# CoE 163

## Computing Architectures and Algorithms

01b: Problem Solving

# PROBLEM SOLVING

**Problem solving** is an important skill to master for us engineers.

Such skill needs knowledge and mastery of a wide range of known algorithms, data structures, and classical problems.

# CONSIDER...

- Find shortest distance from EEEI to CHK
- Find the first 100 prime numbers
- Find shortest length of rubber needed to enclose a set of pins on a corkboard

# SOLVING THE PROBLEM

- How do we solve this problem as humans?
- How do we translate our solution into computer code?
- How do we decompose this problem if it is too big?
- How do we make it fast enough for our purposes?

# PROBLEM STRUCTURE

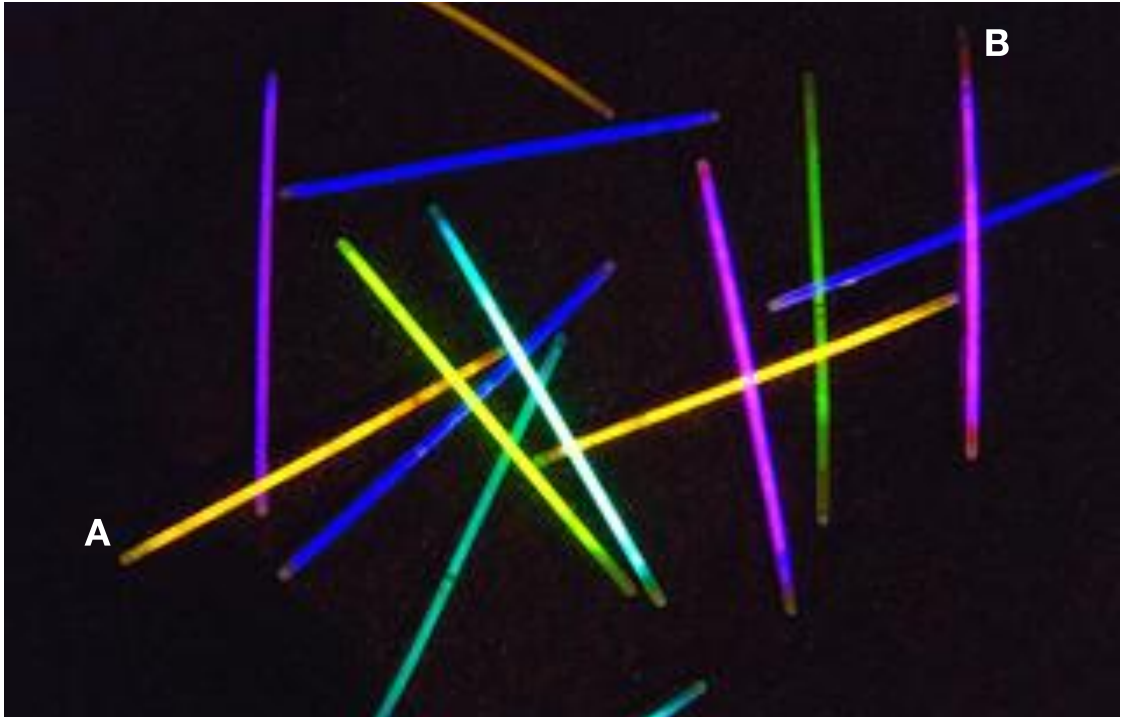
If we are lucky, our problem needs only one algorithm and a basic data structure to solve.

- Graph traversal
- Prime number sieve
- Convex hull

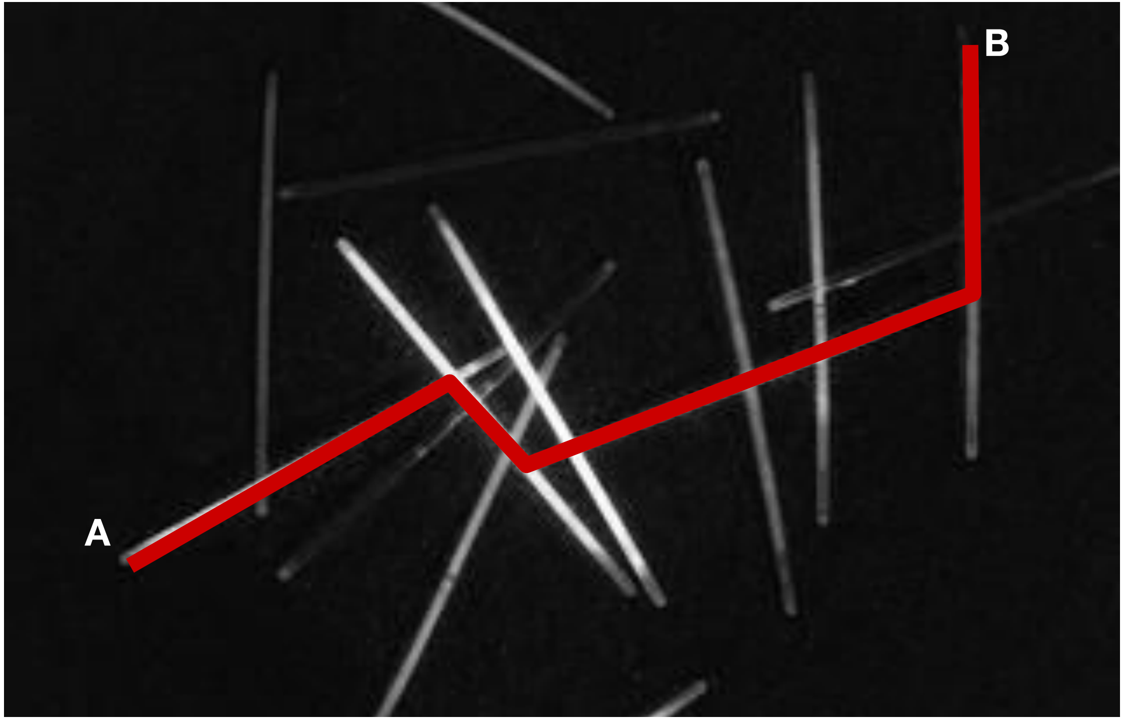
# CONSIDER...

Given pick-up sticks with coordinates of endpoints, find whether a stick A is connected to stick B in some way.

# ARE A AND B CONNECTED?



# ARE A AND B CONNECTED? YES!



# SOLVING...

Human solving is easy, but how do we solve it on a computer?

Maybe do graph traversal. But we haven't checked which sticks cross each other.

Seems like this consists of more than one problem.

# PROBLEM DECOMPOSITION

Most real-life problems need more than one algorithm to be solved.

**Problem decomposition** is the key. With knowledge of the basic problems, anyone can solve a larger problem consisting of multiple components!

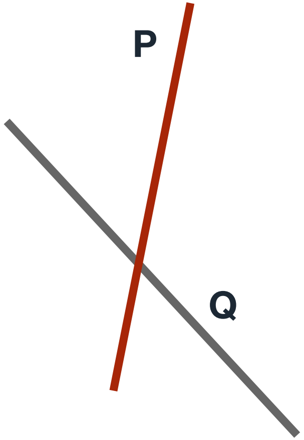
# PROBLEM DECOMPOSITION

- Check whether two sticks cross
- Check whether said sticks are connected

# PROBLEM DECOMPOSITION

- Geometry
  - Line-line intersection
- Graph
  - Graph traversal or transitive closure

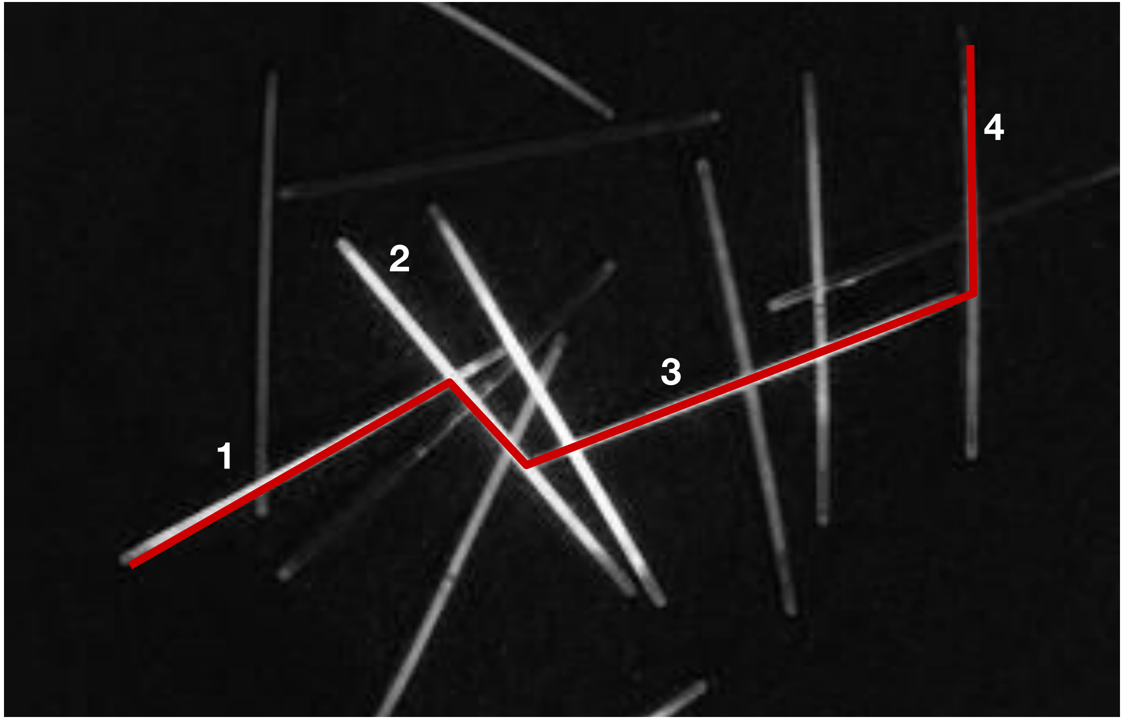
# LINE INTERSECTION



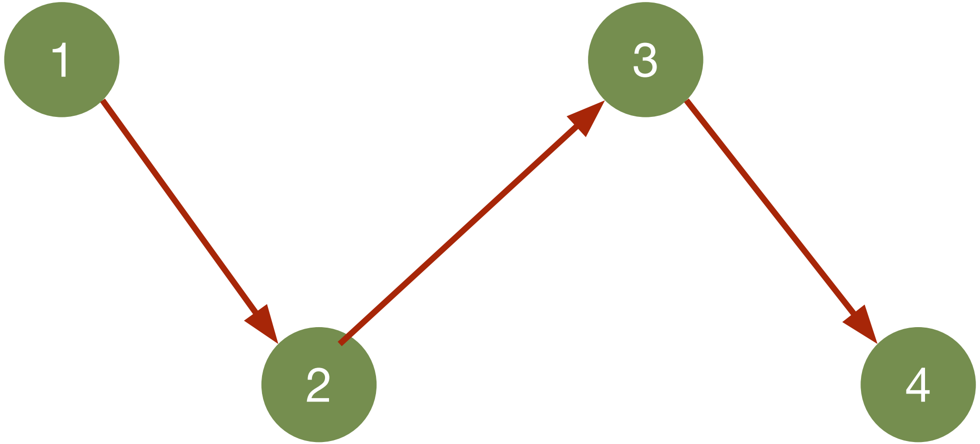

- Compute by solving a linear system equation
- Handle special case if slope of both lines are the same

$$\begin{bmatrix}(P2-P1)_x & (Q1-Q2)_x \\ (P2-P1)_y & (Q1-Q2)_y\end{bmatrix}\begin{bmatrix}s \\ t\end{bmatrix}=\begin{bmatrix}(Q1-P1)_x \\ (Q1-P1)_y\end{bmatrix}$$

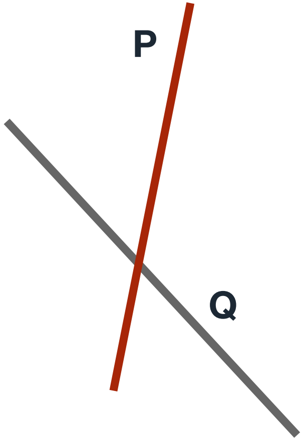
# GRAPH INTERCONNECTION



# GRAPH INTERCONNECTION



# GRAPH INTERCONNECTION




- Create graph vertex for each line
- Connect two vertices if the lines intersect
- Apply depth/breadth-first search to check whether line A is reachable from B
- Alternatively, use Warshall's Algorithm (dynamic programming) to calculate whether a path exists from A to B

# SOLUTION EDGES

Consider **edge cases** - cases deviating from the usual - when formulating solutions.

- Same-sloped sticks

Check whether the problem has certain **limitations** that do not apply.

- Sticks do not self-intersect

# SOLUTION SPEED-UPS

Do this **after solving** when your solution is too slow or does not perform well with respect to the specific application.

- Precreate graph
- Do not store line intersection data, but use it immediately to build the graph
- Use adjacency matrix/graph

# CONSIDER...

A tic-tac-toe game with infinite space (not 3x3) is being played by two players. Check who won given that a k number of consecutive O/X marks is needed for a player to win.

# GAME MECHANICS

- 2 players, 3 lines to win
- X starts the game
- X won!

| | | | | |
|---|---|---|---|---|
| O | | O | | |
| O | | | | |
| X | X | X | | |
| | | | | |
| | | | | X |

# PROBLEM DECOMPOSITION

- Data structure
  - Use a hashmap to save the coordinates
- Complete search
  - Check whether a marked cell is part of a line with k elements

# PROBLEM CONSIDERATIONS

- Space is infinite, so it's not feasible to save the whole board into memory
- Still need to be able to track the location of the markers in some way
- Depends on the number of consecutive marks needed

# SOLVING...

Human solving is easy, but how do we solve it on a computer?

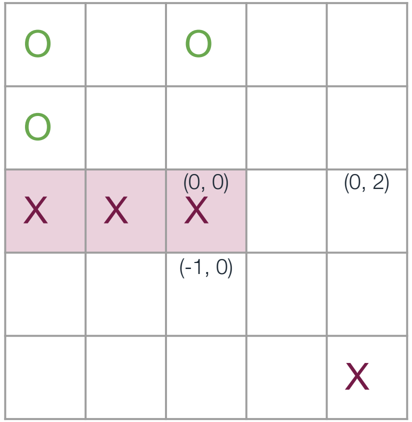
Iterate through each occupied cell and try to traverse a line away from it.

But we cannot save the whole board since space is infinite.

# PROBLEM DECOMPOSITION

- Have a general view of the infinite board
- Check whether a player won by looking at the markers

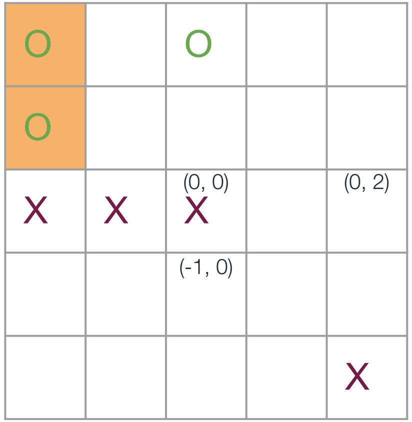
# DATA STRUCTURE




- Save coordinates into a hashmap
- Keys are coordinates and values are the marks on the board

(-2, -2) -> o
(-2, 0) -> o
(-1, -2) -> o
(0, -2) -> x
(0, -1) -> x
(0, 0) -> x
(2, 2) -> x

# COMPLETE SEARCH




- Starting at (-2, -2), traverse downward to check for same marker
  - Once decided, traversal should be downward or upward only
  - Same for similar directions (left to right, upper-left to lower-right, etc.)
- If k consecutive and same markers were found, flag the winner
- If after all the turns are processed and no winning lines were found, flag it as an ongoing game

# SOLUTION EDGES

- Edge cases
  - Ties
    - Technically an illegal game
    - Needs to exhaust all turns to find out
- Limitations
  - Maximum number of consecutive markers k and turns to process

# SOLUTION SPEED-UPS

- Use built-in data structure
  - Implementing your own is tedious!
  - C++ `map` or Python `dict`
- Convert map to a graph?
  - Extra information on node is needed to perform correct traversal
- Assemble map/graph while looping

# TIPS

- Be exposed to a lot of known CS problems, algorithms, and data structures
- Take your time
- Don’t be scared to try or feel defeated
- Feel free to get help (especially online)

# RESOURCES

- uHunt and CPx book for practice solving of known CS problems
- StackOverflow, Wikipedia, GeeksforGeeks, and related websites if you forgot how to implement an algorithm
- EEE 121 materials

# CoE 163

Computing Architectures and Algorithms

01b: Problem Solving