Programming Assignment Report

Author
Rahul Ghosh
login: ghosh128
ID: 5476965

Prof. Nikolaos Papanikolopoulos
Course No.: CSCI 5511

Programming Assignment(100 points)
Due: 12/06/18
Submission Folder Structure:

```
root directory
├── prog_18.pdf
├── report.pdf
├── functionatom.lisp
├── replaceword.lisp
└── eightpuzzle.lisp
```

For each task, the following sections are maintained:

1. Program Implementation Description: Contains a short description of the objective of the task
2. Pseudo Code: high-level steps followed in the algorithm
3. Terminal Session Script: output of the program in some of the test cases.
4. Source Code: actual lisp source code

1 FUNCTIONATOM

1.1 Program Implementation Description

The objective of the function is to take a list of arbitrary depth and return a list with all the elements at a same depth while preserving their order. The basic idea is to recursively call the function on the sub-lists and flatten them. Also a recursive function to append two lists is implemented as higher level lisp functions are not allowed to use.

1.2 Pseudo Code

```
functionatom (list)
    if (list is empty) {return null}
    else
        {if (first_element is an atom)
            {return append(first_element, functionatom(rest_list))}
        else
            {return append(functionatom(first_element), functionatom(rest_list))}
```

1.3 Terminal Session Script

```
[1]> (load "functionatom.lisp")
;; Loading file functionatom.lisp ...
;; Loaded file functionatom.lisp
T
[2]> (functionatom '(a (b c) (((d) e f))))
(A B C D E F)
[3]> (functionatom '(a b c d e f))
(A B C D E F)
[4]> (functionatom '((a (b (c d) e)) ((f))))
(A B C D E F)
[5]> (functionatom '((a b) (c d) (e f)))
(A B C D E F)
[6]> (functionatom '())
NIL
```

1.4 Source Code

```
(defun concatlists (list1 list2)
    (if (null list1)
        list2
        (cons (car list1) (concatlists (cdr list1) list2))))

(defun functionatom (x)
    (if (null x)
        nil
        (if (listp (car x))
            (concatlists (functionatom (car x)) (functionatom (cdr x)))
            (cons (car x) (functionatom (cdr x))))))
```

2 REPLACEWORD

2.1 Program Implementation Description

The objective of the function is to take a symbol and a list, then replace all the instances of the symbol with YYYY, while preserving the structure of the list.

2.2 Pseudo Code

```
replaceword (symbol, list)
    if (list is empty) {return null}
    else
        if (first element is list)
            {return append(replaceword(symbol, first_element),
                           replaceword(symbol, rest_list))}
        else{
            if (first_element == symbol)
                {return append(YYYY, replaceword(symbol, rest_list))}
            else
                {return append(first_element, replaceword(symbol, rest_list))}
            }
```

2.3 Terminal Session Script

```
[1]> (load "replaceword.lisp")
;; Loading file replaceword.lisp ...
;; Loaded file replaceword.lisp
T
[2]> (replaceword NIL NIL)
NIL
[3]> (replaceword NIL '(I AM NIKOS))
(I AM NIKOS)
[4]> (replaceword 'NIKOS '(I AM NIKOS))
(I AM YYYY)
[5]> (replaceword 'NIKOS '(I AM (NIKOS)))
(I AM (YYYY))
```

2.4 Source Code

```
(defun replaceword (symbol X)
    (if (null x)
        nil
        (if (listp (car x))
            (cons (replaceword symbol (car x)) (replaceword symbol (cdr x)))
            (if (equal symbol (car x))
                (cons (setf (car x) 'YYYY) (replaceword symbol (cdr x)))
                (cons (car x) (replaceword symbol (cdr x)))))))
```

3 A^* Search and 8 puzzle

3.1 Program Implementation Description

The objective is to implement the A^* algorithm using the sum of Manhattan distance of all the tiles from their goal positions as the heuristic. It is then used to solve the 8-puzzle problem.

3.1.1 A^* Algorithm

At each step the algorithm picks the node according to the value of the evaluation function 'f' which is equal to the sum of two other parameters – 'g' which is the distance of the current state from the start state and 'h' is the heuristic function which estimates the cost of reaching the goal state from the current state.

At each step the algorithm picks the node/cell having the lowest 'f', and process that node/cell.

3.1.2 8-Puzzle Problem

There are eight tiles in the 8-puzzle with each tile having a number and an empty square in a 3x3 grid. A tile that is next to the empty grid square can be moved into the empty space, leaving its previous position empty in turn. The state of the puzzle is the positions of the tiles in the grid. The objective is to reach the goal state from the start state. The states are stored in the row major format with the empty space 'E' replaced by 0. For e.x goal ((1 2 3) (4 5 6) (7 8 'E')) represented as (1 2 3 4 5 6 7 8 0)
state (('E' 1 3) (4 2 5) (7 8 6)) represented as (0 1 3 4 2 5 7 8 6)

To check if a puzzle is solvable or not, we first write the start state in a linear way. Determine the number of inversions by counting the number of low tiles preceding another tile with lower number(ignore the blank space). The puzzle is solvable only when the number of inversions is even.

For e.g, Puzzle : (('E' 1 3) (4 5 2) (7 8 6))
Corresponding start state : (0 1 3 4 2 5 7 8 6)
Number of inversions : 3 (1:0, 3:1, 4:1, 5:0, 7:1, 8:1, 6:0)
(Tile Number : Count of tiles with lower number preceding it)
Since the number of inversions is odd, this puzzle is not solvable.

3.2 Pseudo Code

```
A*(start, goal)
    Initialize both openList and closedList
    Add the start state to the openList //start.f=0

    while the openList is not empty
        get currentState    //state with the least f value in openList
        remove the currentState from openList
        add the currentState to closedList
        if currentState==goal
            print number of states expanded
            return
        generate successors for currentState
        for successor in successors
            if successor not in closedList
                evaluate the successor  //calculate f, g and h
                if successor in openList
                    if the successor.g < openList state's g
                        add successor to openList
                else
                    add successor to openList
```

3.3 Terminal Session Results

```
Rahuls-MacBook-Pro:Prog_Assgn 2021rahul$ clisp eightpuzzle.lisp
Goal State (1 2 3 4 5 6 7 8 0)
Start State (0 1 3 4 2 5 7 8 6)
level 1
 generated states ((1 0 3 4 2 5 7 8 6) (4 1 3 0 2 5 7 8 6))
level 2
 generated states ((1 3 0 4 2 5 7 8 6) (0 1 3 4 2 5 7 8 6) (1 2 3 4 0 5 7 8 6))
level 3
 generated states ((1 2 3 4 5 0 7 8 6) (1 2 3 0 4 5 7 8 6) (1 2 3 4 8 5 7 0 6)
 (1 0 3 4 2 5 7 8 6))
level 4
 generated states ((1 2 3 4 0 5 7 8 6) (1 2 3 4 5 6 7 8 0) (1 2 0 4 5 3 7 8 6))
number of nodes expanded 4
Rahuls-MacBook-Pro:Prog_Assgn 2021rahul$
Rahuls-MacBook-Pro:Prog_Assgn 2021rahul$ clisp eightpuzzle.lisp
Goal State (1 2 3 4 5 6 7 8 0)
Start State (0 1 3 4 5 2 7 8 6)
The given puzzle is not solvable
```

3.4 Source Code

```
; Get coordinates in a 3x3 grid given the position index in state list
; Input: position_index
; Output: (x, y) coordinates
(defun get_coordinates (index)
    (setf (values q r) (floor index 3))
    (list q r)
)

; Get position index in state list given the coordinates in a 3x3 grid
; Input: (x, y) coordinates
; Output: position_index
(defun get_index (coord)
    (+ (* 3 (nth 0 coord)) (nth 1 coord))
)

; Get new_state from the current state by moving the blank position to the
; new_coordinates
; Input: state, current_coordinates, new_coordinates
; Output: new_state
(defun get_new_state (state coord newcoord)
    (let (new_state)
        (setf new_state (copy-list state))
        (setf (nth (get_index newcoord) new_state)
                    (nth (get_index coord) state))
        (setf (nth (get_index coord) new_state)
                    (nth (get_index newcoord) state))
        new_state
    )
)

; Check if the (x, y) coordinates lie inside the 3x3 grid
```

```
; Input: (x, y) coordinaates)
; Output: Boolean(T if valid, F if not valid)
(defun check_valid_positions (coord)
    (let ((valid nil))
        (if (>= (nth 0 coord) 0)
            (if (<= (nth 0 coord) 2)
                (if (>= (nth 1 coord) 0)
                    (if (<= (nth 1 coord) 2)
                        (setf valid t)))))
        valid
    )
)

; Get successors from the current_state by moving the blank sqaure to the four
; neighbours
; Input: current_state
; Output: successor_list
(defun get_successors (state)
    (let (newstates coord newcoord directions)
        (setf directions '((-1 0) (1 0) (0 -1) (0 1)))
        (setf coord (get_coordinates (position 0 state)))
        (do ((i 0 (+ 1 i)))
            ((> i 3))
            (setf newcoord (mapcar '+ coord (nth i directions)))
            (if (check_valid_positions newcoord)
                (setf newstates (cons
                                    (get_new_state state coord newcoord)
                                    newstates))))
        newstates
    )
)

; Get the state with the lowest f value
; Input: openeList
; Output: sorted openList in increasing order
(defun get_lowest_f_state (opened_list)
    (sort opened_list #'< :key #'second)
    opened_list
)

; Get the manhattan distance between two set of coordinates
; Input: a_coordinates, b_coordinates
; Output: distance
(defun manhattan (a_coord b_coord)
    (+ (abs (- (nth 0 a_coord) (nth 0 b_coord)))
        (abs (- (nth 1 a_coord) (nth 1 b_coord))))
)

; Get the estimate of the cost to reach from current_state
; to goal_state based on manhattan distance
; Input: current_state, goal_state
; Output: cost
(defun heuristic (state goal)
```

```
    (let ((score 0) state_pos correct_pos)
        (do ((i 1 (+ 1 i)))
            ((> i 8) score)
            (setf correct_pos (get_coordinates (position i goal)))
            (setf state_pos (get_coordinates (position i state)))
            (setf score (+ score (manhattan correct_pos state_pos)))
        )
    )
)

; Calculate the f, g and h value for a given node
; Input: current_state, goal, current_state_depth
; Output: list containing f,g and h values
(defun evaluate_node (state goal depth)
    (setf g depth)
    (setf h (heuristic state goal))
    (list state (+ g h) g h)
)

; Given a start_state determine if it is solvable or not
; Input: start_state
; Output: Boolean(T if solvable)
(defun is_solvable (state)
    (let ((count 0))
        (do ((i 0 (+ 1 i)))
            ((> i 8) count)
            (unless (= (nth i state) 0)
                (do ((j i (+ 1 j)))
                    ((> j 8))
                    (when (and (not (= (nth j state) 0))
                               (> ( nth i state ) ( nth j state )))
                        (setf count (+ 1 count))
                    )
                )
            )
        )
        (evenp count)
    )
)

; A* algorithm
(defun Astar (start goal)
    (let (opened closed node state (expanded 0))
        (setf opened (cons (evaluate_node start goal 0) opened))
        (loop while (not (null opened)) do
            (setf ret_val (get_lowest_f_state opened))
            (setf node (car ret_val))
            (setf opened (cdr ret_val))
            (setf state (car node))
            (setf closed (cons node closed))
            (if (equal state goal)
                (progn
                    (format t "number of nodes expanded ~S~%" expanded)
```

```
                (return expanded)
            )
        )
        (setf successors (get_successors state))
        (setf expanded (+ expanded 1))
        (format t "level ~S~% generated states ~S~%" expanded successors)
        (do ((i 0 (+ 1 i)))
            ((> i (- (list-length successors) 1)))
            (setf successor (nth i successors))
            (if (not (assoc successor closed))
                (progn
                    (setf info (evaluate_node successor
                                              goal
                                              (+ (nth 2 node) 1)))
                    (if (assoc successor opened)
                        (if (> (nth 2 (assoc successor opened)))
                                (+ (nth 2 node) 1)
                            (setf opened (cons
                                          (evaluate_node successor
                                                         goal
                                                         (+ (nth 2 node) 1))
                                          opened))
                        )
                        (setf opened (cons
                                      (evaluate_node successor
                                                     goal
                                                     (+ (nth 2 node) 1))
                                      opened))
                    )
                )
            )
        )
    )
)

; Driver function
(defun main ()
    (setf goal '(1 2 3 4 5 6 7 8 0))
    (setf start '(0 1 3 4 2 5 7 8 6))
    (format t "Goal State ~S~%" goal)
    (format t "Start State ~S~%" start)
    (if (is_solvable start)
        (Astar start goal)
        (format t "The given puzzle is not solvable")
    )
)

(main)
```