- Import: the top three lines of codes are used for import.
  - The first line “std::collections” provides the data structures like HashMap, HashSet, and VecDeque; which HashMap is used to store the adjacency list for the graph and degree-related data, HashSet is used for storing unique values like neighbors of a node in the graph, and VecDeque is used to get double-ended queue for Breadth-First Search.
  - The other two lines are getting the CSV reader to parse the data and getting the trait for error handling.
- Struct:
  - the struct VideoGame below is used to represent a video game with its attributes like name, genre, publisher, and ratings. String is used for first three attribute and optional value is used to later two since score could be None if missing.
  - Then, the struct GameGraph is used to represent a graph where nodes are video games, and edges represent relationships.
- Methods: next is the implements for GameGraph.
  - The first method “new” initializes an empty graph.
  - The next method “addEdge” adds a bidirectional edge between two games. If either game does not exist in the adjacency list, it creates a new HashSet for its neighbors. Then, adds the second game as a neighbor of the first and vice versa.
  - The next “bfs” method performs the Breadth-First Search, calculates the

shortest path from a start node to all other nodes. It takes the name of the start game as input, and output a HashMap mapping each reachable game to its distance from the start. It first use a queue to explore neighbors level-by-level, then store distances in a HashMap with the start game at distance 0 and add each unvisited neighbor to the queue and record its distance.

- Next is the “degreeDistribution” method which computes the distribution of vertex degrees. It not takes input, and output a HashMap where the key is a degree, and the value is the count of nodes with that degree.
- The last method “degreeCentrality” calculates the degree centrality for each node. It not takes input, and output a HashMap where the key is a game name and the value is its degree.

- Function:
  - The “buildGraph” function is used to load data from the CSV file and build a graph based on shared genres or publishers. It first initializes the necessary components for reading and processing the dataset. Then, the for loop iterate through each record in the CSV file, extract relevant fields from the CSV row, and create a VideoGame struct and store it in the games HashMap. After that, the nested loop builds up the edges in the graph based on shared attributes.
- Main Function: in the main function, it first initializes the data and build up the graph. Then, it performs the Breadth-First Search, degree distribution, and degree centrality with outputs.

- Test:
  - The first test “testAddEdge” ensures the add_edge method creates a bidirectional edge between two nodes. Creates an empty graph. It first adds an edge between “Game1” and “Game2”. then uses “assert_eq!” to verify that “Game2” is a neighbor of “Game1” and “Game1” is a neighbor of “Game2”.
  - Second test “testBfs” validates that BFS correctly computes the shortest path distances from the starting node. It first constructs a graph with a linear structure from “Game1” to “Game4”. Then runs BFS starting from “Game1” and asserts that distance to “Game1” is 0, distance to “Game2” is 1, distance to “Game3” is 2, distance to “Game4” is 3.
  - Third test “testDegreeDistribution” confirms that the degree distribution is calculated correctly. It first creates a cyclic graph where every node has exactly 2 neighbors, then computes the degree distribution and asserts that there are exactly 4 nodes with a degree of 2.
  - The last test “testDegreeCentrality” ensures that degree centrality is computed correctly for each node. It first creates a star graph where “Game1” connects to 3 other games, then calculates degree centrality, and asserts that “Game1” has a centrality of 3 and all other nodes have a centrality of 1.