The Knapsack Problem

COMP 215 Lecture 8

Greedy Algorithms vs. Dynamic Programming

- Both types of algorithms are generally applied to optimization problems.
- Greedy algorithms tend to be faster.
- A greedy algorithm requires two preconditions:
  - Greedy choice property - making a greedy choice never precludes an optimal solution.
  - Optimal substructure property – an optimal solution to the problem contains optimal solutions to the subproblems.
- If we have the second property then we can develop a DP algorithm.
- If we have neither property then we are in tough shape.

Knapsack Problem

- 0-1 Knapsack Problem...
- Fractional Knapsack Problem...
- Greedy algorithm...

Greedy Knapsack Proof Preview

- Greedy choice property:
  - We need to show that our first greedy choice g_1 is included in some optimal solution O.
- Optimal substructure property:
  - We need to show that O-$\{g_1\}$ is a solution to the problem left over after we make our first greedy choice.

Proof adapted from Vicki Choy's lecture notes at Viginia Tech: http://people.cs.vt.edu/~vchoi/4104/

Greedy Choice Property

- Let $O = \{o_1, o_2, \ldots, o_j\} \subseteq I$ be the optimal solution of problem P.
- Let $G = \{g_1, g_2, \ldots, g_k\} \subseteq I$ be the greedy solution, where the items are ordered according to the greedy choices.
- We need to show that there exists some optimal solution O' that includes the choice g_1.
- CASE 1: g_1 is non-fractional.
  - if g_1 is included in O, then we are done.
  - if g_1 is not included in O then we arbitrarily remove w_{g1} worth of stuff from O and replace it with g_1 to produce O'.
  - O' is a solution, and it is at least as good as O.

Proof Continued...

- CASE 2: g_1 is fractional. (this means $K = f * w_{g1}$ where f is the fraction of g_1 chosen. K is the weight limit.)
  - if O includes $f * w_{g1}$ units of g_1, then we are done.
  - if O includes less than f of g_1, then we remove $f * w_{g1}$ weight from O arbitrarily and replace it with $f * w_{g1}$ units of g_1 to construct O'.
  - O' is a valid solution, and at least as good as O.

Optimal Substructure Proof

- We have shown that there is an optimal solution O' that selects g_1.
- After g_1 is chosen the weight limit becomes $K'' = K - w_{g1,}$ the item set becomes $I'' = I - \{g_1\}$.
- Let P'' be the knapsack problem such that the weight limit is K'' and the item set is I''. We need to show that $O'' = O' - \{g_1\}$ is an optimal solution to P''.
- Proof is by contradiction. Assume that O'' is not a solution to P''. Let Q be an optimal solution that is more valuable than O''.
- Let $R = Q \cup \{g_1\}$. The value of O' is equal to the value of $O'' + g_1$.
- The value of R is greater than the value of $O' = O'' + g_1$.
- Since O' was an optimal solution, this is a contradiction.

Optimal Substructure in the 0-1 Knapsack Problem

- Let O be an optimal subset of all n items with weight limit K.
- We want to show that O contains a solution to all sub-instances (by induction).
  - CASE 1: If O does not contain item n, then it is clearly an optimal subset of the first n-1 items.
  - CASE 2: If O does contain item n, then O-{n} is a solution to the problem instance that includes the first n-1 items, and a weight limit K-w_n.
    - Proof by contradiction, if O – {n} is not a solution, then there must be some other subset Q with higher profit. By adding the nth item to Q, we have a subset of the first n items with higher profit than O, a contradiction.

Dynamic Programming for 0-1 Knapsck

- First, why can't we have a greedy algorithm for this problem? Let's look back at the fractional proof.
- The reasoning on the previous slide leads us to the following recurrence for maximum profit *P* on the first *i* items with a weight limit of *w*:

CASE 1 CASE 2

$$P[i][w] = \begin{cases} \max\left(P[i-1][w],\ p_i + P[i-1][w-w_i]\right) & \text{if } w_i \le w \\ P[i-1][w] & \text{if } w_i > w \end{cases}$$

- $P[i][0] = 0$ and $P[0][w] = 0$.
- Let's write the algorithm!

Analysis

- Running time is in $\Theta(nW)$.
- Is that good?
- Room for improvement?

A Better Algorithm

- We get a more efficient algorithm by only computing necessary entries.
  - The *n*th row requires at most one entry: $P[n][W]$.
  - the *n*-1st row requires at most two entries $P[n-1][W]$ and $P[n-1][W-w_n]$.
  - the *n*-2nd row requires at most four entries, two for each in the *n*-1st row.
  - $1 + 2 + 2^2 + ... + 2^{n-1} = 2^n - 1$
- So regardless of weight limit, this algorithm gives a bound of $\Theta(2^n)$.
- Incorporating the weight limit, we have $O(\min(nW, 2^n))$.