CS 4800: Algorithms & Data

Lecture 8

February 2, 2018

Coin change

- Coin of denominations d_1, d_2, …, d_k
- Wants to make change for n cents using as few coins as possible

Memoization

- Initialize $Best[0] \leftarrow 0$
- $ComputeBest(v)$:
  - If $Best[v]$ is calculated, return $Best[v]$
  - Else
    - $Best[v] \leftarrow 1 + \min_{i=1\ldots k, d_i \leq v} ComputeBest(v - d_i)$
    - Return $Best[v]$

Bottom-up style

- $Best[0] \leftarrow 0$
- For v from 1 to n
  - $Best[v] \leftarrow 1 + \min_{i=1\ldots k, d_i \leq v} Best[v - d_i]$

Knapsack problem

- n items, each with weight w_i and value v_i
- Capacity S
- Pick items with maximum total value without exceeding capacity
- Comparison with log cutter?

First attempt

- Imitate our log cutter solution
- Best(s) : best value from sack size s
- Let the first item in the sack be item i with weight w_i and value v_i
- The rest of the solution should be the best way to fill up capacity $s-w_i$
- If not, we can sub in best solution for $s-w_i$ and get a better solution for w
- What's wrong?
- best($s-w_i$) might use item i and we cannot use i twice

Example

- Capacity W=2
- Two items
  - $w_1 = 1, v_1 = 2$
  - $w_2 = 2, v_2 = 3$
- $best(1) = 2$ (use item 1)
- $best(2) \stackrel{??}{=} 2(from\ item\ 1) + best(1)$

Second attempt

- Our subproblems need not only sack size w but also which items are available
- best(i, w): best value for sack size w using items i, i+1, …, n.

Formulate recursion

- How to compute best(i, s)?
- Either pick item i or not
- $best(i,s) = \max \begin{cases} best(i+1,s) \\ best(i+1,s-w_i)+v_i \end{cases}$
- Order to evaluate?
- From large i to small i

Knapsack DP

- For j $\leftarrow 0\ to\ s$
  - best$[n+1, j] \leftarrow 0$ // base case: no item, no value
- For i $\leftarrow n\ downto\ 1$ // add one item at a time
  - For j $\leftarrow 0\ to\ s$
    - $best[i, j] \leftarrow best[i+1, j]$ // not use item i
    - If j $\geq w_i$ and $best[i+1, j] < v_i + best[i+1, j-w_i]$
      - $best[i, j] \leftarrow v_i + best[i+1, j-w_i]$ // use item i
- Return $best[1, s]$

Remember choices

- For j ← $0\ to\ s$
  - $best[n+1, j] \leftarrow 0$ // base case: no item, no value
- For i ← $n\ downto\ 1$ // add one item at a time
  - For j ← $0\ to\ s$
    - $best[i, j] \leftarrow best[i+1, j]$ // not use item i
    - $picked[i, j] \leftarrow False$
    - If $j \geq w_i$ and $best[i+1, j] < v_i + best[i+1, j-w_i]$
      - $best[i, j] \leftarrow v_i + best[i+1, j-w_i]$
      - $picked[i, j] \leftarrow True$
- Return $best[1, s]$

Retrace the solution

- cap = s
- sol = {}
- For i $\leftarrow 1\ to\ n$
  - If picked[i,cap] then
    - $sol \leftarrow sol \cup \{i\}$
    - $cap \leftarrow cap - w_i$

3 items, capacity 5, $w_1=w_2=w_3=2$, $v_1=9$, $v_2=8$, $v_3=7$

items Cap	{1,2,3}	{2,3}	{3}	none
0				0
1				0
2				0
3				0
4				0
5				0
6				0

Answer = best[{1,2,3}, 5] = 17

3 items, capacity 5, $w_1=w_2=w_3=2$, $v_1=9$, $v_2=8$, $v_3=7$

items Cap	{1,2,3}	{2,3}	{3}	none
0	0	0	0	0
1	0	0	0	0
2	Max(9,8)=9	Max(8,7)=8	7	0
3	9	8	7	0
4	Max(17,15)	15	7	0
5	17	15	7	0
6	Max(24,15)	15	7	0

Answer = best[{1,2,3}, 5] = 17

Blueprint of DP

- Model solution as a sequence of decisions
- Guess the first/last decision to reduce to smaller problems
- Relate subproblems via recurrence
- Either
  - Recursion and memoization
  - Determine evaluation order to build DP table bottom-up
- Solve original problem

Blueprint of DP in log cutter

Subproblems	Best(t): max $ from t-inch log for t=1…n
Guess	Thickness of last board
Recurrence	$Best(t) = \max_{i \leq t}(p_i + Best(t-i))$
Order	Compute Best(t) for t from 1 to n
Solve orig. problem	Return Best(n)

Blueprint of DP in knapsack

Subproblems	
Guess	
Recurrence	
Order	
Solve orig. problem	

Blueprint of DP in knapsack

Subproblems	Best(i, C): max value for sack size C with items {i,i+1, …, n}
Guess	Whether pick item i or not
Recurrence	$Best(i, C) = \max(v_i + Best(i+1, C - w_i), Best(i+1, C))$
Order	Compute Best(i,C) for i from n down to 1
Solve orig. problem	Return Best(1, W)