EECS 2030: Programming Exam 1
Winter 2022

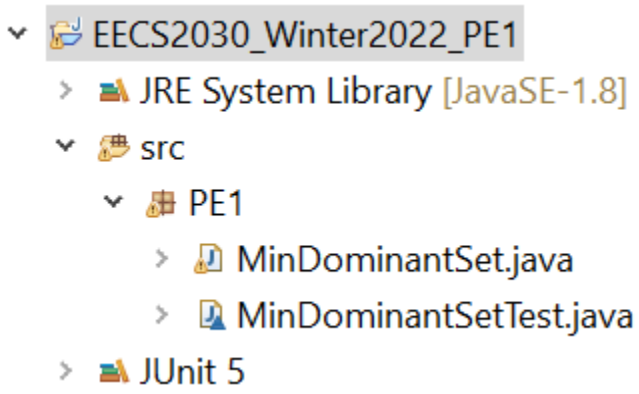
Deadline: Tuesday Feb 13th, 2021 11:59pm

In the first programming exam, we are going to assess your ability for the following skills:

- Ability to design recursive functions.
- Ability to test your code thoroughly, through Junit tests.
- Ability to document your code.

Setting up the environment:

- Download the attached zip file.
- Open eclipse.
- Click on File and select Import
- Choose Existing Projects into Workspace and click Next
- Click on Select Archive File and then Browse.
- Find the zip file that you have already downloaded and click Finish
- Please make sure that you do not already have a project called EECS2030_Winter2022_PE1
- Now you should see the following settings in the package explorer.



Before you start, I would like to remind you that this assignment is different from the labs that you have done until now, as this assignment is an **exam** that you take home. This means you can neither discuss your solution with peers nor get help from them. Therefore, please carefully read the statement at the top of `MinDominantSet.java` before you start doing the assignment. When you are finished you will be needed to sign this statement by writing your information (full name and student number and section) in the gap provided. Please be aware that your assignment WILL NOT be marked if you do not sign the declaration. Also, I would like to inform you that your code will be checked by both a plagiarism detection tool and us to make sure your solution to this problem is unique and not similar to others.

Refusing to follow the policy stated above, is a violation of academic integrity that will have a serious consequences. Please read about the definition of academic integrity and the penalty that one may face, in case they violate academic integrity in https://lassonde.yorku.ca/student-life/academic-support/academic-honesty-integrity .

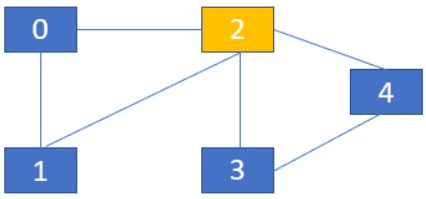
Problem Description:

York university has decided to place a vending machine to sell N95 masks in a couple of buildings in the campus. However, the cost of each vending machine is too high that they cannot place one machine in each building. Therefore, they decided to find the minimum number of machines that they need so that either

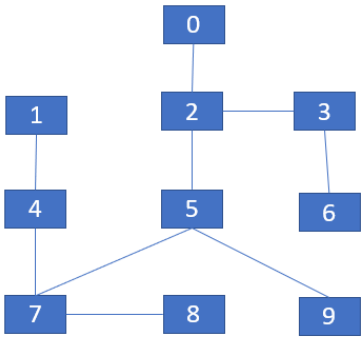
1. a building has a vending machine,
2. or it is 100 meters away from the building that has the machine.

Solving this problem requires the knowledge of recursion. So, they asked you, EECS2030 students, who are familiar with this concept to write an algorithm that tells them how many vending machines they should buy and to which building they should place it. [I trust you that you don't believe a word of what I said. Because nobody has asked us to do so. It's just what occurred to me at 2:00 am on Saturday when I was thinking about designing a programming exam for you 😊]

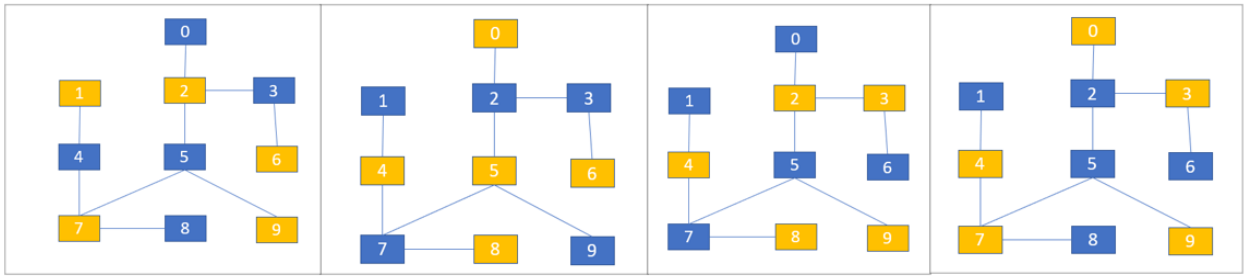
To clarify this problem, let's see some examples. Suppose that we have 5 buildings in the campus. Instead of using the name of the buildings, they are represented by a number. As you see in the picture below, the buildings that are connected with a line, represent the buildings that are 100 meters away from each other. In this example, all the buildings are at most 100 meter away from building 2. Also, the minimum number of vending machine that we need is 1, if we place it in building 2. In this case either the building has a vending machine (e.g., building 2) or they are at most 100 meters away from the building that has a vending machine (e.g., 0, 1, 3, 4).



Now assume that we have 10 buildings that are connected like the figure below:



There are several ways by which you can place 5 machines. In the figures below the orange background shows the buildings in which the machines were placed. Can you do better? Can you use only 4 machines (or even less) to cover all the buildings?



Now that you got familiar with the problem, it is time to solve the problem. There are a few ways by which you can solve this problem. However, we have to solve this problem with the least effective way possible because our hands are tied. Why? Because the majority of you have not passed data structure, algorithm analysis or any artificial intelligence courses.

So, let's get to the least effective way of solving this problem. First, we need to assume that a certain numbers of vending machines (`r`) is available. Then we find the combination of selecting `r` buildings from the possible `n` buildings to place the machines. At this point, when we computed all the combinations, we can select each combination one by one, place the vending machines in there and check if with this arrangement, all the buildings are covered. For example, suppose that we have 5 buildings and 4 vending machines. in this case there are 5 possible combinations as below:
[0 1 2 3], [0, 1, 2, 4], [0, 1, 3, 4] [0, 2, 3, 4], [1, 2, 3, 4]

You may ask why this method is not effective. I tell you why. Assume that we have 40 buildings in the campus, and we are going to provide only 15 machines. In this case the number of combinations that we need to check is 40,225,345,056. This takes a considerable amount of time for processing. The formula for calculating combination of `r` items (i.e. buildings) selected among `n` is as below

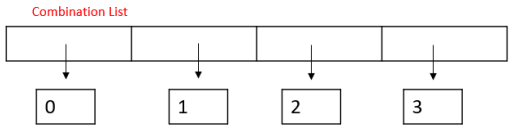
$$C(n,r) = \frac{n!}{r!\,(n-r)!}$$

If you are clear about how the problem is solved, read the following tasks, and start the coding. If you are not ready yet, take a pen and a piece of paper and solve the problem on paper to make it clear for yourself. If you are not familiar with the concept of combination, you can read more in https://www.mathsisfun.com/combinatorics/combinations-permutations.html.

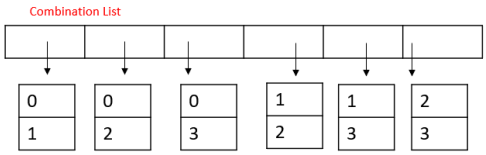
Task 0:

As you see in the starter code, there is an instance variable defined as `ArrayList<Integer[]> combination`. The definition means that we have a list, which at each index of this list, an array of integer is stored. You can assume that we have created a two-dimensional array with this definition. This list is going to store all the combination of `r` buildings selected from `n` buildings. Please see the following lists for clarification:

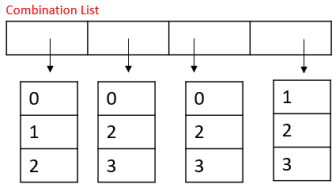
`C(4,1)`



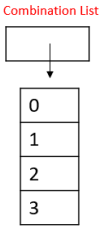
`C(4,2)`



`C(4,3)`



`C(4,4)`



Your job for this task is to implement the default constructor that initializes this list to its `default` value.

Task 1:

In this task you are required to implement `combination` method **recursively**. This method gets two inputs n, r and creates all the combinations of selecting r items from n items. Remember that the items here are the buildings that are represented by integer numbers starting from zero. This method stores the combination in the instance variable (i.e. `combination`) of this class. Please see the figures above on how to store the selections in the list. This method does not return anything.

```
void combination(int n, int r)
```

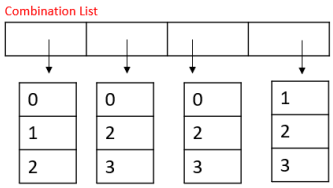
To implement this method, I am quite sure that you need a helper function. Therefore, either the `combination` method or its helper should be recursive in order to receive a point for this task. If it is not recursive, you won't be awarded any grade. Please make the helper function `private`.

Sample Call:

```
MinDominantSet pe1 = new MinDominantSet(4);
pe1.combination(4,3);
```

Corresponding output:

No output, but the combination list gets assigned as shown in the figure below:



Task 2:

There is an overloaded constructor in the starter code. This constructor requires 2 input arguments to construct the object. The first is `size`, which represents the number of buildings in the campus.

The second is a two-dimensional array [size][size], which contains 0 or 1. This array represents the distance between buildings. For example, row 0 (and also column zero), represent the distance between building 0 with other building. If element at `[i][j]` is zero, it means the distance between building `i, j` is not less than 100 meters. If the element at `[i][j]` is 1, it means the distance between these two building is less than or equal to 100 meters.

Your job for this task is to initialize the instance variables with the inputs provided to this constructor. There is no input for the `combination` attribute provided. I am pretty sure that you know you should initialize it to the default value.

Task 3:

For this task you are going to complete the `isEnough` method, with the given method signature as below. You are allowed to implement this method however you want, i.e., recursively or non-recursively. You can also use as many helper methods as you require as long as you make them `private`.

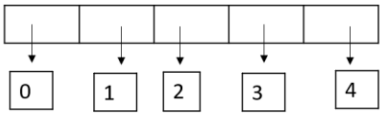
```
public boolean isEnough(int numMachine)
```

This method takes in the number of vending machine that the university is willing to provide. Your job is to find if this number of machines can cover all the buildings. If yes, the method returns true. Otherwise, it returns false. For this task, there are some assumptions that always hold true including:

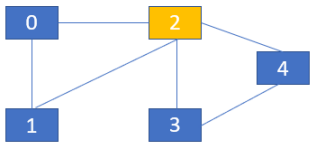
1. the overloaded constructor is called before this method. This means that the `building` and `size` attributes are correctly initialized.
2. `numMachine` $\leq$ `size`
3. `numMachine` $= r$, this means that the length of the array at each element of `combination` arrayList is the same as the number of machines that are provided. And that's right, you should call the `combination(size, numMachine)` in this method.

Sample Input:

numMachine = 1, where `combination` and `building` attributes looks like below respectively from left to right. The third figure is just the graphical representation of the matrix.



	0	1	2	3	4
0	1	1	1	0	0
1	1	1	1	0	0
2	1	1	1	1	1
3	0	0	1	1	1
4	0	0	1	1	1



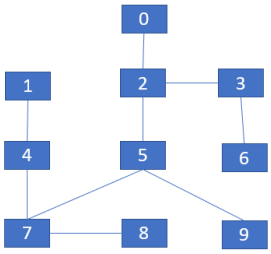
Corresponding output:

True

Sample Input:

NumMachine = 3, if the connection between buildings is as below. Please note that the size of the combination arrayList is 84 because C(9,3) = 84, which made it impossible for us to show it here.

	0	1	2	3	4	5	6	7	8	9
0	1	0	1	0	0	0	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
2	1	0	1	1	0	1	0	0	0	0
3	0	0	1	1	0	0	1	0	0	0
4	0	1	0	0	1	0	0	1	0	0
5	0	0	1	0	0	1	0	1	0	1
6	0	0	0	1	0	0	1	0	0	0
7	0	0	0	0	1	1	0	1	1	0
8	0	0	0	0	0	0	0	1	1	0
9	0	0	0	0	0	1	0	0	0	1



Corresponding output:

false

Marking Scheme:

- [10 points]: For the correctness of task 0.
- [30 points]: for the correctness of task 1.
- [10 points]: for the correctness of task 2.
- [30 points]: for the correctness of task 3.
- [20 points]: For javaDoc, inner documentations, and correct code style (meaningful variable names, correct indentations).

Checklist:

Before you submit, check this checklist to ensure that you have done all we asked you to do.

- ☐ I have finished tasks 0, 1, 2, and 3.
- ☐ I have signed the statement on academic honesty.
- ☐ I have removed/commented out the `main()` method, to ensure that I have tested my methods with the sample tests provided, and my methods signature conforms with the tester's requirement.
- ☐ I have tested my code with more JUnit test cases, which I have written myself as I am aware that a small sample of test cases have been provided.

What and how to Submit:

- You only submit one file, which is the completed `MinDominantSet.java` file.
- Submit your solution in eClass, where you have downloaded these instructions.