Knapsack

CS 5510 F23
Abraham Ladha

In some sense, Knapsack is every problem, a great way into our next unit. You are given a capacity W and a set of several items, each with a weight w_i and value v_i. You want to ~~maximize~~ choose a subset of the items such that $\sum v_i$ is maximum for all possible subsets subject to $\sum w_i < W$.

Suppose $W = 6$, and your items have $\boxed{w|v}$ as

$\boxed{1|12}$ $\boxed{2|17}$ $\boxed{3|18}$ $\boxed{4|35}$

Notice selection of item 3 or 4 eliminates the other one. Choosing items 1, 2 has value 29. 2, 3 has value 35. (1, 2, 3) has max value of 97. This problem has a lot of interesting structure! Lets try to solve it with DP. For simplicity, our first variant of knapsack assumes no duplicates.

Solution

array: $dp[0..W][0..n]$ with $dp[i][j]$ the max you can get from items $1..j$ and with capacity i. We will return $dp[W][n]$.

base cases: $\forall i \; dp[i][0] = 0 \quad \forall j \; dp[0,j] = 0$.
These correspond to if you have no items or no capacity.

recurrence: Some item is chosen last. If you don't choose the n^{th} item, then your item set increases but not your solution, so $dp[W][n] = dp[W][n-1]$. If you do pick the item, then you previously had capacity $W - w_n$ and value $dp[W-w_n][n-1]$, so your new value is $dp[W-w_n, n-1] + v_n$. If your next item has $w_n > W$, we can't pick it at all, so

$$dp[W, n] = \begin{cases} dp[W, n-1] & \text{if } w_n > W \\ \max\left(dp[W, n-1], dp[W-w_n, n-1] + v_n\right) & w_n \le W \end{cases}$$

we generalize from dp[W,n] to dp[i,j] as

$$dp[i,j] = \begin{cases} dp[i,j-1] & w_j > i \\ \max\left(dp[i,j-1], dp[i-w_j, j-1] + v_j\right) & w_j \le i \end{cases}$$

implementation:

```
def knapsack01(W, w_1 .. w_n, v_1 ... v_n):
    dp[0..W][0..n] all zeroes
    for i in 0..W:
        dp[i][0] = 0
    for j in range 0..n:
        dp[0][j] = 0
    for j in 1..n:
        for i in 1..W:
            if w_j ≤ i:
                dp[i,j] = max(dp[i,j-1], dp[i-w_j, j-1] + v_j)
            else:
                dp[i,j] = dp[i,j-1]
    return dp[W,n]
```

runtime: $O(Wn)$, but note this is exponential. (cooking polynomial is not polynomial. W is given as input as k bits so $W \approx 2^k$ so the time in terms of the size of the input is really $O(2^k n)$.

Also note our recurrence does care for slack in the capacity, as if it's really an optimal solution but $\sum w_i < W$ and not equal, the solution will propagate forward.

lets illustrate a table

		w_1	w_2	w_3	w_4
		1	2	3	4
0	0	0	0	0	0
1	0	12	12	12	12
2	0	12	17	[17]	17
3	0	12	29	29	29
4	0	12	29	30	30
5	0	12	29	35	37
W= 6	0	12	29	[47]	47

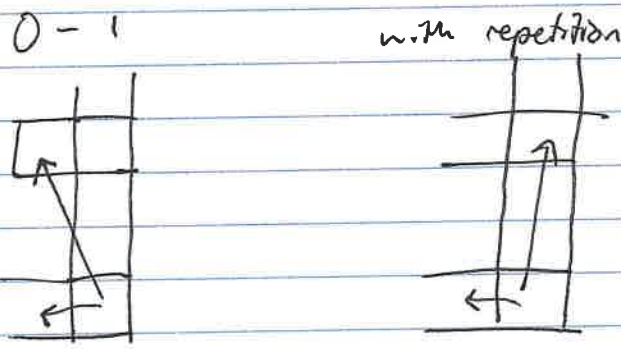
v_1	v_2	v_3	v_4
12	17	18	25

with $dp[6,4] = \max(dp[6,3], dp[6-4,3]+25) = \max(47, 17+25) = 47$

What about knapsack with repetition? In this variant of the problem, you are allowed to choose items an unlimited number of ~~times~~ times. The subproblem should not be in terms of elements $1..j-1$, but in terms of $1..j$. So we replace $j-1$ with j in the recurrence

$$dp[i,j] = \begin{cases} dp[i,j-1] & \text{if } w_j > i \\ \max(dp[i,j-1], dp[i-w_j, j]+v_j) & \text{if } w_j \le i \end{cases}$$

Similar runtime as before with $O(Wn)$. The only difference here is instead of looking in the previous column, it looks above in its own column.



This was still exponential time. For reasons we will see later, we do not think we can improve this. We may be able to use less space though. We give a solution to knapsack which uses $O(W)$ space instead of $O(Wn)$ space, but still $O(Wn)$ time.

solution

array: $dp[0..W]$ with $dp[i]$ = max using all items. will return $dp[W]$

base case $dp[0] = 0$, no capacity, no items.

recurrence

$$dp[i] = \max_{\substack{1 \le j \le n \\ w_j \le i}} \{ dp[i - w_j] + v_j \}$$

basically, just maximize the last item for each possible. similar to min op or min jumps. At each step consider all possible items for a candidate to be last. Each step must consider each item, so computing max takes $\sim n$ steps. For each cell takes $O(W \cdot n)$ steps. Now only $O(W)$ space though.