1. We construct a new graph $G' = (V', E')$ and run the BFS shortest path algorithm on it.

   Let $V' = \{S \subseteq V : |S| = 3\}$ and $E' = \{\{v_1, v_2, v_3\} \to \{w_1, w_2, w_3\} : v_i \to w_i \in E, (v_i), (w_i) \in V'\}$.

   Constructing V' is $O(|V|^3)$ and $|V'| = O(|V|^3)$. Constructing E' is $O(|E|^3)$ because checking that the v_i and w_i are distinct is $O(1)$. $|E'|$ is also $O(|E|^3)$.

   If the BFS shortest path algorithm from (a, b, c) to (x, y, z) does not return ∞, then the puzzle is solvable. The BFS algorithm runs in $O(|E'| + |V'|) = O(|E|^3 + |V|^3)$ time, and our graph computation also takes $O(|E|^3 + |V|^3)$ time, making the entire algorithm $O(|E|^3 + |V|^3)$.

2. We construct a new graph $G' = (V', E')$ and run the BFS shortest path algorithm on it.

   Let $V' = (\{\alpha\} \cup V) \times (\{\beta\} \cup V)$. The first element in V' represents where we originated from and the second element represents our next vertex. (α, v) represents starting a path from v and (v, β) represents ending a path at v.

   To construct E', we do the following.

   i For each edge $v_1 \to v_2 \in E$, we construct the initial edges $(\alpha, v_1) \to (v_1, v_2) \in E'$. This is $O(|E|)$.

   ii For each edge $v_1 \to v_2 \in E$, we construct the terminal edges $(v_1, v_2) \to (v_2, \beta) \in E'$. This is also $O(|E|)$.

   iii For each edge $v_1 \to v_2, v_2 \to v_3 \in E$, we construct the acute edges $(v_1, v_2) \to (v_2, v_3)$ if and only if $\angle v_1 v_2 v_3 < \pi/2$ or $\angle v_1 v_2 v_3 = \pi$. Since we perform the angle check in $O(1)$, this is $O(|E|^2)$.

   If the BFS shortest path algorithm from $(\alpha, \textit{Start})$ to $(\textit{Finish}, \beta)$ does not return ∞, then an acute path exists. The BFS algorithm operates in $O(|E'| + |V'|) = O(|E|^2 + |V|^2)$, but it is well known that for planar graphs, $|E| \leq 3|V| - 6$ (see my response to @744), so $|E| = O(|V|)$, making the algorithm $O(|V|^2)$. We also have $O(|E| + |E| + |E|^2) = O(|V|^2)$ graph construction, making the entire algorithm $O(|V|^2)$.

3. We first state some properties of a shortest rectangle walk.

Proposition 1: We claim that the shortest rectangle walk must consist of alternating expansion and contractions, e.g. it will not have two consecutive expansions or contractions.

Proof. Assume the contrary, so that there exists rectangles (as sets) $R_1, \ldots, R_k$ that form the shortest rectangle walk $R_1 \to R_2 \to \cdots \to R_k$.

When $k = 1$, nothing happens and when $k = 2$, we must have an expansion and a contraction to move between 1×1 rectangles, so assume $k \geq 3$. Assume WLOG that $R_1 \to R_2 \to R_3$ is two consecutive expansions. Then $R_1 \subseteq R_2$ and $R_2 \subseteq R_3$ so $R_1 \subseteq R_3$ and $R_1 \to R_3$ is a valid expansion, forming a shorter walk. A similar argument applies for two consecutive contractions. □

Proposition 2: A smallest rectangle walk exists where each contraction goes to a 1×1 square.

Proof. Let $R_1 \to \cdots R_k$ be a shortest rectangle walk. By the alternating property, we know that R_{2i+1}, where $i \geq 1$, is the result of a contraction. We have $R_{2i} \supseteq R_{2i+1}$ and $R_{2i+1} \subseteq R_{2i+2}$ and $R_{2i+1} \neq \varnothing$, so we pick a representative $r \in R_{2i+1}$ and replace the walk $R_{2i} \to R_{2i+1} \to R_{2i+2}$ with $R_{2i} \to \{r\} \to R_{2i+2}$. □

We give the algorithm WalkLength which takes in a bitmap $M[1..n, 1..n]$, a start coordinate (x, y) and an end coordinate (x', y') and computes the length of a shortest rectangle walk from (x, y) to (x', y'), such that each contraction goes to a 1×1 square.

```
1: procedure WalkLength(M[1..n, 1..n], (x, y), (x', y'))
2:     V ← M
3:     E ← ∅
4:     for i ← 1 to n do
5:         for j ← 1 to n do
6:             v ← (i, j)
7:             E_1 ← Reach(V, v, 1, 1)
8:             E_2 ← Reach(V, v, 1, −1)
9:             E_3 ← Reach(V, v, −1, 1)
10:            E_4 ← Reach(V, v, −1, −1)
11:            E ← E ∪ [{v} × (E_1 ∪ E_2 ∪ E_3 ∪ E_4)]
12:        end for
13:    end for
14:    return 2 · BFS(V, E, (x, y), (x', y'))
15: end procedure
16: procedure Reach(M[1..n, 1..n], (i, j), d_x, d_y)
17:    h ← n
18:    V ← ∅
19:    for dx ← 0 to n do
20:        for dy ← 0 to h − 1 do
21:            v ← (i + dx · d_x, j + dy · d_y)
22:            if v out of bounds or M[v] = 1 then
23:                h ← dy
24:                break
25:            else
26:                V ← V ∪ {v}
27:            end if
28:        end for
29:    end for
30:    return V
31: end procedure
```

Each expansion and contraction representing an edge ($e \in E$) in our graph constitutes a length of 2 in the rectangle walk. Each call to Reach is clearly $O(n^2)$, and this is done in a loop with $O(n^2)$ iterations, so our graph construction algorithm is $O(n^4)$.

We now argue about the correctness of Reach. The two propositions ensure the correctness of our BFS algorithm, so long as our graph connecting 1×1 pixels to other 1×1 pixels is correct.

It initially starts by computing the maximum vertical reach from the pixel, which is bounded above by $h = n$. h will represent the maximum height our rectangle with width $dx + 1$ can potentially be without running into a black pixel, and we update this as we find more and more black pixels. Namely, if a $w \times h$ rectangle at (i, j) contains no black pixels but a $w \times h+1$ rectangle does, then a $w+1 \times h+1$ rectangle definitely contains black pixels so we only need to check the next column of pixels because we have already verified that the previous $w \times h$ rectangle contains only white pixels.

BFS is a BFS algorithm to compute the shortest path from some vertex to another and operates in $O(|E| + |V|)$ time, so the total running time of our algorithm is $O(n^4 + |E| + |V|)$, but $|V| = O(n^2)$ and $|E| = O(n^4)$, so the running time of WalkLength is $O(n^4)$.

Since the pictures and the explanation of the problem in the homework is not unique, we will just claim that we just call the algorithm with the initial vertex being $(1, 1)$ and the ending vertex being (n, n).