CSC410

Assignment 1

Due on 2021/10/07, at 11:59pm

Assignment Format and Guidelines on Submission

This assignment is worth 12% of the total course mark. Submit on Markus and follow these rules:

- This is a group assignment, designed for groups of 3 students. You cannot submit individually. Use Piazza or Quercus to form your groups. If an emergency arises and a group of 3 is missing a member, email your instructor to sort it out before the submission deadline. Markus will be set to accept groups of size 3.
- Each group should submit six files named `gcd.dfy`, `search.dfy`, `rabbit.dfy`, `stackinga.dfy`, `stackingb.dfy`, and `bubble.dfy`. Note that Markus has been setup to accept exactly those files. All the required lemmas, helper functions, etc, should be included in the one file for each problem.
- Each file should contain the code as given in the handout. Changing the algorithm will result in zero marks for that question unless otherwise specified.
- Assume statements and declarations without bodies will result in zero marks for that question. These basically admit facts without proofs into your reasoning and are disallowed for that reason.
- Method signatures for each method should remain exactly as specified in the handout. Changing the method signature to something incompatible with the original will result in zero marks for that question.
- Make sure your final submitted proofs pass through the compile (and not just VSCode) for either Dafny 2 or Dafny 3. VSCode can sometimes be buggy in declaring things verified. You are encouraged, but not required, to work with Dafny 2 as it seems like a more stable version for beginners.

Note that your assignment will be automatically graded. Your function will be called from another function. If you mess with the signature, the call will fail and the autograder will give you a 0 mark.

The submission system will remain open for 12 hours after the deadline, but there is a penalty deduction formula set in Markus that deducts 4% for every hour of late submission up to 12 hours.

Word of Advice

Before you sit behind a Dafny terminal, make sure you have a detailed proof worked out on paper. If you do not have such a (detailed) complete proof, you cannot hack your way through a Dafny proof. You have already seen in class that even when we think our proofs are complete, some intermediate steps (deemed trivial by us) require some more work in Dafny. If you have a complete proof, and are certain about its correctness, but cannot get it through to Dafny, then it most likely means that you are making a leap in reasoning somewhere that seems trivial to you, but not to the prover. We can assure you that this has nothing to do with a *feature* or a *command* that you do not know and have to dig up from a manual/tutorial. Everything you need to know to solve these has already been covered in class and Dafny tutorials.

Problem 0 (25 points)

(Warm up!) Consider the predicate `divides` as defined below:

```
predicate divides(a: nat, b:nat)
   requires a > 0
{
  exists k: nat :: b == k * a
}
```

It formalizes the standard mathematical concept of $a|b$. Encode the following property of *gcd* in Dafny and prove it:

$$\forall k, a, b \in \mathbb{N} : \; k|a \wedge k|b \implies k|(a, b)$$

Note that we want to prove that Euclid's GCD algorithm satisfies the above (and not the mathematical definition of (a, b) (the greatest common divisor of a and b).

Problem 1 (30 points)

An implementation of a special case of a search routine with the precondition and postcondition encoded is given in the file `search.dfy` accompanying this assignment. Give a proof of correctness in Dafny for this search routine. Do not change the preconditions or postconditions. Only add annotations to the body of the method.

Problem 2 (40 points)

Recall the bubble sort example from the second lecture. The goal of this problem is to investigate another property of this algorithm, beyond standard correctness. We want to prove a bound on the maximum number of swaps the algorithm will perform (in order to sort) for any given input. You will find a slightly modified version of the code, which counts and returns the number of swaps. Prove the bound as specified by the postcondition in the provided file.

Problem 3 (40 points)

Consider the following game single player game. Initially, you are given a single stack of n boxes. In each turn of the game, you must perform exactly one of two moves:

1. Pick a stack of $a + b$ boxes and split it into two non-empty stacks of a and b boxes (i.e. $a, b > 0$). This move scores $a \times b$ points.
2. Pick a stack containing only one box and remove it. This actions scores no points.

The game is finished when no more moves are possible. No matter what moves you make, you will eventually run out of turns, and somewhat surprisingly, you will always finish the game scoring exactly $\frac{n(n-1)}{2}$ points. Try out a few runs of the game for some specific games to observe this.

Your task is to prove this outcome for the game. A Dafny encoding of the game is given in the accompanying file `stacking.dfy`. The program is simulating the game. Add the annotations necessary to prove to Dafny that the program (hence the game) terminates and that the postcondition holds. For grading purposes, we divide this assignment into the following two tasks:

(a) (20 points) Prove that the postcondition as specified in `stacking.dfy` and outlined above holds.

(b) (20 points) Prove that the loop terminates by providing the appropriate `decreases` clause for the loop and any argument that may be required to support it so that Dafny has no complaints about termination.

Note: Dafny currently complains about line 14 of the code. This is intentional. You need to add a loop invariant that makes this complaint go away, and then be on your way about proving the problem correct.

This simple program acts *nondeterministically*, but has a *deterministic* outcome. This feature is very commonplace for a lot of concurrent and distributed code. The nature of the hardware platform would mean that many different *moves* may be possible at any point in time by the program, but the programmer fundamentally wants a (single) predictable outcome for the program.

Problem 4 (30 points)

A rabbit is located somewhere on a one-dimensional line, and it moves. It starts at location a. Every second, he moves b units to the right. Therefore, the location of the rabbit at time t is given by the expression $a + t * b$. But, the catch is that $a, b \in \mathbb{N}$ are constants that are unknown to us. We can only know check if the rabbit is at a particular location by querying that location, and we can only query one location at every second.

We can catch the rabbit as follows. The set $\mathbb{N} \times \mathbb{N}$ of pairs of natural numbers is countable [1]. Therefore, there exists a function $unpair : \mathbb{N} \to \mathbb{N} \times \mathbb{N}$ that covers all pairs of natural numbers. [2]

Our strategy is to check the location $x + t \times y$ at time t, where $(x, y) = unpair(t)$. There exists some t_0 such that $(a, b) = unpair(t_0)$. This means that at time t_0 we will check the location $a + t_0 * b$, where the rabbit shall be at time t_0 as well! Therefore, we are guaranteed to catch the rabbit.

Now, you take the high level argument above and the sketch of an implementation that is given to you in file `rabbit.dfy` and turn it into a complete formal argument in Dafny. Complete the implementation of method *unpair* according to our strategy above, and prove that the while loop terminates (i.e. that we eventually catch the rabbit).

Hint: It is much easier to define unpair recursively, instead of defining it as a closed form function. It is also easier to formally reason about such definition in Dafny compared to the closed form.

Note: The purpose is for us to practice using a theorem prover, like Dafny, to do a formal proof that is not related to a program. The puzzle and its solution are effectively provided to you (and you may consult resources online for this as well). We are not asking you to find a solution for the puzzle. We are only asking you to use Dafny to formally prove that the provided solution for the puzzle is indeed correct. One of the main learning objectives of this first chapter of this course is to get you in a comfortable position to think about systems (computer programs or otherwise) in formal terms. This is a generically useful tool for problem solving in computer science.

[1] If you do not know what countable means, then look it up.

[2] Fun fact: this is part of the reasoning for why Dafny can use pairs of natural numbers as a termination measure.