- **Assumption**

  The cleaner will have to walk through two paths across the room. Which one goes in which direction does not matter so we can arbitrarily assign one of the paths to be from (1, 1) to (m, n) and the other one from (m, n) to (1, 1). For convenience purpose, in the following analysis we describe the paths as if they both are from (1, 1) to (m, n).

  The two paths DO NOT intersect with each other when the amount of dirt it sucks up is maximized for any $m > 1$ and $n > 1$, which can be proved by contradiction. Suppose an optimal solution with a total dirt amount of d_{cross} contains two paths that intersect at (i, j), a tile other than (1, 1) and (m, n). We divide the rest of the possible situations into five cases.

  - **Case 1: (i, j) is in the middle.** That is, $1 < i < m$ and $1 < j < n$. Then if we change one of the two paths to stop by at (i+1, j-1) or (i-1, j+1) instead, we obtain a solution with $d_{noncross} = d_{cross} + d_1$ or $d_{noncross} = d_{cross} + d_2$ that are greater than d_{cross}.

o	o	o		
o		o	d_1	
o	o	x	o	o
	d_2	o		o
		o	o	o

  - **Case 2: (i, j) is on the northern border.** That is, $i = 1$ and $1 < j \leq n$. Then we can stop by at (2, j-1) to get a better solution.

o	o	x	o	o
	d_1	o		o
		o		o
		o		o
		o	o	o

  - **Case 3: (i, j) is on the western border.** That is, $1 < i \leq m$ and $j = n$. Then we can stop by at (i-1, 2) to get a better solution.

o				
o	d_1			
x	o	o	o	o
o				o
o	o	o	o	o

  - **Case 4: (i, j) is on the eastern border.** That is, $1 \leq i < m$ and $j = 1$. Then we can stop by at (i+1, n-1) to get a better solution.

o	o	o	o	o
o				o
o	o	o	o	x
			d_1	o
				o

  - **Case 5: (i, j) is on the southern border.** That is, $i = m$ and $1 \leq j < n$. Then we can stop by at (i-1, j+1) to get a better solution.

o	o	o		
o		o		
o		o		
o		o	d_1	
o	o	x	o	o

To sum up, for all solutions that contain two crossed paths, we are always able to find a better solution where an additional tile can be cleaned while the cleaner still cleans every other tile. If the new solution still contains a node that they go across, we can even keep forming a better solution. This contradicts our original assumption that the original solution is optimal.

- **Algorithm**

  Because both paths will eventually reach the southeastern corner, each of them must go across a row at some point.

  Let $f(i, j, k)$ denote the sum of two paths until both paths reach row i at jth and kth column and clean the two tiles (i, j) and (i, k). We store calculated values into a three dimensional array for future use.

  Both (i, j) and (i, k) can come from the west or the north. If they both come from northern tiles, we need to consider the case together because our algorithm does not want a split between different rows. For the other three cases, however, we can incorporate them into two cases, either one of the tiles coming from the west.

  So, there are three ways to reach a state of cleaning (i, j) and (i, k) for the same row. Either both are from the north, or the first tile is from the west, or the second tile is from the west. Some states, however, only have some of the three ways because of the room boundaries.

  Our base case is $f(1, 1, 2) = d(1, 1) + d(1, 2)$. We also need to be careful on room boundaries.

  - **Case 1**: $i = 1$. Then j must be fixed to 2 or else the two paths will intersect. k can be anything between 2 and n. Therefore,

  $$f(1, 2, k) = f(1, 2, k - 1) + d(1, k).$$

  - **Case 2**: $k = 1$. Then (i, 1) must come from (i-1, 1) and (i, 2) from (i-1, 2). Therefore,

  $$f(i, 1, 2) = f(i - 1, 1, 2) + d(i, 1) + d(i, 2).$$

```
sub[1,1,2]=d(1,1)+d(1,2)
for k=2 to n do
  sub[1,1,k]=sub(1,1,k-1)+d(1,k)
end for
for i=2 to n do
  sub[i,1,2]=sub(i-1,1,2)+d(i,1)+d(i,2)
end for
=0
```

After that we will have some nested loops to calculate $f(i, j, k)$ in such sequence that the current calculation only depend on some f we have calculated before and d.

There are two cases for calculating $f(i, j, k)$ where $i > 1$ and $j > 1$.

- **Case 3**: $k = j + 1$. (i, k) is next to (i, j). (i, k) must come from (i-1, k) because the two paths do not intersect and that (i, k-1) is a part of another path. (i, j) can come from either (i-1, j) or (i, j-1). Therefore,

$$f(i, j, k) = max(f(i - 1, j, k) + d(i, j) + d(i, k), f(i, j - 1, k + d(i, j))).$$

- **Case 4**: $k > j + 1$. As mentioned before, there are three possible ways of reaching this state. Therefore,

$$f(i, j, k) = max(f(i-1, j, k) + d(i, j) + d(i, k), f(i, j-1, k) + d(i, j), f(i, j, k-1) + d(i, k)).$$

Our final solution to the problem is $f(m, n-1, n)$.

```
for i=2 to n do
  for j=2 to m-1 do
    for k=j+1 to m do
      if k==j+1 then
        way1=sub[i-1,j,k]+d(i,j)+d(i,k)
        way2=sub[i,j-1,k]+d(i,j)
        sub[i,j,k]=max(way1,way2)
      else
        way1=sub[i-1,j,k]+d(i,j)+d(i,k)
        way2=sub[i,j-1,k]+d(i,j)
        way3=sub[i,j,k-1]+d(i,k)
        sub[i,j,k]=max(way1,way2,way3)
      end if
    end for
  end for
end for
return sub[m,n-1,n]
=0
```

- **Correctness**

  The above DP algorithm is optimal as it has the following three traits.

  - **Complete Choice**: We have considered all the cases where two paths do not intersect. Proved by contradiction above, an optimal solution only contains two paths that do not intersect, so we must have considered all combinations of tiles that might be optimal.
  - **Inductive Structure**: We reduce each problem to several cases, each of which contains a sub-problem. We then pick up the one that yields the most desirable result.
  - **Optimal Substructure**: Although each of our choices includes a selection of either one or two tiles, each final solution must contain $(2m + 2n - 4)$ tiles. For all cases, maximum total amount of dirt is that of the optimal solution for P' plus that of our last DP choice. Therefore according to standard contradiction argument, we can conclude that the DP solution to P is also optimal.

- **Complexity**

  The overall complexity of the algorithm is O(mnk), k being the smaller one between m and n.

  For a m*n room, we calculate $f(i, j, k)$, each of which requires constant time. The whole loop takes O($m * n^2$) in total. However, if we compare m and n before doing these calculations, we can make a diagonal flip in O(mn) if necessary. The overall complexity will then be the smaller one between O(mn^2) and O(m^2n).