

Lesson 9.

# Solving dynamic programs with networkx

## Overview

- In this lesson, we'll revisit a few examples of dynamic programs and solve them with networkx.

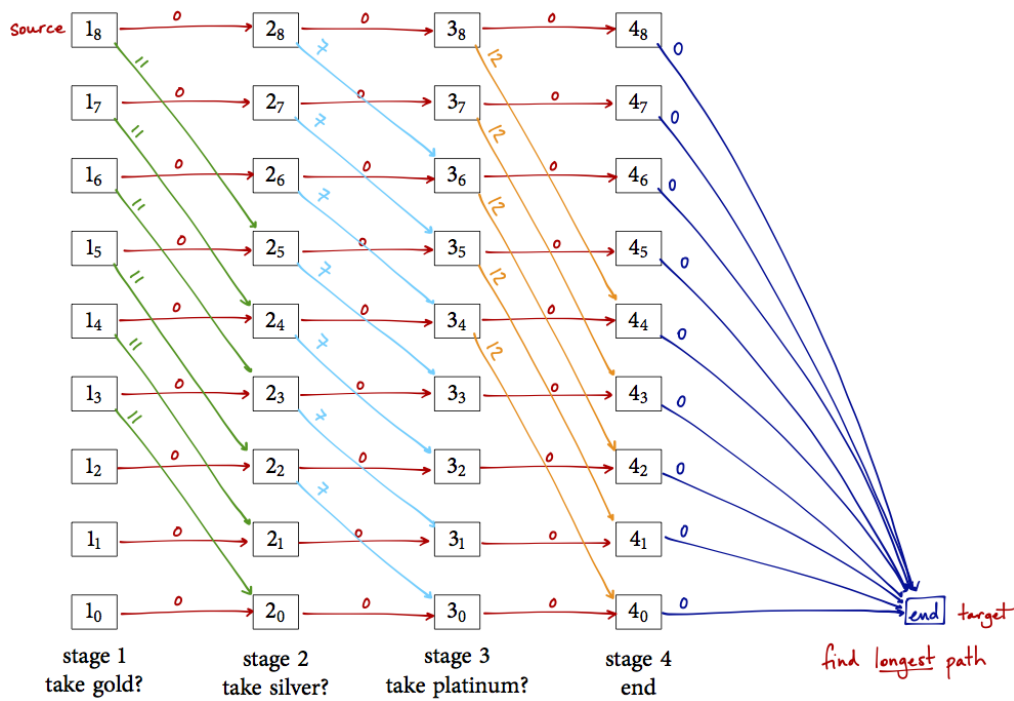
## The knapsack problem, revisited

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 8 kg. 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?

- Recall that we formulated this problem as a dynamic program with the following longest path representation:
  - Stage  $t$ : consider item  $t$
  - Node  $t_n$ :  $n$  kgs left in knapsack at stage  $t$  (with items  $t, t + 1, \dots$  remaining)



DP for knapsack example

- We know how to solve shortest/longest path problems using `networkx`, so we can apply the same ideas here.
- There is a Python data structure that makes this a little easier...

## Tuples

- A **tuple** is like a list, except once it's been defined, it cannot be changed.
- A tuple is written as a sequence of comma-separated items between *round* brackets. For example:

```
In [2]: # Define a tuple corresponding to taking
        # silver with 5 kgs left in the knapsack
        stage = (2, 5)
```

- Tuples are ideal for things like names of nodes — things that you want to make permanent and not accidentally change.

## Back to the knapsack problem...

- We can use a tuple to represent the name of each node in our dynamic program, since each node's name has two distinct parts: the stage and the state.
- Before we do anything, we need to import `networkx` and `bellmanford`:

```
In [3]: import networkx as nx
        import bellmanford as bf
```

- Let's begin by creating an empty graph:

```
In [4]: # Create empty graph
        G = nx.DiGraph()
```

- Next, let's add the stage-state nodes to the graph, using `for` loops. Remember that `range(a, b)` iterates over the integers `a`, `a + 1`, ..., `b - 1`.

```
In [5]: # Add the stage-state nodes
        for t in range(1, 5):
            for n in range(0, 9):
                G.add_node((t, n))
```

- We also need to add the special "end" node:

```
In [6]: # Add the end node
        G.add_node("end")
```

- Now we need to add the edges.
- There are a lot of them, so we'll want to use some `for` loops.
- The best way to use `for` loops depends on the shortest/longest path representation of the DP.
- For example, looking above, we can add all the red edges of length 0 — corresponding to not taking the item — in one fell swoop, like this:

```
In [7]: # Add edges corresponding to not taking an item
# (red edges of length 0)
for t in range(1, 4):
    for n in range(0, 9):
        G.add_edge((t, n), (t + 1, n), length=0)
```

- Next, we can add the green edges of length 11, corresponding to taking item 1 (gold). Don't forget our DP is a *longest* path problem!

```
In [8]: # Add edges corresponding to taking item 1
# (green edges of length 11)
for n in range(3, 9):
    G.add_edge((1, n), (2, n - 3), length=-11)
```

- We can do something similar for the light blue and orange edges as well:

```
In [9]: # Add edges corresponding to taking item 2
# (light blue edges of length 7)
for n in range(2, 9):
    G.add_edge((2, n), (3, n - 2), length=-7)

# Add edges corresponding to taking item 3
# (orange edges of length 12)
for n in range(4, 9):
    G.add_edge((3, n), (4, n - 4), length=-12)
```

- Finally, we can add the edges from the last stage nodes to the special “end” node:

```
In [10]: # Add edges from stage 4 to the end node
for n in range(0, 9):
    G.add_edge((4, n), "end", length=0)
```

- Now, we can solve the dynamic program using the Bellman-Ford algorithm, just as before:

```
In [11]: # Solve DP by solving its shortest path representation using Bellman-Ford
length, nodes, negative_cycle = bf.bellman_ford(G, source=(1, 8), target="end", weight="length")
print("Shortest path length: {0}".format(length))
print("Shortest path: {0}".format(nodes))
```

Shortest path length: -23  
Shortest path: [(1, 8), (2, 5), (3, 5), (4, 1), 'end']

## Interpreting the output

- What is the maximum value we can carry in the knapsack?

The maximum value we can carry in the knapsack is 23, the negative of the shortest path length.

- Which items should we take to obtain this maximum value?

According to the edges in the shortest path, we should take the gold and platinum, but not the silver.

## Practice makes perfect — on your own

- Here are two more examples of dynamic programs we modeled in a previous lesson. Solve them using networkx and interpret the output.

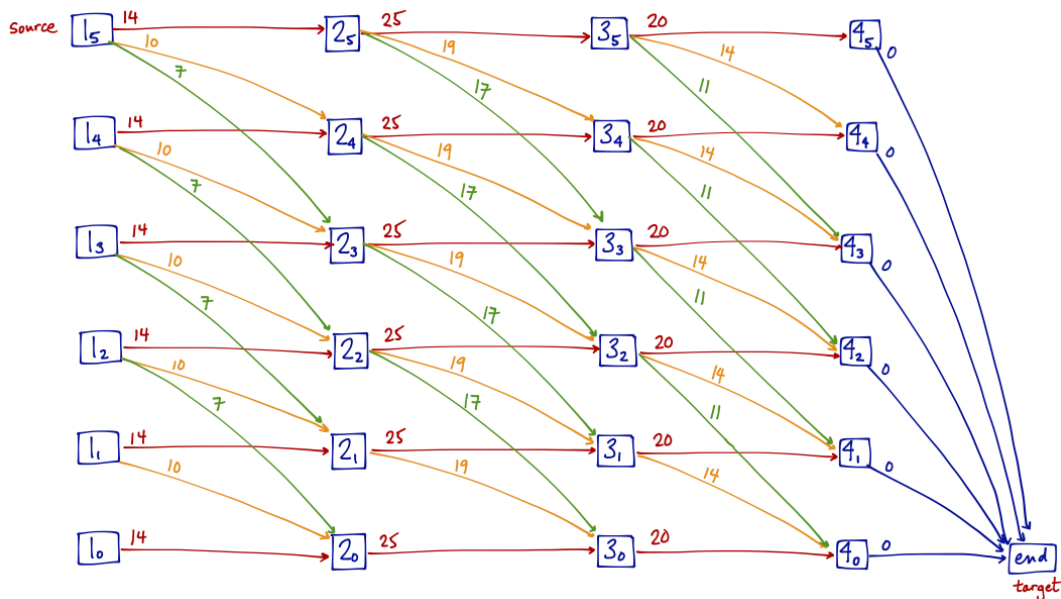
### Assigning patrol cars to precincts

The Simplexville Police Department wants to determine how to assign patrol cars to each precinct in Simplexville. Each precinct can be assigned 0, 1, or 2 patrol cars. The number of crimes in each precinct depends on the number of patrol cars assigned to each precinct:

Precinct	0 patrol cars	1 patrol cars	2 patrol cars
1	14	10	7
2	25	19	17
3	20	14	11

The department has 5 patrol cars. The department's goal is to minimize the total number of crimes across all 3 precincts.

- We formulated this problem as a dynamic program with the following shortest path representation:
  - Stage  $t$ : consider precinct  $t$
  - Node  $t_n$ :  $n$  patrol cars still be assigned at stage  $t$  (with precincts  $t, t + 1, \dots$  remaining)



DP for patrol car example

```
In [12]: # Solve this DP using networkx here
# Create empty graph
G = nx.DiGraph()
```

```

# Add the stage-state nodes
for t in range(1, 5):
    for n in range(0, 6):
        G.add_node((t, n))

# Add the end node
G.add_node("end")

# Add edges corresponding to adding 0 patrol cars - red edges
for n in range(0, 6):
    # precinct 1: length 14
    G.add_edge((1, n), (2, n), length=14)

    # precinct 2: length 25
    G.add_edge((2, n), (3, n), length=25)

    # precinct 3: length 20
    G.add_edge((3, n), (4, n), length=20)

# Add edges corresponding to adding 1 patrol car - orange edges
for n in range(1, 6):
    # precinct 1: length 10
    G.add_edge((1, n), (2, n - 1), length=10)

    # precinct 2: length 19
    G.add_edge((2, n), (3, n - 1), length=19)

    # precinct 3: length 14
    G.add_edge((3, n), (4, n - 1), length=14)

# Add edges corresponding to adding 2 patrol cars - green edges
for n in range(2, 6):
    # precinct 1: length 7
    G.add_edge((1, n), (2, n - 2), length=7)

    # precinct 2: length 17
    G.add_edge((2, n), (3, n - 2), length=17)

    # precinct 3: length 11
    G.add_edge((3, n), (4, n - 2), length=11)

# Add edges from last stage to the end node
for n in range(0, 6):
    G.add_edge((4, n), "end", length=0)

# Solve DP by solving its shortest path representation using Bellman-Ford
length, nodes, negative_cycle = bf.bellman_ford(G, source=(1, 5), target="end", weight="length")
print("Shortest path length: {}".format(length))
print("Shortest path: {}".format(nodes))

```

Shortest path length: 37

Shortest path: [(1, 5), (2, 3), (3, 2), (4, 0), 'end']

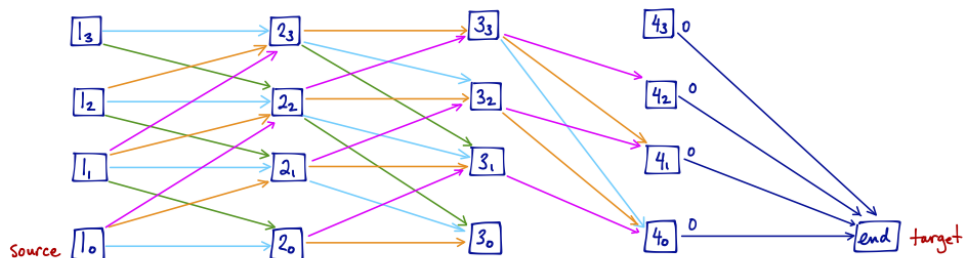
- The minimum number of crimes as a result of assigning the 5 patrol cars to the 3 precincts is 37, the shortest path length.
- To achieve this minimum number of crimes, assign 2 patrol cars to precinct 1, 1 patrol car to precinct 2, and 2 patrol cars to precinct 3.

## Inventory management

The Dijkstra Brewing Company is planning production of its new limited run beer, Primal Pilsner. The company must supply 1 batch next month, then 2 and 4 in successive months. Each month in which the company produces the beer requires a factory setup cost of \$5,000. Each batch of beer costs \$2,000 to produce. Batches can be held in inventory at a cost of \$1,000 per batch per month. Capacity limitations allow a maximum of 3 batches to be produced during each month. In addition, the size of the company's warehouse restricts the ending inventory for each month to at most 3 batches. The company has no initial inventory.

The company wants to find a production plan that will meet all demands on time and minimizes its total production and holding costs over the next 3 months.

- We formulated this problem as a dynamic program with the following shortest path representation:
  - Stage  $t$ : consider precinct  $t$
  - Node  $t_n$ :  $n$  patrol cars still be assigned at stage  $t$  (with precincts  $t, t + 1, \dots$  remaining)



Month	Prod. amt.	Edges	Edge lengths
1	0	$(1_n, 2_{n-1})$ for $n=1,2,3$	$1(n-1)$
1	1	$(1_n, 2_n)$ for $n=0,1,2,3$	$5 + 2(1) + 1(n)$
1	2	$(1_n, 2_{n+1})$ for $n=0,1,2$	$5 + 2(2) + 1(n+1)$
1	3	$(1_n, 2_{n+2})$ for $n=0,1$	$5 + 2(3) + 1(n+2)$
2	0	$(2_n, 3_{n-2})$ for $n=2,3$	$1(n-2)$
2	1	$(2_n, 3_{n-1})$ for $n=1,2,3$	$5 + 2(1) + 1(n-1)$
2	2	$(2_n, 3_n)$ for $n=0,1,2,3$	$5 + 2(2) + 1(n)$
2	3	$(2_n, 3_{n+1})$ for $n=0,1,2$	$5 + 2(3) + 1(n+1)$
3	0	not possible!	
3	1	$(3_n, 4_{n-3})$ for $n=3$	$5 + 2(1) + 1(n-3)$
3	2	$(3_n, 4_{n-2})$ for $n=2,3$	$5 + 2(2) + 1(n-2)$
3	3	$(3_n, 4_{n-1})$ for $n=1,2,3$	$5 + 2(3) + 1(n-1)$

### DP for inventory management example

```
In [13]: # Solve this DP using networkx here
# Create empty graph
G = nx.DiGraph()

# Add the stage-state nodes
for t in range(1, 5):
    for n in range(0, 3):
        G.add_node((t, n))

# Add the end node
G.add_node("end")
```

```

# Add edges corresponding to production in month 1
# 0 batches: green edges
for n in range(1, 4):
    G.add_edge((1, n), (2, n - 1), length=1*(n - 1))

# 1 batch: blue edges
for n in range(0, 4):
    G.add_edge((1, n), (2, n), length=5 + 2*(1) + 1*(n))

# 2 batches: orange edges
for n in range(0, 3):
    G.add_edge((1, n), (2, n + 1), length=5 + 2*(2) + 1*(n + 1))

# 3 batches: purple edges
for n in range(0, 2):
    G.add_edge((1, n), (2, n + 2), length=5 + 2*(3) + 1*(n + 2))

# Add edges corresponding to production in month 2
# 0 batches: green edges
for n in range(2, 4):
    G.add_edge((2, n), (3, n - 2), length=1*(n - 2))

# 1 batch: blue edges
for n in range(1, 4):
    G.add_edge((2, n), (3, n - 1), length=5 + 2*(1) + 1*(n - 1))

# 2 batches: orange edges
for n in range(0, 4):
    G.add_edge((2, n), (3, n), length=5 + 2*(2) + 1*(n))

# 3 batches: purple edges
for n in range(0, 3):
    G.add_edge((2, n), (3, n + 1), length=5 + 2*(3) + 1*(n + 1))

# Add edges corresponding to production in month 3
# 0 batches: not possible!

# 1 batch: blue edges
for n in range(3, 4):
    G.add_edge((3, n), (4, n - 3), length=5 + 2*(1) + 1*(n - 3))

# 2 batches: orange edges
for n in range(2, 4):
    G.add_edge((3, n), (4, n - 2), length=5 + 2*(2) + 1*(n - 2))

# 3 batches: purple edges
for n in range(1, 4):
    G.add_edge((3, n), (4, n - 1), length=5 + 2*(3) + 1*(n - 1))

# Add edges from last stage to the end node
for n in range(0, 4):
    G.add_edge((4, n), "end", length=0)

# Solve DP by solving its shortest path representation using Bellman-Ford
length, nodes, negative_cycle = bf.bellman_ford(G, source=(1, 0), target="end", weight="length")
print("Shortest path length: {}".format(length))
print("Shortest path: {}".format(nodes))

```

Shortest path length: 30

Shortest path: [(1, 0), (2, 0), (3, 1), (4, 0), 'end']

- The minimum total production and holding cost over the next 3 months is 30.
- To achieve this minimum cost, the company should produce 1 batch in month 1, 3 batches in month 2, and 3 batches in month 3.