Homework 2: due by 11:59pm on February 18, 2022 (Total: 100 points)

Zoran Budimlić, Mackale Joyner

Commit and push all work to the GitHub repo at https://classroom.github.com/a/6TSCx57_. In case of problems committing your files, please contact the teaching staff at comp322-staff@mailman.rice.edu before the deadline to get help resolving your issues.

The slip day policy for COMP 322 is similar to that of COMP 321. All students will be given 3 slip days to use throughout the semester. When you use a slip day, you will receive up to 24 additional hours to complete the assignment. You may use these slip days in any way you see fit (3 days on one assignment, 1 day each on 3 assignments, etc.). If you plan to use a slip day, you need to say so in a committed README.md file before the deadline. You should specifically mention how many slip days you plan to use.

Other than slip days, no extensions will be given unless there are exceptional circumstances (such as severe sickness, not because you have too much other work). Such extensions must be requested and approved by the instructor (via e-mail, phone, or in person) before the due date for the assignment. Last minute requests are likely to be denied.

If you see an ambiguity or inconsistency in a question, please seek a clarification on Piazza (remember not to share homework solutions in public posts) or from the teaching staff. If it is not resolved through those channels, you should state the ambiguity/inconsistency that you see, as well as any assumptions that you make to resolve it.

Honor Code Policy: All submitted homeworks are expected to be the result of your individual effort. You are free to discuss course material and approaches to problems with your other classmates, the teaching assistants and the professor, but you should never misrepresent someone else's work as your own. If you use any material from external sources, you must provide proper attribution.

1 Written Assignments (45 points)

Your solution to the written assignment should be submitted as a PDF file named `hw2_written.pdf` in the hw2 directory. This is important — you will be penalized 10 points if you place the file in some other folder or with some other name. The PDF file can be created however you choose. If you scan handwritten text, make sure that the writing is clearly legible in the scanned copy. Your solution to the programming assignment should be submitted in the appropriate location in the hw2 directory.

1.1 Parallelizing the *IsKColorable* Algorithm for Undirected Graphs (25 points)

Consider the pseudo-code of the **IsKColorable** algorithm for undirected graphs shown in Algorithm 1 that you learned in COMP 182. *Your assignment is as follows:*

1. *(10 points) Assume that every comparison operator executed in line 4 takes 1 unit of work, and everything else has zero cost. Create a parallel version of this algorithm using parallel constructs and data structures of your choice that we have learned so far (future tasks, data-driven tasks, async/finish, futures, promises) so as to maximize parallelism, while ensuring that the parallel version always computes the same result as the sequential version. You may change data structures (e.g., replace a scalar variable by an array) if needed. You are* **not** *allowed to directly modify shared memory locations from different tasks (i.e. setting* colorable *to* True *or* False *in different tasks is not allowed). You* **are** *allowed to use functional means of communication between tasks (futures, promises). Also, different tasks modifying different parts of the same array is allowed as well.*
2. *(5 points) What is the big-O for the total WORK that your algorithm performs?*

3. *(5 points) Assume that every comparison operator executed in line 4 takes 1 unit of work, and everything else has zero cost. Is it possible for your parallel algorithm to have a* **larger** *value of WORK than the sequential version for a given set of inputs? If so, describe an example when this can happen. If not, explain why not.*

4. *(5 points) Assume that every comparison operator executed in line 4 takes 1 unit of work, and everything else has zero cost. Is it possible for your parallel algorithm to have a* **smaller** *value of WORK than the sequential version for a given set of inputs? If so, describe an example when this can happen. If not, explain why not.*

Algorithm 1: IsKColorable

Input: Graph $g = (V, E)$ and $k \in \mathbb{N}$.
Output: $\exists f : V \to [k], \forall \{u, v\} \in E, f(u) \neq f(v)$.

```
1 foreach Assignment f of a value in [k] to each node in V do
2     colorable ← True;
3     foreach {u, v} ∈ E do
4         if f(u) = f(v) then
5             colorable ← False;
6             break;
7     if colorable = True then
8         return True;
9 return False;
```

1.2 Parallel Fibonacci using Futures (20 points)

We will now revisit the Fibonacci computation from HW1, but this time we will use futures to compute the Fibonacci function. The parallel implementation of Fibonacci using futures is shown below in Listing 1.

1. What is the big-O of the total amount of Work performed by this algorithm? You are allowed to consult the correct solution from HW1 if it helps you answer this. (5 points)

2. Perform a theoretical big-O analysis for the *critical path length* for a call to fib(n). Include an explanation of the analysis, and state what expression you get for CPL(n) as a function of n. (10 points)

3. Bob the COMP 322 TA had a brilliant idea that we should improve this parallel Fibonacci implementation through memoization using futures. His implementation is given below in Listing 2. What is the CPL(n) of Bob's algorithm (for $n < MaxMemo$), for the first-ever call to *fib*? Is Bob's faster, same, or slower than the sequential Memoized Fibonacci from HW1? You can assume unlimited amount of cores available for the computation. Explain your answer. (5 points)

```
1     public static int fib(int n) throws SuspendableException {
2         doWork(1); // every call to fib counts as 1 unit of work
3
4         if (n <= 0) return 0;
5         else if (n == 1) return 1;
6
7         var f1 = future(() -> fib(n - 1));
8         var f2 = future(() -> fib(n - 2));
9
10        return f1.get() + f2.get();
11    }
```

Listing 1: Parallel Fibonacci using Futures

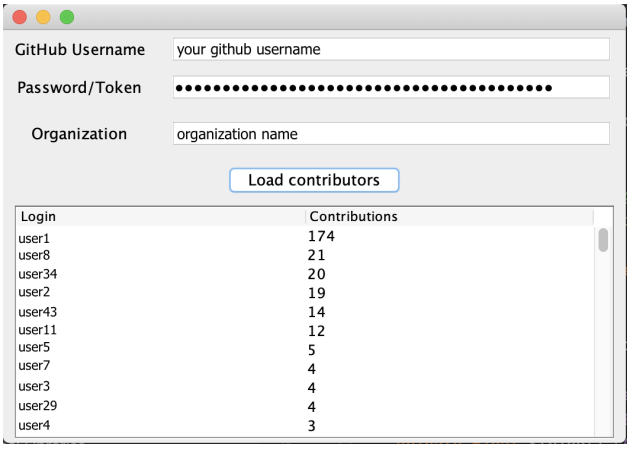
```
1  public class MemoizedParallelFib {
2    private static final int MaxMemo = 1000000; // max memoized results
3    private static final HjFuture<Integer>[] memoized = // array of futures
4      Stream.iterate(0, i -> i+1)    // Stream (0, 1, 2, ... )
5          .limit(MaxMemo)            // Stream (0, 1, 2, ... MaxMemo-1)
6          .map(e->future(()->fib(e)))// Stream of futures to compute
7                                     // (fib(0), fib(1), ... fib(MaxMemo-1))
8          .toArray(HjFuture[]::new); // convert to array
9
10   public static int fib(int n) throws SuspendableException {
11     doWork(1); // every call to fib counts as 1 unit of work
12     if (n <= 0) return 0;
13     else if (n == 1) return 1;
14     else if (n >= MaxMemo) return fib(n - 1) + fib(n - 2);
15     else return memoized[n-1].get() + memoized[n-2].get();
16   }
17 }
```

Listing 2: Memoized Parallel Fibonacci. ©"Bob the COMP 322 TA".

2 Programming Assignment: GitHub Contributions (55 points)

2.1 Introduction

In this assignment, you will implement a concurrent version of a program that fetches and displays all contributors and their total contributions across all repositories under a given organization on GitHub.



The code in your GitHub repository implements a sequential version of this program that causes the user-interface to block when the *Load Contributors* button is clicked. For making the GitHub requests, follow the steps below to generate a token for your account. To test your code, you may find it helpful to use the following organizations that contain sample data:

- revelation
- galaxycats
- moneyspyder
- notch8

Other organizations are provided in the unit tests.

2.2 Setup

See the Lab 1 handout for instructions on HJlib installation for use in this homework. The provided code for HW 2 can be found in your GitHub repository at https://classroom.github.com/a/6TSCx57_.

You can generate a GitHub developer token at https://github.com/settings/tokens/new. Select the repo scope and click Generate Token. Copy and save the token in a safe place.

To start up the program open the main class and click the green arrow to start up the user-interface. Specify your github account username, the token generated above, and an organization from the list above. When you click the Load Contributors button, notice that the user-interface freezes, the button remains selected and you are unable to click in the text fields until the data is loaded.

2.3 Abstract Overhead

Thus far, our abstract metrics have assumed an idealized execution in which there is no overhead in creating *async* or *future* tasks. In the real world, asyncs are not actually free: they consume processor cycles and system memory. In this homework, we will simulate that cost using abstract metrics, and try to implement a parallel algorithm in such a way as to obtain the best work value when taking abstract overheads into account.

2.4 Parallel GitHub Contributions

The goal of this assignment is to implement a parallel GitHub contributions program with two objectives. The user-interface must not freeze when loading the data and all requests and processing of the results must be performed concurrently. Your parallel solution must use futures, data-driven tasks, and streams. All aggregation and post-processing of the results must be performed concurrently using streams.

A correct parallel program should generate the same output as the sequential version, and should not exhibit any data races. It should pass the unit tests provided, and other tests that the teaching staff may use while grading. You are free to add any tests or other code you like under the main/ and test/ directories, but please do not modify the folder structure of the project. Be sure to change the *loadContributorsSeq* function call in the *ContributorsUI* class to your *loadContributorsPar* implementation.

For testing, change the username field in the ContributorsTest class to your GitHub username and the token field to you GitHub token generated above.

2.5 Submitting

To submit, you will need to add, commit, and push all work to the GitHub repository. The repository is located at https://classroom.github.com/a/6TSCx57_. Please open a browser, navigate to the url, and verify that you successfully committed your homework. Don't forget to add a README.md file if you plan to use slip days.

Your submission should include the following in the hw2 directory:

1. 40 points for your completed parallel solution that does not freeze the user-interface and loads and processes all the data concurrently.
2. 5 points for coding style and documentation. We have provided the basic checkstyle rules as part of your Maven project. At a minimum, all code should include basic documentation for each method in each class. You are also welcome to add additional unit tests to test corner cases and ensure the correctness of your implementation.
3. 10 points for a report file formatted as a PDF file named `hw2_report.pdf` in the hw2 directory. The report should contain the following:
   (a) A summary of your parallel algorithm, and the steps that you had to take to unfreeze the user-interface and process the results concurrently.

(b) An explanation as to why you believe that your implementation is correct and data-race-free.

(c) An explanation of what value of work (as a function of N) you expect to see from your implementation, and why.

Hints based on common errors/omissions from past years: Be sure to explain what parallel constructs you used, why, and what subcomputations ran in parallel as a result. Also, justify why you believe that your parallel solution will always produce the same output as the sequential version. In addition, you should explain why there are no data races in your solution. Finally remember to explain what value of work (as a function of N) you expect to see from your implementation, and why.