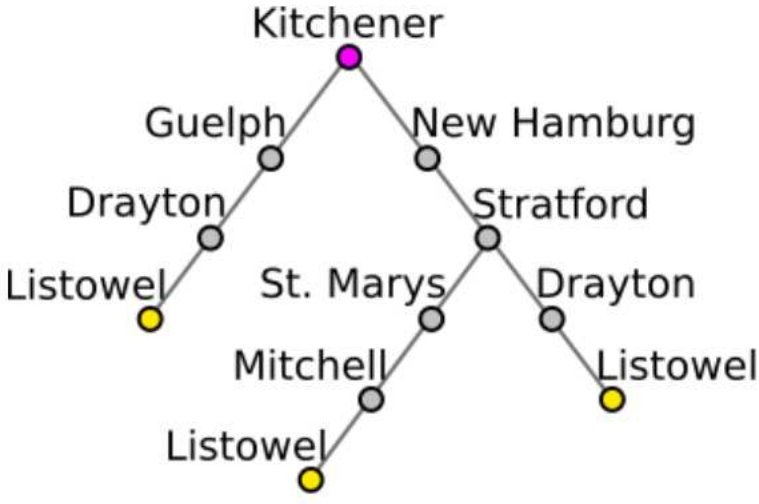
1. Implement Depth-First Search (DFS)

   - It is very similar to BFS, but uses a Last-In-First-Out (LIFO) stack instead of a First-In-First-Out (FIFO) queue.
   - The algorithm can be tested on the search tree generated by `map2searchtree.py`.
   - Hint: Check the list section in the Python datastructures docs.

2. Run `GreedySearch()` with Kitchener as root node (`initi_state`) and Listowel as `goal_name`.

   - Build a new search tree, i.e. don't use the one generated by `map2searchtree.py`.
   - The nodes in this search tree need to include `heuristic`, i.e. the heuristic function. E.g. add `node['heuristic']` (similar to `node['weight']`) to `get_node()`.
   - The heuristic function is given after each node name and corresponds to the red lines on slide 52 ("Greedy Search: Map example") of lecture 2:

   - Only include the follwing locations:
     - Kitchener : 130
     - Guelph : 160
     - Drayton : 100
     - New Hamburg : 110
     - Stratford : 100
     - St. Marys : 130
     - Mitchell : 100
     - Listowel : 0



Build the tree as shown above.

3. Record and compare the paths that three different search agorithms take to reach the goal. The path is the sequence of nodes (cities) visited to get to the goal. Compare `BFS()`, `UCS()` and `GreedySearch()`. Recording the path should only require the addition of a few lines of code to the above functions.

   - Use the following weights (distances) for `UCS()`:
     - Kitchener -> New Hamburg : 90
     - Kitchener -> Guelph : 30
     - New Hamburg -> Stratford : 25
     - Guelph -> Drayton : 100
     - Stratford -> Drayton : 200
     - Stratford -> St. Marys : 30
     - Drayton -> Listowel : 100
     - St. Mary -> Mitchell : 80
     - Mitchell -> Listowel : 100