Problem 67. By starting at the top of the triangle below and moving to adjacent numbers on the row below, the maximum total from top to bottom is 23.

3
7 4
2 4 6
8 5 9 3

That is, 3 + 7 + 4 + 9 = 23.Find the maximum total from top to bottom in triangle.txt (right click and 'Save Link/Target As...'), a 15K text file containing a triangle with one-hundred rows.
NOTE: This is a much more difficult version of Problem 18. It is not possible to try every route to solve this problem, as there are 299 altogether! If you could check one trillion (1012) routes every second it would take over twenty billion years to check them all. There is an efficient algorithm to solve it. ;o)

Knowledge required Dynamic Programming

Solution We will be not solving this using brute force, instead we will be thinking in terms of dynamic programming. This approach can be altogether solve problem-18 and problem-67. We start by taking the triangle input in matrix, where each row of the input is a row in the matrix, all the remaining entries which are not filled by default will be initialized to 0. Then these are the transition states.

$$tri\_sum[i][j] += \begin{cases} tri\_sum[i-1][j] & \text{if j == 0} \\ \max(tri\_sum[i-1][j-1], tri\_sum[i-1][j] & \text{otherwise} \end{cases}$$

Here `tri_sum[i][j]` means the maximum sum that we can get starting from (0,0) to (i, j). The same approach will work for problem-67. The time complexity is O(n^2)

Python Solution

```python
ROWS = COLS = 100

# initial matrix all filled with zeros
tri_sum = [[0 for j in range(COLS)] for i in range(ROWS)]

# read the file and fill the entries in tri_sum
with open('triangle.txt', 'r') as f:
    r = 0
    for line in f.readlines():
        line = list(map(int, line.split()))

        for c, el in enumerate(line):
            tri_sum[r][c] = el
        r += 1

for i in range(ROWS):
    for j in range(COLS):
        if i == 0 and j == 0: continue
        if j == 0:
            tri_sum[i][j] += tri_sum[i - 1][j]
        else:
            tri_sum[i][j] += max(tri_sum[i - 1][j], tri_sum[i - 1][j - 1])

# as we can end at any position in the last
# row we need to print maximum of all those
# values
print(max(tri_sum[ROWS-1]))
```