HW1 Drive Ya Nuts
Discussion and Program Logs
Yixin Sun (Venus)

Q1: Generate

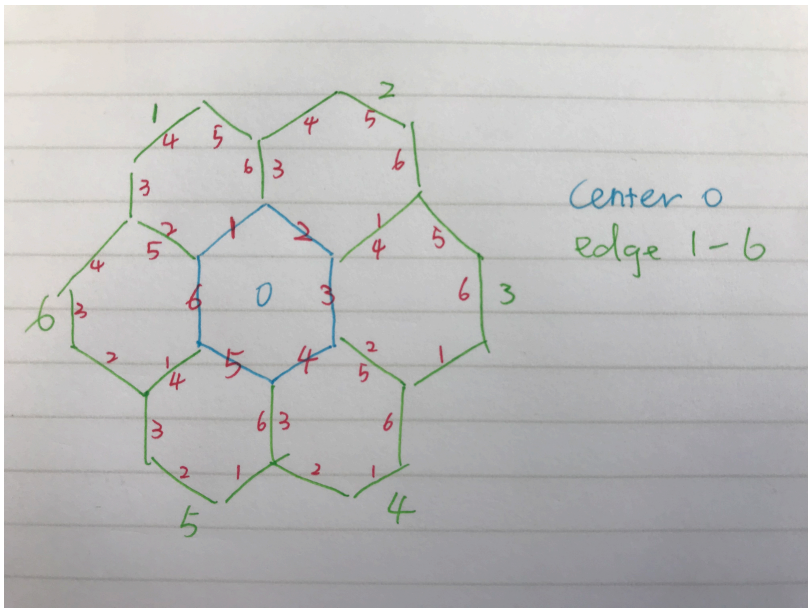
In order to generate a N sized puzzle, the logic I'm using here is to generate all the permutations of lists for number starting from 2 to N, and then select N+1 of them and append 1 to the beginning of each one. And then it will return a list of such lists as nuts representing the puzzle.

```
* (generate 5)
((1 5 3 4 2) (1 2 4 3 5) (1 2 5 3 4) (1 4 5 3 2) (1 4 3 5 2) (1 3 4 5 2))
* (generate 6)
((1 4 6 5 2 3) (1 6 3 2 4 5) (1 4 6 3 5 2) (1 2 5 6 4 3) (1 3 5 6 4 2)
 (1 3 4 5 2 6) (1 3 5 4 6 2))
* (generate 7)
((1 6 2 5 4 3 7) (1 5 7 2 6 4 3) (1 7 6 4 2 5 3) (1 7 5 2 4 3 6)
 (1 7 5 2 4 6 3) (1 2 5 6 3 7 4) (1 3 7 5 6 2 4) (1 2 5 3 7 4 6))
* (generate 8)
((1 4 5 7 8 3 6 2) (1 4 2 3 8 6 7 5) (1 2 5 7 3 4 6 8) (1 8 6 4 5 7 3 2)
 (1 6 2 8 4 7 5 3) (1 8 6 2 5 4 3 7) (1 6 8 2 5 4 7 3) (1 6 4 5 2 8 3 7)
 (1 7 8 6 2 3 5 4))
*
```

Q2: Solve

Since I save my nuts using a list data structure, I have first created a system to organize the positions of the numbers for the nuts. First for the list of nuts, we define the 0-indexed list to present the center nut, and then clock-wisely letting the lists of indexes 1 to (N-1) to present the rest. And then for each edge nut (every nut except the center nut), we let the number with index same as the entire nut's index to present the edge touching the center nut, and clock-wisely positioning the rest of the indexes. As for the center nut, we let the numbers share the same index with the one it's touching. You might have noticed that when I'm referring to these numbers on the nuts, I'm starting my indexes from 1, this is just for convenience with referring to the indexes of the nuts, and I've designed my codes to accommodate that when I'm accessing the specific numbers.

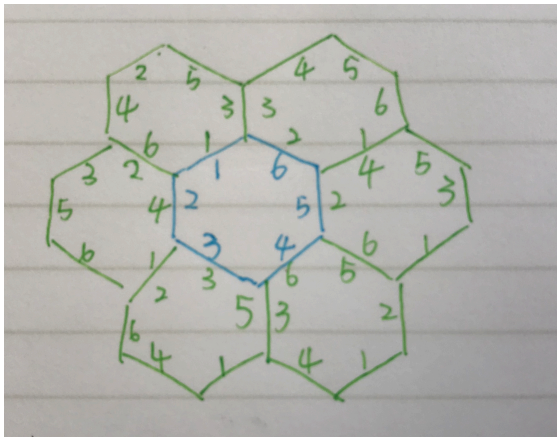
Since the words above is very confusing I figured a graph will be helpful, so here is an illustration of my system with a hexagon puzzle. All the number listed here are indexes, not the actual number. And for the indexes for each individual nut (red ink), they are presented starting from 1 for human convenience. In the code when referring to them, I've written it in the way to take care that the list actually starts with index 0.



So, the standard nut

```
(defvar *nuts* '(
            (1 6 5 4 3 2)
            (1 6 4 2 5 3)
            (1 2 3 4 5 6)
            (1 6 2 4 5 3)
            (1 4 3 6 5 2)
            (1 4 6 2 3 5)
            (1 6 5 3 2 4)))
```

in my system would look like this:



Now about the brute force solution, I basically permutated the position of all the nuts on the board and then align the edge ones to match the center one and see if the board is finished. And the method will return a solution if it finds one.

```
* (solve (generate 6))
((1 6 4 3 2 5) (1 2 4 6 5 3) (2 6 3 4 5 1) (6 1 4 2 5 3) (6 5 4 3 1 2)
 (6 3 5 1 2 4) (1 3 4 6 2 5))
* (solve (generate 6))
NIL
* (solve nuts)
((1 6 2 4 5 3) (1 4 6 2 3 5) (1 6 5 3 2 4) (6 5 2 1 4 3) (1 2 3 4 5 6)
 (1 6 4 2 5 3) (2 1 6 5 4 3))
*
```

Q3: Pruning

The strategy I'm using here is to first put down two nuts (center nut and one to start with) and then using DFS to start putting down the rest of the nuts one by one and "prune" as soon as it's no longer possible to complete the puzzle. In my DFS it will only continue putting down the next one if the previous put-down nut is "correct" in its position. The method will return true as soon as it finds one solution.

```
*  (solve-dfs (generate 6))
NIL
*  (solve-dfs (generate 6))
T
*  (solve-dfs nuts)
T
*  (solve-dfs (generate 7))
T
*  (solve-dfs (generate 8))
NIL
*  (solve-dfs (generate 9))
NIL
*
```

And for bigger-sized Ns, pruning definitely improves the speed of finding solutions.

Q4: Count

Here I've modified my DFS solving method so that instead of returning a Boolean value it collects all the results and return them in a list. After that I just count the number of T's to know how many solutions there are for the puzzle.

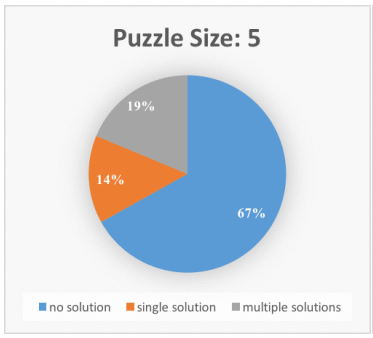
```
* (count-dfs (generate 6))
0
* (count-dfs (generate 6))
2
* (count-dfs nuts)
1
* (count-dfs (generate 7))
0
*
```

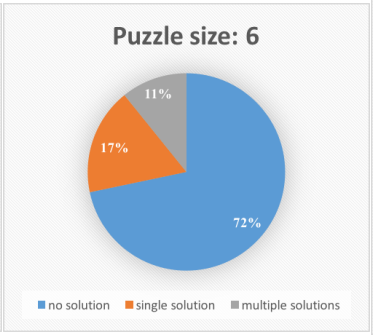
Q5: Graph

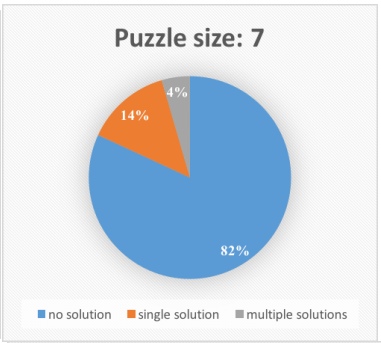
Unfortunately, starting from 11 the heap is exhausted when it's running…

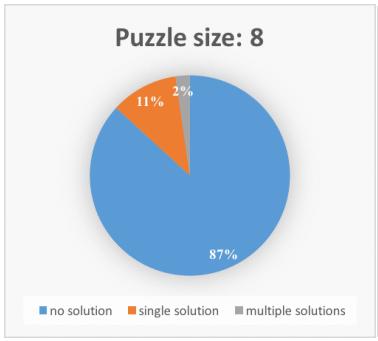
```
* (count-puzzle (solve-puzzle 5 1000))
(669 144 187)
* (count-puzzle (solve-puzzle 6 1000))
(717 175 108)
* (count-puzzle (solve-puzzle 7 1000))
(819 136 45)
* (count-puzzle (solve-puzzle 8 1000))
(868 109 23)
* (count-puzzle (solve-puzzle 9 1000))
(932 58 10)
* (count-puzzle (solve-puzzle 10 1000))
(962 35 3)
*
```

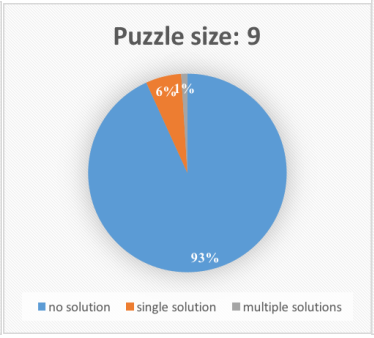
It could be observed here that as the size of the puzzle grows, the amount of solutions, whether it be one or more, decreases.

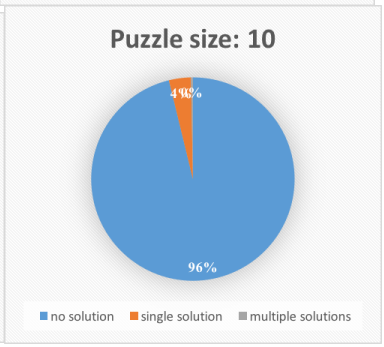












Q6: Main

```
* (main)
((1 6 2 4 5 3) (1 4 6 2 3 5) (1 6 5 3 2 4) (6 5 2 1 4 3) (1 2 3 4 5 6)
 (1 6 4 2 5 3) (2 1 6 5 4 3))
*
```

And it “translates” to: