Mastery II — Data Structures (T. III/21–22)

Directions:

- This mastery examination starts at 2pm on Thursday July 14, 2022. You have until 6pm of the same day to complete the following *four* problems.
- **WHAT IS PERMITTED:** The exam is open- book, notes, Internet, Stackoverflow (and other similar professional sites), etc. Use the Internet in *read-only* mode.
- **WHAT IS *NOT* PERMITTED:** Communication/collaboration of any kind. Using answers from study-aid websites (e.g., Chegg). Obviously, asking a question online is strictly *not* allowed.
- We're providing a starter package. The fact that you're reading this PDF means you have successfully downloaded the starter pack.
- At the top level, the starter package contains 4 folders, one for each problem. We're supplying a single `build.gradle` for the whole mastery.
  To save you some typing, we're also supplying some JUnit tests and driver code in the starter pack. Passing these tests doesn't mean you will pass our further testing. Failing them, however, means you will not pass the real one.

Handing In: To hand in, zip ***only*** the following files as `mastery2.zip` and upload your zip file to Canvas:

`NearFib.java` `PaSBACount.java` `PerfectHiding.java` `RecursivePal.java`

This means your solution code should only be in these files and nowhere else.

Problem 1: Recursive Palindrome (10 points)

An array is a *palindrome* if it reads the same forward and backward. For example, `{4, 0, 4}` is a palindrome but `{2, 0, 4}` is not. From this definition, we know that if `a` is a palindrome, then the first half (precisely the first `len(a)//2` elements) is the reverse of the last half (precisely the last `len(a)//2` elements). Importantly, comparisons are made using the corresponding object's `.equals`.

...let's up the game a bit. An array `a` is said to be a *recursive palindrome* if (i) the array is a palindrome itself and (ii) its first half (i.e., the first `len(a)//2` element) is a *recursive palindrome* or empty. Indeed, notice that how the recursive palindrome definition is recursive. For example:

- `{1, 1, 5, 1, 1}` is a recursive palindrome. Note that the whole array is a palindrome. Further, the first half (i.e., `1, 1`) is a recursive palindrome.
- `{7, 8, 7, 7, 8, 7}` is a recursive palindrome.
- `{2, 0, 4, 0, 2}` is *not* a recursive palindrome. While the whole array is a palindrome, the first half (i.e., `2, 0`) is not a palindrome—and hence *not* a recursive palindrome.
- `{7, 4, 5, 5, 4, 7, 9, 7, 4, 5, 5, 4, 7}` is *not* a recursive palindrome.

Your Task: Inside the class `RecursivePal<T>`, write a public method

```
public class RecursivePal<T> {
    public boolean isRecursivePalindrome(T[] a) { ... }
}
```

that takes as input an array `a` and returns a boolean indicating whether `a` is a recursive palindrome.

Expectations: Promises, Constraints, and Grading

- $1 \leq$ `a.length` $\leq 50,000,000$.
- For full-credit, your code must run in $O(n)$ time, where $n =$ `a.length`. Your code must finish within 1 second on each of the test cases.
- You can write as many helper functions as you'd like.

Problem 2: Near Fibonacci (10 points)

The Fibonacci recurrence is given by

$$f_n = f_{n-1} + f_{n-2},$$

where $f_1 = f_2 = 1$. Therefore, the first several Fibonacci numbers are

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...

Let a nonnegative integer `t` be called a *threshold*. We say that a number `m` is a near Fibonacci number if there is a Fibonacci number `f` such that `abs(m - f)<= t`. In other words, that number `m` is within `t` from a Fibonacci number.

For example, if we use a threshold of `t = 3`, the number 57 is a near Fibonacci because 55 (a Fibonacci number) is only 2 away. By the same reasoning, 86 is also a near Fibonacci number (within 3 from 89). But 45 and 78 are *not*.

Your Task: Inside the class `NearFib`, write a public method

```
public class NearFib {
    public int numNearFib(long[] a, long t) { ... }
}
```

that takes as input an array `a` of numbers and a threshold `t`, and returns the count of the number in `a` that is a near Fibonacci number when the threshold is `t`.

Example: Using `long[] a = {5, 3, 6, 10, 19, 25, 111};`, the answer to `numNearFib(a, 2L)` is 5. This is because 5, 3, 6, 10, and 19 are near Fibonacci numbers.

There are more sample test cases in the `test` directory.

Expectations: Promises, Constraints, and Grading

- $1 \leq$ `a.length` $\leq 20,000,000$.
- $0 \leq$ `a[i]` $\leq 1,000,000,000,000$ and $0 \leq$ `t` $\leq 1,000,000,000,000$. This range does fit in a **`long`**.
- Your code must finish within 2 seconds on each of the test cases.
- You can write as many helper functions as you'd like. You can also write a constructor.

(*Hint:* You may find the `TreeSet` useful.)

Problem 3: The Count of Ones (10 points)

We will count the total of number of ones in an array of packed sorted bit-arrays (PaSBA). A sorted bit-array satisfies the following interface:

```
public interface SortedBitArray {
    // returns the value at index k.
    int get(int k);

    // returns the length of the array.
    int length();
}
```

When a bit-array is sorted, all the 0s come before all the 1s. Storing a bit-array of length n generally requires $O(n)$ space. However, when a bit-array is sorted, it is possible to store it using much less memory, for example, by storing the index at which the run of 0s ends.

A *packed* sorted bit-array (PaSBA), given as a `PackedSortedBitArray` class, satisfies the `SortedBitArray` interface and further guarantees that both its `.get` and `.length` run in worst-case $O(1)$ time.

Your Task: Without modifying the given `PackedSortedBitArray` class or `SortedBitArray` interface, implement the following static method inside a class `PaSBACount`:

```
public static long count(SortedBitArray[] packedArrays)
```

which counts the total number of 1s in the whole array of PaSBAs.

To test your code, you will likely want to use the `PackedSortedBitArray` in the starter pack. The constructor `PackedSortedBitArray(int n, int k)` makes an instance of length `n` where the first index of a 1 is `k`. As an example, the array of PaSBAs on the left compactly encodes the arrays on the right (which are never constructed for real).

```
packedArrays = {
    new PackedSortedBitArray(5, 2),
    new PackedSortedBitArray(3, 1),
    new PackedSortedBitArray(4, 2),
    new PackedSortedBitArray(7, 3)
};
```

```
{
    {0, 0, 1, 1, 1},
    {0, 1, 1},
    {0, 0, 1, 1},
    {0, 0, 0, 1, 1, 1, 1}
}
```

In this example, `packedArrays[1].get(1)` will return 1 and `packedArrays[3].get(2)` will return 0. Furthermore, if we run `count` on this `packedArrays`, the return value should be $3+2+2+4=11$, which is the total number of 1s in the above.

Expectations: Promises, Constraints, and Grading

- $0 \leq$ `packedArrays.length` $\leq 100,000$. The length of each PaSBA—i.e., `packedArrays[i].length()`—is between 0 and $1,000,000,000$.
- For each `SortedBitArray` instance given as part of the input, its `.get` and `.length` run in worst-case $O(1)$ time. It will *not* be any slower than the `PackedSortedBitArray` supplied in your starter pack.
- For full credit, your code must run in $O(n \log k)$ or faster, where n is `packedArrays.length` and k is the length of the longest PaSBA. Your code must finish within 2 seconds on each call to `count`. (*Hint:* some variant of binary search may be useful.)
- Although the number of 1s in each PaSBA will fit in a Java **`int`**, the grand total may be significantly larger than what could be stored in an **`int`**. Notice that the return type of the function is a **`long`**.

Problem 4: The Perfect Hiding Spot (10 points)

The CS student committee for recreation and well-being is renting a huge castle for a week-long party this summer. The entrance opens into Chamber 1. In this chamber, they're planning to set up several large TV monitors so they can play console games without seeing daylight.

Not everyone is a console gamer, though. Your task is to help find the perfect hiding spot for people who seek solitude from the gaming crowd. You will be presented with a map of the castle. The chambers (vertices) are numbered 1 through n. You'll be given a list of one-directional passages from one chamber to another (directed edges). Every passage is labeled with its distance. Importantly, each chamber has exactly one passage that opens into it. More precisely, the input is a directed tree where every vertex, except for vertex 1, has precisely one incoming edge. Note: vertex 1 has no incoming edge.

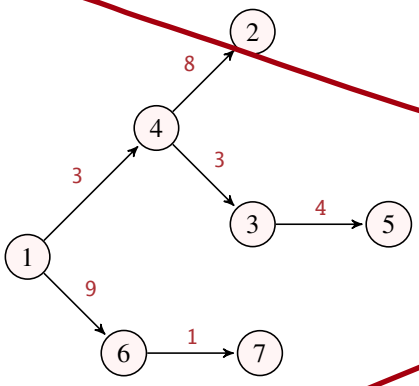
Your Task: Inside the class `PerfectHiding`, you are to implement a static method

```
public static int bestSpotDistance(List<WeightedEdge> passages)
```

where `passages` is a list of directed edges. Because this structure is a tree, the length of the list `passages` is always $n-1$, so you know n from the length of this list. If `e` is a `WeightedEdge` in this list, then there is a passage from Chamber `e.first` to Chamber `e.second` of length `e.cost` meters.

Your static method will compute the distance to the chamber (vertex) farthest away from chamber number 1. If it turns out that you cannot reach any chamber other than 1, the answer will be 0.

Example:



This graph has $n = 7$ nodes and the edges are:

```
(1, 4, 3)   (4, 2, 8)   (4, 3, 3)
(3, 5, 4)   (1, 6, 9)   (6, 7, 1)
```

Here, the notation (u, v, c) denotes a directed edge from u to v with a weight of c meters. Calling `bestSpotDistance` on this input will result in the number 11 because Chamber 2 is the farthest from 1 via $1 \to 4 \to 2$, totaling $3 + 8$ meters.

Expectations: Promises, Constraints, and Grading We'll test your program with n up to $100,000$. On this size of input, your program must finish within 3 seconds to receive full credit. The cost of an edge will be a number between 1 and $1,000$ (inclusive).