Guide

💡 The test suite includes a small randomized stress test, which may take up to a minute to complete.
See section 2 for how to skip it.

1. Some possible `make` commands

💡 TLDR: `make run` for normal testing, `make check` for *Valgrind, `make test` for Address Sanitizer.

- make all
    - Compile 2 versions of the tests
        - `a4test` for normal testing and Valgrind
        - `a4testASAN` for testing with Address Sanitizer
- make run
    - Compile and run `a4test` without checking for any memory errors.
        - The non-stress tests are run first. Iff they succeed, the stress test will run.
- make check
    - Compile and run only the non-stress tests with Valgrind
- make test
    - Compile and run `a4testASAN` with Address Sanitizer

        - The non-stress tests are run first. Iff they succeed, the stress test will run.

- make debug
    - Compile `debug.cpp`
        - **Not a test, but a tool**.
        - It's set up to help quickly reproduce a result with some test data. See section 4.
- make clean
    - Remove all generated executables

2. Some test options

The test program can take command-line arguments. Most helpful for now are:

- Exclude test cases

  ```
  -tce=  or  --test-case-exclude=

  e.g.  ./a4test     -tce="*stress*"
  or    ./a4testASAN --test-case-exclude="*stress*"

  This excludes any test case with the word "stress" in its name, hence the use of asterisks.
  ```

  Chaining with make: `make a4test && ./a4test -tce="*stress*"`.

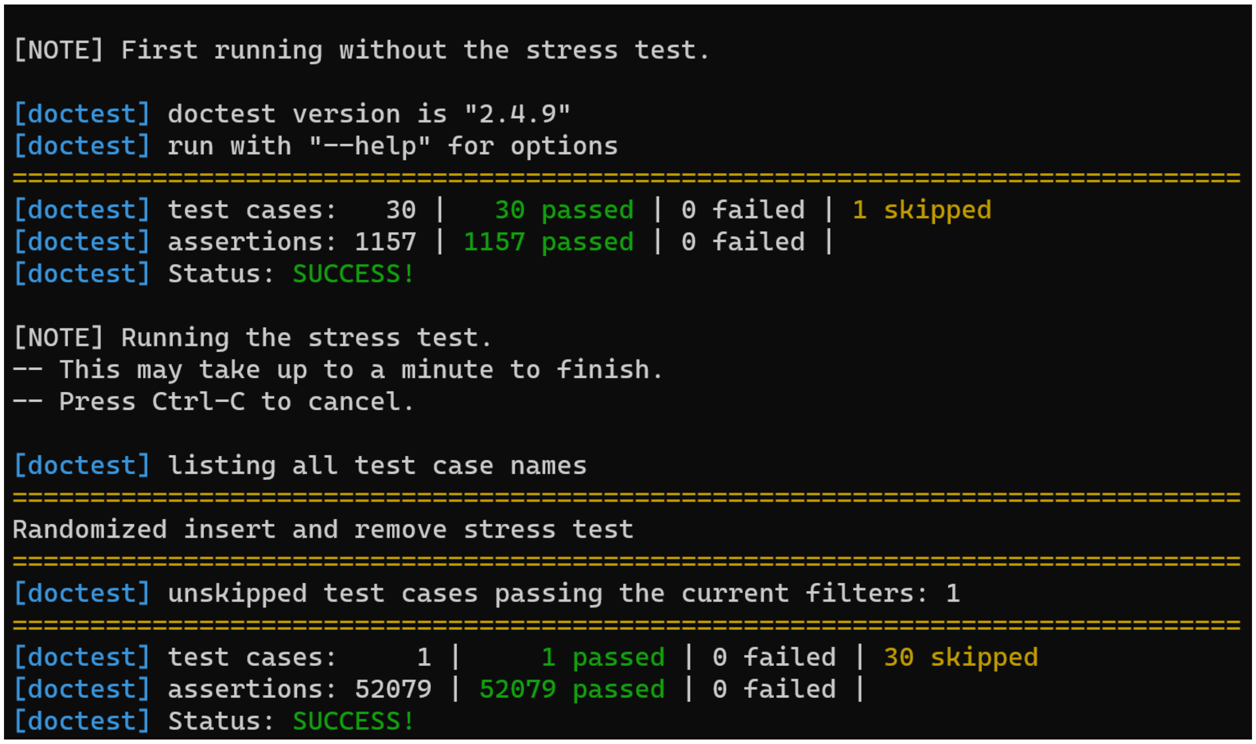
- Select test cases

  ```
  -tc=  or  --test-case=

  e.g.  ./a4test     -tc="*stress*"
  or    ./a4testASAN --test-case="*stress*"

  This runs only test cases with the word "stress" in their names.
  ```

For more, see: doctest/commandline.md at master · doctest/doctest · GitHub

3. Result messages

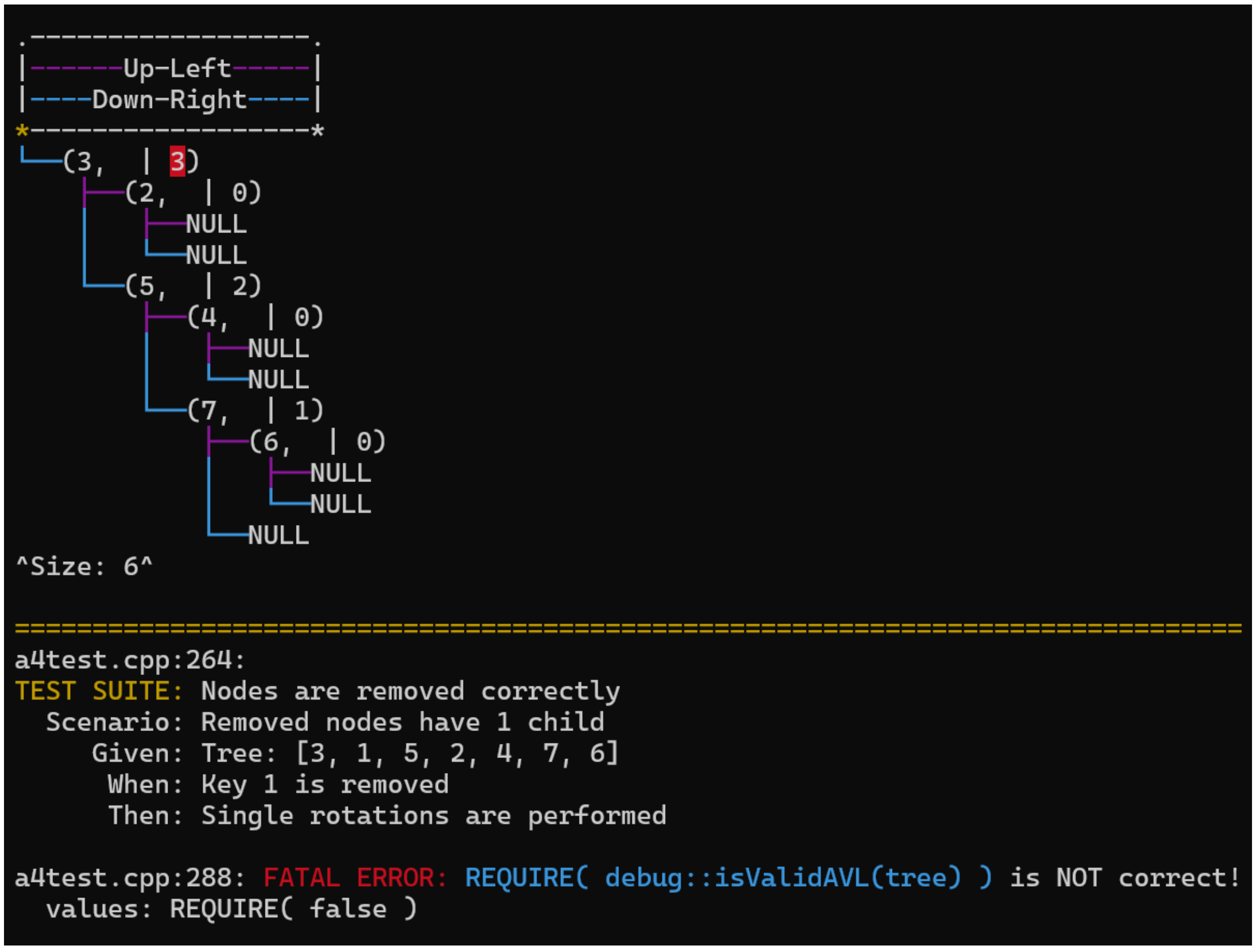
Success



```
[NOTE] First running without the stress test.

[doctest] doctest version is "2.4.9"
[doctest] run with "--help" for options
===============================================================================
[doctest] test cases:   30 |   30 passed | 0 failed | 1 skipped
[doctest] assertions: 1157 | 1157 passed | 0 failed |
[doctest] Status: SUCCESS!

[NOTE] Running the stress test.
-- This may take up to a minute to finish.
-- Press Ctrl-C to cancel.

[doctest] listing all test case names
===============================================================================
Randomized insert and remove stress test
===============================================================================
[doctest] unskipped test cases passing the current filters: 1
===============================================================================
[doctest] test cases:     1 |     1 passed | 0 failed | 30 skipped
[doctest] assertions: 52079 | 52079 passed | 0 failed |
[doctest] Status: SUCCESS!
```

The number of assertions may change.

Error



```
.-------------------.
|-------Up-Left-----|
|----Down-Right-----|
*-------------------*
└──(3,   | 3)
    ├──(2,   | 0)
    │   ├──NULL
    │   └──NULL
    └──(5,   | 2)
        ├──(4,   | 0)
        │   ├──NULL
        │   └──NULL
        └──(7,   | 1)
            ├──(6,   | 0)
            │   ├──NULL
            │   └──NULL
            └──NULL
^Size: 6^

===============================================================================
a4test.cpp:264:
TEST SUITE: Nodes are removed correctly
  Scenario: Removed nodes have 1 child
     Given: Tree: [3, 1, 5, 2, 4, 7, 6]
      When: Key 1 is removed
      Then: Single rotations are performed

a4test.cpp:288: FATAL ERROR: REQUIRE( debug::isValidAVL(tree) ) is NOT correct!
  values: REQUIRE( false )
```

An invalid AVL tree.

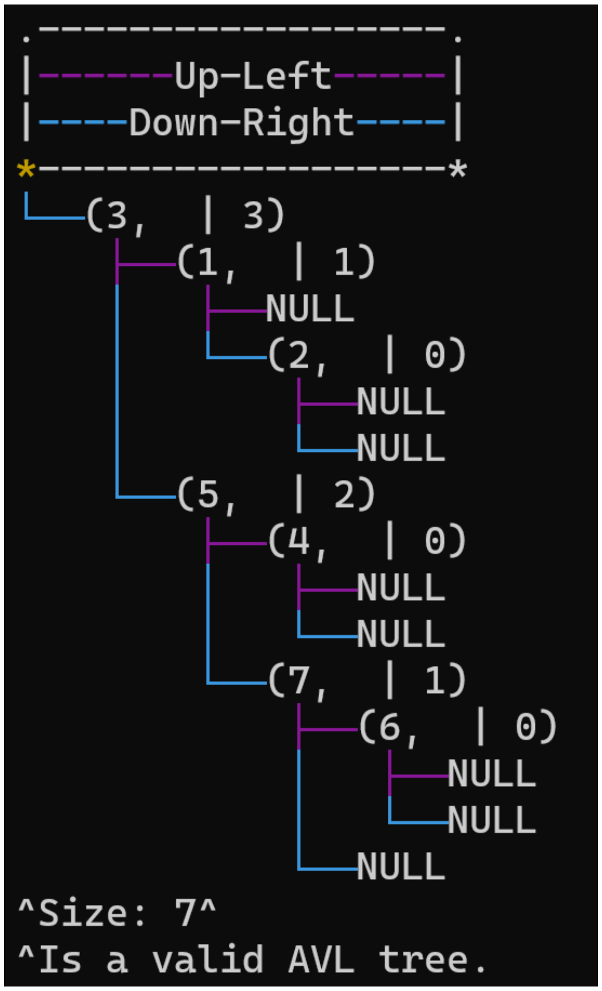
💡 NOTE: The problematic tree is printed **above** the test description, and **after** it's operated on. Some tests may not print any trees.

4. Reproducing results

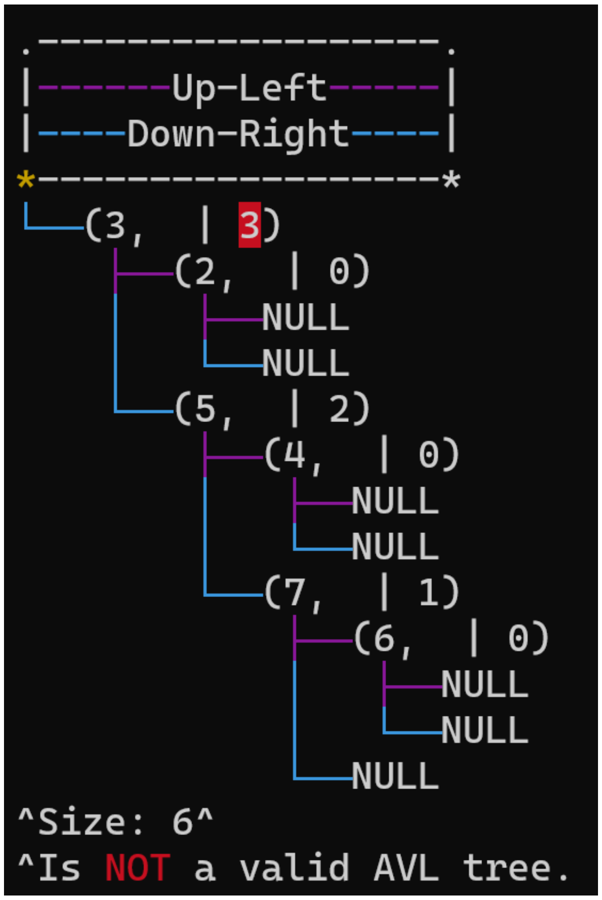
Plugging the above test data into `debug.cpp`, for example:

```
#include "tools.h"
using tools::makeTree;
...
int main() {
    auto tree = makeTree({ 3, 1, 5, 2, 4, 7, 6 });
    cout << tree << treeSpec(tree) << "\n";
    tree.remove(1);
    cout << tree << treeSpec(tree) << "\n";
}
```

And call `make debug`:



```
.-------------------.
|------Up-Left------|
|----Down-Right-----|
*-------------------*
└──(3,   | 3)
    ├──(1,   | 1)
    │   ├──NULL
    │   └──(2,   | 0)
    │       ├──NULL
    │       └──NULL
    └──(5,   | 2)
        ├──(4,   | 0)
        │   ├──NULL
        │   └──NULL
        └──(7,   | 1)
            ├──(6,   | 0)
            │   ├──NULL
            │   └──NULL
            └──NULL
^Size: 7^
^Is a valid AVL tree.
```



```
.-------------------.
|------Up-Left------|
|----Down-Right-----|
*-------------------*
└──(3,   | 3)
    ├──(2,   | 0)
    │   ├──NULL
    │   └──NULL
    └──(5,   | 2)
        ├──(4,   | 0)
        │   ├──NULL
        │   └──NULL
        └──(7,   | 1)
            ├──(6,   | 0)
            │   ├──NULL
            │   └──NULL
            └──NULL
^Size: 6^
^Is NOT a valid AVL tree.
```

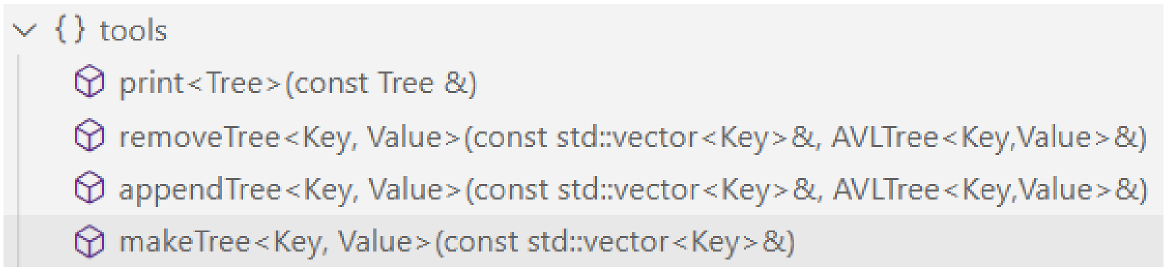
The root node is unbalanced.

5. `tools.h`

All tools in the previously published `tools.cpp` are now updated and moved into `namespace tools` in the new header `tools.h`.

Using these may require prepending their names with `tools::`. For example:

```
#include "tools.h"
int main() {

    // Alternately:
    using tools::isValidAVL;

    auto tree = tools::makeTree({});
    if (isValidAVL(tree)) {
        // ...
    }
    // ...
}
```



```
{} tools
    print<Tree>(const Tree &)
    removeTree<Key, Value>(const std::vector<Key>&, AVLTree<Key,Value>&)
    appendTree<Key, Value>(const std::vector<Key>&, AVLTree<Key,Value>&)
    makeTree<Key, Value>(const std::vector<Key>&)
```

Some of the tools.