CS 278 **Algorithm Analysis**
Assignment # 6 Due Date: November 29, 2001

Note: Create a subdirectory in your home directory and call it `<your_last_name>.6`, where `<your_last_name>` is your real last name (for example, `bhatia` for me). Do all the programs for this assignmnet in that directory. After you are done, submit the code by typing the following command:

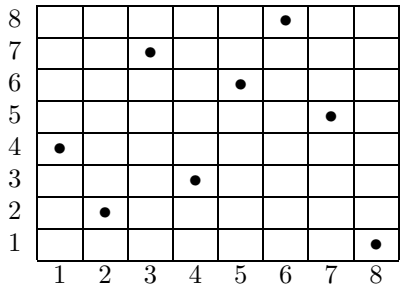
```
~sanjiv/bin/handin <your_last_name>.6 cs278 6
```

Again, do not forget to substitute for `<your_last_name>`. The command should be executed from your home directory.

1. **Eight Queens Problem**

   The problem is to place eight queens on the empty chessboard in such a way that no queen attacks any other queen.

   Board Representation. One natural choice is to represent the position by a list of eight items, each of them corresponding to one queen. Each item in the list will specify a square of the board on which the corresponding queen is sitting. Further, each square can be specified by a pair of coordinates (X and Y) on the board, where each coordinate is an integer between 1 and 8. In the program, you can represent such a pair as (X,Y). The figure below shows one solution of the eight queens problem and its list representation.



Positions specified by the list: [(1,4),(2,2),(3,7),(4,3),(5,6),(6,8),(7,5),(8,1)]

Having chosen this representation, the problem is to find such a list of the form

$$(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4), (x_5, y_5), (x_6, y_6), (x_7, y_7), (x_8, y_8)$$

which satisfies the no-attack requirement. Our procedure solution will have to search for a proper instantiation of the variables x and y. As we know that all the queens have to be in different columns to prevent vertical attacks, we can immediately constrain the choice, and make the search task easier. We can thus fix the x-coordinates so that the solution list will fit the following, more specific template:

$$(1, y_1), (2, y_2), (3, y_3), (4, y_4), (5, y_5), (6, y_6), (7, y_7), (8, y_8)$$

A more economic representation of the board will retain only the Y-coordinates.

At any node in the search tree, the list of queens can look like:

$$[(x, y), \texttt{others}]$$

The following conditions must hold:

- There should be no attack between the queens in the list `others`; *i.e.*, `others` itself must also be a solution.
- x and y must be integers between 1 and 8.
- A queen at square (x, y) must not attack any of the queens in the list `others`.

A recursive `noattack (q, qlist)` function can be defined as follows:

- If `qlist` is empty then return `true`
- If `qlist` is not empty, it has the form `[q1, qlist1]` and two conditions must be satisfied:
  - the queen at `q` must not attack the queen at `q1`, and
  - the queen at `q` must not attack any of the queens in `qlist1`

To specify that a queen at some square does not attack another square is easy: the two squares must not be in the same row, the same column, or the same diagonal. Our solution template guarantees that all the queens are in different columns, so it only remains to specify explicitly that:

- the Y-coordinates of the queens are different, and
- they are not in the same diagonal, either upward or downward; that is, the distance between the squares in the X-direction must not be equal to that in the Y-direction.

Implement two programs to solve the problem of placing eight queens on the chessboard. The first of these programs should use the depth-first search strategy while the second one should be based on breadth-first search, as explained in the class. *Document your programs well.*

Output: At every node from where you can go to the successor node, print the configuration of the state of the chessboard. This will also help you in the debugging process. Keep a counter of the number of nodes expanded and print it as a part of the print statement. For example, the first print statements in depth-first search will look like:

```
0.  0  0  0  0  0  0  0  0
1.  1  0  0  0  0  0  0  0
2.  1  0  1  0  0  0  0  0
 ⋮  ⋮  ⋮  ⋮  ⋮  ⋮  ⋮  ⋮  ⋮
```