Homework 3A

2021315385

이건 / Gun Daniel Lee

Problem explanation

- Find the shortest path that minimizes the sum of weights of all points that the path passes

- Each path must only have at most 1 zero

Problem explanation

- Input:

1. Starting and destination point indices
2. 10-by-10 matrix; weight being an integer between -100 and 100, or -9999 (-9999 means not included)

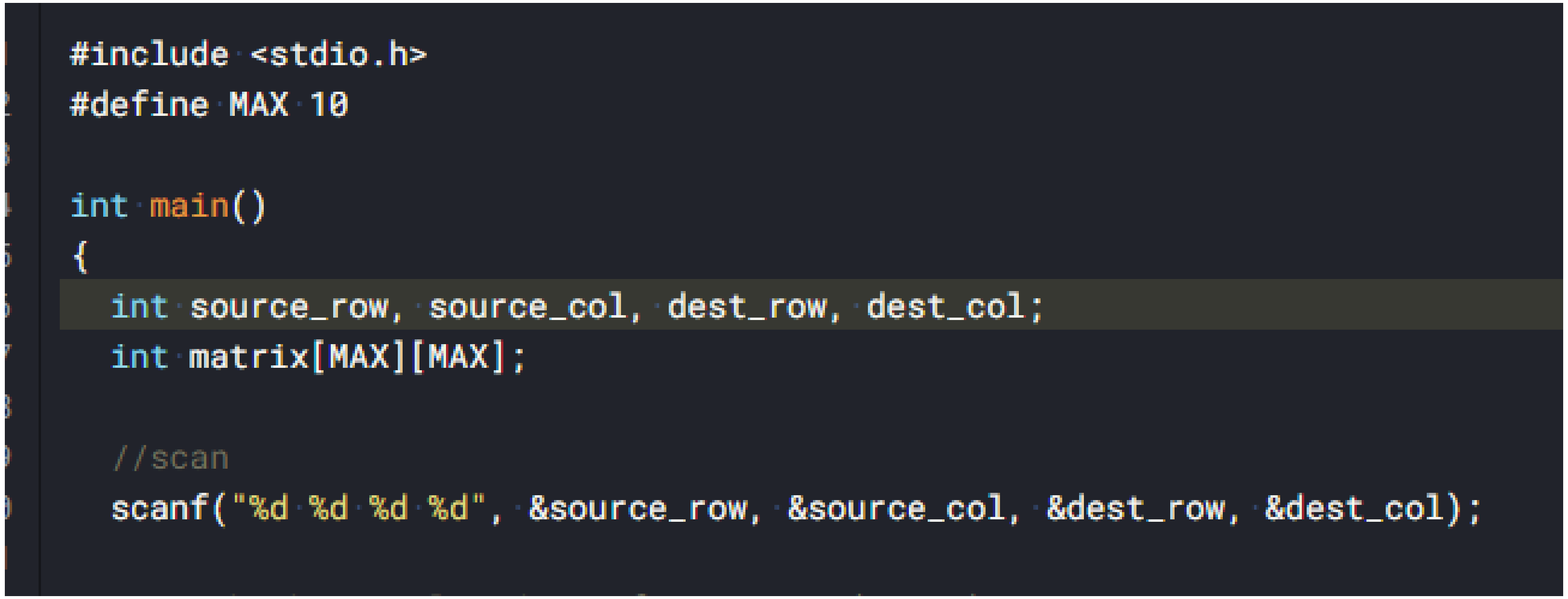
- Output:

1. The sum of minimum weight
2. The series of point indices from starting to destination point including themselves

Solution explanation

source_row, source_col, dest_row, dest_col, matrix:

- Initial inputs of the program
- Scan for the source and destination

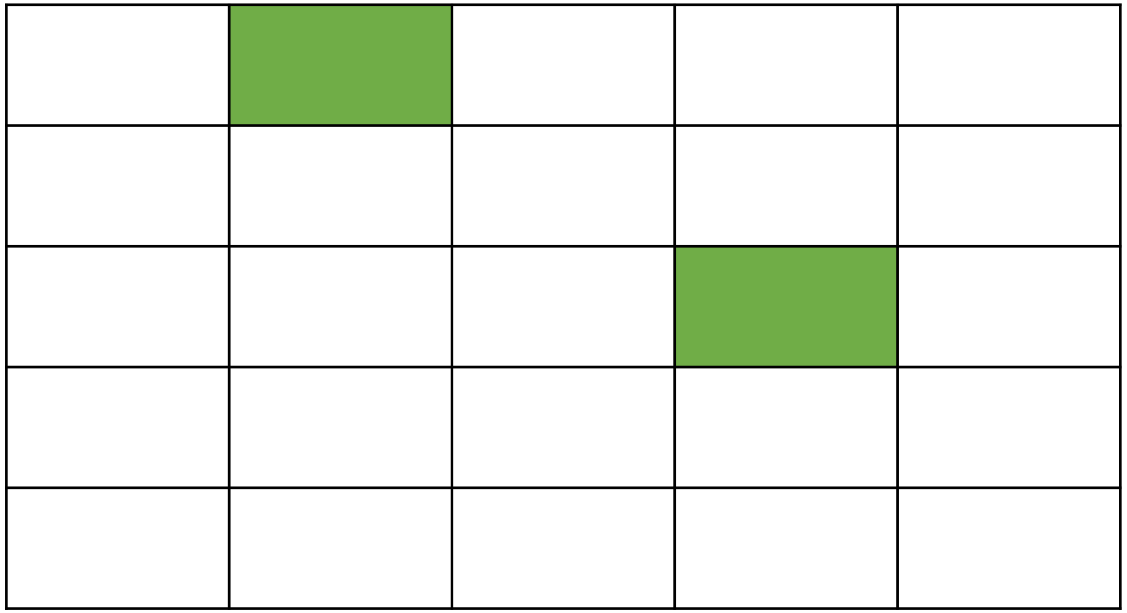


```c
#include <stdio.h>
#define MAX 10

int main()
{
  int source_row, source_col, dest_row, dest_col;
  int matrix[MAX][MAX];

  //scan
  scanf("%d %d %d %d", &source_row, &source_col, &dest_row, &dest_col);
```

Solution explanation

- First, only consider the section of the matrix that includes the minimum path (a rectangular section where the starting and ending point are the top-left most and bottom-right most corner)

- Example:
    - Starting point: 1, 2
    - Ending point: 3, 4



Solution explanation

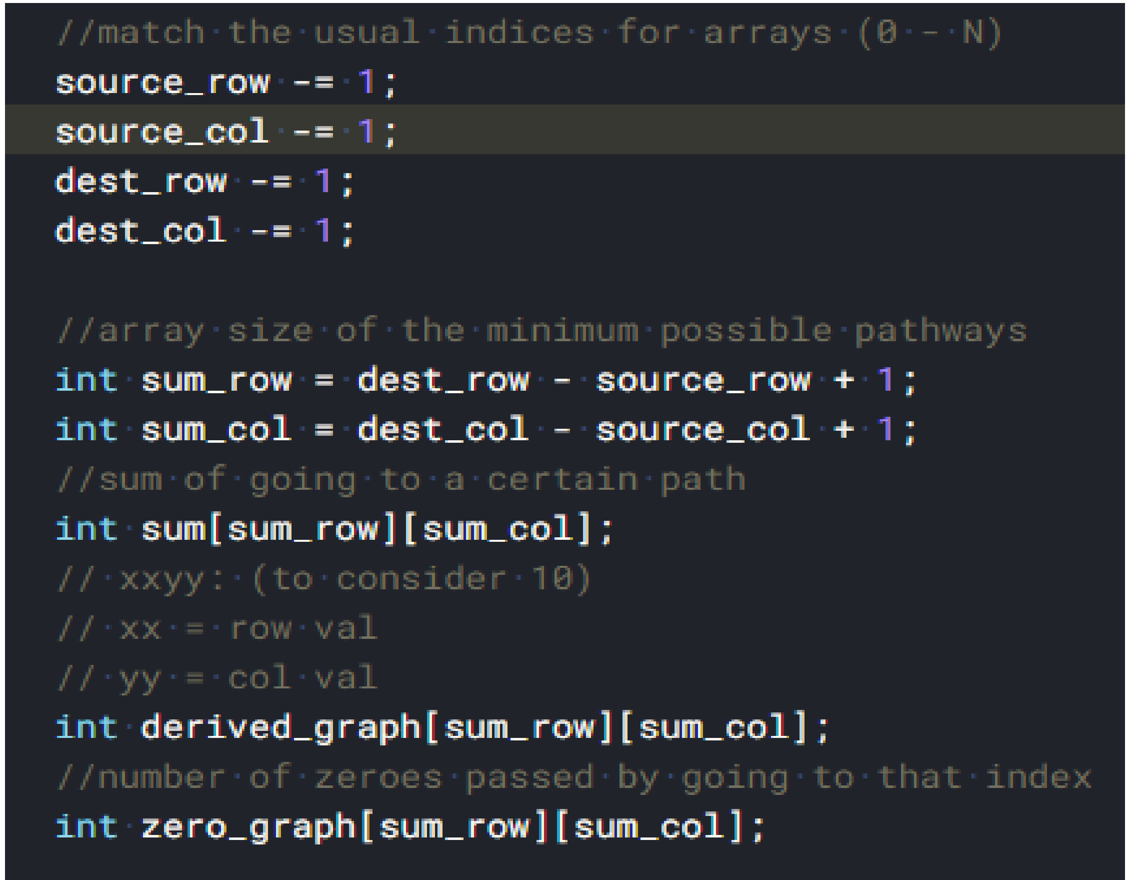
- First, only consider the section of the matrix that includes the minimum path (a rectangular section where the starting and ending point are the top-left most and bottom-right most corner)

- Example:
    - Starting point: 1, 2
    - Ending point: 3, 4

	🟩	🟩	🟩	
	🟩	🟩	🟩	
	🟩	🟩	🟩	

Solution explanation

- Subtract 1, to match the usual array index system
- sum_row: number of rows in the minimum path matrix
- sum_col: number of columns in the minimum path matrix
- Sum matrix: minimum sum matrix
- Derived_graph: data of previous Index
- Zero_graph: number of zeroes until a certain index



```
//match the usual indices for arrays (0 - N)
source_row -= 1;
source_col -= 1;
dest_row -= 1;
dest_col -= 1;

//array size of the minimum possible pathways
int sum_row = dest_row - source_row + 1;
int sum_col = dest_col - source_col + 1;
//sum of going to a certain path
int sum[sum_row][sum_col];
// xxyy: (to consider 10)
// xx = row val
// yy = col val
int derived_graph[sum_row][sum_col];
//number of zeroes passed by going to that index
int zero_graph[sum_row][sum_col];
```

Solution explanation

Sum Matrix

- Contains the minimum sum at a certain index

1	3	-2
5	0	4
-5	12	5

Original Matrix

1	4	2
6	4	6
1	13	9

Sum Matrix

Solution explanation

Derived Matrix

- Contains the index of the previous path (should be the minimum path)
- 4 digits (xxyy): xx being column and yy being row

Solution explanation

Derived Matrix

1	3	-2
5	0	4
-5	12	5

Original Matrix

0000	0000	0001
0000	0001	0002
0100	0200	0102

Derived Matrix

Solution explanation

Zero Matrix

- Contains the number of zeroes passed until the current index

Solution explanation

Zero Matrix

1	3	-2
-2	0	4
5	3	5

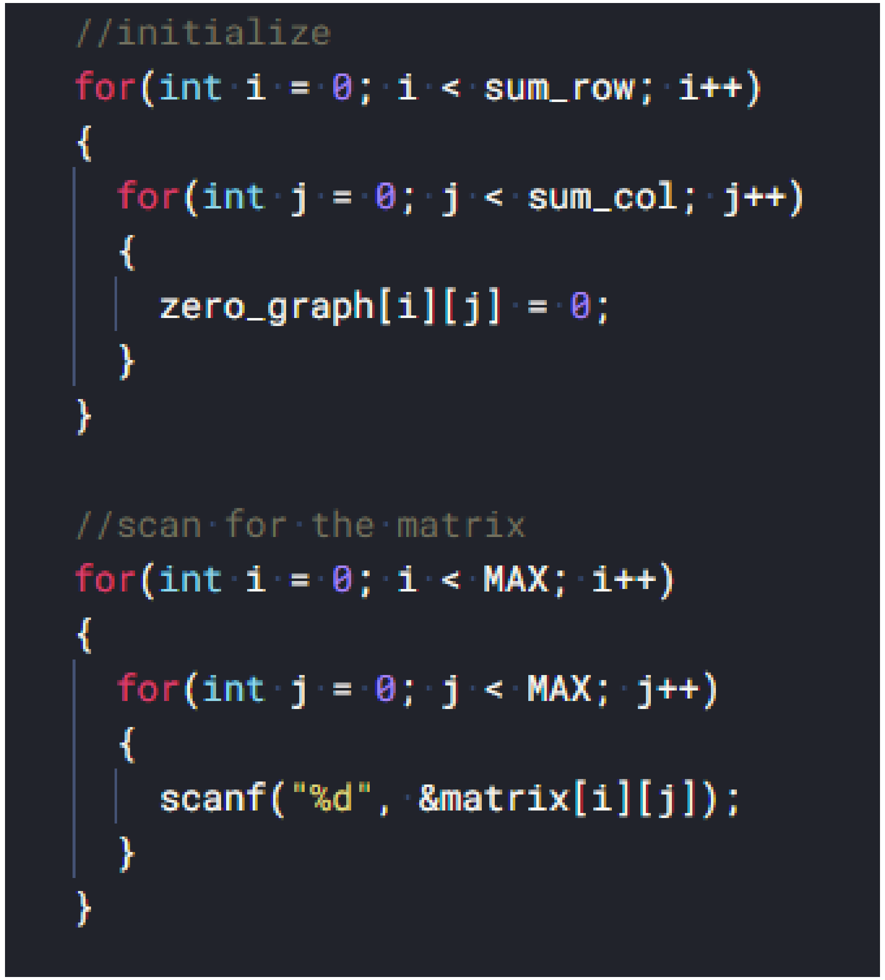
Original Matrix

0	0	0
0	1	0
0	1	1

Zero Matrix

Solution explanation

- Initialize the zero_graph
- Scan for the 10-by-10 matrix

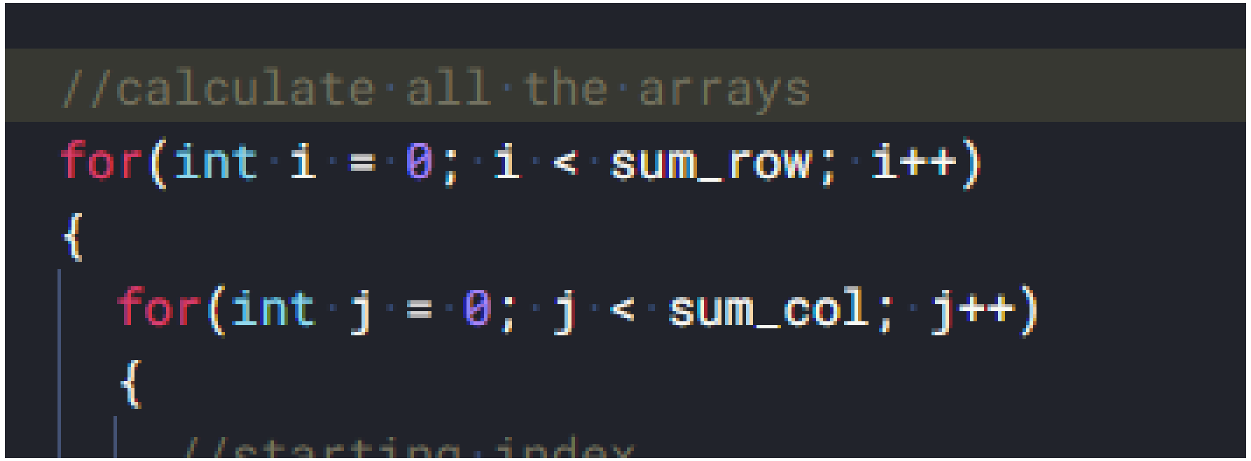


```
//initialize
for(int i = 0; i < sum_row; i++)
{
    for(int j = 0; j < sum_col; j++)
    {
        zero_graph[i][j] = 0;
    }
}

//scan for the matrix
for(int i = 0; i < MAX; i++)
{
    for(int j = 0; j < MAX; j++)
    {
        scanf("%d", &matrix[i][j]);
    }
}
```

Solution explanation

- From the remaining rectangular matrix, slowly just add from the top-left corner, going to the bottom right corner using a nested for loop
- There are four cases:

1. Starting index
2. First row values
3. First column values
4. Remaining values



```
//calculate all the arrays
for(int i = 0; i < sum_row; i++)
{
    for(int j = 0; j < sum_col; j++)
    {
        //starting index
```

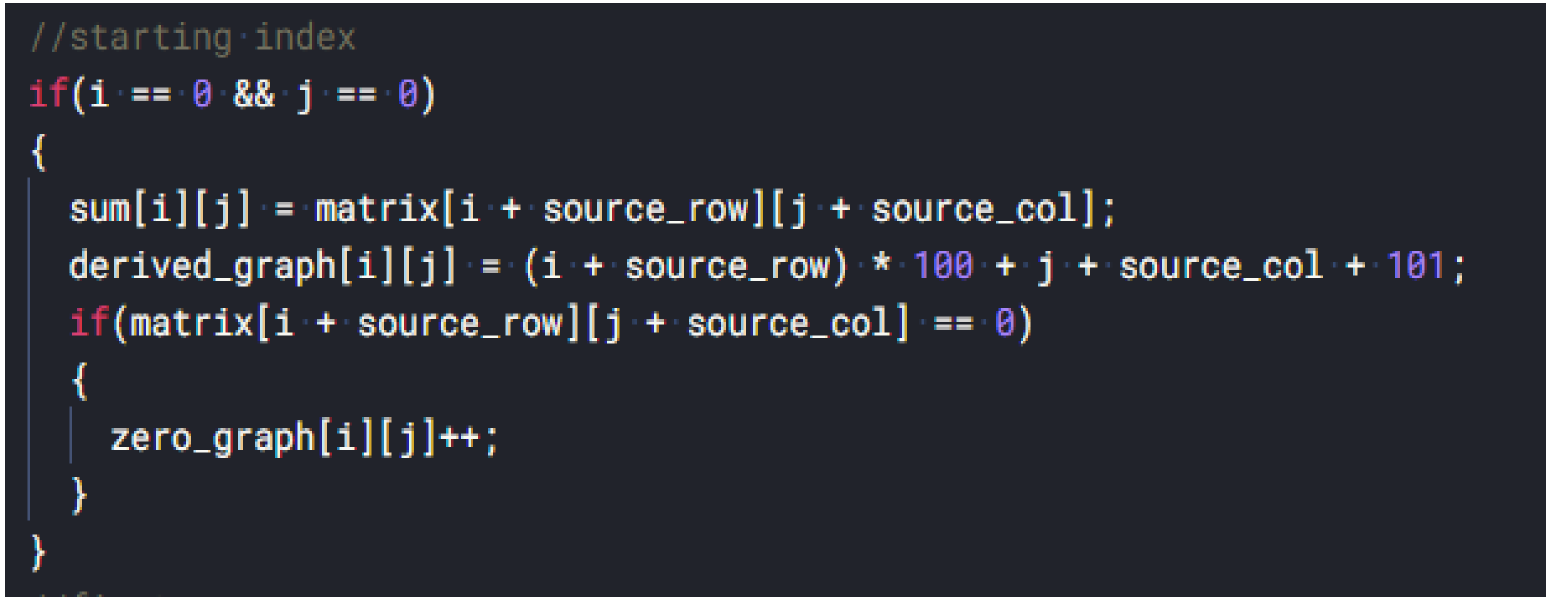
Solution explanation

Case 1: Starting index

- Sum: itself

- Derived: current index + 101 (add 1 to column and row)

- Zero: number of zeroes until now

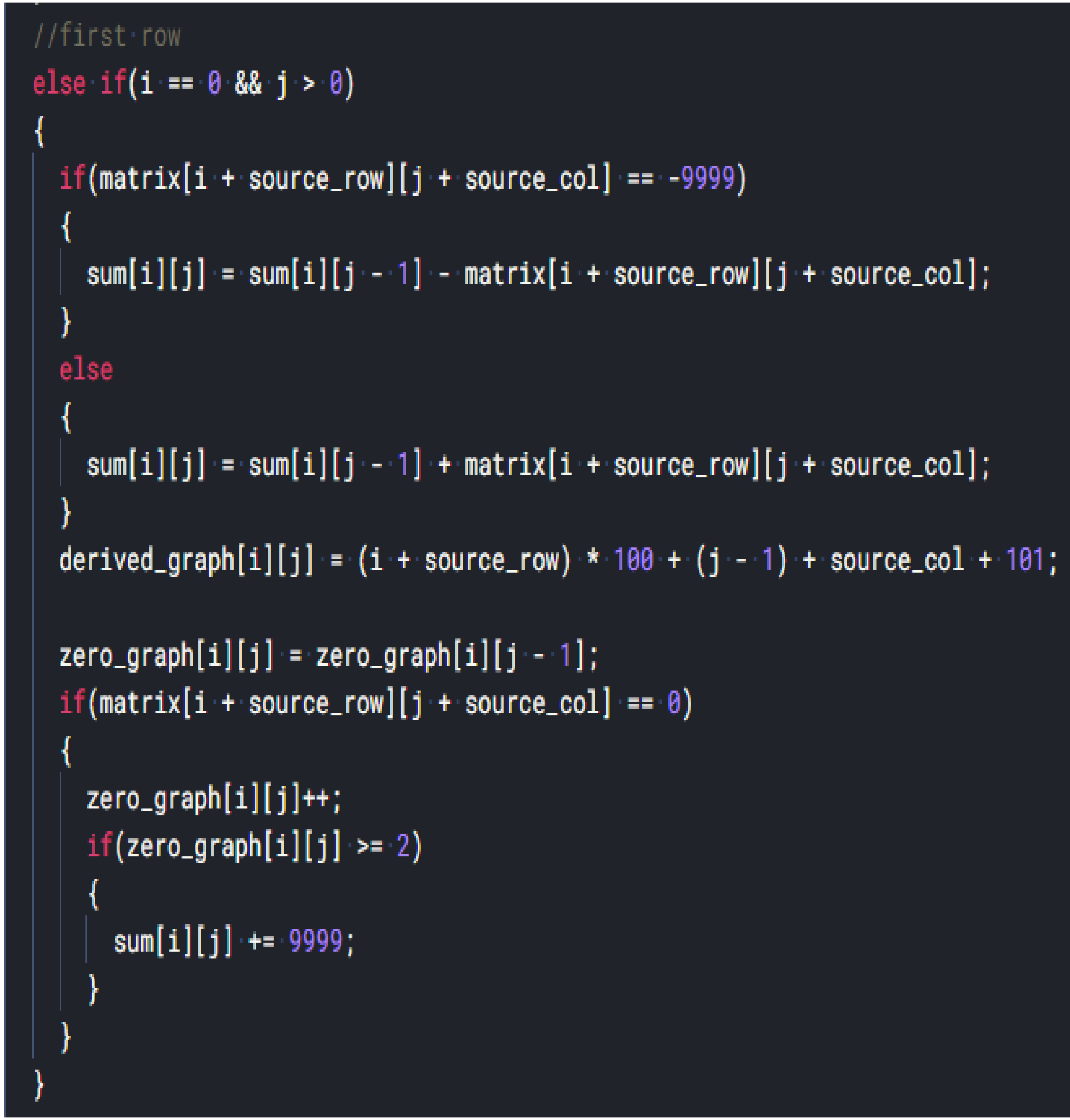


```
//starting index
if(i == 0 && j == 0)
{
  sum[i][j] = matrix[i + source_row][j + source_col];
  derived_graph[i][j] = (i + source_row) * 100 + j + source_col + 101;
  if(matrix[i + source_row][j + source_col] == 0)
  {
    zero_graph[i][j]++;
  }
}
```

Solution explanation

Case 2: First row values

- Sum: previous row's sum + current value(if -9999, add 9999)
- Derived: previous row index + 101 (quick conversion)
- Zero: number of zeroes until now (num_of_zero >= 2, add 9999, eliminate as candidate)

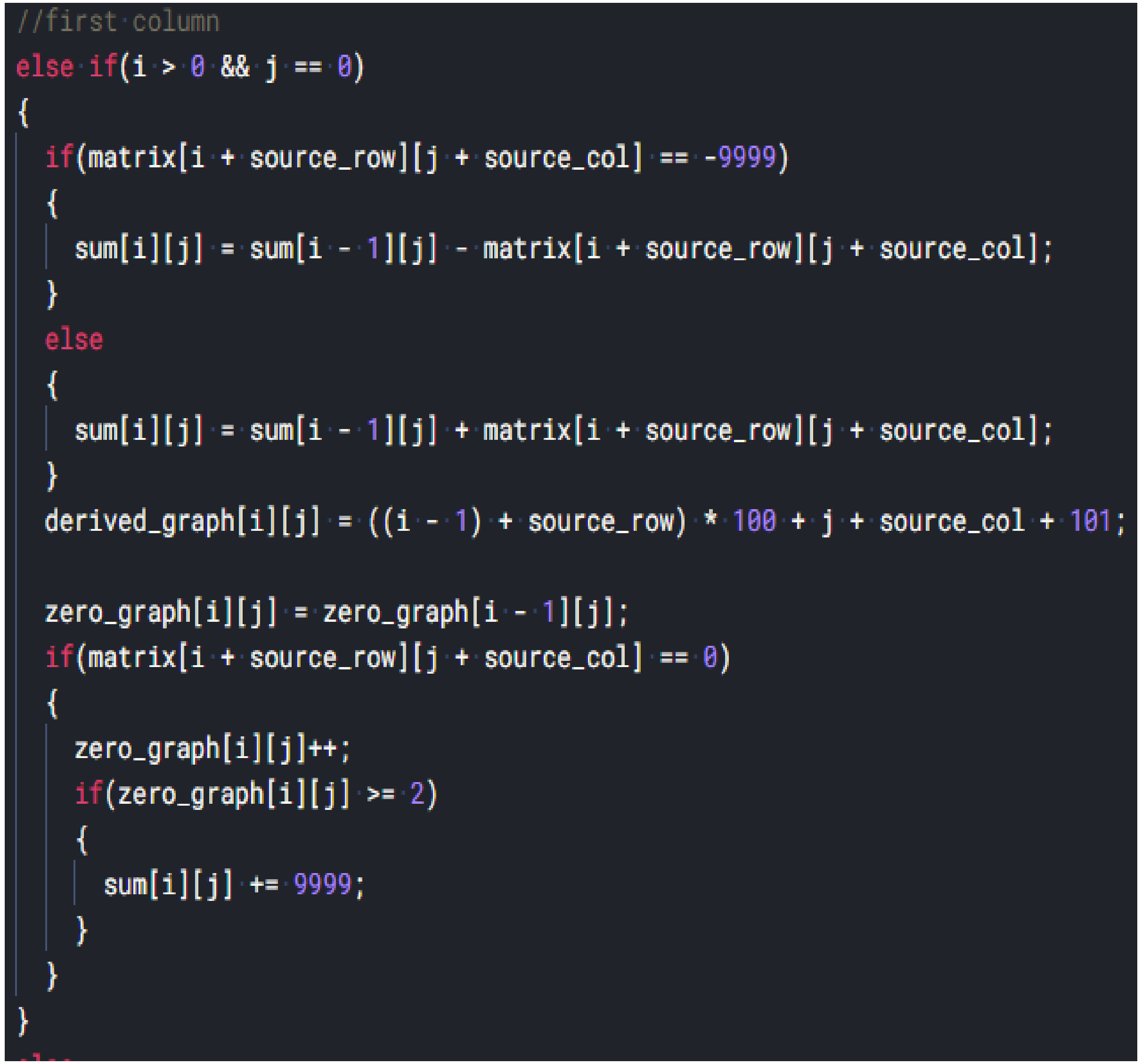


```
//first row
else if(i == 0 && j > 0)
{
    if(matrix[i + source_row][j + source_col] == -9999)
    {
        sum[i][j] = sum[i][j - 1] - matrix[i + source_row][j + source_col];
    }
    else
    {
        sum[i][j] = sum[i][j - 1] + matrix[i + source_row][j + source_col];
    }
    derived_graph[i][j] = (i + source_row) * 100 + (j - 1) + source_col + 101;

    zero_graph[i][j] = zero_graph[i][j - 1];
    if(matrix[i + source_row][j + source_col] == 0)
    {
        zero_graph[i][j]++;
        if(zero_graph[i][j] >= 2)
        {
            sum[i][j] += 9999;
        }
    }
}
```

Solution explanation

Case 3: First column values

- Sum: previous col's sum + current value(if -9999, add 9999)
- Derived: previous col index + 101 (quick conversion)
- Zero: number of zeroes until now (num_of_zero >= 2, add 9999, eliminate as candidate)



```
//first column
else if(i > 0 && j == 0)
{
  if(matrix[i + source_row][j + source_col] == -9999)
  {
    sum[i][j] = sum[i - 1][j] - matrix[i + source_row][j + source_col];
  }
  else
  {
    sum[i][j] = sum[i - 1][j] + matrix[i + source_row][j + source_col];
  }
  derived_graph[i][j] = ((i - 1) + source_row) * 100 + j + source_col + 101;

  zero_graph[i][j] = zero_graph[i - 1][j];
  if(matrix[i + source_row][j + source_col] == 0)
  {
    zero_graph[i][j]++;
    if(zero_graph[i][j] >= 2)
    {
      sum[i][j] += 9999;
    }
  }
}
```

Solution explanation

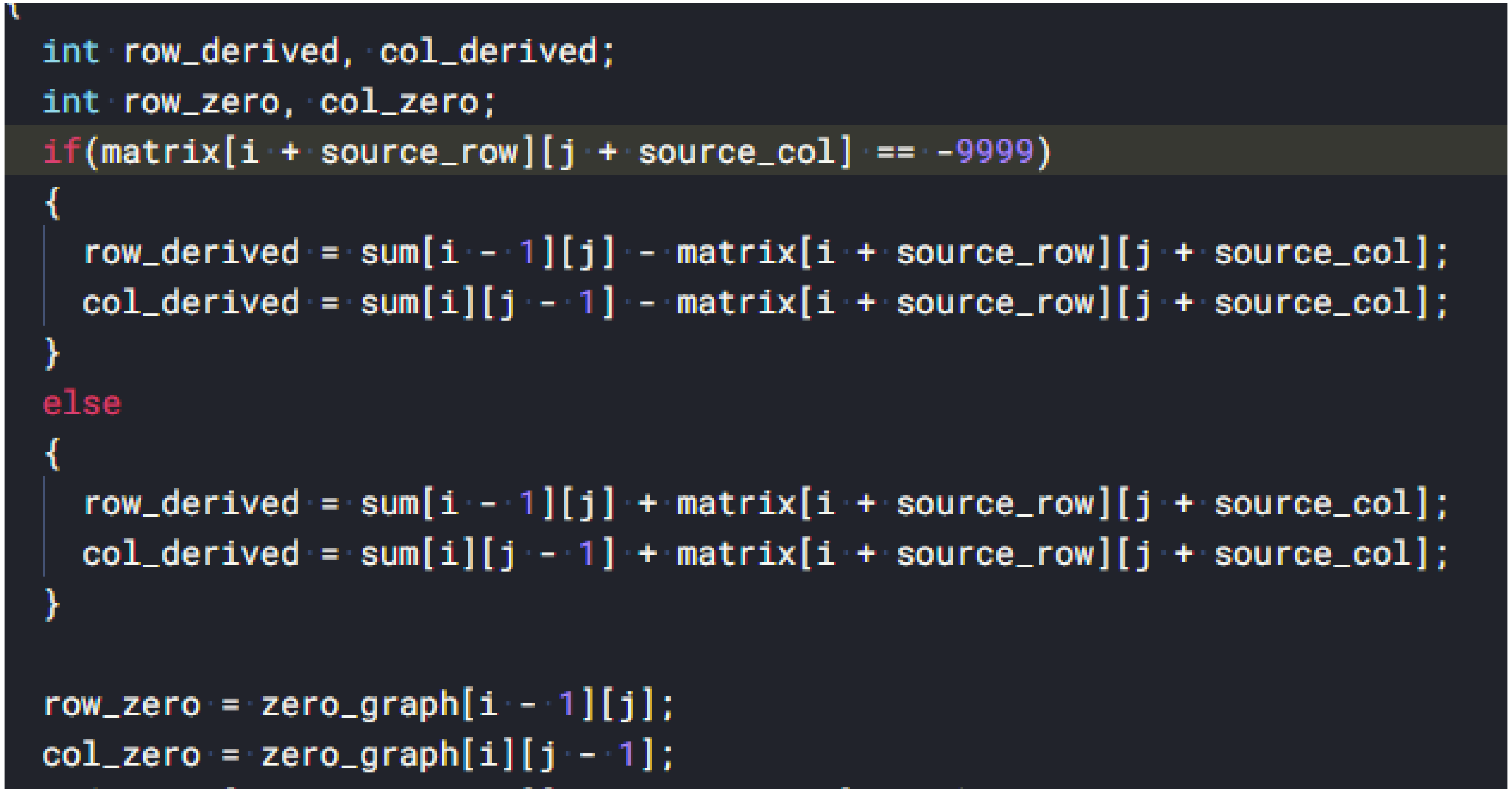
Case 4: Remaining values

- Can receive values from either previous row or previous column

Solution explanation

Case 4: Remaining values

- Row_derived
- row_zero
- Col_derived
- col_zero



```
int row_derived, col_derived;
int row_zero, col_zero;
if(matrix[i + source_row][j + source_col] == -9999)
{
  row_derived = sum[i - 1][j] - matrix[i + source_row][j + source_col];
  col_derived = sum[i][j - 1] - matrix[i + source_row][j + source_col];
}
else
{
  row_derived = sum[i - 1][j] + matrix[i + source_row][j + source_col];
  col_derived = sum[i][j - 1] + matrix[i + source_row][j + source_col];
}

row_zero = zero_graph[i - 1][j];
col_zero = zero_graph[i][j - 1];
```

Solution explanation

Case 4: Remaining values

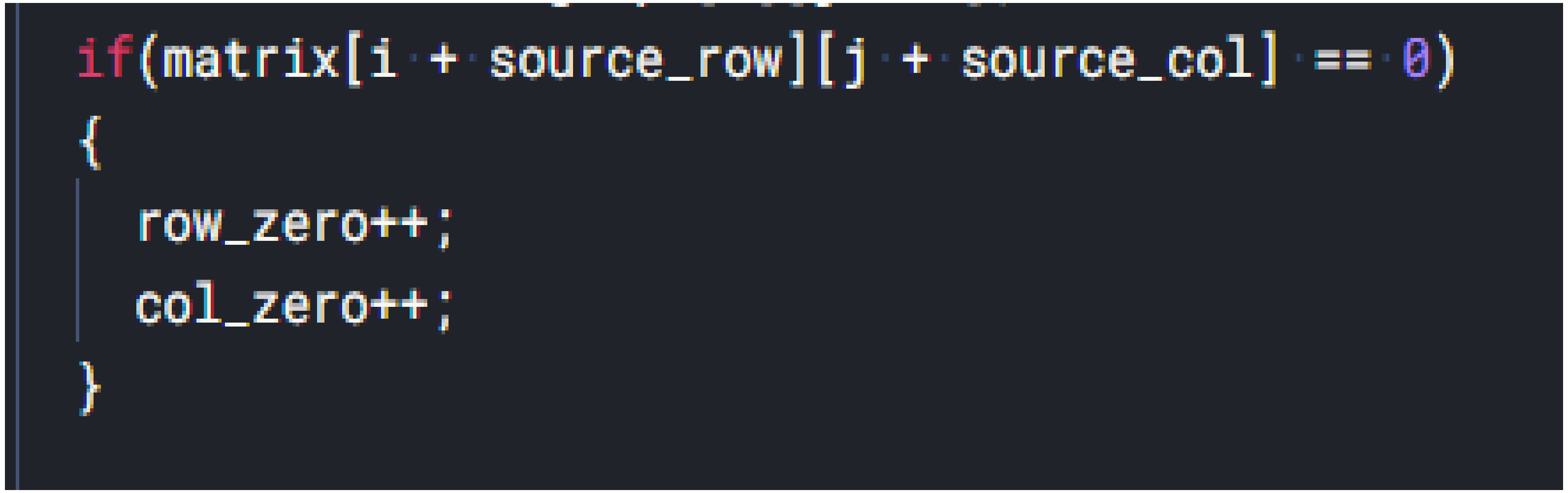
– Row_derived, row_zero: sum and number of zeroes from the previous row

- Col_derived, col_zero: sum and number of zeroes from the previous column
- If current value is -9999, add 9999

Solution explanation

Case 4: Remaining values

- If current value is zero, add a zero to both row_zero and col_zero



```
if(matrix[i + source_row][j + source_col] == 0)
{
  row_zero++;
  col_zero++;
}
```

Solution explanation

Case 4: Remaining values

- Now consider the number of zeroes:

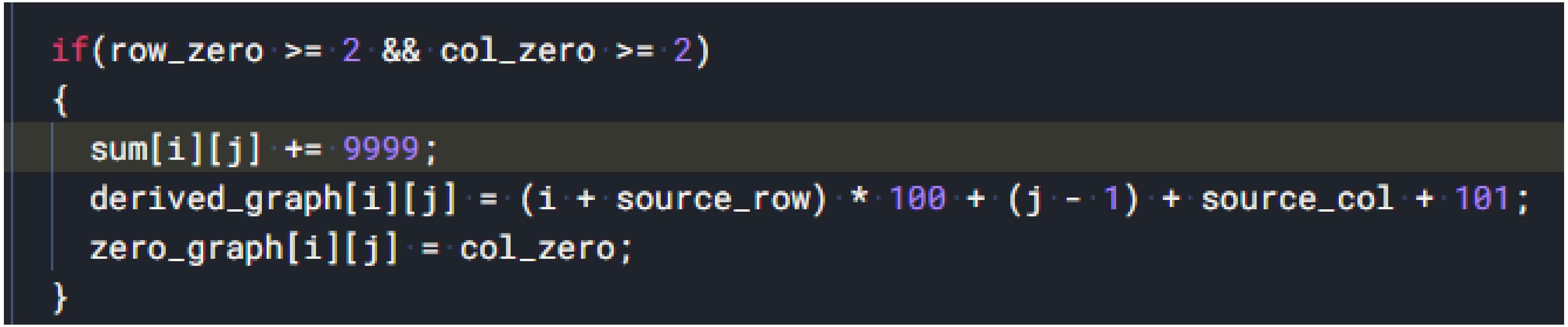
1. Both previous row and column indices have more than two zeroes
2. Only previous row has more than two zeroes
3. Only previous column has more than two zeroes
4. Both have less than two zeroes

Solution explanation

Case 4: Remaining values

Zero Case 1: both row and col have more than 1 zeroes

- Just add 9999
- Eliminates current as a candidate
- Just add previous column as values



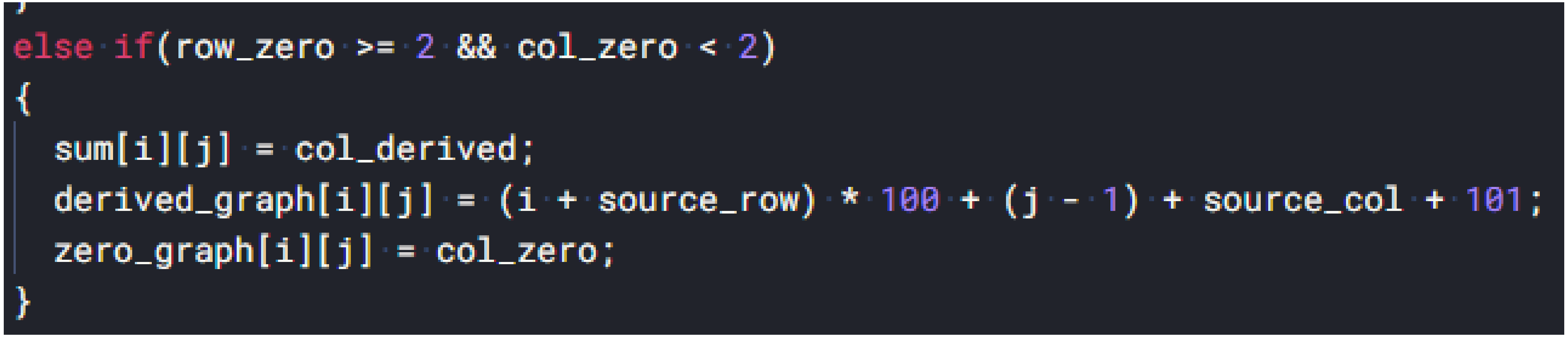
```
if(row_zero >= 2 && col_zero >= 2)
{
  sum[i][j] += 9999;
  derived_graph[i][j] = (i + source_row) * 100 + (j - 1) + source_col + 101;
  zero_graph[i][j] = col_zero;
}
```

Solution explanation

Case 4: Remaining values

Zero Case 2: col has less than 2 zeroes

- Use col_derived and col_zero
- Derive from previous column



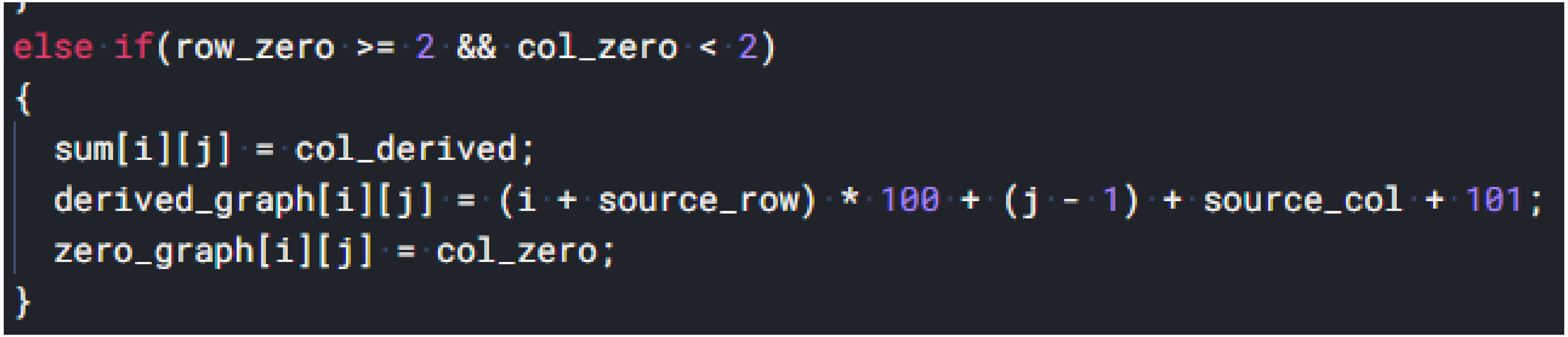
```
}
else if(row_zero >= 2 && col_zero < 2)
{
  sum[i][j] = col_derived;
  derived_graph[i][j] = (i + source_row) * 100 + (j - 1) + source_col + 101;
  zero_graph[i][j] = col_zero;
}
```

Solution explanation

Case 4: Remaining values

Zero Case 3: row has less than 2 zeroes

- Use row_derived and row_zero
- Derive from previous row



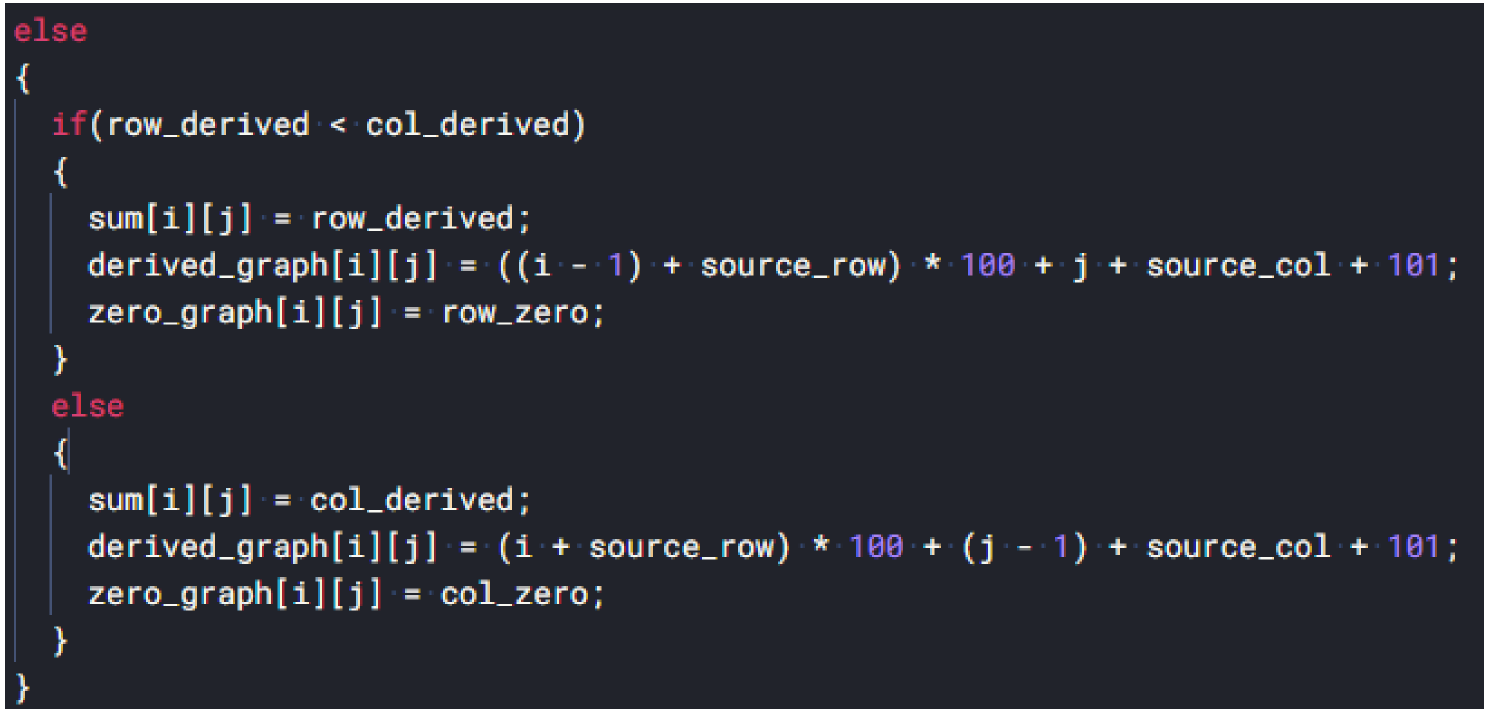
```
}
else if(row_zero >= 2 && col_zero < 2)
{
  sum[i][j] = col_derived;
  derived_graph[i][j] = (i + source_row) * 100 + (j - 1) + source_col + 101;
  zero_graph[i][j] = col_zero;
}
```

Solution explanation

Case 4: Remaining values

Zero Case 4: both less than zero

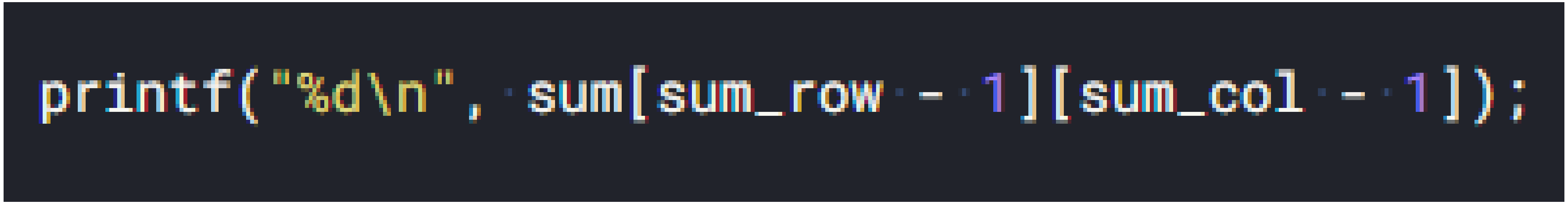
- Find the minimum value between both
- If the same, choose previous column



```
else
{
  if(row_derived < col_derived)
  {
    sum[i][j] = row_derived;
    derived_graph[i][j] = ((i - 1) + source_row) * 100 + j + source_col + 101;
    zero_graph[i][j] = row_zero;
  }
  else
  {
    sum[i][j] = col_derived;
    derived_graph[i][j] = (i + source_row) * 100 + (j - 1) + source_col + 101;
    zero_graph[i][j] = col_zero;
  }
}
```

Solution explanation

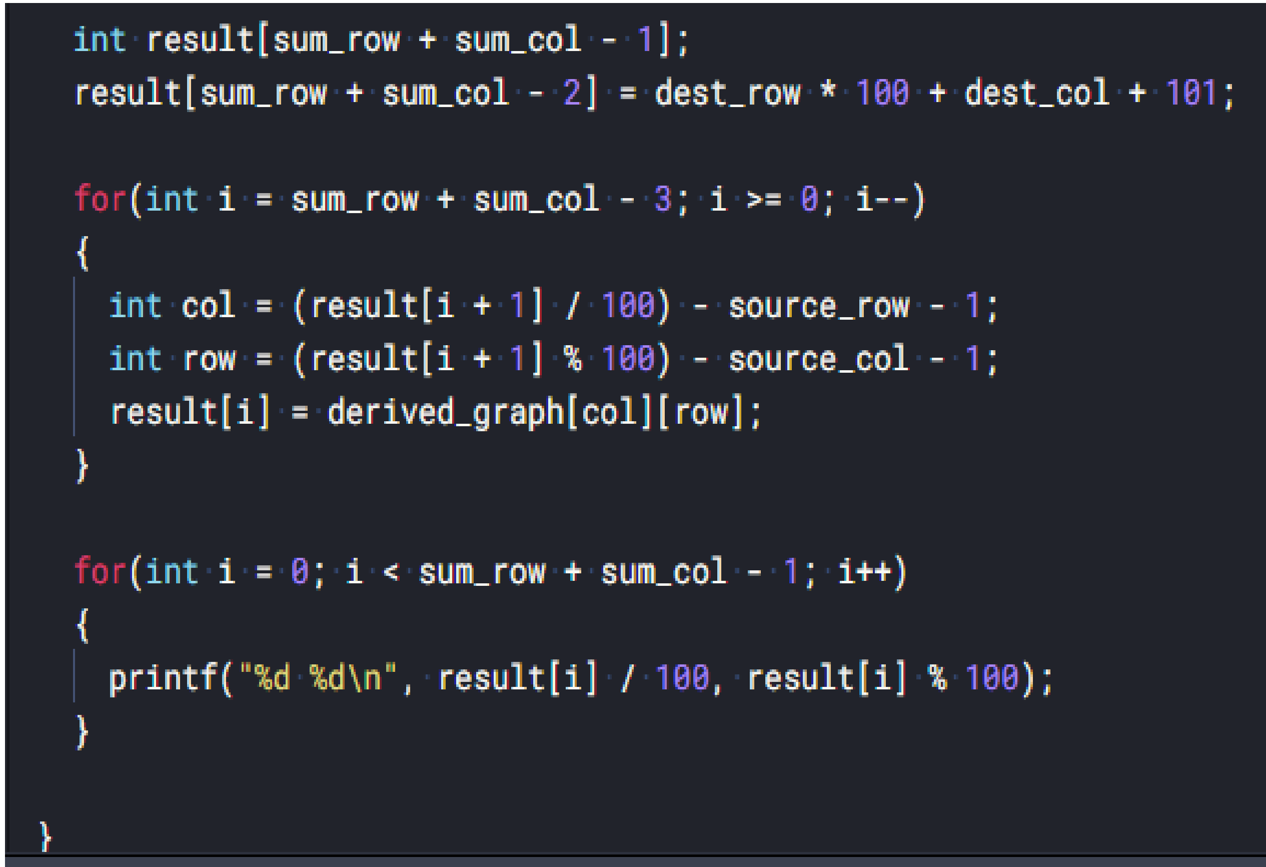
Print resulting sum which is in the destination index (bottom-right corner)



```
printf("%d\n", sum[sum_row - 1][sum_col - 1]);
```

Solution explanation

- result: matrix containing minimum path indices
- Result matrix size: minimum path row + minimum path col – 1
- Result[last index] = destination



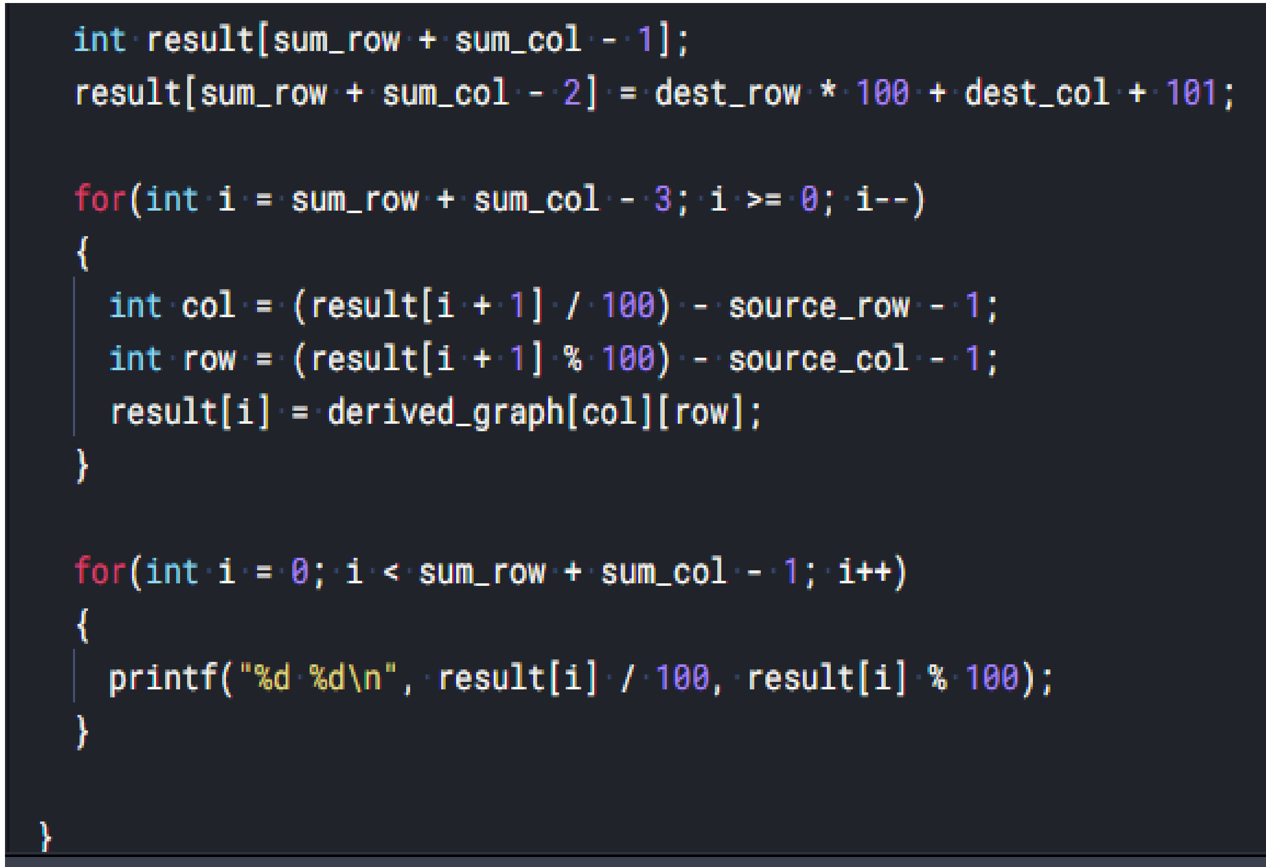
```
  int result[sum_row + sum_col - 1];
  result[sum_row + sum_col - 2] = dest_row * 100 + dest_col + 101;

  for(int i = sum_row + sum_col - 3; i >= 0; i--)
  {
    int col = (result[i + 1] / 100) - source_row - 1;
    int row = (result[i + 1] % 100) - source_col - 1;
    result[i] = derived_graph[col][row];
  }

  for(int i = 0; i < sum_row + sum_col - 1; i++)
  {
    printf("%d %d\n", result[i] / 100, result[i] % 100);
  }

}
```

Solution explanation

- Starting with the destination, retrieve the previous indices of the minimum path using the derived_graph matrix and store from inversely into result

(n – 1 to 0)

- Print the results



```
    int result[sum_row + sum_col - 1];
    result[sum_row + sum_col - 2] = dest_row * 100 + dest_col + 101;

    for(int i = sum_row + sum_col - 3; i >= 0; i--)
    {
        int col = (result[i + 1] / 100) - source_row - 1;
        int row = (result[i + 1] % 100) - source_col - 1;
        result[i] = derived_graph[col][row];
    }

    for(int i = 0; i < sum_row + sum_col - 1; i++)
    {
        printf("%d %d\n", result[i] / 100, result[i] % 100);
    }

}
```

Solution analysis

Pros:

- Consistent for every case
- Only considers minimum paths from the beginning

Cons:

- Not efficient in bigger scaled matrices
- Can use a lot of memory compared to other solutions

Thank you!