que finalice el curso
- ✓ Apoye a nuestro **misión** en edX

Module 14 Assignment: MST

Marcar esta página

currículum
- ✓ Desbloquea el acceso a todas las actividades del curso, incluidas las **tareas calificadas**
- ✓ **Acceso completo** al contenido y los materiales del curso, incluso después de que finalice el curso
- ✓ Apoye a nuestro **misión** en edX

$149

Coding Assignment fecha límite Nov 13, 2023 10:01 -05 Vencidos

Minimum Spanning Trees

For this assignment, you will be implementing **Prim's** algorithm for minimum spanning trees. This assignment has quite a few files in it, so please be sure to read **ALL** of the documentation given to you.

IMPORTANT:

- **You will be given 5 attempts on this assignment, with a 30 minute cooldown between submissions.**
- **Please run your code before each submission to ensure that there are no formatting errors! If there are formatting errors in your code, your code will not be graded and a submission attempt will be logged. For more information, please review the Vocareum overview below.**

Graph Data Structure

Note: This section on the Graph data structure is the exact same as the one in the module 13 coding assignment.

You are provided a **Graph** class. The important methods to note from this class are:

- **getVertices** provides a Set of **Vertex** objects (another class provided to you) associated with a graph.
- **getAdjList** provides a Map that maps **Vertex** objects to Lists of **VertexDistance** objects. This Map is especially important for traversing the graph, as it will efficiently provide you the edges associated with any vertex. For example, consider an adjacency list map where vertex A is associated with a list that includes a **VertexDistance** object with vertex B and distance 2 and another **VertexDistance** object with vertex C and distance 3. This implies that in this graph, there is an edge from vertex A to vertex B of weight 2 and another edge from vertex A to vertex C of weight 3.

Vertex Distance Data Structure

In the **Graph** class, you will be using the **VertexDistance** class implementation that we have provided. This data structure is used by the adjacency list to represent which vertices a vertex is connected to. In Prim's algorithm, you should use this data structure along with a *PriorityQueue*.

Minimum Spanning Trees (MST - Prim's Algorithm)

A MST has three components. By definition, it is a tree, which means that it is a graph that is acyclic and connected. A spanning tree is a tree that connects the entire graph. It must also be minimum, meaning the sum of edge weights of the tree must be the smallest of all possible spanning trees.

By the properties of a spanning tree, any valid MST must have $|V| - 1$ edges in it. However, since all undirected edges are specified as two directional edges, a valid MST for your implementation will have $2(|V| - 1)$ edges in it.

Prim's algorithm builds the MST outward from a single component, beginning with a starting vertex. At each step, the algorithm adds the cheapest edge connected to the incomplete MST that does not cause a cycle. Cycle detection can be handled with a visited set.

Self-Loops and Parallel Edges

In this framework, self-loops and parallel edges work as you would expect. If you recall, self-loops are edges from a vertex to itself. Parallel edges are multiple edges with the same orientation between two vertices. These cases are valid test cases, and you should expect them to be tested. However, most implementations of these algorithms handle these cases automatically, so you shouldn't have to worry too much about them when implementing the algorithms.

General Tips

- There are two stopping conditions for Prim's. The first is when you have found your MST, and the second is when you have considered all reachable vertices from the start. Prim's should continue to run until one of the stopping conditions have been met. How and where should you check for both of these conditions in your code?

- Before returning your MST, it is important to ensure that the MST is valid, as it is possible that no valid MST exists due to a disconnected graph. How can you ensure that your MST is valid? How many edges must a valid MST have in relation to the number of vertices in the undirected graph?

- We highly recommend copying the starter code and working in your preferred IDE in order to have access to features such as code completion, auto-formatting, and much more!

Here are general assignment guidelines that should be followed.

- Do not include any package declarations in your classes.
- Do not change any existing class headers, constructors, instance/global variables, or method signatures. For example, do not add throws to the method headers since they are not necessary. Instead, exceptions should be thrown as follows: `throw new InsertExceptionHere("Error: some exception was thrown");`
- All helper methods you choose to write should be made private. Recall the idea of Encapsulation in Object-Oriented Programming!
- Do not use anything that would trivialize the assignment. (e.g. Don't import/use java.util.ArrayList for an ArrayList assignment.)
- Always be very conscious of efficiency. Even if your method is to be $O(n)$, traversing the structure multiple times is considered inefficient unless that is absolutely required (and that case is extremely rare).
- If applicable, use the generic type of the class; do not use the raw type of the class. For example, use `new LinkedList<Integer>()` instead of `new LinkedList()`.

Use of the following statements should be avoided at all times.

package	System.arraycopy()	clone()
assert()	Arrays class	Array class
Thread class	Collections class	Collection.toArray()
Reflection APIs	Inner or nested classes	Lambda Expressions

The Vocareum (code editor) interface has six main components:

- The **Drop-Down** in the top left. This lets you choose from multiple available files. Note that this drop-down will only be visible in assignments that require multiple files.
- The **Run** button. This will compile your code and run a file scan. Running your code will not count towards your total allowed submission attempts, therefore you are free to run as many times as needed.
- The **Submit** button. This will compile your code, run a file scan, grade your assignment, and output results to console. Note that for most assignments in this class, you will only be allowed a limited number of submissions. A submission is counted when the submit button is clicked, regardless of whether or not your code can compile or if there are any file issues. Therefore, we **highly recommend** that you run your code before submitting to ensure that there are no issues that will prevent your code from being graded and that every submission attempt will generate meaningful results.
- The **Reset** button. This will revert all your changes and reset your code to the default code template.
- The **Code Window**. This is where you will write your code. For large coding assignments, we highly recommend copying the starter code and working in your preferred IDE to have access to features such as code completion, auto-formatting, and much more!
- The **Output Window**. This window will appear whenever you run or submit your code and will display the output for you to view.

For additional help, please visit the Vocareum information page located in the course information module!