**Problem: Easy Pranav is in a Relationship**

Pranav and Surbhi are playing a game on an undirected graph. The graph has N nodes and E edges. Pranav starts at vertex M1 and wants to reach vertex M2. Surbhi starts at vertex N1 and wants to reach vertex N2. The shortest distance between Pranav and Surbhi should never exceed K at any point in time.

Your task is to count the number of valid pairs of Surbhi's start and end positions (N1, N2) such that the game is valid, meaning both can reach their respective destinations without the distance between them exceeding K.

**Input Format**

1. The first line contains three integers N, E, and K, denoting the number of vertices, the number of edges in the graph, and the maximum allowed distance between Pranav and Surbhi, respectively.
2. The next E lines each contain two integers A and B, representing an undirected edge between vertices A and B.
3. The next line contains two integers M1 and M2, denoting the source and destination for Pranav.
4. The next line contains a single integer Q, denoting the number of games.
5. Each of the next Q lines contains two integers N1 and N2, denoting the source and destination for Surbhi for that game.

**Constraints**

* 1 <= N <= 1000
* 1 <= E <= min(1000, N×(N−1)/2)
* 1 <= K <= 1,000,000,000
* 1<=Q<= 50,000
* 1 <= A,B,M1,M2,N1,N2 <= N

**Output Format**

Print a single integer, denoting the number of valid games.

**Sample Input**

6 5 3

1 3

2 3

3 4

4 5

3 6

1 5

1

2 6

**Sample Output**

1

**Explanation**

The game may proceed in the following manner. Pranav moves to vertex 3, and then Surbhi also moves to 3. Pranav moves to 4, and then to 5. Surbhi moves to vertex 6.

**Solution**

To solve this problem, we need to perform the following steps:

1. Use BFS or Dijkstra's algorithm to calculate the shortest distances from M1 and M2 for Pranav, and from each possible N1 and N2 for Surbhi.
2. For each game query, verify if the distance between Pranav and Surbhi never exceeds K throughout their paths.

Here's a detailed implementation:

python

import heapq

from collections import deque, defaultdict

def bfs(graph, start, n):

distances = [float('inf')] \* (n + 1)

distances[start] = 0

queue = deque([start])

while queue:

current = queue.popleft()

for neighbor in graph[current]:

if distances[neighbor] == float('inf'):

distances[neighbor] = distances[current] + 1

queue.append(neighbor)

return distances

def dijkstra(graph, start, n):

distances = [float('inf')] \* (n + 1)

distances[start] = 0

min\_heap = [(0, start)]

while min\_heap:

current\_distance, current\_vertex = heapq.heappop(min\_heap)

if current\_distance > distances[current\_vertex]:

continue

for neighbor, weight in graph[current\_vertex]:

distance = current\_distance + weight

if distance < distances[neighbor]:

distances[neighbor] = distance

heapq.heappush(min\_heap, (distance, neighbor))

return distances

def valid\_game\_count(n, edges, k, m1, m2, queries):

graph = defaultdict(list)

for a, b in edges:

graph[a].append(b)

graph[b].append(a)

dist\_from\_m1 = bfs(graph, m1, n)

dist\_from\_m2 = bfs(graph, m2, n)

valid\_count = 0

for n1, n2 in queries:

dist\_from\_n1 = bfs(graph, n1, n)

dist\_from\_n2 = bfs(graph, n2, n)

valid = True

for i in range(1, n + 1):

if abs(dist\_from\_m1[i] - dist\_from\_n1[i]) > k or abs(dist\_from\_m2[i] - dist\_from\_n2[i]) > k:

valid = False

break

if valid:

valid\_count += 1

return valid\_count

# Input reading

n, e, k = map(int, input().split())

edges = [tuple(map(int, input().split())) for \_ in range(e)]

m1, m2 = map(int, input().split())

q = int(input())

queries = [tuple(map(int, input().split())) for \_ in range(q)]

# Processing and Output

result = valid\_game\_count(n, edges, k, m1, m2, queries)

print(result)

This solution uses BFS to calculate the shortest distances from each of the starting points. It checks if the conditions are met for each query to determine if it's a valid game. The approach ensures efficient handling of the constraints given in the problem.

**Test Cases**

**Test Case 1**

**Input**

6 5 3

1 3

2 3

3 4

4 5

3 6

1 5

1

2 6

**Output**

1

**Test Case 2**

**Input**

6 5 2

1 2

2 3

3 4

4 5

5 6

1 6

2

1 4

3 5

**Output**

2

**Test Case 3**

**Input**

7 7 5

1 2

2 3

3 4

4 5

5 6

6 7

1 7

1 7

2

2 5

3 6

**Output**

2

**Test Case 4**

**Input**

5 4 1

1 2

2 3

3 4

4 5

1 5

3

2 4

1 3

4 5

**Output**

1

**Test Case 5**

**Input**

4 3 2

1 2

2 3

3 4

1 4

2

2 3

3 2

**Output**

2

Test Case 6

Input

107 116 5

2 1

3 2

4 1

5 2

6 2

7 5

8 3

9 2

10 9

11 6

12 1

13 4

14 11

15 4

16 8

17 16

18 6

19 14

20 16

21 1

22 2

23 18

24 14

25 9

26 8

27 19

28 20

29 10

30 20

31 7

32 27

33 19

34 17

35 6

36 27

37 29

38 33

39 37

40 21

41 24

42 38

43 30

44 26

45 16

46 30

47 31

48 43

49 33

50 36

51 34

52 28

53 22

54 36

55 17

56 25

57 20

58 48

59 50

60 4

61 41

62 10

63 15

64 52

65 34

66 65

67 62

68 14

69 16

70 64

71 30

72 69

73 2

74 27

75 8

76 24

77 61

78 24

79 2

80 47

81 41

82 16

83 37

84 24

85 4

86 55

87 4

88 82

89 16

90 71

91 63

92 18

93 77

94 23

95 82

96 29

97 23

98 81

99 63

100 60

101 46

102 92

103 58

104 97

105 53

106 28

107 24

82 84

57 99

77 32

24 3

69 27

51 59

68 2

46 65

87 63

92 90

49 43

146

69 90

54 48

25 34

8 30

8 91

86 38

99 11

61 33

11 87

16 70

86 17

47 83

35 41

67 17

90 41

59 90

23 45

30 47

78 76

9 17

99 94

54 90

36 46

55 47

25 70

48 43

18 26

18 52

66 16

68 87

95 58

2 10

102 70

95 72

39 35

20 30

22 73

51 96

11 105

74 75

106 14

10 55

78 66

106 37

13 105

55 107

56 56

48 50

19 74

54 96

2 73

18 23

39 68

50 88

105 17

94 103

30 35

51 1

100 88

76 5

86 62

43 34

50 90

15 107

57 68

95 58

34 5

12 4

72 61

91 1

77 78

36 39

44 18

78 37

105 85

80 15

39 16

87 20

105 102

19 93

62 6

43 95

49 93

30 53

47 14

53 55

23 88

93 66

105 63

34 35

79 46

49 50

100 29

69 29

23 88

15 84

25 96

4 74

82 33

19 21

85 3

7 39

91 100

37 20

94 2

54 66

47 103

8 39

63 8

68 85

95 14

100 52

2 35

18 83

107 75

35 84

77 42

55 99

73 91

12 98

24 65

56 71

99 102

41 54

3 1

70 29

53 63

80 55

29 29

69 28

35 104

44 44

77 30

35 81

52 46

71 75

43 59

38 73

53 11

20 94

50 89

55 103

83 27

50 44

95 11

3 61

Output 6

14