CISC 340

Introduction to Artificial Intelligence

Lab Booklet #02

Lab Title: Intelligent Searching

Instructors

Memelord Brian Grey

Normie Chad Van Chu M.S.

Objectives

Searching problems are central to gaining an understanding of Artificial Intelligence. In this lab, we will be examining how backtracking search, a modified version of a depth-first tree search, can be used to make intelligent choices. We will be implementing our backtracking search on two classic problem: the map coloring problem and the eight queens problem. While there are many different ways to solve these problems, we'll see that we can get equivalent results using by setting simple, short circuiting constraints on what is an otherwise naïve algorithm.

After completing this lab, you will be able to:

- **Understand the theory of tree traversal**
- **Be able to implement a backtracking search algorithm**
- **Understand the importance of constraint satisfaction in order to short-circuit searches**

Resources

While the lab will explain everything that you need to know, you may want to consult the following sources for additional knowledge:

- https://en.wikipedia.org/wiki/Depth-first_search (https://en.wikipedia.org/wiki/Depth-first_search)
- https://en.wikipedia.org/wiki/Backtracking (https://en.wikipedia.org/wiki/Backtracking)
- https://en.wikipedia.org/wiki/Four_color_theorem (https://en.wikipedia.org/wiki/Four_color_theorem)
- https://en.wikipedia.org/wiki/Eight_queens_puzzle (https://en.wikipedia.org/wiki/Eight_queens_puzzle)

Deliverables

For this lab, you will need to submit:

- Jupyter notebook files (.ipynb), named and coded as instructed.
  - CISC 340 Lab 02 FA19.ipynb

Instructions & Questions

As discussed in class, a backtracking search algorithm can be thought of as a modified version of depth-first tree search algorithm where constraints are put on solution acceptance. Each of these constraints are specific, tailored to the problem attempting to be solved, and, when triggered, signal to the algorithm that this solution, and any further solutions derived from this solution, will be invalid. The algorithm can then backtrack and try different solutions at the next level up on the tree.

Part 1: The Four Color Theorm

In 1976, Kenneth Appel and Wolfgang Haken proved that any map can be colored using only 4 colors with the restriction that no two adjacent areas could have the same color. (Bear in mind, two regions that share a corner are not considered to be adjacent.)

In part 1 of this lab, we will be using a backtracking search paradigm to find acceptable map colorings.

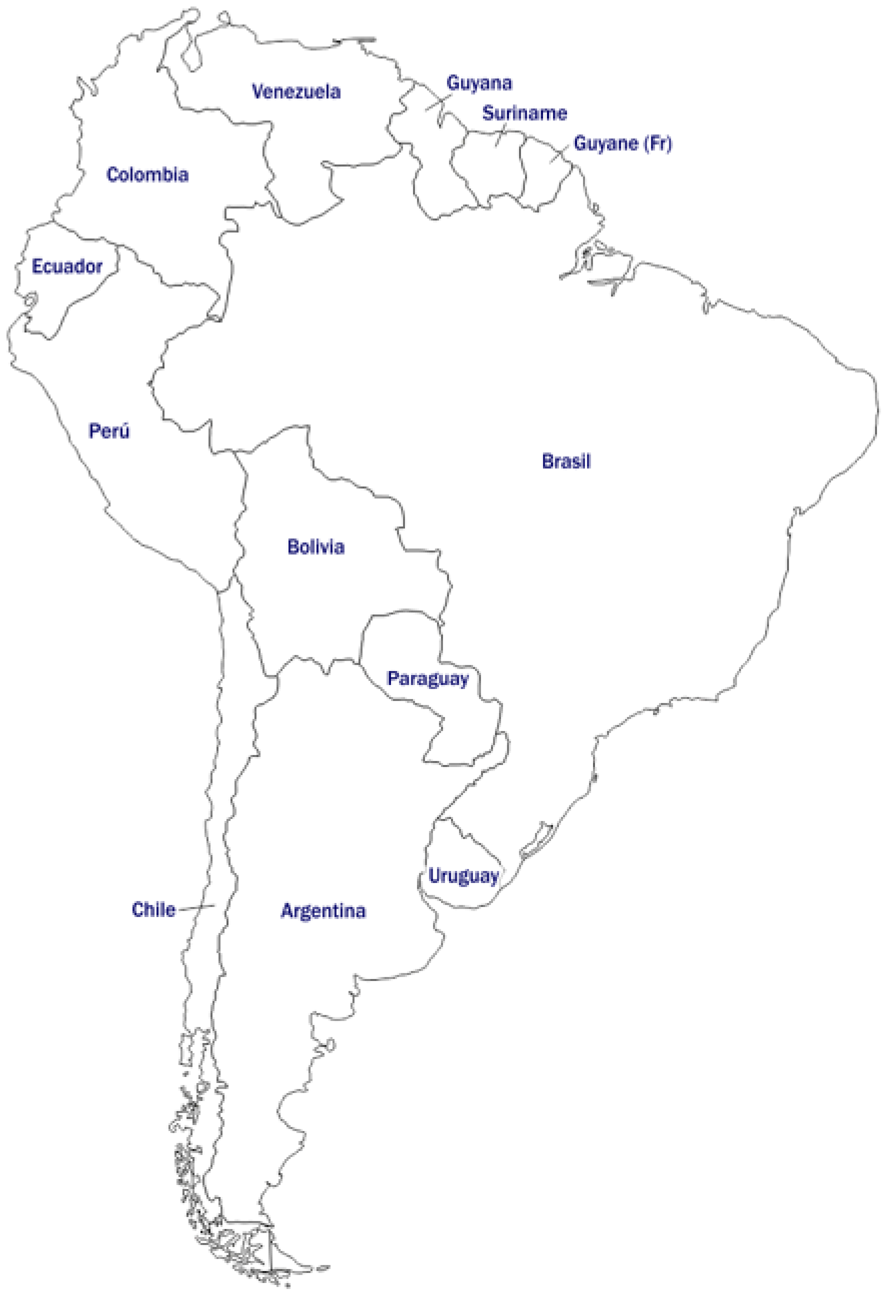
1. Explain, in detail, how you will store the adjacencies that exist in a map and how the data will be structured internally.

2. What condition, expressed in terms of your adjacency storage paradigm, will trigger backtracking in this implementation? You may express this in text, pseudocode, or code. Whatever format you choose, make sure that your description is complete.

3. Translate this map into the appropriate format to be read and used by your algorithm.

In [6]:
```
from IPython.display import Image
Image(filename="./images/SouthAmericaMap.png", format="png")
```

Out[6]:



In [ ]:

4. Implement a function called `colorMap` which takes a translated map as a parameter and returns a list of tuples expressing a valid coloring of the map using 4 colors. The first element of each tuple should be the name the country in the map and the second tuple should be the color. You may write any helper functions that you wish. Be sure to test this code with the map you created in Question 3, above, as well as with any other graph examples that you wish to use.

```
In [ ]:
```

Part 2: The Eight Queens Problem

The Eight Queens Problem is a classical puzzle first solved in 1850. The goal of the puzzle is to be able to place 8 queens on a chess board in such a way that none of them can take each other in a single move.

In part 2 of this lab, we will be using a backtracking search paradigm to valid positionings of queens on a chess board.

5. Explain, in detail, how you will represent the chess board and store the positions of the queens within your program.

6. What condition, expressed in terms of your chess board and queens' positions storage paradigm, will trigger backtracking in this implementation? You may express this in text, pseudocode, or code. Whatever format you choose, make sure that your description is complete.

7. Implement a function called `queens` which takes a positive integer as a parameter and returns a list of positions which solve the problem. The parameter both represent the number of queens to be placed and the dimensions of the chess board. For example, `queens(8)` is requesting a solution to the classic 8 queens problem on a traditional 8 by 8 chessboard. If you called `queens(6)`, this would request a solution where you are trying to place 6 queens on a 6 by 6 chessboard. You may write any helper functions that you wish.

```
In [ ]:
```