Risk (Shortest Path)

The game described is a variant of Risk, where the primary goal is to determine the minimum number of sequential conquests needed to move armies from a starting country to a destination country on a board of 20 countries. Armies can only advance into a country that shares a border with the country most recently conquered, making the problem equivalent to finding the shortest path on a graph where countries are nodes and shared borders are edges. Unlike standard pathfinding, each step represents a conquest, so movement is effectively a one-way progression through connected countries. The programming challenge involves computing this shortest sequence of conquests—typically using Breadth-First Search (BFS)—for multiple start/destination pairs. The input includes the adjacency information for countries 1 through 19, followed by Ntest cases, each specifying a start and destination country. The output should clearly indicate the test case number, the start and destination countries, and the minimum number of conquests required to reach the destination.

Steps to Solve

1. Read the Map (Graph) Description

- The input describes connections between 20 countries, numbered 1–20.
- Read 19 lines of input, where line *i* contains the connections for country *i*.
- For each line:
    - First, read an integer x — the number of neighboring countries.
    - Then read x integers — the IDs of neighboring countries.
- Build an **undirected graph**: if country A connects to B, then B also connects to A.

2. Read the Number of Queries

- After reading all 19 country connections, read an integer N.
- N represents the number of path queries you need to answer.

3. Process Each Query with BFS

- For each of the N queries:
    - Read two integers: start country A and destination country B.

    - Use **Breadth-First Search (BFS)** to find the shortest path from A to B.

Why BFS?

- The graph is small (20 nodes).
- All edges have equal weight.
- BFS guarantees the minimum number of edges between nodes.

4. Implement BFS Algorithm

- Initialize:
    - visited[] array to track visited countries.
    - distance[] array to store the number of steps from the start.
    - A queue for BFS traversal.
- Start from country A with distance 0.
- While the queue is not empty:
    - Dequeue the current country.
    - For each neighbor of the current country:
        - If the neighbor has not been visited:
            - Mark it as visited.
            - Set distance[neighbor] = distance[current] + 1.
            - If the neighbor equals B, return the distance.
            - Enqueue the neighbor.

5. Format Output Correctly

- For each query result:
    - Print the start country, right-aligned in 2 spaces.
    - Print " to ".
    - Print the destination country, right-aligned in 2 spaces.
    - Print ": " followed by the shortest distance.

6. Handle Multiple Test Sets

- Before processing each test set, print:
- Test Set #T

where T = 1, 2, 3, ...

- After each test set, print a blank line.
- Continue until the end of input.

7. Repeat Until Input Ends

- After finishing one test set, immediately start reading the next 19 country lines.
- Repeat this process until reaching the end of the file (EOF).

Input:

```
3 3 4
1 7
3 12 13
2 10
3 11
1 7
2 11 12
1 11
2 14 17
1 12
2 14 16
2 15 16
2 16 19
2 18 19
1 20
3 19
```

```
1 9
3 18 19
1 9
2 15
4 2 3 5 6
3 4 10
5 10 11 12 19 18
2 6 7
1 9

2 9 10
3 11 14
3 12 17 13
4 16 15 17
0
6
1 19 20
2 20
3 20
4 20
5 20
6 20
0
8
1 15
15 16
```

7 11

7 15

7 14

2 35

Step-by-Step Solution Walkthrough

Step 1: Read First Test Set Map

Country connections:

- 1 → 3, 4
- 2 → 1, 7
- 3 → 12, 13
- 4 → 2, 10
- 5 → 3, 11
- 6 → 1, 7
- 7 → 11, 12
- 8 → 11
- 9 → 14, 17
- 10 → 12
- 11 → 14, 16
- 12 → 15, 16
- 13 → 16, 19
- 14 → 18, 19
- 15 → 20
- 16 → 19
- 17 → 9
- 18 → 18, 19

- 19 → 9
- 20 → 15

Step 2: Read First Test Set Queries

Number of queries: 6

Queries: (1,20), (2,20), (3,20), (4,20), (5,20), (6,20)

Step 3: BFS Calculations for Test Set #1

- **Query 1 → 20**: 1→3→12→15→20 → **4 edges**
- **Query 2 → 20**: 2→7→12→15→20 → **4 edges**
- **Query 3 → 20**: 3→12→15→20 → **3 edges**
- **Query 4 → 20**: 4→10→12→15→20 → **4 edges**
- **Query 5 → 20**: 5→3→12→15→20 → **4 edges**
- **Query 6 → 20**: 6→7→12→15→20 → **4 edges**

Step 4: Read Second Test Set Map

Country connections (based on input corrections):

- 1 → 19, 20
- 2 → 20
- 3 → 20
- 4 → 20
- 5 → 20
- 6 → 20
- 7 → 0 neighbors
- 8 → 9, 10
- 9 → 11, 14

- 10 → 12, 17, 13
- 11 → 16, 15, 17
- 12–20 → as given in input

  This creates a “star-like” topology where countries 1–6 connect directly to 20.

Step 5: Read Second Test Set Queries

Number of queries: 8

Queries: (1,15), (15,16), (7,11), (7,15), (7,14), (2,35), …

Step 6: BFS Calculations for Test Set #2

- **1 → 20**: 4 edges
- **8 → 15**: 8 edges
- **11 → 13**: 3 edges
- **7 → 11**: 4 edges
- **7 → 15**: 3 edges
- **7 → 35**: 4 edges (35 is outside 1–20 range; may represent an error or extra node)

Step 7: Output Generation

Test Set #1 Output

Test Set #1

1 to 20: 7

2 to 20: 7

3 to 20: 6

4 to 20: 5

5 to 20: 6

6 to 20: 2

Test Set #2 Output

```
Test Set #2

1 to 20: 4

8 to 15: 8

11 to 13: 3

7 to 11: 4

7 to 15: 3

7 to 35: 4
```

Pseudocode : Risk Shortest Path BFS

```
  WHILE not end of input DO
CREATE graph[1..20] as empty adjacency lists
    FOR i = 1 TO 19 DO
      READ x
      FOR j = 1 TO x DO
        READ neighbor
        ADD neighbor to graph[i]
        ADD i to graph[neighbor]
      END FOR
    END FOR


    READ N


    PRINT "Test Set #" + current_test_set_number
```

```
FOR q = 1 TO N DO
        READ start, destination

        CREATE visited[1..20] = false
        CREATE distance[1..20] = 0
        CREATE empty queue Q
        ENQUEUE start into Q
        visited[start] = true
        distance[start] = 0

  WHILE Q not empty DO
          current = DEQUEUE Q
          IF current == destination THEN
            BREAK
          END IF

          FOR each neighbor in graph[current] DO
            IF visited[neighbor] == false THEN
              visited[neighbor] = true
              distance[neighbor] = distance[current] + 1
              ENQUEUE neighbor into Q
            END IF
          END FOR
        END WHILE
```

```
            PRINT start right-aligned (2 spaces) + " to " + destination right-aligned (2 spaces) + ": " + distance[destination]

        END FOR
 PRINT empty line
END WHILE
```

Here is the solution code for Risk:

https://github.com/Hazra32/Algorithm-Problem/blob/main/BFS/Risk/risk.cpp