Wahrheit und methode

Sebastian Michoń

1 Ideas regarding production

There are many things, that have to be done, and they can be done in any possible order. These are:

1. Procrastinating, while contemplating prose: in other words, proof-reading, sanity-checking.
2. Procrastinating, while contemplating visualizations - detecting problems, detecting imprecision and better ways to do stuff.
3. Contemplating upon used mechanism - dynamic software engineering.
4. Writing mathemagical poetry - articles.
5. Writing post-mathemagical poetry - code, that is.
6. Writing documentation, so that the sanity levels of this app will not reach abyss.

2 Ideas regarding writing

1. **Statement Comprehension has to be playful, algo description - exactly opposite, bridge between both shall be colorful visualization.** Visualization with SCompreh shouldn't contain neither correctness nor complexity proof - just an idea of algorithm. Most mathemagicians should be able to deduce, why is something so-and-so, if they are given method and basic outline of its work (except for refined tricks, i.e. suffix automata or NTT).

3 Ideas regarding implementation

1. **Isolate logic from presentation.** Parts of logic shall be changed only during the beginning - in state maker it shall only be read.
2. In palingnesia (after constructor and BeginningExecutor), the visualization buttons shall be cast to this.buttons.x, where x represents variable presented by button - it reduces pain of writing and debugging StateMaker.
3. One cannot change this ... except from ephemeral variables explicitly after leaving BeginningExecutor.

4 Inner sanctum - Temp.js and its contents

4.1 On the Unmaker

4.1.1 On The Unmaker, I: elements

1. [0, button, color1, color2] - paints button from color1 to color2.

2. [1, button, innerHTML1, innerHTML2] - changes innerHTML of a button from innerHTML1 to innerHTML2.

3. [2, list, element] - appends element to a list.

4. [3, 'variable_name', value1, value2] - changes this.variable_name from value1 to value2.

5. [5, fun, inv_fun, args] - execute fun on arguments args; in order to reverse changes, uses inv_fun on same arguments.

4.1.2 On The Unmaker, II: Inner works

1. Unmaker reverts changes done by state maker, that ought to be reversed (examples of changes, that shall not be reversed are variables changed only once and used only since then or innerHTML that may be constant during whole execution).

2. Unmaker can paint, pop from list, change value or innerHTML to previous value or execute certain function with given arguments. It executes in reversed order compared to StateMaker.

3. StateMaker has array staat, in which changes to "this" that ought to be reversed and changes to buttons are kept in format described above. The values of "this" variables don't change until the end of a function.

4.1.3 On The Maker and Unmaker, III: Use

1. Unmaker actions depend on clicking "Previous" button, they're coded in Temp.js, so one does not have to do anything in code of an algorithm.

4.2 Ephemeral variables

1. Ephemeral variables - this.ephemeral. ..., have no memory - their value at the end of state is always equal to null. Also, in the beginning they are null -> thus, they don't need to participate in transformator.

2. Their aim is to simplify access to variables (particularly staat and passer) within the StateMaker.

3. At the beginning of StateMaker one has to write
staat=this.ephemeral.staat
passer=this.ephemeral.passer

5 Animations - states and/or description

5.1 NTT/FFT animation: states

1. Writing indexes and values of both polynomials to multiply.

2. Formulate either primitive root (NTT) or starting point (FFT)

3. Write first two roots (1 and either function of proot - NTT or starting point - FFT)

4. Construct butterfly

5. Rewrite sequence according to butterfly indices

6. First part of double - calculate the left part of next polynomial - $A_{i,x,2k}$

7. Second part of double - calculate the right part of next polynomial - $A_{i,x,2k+1}$
8. Repeated values of polynomials in points (roots of unity)
9. Multiplication of polynomials
10. Showing inverse roots
11. Ending (After interpolating C(x))- multiply by inversion

6 Legitimate criticism

6.1 General

1. Add full list of concerns regarding content and its backend to this Sprawko -> 3 days
2. Refactor - prever, finito -> 5 days
3. Rethink colors:
   (a) Painter -> mechanism to write -> 1 day
   (b) Showing past which lead to present -> 1 day (green from IEP?)
   (c) One last gold -> Use gold only for ending -> 1 day
   (d) Pair of intertwined numbers - recolor -> 0.5 day
4. Add all Temp.js to this file -> 1 day
5. Refactor dict in base - Its quality is dubious -> 2 days
6. Rethink what to show, and what not to - use of NTT, BinSear etc.
7. Buttons and SCompreh move with scroll -> 1 day
8. Divide NT from system -> 0.5 day
9. statics system -> 0.5 day
10. Variable naming convention -> 2 days
11. Refactor - division of logic and presentation - 6 days
12. Add one function for double transformation of btn and systematic built-in mechanism for handling next move - 2 days
13. returning 2 list - state and next from StateMaker, also NextState+BeginningExecutor - list - 2 days
14. Finishers to usability - 0.5 day

6.2 Specific

1. Proot - What the hell was I thinking when I wrote stateless finisher? -> 0.5 day
2. Perm rep, no rep - problematic height - one during start, second during finish; also, too big start - reps. -> 0.5 day

7 Future

1. Batch one:
   (a) Add Mobius -> 5 days
   (b) Add Totient genral -> 1 day
   (c) Add IEP generalization -> 1 day (just formula)
   (d) Add DLog -> 8 days
   (e) Add Partitions -> 10 days
2. Batch two:
   (a) Add los catalanos -> 5 days
   (b) Add prime counting -> 7 days