Lesson 11.

Drafting a fantasy basketball team

The problem

You're preparing for your upcoming fantasy basketball draft. You wonder: what is the best possible team you can draft?

You have the following data:

- Projected **auction prices** for each player in the NBA.
- The **z-score** for each player: the sum of the number of standard deviations above the mean in the following 9 categories:

    1. points per 36 minutes
    2. 3 point field goals made per 36 minutes
    3. number of rebounds per 36 minutes
    4. number of assists per 36 minutes
    5. number of steals per 36 minutes
    6. number of blocks per 36 minutes
    7. *negative* of the number of turnovers per 36 minutes
    8. field goal percentage
    9. free throw percentage

Your roster must have exactly 12 players, and you have a budget of \$50. You want to maximize the total z-score of your team.

Formulate this problem as a dynamic program by giving its shortest/longest path representation.

Solving the DP

- *Warning.* The code we're about to write isn't the most "Pythonic." However, it matches well with the mathematical notation we've been using in class.
- In the same folder as this notebook, there is a file called `fantasy_basketball_nba2017.csv` with the data described above.
    - The z-scores were computed using projected stats from Basketball Reference.
    - Projected auction prices were taken from Yahoo! Fantasy Sports, normalized to a budget of \$50.
- Let's take a look using pandas. First, let's import pandas:

```
In [2]: # Import pandas
        import pandas as pd
```

- Now we can read the csv file into a pandas DataFrame and inspect the first few rows:

```
In [3]: # Read csv file with data
        df = pd.read_csv('fantasy_basketball_nba2017.csv')

        # Print the first 5 rows of df
        df.head()
```

```
Out[3]:           PLAYER TEAM POSITIONS     ZSCORE  PRICE
        0  Stephen Curry   GS     PG,SG  12.681705     18
        1  Kawhi Leonard   SA     SG,SF   8.994709     16
        2     Chris Paul  LAC        PG   8.485619     15
        3  Anthony Davis   NO      PF,C   8.357714     15
        4   Kevin Durant   GS     SF,PF   7.848493     18
```

- As we can see, we even have some other data: each player's team and the positions each player plays.
- Let's use this data to create the shortest/longest path representation of our DP in networkx.
- As usual, let's import networkx and bellmanford first:

```
In [4]: # Import networkx and bellman ford
        import networkx as nx
        import bellmanford as bf
```

- There are two important constants in our problem: the budget, and the roster size.
- Let's create variables to hold these constants.
- This way, we can easily adapt our code to accomodate similar DPs with different budgets and roster sizes.

```
In [5]: # Create variables to hold constants: budget, roster size
        BUDGET = 50
        ROSTER_SIZE = 12
```

- Next, let's create some lists that correspond to the relevant columns of the dataset.
- Recall that we can grab a column from a DataFrame like this:

```
df['COLUMN_NAME']
```

- The `list()` function turns any list-like object (such as a column of a pandas DataFrame) into a Python list.
- We can apply the `.str.split(",")` method to convert a comma-delimited string into a list. This will be helpful in parsing the positions that a player can play, since many players can play multiple positions.

```
In [6]: # Create a list of players
        players = list(df["PLAYER"])

        # Create a list of zscores
        zscores = list(df["ZSCORE"])

        # Create a list of prices
        prices = list(df["PRICE"])

        # Create a list of positions
```

```
        positions = list(df["POSITIONS"].str.split(","))
```

- Now we can look at player t and his associated data like this:

```
In [7]: # Print out information about player 3 - Anthony Davis
        print(players[3])
        print(zscores[3])
        print(prices[3])
        print(positions[3])
```

```
Anthony Davis
8.35771414362
15
['PF', 'C']
```

- Let's also create a variable that holds the number of players:

```
In [8]: # Create a variable for the number of players
        n_players = len(players)
```

- Now we can use these lists and variables to construct the graph for the dynamic program.
- As usual, we start with an empty graph:

```
In [9]: # Create empty digraph
        G = nx.DiGraph()
```

- Next, let's add the nodes:

```
In [10]: # Add stage-state nodes (t, n1, n2)
         for t in range(0, n_players + 1):
             for n1 in range(0, BUDGET + 1):
                 for n2 in range(0, ROSTER_SIZE + 1):
                     G.add_node((t, n1, n2))

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

- How many nodes do we have in our graph?

```
In [11]: # Print number of nodes in digraph
         print(G.number_of_nodes())
```

```
293710
```

- Now it's time to add the edges.
- Let's start with the edges corresponding to the decision of whether to take a player or not:

```python
In [12]: # Add edges corresponding to the decision of whether to take a player or not
         for t in range(0, n_players):
             for n1 in range(0, BUDGET + 1):
                 for n2 in range(0, ROSTER_SIZE + 1):

                     # Don't take the player
                     G.add_edge((t, n1, n2), (t + 1, n1, n2), length=0)

                     # Take the player if there's enough left in the budget
                     # and there are enough roster spots
                     if n1 - prices[t] >= 0:
                         if n2 - 1 >= 0:
                             G.add_edge((t, n1, n2), (t + 1, n1 - prices[t], n2 - 1), length=-zscores[t])
```

- Now we can add the edges from the last stage to the end node. Remember to only add edges from the last stage if the number of remaining roster spots n_2 is equal to 0!

```python
In [13]: # Add edges from last stage to end,
         # only when number of remaining roster spots is 0
         for n1 in range(0, BUDGET + 1):
             G.add_edge((n_players, n1, 0), "end", length=0)
```

- How many edges do we have in our graph?

```python
In [14]: # Print number of edges
         print(G.number_of_edges())
```

```
550545
```

- Finally, let's solve the shortest path problem we've constructed using the Bellman-Ford algorithm:

```python
In [15]: # Solve the shortest path problem using the Bellman-Ford algorithm
         length, nodes, negative_cycle = bf.bellman_ford(G, source=(0, BUDGET, ROSTER_SIZE), target="end",
                                                         weight="length")

         print("Negative cycle? {0}".format(negative_cycle))
         print("Shortest path length: {0}".format(length))
         print("Shortest path: {0}".format(nodes))
```

```
Negative cycle? False
Shortest path length: -57.45369330886113
Shortest path: [(0, 50, 12), (1, 32, 11), (2, 32, 11), (3, 32, 11), (4, 32, 11), (5, 32, 11), (6, 32, 11), (7, 32, 11), (8
```

- It's easy to see what the maximum possible total z-score is… however, which players should we select to get this maximum total z-score?
- Instead of reading through the path of 400+ nodes to figure out which players to select, let's write some code to do this for us.
- We know that we select a player whenever the number of remaining roster spots n_2 goes down by 1 from stage to stage. So…

```
In [16]: # Print selected players in a more user-friendly format
         # Get number of nodes in shortest path
         n_nodes = len(nodes)

         # Go through each node in the shortest path
         for i in range(n_nodes - 2):

             # Node in current stage
             (t, n1, n2) = nodes[i]

             # Node in next stage
             (next_t, next_n1, next_n2) = nodes[i + 1]

             # If n2 isn't the same from one stage to the next, print the player's info
             if n2 != next_n2:
                 print("Node: {0}  Player: {1}  Positions: {2}, Price: {3}  Z-Score: {4}"
                       .format(nodes[t], players[t], positions[t], prices[t], zscores[t]))
```

```
Node: (0, 50, 12)  Player: Stephen Curry  Positions: ['PG', 'SG'], Price: 18  Z-Score: 12.681704920021767
Node: (8, 32, 11)  Player: Nikola Jokic  Positions: ['PF', 'C'], Price: 9  Z-Score: 6.245534045281088
Node: (9, 23, 10)  Player: Klay Thompson  Positions: ['SG', 'SF'], Price: 8  Z-Score: 5.781494181785728
Node: (11, 15, 9)  Player: Cole Aldrich  Positions: ['C'], Price: 1  Z-Score: 5.689641521236912
Node: (17, 14, 8)  Player: Boban Marjanovic  Positions: ['C'], Price: 1  Z-Score: 4.542112513644865
Node: (23, 13, 7)  Player: Brandan Wright  Positions: ['PF', 'C'], Price: 1  Z-Score: 4.092581777062199
Node: (24, 12, 6)  Player: Jrue Holiday  Positions: ['PG'], Price: 3  Z-Score: 4.0156102748746365
Node: (27, 9, 5)  Player: Dirk Nowitzki  Positions: ['PF', 'C'], Price: 4  Z-Score: 3.798143244965448
Node: (40, 5, 4)  Player: Kelly Olynyk  Positions: ['C'], Price: 2  Z-Score: 2.9855612991757123
Node: (47, 3, 3)  Player: Jeremy Lamb  Positions: ['SG', 'SF'], Price: 1  Z-Score: 2.657902391350232
Node: (51, 2, 2)  Player: Cameron Payne  Positions: ['PG'], Price: 1  Z-Score: 2.487896082671208
Node: (52, 1, 1)  Player: Mike Muscala  Positions: ['PF', 'C'], Price: 1  Z-Score: 2.47551105679133
```

Incorporating other roster constraints

- Fantasy basketball leagues usually have some roster constraints — in particular, on player positions.
- For example, suppose our roster must have exactly 2 players that can play center (C).
- How can we modify our dynamic program to accomodate this? Write a new dynamic program on paper.
- How do we need to modify the code above to solve the new dynamic program?
- A hint:
    - To check if player t can play center, we can write:

```
if "C" in positions[t]:
    ...
```

    - This code does what it looks like: it checks if `"C"` is in the list of positions `positions[t]` that player t can play.

```
In [17]: # Create empty digraph
         H = nx.DiGraph()

         # Add stage-state nodes (t, n1, n2, n3)
         # t = player
```

```python
        # n1 = remaining budget
        # n2 = remaining roster spots
        # n3 = remaining C roster spots
        for t in range(0, n_players):
            for n1 in range(0, BUDGET + 1):
                for n2 in range(0, ROSTER_SIZE + 1):
                    for n3 in range(0, 3):
                        G.add_node((t, n1, n2, n3))

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

        # Add edges corresponding to the decision of whether to take a player or not
        for t in range(0, n_players):
            for n1 in range(0, BUDGET + 1):
                for n2 in range(0, ROSTER_SIZE + 1):
                    for n3 in range(0, 3):

                        # Don't take the player
                        H.add_edge((t, n1, n2, n3), (t + 1, n1, n2, n3), length=0)

                        # Take the player if there's enough left in the budget
                        # and there are enough roster spots
                        if n1 - prices[t] >= 0:
                            if n2 - 1 >= 0:

                                # If the player is a center, we can only add this edge if
                                # there are enough remaining C roster spots
                                if "C" in positions[t]:
                                    if n3 - 1 >= 0:
                                        H.add_edge((t, n1, n2, n3),
                                                   (t + 1, n1 - prices[t], n2 - 1, n3 - 1),
                                                   length=-zscores[t])

                                # Otherwise, the number of remaining C roster spots stays the same
                                else:
                                    H.add_edge((t, n1, n2, n3), (t + 1, n1 - prices[t], n2 - 1, n3),
                                               length=-zscores[t])

        # Add edges from last stage to end,
        # only when number of remaining roster spots is 0 and
        # the number of remaining C roster spots is 0
        for n1 in range(0, BUDGET + 1):
            H.add_edge((n_players, n1, 0, 0), "end", length=0)


        # Solve the shortest path problem using the Bellman-Ford algorithm
        length, nodes, negative_cycle = bf.bellman_ford(H, source=(0, BUDGET, ROSTER_SIZE, 2),
                                                        target="end", weight="length")

        print("Negative cycle? {0}".format(negative_cycle))
        print("Shortest path length: {0}".format(length))
        print("Shortest path: {0}".format(nodes))

        # Print selected players in a more user-friendly format
        # Get number of nodes in shortest path
        n_nodes = len(nodes)

        # Go through each node in the shortest path
        for i in range(n_nodes - 2):
```

```
            # Node in current stage
            (t, n1, n2, n3) = nodes[i]

            # Node in next stage
            (next_t, next_n1, next_n2, next_n3) = nodes[i + 1]

            # If n2 isn't the same from one stage to the next, print the player's info
            if n2 != next_n2:
                print("Node: {0}  Player: {1}  Positions: {2}, Price: {3}  Z-Score: {4}".format(nodes[t], players[t],
```

```
Negative cycle? False
Shortest path length: -54.39369050175551
Shortest path: [(0, 50, 12, 2), (1, 32, 11, 2), (2, 32, 11, 2), (3, 17, 10, 2), (4, 17, 10, 2), (5, 17, 10, 2), (6, 17, 10
Node: (0, 50, 12, 2)  Player: Stephen Curry  Positions: ['PG', 'SG'], Price: 18  Z-Score: 12.681704920021767
Node: (2, 32, 11, 2)  Player: Chris Paul  Positions: ['PG'], Price: 15  Z-Score: 8.485618602625312
Node: (11, 17, 10, 2)  Player: Cole Aldrich  Positions: ['C'], Price: 1  Z-Score: 5.689641521236912
Node: (17, 16, 9, 1)  Player: Boban Marjanovic  Positions: ['C'], Price: 1  Z-Score: 4.542112513644865
Node: (24, 15, 8, 0)  Player: Jrue Holiday  Positions: ['PG'], Price: 3  Z-Score: 4.0156102748746365
Node: (31, 12, 7, 0)  Player: Robert Covington  Positions: ['SF', 'PF'], Price: 3  Z-Score: 3.418952349391036
Node: (32, 9, 6, 0)  Player: Nikola Mirotic  Positions: ['SF', 'PF'], Price: 4  Z-Score: 3.4146385523834835
Node: (47, 5, 5, 0)  Player: Jeremy Lamb  Positions: ['SG', 'SF'], Price: 1  Z-Score: 2.657902391350232
Node: (51, 4, 4, 0)  Player: Cameron Payne  Positions: ['PG'], Price: 1  Z-Score: 2.487896082671208
Node: (59, 3, 3, 0)  Player: Toney Douglas  Positions: ['PG', 'SG'], Price: 1  Z-Score: 2.3923422473096463
Node: (63, 2, 2, 0)  Player: Alan Williams  Positions: ['PF'], Price: 1  Z-Score: 2.34597794227906
Node: (65, 1, 1, 0)  Player: C.J. Miles  Positions: ['SG', 'SF'], Price: 1  Z-Score: 2.2612931039673394
```

Food for thought

- Can the dynamic programs we solved above help with an actual fantasy basketball draft? Why or why not?

- These DPs only give you the best possible roster. They don't model the draft process; in particular, not all players may be available when it's our turn to select, and the DPs don't use actual auction prices.

- These DPs can help plan *during* a draft: as a draft progresses, one can update the DP to remove the players that have been already selected, and use the DP to plan which of the remaining players to focus on.