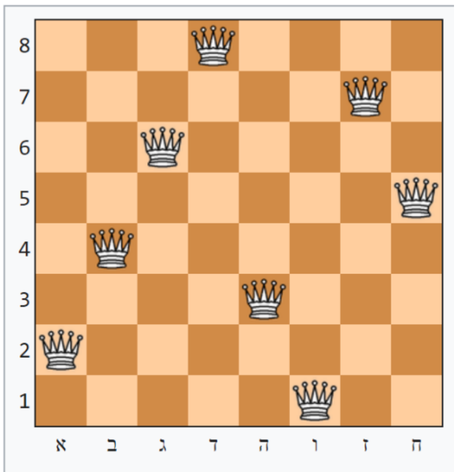
HW #2 – Solving the n-queen problem

Due: Oct. 23, 2023

We have discussed the 8-queens problem (https://en.wikipedia.org/wiki/Eight_queens_puzzle) in class.

Now is your time to have some fun with it!!!



פתרון אפשרי לבעיית שמונה המלכות

I will give you code to use as a base, dfs_queen, for the n-queen problem that uses DFS with backtracking. It works, but isn't nearly as efficient as many other possibilities that we'll discuss in class. Your homework is to implement three additional ways of solving the problem: a very stupid "British Museum" algorithm, one that uses search ordering heuristics to aid in the search (like forward checking) and is based on this code and a second that uses randomization for all elements and then a heuristic repair method (like Hill-climbing) to find a solution based on that initial, random placement. I will then ask you to compare the two and let me know which one worked best over 100 runs. The first method is more likely to be deterministic and the second method will have stochastic elements making the average more important (right?!). Pedagogic node: you could say that the methods that use heuristic repair think more "humanly" (at least more like *me*) and those using search ordering heuristics are more "rational".

The workbook I made about this code is at:

https://colab.research.google.com/drive/1puD0kHn_0jLIZbtEOzNW4Atp2JNHcaYB?usp=sharing

The code that I provided already has two counters:

```
number_of_iterations, number_of_moves
```

which counts the number of iterations and queen moves needed to find the solution.

Similar to homework #1, also add a counter for the time your process ran for solving each of the options.

Your goal: make these numbers as low as possible **without *adding additional loops*.** Secondary goal—try solving as high a n-queens problem as possible.

You **are** encouraged to make the next_row_is_safe function more sophisticated in the second method. Any iteration with the placement of a queen (even a randomly placed one) is considered an iteration, **including** for the first (or subsequent) placement of **all** n queens. I implemented one possibility place_n_queens which implements this idea and places n queens randomly. Note that the two implementations will likely need some tweaks to the code I provided.

Assignment: Compare 4 implementations and their counters:

1. The British Museum—This algorithm is the one to beat. For example, I found that solving the 8-queens problem this way took over 4 billion (!) iterations.
2. DFS with Backtracking – pretty straight forward and deterministic. Given as a baseline but without the timing element.
3. Heuristic Repair/Stochastic Search—can be just like we discussed in class (with random initial placement) or not (only putting down one queen and then doing DFS). Check what works for you!
4. Forward Checking (only in the second week of class).

To show that your best algorithm works well, create a table for the number of `number_of_iterations`, `number_of_moves`, and `time` for each of the queens problems solved starting at n=10. The name could look like:

n=	10	11	12	13
number_of_iterations	100	250	550	1000
number_of_moves	200	300	500	50
time	4	5	15	10

and the goal is to make the table as wide as possible—i.e., n= a big number.

This table should be made for each of the solutions, but some of them (like the British Museum) should be stopped at a low number, and hopefully either the heuristic repair or forward checking options will get you quite far. Note: if you get to 40 for any algorithm, you can stop 😊

Grade Breakdown:

20 points – providing a working solution for the both the updated backtracking, stochastic and British Museum algorithms.

20 points – documentation of logic used – especially for your best option

10 points – solving all n-queens problems up until 25 queens with at least **one** of the algorithms

10 points – solving all n-queens problems up until 40 queens with at least **one** of the algorithms

40 points – how well your code works

Submission: via Git as usual.