You are given an NxN grid where each cell has some gold coins. (Say T[i][j] denotes the number of gold coins at the (i,j)-th cell).
You start at the top-left (1,1) corner of the grid and want to reach the bottom-right (N,N) corner. In doing so, you want to maximise the number of coins that you collect.

**Constraint:** You can only move right or down at each step.

**Example:**

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>70</th>
<th>70</th>
<th>70</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>9</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>
Intuition:

We can only reach a $(x,y)$ from $(x-1, y)$ and $(x, y-1)$ i.e. from the left or top. If we know the optimal cost to reach these points, we know the optimal cost to reach $(x,y)$.

How to define sub-problems?

$$DP(i,j) = \text{Maximum path sum (considering all valid paths) to reach (i,j) starting from (1,1)}$$
Recursive Formulation:

\[
DP(i, j) = \begin{cases} 
T(1,1) & i = 1 \text{ and } j = 1 \\
\max( DP(i-1, j), DP(i, j-1) ) + T(i,j) & \text{otherwise}
\end{cases}
\]

Either take the best path to \((x-1, y)\) and then go to \((x,y)\)
Or take the best path to \((x, y-1)\) and then go to \((x,y)\)
Question 1A: Max Path In Grid

Not Efficient (Overlapping Subproblems):
Dynamic Programming:
Compute and store entries in matrix DP and row-wise

Pseudocode:
Initialize DP[n][n] to 0

for i = 1 to n:
    for j = 1 to n:
        if i == 1 and j == 1:
            DP[i][j] = T[1][1]
        else:
            if i > 1:
                DP[i][j] = max( DP[i][j], DP[i-1][j] + T(i,j) )
            if j > 1:
                DP[i][j] = max( DP[i][j], DP[i][j-1] + T(i,j) )

return DP[n][n]

Runtime:
(# Of Subproblems x Time To Solve A SubProblem)
O(n² x 1) = O(n²)
**Question 1A: Max Path In Grid**

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>70</th>
<th>70</th>
<th>70</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>9</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

![Table](image)

**DP Table**

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>72</th>
<th>142</th>
<th>212</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>102</td>
<td>103</td>
<td>145</td>
<td>217</td>
</tr>
<tr>
<td>103</td>
<td>103</td>
<td>108</td>
<td>146</td>
<td>222</td>
</tr>
<tr>
<td>104</td>
<td>113</td>
<td>155</td>
<td>226</td>
<td></td>
</tr>
</tbody>
</table>
You are given an $N \times N$ grid where each cell has some gold coins. (Say $T[i][j]$ denotes the number of gold coins at the $(i,j)$-th cell). There are two friends A and B. A starts at the top-left $(1,1)$ corner of the grid and want to reach the bottom-right $(N,N)$ corner. B starts at the bottom-left $(N,1)$ corner of the grid and want to reach the top $(1,N)$ corner. In doing so, you want to maximise the number of coins that you collect.

Constraints:
A can only move right or down at each step.
B can only move right or up at each step.
A and B can only have one common square in their paths.

Example:

```
<table>
<thead>
<tr>
<th>2</th>
<th>70</th>
<th>70</th>
<th>70</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>9</td>
<td>4</td>
</tr>
</tbody>
</table>
```
**Question 1B: Max Path In Grid**

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>70</th>
<th>70</th>
<th>70</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>9</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>70</th>
<th>70</th>
<th>70</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>9</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>70</th>
<th>70</th>
<th>70</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>9</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>
Intuition:

A and B MUST meet at at-least one square. Given that they meet at (x,y), what are the possibilities to reach and leave the square?

- A enters from left, B enters from left
- A enters from left, B enters from bottom

Can A leave from the bottom? Can both leave from the right? What's the only choice for B?

- A enters from top, B enters from left

Can B leave from the top? Can both leave from the right? What's the only choice for A?
**Question 1B: Max Path In Grid**

**Algorithm:**

(Case 1) A enters left and leaves from right, B enters from bottom and leaves from top  
(Case 2) B enters left and leaves from right, A enters from top and leaves from bottom

Assume that meeting point is (x,y):

<table>
<thead>
<tr>
<th>(x,y)</th>
<th>(x,y-1)</th>
<th>(x,y+1)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(x-1,y)</td>
<td></td>
</tr>
</tbody>
</table>

**Optimal cost for Case 1:**

Best path to reach from (1,1) to (x-1,y)  
+ Best path to reach from (x,y+1) to (n,n)  
+ Best path to reach from (n,1) to (x+1,y)  
+ Best path to reach from (x,y-1) to (1,n)

**Optimal cost for Case 2:**

Best path to reach from (1,1) to (x-1,y)  
+ Best path to reach from (x+1,y) to (n,n)  
+ Best path to reach from (n,1) to (x,y-1)  
+ Best path to reach from (x,y+1) to (1,n)
Dynamic Programming:
Compute and store entries in matrix DP and row-wise

Pseudocode:
Initialize $DP_1[i][j]$, $DP_2[i][j]$, $DP_3[i][j]$ and $DP_4[i][j]$ to 0

As in part (a) compute $DP_1$, $DP_2$, $DP_3$ and $DP_4$ tables such that:
$DP_1[i][j]$ stores max path sum from $(1,1)$ to $(i,j)$ - only down and right
$DP_2[i][j]$ stores max path sum from $(n,n)$ to $(i,j)$ - only up and left
$DP_3[i][j]$ stores the max path sum from $(n,1)$ to $(i,j)$ - only up and right
$DP_4[i][j]$ stores the max path sum from $(1,n)$ to $(i,j)$ - only down and left

$maxValue = 0$

// Iterate over all possible meeting points
for $i = 2$ to $n-1$:
    for $j = 2$ to $n-1$:
        $case1\_cost = DP_1[i][j-1] + DP_2[i][j+1] + DP_3[i+1][j] + DP_4[i-1][j] + T[i][j]$
        $case2\_cost = DP_1[i-1][j] + DP_2[i+1][j] + DP_3[i][j-1] + DP_4[i][j+1] + T[i][j]$
        $maxValue = \max(maxValue, \max(case1\_cost, case2\_cost))$
Question 2: Pretty Printing

You have \( n \) words such that the length of the \( i \)-th word is given by \( L[i] \).
You want to print these words in a document to minimize the overall cost with the following constraints.

There are at-most \( D \) characters that can be printed in a single line.
A word can’t be split across multiple lines.

**Cost of a line** = \((\text{Number of extra spaces in the line})^2\)

**Total Cost** = Sum of costs for all lines

**Example:**
Suppose, you have the words ['this', 'is', 'a', 'sample', 'list'] and \( D = 7 \)

<table>
<thead>
<tr>
<th>Word</th>
<th>Extra Spaces</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>this</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>is</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>6</td>
<td>36</td>
</tr>
<tr>
<td>sample</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>list</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>Cost</td>
<td></td>
<td>28</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Word</th>
<th>Extra Spaces</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>this</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>is</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>sample</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>list</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Cost</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Cost = ?
Question 2: Pretty Printing

Intuition:

Break this up to into smaller decisions.
You want to print n words.
How many words should we print in the last line?
Suppose we print k words in the last line, what are we left with? Is it a smaller problem to solve?

How to define sub-problems?

DP(i) = Minimum cost to print the first i words
Recursive Formulation:

\[ DP[i] = \begin{cases} 
\min \left( DP[j-1] + \text{Cost To Print Words } j \text{ to } i \text{ in a single line} \right) & i \neq 0 \\
0 & i = 0 
\end{cases} \]

Note that we are looking at the different possibilities for the last line in our document and selecting the best out of these.

Overlapping Subproblems:
**Dynamic Programming**: Compute and store entries in DP starting from $i = 0$ to $n-1$

**Pseudocode**: Initialize DP[] to 0

for $i = 1$ to $n$:
    $\text{minCost} = \infty$
    for $j = 1$ to $i$:
        $\text{Length_Of_Words}_{j \text{ to } i} = \text{Sum Of } L[j], L[j+1], \ldots, L[i]$
        $\text{Spaces_Between_Words} = i - j$
        if $\text{Length_Of_Words}_{j \text{ to } i} + \text{Spaces_Between_Words} > D$:
            $\text{Cost_To_Print}_{j \text{ to } i} = \infty$
        else:
            $\text{Cost_To_Print}_{j \text{ to } i} = (D - \text{Length_Of_Words}_{j \text{ to } i} - \text{Spaces_Between_Words}) ^ 2$

    $\text{minCost} = \min(\text{minCost}, \text{DP}[j-1] + \text{Cost_To_Print}_{j \text{ to } i})$
    $\text{DP}[i] = \text{minCost}$

return $\text{DP}[n]$

**Runtime**: \[
(\# \text{ Of Subproblems } \times \text{Time To Solve A SubProblem})
= O(n \times n^2) = O(n^3)
\]
Further Optimizations

Cost_To_Print_j_to_i can be computed more efficiently!

Let $A[j][i] = \text{Cost\_To\_Print\_j\_to\_i}$

for (i = 0; i < n; i++)
{
    length = L[i]
    if length > D: $A[i][i] = \text{INF}$
    else: $A[i][i] = (D - length)^2$

    for (j = i+1; j < n; j++)
    {
        length += L[j] + 1
        if length > D: $A[i][j] = \text{INF}$
        else: $A[i][j] = (D - length)^2$
    }
}

Now, the time to compute each sub-problem is just $O(n)$ and overall runtime is $O(n^2)$