CS 361 - Sequence Generation

Drexel University Professor Mark Boady

Overview

A **Non-Sum One Difference Zero Sequence** is a special sequence of integers. It is an ordered set of integer that meet certain properties. The name is a little long and traditionally shortened to **NS1D0**.

As far as Professor Boady is aware, these sequences can only be generated by guess and check.

In this assignment, you will try to generate all sequences of a given length n. *Note: Professor Boady will collect the sequences everyone generates to look for patterns.*

Since this is the final assignment, you may use anything you learned in class or in the reading including any provided code. You may use anything from the textbook or anything included in the C++ 17 standard.

You can learn more about the sequences here:

Steiner Triple Systems

Non-Sum-One-Difference-Zero Sequences

NS1D0 Sequences and Anti-Pasch Steiner Triple Systems

NS1D0 Sequences

Let n be an **odd** integer and $n > 1$. An NS1D0(n) sequence is an ordered list of unique integers selected from 0 to $n-1$ that meets the following requirements.

1) The sequence must contain exactly $\frac{n-1}{2}+1$ integers
2) The sequence must start with 0
3) The sequence must end in 1
4) The sequence cannot include $\lceil \frac{n}{2} \rceil$. (The ceiling symbol means round up.)
5) For any number $1 < x < n$ you can have **either** x or $((1-x) \mod n)$, but **not** both.
6) For each integer $j = 1, 2, 3, \cdots, n-1$ only one of the numbers j or $(-j \mod n)$ can be expressed as $(a_k - a_{k-1}) \mod n$ where k in an index into the array $(k = 1, 2, \cdots, (n-1)/2)$. Additionally, only one k value can map to each pair $(j, -j \mod n)$.

An extended of example of finding all NS1D0(7) sequences will be given in a later section.

Your program will take two command line inputs `n` and `filename`.

- The value `n` determines the integers to select from (0 to $n-1$) and target sequence length $\frac{n-1}{2}+1$.

- The `filename` value is the file to write your sequences to.

Both inputs to your program are required. If $n \leq 1$ or even, the program will exit with a message that the input was invalid. There are n values where no sequences exist, but that is not something you can predict. You would just have to try and fail.

The `filename` is also required. If no `filename` is given, exit with a message that the `filename` was missing.

Given valid command line inputs, your program will write **all** NS1D0(n) sequences into the file using the format described below.

Extended Example

The smallest input that generates a meaningful answer is $n = 7$. In this section, we will find all NS1D0(7) sequences by hand.

Note: if you are confused about the negatives in the modular arithmetic there will be a section later to clarify.

Since the value $n = 7$ we much select integers from the following set $\{0, 1, 2, 3, 4, 5, 6\}$.

The length of the sequence is given by rule 1.

$\frac{7-1}{2} + 1 = \frac{6}{2} + 1 = 3 + 1 = 4$

We need to have a sequence with exactly 4 elements.

$[?, ?, ?, ?]$

Rule 2 says the sequence must start with 0.

$[0, ?, ?, ?]$

Rule 3 says the sequence must end with 1.

$[0, ?, ?, 1]$

Rule 4 asy that $\lceil \frac{7}{2} \rceil = 4$ cannot appear in the sequence.

After following rules 1-4, we are left with $\{2, 3, 5, 6\}$ to fill in the last two blanks.

Rule 5 says there are certain pairs that cannot appear in the seqeunce.

Rule 5: For any number $1 < x < n$ you can have **either** $(x \mod n)$ or $((1 - x) \mod n)$, but **not** both.

We can generate the pairs and see what we get. There is no point in testing 4 because it was already eliminated.

- $x = 2$ then $(2 \mod 7) = 2$ and $(1 - 2 \mod 7) = 6$
- $x = 3$ then $(3 \mod 7) = 3$ and $(1 - 3 \mod 7) = 5$
- $x = 5$ then $(5 \mod 7) = 5$ and $(1 - 5 \mod 7) = 3$

- $x = 6$ then $(6 \mod 7) = 6$ and $(1 - 6 \mod 7) = 2$

We cannot select both (2 and 6). We also cannot select both (3 and 5). We look at all the ways we can fill in the last two blanks and eliminate any that break this rule

- $[0, 2, 3, 1]$
- $[0, 2, 5, 1]$
- $[0, 2, 6, 1]$ breaks rule 5
- $[0, 3, 2, 1]$
- $[0, 3, 5, 1]$ breaks rule 5
- $[0, 3, 6, 1]$
- $[0, 5, 2, 1]$
- $[0, 5, 3, 1]$ breaks rule 5
- $[0, 5, 6, 1]$
- $[0, 6, 2, 1]$ breaks rule 5
- $[0, 6, 3, 1]$
- $[0, 6, 5, 1]$

We are down to the following options.

- $[0, 2, 3, 1]$
- $[0, 2, 5, 1]$
- $[0, 3, 2, 1]$
- $[0, 3, 6, 1]$
- $[0, 5, 2, 1]$
- $[0, 5, 6, 1]$
- $[0, 6, 3, 1]$
- $[0, 6, 5, 1]$

We have one rule left. We need to eliminate sequences that break rule 6.

Rule 6: For each integer $j = 1, 2, 3, \cdots, n - 1$ only one of the numbers j or $(-j \mod n)$ can be expressed as $(a_k - a_{k-1}) \mod n$ where k in an index into the array $(k = 1, 2, \cdots, (n - 1)/2)$. Additionally, only one k value can map to each pair $(j, -j \mod n)$.

It helps to generate the pairs $(j \mod n)$ or $((0 - j) \mod n)$ first.

- $j = 1$ then $(1 \mod 7) = 1$ and $((0 - 1) \mod 7) = 6$
- $j = 2$ then $(2 \mod 7) = 2$ and $((0 - 2) \mod 7) = 5$
- $j = 3$ then $(3 \mod 7) = 3$ and $((0 - 3) \mod 7) = 4$
- $j = 4$ then $(4 \mod 7) = 4$ and $((0 - 4) \mod 7) = 3$
- $j = 5$ then $(5 \mod 7) = 5$ and $((0 - 5) \mod 7) = 2$
- $j = 6$ then $(6 \mod 7) = 6$ and $((0 - 6) \mod 7) = 1$

Now we subtract the pairs in the sequence to see if any of these numbers come up.

Test Sequence 1: $[0, 2, 3, 1]$

$(2-0) \mod 7 = 2$

$(3-2) \mod 7 = 1$

$(1-3) \mod 7 = 5$

This sequence fails because 5 and 2 both appear!

Test Sequence 2: $[0, 2, 5, 1]$

$(2-0) \mod 7 = 2$

$(5-2) \mod 7 = 3$

$(1-5) \mod 7 = 3$

This sequence fails because 3 appears twice.

Test Sequence 3: $[0, 3, 2, 1]$

$(3-0) \mod 7 = 3$

$(2-3) \mod 7 = 6$

$(1-2) \mod 7 = 6$

This sequence fails because 6 appears twice.

Test Sequence 4: $[0, 3, 6, 1]$

$(3-0) \mod 7 = 3$

$(6-3) \mod 7 = 3$

$(1-6) \mod 7 = 2$

This sequence fails because 3 appears twice.

Test Sequence 5: $[0, 5, 2, 1]$

$(5-0) \mod 7 = 5$

$(2-5) \mod 7 = 4$

$(1-2) \mod 7 = 6$

This sequence passes the test.

Test Sequence 6: $[0, 5, 6, 1]$

$(5-0) \mod 7 = 5$

$(6-5) \mod 7 = 1$

$(1-6) \mod 7 = 2$

This sequence fails because 5 and 2 both appear!

Test Sequence 7: $[0, 6, 3, 1]$

$(6-0) \mod 7 = 6$

$(3-6) \mod 7 = 4$

$(1-3) \mod 7 = 5$

This sequence passes the test.

Test Sequence 8: $[0, 6, 5, 1]$

$(6-0) \mod 7 = 6$

$(5-6) \mod 7 = 6$

$(1-5) \mod 7 = 3$

This sequence fails because 6 appears twice.

After applying Rule 6 we learn:

- $[0, 2, 3, 1]$ Failed
- $[0, 2, 5, 1]$ Failed
- $[0, 3, 2, 1]$ Failed
- $[0, 3, 6, 1]$ Failed
- $[0, 5, 2, 1]$
- $[0, 5, 6, 1]$ Failed
- $[0, 6, 3, 1]$
- $[0, 6, 5, 1]$ Failed

After eliminating all sequences that break the rules, we are left with two NS1D0(7) sequences.

- $[0, 5, 2, 1]$
- $[0, 6, 3, 1]$

Program Style

Your program must use a minimum of 2 threads. You should do your best to take advantage of as many threads as you can.

If your program was called as

```
./bin/sequence 7 seq7.txt
```

The file `seq7.txt` should contain the following sequences.

```
0, 5, 2, 1
0, 6, 3, 1
```

Put one sequence per line. Seperate values with commas. It doesn't matter if you leave trailing white space or not.

The **order** of the numbers in a sequence matters. The **order** of the sequences in the file does not matter. For example, another valid answer to `./bin/sequence 7 seq7.txt` would be

```
0, 6, 3, 1
0, 5, 2, 1
```

We are just looking for a file contains all valid NS1D0 sequences for the given input n to be written to the file.

Implementation

You are expected to write professional code. Use good variable and function names. Provide comments (in doxygen) explaining how the code works. Documents each function in the header file with comments. You will be graded on style as well as functionality.

Citations

If you use any outside resources, talk about algorithm design with other students, or get help on assignments you **must** cite your resources in the comments. Uncited sources are an Academic Honesty Violation. Cited sources may lead to a deduction depending on the amount of code used, but will not violate Academic Honesty Policies.

You are expect to write the majority of the code yourself and use resources for things like looking up commands. For example, if you forgot how to test if a file can be opened for reading you could look it up and cite a source. If you copy a critical algorithm and cite the code, you may still get a deduction for not developing the code yourself.

Readme

Your readme will include both instructions and reflections on your code. It must be stored on the root of your folder structure. It must include the following. You must use markdown and call it readme.

There is no minimum or maximum length for the short essay questions, you are graded entirely on content. A short but comprehensive answer is better than a long confusing answer.

1) Your name and drexel ID (abc123@drexel.edu)
2) Instructions to run you code.
3) Short Essay Question 1: How did you use concurrency to solve the problem?
4) Short Essay Question 2: How do your threads communicate? Why?
5) Short Essay Question 3: What is the largest value n your program runs in a reasonable time for? (Define *reasonable* yourself)

Makefile

You **must** provide a makefile to compile your code. We will type `make` and it **must** build your program. If there are any compile errors or a makefile is not

provided we cannot test your code. We will test your code on custom folder. No `make run` is needed for this assignment.

You must have the following make targets:

1) `make` - Builds the Program (`bin/sequence`)
2) `make test7` - runs `bin/sequence 7 seq7.txt`
3) `make test9` - runs `bin/sequence 9 seq9.txt`
4) `make test11` - runs `bin/sequence 11 seq11.txt`
5) `make test13` - runs `bin/sequence 13 seq13.txt`
6) `make clean` - Remove compiled code

Other Requirements

If you submission does not meet the following guidelines we will not be able to grade it.

- You **must** use the C++ 17 Standard threads. No other thread libraries (pthreads, boost, etc) may be used. https://en.cppreference.com/w/cpp/header/thread
- Code **must** run on tux and be compiled with g++.
- All code **must** compile using the C++ 17 or above standard. (`--std=c++17`)
- All code **must** be submitted to the course blackboard learn classroom.
- A working makefile **must** be provided.
- Must provide a readme file
- You may use libraries in the C++ 17 standard unless noted elsewhere in the instructions. https://en.cppreference.com/w/cpp/header
- Your code **must** compile. You should always submit stable code, we will not debug code that does not compile.

Grading

This homework is worth 100 points.

Task	Points
Usage of Threads in Design	35
Finds Valid NS1D0 Sequences	35
Overall Style	5
Command Line Input Validation	5
Readme.md: Name and Email	1
Readme.md: Instructions	1
Readme.md: Short Essay 1	4
Readme.md: Short Essay 2	4
Readme.md: Short Essay 3	4
Makefile is correct	3
Required File Structure in blackboard learn	3

Appendix: Modular Arithmetic

The expression $a \mod b$ gives the modulo (remainder of division) from $\frac{a}{b}$. This is easy to define for positive numbers.

The quotient is the fraction rounded down to the nearest integer.

Let's experiment with $a = 9$ and $b = 2$.

$\lfloor \frac{a}{b} \rfloor = \lfloor \frac{9}{2} \rfloor = 4$

The quotient is 4. The remainder is

$a - q * b = 9 - 2 * 4 = 9 - 8 = 1$

Therefore $9 \mod 2 = 1$

What happens when we ask $-1 \mod 2$.

$\lfloor \frac{a}{b} \rfloor = \lfloor \frac{-1}{2} \rfloor = 0$

$-1 - 0 * 2 = -1$

That is what the formula says, but we still have a negative number. There is an alternative solution to this modulo problem.

$(-1 \mod 2) = 1$

Since the first value -1 is less than the second value 2, we can say the mod is $2 - 1 = 1$.

You can also this of this as -1 meaning "one before the denominator of division".

As an example, here is the result of $k \mod 9$ for all values of k from -18 to 18. Notice how with this definition of mod for negatives, we always get a result between 0 and $b - 1$ for $a \mod b$.

$(-18 \mod 9) = 0$

$(-17 \mod 9) = 1$

$(-16 \mod 9) = 2$

$(-15 \mod 9) = 3$

$(-14 \mod 9) = 4$

$(-13 \mod 9) = 5$

$(-12 \mod 9) = 6$

$(-11 \mod 9) = 7$

$(-10 \mod 9) = 8$

$(-9 \mod 9) = 0$

$(-8 \mod 9) = 1$

$(-7 \mod 9) = 2$

$(-6 \mod 9) = 3$

$(-5 \mod 9) = 4$

$(-4 \mod 9) = 5$

$(-3 \mod 9) = 6$

$(-2 \mod 9) = 7$

$(-1 \mod 9) = 8$

$(0 \mod 9) = 0$

$(1 \mod 9) = 1$

$(2 \mod 9) = 2$

$(3 \mod 9) = 3$

$(4 \mod 9) = 4$

$(5 \mod 9) = 5$

$(6 \mod 9) = 6$

$(7 \mod 9) = 7$

$(8 \mod 9) = 8$

$(9 \mod 9) = 0$

$(10 \mod 9) = 1$

$(11 \mod 9) = 2$

$(12 \mod 9) = 3$

$(13 \mod 9) = 4$

$(14 \mod 9) = 5$

$(15 \mod 9) = 6$

$(16 \mod 9) = 7$

$(17 \mod 9) = 8$

$(18 \mod 9) = 0$