Dynamic Programming

- The Knapsack Problem

- Dynamic Programming Formulation

- Implementation

- Fractional Knapsack and Subset Sum Problem

Imdad ullah Khan

Knapsack Problem: Dynamic Programming

Input: A set $\mathcal{U}$ of objects $\{a_1, \ldots, a_n\}$ with

- integral weights $\{w_1, \ldots, w_n\}$ and
- positive values $\{v_1, \ldots, v_n\}$ and
- a positive integer C (capacity)

**Output:** A subset $S \subset \mathcal{U}$ with total weight $\leq C$ and maximum total value

- Fix an order on objects $a_1, \ldots, a_n$
- $\mathrm{OPT\text{-}SET}(k, c)$ is the max value feasible subset of $\mathcal{U}[1 \ldots k]$ and c
- $\mathrm{OPT\text{-}VAL}(k, c)$ is the total value of $\mathrm{OPT\text{-}SET}(k, c)$

Out goal is to find $\mathrm{OPT\text{-}VAL}(n)$ (and $\mathrm{OPT\text{-}SET}(n)$)

Knapsack Problem: Dynamic Programming

$$\text{OPT-VAL}(k, c) = \max \begin{cases} 0 & \text{if } k = 0 \\ 0 & \text{if } c = 0 \\ \text{OPT-VAL}(k-1, c-w_k) & \text{if } a_k \in \text{OPT-SET}(k, c) \\ \text{OPT-VAL}(k-1, c) & \text{if } a_k \notin \text{OPT-SET}(k, c) \end{cases}$$

This is a two variable recurrence. Need a 2-dimensional memo
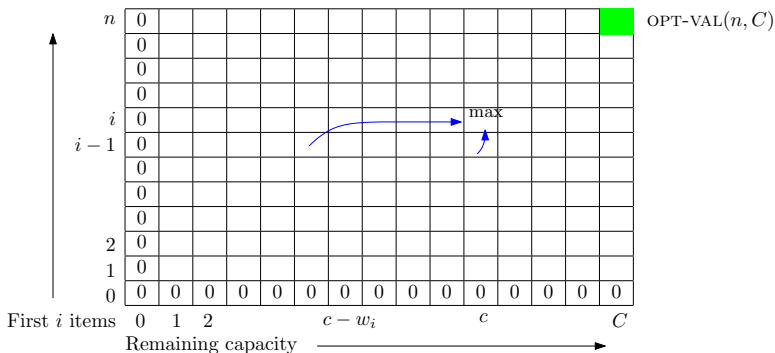
Knapsack Problem: Dynamic Programming

$$\text{OPT-VAL}(k, c) = \max \begin{cases} 0 & \text{if } k = 0 \\ 0 & \text{if } c = 0 \\ \text{OPT-VAL}(k - 1, c - w_k) & \text{if } a_k \in \text{OPT-SET}(k, c) \\ \text{OPT-VAL}(k - 1, c) & \text{if } a_k \notin \text{OPT-SET}(k, c) \end{cases}$$
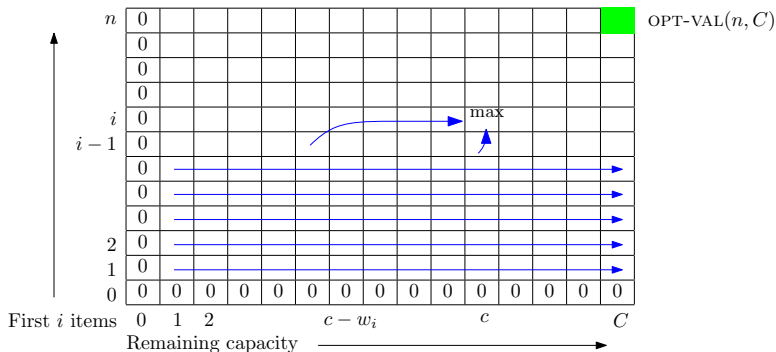
Fill in the memo solutions table bottom to top, left to right

Knapsack Problem: Dynamic Programming

Algorithm Knapsack with memoization, n, C

    for $i = 0$ to n **do**   ▷ Initially $\text{OPT-SET}[i][c]$'s are unknown
      for $c = 0$ to C **do**
        $\text{OPT}[i][c] \leftarrow -\infty$

    for $c = 0$ to C **do**
      $\text{OPT}[0][c] \leftarrow 0$ ▷ when $i = 0 \implies U = \emptyset$, then $\text{OPT}[0][\cdot] = 0$

    for $i = 0$ to n **do**
      $\text{OPT}[i][0] \leftarrow 0$ ▷ when $c = 0 \implies$ no capacity, then $\text{OPT}[\cdot][0] = 0$

    for $i = 1$ to n **do**
      for $c = 0$ to C **do**
        if $\text{OPT}[i-1][c - w_i] + v_i \geq \text{OPT}[i-1][c]$ and $c \geq w_i$ **then**
          $\text{OPT}[i][c] \leftarrow \text{OPT}[i-1][c - w_i] + v_i$
        else
          $\text{OPT}[i][c] \leftarrow \text{OPT}[i-1][c]\}$

    return $\text{OPT}[n][C]$

Knapsack Problem: Dynamic Programming

$$\text{OPT-VAL}(k, c) = \max \begin{cases} 0 & \text{if } k = 0 \\ 0 & \text{if } c = 0 \\ \text{OPT-VAL}(k-1, c - w_k) & \text{if } a_k \in \text{OPT-SET}(k, c) \\ \text{OPT-VAL}(k-1, c) & \text{if } a_k \notin \text{OPT-SET}(k, c) \end{cases}$$

$a_i \in \text{OPT-SET}(i, c)$ iff $\text{OPT-VAL}[i-1][c - w_i] + v_i \geq \text{OPT-VAL}[i-1][c]$

Backtrack from $\text{OPT-VAL}(n, C)$ to see whether or not a_i is included

---

function FIND-SET(i,c)
   if $i = 0$ or $c = 0$ **then**
      return $\emptyset$
   else
      if $OptVal[i-1][c - w_i] + v_i \geq OptVal[i-1][c]$ **then**
         return $a_i \cup$ FIND-SET($i-1, c - w_i$)
      else
         return FIND-SET($i-1, c$)

Knapsack Problem: Dynamic Programming

function FIND-SET(i,c)
  if $i = 0$ or $c = 0$ **then**
    return $\emptyset$
  else
    if $OptVal[i-1][c-w_i] + v_i \geq OptVal[i-1][c]$ **then**
      return $a_i \cup$ FIND-SET$(i-1, c-w_i)$
    else
      return FIND-SET$(i-1, c)$

5	0	3	3	4	4	8	11	11	12	12	12	13	13	13	16	16
4	0	3	3	4	4	8	11	11	12	12	12	13	13	13	13	13
3	0	3	3	3	3	8	11	11	11	11	11	11	13	13	13	13
2	0	3	3	3	3	3	3	5	5	5	5	5	5	5	5	5
1	0	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

ID	weight	value
a_1	1	3
a_2	6	2
a_3	5	8
a_4	2	1
a_5	8	5

$C = 15$

Knapsack Problem: Dynamic Programming

Runtime:

- Each entry is filled in $O(1)$ if the two required entries are already filled

- FIND-SET(n, C) takes $O(n)$ time

- Total runtime is $O(nC)$

  pseudo-polynomial time

  - C is the input, not size of input
  - C can be expressed in $\log C$ bits

- So it is exponential in size of one input parameter

Note we required C to be integer, as memo is indexed by it