# 0-1 Knapsack Problem

Given _____, _____, and arrays _____, _____

Find _____ (rarely _____)

Have _____ of each item.

( , , ) , ( , , ) , ( , , ) , ( , , )

How do we solve it?

1. _____
   _____
2. _____
   _____
3. _____
   _____

Dynamic Programming ( DP ) Solution

Solve all smaller problems (from problem size 0 to current problem size) => table

Observations:

- Does order of picked items matter? (e.g. 1,2,3 vs 2,1,3?) _____
-
- What is a problem of size 0? _____, _____
- What is a problem of size 1? Can I have a single problem of size 1 (regarding items)?

Implementation:

*row/column indexes* = 0

*item's weight > current weight (column idx)* = value of cell 1 row above current cell

*item's weight ≤ current weight (column idx)* = max between value of cell 1 row above current cell & value of current item + value of cell 1 row above it and current weight - current item's weight columns before current cell

Equations:

sol = matrix/2D array storing answers for (all) smaller pbs TC:

w[i] = weight of current item SC:

v[i] = value of current item

k = current weight (column indexes)

sol[0][k] = _____, for _____

sol[i][0] = _____ , for _____

sol[i][k]:

If k < w[i] : _____

If k ≥ w[i] : _____

| ID | Weight | Value |
|---|---|---|
| 1 | | |
| 2 | | |
| 3 | | |
| 4 | | |

Max Weight =

sol table below. Indicate in table picked item with "*" and not picked with "."

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |

sol[0][5] = _____

sol[1][4] = _____

sol[3][5] = _____

sol[4][8] = _____

sol[3][5] = max {_____ , _____}

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | |

Backtracking:

Final Value Achieved: _____ Items Picked: _____

DP solution variations:

- Space saving:

Knapsack problem variations:

1. _____
2. _____
3. _____

```
/*Arrays v and w have info from index 1: first item has value v[1] and weight w[1]
*/
int knapsack01(int W, int n, int * v, int * w){
   int sol[n+1][W+1];
   for(k=0; k<=W; k++) { sol[0][k] = 0;}
   for(i=1; i<=n; i++) {
       for(k=0;k<=W;k++) {
            sol[i][k] = sol[i-1][k]; // solution without item i
              if (k>w[i]) {
                with_i = v[i]+sol[i-1][k-w[i]];
                   if (sol[i][k] < with_i) {   // better choice
                     sol[i][k] = with_i;       // keep it
                }
            }
         }// for k
      }// for i
      return sol[n][W];
}     // Time: Θ(_____)   Space: Θ(_____)   pseudo polynomial in W
      // need Θ(n) bits to store n items (values and weights) , but only log2(W) bits to store value W
```

Greedy:

| Max Weight = | | |
|---|---|---|
| ID | Weight | Value |
| 1 | | |
| 2 | | |
| 3 | | |
| 4 | | |

Criterion: _____

Work:

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

Criterion: _____

Work:

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

Optimality:

0/1 Knapsack:

Fractional Knapsack:

Other variations:

Job Scheduling:

| Criteria | Max **Total Value/Profit** | Max **Number of Jobs** |
|---|---|---|
| Max Value/Length | | |
| Max Value | | |
| Min Length | | |
| Finishes Last | | |
| Starts First | | |
| Finishes First | | |
| Starts Last | | |

Difference between Greedy and Dynamic Programming:

| Greedy: | Dynamic Programming |
|---|---|
| | |

Brute Force

Idea:

How many possible combinations of items are there for N items? _____

How to generate all possible combinations of items?

- Idea
- TC