CPSC 313 — Solutions for Assignment #1

The Problem To Be Solved: Grindelwald Gaggle Computation

As noted on the assignment, the ***Grindelwald Gaggle*** is a sequence of numbers $\mathcal{G}_0, \mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3, \ldots$ such that, for every non-negative integer i,

$$\mathcal{G}_i = \begin{cases} 1 & \text{if } i = 0, \\ 2 & \text{if } i = 1, \\ 3 & \text{if } i = 2, \\ 4 & \text{if } i = 3, \\ 2\mathcal{G}_{i-1} - 2\mathcal{G}_{i-3} + \mathcal{G}_{i-4} & \text{if } i \geq 4 \text{ and } i \text{ is even}, \\ \mathcal{G}_{i-1} + 3\mathcal{G}_{i-2} - 5\mathcal{G}_{i-3} + 2\mathcal{G}_{i-4} & \text{if } i \geq 4 \text{ and } i \text{ is odd}. \end{cases}$$

These can be used to define the following computational problem:

> **Grindelwald Gaggle Computation**
>
> *Precondition:* A non-negative integer $\mathtt{n}$ is given as input.
>
> *Postcondition:* The $\mathtt{n}^{\text{th}}$ Grindelwald number, $\mathcal{G}_{\mathtt{n}}$, is returned as output.

A Simple — But Slow — Algorithm for This Computation

The initial questions concerned the `sHuff` algorithm, which is shown in Figure 1 on page 2.

1. You were first asked to give a reasonably short proof that the function $f(\mathtt{n}) = \mathtt{n}$ is a ***bound function*** for this recursive algorithm.

   Solution: The function $f(\mathtt{n}) = \mathtt{n}$ is certainly a well-defined integer-valued total function of the input(s) for this algorithm.

```
integer sGrin (integer n) {
// Assertion:  A non-negative integer n has been given as input.
 1. if (n == 0) {
 2.   return 1
 3. } else if (n == 1) {
 4.   return 2
 5. } else if (n == 2) {
 6.   return 3
 7. } else if (n == 3) {
 8.   return 4
 9. } else if ((n mod 2) == 0) {
10.   return 2 × sGrin(n − 1) − 2 × sGrin(n − 3) + sGrin(n − 4)
   } else {
11.   return sGrin(n − 1) + 3 × sGrin(n − 2) − 5 × sGrin(n − 3)
                                                  +2 × sGrin(n − 4)
   }
}
```

Figure 1: A Slow Algorithm for the "Grindelwald Gaggle Computation" Problem

- The algorithm is only recursively executed at line 10, with inputs $\mathtt{n}-1$, $\mathtt{n}-3$ and $\mathtt{n}-4$, or at line 11, with inputs $\mathtt{n}-1$, $\mathtt{n}-2$, $\mathtt{n}-3$ and $\mathtt{n}-4$ — so that (since it is the identity function) the value of f is decreased by *at least* one every time the algorithm calls itself recursively.
- If the algorithm is called when the precondition for the "Grindelwald Gaggle Computation Problem" is satisfied when the `sGrin` algorithm is executed then $\mathtt{n} \geq 0$. On the other hand, if $f(\mathtt{n}) \leq 0$ then $\mathtt{n} \leq 0$ as well — so that $\mathtt{n} = 0$.

  In this case the test at line 1 is passed and the algorithm ends — without ever having called itself recursively — after the execution of the step at line 2.

Since all the properties of a "bound function for a recursive algorithm" are satisfied it follows that the function $f(\mathtt{n}) = \mathtt{n}$ is a bound function for the `sGrin` algorithm, as claimed.

2. You were asked to prove that the `sGrin` algorithm correctly solves the "Grindelwald Gaggle Computation" problem.

   Solution: One can see by inspection of the code that an execution of the `sGrin` algorithm has no undocumented side-effects — it does not access additional inputs, create additional outputs, change the input values, or access or modify any global data. It is therefore necessary and sufficient to prove the following.

   Claim: For every non-negative integer $\mathtt{n}$, if the `sGrin` algorithm is executed on input $\mathtt{n}$ then this execution of the algorithm eventually halts, and the $\mathtt{n}^{\text{th}}$ Grindelwald number $\mathcal{G}_{\mathtt{n}}$ is returned as output.

   Proof. By induction on $\mathtt{n}$. The strong form of mathematical induction will be used and the cases $\mathtt{n} = 0$, $\mathtt{n} = 1$, $\mathtt{n} = 2$ and $\mathtt{n} = 3$ will be considered in the basis.

   Basis: If $\mathtt{n} = 0$ then the test at line 1 is checked and the algorithm halts after returning 1 when executing the step at line 2. Since $\mathcal{G}_0 = 1$, as given above, the claim is satisfied in this case.

   If $\mathtt{n} = 1$ then the test at line 1 is checked and failed, and then the test at line 3 is checked and passed. The execution ends with the value 2 being returned after the execution of the step at line 4. Since $\mathcal{G}_1 = 2$, as given above, it follows that the claim is satisfied in this case as well.

   If $\mathtt{n} = 2$ then the tests at lines 1 and 3 are both checked and failed; the test at line 5 is then checked and passed. The execution ends with the value 3 being returned after the execution of the step at line 6. Since $\mathcal{G}_2 = 3$, as given above, the claim is also satisfied in this case.

   If $\mathtt{n} = 3$ then the tests at lines 1, 3 and 5 are all checked and failed. The test at line 7 is then checked and passed, and the execution of the algorithm ends with the value 4 being returned after the execution of the step at line 8. Since $\mathcal{G}_3 = 4$, as given above, the claim is satisfied in this case too.

   Inductive Step: Let k be an integer such that $k \geq 3$. It is necessary and sufficient to use the following

   > Inductive Hypothesis: If i is an integer such that $0 \leq i \leq k$, and the `sGrin` algorithm is executed with input i, then this execution of the algorithm eventually ends and the i^{th} Grindelwald number, $\mathcal{G}_i$, is returned as output.

   to prove the following

   > Inductive Claim: If the `sGrin` algorithm is executed with input $k + 1$, then this execution of the algorithm eventually ends and the $k+1^{\text{st}}$ Grindelwald number, $\mathcal{G}_{k+1}$, has been returned as output.

With that noted, consider the execution of the sGrin algorithm on input $k+1$. Either $k+1$ is even or $k+1$ is odd; these cases are considered separately below.

If $k+1$ is even then, since $k \geq 3$, $k+1 \geq 4$ and the tests at lines 1, 3, 5 and 7 are all checked and failed; the test at line 9 is now checked and passed, so that the step at line 10 is executed.

- Since $\mathtt{n} = k+1 \geq 4$, $3 \leq \mathtt{n}-1 = k \leq k$ and it follows by the Inductive Hypothesis that the recursive execution of the sGrin algorithm with input $\mathtt{n}-1$ eventually halts, with the value $\mathcal{G}_{\mathtt{n}-1} = \mathcal{G}_k$ returned as output.
- Since $\mathtt{n} = k+1 \geq 4$, $1 \leq \mathtt{n}-3 = k-2 \leq k$ and it follows by the Inductive Hypothesis that the recursive execution of the sGrin algorithm with input $\mathtt{n}-3$ eventually halts, with the value $\mathcal{G}_{\mathtt{n}-3} = \mathcal{G}_{k-2}$ returned as output.
- Since $\mathtt{n} = k+1 \geq 4$, $0 \leq \mathtt{n}-4 = k-3 \leq k$ and it follows by the Inductive Hypothesis that the recursive execution of the sGrin algorithm with input $\mathtt{n}-4$ eventually halts, with the value $\mathcal{G}_{\mathtt{n}-4} = \mathcal{G}_{k-3}$ returned as output.
- One can now see by inspection of the code that the execution of the algorithm halts after this step, and that the value returned is $2\mathcal{G}_k - 2\mathcal{G}_{k-2} + \mathcal{G}_{k-3}$.

Since $k+1$ is even and greater than or equal to four this value is $\mathcal{G}_{k+1}$, as required to establish the Inductive Claim in this case.

If $k+1$ is odd, instead, then, since $k \geq 3$, $k+1 \geq 4$ and the tests at lines 1, 3, 5 and 7 are all checked and failed. The test at line 9 is checked and failed as well since $\mathtt{n} = k+1$ is odd, and the execution of the algorithm continues with the step at line 11.

- Since $\mathtt{n} = k+1 \geq 4$, $3 \leq \mathtt{n}-1 = k \leq k$ and it follows by the Inductive Hypothesis that the recursive execution of the sGrin algorithm with input $\mathtt{n}-1$ eventually halts, with the value $\mathcal{G}_{\mathtt{n}-1} = \mathcal{G}_k$ returned as output.
- Since $\mathtt{n} = k+1 \geq 4$, $2 \leq \mathtt{n}-2 = k-1 \leq k$ and it follows by the Inductive Hypothesis that the recursive execution of the sGrin algorithm with input $\mathtt{n}-2$ eventually halts, with the value $\mathcal{G}_{\mathtt{n}-2} = \mathcal{G}_{k-1}$ returned as output.
- Since $\mathtt{n} = k+1 \geq 4$, $1 \leq \mathtt{n}-3 = k-2 \leq k$ and it follows by the Inductive Hypothesis that the recursive execution of the sGrin algorithm with input $\mathtt{n}-3$ eventually halts, with the value $\mathcal{G}_{\mathtt{n}-3} = \mathcal{G}_{k-2}$ returned as output.
- Since $\mathtt{n} = k+1 \geq 4$, $0 \leq \mathtt{n}-4 = k-3 \leq k$ and it follows by the Inductive Hypothesis that the recursive execution of the sGrin algorithm with input $\mathtt{n}-4$ eventually halts, with the value $\mathcal{G}_{\mathtt{n}-4} = \mathcal{G}_{k-3}$ returned as output.
- One can now see by inspection of the code that the execution of the algorithm halts after this step, and that the value returned is $\mathcal{G}_k + 3\mathcal{G}_{k-1} - 5\mathcal{G}_{k-2} + 2\mathcal{G}_{k-3}$.

Since $k+1$ is odd and greater than or equal to four this value is $\mathcal{G}_{k+1}$, as required to establish the Inductive Claim in this case too.

Since the Inductive Claim has been established in every case this completes the Inductive Step and establishes the claim. □

It follows that the `sGrin` algorithm correctly solves the "Grindelwald Gaggle Computation" problem.

3. You were asked to write a Java program, `SGrindelwald.java`, satisfying a variety of properties.

   Solution: This program is available as a separate file.

4. Let $T_{\mathsf{sGrin}}(n)$ be the number of steps included in the execution of the algorithm, `sGrin`, shown in Figure 1 on input n, for a non-negative integer n — assuming that the ***uniform cost criterion*** is used to define this and the only steps counted are the numbered steps shown in Figure 1.

   You were asked to give a ***recurrence*** for $T_{\mathsf{sGrin}}(n)$.

   Note: If your answer is correct then you should be able to use your recurrence to show that $T_{\mathsf{sGrin}}(0) = 2$, $T_{\mathsf{sGrin}}(1) = 3$, $T_{\mathsf{sGrin}}(2) = 4$, $T_{\mathsf{sGrin}}(3) = 5$, $T_{\mathsf{sGrin}}(4) = 16$, and $T_{\mathsf{sGrin}}(5) = 34$.

   Solution: If $n = 0$ then two steps — the steps at lines 1 and 2 — are performed when this algorithm is executed on input n. Thus $T_{\mathsf{sGrin}}(0) = 2$, as noted above.

   If $n = 1$ then three steps — the steps at lines 1, 3 and 4 — are performed when the algorithm is executed on input n. Thus $T_{\mathsf{sGrin}}(1) = 3$, as noted above.

   If $n = 2$ then four steps — the steps at lines 1, 3, 5 and 6 — are performed when the algorithm is executed on input n. Thus $T_{\mathsf{sGrin}}(2) = 4$, as noted above.

   If $n = 3$ then five steps — the steps at lines 1, 3, 5, 7 and 8 — are performed when the algorithm is executed on input n. Thus $T_{\mathsf{sGrin}}(3) = 5$, as noted above.

   If $n \geq 4$ and n is even then six steps — the steps at lines 1, 3, 5, 7, 9 and 10 — are performed. However, step 10 also includes recursive applications on inputs $n-1$, $n-3$ and $n-4$. Thus

   $$T_{\mathsf{sGrin}}(n) = 6 + T_{\mathsf{sGrin}}(n-1) + T_{\mathsf{sGrin}}(n-3) + T_{\mathsf{sGrin}}(n-4)$$

   in this case. For example (since $4 \geq 4$ and 4 is even)

   $$\begin{aligned} T_{\mathsf{sGrin}}(4) &= 6 + T_{\mathsf{sGrin}}(3) + T_{\mathsf{sGrin}}(1) + T_{\mathsf{sGrin}}(0) \\ &= 6 + 5 + 3 + 2 \\ &= 16, \end{aligned}$$

as noted above.

Finally, if $n \geq 4$ and n is odd then six steps — the steps at lines 1, 3, 5, 7, 9 and 11 — are performed. However, step 11 also includes recursive applications on inputs $n-1$, $n-2$, $n-3$ and $n-4$. Thus

$$T_{\mathsf{sGrin}}(n) = 6 + T_{\mathsf{sGrin}}(n-1) + T_{\mathsf{sGrin}}(n-2) + T_{\mathsf{sGrin}}(n-3) + T_{\mathsf{sGrin}}(n-4)$$

in this case. For example (since $5 \geq 4$ and 5 is odd)

$$\begin{aligned} T_{\mathsf{sGrin}}(5) &= 6 + T_{\mathsf{sGrin}}(4) + T_{\mathsf{sGrin}}(3) + T_{\mathsf{sGrin}}(2) + T_{\mathsf{sGrin}}(1) \\ &= 6 + 16 + 5 + 4 + 3 \\ &= 34, \end{aligned}$$

as noted above.

In summary, if $n \in \mathbb{N}$ then

$$T_{\mathsf{sGrin}}() = \begin{cases} 2 & \text{if } n = 0, \\ 3 & \text{if } n = 1, \\ 4 & \text{if } n = 2, \\ 5 & \text{if } n = 3, \\ 6 + T_{\mathsf{sGrin}}(n-1) + T_{\mathsf{sGrin}}(n-3) + T_{\mathsf{sGrin}}(n-4) & \text{if } n \geq 4 \text{ and } n \text{ is even,} \\ 6 + T_{\mathsf{sGrin}}(n-1) + T_{\mathsf{sGrin}}(n-2) + T_{\mathsf{sGrin}}(n-3) & \\ \qquad + T_{\mathsf{sGrin}}(n-4) & \text{if } n \geq 4 \text{ and } n \text{ is odd.} \end{cases}$$

5. You were asked to use your recurrence to prove that $T_{\mathsf{sGrin}}(n) \geq \left(\frac{3}{2}\right)^n$ for every non-negative integer n. Thus the number of steps executed by the `sGrin` algorithm is at least ***exponential*** in its input.

   Solution: It is necessary and sufficient to prove the following

   Claim: $T_{\mathsf{sGrin}}(n) \geq \left(\frac{3}{2}\right)^n$ for every integer n such that $n \geq 0$.

   Proof. By induction on n. The strong form of mathematical induction will be used and the cases that $n = 0$, $n = 1$, $n = 2$ and $n = 3$ will be considered in the basis.

   Basis: If $n = 0$ then

$$\begin{aligned} T_{\mathsf{sGrin}}(n) &= T_{\mathsf{sGrin}}(0) \\ &= 2 && \text{(as noted above)} \\ &\geq 1 \\ &= \left(\frac{3}{2}\right)^0 \end{aligned}$$

$$= \left(\tfrac{3}{2}\right)^n$$

as required in this case.

If $n = 1$ then

$$\begin{aligned} T_{\text{sGrin}}(n) &= T_{\text{sGrin}}(1) \\ &= 3 && \text{(as noted above)} \\ &\geq \tfrac{3}{2} \\ &= \left(\tfrac{3}{2}\right)^1 \\ &= \left(\tfrac{3}{2}\right)^n \end{aligned}$$

as required in this case, as well.

If $n = 2$ then

$$\begin{aligned} T_{\text{sGrin}}(n) &= T_{\text{sGrin}}(2) \\ &= 4 && \text{(as noted above)} \\ &\geq \tfrac{9}{4} \\ &= \left(\tfrac{3}{2}\right)^2 \\ &= \left(\tfrac{3}{2}\right)^n \end{aligned}$$

as required in this case too.

If $n = 3$ then

$$\begin{aligned} T_{\text{sGrin}}(n) &= T_{\text{sGrin}}(3) \\ &= 5 && \text{(as noted above)} \\ &\geq \tfrac{27}{8} \\ &= \left(\tfrac{3}{2}\right)^3 \\ &= \left(\tfrac{3}{2}\right)^n, \end{aligned}$$

also as required in this case.

Inductive Step: Let k be an integer such that $k \geq 3$. It is necessary and sufficient to use the following

> Inductive Hypothesis: $T_{\text{sGrin}}(i) \geq \left(\tfrac{3}{2}\right)^i$ for every integer i such that $0 \leq i \leq k$.

to prove the following

> Inductive Claim: $T_{\text{sGrin}}(k+1) \geq \left(\tfrac{3}{2}\right)^{k+1}$.

Suppose first that $n = k + 1$ is even. Then, since $k \geq 3$, $0 \leq n-1, n-3, n-4 \leq k$, and it follows by the Inductive Hypothesis that

$$T_{\mathsf{sGrin}}(n-1) \geq \left(\tfrac{3}{2}\right)^{n-1},\ T_{\mathsf{sGrin}}(n-3) \geq \left(\tfrac{3}{2}\right)^{n-3},\ \text{and}\ T_{\mathsf{sGrin}}(n-4) \geq \left(\tfrac{3}{2}\right)^{n-4}.$$

Thus

$$\begin{aligned}
T_{\mathsf{sGrin}}(n) &= T_{\mathsf{sGrin}}(k+1) \\
&= 6 + T_{\mathsf{sGrin}}(n-1) + T_{\mathsf{sGrin}}(n-3) + T_{\mathsf{sGrin}}(n-4) \\
&\qquad\qquad \text{(by the above recurrence)} \\
&\geq 6 + \left(\tfrac{3}{2}\right)^{n-1} + \left(\tfrac{3}{2}\right)^{n-3} + \left(\tfrac{3}{2}\right)^{n-4} \qquad \text{(as noted above)} \\
&= 0 + \left(\tfrac{3}{2}\right)^{n-4} \cdot \left(\left(\tfrac{3}{2}\right)^{3} + \tfrac{3}{2} + 1\right) \\
&= \left(\tfrac{3}{2}\right)^{n-4} \cdot \left(\tfrac{27}{8} + \tfrac{12}{8} + \tfrac{8}{8}\right) \\
&= \left(\tfrac{3}{2}\right)^{n-4} \cdot \tfrac{94}{16} \\
&\geq \left(\tfrac{3}{2}\right)^{n-4} \cdot \tfrac{81}{16} \\
&= \left(\tfrac{3}{2}\right)^{n-4} \cdot \left(\tfrac{3}{2}\right)^{4} \\
&= \left(\tfrac{3}{2}\right)^{n}
\end{aligned}$$

as required to establish the Inductive Claim in this case.

Suppose, instead, that $n = k + 1$ is odd. Once again, since $k \geq 3$,

$$0 \leq n-1, n-2, n-3, n-4 \leq k,$$

and it follows by the Inductive Hypothesis that

$$T_{\mathsf{sGrin}}(n-1) \geq \left(\tfrac{3}{2}\right)^{n-1},\ T_{\mathsf{sGrin}}(n-2) \geq \left(\tfrac{3}{2}\right)^{n-2},\ T_{\mathsf{sGrin}}(n-3) \geq \left(\tfrac{3}{2}\right)^{n-3},$$
$$\text{and}\ T_{\mathsf{sGrin}}(n-4) \geq \left(\tfrac{3}{2}\right).$$

Thus

$$\begin{aligned}
T_{\mathsf{sGrin}}(n) &= T_{\mathsf{sGrin}}(k+1) \\
&= 6 + T_{\mathsf{sGrin}}(n-1) + T_{\mathsf{sGrin}}(n-2) + T_{\mathsf{sGrin}}(n-3) + T_{\mathsf{sGrin}}(n-4) \\
&\qquad\qquad \text{(by the above recurrence)} \\
&\geq 6 + \left(\tfrac{3}{2}\right)^{n-1} + \left(\tfrac{3}{2}\right)^{n-2} + \left(\tfrac{3}{2}\right)^{n-3} + \left(\tfrac{3}{2}\right)^{n-4} \qquad \text{(as noted above)} \\
&= 0 + \left(\tfrac{3}{2}\right)^{n-4} \cdot \left(\left(\tfrac{3}{2}\right)^{3} + \left(\tfrac{3}{2}\right)^{2} + \tfrac{3}{2} + 1\right) \\
&= \left(\tfrac{3}{2}\right)^{n-4} \cdot \left(\tfrac{27}{8} + \tfrac{18}{8} + \tfrac{12}{8} + \tfrac{8}{8}\right) \\
&= \left(\tfrac{3}{2}\right)^{n-4} \cdot \tfrac{130}{16}
\end{aligned}$$

$$\geq \left(\tfrac{3}{2}\right)^{n-4} \cdot \tfrac{81}{16}$$
$$= \left(\tfrac{3}{2}\right)^{n-4} \cdot \left(\tfrac{3}{2}\right)^{4}$$
$$= \left(\tfrac{3}{2}\right)^{n}$$

as required to establish the Inductive Claim in this case as well.

Since the Inductive Claim has been established in every case, this completes the Inductive Step and establishes the claim. □

An Asymptotically Faster Algorithm for This Computation

The next set of questions concerns the other algorithm for the "Grindelwald Gaggle Computation" problem shown in Figure 2 on page 10.

6. You were asked to state a ***loop invariant*** for the `while` loop at lines 15–19 of this algorithm.

   Solution: The following is a ***loop invariant*** for this `while` loop that satisfies the properties listed in the assignment.

   Loop Invariant

   1. `n` is an integer input such that $\texttt{n} \geq 4$.
   2. `i` is an integer variable such that $4 \leq i \leq \texttt{n} + 1$.
   3. `G` is a (variable) integer array with length $\texttt{n} + 1$.
   4. $\texttt{G}[h] = \mathcal{G}_h$ for every integer h such that $0 \leq h \leq \texttt{i} - 1$.

7. You were asked to prove that your answer for the previous question really *is* a loop invariant for the `while` loop in this algorithm.

   Solution: Since the loop test at line 15 has no side-effects, *Loop Theorem #1* can be applied to prove this.

   With that noted, consider an execution of this algorithm beginning with the precondition for the "Grindelwald Gaggle Computation" problem satisfied — so that `n` is an integer input such that $\texttt{n} \geq 0$.

   - Since the precondition for the above problem is satisfied, `n` is an integer input such that $\texttt{n} \geq 0$. There is nothing to be proved, at all, if the loop is never reached and executed, so we may assume that the tests at lines 1, 3, 5 and 7 are all checked and failed (so that execution would continue with the step at line 9). Thus $\texttt{n} \notin \{0, 1, 2, 3\}$. The value of `n` is not changed before the `while` loop is reached so that $\texttt{n} \geq 4$, as needed to satisfy part 1 of the propose loop invariant.

```
integer fGrin (integer n) {
// Assertion:  A non-negative integer n has been given as input.
 1. if (n == 0) {
 2.   return 1
 3. } else if (n == 1) {
 4.   return 2
 5. } else if (n == 2) {
 6.   return 3
 7. } else if (n == 3) {
 8.   return 4
    } else {
 9.   int[] G := new int[n + 1]
10.   G[0] := 1
11.   G[1] := 2
12.   G[2] := 3
13.   G[3] := 4
14.   int i := 4
15.   while (i ≤ n) {
16.     if (i mod 2 == 0) {
17.       G[i] := 2 × G[i − 1] − 2 × G[i − 3] + G[i − 4]
        } else {
18.       G[i] := G[i − 1] + 3 × G[i − 2] − 5 × G[i − 3] + 2 × G[i − 4]
        }
19.     i := i + 1
      }
20.   return G[n]
    }
}
```

Figure 2: Another Algorithm for the "Grindelwald Gaggle Computation" Problem

- The integer variable `i` is initialized to have value 4 when the step at line 14 is executed and its value is not changed before the `while` loop is reached. Since $4 \leq 4 = \mathtt{i} \leq 5 \leq \mathtt{n} + 1$, part 2 of the proposed loop is also satisfied when the `while` loop is first reached.

- The integer array `G` is initialized to have length $\mathtt{n}+1$ at line 9. Since the type and length of this array are not changed, part 3 of the proposed loop invariant is also satisfied when the `while` loop is first reached.
- $\mathtt{G}[0]$ is set to have value $\mathcal{G}_0 = 1$ at line 10, $\mathtt{G}[1]$ is set to have value $\mathcal{G}_1 = 2$ at line 11, $\mathtt{G}[2]$ is set to have value $\mathcal{G}_2 = 3$ at line 12, and $\mathtt{G}[3]$ is set to have value $\mathcal{G}_3 = 4$ at line 13. The values of these array entries are not changed before the `while` loop is reached, so $\mathtt{G}[h] = \mathcal{G}_h$ for every integer h such that $0 \leq h \leq 3 = \mathtt{i} - 1$ at this point. That is, part 4 of the proposed loop invariant is also satisfied at this point.

Since this is the only loop in the algorithm, it is only reached once if it is reached at all. Thus the proposed loop invariant is satisfied *every* time the `while` loop is reached and executed.

Now consider an execution of the *body* of the `while` loop when the proposed loop invariant is initially satisfied.

- Since the value of `n` is not changed when the steps at lines 16–19 are executed, part 1 of the proposed loop invariant is still satisfied when the bottom of the loop boy is reached (if it is reached at all) since it was satisfied when this execution of the loop body began.
- Since part 2 of the proposed loop invariant is initially satisfied, `i` is an integer variable such that $4 \leq \mathtt{i} \leq \mathtt{n}+1$ when this execution of the loop body begins. Furthermore, since the loop test at line 15 has been checked and passed, $\mathtt{i} \leq \mathtt{n}$. The value of `i` is unchanged when the steps at lines 16–18 are executed but it is increased by one when the step at line 19 is executed. Thus $5 \leq \mathtt{i} \leq \mathtt{n}+1$ when the bottom of the loop body is reached (if it is reached, at all), so part 2 of the proposed loop invariant is also satisfied at this point.
- Since the type and length of the integer array `G` are not changed by an execution of the loop body, part 3 of the proposed loop invariant is satisfied at the bottom of this execution of the loop boy because it was satisfied when this execution of the loop body began.
- Since part 4 of the proposed loop invariant is initially satisfied, $\mathtt{G}[h] = \mathcal{G}_h$ for every integer h such that $0 \leq h \leq \mathtt{i} - 1$ when this execution of the loop body begins. Now $0 \leq \mathtt{i} \leq \mathtt{n}$ at this point, as noted above.

  Suppose that `i` is even. Then the test at line 16 is passed so that execution of the algorithm continues with the step at line 17. Since $0 \leq \mathtt{i}-1, \mathtt{i}-3, \mathtt{i}-4 \leq \mathtt{i}-1$ at this point, $\mathtt{G}[\mathtt{i}]$ is set to have value

$$\begin{aligned} 2 \times \mathtt{G}[\mathtt{i}-1] - 2 \times \mathtt{G}[\mathtt{i}-3] + \mathtt{G}[\mathtt{i}-4] \\ &= 2 \times \mathcal{G}_{\mathtt{i}-1} - 2 \times \mathcal{G}_{\mathtt{i}-3} + \mathcal{G}_{\mathtt{i}-4} \\ &= \mathcal{G}_{\mathtt{i}} \end{aligned}$$

since $\texttt{i} \geq 4$ and $\texttt{i}$ is even.

On the other hand, if $\texttt{i}$ is odd then the test at line 16 fails and execution of the algorithm continues with the step at line 18. In this case, since $0 \leq \texttt{i} - 1, \texttt{i} - 2, \texttt{i-3}, \texttt{i} - 4 \leq \texttt{i} - 1$, $\texttt{G}[\texttt{i}]$ is set to have value

$$\begin{aligned}\texttt{G}[\texttt{i} - 1] + 3 \times \texttt{G}[\texttt{i} - 2] - 5 \times \texttt{G}[\texttt{i} - 3] + 2 \times \texttt{G}[\texttt{i} - 4] \\ = \mathcal{G}_{\texttt{i}-1} + 3 \times \mathcal{G}_{\texttt{i}-2} - 5 \times \mathcal{G}_{\texttt{i}-3} + 2 \times \mathcal{G}_{\texttt{i}-4} \\ = \mathcal{G}_{\texttt{i}}\end{aligned}$$

sine $\texttt{i} \geq 4$ and $\texttt{i}$ is odd.

Thus $\texttt{G}[h] = \mathcal{G}_h$ for every integer h such that $0 \leq h \leq \texttt{i}$ immediately before the step at line 19 is reached and executed.

Since the value of $\texttt{i}$ is increased by one at this point $\texttt{G}[h] = \mathcal{G}_h$ for every integer h such that $0 \leq h \leq \texttt{i} - 1$ and part 4 of the proposed loop invariant is also satisfied when the bottom of the body of the `while` loop is reached.

Thus the proposed loop invariant is also satisfied when the bottom of the loop body is reached. It now follows by Loop Theorem #1 that the proposed loop invariant really *is* a loop invariant for the `while` loop in this algorithm.

8. You were asked to use this to prove that this algorithm is ***partially correct***.

   Solution: Consider an execution of this algorithm that begins with the precondition for the "Grindelwald Gaggle Computation" problem satisfied. Then $\texttt{n}$ is an integer input such that $\texttt{n} \geq 0$ so that either $\texttt{n} = 0$, $\texttt{n} = 1$, $\texttt{n} = 2$, $\texttt{n} = 3$, or $\texttt{n} \geq 4$.

   - If $\texttt{n} = 0$ then the test at line 1 is passed and the value $\mathcal{G}_0 = 1$ is returned when the step at line 2 is executed, as required to satisfy the postcondition for the "Grindelwald Gaggle Computation" problem.
   - If $\texttt{n} = 1$ then the test at line 1 fails, the test at line 3 is checked and passed, and the step at line 4 is executed: The value $\mathcal{G}_1 = 2$ is returned, as required to satisfy the postcondition for this problem once again.
   - If $\texttt{n} = 2$ then the tests at lines 1 and 3 are checked and fail, the test at line 5 is checked and passed, and the step at line 6 is executed. The value $\mathcal{G}_2 = 3$ is returned, as required to satisfy the postcondition for this problem in this case too.
   - If $\texttt{n} = 3$ then the tests at lines 1, 3 and 5 are checked and fail, the test at line 7 is checked and passed, and the step at line 8 is executed. The value $\mathcal{G}_3 = 4$ is returned, as required to satisfy the postcondition for this problem.
   - Finally, if $\texttt{n} \geq 4$ then the tests at lines 1, 3, 5 and 7 are all checked and fail. Execution continues with the execution of the steps at lines 9–14. The `while` loop is then reached and executed.

If the execution of the `while` loop does not halt then this execution of the algorithm does not half either, and there is nothing more to be proved. We may therefore assume that the execution of the algorithm eventually halts, so that the step at line 20 is eventually reached and executed.

At this point $0 \leq \mathtt{i} \leq \mathtt{n} + 1$ because part 2 of the loop invariant is satisfied. On the other hand, the test at line 15 must have been checked and failed, so that $\mathtt{i} > \mathtt{n}$ as well. Since `i` and `n` are both integer-valued it follows that $\mathtt{i} = \mathtt{n} + 1$, and it follows by part 4 of the loop invariant that $\mathtt{G}[h] = \mathcal{G}_h$ for every integer h such that $0 \leq h \leq \mathtt{n}$.

In particular, $\mathtt{G}[\mathtt{n}] = \mathcal{G}_{\mathtt{n}}$, so that $\mathcal{G}_{\mathtt{n}}$ is returned as output when the step at line 20 is executed: The postcondition for the problem is satisfied in this case too.

Since either the computation fails to halt or the postcondition is satisfied on termination in every case, it follows that this algorithm is partially correct.

9. You were asked to state a ***bound function*** for the `while` loop in this algorithm and prove that your answer is correct.

   Solution: Consider the function $f(\mathtt{n}, \mathtt{i}) = \mathtt{n} - \mathtt{i} + 1$.

   - This is a well-defined integer-valued function of this algorithm's inputs and local variables (when the loop is reached and executed).
   - During an execution of the loop body the value of `n` is not changed, but the value of `i` is *increased* by one when the statement at line 19 is executed. Thus the value of this function is *decreased* by one during any execution of the body of this loop.
   - If the value of this function is less than or equal to zero then $\mathtt{i} \geq \mathtt{n} + 1$ and the test at line 15 fails, if checked — causing the execution of the loop to end.

   It therefore follows by the definition of a "bound function for a `while` loop" that this function is a bound function for the `while` loop in this algorithm.

10. You were asked to prove that if this algorithm is executed when the precondition for the "Grindelwald Gaggle Computation" problem is satisfied, and the `while` loop is reached and executed, then the execution of the loop eventually ends.

    Solution: Loop Theorem #2 from Lecture #3 can be used to prove this.

    - The loop test at line #15 has no side-effects, that is, it does not change the values of inputs, local variables or global data — or (for that matter) cause any exceptions to be thrown.
    - As established above, this `while` loop has a bound function.
    - The loop body (lines #16–19) does not include any loops or statements that might fail to halt, so every execution of this loop body halts.

It follows by Loop Theorem #2 that every execution of the `while` loop, that is part of an execution of this algorithm starting with the precondition for its problems satisfied, will halt.

11. Using what has been proved so far, you were asked to complete a proof that the `fGrin` algorithm correctly solves the "Grindelwald Gaggle Computation" problem.

    Solution: Since it has already been shown that the `fGrin` algorithm is partially correct, it is necessary and sufficient to prove that if this algorithm is executed with the precondition for the "Grindelwald Gaggle Computation" problem initially satisfied, then this execution of the algorithm terminates.

    Since the the above problem's precondition is satisfied when the execution of the algorithm begins, `n` is an integer input such that $\texttt{n} \geq 0$. It has already been the execution of the algorithm terminates if $0 \leq \texttt{n} \leq 3$, so it suffices to consider an execution when $\texttt{n} \geq 4$.

    In this case the tests at lines 1, 3, 5 and 7 are all checked and failed. The steps at lines 9–14 are executed, and the `while` loop at lines 15–19 is reached and executed. As proved above, this execution of the loop eventually ends — and the execution of the algorithm also ends, after the execution of the step at line 20, as needed to establish the correctness of this algorithm.

12. Let $T_{\text{fGrin}}(n)$ be the number of steps executed by the algorithm `fGrin` when executed on input n, for a natural number n — as defined using the Uniform Cost Criterion (and counting only the numbered steps shown in Figure 2) — so that, for example, $T_{\text{fGrin}}(0) = 2$, because the steps at lines 1 and 2 are carried out when this algorithm is executed on input 0.

    Using techniques introduce in this course, you were asked to give an ***upper bound*** for $T_{\text{fGrin}}(n)$, as a function of n, that is as precise as you can.

    Solution: As noted above, $T_{\text{fGrin}}(0) = 2$ because the steps at lines 1 and 2 are carried out when the algorithm is executed on input 0. Similarly, $T_{\text{fGrin}}(1) = 3$ because the steps at lines 1, 3 and 4 are carried out when the algorithm is executed on input 1, $T_{\text{fGrin}}(2) = 4$ because the steps at lines 1, 3, 5 and 6 are carried out when the algorithm is executed on input 2, and $T_{\text{fGrin}}(3) = 5$ because the steps at lines 1, 3, 5, 7 and 8 are carried out when the algorithm is executed on input 3. It now suffices to consider the case that $n \geq 4$.

    In this case ten steps — the steps at lines 1, 3, 5, 7, and 9–14 — are executed before the `while` loop is reached executed, and one more step — at line 20 — is carried out after the execution of this loop.

    As noted above, $f(\texttt{n}, \texttt{i}) = \texttt{n} - \texttt{i} + 1$ is a bound function for the `while` loop in this algorithm, and (when executed on input n) $n - 4 + 1 = n - 3$ is the initial value of this function. Thus there are at most $n - 3$ executions of the loop body (at lines 16–19) and at most $n - 2$ executions of the loop test (at line 15). Each execution of the loop body

includes three steps — either at lines 16, 17 and 19 or at lines 16, 18 and 19 — while each execution of the loop test includes one step. Thus the total number of steps included in the execution of the loop is at most $3(n-3)+1(n-2) = 4n-11$. Since this execution of the algorithm includes eleven more steps, as noted above, it follows that $T_{\text{fGrin}}(n) \leq 4n$ when $n \geq 4$.

In summary, if $n \in \mathbb{N}$ then

$$T_{\text{fGrin}}(n) \leq \begin{cases} 2 & \text{if } n = 0, \\ 3 & \text{if } n = 1, \\ 4 & \text{if } n = 2, \\ 5 & \text{if } n = 3, \\ 4n & \text{if } n \geq 4. \end{cases}$$

13. You were asked to write a Java program, `FGrindelwald.java`, satisfying a variety of properties.

    Solution: The program is available as a separate file.

Finding a Closed Form

14. You were asked to find an expression for the $\mathtt{n}^{\text{th}}$ Grindelwald number, as a function of `n`, in ***closed form*** — so that it does not include a summation or recurrence — and prove that your answer is correct.

    Solution: A closed form for $\mathcal{G}_n$ is given by the following.

    Claim: $\mathcal{G}_n = n + 1$ for every integer $n \geq 0$.

    Proof. By induction on n. The strong form of mathematical induction will be used and the cases that $n = 0$, $n = 1$, $n = 2$ and $n = 3$ will be considered in the basis.

    Basis: If $n = 0$ then

$$\begin{aligned} \mathcal{G}_n &= \mathcal{G}_0 \\ &= 1 && \text{(by the recurrence at the beginning)} \\ &= 0 + 1 \\ &= n + 1 \end{aligned}$$

    as claimed.

    If $n = 1$ then

$$\mathcal{G}_n = \mathcal{G}_1$$

$$
\begin{aligned}
&= 2 && \text{(by the recurrence at the beginning)} \\
&= 1 + 1 \\
&= n + 1
\end{aligned}
$$

as claimed, for this case too.

If $n = 2$ then

$$
\begin{aligned}
\mathcal{G}_n &= \mathcal{G}_2 \\
&= 3 && \text{(by the recurrence at the beginning)} \\
&= 2 + 1 \\
&= n + 1
\end{aligned}
$$

as claimed, for this case as well.

If $n = 3$ then

$$
\begin{aligned}
\mathcal{G}_n &= \mathcal{G}_3 \\
&= 4 && \text{(by the recurrence at the beginning)} \\
&= 3 + 1 \\
&= n + 1
\end{aligned}
$$

as as claimed for this case.

Inductive Step: Let k be an integer such that $k \geq 3$. It is necessary and sufficient to use the following

Inductive Hypothesis: $\mathcal{G}_h = h + 1$ for every integer h such that $0 \leq h \leq k$.

to prove the following

Inductive Claim: $\mathcal{G}_{k+1} = (k + 1) + 1$.

Now, since $k \geq 3$, $0 \leq k, k-2, k-3 \leq k$ and it follows by the Inductive Hypothesis that $\mathcal{G}_k = k + 1$, $\mathcal{G}_{k-1} = (k-1) + 1 = k$, $\mathcal{G}_{k-2} = (k-2) + 1 = k - 1$ and $\mathcal{G}_{k-3} = (k-3) + 1 = k - 2$.

Either $k + 1$ is even or $k + 1$ is odd. These cases are considered separately below.

If $k + 1$ is even then, since $k + 1 \geq 4$,

$$
\begin{aligned}
\mathcal{G}_{k+1} &= 2\mathcal{G}_k - 2\mathcal{G}_{k-2} + \mathcal{G}_{k-3} && \text{(by the recurrence at the beginning)} \\
&= 2(k+1) - 2(k-1) + (k-2) \\
& && \text{(since } \mathcal{G}_k = k+1, \mathcal{G}_{k-2} = k-1, \text{ and } \mathcal{G}_{k-3} = k-2\text{)} \\
&= (2 - 2 + 1) \cdot k + (2 + 2 - 2) && \text{(reordering terms)}
\end{aligned}
$$

$$= k + 2$$
$$= (k + 1) + 1$$

as required to establish the Inductive Claim in this case.

If $k + 1$ is odd, instead, then, since $k + 1 \geq 4$,

$$\begin{aligned}
\mathcal{G}_{k+1} &= \mathcal{G}_k + 3\mathcal{G}_{k-1} - 5\mathcal{G}_{k-2} + 2\mathcal{G}_{k-3} && \text{(by the recurrence at the beginning)} \\
&= (k + 1) + 3k - 5(k - 1) + 2(k - 2) \\
&\qquad \text{(since } \mathcal{G}_k = k + 1, \mathcal{G}_{k-1} = k, \mathcal{G}_{k-2} = k - 1 \text{ and } \mathcal{G}_{k-3} = k - 2) \\
&= (1 + 3 - 5 + 2)k + (1 + 0 + 5 - 4) && \text{(reordering terms)} \\
&= k + 2 \\
&= (k + 1) + 1
\end{aligned}$$

as required to establish the Inductive Claim in this case too.

Since the Inductive Claim has been established in every possible case, this completes the Inductive Step and establishes the claim. □