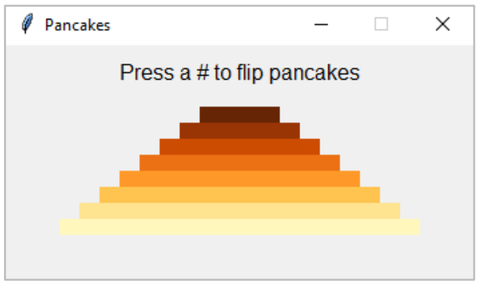
CSC 3510
Introduction to Artificial Intelligence
Florida Southern College

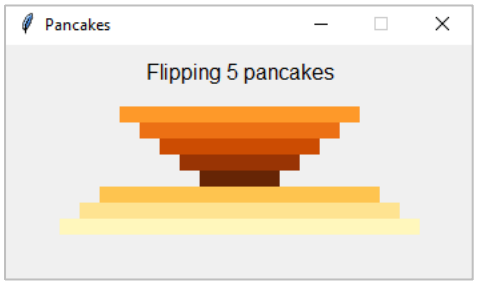
HW3: Informed Search
Due: Thursday, March 31, 2022

The purpose of this assignment is to ensure that you are able to:
- understand the similarities and differences of greedy best-first search and A* search
- program informed search algorithms in Python
- (optionally) work in collaboration with a peer to efficiently solve problems

1. Inspired by Bill Gates' 1979 discrete mathematics research paper discussing pancakes, you have decided to develop a Python pancake program. The goal of your program is to take a stack of pancakes of varying width and apply **greedy best-first search (GBFS)** to find a series of flips that will result in a sorted stack from smallest to largest width.

   Starter code has been provided for you (`pancakes.py`). To run the code, you will first need to install `matplotlib` via `pip`. The portions of code that you need to edit are identified by comments, e.g. `# ***ENTER CODE HERE***`, `# ***MODIFY CODE HERE***`. Here is what the GUI should look like at the start if programmed correctly *(left)* and an example of manually flipping some of the pancakes *(right)*:





   Also, here is a summary of the programming tasks that you must complete:

   (a) Draw pancakes in the `guisetup` function using `Line` objects from the graphics library. Try to match the style of the pancakes shown in the images above.

   (b) Move pancakes in the `flip` function. If programmed correctly, flipping three pancakes from the initial stack should result in the image on the right above.

   (c) Update the `cost` function so that cost is defined as the number of pancakes in the wrong position. The correct cost of the stack in the image on the right above is 4.

   (d) Implement greedy best-first search in the `gbfs` function. Your code will be evaluated based on its ability to correctly solve pancake stacks of varying size (3-9 pancakes) and with random manual shuffling. The output of the search algorithm should include print statements that describe how long the search took and what the solution is. Sample output is given on the next page. In this arbitrary example, the proposed solution is to flip 4 pancakes, then 2 pancakes, then 3 pancakes.

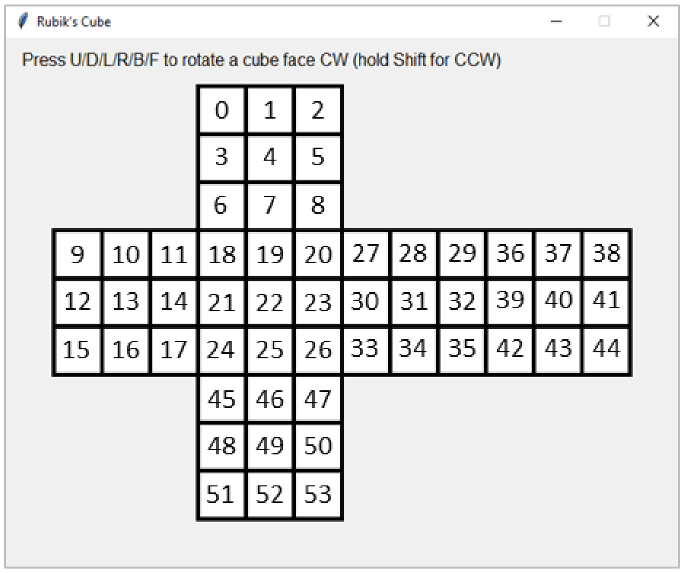
```
Running greedy best-first search...
  searched 6 paths
  solution: 423
```

BONUS. You will be eligible for extra credit if you successfully implement one or both of the additional programming tasks described below:

(e) For large stacks of pancakes with high initial cost, the solution may be too long to manually check. After running GBFS, enable the computer to demonstrate the solution if the user presses a key not already used in the main while loop (line 27). It is recommended that you incorporate a custom function and appropriate timing delays so the solution is understandable to the viewer.

(f) One of the optional command-line input arguments provided in the code is a seed for randomly shuffling the stack of pancakes before the user takes control (so that you can quickly get to the same node for searching, for example). Correctly implement this feature by drawing the original (unshuffled) stack, then move the pancakes based on the shuffling given from the random seed.

2. For this problem, you will solve a 3x3 Rubik's cube using **A* search**. Starter code has been provided for you (`rubiks.py`). The portions of code that you need to edit are identified by comments, e.g. `# ***ENTER CODE HERE***`, `# ***MODIFY CODE HERE***`. To assist you, the image below shows the underlying indices for each square; you will need these numbers for keeping track of the `state` of the cube.

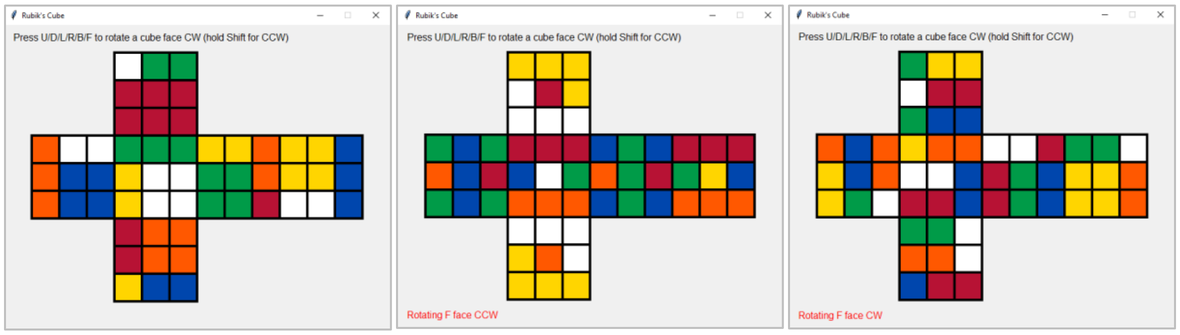


Rubik's Cube

Press U/D/L/R/B/F to rotate a cube face CW (hold Shift for CCW)

A few sample cubes are also included (`state01.txt`, `state02.txt`, `state03`.txt). Keep in mind that your code will be tested on other cubes as well, so it may benefit you to mix up the cube yourself for testing and debugging purposes.

Here is what the sample cubes should look like at the start if programmed correctly:



Rubik's Cube

Press U/D/L/R/B/F to rotate a cube face CW (hold Shift for CCW)

Rubik's Cube

Press U/D/L/R/B/F to rotate a cube face CW (hold Shift for CCW)

Rotating F face CCW

Rubik's Cube

Press U/D/L/R/B/F to rotate a cube face CW (hold Shift for CCW)

Rotating F face CW

Also, here is a summary of the programming tasks that you must complete:

(a) Color the cube when the program starts based on the initial state (if one is provided).

(b) Implement A* search in the `astar` function. Your code will be evaluated based on its ability to correctly solve multiple mixed cubes of varying difficulty. The output of the search algorithm should include print statements that describe how long the search took and what the solution is. Sample output is given on the next page. In the GUI, the user should be able to "type" the solution and see the cube solved in real-time.

```
Running A* search...
  searched 6 paths
  solution: BULL
```

(c) In order to correctly complete (b), you will need to implement the `cost` function. For this problem, define $g(s)$ as the number of moves (rotations of the cube) to reach a given state s and define $h(s)$ as the average number of incorrect color per face. Note that the center color on each face will never change!

(d) During A* search, you will need to simulate a sequence of moves in order to identify the resulting state for computing cost. Modify the `simulate` function to accomplish this. You should call the provided `rotate` function as part of your solution.
NOTE: We are not actually updating the GUI here!

(e) Is our current heuristic $h(s)$ admissible? Clearly explain your reasoning.

(f) Give an example of a heuristic that would be better. Explain your thought process.

BONUS. Implement the better heuristic in a function called `bettercost`.

Some important notes:

- Your code will be tested on a variety of cubes; part of the credit you can earn is based on the ability to correctly solve those cubes.
- If your code cannot run due to error, you will automatically receive a penalty (minimum 5 points for minor errors, 10 points for major errors); this is *in addition* to the points assigned to each of the required tasks above, so TEST, TEST, TEST your code!