Artificial Intelligence Assignment Report

TES 3111 / TIC 3151 Artificial Intelligence

Trimerter 1 2017/2018

Madam Amy Lim

Names	ID
Ng Chen Hon	1132702936
Lim Hern Rieh	1142700558
Ng Chin Ann	1142701684

Codes with Comments (Printout of LISP program with documentation)

```
; A function that get city than return the cities they ar
e connected to
(DEFUN GET_ADJACENT_CITIES (CITY) (
    COND
    ((STRING-EQUAL CITY "ZERIND")
     `("ORADEA" "ARAD")
    )
```

```
    ((STRING-EQUAL CITY "ORADEA")
     `("ZERIND" "SIBIU")
    )
    ((STRING-EQUAL CITY "ARAD")
     `("ZERIND" "SIBIU" "TIMISOARA")
    )
    ((STRING-EQUAL CITY "SIBIU")
     `("ORADEA" "ARAD" "FAGARAS" "RIMNICU VILCEA")
    )
    ((STRING-EQUAL CITY "FAGARAS")
     `("SIBIU" "BUCHAREST")
    )
    ((STRING-EQUAL CITY "RIMNICU VILCEA")
     `("SIBIU" "CRAIOVA" "PITESTI")
    )
    ((STRING-EQUAL CITY "TIMISOARA")
     `("ARAD" "LUGOJ")
    )
    ((STRING-EQUAL CITY "LUGOJ")
     `("TIMISOARA" "MEHADIA")
    )
    ((STRING-EQUAL CITY "MEHADIA")
     `("DROBETA" "LUGOJ")
    )
    ((STRING-EQUAL CITY "DROBETA")
     `("CRAIOVA" "MEHADIA")
    )
    ((STRING-EQUAL CITY "CRAIOVA")
```

```
      `("DROBETA" "PITESTI" "RIMNICU VILCEA")
     )
     ((STRING-EQUAL CITY "PITESTI")
      `("CRAIOVA" "RIMNICU VILCEA" "BUCHAREST")
     )
     ((STRING-EQUAL CITY "BUCHAREST")
      `("FAGARAS" "PITESTI" "URZICENI" "GIURGIU")
     )
     ((STRING-EQUAL CITY "GIURGIU")
      `("BUCHAREST")
     )
     ((STRING-EQUAL CITY "URZICENI")
      `("BUCHAREST" "HIRSOVA" "VASLUI")
     )
     ((STRING-EQUAL CITY "HIRSOVA")
      `("URZICENI" "EFORIE")
     )
     ((STRING-EQUAL CITY "EFORIE")
      `("HIRSOVA")
     )
     ((STRING-EQUAL CITY "VASLUI")
      `("IASI" "URZICENI")
     )
     ((STRING-EQUAL CITY "NEAMT")
      `("IASI")
     )
     ((STRING-EQUAL CITY "IASI")
      `("VASLUI" "NEAMT")
```

```
    )
))

; orders the list of cities according to alphabet ascendi
ng or descending
(DEFUN ORDER (LIST ORDER) (COND
    ((STRING-EQUAL ORDER "ASCENDING")
     (SORT LIST #'STRING-LESSP))
    ((STRING-EQUAL ORDER "DESCENDING")
     (SORT LIST #'STRING-GREATERP))
))

; a class for BFS states
(DEFCLASS BFS () (
    (CLOSED :ACCESSOR GET-CLOSED :WRITER SET-CLOSED :INIT
FORM (LIST))
    (OPEN :ACCESSOR GET-OPEN :WRITER SET-OPEN :INITFORM (
LIST))
    (MAP :ACCESSOR GET-MAP :WRITER SET-MAP :INITFORM (LIS
T))
    (CURRENT_CHECK :ACCESSOR GET-CURRENT_CHECK :WRITER SE
T-CURRENT_CHECK :INITFORM "")
    (START :ACCESSOR GET-START :WRITER SET-START :INITFOR
M "")
    (GOAL :ACCESSOR GET-GOAL :WRITER SET-GOAL :INITFORM "
")
    (TEMP_MAP :ACCESSOR GET-TEMP_MAP :WRITER SET-TEMP_MAP
 :INITFORM (LIST))
```

```
    (NAME :ACCESSOR GET-NAME :WRITER SET-NAME :INITFORM "
" :INITARG :INIT-NAME)
    (ORDER :ACCESSOR GET-ORDER :WRITER SET-ORDER :INITFOR
M "")
))

; resets a BFS object
(DEFMETHOD RESET ((OBJECT BFS)) (
    PROGN (SET-CLOSED (LIST) OBJECT)
    (SET-OPEN (LIST) OBJECT)
    (SET-MAP (LIST) OBJECT)
    (SET-CURRENT_CHECK "" OBJECT)
    (SET-START  "" OBJECT)
    (SET-GOAL "" OBJECT)
    (SET-TEMP_MAP (LIST) OBJECT)
))

; combines open and closed list of a BFS object then retu
rn it
(DEFMETHOD GET_NODES_IN_OPEN_CLOSED ((OBJECT BFS)) (
    APPEND  (GET-OPEN OBJECT) (GET-CLOSED OBJECT)
))

; check if the CITY_TO_CHECK which is a city is in open o
r closed lists in BFS
(DEFMETHOD FIND_IF_IN_OPEN_OR_CLOSED ((OBJECT BFS) CITY_T
O_CHECK) (
    COND
```

```
    ((GET_NODES_IN_OPEN_CLOSED OBJECT) (
        DOLIST (N (GET_NODES_IN_OPEN_CLOSED OBJECT)) (
            IF (STRING-EQUAL N CITY_TO_CHECK) (RETURN-FRO
M FIND_IF_IN_OPEN_OR_CLOSED T)
        )
    ))
    (T NIL)
))

; add parent-children pair into the map, to represent the
 connected of parent to children, a tree structure in a l
ists in BFS
; so its like (parent, (children ....))
(DEFMETHOD ADD_TO_MAP ((OBJECT BFS)) (
    SET-MAP (APPEND (GET-MAP OBJECT) (LIST (CONS (GET-CUR
RENT_CHECK OBJECT) (LIST (GET-TEMP_MAP OBJECT))))) OBJECT
))

; add the current checking city to closed list in BFS
(DEFMETHOD ADD_TO_CLOSED ((OBJECT BFS)) (
    SET-CLOSED (APPEND (GET-CLOSED OBJECT) (LIST (GET-CUR
RENT_CHECK OBJECT))) OBJECT
))

; add a city to open list in BFS
(DEFMETHOD PUSH_BACK ((OBJECT BFS) CITY) (
    SET-OPEN (APPEND (GET-OPEN OBJECT) (LIST CITY)) OBJEC
T
```

```
))

; remove a city from the queue of open list of BFS
; use it as current checking city
(DEFMETHOD POP_FRONT ((OBJECT BFS)) (
    IF (> (LENGTH (GET-OPEN OBJECT)) 0) (
                        PROGN
            (SET-CURRENT_CHECK (CAR (GET-OPEN OBJECT)) OB
JECT)
                        (SET-OPEN (CDR (GET-OPEN OBJECT))
 OBJECT)
                        ()
            (RETURN-FROM POP_FRONT T)
    ) NIL
))

; this is a function to add current children connected to
 the current checking city (parent)
(DEFMETHOD INSERT_TEMP_MAP ((OBJECT BFS) CITY) (
    SET-TEMP_MAP (APPEND (GET-TEMP_MAP OBJECT) (LIST CITY
)) OBJECT
))

; this method will check the current checking city (paren
t), and add its child not included in open or closed list
 to open list and into the children list of this parent
(DEFMETHOD DISCOVER ((OBJECT BFS)) (
    PROGN
```

```
    (DOLIST (N (ORDER (GET_ADJACENT_CITIES (GET-CURRENT_C
HECK OBJECT)) (GET-ORDER OBJECT))) (
        IF(NOT (FIND_IF_IN_OPEN_OR_CLOSED OBJECT N)) (
            PROGN
                (INSERT_TEMP_MAP OBJECT N)
                (PUSH_BACK OBJECT N)
        )
    ))
    (ADD_TO_MAP OBJECT)
    (ADD_TO_CLOSED OBJECT)
    (SET-TEMP_MAP (LIST) OBJECT)
))

; calls an iteration on a city from open list, return tru
e if got city to check, return nil if there is no city le
ft to check
(DEFMETHOD ITERATE ((OBJECT BFS)) (
    IF (POP_FRONT OBJECT) (
        PROGN
            (DISCOVER OBJECT)
            (FORMAT T "~A Closed List : ~S ~%" (GET-NAME
OBJECT) (GET-CLOSED OBJECT))
            (FORMAT T "~A Open List : ~S ~%" (GET-NAME OB
JECT) (GET-OPEN OBJECT))
            (FORMAT T "~A Parent-Child Map List : ~S ~%"
(GET-NAME OBJECT) (GET-MAP OBJECT))
            (RETURN-FROM ITERATE T)
    ) NIL
```

```
))

; init the states of BFS
(DEFMETHOD INIT ((OBJECT BFS)) (
    PROGN
    (PUSH_BACK OBJECT (GET-START OBJECT))
    (POP_FRONT OBJECT)
    (DISCOVER OBJECT)
))

; check both BFS objects if their open and closed lists h
as one common city
(DEFMETHOD FIND_SAME_CITY_IN_BOTH_BFS((BEGIN BFS) (END BF
S)) (
    PROGN
    (DOLIST (N (GET_NODES_IN_OPEN_CLOSED BEGIN)) (
        IF(FIND_IF_IN_OPEN_OR_CLOSED END N) (
            RETURN-FROM FIND_SAME_CITY_IN_BOTH_BFS N
        )
    ))
    NIL
))

; find the parent city by checking its child through the
parent-children pair
(DEFMETHOD FIND_PARENT_WITH_CHILD ((OBJECT BFS) CITY) (
        PROGN
        (DOLIST (N (GET-MAP OBJECT)) (
```

```
                DOLIST(M (NTH 1 N)) (
                        IF(STRING-EQUAL M CITY) (
                RETURN-FROM FIND_PARENT_WITH_CHILD (CAR N
)
                        )
        )
    ))
    NIL
))

; trace from the intersected city back to the root and fo
rm a route from root to intersected city
(DEFMETHOD TRACE_PATH_FROM_CITY_BACK_TO_ROOT((OBJECT BFS)
 CITY) (
    LET ((ROUTE (LIST)) (TEMP CITY))
    (LOOP
        (SETQ TEMP (FIND_PARENT_WITH_CHILD OBJECT TEMP))
        (SETQ ROUTE (CONS TEMP ROUTE))
        (WHEN (STRING-EQUAL TEMP (GET-START OBJECT)) (RET
URN-FROM TRACE_PATH_FROM_CITY_BACK_TO_ROOT ROUTE))
    )
))

; get route from both BFS objects from starting city to i
ntersected city to goal city
(DEFMETHOD FIND_PATH_FROM_BOTH_BFS((BEGIN BFS) (END BFS)
CITY) (
        LET ((ROUTE_START (LIST)) (ROUTE_END (LIST)) (ROU
```

```
TE_COMPLETE (LIST)))
        (SETQ ROUTE_START (TRACE_PATH_FROM_CITY_BACK_TO_R
OOT BEGIN CITY))
        (SETQ ROUTE_END (TRACE_PATH_FROM_CITY_BACK_TO_ROO
T END CITY))
        (SETQ ROUTE_END (REVERSE ROUTE_END))
        (SETQ ROUTE_COMPLETE (APPEND ROUTE_START (LIST CI
TY) ROUTE_END))
    (RETURN-FROM FIND_PATH_FROM_BOTH_BFS ROUTE_COMPLETE)
))

; the culmination of bidirectional search with BFS
(DEFMETHOD BDS_BFS(START GOAL ORDER (OBJ1 BFS) (OBJ2 BFS)
) (
    LET ((FOUND_CITY "") (POP_START T) (POP_END T))
    (PROGN
        (SET-START START OBJ1)
        (SET-GOAL GOAL OBJ1)
        (SET-ORDER ORDER OBJ1)
        (INIT OBJ1)

        (SET-START GOAL OBJ2)
        (SET-GOAL START OBJ2)
        (SET-ORDER ORDER OBJ2)
        (INIT OBJ2)

        (LOOP
            (SETQ POP_START (ITERATE OBJ1))
```

```
            (SETQ POP_END (ITERATE OBJ2))

            (IF(AND (NOT POP_START) (NOT POP_END)) (
                RETURN-FROM BDS_BFS NIL
            ))

            (SETQ FOUND_CITY (FIND_SAME_CITY_IN_BOTH_BFS
OBJ1 OBJ2))

            (IF (NOT (NULL FOUND_CITY)) (
                PROGN
                (FORMAT T "Number of Nodes Generated : ~D
  ~%" (+ (LENGTH (GET-CLOSED OBJ1)) (LENGTH (GET-OPEN OBJ
1)) (LENGTH (GET-CLOSED OBJ2)) (LENGTH (GET-OPEN OBJ2))))
                (RETURN-FROM BDS_BFS (FIND_PATH_FROM_BOTH
_BFS OBJ1 OBJ2 FOUND_CITY))
            ))
        )
        (RETURN-FROM BDS_BFS NIL)
    )
))

(defun extract-path-from-goal (goal start)
  (setf path-goal nil)
  (setf path-goal (cons goal path-goal))
  (loop
    (setf goal (intern goal))
    (setf goal (get goal 'previous-goal))
```

```
    (setf path-goal (cons goal path-goal))
    (if (string-equal goal start) (return))
  )
)


(defun extract-path-from-start (goal start)
  (setf path-start nil)
  (setf path-start (cons goal path-start))
  (loop
    (setf goal (intern goal))
    (setf goal (get goal 'previous-start))
    (setf path-start (cons goal path-start))
    (if (string-equal goal start) (return))
  )
)

(defun bds-dfs (start goal order forbidden-city)
  (setf path-found 'false)
  (setf no-solution-start 'false) ; SET NO SOLUTION FOUND
 FOR START AS FALSE
  (setf no-solution-goal 'false) ; SET NO SOLUTION FOUND
FOR GOAL AS FALSE
  (setf n 0) ; COUNTER TO KEEP TRACK OF NUMBER OF ITERATI
ONS

  (setf open-start (list start)) ; INITIALIZE START NODE
  (setf closed-start nil)        ; INITIALIZE CLOSED LIST
```

```
 FOR START

  (setf open-goal (list goal)) ; INITIALIZE GOAL NODE
  (setf closed-goal nil)       ; INITIALIZE CLOSED LIST F
OR GOAL

  (setf output-start-search-tree nil) ; INITIALIZE LIST T
O OUTPUT SEARCH TREE FROM START
  (setf output-goal-search-tree nil)  ; INITIALIZE LIST T
O OUTPUT SEARCH TREE FROM GOAL

  ; PRINTING OUT THE INITIAL INIFORMATION BEFORE BIDIRECT
IONAL SEARCH USING DEPTH-FIRST-SEARCH
  (format t "~%Closed List for Start after Iteration ~d:
~a" n closed-start)
  (format t "~%Open List for Start after Iteration ~d: ~a
" n open-start)
  (format t "~%Closed List for Goal after Iteration ~d: ~
a" n closed-goal)
  (format t "~%Open List for Goal after Iteration ~d: ~a"
 n open-goal)
  (format t "~%Start Search Tree at Iteration ~d: ~a" n o
utput-start-search-tree)
  (format t "~%Goal Search Tree at Iteration ~d: ~a" n ou
tput-goal-search-tree)

  (loop
    ;;; DEPTH FIRST SEARCH FROM START
```

```
    ; IF OPEN LIST FOR START IS NULL AND STILL NO SOLUTIO
N IS FOUND, RETURN NO SOLUTION
    (if (null open-start) (progn (setf no-solution-start
'true)(return "NO SOLUTION FOUND")))

    ; NEXT CITY TO EXPLORE FROM START
    (setf to-explore-from-start (car open-start))

    ; CITIES THAT ARE WAITING TO BE EXPLORED (THE FRINGE)
    (setf open-start (cdr open-start))

    (dolist (x forbidden-city)
    ; IF THE CITY TO BE EXPLORED IS IN FORBIDDEN LIST
      (if (string-equal to-explore-from-start x)
        ; SKIP THE CITY AND PROCEEED TO NEXT AVAILABLE CI
TY
        (multiple-value-setq (to-explore-from-start open-
start) (values (car open-start) (cdr open-start)))
      )
    )

    ; STORE EXPLORED CITIES INTO CLOSED LIST
    (setf closed-start (cons to-explore-from-start closed
-start))

    ; GET THE SUCCESSOR CITIES OF CURRENT EXPLORED CITY
    (setf descendants-start (ORDER (GET_ADJACENT_CITIES t
```

```
o-explore-from-start) order))

    ; CHECK AND FILTER CITIES WHICH ARE EXPLORED
    (setf descendants-start (set-difference descendants-s
tart closed-start :test #'string=))
    (setf descendants-start (reverse descendants-start))

    ; APPEND ALL THE SUCCESSOR CITIES OF CURRENT EXPLORED
 CITY IN OPEN LIST OF START AND DEDUPLICATE
    (setf open-start (append descendants-start (reverse (
set-difference open-start descendants-start :test #'strin
g=))))

    ; ASSIGN EACH NODES TO BE EXPLORED WITH THEIR ANTECED
ANT TO KEEP TRACK IF SOLUTION PATH FOUND
    (dolist (x descendants-start)
      (setf (get (intern x) 'previous-start) to-explore-f
rom-start)
    )



    ;;; DEPTH FIRST SEARCH FROM GOAL

    ; IF OPEN LIST FOR GOAL IS NULL AND STILL NO SOLUTION
 IS FOUND, RETURN NO SOLUTION
    (if (null open-goal) (progn (setf no-solution-goal 't
rue)(return nil)))
```

```
    ; NEXT CITY TO EXPLORE FROM GOAL
    (setf to-explore-from-goal (car open-goal))

    ; CITIES THAT ARE WAITING TO BE EXPLORED (THE FRINGE)
    (setf open-goal (cdr open-goal))

    (dolist (x forbidden-city)
    ; IF THE CITY TO BE EXPLORED IS IN FORBIDDEN LIST
     (if (string-equal to-explore-from-goal x)
      ; SKIP THE CITY AND PROCEEED TO NEXT AVAILABLE CITY
      (multiple-value-setq (to-explore-from-goal open-goa
l) (values (car open-goal) (cdr open-goal)))
     )
    )

    ; STORE EXPLORED CITIES INTO CLOSED LIST
    (setf closed-goal (cons to-explore-from-goal closed-g
oal))

    ; GET THE SUCCESSOR CITIES OF CURRENT EXPLORED CITY
    (setf descendants-goal (ORDER (GET_ADJACENT_CITIES to
-explore-from-goal) order))

    ; CHECK AND FILTER CITIES WHICH ARE EXPLORED
    (setf descendants-goal (set-difference descendants-go
al closed-goal :test #'string=))
    (setf descendants-goal (reverse descendants-goal))
```

```
    ; APPEND ALL THE SUCCESSOR CITIES OF CURRENT EXPLORED
 CITY IN OPEN LIST OF GOAL AND DEDUPLICATE
    (setf open-goal (append descendants-goal (reverse (se
t-difference open-goal descendants-goal :test #'string=))
))

    ; ASSIGN EACH NODES TO BE EXPLORED WITH THEIR ANTECED
ANT TO KEEP TRACK IF SOLUTION PATH FOUND
    (dolist (x descendants-goal)
      (setf (get (intern x) 'previous-goal) to-explore-fr
om-goal)
    )

    ; TOO KEEP TRACK OF NUMBER OF ITERATIONS
    (incf n)

    (format t "~%Closed List for Start after Iteration ~d
: ~a" n closed-start) ; LIST STORING EXPLORED CITIES FROM
 START IN CURRENT ITERATION
    (format t "~%Open List for Start after Iteration ~d:
~a" n open-start)     ; LIST STORING CITIES TO BE EXPLORE
D FROM START (THE FRINGE) IN CURRENT ITERATION
    (format t "~%Closed List for Goal after Iteration ~d:
 ~a" n closed-goal)   ; LIST STORING EXPLORED CITIES FROM
 GOAL IN CURRENT ITERATION
    (format t "~%Open List for Goal after Iteration ~d: ~
a" n open-goal)       ; LIST STORING CITIES TO BE EXPLORE
```

```
D FROM GOAL (THE FRINGE) IN CURRENT ITERATION

    ; CONSTRUCT THE SEARCH TREE FROM START
    (setf start-search-tree (cons (cons to-explore-from-s
tart (cons descendants-start nil)) nil))
    (setf output-start-search-tree (append start-search-t
ree output-start-search-tree))
    (format t "~%Start Search Tree at Iteration ~d: ~a" n
 output-start-search-tree)

    ; CONSTRUCT THE SEARCH TREE FROM GOAL
    (setf goal-search-tree (cons (cons to-explore-from-go
al (cons descendants-goal nil)) nil))
    (setf output-goal-search-tree (append goal-search-tre
e output-goal-search-tree))
    (format t "~%Goal Search Tree at Iteration ~d: ~a" n
output-goal-search-tree)

    ; IF SEARCH FROM START OR SEARCH FROM GOAL MEET EACH
OTHER IN A NODE, GET THE INTERSECTED NODE
    (dolist (city closed-goal)
      (if (string-equal city (find city closed-start :tes
t #'string=))
        (progn (setf to-explore-from-goal city) (setf to-
explore-from-start city) (setf path-found 'true))
      )
    )
```

```
    (format t "~%")

    ; CONSTRUCT THE SOLUTION PATH FROM START AND GOAL
    (if (equal path-found 'true)
      (progn (extract-path-from-goal to-explore-from-goal
 goal)
        (extract-path-from-start to-explore-from-start st
art)
        (return "path found")
      )
    )
  )
)


(DEFUN MAINMENU()
    (SETF BFS_1 (MAKE-INSTANCE `BFS))
    (SETF BFS_2 (MAKE-INSTANCE `BFS))

    (FORMAT T "ENTER ORIGIN CITY NAME:~%") ;; GET NAME OF
 ORIGIN CITY
    (SETQ ORIGIN_CITY (READ-LINE)) ;; EXAMPLE INPUT: ARAD

    (FORMAT T "ENTER DESTINATION CITY NAME:~%") ;; GET NA
ME OF DESTINATION CITY
    (SETQ DESTINATION_CITY (READ-LINE)) ;; EXAMPLE INPUT:
 NEAMT
```

```
    (FORMAT T "CHOOSE SORT ORDER: ~%1. ASCENDING ~%2. DES
CENDING~%ANSWER: ")  ;; CHOOSE SORT ORDER: ASCENDING OR D
ESCENDING
    (SETQ SORTORDERCHOOSE (READ-LINE))
    (IF (STRING-EQUAL SORTORDERCHOOSE "1")
        (PROGN
            (SETQ SORTORDERCHOOSE "ASCENDING")
        )
        (PROGN
            (IF (STRING-EQUAL SORTORDERCHOOSE "2")
                (PROGN
                    (SETQ SORTORDERCHOOSE "DESCENDING")
                )
                (PROGN
                    (FORMAT T "INVALID INPUT. PLEASE TRY
AGAIN~%")
                    (MAINMENU)
                )
            )
        )
    )
    (FORMAT T "CHOOSE SEARCH ALGORITHM: ~%1. BREADTH-FIRS
T SEARCH (BFS) ~%2. DEPTH-FIRST SEARCH (DFS)~%ANSWER: ")
;; CHOOSE SEARCH ALGORITHM: BFS OR DSF
    (SETQ ALGORITHMTYPE (READ-LINE))
    (IF (STRING-EQUAL ALGORITHMTYPE "1")
        (PROGN
            (SETF BFS_1 (MAKE-INSTANCE `BFS :INIT-NAME "H
```

```
EAD"))
            (SETF BFS_2 (MAKE-INSTANCE `BFS :INIT-NAME "T
AIL"))
            (SETF SOLUTION_PATH (BDS_BFS ORIGIN_CITY DEST
INATION_CITY SORTORDERCHOOSE BFS_1 BFS_2))
            (FORMAT T "Solution Path : ~S  ~%" SOLUTION_P
ATH)
            (IF (NOT (NULL SOLUTION_PATH)) (FORMAT T "Pat
h Cost : ~D" (- (LENGTH SOLUTION_PATH) 1)))
         )
         (PROGN
            (IF (STRING-EQUAL ALGORITHMTYPE "2")
                (PROGN
                    (setq FORBIDDEN_CITY_LIST (list))
                    (FORMAT T "ENTER NAME OF FORBIDDEN CI
TY (ENTER 'SKIP' TO SKIP THIS):~%") ;; GET FIRST FORBIDDE
N CITY
                    (SETQ FORBIDDEN_CITY_1 (READ-LINE))
                    (IF (STRING-EQUAL FORBIDDEN_CITY_1 "S
KIP")
                        (PROGN )
                        (PROGN
                            (setq FORBIDDEN_CITY_LIST (li
st FORBIDDEN_CITY_1))
                            (FORMAT T "ENTER ANOTHER NAME
 OF FORBIDDEN CITY (ENTER 'SKIP' TO SKIP THIS):~%") ;; GE
T SECOND FORBIDDEN CITY
                            (SETQ FORBIDDEN_CITY_2 (READ-
```

```
LINE))
                                (IF (STRING-EQUAL FORBIDDEN_C
ITY_2 "SKIP")
                                    (PROGN )
                                    (PROGN
                                        (setq FORBIDDEN_CITY_
LIST (list FORBIDDEN_CITY_1 FORBIDDEN_CITY_2))
                                    )
                                )
                            )
                        )


                        (bds-dfs ORIGIN_CITY DESTINATION_CITY
 SORTORDERCHOOSE FORBIDDEN_CITY_LIST)

                        (setf start-nodes-generated (+ (list-
length open-start) (list-length closed-start)))
                        (setf goal-nodes-generated (+ (list-l
ength open-goal) (list-length closed-goal)))
                        (setf total-nodes-generated (+ start-
nodes-generated goal-nodes-generated))
                        (format t "~%Total Number of Nodes Ge
nerated: ~d" total-nodes-generated)

                        ;ADD IF NO SOLUTION FOUND
                        (if (or (equal no-solution-start 'tru
e) (equal no-solution-goal 'true)) (format t "~%Cannot Fi
```

```
nd Solution")
                        (progn
                            (setf solution-path (remove-d
uplicates (append path-start (reverse path-goal)) :test #
'string=))
                            (format t "~%Solution Path: ~
a" solution-path)
                (setf path-cost (- (list-length solution-pa
th) 1))
                        (format t "~%Path Cost: ~d" path-
cost)
                        )
                    )


                )
                (PROGN
                    (FORMAT T "TRY AGAIN~%")
                    (MAINMENU)
                )
            )
        )
    )
)

(MAINMENU)
```

Sample Outputs (Output from executing the program)

Ascending BFS

1. Sample Inputs

> *ARAD*
>
> *IASI*
>
> *1*
>
> *1*

2. Sample Outputs

```
HEAD Closed List : ("ARAD" "SIBIU")
HEAD Open List : ("TIMISOARA" "ZERIND" "FAGARAS" "ORADEA"
 "RIMNICU VILCEA")
HEAD Parent-Child Map List : (("ARAD" ("SIBIU" "TIMISOARA
" "ZERIND"))
                              ("SIBIU" ("FAGARAS" "ORADEA
" "RIMNICU VILCEA")))
TAIL Closed List : ("IASI" "NEAMT")
TAIL Open List : ("VASLUI")
TAIL Parent-Child Map List : (("IASI" ("NEAMT" "VASLUI"))
 ("NEAMT" NIL))
HEAD Closed List : ("ARAD" "SIBIU" "TIMISOARA")
HEAD Open List : ("ZERIND" "FAGARAS" "ORADEA" "RIMNICU VI
```

```
LCEA" "LUGOJ")
HEAD Parent-Child Map List : (("ARAD" ("SIBIU" "TIMISOARA
" "ZERIND"))
                             ("SIBIU" ("FAGARAS" "ORADEA
" "RIMNICU VILCEA"))
                             ("TIMISOARA" ("LUGOJ")))
TAIL Closed List : ("IASI" "NEAMT" "VASLUI")
TAIL Open List : ("URZICENI")
TAIL Parent-Child Map List : (("IASI" ("NEAMT" "VASLUI"))
 ("NEAMT" NIL) ("VASLUI" ("URZICENI")))
HEAD Closed List : ("ARAD" "SIBIU" "TIMISOARA" "ZERIND")
HEAD Open List : ("FAGARAS" "ORADEA" "RIMNICU VILCEA" "LU
GOJ")
HEAD Parent-Child Map List : (("ARAD" ("SIBIU" "TIMISOARA
" "ZERIND"))
                             ("SIBIU" ("FAGARAS" "ORADEA
" "RIMNICU VILCEA"))
                             ("TIMISOARA" ("LUGOJ")) ("Z
ERIND" NIL))
TAIL Closed List : ("IASI" "NEAMT" "VASLUI" "URZICENI")
TAIL Open List : ("BUCHAREST" "HIRSOVA")
TAIL Parent-Child Map List : (("IASI" ("NEAMT" "VASLUI"))
 ("NEAMT" NIL) ("VASLUI" ("URZICENI"))
                             ("URZICENI" ("BUCHAREST" "H
IRSOVA")))
HEAD Closed List : ("ARAD" "SIBIU" "TIMISOARA" "ZERIND" "
FAGARAS")
HEAD Open List : ("ORADEA" "RIMNICU VILCEA" "LUGOJ" "BUCH
```

```
AREST")
HEAD Parent-Child Map List : (("ARAD" ("SIBIU" "TIMISOARA
" "ZERIND"))
                              ("SIBIU" ("FAGARAS" "ORADEA
" "RIMNICU VILCEA"))
                              ("TIMISOARA" ("LUGOJ")) ("Z
ERIND" NIL) ("FAGARAS" ("BUCHAREST")))
TAIL Closed List : ("IASI" "NEAMT" "VASLUI" "URZICENI" "B
UCHAREST")
TAIL Open List : ("HIRSOVA" "FAGARAS" "GIURGIU" "PITESTI"
)
TAIL Parent-Child Map List : (("IASI" ("NEAMT" "VASLUI"))
 ("NEAMT" NIL) ("VASLUI" ("URZICENI"))
                              ("URZICENI" ("BUCHAREST" "H
IRSOVA"))
                              ("BUCHAREST" ("FAGARAS" "GI
URGIU" "PITESTI")))
Number of Nodes Generated : 18
Solution Path : ("ARAD" "SIBIU" "FAGARAS" "BUCHAREST" "UR
ZICENI" "VASLUI" "IASI")
Path Cost : 6
```

Descending BFS

1. Sample Inputs

> *ARAD*

> *IASI*
>
> *2*
>
> *1*

2. Sample Outpus

```
HEAD Closed List : ("ARAD" "ZERIND")
HEAD Open List : ("TIMISOARA" "SIBIU" "ORADEA")
HEAD Parent-Child Map List : (("ARAD" ("ZERIND" "TIMISOAR
A" "SIBIU")) ("ZERIND" ("ORADEA")))
TAIL Closed List : ("IASI" "VASLUI")
TAIL Open List : ("NEAMT" "URZICENI")
TAIL Parent-Child Map List : (("IASI" ("VASLUI" "NEAMT"))
 ("VASLUI" ("URZICENI")))
HEAD Closed List : ("ARAD" "ZERIND" "TIMISOARA")
HEAD Open List : ("SIBIU" "ORADEA" "LUGOJ")
HEAD Parent-Child Map List : (("ARAD" ("ZERIND" "TIMISOAR
A" "SIBIU")) ("ZERIND" ("ORADEA"))
                              ("TIMISOARA" ("LUGOJ")))
TAIL Closed List : ("IASI" "VASLUI" "NEAMT")
TAIL Open List : ("URZICENI")
TAIL Parent-Child Map List : (("IASI" ("VASLUI" "NEAMT"))
 ("VASLUI" ("URZICENI")) ("NEAMT" NIL))
HEAD Closed List : ("ARAD" "ZERIND" "TIMISOARA" "SIBIU")
HEAD Open List : ("ORADEA" "LUGOJ" "RIMNICU VILCEA" "FAGA
RAS")
HEAD Parent-Child Map List : (("ARAD" ("ZERIND" "TIMISOAR
A" "SIBIU")) ("ZERIND" ("ORADEA"))
```

```
                                ("TIMISOARA" ("LUGOJ")) ("S
IBIU" ("RIMNICU VILCEA" "FAGARAS")))
TAIL Closed List : ("IASI" "VASLUI" "NEAMT" "URZICENI")
TAIL Open List : ("HIRSOVA" "BUCHAREST")
TAIL Parent-Child Map List : (("IASI" ("VASLUI" "NEAMT"))
 ("VASLUI" ("URZICENI")) ("NEAMT" NIL)
                                ("URZICENI" ("HIRSOVA" "BUC
HAREST")))
HEAD Closed List : ("ARAD" "ZERIND" "TIMISOARA" "SIBIU" "
ORADEA")
HEAD Open List : ("LUGOJ" "RIMNICU VILCEA" "FAGARAS")
HEAD Parent-Child Map List : (("ARAD" ("ZERIND" "TIMISOAR
A" "SIBIU")) ("ZERIND" ("ORADEA"))
                                ("TIMISOARA" ("LUGOJ")) ("S
IBIU" ("RIMNICU VILCEA" "FAGARAS"))
                                ("ORADEA" NIL))
TAIL Closed List : ("IASI" "VASLUI" "NEAMT" "URZICENI" "H
IRSOVA")
TAIL Open List : ("BUCHAREST" "EFORIE")
TAIL Parent-Child Map List : (("IASI" ("VASLUI" "NEAMT"))
 ("VASLUI" ("URZICENI")) ("NEAMT" NIL)
                                ("URZICENI" ("HIRSOVA" "BUC
HAREST")) ("HIRSOVA" ("EFORIE")))
HEAD Closed List : ("ARAD" "ZERIND" "TIMISOARA" "SIBIU" "
ORADEA" "LUGOJ")
HEAD Open List : ("RIMNICU VILCEA" "FAGARAS" "MEHADIA")
HEAD Parent-Child Map List : (("ARAD" ("ZERIND" "TIMISOAR
A" "SIBIU")) ("ZERIND" ("ORADEA"))
```

```
                                ("TIMISOARA" ("LUGOJ")) ("S
IBIU" ("RIMNICU VILCEA" "FAGARAS"))
                                ("ORADEA" NIL) ("LUGOJ" ("M
EHADIA")))
TAIL Closed List : ("IASI" "VASLUI" "NEAMT" "URZICENI" "H
IRSOVA" "BUCHAREST")
TAIL Open List : ("EFORIE" "PITESTI" "GIURGIU" "FAGARAS")

TAIL Parent-Child Map List : (("IASI" ("VASLUI" "NEAMT"))
 ("VASLUI" ("URZICENI")) ("NEAMT" NIL)
                                ("URZICENI" ("HIRSOVA" "BUC
HAREST")) ("HIRSOVA" ("EFORIE"))
                                ("BUCHAREST" ("PITESTI" "GI
URGIU" "FAGARAS")))
Number of Nodes Generated : 19
Solution Path : ("ARAD" "SIBIU" "FAGARAS" "BUCHAREST" "UR
ZICENI" "VASLUI" "IASI")
Path Cost : 6
```

Ascending DFS with Forbidden Cities

1. Sample Inputs

> *ARAD*
>
> *IASI*
>
> *1*
>
> *2*

> *FAGARAS*
>
> *SKIP*

2. Sample Outputs

```
Closed List for Start after Iteration 0: NIL
Open List for Start after Iteration 0: (ARAD)
Closed List for Goal after Iteration 0: NIL
Open List for Goal after Iteration 0: (IASI)
Start Search Tree at Iteration 0: NIL
Goal Search Tree at Iteration 0: NIL

Closed List for Start after Iteration 1: (ARAD)
Open List for Start after Iteration 1: (SIBIU TIMISOARA Z
ERIND)
Closed List for Goal after Iteration 1: (IASI)
Open List for Goal after Iteration 1: (NEAMT VASLUI)
Start Search Tree at Iteration 1: ((ARAD (SIBIU TIMISOARA
 ZERIND)))
Goal Search Tree at Iteration 1: ((IASI (NEAMT VASLUI)))

Closed List for Start after Iteration 2: (SIBIU ARAD)
Open List for Start after Iteration 2: (FAGARAS ORADEA RI
MNICU VILCEA TIMISOARA ZERIND)
Closed List for Goal after Iteration 2: (NEAMT IASI)
Open List for Goal after Iteration 2: (VASLUI)
Start Search Tree at Iteration 2: ((ARAD (SIBIU TIMISOARA
 ZERIND))
```

```
                                        (SIBIU (FAGARAS ORADEA
 RIMNICU VILCEA)))
Goal Search Tree at Iteration 2: ((IASI (NEAMT VASLUI)) (
NEAMT NIL))

Closed List for Start after Iteration 3: (ORADEA SIBIU AR
AD)
Open List for Start after Iteration 3: (ZERIND RIMNICU VI
LCEA TIMISOARA)
Closed List for Goal after Iteration 3: (VASLUI NEAMT IAS
I)
Open List for Goal after Iteration 3: (URZICENI)
Start Search Tree at Iteration 3: ((ARAD (SIBIU TIMISOARA
 ZERIND))
                                        (SIBIU (FAGARAS ORADEA
 RIMNICU VILCEA)) (ORADEA (ZERIND)))
Goal Search Tree at Iteration 3: ((IASI (NEAMT VASLUI)) (
NEAMT NIL) (VASLUI (URZICENI)))

Closed List for Start after Iteration 4: (ZERIND ORADEA S
IBIU ARAD)
Open List for Start after Iteration 4: (RIMNICU VILCEA TI
MISOARA)
Closed List for Goal after Iteration 4: (URZICENI VASLUI
NEAMT IASI)
Open List for Goal after Iteration 4: (BUCHAREST HIRSOVA)
Start Search Tree at Iteration 4: ((ARAD (SIBIU TIMISOARA
 ZERIND))
```

```
                                        (SIBIU (FAGARAS ORADEA
 RIMNICU VILCEA)) (ORADEA (ZERIND))
                                        (ZERIND NIL))
Goal Search Tree at Iteration 4: ((IASI (NEAMT VASLUI)) (
NEAMT NIL) (VASLUI (URZICENI))
                                    (URZICENI (BUCHAREST HI
RSOVA)))

Closed List for Start after Iteration 5: (RIMNICU VILCEA
ZERIND ORADEA SIBIU ARAD)
Open List for Start after Iteration 5: (CRAIOVA PITESTI T
IMISOARA)
Closed List for Goal after Iteration 5: (BUCHAREST URZICE
NI VASLUI NEAMT IASI)
Open List for Goal after Iteration 5: (FAGARAS GIURGIU PI
TESTI HIRSOVA)
Start Search Tree at Iteration 5: ((ARAD (SIBIU TIMISOARA
 ZERIND))
                                        (SIBIU (FAGARAS ORADEA
 RIMNICU VILCEA)) (ORADEA (ZERIND))
                                        (ZERIND NIL) (RIMNICU
VILCEA (CRAIOVA PITESTI)))
Goal Search Tree at Iteration 5: ((IASI (NEAMT VASLUI)) (
NEAMT NIL) (VASLUI (URZICENI))
                                    (URZICENI (BUCHAREST HI
RSOVA))
                                    (BUCHAREST (FAGARAS GIU
RGIU PITESTI)))
```

```
Closed List for Start after Iteration 6: (CRAIOVA RIMNICU
 VILCEA ZERIND ORADEA SIBIU ARAD)
Open List for Start after Iteration 6: (DROBETA PITESTI T
IMISOARA)
Closed List for Goal after Iteration 6: (GIURGIU BUCHARES
T URZICENI VASLUI NEAMT IASI)
Open List for Goal after Iteration 6: (PITESTI HIRSOVA)
Start Search Tree at Iteration 6: ((ARAD (SIBIU TIMISOARA
 ZERIND))
                                   (SIBIU (FAGARAS ORADEA
 RIMNICU VILCEA)) (ORADEA (ZERIND))
                                   (ZERIND NIL) (RIMNICU
VILCEA (CRAIOVA PITESTI))
                                   (CRAIOVA (DROBETA PITE
STI)))
Goal Search Tree at Iteration 6: ((IASI (NEAMT VASLUI)) (
NEAMT NIL) (VASLUI (URZICENI))
                                  (URZICENI (BUCHAREST HI
RSOVA))
                                  (BUCHAREST (FAGARAS GIU
RGIU PITESTI)) (GIURGIU NIL))

Closed List for Start after Iteration 7: (DROBETA CRAIOVA
 RIMNICU VILCEA ZERIND ORADEA SIBIU
                                        ARAD)
Open List for Start after Iteration 7: (MEHADIA PITESTI T
IMISOARA)
```

```
Closed List for Goal after Iteration 7: (PITESTI GIURGIU
BUCHAREST URZICENI VASLUI NEAMT IASI)
Open List for Goal after Iteration 7: (CRAIOVA RIMNICU VI
LCEA HIRSOVA)
Start Search Tree at Iteration 7: ((ARAD (SIBIU TIMISOARA
 ZERIND))
                                  (SIBIU (FAGARAS ORADEA
 RIMNICU VILCEA)) (ORADEA (ZERIND))
                                  (ZERIND NIL) (RIMNICU
VILCEA (CRAIOVA PITESTI))
                                  (CRAIOVA (DROBETA PITE
STI)) (DROBETA (MEHADIA)))
Goal Search Tree at Iteration 7: ((IASI (NEAMT VASLUI)) (
NEAMT NIL) (VASLUI (URZICENI))
                                 (URZICENI (BUCHAREST HI
RSOVA))
                                 (BUCHAREST (FAGARAS GIU
RGIU PITESTI)) (GIURGIU NIL)
                                 (PITESTI (CRAIOVA RIMNI
CU VILCEA)))

Closed List for Start after Iteration 8: (MEHADIA DROBETA
 CRAIOVA RIMNICU VILCEA ZERIND ORADEA
                                       SIBIU ARAD)
Open List for Start after Iteration 8: (LUGOJ PITESTI TIM
ISOARA)
Closed List for Goal after Iteration 8: (CRAIOVA PITESTI
GIURGIU BUCHAREST URZICENI VASLUI NEAMT
```

```
                                                    IASI)
Open List for Goal after Iteration 8: (DROBETA RIMNICU VI
LCEA HIRSOVA)
Start Search Tree at Iteration 8: ((ARAD (SIBIU TIMISOARA
 ZERIND))
                                   (SIBIU (FAGARAS ORADEA
 RIMNICU VILCEA)) (ORADEA (ZERIND))
                                   (ZERIND NIL) (RIMNICU
VILCEA (CRAIOVA PITESTI))
                                   (CRAIOVA (DROBETA PITE
STI)) (DROBETA (MEHADIA))
                                   (MEHADIA (LUGOJ)))
Goal Search Tree at Iteration 8: ((IASI (NEAMT VASLUI)) (
NEAMT NIL) (VASLUI (URZICENI))
                                  (URZICENI (BUCHAREST HI
RSOVA))
                                  (BUCHAREST (FAGARAS GIU
RGIU PITESTI)) (GIURGIU NIL)
                                  (PITESTI (CRAIOVA RIMNI
CU VILCEA))
                                  (CRAIOVA (DROBETA RIMNI
CU VILCEA)))

Total Number of Nodes Generated: 22
Solution Path: (ARAD SIBIU RIMNICU VILCEA CRAIOVA PITESTI
 BUCHAREST URZICENI VASLUI IASI)
Path Cost: 8
```

Descending DFS with Forbidden Cities

1. Sample Inputs

> *ARAD*
>
> *IASI*
>
> *2*
>
> *2*
>
> *SIBIU*
>
> *GIURGIU*

2. Sample Outputs

```
Closed List for Start after Iteration 0: NIL
Open List for Start after Iteration 0: (ARAD)
Closed List for Goal after Iteration 0: NIL
Open List for Goal after Iteration 0: (IASI)
Start Search Tree at Iteration 0: NIL
Goal Search Tree at Iteration 0: NIL

Closed List for Start after Iteration 1: (ARAD)
Open List for Start after Iteration 1: (ZERIND TIMISOARA
SIBIU)
Closed List for Goal after Iteration 1: (IASI)
Open List for Goal after Iteration 1: (VASLUI NEAMT)
Start Search Tree at Iteration 1: ((ARAD (ZERIND TIMISOAR
A SIBIU)))
```

```
Goal Search Tree at Iteration 1: ((IASI (VASLUI NEAMT)))

Closed List for Start after Iteration 2: (ZERIND ARAD)
Open List for Start after Iteration 2: (ORADEA TIMISOARA
SIBIU)
Closed List for Goal after Iteration 2: (VASLUI IASI)
Open List for Goal after Iteration 2: (URZICENI NEAMT)
Start Search Tree at Iteration 2: ((ARAD (ZERIND TIMISOAR
A SIBIU)) (ZERIND (ORADEA)))
Goal Search Tree at Iteration 2: ((IASI (VASLUI NEAMT)) (
VASLUI (URZICENI)))

Closed List for Start after Iteration 3: (ORADEA ZERIND A
RAD)
Open List for Start after Iteration 3: (SIBIU TIMISOARA)
Closed List for Goal after Iteration 3: (URZICENI VASLUI
IASI)
Open List for Goal after Iteration 3: (HIRSOVA BUCHAREST
NEAMT)
Start Search Tree at Iteration 3: ((ARAD (ZERIND TIMISOAR
A SIBIU)) (ZERIND (ORADEA))
                                    (ORADEA (SIBIU)))
Goal Search Tree at Iteration 3: ((IASI (VASLUI NEAMT)) (
VASLUI (URZICENI))
                                  (URZICENI (HIRSOVA BUCH
AREST)))

Closed List for Start after Iteration 4: (TIMISOARA ORADE
```

```
A ZERIND ARAD)
Open List for Start after Iteration 4: (LUGOJ)
Closed List for Goal after Iteration 4: (HIRSOVA URZICENI
 VASLUI IASI)
Open List for Goal after Iteration 4: (EFORIE BUCHAREST N
EAMT)
Start Search Tree at Iteration 4: ((ARAD (ZERIND TIMISOAR
A SIBIU)) (ZERIND (ORADEA))
                                    (ORADEA (SIBIU)) (TIMI
SOARA (LUGOJ)))
Goal Search Tree at Iteration 4: ((IASI (VASLUI NEAMT)) (
VASLUI (URZICENI))
                                  (URZICENI (HIRSOVA BUCH
AREST)) (HIRSOVA (EFORIE)))

Closed List for Start after Iteration 5: (LUGOJ TIMISOARA
 ORADEA ZERIND ARAD)
Open List for Start after Iteration 5: (MEHADIA)
Closed List for Goal after Iteration 5: (EFORIE HIRSOVA U
RZICENI VASLUI IASI)
Open List for Goal after Iteration 5: (BUCHAREST NEAMT)
Start Search Tree at Iteration 5: ((ARAD (ZERIND TIMISOAR
A SIBIU)) (ZERIND (ORADEA))
                                    (ORADEA (SIBIU)) (TIMI
SOARA (LUGOJ)) (LUGOJ (MEHADIA)))
Goal Search Tree at Iteration 5: ((IASI (VASLUI NEAMT)) (
VASLUI (URZICENI))
                                  (URZICENI (HIRSOVA BUCH
```

```
AREST)) (HIRSOVA (EFORIE)) (EFORIE NIL))

Closed List for Start after Iteration 6: (MEHADIA LUGOJ T
IMISOARA ORADEA ZERIND ARAD)
Open List for Start after Iteration 6: (DROBETA)
Closed List for Goal after Iteration 6: (BUCHAREST EFORIE
 HIRSOVA URZICENI VASLUI IASI)
Open List for Goal after Iteration 6: (PITESTI GIURGIU FA
GARAS NEAMT)
Start Search Tree at Iteration 6: ((ARAD (ZERIND TIMISOAR
A SIBIU)) (ZERIND (ORADEA))
                                   (ORADEA (SIBIU)) (TIMI
SOARA (LUGOJ)) (LUGOJ (MEHADIA))
                                   (MEHADIA (DROBETA)))
Goal Search Tree at Iteration 6: ((IASI (VASLUI NEAMT)) (
VASLUI (URZICENI))
                                  (URZICENI (HIRSOVA BUCH
AREST)) (HIRSOVA (EFORIE)) (EFORIE NIL)
                                  (BUCHAREST (PITESTI GIU
RGIU FAGARAS)))

Closed List for Start after Iteration 7: (DROBETA MEHADIA
 LUGOJ TIMISOARA ORADEA ZERIND ARAD)
Open List for Start after Iteration 7: (CRAIOVA)
Closed List for Goal after Iteration 7: (PITESTI BUCHARES
T EFORIE HIRSOVA URZICENI VASLUI IASI)
Open List for Goal after Iteration 7: (RIMNICU VILCEA CRA
IOVA GIURGIU FAGARAS NEAMT)
```

```
Start Search Tree at Iteration 7: ((ARAD (ZERIND TIMISOAR
A SIBIU)) (ZERIND (ORADEA))
                                    (ORADEA (SIBIU)) (TIMI
SOARA (LUGOJ)) (LUGOJ (MEHADIA))
                                    (MEHADIA (DROBETA)) (D
ROBETA (CRAIOVA)))
Goal Search Tree at Iteration 7: ((IASI (VASLUI NEAMT)) (
VASLUI (URZICENI))
                                   (URZICENI (HIRSOVA BUCH
AREST)) (HIRSOVA (EFORIE)) (EFORIE NIL)
                                   (BUCHAREST (PITESTI GIU
RGIU FAGARAS))
                                   (PITESTI (RIMNICU VILCE
A CRAIOVA)))

Closed List for Start after Iteration 8: (CRAIOVA DROBETA
 MEHADIA LUGOJ TIMISOARA ORADEA ZERIND
                                          ARAD)
Open List for Start after Iteration 8: (RIMNICU VILCEA PI
TESTI)
Closed List for Goal after Iteration 8: (RIMNICU VILCEA P
ITESTI BUCHAREST EFORIE HIRSOVA
                                        URZICENI VASLUI
IASI)
Open List for Goal after Iteration 8: (SIBIU CRAIOVA GIUR
GIU FAGARAS NEAMT)
Start Search Tree at Iteration 8: ((ARAD (ZERIND TIMISOAR
A SIBIU)) (ZERIND (ORADEA))
```

(ORADEA (SIBIU)) (TIMI
SOARA (LUGOJ)) (LUGOJ (MEHADIA))
(MEHADIA (DROBETA)) (D
ROBETA (CRAIOVA))
(CRAIOVA (RIMNICU VILC
EA PITESTI)))
Goal Search Tree at Iteration 8: ((IASI (VASLUI NEAMT)) (
VASLUI (URZICENI))
(URZICENI (HIRSOVA BUCH
AREST)) (HIRSOVA (EFORIE)) (EFORIE NIL)
(BUCHAREST (PITESTI GIU
RGIU FAGARAS))
(PITESTI (RIMNICU VILCE
A CRAIOVA))
(RIMNICU VILCEA (SIBIU
CRAIOVA)))

Closed List for Start after Iteration 9: (RIMNICU VILCEA
CRAIOVA DROBETA MEHADIA LUGOJ TIMISOARA
ORADEA ZERIND A
RAD)
Open List for Start after Iteration 9: (SIBIU PITESTI)
Closed List for Goal after Iteration 9: (CRAIOVA RIMNICU
VILCEA PITESTI BUCHAREST EFORIE HIRSOVA
URZICENI VASLUI
IASI)
Open List for Goal after Iteration 9: (DROBETA GIURGIU FA
GARAS NEAMT)

```
Start Search Tree at Iteration 9: ((ARAD (ZERIND TIMISOAR
A SIBIU)) (ZERIND (ORADEA))
                                    (ORADEA (SIBIU)) (TIMI
SOARA (LUGOJ)) (LUGOJ (MEHADIA))
                                    (MEHADIA (DROBETA)) (D
ROBETA (CRAIOVA))
                                    (CRAIOVA (RIMNICU VILC
EA PITESTI))
                                    (RIMNICU VILCEA (SIBIU
 PITESTI)))
Goal Search Tree at Iteration 9: ((IASI (VASLUI NEAMT)) (
VASLUI (URZICENI))
                                   (URZICENI (HIRSOVA BUCH
AREST)) (HIRSOVA (EFORIE)) (EFORIE NIL)
                                   (BUCHAREST (PITESTI GIU
RGIU FAGARAS))
                                   (PITESTI (RIMNICU VILCE
A CRAIOVA))
                                   (RIMNICU VILCEA (SIBIU
CRAIOVA)) (CRAIOVA (DROBETA)))

Total Number of Nodes Generated: 24
Solution Path: (ARAD TIMISOARA LUGOJ MEHADIA DROBETA CRAI
OVA RIMNICU VILCEA PITESTI BUCHAREST
                URZICENI VASLUI IASI)
Path Cost: 11
```