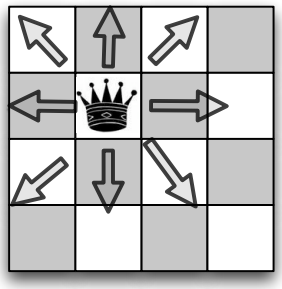
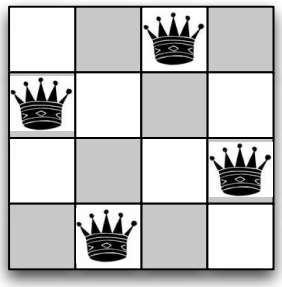
Work on the following problems in groups of three. Make sure everyone in your group understands the problem before moving on to the next; you're not necessarily expected to finish all the problems.

I. Search Space Formulation

The *n*-queens problem is a popular problem originally proposed by a chess player in 1848. It requires putting *n* queens on an *n* x *n* chess board such that none of the queens is attacking any of the other queens. In case you're unfamiliar with chess, a queen can move in any direction for any number of squares:





Concretely, the picture on the right, above, shows a solution to the 4-queens problem. Finding a solution to the *n*-queens problem can be thought of as a search problem.

1) Formulate *n*-queens as a search problem. This means you should specify:

a. Start state:

b. Successor function:

c. Search space (what does a valid state in the search tree look like)

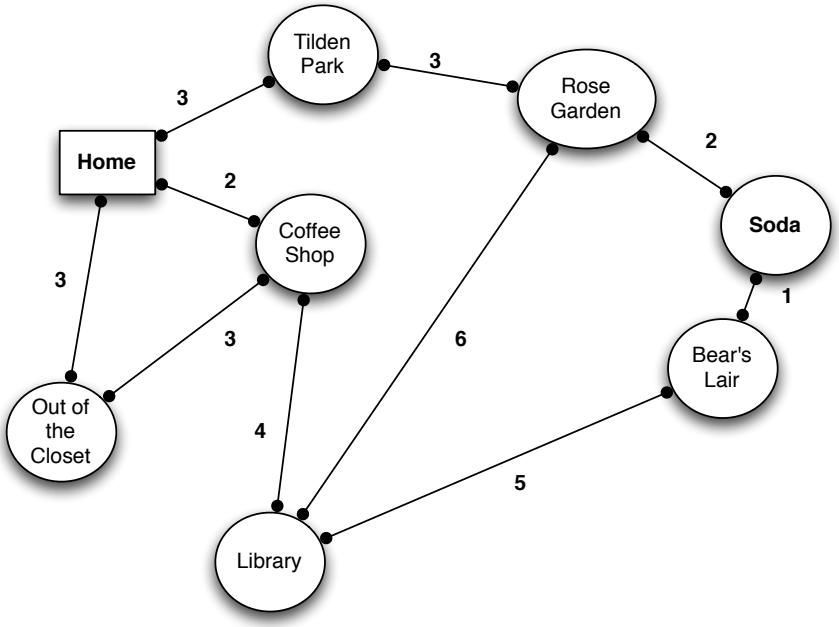
d. Goal test

2) Like many search problems, the *n*-queens problem can be parameterized in a number of different ways to create a valid search problem. Come up with a different way of formulating it as a search problem, specifying the same four items as in (1). Your new formulation shouldn't just change the way you encode the problem but should also change the successor function.

4) Does either of your search formulations have any advantages over the other? Think about this in terms of memory and time requirements, and the number of nodes you'd expect to expand before finding a solution.

5) *N*-queens can also be solved using recursion. For a little Python practice, write a recursive program to solve *n*-queens. (** Come back to this one after you've done the rest of the worksheet or solve it on your own after class.)

II. Search techniques



Maia just moved to a new apartment, and has to figure out the shortest paths to various places from her new place.

1.) Draw the first two levels of the search tree for creating a route from Home to Soda. Assume for all parts of this problem that when we expand a node, successors are ordered alphabetically from left to right.

2.) What happens if we naively try to do depth-first tree search for this problem?

3.) How can we get around the problem in (2)?

4.) Draw the search tree for the following approaches for finding a route from Home to Soda, assuming a graph search algorithm, and show only the nodes expanded by each search. How many nodes are expanded in each search? What is the cost of the route found by each search?

a. Breadth-first search

b. Depth-first search

c. Greedy search

d. Uniform-cost search

5.) Which of the above methods techniques found the shortest route? Did any find the shortest possible route? Are any of them guaranteed to find the shortest possible route?

6.) For each search technique, give one pro and one con of using the technique. Hint: think about things like optimality, time and space complexity, whether the technique is guaranteed to terminate, etc.

a. Breadth first search

Pro:

Con:

b. Depth first search

Pro:

Con:

c. Uniform cost search

Pro:

Con:

III. Formulating problems as search

Think of a problem that doesn't involve route-planning but that can be formulated as search; for example, can your favorite game be thought of as a search problem? How abut planning your schedule? With your group, pick a problem that seems interesting and come up with:

a. The start state

b. Successor function

c. State space

d. How to determine if you're in the goal state