CS3000: Algorithms & Data
Jonathan Ullman

Lecture 8:

- Dynamic Programming: Knapsacks, ~~Edit Distance~~

Oct 2, 2018

Midterm I

- In class on Tuesday Oct 16
- Types of questions similar to HW, but shorter
- Topics:
  - Asymptotics, Recurrences, Proof by Induction
  - Divide-and-Conquer Algs
  - Dynamic Programming Algs
- HW3 due 10/5, HW4 due 10/12
- One-page cheat sheet

Tug-of-War, Subset-Sum, Knapsack

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. $\{21,42,33,52\}$

$\{21, 42\}$ vs. $\{33, 52\}$

63 85

$\{33, 42\}$ vs. $\{21, 52\}$

75 vs. 73

The Knapsack Problem

$2n+1$ numbers

- **Input:** n items for your knapsack
  - value v_i and a weight $w_i \in \mathbb{N}$ for n items
  - capacity of your knapsack $T \in \mathbb{N}$
- **Output:** the most valuable subset of items that fits in the knapsack

  $\underset{\substack{S \subseteq \{1,\dots,n\} \\ \text{s.t. } W_S \leq T}}{\operatorname{argmax}} \quad V_S$

  - 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

- **SubsetSum:** $v_i = w_i$
- Tug-of-War is a special case of subset sum $\left(T = \frac{1}{2}\sum_{i=1}^{n} w_i\right)$

Is Dynamic Programming Necessary?

- Want to maximize **bang-for-buck**, right?
  - Items with large $\frac{v_i}{w_i}$ seem like good choices
  - Wont always give optimal solution

$n = 3$ $\quad v_1 = 6 \quad w_1 = 5 \quad T = 8$

$v_2 = 4 \quad w_2 = 4$

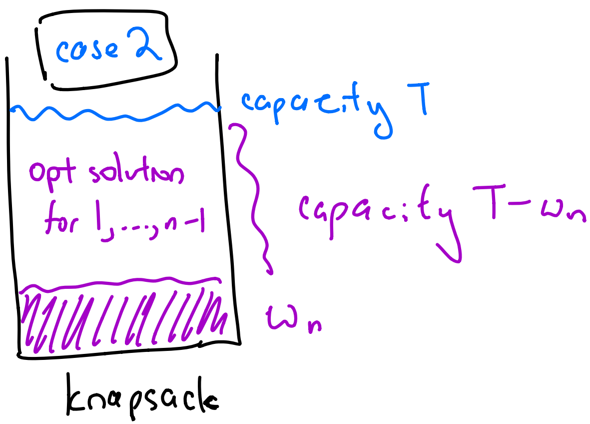
$v_3 = 4 \quad w_3 = 4$

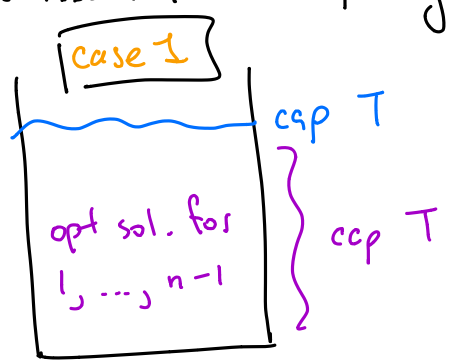
heuristic chooses $S = \{1\}$, $V_S = 6$

opt is $S = \{2, 3\}$, $V_S = 8$

Dynamic Programming

- Let $O \subseteq \{1, \ldots, n\}$ be the **optimal** subset of items
- **Case 1:** $n \notin O$ (If $w_n > T$ then n is not in the knapsack)
  - O is the optimal solution for items $\{1, \ldots, n-1\}$ w. capacity T
- **Case 2:** $n \in O$
  - O is $\{n\} \cup \{$opt. for items $1 \ldots n-1$ and capacity $T - w_n\}$





Dynamic Programming

add variables

$0 \le j \le n$
$0 \le S \le T$ } $O(nT)$ subproblems

- 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:** $j \notin O_{j,S}$
  - $OPT(j, S) = OPT(j-1, S)$

- **Case 2:** $j \in O_{j,S}$
  - $OPT(j, S) = v_j + OPT(j-1, S - w_j)$

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

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 $S - w_j$

Dynamic Programming

- 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:** $i \notin O_{j,S}$
  - 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:

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

Base Cases:

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

Ask the Audience

$OPT(3,8) =$

$\max\{ OPT(2,8), 8 + OPT(2,3) \}$

$= \max\{ 9, 8+5 \}$

$= 13$

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

OPT(items, capacity)

items									
3	0	4	4	5	9	9	12	12	13
2	0	4	4	5	9	9	9	9	9
1	0	4	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

Knapsack (“Bottom-Up”)

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

  return M[n,T]
```

Dynamic Programming

- Let $\boldsymbol{O_{j,S}}$ be 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

$$V_{O_{j,S}} = V_{O_{j-1,S}} \Rightarrow O_{j,S} = O_{j-1,S}$$

- **Case 2:** $j \in O_{j,S}$
  - Use i + opt. solution for items 1 to j-1 and size $S - w_j$

$$V_{O_{j,S}} = V_{O_{j-1,S-w_j}} + v_j \Rightarrow O_{j,S} = \{j\} \cup O_{j-1,S-w_j}$$

Caveat: Both might be true

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 ∅
  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)
```

SLS Wrapup

- Can solve knapsack problems in time/space $O(nT)$
  - Brute force algorithms runs 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