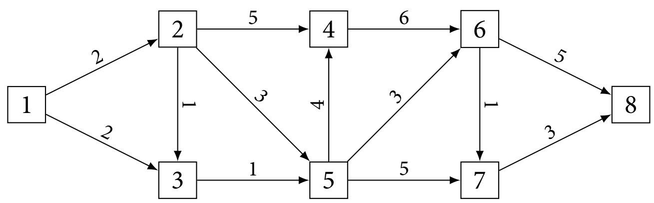
Lesson 13.

The principle of optimality and formulating DP recursions

0 Warm up

Example 1. Consider the following directed graph. The labels on the edges are edge lengths.



In this order:

a. Find the shortest path from node 1 to node 8.

b. Find the shortest path from node 3 to node 8.

c. Find the shortest path from node 5 to node 8.

1 The principle of optimality

- In Example 1, we found that the shortest path from node 1 to node 8 is $1 \to 3 \to 5 \to 6 \to 7 \to 8$ with length 10
- Now, consider paths from node 3 to node 8
  - For example, $3 \to 5 \to 6 \to 7 \to 8$ is such a path with length 8
- Could there be a shorter path from node 3 to node 8?
  - Suppose we had a path from 3 to 8 with length < 8
  - Consider edge $(1, 3)$ + this path

$\Rightarrow$

The principle of optimality (for shortest path models)

In a directed graph with no negative cycles, optimal paths must have optimal subpaths.

- Consider a directed graph (N, E) with target node $t \in N$ and edge lengths c_{ij} for $(i, j) \in E$
- By the principle of optimality, the shortest path from node i to node t must be:

edge (i, j) + shortest path from j to t for some $j \in N$ such that $(i, j) \in E$

- How can we exploit this?
- Let $f(i) =$
- Then we can write the following **boundary conditions** and **recursion**
- For example, in Example 1, $f(5)$ is

2 Formulating DP recursions

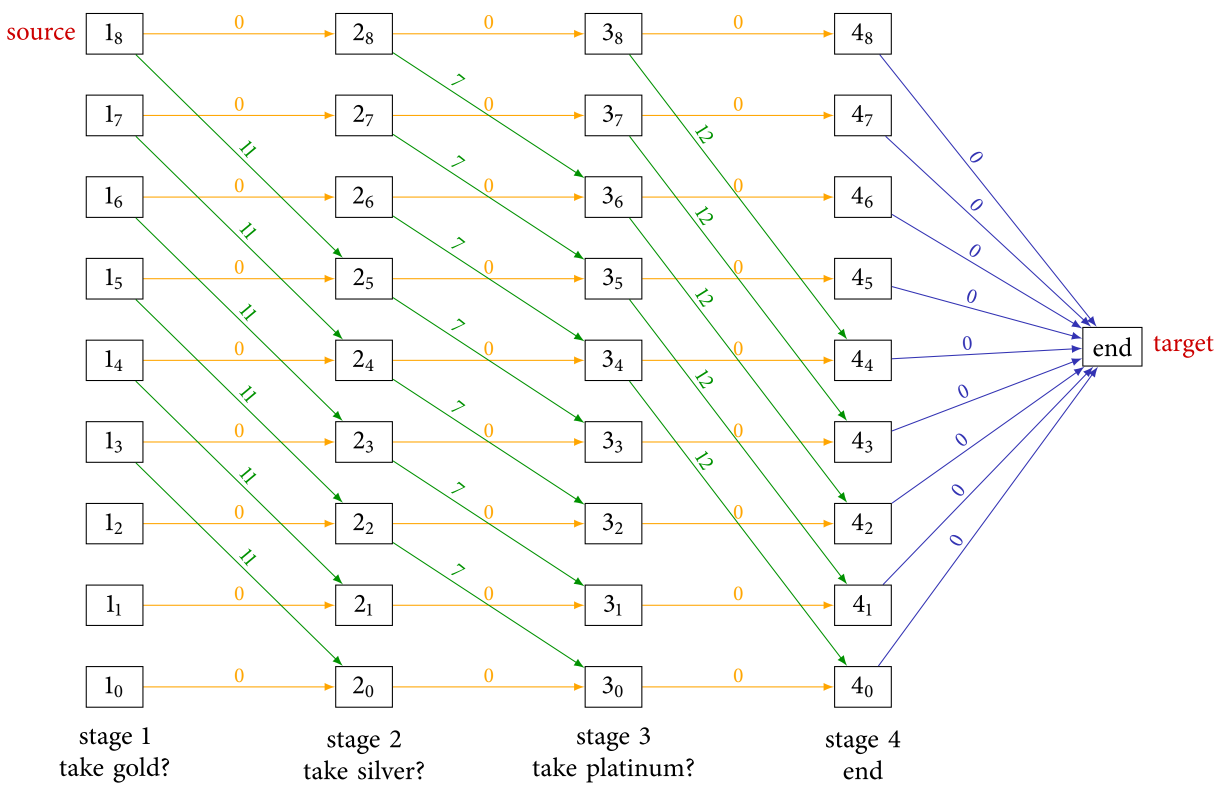
- Dynamic programs are not usually given as shortest/longest path problems as we have done over the past few lessons
- Instead, DPs are usually given as **recursions**
- Let's revisit the following knapsack problem that we studied back in Lesson 7

Example 2. You are a thief deciding which precious metals to steal from a vault:

	Metal	Weight (kg)	Value
1	Gold	3	11
2	Silver	2	7
3	Platinum	4	12

You have a knapsack that can hold at most 8kg. If you decide to take a particular metal, you must take all of it. Which items should you take to maximize the value of your theft?

- We formulated the following dynamic program for this problem by giving the following longest path representation:



- Let's formulate this as a dynamic program, but now by giving its recursion representation
- Let

$$w_t = \text{weight of metal } t \qquad v_t = \text{value of metal } t \qquad \text{for } t = 1, 2, 3$$

- Stages:

- States:

- Allowable decisions x_t at stage t and state n:

- Reward of decision x_t at stage t and state n:

- Reward-to go function $f_t(n)$ at stage t and state n:

- Boundary conditions:

- Recursion:

- Desired reward-to-go function value:

- In general, to formulate a DP by giving its recursive representation:

Dynamic program – recursive representation

- **Stages** $t = 1, 2, \dots, T$ and **states** $n = 0, 1, 2, \dots, N$
- Allowable **decisions** x_t at stage t and state n $\quad (t = 1, \dots, T-1;\ n = 0, 1, \dots, N)$
- **Cost** of decision x_t at stage t and state n $\quad (t = 1, \dots, T;\ n = 0, 1, \dots, N)$
- **Cost-to-go** function $f_t(n)$ at stage t and state n $\quad (t = 1, \dots, T;\ n = 0, 1, \dots, N)$
- **Boundary conditions** on $f_T(n)$ at state n $\quad (n = 0, 1, \dots, N)$
- **Recursion** on $f_t(n)$ at stage t and state n $\quad (t = 1, \dots, T-1;\ n = 0, 1, \dots, N)$

$$f_t(n) = \min \Big\{ (\text{cost of decision at stage } t) + f_{t+1}(\text{new state at stage } t+1) \Big\}$$

- **Desired cost-to-go function value**

3 Solving DP recursions

- To improve our understanding of how this recursive representation works, let's solve the DP we just wrote for the knapsack problem
- We solve the DP backwards:
  - start with the boundary conditions in stage T
  - compute values of the cost-to-go function $f_t(n)$ in stages $T-1, T-2, \dots, 3, 2$
  - …until we reach the desired cost-to-go function value
- Stage 4 computations – boundary conditions:

- Stage 3 computations:

$f_3(8) =$

$f_3(7) =$

$f_3(6) =$

$f_3(5) =$

$f_3(4) =$

$f_3(3) =$

$f_3(2) =$

$f_3(1) =$

$f_3(0) =$

- Stage 2 computations:

$f_2(8) =$

$f_2(7) =$

$f_2(6) =$

$f_2(5) =$

$f_2(4) =$

$f_2(3) =$

$f_2(2) =$

$f_2(1) =$

$f_2(0) =$

- Stage 1 computations – desired cost-to-go function: