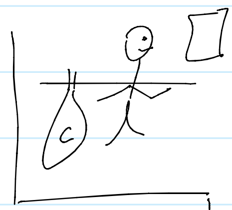
Knapsack Problem

Sunday, October 9, 2022 7:54 PM

Recall: Dynamic Programming

1] 0/1 knapsack Problem



u_1	u_2	u_3	u_4	u_5	
0	0	0	0	0	
2	3	4	5	7	KG
3	4	5	7	8	\$

Given a set of items
$U = \{u_1, u_2, u_3, \ldots, u_n\}$
and knapsack S of capacity C.
Each item u_i has a size s_i
and a value $v_i \in \mathbb{Z}^+$

Find the max value that can be packed

$$\max \sum_{u_i \in S} v_i \quad \text{s.t.} \quad \sum_{u_i \in S} s_i \leq C$$

2] Greed Algorithm

1. Compute the ratio $r_i = \frac{v_i}{s_i}$ for all items
2. Sort the items in a decreasing order by r_i

e.g.

	u_1	u_2	u_3	u_4	u_5		
	2	3	4	5	7	KG	
	3	4	5	7	8	\$	
$r_i = \frac{v_i}{s_i} \rightarrow$	$3/2$	$4/3$	. . . .		$8/7$		
	1.5	1.33	1.25	1.4	1.13	$\leftarrow$	$\theta(n)$
	1	3	4	2	5	$\leftarrow$	$\theta(n \log n)$

3] Note : the greedy algorith is fast

time complexity : $\Theta(n \log n)$ for sorting.

but does not always give the optimal solution. why?

(HW)

S:	6	5	5	kg.
v:	12	8	7	$
r:	2	1.6	1.4	
	1			

$C = 10$

4] Subproblem:

let $V[i,j]$ = max value obtained by filling a knapsack of capacity j with items $\underbrace{\{u_1, u_2, \dots, u_i\}}_{\text{first } i \text{ items}}$

1. $$V[i,j] = \begin{cases} 0 & \text{if } i=0 \text{ or } j=0 \\ V[i-1, j] & \text{if } j < s_i \\ \max\{\underbrace{V[i-1, j]}_{\text{leave it}}, \underbrace{V[i-1, j-s_i]+v_i}_{\text{take it}}\} & \underline{\text{otherwise}} \end{cases}$$

2. Return $V[n, C]$

5] e.g.

$u_i \rightarrow$	u_1	u_2	u_3	u_4
size: (s_i)	2	3	4	5
Value: (v_i)	3	4	5	7

$C = 9$,

V

$i \downarrow$ / $j \rightarrow$	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	0	3	3	3	3	3	3	3	3
2	0	0	3	+4 4	4	+4 7	7	7	7	7
3	0	0	3	4	+5 5	7	+5 8	+5 9	9	+5 12

1	0		3	3				3		3
2	0	0	3	4 4	4	+4 7	7	7.	7	7
3		0	3	4	-5 5	7	+5 8	+5	9	+5 12
4	0		3	4	5	7	8	+7 10	11	12

```
Algorithm Knapsack
Input: A set of items U = {u1,u2,…,un} with sizes s1,s2,…,sn and
   values v1,v2,…,vn, respectively and knapsack capacity C.
```

Output: the maximum value of $\sum_{u_i \in S} v_i$ subject to $\sum_{u_i \in S} s_i \le C$

```
  for i := 0 to n do        ] n+1 steps
    V[i,0] := 0;
  for j := 0 to C do        ] C+1 steps
    V[0,j] := 0;
  for i := 1 to n do
    for j := 1 to C do
      V[i,j] := V[i-1,j];
      if si ≤ j then
        V[i,j] := max{V[i,j], V[i-1,j-si]+vi}
    end for;
  end for;
  return V[n,C];
```

C n   n C steps

6) Time Complexity

To fill the V table, we have $(n+1) \times (C+1)$ cells in the table, each may need one comparison.

∴ the time complexity is $\Theta(nC)$

Pseudo-polynomi in t r s o- nput ize.

7) Space Complexity :

for this algorith, we use $\Theta(nC)$ space

However, we only need the last row of the table

So, the algorithm can be optimized to use $\Theta(C)$ space.