CS 312: Algorithms

Subset Sum

Dan Sheldon

Mount Holyoke College

Last Compiled: November 7, 2018

Dynamic Programming Recipe

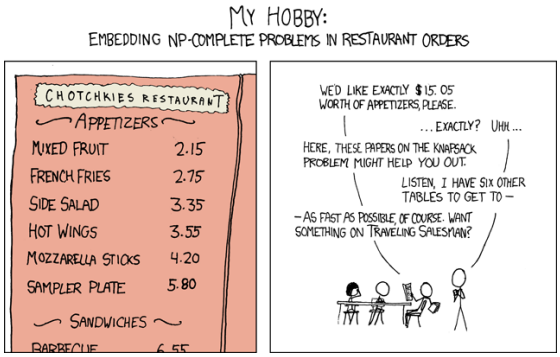
- Step 1: Devise simple recursive algorithm
  - Make *one decision* by trying all possibilities
  - Use a recursive solver to evaluate the value of each
  - Problem: it does redundant work, often exponential time
- Step 2: Write recurrence for optimal value
- Step 3: Design iterative algorithm

- Subset Sum: need to "add a variable"

Problem Formulation

- Example on board
- **Input**
  - Items $1, 2, \ldots, n$
  - 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 ...

- 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 $\{1, 2, \ldots, j\}$ when the remaining capacity is w

- SubsetSum(j,w)
  if $j = 0$ **then** return 0
  ▷ Case 1: $j \notin O$
  val1 = SubsetSum($j-1$, w)
  ▷ Case 2: $j \in O$
  if $w_j \leq w$ **then**
  val2 = w_j + SubsetSum($j-1$, $w-w_j$)
  else
  val2 = 0
  end if
  return max(val1, val2)

Recurrence

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

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

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

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

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

Step 3: Iterative Algorithm

- SubsetSum(n,W)

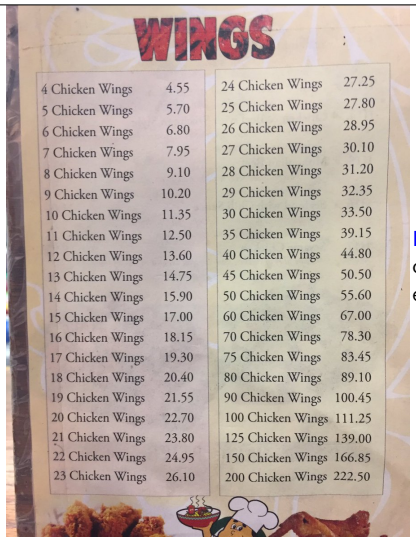
  Initialize array $M[0..n, 0..W]$ to hold optimal values of subproblems
  Set $M[0, w] = 0$ for $w = 0, \ldots, W$
  for $j = 1$ to n **do**
  for $w = 0$ to W **do**
  Use recurrence from previous slide to compute $M[j, w]$
  end for
  end for
  return $M[n, W]$

- 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



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