More Dynamic Programming: Knapsack, Optimal Binary Trees

- Read section 8.2 (pages 283-289)
- Describe the two-dimensional dynamic programming solution to the Knapsack problem.
  - What is the recurrence relation?
  - What are the initial conditions?
- Why is a bottom-up implementation of this programming solution not optimal?
- How do **memory functions** help with a top-down implementation that still benefits from the dynamic programming approach?
- Create a small instance of the knapsack problem and solve it using this algorithm.
- Practice problems: 8.2.4, 8.2.8
- Read section 8.3 (pages 297-302)
  - In this section we are looking for **optimal binary search trees**. In what way are these trees *optimal*?
  - What does $C(i, j)$ represent, and what is the basic recurrence relation for it? How does this relation come about?
  - What should $C(i, i)$ be? What should $C(i, j)$ be for $j < i$?
  - Describe how we fill the entries in the dynamic programming table, for this problem.
- We want to store the letters A, B, C, D, E, F in an optimal binary search tree. They have frequencies 0.2, 0.15, 0.1, 0.2, 0.1, 0.25. What is the optimal binary search tree for these letters?
- Practice problems: 8.3.5, 8.3.7