Module Code
COM00016C

BEng, BSc, MEng and MMath Degree Examinations 2023–2024

DEPARTMENT OF COMPUTER SCIENCE

Software 2

Software Lab Week 10

Recommended Time:

FOUR Hours

Allocation of Marks:

Question 1 is worth 70 marks and question 2 is worth 30 marks.

Instructions:

Questions 1 and 2 are independent and can be answered in any order using Java 8 or above (preferably OpenJDK-17).

Download the required source files from the VLE. In the project, you will find the `lib` folder containing the `jar` files needed for unit testing. Do not delete or move them. In the `src\exercise` folder, you will find an incomplete implementation of the class `QTreeAssetManager`, the interfaces `IAsset` and `IAssetManager`, and some unit tests. You should implement your solutions within that folder.

As being able to test the robustness and validity of a program is the learning objective of this week, the unit tests provided are quite simple and do not cover edge cases. You are encouraged to add more tests to the files to ensure the correctness of your code. For once, you will not be able to use Gradescope to check your progress.

Once completed, upload the four requested files (`BoundingBox.java`, `Asset.java`, `QTreeAssetManager.java`, and `Question2.java`) to the Gradescope submission point `SOF2 week 10 Software lab` on the VLE.

Finally, you will not see the outcome of the tests until the submission deadline has passed, that is Monday 6 May at 9:30 AM.

1 (70 marks) QuadTree

The company we are working for has a set of assets. An asset is some object (a warehouse, a field, a building, etc.) which covers an area. The company is interested in finding quickly all assets they have in a given area. For this we index the assets using a QuadTree (defined in Question 1.(iii)). An area is defined as a rectangular bounding box.

(i) [20 marks] First define and implement a class `BoundingBox` representing a rectangular area. A `BoundingBox` has four **public** attributes of type `int` named `x`, `y`, `width`, and `height`. The point `(x,y)` represent the top left corner of the bounding box as shown in Figure 1. We are using a discrete plan where the smallest area has a width and height of 1, therefore the bottom right corner of a bounding box is at position `(x+width,y+height)`. In addition, to ensure non overlapping of bounding box in a quadtree subdivision of space, the bottom edge and the right edge of the bounding box is not part of the bounding box, i.e. any point P with coordinate (x+width, y_P) or (x_P, y+height) is outside the bounding box.

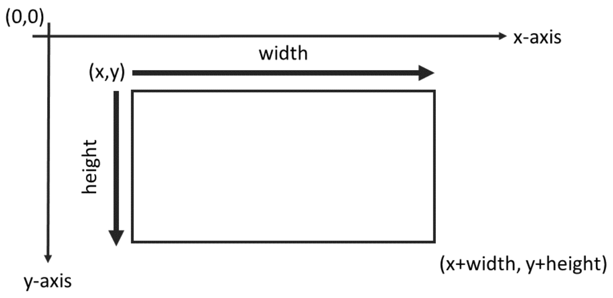

Figure 1: A boundingBox coordinates system.

You must use the file `BoundingBox.java` provided and complete the implementation of the following methods:

(a) [5 marks] `BoundingBox(int x, int y, int width, int height)` a constructor which throws an `IllegalArgumentException` if the width or height are not greater or equal to 1.

(b) [4 marks] `boolean equals(Object other)` which returns `true` if `other` has the same width, height and top-left corner, `false` otherwise.

(c) [3 marks] `Boolean contains(int x, int y)` which returns `true` if the point (x, y) is contained in the bounding box, `false` otherwise.

(d) [4 marks] `Boolean contains(BoundingBox box)` which returns `true` if the area covered by `box` is entirely comprised in the bounding box, `false` otherwise. Note, A bounding box is contained within itself.

(e) [4 marks] `Boolean intersects(BoundingBox box)` which returns `true` if some part of the area covered by `box` is also part of the bounding box, `false` otherwise. Two bounding boxes A and B do not intersect if at least one of the following condition is true:

- $B.x + B.width \leq A.x$
- $B.y + B.height \leq A.y$
- $A.x + A.width \leq B.x$
- $A.y + A.height \leq B.y$

Note: To test your class, we have provided a JUnit test class in the file `BoundingBoxTest.java`.

(ii) [20 marks] Implement the class `Asset` which implements the interface `IAsset`. The class `Asset` has at least the following attributes with package visibility (not private):

- `String content` represents a single asset of the company such as a warehouse. Note, we do not have to implement a warehouse class, for simplicity we will use a `String`. For example, `"Warehouse 51"`.
- `BoundingBox area` is the area used/covered by that asset.

You must use the file `Asset.java` provided and complete the implementation of the following methods:

(a) [3 marks] `Asset(String content, BoundingBox area)` where `content` is the asset (such as `"Warehouse 51"`), and `area` is the area covered by that asset. The constructor must throw an `IllegalArgumentException` if `content` or `area` are `null`.

(b) [4 marks] `boolean equals(Object other)` which returns `true` if `other` has the same content and the same bounding box, `false` otherwise.

(c) [2 marks] `String getContent()` which returns the content of the asset.

(d) [2 marks] `BoundingBox getArea()` which returns the area covered by the asset.

(e) [4 marks] `Boolean intersect(BoundingBox region)` which returns `true` if the area covered by the asset intersects `region`, `false` otherwise.

(f) [5 marks] `Boolean isContainedIn(BoundingBox region)` which returns `true` if the asset's area is contained inside `region`, `false` otherwise.

Note: To test your class, we have provided a JUnit test class in the file `AssetTest.java`.

(iii) [30 marks]

QuadTrees are a straightforward spatial indexing technique. In a QuadTree, each node represents a bounding box covering some part of the space being indexed, with the root node covering the entire area. Each node is either:

- a leaf node, in which case it contains **one or more** entries, and no children,
- or it is an internal node, in which case it has exactly four children. One child for each quadrant obtained by dividing the area covered in half along both axes (as shown in Figure 2). In addition, the node contains **zero or more** entries.

An entry is store in a node, if the entry's area intersects more than one children area, or if the node is at the maximum depth of the QuadTree.

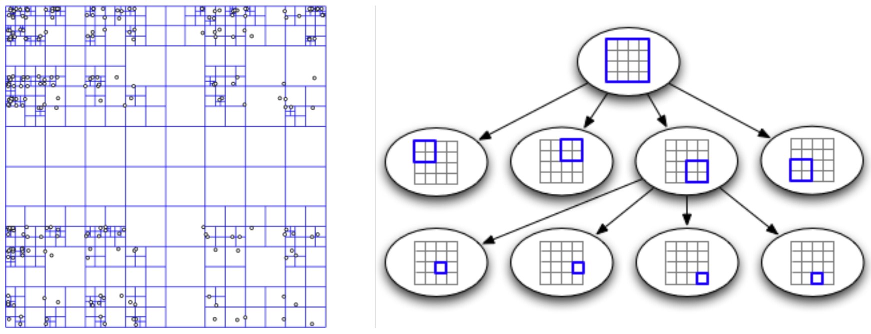
(a) (b)

Figure 2: a) a representation of how a QuadTree divides an indexed area (Source: Wikipedia), b) a representation of how a QuadTree is structured internally.

We have provided an incomplete implementation of a QuadTree in the class `QTreeAssetManager`. You must complete the implementation of the following methods from the interface `IAssetManager`.

(a) [15 marks] `Set<IAsset> getAssets()` which returns all assets contained in that QuadTree.

(b) [15 marks] `Set<IAsset> getAssets(BoundingBox region)` which returns all assets contained in the QuadTree that intersect the given `region`.

hint: To query a QuadTree, starting at the root node, examine each asset in that node to see if it intersects with the query region. If it does add the entry to the set of entries to be returned. Then examine each child node, and check if it intersects the area being queried for. If it does, recurse into that child node.

Note: test your code using the JUnit tests in `QTreeAssetManagerTest.java`.

2 (30 marks) Dynamic Programming

The code for this question must be written in the provided file `Question2.java`.

The rod cutting problem: suppose you have a rod of length n, and you want to cut up the rod and sell the pieces in a way that maximises the total amount of money you get. A piece of length i is worth £p_i.

Table 1: An example of a price list.

length i	1	2	3	4	5	6
price p_i	1	5	8	9	10	17

For example, if you have a rod of length 4, there are eight different ways to cut it, and given the price list in Table 1, the best strategy is cutting it into two pieces of length 2, which gives you £10 (see Figure 3).

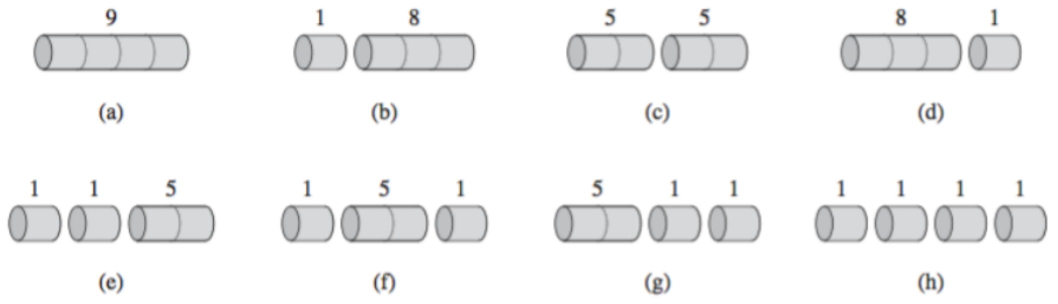

Figure 3: The 8 possible ways of cutting up a rod of length 4. Given the price list in Table 1, the optimal strategy is part (c) – cutting the rod into two pieces of length 2 – which as a total value of £10.

We can view the problem recursively as follows:

- First, cut a piece off the left end of the rod, and sell it.
- Then, find the optimal way to cut the remainder of the rod.

However, a naive recursive implementation to compute the maximised profit of cutting a rod of size `n` given a price list `p` is inefficient ($O(2^n)$. A better approach is to use dynamic programming as shown in Algorithm 1 where the algorithm complexity is $O(n^2)$.

(i) [15 marks] Write a **static** method which implements Algorithm 1. The method signature is `int rodCutting(int[] p, int n)` and the method must be public. In addition, the method must throw an `IllegalArgumentException` if at least one of the following conditions is true:

- the array `p` contains a price less or equal to zero,
- the size of array `p` is strictly less than `n`,
- `n < 0`.

Note: test your code using the JUnit tests in `Question2_i_Test.java`.

Algorithm 1 Dynamic Programming algorithm to solve the rod cutting problem.

```
function MEMOIZEDCUTROD(p, n)
    let $r[0..n]$ be a new array
    for $i := 0$ to $n$ do
        $r[i] := -\infty$
    end for
    return memoizedCutRodAux(p, n, r)
end function

function MEMOIZEDCUTRODAUX(p,n,r)
    if $r[n] \geq 0$ then
        return $r[n]$
    end if
    if $n = 0$ then
        $q := 0$
    else
        $q = -\infty$
        for $i := 1$ to $n$ do
            $q := max(q, p[i] + memoizedCutRodAux(p, n - i, r))$
        end for
    end if
    $r[n] := q$
    return q
end function
```

(ii) [15 marks] Write a **static** method which implements a modified version of Algorithm 1 such that the price list contains only a subset of pieces size. For example, only pieces of size 2, 4, 5 and 8 can be sold. Any other length must be discarded (waste). The method signature is `int rodCutting(Map<Integer, Integer> p, int n)` and the method must be public. The map `p` has the length of a piece for keys and the price of the piece for value. For example, given Table 2 the key 8 is mapped to value 11. In addition, the method must throw an `IllegalArgumentException` if one of the following condition is true:

- the map `p` contains a price or a length less or equal to zero,
- `n < 0`.

Table 2: An example of a price list where only some specific rod length can be sold. In this example, only four sizes are allowed [2, 4, 5, 8].

length i	2	4	5	8
price p_i	2	5	9	11

Note: test your code using the JUnit tests in `Question2_ii_Test.java`.

End of examination paper