CSCE476/876 **Spring 2003**

Homework 4

Assigned on: Wednesday February 25, 2003.

Due: Exercices 1, 2, 3, 4, and 5. Pen and paper. Monday, March 3, 2003.
Exercices 6 and 7. Programming. Friday, March 14, 2003.

Pen and paper part should be returned in class.
Programming assignment should be submitted with `handin`.

---

1 AIMA, Exercise 2.4, Page 50. (5 points)

2 AIMA, Exercise 3.2, Page 87. (5 points)

3 AIMA, Exercise 3.9, Page 89. (10 points)

4 Basic search strategies (Total 12 points)

Assume you have the following search space:

State	Next-state	Cost
A	B	4
A	C	1
B	D	3
B	E	8
C	C	0
C	D	2
C	F	6
D	C	2
D	E	4
E	G	2
F	G	8

1. Draw the state space of this problem as a directed graph in which the states are represented as nodes and actions/operators as directed edges labeled by their cost. (2 points)

2. Assume that the initial state is A and the goal state is G. Use each of the following search strategies, *both with and without a loop control*[1], to find a path from the initial state to the goal state.

[1]Consider the most elaborate strategy for loop control, i.e. the third one in AIMA.

- Breadth-first search (2.5 points)
- Depth-first search (2.5 points)
- Uniform cost search (2.5 points)
- Iterative deepening search (2.5 points)

At each step of a search algorithm, report:

- which node is being expanded, and
- the content of the fringe.

Also, report the solution path found.

5 Evaluation function (Total 8 points)

Adapted from AIMA, Exercise 4.1.

With $g(n)$ being the path length,

1. Suppose that we run a greedy search algorithm with $h(n) = -g(n)$. What sort of search will the greedy search emulate?
   Explain. *(4 points)*

2. Suppose that we run a search algorithm with $h(n) = g(n)$. What sort of search will the greedy search emulate?
   Explain. *(4 points)*

6 A special set-difference (Total 10 points)

Amy Davis, a former graduate student at CSE, needed to implement the following function for her experiments.

The function should remove some specified elements of a set from a partition of the set. It takes two lists, a list of atoms `lost` and a partition of the set `partition` (with `lost` $\subseteq \bigcup_{S \in \texttt{partition}} S$). It returns a new list obtained by removing, in the elements of `partition`, the elements of `lost`.

Example: Given,

`(setf lost '(a b c d)`, and

`(setf partition '((a b f) (c) (h d j k))`,

`(amys-set-difference partition lost)` should return `((f) (h j k))`.

You are asked to implement the function `amys-set-difference` in two manners:

1. iteratively, and *(5 points)*

2. recursively. *(5 points)*

Hint: check the definition of `remove-if`, which may be useful at some point.

7 Data structures in Common Lisp (Total 50 points)

Using `defstruct` (see LWH, Chapter 13), create data structures in Common Lisp to represent the map of Romania. Include the information about the distances between two cities linked by a road as well as the distance from any given city to Bucharest as indicated in Figure 1.

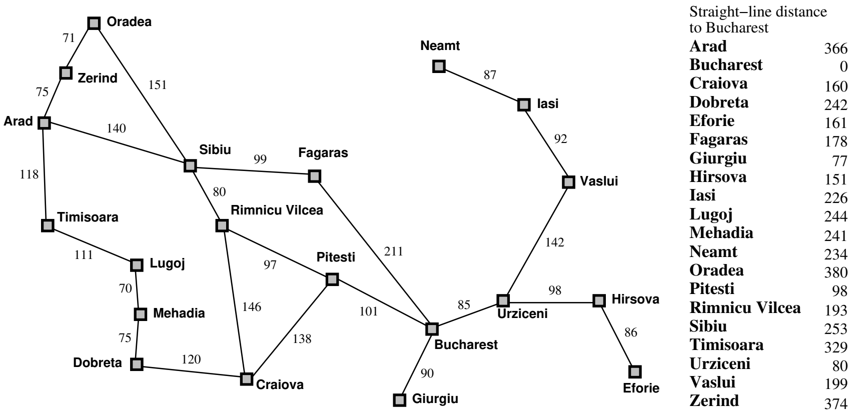

Figure 1: *Map of Romania with road distances in kilometers and straight-line distances to Bucharest. AIMA page 95.*

Indications (follow illustration in Figure 2):

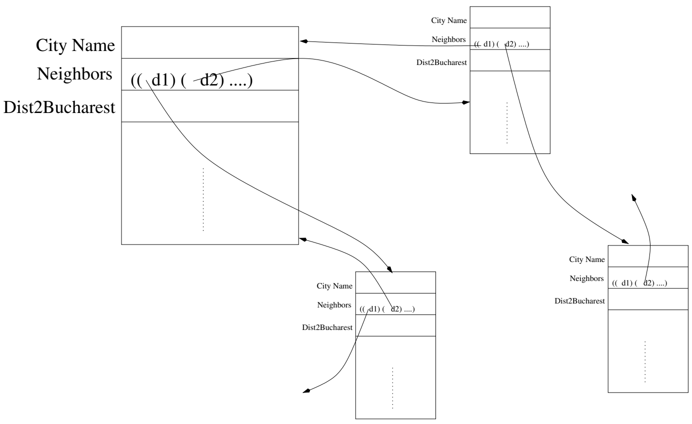

Figure 2: *Data structures.*

- Create a data structure for a city using `defstruct`.
- Include an attribute to store the name of the city.
- Include an attribute to store the neighboring cities.

- Include an attribute that provides the value of the straight-line distance to Bucharest.
- Create a global variable that stores all the cities. Use `defvar` to declare the global variables. Implement it this in **two different ways**: a list `*all-cities-list*` and a hash-table `*all-cities-htable*` [2]. Use the name of a city as key and the structure as value. For sake of clarity, you are *not* asked to implement a hash-table (which you probably did in CSCE310) but to use a hash-table in Lisp.
- After creating structures for all the cities, loop through them again in order to include, in the relevant attribute of a city, a reference its neighboring cities. Store these neighbors as an association list of the structure of a neighbor and the distance between the two (see LWH, page 31).

Tasks:

1. (10 points) Design, implement and test your map.
2. (5 points) Write a function `all-cities-from-list` that takes a global variable, `*all-cities-list*`, and returns a list of all *names* of cities on the map.
3. (5 points) Write a and `all-cities-from-htable` that takes a global variable, `*all-cities-htable*` and returns a list of all the *structures* of cities on the map.
4. (5 point) Write two functions `get-city-from-list` and `get-city-from-htable` that take the *name* of a city as input and return the corresponding structure (by accessing a global variable, `*all-cities-list*` and `*all-cities-htable*`, respectively).
5. (5 points) Write two functions `neighbors-using-list` and `neighbors-using-htable` that take the *name* of a city as input and return the list of structures of its direct neighbors. `neighbors-using-list` and `neighbors-using-htable` should use `get-city-from-list` and `get-city-from-htable`, respectively.
6. (10 points) Using `*all-cities-htable*`, write a function `neighbors-within-d` that takes the name of a city `my-city` and a number `distance`, then returns, for all direct neighbors within `distance` from `my-city` ($\leq$), an association list of the structures of the neighbors of `my-city` and their distance to `my-city`.
7. (10 points) Using `*all-cities-htable*`, write a function `neighbors-p` that takes the *name* of two cities `city-1` and `city-2`, and returns the distance between them if they are directly connected or nil if they are not.

Note that the global variables should always be passed as arguments to these functions (becoz it is cleaner).

[2] Check documentation on hash-tables in `http://www.cs.cmu.edu/afs/cs.cmu.edu/project/ai-repository/ai/html/cltl/cltl2.html` and `http://www.xanalys.com/software_tools/reference/HyperSpec/Body/fun_make-hash-table.html`.