Fast and Slow Pointers: Introduction

Let's go over the Fast and Slow Pointers pattern, its real-world applications, and some problems we can solve with it.

We'll cover the following

Overview

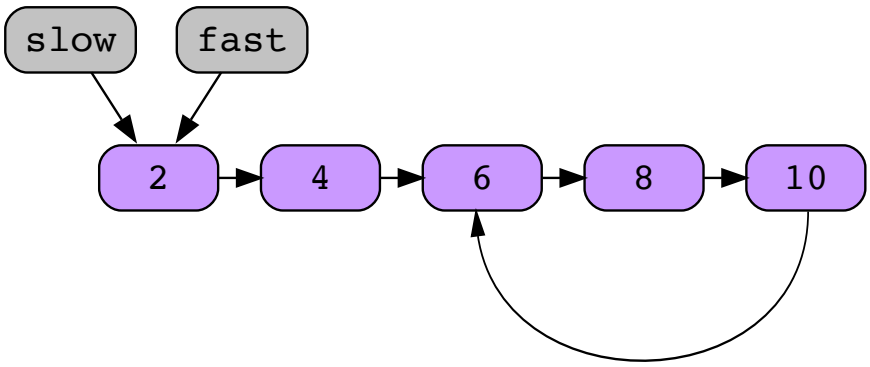
Similar to the two pointers pattern, the **fast and slow pointers** pattern uses two pointers to traverse an iterable data structure at different speeds. It's usually used to identify distinguishable features of directional data structures, such as a linked list or an array.

The pointers can be used to traverse the array or list in either direction, however, one moves faster than the other. Generally, the slow pointer moves forward by a factor of one, and the fast pointer moves by a factor of two in each step. However, the speed can be adjusted according to the problem statement.

Unlike the two pointers approach, which is concerned with data values, the fast and slow pointers approach is used to determine data structure traits using indices in arrays or node pointers in linked lists. The approach is commonly used to detect cycles in the given data structure, so it's also known as Floyd's cycle detection algorithm.

The key idea is that the pointers start at the same location, but they move forward at different speeds. If there is a cycle, the two are bound to meet at some point in the traversal. To understand the concept, think of two runners on a track. While they start from the same point, they have different running speeds. If the race track is a circle, the faster runner will overtake the slower one after completing a lap. On the other hand, if the track is straight, the faster runner will end the race before the slower one, hence never meeting on the track again. The fast and slow pointers pattern uses the same intuition.

Here's how the pointers move along a list with a cycle:



Increment the **slow** pointer by 1 and the **fast** pointer by 2.
The curved arrow represents a cycle in the linked list.

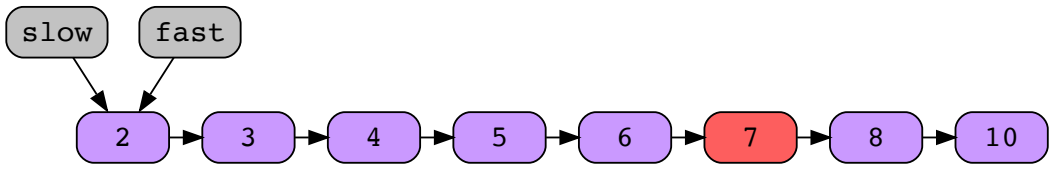
1 of 4

Examples

The following example illustrates a problem that can be solved with this approach:

Delete nth node from the end of a list

n = 3



1 of 7

Does my problem match this pattern?

- Yes, if either of these conditions is fulfilled:
  - Either as an intermediate step, or as the final solution, the problem requires identifying:
    - the first x % of the elements in a linked list, or,
    - the element at the k-way point in a linked list, for example, the middle element, or the element at the start of the second quartile, etc.
    - the k^{th} last element in a linked list
  - Solving the problem requires detecting the presence of a cycle in a linked list.
  - Solving the problem requires detecting the presence of a cycle in a sequence of symbols.
- No, if either of these conditions is fulfilled:
  - The input data cannot be traversed in a linear fashion, that is, it's neither in an array, nor in a linked list, nor in a string of characters.
  - The problem can be solved with two pointers traversing an arr or a linked list at the same pace.

Real-world problems

Many problems in the real world use the fast and slow pointers pattern. Let's look at some examples.

- **Symlink verification:** Fast and slow pointers can be used in a symlink verification utility in an operating system. A symbolic link, or symlink, is simply a shortcut to another file. Essentially, it's a file that points to another file. Symlinks can easily create loops or cycles where shortcuts point to each other. To avoid such occurrences, a and slow pointers can detect a loop in the symlinks by moving along the connected files or directories at different speeds.
- **Compiling an object-oriented program:** Usually, programs are not contained in a single file. Particularly, for large applications, modules can be divided into different files for better maintenance. Dependency relationships are then defined to specify the order of compilation for these files. However, sometimes, there might be cyclic dependencies that can lead to an error. Fast and slow pointers can be used to identify and remove these cycles for seamless compilation and execution of the program.

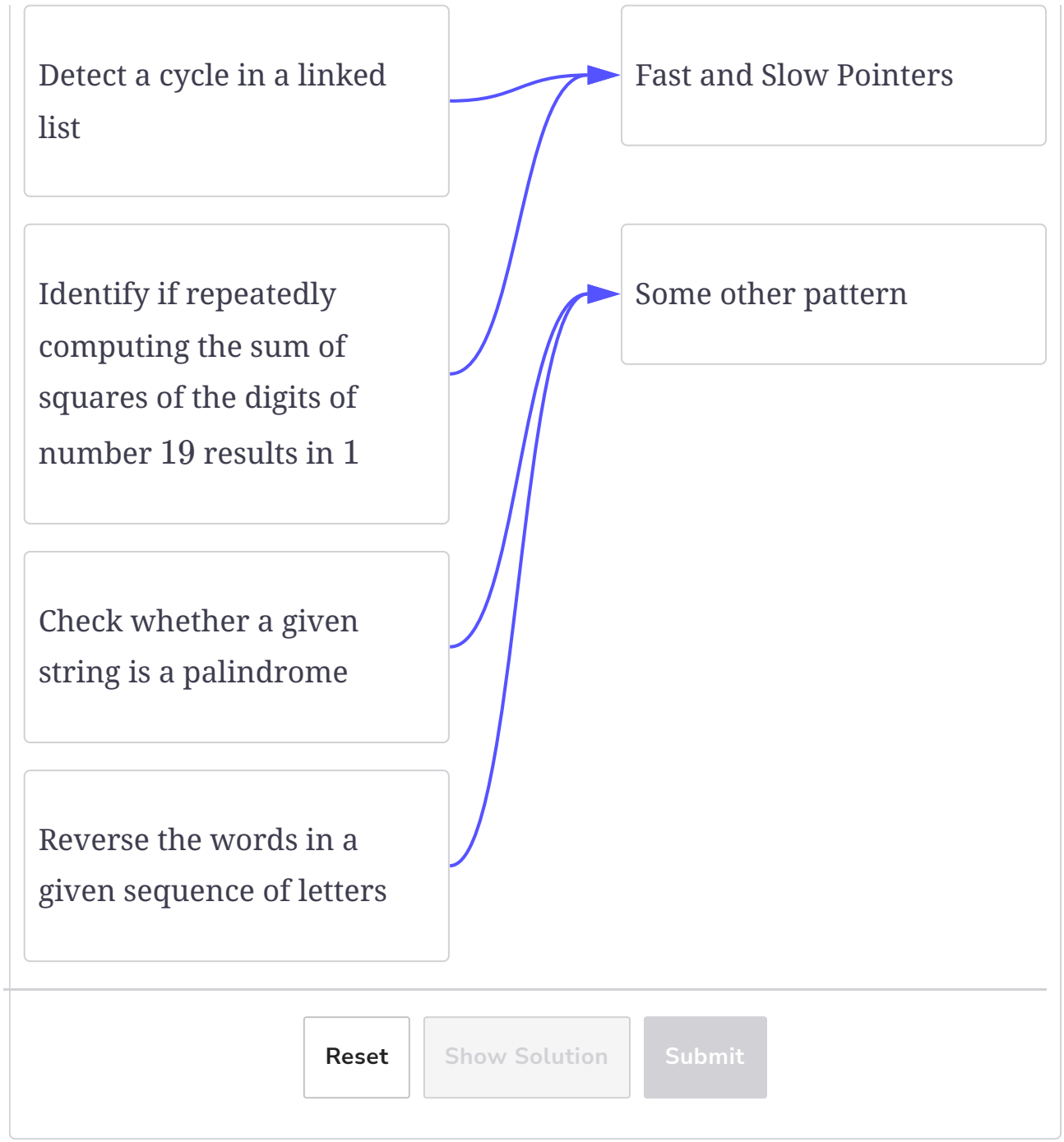
Strategy time!

Match the problems that can be solved using the fast and slow pointers pattern.

> **Note:** Select a problem in the left-hand column by clicking it, and then click one of the two options in the right-hand column.

Match The Answer

Select an option from the left-hand side



Reset | Show Solution | Submit

Back

Valid Palindrome II

Next

Happy Number