Dynamic Programming: knapsack

Planning for Mars One Mission

- Mars one spaceship has a fixed weight limit
- There are a lot of different things to choose from to bring on board
- They each has a weight and a value
- How can we maximize the value of the things we bring while respecting the weight limit?

0-1 knapsack problem definition

- Input: a weight limit W, and a set of items with their specific weights and values:

Item	Weight (w)	Value (v)
1	w_1	v_1
2	w_2	v_2
…	…	…
n	w_n	v_n

- Goal: find a subset S$\subset \{1,2,\ldots,n\}$ that maximizes $\sum_{i\in S} v_i$ and satisfies $\sum_{i\in S} w_i \leq W$

Also called knapsack problem without repetition

Brute force?

- Enumerate all possible subsets

- Time complexity?

Is it good to be greedy?

- Greedily choose the most valuable item?

- Greedily choose the most value/weight item?

We can construct examples where such greedy strategies won't return optimal solution.

Optimal substructure

- Our problem size is determined by two things:
  - n, the number of items to choose from
  - W, the weight limit (if W=0, the problem is trivial)

- Define $V(n, W) =$ the optimal value achievable using item 1,2,…, n, with weight limit W

- How can we create smaller subproblems that help us solve this original problem?

- Let's focus on the last object. There are two possible options for this object:

Option 1: Use this object:

- we will reduce n to n-1
- W will reduce to $W - w_n$

Subproblem: $V(n-1, W - w_n)$

The best value achievable for this option is:

$$V(n-1, W - w_n) + v_n$$

Option 2. Do not use this object:

- We will reduce n to n-1
- W will remain the same

Subproblem:
$V(n-1, W)$

The best value achievable with this option:

$$V(n-1, W)$$

Considering both options

$$V(n,W) = \max \begin{cases} V(n-1, W-w_n) + v_n \\ V(n-1, W) \end{cases}$$

Use item n

Discard item n

Edge case: $w_n > W$, we can only discard item n

Base case?

$V(i,0) = 0 \text{ for } i = 0, \dots, n$

$V(0,j) = 0 \text{ for } j = 0, \dots, W$

```
Knapsack(n,W)
for i=0 to n          V(i,0)=0
for w=1 to W          V(0,w)=0

for i=1 to n
      for w=1 to W
            V(i,w) = max { V(i-1,w-w_i)+v_i
                         { V(i-1,w))

Return V(n,W)
```

* Ignoring edge cases

Runtime? O(nW)

Memoization

```
Knapsack_m(n,W)
if n=0 V(n,W)=0
if W=0 V(n,W)=0
if V(n,W) is undefined
```

$$V(n,W) = \max \begin{cases} \text{Knapsack\_m}(i-1, w-w_i)+v_i \\ \text{Knapsack\_m}(i-1, w)) \end{cases}$$

```
Return V(n,W)
```

* Ignoring edge cases

Same asymptotic runtime: $O(nW)$

Practical Advantage: we do not always have to go through all w values, skipping many unnecessary subproblems

Unbounded knapsack problem

- Same general premise, but we have an unbounded number of all items
- Input: a weight limit W, and n possible items, each item has unbounded supply
- Goal: achieve maximum value within weight limit

Subproblem

- Because of the infinite supply, we will not be able to reduce the size n
- But we can reduce the weight limit by putting items in the sack
- Consider the first selection of item to place in the sack, we have n choices, one for each item

- Suppose your first choice is item i:

We would get its value v_i

And we will reach a smaller weight limit (subproblem): $W-w_i$

How do you proceed from here?