```
#include<iostream>
using namespace std;

class sparse_matrix{
	private:
		int rows;
		int cols;
		int non_zero;
		int sp_row[100];
		int sp_col[100];
		int sp_val[100];
		int mat[100][100];
	public:
		int sp_mat[100][3];
		int sp_tran[100][3];
		int fast_tran[100][3];
		void read_sparse_matrix(int r, int c);
		void display_matrix();
		void display_sparse_matrix(int sp[100][3]);
		void simple_transpose();
		void fast_transpose();
		void add_sparse_matrix();
		void multiply_sparse_matrix();
		void optimized_multiply_sparse_matrix();
};

void sparse_matrix::read_sparse_matrix(int r, int c){
	rows = r;
	cols = c;
	non_zero = 0;
```

```
    int k = 0;   //for iteration of sp_row[] sp_col[] and sp_val[]
    cout << "Enter " << rows*cols << " elements for "<< rows << "X" << cols << " matrix: \n";
    for(int i=0; i<rows; i++){
        for(int j=0; j<cols; j++){
            cin >> mat[i][j];
            if(mat[i][j] != 0){
                non_zero++;
                sp_row[k] = i;
                sp_col[k] = j;
                sp_val[k] = mat[i][j];
                k++;
            }
        }
    }
    k = 0;
    sp_mat[0][0] = rows;
    sp_mat[0][1] = cols;
    sp_mat[0][2] = non_zero;
    for(int i=1; i<=non_zero; i++){
        for(int j=0; j < 3; j++){
            if(j==0) sp_mat[i][j] = sp_row[k];
            else if(j==1) sp_mat[i][j] = sp_col[k];
            else if(j==2) sp_mat[i][j] = sp_val[k];
        }
        k++;
    }
}

void sparse_matrix::display_matrix(){
    for(int i=0; i<rows; i++){
```

```
		for(int j=0; j<cols; j++){
			cout << mat[i][j] << " ";
		}
		cout << "\n";
	}
}

void sparse_matrix::display_sparse_matrix(int sp[100][3]){
	for(int i=0; i<=sp[0][2]; i++){
		for(int j=0; j<3; j++){
			cout << sp[i][j] << " ";
		}
		cout << "\n";
	}
}


void sparse_matrix::simple_transpose(){
	sp_tran[0][0] = cols;
	sp_tran[0][1] = rows;
	sp_tran[0][2] = non_zero;
	if(non_zero == 0) return;
	int i = 1;
	for(int j=0; j<cols; j++){
		for(int k=1; k<=non_zero; k++){
			if(sp_mat[k][1] == j)
			{
				sp_tran[i][0] = sp_mat[k][1];
				sp_tran[i][1] = sp_mat[k][0];
				sp_tran[i][2] = sp_mat[k][2];
```

```
                        i++;
                    }
                }
            }
            cout << "Simple Transpose of the matrix is = \n";
            display_sparse_matrix(sp_tran);
}


void sparse_matrix::fast_transpose() {
    fast_tran[0][0] = cols;
    fast_tran[0][1] = rows;
    fast_tran[0][2] = non_zero;

    if (non_zero == 0) return;  // If matrix is empty -> return

    int freq[100] = {0};   // Frequency of elements in each column
    int index[100];  // Index for positions in the transpose matrix

    // Step 1: Count the frequency of each column in the original matrix
    for (int i = 1; i <= non_zero; i++) {
        freq[sp_mat[i][1]]++;
    }

    // Step 2: Calculate the starting index for each column in the transposed matrix
    index[0] = 1;
    for (int i = 1; i < cols; i++) {
        index[i] = index[i - 1] + freq[i - 1];
    }
    // Step 3: Place elements in the transposed matrix based on the calculated indices
    for (int i = 1; i <= non_zero; i++) {
        int col = sp_mat[i][1];
        int pos = index[col];

        fast_tran[pos][0] = col;
        fast_tran[pos][1] = sp_mat[i][0];
        fast_tran[pos][2] = sp_mat[i][2];

        index[col]++;  // Increment index for the next element in the same column
    }

    cout << "Fast transpose of matrix = \n";
    display_sparse_matrix(fast_tran);
}

void sparse_matrix::add_sparse_matrix() {
    int row1, col1, row2, col2;
    cout << "Enter row of second matrix to add with current matrix: ";
    cin >> row2;
    cout << "Enter column of second matrix to add with current matrix: ";
    cin >> col2;
    row1 = rows;  // Set row1 to the number of rows in the current matrix
    col1 = cols;  // Set col1 to the number of columns in the current matrix

    // Check if the dimensions of the matrices match
    if(row1 != row2 || col1 != col2) {
        cout << "Matrices cannot be added\n";
        return;
    }
    sparse_matrix B;
    B.read_sparse_matrix(row2, col2);  // Read the second sparse matrix
    cout << "\n\nOriginal matrix B: \n";
    B.display_matrix();
    cout << "\n\nSparse matrix B: \n";
    B.display_sparse_matrix(B.sp_mat);

    cout << "\n\nAdding two sparse matrices: \n";
    cout << "Sparse matrix A: \n";
    display_sparse_matrix(sp_mat);
    cout << "\n  +  \n\nSparse matrix B: \n";
    display_sparse_matrix(B.sp_mat);

    sparse_matrix C;  // Resultant matrix

    int i = 1, j = 1, k = 1;
    // Loop through the non-zero elements of both matrices
    while(i <= sp_mat[0][2] && j <= B.sp_mat[0][2]) {
        if(sp_mat[i][0] == B.sp_mat[j][0]){
                if(sp_mat[i][1] == B.sp_mat[j][1]) {  // If columns match, add the values
                C.sp_mat[k][0] = sp_mat[i][0];
                C.sp_mat[k][1] = sp_mat[i][1];
                C.sp_mat[k][2] = sp_mat[i][2] + B.sp_mat[j][2];
                i++;
                j++;
                k++;
            }
            else{
                if(sp_mat[i][1] < B.sp_mat[j][1]) {  // If the current element in A comes before B
```

```
                     C.sp_mat[k][0] = sp_mat[i][0];
                     C.sp_mat[k][1] = sp_mat[i][1];
                     C.sp_mat[k][2] = sp_mat[i][2];
                     i++;
                     k++;
                  }
                  else {  // If the current element in B comes before A
                     C.sp_mat[k][0] = B.sp_mat[j][0];
                     C.sp_mat[k][1] = B.sp_mat[j][1];
                     C.sp_mat[k][2] = B.sp_mat[j][2];
                     j++;
                     k++;
                  }
                     }
             }
else{
   if(sp_mat[i][0] < B.sp_mat[j][0]){
            C.sp_mat[k][0] = sp_mat[i][0];
            C.sp_mat[k][1] = sp_mat[i][1];
            C.sp_mat[k][2] = sp_mat[i][2];
            k++;
            i++;
                     }
                     else{
                              C.sp_mat[k][0] = B.sp_mat[j][0];
            C.sp_mat[k][1] = B.sp_mat[j][1];
            C.sp_mat[k][2] = B.sp_mat[j][2];
            j++;
            k++;
                     }
```

```
                        }
    }

    // If there are remaining elements in A, add them to the result
    while(i <= sp_mat[0][2]) {
        C.sp_mat[k][0] = sp_mat[i][0];
        C.sp_mat[k][1] = sp_mat[i][1];
        C.sp_mat[k][2] = sp_mat[i][2];
        i++;
        k++;
    }

    // If there are remaining elements in B, add them to the result
    while(j <= B.sp_mat[0][2]) {
        C.sp_mat[k][0] = B.sp_mat[j][0];
        C.sp_mat[k][1] = B.sp_mat[j][1];
        C.sp_mat[k][2] = B.sp_mat[j][2];
        j++;
        k++;
    }

    C.sp_mat[0][2] = k-1;  // Set the number of non-zero elements in the result
    cout << "\n\nResultant sparse matrix: \n";
    C.display_sparse_matrix(C.sp_mat);
    cout << "\n\n";
}

void sparse_matrix::multiply_sparse_matrix() {
    int row1, col1, row2, col2;
    cout << "Enter row of second matrix to multiply with current matrix: ";
```

```
cin >> row2;
cout << "Enter column of second matrix to multiply with current matrix: ";
cin >> col2;
row1 = rows;  // Set row1 to the number of rows in the current matrix
col1 = cols;  // Set col1 to the number of columns in the current matrix

// Check if the matrices can be multiplied
if(col1 != row2) {
    cout << "Matrices cannot be multiplied\n";
    return;
}

sparse_matrix B;
B.read_sparse_matrix(row2, col2);  // Read the second sparse matrix
cout << "\n\nOriginal matrix B: \n";
B.display_matrix();
cout << "\n\nSparse matrix B: \n";
B.display_sparse_matrix(B.sp_mat);
cout << "\n\nTranspose of matrix B: \n";
B.fast_transpose();  // Compute the transpose of matrix B

sparse_matrix C;  // Resultant matrix
C.sp_mat[0][0] = row1;
C.sp_mat[0][1] = col2;
C.sp_mat[0][2] = 0;

int k = 0;  // Counter for non-zero elements in the result matrix
int pos = 1;  // Position in the resultant matrix

// Loop through non-zero elements of A and B's transpose
```

```
    for(int i = 1; i <= sp_mat[0][2]; i++) {
        for(int j = 1; j <= B.fast_tran[0][2]; j++) {
            if(sp_mat[i][1] == B.fast_tran[j][1]) {  // Match the column of A with the row of B's transpose
                C.sp_mat[pos][0] = sp_mat[i][0];
                C.sp_mat[pos][1] = B.fast_tran[j][0];
                C.sp_mat[pos][2] = sp_mat[i][2] * B.fast_tran[j][2];

                // Combine entries if they have the same row and column
                if((C.sp_mat[pos-1][0] == C.sp_mat[pos][0] && C.sp_mat[pos-1][1] == C.sp_mat[pos][1]) && pos != 1) {
                    C.sp_mat[pos-1][2] += C.sp_mat[pos][2];
                    pos--;
                    k--;
                }
                pos++;
                k++;
            }
                        }
    }

    C.sp_mat[0][2] = k;  // Set the number of non-zero elements in the result matrix
    cout << "\n\nResultant sparse matrix: \n";
    C.display_sparse_matrix(C.sp_mat);
    cout << "\n\n";
}



int main(){
            sparse_matrix A,TA;
```

```
int rows;
int cols;
cout << "Enter number of rows: ";
cin >> rows;
cout << "Enter number of columns: ";
cin >> cols;
cout << "\n\nEnter input for matrix A: \n";
A.read_sparse_matrix(rows, cols);
cout << "\n\nOriginal matrix A: \n";
A.display_matrix();
cout << "\n\nSparse matrix A: \n";
A.display_sparse_matrix(A.sp_mat);
ask:
        int choice;
        cout << "\n\nWhat you want to do? \nEnter: \n";
        cout << "1 - To transpose a sparse matrix\n";
        cout << "2 - To fast-transpose a sparse matrix\n";
        cout << "3 - To add two sparse matrix\n";
        cout << "4 - To multiply two sparse matrix\n";
        cout << "5 - To exit\n";
        cin >> choice;

        switch (choice){
                case 1:
                        A.simple_transpose();
                        cout << "\n\n";
                        break;
                case 2:
                        A.fast_transpose();
                        cout << "\n\n";
                break;
            case 3:
                A.add_sparse_matrix();
                cout << "\n\n";
                break;
            case 4:
                A.multiply_sparse_matrix();
                cout << "\n\n";
                break;
            case 5:
                exit(0);
            default:
                goto ask;
        }
        goto ask;
    return 0;
}
```