Given a set of disjoint polygon obstacles in the plane with \( n \) vertices total.

Find the path from \( s \) to \( t \) that avoids interior of the obstacles.

Fig. 154: Shortest paths and the visibility graph.
Today: $O(n^2)$ time algorithm
(there is a much harder $O(n \log n)$ time alg.)

Claim: Shortest path is a polygonal curve with vertices from $s$ to $t$ vertices of polygons.

If path did
Call \( p + q \) are mutually visible if segment \( pq \) avoids interiors of all obstacles.

**Visibility graph:**
vertices = \( s, t, + \) polygon vertices
edge \( uv \) iff \( u + v \) are mutually visible

May not be planar.
Could have \( \Theta(n^2) \) edges.
If given visibility graph weight edges by their length, use Dijkstra in
$O(E \log V) = O(n^2 \log n)$ time.

with Fibonacci heap in
$O(E + V \log V) = O(n^2)$. 
Constructing visibility graph:

Focus on just line segments.

Do angular sweep of rays from all vertices. Every time a ray hits a second vertex, check if pair is visible.
Point $v = (v_a, v_b)$ has dual line $v^\ast: y = v_a x - v_b$.

If a ray goes from $u$ to $v$, then $u^\ast$ and $v^\ast$ intersect at dual to that ray's line.

$x$-coordinate of line's dual is the slope.

$\Rightarrow$ Dual vertices in arrangement sorted left to right are dual to our events in increasing slope order.
For each (primal) vertex $v$, store

$f(v)$: first segment hit by forward bullet path from $v$

$b(v)$: $v$ with backward path

Event: $v$'s ray points to $w$:
- if $v$ and $w$ share a segment, they are mut. vis.
- Invisible: $f(v)$ is closer to $v$ than $w$: just move on
- segment entry: about to start sweeping W's segment as first thing V sees:
  add edge \( uv \) to graph \( f(v) \leftarrow w \)'s segment

- segment exit
  add \( wv \) to graph \( f(v) \leftarrow f(w) \)
Use a topological plane sweep of dual arrangement in $O(n^2)$ time total.

Simple Polygon Robots:

Given $O(1)$ complexity polygon robot & polygon obstacles with $n$ vertices.

Find free space in $O(n \log^2 n)$. It has complexity.

Find shortest route in free space in $O(n)$. 
\[ O(n^2) \text{ time}, \]
Range Searching

Given \( n \) points \( P \) in \( \mathbb{R}^d \) and a collection of range shapes. (rectangles, balls, etc.)

Want to preprocess \( P \) so that given a query range \( Q \), we can quickly answer something about \( P \cap Q \).
Range reporting: Return $p \in Q$

Range counting: Return $|p \cap Q|$

Today's orthogonal rectangular range queries: $Q$ is an axis-aligned rectangle.
Most data structures store canonical subsets $\{P_1, \ldots, P_k\}$.

Each $P_i \in P$.

$P \cap Q$ will be a disjoint union of some of these subsets.

Quick to find $Q$'s subsets and aggregate their info.

Subsets in data structure do overlap.
Don't want too much overlap to save space, but enough for quick queries.

Typically define subsets using a partition tree:

- root
- usually binary
- leaves correspond to points of $P$
- internal node $v$ has subset $P_v$ of descendant points
ID: "interval queries"

\[ P = \{ p_1, p_2, \ldots, p_n \subseteq \mathbb{R} \} \]

\[ Q = [x_{lo}, x_{hi}] \]

Sort points left to right and store as leaves in a balanced partition tree.

Each node stores the largest
point in subtree to the left.

Don't store canonical subsets explicitly. Can just look at leaves if you need to.

Binary tree with $n$ leaves $\Rightarrow O(n)$ space.
To do a query $Q = [x_{l}, x_{h}]$:

Find rightmost leaf $a < x_{l}$ and leftmost leaf $v > x_{h}$.

Canonical subsets for $Q$:

From where the search paths diverge, when step left for $x_{l}$, add subset for right child.

When right for $x_{h}$, use subset for left child.
To quickly count, store size of can, subsets on each node + add sizes.

Count in $O(\log n)$ time.

Report $k$ points.

$O(\log n + k)$