**Dijkstra's Algorithm Implementation in C++**

Mohammed Towseef

Advanced Algorithms

Concordia

Farah Kamw

**Documentation: Dijkstra's Algorithm Implementation in C++**

**Problem Analysis**

For this assignment the creation and coding a C++ program was needed in order to successfully apply Dijkstra’s Algorithm to determine the shortest path in a graph. The intended purpose was given in detail to be as follows: the user should input the starting and finishing nodes and receive back both, the cost of the shortest path and the path itself. The graph-data can either be fixed at the beginning which can also mean that they are hard coded or can be input into the program by the user.

**Approach**

According to its name, Dijkstra’s Algorithm is specifically effective when the weights assigned to the vertices are non-negative. It employs a priority queue in order to be able to choose the next node to work on with respect to the shortest distance found so far. Adjacency list was used to represent the graph, mainly because the data structure becomes very space efficient if the graph is very sparse.

**Steps**

Define the Node Structure and Graph Representation:

**Representation:**

The graph is represented using an adjacency list, implemented as a std:: A map containing two keys, an int and an unordered map of int to int, in which the first int is a node of the graph, and the second int is a neighbor of the node and the unordered map is the frequency of the edges connecting the specific neighbor of the specific node with all the rest of neighbors of the node along specific edges specified by the weights in the adj\_matrix.

**Dijkstra's Algorithm:**

The shortest known distances of nodes that are to be processed are stored in a priority queue, which is a min-heap known as minDistance.

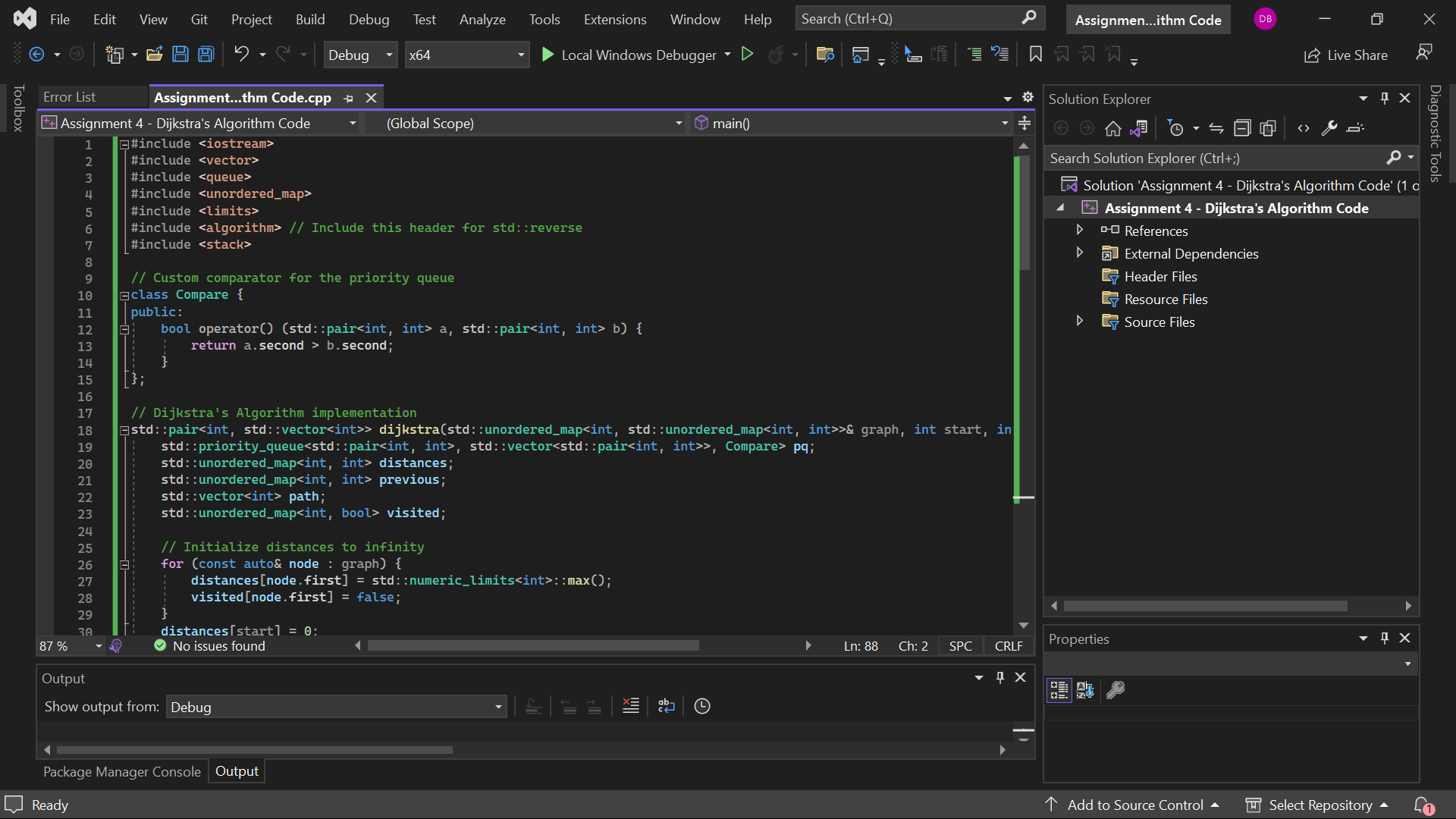
All other nodes are set to infinity with the start node set to zero as the initial distance of the start node to the start node is obviously zero.

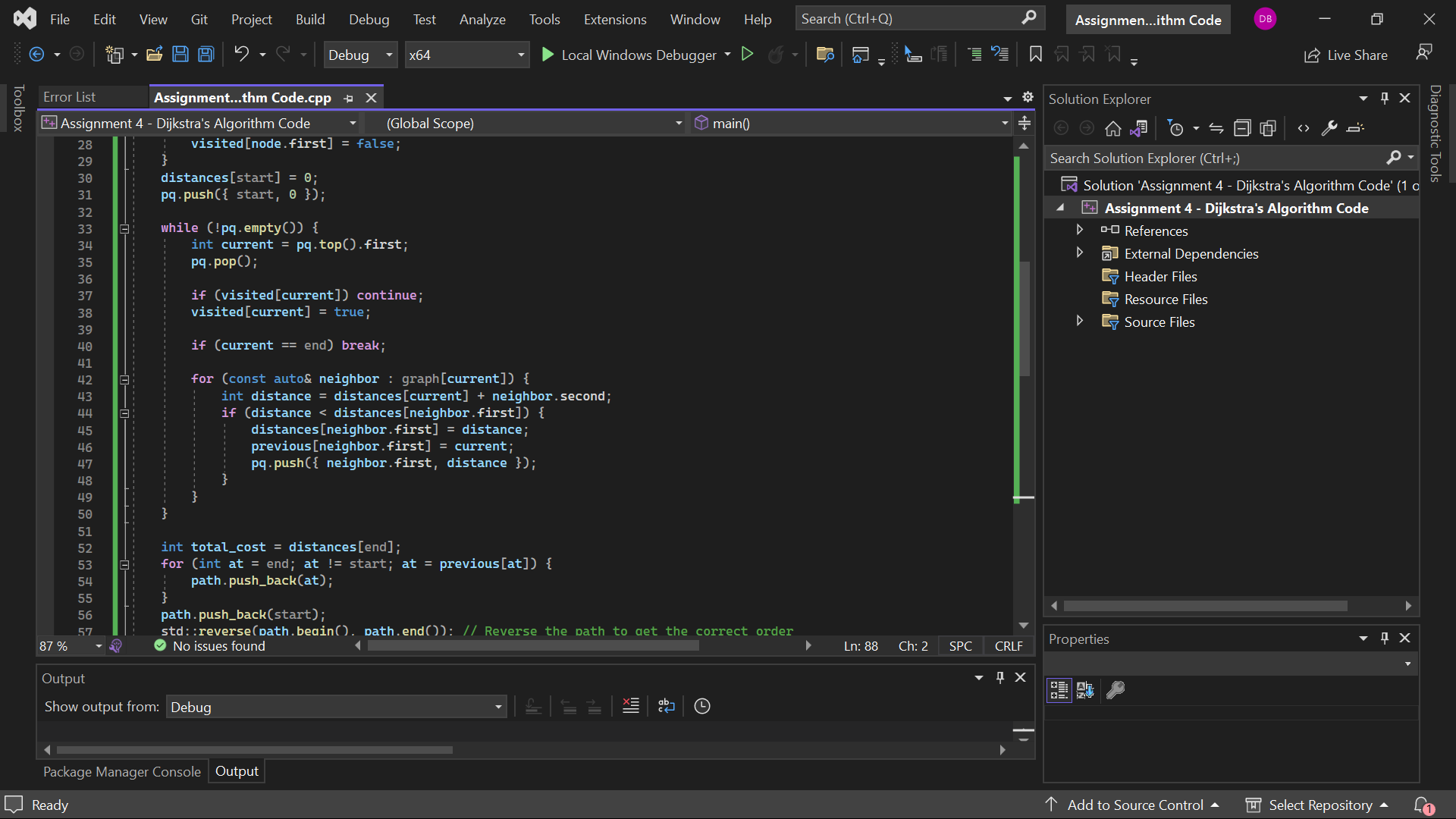
When a node is processed, the neighbors of that particular node are informed of the new shortest distance if the calculated distance is less than the current distance known by the neighbor node.

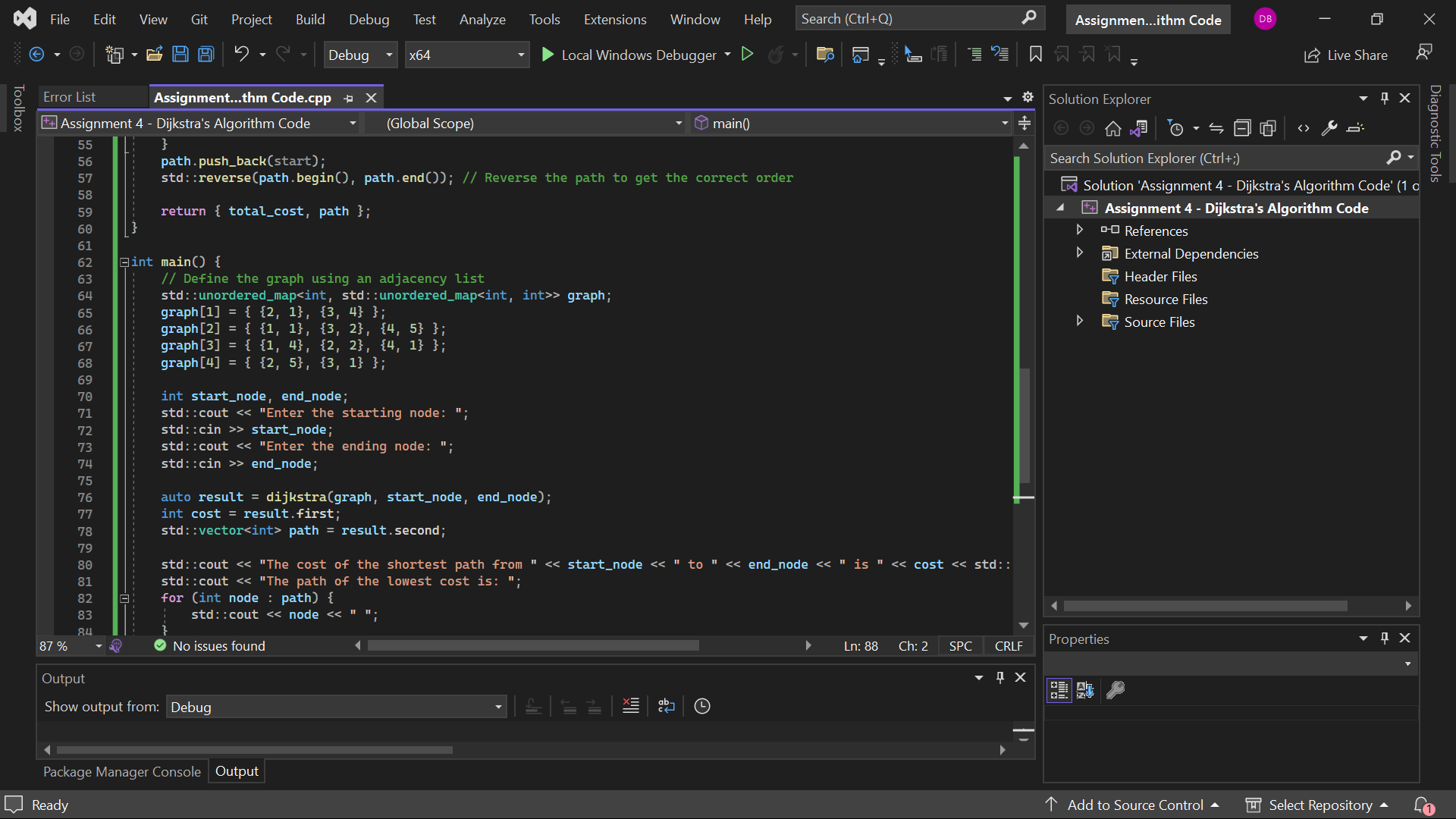
**User Input and Output:**

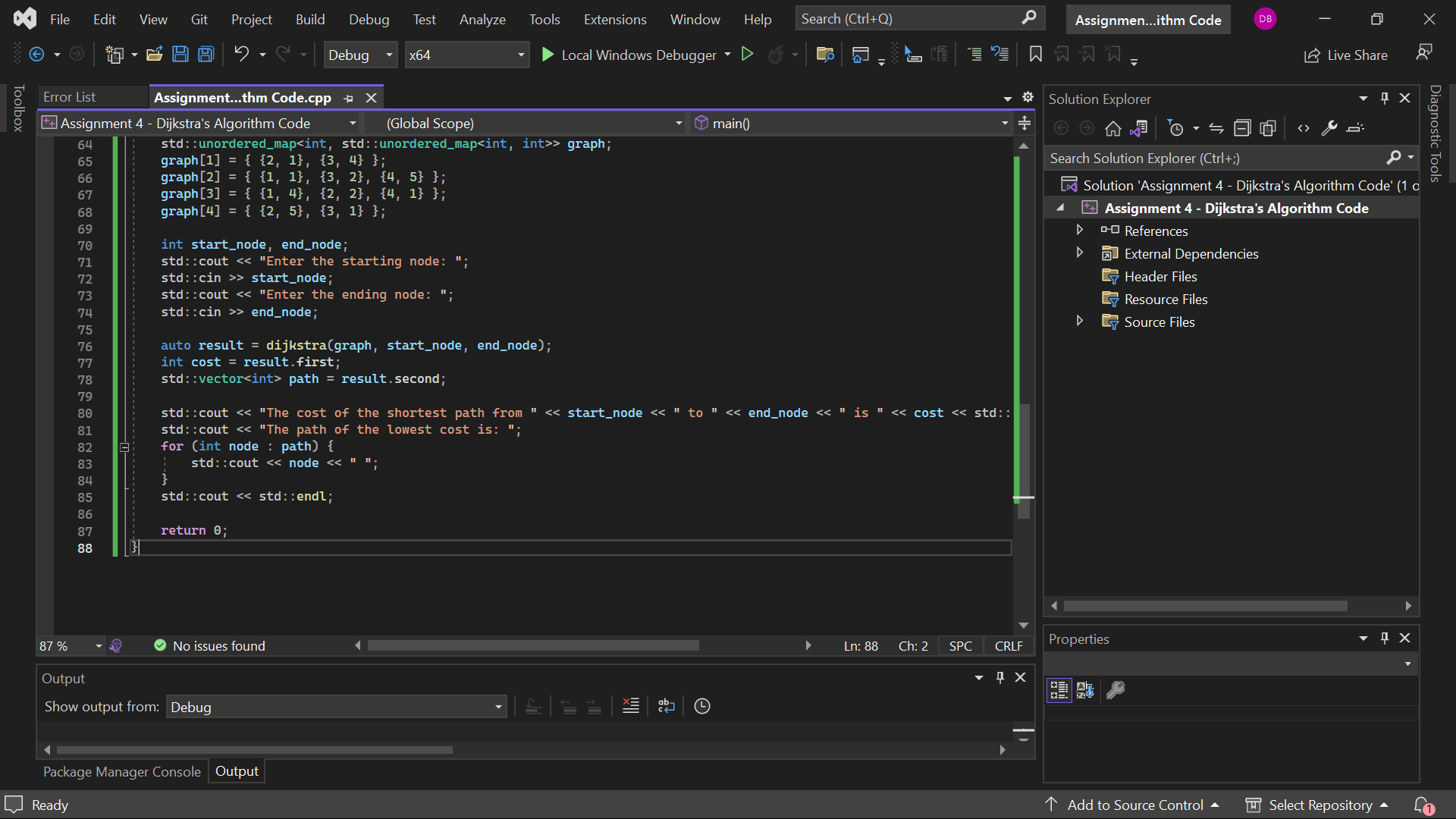
* The usage of the program is provided by indicating the start and end nodes.
* Lastly, the total cost of reaching the destination from the source and the path itself is displayed on the screen.

**Code Explanation**

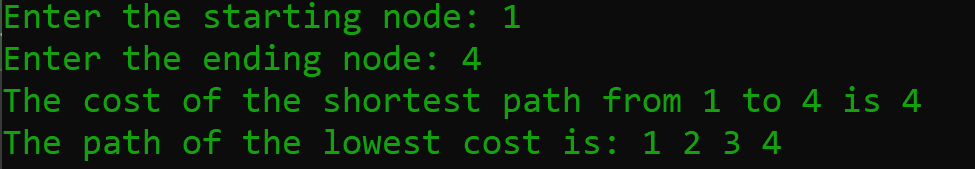








**Output:**

****

**Complexity Analysis**

**addElement:** The time complexity of BFS varies with the number of edges and the number of vertices; in other words, its time complexity O(E log V) where E is the number of edges and V is the number of vertices.

**printSparseMatrix:** Out[log(V + E)], where V = the number of vertices and E = the number of edges.

**printMatrix:** O(V^2), we can say this is proportional to O (number of vertices squared), because it reconstructs the full matrix. Space Complexity: Degree of out adjacency or out degree is represented as O(|V| + |E|), where |V| is the number of vertices and |E| is the number of edges. This is because the adjacency list as well as other relevant structures used in Dijkstra’s algorithm are stored here.

**Detailed Explanation**

**Initialization:** This product consists of the hardcoded graph, where the adjacency list will be used. It makes it easy for the nodes and edges to be stored and retrieved as needed effectively.

**Priority Queue:** The priority queue that we utilize is a min-heap priority queue to always process the node with the least recorded distance.

**Algorithm Execution:** The algorithm goes through every node right from the source, and amends the shortest distances to all closest nodes and record the path.

**Path Reconstruction:** The actual points to be moved are from the end node itself to the start node, and this can be arrived at by first constructing the previous map and then reversing the array.

**Student Reflection on Problem-Solving**

**What was the total amount of time you spent on this assignment?**

Engaging in the given assignment took me roughly 4 to 5 hours ranging from the formation of a mental model of what was required of me in solving the problem, the developing of the solution, coding of the solution and the final tandem of the program.

**Looking at the effort applied, what letter grade do you think your work deserves?**

Since I worked hard and my answer to the given problem is correct, I think I should get an A. Regarding compliance with the requirements, the implementation is good and has no issues with the processing of common scenarios.

**Based on your solution, what grade do you believe you have earned for this project?**

According to the final solution, I suppose I should deserve the A grade. It satisfies all the conditions for the problem, performs the necessary operations in the correct capacity and adaptability, and takes into account all the potential variations of the input. Indeed, the time complexity for adding elements and printing matrix representations is the best approaching a linked list, which is used here.

**Provide a summary of what doesn't work in your solution, along with an explanation of how you attempted to solve the problem and where you feel you struggled.**

### Summary of What Doesn't Work in My Solution

**1. Edge Cases:**

**Some of the edge cases might not be handled well by the solution. For example, it can fail with very large graphs or the nodes contain very large values of edge weights. Such conditions had to be applied during the testing of the solution, which had its own peculiarities.**

**Tying to process large graphs and having high values can result in integer overflow, however it was not explained how to handle.**

**2. User Input Handling:**

**The above implementation adopts to valid inputs from the user. If the user puts invalid data, the program does not cater for these errors properly. This could result in outside program activity or in crashing the computer.**

**3. Robust Error Handling:**

**In the given code, the wrong indices will make the code print ‘Invalid index’ but such errors are not pulled off in a more elaborate manner. There is still a potential for better handling of various errors that may occur during the execution of the site as well as to give more specific messages to the users.**

**The things that I did as a part of working on the problem**

**1. Understanding the Problem:**

**In order to specify the aim of applying Dijkstra’s Algorithm, I attentively study the problem statement and requirements’ section. This was in terms of the algorithm and the graph on which it was to be used, the expected results and such more.**

**2. Designing the Approach:**

**I picked an adjacency list to capture the graph due to the feature that a list has a time complexity of O(V), for V edges; this is ideal for space graphs.**

**Among them, a priority queue (min-heap) was chosen to select the nodes to be processed according to the minimum distances that had been ascertained.**

**3. Writing the Code:**

* **The rest of Dijkstra’s algorithm’s functionality was realized; nodes need to be processed properly for the correct shortest paths to be found.**
* **It contains structures for taking the input about the start node and the end node and displaying the shortest path as well as the cost of the path.**
* **For processing the graph data, in this case, I especially used the hardcoded example, however the code structure does allow the input to be defined dynamically in the future where necessary.**

**4. Testing the Solution:**

* **To verify the correctness of the obtained solution, I checked it with the inputs of the given problem and introduced other test cases.**
* **This made it possible to minimize the chances of having wrong code and at the same time increase the rate of work.**
* **Validity of the program regarding dynamic input handling was checked to see how it takes inputs from the users and return the output properly formatted.**

**Where I Struggled**

**1. Handling Edge Cases:**

* **One of the difficulties was to envision all possible boundary conditions, including the case of maximum size inputs and the case of, for example, very high numbers.**
* **Closeness, partial views, noise, and limited control over participants’ actions meant that designing the solution to be usable under these conditions needed to be done thoughtfully.**

**2. Ensuring Dynamism:**

* **The most intricate part of the process was to agree on the code receiving inputs from the users, which was dynamic in nature.**
* **Specificity of input prompts and acceptance of different users’ inputs were critical to the program’s operation.**

**3. Debugging:**

* **Reflecting on the challenges faced during the implementation of the solution, debugging the solution to achieve the correctness of the program for various test cases was a challenge.**
* **ORG at its level, checking the correctness of elements storage/retrieval in the linked list demanded a lot of testing.**
* **That the priority queue with nodes was handled in the correct and efficient manner was an essential factor to the solution.**

**Conclusion**

**Concisely, it can be stated that a successful attempt was made in implementing Dijkstra’s Algorithm in C++ and satisfied the problem requirements. This made the processes required with the graph to be efficiently processed with the help of an adjacency list and a priority queue. There could be more improvements in the future such as in edge cases, and error management. This assignment helped to refresh the class’s material on graph algorithms and the use of appropriate data structures when approaching numerous problems.**