# CS3000: Algorithms & Data
Paul Hand

Lecture 10:

- Dynamic Programming: Knapsack Problems

Feb 11, 2019

# Dynamic Programming

# Dynamic Programming Recap

- Express the optimal solution as a **recurrence**
  - Identify a small number of subproblems
  - Relate the optimal solution on subproblems

- Efficiently solve for the **value** of the optimum
  - Simple implementation is exponential time
  - Top-Down: store solution to subproblems
  - Bottom-Up: iterate through subproblems in order

- Find the **solution** using the table of **values**

# Dynamic Programming: Knapsack Problems

# Tug-of-War

- We have $n$ students with weights $w_1, \ldots, w_n \in \mathbb{N}$, need to split as evenly as possible into two teams
    - e.g. $\{21,42,33,52\}$

Q

| Group 1 | Group 2 |
|---|---|
| 21, 42 | 33, 52 |
| = 63 | = 85 |
| 42, 33 | 21, 52 |
| 75 | 73 |

Soln to TOW problem

# The Knapsack Problem

- **Input:** $n$ items for your knapsack
  - value $v_i$ and a weight $w_i \in \mathbb{N}$ for $n$ items
  - capacity of your knapsack $T \in \mathbb{N}$

assuming these are natural #s

size

- **Output:** the most valuable subset of items that fits in the knapsack
  - Subset $S \subseteq \{1, \dots, n\}$
  - Value $V_S = \sum_{i \in S} v_i$ as large as possible
  - Weight $W_S = \sum_{i \in S} w_i$ at most $T$

Could write as optimization problem

$$\max_{\substack{S \subseteq \{1 \cdots n\} \\ W_S \leq T}} V_S$$

- **SubsetSum:** $v_i = w_i$

is there a subset that adds up to T?

Tug of War: $T = \frac{1}{2} \sum_{i=1}^{n} V_i$

# Solve this Knapsack by hand

- Total Weight T = 10
  What collection of items maximizes value with total weight of at most T?

| $i$ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| $v_i$ | 10 | 40 | 30 | 50 |
| $w_i$ | 5 | 4 | 6 | 3 |

$S = \{2, 4\}$

$V_S = 40 + 50 = 90$

$W_S = 4 + 3 = 7 \leq 10$

Heuristic

Bang for Buck $\frac{V_i}{W_i}$ 2 10 ☆ 5 $\frac{50}{3} > 10$ ☆

# Is Dynamic Programming Necessary?

What idea is your problem going to be based on?

- Want to maximize **bang-for-buck**, right?
  - Items with large $\frac{v_i}{w_i}$ seem like good choices
  - Design a Knapsack problem where selecting items in decreasing order of bang-for-buck (subject to the weight constraint) gives the incorrect result.

A single item w/ large b-f-b but doesn't consume much of knapsack

$T = 10$

| $i$ | 1 | 2 | 3 | ~~4~~ |
|---|---|---|---|---|
| $V_i$ | 10 | 15 | 24 | |
| $W_i$ | 5 | 5 | 7 | |
| $\frac{V_i}{W_i}$ | 2 | 3 | $>3$ | |

Greedy: $\{3\}$ val 24

Other $\{1,2\}$ Val 25

# Dynamic Programming

What can you say about a solution to a smaller problem in each of these cases?

How do you relate these cases to simpler version of Knapsack

- Let $O \subseteq \{1, \ldots, n\}$ be the **optimal** subset of items for a knapsack of size T
- **Case 1:** $n \notin O$

If n is not in Knapsack of size T, then then O is optimal soln to a Knapsack problem on $\{1, \ldots, n-1\}$ with size T

- **Case 2:** $n \in O$

If n is in Knapsack of size T, then O is an optimal Soln on $\{1 \ldots n-1\}$ with size $T - W_n$. together with $\{n\}$

Find a "smaller" knapsack problem whose sol'n is useful to you.

variable has been added

# Dynamic Programming

optimal value of Knapsack of size S using only items 1 ... j

$O_{j,S}$ is the optimal set

- Let $\mathbf{OPT}(\boldsymbol{j}, \boldsymbol{S})$ be the **value** of the optimal subset of items $\{1, \dots, j\}$ in a knapsack of size $S$
- **Case 1:** $j \notin O_{j,s}$

$$O_{j-1,S} = O_{j,S}$$

$$opt(j-1,S) = opt(j,S)$$

- **Case 2:** $j \in O_{j,s}$

$$O_{j,S} = \{j\} + O_{j-1,S-w_j}$$

$$opt(j,S) = v_j + opt(j-1,S-w_j)$$

# Dynamic Programming

- Let $\mathbf{OPT}(\boldsymbol{j}, \boldsymbol{S})$ be the **value** of the optimal subset of items $\{1, \ldots, j\}$ in a knapsack of size $S$
- **Case 1:** $j \notin O_{j,S}$
  - Use opt. solution for items 1 to j-1 and size S
- **Case 2:** $j \in O_{j,S}$
  - Use $i$ + opt. solution for items 1 to j-1 and size $S - w_j$

# Dynamic Programming

- Let $\mathbf{OPT}(\boldsymbol{j}, \boldsymbol{S})$ be the **value** of the optimal subset of items $\{1, \ldots, j\}$ in a knapsack of size $S$
- **Case 1:** $i \notin O_{j,S}$
  - Use opt. solution for items 1 to j-1 and size S
- **Case 2:** $i \in O_{j,S}$
  - Use $i$ + opt. solution for items 1 to j-1 and size $S - w_j$

**Recurrence:**

$$\mathrm{OPT}(j,S) = \begin{cases} \max\{\overbrace{OPT(j-1,S)}^{\text{case 1}}, \overbrace{v_j + OPT(j-1,S-w_j)}^{\text{case 2}}\} & \text{if } w_j \le S \\ OPT(j-1,S) & \text{if } w_j > S \end{cases}$$

Can't afford j, not in set

**Base Cases:**

$$\mathrm{OPT}(j,0) = \mathrm{OPT}(0,S) = 0$$

# Activity

$$\mathrm{OPT}(j,S) = \begin{cases} \max\{\underline{OPT(j-1,S)}, v_j + OPT(j-1,S-w_j)\} & \text{if } w_j \le S \\ OPT(j-1,S) & \text{if } w_j > S \end{cases}$$

where do you start?

$opt(1,1) = \begin{cases} \max(0, v_1 + 0) \\ - - - \end{cases}$

$= \max(0, v_1) = v_1 = 1$

- Input: $T = 8, n = 3$
  - $w_1 = 1, v_1 = 4$
  - $w_2 = 3, v_2 = 5$
  - $w_3 = 5, v_3 = 8$

$OPT(j,S)$ optimal value of Knapsack on $1 \cdots j$ w/ size $S$

① Check if you can afford the item (which branch of is relevant)

② Look up $opt(j-1,S)$, $v_j + opt(j-1,S-w_j)$

Take max

$j$ ↑

items

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **3** | 0 | 4 | 4 | | | | | | $\boxed{13}$ |
| **2** | 0 | $\boxed{4}$ | $\boxed{4}$ | $\boxed{5}$ | . | - - | - | - | |
| **1** | 0 | 4 | $\boxed{4}$ | 4 | 4 | 4 | 4 | 4 | 4 |
| **0** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **-** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |

capacities

$S$ →

# Knapsack ("Bottom-Up")

```
// All inputs are global vars
FindOPT(n,T):
  M[0,~~S~~T] ← 0, M[j,0] ← 0
  for (j = 1,…,n):
    for (s = 1,…,T):
      if (w_j > ~~S~~s): M[j,~~S~~s] ← M[j-1,~~S~~s]
      else: M[j,s] ← max{M[j-1,~~S~~s],v_j + M[j-1,~~S~~s-w_j]}

  return M[n,T]
```

base cases

T ↖ put 0's in entire row

Activity: What is the runtime of this algorithm?

nT

/

depends on size of the knapsack

How much memory does it take?

nT

# Dynamic Programming

- Let $\boldsymbol{O_{j,S}}$ be the **optimal subset of items** $\{1, \dots, j\}$ in a knapsack of size $S$
- **Case 1:** $j \notin O_{j,S}$
    - Use opt. solution for items 1 to j-1 and size S
- **Case 2:** $j \in O_{j,S}$
    - Use $i$ + opt. solution for items 1 to j-1 and size $S - w_j$

# Filling the Knapsack

```
// All inputs are global vars
// M[0:n,0:T] contains solutions to subproblems
FindSol(M,n,T):
  if (n = 0 or T = 0): return $\emptyset$
  else:
    if ($w_n$ > T): return FindSol(M,n-1,T)
    else:
      if (M[n-1,T] > $v_n$ + M[n-1,T-$w_n$]):
        return FindSol(M,n-1,T)
      else:
        return {n} + FindSol(M,n-1,T-$w_n$)
```

# Knapsack Wrapup

- Can solve knapsack problems in time/space $O(nT)$
  - Brute force algorithms run in time $O(2^n)$

- Dynamic Programming:
  - Decide whether the $n^{th}$ item goes in the knapsack
- Can solve subset-sum and tug-of-war