CSCI 4150 Introduction to Artificial Intelligence, Fall 2000
Assignment 2 (52 points): out Friday September 8, due Friday September 15

We will most likely ask you to turn in some parts of this assignment on paper and some parts electronically. Stay tuned for details.

The only addition to the set of functions you should know are `map` and `apply`. Again, you may not use the "more advanced" features of Scheme, in particular `set!` and its variants and any iterative control forms.

1. (3 points) Write a function `(mag x)` using the that returns the magnitude of a vector `x` represented as a list of numbers. The vector `x` may be of any dimension. For example:

   ```
   (mag '(3 4))              ==>   5
   (mag '(4 -2 8 7 11))      ==>  15.937377450509228
   ```

   You function should be nonrecursive and should use the `map` and `apply` functions.

2. (10 points) Write the following functions:

   (a) `(positions lst e)` which returns a list of numbers corresponding to the position of every occurrence of element `e` in the list `lst`. For example:

   ```
   (positions '(1 3 5 3 3 7 2) 3)  ==>  (1 3 4)
   ```

   The order of the elements in the returned list is not important.

   (b) `(positions-list lst elements)` which returns a list of positions of every occurrence of an element of the `elements` list in the list `lst`. For example:

   ```
   (define test-list '(a b c e b c f k l m o c k f a c))
   (positions-list test-list '(b c k))  ==> (4 1 15 11 5 2 12 7)
   ```

   Your function should not be recursive and should use `map` and/or `apply`. The order of the elements in the returned list is not important.

3. (8 points) Write a function `(swap lst i j)` that returns a new list based on `lst` except that elements `i` and `j` are swapped. Assume that the first element of a list is element 0. For example:

   ```
   (swap '(a b c d e f g h) 3 7)  ==>  (a b c h e f g d)
   ```

4. (16 points) Another "toy problem" for search algorithms is the 8 queens problem as described in Section 3.3 of our text. In this question, you will write functions to solve the N queens problem with our breadth-first search algorithm. Assume we have defined a global variable `nqueens` in the following description.

   The approach we will take in this question is to place a queen in each column starting from the right side of the board. We will represent the state of the board by a list of integers, each of which is in the range 1 to `nqueens`. The last element of this list is the queen position (i.e. row) for the rightmost column, the second to last element is for the column second from the right, and so on. We will ensure that whenever we generate a node, that the corresponding state is valid. Therefore, when the state has a length of `nqueens`, we have solved the problem.

   (a) Write a function `(nq-valid?  state)` which determines whether the given state is a valid state for the N queens problem. Assume that you only need to check whether the first element violates any constraints. (I.e. assume that `(cdr state)` is a valid state.) You must check whether this new queen is on the same row or diagonal as another queen that has already been placed.

(b) Write a function (nq-child-states state) which takes a valid state and returns a list of all valid states that result from placing a queen in the next column to the left.

(c) On the course home page, I have provided the functions/variables nq-start, nq-goal?, and nq-children. The last will call your nq-child-states function. Use these definitions with your code from the first two parts of this question to solve this problem using the breadth-first search function from class (which you will also find on the course home page). How many nodes are evaluated in order to find a solution for the 6 queens problem? the 8 queens problem?

5. (12 points) There are three ways to deal with repeated states in search problem:

- Don't return to the state you just came from.
- Don't create paths (through the state space) with cycles.
- Don't generate any state that was previously generated.

These approaches are in order of increasing effectiveness and computational overhead. In this problem, you will use the first method to improve the breadth first search implementation we did in class.

(a) Write a function (remove-repeat grandparent-node node-list). In the context of our search algorithms, the node-list is a list of children of the current node. The grandparent-node is the current node's parent.

Your function should return a list of all nodes from node-list whose state is different from the grandparent-node. Assume that nodes are lists in which the first element is the state.

(b) Modify the breadth-first search implementation from class to incorporate your remove-repeat function. Call this new function rr-bfs. Be sure to rename the helper function and any recursive calls in these functions.

Note that because the parent of the root node is (), this creates a special case that you must handle either in your modified breadth-first search or in your remove-repeat function.

Solve the missionaries and cannibals problem with this modified breadth-first search implementation. How many nodes are visited in order to find a solution?

6. (3 points) This problem is for those of you who are looking for a little more challenging problem or just have some extra time on your hands. Do this problem last because it's only worth 3 points!

Write a procedure (permute x) which takes a list x and produces a list of all the permutations of x. For example:

```
(permute '(a b c)) ==> ((a b c) (a c b) (b a c) (b c a) (c a b) (c b a))
```

The order is not important so long as you generate all permutations.