CS 31: Algorithms (Spring 2019): Lecture 7

Date: 10th April, 2019

Topic: Dynamic Programming 2: The Knapsack Problem

Disclaimer: These notes have not gone through scrutiny and in all probability contain errors. Please notify errors on Piazza/by email to deeparnab@dartmouth.edu.

---

1 Knapsack Problem.

KNAPSACK
Input: n items; item j has profit p_j and weight w_j. A knapsack of capacity B. All of these are positive integers.
Output: Find the subset $S \subseteq \{1, 2, \ldots, n\}$ which maximizes $\sum_{j \in S} p_j$ and "fits" in the knapsack; that is, $\sum_{j \in S} w_j \leq B$.

As in the Subset Sum case, let us look at an instance of Knapsack $((p_1, w_1), \ldots, (p_n, w_n); B)$ and consider an optimal solution $S \subseteq \{1, \ldots, n\}$ for the problem. Can we break this up into solutions to smaller instances of the same problem?

Just like in subset sum, let us ask whether the *last* element n is in the subset or not. If it is not, then S in fact is also the optimal solution for the smaller instance $((p_1, w_1), \ldots, (p_{n-1}, w_{n-1}); B)$. If not, then a better solution for this smaller instance would be a better solution for the original instance.

If the last element is in S, and here is the simple but crucial observation, then $S - n$ must be the optimal solution for the smaller instance $((p_1, w_1), \ldots, (p_{n-1}, w_{n-1}); B - w_n)$. In other words, the placement of the last item only reduces the size of the knapsack available to the other items, but doesn't affect the problem in any other way.

In summary, if the solution S doesn't contain the n item, then S itself is an optimal solution to the smaller instance $((p_1, w_1), \ldots, (p_{n-1}, w_{n-1}); B)$; otherwise, $S - n$ is an optimal solution to the smaller instance $((p_1, w_1), \ldots, (p_{n-1}, w_{n-1}); B - w_n)$. Good, we have made progress – now we will formalize all these things using the 7-step method described last time.

To make things precise, for optimization problems like knapsack (unlike decision problems like subset sum) which try to maximize/minimize something, it helps to spell out the *candidate/feasible* solutions of an instance. To that end, we use the definition

$$\mathsf{Cand}(m, b) : \text{all possible subsets of } \{1,2,\ldots,m\} \text{ whose total weight is } \leq b.$$

We will write a recurrence for $F(m, b)$ which is the maximum profit subset in $\mathsf{Cand}(m, b)$.

1. *Definition:*

   For any $0 \leq m \leq n$ and $0 \leq b \leq B$, let $\mathsf{Cand}(m, b)$ be all subsets $S \subseteq \{1, \ldots, m\}$ which fit in a knapsack of capacity b, that is, $\sum_{j \in S} w_j \leq b$.

Define $F(m, b) = \max_{S \in \mathsf{Cand}(m,b)} \sum_{j \in S} p_j$.

We are interested in $F(n, B)$.

2. *Base Cases:*

   $F(0, b) = 0$ for all $0 \leq b \leq B$; an empty set gives profit 0.

   $F(m, 0) = 0$ for all $0 \leq m \leq n$; an empty set gives profit 0.

3. *Recursive Formulation:*

   For all $m \geq 1, b \geq 1$:

$$F(m, b) = \max\left(F(m-1, b),\ F(m-1, b-w_m) + p_m \right)$$

4. *English Explanation / Formal Proof:*

   Let's consider the optimal way of picking items of type in subset $\{1, 2, \ldots, m\}$.

   Case 1: This optimal set may not contain item m, in which case we should have $F(m, b) = F(m-1, b)$.

   Case 2: This contains item m. In that case, there is a way of picking items from $\{1, 2, \ldots, m-1\}$ giving value $F(m, b) - p_m$ which fit in a knapsack of capacity $b - w_m$.

   Conversely, the optimal solution to $F(m-1, b)$ is also a solution to $F(m, b)$. And, the solution to $F(m-1, b-w_m)$ can be appended with the last item.

   Formal Proof:

   ($\leq$): Let S be the subset in $\mathsf{Cand}(m, b)$ such that $p(S) = F(m, b)$.
   Case 1: S doesn't contain item m. Then $S \in \mathsf{Cand}(m-1, b)$ and so $F(m-1, b) \geq p(S) = F(m, b)$, since $F(m-1, b)$ is the *maximum* over all sets in $\mathsf{Cand}(m-1, b)$.
   Case 2: S contains item m. Then $S \setminus j$ lies in $\mathsf{Cand}(m-1, b-w_m)$ and $p(S \setminus j) = p(S) - p_m = F(m, b) - p_m$. Thus, $F(m-1, b-w_m) \geq F(m, b) - p_m$, since $F(m-1, b-w_m)$ is the *maximum* over all sets in $\mathsf{Cand}(m-1, b-w_m)$.

   ($\geq$): Let S be the subset in $\mathsf{Cand}(m-1, b)$ such that $p(S) := \sum_{i \in S} p_i = F(m-1, b)$. Observe S also lies in $\mathsf{Cand}(m, b)$. Thus, $F(m, b) \geq p(S) = F(m-1, b)$ since $F(m, b)$ is the *maximum* over all sets in $\mathsf{Cand}(m, b)$.
   Similarly, let S be the subset in $\mathsf{Cand}(m-1, b-w_m)$ such that $p(S) = F(m-1, b-w_m)$. Form $S' = S + m$. Note that $S \in \mathsf{Cand}(m, b)$ since $w(S') \leq b$, and $p(S') = F(m-1, b-w_m) + p_m$. Thus, $F(m, b) \geq F(m-1, b-w_m) + p_m$.

5. *Pseudocode for computing $F[n, B]$ and recovery pseudocode:*

```
1: procedure KNAPSACK(B,(p_1, w_1), ··· , (p_n, w_n)):
2:     ▷ Returns the subset of items of type 1, ..., n which fits in knapsack of capacity
   B and gives maximum profit.
3:     Allocate space F[0 : n, 0 : B]
4:     F[0, b] ← 0 for all 0 ≤ b ≤ B ▷ Base Case
5:     F[m, 0] = 0 for all 0 ≤ m ≤ n. ▷ Base Case
6:     for 1 ≤ m ≤ n do:
7:         for 1 ≤ b ≤ B do:
8:             if b − w_m ≥ 0 then :
9:                 F[m, b] ← max(F[m − 1, b], F[m − 1, b − w_m] + p_m)
10:                ▷ Note F[m − 1, b − w_m] is set before F[m, b] in this ordering.
11:            else: ▷ Implicitly, in this case F[m − 1, b − w_m] = −∞
12:                F[m, b] ← F[m − 1, b]
13:    ▷ F[n, B] now contains the value of the optimal subset
14:    ▷ Below we show the recovery pseudocode

15:    m ← n; b ← B; S ← ∅.
16:    ▷ Invariant: ∑_{j∈S} w_j + b ≤ B and F[m, b] + ∑_{j∈S} p_j = F[n, B]
17:    while m > 0 do:
18:        if F[m, b] = F[m − 1, b] then:
19:            m ← m − 1
20:        else: ▷ We know F[m, b] = F[m, b − w_m] + p_m.
21:            S ← S + m
22:            b ← b − w_m.
23:            m ← m − 1
24:    return S
```

Note that in the recovery the invariant always holds and at the end since $F[0, k] = 0$, we have $p(S) = F[n, B]$.

6. *Running time and space* The above pseudocode take $\Theta(nB)$ time and space where n is the number of items.