

CS 312: Algorithms

Subset Sum

Dan Sheldon

Mount Holyoke College

Last Compiled: November 7, 2018

Dynamic Programming Recipe

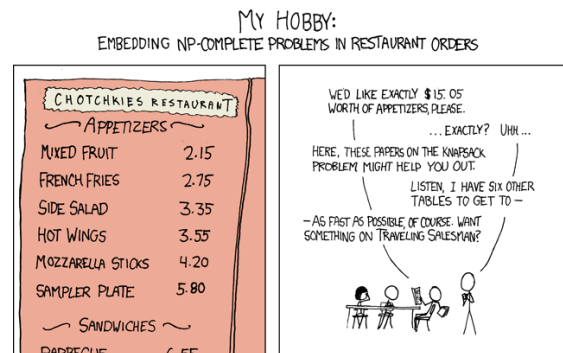
- ▶ **Step 1:** Devise simple recursive algorithm
 - ▶ Make *one decision* by trying all possibilities
 - ▶ Use a recursive solver to evaluate the value of each
 - ▶ **Problem:** it does redundant work, often exponential time
- ▶ **Step 2:** Write recurrence for optimal value
- ▶ **Step 3:** Design iterative algorithm

- ▶ Subset Sum: need to “add a variable”

Problem Formulation

- ▶ Example on board
- ▶ **Input**
 - ▶ Items $1, 2, \dots, n$
 - ▶ Weights w_i for all items (integers)
 - ▶ Capacity W
- ▶ **Goal:** select a subset S whose total weight is as large as possible without exceeding W .
- ▶ Subset Sum: need to “add a variable” to recurrence

Subset Sum



Step 1: Recursive Algorithm, First Try

- ▶ Let O be an optimal solution
 - ▶ If $n \notin O$, then recurse on first $n - 1$ items
 - ▶ If $n \in O$, then ...
- ▶ $\text{SubsetSum}(j)$
 - if** $j = 0$ **then** return 0
 - ▶ **Case 1:** $j \notin O$
val1 = ???
 - ▶ **Case 2:** $j \in O$
if $w_j \leq W$ **then**
val2 = ???
 - else**
val2 = 0
 - end if**
 - return max(val1, val2)
- ▶ Exercise: what goes wrong?
Cannot express val2 recursively: the remaining capacity no longer W .
- ▶ Exercise: fix it

Step 1: Recursive Algorithm, Add a Variable

- ▶ Find value of optimal solution O on items $\{1, 2, \dots, j\}$ when the remaining capacity is w
- ▶ $\text{SubsetSum}(j, w)$
 - if** $j = 0$ **then** return 0
 - ▶ **Case 1:** $j \notin O$
val1 = $\text{SubsetSum}(j - 1, w)$
 - ▶ **Case 2:** $j \in O$
if $w_j \leq w$ **then**
val2 = $w_j + \text{SubsetSum}(j - 1, w - w_j)$
 - else**
val2 = 0
 - end if**
 - return max(val1, val2)

Recurrence

- ▶ Let $\text{OPT}(j, w)$ be the maximum weight of any subset of items $\{1, \dots, j\}$ that does not exceed w

base case: $\text{OPT}(0, w) = 0$ for all w

if $w_j \leq w$ then: $\text{OPT}(j, w) = \max \left\{ \begin{array}{l} \text{OPT}(j-1, w), \\ w_j + \text{OPT}(j-1, w-w_j) \end{array} \right\}$

else: $\text{OPT}(j, w) = \text{OPT}(j-1, w)$

- ▶ Questions
 - ▶ Do we need a base case for $\text{OPT}(j, 0)$? **No.**
 - ▶ What is overall optimum to original problem? $\text{OPT}(n, W)$

Step 3: Iterative Algorithm

- ▶ $\text{SubsetSum}(n, W)$

Initialize array $M[0..n, 0..W]$ to hold optimal values of subproblems

Set $M[0, w] = 0$ for $w = 0, \dots, W$

for $j = 1$ to n **do**

for $w = 0$ to W **do**

 Use recurrence from previous slide to compute $M[j, w]$

end for

end for

return $M[n, W]$

- ▶ [Example on board.](#)
- ▶ Running Time? $\Theta(nW)$. Note: this is "pseudopolynomial". Not strictly polynomial, because it can be exponential in the number of *bits* used to represent the values.

A Related Problem: Knapsack

- ▶ n items
- ▶ weights w_i
- ▶ values v_i

Find the subset of items with total weight at most W

WINGS			
4 Chicken Wings	4.55	24 Chicken Wings	27.25
5 Chicken Wings	5.70	25 Chicken Wings	27.80
6 Chicken Wings	6.80	26 Chicken Wings	28.95
7 Chicken Wings	7.95	27 Chicken Wings	30.10
8 Chicken Wings	9.10	28 Chicken Wings	31.20
9 Chicken Wings	10.20	29 Chicken Wings	32.35
10 Chicken Wings	11.35	30 Chicken Wings	33.50
11 Chicken Wings	12.50	35 Chicken Wings	39.15
12 Chicken Wings	13.60	40 Chicken Wings	44.80
13 Chicken Wings	14.75	45 Chicken Wings	50.50
14 Chicken Wings	15.90	50 Chicken Wings	55.60
15 Chicken Wings	17.00	60 Chicken Wings	67.00
16 Chicken Wings	18.15	70 Chicken Wings	78.30
17 Chicken Wings	19.30	75 Chicken Wings	83.45
18 Chicken Wings	20.40	80 Chicken Wings	89.10
19 Chicken Wings	21.55	90 Chicken Wings	100.45
20 Chicken Wings	22.70	100 Chicken Wings	111.25
21 Chicken Wings	23.80	125 Chicken Wings	139.00
22 Chicken Wings	24.95	150 Chicken Wings	166.85
23 Chicken Wings	26.10	200 Chicken Wings	222.50

Exercise: Find the cheapest way to buy exactly n chicken wings