Knapsack problem:

Backpack with total capacity B

& n objects with:

- integer weights $w_1, \ldots, w_n$
- integer values $v_1, \ldots, v_n$

Goal: Find subset S of objects that:

a) fits in the backpack

& b) maximizes the total value.

In other words, find $S \subset \{1, \ldots, n\}$ where:

a) $\sum_{i \in S} w_i \leq B$

& b) maximizes $\sum_{i \in S} v_i$

Application: scheduling jobs.

Two versions:

1) Without repetition: one copy of each object
2) With repeats: unlimited supply of each object.

Version 1: one copy of each object.

What about greedy approach?

Example: 4 objects: 1, 2, 3, 4

Values: 15, 10, 8, 1

weights: 15, 12, 10, 5

total capacity $B = 22$

Greedy: take most valuable/per unit of weight

Sort by $r_i = \frac{v_i}{w_i}$. Note, $r_1 > r_2 > r_3 > r_4$.

greedy solution: objects 1 & 4

total value = 16.

optimal solution: objects 2 & 3

total value = 18.

Dynamic Programming approach:

Step 1: Define the subproblem in words.

initial attempt: Prefix of input.

Let $K(j)$ = max value achievable using a subset of objects $1, \dots, j$.

Step 2: Express $K(j)$ in terms of $K(1), \dots, K(j-1)$.

Consider $K(j)$: two options – use object j or not.

If don't use object j, then $K(j) = K(j-1)$.

If use object j, then can we add j to the optimal for $K(j-1)$?

Need to know how much weight is available?

if include j then want optimal solution with weight $\leq B - w_j$.

So need to keep track of weight available.

Attempt 2—

Step 1: Subproblem definition

For b & j where $0 \le b \le B$ & $0 \le j \le n$,

let $K(b,j)$ = max value achievable using a subset of objects $1,\ldots,j$ & total weight $\le b$.

Goal: compute $K(B,n)$.

Step 2: recurrence relation—

For $K(b,j)$:

either:

- Use object j then want optimal solution for subset of objects $1,\ldots,j-1$ with total weight $\le b-w_j$

  then $K(b,j) = K(b-w_j, j-1) + v_j$

- Don't use object j

  then $K(b,j) = K(b, j-1)$

Therefore,

if $w_j \le b$,

$$K(b,j) = \max\{v_j + K(b-w_j, j-1),\ K(b,j-1)\}$$

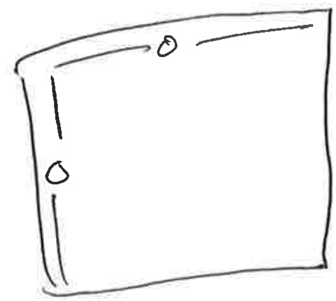
if $w_j > b$,

$$K(b,j) = K(b,j-1).$$

Base cases:

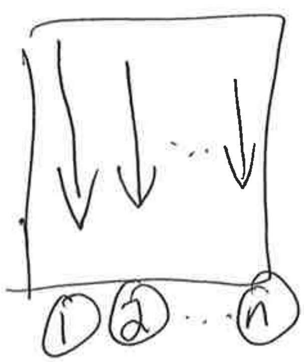
$$K(b,0) = 0$$

$$K(0,j) = 0$$

$K =$ 

Recurrence for $K(b,j)$ uses $K(?, j-1)$

So fill table from $j = 0 \to n$

$K =$ 

KnapsackNoRepeat$(B, w_1, \ldots, w_n, v_1, \ldots, v_n)$:

For $j = 0 \to n$: $K(0, j) = 0$

For $b = 0 \to B$: $K(b, 0) = 0$

For $j = 1 \to n$

  For $b = 1 \to B$

    if $w_j > b$,

    then $K(b, j) = K(b, j-1)$.

    else $K(b, j) = \max\{ v_j + K(b - w_j, j-1), K(b, j-1) \}$

Return$(K(B, n))$

<u>Running time</u>:

outer for loop of size $O(n)$

$\times$ inner loop of size $O(B)$

$\Rightarrow O(nB)$ time.

Version 2: unlimited supply of each object

Try same subproblem as before:

$K(b,j)$ = max value achievable using subset of objects $1,\dots,j$ (possibly with repeats) & total weight $\le b$.

if $w_j \le b$:

either:

- don't include j so $K(b,j) = K(b,j-1)$.
- or include j, but how many times?

adding one more copy then:

$$K(b,j) = v_j + K(b-w_j, j)$$

j instead of $j-1$

so can use j again.

Therefore,

if $w_j \le b$,

$$K(b,j) = \max\{K(b,j-1),\ v_j + K(b-w_j, j)\}$$

if $w_j > b$,

$$K(b,j) = K(b,j-1).$$

<u>KnapsackRepeat</u> $(B, w_1, \ldots, w_n, v_1, \ldots, v_n)$:

For $j = 0 \to n$, $K(0,j) = 0$

For $b = 0 \to B$, $K(b,0) = 0$

For $j = 1 \to n$,

  For $b = 1 \to B$,

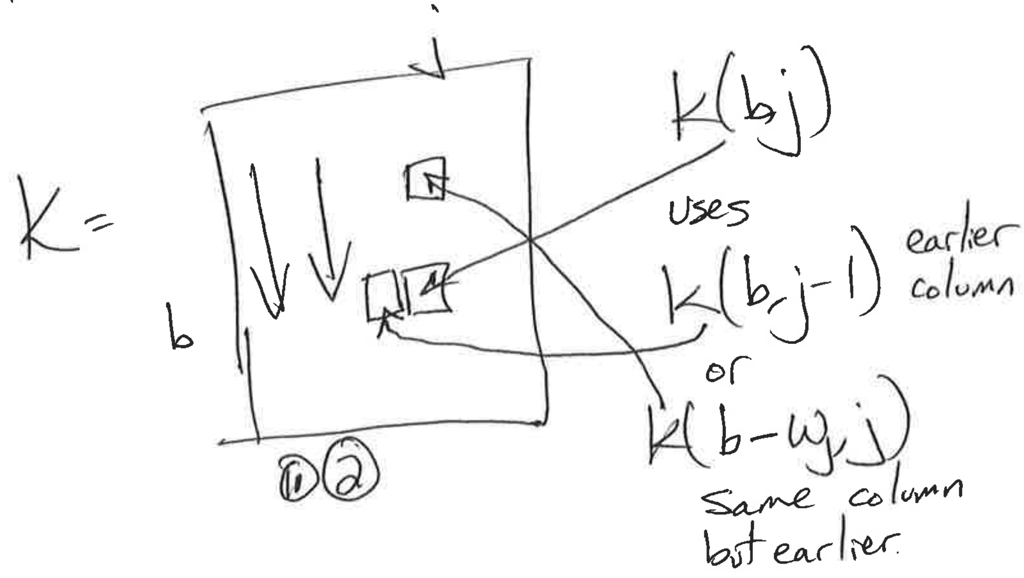
    if $w_j > b$

      then $K(b,j) = K(b,j-1)$

      else $K(b,j) = \max\{K(b,j-1),\ v_j + K(b-w_j, j-1)\}$

Return $(K(B,n))$.

Filling the table:



Running time: $O(nB)$ as before.

Alternative DP for version with repeats
- Don't need to keep track of which objects considered.

1. Let $K(b)$ = max value achievable using total weight $\le b$ & all objects $1, \dots, n$ allowed

2. Recurrence: try all possibilities for "last" object added.
so try all ℓ where $1 \le \ell \le n$ & $w_\ell \le b$

Hence, $K(b) = \max\limits_{\ell} \{ K(b - w_\ell) + v_\ell : 1 \le \ell \le n, w_\ell \le b \}$

K is one-dimensional but each entry takes $O(n)$ time to fill in
so $O(nB)$ time.

Our algorithm had running time $O(nB)$.

Let $V = \sum_{i=1}^{n} v_i$ = total value of all objects.

Consider the version without repeats then the max value is $\leq V$.

<u>Exercise (hard)</u>: Design DP algorithm with running time $O(nV)$.

Also, discussion about the fact that knapsack (we'll see later in the course) is NP-hard.