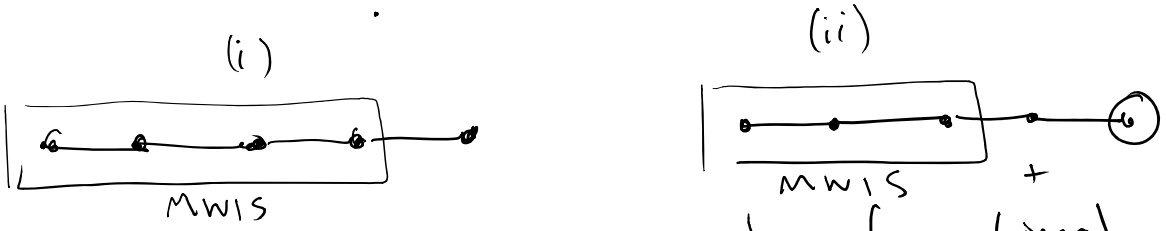
S. KIMMEL

To Create D. P. (dynamic programming) algorithm:

1. Think of form of optimal solution.
   - WMIS on line: $v_n \in S$ (ii) or $n \notin S_n$ (i)

2. Optimal soln in terms of smaller soln? (Multiple cases)



3. Create recurence relating value of optimal solution to smaller solutions.

$$A[k] = \max\{\underbrace{A[k-1]}_{(i)}, \underbrace{A[k-2] + w_k}_{(ii)}\}$$

$A[k]$: weight of MWIS on G_k

4. Store values in an array using for-loop

5. Work backwards through array to reconstruct optimal solution

S. KIMMEL

Dynamic Programming - More Practice

Knapsack Problem $K(v_1, \dots v_n, w_1, \dots w_n, W)$

Input:
- n items, each has
  - value v_i
  - size w_i

  $\} \in \mathbb{Z}^+$
- Capacity W

Output: A subset $S \subseteq \{1, 2, \dots, n\}$ 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

Applications

- Cargo trucks
- Investments
  - v_i = expected yield %
  - w_i = min investment amount
  - W = amount to invest

S. KIMMEL

Some notation

- Let $K_{i,R}$ be subproblem on 1st i items, with capacity R.
- Solution to $K_{i,R}$ is a set $S \subseteq \{1, 2, \ldots, i\}$: $W(S) \leq R$.
- Optimal solution is solution with largest $V(S)$

1. Let S be the optimal solution $K_{n,W}$

Only two possibilities { (i) $n \notin S$ → 2. How to relate to optimal solution of smaller problem?
(ii) $n \in S$ →

(i) If S is optimal soln to $K_{n,W}$ and $n \notin S$, then S is optimal soln to $K_{n-1,W}$

Pf: For contradiction, suppose S is not opt soln for $K_{n-1,W}$ but S is optimal soln to $K_{n,W}$ and $n \notin S$. Note S is a solution to $K_{n-1,W}$ but since it is not optimal, $\exists$ a a soln S' for $K_{n-1,W}$ such that $V(S') > V(S)$. But S' is also a solution to $K_{n,W}$, and since $V(S') > V(S)$, S is not optimal for $K_{n,W}$, a contradiction.

(Recall, if trying to prove $P \rightarrow Q$ by contradiction, assume $P \wedge \neg Q$ then try to show $\neg P$.

(ii) If S is opt. soln for $K_{n,w}$ and $n \in S$, then $S - \{n\}$ is opt soln for $K_{n-1, w-w_n}$.

Pf: Suppose for contradiction $S - \{n\}$ is not opt, but S is opt. for $K_{n,w}$ and $n \in S$. Then $\exists$ S', a soln for $K_{n-1, w-w_n}$ s.t. $V(S') > V(S - \{n\})$. But then $S' + \{n\}$ is a soln to $K_{n,w}$, and $V(S' + \{n\}) > V(S)$, so S is not opt. for $K_{n,w}$, a contradiction

Consequence: Let $V[i,r]$ be value of optimal soln to $K_{i,r}$
If S is opt for $K_{i,r}$ and $i \notin S$, $\Rightarrow$

$$V[i-1, r] = V[i, r]$$

If S is opt for $K_{i,r}$ and $i \in S$ $\Rightarrow$

$$V[i-1, r-w_i] = V[i,r] - v_i$$

3 Recurrence Relation

Base Case: $\boxed{V[0,r]=0}$

$V[i,0]=0$

$$V[i,r] = \max\{V[i-1,r],\ V[i-1, r-w_i]+v_i\}$$

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

What is run time?

```
for r = 0 to W:
    V[0, r] = 0
for i = 1 to n
    V[i, 0] = 0
for i = 1 to n
for r = 1 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

Pf of correctness: What are we trying to prove?

Let $V_{i,r}$ be optimal value for $K_{i,r}$.

Loop Invariant: $V[i',r'] = V_{i',r'}$ for all $0 \le i' < i$, $0 \le r' \le r$.

Initialization: $i = 1$ $r = 1$ ✓

Maintenance: Since $i-1 < i$, $V[i-1, r] = V_{i-1,r}$

$V[i-1, r-w_i] = V_{i-1, r-w_i}$

Now one of (i) or (ii) must be true, so using previous proof, either $V[i,r] = V[i-1,r]$ or $V[i,r] = V[i-1, r-w_i] + v_i$

but want optimal, so choose larger.

Termination: All $V[i,r] = V_{i,r}$ for $1 \le i \le n$, $1 \le i \le W$

S. KIMMEL

5 Write pseudocode to get S using array V

What is runtime?

```
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

S. KIMMEL

Pf that loop is correct. Let S be opt solution. Can assume $V[i,w]$ is value of opt. sol. on $K_{i,w}$

Loop Invariant: • S' contains all elements of S larger than i.

• $r = W - W(S')$

Initially: $i = n$, $S = \phi$, $r = W$

Maintenance: Suppose loop invariant is true going into loop. Then S' contains elements of S larger than i, so we just need to figure out those less than or equal to i. Since we've already used up capacity $W(S')$, we need to solve $K_{i, W-W(S')} = K_{i,r}$. to figure out remaining elements

We now determine if $i \in$ opt. Soln for $K_{i,r}$. Only two options (i), or (ii) and at least one is true, so either $V[i,r] = V[i-1, r]$ or $V[i,r] = V[i-1, r-w_i] + v_i$, and which one it is tells us whether to add i or not. In either case invariant is maintained