**Length of a Longest Path in a Grid Problem**

*Find the length of a longest path in a rectangular grid.*

**Input:** Integers *n* and *m*, an *n* × (*m*+1) matrix *Down*, and an (*n*+1) × *m* matrix *Right*.

**Output:** The length of a longest path from source (0, 0) to sink (*n*, *m*) in the *n* × *m* rectangular grid whose edge weights are defined by the matrices *Down* and *Right*.

Imagine you are a tourist in Midtown Manhattan, and you want to see as many sights as possible on your way from the corner of 59th Street and 8th Avenue to the corner of 42nd Street and 3rd Avenue (Figure below). However, you are short on time, and at each intersection, you can only move south (↓) or east (→). You can choose from many different paths through the map, but no path will visit all the sights. The challenge of finding a legal path through the city that visits the most sights is called the **Manhattan Tourist Problem**.

**Input Format.** The first line of the input contains the integers *n* and *m* (separated by a space). The next *n* lines (each with *m* + 1 space-delimited numbers) represent the matrix *Down*. The next line is the “-” symbol (to separate the matrices *Down* and *Right*). The next *n* + 1 lines (each with *m* space-delimited numbers) represent the matrix *Right*.

**Output Format.** The length of a longest path from source (0, 0) to sink (*n*, *m*) in the *n* × *m* rectangular grid whose edge lengths are defined by the matrices *Down* and *Right*.

**Constraints.** 1 ≤ *n* ≤ 20; 1 ≤ *m* ≤ 20

**SAMPLE DATASET:**

Input:

4 4

1 0 2 4 3

4 6 5 2 1

4 4 5 2 1

5 6 8 5 3

-

3 2 4 0

3 2 4 2

0 7 3 3

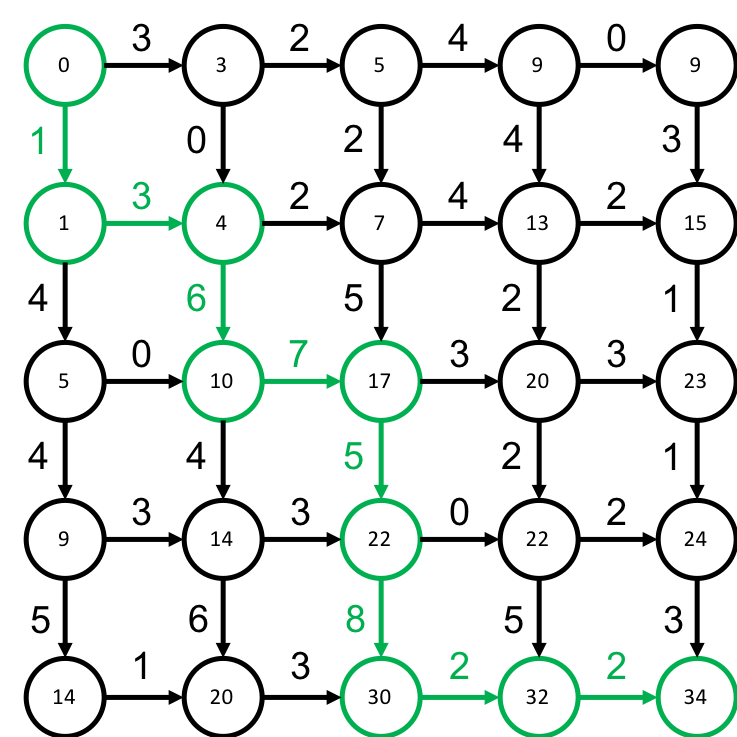
3 3 0 2

1 3 2 2

Output:

34

The sample dataset is not actually run on your code. The longest path in the grid shown below in green has length 34.



**TEST DATASET 1:**

Input:

2 2

20 0 0

20 0 0

-

0 0

0 0

10 10

Output:

60

This dataset checks that your code considers the first column of *Down* and the last row of *Right*. If there is some off-by-one error (possibly due to 0/1 indexing confusion) in updating the dynamic programming matrix it could be possible to miss these edges.

**TEST DATASET 2:**

Input:

2 2

0 0 20

0 0 20

-

10 10

0 0

0 0

Output:

60

This dataset checks that your code considers the last column of *Down* and the first row of *Right*. If there is some off-by-one error (possibly due to 0/1 indexing confusion) in updating the dynamic programming matrix it could be possible to miss these edges. Similar to Test Dataset 1.

**TEST DATASET 3:**

Input:

2 2

20 0 0

0 0 0

-

0 30

0 0

0 0

Output:

30

This dataset checks that you are not using some sort of greedy approach to solving this problem. If the movement with the highest weight is chosen at each step the first entry of *Down* should be chosen. This will result in missing the higher total weight that just goes all the way right then down.

**TEST DATASET 4:**

Input:

5 3

20 5 0 10

0 5 10 0

10 10 0 15

0 20 20 25

30 10 5 30

-

0 30 15

10 20 10

10 10 20

20 25 30

15 35 40

15 10 25

Output:

175

This dataset checks that your code can correctly parse and use inputs in which *n* and *m* are not the same. In all previous test datasets *n* and *m* were the same. If your output doesn’t match the correct output make sure that your dynamic programming matrix has the dimensions . In previous datasets your code could get away with creating a dynamic programming matrix with dimensions or , but implementations relying on those dimensions will fail this dataset.

**TEST DATASET 5:**

Input:

3 5

0 5 10 0 10 10

15 0 20 20 25 30

10 5 30 15 0 20

-

0 30 15 10 20

10 10 10 20 20

25 30 15 35 40

15 10 25 15 20

Output:

180

This dataset checks that your code can correctly parse and use inputs in which *n* and *m* are not the same. This dataset is similar to Test Dataset 4 except that the larger dimension is now *m* instead of *n*.