Goal: Fill in the `findAnswer` Method

```java
public class Rasterer {
  private static final double MAX_X = 100;
  private static final double MAX_Y = 100;

  public static List<String> findAnswer(double x1, double y1, double x2, double y2) {
    // TODO
    return null;
  }
}
```

What are some potentially useful helper methods?

Arbitrary Challenging Problem of the Day

Find all labels that overlap the query.

```java
findAnswer(20, 90, 40, 60)
/*AA, "AB", "BA", "BB"*/
```

Problem Decomposition in Software Engineering

**Decomposition.** Taking a complex task and breaking it into smaller parts. This is the heart of computer science. Using appropriate abstractions makes problem solving vastly easier.

**Perspective 1:** Software engineering.

- **Eliminating special cases** in k-d tree nearest made code simpler and more obvious.
- **Modularization** is decomposition for managing software complexity at a project level.
  - **Autocomplete.** Efficient search bar prefix queries.
  - **Heap.** Efficient priority queue for route finding.
  - **K-d Tree.** Efficient 2-d nearest neighbors to find start and goal vertices.
  - **A* Search.** Efficient route finding.
  - **Rasterer.** Efficient map tile display.

Problem Decomposition in CS Theory

**Decomposition.** Taking a complex task and breaking it into smaller parts. This is the heart of computer science. Using appropriate abstractions makes problem solving vastly easier.

**Perspective 2:** Computational complexity theory.

- **Reduction.** Using an algorithm for Problem Q to solve Problem P.

  "If any subroutine for task Q can be used to solve P, we say P reduces to Q."
Graph Problems and Their Solutions

Paths. Find a path from s to every reachable vertex.
- Depth-first search. $O(V + E)$ runtime with adjacency list.

Unweighted Single-Source Shortest Paths.
- Find a shortest path from s to every reachable vertex.
  - Breadth-first search. $O(V + E)$ runtime with adjacency list.

Weighted Single-Source Shortest Paths.
- Find a shortest path from s to every reachable vertex.
  - Dijkstra's algorithm. $O(E \log V + V \log V)$ runtime with adjacency list.

Weighted Single-Pair Shortest Paths.
- Find a shortest path from s to a single goal vertex.
  - A* search. Dijkstra's algorithm with $h(v, \text{goal})$ as priority. Runtime depends on heuristic.

Algorithm for Finding a Shortest Paths Tree
Given a weighted, directed graph with integer edge weights between 1 and 5, find the single-source shortest paths tree from s to every other vertex in the graph.
Your algorithm should be faster than Dijkstra's algorithm.

Dijkstra's Runtime Analysis
PQ.add(s, 0)
For all other vertices v, PQ.add(v, infinity)
While PQ is not empty:
  p = PQ.removeSmallest()
  Relax all edges from p
Relaxing an edge (v, w) with weight:
If distTo[w] > distTo[v] + weight:
  distTo[w] = distTo[v] + weight
  edgeTo[w] = v
  PQ.changePriority(w, distTo[w])

Overall: $O(V \log V + V \log V + E \log V)$.
Simple: $O(V \log V + E \log V)$.
Assuming $E > V$, this is just $O(E \log V)$ for connected graphs.

Reductions
Given a graph G, we created a new graph $G'$, then fed it to a related (but different) algorithm, and finally interpreted the result.

Preprocess
BreadthFirstPaths
BFS
Postprocess
SPT of G
Topological Ordering

Suppose we have tasks 0 through 7, where an arrow from \( v \) to \( w \) indicates that \( v \) must happen before \( w \).

Which graph algorithm could we use to find a valid ordering for these tasks?

Valid orderings include: \([0, 2, 1, 3, 5, 4, 7, 6]\), \([2, 0, 3, 5, 1, 4, 6, 7]\), …

Reduction to Dijkstra's

To find a vertical seam...

**Vertex.** Pixel in image.

**Edge.** Cost to go from a pixel to its 3 downward neighbors.

**Weight.** Energy function of 8 neighboring pixels.

**Seam.** Shortest path (sum of weights) from top to bottom.

Formal Problem Statement

Using AStarSolver, find the seam from any top vertex to any bottom vertex.

Given a digraph with positive edge weights, and two distinguished subsets of vertices \( S \) and \( T \), find a shortest path from any vertex in \( S \) to any vertex in \( T \).

Your algorithm should run in time proportional to \( E \log V \) in the worst case.