Proof Techniques – Contradiction

- for the maintain the invariant step –
  - want to show: if the invariant holds after *k* iterations, it still holds after *k*+1 iterations
  - strategy – proof by contradiction
    - assume the invariant holds after *k* iterations
    - assume this is where things go wrong – the invariant *doesn't* hold after *k*+1 iterations
    - deduce that something you know is true isn't (contradiction)

Invariants by Contradiction – Example

```
max ← A[0]
for i = 1 to n-1 do
  if A[i] > max then
    max ← A[i]
return max
```

loop invariant –
after *i* iterations (i.e. at the start of iteration *i*+1), `max` is the largest element in `A[0..i]`

---

maintain the invariant –

- assume that after iteration *k* (at the start of iteration *k*+1) `max` is the largest element in `A[0..k]`
- assume, for contradiction, that after iteration *k*+1 (i.e. at the start of iteration *k*+2) `max` is *not* the largest element in `A[0..k+1]`
- ...what happened?
  - either the largest element in `A[0..k+1]` is in `A[0..k]` or it is `A[k+1]`
  - it cannot be in `A[0..k]` because the loop invariant gives us that `max` is the largest in `A[0..k]` and the loop doesn't update `max` if `A[k+1]` is smaller
  - it cannot be `A[k+1]` because the loop would have updated `max`
- ...so the assumption that this is where the algorithm fails to maintain the invariant must be false and the invariant holds

How to Design Greedy Algorithms

- establish the problem
  - for optimization problems, identify "legal solution" separate from "optimal solution"
- identify avenues of attack
  - patterns – iterative patterns + series-of-choices interpretation
    - what the decision is about – next input item (process input) or next output item (produce output)
  - flavors
    - type of decision (picking a subset, ordering, labeling, …)
  - greedy choice – by what criteria can we pick an alternative?
    - goal is to identify possible options for the greedy choice
  - counterexamples – rule out incorrect greedy choices
    - goal is to narrow down the possibilities for the correct greedy choice (if there is one)
- define the algorithm
  - iterative algorithm steps – main steps: while not done, make the next choice
- show termination and correctness
  - loop invariant patterns – we haven't gone wrong yet...
    - for optimization problems – ...by staying ahead
    - in general – ...by having a correct solution so far
  - commonly use proof by contradiction for the "maintain the invariant" step

Non-Optimization Greedy

- e.g. assigning people to groups so as to avoid conflicts
- goal is to find a legal solution (according to some criteria for legality)
- greedy choice is to pick any legal alternative
  - finding a counterexample means greedy isn't viable for this algorithm
- loop invariant follows typical iterative patterns
  - *after k iterations, the solution so far is legal*
- direct proof may be possible for maintaining the invariant, or use proof by contradiction
  - must show both that the resulting solution so far is legal and that there is at least one legal alternative to pick

Loop Invariants for Greedy Optimization

- need to address both *legality* and *optimality*
- the legality part is typically a direct claim, following the pattern for iterative algorithms
  - *after k iterations, the solution so far is legal*

Loop Invariants for Greedy Optimization

- need to address both *legality optimality*
- for the optimality part, a outright claim of optimality is often not strong enough to be able to show maintaining the invariant
  - *after k iterations, the solution so far is the best*
- instead, a staying ahead argument is often employed
  - *after k iterations, the solution so far is at least as good as any optimal solution*
- when the optimization quantity is the same as the measure of progress, an indirect claim is needed
  - *after k iterations, some other property of the solution so far is at least as good as that property for any optimal solution*
- with a staying ahead invariant, the "final answer" step must then show that why that property being at least as good when the loop exits means that the overall solution is optimal

Boston to Seattle

You're planning to drive from Boston to Seattle on I-90 this summer, and have a GPS programmed with the locations of gas stations along the way. Assuming that your car can go 400 miles on a tank of gas, determine where you should stop for gas in order to make as few stops as possible.

You're planning to drive from Boston to Seattle on I-90 this summer, and have a GPS programmed with the locations of gas stations along the way. Assuming that your car can go 400 miles on a tank of gas, determine where you should stop for gas in order to make as few stops as possible.

- series-of-choices patterns and flavors

task	iterative pattern	choice
subset	process input	include element in the solution or not
	produce output	which element to include in the solution next
ordering	process input	where to add the element into the solution-so-far
	produce output	which element to append to the solution-so-far
labeling	process input	the label to give the element

What will be the choice in the series of choices?

include the element in the solution or not
which element to include in the solution next
where to add the element into the solution-so-far
which element to append to the solution-so-far
the label to give the element
whether to stop at the current gas station
which gas station to stop at next
where the current gas station goes in the list of stops
which gas station to add to the end of the list of stops
assign "stop" or "don't stop" to the station

"gas stations to stop at" is a subset task

there is an ordering to the gas stations along the route, but this is dictated – there's only one possible ordering of a particular set of stops

subset wording, process input
subset wording, produce output
ordering wording
labeling wording