Q1

This problem involves finding the shortest weighted path in an undirected graph while mandating traversal through specific edges. This adds complexity to the traditional shortest-path problem because the algorithm must ensure that a subset of edges is included in the path. Personally, the key considerations include handling the undirected nature of the graph, efficiently tracking which required edges have been traversed, and ensuring the solution is computationally feasible for large graph.

Firstly, it is quite sure to use Dijkstra's algorithm's thought in the problem. However i need to add details into the algorithm and made it fit for the problem. Firstly, an adjacency list is used to represent the graph, where each node maintains a list of neighbors, their respective weights, and edge indices(this is important because i need to use this specifically) This facilitates quick access to connected nodes and edge properties. Since the graph is undirected, edges must be accessible in both directions, which is handled by appending neighbors for both (u, v) and (v, u). Additionally, a priority queue is utilized to implement Dijkstra's algorithm, maintaining the smallest current path cost and the set of edges covered so far. Each state is uniquely defined by the current node and the set of covered edges, ensuring correct traversal logic. The required edges are stored in a set for efficient membership checks. Each time a new edge is traversed, it is conditionally added to the set of covered edges.

The algorithm returns the minimum path cost if all required edges are covered upon reaching the destination; otherwise, it outputs -1, indicating no valid path exists. While coding, i made a mistake at first, Edge vs. Node Coverage i should check required edge coverage instead of node visitation, which explains why i did not pass questions. Also another strategy to improve efficiency, the combination of covered edges and nodes significantly increases the state space, necessitating careful pruning with a visited set to avoid redundant computations.

Q2

Firstly similar to the question that we have done in the homework, the best method to think of is to know that it is better to traverse from the end point, with the initial HP=0 and till the start point, get the HPs choose the smallest one. Now, unlike the previous homework, the graph is undirected, simple. There are specific constraints on how the HP and SP values are updated as we move through the nodes：

SP'=SP+1

$$\mathrm{HP'} = \mathrm{HP} + \left\lfloor \frac{a_i}{\mathrm{SP}+1} \right\rfloor$$

We approach the problem using a modified **Dijkstra's algorithm**, where the cost is the HP value. The graph is represented as an adjacency list, where each node has a list of neighboring nodes and edge weights.We start by creating an empty adjacency list graph, where each node has a list of its connected neighbors and the corresponding edge weights. Each edge is read and added to the graph in both directions since the graph is undirected. A priority queue pq is used to store nodes and their associated costs (HP and SP). The queue stores tuples in the form (current HP, current SP, current node). A visited set is used to ensure that we do not revisit nodes with the same SP, preventing redundant calculations and cycles. For each node, we explore all its neighbors and calculate the new HP by applying the formula. If we reach the target node t, we return the current HP as the answer. We handle the input, parse the number of nodes, edges, and the starting and ending nodes. Then, we run the function to get the result,which is the correct answer.