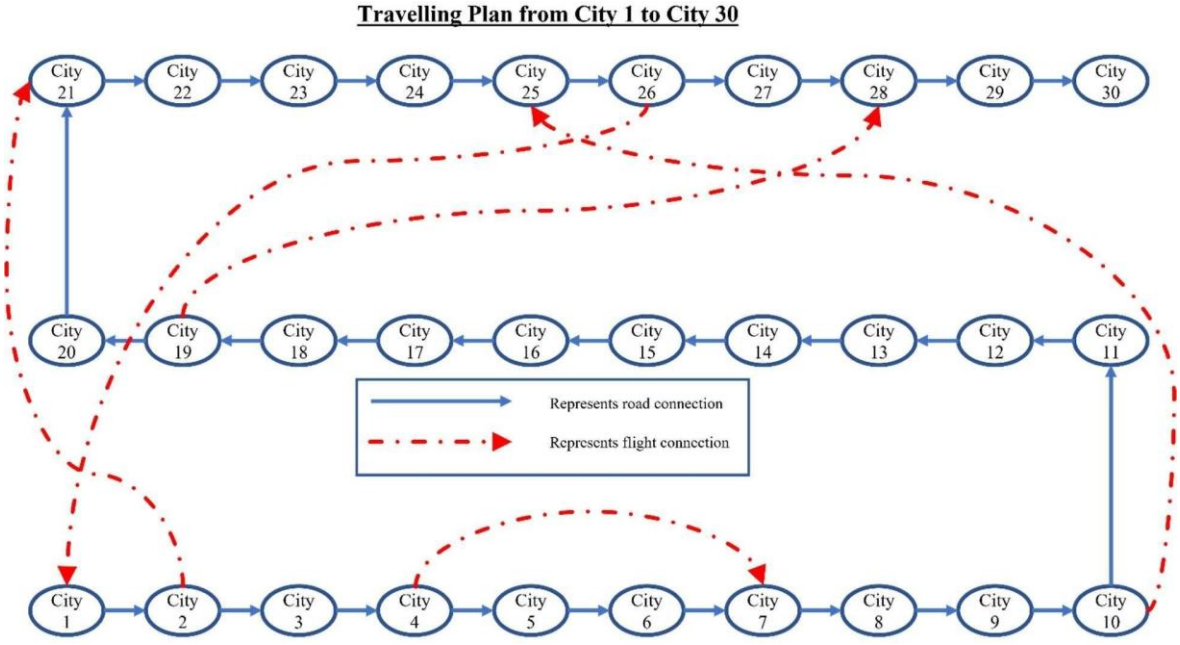
CS-2001 Data Structures
Semester Project: Emergency Travelling Plan

A traveler is visiting different places and suddenly receives a call from the office to reach back office due to an emergency meeting. Now the traveler is required to move back as soon as possible but he has certain constraints. There are some cities that are connected through road networks, and some are also connected through air connections. The traveler may not stop in a city where there is an aerial route taking him back to previous visited cities. As he is in far remote areas and the roads are not that good so he can travel to a maximum of 6 cities in one day if he travels by road. Once he reaches the city having the aerial route, he reaches the other destination of the aerial route by the end of the same day.

Let us take an example given in the figure given below. In figure all cities are connected through road network shown with arrow. There are some cities that also have aerial route shown as dotted lines such as from city 2 to city 21. Consider the traveler wants to move from city 1 to city 30 in as minimum days as possible. The traveler can travel up to six cities by road, but there is an aerial route available at city 2 so take one day to travel to city 2, take a flight to city 21 on day 1. After that he can take the road, but he should not stop at city 26 as he does not want to fly back (In case of such case the traveler should either stay on previous city or next city in case if maximum 6 cities are not covered). So, on day 2, he reaches city 27, covering 6 cities by road and on 3rd day he reaches his destination i.e., city 30. (Same can be achieved by travelling from city 1 to city 7 then on second day reaching city 25 through city 10 and on third day reaching destination). So, in total the minimum days that is required to reach destination is 3 days.



F igure: Emergency Travel plan

Hints:

- Start number of cities with zero i.e., city 0 and last city will be city (N-1) where is N is the number of cities.

Input

The first line should contain the number of cases. For each test case, the starting line contains the number of cities, the next line shows the number of aerial routes (*n*), and the subsequent *n* lines will show the source and destination of the aerial route, separated by a space.

Output

There would be as many outputs as the number of test cases. The output displays the minimum number of days required to travel from city 1 to city N.

Sample input	**Sample Output**
2 30 5 2 21 4 7 10 25 19 28 26 0 98 3 5 29 35 23 24 95	3 4

RUBRIC:

Graph Creation (18 marks):

- The code correctly creates a graph representation of cities and their connections (both road and aerial routes).
- The graph includes all cities and their corresponding edges based on the given input.
- The graph representation is appropriate for efficient traversal.

BFS Implementation (18 marks):

- The Breadth-First Search (BFS) algorithm is correctly implemented for traversing the graph.
- BFS is appropriately applied to explore cities and reach the destination within the specified constraints.
- The algorithm handles both road and aerial routes seamlessly.

Input Handling (14 marks):

- The program correctly parses and handles input for the number of test cases, the number of cities, and aerial routes.
- Input validation is implemented to handle potential errors or unexpected input formats.

- The program gracefully handles different input scenarios.

Output Generation (14 marks):

- The code produces the correct output, displaying the minimum number of days required to travel from city 1 to city N.
- The output format is clear and consistent across test cases.
- Output for both the provided example and additional test cases is accurate.

Code Organization and Readability (15 marks):

- The code is well-organized, with clear and meaningful variable names.
- Functions and logic are modular, promoting readability and maintainability.
- There are sufficient code comments explaining complex sections of the code.

Robustness and Error Handling (9 marks):

- The code handles edge cases and unexpected scenarios gracefully.
- There are appropriate error messages or feedback for users in case of invalid inputs or exceptional conditions.
- The program does not crash unexpectedly.

Efficiency (9 marks):

- The code is reasonably efficient in terms of time and space complexity.
- It avoids unnecessary computations or redundant data structures.
- The chosen algorithm scales well for larger inputs.

Code Comments (5 marks):

- Code comments are present throughout the code, explaining key sections, algorithmic steps, and any non-trivial logic.
- Comments enhance code readability and provide additional insights into the implementation.