The Traveling Salesperson Problem (TSP)

Background:
The Traveling Salesperson Problem imagines that a salesperson, starting at his home or office, needs to visit a number of customers at different locations. The goal is for the salesperson to visit each location exactly once and end up back where he started, as efficiently as possible, without retracing any part of the route.

The problem: Given a graph of nodes and edges, where the weights of the edges represent some cost (distance, time, money, etc.), find the set of edges that make the lowest-cost “tour” of all the nodes. A tour is a route that visits each node exactly once and returns to the starting node.

History: The problem was first formulated in 1930 and is one of the most intensively studied problems in optimization because it has many applications in the real world. Almost any business that involves transportation of any kind is concerned with this problem because there are direct costs (in time, money, resources) to taking inefficient paths to visit all the nodes you need to get to. For example, a mail carrier delivering mail must go to each house exactly once, needs to end up at the starting location, and wants to minimize the length of the route. When you are running errands to different stores, or picking up friends to bring them to your place for a party, you need to visit each location once, return to where you started, and want to minimize your route. Can you think of a similar problem or example from real life?

Example:

This graph shows the direct routes between stores, and the distance of each route. You can assume any node is your home, the starting point.

Here are some possible routes, with their total distances, that visit every node once and return to wherever you started are:

- BADEC 2 + 12 + 10 + 3 + 4 = 31
- AEDCB 5 + 10 + 3 + 4 + 2 = 24

The shortest route that visits every node once and returns to wherever you started is:
- AECDB 5 + 3 + 3 + 8 + 2 = 21

Try It Out!
For all of the examples below, you’re also going to try to solve the traveling salesperson problem to find the shortest route that visits every vertex once and returns to where you started. You are encouraged to mark up these diagrams as you go.

...but you should be thinking...
- **Remember**: In computer science, “solving a problem” doesn’t mean finding an answer to an instance of a problem; it means finding an algorithm that might be able to solve any instance of that problem.
- As you look at the problems, your brain is working to find a solution. You might think you’re just trying “random stuff” but you’re not. You are using your human intelligence to help you.
- **Think about your own thinking process.**
- **Could you express a way to solve this route-finding problem as an algorithm?**

**Directions**
- Find the shortest path that visits each node exactly once (i.e. makes a cycle) in each of the graphs below.
- Highlight the route and make a note of the total distance.
- When you’re done, compare with a partner to see if you found the same things.
- In the “Algorithms Notes” area, jot down a few ideas for how an algorithm to find the shortest route might work. Maybe make a few notes about what’s potentially tricky, what things you want to be sure to remember.

**Algorithm Notes:**
In this final algorithm detour, you are introduced to the Traveling Salesperson Problem, a classic problem in computer science for which there is no known algorithm to solve it, other than brute force. The number of possible solutions grows extremely fast, even for small inputs, and quickly becomes "unreasonable" to solve, making it a computationally hard problem. The ideas of computationally hard problems are leveraged for encryption to make ciphers that take an unreasonable amount of time to crack (as in thousands of trillions of years), but computationally hard problems are also important in their own right. There are many problems for which we wish we had reasonable algorithmic solutions - especially in medical fields - and we're still on the hunt to find them. No one has yet mathematically proven whether or not the problems we currently think are "hard" actually are.

Reasonable or Unreasonable Time

Computers work fast, but they have limits. In computer science, we have an actual mathematical hard line between reasonable and unreasonable runtimes.

"Reasonable" means the number of things the computer has to do is proportional to the size of the input to the problem. For example, the Minimum Spanning Tree and Shortest Path Problems are “reasonable” because they had algorithms that solved them by considering every edge in the graph once. The amount of time it takes is proportional to the number of edges. If the number of edges is $n$, even if there was an algorithm that had to look at the edge $n^2$ times, or $n^3$ times, that’s still reasonable.

"Unreasonable" means the number of things the computer has to do grows as an exponent of the size of the input. So if you discovered that an algorithm made the computer do $2^n$ things, that’s not reasonable, because it means every time the size of the input ($n$) gets bigger, the solution gets massively further out of reach. $n!$ is another running time that is considered unreasonable. In real life, “unreasonable” problems would take a modern computer trillions and trillions of years to churn through all the possibilities.

So the brute force solution to TSP is unreasonable -- at least as far as we know! In fact, it’s an open question how to solve this problem efficiently. If anyone finds a solution that runs in a reasonable time, a lot of security and encryption algorithms are in trouble, because many are based on the fact that this (and related problems) are unreasonable to solve.

Reflection:

1. Describe what it means for a problem to be “computationally hard.”
2. What strategies do people use to solve large computationally hard problems?
3. Why are computationally hard problems important in encryption strategies?
4. If the TSP is unsolvable for finding an exact solution, how do you think package delivery companies optimize their delivery routes?