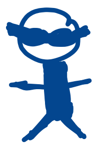
Dynamic Programming [chapter 6]

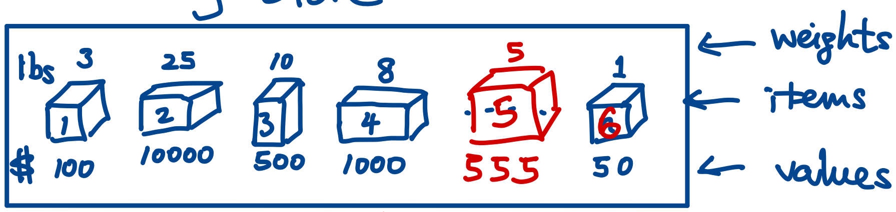
Knapsack Problem



Thief

can only hold 20 lbs

Jewelry Store



Values = {100, 10000, 500, 1000, 555, 50}
Weights = {3, 25, 10, 8, 5, 1}

Input : 1) set of items 1, 2, ... n,
Each item i has a value v_i and weight w_i. $W = 20$

2) The maximum capacity W of knapsack.

Output : Subset $S \subseteq \{1, 2, \dots n\}$ s.t.

a) $\sum_{i \in S} w_i \le W$ and

b) $\sum_{i \in S} v_i$ is as large as possible subject to a).

Defining. OPT(j) - most value we can steal if only allowed to steal items {1, 2, ..., j} and bag of weight W.

$j = 1, \dots, n$.

Ex OPT(1) = 100, OPT(2) = 100, OPT(3) = 600, OPT(4) = 1500 ....

$$\mathrm{OPT}(j) \begin{cases} \text{select item } j & - \quad v_j + \mathrm{OPT}(j-1) \quad \times \\ \text{don't select item } j & - \quad \mathrm{OPT}(j-1). \end{cases}$$

We need to define subproblem considering not only the items, but also the weight. $\Rightarrow$ nW subproblems.

$OPT(j, s)$ – the max. value we can steal if

$1 \le j \le n$ a) only allowed to steal items $\{1, \dots, j\}$. and

$1 \le s \le W$ b) knapsnack of capacity s.

↖ if all $OPT(j, s)$ were known, we would output $OPT(n, W)$.

$OPT(j, s)$

- select item j $= v_j + OPT(j-1, s-w_j)$
- don't select item j $= OPT(j-1, s)$

OPT :

Initialization

for $s = 1$ to W.

  if $s < w_1$

   $OPT(1, s) = 0$

  else

   $OPT(1, s) = v_1$

item: \ weight	1	2	3	- - - $OPT(1, w_1)$	s		W
1	0	0	0	0 .... v_1	v_1	v_1 . - - -	v_1
2							
3							
⋮							
j							
⋮							
n							

for $j = 2$ to n.

  for $s = 2$ to W.

$$OPT(j, s) = \max\left(v_j + OPT(j-1, s-w_j),\; OPT(j-1, s) \right)$$

endfor
endfor
return OPT(n, W). ← max value.

Runtime : $O(nW)$

- not polynomial in $\log W$.
- psedopolynomial.

Longest Common Subsequence.

X - A B B A D C A

Y - B A C A D B B

Is BBC a subseq. of X ? Yes Y? No.

Is BDC a subseq of X ? Yes Y? No.

Is BAD a subseq of X? Yes Y? Yes.

↑ common subseq.

BACA is ~~a~~ the longest common subsequence.

Input: Two strings X and Y, X with length n and Y with length m.

LCS(X,Y)

Output : The longest Common subsequence of X and Y.
(length)

$O(nm)$ algorithm to Compute $LCS(X, Y)$ using DP.

$OPT(i,j)$ - the max length of $LCS(X_i, Y_j)$

$$X_i = x_1 x_2 \dots x_i$$
$$Y_j = y_1 y_2 \dots y_j$$

Example:

X_3- $\boxed{A\ B\ B}$ A D C A $\qquad$ $OPT(3,5) = 1$

Y_5- $\boxed{B\ A\ C\ A\ D}$ B B

We will compute $OPT(i,j)$ for all $1 \le i \le n$, $1 \le j \le m$.

and output $OPT(n,m)$.

$$OPT(1,1) = \begin{cases} 0 & \text{if } x_1 \neq y_1 \\ 1 & \text{if } x_1 = y_1 \end{cases}$$

$$OPT(i,j) = \begin{cases} 0 & \text{if } i=0 \text{ or } j=0 \\ 1 + OPT(i-1, j-1) & \text{if } x_i = y_j \\ \max\left(OPT(i-1,j), OPT(i,j-1)\right) & \text{if } x_i \neq y_j \end{cases}$$

Algorithm:

Initialize a matrix OPT – $n+1$ rows and $m+1$ cols.
S

for $i = 0$ to n

for $j = 0$ to m.

if $i=0$ or $j=0$: $OPT(i,j) = 0$.
$S(i,j) = \phi$

else if $x_i = y_j$: $OPT(i,j) = OPT(i-1, j-1) + 1$
$S(i,j) = S(i-1, j-1)\, x_i$

else if $OPT(i-1, j) > OPT(i, j-1)$:

$OPT(i,j) = OPT(i-1, j)$
$S(i,j) = S(i-1, j)$

else: $OPT(i,j) = OPT(i, j-1)$.
$S(i,j) = S(i, j-1)$

end for
endfor

return $OPT(n,m)$ = length of $LCS(X, Y)$.
$S(n,m)$

What if we want to output $LCS(X,Y)$?

$S[i,j] = LCS(X_i, Y_j)$ is string.

$OPT(n,m) = \text{length}(S[n,m])$.