S. KIMMEL

Goals

- Create an algorithm for Knapsack
- Prove correctness of dynamic programming alg.
- Describe why prefix-free codes are good.

Knapsack Problem

Input: • n items, each has
- value v_j
- size w_j

$\left.\begin{matrix} \\ \\ \end{matrix}\right\} \mathbb{Z}^+$

• Capacity W

Output: A subset $S \subseteq \{1, 2, \ldots, n\}$ that maximizes $\sum_{i \in S} v_i = V(S)$ and satisfies $w(S) = \sum_{i \in S} w_i \leq W$

$w(S) = \sum_{i \in S} w_i \leq W$: constraints

$\sum_{i \in S} v_i = V(S)$: objective function

$K_{i,r}$ = optimal subset if only 1st i items are allowed, capacity r.

1. Form of optimal solution S of $K_{n,w}$
   (i) $n \notin S$ (ii) $n \in S$

2. Optimal Soln in terms of smaller Soln
   (i) If $n \notin S$, S is optimal for $K_{n-1,w}$
   (ii) If $n \in S$, $S - \{n\}$ is optimal for $K_{n-1,w-w_n}$
   (Proof by contradiction)

S.KIMMEL

3. Recurrence Relation for $V_{i,r}$ = value of objective function of $K_{i,r}$

4. Write code to store $V_{i,r}$ in array

5. Write code to work backwards through array to recover $K_{n,w}$

S. KIMMEL

3 Recurrence Relation for objective function

Base Cases: $V_{0,r} = 0 \quad \forall r \in \{0, \dots W\}$

$$V_{i,R} = \max\left\{ V_{i-1,R} \ (i), \ V_{i-1,R-w_i} + v_i \ (ii) \right\}$$

4 Create a for-loop to fill out. (include base case).

```
for r = 0 to W:
    V[0, r] = 0

for i = 1 to n:
    for r = 0 to W:
        V[i, r] = max { V[i-1, r], V[i-1, r - w_i] + v_i }
```

This option Not allowed if $r - w_i < 0$

S. KIMMEL

5 Write pseudocode to get S using array V

```
S = ∅
i = n
r = W
while i > 0:
    if V[i, r] = V[i-1, r-w_i] + v_i
        S = S + i
        r = r - w_i
    i --
```

Need check that doesn't go out of bounds

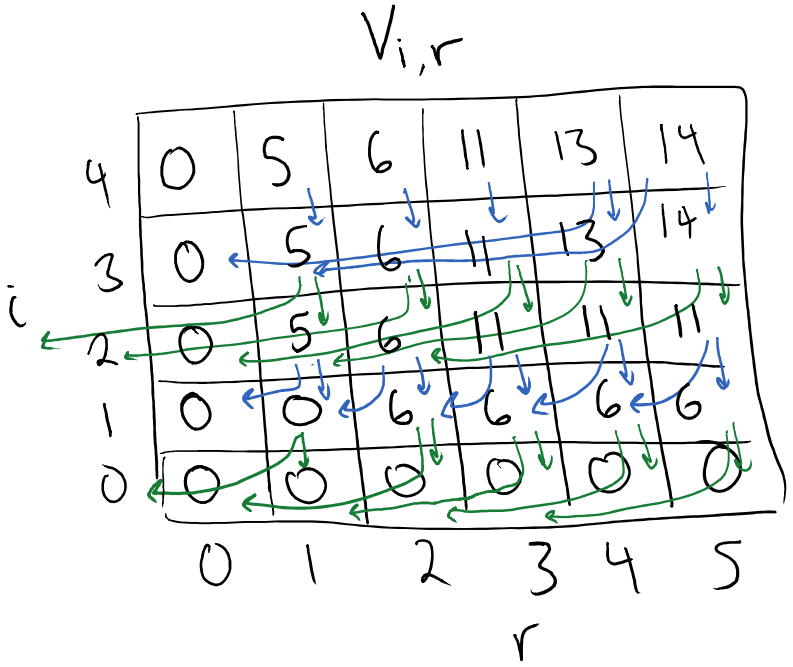
EX: $W = 5$

$v_1 = 6 \quad w_1 = 2$

$v_2 = 5 \quad w_2 = 1$

$v_3 = 8 \quad w_3 = 3$

$v_4 = 7 \quad w_4 = 4$



Q: What is the runtime of DP knapsack algorithm?

A) $O(n)$ B) $O(W)$ C) $O(n+W)$ D) $O(nW)$