POETRY OF PROGRAMMING

CLOJURE PROJECTS

v2023.06.20

Solving these problems may require a bit of thinking and planning. This is exactly the challenge: decomposing the problem into simpler tasks.

(1) **Creating interactive drawings and animations.** Using the `Quil` library available at `http://quil.info/`, which is the `Clojure` version of the `Processing` library. More than one student can do a Quil project, assuming that they develop a different sketch. Look at the examples to get some inspiration and check the source code for ideas.

(2) **Live coding music project.** The `Overtone` library is a programming interface to a synthesizer, see `https://overtone.github.io/`. The library itself can be found at `https://github.com/overtone/overtone/`. The main difficulty could be to get it working on a given computer, see the wiki for installation `https://github.com/overtone/overtone/wiki/Installing-Overtone`. A very early demo: `https://vimeo.com/22798433`.

(3) **Solving a logic puzzle with the `core.logic` library.** See: `https://github.com/clojure/core.logic/wiki/A-Core.logic-Primer`

(4) **Australian Voting** (Adapted from [1]) Australian ballots require that voters rank all the candidates in order of choice. Initially only the first choices are counted, and if one candidate receives more than 50% of the vote then that candidate is elected. However, if no candidate receives more than 50%, all candidates tied for the lowest number of votes are eliminated. Ballots ranking these candidates first are recounted in favor of their highest-ranked non-eliminated candidate. This process of eliminating the weakest candidates and counting their ballots in favor of the preferred non-eliminated candidate continues until one candidate receives more than 50% of the vote, or until all remaining candidates are tied.

Input: ballots for a given list of candidates. Output: the winner of the election.

```
(au-voting-winner ["John Doe" "Jane Smith" "Jane Austen"]
                  [[1 2 3]
                   [2 1 3]
                   [2 3 1]
                   [1 2 3]
                   [3 1 2]] )
"John Doe"
```

(5) **Check the check** (Adapted from [1]) Input: a chess board position Output: yes if the king is in check, no if not in check.

```
"..k....
 ppp.pppp
 ........
 .R...B..
 ........
 ........
 PPPPPPPP
 K......."
```

The black king is in check.

(6) **Wordfinder** Input: a $m \times n$ grid of letters and a word. Output: find the location(s) of the word in the grid (in columns, rows and diagonals), give the sequence of coordinates for the letters of the word.

```
(wordfinder ["ahk" "pet" "klk" "ili" "pot"] "hello")
([1 2] [2 2] [3 2] [4 2] [5 2])
```

(7) **The Caesar shift cipher**

The Caeser cipher is the simplest form of encryption, where each letter is substituted by another letter from the alphabet shifted by n letters. For example, "hello" can be encrypted as "ifmmp" when using the $n = 1$ shift cipher. Write functions that produce encrypter and decrypter functions for a given n. Write another function that performs a brute-force attack on the cipher by trying all possible shifts.

(8) **The halving method for finding roots**

The *root* of function is a value for x such that $f(x) = 0$. Write a CLJ function find-root that takes a continuous real function $f : \mathbb{R} \to \mathbb{R}$ and two real numbers a, b such that $f(a) < 0$ and $f(b) > 0$. This way f is bound to cross the x-axis at least once, and find-root can find a root by systematically halving the $[a, b]$ interval and calling itself recursively. It should work up to some predefined level of precision.

(9) **Efficient Collatz**

Calculate the return time of integers in the Collatz conjecture as efficiently as possible. This involves storing the return time for each intermediate number.

(10) **Maze solver** A maze is described by a string. Character # represents wall, . path, S the start point, and D the destination.

```
S...
###.
....
.#.#
...D
```

Write a program that outputs a path from start to destination. For instance, using o for the actual path taken.

```
Sooo
###o
..oo
.#o#
..oD
```

(11) **Text search** Implement the Knuth-Morris-Pratt algorithm. `https://en.wikipedia.org/wiki/Knuth-Morris-Pratt_algorithm`

(12) **Pseudo random number generator** Implement an RNG to generate sequences of pseudo random numbers, repeatably starting from a seed. You can use the classic `https://en.wikipedia.org/wiki/Middle-square_method`. See for instance `https://youtu.be/u0t-6lUvXHo`.

(13) **Map generation** Using Perlin-noise (`https://en.wikipedia.org/wiki/Perlin_noise`), create believable 2D (not necessarily 3D) maps of terrains. See for instance, the Minetest game `https://www.minetest.net/`, see a gallery here `https://wiki.minetest.net/Map_generator#Gallery`.

(14) **Biome generation** Same context as for generating terrains, but for this project the task is to generate boundaries of biomes by using Voronoi diagrams `https://en.wikipedia.org/wiki/Voronoi_diagram`.

(15) **Conway's Game of Life** implement this famous cellular auotomaton (or any other). `https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life`

References

[1] S.S. Skiena and M.A. Revilla. *Programming Challenges: The Programming Contest Training Manual*. Texts in Computer Science. Springer New York, 2006.