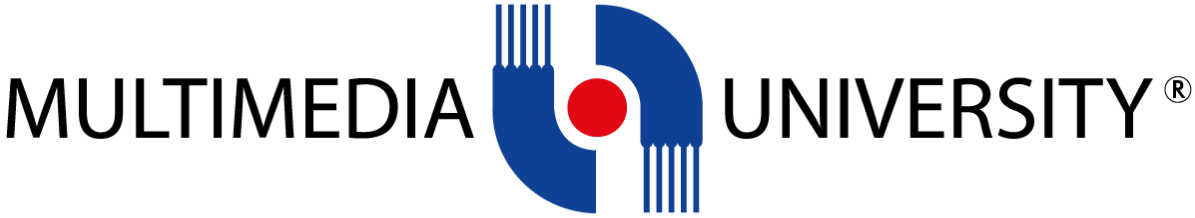

TIC3151 - Artificial Intelligence

Artificial Intelligence Assignment Documentation

Bidirectional Search Algorithm with BFS and modified DFS

Prepared by:

No	Name	Student ID	E-mail
1	Kin Zhi Qing	1142700059	zhiqingkin96@hotmail.com
2	Chong Hoe Ren	1142701180	nelsonc1006@gmail.com
3	Chai Ying Hua	1141328508	yinghuachai0@gmail.com

Table of Contents

1. Execution guides.

```
(defun main()
  (compile-file "/Users/Lim/Documents/gitRepo/ai-bidirectional-search/AI/input-test.lisp"
:load t)
  (load "/Users/Lim/Documents/gitRepo/ai-bidirectional-search/AI/input-test.fasl")

----------------------------------------------------------------
Change all the path in main.lisp file to perform compile and load
----------------------------------------------------------------
```

User is required to change the directory path contains in mail.lisp in order to run the program. Afterwards, refer the sample below to execute the program. Ignore the warning message during the compilation.

```
CG-USER(2): (main)

Select bfs or dfs algorithm (case-sensitive): bfs
You had chosen BFS  Ascending or Descending (asc/dsc)?: asc

(warning message)..........................................

 Search Tree contains:
 #2A((ARAD ZERIND ORADEA SIBIU FAGARAS RIMNIOU_VILOEA TIMISOARA LUGOJ MEHADIA DOBRETA
CRAIOVA PITESTI
      BUCHAREST GIURGIU URZIOENI HIRSOVA ETORIE VASLUI IASI NEAMT)
     (NIL ARAD SIBIU ARAD SIBIU SIBIU ARAD TIMISOARA LUGOJ MEHADIA RIMNIOU_VILOEA CRAIOVA
FAGARAS
      BUCHAREST BUCHAREST URZIOENI HIRSOVA URZIOENI VASLUI NIL))
 OPEN LIST contains  : (ETORIE IASI)
 CLOSE LIST contains : (ARAD ZERIND SIBIU TIMISOARA ORADEA FAGARAS RIMNIOU_VILOEA LUGOJ
BUCHAREST
                       CRAIOVA PITESTI MEHADIA GIURGIU URZIOENI DOBRETA HIRSOVA VASLUI)

 Solution path = IASI  VASLUI  URZIOENI  BUCHAREST  FAGARAS  SIBIU  ARAD
 Pathcost      = 6
 Total Node    = 23
NIL
CG-USER(3):
```

2. Source code with documentation and explanation

2.1 Main.cl

```
;;-------------------------------------------------------------------------------------------
;; Function: Main
;;-------------------------------------------------------------------------------------------
;; Description:
;; 1. Call (preload), (make-open-list), (make-close-list) in openclose.lisp
;; 2. Insert "Arad" into Open-list
;; 3. gettype = type of search algo (BFS/DFS), from inputtest.lisp
;; 4. If gettype = "DFS"
;;     Insert forbidden city (for1, for2) into close-list
;;
;; 5. When (check-end) = NIL (means Goal not found, Open-list not empty)
;;     Do
;;     i.   get-current, input = NOTHING, output = Node(eg. Arad) or NIL(open-list is empty)
;;     ii.  get-parent, input = Node(eg. Arad), output = x(type: number)
;;     iii. get-child, input = x(type: number, 1-20),
;;                     output = List of Child Node(eg. [Timisoara, Sibiu, Zerind])
;;     iv.  check-duplicate, input = list of node,
;;                           output = insert Child-node into Open-list,
;;                                    remove Parent-node from Open-list,
;;                                    insert Parent-node into Close-list
;;-------------------------------------------------------------------------------------------
;; Written by: Chong Hoe Ren and Zhi Qing Kin
;;-------------------------------------------------------------------------------------------


(defun main()
  (compile-file
"/Users/Lim/Documents/gitRepo/ai-bidirectional-search/AI-final/input-test.lisp" :load t)
  (load "/Users/Lim/Documents/gitRepo/ai-bidirectional-search/AI-final/input-test.fasl")
  (compile-file
"/Users/Lim/Documents/gitRepo/ai-bidirectional-search/AI-final/check-duplicate.cl" :load t)
  (load
"/Users/Lim/Documents/gitRepo/ai-bidirectional-search/AI-final/check-duplicate.fasl")
  (compile-file "/Users/Lim/Documents/gitRepo/ai-bidirectional-search/AI-final/check-end.cl"
:load t)
  ….
```

Code snippet above is created to compile and load entire program with "*compile-file*" and "*load*" commands. As the functions are distributed among different files, repetition of compile and load in debug window could be troublesome for user to execute and test the program. The approach improve compiling efforts and user can compile the whole program with only one files.

```
(setf pathcost -1)
(setf totalnode 1)
(preload)
(getinput)
(make-open-list 'Arad)
(make-close-list)
```

After source file (.lisp, .cl) are compiled and binary (.fasl) are loaded, an amount of variables are declared. Pathcost and totalnode variable serve purpose of displaying total number of nodes generated and display path cost. Close list and Open list with element ‘Arad as initial states are instantiated.

```
(when (string-equal gettype "dfs")
     (progn
     (insert-front-close for1)
     (insert-front-close for2)))

(loop
     (check-duplicate (get-child (get-parent (get-current))))
     (setf result (check-end))
     (when (not (equal result NIL))(return result)))
(format t "~% Solution path = ")
(printpath result)
(format t "~% Pathcost     = ~A" pathcost)
(format t "~% Total Node   = ~A" totalnode))
```

The program request for user choice on implement bidirectional search algorithms on traversing cities. Steps of search tree construction, path cost and total number of nodes generated will be displayed with code snippet in this file. The details of open close list, breadth-first search and depth first search implementation will be discussed in following sections.

2.2 Openclose.lisp

```
;;; ===================================
;;; File Description: CLOS for OPEN LIST
;;; -----------------------------------
;;; Written by: Chai Ying Hua
;;; ===================================

;;; ----------------------------------------
;;; Class: Open List
;;; ----------------------------------------
;;; Description: Stores new discovered nodes
;;; Accessor/Slot-value: open-list
;;; ----------------------------------------

(defclass openlist ()
  ((open-list
        :initarg :open-list
        :accessor   open-list )))


;;; ------------------------------------------------------------
;;; Class: Close List
;;; ------------------------------------------------------------
;;; Description: Stores discovered nodes for prevent duplication
;;; Accessor/Slot-value: open-list
;;; ------------------------------------------------------------

(defclass closelist ()
  ((close-list
        :initarg :close-list
        :accessor   close-list )))
```

The open list and close list in this program is implemented with Common Lisp Object System (CLOS) which is an object-oriented programming facilities of Lisp. The open list possess functionality to storing new discovered nodes and close list stored discovered nodes to prevent duplication.

The implementation method is used in order to gain control of objects with separation of concerns and modularity. In addition, it increase the readability and testability of program as we are easily debug and identify root cause of faulty in program by focus on single function in single files.

```
;;; -----------------------------------------------
;;; Declare variable for found
;;; -----------------------------------------------

(defun preload ()
  (defvar found nil)
)

;;; -----------------------------------------------
;;; Function: Instantiate open list and close list
;;; -----------------------------------------------
;;; Description:
;;;    open - instance of standard class openlist with start node
;;;    close - instance of standard class closelist
;;; -----------------------------------------------

(defun make-open-list (start)
  (setf open (make-instance 'openlist  :open-list (make-list 1 :initial-element start))))

(defun make-close-list()
  (setf close (make-instance 'closelist  :close-list '())))
```

In the following codes snippet, the program possess general function to instantiate open list and close list with the use of "*make-instance*". Lisp is known as second oldest language with functional and procedural paradigm, the function is not restricted and unable to declared with private which only can be access by specific class. Therefore, the creation of object is declared with public in order to ease the procedural calls among files.

The function preload possess variable found which will be used by find methods to search for specific elements in list.

```
;;; -----------------------------------------
;;; Functions (Setter): insert-front-open
;;; -----------------------------------------
;;; Description:
;;;  1. Insert element in front of OPEN List
;;;
;;; Input: Numerical or String
;;; Output: Return list contains inserted element
;;; -----------------------------------------

(defmethod insert-front-open (element)
  (setf (open-list open) (push element (open-list open))))

;;; -----------------------------------------
;;; Functions (Setter): insert-back-open
;;; -----------------------------------------
;;; Description:
;;;  1. Insert element in back of OPEN List
;;;
;;; Input: Numerical or String
;;; Output: Return OPEN list contains inserted element
;;; -----------------------------------------

(defmethod insert-back-open (element)
  (setf (open-list open) (reverse (cons element (reverse (open-list open))))))

;;; -----------------------------------------
;;; Functions (Getter): find-elements-open
;;; -----------------------------------------
;;; Input: NONE
;;; Output: Find element in OPEN list
;;; -----------------------------------------

(defmethod find-elements-open (element)
  (if (not (eq nil (open-list open)))
       (progn
       (setf found (find element (open-list open)))
       (if (eq nil found)
       (setf found nil)
              (setf found t))
              (format t " "))))
```

These functions provide functionality to insert element from front or back and search for specific element in the list. These function are created only for OPEN LIST. The naming and creation of function are strongly concern on codes readability and maintainability with defensive programming. This is because the naming of method give a great overview what the function trying to achieve. Moreover, it doesn't provide flexibility for other list to call this function because extra effort is required to protect data integrity of open list during the tree traversal.

```
;;; -----------------------------------------
;;; Functions (Setter): insert-front-close
;;; -----------------------------------------
;;; Description:
;;;  1. Insert element in front of CLOSE List
;;;
;;; Input: Numerical or String
;;; Output: Return CLOSE list contains inserted element
;;; -----------------------------------------

(defmethod insert-front-close (element)
  (if (eq nil (close-list close))
       (setf (close-list close) (make-list 1 :initial-element element))
       (setf (close-list close) (push element (close-list close)))))

;;; -----------------------------------------
;;; Functions (Setter): insert-back-close
;;; -----------------------------------------
;;; Description:
;;;  1. Insert element in back of OPEN List
;;;
;;; Input: Numerical or String
;;; Output: Return list contains inserted element
;;; -----------------------------------------

(defmethod insert-back-close (element)
  (if (eq nil (close-list close))
       (setf (close-list close) (make-list 1 :initial-element element))
       (setf (close-list close) (reverse (cons element (reverse (close-list close)))))))

;;; -----------------------------------------
;;; Functions (Getter): find-elements-close
;;; -----------------------------------------
;;; Input: NONE
;;; Output: Find element in OPEN list
;;; -----------------------------------------
(defmethod find-elements-close (element)
  (if (not (eq nil (close-list close)))
       (progn
       (setf found (find element (close-list close)))
       (if (eq nil found)
       (setf found nil)
       (setf found t))) (format t " ")))
```

These functions possess same purpose as mentioned. They are created only for CLOSE LIST. The container possess its own function to improve data integrity and prevent affect code readability during debugging process.

2.3 Map.lisp

```
;;; -------------------------------------------------------------
;;; Function: The map
;;; -------------------------------------------------------------
;;; Description:
;;; 1. The map is make up with two-dimensional array with 21x21
;;; 2. The map possess path cost values from nodes to nodes
;;; -------------------------------------------------------------
;;; Written by: Chong Hoe Ren
;;; -------------------------------------------------------------

(setf map (make-array '(21 21)
    :initial-contents '((NIL Arad Zerind Oradea Sibiu Fagaras Rimniou_Viloea Timisoara Lugoj
Mehadia Dobreta Craiova Pitesti Bucharest Giurgiu Urzioeni Hirsova Etorie Vaslui Iasi Neamt)
   (Arad NIL 75 NIL 140 NIL NIL 118 NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL)
   (Zerind 75 NIL 71 NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL)
   (Oradea NIL 71 NIL 151 NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL)
   (Sibiu 140 NIL 151 NIL 99 80 NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL)
   (Fagaras NIL NIL NIL 99 NIL NIL NIL NIL NIL NIL NIL 211 NIL NIL NIL NIL NIL NIL NIL)
    (Rimniou_Viloea NIL NIL NIL 80 NIL NIL NIL NIL NIL NIL 146 97 NIL NIL NIL NIL NIL NIL
NIL)
   (Timisoara 118 NIL NIL NIL NIL NIL NIL 111 NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL
NIL)
   (Lugoj NIL NIL NIL NIL NIL NIL 111 NIL 70 NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL)
   (Mehadia NIL NIL NIL NIL NIL NIL NIL 70 NIL 75 NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL)
   (Dobreta NIL NIL NIL NIL NIL NIL NIL NIL 75 NIL 120 NIL NIL NIL NIL NIL NIL NIL NIL NIL)
   (Craiova NIL NIL NIL NIL NIL 146 NIL NIL NIL 120 NIL 138 NIL NIL NIL NIL NIL NIL NIL NIL)
    (Pitesti NIL NIL NIL NIL NIL 97 NIL NIL NIL NIL 138 NIL 101 NIL NIL NIL NIL NIL NIL NIL)
   (Bucharest NIL NIL NIL NIL 211 NIL NIL NIL NIL NIL NIL 101 NIL 90 85 NIL NIL NIL NIL NIL)
   (Giurgiu NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL 90 NIL NIL NIL NIL NIL NIL NIL)
    (Urzioeni NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL 85 NIL NIL 98 NIL 142 NIL NIL)
   (Hirsova NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL 98 NIL 86 NIL NIL NIL)
    (Etorie NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL 86 NIL NIL NIL NIL)
    (Vaslui NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL 142 NIL NIL NIL 92 NIL)
    (Iasi NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL 92 NIL 87)
   (Neamt NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL 87 NIL))))
```

	Arad	Zerind	Oradea	Sibiu	Fagaras	Rimniou	Timisoara	Lugoj	Mehadia	Dobreta	Craiova	Pitesti	Bucharest	Giurgiu	Urzioeni	Hirsova	Etorie	Vaslui	Iasi	Neamt
Arad		75		140			118													
Zerind	75		71																	
Oradea		71		151																
Sibiu	140		151		99	80														
Fagaras				99									211							
Rimniou				80							146	97								
Timisoara	118							111												
Lugoj							111		70											
Mehadia								70		75										
Dobreta									75		120									
Craiova						146				120		138								
Pitesti						97					138		101							
Bucharest					211							101		90	85					
Giurgiu													90							
Urzioeni													85			98		142		
Hirsova															98		86			
Etorie																86				
Vaslui															142				92	
Iasi																		92		87
Neamt																			87	

The map is implemented with two dimensional array. The values between column and rows are path cost and indicates they are PATH.

2.4 check-duplicate.cl

```
;;-------------------------------------------------------------------------
;; Function: Check Duplicate
;;-------------------------------------------------------------------------
;; Description:
;;  1. Check if element to be inserted into Open-list is already in Close-list
;;  2. Check if element to be inserted into Open-list is already in Open-list
;;  3. Remove current(Parent) from Open, and insert it into Close-list
;;-------------------------------------------------------------------------
;; Written by: Chong Hoe Ren
;;-------------------------------------------------------------------------

(defun check-duplicate(list)
  (setf parent (get-current))
  (dolist (n list)
       (setf element (car list))
       (if (eq (find-elements-close element) NIL)
       (progn
       (setf totalnode (+ totalnode 1))
       (insert-back-open element)
       (setparent element parent)))
       (setf list (cdr list)))
  (remove-element-open parent)
  (insert-back-close parent)
  (format t "~% Search Tree contains: ~% ~a" tempmap)
  (format t "~% OPEN LIST contains  : ~a" (open-list open))
  (format t "~% CLOSE LIST contains : ~a~%" (close-list close)))
```

The function above prevent duplication insert into open list and close list if open list already possess the elements. If the discovered node is not found in open list, the nodes will be traversed and set as parent. Afterwards, the previous parent node will be insert into close list marked as visited nodes. The contents of search tree, open list and close list will be print once new node is traversed to satisfy project requirement.

2.5 Check-end.cl

```
;;-------------------------------------------------------------------------
;; Function: Check whether reach the end state (check-end)
;;-------------------------------------------------------------------------
;; Description:
;;  1. Check if Open-list is empty
;;       If Open-list is empty, return "NOT FOUND"
;;     Else Check if Goal(Iasi) is in Open-list
;;      If Goal is in Open-list, return "FOUND"
;;     Else return "NIL"
;;-------------------------------------------------------------------------
;; Written by: Chong Hoe Ren
;;-------------------------------------------------------------------------

(defun check-end()
  (if (eq (get-current) NIL)
       (return-from check-end "NOT FOUND")
       (progn
       (setf list (open-list open))
       (setf result NIL)
       (dolist (n list)
       (setf element (car list))
       (if (eq (check-goal element) T)
              (return-from check-end element))
       (setf list (cdr list)))
       (return-from check-end result))))
```

2.6 Check-goal.cl

```
;;---------------------------------------
;; Function: Check Goal
;;---------------------------------------
;; Description:
;; 1. Check if input node == Goal(Iasi)
;;      If Yes, return T
;;     Else return NIL
;;---------------------------------------
;; Input: Node (eg. Arad)
;; Output: T, NIL
;;---------------------------------------
;; Written by: Chong Hoe Ren
;;---------------------------------------

(defun check-goal(element)
  (if (equal element 'Iasi)
       (return-from check-goal T)
       (return-from check-goal NIL)))
              (return-from check-end element))
       (setf list (cdr list)))
       (return-from check-end result))))
```

2.7 get-child.cl and get-parent.cl

```
;;-------------------------------------------------------------
;; Function: Get Child Node
;;-------------------------------------------------------------
;; Description:
;; 1. For (y = 1 to 20)
;;      If map(x,y) not equal to NIL
;;      (Cons) map(0,y) into result list
;; 2. (x) = input, parent row
;; 3. (y) = child row
;; 4. (x,y) = path cost, type: number
;; 5. (0,y) = child, type: string (eg. Arad)
;;-------------------------------------------------------------
;; Input: x, type: number (1-20)
;; Output: list of child node (eg. [Timisoara, Sibiu, Zerind])
;;-------------------------------------------------------------
;; Written by: Chong Hoe Ren
;;-------------------------------------------------------------

(defun get-child(x)
    (if (equal getacc 'asc)
    (loop for y from 1 to 20
       if(not (equal (aref map x y) NIL))
        collect (aref map 0 y))
    (loop for y from 20 downto 1
        if(not (equal (aref map x y) NIL))
        collect (aref map 0 y))))
```

```
;;; ---------------------------------------
;;; Function: Get Position of Parent in map
;;; ---------------------------------------
;;; Description:
;;; 1. For (x = 1 to 20)
;;;    If map(x,0) equal to input-node
;;;    return x
;;;
;;; 2. map(x,0) = Parent row in map
;;; ---------------------------------------
;;; Input: Node (eg. Arad)
;;; Output: x type:number
;;; ---------------------------------------
;;; Written by: Chong Hoe Ren
;;; ---------------------------------------

(defun get-parent(element)
  (loop for x from 1 to 20
       if(equal (aref map x 0) element)
       do(return-from get-parent x)))
```

These function attempt to identify traversed nodes as parent and child, it loops through the map (refer section 2.3) to find connected cities and return x as parent nodes and y as child nodes.

2.9 Input-test.cl

```
;;; -------------------------------------------------------------------------
;;; Function: Obtain input from user for
;;;           - using BFS or DFS
;;;           - sort with ascending or descending order
;;; -------------------------------------------------------------------------
;;; Description:
;;;    1. User select bfs will proceed to 3.
;;;    2. User select dfs will required to insert forbidden city and proceed to 3.
;;;    3. User able to select ascending or descending order.
;;; -------------------------------------------------------------------------
;;; Written by: Zhi Qing Kin
;;; -------------------------------------------------------------------------


(defun getinput()
  (format t "Select bfs or dfs algorithm (case-sensitive): ")
  (setq gettype(read))
  (cond ((string-equal gettype "dfs")
        (progn
        (format t "forbidden city 1: ")
        (setq for1(read))
        (format t "forbidden city 2: ")
        (setq for2(read))
        (format t "forbidden city one: ~A  forbidden city two: ~A  " for1 for2)))
        ((string-equal gettype "bfs") (format t "You had chosen ~A  " gettype))
        (t (format t "wrong input")))

  (format t "Ascending or Descending (asc/dsc)?: ")
  (setq getacc(read))
  (cond ((string-equal getacc "asc") (print "The algorithm will run in ascending order. "))
        ((string-equal getacc "dsc") (print "The algorithm will run in descending order. "))
        (t (format t "wrong input"))
  ))
```

The function is declared in main.cl (section 2.1) to obtain input from user to perform interactions. The user can choose to perform following interactions in user friendly manner :

- Choose to implement bidirectional algorithms using BFS or modified DFS.
- User can select most two forbidden cities if select modified DFS.
- User can choose sort nodes in ascending or descending order.

2.10 setparent.cl and tempmap.cl

```
;;; -----------------------------------------------------------------------
;;;  Function: Set Parent
;;; -----------------------------------------------------------------------
;;;  Description:
;;;   1. The function set specific nodes as parents
;;; -----------------------------------------------------------------------
;;;  Written by: Zhi Qing Kin
;;; -----------------------------------------------------------------------


(defun setparent(element parent)
  (loop for x from  0 to 19
        if(equal (aref tempmap 0 x) element)
        do(setf(aref tempmap 1 x) parent))
)
```

```
;;; -----------------------------------------------------------------------
;;;  Function: Temporary Map
;;; -----------------------------------------------------------------------
;;;  Description:
;;;   1. The function is external container to display solution path.
;;; -----------------------------------------------------------------------
;;;  Written by: Zhi Qing Kin
;;; -----------------------------------------------------------------------


(setf tempmap (make-array '(2 20)
    :initial-contents '((Arad Zerind Oradea Sibiu Fagaras Rimniou_Viloea Timisoara Lugoj
Mehadia Dobreta Craiova Pitesti Bucharest Giurgiu Urzioeni Hirsova Etorie Vaslui Iasi Neamt)
       (NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL))))
```

2.11 printlist.cl and printpath.cl

```
;;; ---------------------------------------------------------------------------
;;;  Function: Print List
;;; ---------------------------------------------------------------------------
;;;  Description:
;;;   1. The function print the containing element in Open and Close list
;;; ---------------------------------------------------------------------------
;;;  Written by: Zhi Qing Kin
;;; ---------------------------------------------------------------------------


(defun printlist()
  (print (open-list open))
  (close-list close))
```

```
;;; ---------------------------------------------------------------------------
;;;  Function: Print Path
;;; ---------------------------------------------------------------------------
;;;  Description:
;;;   1. The function print all the elements in temporary maps as solution path
;;; ---------------------------------------------------------------------------
;;;  Written by: Zhi Qing Kin
;;; ---------------------------------------------------------------------------


(defun printpath(element)
(if(not (equal element NIL))
       (progn
       (setf pathcost (+ pathcost 1))
       (format t "~A  " element)
       (loop for x from  0 to 19
       if(equal (aref tempmap 0 x) element)
       do(printpath (aref tempmap 1 x))))))
```

These function print element contains in list. However, both serve different **purposes**. The purpose are state as follow:

- **Printlist** function print the construction and update of content in close and open lists during traversal.

- **Printpath** function print elements stores in temporary maps in order to display solution path at the end of program.

3. Expected Output

3.1 Breadth first search with Ascending

```
CG-USER(2): (main)

Select bfs or dfs algorithm (case-sensitive): bfs
You had chosen BFS  Ascending or Descending (asc/dsc)?: asc


 Search Tree contains:
 #2A((ARAD ZERIND ORADEA SIBIU FAGARAS RIMNIOU_VILOEA TIMISOARA LUGOJ MEHADIA DOBRETA
CRAIOVA PITESTI
      BUCHAREST GIURGIU URZIOENI HIRSOVA ETORIE VASLUI IASI NEAMT)
     (NIL ARAD SIBIU ARAD SIBIU SIBIU ARAD TIMISOARA LUGOJ MEHADIA RIMNIOU_VILOEA CRAIOVA
FAGARAS
      BUCHAREST BUCHAREST URZIOENI HIRSOVA URZIOENI VASLUI NIL))
 OPEN LIST contains  : (ETORIE IASI)
 CLOSE LIST contains : (ARAD ZERIND SIBIU TIMISOARA ORADEA FAGARAS RIMNIOU_VILOEA LUGOJ
BUCHAREST CRAIOVA PITESTI MEHADIA GIURGIU URZIOENI DOBRETA HIRSOVA VASLUI)

 Solution path = IASI  VASLUI  URZIOENI  BUCHAREST  FAGARAS  SIBIU  ARAD
 Pathcost      = 6
 Total Node    = 23
NIL
CG-USER(3):
```

3.2 Breadth first search with Descending

```
CG-USER(2): (main)

Select bfs or dfs algorithm (case-sensitive): bfs
You had chosen BFS  Ascending or Descending (asc/dsc)?: dsc


 Search Tree contains:
 #2A((ARAD ZERIND ORADEA SIBIU FAGARAS RIMNIOU_VILOEA TIMISOARA LUGOJ MEHADIA DOBRETA
CRAIOVA PITESTI
      BUCHAREST GIURGIU URZIOENI HIRSOVA ETORIE VASLUI IASI NEAMT)
     (NIL ARAD ZERIND ARAD SIBIU SIBIU ARAD TIMISOARA LUGOJ CRAIOVA PITESTI RIMNIOU_VILOEA
PITESTI
      BUCHAREST BUCHAREST URZIOENI NIL URZIOENI VASLUI NIL))
 OPEN LIST contains  : (HIRSOVA IASI)
 CLOSE LIST contains : (ARAD TIMISOARA SIBIU ZERIND LUGOJ RIMNIOU_VILOEA FAGARAS ORADEA
MEHADIA PITESTI CRAIOVA BUCHAREST DOBRETA URZIOENI GIURGIU VASLUI)

 Solution path = IASI  VASLUI  URZIOENI  BUCHAREST  PITESTI  RIMNIOU_VILOEA  SIBIU  ARAD
 Pathcost      = 7
 Total Node    = 22
CG-USER(3):
```

3.3 Depth first search with Ascending without forbidden cities

```
CG-USER(2): (main)

Select bfs or dfs algorithm (case-sensitive): dfs
forbidden city 1: a
forbidden city 2: b
forbidden city one: A  forbidden city two: B  Ascending or Descending (asc/dsc)?: asc


 Search Tree contains:
 #2A((ARAD ZERIND ORADEA SIBIU FAGARAS RIMNIOU_VILOEA TIMISOARA LUGOJ MEHADIA DOBRETA
CRAIOVA PITESTI
      BUCHAREST GIURGIU URZIOENI HIRSOVA ETORIE VASLUI IASI NEAMT)
     (NIL ARAD NIL ARAD BUCHAREST PITESTI ARAD TIMISOARA LUGOJ MEHADIA DOBRETA CRAIOVA
PITESTI
      BUCHAREST BUCHAREST URZIOENI NIL URZIOENI VASLUI NIL))
 OPEN LIST contains  : (ZERIND SIBIU RIMNIOU_VILOEA RIMNIOU_VILOEA FAGARAS GIURGIU HIRSOVA
IASI)
 CLOSE LIST contains : (B A ARAD TIMISOARA LUGOJ MEHADIA DOBRETA CRAIOVA PITESTI BUCHAREST
URZIOENI VASLUI)

 Solution path = IASI  VASLUI  URZIOENI  BUCHAREST  PITESTI  CRAIOVA  DOBRETA  MEHADIA
LUGOJ  TIMISOARA  ARAD
 Pathcost      = 10
 Total Node    = 18
NIL
CG-USER(5):
```

3.4 Depth first search with Descending without forbidden cities

```
CG-USER(2): (main)

Select bfs or dfs algorithm (case-sensitive): dfs
forbidden city 1: a
forbidden city 2: b
forbidden city one: A  forbidden city two: B  Ascending or Descending (asc/dsc)?: dsc


 Search Tree contains:
 #2A((ARAD ZERIND ORADEA SIBIU FAGARAS RIMNIOU_VILOEA TIMISOARA LUGOJ MEHADIA DOBRETA
CRAIOVA PITESTI
      BUCHAREST GIURGIU URZIOENI HIRSOVA ETORIE VASLUI IASI NEAMT)
     (NIL ARAD ZERIND ORADEA SIBIU PITESTI LUGOJ MEHADIA DOBRETA CRAIOVA RIMNIOU_VILOEA
BUCHAREST
      FAGARAS BUCHAREST BUCHAREST URZIOENI HIRSOVA URZIOENI VASLUI NIL))
 OPEN LIST contains  : (IASI)
 CLOSE LIST contains : (B A ARAD ZERIND ORADEA SIBIU FAGARAS BUCHAREST PITESTI
RIMNIOU_VILOEA CRAIOVA DOBRETA MEHADIA LUGOJ TIMISOARA GIURGIU URZIOENI HIRSOVA ETORIE
VASLUI)

 Solution path = IASI  VASLUI  URZIOENI  BUCHAREST  FAGARAS  SIBIU  ORADEA  ZERIND  ARAD
 Pathcost      = 8
 Total Node    = 23
NIL
CG-USER(6):
```