Homework 2 - Searching
COEN 266, Spring 2018

Overview

For this assignment, you will be implementing an algorithm that will sort a list of locations on a map into order using only swap operations, so that they form a continuous path through adjacent locations. For example, consider this portion of the map of the US:

If your algorithm were given the input:

```
(Tennessee Iowa Kentucky North-Carolina Missouri)
```

It might find the path:

```
(North-Carolina Tennessee Kentucky Missouri Iowa)
```

Problem Space

Initial State

The initial state of your search algorithm will be an ordered list of locations.

Goal State

A goal state will be a list of locations where any two locations adjacent to each other in the list are adjacent to each other in the map. Physical adjacency information is provided on the DC workstations in the file `/scratch/COEN266/map.scm`. Copy this file into your assignment. The file defines an adjacency-map which is a list of lists. Each of these sublists includes a location as its first item and all locations that are adjacent to that location as its remaining items.

Transition Model

Each state may be transmuted to another state by switching two locations. For example:

```
(North-Carolina Tennessee Kentucky Missouri Iowa)
```

may be modified to:

```
(North-Carolina Iowa Kentucky Missouri Tennessee)
```

by swapping the 2nd and 5th items in the list.

Cost Model

The goal of your search is to find a solution that requires the fewest number of swaps.

Basic Assignment (80 points)

Implement an iterative-deepening DFS search. The function should be named `id-dfs` and should take one parameter: the initial list of locations. Your depth cutoff should start at 1 and increment to the length of the initial state - 1 (any valid solution, if possible, will occur within that limit).

`id-dfs` should return `#f` if those locations cannot be ordered into a valid solution state. Otherwise, it should return a list containing two items. The first item will be the original list of locations sorted into a valid solution state. The second will be a list of pairs indicating which items need to be swapped in order to reach the solution state.

For example:

```
scheme@(guile-user)> (id-dfs '(Tennessee Iowa Kentucky North-Carolina Missouri))
$1 = ((North-Carolina Tennessee Kentucky Missouri Iowa) ((1 2) (1 4) (4 5)))
scheme@(guile-user)> (id-dfs '(California))
$2 = ((California) ())
scheme@(guile-user)> (id-dfs '(Arizona Alaska))
$3 = #f
```

The input and output format are mandated, and should be strictly adhered to. You may decide to use a different state representation in your functions, and this may require you to have wrapper functions that perform this conversion.

All functions should be implemented in a file named `hw2.scm`, or in files included from `hw2.scm`, so that I only have to load `hw2.scm` in order to run your code. Please do not include tests or other extraneous items in your code (but do include comments).

Suggested Progression

I would recommend implementing the assignment in the following order:

1. Decide on a state representation. In my opinion the easiest choice is for a state to include the current ordered list of locations and the swaps performed so far to get there, which looks like the output format of id-dfs. For example:

```
((Alabama Mississippi Florida) ((1 2) (2 3)))
```

This is not a requirement, but it is likely to simplify the code quite a bit.

2. Implement a "perform swap" function that will swap the values at two indices in a list. For example

```
(swap-elements 2 3 '(a b c d))
```

should return

```
(a c b d)
```

This function can take advantage of the functions you wrote in assignment 1 to retrieve and replace an element at a specific index in a list.

3. Implement a function that will determine if a state is adjacent to another state. For example:

```
(is-adjacent? 'Florida 'Georgia)
```

should return

```
#t
```

4. Implement a function to get all possible children of a state. This will require that you evaluate all possible swaps from a given state. For example, with a list of size 5 that would include each of the following swaps:

```
(1 2) (1 3) (1 4) (1 5) (2 3) (2 4) (2 5) (3 4) (3 5) (4 5)
```

For example, the following call:

```
(get-children '((Alabama Arizona Alaska) ()))
```

Should return:

```
(((Arizona Alabama Alaska) ((1 2)))
 ((Alaska Arizona Alabama) ((1 3)))
 ((Alabama Alaska Arizona) ((2 3))))
```

5. Implement a function to determine if a state is a goal state. For example:

```
(is-goal-state? '((Alabama Alaska)()))
```

should return

```
#f
```

6. Implement a depth-first search algorithm.

7. Implement a depth-limited DFS search algorithm.

Advanced Assignment (100 points possible)

In addition to the basic assignment, implement an A* search of the same problem space. In order to do this, you will have to come up with a reasonable heuristic for optimistically estimating the number of swaps required to complete a solution. If you choose to attempt this option, please provide a brief text description of your heuristic, either as comments in the Scheme source file or as a separate .pdf or .txt document.

For this assignment, your function should be named `A*`, and should take the same parameters and have the same return value format as those used for the basic assignment.

Since an unsolvable initial state with a standard A* algorithm would iterate indefinitely, your code won't be tested with unsolvable states. You can choose to implement a maximum # of steps terminating condition, if you wish, but it is not required.

Extra Credit Opportunities

Opportunity #1 - Wildcard support (20 points)

In addition to the advanced assignment, for extra credit incorporate support for a "Wildcard" location. That location may be used to represent any location for the purposes of a solution, and should be represented in the solution by a list with Wildcard as the first element and the location it represents as the second element. For example, consider the input:

```
(Wildcard Oregon Arizona)
```

A possible solution would be:

```
((Oregon (Wildcard California) Arizona) ((1 2)))
```

Another valid solution would be:

```
((Oregon (Wildcard Nevada) Arizona) ((1 2)))
```

Opportunity #2 - Find my bugs (up to 10 points)

For each significant flaw in the assignment that you are the first student to report, I will add 5 points to your assignment grade.