Algorithm Design and Analysis

Dynamic Programming II

11/2/2022

Knapsack Problem

The Knapsack Problem

Input : A knapsack of size W > 0 (integer)
n different indivisible items
item i has weight $w_i > 0$ and value $v_i > 0$ (ints)

Goal: Fill the knapsack (without overloading) so as to maximize the total value

Knapsack Example

Item	~~Weight~~ Value	~~Value~~ Weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

W = 11

Knapsack Example

Item	~~Weight~~ Value	~~Value~~ Weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

Optimal : {3, 4} with value 40

W = 11

Knapsack Example

Item	~~Weight~~ Value	~~Value~~ Weight
1	1	1
2	6	$\boxed{2}$
3	18	5
4	22	6
5	28	$\boxed{7}$

Optimal : {3, 4} with value 40

Greedy: Pick repeatedly the item with largest $\frac{v_i}{w_i}$ ratio

$\{5, 2, 1\} \Rightarrow 35$

W = 11

Knapsack Example

Item	Weight	Value
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

Optimal : {3, 4} with value 40

Greedy: Pick repeatedly the item with largest $\frac{v_i}{w_i}$ ratio

{5, 2, 1} => value 35 ! Not optimal

Show that Greedy can be arbitrarily bad.

W = 11

DP : Structure of the Optimal (First Attempt)

OPT[i] : Optimal solution considering the items $\{1,2,3,\cdots i\}$ and knapsack of size W

DP : Structure of the Optimal (First Attempt)

OPT[i] : Optimal solution considering the items $\{1,2,3, \cdots i\}$ and knapsack of size W

Case 1: item i is not part of the solution OPT[i]

=> OPT[i-1] = OPT [i]

DP : Structure of the Optimal (First Attempt)

OPT[i] : Optimal solution considering the items $\{1,2,3,\cdots i\}$ and knapsack of size W

Case I: item i is not part of the solution OPT[i]
=> OPT[i-1] = OPT [i]

Case II: item i is part of the solution OPT[i]

DP : Structure of the Optimal (First Attempt)

OPT[i] : Optimal solution considering the items $\{1,2,3,\cdots i\}$ and knapsack of size W

Case I: item i is not part of the solution OPT[i]

=> OPT[i-1] = OPT [i]

Case II: item i is part of the solution OPT[i]

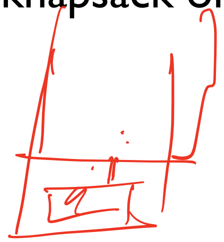
- Inclusion of i does not mean that we need to reject item $i-1$
- But, we also cannot reduce to OPT[i-1]

  We need to pack as much value as possible in a knapsack of size $W - w_i$

DP : Structure of the Optimal (First Attempt)

OPT[i] : Optimal solution considering the items $\{1,2,3,\cdots i\}$ and knapsack of size W

opt [i]



Case I: item i is not part of the solution OPT[i]

=> OPT[i-1] = OPT [i]

Case II: item i is part of the solution OPT[i]

- Inclusion of i does not mean that we need to reject item $i-1$
- But, we also cannot reduce to OPT[i-1]

  We need to pack as much value as possible in a knapsack of size $W - w_i$

Moral of the Story : One parameter not enough to capture the problem !

DP : Structure of the Optimal (Second Attempt)

OPT[i, w] : Optimal solution considering the items $\{1,2,3,\cdots i\}$ and knapsack of size w for all $i = 0, 1, 2, .., n$, $W = 0, 1, 2, \ldots, W$

↳ smaller subproblems

$\text{OPT}(n, W)$: final answer

Case I: item i is not part of OPT[i, w]

$$\text{OPT}[i, w] = \text{OPT}[i-1, w]$$

Case II : item i is part of OPT[i, w]

$$\text{OPT}[i, w] = \text{OPT}[i-1, w - w_i] + v_i$$

$[w \geq w_i]$

Base case :- $\text{OPT}[0, w] = 0 \;\; \forall \; w = 0, 1, 2, \ldots, W.$

Recursive (with Memoization) Algorithm

$M[i, w]$: 2-D Array of size $n \times W$, initialize to -1

$(n+1) \times (W+1)$

Knap (i, w)

If $M[i, w]$== invalid Return $M[i, w]$

If $(i == 0)$

$M[i, w] = 0$ [base case]

else if $(w_i > w)$

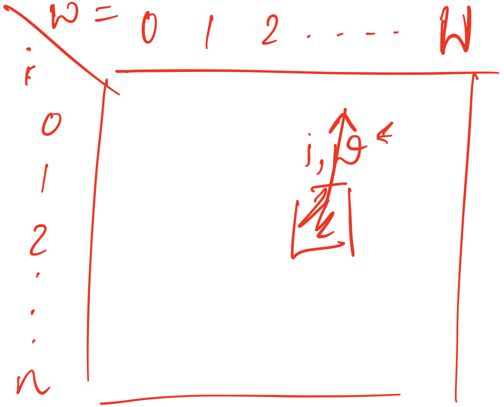
$M[i, w]$ = Knap$(i-1, w)$

else

$M[i, w]$ = max{Knap$(i-1, w)$,

Knap$(i-1, w-w_i) + v_i$}

Return $M[i, w]$



Knap[n, W]

Dynamic Programming Algorithm

$M[i, w]$: 2-D Array of size $n \times W$ (n+1) (W+1)

Knap (n, W)

for $w = 0$ *to* W

$M[0, w] = 0$ } Base case

for $i = 1$ *to* n

for $w = 0$ *to* W

if ($w_i > w$)

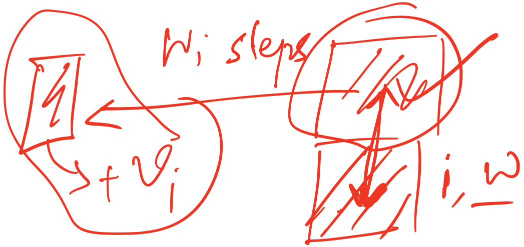
$M[i, w] = M[i-1, w]$

else

$M[i, w] = \max\{ M[i-1, w],$

$M[i-1, w - w_i] + v_i\}$

Return $M[n, W]$



Opt(i, w)
for all i,
all w = 0, .. W

Runtime:-
O(nW)

i \ w	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	0	0	3	3	3
2	0	0	0	2	3	3	3
3	0	0	4	4	4	6	7
4	0	0	4	4	4	8	8

Item	Weight	Value
1	4	3
2	3	2
3	2	4
4	3	4

W=6

{item 4, item 3}

if $(w_i > w)$

$$M[i,w] = M[i-1,w]$$

else

$$M[i,w] = \max\{ M[i-1,w], M[i-1, w-w_i] + v_i\}$$

Runtime Discussion

$M[i,w]$: 2-D Array of size $n \times W$

Knap (n, W)

$$
\left.
\begin{array}{l}
\text{for } w = 0 \text{ to } W \\
\quad M[0,w] = 0
\end{array}
\right\} O(W)
$$

$$
\left.
\begin{array}{l}
\text{for } i = 1 \text{ to } n \\
\quad \text{for } w = 0 \text{ to } W \\
\quad\quad \text{if } (w_i > w) \\
\quad\quad\quad M[i,w] = M[i-1,w] \\
\quad\quad \text{else} \\
\quad\quad\quad M[i,w] = \max\{ M[i-1,w], \\
\quad\quad\quad\quad\quad M[i-1, w-w_i] + v_i\}
\end{array}
\right\} O(nW)
$$

Return $M[n, W]$

$O(nW)$

pseudopolynomial.

There are essentially two loops : outer one running for n iterations and the inner one running for W iterations. Inside, there is only 4 operations