Function: compute_activities_times()

Description:

This function calculates the earliest and latest start times, as well as the earliest and latest end times, for activities in a directed acyclic graph (DAG). It utilizes a topological sorting algorithm to determine the order in which the activities should be executed.

Parameters:

- **self:** An instance of the class that contains the graph and its associated properties.

Returns: None

Raises:

- **ValueError:** If the graph contains cycles and is not a valid DAG.

Documentation:

The compute_activities_times() function is used to compute the scheduling and timing information for activities within a directed acyclic graph (DAG). By executing this function, the earliest start time, earliest end time, latest start time, and latest end time for each activity are determined.

The function initiates the process by performing a topological sort on the graph using the topological_sort_graph() function. In case the graph contains cycles, signifying that it is not a valid DAG, a ValueError is raised.

Next, the total number of activities is computed by subtracting 2 from the total number of vertices in the graph. This calculation assumes that the graph includes two additional vertices that symbolize the source and the sink.

Subsequently, the function proceeds to iterate through the sorted vertices, commencing from index 1. For each vertex, it determines the earliest start time by selecting the maximum earliest end time among its incoming vertices. The earliest end time is then computed by adding the duration of the current activity to its earliest start time.

Upon completion of processing all the vertices, the last activity is identified using the number of activities. The latest end time for the last activity is set equal to the corresponding earliest end time, and the latest start time is obtained by subtracting the duration of the last activity from the latest end time.

Following this, the function iterates over the sorted vertices in reverse order, starting from the last activity and moving towards the first. For each vertex, it calculates the latest end time by selecting the minimum latest start time among its outgoing vertices. The latest start time is then obtained by subtracting the duration of the current activity from the latest end time.

Finally, the function updates the relevant timing attributes (earliest_start_time, earliest_end_time, latest_start_time, and latest_end_time) of the graph object for each activity.

Function: topological_sort_graph(graph_to_sort: DirectedGraph)

Description:

This function performs a topological sort on a directed graph to determine a linear ordering of its vertices. The topological sorting order represents a sequence in which each vertex appears before its successors in the graph.

Parameters:

graph_to_sort: An instance of the DirectedGraph class representing the graph to be sorted.

Returns:

sorted_vertices: A list of vertices in the graph in topologically sorted order. If the graph contains cycles and cannot be sorted, None is returned.

Documentation:

The topological_sort_graph() function performs a topological sort on a directed graph to obtain a linear ordering of its vertices. It returns a list of vertices in the topologically sorted order, or None if the graph contains cycles and cannot be sorted.

The function starts by initializing an empty sorted_vertices list and a vertices_queue queue to hold vertices with no incoming edges.

A degree dictionary is created to track the in-degree of each vertex in the graph.

The function then iterates over each vertex in the graph, calculating its in-degree using the get_in_degree() method of the graph_to_sort object. If a vertex has an in-degree of 0, indicating no incoming edges, it is added to the vertices_queue

Next, the function enters a while loop that continues until the vertices_queue is empty. In each iteration, a vertex is dequeued from the vertices_queue and added to the sorted_vertices list.

For each outgoing vertex of the current vertex, the in-degree is decremented. If the in-degree of an outgoing vertex becomes 0, it is added to the vertices_queue.

Once the while loop completes, the function checks if the number of sorted vertices is equal to the total number of vertices in the graph. If they are not equal, indicating the presence of cycles, sorted_vertices is set to None.

Finally, the function returns the sorted_vertices list, representing the vertices of the graph in topologically sorted order. If the graph contains cycles and cannot be sorted, None is returned.