Six Degrees of Kevin Bacon

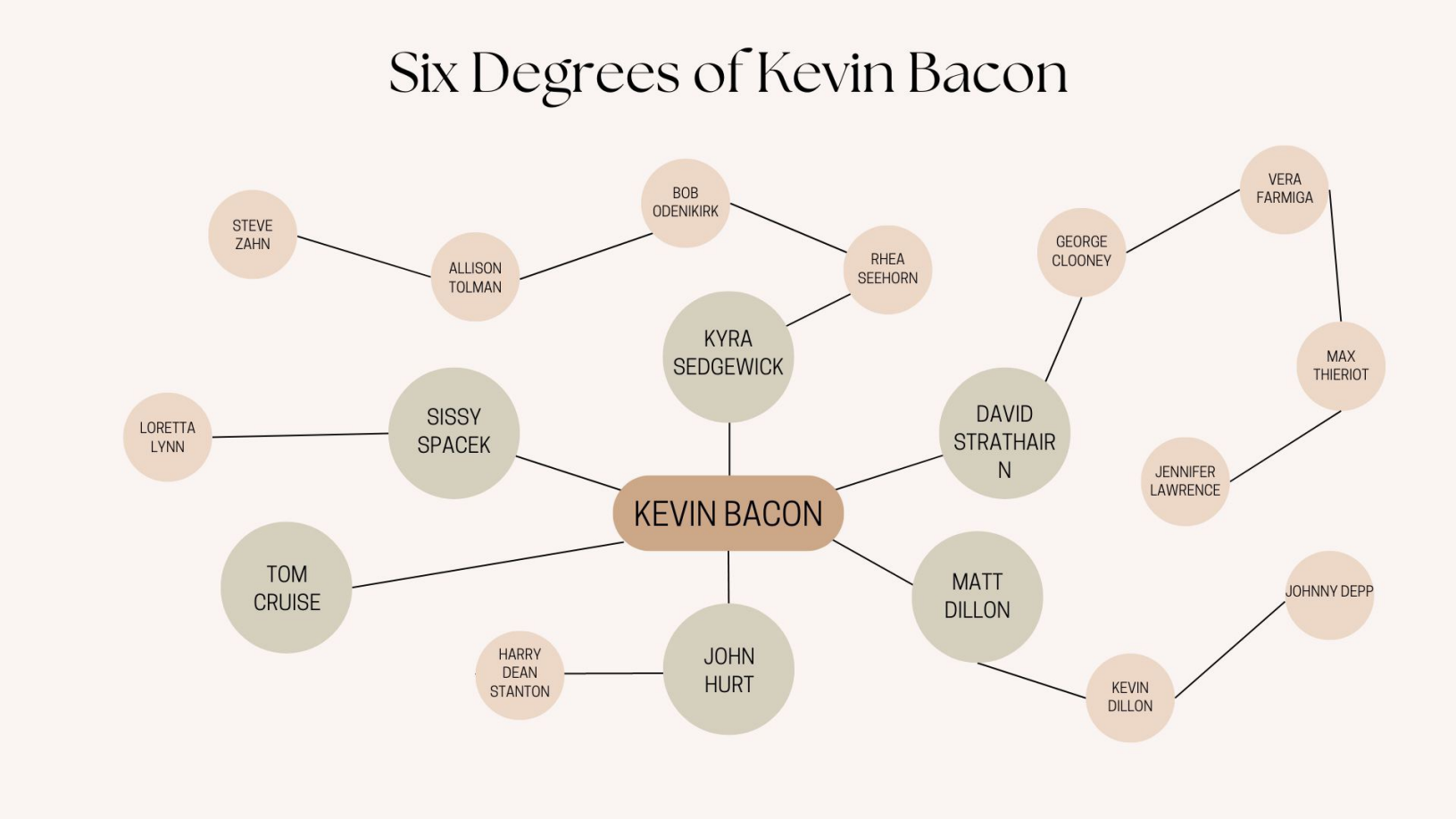
Ezer Patlan

CS460

Summer 2024

Six Degrees of Kevin Bacon

- It involved in the Hollywood film industry can be linked through their film roles to Bacon within six steps.



Breadth First Search Algorithm

- Starting from the root, all the nodes at a particular level are visited first and then the nodes of the next level are traversed till all the nodes are visited.

- To do this a queue is used. All the adjacent unvisited nodes of the current level are pushed into the queue and the nodes of the current level are marked visited and removed from the queue.

Time and Space Complexity

- Time Complexity of BFS Algorithm: O(V + E)
  - It explores all the vertices and edges in the graph
  - In the worst case, it visits every vertex and edge once

- Space Complexity of BFS Algorithm: O(V)
  - BFS uses a queue to keep track of the vertices that need to be visited.
  - In the worst case, the queue can contain all the vertices in the graph

BFS Algorithm

Shortest Path

```python
def shortest_path(source, target):
    """
    Returns the shortest list of (movie_id, person_id) pairs
    that connect the source to the target, using BFS.

    source and target are unique IMDB actor ID's

    If no possible path, returns None.
    """

    # Initialise a QueueFrontier for BFS, with the starting actor ID:
    start = Node(source, None, None)
    frontier = QueueFrontier()
    frontier.add(start)

    # Initialise an empty explored set to hold explored states (actors):
    explored = set()

    # Loop until a solution is found, or Frontier is empty(no solution):
    while True:

      if len(explored) % 100 == 0:
        print('Actors explored to find solution: ', len(explored))
        print('Nodes left to expand in Frontier: ', len(frontier.frontier))

      # Check for empty Frontier and return with no path if empty
      if frontier.empty():
        print('Frontier is Empty - No Connection Between Actors!')
        print(len(explored), 'actors explored to with no solution found!')
        return None

      # Otherwise expand the next node in the Queue,
      # add it to the explored states and get set of movies and
      # actors for the actor in the current node:
      curr_node = frontier.remove()
      explored.add(curr_node.state)
```

```python
    for action, state in neighbors_for_person(curr_node.state):

      # If state (actor) is the target actor
      # then solution has been found, return path:
      if state == target:
        print('Solution Found!')
        print(len(explored), 'actors explored to find solution!')
        # Create path from source to target
        path = []
        path.append((action, state))

        # Add action and state to path until back to start node
        while curr_node.parent != None:
          path.append((curr_node.action, curr_node.state))
          curr_node = curr_node.parent

        path.reverse()

        return path

      # Otherwise add the new states to explore to the frontier:
      if not frontier.contains_state(state) and state not in explored:
        new_node = Node(state, curr_node, action)
        frontier.add(new_node)
```

BFS Algorithm

Shortest Path

```
def shortest_path(source, target):
    """
    Returns the shortest list of (movie_id, person_id) pairs
    that connect the source to the target, using BFS.

    source and target are unique IMDB actor ID's

    If no possible path, returns None.
    """

    # Initialise a QueueFrontier for BFS, with the starting actor ID:
    start = Node(source, None, None)
    frontier = QueueFrontier()
    frontier.add(start)

    # Initialise an empty explored set to hold explored states (actors):
    explored = set()

    # Loop until a solution is found, or Frontier is empty(no solution):
    while True:

      if len(explored) % 100 == 0:
        print('Actors explored to find solution: ', len(explored))
        print('Nodes left to expand in Frontier: ', len(frontier.frontier))

      # Check for empty Frontier and return with no path if empty
      if frontier.empty():
        print('Frontier is Empty - No Connection Between Actors!')
        print(len(explored), 'actors explored to with no solution found!')
        return None

      # Otherwise expand the next node in the Queue,
      # add it to the explored states and get set of movies and
      # actors for the actor in the current node:
      curr_node = frontier.remove()
      explored.add(curr_node.state)
```

- Shortest Path:
  - BFS can be used to find the shortest path between two nodes in an unweighted graph
  - Keep track of the parent of each node during the traversal

BFS Algorithm

Shortest Path

```
for action, state in neighbors_for_person(curr_node.state):

  # If state (actor) is the target actor
  # then solution has been found, return path:
  if state == target:
    print('Solution Found!')
    print(len(explored), 'actors explored to find solution!')
    # Create path from source to target
    path = []
    path.append((action, state))

    # Add action and state to path until back to start node
    while curr_node.parent != None:
      path.append((curr_node.action, curr_node.state))
      curr_node = curr_node.parent

    path.reverse()

    return path

  # Otherwise add the new states to explore to the frontier:
  if not frontier.contains_state(state) and state not in explored:
    new_node = Node(state, curr_node, action)
    frontier.add(new_node)
```

- Connected Components:
  - BFS can be used to identify connected components in a graph
  - Each connected component is a set of nodes that can be reached from each other

Functions

Person ID for Name

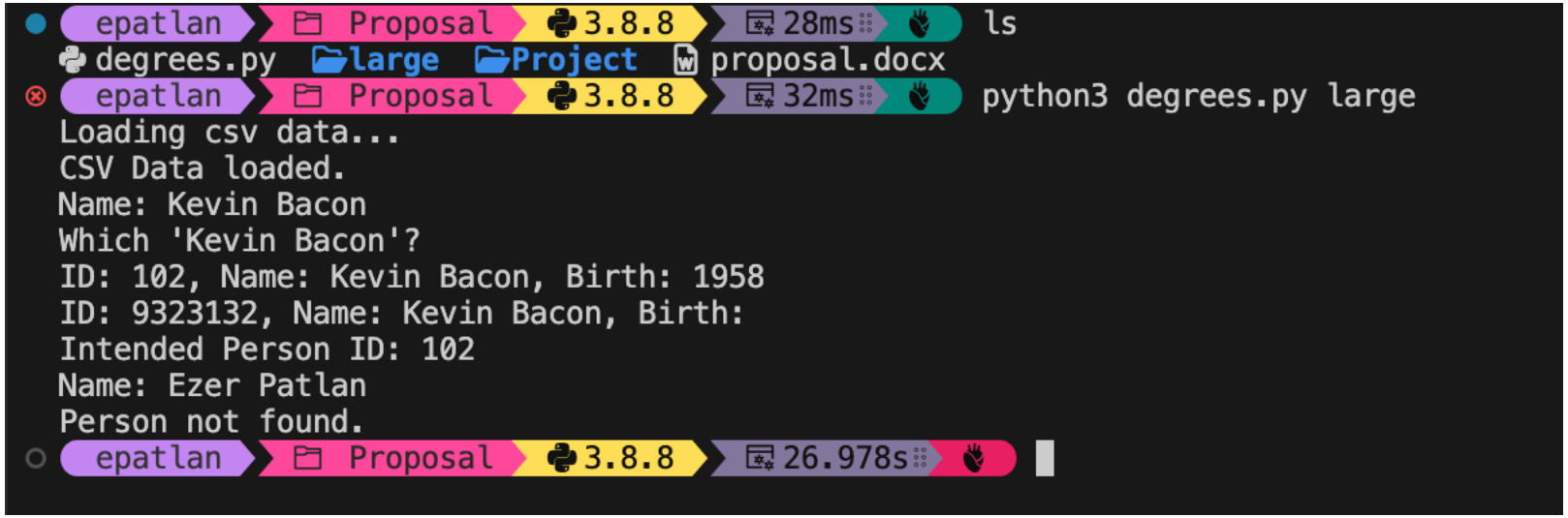
```python
def person_id_for_name(name):
    """
    Returns the IMDB id for a person's name,
    resolving ambiguities as needed.
    """

    person_ids = list(names.get(name.lower(), set()))
    #print(list(names.get(name.lower())))
    if len(person_ids) == 0:
        return None
    elif len(person_ids) > 1:
        print(f"Which '{name}'?")
        for person_id in person_ids:
            person = people[person_id]
            name = person["name"]
            birth = person["birth"]
            print(f"ID: {person_id}, Name: {name}, Birth: {birth}")
        try:
            person_id = input("Intended Person ID: ")
            if person_id in person_ids:
                return person_id
        except ValueError:
            pass
        return None
    else:
        return person_ids[0]
```

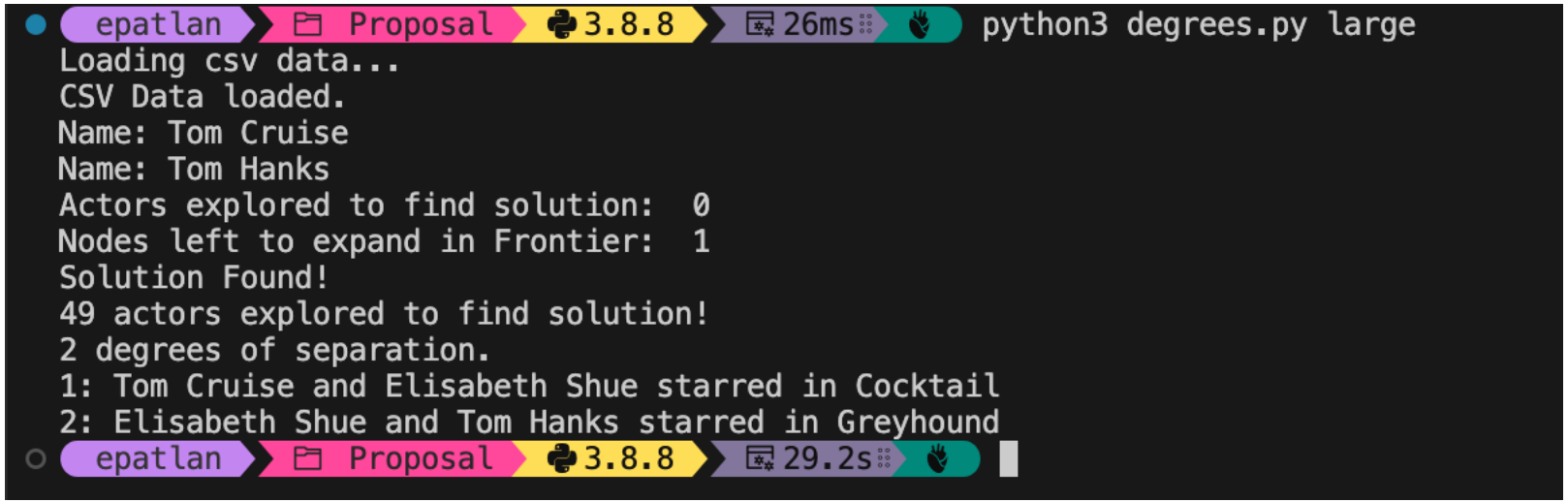
Neighbors for person

```python
def neighbors_for_person(person_id):
    """
    Returns (movie_id, person_id) pairs for people
    who starred with a given person.
    """
    movie_ids = people[person_id]["movies"]
    neighbors = set()
    for movie_id in movie_ids:
        for person_id in movies[movie_id]["stars"]:
            neighbors.add((movie_id, person_id))
    return neighbors
```

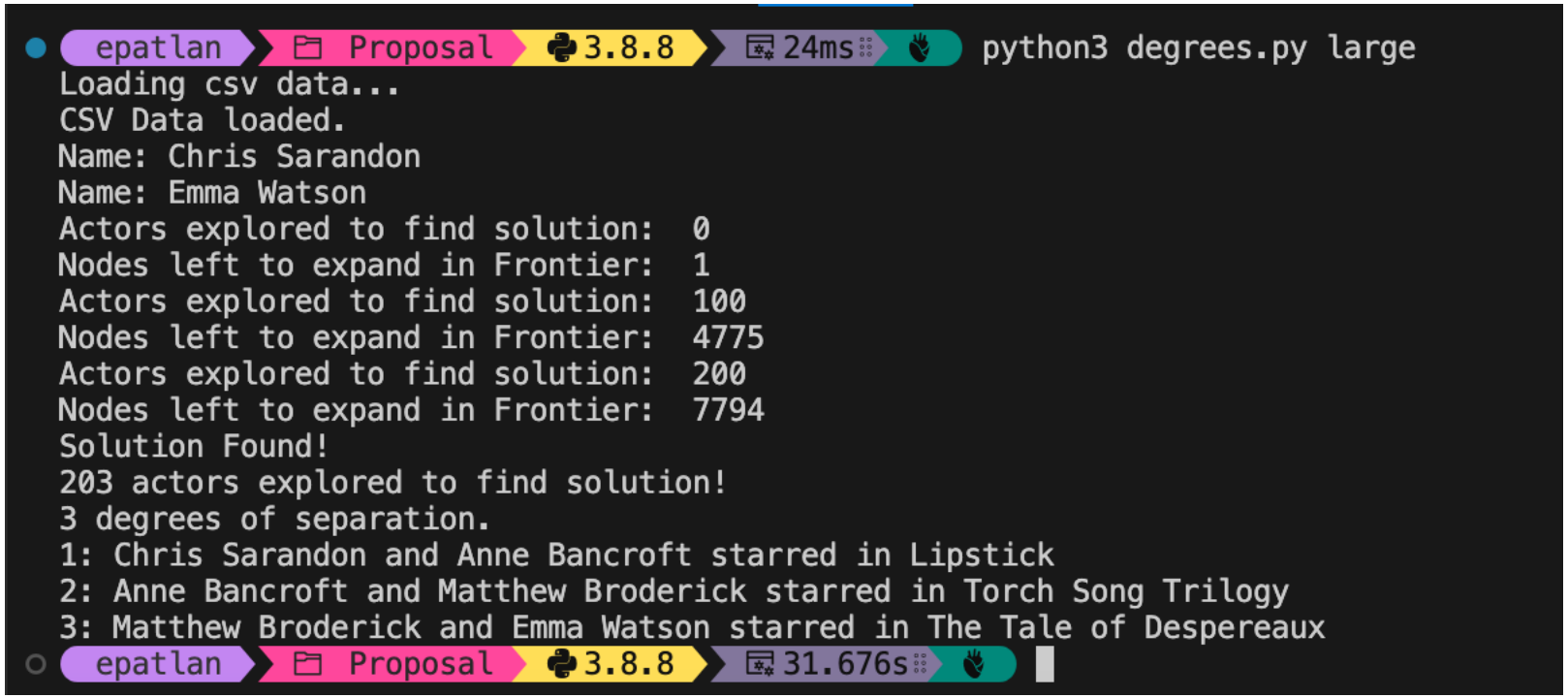
0 Degree of separation



2 Degrees of seperation



3 Degrees of seperation



Thank You