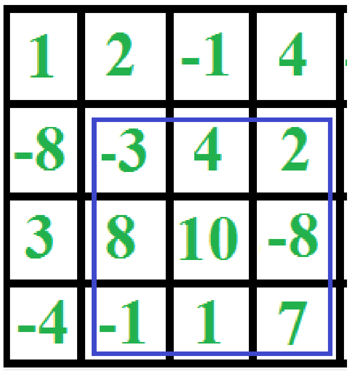
Print maximum sum square sub-matrix of given size

Given an N x N matrix, find a k x k submatrix where k <= N and k >= 1, such that sum of all the elements in submatrix is maximum. The input matrix can contain zero, positive and negative numbers.

For example consider below matrix, if k = 3, then output should print the sub-matrix enclosed in blue.



We strongly recommend you to minimize your browser and try this yourself first.

A Simple Solution is to consider all possible sub-squares of size k x k in our input matrix and find the one which has maximum sum. Time complexity of above solution is O(N^2k^2).

We can solve this problem in O(N^2) time. This problem is mainly an extension of this problem of printing all sums. The idea is to preprocess the given square matrix. In the preprocessing step, calculate sum of all vertical strips of size k x 1 in a temporary square matrix stripSum[][]. Once we have sum of all vertical strips, we can calculate sum of first sub-square in a row as sum of first k strips in that row, and for remaining sub-squares, we can calculate sum in O(1) time by removing the leftmost strip of previous subsquare and adding the rightmost strip of new square.

Below is C++ implementation of above idea.

```
// An efficient C++ program to find maximum sum
// sub-square matrix
#include <bits/stdc++.h>
using namespace std;

// Size of given matrix
#define N 5

// A O(n^2) function to the maximum sum sub-
// squares of size k x k in a given square
// matrix of size n x n
void printMaxSumSub(int mat[][N], int k)
{
    // k must be smaller than or equal to n
    if (k > N) return;

    // 1: PREPROCESSING
    // To store sums of all strips of size k x 1
    int stripSum[N][N];

    // Go column by column
    for (int j=0; j<N; j++)
    {
        // Calculate sum of first k x 1 rectangle
        // in this column
        int sum = 0;
        for (int i=0; i<k; i++)
            sum += mat[i][j];
        stripSum[0][j] = sum;

        // Calculate sum of remaining rectangles
        for (int i=1; i<N-k+1; i++)
        {
            sum += (mat[i+k-1][j] - mat[i-1][j]);
            stripSum[i][j] = sum;
        }
    }

    // max_sum stores maximum sum and its
    // position in matrix
    int max_sum = INT_MIN, *pos = NULL;

    // 2: CALCULATE SUM of Sub-Squares using stripSum[][]
    for (int i=0; i<N-k+1; i++)
    {
        // Calculate and print sum of first subsquare
        // in this row
        int sum = 0;
        for (int j = 0; j<k; j++)
            sum += stripSum[i][j];

        // Update max_sum and position of result
        if (sum > max_sum)
        {
            max_sum = sum;
            pos = &(mat[i][0]);
        }

        // Calculate sum of remaining squares in
        // current row by removing the leftmost
        // strip of previous sub-square and adding
        // a new strip
        for (int j=1; j<N-k+1; j++)
        {
            sum += (stripSum[i][j+k-1] - stripSum[i][j-1]);

            // Update max_sum and position of result
            if (sum > max_sum)
            {
                max_sum = sum;
                pos = &(mat[i][j]);
            }
        }
    }

    // Print the result matrix
    for (int i=0; i<k; i++)
    {
        for (int j=0; j<k; j++)
            cout << *(pos + i*N + j) << " ";
        cout << endl;
    }
}

// Driver program to test above function
int main()
{
    int mat[N][N] = {{1, 1, 1, 1, 1},
        {2, 2, 2, 2, 2},
        {3, 8, 6, 7, 3},
        {4, 4, 4, 4, 4},
        {5, 5, 5, 5, 5},
    };
    int k = 3;

    cout << "Maximum sum 3 x 3 matrix is\n";
    printMaxSumSub(mat, k);

    return 0;
}
```

Output:

```
Maximum sum 3 x 3 matrix is
8 6 7
4 4 4
5 5 5
```

Related Articles:

Given an n x n square matrix, find sum of all sub-squares of size k x k

Maximum sum rectangle in a 2D matrix