MATH 182: HOMEWORK 4

Due Friday Dec 2 at 12pm.
Submit to "Homework 4 Programming" the following:

- The file(s) containing your code.
- The file `submission_a.txt`
- The file `submission_b.txt`
- The file `submission_c.txt`

Problem A: Depth first search

Let `cycle_or_toposort` which takes as input a directed graph `G`, and does the following:

- If `G` has a directed cycle, return a pair `("cycle", C)` where `C` is a cycle in `G`.
- If `G` is acyclic, return a pair `("toposort", T)` where `T` is a topological sort of `G`.

Implement `cycle_or_toposort` with a **single** depth first search.

The input graph `G` is represented as an array, where `G[i]` is an array, and it contains `j` iff there is an edge `(i, j)` in the graph.

For instance, if `data_a.txt` was the following:

```
[[], [0], [0, 1], [0, 1, 2]]
[[2], [0], [1]]
[[], [], [], [], []]
[[1, 2, 3], [0, 2, 3], [0, 1, 3], [0, 1, 2]]
[[]]
```

Then `submission_a.txt` could be the following:

```
("toposort", [0, 1, 2, 3])
("cycle", [2, 1, 0])
("toposort", [4, 1, 0, 3, 2])
("cycle", [2, 0])
("toposort", [0])
```

(Hint: Modify the algorithm for topological sort. At every step, the stack should only contain the vertices of the current branch, so to see if you have a cycle, check if the current vertex is already on the stack.)

Problem B: Network flows

Implement the function `max_flow` which takes as input a weighted graph `G` with source `0` and sink `1`, and outputs the max flow from `0` to `1`.

The weighted graph is represented as an array `G` where `G[i]` is an array of pairs `(j, w)` where `(i, j)` is an edge in the graph with weight `w`.

For instance, if `data_b.txt` was the following:

```
[[], []]
[[(1, 7)], []]
[[(2, 9)], [], [(1, 10)]]
[[(2, 1), (3, 1)], [], [(3, 1), (4, 1)], [(1, 1)], [(1, 1)]]
[[(2, 1), (5, 1)], [], [(3, 1), (4, 1)], [(1, 1), (2, 1)], [(1, 1)], [(3, 1)]]
```

then `submission_b.txt` would be the following:

```
0
7
19
2
2
```

Problem C: A* algorithm

Consider the Torus puzzle, which is similar to the Fifteen puzzle, except for the following two differences: it's on a 3×3 grid, and the pieces can "wrap around". More precisely:

- A position of the Torus puzzle is an arrangement of the numbers from $\{0, 1, 2, \ldots, 8\}$ into a 3×3 grid (we will consider 8 as the empty tile).
- The solved position is where the first row is $0, 1, 2$, the second row is $3, 4, 5$, and the third row is $6, 7, 8$.
- To do a move, switch the 8 with any tile "adjacent" to it in any of the four directions (which are all possible, since we are allowing tiles to wrap around). For instance, if the 8 is at the top right, we are allowed to switch it with the bottom right tile.

Let `lower_bound` be the function which does the following:

- For each of the eight non-empty tiles, calculate the minimum number of moves required to put it in its correct place (it should only require 2 at most).
- Add those eight numbers up, and return their sum.

Using `lower_bound` as a heuristic for the A* algorithm, implement the function `optimal_solution` which takes as input a position of the Torus Puzzle, and outputs the minimum number of moves required to solve it.

For instance, if `data_c.txt` was the following:

```
[[0, 1, 2], [3, 4, 5], [6, 7, 8]]
[[0, 1, 2], [3, 4, 5], [6, 8, 7]]
[[0, 1, 2], [3, 4, 8], [6, 7, 5]]
[[0, 1, 2], [3, 4, 5], [8, 7, 6]]
[[0, 1, 8], [3, 4, 5], [6, 7, 2]]
[[0, 1, 2], [3, 8, 4], [6, 7, 5]]
```

Then `submission_c.txt` would be the following:

```
0
1
1
1
1
2
```