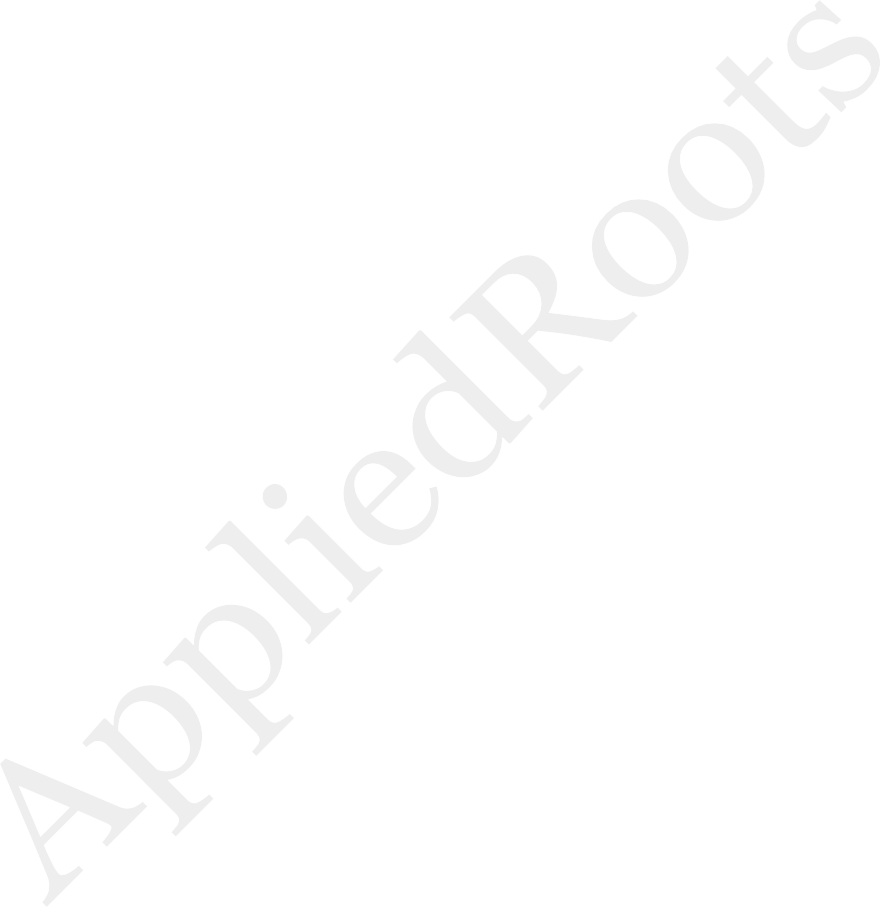
Greedy Algorithms

CLRS 16.2

# Greedy Algorithms

* + We have previously discussed how to speed up optimization problems using the technique of dynamic programming:
    - The problem must have the optimal substructure property: the optimal solution to the problem contains within it optimal solutions to smaller subproblems.
    - Typically the number of different subproblems is polynomial, but the recursive algorithm implementing the recursive formulation of the problem is exponential. This is because of overlapping calls to same subproblem.
    - Idea: If same subproblem is solved several times, use table to store result of a subproblem the first time it is computed and never compute it again.
    - Alternatively, we can think about filling up a table of subproblem solutions from the bottom.
  + Another, simpler and often less powerful (and less well-defined), technique that uses the same feature of optimal substructure is greediness.
  + Like in the case of dynamic programming, we will introduce greedy algorithms via an example.

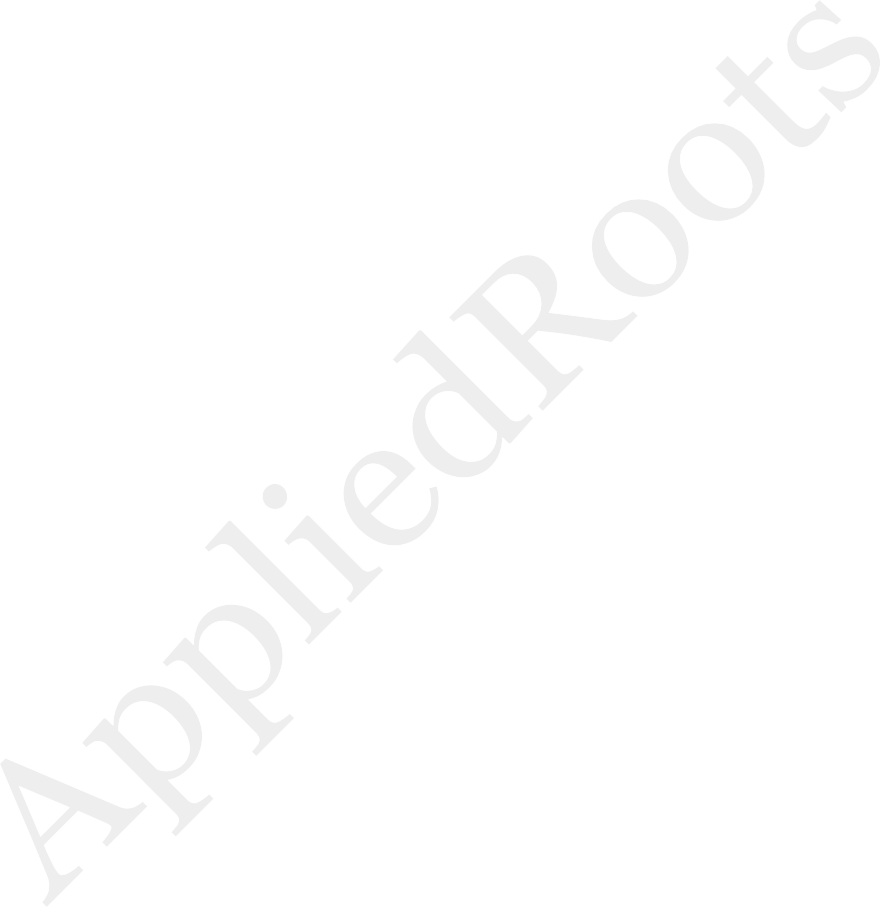
# Comparison of greedy technique with dynamic programming

* + Both techniques use optimal substructure (optimal solution “contains optimal solution for subproblems within it”).
  + In dynamic programming, solution depends on solution to subproblems.That is, compute the optimal solutions for each possible choice and thencompute the optimal way to combine things together.
  + In greedy algorithm we choose what looks like best solution at any given moment and recurse (choice does not depend on solution to subproblems).

Note: Shortsightedness: Always go for seemingly next best thing, optimizing the present without regard for the future, and never change past choices.

* + Any problem that can be solved by a greedy algorithm can be solved by dynamic programming, but not the other way around.
  + It is often hard to figure out when being greedy works!
  + How do we know if being greedy works? Try dynamic programming first and understand the choices. Try to find out if there is a locally best choice, i.e. a choice that looks better than the others (without computing recursive solutions to subproblems). Now try to prove that it works correctly.
  + Greedy correctness proof: It is enough to prove that there exists an optimal solution which contains the greedy choice. That is, prove 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 get an optimal solution for the original problem. Typically this is proved by contradiction.

Example:

* + 0 − 1 knapsack problem: Given n items, with item i being worth $ vi and having weight wi pounds, fill knapsack of capacity w pounds with maximal value.
  + Fractional knapsack problem: As 0−1 knapsack problem but we can take fractions of items.
  + Problems appear very similar, but only fractional knapsack problem can be solved greedily:

vi

* + - Compute value per pound wi for each item – Sort items by value per pound.
    - Fill knapsack greedily (take objects in order) 𝖴

O(nlogn) time, easy to show that solution is optimal.

* + Example that 0 − 1 knapsack problem cannot be solved greedily:

## $60=

$100

20

## $120

30 $220=

## $160

20

|  |
| --- |
| 30 |
| 20 |

1010

|  |  |  |
| --- | --- | --- |
| Items in value  per pound order | Optimal solution  for knapsack of size 50 | Greedy solution  for knapsack of size 50 |

Note: In fractional knapsack problem we can take  of $120 object and get $240 solution.

* + 0 − 1 knapsack problem can be solved in time O(n · w) using dynamic-programming (homework).