Dynamic Programming
0/1 knapsack problem

0/1 Knapsack Problem

0/1 Knapsack Problem

Formal description: Given two n-tuples of positive numbers

$$\langle v_1, v_2, \ldots, v_n \rangle \quad \text{and} \quad \langle w_1, w_2 \ldots, w_n \rangle,$$

and $W > 0$, we wish to determine the subset $T \subseteq \{1, 2, \ldots, n\}$ (of files to store) that

$$\text{maximizes} \quad \sum_{i \in T} v_i,$$

$$\text{subject to} \quad \sum_{i \in T} w_i \leq W.$$

0/1 Knapsack Problem

- Optimization Problem
- Brute force: Try all 2^n possible subsets T

Divide and Conquer Approach?

Divide and Conquer approach

1. Partition the problem into subproblems.
2. Solve the subproblems.
3. Combine the solutions to solve the original one.

Knapsack Problem :

Subproblems are not independent

Divide and-conquer algorithm repeatedly solves the common subsubproblems.

Thus, it does more work than necessary!

Better Solution : Dynamic Programming

Solution using Dynamic Programming

- Step 1 : Decompose the problem into smaller problems
- Step 2 : Recursively define the value of an optimal solution in terms of solutions to smaller problems

Initial Settings: Set

$$
\begin{array}{lll}
V[0,w]=0 & \text{for } 0 \le w \le W, & \text{no item} \\
V[i,w]=-\infty & \text{for } w<0, & \text{illegal}
\end{array}
$$

Recursive Step: Use

$$V[i,w]=\max(V[i-1,w],v_i+V[i-1,w-w_i])$$

for $1 \le i \le n, 0 \le w \le W$.

- Step 3: Bottom-up computing V[i,w] (using iteration, not recursion).

Bottom: $V[0, w] = 0$ for all $0 \leq w \leq W$.

Bottom-up computation: Computing the table using

$$V[i, w] = \max(V[i-1, w], v_i + V[i-1, w-w_i])$$

row by row.

V[i,w]	w=0	1	2	3	⋯	⋯	W	
i= 0	0	0	0	0	...	...	0	bottom
1								
2								
⋮								
n								

up

Example

Let $W = 10$ and

i	1	2	3	4
v_i	10	40	30	50
w_i	5	4	6	3

V[i, w]	0	1	2	3	4	5	6	7	8	9	10
i = 0											
i = 1											
i = 2											
i = 3											
i = 4											

Example

Let $W = 10$ and

i	1	2	3	4
v_i	10	40	30	50
w_i	5	4	6	3

Bottom: $V[0, w] = 0$ for all $0 \leq w \leq W$.

Bottom-up computation: Computing the table using

$$V[i, w] = \max(V[i-1, w], v_i + V[i-1, w - w_i])$$

row by row.

V[i, w]	0	1	2	3	4	5	6	7	8	9	10	
i = 0	0	0	0	0	0	0	0	0	0	0	0	
i = 1	0											
i = 2	0											
i = 3	0											
i = 4	0											Profit V[n,W]

Example

Let $W = 10$ and

i	1	2	3	4
v_i	10	40	30	50
w_i	5	4	6	3

Bottom: $V[0, w] = 0$ for all $0 \le w \le W$.

Bottom-up computation: Computing the table using

$$V[i, w] = \max(V[i-1, w], v_i + V[i-1, w-w_i])$$

row by row.

V[i, w]	0	1	2	3	4	5	6	7	8	9	10
i = 0	0	0	0	0	0	0	0	0	0	0	0
i = 1	0	0	0	0	0	10	10	10	10	10	10
i = 2	0										
i = 3	0										
i = 4	0										

Profit V[n,W]

Example

Let $W = 10$ and

i	1	2	3	4
v_i	10	40	30	50
w_i	5	4	6	3

Bottom: $V[0, w] = 0$ for all $0 \leq w \leq W$.

Bottom-up computation: Computing the table using

$$V[i, w] = \max(V[i-1, w], v_i + V[i-1, w - w_i])$$

row by row.

V[i, w]	0	1	2	3	4	5	6	7	8	9	10
i = 0	0	0	0	0	0	0	0	0	0	0	0
i = 1	0	0	0	0	0	10	10	10	10	10	10
i = 2	0	0	0	0	40	40	40	40	40	50	50
i = 3											
i = 4											

Profit V[n,W]

Example

Let $W = 10$ and

i	1	2	3	4
v_i	10	40	30	50
w_i	5	4	6	3

Bottom: $V[0, w] = 0$ for all $0 \le w \le W$.

Bottom-up computation: Computing the table using

$$V[i, w] = \max(V[i-1, w], v_i + V[i-1, w-w_i])$$

row by row.

V[i, w]	0	1	2	3	4	5	6	7	8	9	10
i = 0	0	0	0	0	0	0	0	0	0	0	0
i = 1	0	0	0	0	0	10	10	10	10	10	10
i = 2	0	0	0	0	40	40	40	40	40	50	50
i = 3	0	0	0	0	40	40	40	40	40	50	70
i = 4	0										

Profit V[n,W]

Example

Let $W = 10$ and

i	1	2	3	4
v_i	10	40	30	50
w_i	5	4	6	3

Bottom: $V[0, w] = 0$ for all $0 \leq w \leq W$.

Bottom-up computation: Computing the table using

$$V[i, w] = \max(V[i-1, w], v_i + V[i-1, w - w_i])$$

row by row.

V[i, w]	0	1	2	3	4	5	6	7	8	9	10
i = 0	0	0	0	0	0	0	0	0	0	0	0
i = 1	0	0	0	0	0	10	10	10	10	10	10
i = 2	0	0	0	0	40	40	40	40	40	50	50
i = 3	0	0	0	0	40	40	40	40	40	50	70
i = 4	0	0	0	50	50	50	50	90	90	90	90

Profit V[n,W]

Optimal Subset

incl[i,w] = 1 if i is included
= 0 otherwise

V[i, w]	0	1	2	3	4	5	6	7	8	9	10
i = 0	0	0	0	0	0	0	0	0	0	0	0
i = 1	0	0	0	0	0	10	10	10	10	10	10
i = 2	0	0	0	0	40	40	40	40	40	50	50
i = 3	0	0	0	0	40	40	40	40	40	50	70
i = 4	0	0	0	50	50	50	50	90	90	90	90

Profit V[n,W]

incl	0	1	2	3	4	5	6	7	8	9	10
i = 0	0	0	0	0	0	0	0	0	0	0	0
i = 1	0	0	0	0	0	1	1	1	1	1	1
i = 2	0	0	0	0	1	1	1	1	1	1	1
i = 3	0	0	0	0	0	0	0	0	0	0	1
i = 4	0	0	0	1	1	1	1	1	1	1	1

Let $W = 10$ and

Optimal Subset

i	1	2	3	4
v_i	10	40	30	50
w_i	5	4	6	3

incl[i,w] = 1 if i is included
= 0 otherwise

incl	0	1	2	3	4	5	6	7	8	9	10
i = 0	0	0	0	0	0	0	0	0	0	0	0
i = 1	0	0	0	0	0	1	1	1	1	1	1
i = 2	0	0	0	0	1	1	1	1	1	1	1
i = 3	0	0	0	0	0	0	0	0	0	0	1
i = 4	0	0	0	1	1	1	1	1	1	1	1

Optimal Subset

Let $W = 10$ and

i	1	2	3	4
v_i	10	40	30	50
w_i	5	4	6	3

incl	0	1	2	3	4	5	6	7	8	9	10
i = 0	0	0	0	0	0	0	0	0	0	0	0
i = 1	0	0	0	0	0	1	1	1	1	1	1
i = 2	0	0	0	0	1	1	1	1	1	1	1
i = 3	0	0	0	0	0	0	0	0	0	0	1
i = 4	0	0	0	1	1	1	1	1	1	1	1

w = W

for i = n to 1

If incl[i, w] ==1

include item i

$w = w - w_i$

w = 10

i = 4; incl[4, 10] = 1

item 4 included; w = 7

i = 3; incl[3, 7] = 0

i = 2; incl[2, 7] = 1

item 2 included; w = 3

i = 1; incl[1, 3] = 0

Optimal Subset { 2,4}

Profit V[n,W]

Algorithm

```
KnapSack(v, w, n, W)
{
    for (w = 0 to W) V[0, w] = 0;
    for (i = 1 to n)
        for (w = 0 to W)
            if ((w[i] ≤ w) and (v[i] + V[i − 1, w − w[i]] > V[i − 1, w]))
            {
                V[i, w] = v[i] + V[i − 1, w − w[i]];
                  incl[i, w] = 1;
            }
            else
            {
                V[i, w] = V[i − 1, w];
                  incl[i, w] = 0;
            }
    K = W;
    for (i = n downto 1)
        if ( incl [i, K] == 1)
        {
            output i;
            K = K − w[i];
        }
    return V[n, W];
}
```

Exercise

1.

Item	Weight	value
1	2	12
2	1	10
3	3	20
4	2	15

W = 5

2.

Item	Weight	value
1	7	42
2	3	12
3	4	40
4	5	25

W = 10