المملكة العربية السعودية
وزارة التعليم العالي
جامعة الملك عبد العزيز
كلية الحاسبات وتقنية المعلومات

Department of Computer Science

CPCS 324: Algorithms and Data Structures (II)

Spring 2023

Group Project – Part I (15%)

Issued Sunday 9th -04-2023

Objective:

- Provide an opportunity to work with complex data structures
- Develop advanced programming skills.

Learning Outcomes:

- CLO #8: Compare select (covered) algorithms for the same problem.
- CLO #9: Write code to implement select (covered) greedy algorithm(s) in programming language and environment of choice.

Required Tasks (Total 15 points): **Due Date:** ***Sunday 7th -05-2023, 11:59PM***

Given the following problem:

"‥ a problem like phone network design. You have a business with several offices; you want to lease phone lines to connect them up with each other, and the phone company charges different amounts of money to connect different pairs of cities. You want a set of lines that connects all your offices with a minimum total cost. It should be a spanning tree, since if a network isn't a tree you can always remove some edges and save money." [1]

---

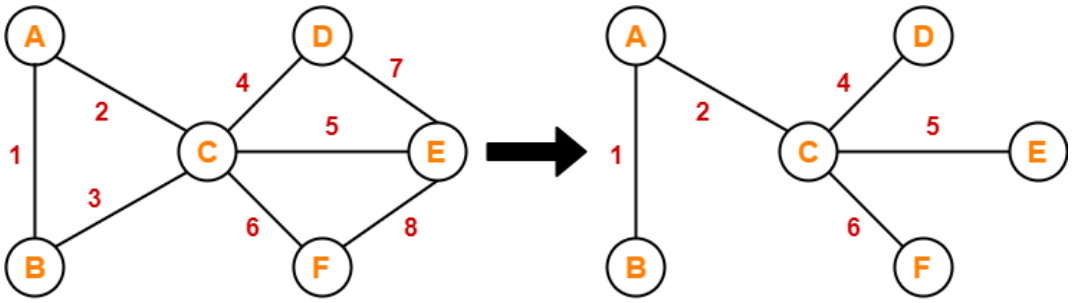
follow the design presented in ***Appendix I*** and implement it -*as instructed*- to do the following:

Requirement 1:

Compute the minimum spanning tree and print it using two algorithms as follows:

- Define a function called readGraphFromFile(filename).

- Store the following graph[2] within a file – the format of the file is shown in Appendix II.



Given Graph **Minimum Spanning Tree (MST)**

(Cost = 18 units)

Ps. Only store the left-hand side of the Figure, the resulting MST is presented here for your information.

- Compute the minimum spanning tree using Kruskal algorithm and print the result as in the following Figure:

> The phone network (minimum spanning tree) generated by Kruskal algorithm is as follows:
>
> Office No. A – Office No. B : line length: 5
>
> Office No. A – Office No. C : line length: 10
>
> ……..
>
> The cost of designed phone network: 18

- Compute the minimum spanning tree using Min-heap based Prim algorithm and print the result as in the following Figure:

> The phone network (minimum spanning tree) generated by min-heap based Prim algorithm is as follows:
>
> Office No. A – Office No. B : line length: 5
>
> Office No. A – Office No. C : line length: 10
>
> ……..
>
> The cost of designed phone network: 18

Required Output:

The computed output of both algorithms, as shown in the figures.

Requirement 2:

1. Define a function called make_graph() to randomly generate Graphs, assuming that the edge weights are integers, for the following cases:
    - n=1000 and m={10000, 15000, 25000},
    - n=5000 and m={15000, 25000},
    - n=10000 and m={15000, 25000},
2. Perform an experimental comparison of two of the minimum-spanning-tree algorithms, Kruskal and min-heap- based Prim. Develop an extensive set of experiments to test and plot the running times of these algorithms using the above randomly generated graphs.

Required Output:

- Plot the running times of the implemented algorithms above of tasks 2**.** Comment on each plot in no more than one paragraph

---

Instructions:

1. Each group must have only *four* students.
2. Students are required to submit *a soft copy* of their work – the code must be submitted using Github
3. Avoid *plagiarism* (Avoid copying work from your class fellows).
4. Submit the Project by the given deadline to avoid deduction of marks.
5. Only one member of the group should submit the project under its slot in the course Blackboard page.
6. Ensure to meet all the *rubric* requirements.
7. Be Creative!!

Expectations

1. Students are expected to submit *three* deliverables:
    - A source code for the implementation of the algorithm(s) uploaded in GitHub such that ***(required output points)***.
        1. The code is clear and documented (commented)
        2. The function are well structured.
    - A report (.docx) that includes the following:
        1. A small introduction (one paragraph)
        2. Clear screenshots for the outputs with an explanation for each figure ***(required output points)***
        3. Difficulties faced during the phase design
        4. Conclusion
2. A final oral presentation (with slides) for executing the project phases in pre-assigned date and time by the instructor**.**

Consequences / Penalties:

Plagiarism/Copying or Outsourcing will not be tolerated. **If a student is caught cheating, then the grade of the project for all students knowingly involved will be *Zero*.** Furthermore, based on the severity of the case, the entire course grade for the student may be lowered a full letter grade, an **"F"** in the course, dismissal from an academic unit, revocation of admission, suspension from the university, etc.

Resources/Recommended Software:

- Git
- Visual Studio Code (*recommended*: install Git first)

Course Textbook: Anany Levitin, Introduction to The Design and Analysis of Algorithms, Addison-Wesley (Pearson International Edition), 3rd edition, 2011. ISBN: 027376411X

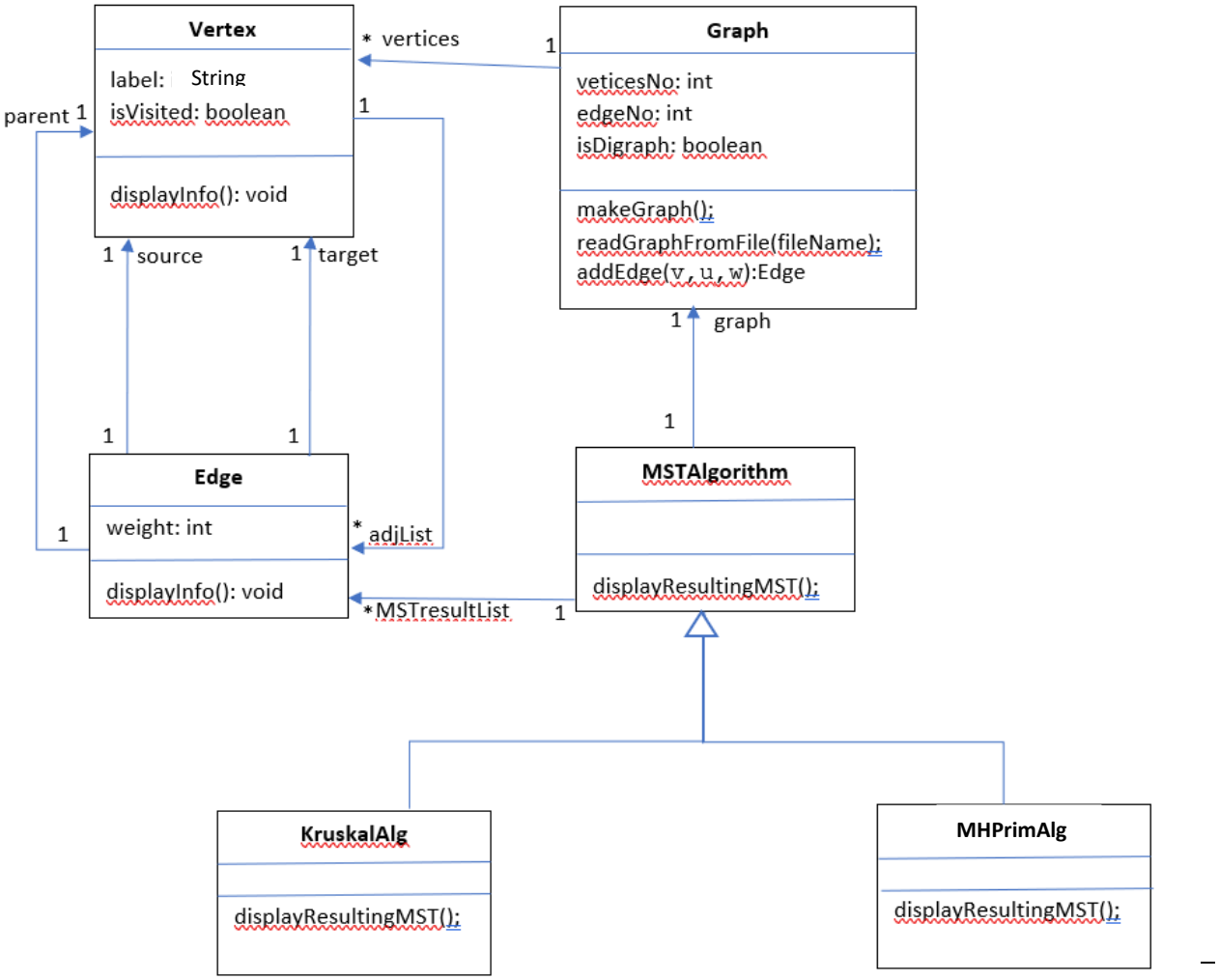
References

1 Applications of Minimum Spanning Tree Problem - GeeksforGeeks
2 Kruskal's Algorithm Example with Solution | Gate Vidyalay

Appendix I

Use the TWO Class diagrams presented below to implement the Phone Network Design Application. The relationship between the two parts of the system is shown in the following package diagram.

The following class diagram represents the Graph Framework:



1. Use the above classes, their attributes and member functions in your implementation.

PS. Marks will be **deducted** in case any of the elements of the class diagrams was not used as defined, or *extra* variables were used for the same purpose of those elements.

2. According to UML notation, the name on the side of an association should be used as a member variable of the class. For example, `parent` on the association between `Vertex` and `Edge` classes should be defined as a member variable within the Class `Edge`, the variable's type is `Vertex` and its name is `parent`.

3. Do not use the Pair class in Java as it is defined for other complex applications. You can use classes defined for representing key-value pairs like Map.

4. The list `adjList` on the side of the association between Vertex and Edge is an abstraction. Use appropriate class/classes to define it as a linked list.

5. Do not use priority queue to sort the edges when implementing Kruskal algorithm.

6. The parameters of the member functions are left to the students to be defined/modified as appropriate.

Classes documentation:

==========

`Graph`: is a class. It defines the structure of a graph. It contains the following attributes and functions:

`veticesNo`: number of vertices of the graph. It should be incremented whenever a function a new object is added to the vertex list.

`edgeNo`: number of edges of the graph.

`isDigraph`: a boolean that is set to true if the graph is directed graph, and set to false if the graph is undirected.

Attributes displayed via association relationship in the diagram: `vertices` attribute is a list/vector representing the list of vertices of a graph.

`makeGraph()`: this function takes as parameters the number of vertices and the number of edges. It is responsible for creating a graph object with the specified parameters and randomly initializes the vertices' labels, creating edges that connects the created vertices randomly and assigning them random weights. Make sure that the resulting graph is connected.

`readGraphFromFile(fileName)` reads the edges and vertices from the text file whose name is specified by the parameter `filename` and place data in a `Graph` object. In this project, you need to create a text file that contains the graph presented in **requirement 1**. The file format is shown in **Appendix II**. It is responsible for doing some preprocessing then call the `addEdge()` method to determine the position of the Edge **.**

`addEdge(v,u,w)`: is a function that creates an edge object and passes the source vertex v, the target vertex and w the vertex weight as parameters, assigns the target vertex to the adjacent list of the source vertex and if the graph is undirected then it will add the source vertex to the adjacent list of the target

vertex. It increments the `EdgeNo` by one in case it is a directed graph and by two if it is an undirected graph.

==========

`Vertex`: is a class that represents a graph vertex. It contains the following attributes and functions:

label: a number that represents the vertex label.

`isVisited`: holds a boolean value that is initialized by true and then set to true if the current vertex is visited within a certain operation like graph traversal.

Attributes displayed via association relationship in the diagram: `adjList` attribute should be an implementation of a linked list as it represents the adjacency list of a vertex within a class.

`displayInfo()` method is responsible for displaying the information of the class attributes.

==========

`Edge`: is a class that represents a graph edge. It contains the following attributes and functions:

Attributes displayed via association relationship in the diagram: source vertex, destination vertex and parent vertex; all of the type `Vertex`. The parent vertex represents the parent of target vertex/source vertex depending on the application.

`weight`: is an integer value, that holds the weight assigned to the edge connecting the source and target vertices.

`displayInfo()` method is responsible for displaying the information of the class attributes.

==========

`MSTAlgorithm`: is a superclass representing the general characteristics of an algorithm for computing the minimum spanning tree. It has three subclasses: `KruskalAlg` and `MHPrimAlg`. It contains the following attributes and functions:

Attributes displayed via association relationship in the diagram: `MSTResultList` attribute is a list of objects of the type `Edge`. It stores the parent of the vertex and the weight needed by the MST algorithm.

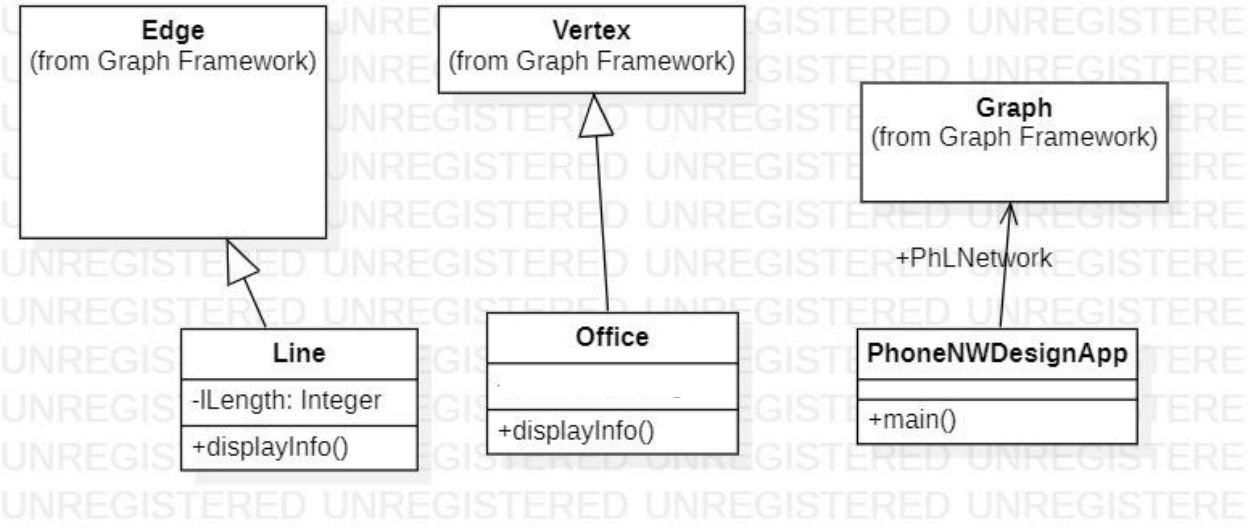
`displayResultingMST()`**:** it is an abstract function that should be implemented by the subclasses' polymorphic functions.

==========

`KruskalAlg`: is a subclass of `MSTAlgorithm`. It implements the polymorphic operation `displayResultingMST()`. Make sure it calls the `displayInfo()` method of the `Vertex` and `Edge` classes.

`MHPrimAlg`: is a subclass of `MSTAlgorithm`. It implements the polymorphic operation `displayResultingMST()`. Make sure it calls the `displayInfo()` method of the `Vertex` and `Edge` classes.

The following class diagram shows the representation of the PhoneNetworkApp



`PhNWDesignApp`: is a class. It is the starting point of the program and contains the main function.

`main()`: this function should be responsible for running the readGraphFromFile method for requirement 1 and running the make graph function for requirement 2 to initialize the graph and invoking the 2 algorithm and displaying the returned results and the measured running time. The output should show the requested comparison between the results & run time of the algorithms.

Important: Make sure that you create objects of the `Graph`, `Office` and `Line` classes, ***do not create objects*** of Vertex and Edge neither in `PhoneNetworkApp` package or `Graph` package. Two grades will be deducted if objects of Vertex or Edge are created anywhere in the application.

Hint: Graph should have `createVertex()` & `createEdge()` methods that need to be overridden by `Office` & `Line` subclasses.

Office: it is a subclass of Vertex, it inherits all attributes, operations & relationships.

`setLabel()` method (alternatively set the correct value within the relevant constructor) to store O1, O2,… or O15 – starts with O followed by a unique number.

Override/complete the `displayInfo()` method to display the information of the class attributes.

Line: it is a subclass of Edge, it inherits all attributes, operations & relationships.

The lLength attribute represents the line length and it is 5 times the weight of the corresponding edge.

Override/complete the `displayInfo()` method to display the information of the class attributes.

Appendix II

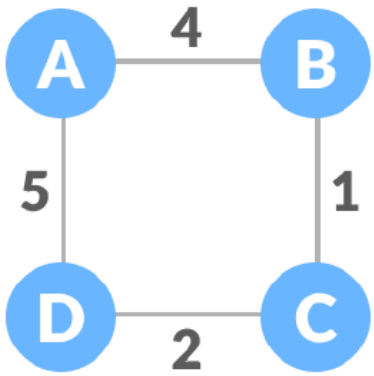


Figure 1: Graph

Format of the text file for the graph shown in Figure 1:

```
digraph 0
4 4
A B 4
B C 1
C D 2
D A 5
```

Notes:
1- In the first line digraph 0 means it is an undirected graph. In the case of a directed graph, the first line should be: digraph 1
2- The second line has two numbers the first is the number of vertices and the second is the number of edges
3- The characters A, B, C, etc. are vertices names and the numbers are weights.
4- This is a suggested format for the text file, other formats are acceptable as long as the file contains the same information