Comp 251: Assignment 2

Answers must be returned online by March 7th (11:59:59pm), 2024.

General instructions (Read carefully!)

- **Important:** All of the work you submit must be done by only you, and your work must not be submitted by someone else. Plagiarism is academic fraud and is taken very seriously. For Comp251, we will use software that compares programs for evidence of similar code. This software is very effective and it is able to identify similarities in the code even if you change the name of your variables and the position of your functions. The time that you will spend modifying your code, would be better invested in creating an original solution.

  Please don't copy. We want you to succeed and are here to help. Here are a couple of general guidelines to help you avoid plagiarism:

  Never look at another assignment solution, whether it is on paper or on the computer screen. Never share your assignment solution with another student. This applies to all drafts of a solution and to incomplete solutions. If you find code on the web, or get code from a private tutor, that solves part or all of an assignment, do not use or submit any part of it! A large percentage of the academic offenses in CS involve students who have never met, and who just happened to find the same solution online, or work with the same tutor. If you find a solution, someone else will too. The easiest way to avoid plagiarism is to only discuss a piece of work with the Comp251 TAs, the CS Help Centre TAs, or the COMP 251 instructors.

- Your solution must be submitted electronically on ed-Lessons.

- To some extent, collaborations are allowed. These collaborations should not go as far as sharing code or giving away the answer. You must indicate on your assignments (i.e. as a comment at the beginning of your java source file) the names of the people with whom you collaborated or discussed your assignments (including members of the course staff). If you did not collaborate with anyone, you write "No collaborators". If asked, you should be able to orally explain your solution to a member of the course staff. At the end of this document, you will find a check-list of the behaviours/actions that are allowed during the development of this assignment.

- This assignment is due on March 7^{th} at 11h59:59 pm. It is your responsibility to guarantee that your assignment is submitted on time. We do not cover technical issues or unexpected difficulties you may encounter. Last minute submissions are at your own risk.

- This assignment includes a programming component, which counts for 100% of the grade, and an optional long answer component designed to prepare you for the exams. This component will not be graded, but a solution guide will be published.

- Multiple submissions are allowed before the deadline. We will only grade the last submitted file. Therefore, we encourage you to submit as early as possible a preliminary version of your solution to avoid any last minute issue.
- Late submissions can be submitted for 24 hours after the deadline, and will receive a flat penalty of 20%. We will not accept any submission more than 24 hours after the deadline. The submission site will be closed, and there will be no exceptions, except medical.
- In exceptional circumstances, we can grant a small extension of the deadline (e.g. 24h) for medical reasons only.
- Violation of any of the rules above may result in penalties or even absence of grading. If anything is unclear, it is up to you to clarify it by asking either directly the course staff during office hours, by email at (`cs251-winter@cs.mcgill.ca`) or on the discussion board on our discussion board (recommended). Please, note that we reserve the right to make specific/targeted announcements affecting/extending these rules in class and/or on one of the communication channels used in the course. It is your responsibility to monitor MyCourses and the discussion board for announcements.
- The course staff will answer questions about the assignment during office hours or in the online forum. We urge you to ask your questions as early as possible. We cannot guarantee that questions asked less than 24h before the submission deadline will be answered in time. In particular, we will not answer individual emails about the assignment that are sent the day of the deadline.

Programming component

- You are provided some starter code that you should fill in as requested. Add your code only where you are instructed to do so. You can add some helper methods. Do not modify the code in any other way and in particular, do not change the methods or constructors that are already given to you, do not import extra code and do not touch the method headers. The format that you see on the provided code is the only format accepted for programming questions. **Any failure to comply with these rules will result in an automatic 0.**
- Public tests cases are available on ed-Lessons. You can run them on your code at any time. If your code fails those tests, it means that there is a mistake somewhere. Even if your code passes those tests, it may still contain some errors. We will grade your code with a more challenging, private set of test cases. We therefore highly encourage you to modify that tester class, expand it and share it with other students on the discussion board. Do not include it in your submission.
- Your code should be properly commented and indented.
- **Do not change or alter the name of the files you must submit, or the method headers in these files**. Files with the wrong name will not be graded. Make sure you are not changing file names by duplicating them. For example, main (2).java will not be graded.
- **Do not add any package or import statement that is not already provided**
- Please submit only the individual files requested.
- **You will automatically get 0 if the files you submitted on ed-Lessons do not compile, since you can ensure yourself that they do. Note that public test cases do not cover every situation and your code may crash when tested on a method**

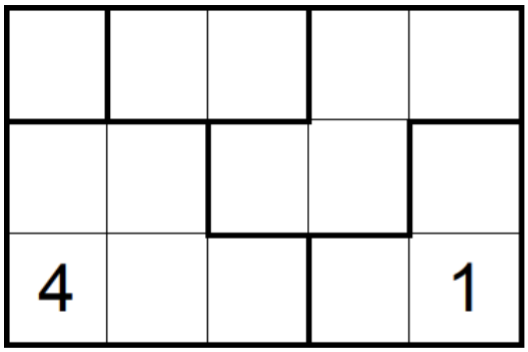
that is not checked by the public tests. This is why you need to add your own test cases and compile and run your code from command line on linux.

Homework

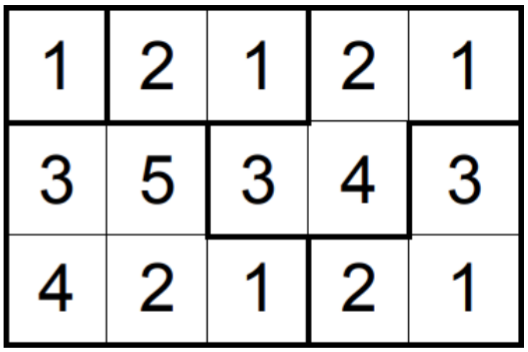
Exercise 1 (60 points)**.** ***Complete Search***

Normally, in a sudoku, you are given a 9×9 grid with some cells filled with numbers and others with empty values. The normal job is to replace the empty cells with a digit, such the grid becomes valid.The grid is valid if every row, every column, and every 3x3 sub-grid contains exactly 9 distinct numbers (1 through 9) with no duplicate numbers. To make things a bit more interesting, for this question of the assignment, we will work on a modification of the sudoku game. As in Sudoku, this modification will place numbers in a grid (where the number of rows (R) and columns (C) will be in the following limits: $1 \leq R, C \leq 7$); but this grid will not be constrained to size 9 X 9. Furthermore, the grid will have outlined regions (not constrained to be a 3X3 region as in the standard sudoku) that must contain the numbers from 1 to n, where n is the number of the cells (i.e., squares) in the region. The most interesting new feature comes from the fact that the same number can never touch itself, not even diagonally.

The figure below depicts the initial incomplete grid and the grid with an accepted solution. Please note that this solution is unique (i.e., there is no other solution to the puzzle based on the described constrains). Furthermore, for this assignment, you are guaranteed that your solution will be tested against problems that have a unique solution.



Incomplete Grid



Solution Grid

We will model the problem in java using an Object Oriented Approach. Particularly, we will model the Sudoku `Board` as an object that has regions (i.e., the outlined areas in the board) and a values (i.e., and `int[][]` matrix containing the digits in each cell). Each `Region` is an object with two private attributes 1) An array of `Cells` (i.e., the squares of the grid) and 2) an `int` representing the number of cells. Finally, a `Cell` object will represent each square in the board, where the private attributes of the class denotes the position of the square in the board. Each of those classes has a set of getters and setter methods to access/modify their private information.

The above mentioned classes have been coded inside the Java Class `Game`. This class has a private attribute named `sudoku` of type `Board` that will contain all the initial information needed to complete the Sudoku puzzle. Inside the Class `Game`, you will also find the function `solver`. The signature of the function `solver` in the java file `Game.java` is as follows.

```
public int[][] solver() {
    //To Do => Please start coding your solution here
    return sudoku.getValues();
}
```

Your job for this part of the assignment is to complete the function `solver`, which takes an incomplete puzzle grid (stored initially in the `board_values` attribute of the object `sudoku`) and returns the output solution grid (i.e., the same `board_values` attribute of the object `sudoku`, but this time with all the cells filled). Please notice that a cell in the board is empty if it contains a 0 (i.e., zero value).

Note: method signatures

You should not change any of the method signatures provided in the template (because we need them for the auto-grader calls); however, you can (i) change the method bodies, (ii) add new fields and/or methods to any of the classes, (iii) create as many helper functions as you need. It is your responsibility to guarantee that any change that you are producing to our template does not affect the auto-grader.

Note: main function

We have already implemented a `main` function to read the data from the files, to create all the initial information (i.e., the incomplete grid and the description of its separate regions) and finally to create the `Board` object called `sudoku` (which will contain all the initial information of the sudoku game). Please note that this function will not be graded, and it is there only to make sure that all of the Comp251 students understand the input of the problem and to test their own code.

Exercise 2 (40 points)**.** ***Divide and Conquer***

During our Comp251 lectures, we defined the Fibonacci sequence as follows:
$fib_1 = 1$
$fib_2 = 2$
$fib_n = fib_{n-2} + fib_{n-1}$
Generating the following sequence $1, 1, 2, 3, 5, 8, 13, 21 \ldots$

We can also define the Fibonacci sequence using letters. For example, if:
$fib_1 = X$
$fib_2 = Y$
$fib_n = fib_{n-2} + fib_{n-1}$

we will get now the sequence $X, Y, XY, YXY, XYYXY, YXYXYYXY, XYYXYYXYXYYXY \ldots$, by considering "+" as string concatenation,

For this question, you will need to implement a divide and conquer algorithm than given N and K as parameters, your function returns the K-th letter in the N-th string in the fibonacci sequence. For example, if $N = 7$ and $K = 7$, your algorithm must return X (please notice that the N=7 string in the sequence is "XYYXYY**X**YXYYXY", and the K=7 letter in that string is "X", which is bold).

HINT 1: Please check if there is a relation between the two sequences (i.e., the numeric and letter-based Fibonacci sequences) shown in this question

HINT 2: The integers could be very big, then please use BIGINTEGERS
HINT 3: Please do not even try concatenating the strings, this approach will never succeed

You will need to submit your `A2_Q2.java` source file to the `Assignment 2 => Q2 - Divide and Conquer` lesson in Ed-Lessons.

Exercise 3 (60 points)**.** ***Dynamic Programming***

Given my totally wrong spending habits, I recently moved from paying everything with my (overloaded and high interest) credit card to paying with paper money. With this change, I have improved my mindful spending and budgeting habits. An inconvenience appeared when I tried a vending machine (located in Trottier building) that do not return change. Being as lazy as I am (and given the cold winters in Montreal), I am planning to keep using that machine (because it has great products and I do not have to exit the building :P). Of course, I would like to minimize the amount that I give to the machine, and while paying the minimum amount that is at least as much as the value of the item that I intend to buy, I would also like to minimize the number of bills that I pay out.

As a Comp251 student, you have been commissioned (by me) to complete the function `change` in the `A2_Q3` class. This function receive as parameters, an array of integers (called `values_bills`) representing the value in **cents** of each bill that I have. Please notice that the values of this array are not limited to the denominations of our Canadian bill system and it can contain any number of cents. Given that I have an (very low) academic-based salary, I guarantee that I will never have more than one hundred bills with me. `change` receives as a second parameter the integer `price`, which represent the price in **cents** of the item that I intent to buy in the machine. Please notice that the machine does not contain items that exceed the price of 100 dollars. For this problem, you can safely assume that the total value of my bills will always be equal or greater than the price of the item, and no bill will have a value greater than 100 dollars. The function `change` must return an integer array of size two, where the first and second elements represents the total amount paid (in cents), and the total number of bills that I used.

Let's see one example to make sure that everything is clear. On Tuesday (February 6th), I bought an item that costed 14 dollars (i.e., 1400 cents). On that day, I had in my wallet 3 bills with the following values [500, 1000, 2000]. Then, I paid (to the vending machine) a total of 1500 cents and I used two bills. An array with these two values (i.e., [1500,2]) is what is expected as answer from your `change` function.

You will need to submit your `A2_Q3.java` source file to the `Assignment 2 => Q3 - DP` lesson in Ed-Lessons.

Exercise 4 (40 points)**.** ***Greedy***

In this exercise, you will plan your homework with a greedy algorithm. The input is a list of homeworks defined by two arrays: deadlines and weights (the relative importance of the homework towards your final grade). These arrays have the same size and they contain integers between 1 and 100. The index of each entry in the arrays represents a single homework, for example, `Homework 2` is defined as a homework with deadline `deadlines[2]` and weight `weights[2]`. Each homework takes exactly one hour to complete.

Your task is to output a `homeworkPlan`: an array of length equal to the last deadline. Each entry in the array represents a one-hour timeslot, starting at 0 and ending at 'last deadline -

1'. For each time slot, `homeworkPlan` indicates the homework which you plan to do during that slot. You can only complete a single homework in one 1-hour slot. The homeworks are due at the beginning of a time slot, in other words if an assignment's deadline is `x`, then the last time slot when you can do it is `x - 1`. For example, if the homework is due at t=14, then you can complete it before or during the slot t=13. If your solution plans to do `Homework 2` first, then you should have `homeworkPlan[0]=2` in the output. Note that sometimes you will be given too much homework to complete in time, and that is okay.

Your homework plan should maximize the sum of the weights of completed assignments.

To organize your schedule, we give you a class `HW_Sched.java`, which defines an `Assignment` object, with a number (its index in the input array), a weight and a deadline.

The input arrays are unsorted. As part of the greedy algorithm, the template we provide sorts the homeworks using Java's `Collections.sort()`. This sort function uses Java's `compare()` method, which takes two objects as input, compares them, and outputs the order they should appear in. The template will ask you to override this `compare()` method, which will alter the way in which Assignments will be ordered. You have to determine what comparison criterion you want to use. Given two assignments A1 and A2, the method should output:

- 0, if the two items are equivalent
- 1, if a1 should appear after a2 in the sorted list
- -1, if a2 should appear after a1 in the sorted list

The `compare` method (called by `Collections.sort()`) should be the only tool you use to re-organize lists and arrays in this problem. You will then implement the rest of the `SelectAssignments()` method.

What To Submit?

Attached to this assignment are java template files. You have to submit only this java files. Please DO NOT zip (or rar) your files, and do not submit any other files.

Where To Submit?

You need to submit your assignment in ed - Lessons. Please note that you do not need to submit anything to myCourses.

When To Submit?

Please do not wait until the last minute to submit your assignment. You never know what could go wrong during the last moment. Please also remember that you are allowed to have multiple submission. Then, submit your partial work early and you will be able to upload updated versions later (as far as they are submitted before the deadline).

How will this assignment be graded?

Each student will receive an overall score for this assignment. This score is the combination of the passed open and private test cases for the questions of this assignment. The open cases correspond to the examples given in this document plus other examples. These cases will be run with-in your submissions and you will receive automated test results (i.e., the autograder output) for them. You MUST guarantee that your code passes these cases. In general, the private test cases are inputs that you have not seen and they will test the correctness of your algorithm on those inputs once the deadline of the assignment is over; however, for this assignment you will have information about the status (i.e., if it passed or not) of your test. Please notice that not all the test cases have the same weight.