Triangle

This is a perfect fit for DP. Two standard ways - both rely on the same recurrence:

Let best[r][c] be the minimum path sum to reach cell (r, c).

Transition (because you can come from (r-1, c-1) or (r-1, c) if they exist):

- best[r][c] = triangle[r][c] + min(best[r-1][c-1], best[r-1][c])
- Treat out-of-bounds parents as +∞.
- Base: best[0][0] = triangle[0][0].

Answer: min over the last row of best.

Two practical implementations:

1) Top-down DP (triangle-sized table):

- Build a 2D DP array with the same shape as triangle.
- Initialize the top.
- For each row r from 1 to n-1, fill:
  - Left edge: best[r][0] = triangle[r][0] + best[r-1][0].
  - Middle cells: use the min of two parents.
  - Right edge: best[r][r] = triangle[r][r] + best[r-1][r-1].
- Take the minimum in the last row.

Time: O(total_cells)

Space: O(total_cells)

2) Bottom-up with O(n) extra space (the follow-up):

Key idea: compress into one array dp[c] representing "best from current row downwards".

- Start from the last row: set dp[c] = triangle[last][c] for all c.
- Move upward row by row:
  - For row r from n-2 down to 0:
    - For column c from 0 ... r:
      - dp[c] = triangle[r][c] + min(dp[c], dp[c+1]).
- At the end, dp[0] is the answer.

Why this works: dp[c] and dp[c+1] are the best path sums from the two adjacent children below; you update them in-place to be the best from the current cell.

Time: O(total_cells)

Space: O(n) where n is the number of rows.