Tasks

- 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.
  - Hint: Check the list section in the Python datastructures docs (https://docs.python.org/3/tutorial/datastructures.html).
- 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 needs to include `h` , i.e. the heuristic function. E.g. add `self.h` (similar to `self.weight`) to `Node` .
  - Only include the locations listed below. The heuristic function is given after each node name and corresponds to the red lines on slide 40 of lecture 2:
    - Kitchener : 130
    - Guelph : 160
    - Drayton : 100
    - New Hamburg : 110
    - Stratford : 100
    - St. Marys : 130
    - Mitchell : 100
- Record and return the path (i.e. sequence of nodes) the search algorithm took to reach the goal.
  - Hint: Add `self.path = []` to `Node` and then record the parent and parent's parent (and so on) when a node is added to the `frontier` . `list.extend()` might prove useful. See the Python datastructures docs (https://docs.python.org/3/tutorial/datastructures.html) for more info about Python lists.

Code provided

Search algorithms

Breadth-First Search (BFS)

Question 1

In [2]:

```
# Depth-first search
def DFS(init_state, goal_name):
    """Depth-First Search (DFS)

    Arguments
    ---------
    init_state : the root node of a search tree
    goal_name  : A string, the name of a node, e.g. tree.childrend[0].name
    """

    frontier = [init_state]
    explored = []

    while len(frontier):
        state = frontier.pop()  # dequeue
        explored.append(state.name)

        if state.name == goal_name:
            return True

        for child in state.children:
            if child.name not in explored:
                # enqueue: insert node at the beginning
                frontier.append(child)

    return False
```

Question 2)Greedy Search

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 needs to include h, i.e. the heuristic function. E.g. add self.h (similar to self.weight) to Node. Only include the locations listed below. The heuristic function is given after each node name and corresponds to the red lines on slide 40 of lecture 2: Kitchener : 130 Guelph : 160 Drayton : 100 New Hamburg : 110 Stratford : 100 St. Marys : 130 Mitchell : 100

In [4]:
```
# Node with weight
class Node:
    def __init__(self, name, h=None):
        self.children = []
        self.name = name
        self.h = h
        self.path =[]


def add_child(node, name, h):
    node.children.append(Node(name, h))
```

In [5]:
```
# Building the entire search tree based on the map
tree = Node('Kitchener',130)
print(tree)
```

```
<__main__.Node object at 0x0000026D8F8D9E08>
```

In [6]:
```
# Children of Kitchner: Guelph,New Hamburg
add_child(tree, 'Guleph',160)
add_child(tree, 'New Hamburg',110)
```

In [7]:
```
# Children of Guleph: Drayton , Kitchener
add_child(tree.children[0], 'Drayton', 100)
add_child(tree.children[0], 'Kitchener', 130)
```

In [8]:
```

# Children of 'New Hamburg: Stratford , Kitchener
add_child(tree.children[1], 'Stratford', 100)
add_child(tree.children[1], 'Kitchener', 130)
```

In [9]:
```
# Children of  Drayton:,Listowel,Stratford,Guleph


add_child(tree.children[0].children[0], 'Listowel', 0)
add_child(tree.children[0].children[0], 'Stratford', 100)
add_child(tree.children[0].children[0], 'Guleph', 160)
```

In [10]:
```
# Children of kITCHNER:,Guleph & New Hamburg
add_child(tree.children[0].children[1], 'Guleph', 160)
add_child(tree.children[0].children[1], ' New Hamburg', 110)
```

In [11]:
```
# Children of  Stratford:Drayton,New Hamburg , St.Marys
add_child(tree.children[1].children[0], 'Drayton', 100)
add_child(tree.children[1].children[0], 'New Hamburg ', 110)
add_child(tree.children[1].children[0], ' St.Marys', 130)
```

In [12]:
```
# Children of  Kitchner:Guleph & New Hamburg
add_child(tree.children[1].children[1], 'Guleph', 160)
add_child(tree.children[1].children[1], 'New Hamburg ', 110)
```

In [13]:
```

# Children of  Listowel:Mitchell,Drayton
add_child(tree.children[0].children[0].children[0], 'Mitchell', 100)
add_child(tree.children[0].children[0].children[0], 'Drayton', 100)
```

In [14]:
```
# Children of  Stratford:Drayton,New Hamburg , St.Marys
add_child(tree.children[0].children[0].children[1], 'Drayton', 100)
add_child(tree.children[0].children[0].children[1], 'New Hamburg ', 110)
add_child(tree.children[0].children[0].children[1], ' St.Marys', 130)
```

In [15]:
```
#So just add all the missing nodes u shud have 0,1,00,01,10,11,000,001,010,011,100,101,111
# Children of Guleph: Drayton , Kitchener
add_child(tree.children[0].children[0].children[2], 'Drayton', 100)
add_child(tree.children[0].children[0].children[2], 'Kitchener', 130)
```

In [16]:
```
# Children of Guleph: Drayton , Kitchener
add_child(tree.children[0].children[1].children[0], 'Drayton', 100)
add_child(tree.children[0].children[1].children[0], 'Kitchener', 130)
```

In [17]:
```
# Children of 'New Hamburg: Stratford , Kitchener
add_child(tree.children[0].children[1].children[1], 'Stratford', 100)
add_child(tree.children[0].children[1].children[1], 'Kitchener', 130)
```

In [18]:
```
# Children of  Drayton:,Listowel,Stratford,Guleph


add_child(tree.children[1].children[0].children[0], 'Listowel', 0)
add_child(tree.children[1].children[0].children[0], 'Stratford', 100)
add_child(tree.children[1].children[0].children[0], 'Guleph', 160)
```

In [19]:
```
# Children of 'New Hamburg: Stratford , Kitchener
add_child(tree.children[1].children[0].children[1], 'Stratford', 100)
add_child(tree.children[1].children[0].children[1], 'Kitchener', 130)
```

In [20]:
```
# Children of 'St.Mary: Stratford & Mitchell
add_child(tree.children[1].children[0].children[2], 'Stratford', 100)
add_child(tree.children[1].children[0].children[2], 'Mitchell ', 100)
```

In [21]:
```
# Children of Guleph: Drayton , Kitchener
add_child(tree.children[1].children[1].children[0], 'Drayton', 100)
add_child(tree.children[1].children[1].children[0], 'Kitchener', 130)
```

In [22]:
```
# Children of 'New Hamburg: Stratford , Kitchener
add_child(tree.children[1].children[1].children[1], 'Stratford', 100)
add_child(tree.children[1].children[1].children[1], 'Kitchener', 130)
```

In [23]:
```
# Children of  Mitchell:St.Mary and Listowel
add_child(tree.children[0].children[0].children[0].children[0], 'St.Mary', 130)
add_child(tree.children[0].children[0].children[0].children[0], 'Listowel', 0)
```

In [24]:
```
# Children of  Drayton:,Listowel,Stratford,Guleph

add_child(tree.children[0].children[0].children[0].children[1], 'Listowel', 0)
add_child(tree.children[0].children[0].children[0].children[1], 'Stratford', 100)
add_child(tree.children[0].children[0].children[0].children[1], 'Guleph', 160)
```

QUESTION 3

Record and return the path (i.e. sequence of nodes) the search algorithm took to reach the goal. Hint: Add self.path = [] to Node and then record the parent and parent's parent (and so on) when a node is added to the frontier. list.extend() might prove useful. See the Python datastructures docs for more info about Python lists.

In [25]:
```
# Greedy helper

def find_min_h(frontier):
    # Helper func to find min of h (the heuristic function)
    min_h_i = 0
    if len(frontier) > 1:
        min_h_i = 0
        min_h = frontier[0].h
        for i, state in enumerate(frontier):
            if state.h < min_h:
                min_h_i = i
                min_h = state.h
    return min_h_i


def GreedySearchPath(init_state, goal_name):
    frontier = [init_state]
    explored = []

    while len(frontier):
        state = frontier.pop(find_min_h(frontier))
        explored.append(state.name)

        if state.name == goal_name:
            state.path.append(state.name)
            print(state.path)
            return True

        for child in state.children:
            if child.name not in explored:
                child.path.extend(state.path)
                child.path.append(state.name)
                frontier.append(child)
    return False
```

In [26]:
```
GreedySearchPath(tree, "Listowel")
```

```
['Kitchener', 'New Hamburg', 'Stratford', 'Drayton', 'Listowel']
```

Out[26]: True

In [ ]:

In [ ]: