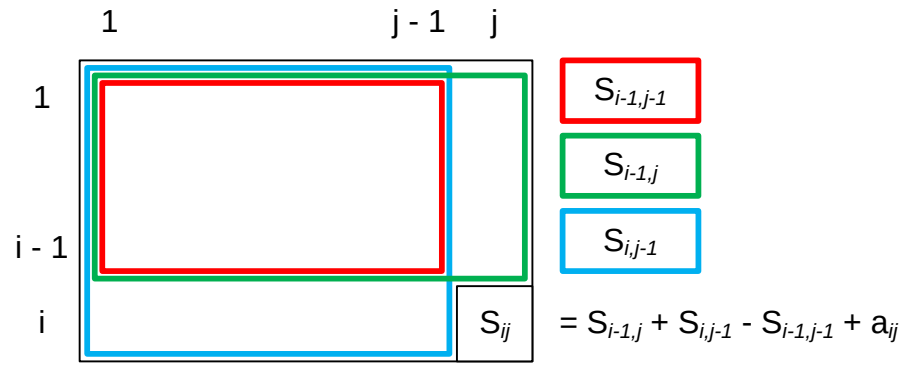
Dynamic Programming (square)

E-OLYMP **683. Partial matrix sum** The matrix of integer a_{ij} is given, where $1 \le i \le n$, $1 \le j \le m$. For all i, j find

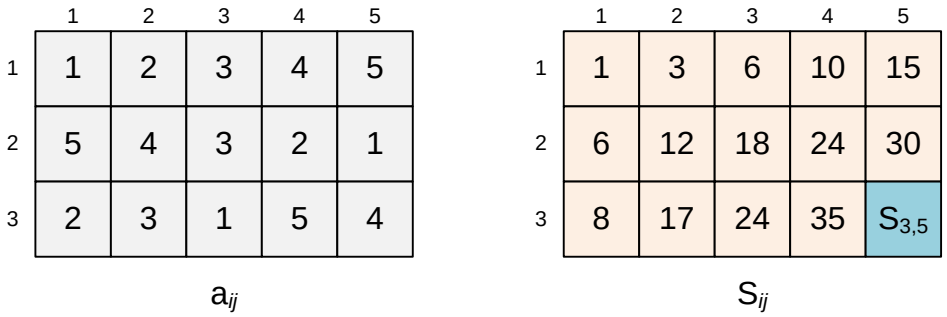
$$s_{i,j} = \sum_{k=1,t=1}^{k \le i, t \le j} a_{k,t}.$$

► Let a[i][j] be the input array, s[i][j] be the array of partial sums. We shall fill the array s in ascending order of lines, and the cells in each row – in ascending order of columns. Suppose we calculated this way all the values of s array till s[i][j].

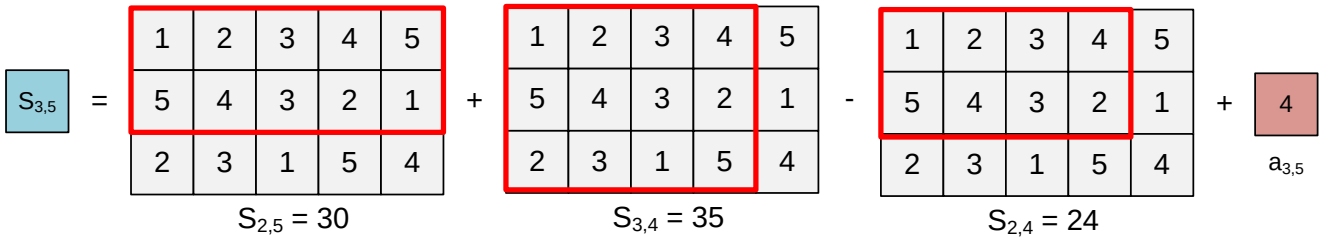


Then s[i][j] = s[i – 1][j] + s[i][j – 1] – s[i – 1][j – 1] + a_{ij}.

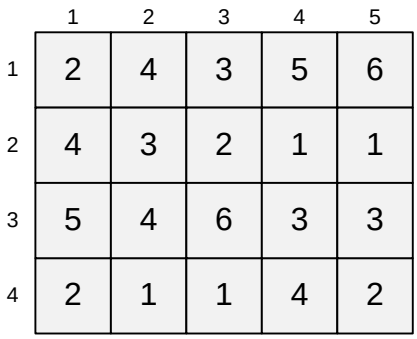
Consider for the given example how to calculate the value of s[3][5].



s[3][5] = s[2][5] + s[3][4] – s[2][4] + a_{35} = 30 + 35 – 24 + 4 = 45

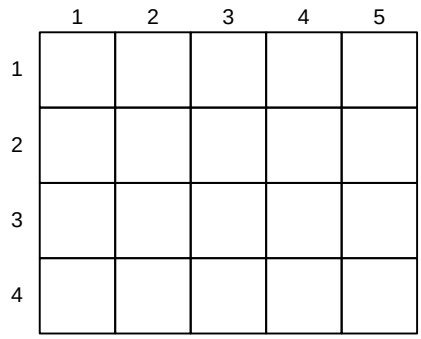


Exercise. For the given array a_{ij} compute the values of array s_{ij}.



	1	2	3	4	5
1	2	4	3	5	6
2	4	3	2	1	1
3	5	4	6	3	3
4	2	1	1	4	2

a_{ij}



	1	2	3	4	5
1					
2					
3					
4					

S_{ij}

Declare two two-dimensional arrays.

```
#define MAX 1001
int a[MAX][MAX], s[MAX][MAX];
```

Read the input data.

```
scanf("%d %d",&n,&m);
for(i = 1; i <= n; i++)
for(j = 1; j <= m; j++)
  scanf("%d",&a[i][j]);
```

Calculate the partial sums.

```
memset(s,0,sizeof(s));
for(i = 1; i <= n; i++)
for(j = 1; j <= m; j++)
  s[i][j] = s[i][j-1] + s[i-1][j] - s[i-1][j-1] + a[i][j];
```

Print the resulting array of partial sums.

```
for(i = 1; i <= n; i++)
{
  for(j = 1; j <= m; j++)
    printf("%d ",s[i][j]);
  printf("\n");
}
```

E-OLYMP **4018. Turtle** There is a turtle in the left top corner of rectangular table of size $n \times m$. Each cell of the table contains some amount of acid. Turtle can move right or down, its route terminates in right bottom cell of the table.

Each milliliter of acid brings turtle some amount of damage. Find the smallest possible value of damage that will receive a turtle after a walk through the table.

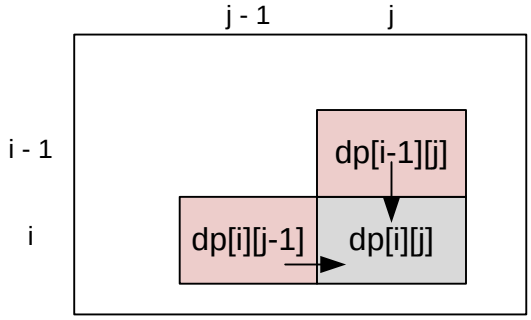
► Let a[i][j] contains the amount of damage for the turtle after visiting the cell (i, j). Let dp[i][j] contains the minimum possible damage for the turtle during the route from (1, 1) to (i, j).

Consider the base cases:

- dp[1][1] = a[1][1];
- dp[i][1] = dp[i – 1][1] + a[i][1], $1 < i \le n$ (first column);
- dp[1][j] = dp[1][j – 1] + a[1][j], $1 < j \le m$ (first row);

One can get into the cell (i, j) either from $(i - 1, j)$ or from $(i, j - 1)$. Since the damage is minimized, then

$$dp[i][j] = \min(dp[i - 1][j], dp[i][j - 1]) + a[i][j]$$



a_{il}

5	9	4	3
3	1	6	9
8	6	8	12

dp_{il}

5	14	18	21
8	9	15	24
16	15	23	35

First column:

- dp[2][1] = dp[1][1] + a[2][1] = 5 + 3 = 8,
- dp[3][1] = dp[2][1] + a[3][1] = 8 + 8 = 16.

First row:

- dp[1][2] = dp[1][1] + a[1][2] = 5 + 9 = 14,
- dp[1][3] = dp[1][2] + a[1][3] = 14 + 4 = 18.

Calculate the values of some non-border cells:

dp[2][2] = min(dp[1][2], dp[2][1]) + a[2][2] = min(14, 8) + 1 = 8 + 1 = 9

dp[3][4] = min(dp[2][4], dp[3][3]) + a[3][4] = min(24, 23) + 12 = 23 + 12 = 35

The desired path for the turtle is:

a_{ij}

	1	2	3	4
1	5	9	4	3
2	3	1	6	9
3	8	6	8	12

dp_{ij}

	1	2	3	4
1	5	14	18	21
2	8	9	15	24
3	16	15	23	35

Exercise. Given matrix a[i][j], find the values of matrix dp[i][j].

a_{il}

	1	2	3	4
1	3	2	6	5
2	4	6	1	8
3	5	3	4	5
4	7	3	3	5

dp_{il}

	1	2	3	4
1				
2				
3				
4				

E-OLYMP **4019. Turtle restoring** The turtle wants to pass the rectangular table as quickly as possible from top left corner to bottom right corner along the route with the least losses.

Print in the first line the minimal possible turtle's damage. In the next lines print the cells coordinates along which the appropriate path runs. Print the coordinates in the order like they appear on the route.

► Let a[i][j] contains the amount of damage for the turtle after visiting the cell (i, j). Let dp[i][j] contains the minimum possible damage for the turtle during the route from (1, 1) to (i, j).

Consider the base cases:

- dp[1][1] = a[1][1];
- dp[i][1] = dp[i – 1][1] + a[i][1], $1 < i \le n$ (first column);
- dp[1][j] = dp[1][j – 1] + a[1][j], $1 < j \le m$ (first row);

One can get into the cell (i, j) either from $(i - 1, j)$ or from $(i, j - 1)$. Since the damage is minimized, then

$$dp[i][j] = \min(dp[i - 1][j], dp[i][j - 1]) + a[i][j]$$

Start the movement from the right lower corner (n, m) to the left upper corner (1, 1) along the path of minimum damage. Initialize (i, j) = (n, m). From the cell (i, j) we can move either to (i – 1, j) or to (i, j – 1) depending on which of these values is less. If dp[i – 1][j] = dp[i][j – 1], then the movement can be continued into any of these two cells.

a_{il}

5	9	4	3
3	1	6	9
8	6	8	12

dp_{il}

5	14	18	21
8	9	15	24
16	15	23	45

path

5	14	18	21
8	9	15	24
16	15	23	45

Declare the arrays.

```
#define MAX 1010
int a[MAX][MAX], dp[MAX][MAX];
```

Read the input data.

```
scanf("%d %d", &n, &m);
for (i = 1; i <= n; i++)
for (j = 1; j <= m; j++)
  scanf("%d", &a[i][j]);
```

Initialize the first row and the first column of array dp.

```
dp[1][1] = a[1][1];
for (i = 2; i <= n; i++)
  dp[i][1] = dp[i - 1][1] + a[i][1];
for (j = 2; j <= m; j++)
  dp[1][j] = dp[1][j - 1] + a[1][j];
```

Find the minimum possible damage for the turtle for each cell.

```
for (i = 2; i <= n; i++)
for (j = 2; j <= m; j++)
  dp[i][j] = min(dp[i - 1][j], dp[i][j - 1]) + a[i][j];
```

Print the minimum damage with which you can reach the lower right corner.

```
printf("%d\n", dp[n][m]);
```

Initialize $(i, j) = (n, m)$. Start the movement from the right lower corner to the left upper corner.

```
i = n; j = m;
```

Continue moving until we reach the first row or first column. Push the point (i, j) into the *path* array.

```
while (i > 1 && j > 1)
{
  path.push_back(make_pair(i, j));
  if (dp[i - 1][j] + a[i][j] == dp[i][j]) i--; else j--;
}
```

Move to the cell (1, 1) either by the first row or by the first column.

```
while (i > 1) path.push_back(make_pair(i, j)), i--;
while (j > 1) path.push_back(make_pair(i, j)), j--;
path.push_back(make_pair(1, 1));
```

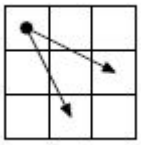
The path should be reversed.

```
reverse(path.begin(), path.end());
```

Print the path found.

```
for (i = 0; i < path.size(); i++)
  printf("%d %d\n", path[i].first, path[i].second);
```

E-OLYMP **4021. Knight move** The rectangular board of size $n \times m$ (n rows and m columns) is given. The chess knight is located in the left upper corner. You must relocate it to the right lower corner. The knight can move only either two cells down and one right, or one cell down and two cells right.

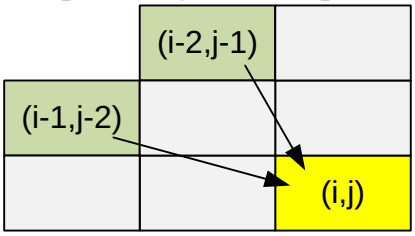


Find the number of knight paths from the left upper corner to the right lower corner.

► Let dp[i][j] contains the number of ways to run from left upper corner – the cell with coordinates (1, 1) into right lower corner – the cell with coordinates (n, m). Assign zeroes to array dp and set dp[1][1] = 1.

According to the knight moves, we can come into cell (i, j) only either from (i – 1, j – 2) or from (i – 2, j – 1). So

$$dp[i][j] = dp[i - 1][j - 2] + dp[i - 2][j - 1]$$



Fill the array dp for the board of size 7 * 7:

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0
2	0	0	0	1	0	0	0	0
3	0	0	1	0	0	1	0	0
4	0	0	0	0	2	0	0	1
5	0	0	0	1	0	0	3	0
6	0	0	0	0	0	3	0	0
7	0	0	0	0	1	0	0	6

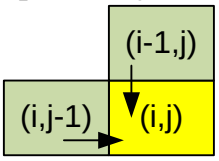
E-OLYMP **1704. Clever turtle** There is a field of cellular size $m \times n$. The turtle sits in the lower left corner. It can go only right or up. Before getting to the top right

corner, it is interested in the question: how many ways are there to get from the origin to the upper right corner?

► Consider two dimensional array dp, where dp[i][j] equals to the number of ways for the turtle to go from (1, 1) to (i, j). Let dp[1][1] = 1.

The turtle can get into the cell (i, j) either from (i – 1, j) or from (i, j – 1). Hence

$$dp[i][j] = dp[i - 1][j] + dp[i][j - 1]$$



Initialize array dp with 0. For us it will be significant to put zeros into zero line and zero column of array dp. If for example i = 1, then dp[i – 1][j] = dp[0][j] = 0 and the dynamic equation turns into dp[i][j] = dp[i][j – 1] (the first line is recalculated this way). If j = 1, the dynamic equation turns into dp[i][j] = dp[i – 1][j] (the first column is recalculated this way). Given that dp[1][1] = 1, one can conclude that all the cells of the first line and of the first column will contain 1.

Fill the array dp for the field of the size 4 * 3:

i \ j	0	1	2	3
0	0	0	0	0
1	0	1	1	1
2	0	1	2	3
3	0	1	3	6
4	0	1	4	10

E-OLYMP **5101. Hodja Nasreddin** Hodja Nasreddin is located in the upper left corner of the table of the size $n \times n$, and his donkey is located in the lower right corner. Hodge goes only to the right or down, a donkey goes only to the left or up.

In how many ways they can meet in one cell? (Two ways are considered different if Hodja or donkey has different routes).

► Let a[i][j] contains the number of ways for Hodja Nasreddin to go from (1, 1) to (i, j).

Let b[i][j] contains the number of ways for donkey to go from (n, n) to (i, j).

The number of ways for Hodja Nasreddin and donkey to meet in the cell (i, j) equals to a[i][j] * b[i][j]. To get the answer you need to find the sum of products modulo 9929:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij} \cdot b_{ij} \bmod 9929$$

Let's build the arrays for n = 3:

1	1	1
1	2	3
1	3	6

a_{ij}

6	3	1
3	2	1
1	1	1

b_{ij}

6	3	1
3	4	3
1	3	6

$a_{ij} \cdot b_{ij}$

$$\sum_{i=1}^{3}\sum_{j=1}^{3} a_{ij} \cdot b_{ij} = (6 + 3 + 1) + (3 + 4 + 3) + (1 + 3 + 6) = 30$$