- Why should we study greedy algorithm? Because for many optimization problems, using dynamic programming to determine the best choices is overkill.
- Like dynamic-programming algorithms, greedy algorithms typically apply to optimization problems in which we make a set of choices in order to arrive at an optimal solution.
- The idea of a greedy algorithm is to make each choice in a locally optimal manner.
- P414 A **greedy algorithm** always makes the choice that looks best at the moment. That is, it makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution.
- P414 We shall arrive at the greedy algorithm by first considering a dynamic-programming approach and then showing that we can always make greedy choices to arrive at an optimal solution.

16.1 An activity-selection problem

- P415 This problem is the problem of scheduling several competing activities that require exclusive use of a common resource, with a goal of selecting a maximum-size set of mutually compatible activities.
- The activities are sorted in monotonically increasing order of finish time.
- Read P415 circled paragraph
- P416 circled paragraph
- The cut-and-paste method is to show optimal substructure of a problem
- P417 Intuition suggests that we should choose an activity that leaves the resource available for as many other activities as possible. Now, of the activities we end up choosing, one of them must be the first one to finish. Our intuition tells us, therefore, to choose the activity in S with the earliest finish time, since that would leave the resource available for as many of the activities that follow it as possible.
- In other words, since **the activities are sorted in monotonically increasing order by finish time**, the greedy choice is a1.
- P417 circled paragraph
- P418 Theorem 16.1
- A greedy algorithm doesn't need to work bottom-up, like a table-based dynamic-programming algorithm. Instead, it can work top-down, choosing an activity to put into the optimal solution and then solving the subproblem of choosing activities from those that are compatible with those already chosen.
- Greedy algorithms typically have this top-down design: make a choice and then solve a subproblem, rather than the bottom-up technique of solving subproblems before making a choice.
- P419 RECURSIVE-ACTIVITY-SELECTOR -> running time $\theta(n)$ because over all recursive calls, each activity is examined exactly once in the while loop.
- P419 circled paragraph
- The procedure GREEDY-ACTIVITY-SELECTOR is an iterative version of the procedure RECURSIVE-ACTIVITY-SELECTOR.
- P421 GREEDY-ACTIVITY-SELECTOR -> running time also $\theta(n)$
- P421 circled paragraph

16.2

- P423 More generally, we design greedy algorithms according to the following sequence of steps:
  1. Cast the optimization problem as one in which we make a choice and **are left with only one subproblem to solve**.
  2. Prove that there is always an optimal solution to the original problem that makes the greedy choice, so that the greedy choice is always safe.
  3. **Demonstrate optimal** substructure by showing that, having made the greedy choice, what remains is a subproblem with the property that if we combine the optimal solution to the subproblem with the greedy choice we have made, we arrive at an optimal solution to the original problem.

  We see that beneath every greedy algorithm, there is almost always a more cumbersome dynamic-programming solution.

- P424 If we can show that the problem has both: 1. Greedy-choice property 2. Optimal substructure, we are well on the way to developing a greedy algorithm for it.
- P424 Greedy-choice property: when we are considering which choice to make, we make the choice that looks best in the current problem, without considering results from subproblems.
- P424 circled paragraph
- Unlike dynamic programming, which solves the subproblem before making the first choice, a greedy algorithm makes its first choice before solving any subproblems.
- P425 Greedy vs. dynamic programming (example: 0-1 knapsack problem vs. fractional knapsack problem)
- Although the problems are similar, we can solve the fractional knapsack problem by a greedy strategy, but we cannot solve the 0-1 problem by such a strategy.
- P426 circled paragraph
- Fractional knapsack problem: compute & sort the value per pound for each item. Start taking item with greatest value per pound, then the next greatest value per pound, etc until space is full
- In the 0-1 problem, when we consider whether to include an item in the knapsack, we must compare the solution to the subproblem that includes the item with the solution to the subproblem that excludes the item before we can make the choice.