

Algorithmic programming

...and testing your suffix trees...

KISS

- ***Keep It Simple, Stupid!***
 - You know the algorithm
 - You know what you need
 - *Implement that!*
 - *Don't get inventive!*
- Choose simple solutions first
 - Simple (but perhaps slow) solutions in 1st iteration
 - Then improve them
 - Premature optimization is the root of all evil!

Example...

- Implement a simple compressed trie
 - Insert strings one at a time (using **slowscan**)
 - Test correctness
- Use it to build a suffix tree
 - Insert *suffixes* one at a time
 - Test correctness
- Use McCreight's algorithm
 - Use **fastscan** and test correctness
 - Add suffix links and test correctness

Go easy on the OO modeling

- This is **not** an OO course
 - Don't do OO modeling
 - Attributes determined by algorithm
 - If you need a pointer, use a bloody pointer!
 - e.g. a suffix link or an edge is not an “object” with identity, attributes and methods
- OO (as you know it) is for modeling and architecture
 - Use it for data structure interfaces
 - Ignore it for the implementation
 - (I am exaggerating here...)

Defensive programming

- Annotate your code with assertions
 - Be able to turn them on/off for efficiency in production code
 - But test *everything* you reasonably can
 - Pre- and post-conditions
 - Invariants
 - Remember to test cases that “can’t happen”!
 - **Do this as you write the code, not afterwards!**

Aggressive testing

- Test early and test often
 - Testing suite you can run after each change
 - *Automate your testing!*
 - Unit testing of all structures and routines
 - Regression testing
 - Test common and boundary cases
 - “Random” crash testing
- Never ever delete a (non-redundant) test!
 - Let the testing suite grow
 - Don't delete a test after you have used it

Make debugging easy

- Add code for inspection
 - Be able to print algorithmic state
 - Be able to print data structures
 - Make sure you can turn this on and off (but don't ever delete inspection code!)

Example

```
> ~mailund/suffix_tree-2.0/print-structure.py abaab
```

The tree structure:

```
`- a
|
| ` - b
|   |
|   | ` - aab$ : abaab$
|   | ` - $ : ab$
|   |
|   ` - ab$ : aab$
|
| ` - b
|   |
|   | ` - aab$ : baab$
|   | ` - $ : b$
|
| ` - $ : $
```

- ...or consider e.g. Graphviz
<www.graphviz.org> for visualization

Make debugging easy

- Use your test suite for debugging
 - *First* you build a test to trigger a bug
 - *Then* you fix it
 - ...recurse as necessary

Test by profiling

- Annotate the code for timing
 - Plot time used for various routines
 - Compare with expected running times
 - Be able to turn this profiling code on/off
- For McCreight's algorithm:
 - Linear time for **slowscan**
 - Linear time for **fastscan**
 - Linear time for total construction

Optimize using profiling

- ***Never ever optimize before profiling!***
- Use profiler tools to find hotspots
- Focus on the hotspots
- Only optimize if you expect to gain from it!
- You won't gain from the micro-optimizations!