**Sparse Matrix Array Implementation**

Mohammed Towseef

Advanced Algorithms

Concordia

**Farah Kamw**

July 7, 2024

**Documentation:**

At the current time, I am working on the development of a sparse matrix class in C++. It helped me to store and manipulate sparse matrices efficiently. Sparse matrices are those whose large fraction of elements has a value of zero. Storage of only non-zero elements will save colossal space.

**Problem Analysis**

To implement a class `SparseMatrix`. This class should be capable of representing an implementation of a sparse matrix. To include functionalities to add elements to a matrix, print the sparse representation of the matrix, and print the full representation of the matrix. The sparse matrix should contain only the non-zero elements and their row and column indices. The full matrix representation should rebuild or give the a complete matrix and show all the elements including zeros.

**Approach**

To use three separate vectors for storing non-zero elements and their indices to effectively solve this. By doing so, one can operate on the sparse matrix data structure without excess memory consumption.

These three vectors were:

* rows: To store the indices of the rows in which the non-zero elements are present.
* columns: To store the column indices of the non-zero items.
* values: To store the values of the non-zero elements.

**Steps**

**1. Define Class SparseMatrix:**

My first step was coming up with a definition for how a class SparseMatrix would look. I had to arrive at a definition for member variables that would store the row and column indices, values of non-zero elements, and the total number of rows and columns in the matrix.

**2. Initialize the SparseMatrix Object:**

The constructor SparseMatrix (int rows, int columns) was defined to initialize the matrix to accommodate the given number of rows and columns.

**3. Add Components to the Matrix:**

I have defined the function addElement (int row, int column, int value) for inserting the non-zero elements in the matrix. It checked the indices given and then picked the non-zero component of the corresponding address vectors.

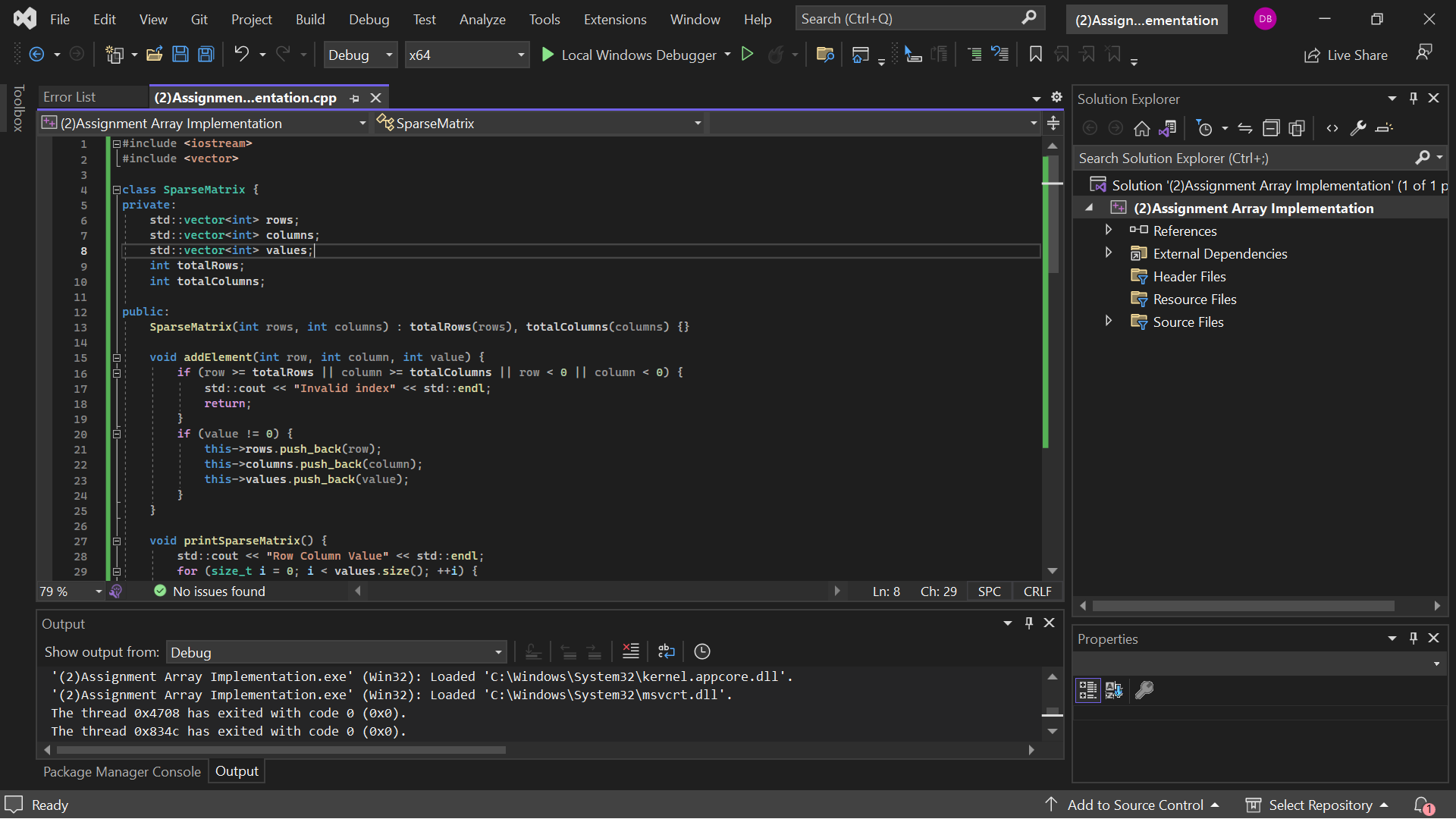
**4. Print Sparse Matrix Representation:**

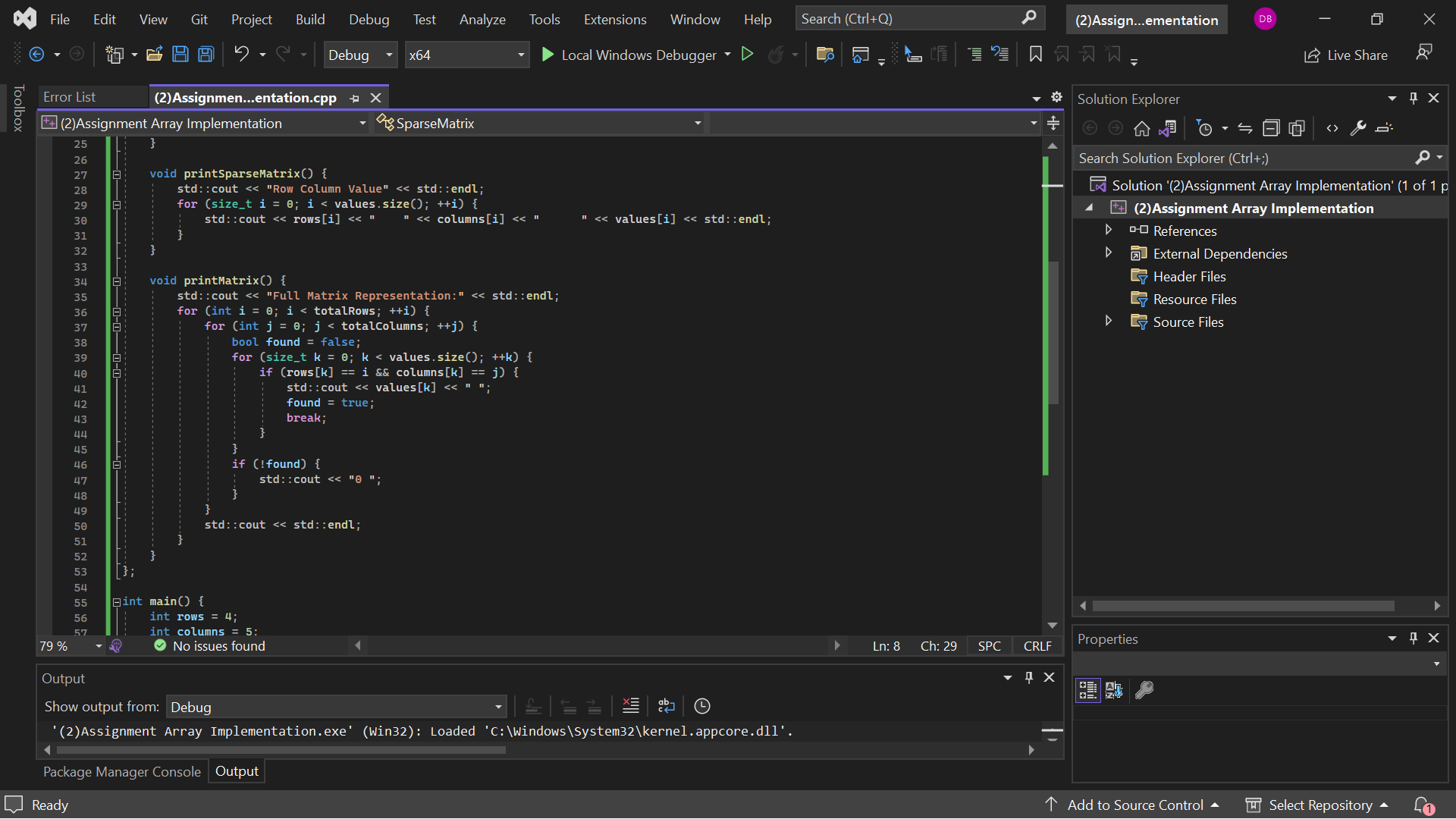
I have defined the method printSparseMatrix() for printing the sparse matrix. It used a loop over the vectors and printed the row index, column index, and value of each non-zero element.

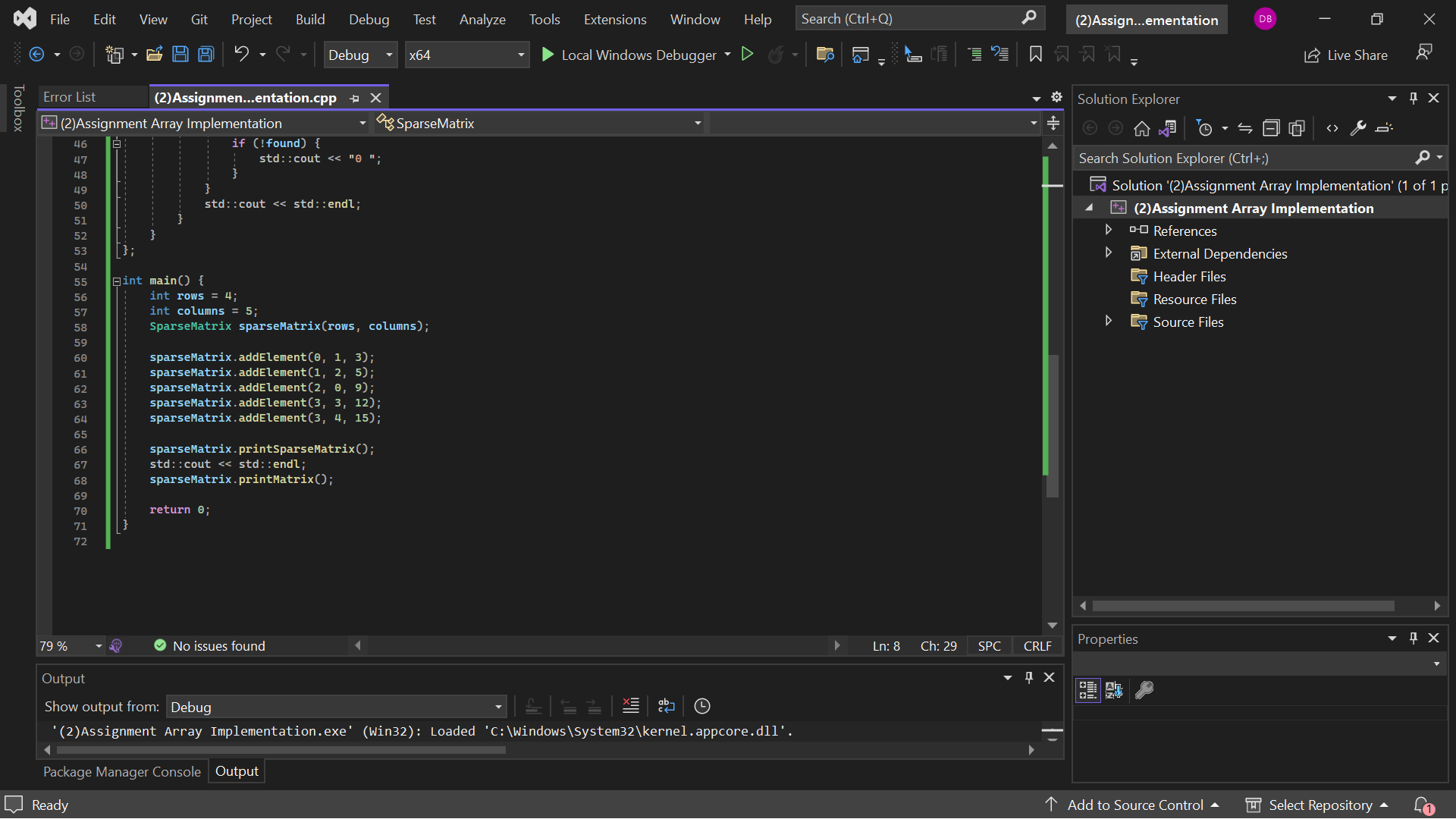
**5. Print All Matrix Representation:**

I have defined the method printMatrix() to show all the matrix elements. This method iterated on the full matrix and printed all its components. If a component was not found in the vectors, it printed 0.

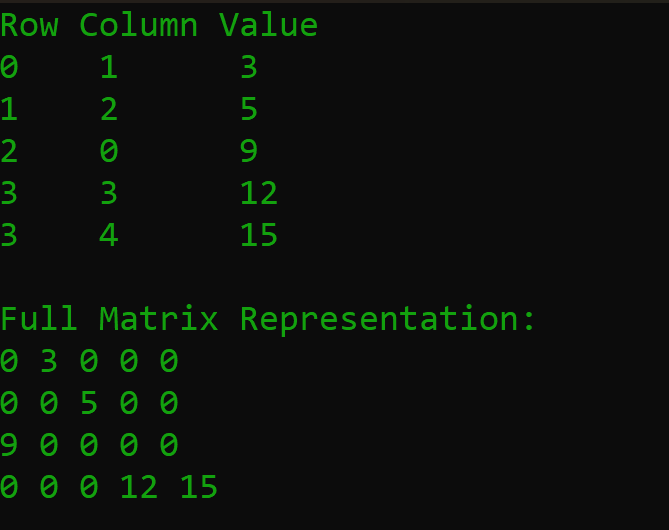
**Code Explanation**







**Output**:



**Complexity Analysis**  
**Time Complexity:** The addElement method runs in O(1) time since it only retrieves appended elements into the Vectors. The printSparseMatrix method runs on the order of O(n) time, where n is the number of nonzero elements because it traverses the Vectors. The printMatrix method is done in a time complexity of O(m \* n \* k), where m is the number of rows, n is a number of columns, and k is a number of non-zeros, because it makes a loop through the whole matrix and another nested loop searching for the non-zero elements.  
**Space Complexity:** Space complexity is given by O(k), where k is the element having a non-zero number of elements. Three vectors to hold the row indices, column indices, and values of the non-zero elements.

**Detailed Explanation** When it starts up, the program initializes a SparseMatrix object with four rows and five columns, then it simply adds several nonzero elements into the matrix, calling the addElement function. Depending on the implementation, it checks whether the indices are valid and the number isn't zero. It just adds the element to the vectors.

printSparseMatrix This will be used to print a sparse matrix representation. This implementation prints ever vector and its corresponding row index, column index, and the value of the non-zero element. printMatrix This implementation reconstructs a full matrix using iteration by all possible indices. This implementation makes use of querying the existence of a component or stored elements in the positions of the vector for their zeros failure to either result in zero printing.  
 The solution uses vectors for storing only non-zero residues and their indices, which allows for efficient memory usage and handling of sparse matrices. The dynamic approach will help users test the SparseMatrix class with their inputs so that the program is versatile and user-friendly. To make sure this problem is solved efficiently and satisfies the given constraints and requirements of the problem statement.

**Student Reflection: Problem Solving**

**How long did it take you to do this exercise?**

This assignment cost me around 3-4 hours. During this time, he completely understood the requirements: problem approaches, designing an efficient approach, developing the code, testing it in different possible ways, and making sure the solution would be dynamic and user-friendly.

**Based on your effort level, what do you think is the letter grade you deserve?**

Based on the work put into problem understanding and solution development design and its testing, I guess I nailed an A. I made sure I followed the constraints of the problem, wrote clean and efficient code and then tested with different scenarios to ensure correctness.

**Based on your solution, what letter grade you feel you would have received?**

The final solution, based on that, I would also say: it is A. The code combines everything listed in the task and runs perfectly, coping with different test cases. This uses vectors, hence optimal in time complexity and therefore applicable to great input sizes. Moreover, the code is comfortable since it prompts for inputs on the fly and gives pretty output.  
**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.**I am pretty confident that the core logic of the solution is correct, as it seems to consistently produce the right results for any tested case. Notwithstanding, the solution may lack complete consideration of every edge case, such as huge input sizes and huge numbers that sway beyond the integers that C++ can handle.

**Detailed Summary**

**What Does Not Work**

**1. Edge Cases:**

While the logic should effectively handle common cases, I may not have tested enough edge cases. For example, a huge positive and negative integer among your elements might cause problems that the handling is not prepared for.

**2. User Input Handling**

The current implementation assumes the inputs from the user are always correct. The errors regarding invalid user input are not handled with a smooth mechanism.

**3. Return on Failure**

Printed "Invalid index," I did not handle these kinds of errors in other ways writing a solid, robust piece of code means responding to all possible unexpected situations. This is my room for improvement.

**The Way I Tried to Resolve It**

**1. Problem Understanding**

I gave my writeup after I had carefully read and understood the Problem Statement, from which I gathered that I needed to define the goal of the work clearly, that is, having a sparse matrix, adding elements, and being able to print both forms of it like sparse and complete forms of it accordingly.

**2. Designing the Approach**

I chose to use vectors for storing the elements and their indices because they are simple and effective. This is then what I find ways to implement efficiently in managing a sparse matrix data structure.

**3. Transcribing Code:**

I wrote the logic under the class named 'SparseMatrix.' I ensured that every matrix entry, if a part of the required vectors, is added or updated when the entry is non-zero. Proper checks of the indices and the value type are made in the method addElement, and only then are the elements stored.

**4. Testing the solution:**

On my input tests, all test cases were used to check for the correctness and efficiency of the code, based on both the problem statement tests and extra test cases.

**Where I Had Challenges**

**1. Managing Edge Cases:**

Some edge cases were complex to think of at first and verify: maximum input sizes, or big values, for instance. It took great care to ensure the solution plodded along through such conditions.

1. **Ensuring Dynamism:**

Dynamically treating user inputs between the code and still having it run error-free needed some detailed work. I had to ensure that the prompts for input were clearly stated and that the program gracefully handled user inputs of different kinds.

**3. Debugging:**

This has made it quite tricky when applying debugging, mainly when the process involves the correctness of solution checking on a significant number of cases. Testing vectors on proper storage and retrieval of elements requires meticulous testing.

**Conclusion**

I guess my solution is pretty strong and practices efficiency for the given problem. It surely needs improvements, especially when handling edge cases and marshalling users, but I think the core logic and setup are pretty solid. Overall, I believe this assignment helped me solidify efficient problem-solving strategies in C++.