Torus Lot

codility
WE TEST CODERS

Another month has passed and it is time to show how the previous Codility challenge, code-named T (Tau), can be solved. You can still give it a try, but no certificate will be granted.

This problem is a variant of a problem posed by J. Bentley in his *Programming Pearls*. The original problem was to find, in a given matrix, a rectangular region with a maximal sum of elements. We will show how this original problem can be solved, and then we will extend the solution to the challenge problem. Expected time complexity of the solution is $O(M^3 + N^3)$, although a faster solution is known (see H. Tamaki and T. Tokuyama, 2000).

Let C be the given matrix of $M \times N$ integers. There are $O(N^2 \cdot M^2)$ rectangular fragments of the matrix. Moreover, a na ive way to calculate the elements in the rectangle requires $O(N \cdot M)$ time. To obtain the desired time complexity we have to reduce the number of rectangles considered, and compute the sum of elements in a rectangle more quickly.

First, let us focus on the latter problem. We can compute the sum of elements in a given rectangle in constant time, after $O(N \cdot M)$-time pre-processing. The idea is to compute another $(M + 1) \times (N + 1)$ matrix sum, such that:

$$sum[x][y] = \sum_{i=0}^{x-1} \sum_{j=0}^{y-1} C[i][j]$$

Then, the sum of elements in $C[i_1..i_2][j_1..j_2]$ equals:

$$sum[i_2 + 1][j_2 + 1] - sum[i_1][j_2 + 1] - sum[i_2 + 1][j_1] + sum[i_1][j_1]$$

For the time being, let us fix the indices $0 \leqslant i_1 \leqslant i_2 < M$. How can we find the maximal-sum rectangular submatrix $C[i_1..i_2][j_1..j_2]$? Think about a one-dimensional array $C'[j] = \sum_{i=i_1}^{i_2} C[i][j]$. The sum of elements in $C[i_1..i_2][j_1..j_2]$ is equal to the sum of elements in $C'[j_1..j_2]$. In this way, the problem reduces to another problem posed by Bentley: that of finding a maximal-sum contiguous fragment of a one-dimensional array. What is interesting is that this problem can be solved in $O(N)$ time. So, by solving this one-dimensional problem for all possible $O(M^2)$ pairs of indices i_1 and i_2 we obtain an $O(M^2 \cdot N)$ time solution of the two-dimensional problem posed by Bentley.

Now let us deal with the one-dimensional problem. How can we find the maximal-sum contiguous fragment $C'[j_1..j_2]$ of array C'? Let us define array sum' as follows: $sum'[j] = sum[i_2 + 1][j] - sum[i_1][j]$. The sum of elements in $C'[j_1..j_2]$ equals $sum'[j_2 + 1] - sum'[j_1]$. If we fix j_2, what then is the optimal value of j_1? Clearly, it is an index such that $0 \leqslant j_1 \leqslant j_2$ and $sum'[j_1]$ is minimal. So, to find the optimal values of j_1 and j_2, it is sufficient to scan all possible values $sum'[j_2]$, keeping track of the minimum value $sum'[j_1]$ found so far.

After solving the two-dimensional problem posed by Bentley, it is time to see how to extend the solution for the certificate. Each rectangular submatrix corresponds to not one but four rectangles on a torus. In the following figure they are numbered from 1 to 4.

4	3	4
2	1	2
4	3	4

The solution presented so far finds the maximal-sum rectangle of type 1. To find the maximal-sum rectangle of type 2 for given indices i_1 and i_2, it is enough to find the minimal-sum rectangle of type 1. To find the maximal-sum rectangles of type 3 or 4, one should use the following array sum' instead: $sum'[j] = sum[M][j] - sum[i_2 + 1][j] + sum[i_1][j]$.

The overall time complexity of the presented solution is $O(M^2 \cdot N)$. We can optimize it slightly, transposing the input matrix if $M > N$. This way, we obtain a solution running in $O(\min(M, N)^3)$ time.

Here is an implementation of the presented solution in Python. Arrays C' and sum' are not defined explicitly; equivalent expressions referring to the array sum are used instead.

1: Model solution — $O(\min(M, N)^3)$

```
1       def torus_lot(C):
2           M = len(C)
3           N = len(C[0])
4
5           if M > N:
6               # Transpose C
7               C1 = [([0] * M)] * N
8               for j in range(N):
9                   C1[j] = C1[j][:]
10                  for i in range(M):
11                      C1[j][i] = C[i][j]
12              C = C1
13              M = len(C)
14              N = len(C[0])
15
16          # sum[i][j] = profit of a rectangle [0..i-1]x[0..j-1]
17          sum = [([0] * (N+1))] * (M+1)
18          for i in range(1, M+1):
19              sum[i] = sum[0][:]
20              for j in range(1, N+1):
21                  sum[i][j] = sum[i-1][j] + sum[i][j-1] - sum[i-1][j
                        -1] + C[i-1][j-1]
22
23          Res = 0 # Maximum profit of a rectangle
24
25          for i1 in xrange(M):
26              for i2 in xrange(i1+1, M+1):
27                  Min1  = 0  # Minimum profit of a rectangle [i1..i2
                        -1]x[0..j-1]
28                  MinJ1 = 0  # Maximum such j
29                  MaxP1 = 0  # Maximum profit of a rectangle [i1..i2
                        -1]x[j'..j-1]
30
31                  Max1  = 0  # Maximum profit of a rectangle [i1..i2
                        -1]x[0..j-1]
32                  MaxJ1 = 0  # Minimum such j
```

```
33                  MinC1 = 0  # Minimum profit of a rectangle [i1..i2
                        -1]x[j'..j-1]
34
35                  Min2  = 0  # Minimum profit of a rectangle [0..i1-1,
                        i2..M-1]x[0..j-1]
36                  MinJ2 = 0  # Maximum such j
37                  MaxP2 = 0  # Maximum profit of a rectangle [0..i1-1,
                        i2..M-1]x[j'..j-1]
38
39                  Max2  = 0  # Maximum profit of a rectangle [0..i1-1,
                        i2..M-1]x[0..j-1]
40                  MaxJ2 = 0  # Minimum such j
41                  MinC2 = 0  # Minimum profit of a rectangle [0..i1-1,
                        i2..M-1]x[j'..j-1]
42
43                  for j in xrange(1,N+1):
44                      Profit1 = sum[i2][j] - sum[i1][j]
45                      if Profit1 <= Min1:
46                          Min1  = Profit1
47                          MinJ1 = j
48                      if (Profit1 - Min1 > MaxP1):
49                          MaxP1 = Profit1 - Min1
50
51                      if Profit1 > Max1:
52                          Max1  = Profit1
53                          MaxJ1 = j
54                      if (Profit1 - Max1 < MinC1):
55                          MinC1 = Profit1 - Max1
56
57                      Profit2 = sum[M][j] - Profit1
58                      if Profit2 <= Min2:
59                          Min2  = Profit2
60                          MinJ2 = j
61                      if (Profit2 - Min2 > MaxP2):
62                          MaxP2 = Profit2 - Min2
63
64                      if Profit2 > Max2:
65                          Max2  = Profit2
66                          MaxJ2 = j
67                      if (Profit2 - Max2 < MinC2):
68                          MinC2 = Profit2 - Max2
69
70                  Res = max(Res, MaxP1, MaxP2, Profit1 - MinC1,
                        Profit2 - MinC2)
71
72              return Res
```