**Logical Part**

**Task 1A :**

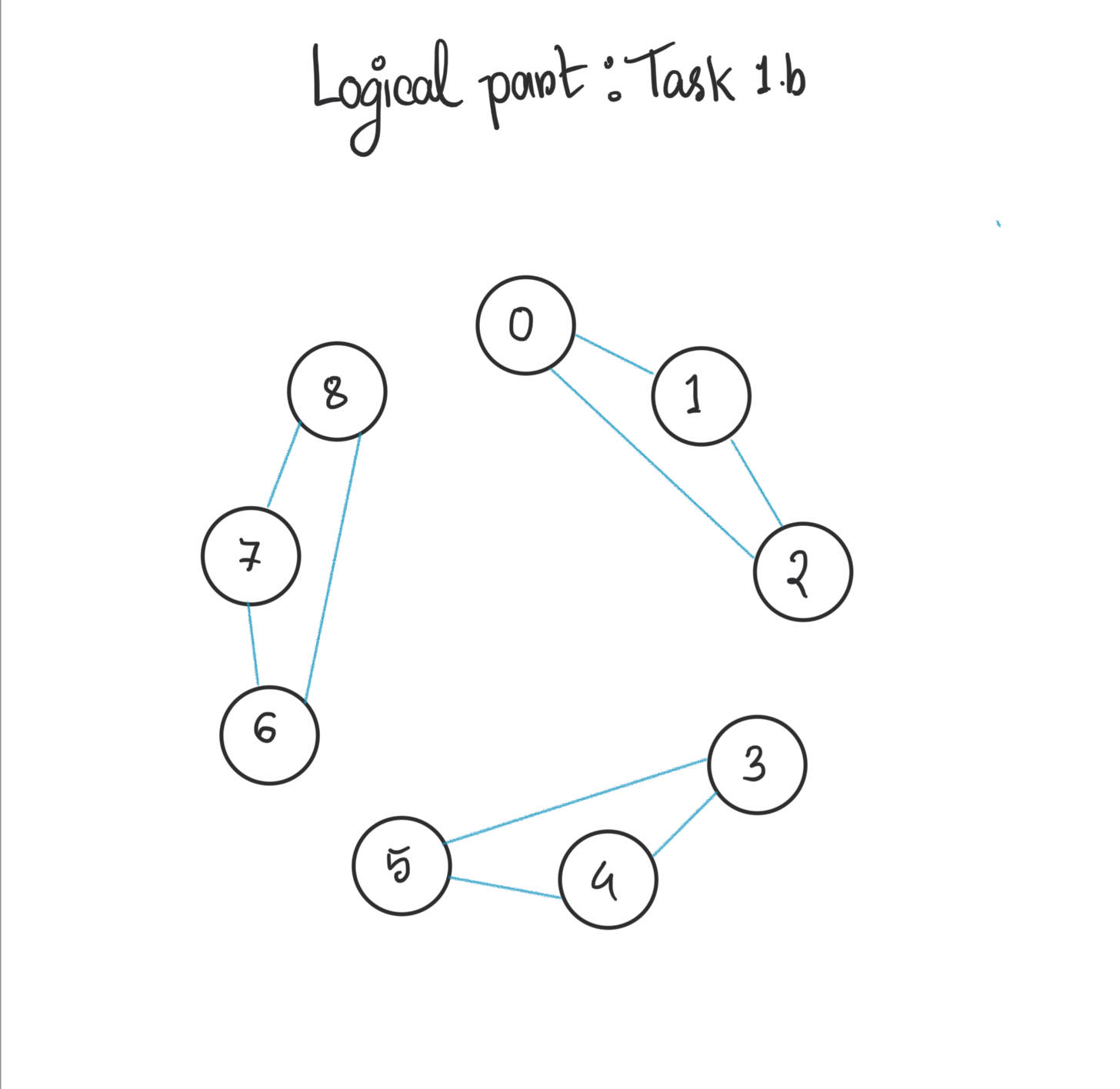
**code link:** [**https://github.com/Fa-riya/Altair-Software-Subteam-Selection/blob/main/210041103\_Fariya%20Ahmed\_week%201\_Logical\_Task%201\_A.cpp**](https://github.com/Fa-riya/Altair-Software-Subteam-Selection/blob/main/210041103_Fariya%20Ahmed_week%201_Logical_Task%201_A.cpp)

I used Depth-First Search Traversal method to traverse the graph in this code. First, I created an adjacency list of the graph using an array of integer vectors named edges. The vector at a particular index contains the other vertices that specific vertex is connected to.

Then I created a bool array named visited\_vertex which holds true or false depending on whether that vertex has been visited or not. At first, I set all values to false. Then started dfs traversal to visit these vertices starting with vertex 0 as it is the first vertex in the edge list. After setting visited\_vertex of that index to true, the function then traverses its adjacency list to mark the adjacent vertices as visited. As it is a recursive call, the function keeps getting called until it finds a vertex that has been visited before or if all the neighboring vertices of a certain vertex has been traversed through recursion. Then the function returns and continues traversing the adjacency list of the previous vertex. If any vertex is not connected anyhow to the first vertex ,then the algorithm won’t mark it as visited. After the traversal ends, the function returns.

In the path function I called the dfs function. After traversal the path function then checks if all values in the visited\_vertex have turned to true. If yes that means all the vertices can be visited so the function returns true. Otherwise, there is at least one vertex that cannot be visited so the function returns false. Finally, I printed this value as answer.

**Task 1B: Graph-**

****

**Task 2:**

1. **Dijkstra Algorithm:**

Dijkstra’s algorithm is a shortest path algorithm which works by repeatedly selecting the vertex with the minimum distance from the source. It is used in various fields such as navigation systems, network routing, GPS etc to find shortest path in maps or telephone networks.

The time complexity of this algorithm is O ((V+E) log V) where E is the number of edges and V is the number of vertices. The Space complexity for this algorithm is O(V).

* Highly efficient for small or medium graphs which have fewer edges. However, it’s not recommended for larger graphs
* It works with graphs with non-negative edge weights, it does not work properly if there are negative edges in the graph.
* It is the best shortest path algorithm for graphs with weighted edges

1. **Bellman Ford Algorithm:**

Bellman Ford Algorithm is another shortest path algorithm which can work with negative edges. It first overestimates the length of path to all vertices, then iterates to relax those estimates by finding new paths that are shorter than the paths found or estimated before. It can work on graphs with negative edges which is why it is more versatile.

The time complexity for this algorithm is O(VE) however the best case complexity is O(E). The space complexity is O(V).

* Can handle negative edge graphs and detect negative cycles, can be used when Dijkstra fails.
* It is slower and requires more iterations.
* Good for small or medium graphs but inefficient for larger graphs because of the time complexity.

1. **Floyd-Warshall Algorithm:**

It is an all-pair SP algorithm that works on graphs with non-negative and negative edges. The time complexity for this algorithm is O(V^3) and space complexity is O(V^2).

* Works for graphs with both positive and negative weighted edges.
* Inefficient for larger graphs.
* Consumes large amount of memory because of the 3D matrix created to track paths between all pairs of vertices.
* Great for ASPS.
* Can be used to check whether an undirected graph is bipartite.

1. **A\* Algorithm:**

A\* algorithm is like a smarter version of the Dijkstra algorithm which can work on graphs with negative paths. However, it might not always produce the shortest path. It is commonly used in various applications, games, robotics and GPS navigation systems. The time complexity for A\* is O(E) and space complexity is O(V).

* Works on both unweighted and weighted graphs with positive and negative edges.
* Highly efficient, outperforms other algorithms.
* Great for larger graphs.
* May not always find the shortest path.

To summarize, if there is one source and one destination then it is best to use A\* algorithm. If there are multiple destinations, Dijkstra is great for graphs with non-negative edges which are weighted. But for negative edges, Bellman Ford is used in this case. For ASPS (path between every pair of nodes) it is best to use Floyd-Warshall algorithm.

**Microcontroller Part**

**Task 1:**

Tinkercad link: <https://www.tinkercad.com/things/2IaUWSEYpZU-exquisite-uusam/editel?sharecode=FCO3kH3Q7FwdN-mc60ousqjF7I0Dt0VJeqwxiTw7i0Q>

**Task description:**

Using two Arduino Uno, I used I2C to communicate between the boards. I used Wire.h library for this. During transmission the strings are converted into bytes.

The master firsts sends data to the slave “Hi from Master” which is printed on the serial monitor on the slave Arduino. Then master sends a request to slave to send “Successful”. Master receives the message and prints it on the serial monitor.

The slave Arduino first receives “Hi from Master” without any request. The data is printed on the serial monitor after being received by slave. Then the master requests data from slave and slave sends the message “Successful” through .

**Problems I faced during the task:**

1. As this was my first time using Arduino, I had to first learn the basic setups
2. During running simulation, Tinkercad was showing “Invalid Header file” error even though all the header files were valid. This was due to an error in function parameters which took me one hour to resolve.
3. Transmitted datatype was byte so I had to convert my message into bytes and after receiving reconvert them into a string to print on the serial monitor.

**Task 2:**

Tinkercad link: <https://www.tinkercad.com/things/4fTwfquDbkJ-terrific-leelo/editel?sharecode=WvOHSF1bZYaM96U6i7WozbBMbwyzf-4_l9K3_T-KQLs>

**Task description:**

Using an Arduino Uno and a DC motor with encoder I controlled the motor with the Arduino and printed the speed in rpm in the serial monitor. To calculate speed in rpm I took the position of the motor every 1 second and the time elapsed in simulation using millis() function. Then I printed the result in the serial monitor.

**Problems I faced during the task:**

1. I wanted to calculate the instantaneous speed in rpm however an error occurred which I couldn’t debug , printing unrealistic values. So I printed the average speed in rpm as the simulation ran
2. The rpm is shown in integer format even though the calculations are done in float format. This was done to simplify the look. But there is a slight error/deviation from the original value which can be ignored.