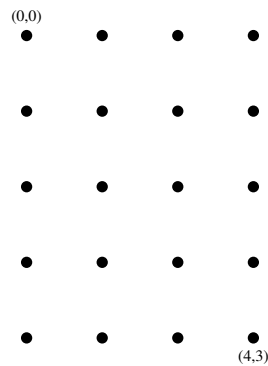
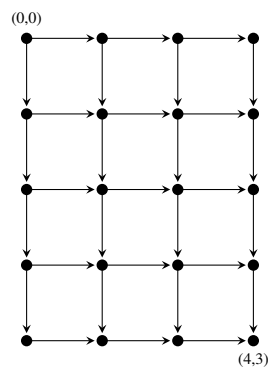


### Least Cost Path in a Rectangular Grid

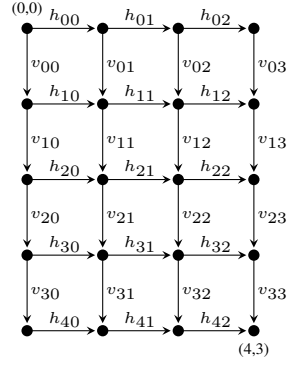
Consider a rectangular grid points/nodes at locations  $(i, j)$  in a matrix where  $i$  and  $j$  are integers with  $0 \leq i \leq m$  and  $0 \leq j \leq n$ . Here, as usual for locations in a matrix, the index  $i$  refers to a row and index  $j$  refers to a column. For example, we can take  $m = 4$  and  $n = 3$  and our grid is shown in the following figure.



Assume that for each node we connect it to its neighbor on its right (if there is one) by a horizontal arrow, and to its neighbor below by a vertical arrow (if there is one).

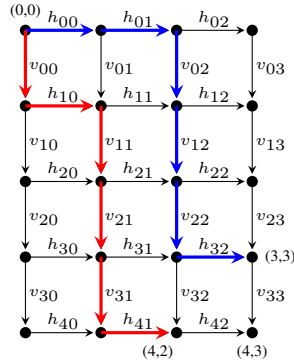


Assume each horizontal arrow from  $(i, j)$  to  $(i, j + 1)$  is assigned a cost denoted by  $h_{ij}$ , and each vertical arrow from  $(i, j)$  to  $(i + 1, j)$  is assigned a cost  $v_{ij}$ .



The problem is to find the least cost path from  $(0, 0)$  to  $(m, n)$  along a sequence of arrows, where the cost of a path is the sum of the costs at all of its individual arrows. This is an example of a problem that can be solved using dynamic programming. The basic idea is to work backwards. In order to get from  $(0, 0)$  to  $(m, n)$  we first have to first visit either the node  $(m, n - 1)$  or  $(m - 1, n)$ .

For example, suppose we already know the best path from  $(0, 0)$  to  $(4, 2)$  shown in red, and the best path from  $(0, 0)$  to  $(3, 3)$  shown in blue, in the following figure.



Then we need only decide between the completions of two possible paths.

The best path from  $(0, 0)$  to  $(4, 3)$  would involve following the red path and then a final horizontal move or taking the blue path followed by a final vertical. Assuming the cost of the red path is  $C_{4,2}$  the cost of it plus the final move is

$$C_{4,2} + h_{42}$$

and if the cost of the blue path is  $C_{3,3}$  then the cost of that path plus the final move is

$$C_{3,3} + v_{33}.$$

Comparing these two costs leads to our decision as to which path to take. If there is a tie, we can take either path. We then conclude that

$$C_{4,3} = \min\{C_{4,2} + h_{42}, C_{3,3} + v_{33}\}.$$

We refer a node  $(i', j')$  as a *predecessor* of a node  $(i, j)$  if there is an arrow pointing from  $(i', j')$  to  $(i, j)$ . Observe that a node can have 0, 1 or 2 predecessors.

- $(0, 0)$  has 0 predecessors,
- $(i, 0)$  has 1 predecessor if  $i > 0$ ,
- $(0, j)$  has 1 predecessor if  $j > 0$ , and
- $(i, j)$  has 2 predecessors if  $i > 0$  and  $j > 0$ .

For any node  $(i, j)$  we denote the cost of an optimal path from  $(0, 0)$  to  $(i, j)$  by  $C_{i,j}$

We define a *direction*  $d_{i,j}$  associated with any node with at least one predecessor. If there is a least cost path from  $(0, 0)$  to  $(i, j)$  whose final step is from  $(i - 1, j)$  to  $(i, j)$  we define  $d_{i,j} = V$ . Otherwise, there must be a least cost path from  $(0, 0)$  to  $(i, j)$  whose final step is from  $(i, j - 1)$  to  $(i, j)$  and we define  $d_{i,j} = H$ .

**Key observation.** Once we have determined  $d_{i,j}$  for every node  $(i, j)$  we can find an optimal path from  $(0, 0)$  to  $(m, n)$  working backwards from  $(m, n)$  to find out least cost path from  $(0, 0)$  to  $(m, n)$ . I.e. if  $d_{m,n} = H$  then the optimal path visited  $(m, n - 1)$  and if  $d_{m,n} = V$  then our optimal path visited  $(m - 1, n)$  before reaching  $(m, n)$  in the next step. Whichever node we determine to have visited, we repeat this process to determine the previously visited node, until we get to  $(0, 0)$ .

## Getting started

To get started with the algorithm, note that there is only one path along the right hand border of our grid from  $(0, 0)$  to  $(m, 0)$  so we can initialize  $C_{0,0} = 0$  and  $C_{i,0} = \sum_{p=0}^{m-1} v_{p0}$  for  $i = 1, \dots, m$ , and similarly there is only one path across the top from  $(0, 0)$  to  $(0, n)$  we can immediately calculate the values of  $C_{0,j} = \sum_{p=0}^{n-1} h_{0p}$  for  $j = 1, \dots, n$ .

Now whenever we know the values of  $C_{i',j'}$  for every predecessor of  $(i', j')$  of  $(i, j)$  so we can calculate  $C_{i,j}$  and  $d_{i,j}$ , so we proceed to compute these quantities for the nodes in the following order

$$\begin{array}{ccccccccccc}
& (1, 1) & \rightarrow & (1, 2) & \rightarrow & (1, 3) & \cdots & \rightarrow & (1, n) & \rightarrow & \\
\rightarrow & (2, 1) & \rightarrow & (2, 2) & \rightarrow & (2, 3) & \cdots & \rightarrow & (2, n) & \rightarrow & \\
\rightarrow & (3, 1) & \rightarrow & (3, 2) & \rightarrow & (3, 3) & \cdots & \rightarrow & (3, n) & \rightarrow & \\
\rightarrow & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \rightarrow & \\
\rightarrow & (m-1, 1) & \rightarrow & (m-1, 2) & \rightarrow & (m-1, 3) & \cdots & \rightarrow & (m-1, n-1) & \rightarrow & \\
\rightarrow & (m, 1) & \rightarrow & (m, 2) & \rightarrow & (m, 3) & \cdots & \rightarrow & (m, n) & \rightarrow & 
\end{array}$$

Observe that if we use this ordering of the nodes, whenever we reach node  $(i, j)$  we will have already found  $C_{i', j'}$  and  $d_{i', j'}$  for its two predecessors.

### Pseudo-Code

We can now write some pseudo-code for finding a solution to our problem in the general case. The inputs to our problem are:

- $m, n$
- an  $m \times (n+1)$  matrix  $V = (v_{ij})_{i=0, \dots, m-1}^{j=0, \dots, n}$  with  $v_{ij}$  giving the cost of a vertical (down) move from  $(i, j)$  to  $(i+1, j)$
- an  $(m+1) \times n$  matrix  $H = (h_{ij})_{i=0, \dots, m}^{j=0, \dots, n-1}$  with  $h_{ij}$  giving the cost of a horizontal (right) move from  $(i, j)$  to  $(i, j+1)$ .

We'll use an  $(m+1) \times (n+1)$  matrix  $C = (c_{ij})_{i=0, \dots, m}^{j=0, \dots, n}$  to store the *cost* of a shortest path from  $(0, 0)$  to  $(i, j)$  for  $i = 0, \dots, m$  and  $j = 0, \dots, n$ .

We'll use an  $(m+1) \times (n+1)$  matrix of strings  $D = (d_{ij})_{i=0, \dots, m}^{j=0, \dots, n}$  to store, at each node  $(i, j)$  the value "H" if there is a best path from  $(0, 0)$  to  $(i, j)$  whose last step is a move from predecessor  $(i, j-1)$  to  $(i, j)$  otherwise the value is "V" since there must be a best path from  $(0, 0)$  to  $(i, j)$  whose last step is a move from the predecessor  $(i-1, j)$  to  $(i, j)$ .

Now we proceed as follows to calculate the terms in the matrices  $C$  and  $D$ .

```

Initialize  $c_{0,0} = 0$ 
Initialize  $c_{i,0} = \sum_{p=0}^{i-1} v_{p0}$  and  $d_{i,0} = \text{"V"}$  for  $i = 1, \dots, m$ .
Initialize  $c_{0,j} = \sum_{p=0}^{j-1} h_{0p}$  and  $d_{0,j} = \text{"H"}$  for  $j = 1, \dots, n$ .
For  $i = 1, \dots, m$ :
  For  $j = 1, \dots, n$ :
    If  $C_{i-1,j} + v_{i-1,j} \leq C_{i,j-1} + h_{i,j-1}$ 
      Take  $C_{i,j} = C_{i-1,j} + v_{i-1,j}$  and  $d_{i,j} = \text{"V"}$ 
    Else

```

Take  $C_{i,j} = C_{i,j-1} + h_{i,j-1}$  and  $d_{i,j} = \text{"H"}$

Observe that once we run this algorithm, for every node  $(i, j)$  except for  $(0, 0)$  it is the case that

- either  $(i - 1, j)$  is a predecessor of  $(i, j)$  and  $d_{i,j} = \text{"V"}$ , or
- $(i, j - 1)$  is a predecessor of  $(i, j)$  and  $d_{i,j} = \text{"H"}$ .

This gives us the cost  $C_{i,j}$  of the optimal path from  $(0, 0)$  to  $(i, j)$  with the cost of  $(0, 0)$  to  $(m, n)$  as a special case. Next we carry out the following steps to find the actual optimal path - the list of nodes visited by the optimal path.

```

Initialize list of OptimalPathNodes list as [(m, n)]
While length of Optimal PathNodes list is less than m + n + 1
    Let (i, j) be the last element in OptimalPathNodes.
    If (i - 1, j) is a predecessor of (i, j) and d(i,j) = "V"
        Append (i - 1, j) to OptimalPathNodes
    Else
        Append (i, j - 1) to OptimalPathNodes
Reverse the OptimalPathNodes list
Return OptimalPathNodes list

```

## Solving by brute-force

For small values of  $m$  and  $n$  it is possible to enumerate every possible path from  $(0, 0)$  to  $(m, n)$ . Since every such path must contain  $m$  horizontal moves and  $n$  vertical moves the paths are in one-to-one correspondence with the permutations of the list consisting of  $m$  values of "H" and  $n$  values of "V". The number of such permutations is

$$\binom{m+n}{n}.$$

There is a function in the *sympy* package that can be used to generate all permutations of a list that contains repeated elements. The following code

```

import numpy as np
m=3
n=2
L=["V" for i in range(m)]+["H" for i in range(n)]
allpaths=multiset_permutations(L)

```

```
for a in allpaths:  
    print(a)
```

yields the output

```
['H', 'H', 'V', 'V', 'V']  
['H', 'V', 'H', 'V', 'V']  
['H', 'V', 'V', 'H', 'V']  
['H', 'V', 'V', 'V', 'H']  
['V', 'H', 'H', 'V', 'V']  
['V', 'H', 'V', 'H', 'V']  
['V', 'H', 'V', 'V', 'H']  
['V', 'V', 'H', 'H', 'V']  
['V', 'V', 'H', 'V', 'H']  
['V', 'V', 'V', 'H', 'H']
```

To get the optimal path, we simply compute the cost of each path and take a path that gives the lowest cost.