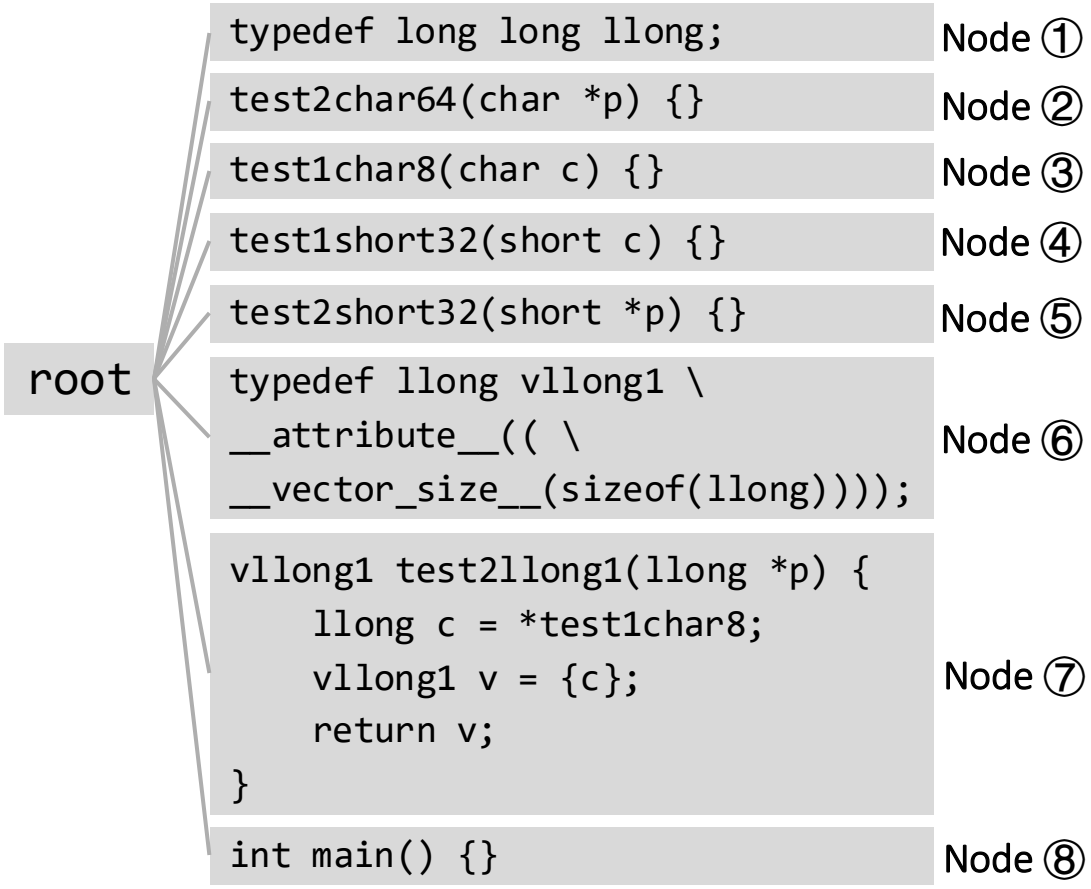
```
typedef long long llong;
test2char64(char *p) {}
test1char8(char c) {}
test1short32(short c) {}
test2short32(short *p) {}
typedef llong vllong1 \
__attribute__(( \
__vector_size__(sizeof(llong))));
vllong1 test2llong1(llong *p) {
    llong c = *test1char8;
    vllong1 v = {c};
    return v;
}
int main() {}
```

```
>> gcc -O0 test.c ; ./a.out
>> gcc -O3 test.c ; ./a.out
internal compiler error
```

Target: minimize the program to a minimal one that still triggers the bug

[1] https://gcc.gnu.org/bugzilla/show_bug.cgi?id=71626

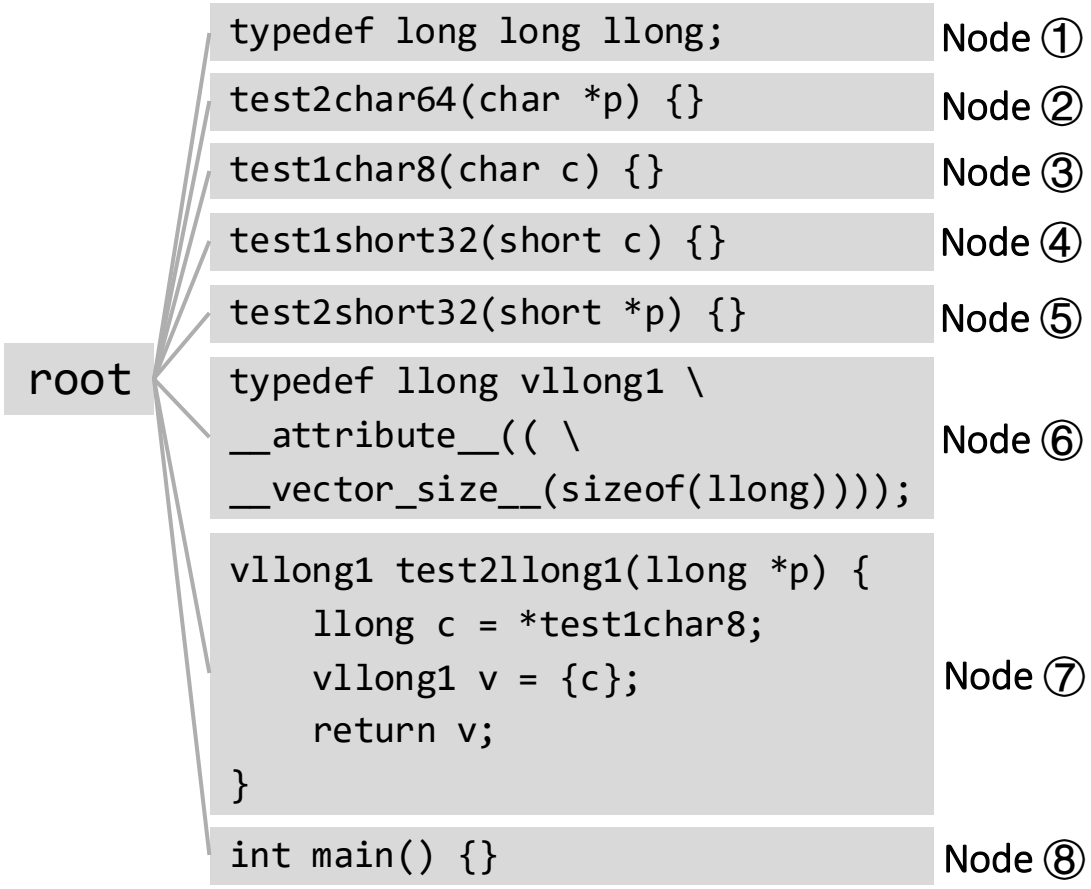
Example



Target: minimize the program to a minimal one that still triggers the bug

Approach: tree-based minimizer (HDD or Perses)

Example

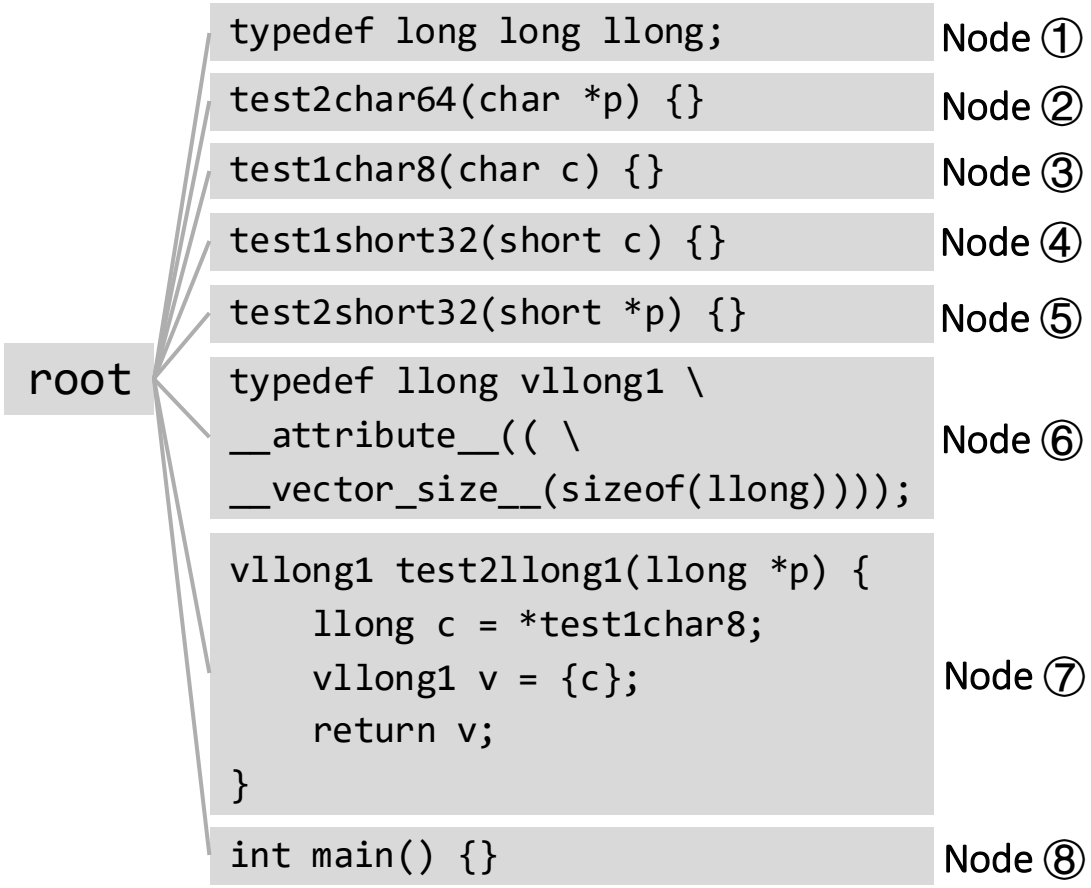


Target: minimize the program to a minimal one that still triggers the bug

Approach: tree-based minimizer (HDD or Perses)

- First step: minimize the list of tree nodes with Delta Debugging

Example



Target: minimize the program to a minimal one that still triggers the bug

Approach: tree-based minimizer (HDD or Perses)

- First step: minimize the list of tree nodes with Delta Debugging
- Delta Debugging algorithms:
  - Minimizing Delta Debugging (ddmin)
  - Probabilistic Delta Debugging (ProbDD)

Example

root

```
typedef long long llong;
```
Node ①

```
test2char64(char *p) {}
```
Node ②

```
test1char8(char c) {}
```
Node ③

```
test1short32(short c) {}
```
Node ④

```
test2short32(short *p) {}
```
Node ⑤

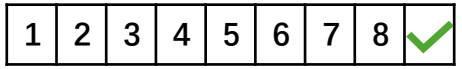
```
typedef llong vllong1 \
__attribute__(( \
__vector_size__(sizeof(llong))));
```
Node ⑥

```
vllong1 test2llong1(llong *p) {
    llong c = *test1char8;
    vllong1 v = {c};
    return v;
}
```
Node ⑦

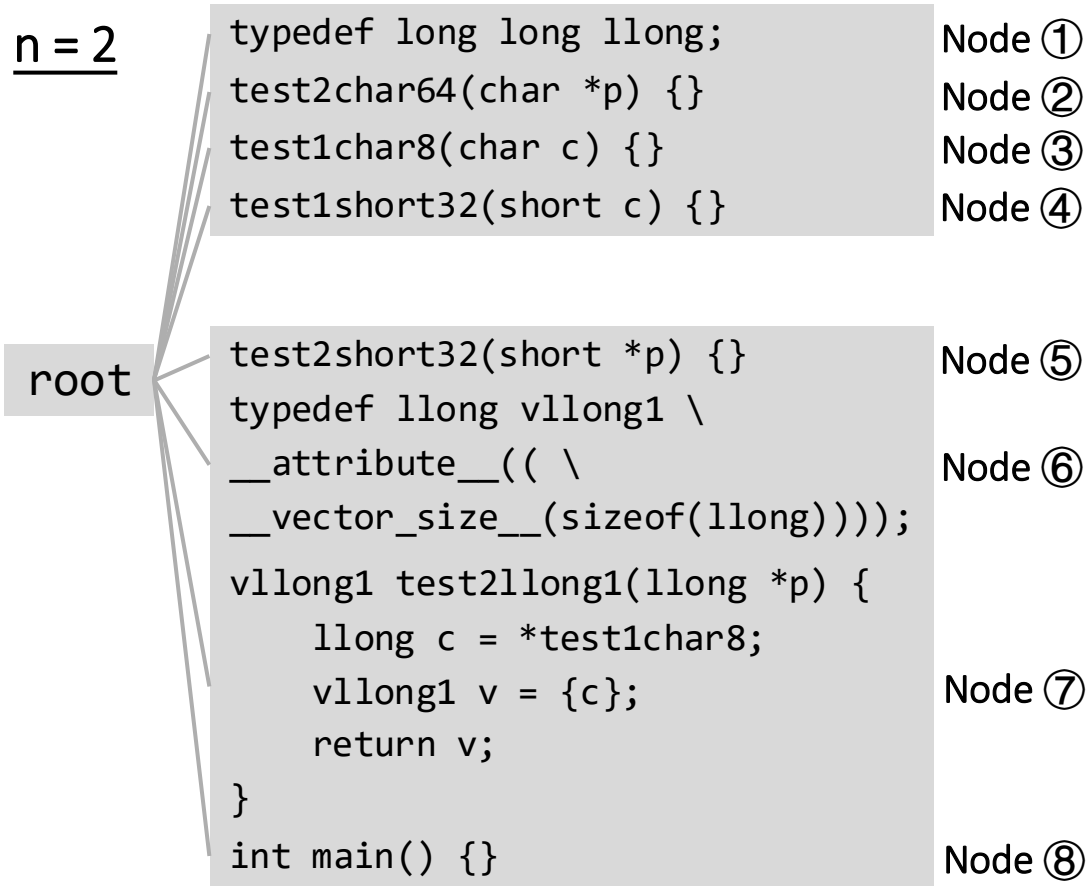
```
int main() {}
```
Node ⑧

Minimizing Delta Debugging (ddmin):

- Binary-search style minimization
- Divide the list evenly by length

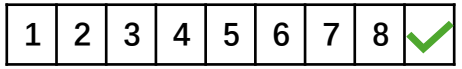


Example



Minimizing Delta Debugging (ddmin):

- Binary-search style minimization
- Divide the list evenly by length



Example

n = 2

root

```
typedef long long llong;
test2char64(char *p) {}
test1char8(char c) {}
test1short32(short c) {}
```

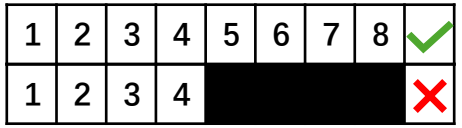
Node ①
Node ②
Node ③
Node ④

```
test2short32(short *p) {}
typedef llong vllong1 \
__attribute__(( \
__vector_size__(sizeof(llong))));
vllong1 test2llong1(llong *p) {
    llong c = *test1char8;
    vllong1 v = {c};
    return v;
}
int main() {}
```

Node ⑤
Node ⑥
Node ⑦
Node ⑧

Minimizing Delta Debugging (ddmin):

- Binary-search style minimization
- Divide the list evenly by length



Example

n = 2

```
typedef long long llong;
test2char64(char *p) {}
test1char8(char c) {}
test1short32(short c) {}
```

Node ①
Node ②
Node ③
Node ④

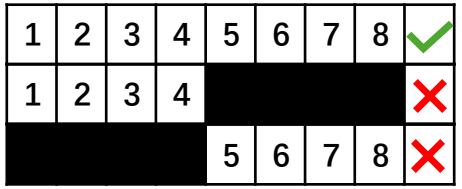
root

```
test2short32(short *p) {}
typedef llong vllong1 \
__attribute__(( \
__vector_size__(sizeof(llong))));
vllong1 test2llong1(llong *p) {
    llong c = *test1char8;
    vllong1 v = {c};
    return v;
}
int main() {}
```

Node ⑤
Node ⑥
Node ⑦
Node ⑧

Minimizing Delta Debugging (ddmin):

- Binary-search style minimization
- Divide the list evenly by length



Example

n = 2

root

```
typedef long long llong;
test2char64(char *p) {}
test1char8(char c) {}
test1short32(short c) {}
```

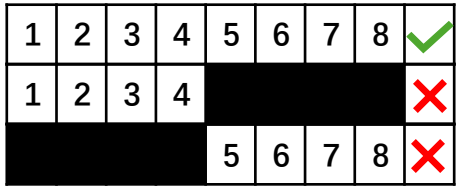
Node ①
Node ②
Node ③
Node ④

```
test2short32(short *p) {}
typedef llong vllong1 \
__attribute__(( \
__vector_size__(sizeof(llong))));
vllong1 test2llong1(llong *p) {
    llong c = *test1char8;
    vllong1 v = {c};
    return v;
}
int main() {}
```

Node ⑤
Node ⑥
Node ⑦
Node ⑧

Minimizing Delta Debugging (ddmin):

- Binary-search style minimization
- Divide the list evenly by length



Example

n = 2

n = 4

root

```
typedef long long llong;
test2char64(char *p) {}
```
Node ①
Node ②

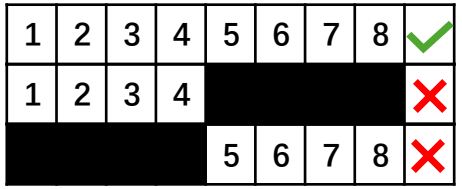
```
test1char8(char c) {}
test1short32(short c) {}
```
Node ③
Node ④

```
test2short32(short *p) {}
typedef llong vllong1 \
__attribute__(( \
__vector_size__(sizeof(llong))));
```
Node ⑤
Node ⑥

```
vllong1 test2llong1(llong *p) {
    llong c = *test1char8;
    vllong1 v = {c};
    return v;
}
int main() {}
```
Node ⑦
Node ⑧

Minimizing Delta Debugging (ddmin):

- Binary-search style minimization
- Divide the list evenly by length



Example

n = 2

n = 4

root

```
typedef long long llong;
test2char64(char *p) {}
```
Node ①
Node ②

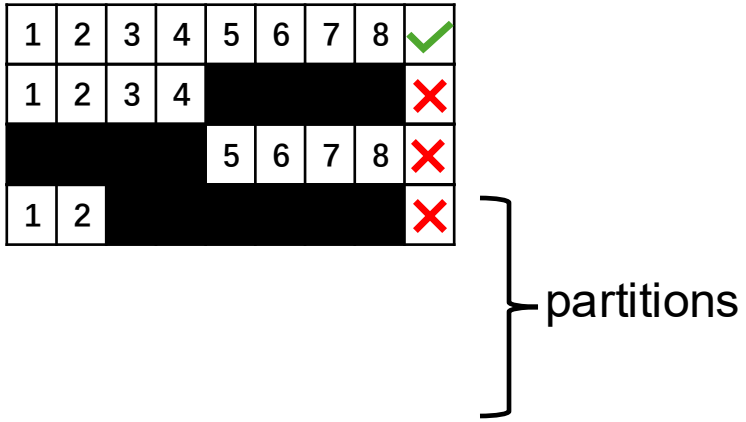
```
test1char8(char c) {}
test1short32(short c) {}
```
Node ③
Node ④

```
test2short32(short *p) {}
typedef llong vllong1 \
__attribute__(( \
__vector_size__(sizeof(llong))));
```
Node ⑤
Node ⑥

```
vllong1 test2llong1(llong *p) {
    llong c = *test1char8;
    vllong1 v = {c};
    return v;
}
int main() {}
```
Node ⑦
Node ⑧

Minimizing Delta Debugging (ddmin):

- Binary-search style minimization
- Divide the list evenly by length



Example

n = 2

n = 4

root

```
typedef long long llong;
test2char64(char *p) {}
```
Node ①
Node ②

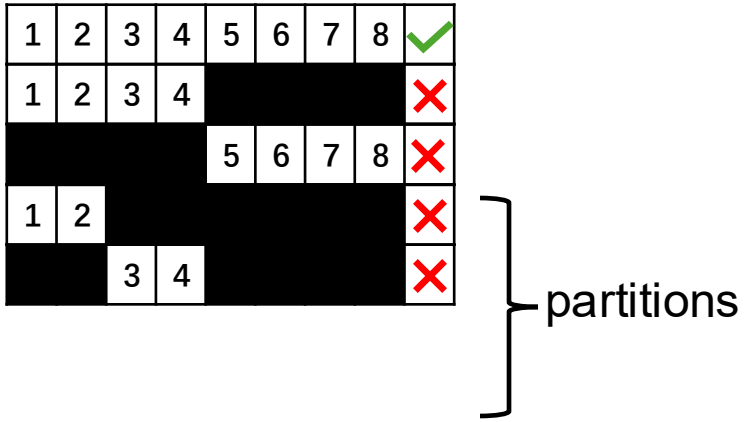
```
test1char8(char c) {}
test1short32(short c) {}
```
Node ③
Node ④

```
test2short32(short *p) {}
typedef llong vllong1 \
__attribute__(( \
__vector_size__(sizeof(llong))));
```
Node ⑤
Node ⑥

```
vllong1 test2llong1(llong *p) {
    llong c = *test1char8;
    vllong1 v = {c};
    return v;
}
int main() {}
```
Node ⑦
Node ⑧

Minimizing Delta Debugging (ddmin):

- Binary-search style minimization
- Divide the list evenly by length



Example

n = 2

n = 4

root

```
typedef long long llong;
test2char64(char *p) {}
```
Node ①
Node ②

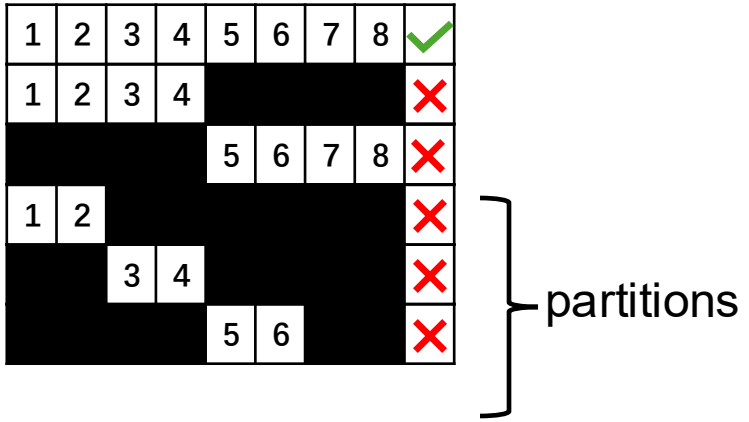
```
test1char8(char c) {}
test1short32(short c) {}
```
Node ③
Node ④

```
test2short32(short *p) {}
typedef llong vllong1 \
__attribute__(( \
__vector_size__(sizeof(llong))));
```
Node ⑤
Node ⑥

```
vllong1 test2llong1(llong *p) {
    llong c = *test1char8;
    vllong1 v = {c};
    return v;
}
int main() {}
```
Node ⑦
Node ⑧

Minimizing Delta Debugging (ddmin):

- Binary-search style minimization
- Divide the list evenly by length



Example

n = 2

n = 4

root

```
typedef long long llong;
test2char64(char *p) {}
```
Node ①
Node ②

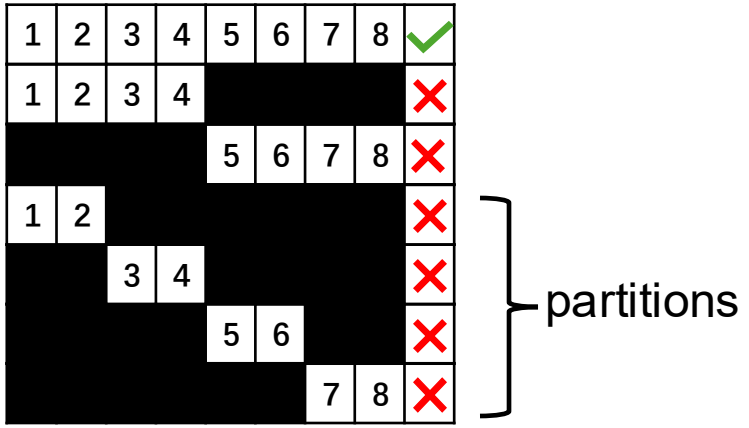
```
test1char8(char c) {}
test1short32(short c) {}
```
Node ③
Node ④

```
test2short32(short *p) {}
typedef llong vllong1 \
__attribute__(( \
__vector_size__(sizeof(llong))));
```
Node ⑤
Node ⑥

```
vllong1 test2llong1(llong *p) {
    llong c = *test1char8;
    vllong1 v = {c};
    return v;
}
int main() {}
```
Node ⑦
Node ⑧

Minimizing Delta Debugging (ddmin):

- Binary-search style minimization
- Divide the list evenly by length



Example

n = 2

n = 4

root

```
typedef long long llong;
test2char64(char *p) {}
```
Node ①
Node ②

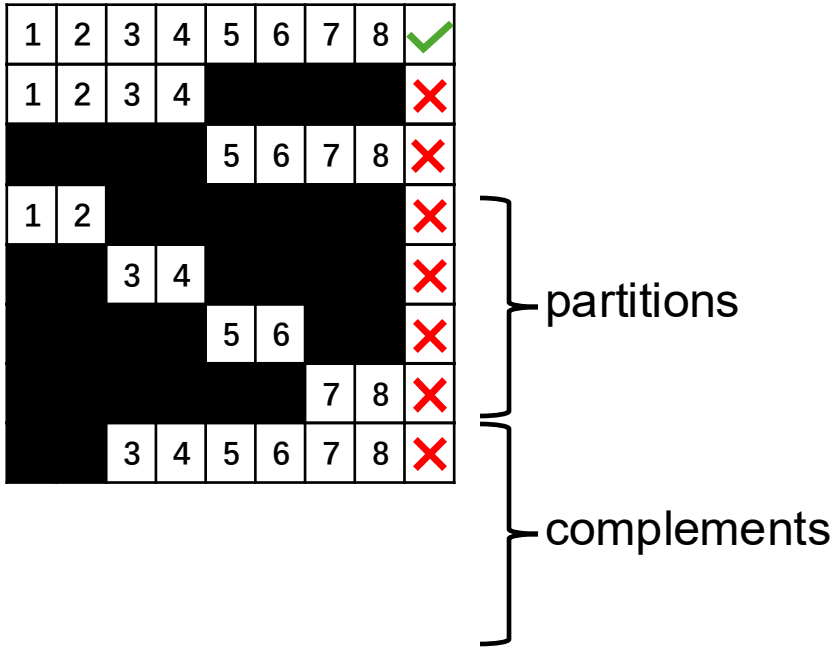
```
test1char8(char c) {}
test1short32(short c) {}
```
Node ③
Node ④

```
test2short32(short *p) {}
typedef llong vllong1 \
__attribute__(( \
__vector_size__(sizeof(llong))));
```
Node ⑤
Node ⑥

```
vllong1 test2llong1(llong *p) {
    llong c = *test1char8;
    vllong1 v = {c};
    return v;
}
int main() {}
```
Node ⑦
Node ⑧

Minimizing Delta Debugging (ddmin):

- Binary-search style minimization
- Divide the list evenly by length



Example

n = 2

n = 4

root

```
typedef long long llong;
test2char64(char *p) {}
```
Node ①
Node ②

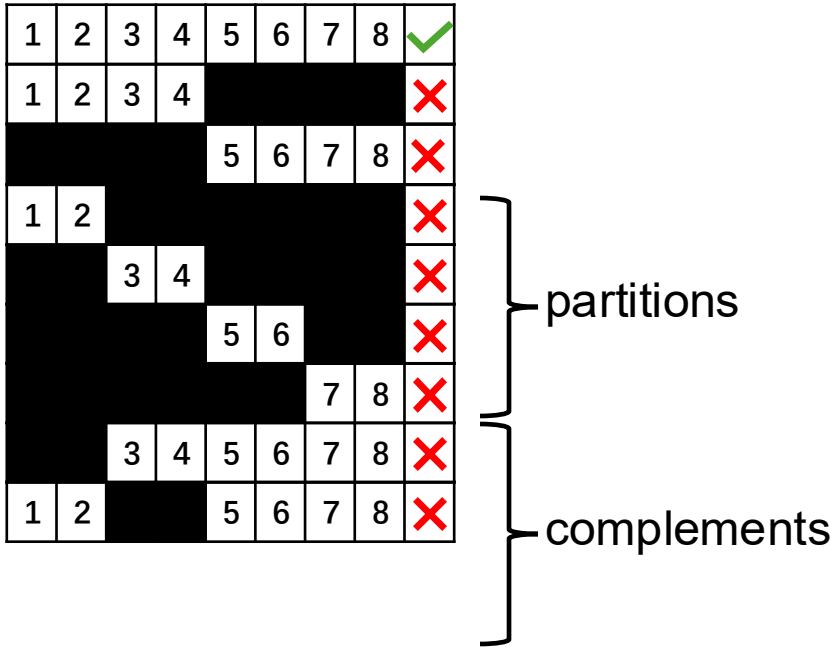
```
test1char8(char c) {}
test1short32(short c) {}
```
Node ③
Node ④

```
test2short32(short *p) {}
typedef llong vllong1 \
__attribute__(( \
__vector_size__(sizeof(llong))));
```
Node ⑤
Node ⑥

```
vllong1 test2llong1(llong *p) {
    llong c = *test1char8;
    vllong1 v = {c};
    return v;
}
int main() {}
```
Node ⑦
Node ⑧

Minimizing Delta Debugging (ddmin):

- Binary-search style minimization
- Divide the list evenly by length



Example

n = 2

n = 4

root

```
typedef long long llong;
test2char64(char *p) {}
```
Node ①
Node ②

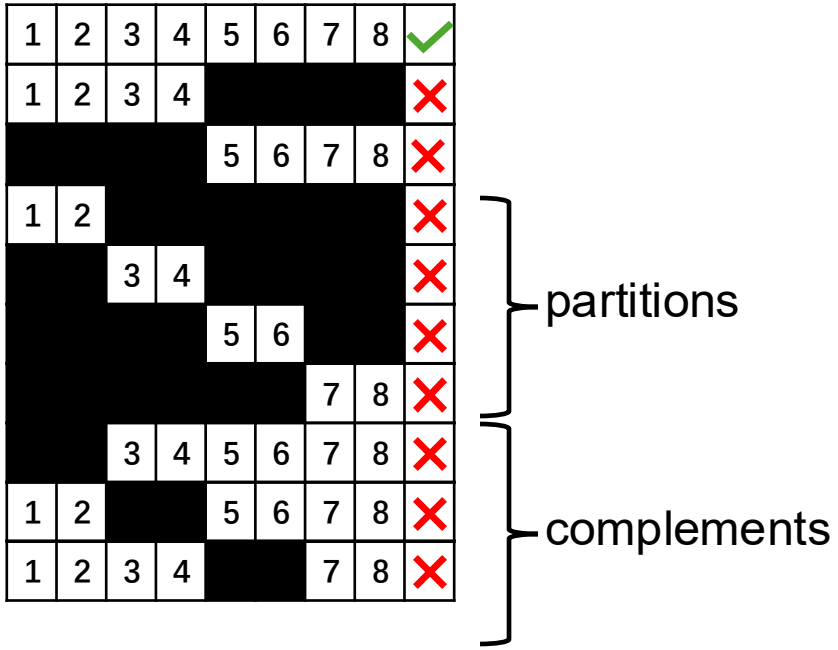
```
test1char8(char c) {}
test1short32(short c) {}
```
Node ③
Node ④

```
test2short32(short *p) {}
typedef llong vllong1 \
__attribute__(( \
__vector_size__(sizeof(llong))));
```
Node ⑤
Node ⑥

```
vllong1 test2llong1(llong *p) {
    llong c = *test1char8;
    vllong1 v = {c};
    return v;
}
int main() {}
```
Node ⑦
Node ⑧

Minimizing Delta Debugging (ddmin):

- Binary-search style minimization
- Divide the list evenly by length



Example

n = 2

n = 4

root

```
typedef long long llong;
test2char64(char *p) {}
```
Node ①
Node ②

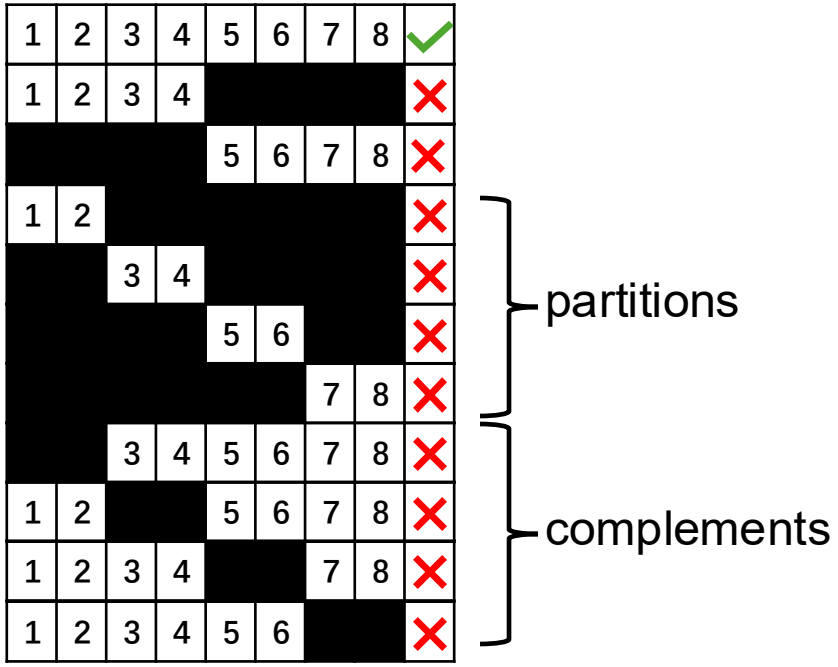
```
test1char8(char c) {}
test1short32(short c) {}
```
Node ③
Node ④

```
test2short32(short *p) {}
typedef llong vllong1 \
__attribute__(( \
__vector_size__(sizeof(llong))));
```
Node ⑤
Node ⑥

```
vllong1 test2llong1(llong *p) {
    llong c = *test1char8;
    vllong1 v = {c};
    return v;
}
int main() {}
```
Node ⑦
Node ⑧

Minimizing Delta Debugging (ddmin):

- Binary-search style minimization
- Divide the list evenly by length



Example

n = 2

n = 4

root

```
typedef long long llong;
test2char64(char *p) {}
```
Node ①
Node ②

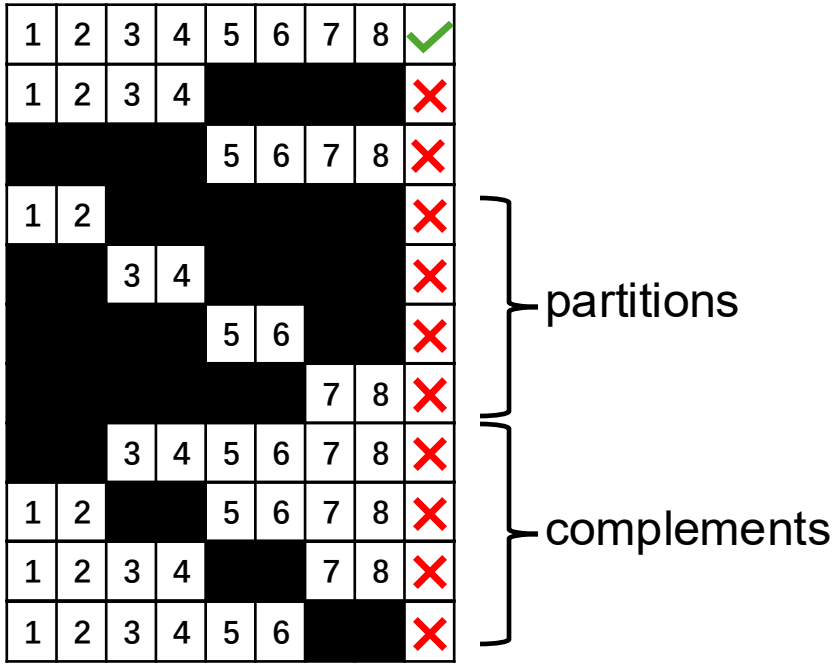
```
test1char8(char c) {}
test1short32(short c) {}
```
Node ③
Node ④

```
test2short32(short *p) {}
typedef llong vllong1 \
__attribute__(( \
__vector_size__(sizeof(llong))));
```
Node ⑤
Node ⑥

```
vllong1 test2llong1(llong *p) {
    llong c = *test1char8;
    vllong1 v = {c};
    return v;
}
int main() {}
```
Node ⑦
Node ⑧

Minimizing Delta Debugging (ddmin):

- Binary-search style minimization
- Divide the list evenly by length



Example

n = 2
n = 4
n = 8

root

```
typedef long long llong;
```
Node ①

```
test2char64(char *p) {}
```
Node ②

```
test1char8(char c) {}
```
Node ③

```
test1short32(short c) {}
```
Node ④

```
test2short32(short *p) {}
```
Node ⑤

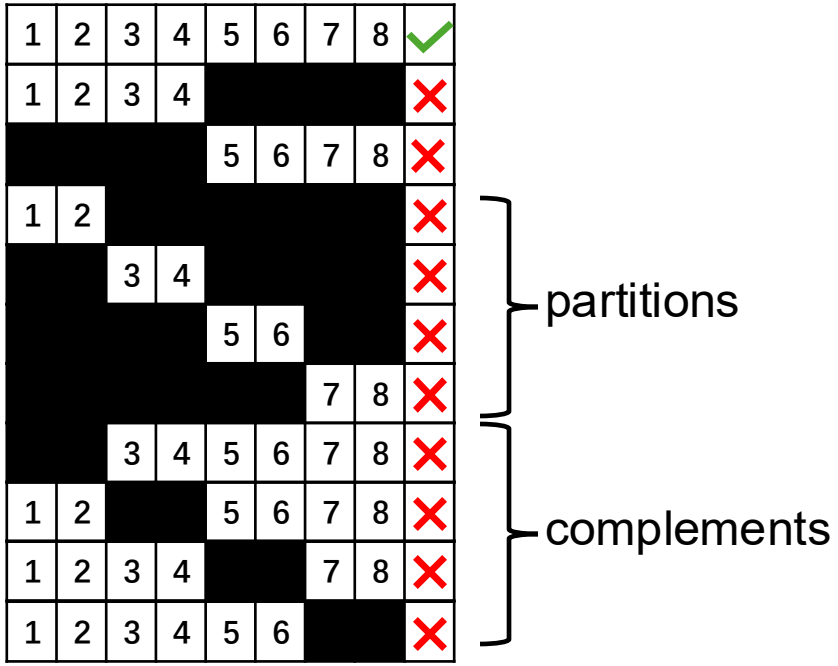
```
typedef llong vllong1 \
__attribute__(( \
__vector_size__(sizeof(llong))));
```
Node ⑥

```
vllong1 test2llong1(llong *p) {
    llong c = *test1char8;
    vllong1 v = {c};
    return v;
}
```
Node ⑦

```
int main() {}
```
Node ⑧

Minimizing Delta Debugging (ddmin):

- Binary-search style minimization
- Divide the list evenly by length



Example

n = 2
n = 4
n = 8

root

```
typedef long long llong;
```
Node ①

```
test1char8(char c) {}
```
Node ③

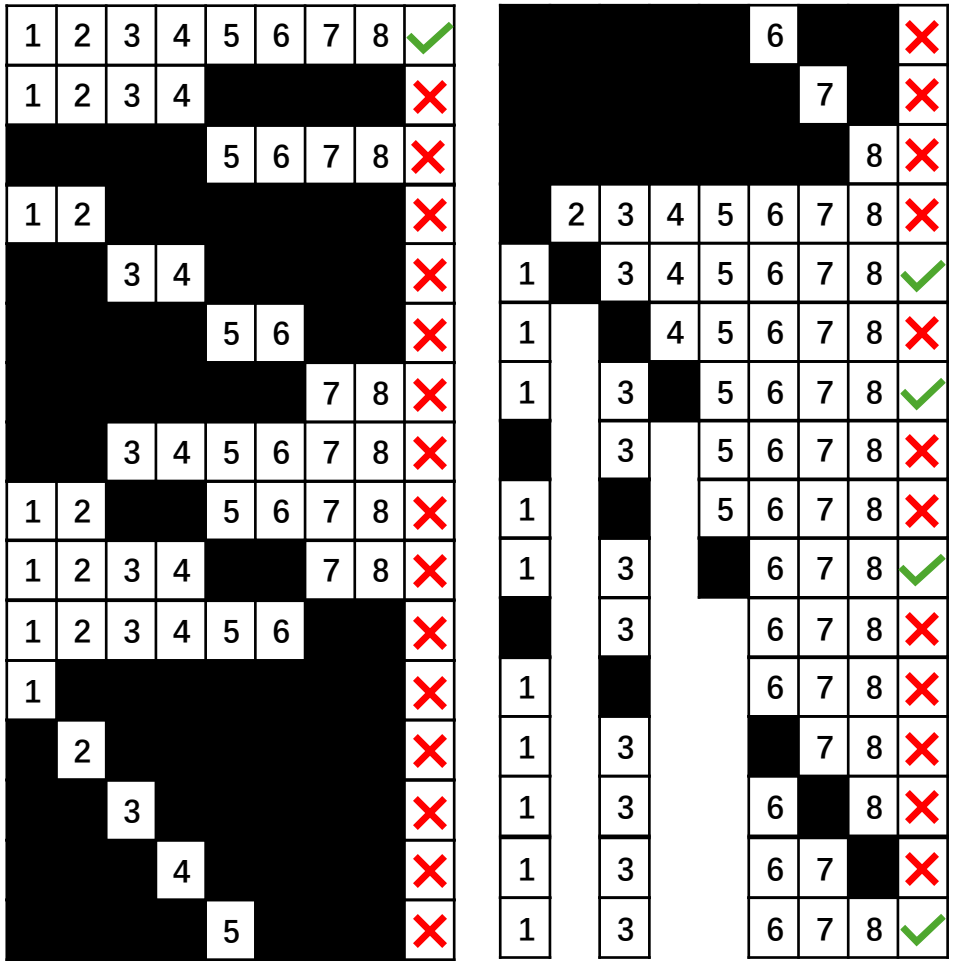
```
typedef llong vllong1 \
__attribute__(( \
__vector_size__(sizeof(llong))));
```
Node ⑥

```
vllong1 test2llong1(llong *p) {
    llong c = *test1char8;
    vllong1 v = {c};
    return v;
}
```
Node ⑦

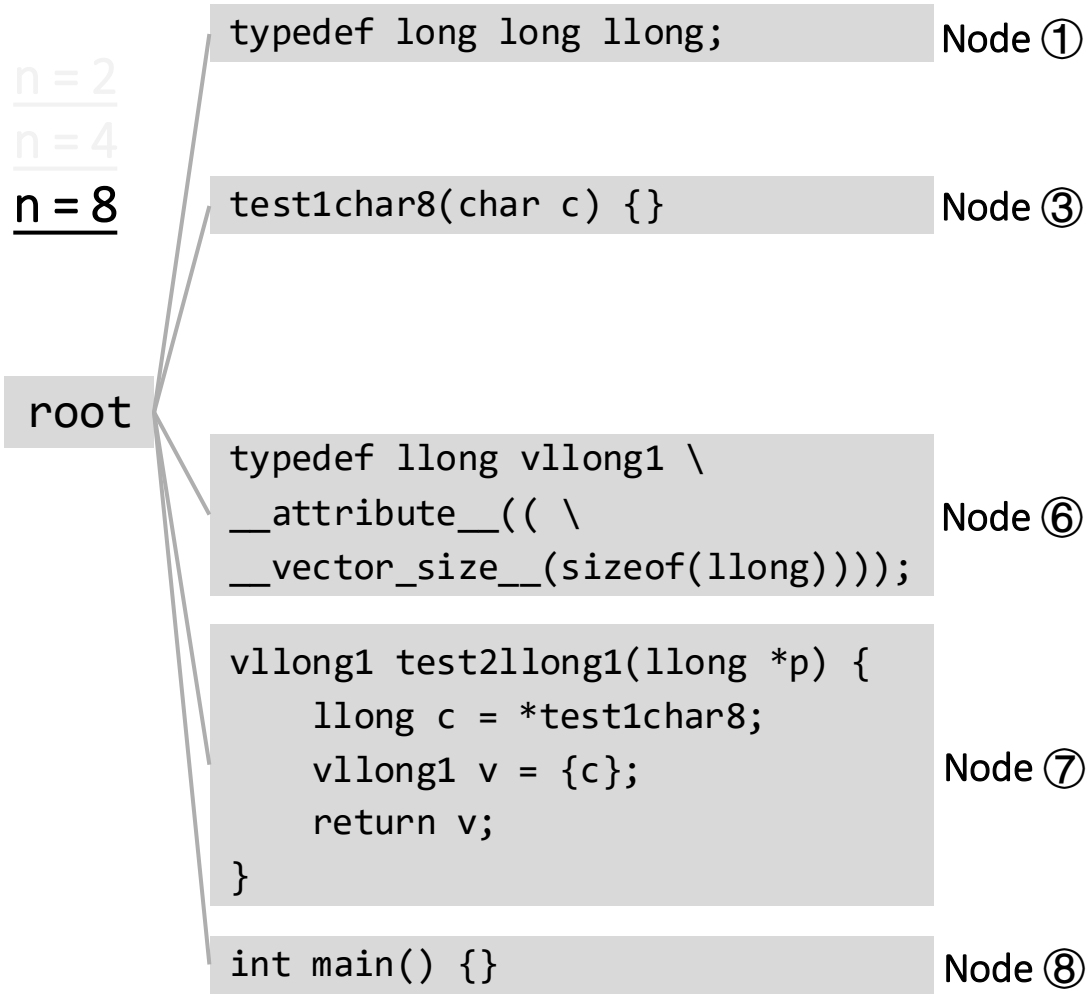
```
int main() {}
```
Node ⑧

Minimizing Delta Debugging (ddmin):

- Binary-search style minimization
- Divide the list evenly by length



Example



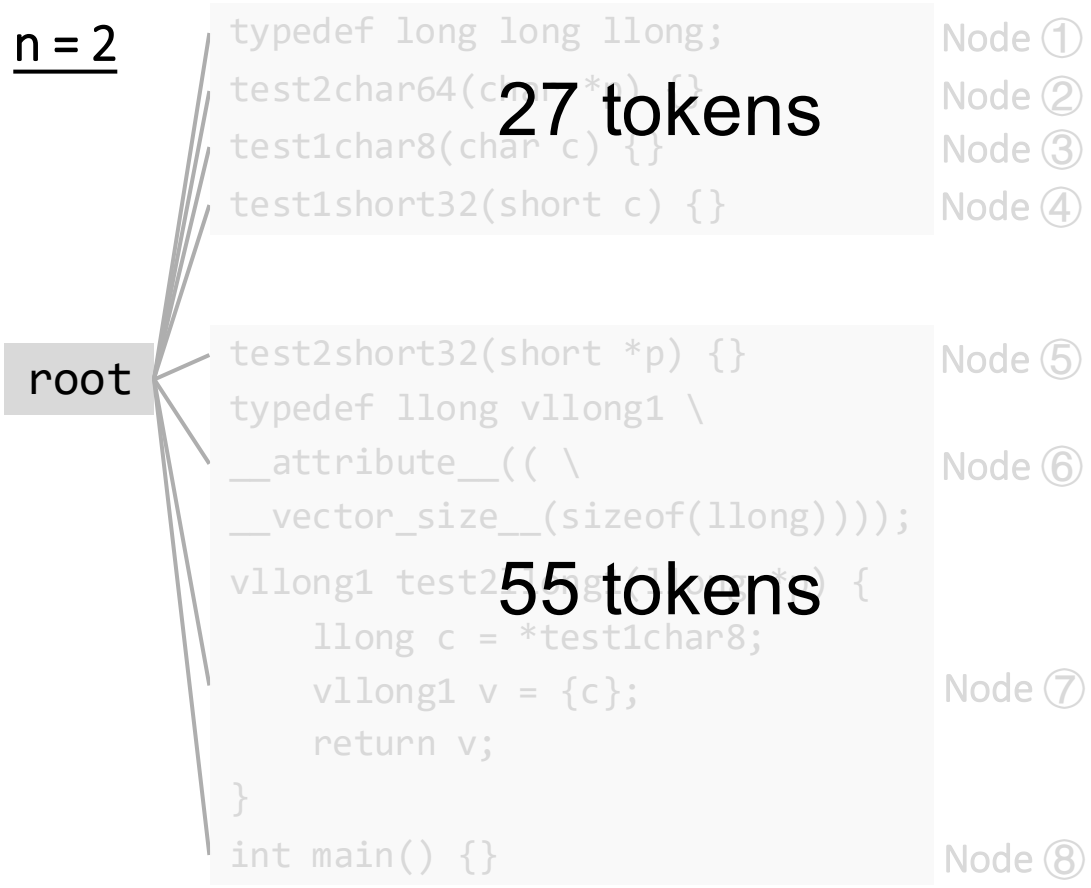
Minimizing Delta Debugging (ddmin):

- Binary-search style minimization
- Divide the list evenly by length

In the example:

- Elements are different in size

Example



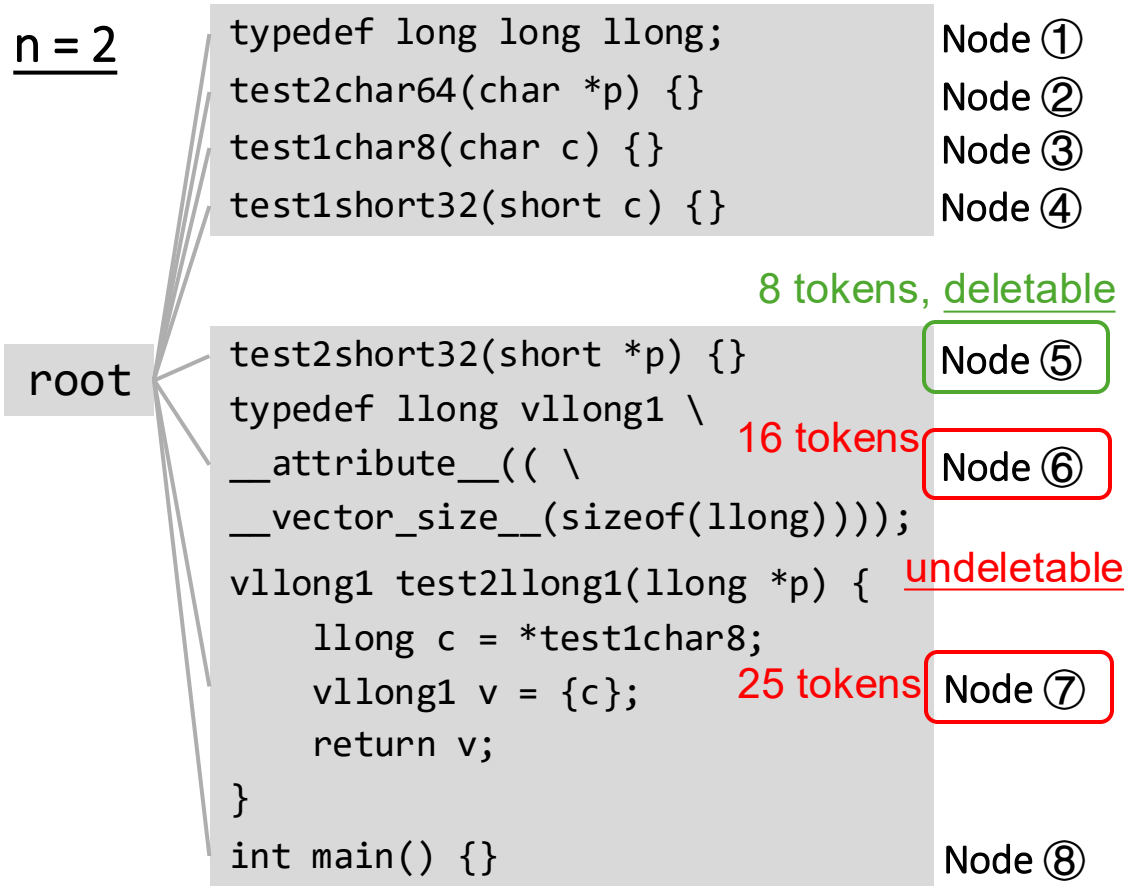
Minimizing Delta Debugging (ddmin):

- Binary-search style minimization
- Divide the list evenly by length

In the example:

- Elements are different in size
- Fail to achieve the actual evenness
- The unevenness can lead to significant decrease in efficiency

Example



Minimizing Delta Debugging (ddmin):

- Binary-search style minimization
- Divide the list evenly by length

In the example:

- Elements are different in size
- Fail to achieve the actual evenness
- The unevenness can lead to significant decrease in efficiency

Intuitive explanation:

- Larger elements are less likely to be deleted than smaller ones
- Small elements fail to be deleted when divided with large non-deletable elements in the same partition

Weighted Delta Debugging (WDD)

Observation:

- Elements are different in size
- Larger elements are more likely to include the buggy code, thus less likely to be deleted (verified by RQ1 in the paper)

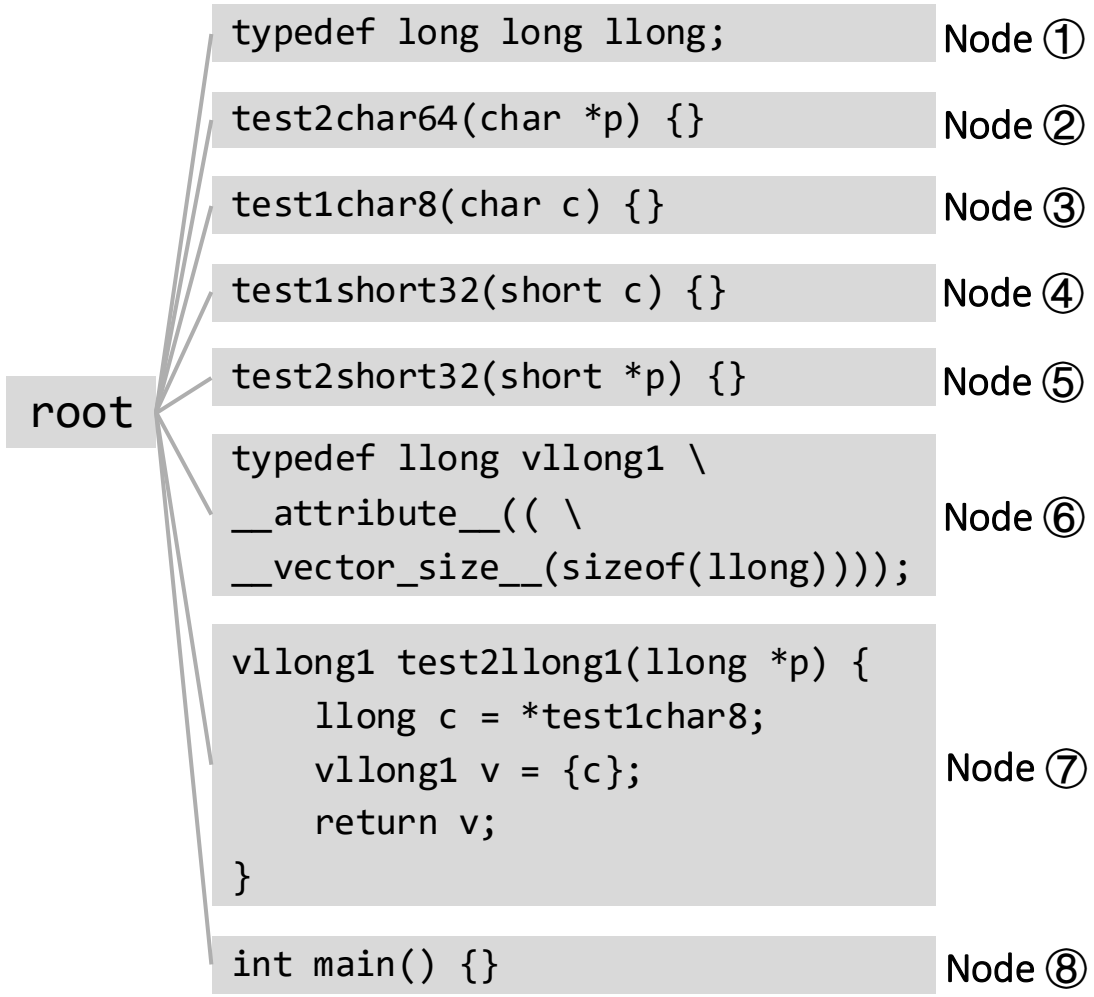
Insight:

- Assign each element a weight according to its size
  - size, *i.e.*, number of tokens
- Handle elements differently based on weight

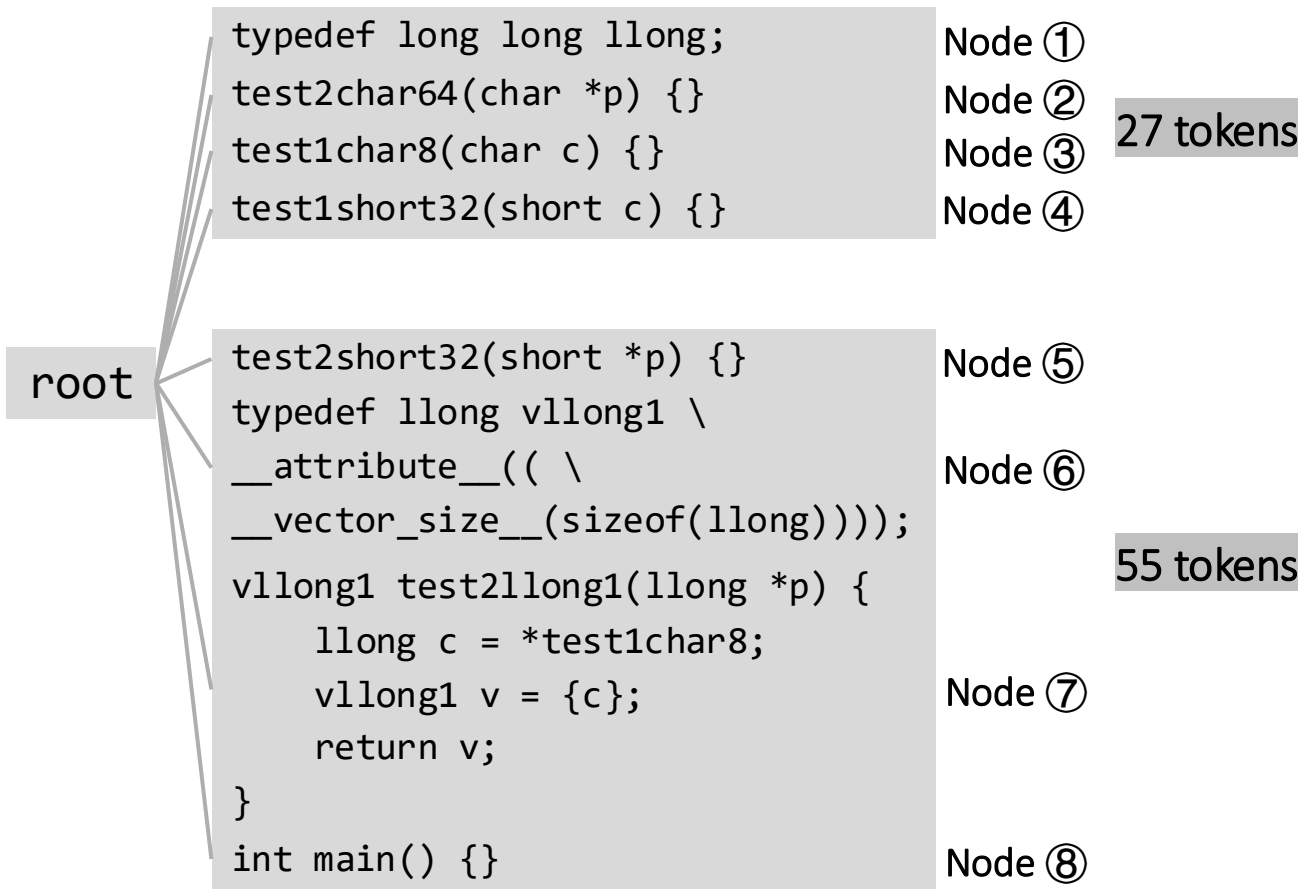
Contributions:

- Weighted ddmin (Wddmin)
- Weighted ProbDD (WProbDD)

Weighted Delta Debugging (WDD)



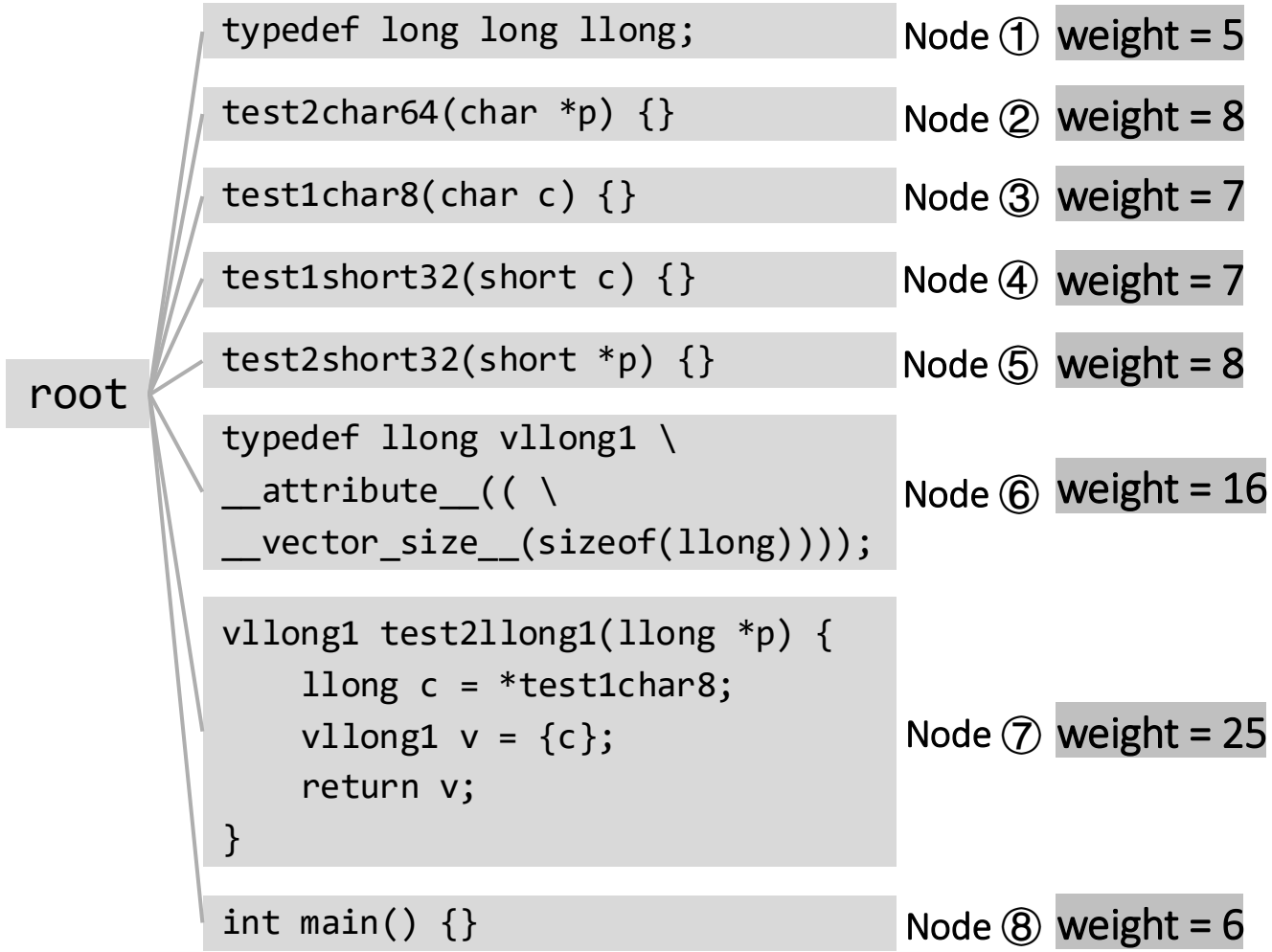
Weighted Delta Debugging (WDD)



ddmin: divide the list evenly by *length*

- |①②③④| = |⑤⑥⑦⑧|= 4

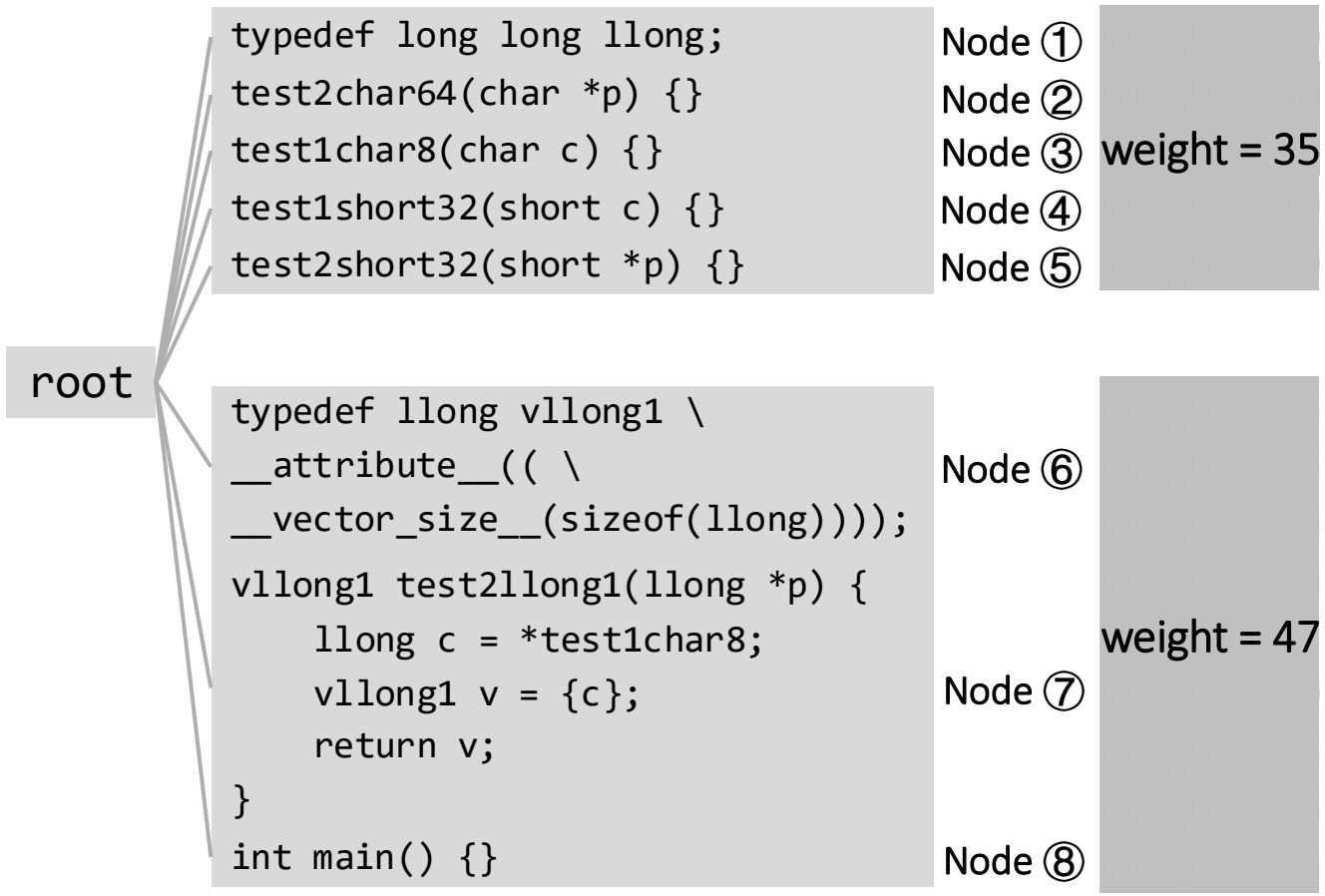
Weighted Delta Debugging (WDD)



ddmin: divide the list evenly by *length*

- |①②③④| = |⑤⑥⑦⑧|= 4

Weighted Delta Debugging (WDD)



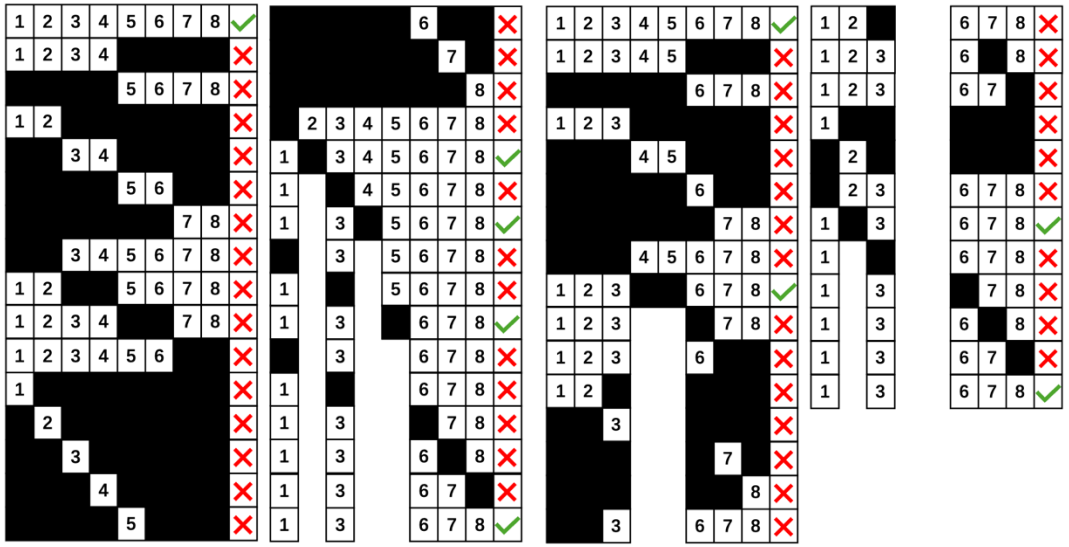
ddmin: divide the list evenly by *length*

- |①②③④| = |⑤⑥⑦⑧|= 4

weight

Wddmin: divide the list evenly by *weight*

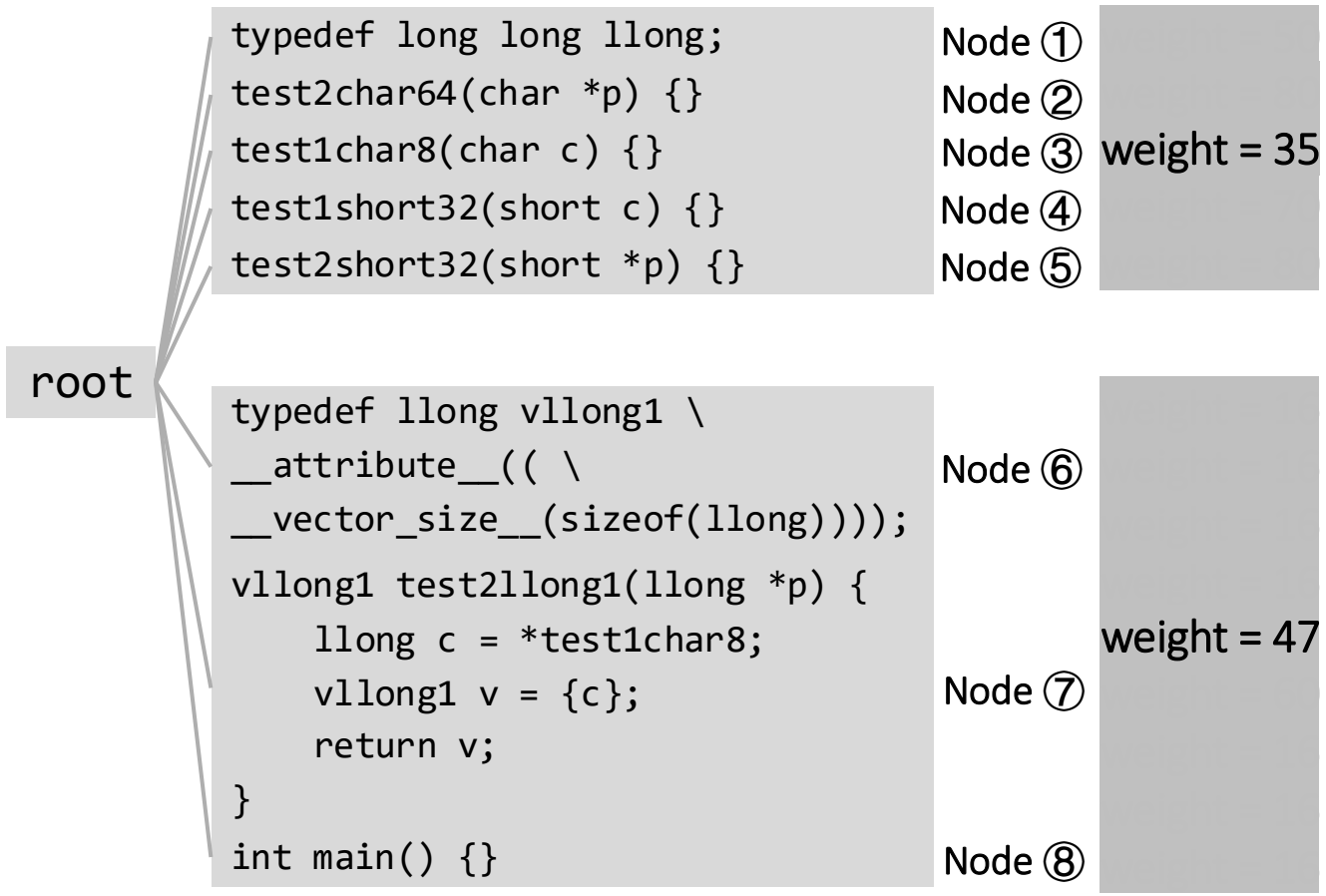
- weight(①②③④⑤) ≈ weight(⑥⑦⑧)



ddmin: 30 tests

Wddmin: 26 tests

Weighted Delta Debugging (WDD)



ddmin: divide the list evenly by *length*

- |①②③④| = |⑤⑥⑦⑧|= 4

↓ weight

Wddmin: divide the list evenly by *weight*

- weight(①②③④⑤) ≈ weight(⑥⑦⑧)

ProbDD: *probability-guided* Delta Debugging

↓ weight

WProbDD: *weighted-probabilistic model-guided* Delta Debugging

- Always towards optimal result (maximize weights can be deleted)
- more details in our paper

Evaluation

- Wddmin
- WProbDD

Applications:

- HDD
  - HDD-ddmin *v.s.* HDD-Wddmin
  - HDD-ProbDD *v.s.* HDD-WProbDD
- Perses
  - Perses-ddmin *v.s.* Perses-Wddmin
  - Perses-ProbDD *v.s.* Perses-WProbDD

Benchmarks:

- 32 C programs trigger bugs in GCC/LLVM
  - 77,723 tokens on average
- 30 XML files trigger bugs in Basex
  - 20,197 tokens on average

Metrics:

- Efficiency
  - Processing time (seconds)
- Effectiveness
  - Size of the minimized result (tokens)

ddmin *v.s.* Wddmin

Efficiency (time):

Benchmark	HDD-*ddmin*	HDD-*Wddmin*	Perses-*ddmin*	Perses-*Wddmin*
C	39,108	18,022	4,582	4,169
XML	3,152	2,621	1,295	1,273

Effectiveness (size):

Benchmark	HDD-*ddmin*	HDD-*Wddmin*	Perses-*ddmin*	Perses-*Wddmin*
C	518	477	281	278
XML	98	82	38	38

Slightly smaller results

Wddmin outperforms ddmin mainly in *efficiency*:

- HDD and Perses use ***51.31%*** and ***7.47%*** less time, respectively

Perses has other deletion strategies aside from Delta Debugging.

ProbDD *v.s.* WProbDD

Efficiency (time):

Benchmark	HDD-*ProbDD*	HDD-*WProbDD*	Perses-*ProbDD*	Perses-*WProbDD*
C	6,042	5,384	3,455	2,975
XML	3,004	2,574	1,838	1,813

Effectiveness (size):

Benchmark	HDD-*ProbDD*	HDD-*WProbDD*	Perses-*ProbDD*	Perses-*WProbDD*
C	567	488	280	273
XML	89	80	37	37

WProbDD outperforms ProbDD in:

- Efficiency:
  - HDD and Perses use ***11.98%*** and ***9.72%*** less time, respectively
- Effectiveness:
  - HDD and Perses generate ***13.4%*** and ***2.2%*** smaller results, respectively

Weighted Delta Debugging (WDD)

Observation:

- Elements are different in size
- Larger elements are more likely to include the buggy code, thus less likely to be deleted (verified by RQ1 in the paper)

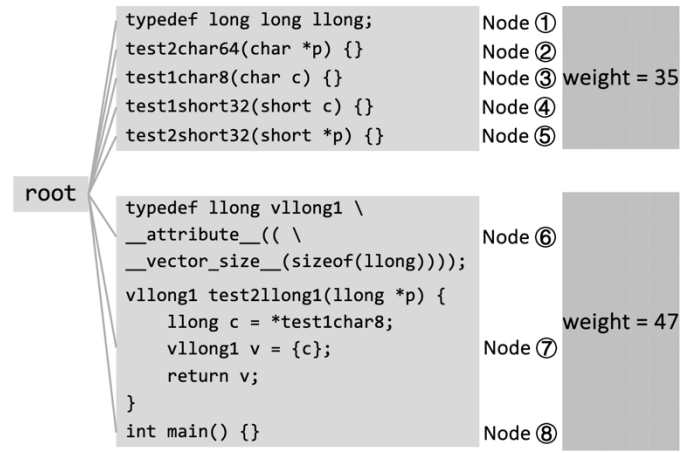
Insight:

- Assign each element a weight according to its size
  - size, *i.e.*, number of tokens
- Handle elements differently based on weight

Contributions:

- Weighted ddmin (Wddmin)
- Weighted ProbDD (WProbDD)

Weighted Delta Debugging (WDD)



ddmin: divide the list evenly by *length*

- |①②③④| = |⑤⑥⑦⑧| = 4

↓ weight

Wddmin: divide the list evenly by *weight*

- weight(①②③④⑤) ≈ weight(⑥⑦⑧)

ProbDD: *probability-guided* Delta Debugging

↓ weight

WProbDD: *weighted-probabilistic model-guided* Delta Debugging

- Always towards optimal result (maximize weights can be deleted)
- more details in our paper

ddmin *v.s.* Wddmin

Efficiency (time):

Benchmark	HDD-*ddmin*	HDD-*Wddmin*	Perses-*ddmin*	Perses-*Wddmin*
C	39,108	18,022	4,582	4,169
XML	3,152	2,621	1,295	1,273

Effectiveness (size):

Benchmark	HDD-*ddmin*	HDD-*Wddmin*	Perses-*ddmin*	Perses-*Wddmin*
C	518	477	281	278
XML	98	82	38	38

Slightly smaller results

Wddmin outperforms ddmin mainly in *efficiency*:

- HDD and Perses use ***51.31%*** and ***7.47%*** less time, respectively

Perses has other deletion strategies aside from Delta Debugging.

ProbDD *v.s.* WProbDD

Efficiency (time):

Benchmark	HDD-*ProbDD*	HDD-*WProbDD*	Perses-*ProbDD*	Perses-*WProbDD*
C	6,042	5,384	3,455	2,975
XML	3,004	2,574	1,838	1,813

Effectiveness (size):

Benchmark	HDD-*ProbDD*	HDD-*WProbDD*	Perses-*ProbDD*	Perses-*WProbDD*
C	567	488	280	273
XML	89	80	37	37

WProbDD outperforms ProbDD in:

- Efficiency:
  - HDD and Perses use ***11.98%*** and ***9.72%*** less time, respectively
- Effectiveness:
  - HDD and Perses generate ***13.4%*** and ***2.2%*** smaller results, respectively