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
```


Common Elements – Approaches and Flavors

- as with iterative algorithms, there are two main approaches to the series of choices

- **process input**, where the choice or decision is about what to do with each input element in turn
- **produce output**, where the choice or decision is about what what the next output element is

- there are also several common types of tasks which lead to slightly different flavors for the algorithm components

- a **subset**, where the task to is select a subset of the input items subject to some subset membership constraint
- a **sequence**, where the task is to produce an ordering of all or a subset of the input items
- a **labelling**, where the task is to assign labels to the input items

  - these can overlap – pick what best aligns with the primary task
  - not an exhaustive list

Paradigms

- how many alternatives need to be considered for each decision leads to fundamentally different algorithmic paradigms

- If only **one** alternative needs to be considered, the formulation can be iterative. The key focus for the algorithm is determining how to pick that right alternative for each decision, and showing that the series of choices made leads to a correct solution. Greedy algorithms are of this type and will be considered in chapter 7.
- If **more than one** alternative needs to be considered, the formulation is typically recursive. The key focus for the algorithm is how to avoid an exponential blowup in running time. Backtracking, branch-and-bound, and dynamic programming algorithms are of this type and will be considered in chapters 8 to 10.

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!
  - finding counterexamples is an important technique for identifying incorrect greedy choices

Proof Techniques – Incorrectness

One counterexample is all that is needed to prove an algorithm incorrect.

Properties of a good counterexample.

- simple, which often means small
- verifiable – need to be able to compute the algorithm's output and give a better answer

Strategies.

- think exhaustively – can often enumerate all possible inputs of a small size
- hunt for weakness – look for a case where the algorithm's choice is the wrong thing to do
- try inputs with duplicates or ties, as that neutralizes the algorithm's choice
- seek extremes rather than uniformity

Counterexamples

Find a counterexample to prove the following statement false:

`a + b >= min(a,b)`

- both numbers negative e.g. -5, -2
  - -5 + -2 = -7 ≥ min(-5,-2) = -5 → false

Counterexamples

1-2. [3] Show that $a \times b$ can be less than $\min(a, b)$.

1-3. [5] Design/draw a road network with two points a and b such that the fastest route between a and b is not the shortest route.

1-4. [5] Design/draw a road network with two points a and b such that the shortest route between a and b is not the route with the fewest turns.

1-5. [4] The subset sum problem is as follows: given a set of integers $S = \{s_1, s_2, \ldots, s_n\}$, and a target number T, find a subset of S that adds up exactly to T. For example, there exists a subset within $S = \{1, 2, 5, 9, 10\}$ that adds up to $T = 22$ but not $T = 23$.

Find counterexamples to each of the following algorithms for the subset sum problem. That is, give an S and T where the algorithm does not find a solution, even though a solution exists.

(a) Pick elements of S in left to right order if they fit,

(b) Pick elements of S from smallest to largest, that is,

(c) Pick elements of S from largest to smallest.

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?
  - counterexamples – rule out incorrect greedy choices
- define the algorithm
  - iterative algorithm steps
- show termination and correctness
  - loop invariant patterns
    - for optimization problems – staying ahead
    - in general – we haven't gone wrong yet
  - commonly use proof by contradiction for the "maintain the invariant" step
- determine efficiency

Proof Techniques – Contradiction

- assume that what you want to prove is false
- develop logical consequences from this assumption, until you get to one that is demonstrably false
- since there were no flaws in the deduction, the assumption that what you want to prove is false must have been faulty and thus what you want to prove is true