

Solution — Even Matrices

1 Modeling

The modeling part of the exercise is straightforward. You are given a matrix of numbers

$$M = \begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,n} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,n} \end{pmatrix}$$

and you are required to calculate the number of quadruples (i_1, i_2, j_1, j_2) where $1 \leq i_1 \leq i_2 \leq n$ and $1 \leq j_1 \leq j_2 \leq n$ and for which the sum

$$\sum_{i=i_1}^{i_2} \sum_{j=j_1}^{j_2} x_{i,j}$$

is even.

2 Algorithm Design

By looking at the constraints we see that n is at most 200. On the other hand, what is the maximum number of quadruples (i_1, i_2, j_1, j_2) with the required properties? In the extreme case when all the bits of M are zeros then any quadruple is fine, which means that the answer can be of order $\Theta(n^4)$. However, if $n = 200$ then even just going through all the quadruples is infeasible (200^4 is more than a billion). As the bounds on n in the first and the second test set are smaller, this might be a strategy that brings us partial points and thus let us explore this idea further.

$O(n^6)$ solution (20 points). For a given quadruple (i_1, i_2, j_1, j_2) , how much time do we need to check that $\sum_{i=i_1}^{i_2} \sum_{j=j_1}^{j_2} x_{i,j}$ is even? Assuming we have the matrix stored in a data structure with random access, we can do the check in roughly $(i_2 - i_1)(j_2 - j_1)$ time. In the worst case this can take quadratic time, e.g. if $i_2 = j_2 = \Theta(n)$ and $i_1 = j_1 = 1$. Therefore, to iterate over all possible quadruples and check if the elements of a submatrix induced by a quadruple sum to an even number takes $O(n^6)$ time. Since the first test set guarantees that n is at most 10, this brute-force solution passes the first test set. The second test set puts a bound of $n \leq 50$, which implies that a faster than $\Theta(n^6)$ solution is needed.

$O(n^4)$ solution (70 points). Let us try to improve the $O(n^6)$ solution such that it passes the second test set as well. Since $n \leq 50$ we can at least iterate through all $\Theta(n^4)$ possible quadruples in time. Hence it suffices to improve the part of the algorithm that checks the parity of the sum of elements in a fixed submatrix. Drawing inspiration from the problem *Even Pairs* we try achieve this by using the technique of *precomputation* (see lecture 1 slides). Let us define $P_{i,j}$, for any $1 \leq i, j \leq n$ as follows:

$$P_{i,j} = \sum_{i'=1}^i \sum_{j'=1}^j x_{i',j'},$$

and we set $P_{i,0}$ and $P_{0,i}$ to be zero for any $i \in \{0, \dots, n\}$. It is not hard to see that

$$\sum_{i=1}^{i_2} \sum_{j=1}^{j_2} x_{i,j} = P_{i_2,j_2} - P_{i_2,j_1-1} - P_{i_1-1,j_2} + P_{i_1-1,j_1-1}. \quad (1)$$

Thus, computing the parity of the sum of elements of a fixed submatrix can be done in $O(1)$ time, once we know all the values $P_{i,j}$. Similarly as in (1), we can compute a particular value $P_{i,j}$ in constant time if we have the values $P_{i',j'}$, for $i' \leq i$ and $j' \leq j$. Namely,

$$P_{i,j} = x_{i,j} + P_{i-1,j} + P_{i,j-1} - P_{i-1,j-1}. \quad (2)$$

This shows that we do the precomputation in quadratic time and thus the whole algorithm can be implemented in $O(n^4)$.

$O(n^3)$ solution (100 points). By generalizing the approach from the problem *Even Pairs* we were able to design an algorithm which works in quartic time and achieves seventy points. However, to come up with a solution faster than $\Theta(n^4)$, and thus achieve the full number of points, some creativity and new ideas are required. The key new ingredient is the idea of *dimensionality reduction*. We reduce our 2-dimensional problem to the 1-dimensional problem of calculating the number of pairs (i, j) in a given array y_1, \dots, y_n such that $\sum_{k=i}^j y_k$ is even. Observe that the 1-dimensional problem is equivalent to *Even Pairs*, which we know how to solve in $O(n)$ time.

Let us define a sequence of variables Y_{i_1,i_2} , for $1 \leq i_1 \leq i_2 \leq n$ as follows:

$$Y_{i_1,i_2} = \# \text{ of pairs } (j_1, j_2) \text{ such that } 1 \leq j_1 \leq j_2 \leq n \text{ and } \sum_{i=1}^{i_2} \sum_{j=1}^{j_2} x_{i,j} \text{ is even.}$$

By definition the solution to the problem is equal to $\sum_{i_1=1}^n \sum_{i_2=i_1}^n Y_{i_1,i_2}$. Observe that for a fixed i_1 and i_2 , calculating Y_{i_1,i_2} is very similar to *Even Pairs*; the main difference being that instead of an array of numbers we have a matrix where each column plays a role of a single cell in the 1-dimensional version. This motivates the definition of an array $S_{i_1,i_2} = (s_1, \dots, s_n)$ where

$$s_j = \sum_{i=i_1}^{i_2} x_{i,j}.$$

Finally, the main observation is that Y_{i_1,i_2} is equal to the number of pairs (j_1, j_2) , where $1 \leq j_1 \leq j_2 \leq n$, such that $\sum_{j=j_1}^{j_2} s_j$ is even. This shows that the calculation of Y_{i_1,i_2} can be reduced to *Even Pairs* and since we can create the array S_{i_1,i_2} in linear time (by using precomputation as in the previous solution) we can calculate Y_{i_1,i_2} in linear time. Iterating this procedure over all pairs (i_1, i_2) ($1 \leq i_1 \leq i_2 \leq n$) we obtain an $O(n^3)$ solution.

3 Implementation

We use a global two-dimensional array `int M[201][201]` to store the input values and similarly `int pM[201][201]` stores the values of the precomputed matrix P, where `pM[i][j]` contains the value P_{ij} .

By using (2) we calculate the values of pM with the following piece of code which runs in $O(n^2)$ time.

```
1 // First we initialize the values of pM[0][i] and pM[i][0] to zero.
2 for (int i = 0; i <= n; ++i){
3     pM[i][0] = 0;
4     pM[0][i] = 0;
5 }
6
7 // Here we compute pM[i][j] by using the values pM[i'][j'] for i' <= i
8 // and j' <= j.
9 for (int i = 1; i <= n; ++i)
10     for (int j = 1; j <= n; ++j)
11         pM[i][j] = pM[i-1][j] + pM[i][j-1] - pM[i-1][j-1] + M[i][j];
```

Iteration through all possible submatrices can be easily done with four nested loops. The following snippet of code does that and calculates the sum of the elements of a submatrix in constant time by using precomputed matrix P.

```
1 // The variable counter will keep track of the number of quadruples
2 // whose corresponding submatrix has even sum.
3 int counter = 0;
4
5 for (int i1 = 1; i1 <= n; ++i1){
6     for (int j1 = 1; j1 <= n; ++j1){
7         for (int i2 = i1; i2 <= n; ++i2){
8             for (int j2 = j1; j2 <= n; ++j2){
9                 // qtuple is the element sum of the submatrix given by (i1, j1, i2, j2).
10                 int qtuple;
11                 qtuple = pM[i2][j2] - pM[i2][j1-1] - pM[i1-1][j2] + pM[i1-1][j1-1];
12                 if (qtuple % 2 == 0)
13                     ++counter;
14             }
15         }
16     }
17 }
18
19 // Output the result.
20 std::cout << counter << std::endl;
```

4 Appendix

The following code is an implementation of the $O(n^3)$ solution explained in Section 2.

```
1 #include <iostream>
2
3 // Input matrix.
4 int M[201][201];
5 // pm[i][j] = sum of elements in the submatrix (1, 1, i, j).
6 int pM[201][201];
```

```

7
8 int main(){
9     int T; std::cin >> T;
10
11     while (T > 0){
12         int n; std::cin >> n;
13
14         for (int i = 1; i <= n; ++i)
15             for (int j = 1; j <= n; ++j)
16                 std::cin >> M[i][j];
17
18         for (int i = 0; i <= n; ++i){
19             pM[0][i] = 0;
20             pM[i][0] = 0;
21         }
22
23         for (int i = 1; i <= n; ++i)
24             for (int j = 1; j <= n; ++j)
25                 pM[i][j] = pM[i-1][j] + pM[i][j-1] - pM[i-1][j-1] + M[i][j];
26
27         int solution = 0;
28         for (int i1 = 1; i1 <= n; ++i1){
29             for (int i2 = i1; i2 <= n; ++i2){
30                 // We reduce the problem to one dimension.
31                 int S[201]; // We do Even Pairs on array S.
32                 int pS[201]; // pS contains partial sums of S.
33                 pS[0] = 0;
34                 for (int k = 1; k <= n; ++k){
35                     S[k] = pM[i2][k] - pM[i2][k-1] - pM[i1-1][k] + pM[i1-1][k-1];
36                     pS[k] = pS[k-1] + S[k];
37                 }
38
39                 // Do Even Pairs O(n) algorithm on array S.
40                 int onedim_sol = 0;
41                 int even = 0, odd = 0;
42                 for (int j = 1; j <= n; ++j){
43                     // even = # of partial sums of array (S[1], ..., S[j-1]) that are even.
44                     // odd = # of partial sums of array (S[1], ..., S[j-1]) that are odd.
45                     if (pS[j] % 2 == 0){
46                         onedim_sol += even + 1;
47                         ++even;
48                     }
49                     else {
50                         onedim_sol += odd;
51                         ++odd;
52                     }
53                 }
54                 solution += onedim_sol;
55             }
56         }
57         std::cout << solution << std::endl;
58         --T;
59     }
60     return 0;
61 }

```