```cpp
// Copyright © 2020 AlgoExpert, LLC. All rights reserved.

using namespace std;

struct InfoMatrixItem {
  int numZeroesBelow;
  int numZeroesRight;
};

bool hasSquareOfZeroes(
  vector<vector<InfoMatrixItem>>& infoMatrix,
  int r1,
  int c1,
  int r2,
  int c2,
  unordered_map<string, bool> &cache
);
bool isSquareOfZeroes(
  vector<vector<InfoMatrixItem>>& infoMatrix,
  int r1,
  int c1,
  int r2,
  int c2
);
vector<vector<InfoMatrixItem>> preComputedNumOfZeroes(vector<vector<int>> matrix);

// O(n^3) time | O(n^3) space - where n is the height and width of the matrix
bool squareOfZeroes(vector<vector<int>> matrix) {
  vector<vector<InfoMatrixItem>> infoMatrix = preComputedNumOfZeroes(matrix);
  int lastIdx = matrix.size() - 1;
  unordered_map<string, bool> cache;
  return hasSquareOfZeroes(infoMatrix, 0, 0, lastIdx, lastIdx, cache);
}

// r1 is the top row, c1 is the left column
// r2 is the bottom row, c2 is the right column
bool hasSquareOfZeroes(
  vector<vector<InfoMatrixItem>>& infoMatrix,
  int r1,
  int c1,
  int r2,
  int c2,
  unordered_map<string, bool> &cache
) {
  if (r1 >= r2 || c1 >= c2) return false;

  string key = to_string(r1) + '-' + to_string(c1) + '-' + to_string(r2) + '-' + to_string(c2);
  if (cache.find(key) != cache.end()) return cache[key];

  cache[key] =
    isSquareOfZeroes(infoMatrix, r1, c1, r2, c2) ||
    hasSquareOfZeroes(infoMatrix, r1 + 1, c1 + 1, r2 - 1, c2 - 1, cache) ||
    hasSquareOfZeroes(infoMatrix, r1, c1 + 1, r2 - 1, c2, cache) ||
    hasSquareOfZeroes(infoMatrix, r1 + 1, c1, r2, c2 - 1, cache) ||
    hasSquareOfZeroes(infoMatrix, r1 + 1, c1 + 1, r2, c2, cache) ||
    hasSquareOfZeroes(infoMatrix, r1, c1, r2 - 1, c2 - 1, cache);

  return cache[key];
}

// r1 is the top row, c1 is the left column
// r2 is the bottom row, c2 is the right column
bool isSquareOfZeroes(
  vector<vector<InfoMatrixItem>>& infoMatrix,
  int r1,
  int c1,
  int r2,
  int c2
) {
  int squareLength = c2 - c1 + 1;
  bool hasTopBorder = infoMatrix[r1][c1].numZeroesRight >= squareLength;
  bool hasLeftBorder = infoMatrix[r1][c1].numZeroesBelow >= squareLength;
  bool hasBottomBorder = infoMatrix[r2][c1].numZeroesRight >= squareLength;
  bool hasRightBorder = infoMatrix[r1][c2].numZeroesBelow >= squareLength;
  return hasTopBorder && hasLeftBorder && hasBottomBorder && hasRightBorder;
}

vector<vector<InfoMatrixItem>> preComputedNumOfZeroes(vector<vector<int>> matrix) {
  vector<vector<InfoMatrixItem>> infoMatrix;
  for (int i = 0; i < matrix.size(); i++) {
    vector<InfoMatrixItem> inner;
    for (int j = 0; j < matrix[i].size(); j++) {
      int numZeroes = matrix[i][j] == 0 ? 1 : 0;
```

```
84          inner.push_back(InfoMatrixItem {numZeroes, numZeroes});
85        }
86        infoMatrix.push_back(inner);
87      }
88
89      int lastIdx = matrix.size() - 1;
90      for (int row = lastIdx; row >= 0; row--) {
91        for (int col = lastIdx; col >= 0; col--) {
92          if (matrix[row][col] == 1) continue;
93          if (row < lastIdx) {
94            infoMatrix[row][col].numZeroesBelow += infoMatrix[row + 1][col].numZeroesBelow;
95          }
96          if (col < lastIdx) {
97            infoMatrix[row][col].numZeroesRight += infoMatrix[row][col + 1].numZeroesRight;
98          }
99        }
100     }
101
102     return infoMatrix;
103   }
```