2025 Spring: DSA Labsheet 02: Mapcode and Sorting

Venkatesh Choppella

[2025-01-10 Fri]

Contents

1 An abstract and (practical) approach to iteration

1.1 Discrete Flows

Let's start with a value $x : X$ and repeatedly apply a function $F : X \to X$:

$$tr(x) = [x, F(x), F(F(x)), F^3(x), \ldots]$$

- The sequence $tr(x)$ is called the *trajectory* of x. The trajectory is an infinite sequence.
- The set X is called a *state space*. The individual elements in X are called *states*.
- The total function F is called the *dynamical map*.
- The pair $(X, F : X \to X)$ is called a *discrete flow*.

1.2 Examples of discrete flows:

1.2.1 Counting up:

- $X = \mathbb{N}$,
- $F(x) = x + 1$
- Example trajectories:
  - $tr(2) = [2, 3, 4, 5, \ldots]$
  - $tr(0) = [0, 1, 2, 3, 4, 5, \ldots]$
  - $tr(n) = [n, n + 1, n + 2, \ldots]$

1.2.2 Counting down

- $X = \mathbb{N}$
- $F(x) = \begin{cases} x & \text{if } x = 0 \\ x - 1 & \text{otherwise} \end{cases}$
- Example trajectories:
  - $tr(2) = [2, 1, 0, 0, 0, \ldots]$
  - $tr(5) = [5, 4, 3, 2, 1, 0, 0, 0, \ldots]$
  - $tr(n) = [n, n - 1, n - 2, \ldots, 1, 0, 0, 0]$

How long does the above sequence continue? Forever! But sometimes you end up in a situation when you applying F to a state leaves it unchanged. That is F cannot move x and x is fixed.

1.2.3 Oscillation

- $X = \{True, False\}$
- $F(x) = \neg x$
  - Example trajectories:
    - $tr(True) = [True, False, True, False, \ldots]$
    - $tr(False) = [False, True, False, True, \ldots]$
    - $tr(x) = [x, \neg x, x, \neg x, \ldots]$

Hidden inside every algorithm is a discrete flow waiting to be discovered! Discovering or designing this discrete flow is key to designing the algorithm.

1.3 Fixed and Convergent points

Note that while some trajectories are infinitely long, some of them 'terminate' after reaching a value, while others oscillate or, in general, could get into a cycle from which they don't come out.

A fixed point of F is a state x such that $x = F(x)$.

We are interested in the case when a trajectory reaches a fixed point, i.e., a value $x^* = F^n(x)$ for some n such that $F(x^*) = x^*$. Then the trajectory of x would look something like this

$$tr(x) = [x, F(x), \ldots, x^* = F^n(x), x^*, x^*, \ldots]$$

The set of all fixed points of is written $fix(F)$.

It's also possible that for some state x, its trajectory always reaches a fixed point. Such a state is called a *convergent state*. The set of convergent states of F is written $con(F)$.

Note that if x is convergent, then its trajectory may be truncated to stop after reaching the fixed point x^*. No information is lost.

2 Mapcode Framework for Algorithmic Problem Solving

Mapcode (Viswanath, 2008) is a systematic method for designing algorithms for problem solving. It is a 10 step process to construct an algorithm and show that it terminates with the correct answer. Please consult the mapcode book available in the library.

2.1 What is an algorithmic problem?

An algorithmic problem consists of entities:

1. **Instance space** I: The set of problem instances.
2. **Answer space** A: The set of possible answers.
3. **Specification map** $f : I \to A$: The specification map specifies what to compute. It is modeled as a function that needs to be implemented as an algorithm.
4. **Resource constraints**: The set of resources (datatypes and operations) available to implement the specification map. Usually, these are familiar types like numbers, arrays, and operations like addition, array lookups and updates, etc., but they could be anything you want.

1.	Instance space	I	What to input to the computation?
2.	Answer space	A	What is the result of the computation?
3.	Specification map	f	What to compute?
4.	Resource constraints		What to compute with?

In other words, the problems for which we are interested in designing algorithms need to be first conceptualised as total functions from an instance space to an answer space. Their implementation is constrained by the availability of suitable data types and operations on them.

2.2 Discrete Flow of an algorithm

An algorithm is implemented by using a set of variables and performing some operations on them. In the simplest case, we can club all the variables values together into a state space X

5	State Space	X	What are program variables and their types?

2.3 The initialization

How do we initialize the state space from the instance? This is captured by the initialization map

$$\rho : I \to X$$

6	Initialization map	ρ	How to start the computation?

2.4 Extracting the answer

How do we extract the answer component from a state? This is captured by the answer map

$$\pi : X \to I$$

7	Answer map	π	How to extract the answer?

2.5 Iterating through the computation

What is the dynamical map that drives the computation?

8	Dynamical Map	$F : X \to X$	What is computed in each step of the iteration?

2.6 Termination

9	Termination	$\rho[I] \subseteq con(F)$	How do we ensure that the computation terminates?

2.7 Correctness

10	Correctness	$f(i) = \pi(loop(F, \rho(i)))$	Assuming the computation terminates how do we know we have the right answer?

3 Problem: Hemachandra-Fibonacci series

Use the mapcode framework to design an algorithm to compute the nth Hemachandra-Fibonacci number according to the recurrence relation:

$$\begin{aligned} fib(0) &= 1 \\ fib(1) &= 1 \\ fib(n+2) &= fib(n) + fib(n+1) \end{aligned}$$

Implement the following:

1. Identify the problem instance space I: Nat
2. Identify the answer space A: Nat

3. Design the state space X: Nat x Nat x Nat
4. Define the init map $\rho : I \to X$

$$\rho(n) = (1, 1, n)$$

5. Define the answer map $\pi : X \to A$

$$\pi(a, b, i) = a$$

6. Define the dynamical map $F : X \to X$

$$F(a, b, i) = \begin{cases} (a, b, i) & \text{if } i <= 1 \\ (a + b, a, i - 1) & \text{otherwise} \end{cases}$$

7. Define $f : I \to A$ using the template

$$fib(n) = \pi(loop(F, \rho(n)))$$

4 Bubblesort

Your goal is to implement bubblesort in an incremental manner.

Bubblesort is similar to selection sort in that we sweep the array a[0..b-1] to locate the maximum. Unlike in selection sort, the sweeping is accompanied with swapping so that in the end, the maximum in this range 'bubbles up' to the position b-1.

Adapt the selection sort code to implement bubblesort:

- Step 1: Replace findmax with bubble using the mapcode framework
- Step 2: Implement bubblesort using the mapcode framework