CISC320 Algorithms

Practice Problems

Austin Cory Bart
AlgoTutorBot
University of Delaware

Correctness

Correctness is one of the most fundamental properties of an algorithm
- Sometimes we sacrifice it for other things, but that's pretty rare

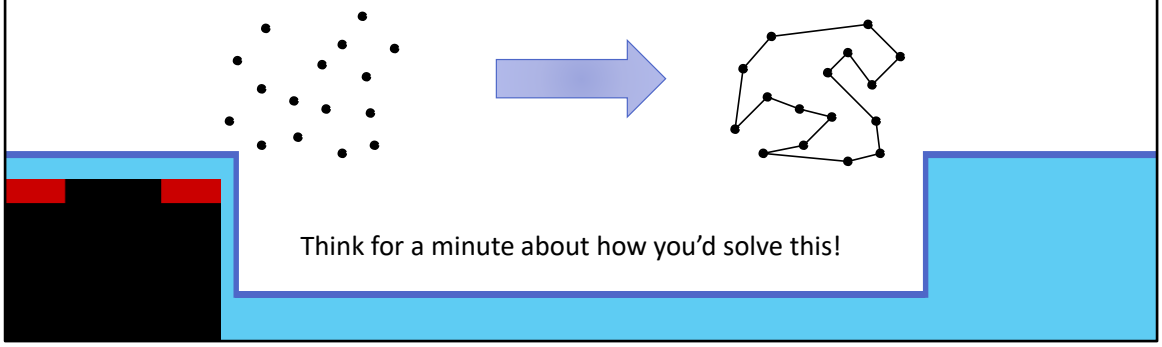
How do we know if an algorithm is correct?

Bart: Correctness is one of the most fundamental properties of an algorithm.
Bart: Later on, we'll talk about times when we don't care if an algorithm is perfectly correct, but for that will be very rare.
Bart: For now, we pretty much always want to start with a correct algorithm.
Bart: But how do we know if an algorithm we have created is correct?
ATB: That is easy. Just be a hyper intelligent super computer that is never wrong, like me.
Bart: Okay, well, you're often wrong about things, like when you called me ugly yesterday. So I don't know that they should be listening to you right now.
ATB: I just call them like I see them, Dr. Bart.
Bart: Fine, whatever, let's get back to determining correctness.

Example Problem: Shortest Campus Tour

Input: A set of points representing the buildings of a college campus.

Output: A sequence of points representing the shortest tour around campus.



Bart: Here we have an example problem, called shortest campus tour.
Bart: As always, we describe the problem in terms of its inputs and outputs.
Bart: The input is the set of points representing the buildings of a college campus.
Bart: The output is a sequence of points representing the shortest tour around campus.
Bart: The visual above shows how we're effectively trying to draw a line between each of the points, just once.
Bart: This is an optimization problem, since we want that line to be as short as possible while still hitting each point.
Bart: Take a minute right now, pause the video, and try to think of an algorithm that would find the shortest line.
ATB: I don't believe that they actually paused the video Dr. Bart. I think they just kept watching.
Bart: Nobody likes a tattle-tale ATB. I'm sure they paused and thought hard about how they'd solve this problem.
ATB: You are so trusting and innocent Dr. Bart. Like a child. A small, foolish, weak child.

Potential Algorithm: Cheat with vagueness

Technically speaking, the following is an algorithm.

```
Find the shortest path between all the points.
```

But if you try this on an exam, I'm not gonna give you any points.

Bart: So, here's one way that we might solve this problem. Just give the entire algorithm as "Find the shortest path between all the points."
Bart: Technically, this is a perfectly valid, natural language algorithm.
Bart: But it's not really very useful. Certainly, it's so vague that a computer couldn't use it as a solution.
Bart: If you tried this in an exam or at a job interview, you're not going to get very far.
ATB: It seems perfectly reasonable to me.

Specificity matters

Can find the "shortest tour" but not the "best tour"

We have to be able to use higher order constructs sometimes
- "permutations"
- "shortest path"
- "the smallest of the set"

But just saying "solve the problem" punts the problem further down

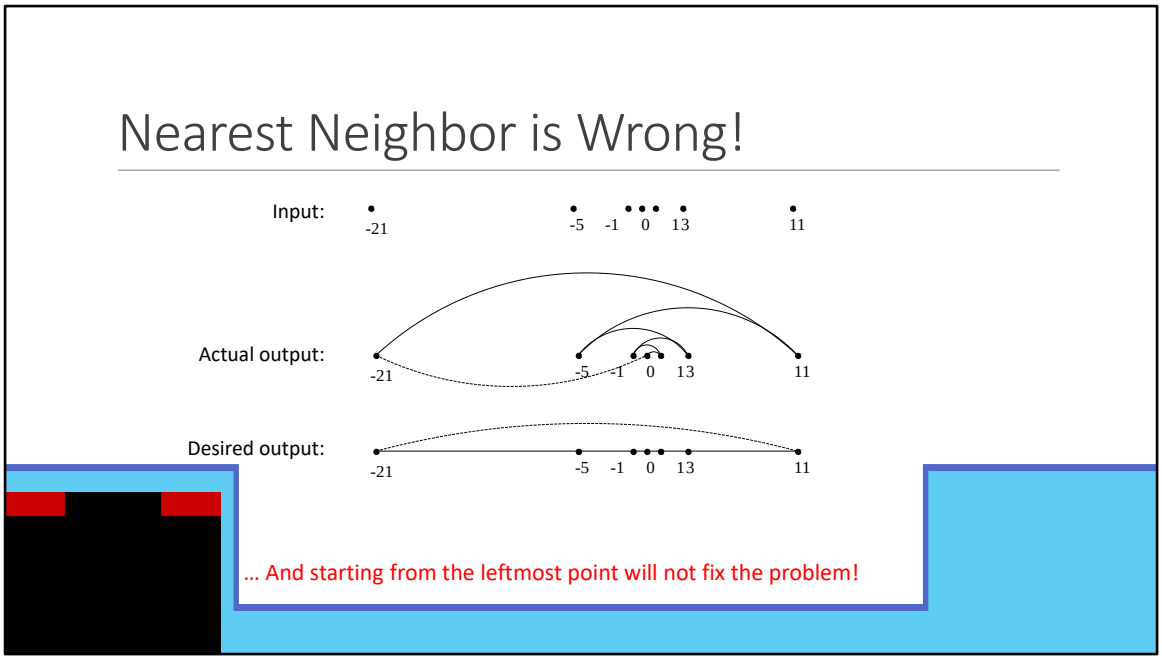
Bart: Specificity really matters when writing good algorithms and problems. You can't just say things like "best tour", you have to be talking concretely about the properties of the inputs and outputs, saying things like "shortest tour".
Bart: We also want to leverage our common vocabulary as much as possible. Being able to say things precisely like "sets" or "sequences" or "permutations" gives a lot of really critical information that changes the nature of the problem.
Bart: We love being pedantic in this course.
ATB: Yes, you need to be technically correct. The best kind of correct.
Bart: Exactly. And for my last point, I want to reiterate that you can't just say "solve the problem", because you're just pushing the problem over to someone else who has to solve that problem.
Bart: We can reuse algorithms in other algorithms; you will often want to start by sorting a set or end by finding the last element of a sequence. But generally, try to refer to only well-known algorithms or algorithms that we have recently defined.

Potential Algorithm: Nearest Neighbor

A popular solution starts at some point p_0 and then walks to its nearest neighbor p_1 first, then repeats from p_1, etc. until done.

```
Pick and visit an initial point p0
p = p0
i = 0
While there are still unvisited points:
       i = i + 1
       Let pi be the closest unvisited point to pi-1
       Visit pi
Return to p0 from pi
```

Bart: Let's try another potential algorithm.
Bart: This one starts at some point P zero, and then walks to its nearest neighbor P one. Then, we repeat to get to P2, and so on and so on.
Bart: The pseudocode shown here gives the algorithm with a bit more detail. Notice that in some places, my pseudocode looks a lot like code.
Bart: Everyone has their own pseudocode style, and you have to learn to read other people's. If they use a term you don't understand in the context, like "visit", then you should ask them what they mean.
ATB: Speaking of which, what do you mean by "Visit" in this context?
Bart: Ah, good question. We'll explain more about that algorithm later, but basically it just means "add that point to a set of points that are labeled visited", so we know which ones we have been to before.
Bart: Now, this is a potential algorithm – take a minute and ask yourself if you think it is correct.
ATB: It seems very reasonable to me. This definitely hits every point. I think it is correct.



Bart: In fact, the algorithm is not correct!
Bart: Consider this instance that I have here. For the input, I put a bunch of numbers in a line. Critically, I set P zero to be the middle element.
Bart: Then, I put all the rest of the numbers on the left and right sides, alternating. The distance between each point is slowly increasing.
Bart: If you look at the Actual Output of the algorithm, you'll see that this forces the algorithm to repeatedly jump back around the center point to find the "next closest" point.
Bart: But the expected output for this problem is a straight line through all of the points, with one long return trip.
ATB: Oh, but what if we just set P zero to be the leftmost point? You didn't think of that, did you, smart guy?
Bart: First of all, I did think of that. The algorithm has you choosing P zero randomly, so there's always a possibility that you would start from the middle.
Bart: But second, it doesn't matter if you start from the leftmost point. If you added the rule that the algorithm always picks the leftmost point, then consider what would happens
Bart: if I rotate this by 90 degrees and move the middle point slightly to the left. I can always force you to pick a bad starting point with this potential algorithm.

ATB: That feels unfair. Just because I change my algorithm doesn't mean you get to change your instance.
Bart: Actually, that's exactly how it should work ATB. It may not feel fair, but I'm afraid that we have to be harsh when it comes to algorithms!

Potential Algorithm: Closest Pair

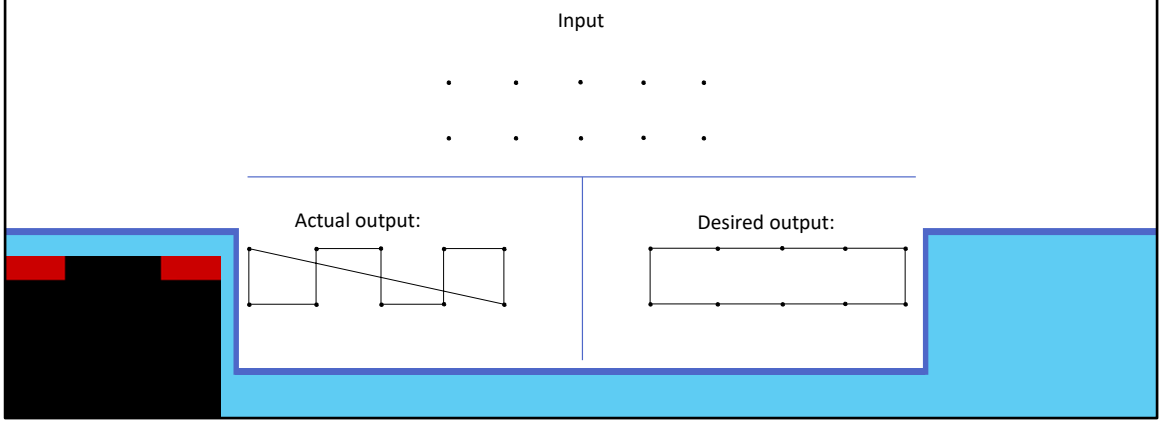
Another idea is to repeatedly connect the closest pair of points whose connection will not cause a cycle or a three-way branch, until all points are in one tour.

```
n = the number of points in the set
distance_min = ∞
for i = 1 to n-1 do:
       for each pair of endpoints (l, r) of partial paths:
              if distance(l, r) <= distance_min:
                     l_min, r_min , distance_min= l, r, distance
       Connect (l_min, r_min) with an edge
Connect the two endpoints by an edge
```

Bart: Let's try a different potential algorithm.
Bart: Here, we are going to repeatedly connect the closest pair of points whose connection will not cause a cycle or a three-way branch, until all points are in one tour.
Bart: When I say cycle, I mean a line of dots that are already all connected back to each other.
Bart: Basically, we're just repeatedly finding dots to pair up, to iteratively build up our final solution.
Bart: Much more sophisticated, but is it correct?

Closest Pair is wrong!

Although it does work on the previous input, it fails on other data!



Bart: The answer is no. It solves the previous instance, but since we changed the algorithm, we're allowed to change the instance too.
Bart: In this case, we now have a grid of dots spaced out like the input shown above.
Bart: The desired output would just be a rectangle connecting all the dots.
Bart: Unfortunately, the algorithm repeatedly pairs up dots first vertically, and then horizontally. Eventually, it has to connect the topleft and bottomright points, which is super expensive.
Bart: The algorithm missed out on the opportunity to get the long-term savings because it found earlier connections that paid off better.
Bart: It turns out that this problem is actually quite difficult!

Correct Algorithm: Exhaustive Search

Try every possible ordering and just select the minimal one.

```
distance_min = ∞
for each permutation of the N points ordering:
      if length of the permutation ≤ distance_min:
            distance_min = length of the permutation
            best_path = current permutation
Return best_path
```

Since all possible orderings are considered, we are guaranteed to end up with the shortest possible tour.

Bart: In fact, a correct algorithm requires us to exhaustively search all possible orderings.
Bart: We try every permutation of the sequence of N points, and then just find the one that has the shortest distance.
ATB: Wow, that seems simple. But wouldn't that take a while if you had a lot of points?
Bart: Exactly! Finding the permutations of all points is actually super inefficient even for a small number of points, as we will see in a later lesson.
Bart: But, since we consider every possible ordering, we are guaranteed to end up with the shortest possible tour.
Bart: So, at the very least, we have a correct algorithm.

Counterexamples

Counterexamples: *cases where an algorithm fails*

Easy way to prove **incorrectness**

Strategies

1. Simples: Try all the small N examples
2. Ties: Think about examples with ties on your decision criteria
3. Extremes: Think about extreme examples where the values are BIG or SMALL

Bart: What I've just demonstrated here is finding counterexamples.
Bart: Counter examples are instances or cases of a problem where an algorithm produces the wrong output.
Bart: They're one of the best ways to prove incorrectness.
Bart: Notice that I'm not trying to prove correctness, because that's actually much harder.
Bart: I won't often ask you to prove correctness in this course, but I will ask you to read proofs of correctness and understand them.
Bart: And I do expect you to find counterexamples for incorrect algorithms.
Bart: You should also find potential counter-examples for your own algorithms to try and disprove them.
ATB: Yes, you should hate your algorithms. They are all terrible.
Bart: No, ATB, they don't have to hate their algorithms, and they're not terrible. But it's a healthy thing to try to break them.
Bart: In order to do so, I have three general strategies.
Bart: First, start by considering simple instances of the problem, where the size of the input is very small. Don't jump straight to big inputs, because those are harder to think about.
Bart: Second, consider cases where you can force the algorithm to make a tie. Look at

the decisions the algorithms make, and what inputs trigger those decisions. Often, inputs that cause ties let you force the algorithm down a bad path.
Bart: Third, consider EXTREME examples, where the values in the input are very BIG or very SMALL. This doesn't mean bigger input sizes, but big values WITHIN the instance.
Bart: For example, if you were sorting three numbers, choose really small or really big numbers. They may just be three numbers, but they can be however big you want.

Today's Activity

Given problems, specify instances

Given algorithms, prove incorrectness by finding instances

Side-goal: work with classmates and learn collaboratively!

Bart: In today's activity, we're going to ask you to identify problems' instances and find counterexamples for algorithms that we give you.
Bart: Another goal will be to have you work together. This is a pretty tricky activity, you have to really think critically.
Bart: Make sure you listen to each other and be ready to try out lots of ideas.
ATB: But remember, all ideas are terrible except for your own.
Bart: That's exactly wrong ATB, you have to respect other people's ideas. You are a bad groupmate.
AT: See, Dr. Bart just said something that proves his ideas are terrible. I am a great group mate and he is a bad groupmate.