Assignment 2

This assignment is worth 15% of the total course marks.

Due date: The assignment is due at **9am on Monday 22 October.** Late assignments will attract a penalty of 10% of the total assignment mark per day, and submissions more than 5 days late will not be marked. Only extensions on Medical Grounds or Exceptional Circumstances will be considered, and in those cases students need to submit an application for extension of progressive assessment form (http://www.uq.edu.au/myadvisor/forms/exams/progressive-assessment-extension.pdf) to the lecturer (email is acceptable) or the ITEE Enquiries desk on Level 4 of GPSouth Building) prior to the assignment deadline.

Resources:

The Assignment 2 zip file, available from Blackboard (under Learning Resources/Course Materials), contains the Java classes and interfaces that are mentioned in this handout. You will need those source files to complete this assignment.

Your Java Build Path will need to include the JUnit 4 Library to run the JUnit tests.

Problem overview:

The façade of a building is to consist of a number of horizontal *platforms*. The positioning of a platform p is defined by: its horizontal start distance (in meters) *p.start* from the left hand corner of the building; its horizontal finish distance *p.end* from the left hand corner of the building; and its height *p.height* from the flat base of the building. For convenience we will denote the position of a platform p by the triple <p.start, p.end, p.height>.

For example, Figure 1 depicts a façade consisting of four platforms: $p_1 = <1,3,2>$, $p_2 = <2,5,4>$, $p_3 = <2,4,1>$ and $p_4 = <4,7,3>$.

Each platform p must have a non-zero width (i.e. p.end - p.start > 0), and p.start, p.end and p.height may be non-negative real values, for example, <1.3, 4.23, 6.7> is a valid position for a platform.

No two distinct platforms p_1 and p_2 can coexist at the same height in the same interval. That is the closed intervals $[p_1.start, p_1.end]$ and $[p_2.start, p_2.end]$ cannot overlap if $p_1.height$ equals $p_2.height$. For instance, we could not add an extra platform $p_5 = <3,4,4>$ to those already in Figure 1 since it has the same height as p_2 and interval [3,4] overlaps with [2,5]. We could however add another platform $p_5 = <5.5,7,4>$ at the same level as p_2 since interval [2,5] and [5.5,7] do not overlap (i.e. they have an empty intersection).

Even though each platform has an absolute height to the bottom of the building, the architect is interested in knowing, for each platform p and horizontal position $x \in [p.start, p.end]$ along p, the distance from x to the closest platform directly below p, if there is one, or the base of the building otherwise. This distance is known as the *propping height* of p at x. Formally, we define:

proppingHeight(p, x, P) =

$$p.height - maximum\ (\{p_i{:}P \mid p_i.height < p.height \wedge x \in [p_i.start, p_i.end] \cdot p_i.height\ \} \cup \{0\ \})$$

where p is a platform from the set of platforms P and x is an element in the closed interval [p.start, p.end]. The set $\{p_i{:}P \mid p_i.height < p.height \wedge x \in [p_i.start, p_i.end] \cdot p_i.height\ \}$ contains the height of each platform directly below p at position x. The maximum of a set is the greatest element in that set.

For example, the propping height of p_1 from Figure 1 is 2 for x ∈ [1,2) and 1 for x ∈ [2,3], where P={p_1,p_2,p_3,p_4}. Similarly the propping height of p_2 is 2 for x ∈ [2,3]; 3 for x ∈ (3,4) and 1 for x ∈ [4,5]. The propping height of p_3 is 1 for x ∈ [2,4], and the propping height of p_4 is 2 at the point x=4 and 3 for *x* ∈ (4,7].

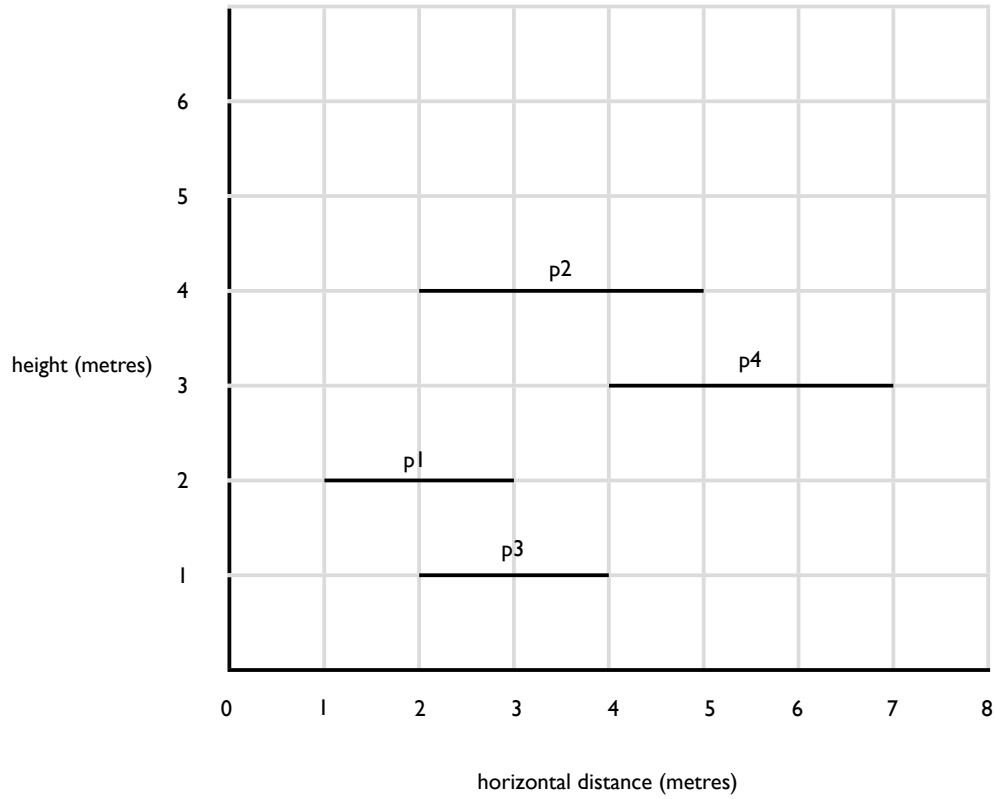

Figure 1: Facade consisting of platforms p_1 = <1,3,2>, p_2 = <2,5,4>, p_3 = <2,4,1> and p_4 = <4,7,3>.

Task 1: Algorithm design and implementation [11 marks]

Your task is to design and implement an efficient algorithm `getProppingHeights` that takes as input an unordered sequence of platforms P = <p_1, … , p_n>, for some n ≥ 0 and returns a mapping from each platform p_i ∈ P to a sequence

< <s_1, e_1, h_1>, … , < s_m, e_m, h_m> >

of triples such that:

- s_1 = p_i.start and e_m = p_i.end,
- for each j∈ [1,..,m-1], e_j = s_{j+1} and $h_j \neq h_{j+1}$,
- for each j∈ [1,..,m] we have that $e_j - s_j > 0$, and
- for each j∈ [1,..,m] and x ∈(s_j, e_j) we have that proppingHeight(p_i, x, P) = h_j.

For example, the output of `getProppingHeights(<p1, p2, p3, p4>)` where p_1 = <1,3,2>, p_2 = <2,5,4>, p_3 = <2,4,1> and p_4 = <4,7,3> would be a mapping containing the following (key, entry) pairs:

{(p_1, <<1,2,2>, <2,3,1>>), (p_2, <<2,3,2>, <3,4,3>, <4,5,1>>), (p_3, <<2,4,1>>), (p_4, <<4,7,3>>)}

Your algorithm should be as efficient as possible.

First, I recommend that you sketch out your algorithm in pseudocode using any of the ADTs or algorithms covered in lectures. (This will help you to make sure that you get the idea correct before you get stuck into the implementation details). After you have done that you must implement your algorithm in java in the method `getProppingHeights` that belongs to the class `a2.HeightCalculator` that is available in the assignment zip file.

You must not change the name or signature of the method `getProppingHeights`, or the name or package of the class to which it belongs. Do not change class `a2.Position` in any way. You should also not write any code that is operating-system specific (e.g. by hard-coding in newline characters etc.), since we will batch test your code on a Unix machine. Your source files should be written using ASCII characters only.

You may (and should) use the Java 7 SE API in your implementation, but no other libraries should be used. (It is not necessary and makes marking hard.)

Any extra classes that you write should be included in the file `a2.HeightCalculator` as private nested classes. Any additional class methods should also be private.

To help you get your solution right, you should write you own JUnit tests in `a2.test.HeightCalculatorTest` (some basic tests are included to get you started). Tests that you write will not be marked. We will test your implementation with our own tests.

HINT: Consider pre-processing the inputs.

Task 2: Time complexity analysis and justification [4 marks]

What is the time complexity of your algorithm from Task 1?

Justify your answer. You may wish to use your pseudocode to help you to justify your analysis (assuming that your pseudocode is actually realised by your implementation).

In your justification, explain why you have chosen the data structures and algorithms (from the java API etc.) that you have to make your algorithm time efficient; and how the choices that you have made effect the time complexity of your solution.

Express your answers in big-Oh notation, making the bounds as tight as possible, and simplifying your answer as much as possible. If the expected time complexity differs from the worst-case time complexity (because you use Hash Tables for instance), then your time complexity analysis should include an acknowledgement of this and a discussion of both.

Task 3: COMP7505 ONLY: Space complexity analysis and justification [3 marks]

What is the space complexity of your algorithm from Task 1?

Justify your answer. You may wish to use your pseudocode to help you to justify your analysis (assuming that your pseudocode is actually realised by your implementation).

In your justification, explain how the data structures and algorithms (from the java API etc.) that you have chosen to use affect the space complexity of your solution.

Express your answers in big-Oh notation, making the bounds as tight as possible, and simplifying your answer as much as possible.

Practical considerations:

If necessary, there may be some small changes to the files that are provided, up to 1 week before the deadline, in order to make the requirements clearer, or to tweak test cases. These updates will be clearly announced on the Announcements page of the course Blackboard site, and during the lectures.

Submission: Submit you source code file `HeightCalculator.java`, as well as your written answers in `report.pdf` electronically using Blackboard according to the exact instructions on the Blackboard website:

https://learn.uq.edu.au/

Answers to Tasks 2 and 3 should be clearly labelled and included in file `report.pdf`.

You can submit your assignment multiple times before the assignment deadline but only the last submission will be saved by the system and marked. Only submit the files listed above. You are responsible for ensuring that you have submitted the files that you intended to submit in the way that we have requested them. You will be marked on the files that you submitted and not on those that you intended to submit. Only files that are submitted according to the instructions on Blackboard will be marked

Evaluation: Your assignment will be given a mark according to the following marking criteria. For COMP3506 the total marks possible is 15, and for COMP7505 the total marks possible is 18. The assignment is worth 15% for both courses.

Code must be clearly written, consistently indented, and commented. Java naming conventions should be used, and lines should not be excessively long (> 80 chars). Marks will be deducted in code analysis and efficiency questions if we cannot read and understand your solution to check your analysis or evaluate the efficiency of your solution.

Task 1: [11 marks]

Given that the solution does not violate any restrictions (using forbidden libraries etc), [3 marks] will be allocated for time efficiency and the remaining [8 marks] will be allocated based on the outcome of running test cases:

- All of our tests pass — 8 marks
- At least 85% of our tests pass — 7 marks
- At least 75% of our tests pass — 6 marks
- At least 65% of our tests pass — 5 mark
- At least 50% of our tests pass — 4 marks
- At least 35% of our tests pass — 3 mark
- At least 25% of our tests pass — 2 mark
- At least 15% of our tests pass — 1 mark
- Work with little or no academic merit — 0 marks

Note: code submitted with compilation errors will result in zero marks in this section. Code that violates any stated restrictions will receive zero marks. To receive any marks at all for efficiency the code must be mostly correct and easy to read and comprehend.

Task 2: [4 marks]

- The algorithm analysis and justifications are correct and complete. 4 marks
- The algorithm analysis and justifications contain only very minor errors or omissions. 3 marks
- The algorithm analysis and justifications are reasonable but contain some errors or omissions. 2 marks
- The algorithm analysis and justifications contain many errors or omissions or a major error or omission. 1 mark
- Work with little or no academic merit 0 marks

A mark of 0 will be given for this part if no reasonable attempt was made to complete Task 1 or the solution to Task 1 is difficult to read and comprehend, or incorrect.

Task 3: COMP7505 ONLY [3 marks]

- The algorithm analysis and justifications are correct and complete. 3 marks
- The algorithm analysis and justifications are reasonable but contain some errors or omissions. 2 marks
- The algorithm analysis and justifications contain many errors or omissions or a major error or omission. 1 mark
- Work with little or no academic merit 0 marks

A mark of 0 will be given for this part if no reasonable attempt was made to complete Task 1 or the solution to Task 1 is difficult to read and comprehend, or incorrect.

School Policy on Student Misconduct: You are required to read and understand the School Statement on Misconduct, available on the School's website at:

http://ppl.app.uq.edu.au/content/3.60.04-student-integrity-and-misconduct

This is an individual assignment. If you are found guilty of misconduct (plagiarism or collusion) then penalties will be applied.

If you are under pressure to meet the assignment deadline, contact the course coordinator **as soon as possible**.