Maximum size square sub-matrix with all 1s

Given a binary matrix, find out the maximum size square sub-matrix with all 1s.

For example, consider the below binary matrix.

```
   0  1  1  0  1
   1  1  0  1  0
   0  1  1  1  0
   1  1  1  1  0
   1  1  1  1  1
   0  0  0  0  0
```

The maximum square sub-matrix with all set bits is

```
    1  1  1
    1  1  1
    1  1  1
```

Algorithm:
Let the given binary matrix be M[R][C]. The idea of the algorithm is to construct an auxiliary size matrix S[][] in which each entry S[i][j] represents size of the square sub-matrix with all 1s including M[i][j] where M[i][j] is the rightmost and bottommost entry in sub-matrix.

```
1) Construct a sum matrix S[R][C] for the given M[R][C].
     a) Copy first row and first columns as it is from M[][] to S[][]
     b) For other entries, use following expressions to construct S[][]
         If M[i][j] is 1 then
            S[i][j] = min(S[i][j-1], S[i-1][j], S[i-1][j-1]) + 1
         Else /*If M[i][j] is 0*/
            S[i][j] = 0
2) Find the maximum entry in S[R][C]
3) Using the value and coordinates of maximum entry in S[i], print
   sub-matrix of M[][]
```

For the given M[R][C] in above example, constructed S[R][C] would be:

```
   0  1  1  0  1
   1  1  0  1  0
   0  1  1  1  0
   1  1  2  2  0
   1  2  2  3  1
   0  0  0  0  0
```

The value of maximum entry in above matrix is 3 and coordinates of the entry are (4, 3). Using the maximum value and its coordinates, we can find out the required sub-matrix.

```
#include<stdio.h>
#define bool int
#define R 6
#define C 5

void printMaxSubSquare(bool M[R][C])
{
```

```
  int i,j;
  int S[R][C];
  int max_of_s, max_i, max_j;

  /* Set first column of S[][]*/
  for(i = 0; i < R; i++)
     S[i][0] = M[i][0];

  /* Set first row of S[][]*/
  for(j = 0; j < C; j++)
     S[0][j] = M[0][j];

  /* Construct other entries of S[][]*/
  for(i = 1; i < R; i++)
  {
    for(j = 1; j < C; j++)
    {
      if(M[i][j] == 1)
        S[i][j] = min(S[i][j-1], S[i-1][j], S[i-1][j-1]) + 1;
      else
        S[i][j] = 0;
    }
  }

  /* Find the maximum entry, and indexes of maximum entry
     in S[][] */
  max_of_s = S[0][0]; max_i = 0; max_j = 0;
  for(i = 0; i < R; i++)
  {
    for(j = 0; j < C; j++)
    {
      if(max_of_s < S[i][j])
      {
         max_of_s = S[i][j];
         max_i = i;
         max_j = j;
      }
    }
  }

  printf("\n Maximum size sub-matrix is: \n");
  for(i = max_i; i > max_i - max_of_s; i--)
  {
    for(j = max_j; j > max_j - max_of_s; j--)
    {
      printf("%d ", M[i][j]);
    }
    printf("\n");
  }
}

/* UTILITY FUNCTIONS */
/* Function to get minimum of three values */
int min(int a, int b, int c)
{
  int m = a;
  if (m > b)
    m = b;
  if (m > c)
    m = c;
  return m;
}

/* Driver function to test above functions */
int main()
{
  bool M[R][C] =  {{0, 1, 1, 0, 1},
                   {1, 1, 0, 1, 0},
                   {0, 1, 1, 1, 0},
                   {1, 1, 1, 1, 0},
                   {1, 1, 1, 1, 1},
                   {0, 0, 0, 0, 0}};

  printMaxSubSquare(M);
```

```
    getchar();
}
```

Time Complexity: O(m*n) where m is number of rows and n is number of columns in the given matrix.
Auxiliary Space: O(m*n) where m is number of rows and n is number of columns in the given matrix.
Algorithmic Paradigm: Dynamic Programming