HW 10

- give the steps of the backtracking process from class, not just a statement of the algorithm or a narrative account of your reasoning process
  - establish the problem
    - specifications
    - examples
  - identify avenues of attack
    - targets
    - paradigms and patterns
    - the series of choices
  - define the algorithm
    - size
    - generalize / define subproblems: partial solution, alternatives, subproblem
    - base case(s)
    - main case
    - top level: initial subproblem, setup, wrapup
    - special cases
    - algorithm
  - show termination and correctness
    - termination: making progress, the end is reached
    - correctness: establish the base case, show the main case, final answer
  - determine efficiency
    - implementation
    - time and space
    - room for improvement
- only #3, #4 were graded (or a different problem if you didn't hand in #3, #4)
  - review those comments and consider their application to the other problems

HW 10

- there are different ways to structure elements of backtracking algorithms – be careful to be consistent with your choices
  - the subproblem solution can be either a complete solution (including the partial solution) or just the solution for the rest of the problem (not including the partial solution)
    - the framework and examples discussed in class use the former (complete solution) for backtracking and the latter (rest of the problem) for dynamic programming
  - for find-the-best-solution tasks, either return the best solution or update a global best-so-far
    - return the best: base case returns the complete solution it got, main case maintains a best-so-far solution from amongst the subproblems it generates and returns that
    - update global: base case checks its complete solution against the global best so far and updates accordingly, nothing is returned

HW 10

- for any algorithm, highly detailed pseudocode is not helpful in understanding the concept and its correctness
  - words are good!
  - introduce notation only when it is clearer than words
  - especially stick with words when notation forces you into unnecessary implementation decisions
  - don't be tempted to overload notation with additional meanings
  - e.g.
    - good: "remove contractor *c* from the available contractors" or "mark contractor *c* as unavailable" or "`available.remove(c)`"
      - all convey that *c* is no longer available
    - confusing: `unavailable ← c`
      - ← is a common convention in pseudocode for assignment, so using it to mean adding *c* to a collection is confusing

HW 10

- pruning
  - pruning if the partial solution cost > best-so-far cost is safe (and cheap)...
    - ...but any partial solution cost will generally be lower than a complete solution cost because there are fewer jobs with contractors assigned in a partial solution (unassigned jobs have a cost of 0)
  - pruning based on some estimate of the cost of the rest of the solution is branch and bound

HW 10

- bound functions
  - be aware of running time – the bound function must be evaluated for every subproblem even if no pruning occurs
    - global best value × number of choices left is O(1) to compute
    - summing something over the remaining choices is $\Omega(n\text{-}k)$ (k choices made, $n\text{-}k$ choices left)
      - $O(n\text{-}k)$ if the value being summed can be determined in O(1) for each choice
  - there is often a tradeoff between pruning quality and speed – find the best balance
    - global best value × number of choices left is O(1) to compute – but may not be very close to the actual solution cost
  - possible $\Omega(n\text{-}k)$ strategies
    - choose the best alternative for each of the remaining choices ignoring legality
      - safe – no legal alternative can be better than the best possible one
    - choose the best legal alternative (based on the current partial solution) for each of the remaining choices
      - safe – later alternatives, when fewer alternatives are available, can't be better than the best alternative now
    - greedy solution
      - not safe – picking a good alternative now may force a worse alternative later