CSE408
Knapsack problem

Lecture # 35

Dynamic Programming

D*ynamic Programming* is a general algorithm design technique for solving problems defined by or formulated as recurrences with overlapping sub instances

- Invented by American mathematician Richard Bellman in the 1950s to solve optimization problems and later assimilated by CS

- "Programming" here means "planning"

- Main idea:
  - set up a recurrence relating a solution to a larger instance to solutions of some smaller instances
  - solve smaller instances once
  - record solutions in a table
  - extract solution to the initial instance from that table

Knapsack Problem by DP

Given n items of

integer weights: w_1 w_2 … w_n

values: v_1 v_2 … v_n

a knapsack of integer capacity W

find most valuable subset of the items that fit into the knapsack

Consider instance defined by first i items and capacity j ($j \leq W$).

Let $V[i,j]$ be optimal value of such an instance. Then

$$V[i,j] = \begin{cases} \max\{V[i-1,j],\ v_i + V[i-1,j-w_i]\} & \text{if } j-w_i \geq 0 \\ V[i-1,j] & \text{if } j-w_i < 0 \end{cases}$$

Initial conditions: $V[0,j] = 0$ and $V[i,0] = 0$

Knapsack Problem by DP (example)

Example: Knapsack of capacity $W = 5$

item	weight	value
1	2	$12
2	1	$10
3	3	$20
4	2	$15

$w_1 = 2, v_1 = 12$ 1

$w_2 = 1, v_2 = 10$ 2

$w_3 = 3, v_3 = 20$ 3

$w_4 = 2, v_4 = 15$ 4

	capacity j				
	0	1	2	3	4	5
0	0	0	0	0	0	
1	0	0	12	12	12	
2	0	10	12	22	22	22
3	0	10	12	22	30	32
4	0	10	15	25	30	37

Backtracing finds the actual optimal subset, i.e. solution.

Knapsack Problem by DP (pseudocode)

Algorithm DPKnapsack(*w*[*1..n*], *v*[*1..n*], *W*)

var *V*[*0..n,0..W*], *P*[*1..n,1..W*]: int

for *j := 0* to *W* do

V[*0,j*] := *0*

for *i := 0* to *n* do

V[*i,0*] := *0*

for *i := 1* to *n* do

for *j := 1* to *W* do

if *w*[*i*] $\leq$ *j* and *v*[*i*] + *V*[*i-1,j-w*[*i*]] > *V*[*i-1,j*] then

V[*i,j*] := *v*[*i*] + *V*[*i-1,j-w*[*i*]]; *P*[*i,j*] := *j-w*[*i*]

else

V[*i,j*] := *V*[*i-1,j*]; *P*[*i,j*] := *j*

return *V*[*n,W*] and the optimal subset by backtracing

Running time and space:
O(nW).

L
P
U

Thank You !!!