Algorithms & Data Structures: Lab 16

week of 25th February 2019

1 Setup

1.1 Saving your work from last week

By now you should be familiar with the operations needed to save your work. Make sure you commit your work to version control often, and always have a backup copy, ideally remotely (for example in your own account on the department's gitlab installation.)

1.2 Downloading this week's distribution

Once you have successfully saved your changes from last week, you can get my updates by doing

```
git pull
```

which *should* automatically merge in new content. After the `git pull` command, you should have a new directory containing this week's material (named `16/`) alongside your existing directories.

2 Linear Congruential generator

Implement a Linear congruential pseudorandom number generator, supporting the operations (`next!` and `seed!`) described in the lecture. You are provided with skeleton files and test files, as usual; running `make test` in the `cpp/` or `java/` directory should provide you with a test failure report, and once you have successfully implemented such a generator, you should be able to rerun the tests I provide with success.

One of the issues described in the lecture relates to the low bits of an pseudorandom sequence from an LCG having very little entropy (*e.g.* the lowest bit of the numbers alternating between 0 and 1). One way of mitigating this is to have a larger internal state, and return to the user only the higher-entropy bits. For example, Java's `java.util.Random` and POSIX's `lrand48` both implement a 48-bit internal state, and the `next` method returns the state shifted right by 16 bits. Extend your implementation to support this; define a new constructor with signature

C++ `LCG::LCG(uint64_t _a, uint64_t _c, uint64_t _m, uint64_t seed, uint64_t _shift)`

Java `LCG(long _a, long _c, long _m, long seed, long _shift)`

Make sure that your new implementation continues to work with the existing tests. You will have to write your own tests for non-zero values of shift.

3 Xorshift generator

Implement a xorshift random number generator; specifically, the general version (with arbitrary shifts a, b and c) of `unsigned long xor` given on page 4 of George Marsaglia's "Xorshift RNGs", published by the Journal of Statistical Software in July 2003.

Your implementation should maintain the one unsigned 32-bit word of state, and update and return it according to:

x ⟵ x ⊕ LEFTSHIFT(x,a)
x ⟵ x ⊕ RIGHTSHIFT(x,b)
x ⟵ x ⊕ LEFTSHIFT(x,c)
return x

A skeleton implementation and a single basic test is provided.

4 Pseudorandom number generators submission

Submit your work on pseudorandom number generators, along with answers to the optional extra questions, to the submission area for this lab on the VLE by **16:00** on **Friday 1st March**. As usual, you may submit more than once, and your highest score is retained.