A Series of Choices

- divide-and-conquer works by dividing the task into independent subproblems which are solved separately
- an alternative is to build up a solution incrementally by making a series of choices

Given a collection of events with start time $s(i)$ and finish time $f(i)$ $(0 \leq s(i) \leq f(i))$, find the largest set of non-overlapping events.

```
initialize an empty set of events
repeatedly
  select a non-overlapping event
until no non-overlapping events remain
```


Greedy Algorithms

- iterative
- always make a *local* decision
  - each choice is made without consideration of future possibilities
- often, but not exclusively, applied to optimization problems
  - goal is to find the best solution among (generally) many legal solutions
  - for non-optimization problems, goal is to find a legal solution among (generally) many invalid (non-)solutions
- don't work for everything – requires
  - *greedy choice property* – that a globally optimal/legal solution can be found by making local choices
  - *optimal substructure property* – that an optimal/legal solution can be constructed from optimal/legal solutions of subproblems

a correctness proof is essential!

15.5 Steps to Greedy Success

establishing the problem

1. specifications
2. examples

brainstorming ideas

3. targets
4. tactics
5. approaches

5.5 greedy choice

- greedy strategies
- counterexamples

defining the algorithm

6. main steps
7. exit condition
8. setup
9. wrapup
10. special cases
11. algorithm

showing correctness

12. termination
    - measure of progress
    - making progress
    - reaching the end
13. correctness
    - loop invariant
    - establish the loop invariant
    - maintain the loop invariant
    - final answer

determining efficiency

14. implementation
15. time and space

Approaches

Some common flavors of tasks –

- select a subset of the input items
- order the input items
- assign labels to the input items

task	iterative pattern	main steps structure	**choice**
"select a subset"	process input	for each element, decide whether to include in the solution or not include in the solution	include or not?
	produce output	repeatedly find elements to include in the solution	which element is next?
"order the input items"	process input	for each element, add to the solution in the proper order	where does the element go in the ordering?
	produce output	repeatedly find the next element in the ordered solution	which is the next element in the ordering?
"assign labels"	process input	for each element, determine its label	what is the label?

if none of these flavors is applicable, fall back on the basic iterative patterns – process input and produce output

Greedy Choice

5.5 *Greedy choice.*
The main steps involve repeatedly making a local decision. On what basis is that choice made?

(a) *Greedy strategies.* Identify plausible greedy strategies for making each choice.

(b) *Counterexamples.* Narrow down the candidates by looking for counterexamples to eliminate plausible-but-incorrect greedy strategies.

- each iteration makes the best choice available at the time, using only past or current information
- cannot factor in future possibilities! (or undo past choices in light of current information)

- the challenge of greedy algorithms is typically showing that a particular greedy choice leads to the desired solution
  - look for counterexamples to quickly eliminate plausible but incorrect greedy choices

Loop Invariant

For optimization problems, address both *legality* and *optimality*. Two common patterns for the optimality part –

- *staying ahead* – our algorithm's partial solution after *k* steps is at least as good as any optimal solution after *k* steps
  - need a specific measure in order to define "at least as good as"
  - measure may be more closely connected to the greedy choice criterion than the quantity being optimized
- *we haven't gone wrong yet* – our algorithm's partial solution is still consistent with an optimal solution for the whole problem

Establishing and maintaining –

- establish using both *k*=0 and *k*=1 instead, as *k*=0 is too trivial by itself (it doesn't make use of the greedy choice)
- proof by contradiction is often effective for maintaining