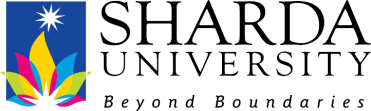
**Artificial Intelligence Lab**

CSP-472

B.Tech.-7th Semester

**Submitted by:**

**Atul Panwar (2019006131)**



Sharda School of Engineering & Technology

Department of Computer Science & Engineering

Table of Contents

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No.** | **Title** | **Date** | **Sign** |
| 1.1 | Implementation of the Water Jug Problem. | 20/07/22 |  |
| 1.2 a | Write a LISP function to compute the sum of squares. | 27/07/22 |  |
| 1.2 b | Write a LISP function to compute the difference of squares. (if x > y return x2 – y2, Otherwise y2 – x2). | 27/07/22 |  |
| 1.2 c | Write a Recursive LISP function which takes one argument as a list and returns the last element of the list. (Do not use the last predicate.) | 03/08/22 |  |
| 1.2 d | Write a Recursive LISP function which takes one argument as a list and return list except the last element of the list. (Do not use butlast.) | 03/08/22 |  |
| 1.2 e | Write a Recursive LISP function which takes one argument as a list and returns the reverse of the list. (Do not use reverse predicate). | 8/08/22 |  |
| 1.2 f | Write a Recursive LISP function which takes two arguments first an atom second a list returns a list after removing the first occurrence of that atom within the list. | 22/08/22 |  |
| 1.2 g | Write a Recursive LISP function which appends two lists together. | 22/08/22 |  |
| 1.2 h | Write a recursive LISP function which takes 2 lists as arguments and returns a list containing alternate elements from each list. | 29/08/22 |  |
| 1.3 a | Write a function that compute the factorial of a number.(factorial of 0 is 1, and factorial of n is n\*(n-1)\*...1.Factorial is defined only for integers greater than or equal to 0.) | 05/09/22 |  |
| 1.3 b | Write a function that evaluates a fully parenthesized infix arithmetic expression. For examples, (infix (1+ (2\*3))) should return 7. | 19/09/22 |  |
| 1.3 c | Write a function that performs a depth first traversal of a binary tree. The function should return a list containing the tree nodes in the order they were visited. | 19/09/22 |  |
| 1.3 d | Write a LISP program for the water jug problem. | 26/09/22 |  |
| 1.3 e | Write a LISP program that determines whether an integer is prime. | 03/10/22 |  |
| 1.4 | Refer following figure as map with distance details, Write a program in your preferred language to generate path from ARAD to BUCHREST, analyze result obtained by: a)Depth First Search b)Breadth First Search c)Uniform Cost Search | 10/10/22 |  |
| 1.5 | Write a program in your preferred language to generate steps to solve the Tower of Hanoi problem. | 31/10/22 |  |
| 1.6 | Write a program in your preferred language to solve the 8 puzzle Problem-using A\* algorithm Guidelines for continuous assessment. | 31/10/22 |  |
| 1.7 | Write PROLOG program to Program to categorize animal characteristics. | 07/11/22 |  |
| 1.8 | Write a PROLOG program that answers questions about family members and relationships including predicates and rules which define sister, brother, father, mother, grandchild, grandfather and uncle. The program should be able to answer queries such as the following:  father(x, Amit)  grandson(x, y)  uncle (sumit, puneet)  mother (anita, x) | 07/11/22 |  |

**1.1 Aim- Write a program to solve the water jug problem using LISP.**

**Program:**

(defvar state);declare global variable state

(defun start ()

(data) (checkB))

(defun data()

(

format t "~%WATER JUG PROBLEM")

(format t "~%FILL JUGNAME: (fillJUGNAME)")

(format t "~%EMPTY JUGNAME: (emptyJUGNAME)")

(format t "~%PUT A TO B : (putAtoB)")

(format t "~%~%GIVE THE STARTING STATE OF EACH JUG IN FORM (A B C) E.G (0 0 0)")

(format t "~%MAX A=7,MAX B=4 AND MAX C=2: ")

(setf state (read));SET VALUE AT VARIABLE STATE

)

(format t "~%STARTING CONDITION OF EACH JUG: ~3a" state)

;FILLING A

(defun fillA () (let((x(first state))(y(second state))(z(third state)))

(if (< x 7) (let ((newstate (list 7 y z)))

(prog1 state (format t "~%NEW STATE: ~3a" newstate ))

(setf (first state) 7)))

(if (>= x 7) (format t "~%A IS ALREADY FULL")))

(checkB))

;FILLING B

(defun fillB () (let((x(first state))(y(second state))(z(third state)))

(if (< y 4) (let ((newstate (list x 4 z)))

(prog1 state (format t "~%NEW STATE: ~3a" newstate ))

(setf (second state) 4)))

(if (>= y 4)(format t "~%B IS ALREADY FULL")))

(checkB))

;FILLING C

(defun fillC () (let((x(first state))(y(second state))(z(third state)))

(if (< z 2) (let ((newstate (list x y 2)))

(prog1 state (format t "~%NEW STATE: ~3a" newstate ))

(setf (third state) 2)))

(if (>= z 2) (format t "~%C IS ALREADY FULL")))

(checkB) )

;EMPTY A

(defun emptyA () (let((x(first state))(y(second state))(z(third state)))

(if (> x 0) (let ((newstate (list 0 y z)))

(prog1 state (format t "~%NEW STATE: ~3a" newstate ))

(setf (first state) 0)))

(if (<= x 0) (format t "~%A IS ALREADY EMPTY")))

(checkB))

;EMPTY Β

(defun emptyB () (let((x(first state))(y(second state))(z(third state)))

(if (> y 0) (let ((newstate (list x 0 z)))

(prog1 state (format t "~%NEW STATE: ~3a" newstate ))

(setf (second state) 0)))

(if (<= y 0) (format t "~%B IS ALREADY EMPTY")))

(checkB))

;EMPTY C

(defun emptyC () (let((x(first state))(y(second state))(z(third state)))

(if (> z 0) (let ((newstate (list x y 0)))

(prog1 state (format t "~%NEW STATE: ~3a" newstate ))

(setf (third state) 0)))

(if (<= z 0) (format t "~%C IS ALREADY EMPTY")))

(checkB))

;PUT A TO B

(defun putAtoB () (let((x(first state))(y(second state))(z(third state)))

(cond((<= x 0) (format t"~%A IS ALREADY EMPTY"))

((>= y 4) (format t"~%B IS FULL"))

((and (> x 0) (< y 4))

(let ((local (- 4 y)))

(if (or (> x local)(= x local))

(let ((newstate (list (- x (- 4 y)) 4 z)))

(setf (first state) (first newstate))

(setf (second state) (second newstate))

(setf (third state) (third newstate))

(format t "~%NEW STATE: ~3a" newstate))

(let ((newstate (list 0 (+ y x) z)))

(setf (first state) (first newstate))

(setf (second state) (second newstate))

(setf (third state) (third newstate))

(format t "~%NEW STATE: ~3a" newstate))

)))))

(checkB))

;PUT B TO A

(defun putBtoA () (let((x(first state))(y(second state))(z(third state)))

(cond((>= x 7) (format t"~%A IS FULL"))

((<= y 0) (format t"~%B IS EMPTY"))

((and (> y 0) (< x 7))

(let ((local (- 7 x)))

(if (or (> x local)(= x local))

(let ((newstate (list 7 (- y (- 7 x)) z)))

(setf (first state) (first newstate))

(setf (second state) (second newstate))

(setf (third state) (third newstate))

(format t "~%NEW STATE: ~3a" newstate))

(let ((newstate (list (+ y x) 0 z)))

(setf (first state) (first newstate))

(setf (second state) (second newstate))

(setf (third state) (third newstate))

(format t "~%NEW STATE: ~3a" newstate))

)))))

(checkB))

;PUT A TO C

(defun putAtoC () (let((x(first state))(y(second state))(z(third state)))

(cond((<= x 0) (format t"~%A IS EMPTY"))

((>= z 2) (format t"~%C IS FULL"))

((and (> x 0) (< z 2))

(let ((local (- 2 z)))

(if (or (> x local)(= x local))

(let ((newstate (list (- x (- 2 z)) y 2)))

(setf (first state) (first newstate))

(setf (second state) (second newstate))

(setf (third state) (third newstate))

(format t "~%NEW STATE: ~3a" newstate))

(let ((newstate (list 0 y (+ z x))))

(setf (first state) (first newstate))

(setf (second state) (second newstate))

(setf (third state) (third newstate))

(format t "~%NEW STATE: ~3a" newstate))

)))))

(checkB))

;PUT C TO A

(defun putCtoA () (let((x(first state))(y(second state))(z(third state)))

(cond((>= x 7) (format t"~%A IS FULL"))

((<= z 0) (format t"~%C IS EMPTY"))

((and (> z 0) (< x 7))

(let ((local (- 7 x)))

(if (or (> x local)(= x local))

(let ((newstate (list 7 y (- z (- 7 x)))))

(setf (first state) (first newstate))

(setf (second state) (second newstate))

(setf (third state) (third newstate))

(format t "~%NEW STATE: ~3a" newstate))

(let ((newstate (list (+ z x) y 0)))

(setf (first state) (first newstate))

(setf (second state) (second newstate))

(setf (third state) (third newstate))

(format t "~%NEW STATE: ~3a" newstate))

)))))

(checkB))

;PUT B TO C

(defun putBtoC () (let((x(first state))(y(second state))(z(third state)))

(cond((<= y 0) (format t"~%B IS EMPTY"))

((>= z 2) (format t"~%C IS FULL"))

((and (> y 0) (< z 2))

(let ((local (- 2 z)))

(if (or (> y local)(= y local))

(let ((newstate (list x (- y (- 2 z)) 2)))

(setf (first state) (first newstate))

(setf (second state) (second newstate))

(setf (third state) (third newstate))

(format t "~%NEW STATE: ~3a" newstate))

(let ((newstate (list x 0 (+ z y))))

(setf (first state) (first newstate))

(setf (second state) (second newstate))

(setf (third state) (third newstate))

(format t "~%NEW STATE: ~3a" newstate))

)))))

(checkB))

;PUT C TO B

(defun putCtoB () (let((x(first state))(y(second state))(z(third state)))

(cond((>= y 4) (format t"~%B IS FULL"))

((<= z 0) (format t"~%C IS EMPTY"))

((and (> z 0) (< y 4))

(let ((local (- 4 y)))

(if (or (> y local)(= y local))

(let ((newstate (list x 4 (- z (- 4 y)))))

(setf (first state) (first newstate))

(setf (second state) (second newstate))

(setf (third state) (third newstate))

(format t "~%NEW STATE: ~3a" newstate))

(let ((newstate (list x (+ y z) 0)))

(setf (first state) (first newstate))

(setf (second state) (second newstate))

(setf (third state) (third newstate))

(format t "~%NEW STATE: ~3a" newstate))

)))))

(checkB))

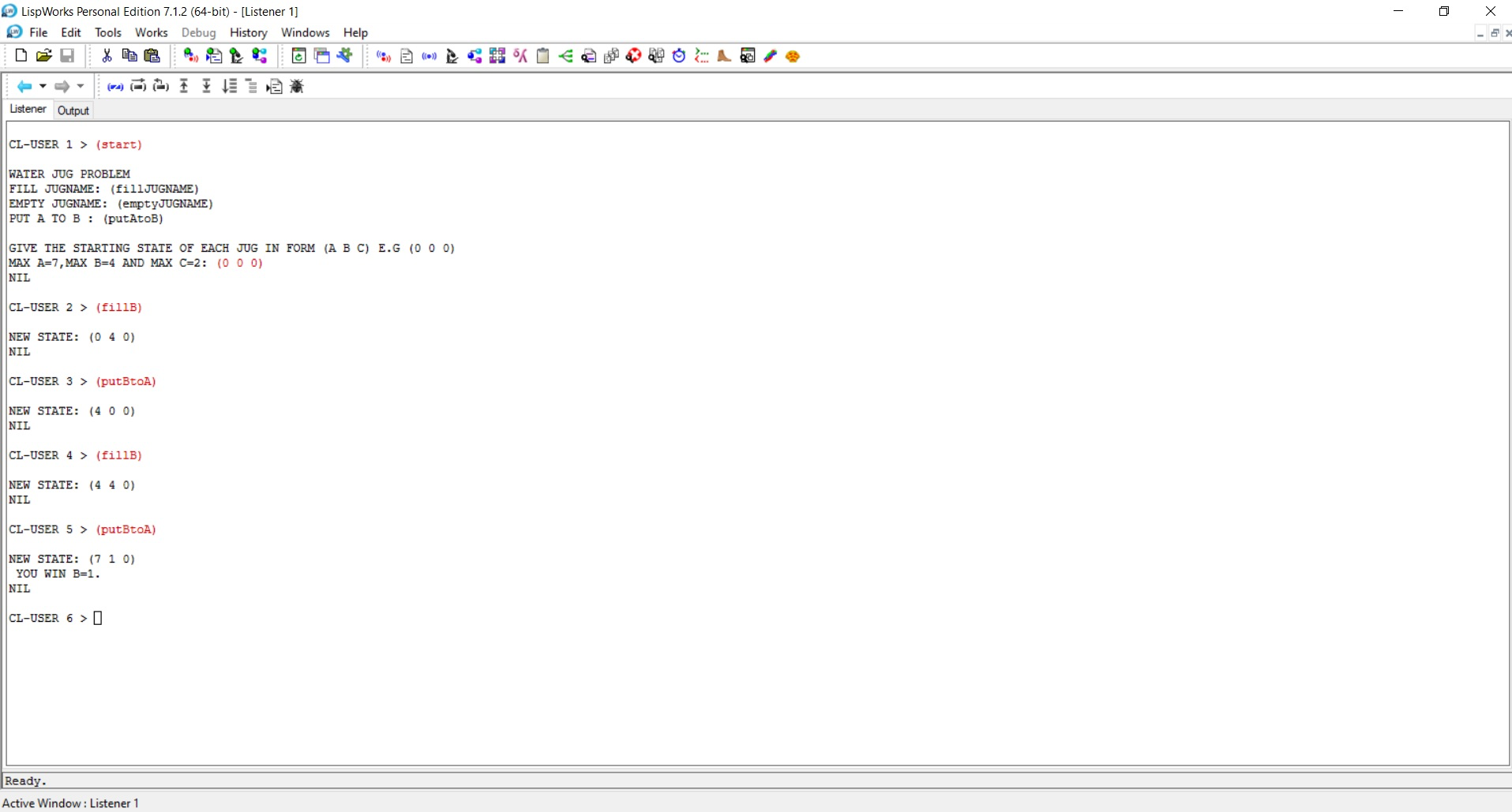
;CHECK IF B=1

(defun checkB () (let ((y(second state)))

(if (= y 1) (format t "~% YOU WIN B=1."

))))

**Output-**



**1.2 INTRODUCTION TO LISP, AND BASIC PROGRAMMING IN LISP**

* 1. **Aim- Define a LISP function to compute sum of squares**

(defun sumsqr (x y) (+(\* x x)(\* y y)))

Write (sumsqr 2 3);

**OUTPUT** : sumsqr = 13

* 1. **Aim- Write a LISP function to compute the difference of squares. (if x > y return x2 – y2, Otherwise y2 – x2).**

defun diffsqr (x y) (if(> x y)

(-(\* x x) (\* y y))

(-(\* y y) (\* x x))))

Write (diffsqr 2 3)

**OUTPUT** : DIFFSQR 5

* 1. **Aim- Write a Recursive LISP function which takes one argument as a list and returns the last element of the list. (Do not use the last predicate.)**

defun last\_element(ab\_list) (first(reverse ab\_list)))

Write (last\_element (a b c d))

**OUTPUT** : LAST\_ELEMENT D

* 1. **Aim- Write a Recursive LISP function which takes one argument as a list and return list except the last element of the list. (Do not use butlast.)**

(defun not\_last(ab\_list) (reverse(rest(reverse ab\_list))))

Write (not\_last '(a b c d e))

**OUTPUT** : NOT\_LAST (A B C D)

* 1. **Aim- Write a Recursive LISP function which takes one argument as a list and returns the reverse of the list. (Do not use reverse predicate).**

(defun list-append (L1 L2) "Append L1 by L2."

(if (null L1)

L2

(cons (first L1) (list-append (rest L1) L2))))

(defun show-list-reverse (L)

"Create a new list containing the elements of L in reversed order." (if (null L)

nil

(list-append (show-list-reverse (rest L)) (list (first L)))))

Write (show-list-reverse '(1 2 3 4))

**OUTPUT** :

LIST-APPEND

SHOW-LIST-REVERSE (4 3 2 1)

* 1. **Aim- Write a Recursive LISP function which takes two arguments: first an atom second a list returns a list after removing the first occurrence of that atom within the list.**

(defun remove(lst elt) (cond((null lst)nil) ((equal(first lst)elt)(rest lst)) (elt(cons(first lst) (remove(rest lst)elt)))))

Write (remove '(1 2 3 3 4 4)'3)

**OUTPUT** :

REMOVE (1 2 3 4)

* 1. **Aim- Write a Recursive LISP function which appends two lists together.**

(defun append\_list( list1 list2)(if(nulllist1)

list2

(cons(firstlist1)(append\_list(restlist1)list2))))(print(append\_list'(12)'(34)))

**OUTPUT** :

(1 2 3 4)

* 1. **Aim- Write a recursive LISP function which takes 2 lists as arguments and returns a list containing alternate elements from each list.**

(defunalternate\_list(list1list2)

(cond((and(endp list1)(endp list2))nil)((endplist1)list2)

((endplist2)list1)

(T (cons (car list1)(alternate\_list list2 (cdr list1))))))(print(alternate\_list'(a bc)'(123)))

**OUTPUT** :

(A 1 B 2 C 3)

**1.3 INTRODUCTION TO LISP, AND BASIC PROGRAMMING IN LISP**

* 1. **Aim- Write a function that computes the factorial of a number.(factorial of 0 is 1, and factorial of n is n\*(n-1)\*...1.Factorial is defined only for integers greater than or equal to 0.)**

(defun factorial (N)

"Compute the factorial of N." (if (= N 1)

1

(\* N (factorial (- N 1))))) Write (factorial 5)

**OUTPUT:**

FACTORIAL 120

* 1. **Aim- Write a function that evaluates a fully parenthesized infix arithmetic expression. For examples, (infix (1+ (2\*3))) should return 7.**

(defun infix (var)(if(atomvar)

var

(eval(list(infix(secondvar))(infix(firstvar))(infix(thirdvar))

)

)

)

)

(print(infix'(25 +(5\*15))))

**OUTPUT** :

7

* 1. **Aim- Write a function that performs a depth first traversal of a binary tree. The function should return a list containing the tree nodes in the order they were visited.**

(defun by-levels (ts)(if(nullts)

'()

(append

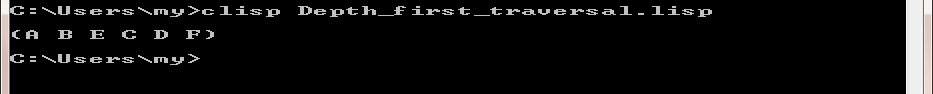
(mapcar#'(lambda(x)(if(listp x)(carx)x))ts)

(by-levels(mapcan#'(lambda(x) (if(listpx)(cadrx) '())) ts)))))

(defun leafs-of-tree-by-levels (tree)(by-levels(listtree)))

(print(leafs-of-tree-by-levels'(a((b(cd))(e(f))))) )

**OUTPUT** :

****

* 1. **Aim- Write a LISP program for the water jug problem.**

(defvar state);declare global variable state

(defun start ()

(data) (checkB))

(defun data()

(

format t "~%WATER JUG PROBLEM")

(format t "~%FILL JUGNAME: (fillJUGNAME)")

(format t "~%EMPTY JUGNAME: (emptyJUGNAME)")

(format t "~%PUT A TO B : (putAtoB)")

(format t "~%~%GIVE THE STARTING STATE OF EACH JUG IN FORM (A B C) E.G (0 0 0)")

(format t "~%MAX A=7,MAX B=4 AND MAX C=2: ")

(setf state (read));SET VALUE AT VARIABLE STATE

)

(format t "~%STARTING CONDITION OF EACH JUG: ~3a" state)

;FILLING A

(defun fillA () (let((x(first state))(y(second state))(z(third state)))

(if (< x 7) (let ((newstate (list 7 y z)))

(prog1 state (format t "~%NEW STATE: ~3a" newstate ))

(setf (first state) 7)))

(if (>= x 7) (format t "~%A IS ALREADY FULL")))

(checkB))

;FILLING B

(defun fillB () (let((x(first state))(y(second state))(z(third state)))

(if (< y 4) (let ((newstate (list x 4 z)))

(prog1 state (format t "~%NEW STATE: ~3a" newstate ))

(setf (second state) 4)))

(if (>= y 4)(format t "~%B IS ALREADY FULL")))

(checkB))

;FILLING C

(defun fillC () (let((x(first state))(y(second state))(z(third state)))

(if (< z 2) (let ((newstate (list x y 2)))

(prog1 state (format t "~%NEW STATE: ~3a" newstate ))

(setf (third state) 2)))

(if (>= z 2) (format t "~%C IS ALREADY FULL")))

(checkB) )

;EMPTY A

(defun emptyA () (let((x(first state))(y(second state))(z(third state)))

(if (> x 0) (let ((newstate (list 0 y z)))

(prog1 state (format t "~%NEW STATE: ~3a" newstate ))

(setf (first state) 0)))

(if (<= x 0) (format t "~%A IS ALREADY EMPTY")))

(checkB))

;EMPTY Β

(defun emptyB () (let((x(first state))(y(second state))(z(third state)))

(if (> y 0) (let ((newstate (list x 0 z)))

(prog1 state (format t "~%NEW STATE: ~3a" newstate ))

(setf (second state) 0)))

(if (<= y 0) (format t "~%B IS ALREADY EMPTY")))

(checkB))

;EMPTY C

(defun emptyC () (let((x(first state))(y(second state))(z(third state)))

(if (> z 0) (let ((newstate (list x y 0)))

(prog1 state (format t "~%NEW STATE: ~3a" newstate ))

(setf (third state) 0)))

(if (<= z 0) (format t "~%C IS ALREADY EMPTY")))

(checkB))

;PUT A TO B

(defun putAtoB () (let((x(first state))(y(second state))(z(third state)))

(cond((<= x 0) (format t"~%A IS ALREADY EMPTY"))

((>= y 4) (format t"~%B IS FULL"))

((and (> x 0) (< y 4))

(let ((local (- 4 y)))

(if (or (> x local)(= x local))

(let ((newstate (list (- x (- 4 y)) 4 z)))

(setf (first state) (first newstate))

(setf (second state) (second newstate))

(setf (third state) (third newstate))

(format t "~%NEW STATE: ~3a" newstate))

(let ((newstate (list 0 (+ y x) z)))

(setf (first state) (first newstate))

(setf (second state) (second newstate))

(setf (third state) (third newstate))

(format t "~%NEW STATE: ~3a" newstate))

)))))

(checkB))

;PUT B TO A

(defun putBtoA () (let((x(first state))(y(second state))(z(third state)))

(cond((>= x 7) (format t"~%A IS FULL"))

((<= y 0) (format t"~%B IS EMPTY"))

((and (> y 0) (< x 7))

(let ((local (- 7 x)))

(if (or (> x local)(= x local))

(let ((newstate (list 7 (- y (- 7 x)) z)))

(setf (first state) (first newstate))

(setf (second state) (second newstate))

(setf (third state) (third newstate))

(format t "~%NEW STATE: ~3a" newstate))

(let ((newstate (list (+ y x) 0 z)))

(setf (first state) (first newstate))

(setf (second state) (second newstate))

(setf (third state) (third newstate))

(format t "~%NEW STATE: ~3a" newstate))

)))))

(checkB))

;PUT A TO C

(defun putAtoC () (let((x(first state))(y(second state))(z(third state)))

(cond((<= x 0) (format t"~%A IS EMPTY"))

((>= z 2) (format t"~%C IS FULL"))

((and (> x 0) (< z 2))

(let ((local (- 2 z)))

(if (or (> x local)(= x local))

(let ((newstate (list (- x (- 2 z)) y 2)))

(setf (first state) (first newstate))

(setf (second state) (second newstate))

(setf (third state) (third newstate))

(format t "~%NEW STATE: ~3a" newstate))

(let ((newstate (list 0 y (+ z x))))

(setf (first state) (first newstate))

(setf (second state) (second newstate))

(setf (third state) (third newstate))

(format t "~%NEW STATE: ~3a" newstate))

)))))

(checkB))

;PUT C TO A

(defun putCtoA () (let((x(first state))(y(second state))(z(third state)))

(cond((>= x 7) (format t"~%A IS FULL"))

((<= z 0) (format t"~%C IS EMPTY"))

((and (> z 0) (< x 7))

(let ((local (- 7 x)))

(if (or (> x local)(= x local))

(let ((newstate (list 7 y (- z (- 7 x)))))

(setf (first state) (first newstate))

(setf (second state) (second newstate))

(setf (third state) (third newstate))

(format t "~%NEW STATE: ~3a" newstate))

(let ((newstate (list (+ z x) y 0)))

(setf (first state) (first newstate))

(setf (second state) (second newstate))

(setf (third state) (third newstate))

(format t "~%NEW STATE: ~3a" newstate))

)))))

(checkB))

;PUT B TO C

(defun putBtoC () (let((x(first state))(y(second state))(z(third state)))

(cond((<= y 0) (format t"~%B IS EMPTY"))

((>= z 2) (format t"~%C IS FULL"))

((and (> y 0) (< z 2))

(let ((local (- 2 z)))

(if (or (> y local)(= y local))

(let ((newstate (list x (- y (- 2 z)) 2)))

(setf (first state) (first newstate))

(setf (second state) (second newstate))

(setf (third state) (third newstate))

(format t "~%NEW STATE: ~3a" newstate))

(let ((newstate (list x 0 (+ z y))))

(setf (first state) (first newstate))

(setf (second state) (second newstate))

(setf (third state) (third newstate))

(format t "~%NEW STATE: ~3a" newstate))

)))))

(checkB))

;PUT C TO B

(defun putCtoB () (let((x(first state))(y(second state))(z(third state)))

(cond((>= y 4) (format t"~%B IS FULL"))

((<= z 0) (format t"~%C IS EMPTY"))

((and (> z 0) (< y 4))

(let ((local (- 4 y)))

(if (or (> y local)(= y local))

(let ((newstate (list x 4 (- z (- 4 y)))))

(setf (first state) (first newstate))

(setf (second state) (second newstate))

(setf (third state) (third newstate))

(format t "~%NEW STATE: ~3a" newstate))

(let ((newstate (list x (+ y z) 0)))

(setf (first state) (first newstate))

(setf (second state) (second newstate))

(setf (third state) (third newstate))

(format t "~%NEW STATE: ~3a" newstate))

)))))

(checkB))

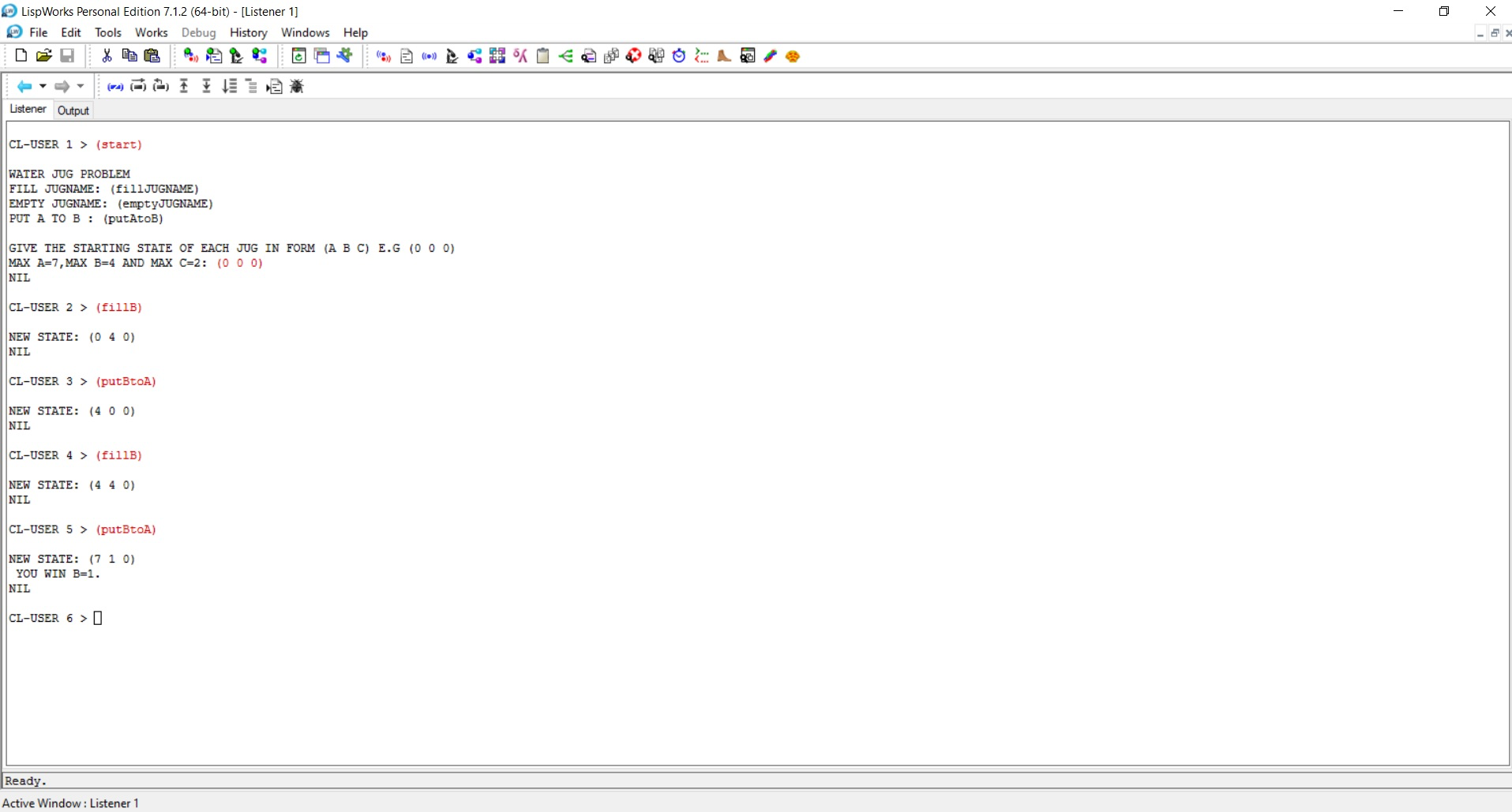
;CHECK IF B=1

(defun checkB () (let ((y(second state)))

(if (= y 1) (format t "~% YOU WIN B=1."

))))

**Output:**



* 1. **Aim- Write a LISP program that determines whether an integer is prime.**

(defun is-prime (n &optional (d (- n 1)))(or (=d1)

(and(/= (remn d)0)

(is-primen(-d1)))))

(if (equal (is-prime 3) T)(print "It is prime")(print"Itisnotprime")

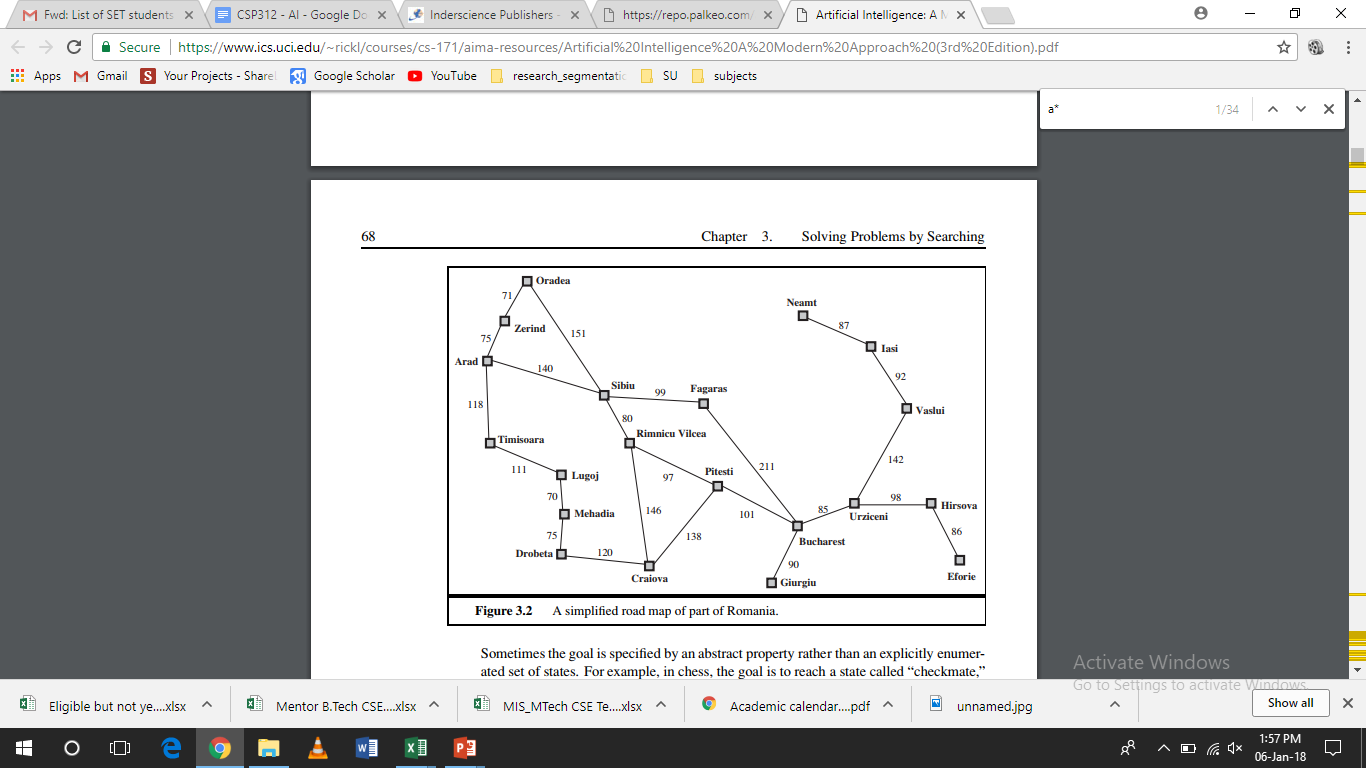
)

**OUTPUT** :

****

**1.4 Aim- Refer following figure as map with distance details, Write a program in your preferred language to generate path from ARAD to BUCHREST, analyze result obtained by:**

1. **Depth First Search**
2. **Breadth First Search**
3. **Uniform Cost Search**

****

**(A.) DFS**

class node:

def init (self, name):self.explored = 0self.name = nameself.neighbours ={}

nodes = {}start='Arad'

goal = 'Bucharest'explored = []frontier=[]

path =[]

f=open("Data(DFS).txt","r")forlinein f:

line=line.strip()

node1, node2, distance = line.split(",")ifnode1 notin nodes:

nodes[node1] = node(node1)ifnode2 notin nodes:

nodes[node2] = node(node2)nodes[node1].neighbours[node2]=distancenodes[node2].neighbours[node1]=distance

def initFrontier():frontier.append(start)nodes[start].parent=''

defchoosenode():node = frontier.pop()iftestgoal(node):

print (goal)

pathcost =calpath(goal)

print ("path cost is {}".format(pathcost))print ("path selected is {}".format(path))exit()

return node

def calpath(cnode):path.append(cnode)

if nodes[cnode].parent == '':return 0

else:

cparent = nodes[cnode].parentpathcost=

calpath(cparent)+int(nodes[cnode].neighbours[cparent])return pathcost

def testgoal(curnode):ifcurnode==goal:

return TruereturnFalse

def graphsearch():ifnotfrontier:

print ("failure")exit()

curnode = choosenode()nodes[curnode].explored = 1explored.append(curnode)

for neighbour in nodes[curnode].neighbours.keys():ifneighbour in frontier:

continue

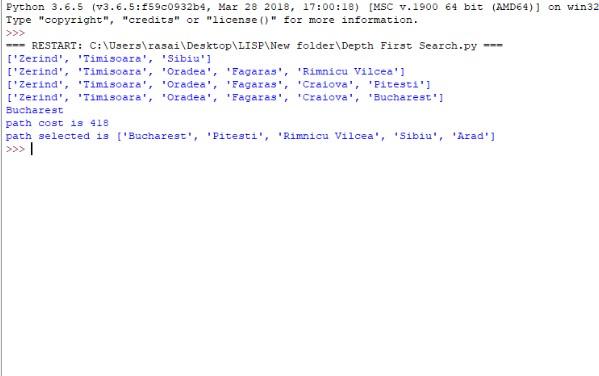
ifneighbourinexplored:continue

frontier.append(neighbour)nodes[neighbour].parent=curnode

initFrontier()while True:

graphsearch()print(frontier)

**OUTPUT:**



**(B.) BFS**

graph={

'arad':['sibiu','zerind','timisoara'],

'sibiu':['oradea','fagaras','rimnicu'],

'zerind':['arad','oradea'],

'timisoara':['arad','lugoj'],

'oradea':['zerind','sibiu'],

'fagaras':['sibiu','bucharest']

}

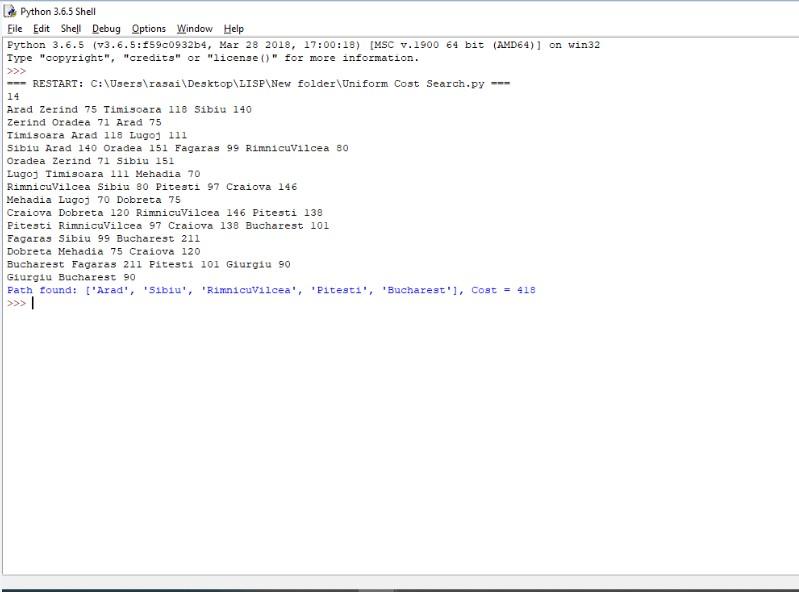
def bfs(graph, start, end):queue = []queue.append([start])whilequeue:

path = queue.pop(0)node=path[-1]

if node == end:return path

for adjacent in graph.get(node, []):new\_path = list(path)new\_path.append(adjacent)queue.append(new\_path)

print(bfs(graph,'arad','bucharest'))



**(C.) UCS**

importqueueasQ

def search(graph, start, end):ifstartnotingraph:

raise TypeError(str(start)+'notfoundingraph!')return

ifendnotingraph:

raiseTypeError(str(end)+'notfoundingraph!')return

queue = Q.PriorityQueue()queue.put((0,[start]))

while not queue.empty():node =queue.get()

current= node[1][len(node[1]) -1]

ifendinnode[1]:

print("Path found: " + str(node[1]) + ", Cost = " + str(node[0]))break

cost= node[0]

for neighbor in graph[current]:temp = node[1][:]temp.append(neighbor)

queue.put((cost+graph[current][neighbor],temp))

defreadGraph():lines = int( input() )graph={}

for line in range(lines):line=input()

tokens = line.split()node = tokens[0]graph[node]={}

fori inrange(1,len(tokens)-1,2):

#print(node,tokens[i],tokens[i+1])

#graph.addEdge(node,tokens[i],int(tokens[i+1]))graph[node][tokens[i]]=int(tokens[i+1])

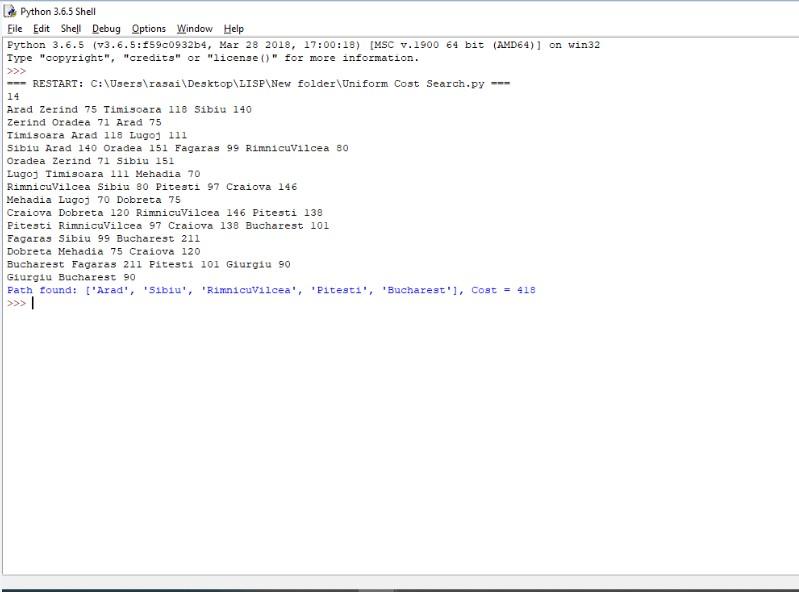
returngraph

defmain():

graph = readGraph()search(graph,'Arad','Bucharest')

if name\_\_==" main ":main()

**OUTPUT-**



**1.5 Aim- Write a program in your preferred language to generate steps to solve the Tower of Hanoi problem.**

defTowerOfHanoi(n,from\_rod,to\_rod,aux\_rod):ifn==1:

print("Movedisk1 fromrod",from\_rod,"torod",to\_rod)return

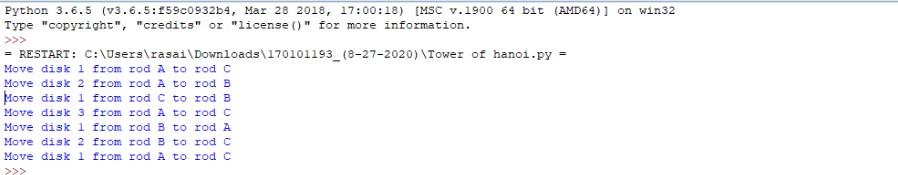
TowerOfHanoi(n-1,from\_rod,aux\_rod,to\_rod)

print("Movedisk",n,"fromrod",from\_rod,"torod",to\_rod)TowerOfHanoi(n-1,aux\_rod,to\_rod,from\_rod)

n =3

TowerOfHanoi(n,'A','C','B')

**Output:**



**1.6 Aim- Write a program in your preferred language to solve the 8 puzzle Problem-using A\* algorithm.**

class Node:

def init\_\_(self,data,level,fval):self.data=data

self.level = levelself.fval=fval

def generate\_child(self):

x,y=self.find(self.data,'\_')

val\_list=[[x,y-1],[x,y+1],[x-1,y],[x+1,y]]children=[]

for i inval\_list:

child=self.shuffle(self.data,x,y,i[0],i[1])if childisnotNone:

child\_node=Node(child,self.level+1,0)children.append(child\_node)

return children

def shuffle(self,puz,x1,y1,x2,y2):

if x2>=0 and x2<len(self.data) and y2>=0 and y2<len(self.data):temp\_puz=[]

temp\_puz = self.copy(puz)temp=temp\_puz[x2][y2]

temp\_puz[x2][y2]=temp\_puz[x1][y1]temp\_puz[x1][y1]=temp

return temp\_puzelse:

return None

def copy(self,root):

"""Copyfunctionto createasimilarmatrixofthegivennode"""temp=[]

for i in root:t=[]

for j in i:t.append(j)

temp.append(t)returntemp

def find(self,puz,x):

for i in range(0,len(self.data)):forjin range(0,len(self.data)):

if puz[i][j]==x:returni,j

classPuzzle:

def init\_\_(self,size):

self.n = sizeself.open = []self.closed=[]

def accept(self):

puz=[]

for i in range(0,self.n):temp = input().split(" ")puz.append(temp)

return puz

deff(self,start,goal):

returnself.h(start.data,goal)+start.leveldef h(self,start,goal):

temp =0

for i in range(0,self.n):for jinrange(0,self.n):

ifstart[i][j]!=goal[i][j]andstart[i][j]!='\_':temp+=1

return temp

def process(self):

print("Enter the start state matrix \n")start=self.accept()

print("Enter the goal state matrix \n")goal=self.accept()

start = Node(start,0,0)start.fval = self.f(start,goal)self.open.append(start)print("\n\n")

whileTrue:

cur = self.open[0]print("")

print("|")

print("|")

print(" \\\'/ \n")foriincur.data:

for j in i:print(j,end="")

print("")

if(self.h(cur.data,goal)==0):break

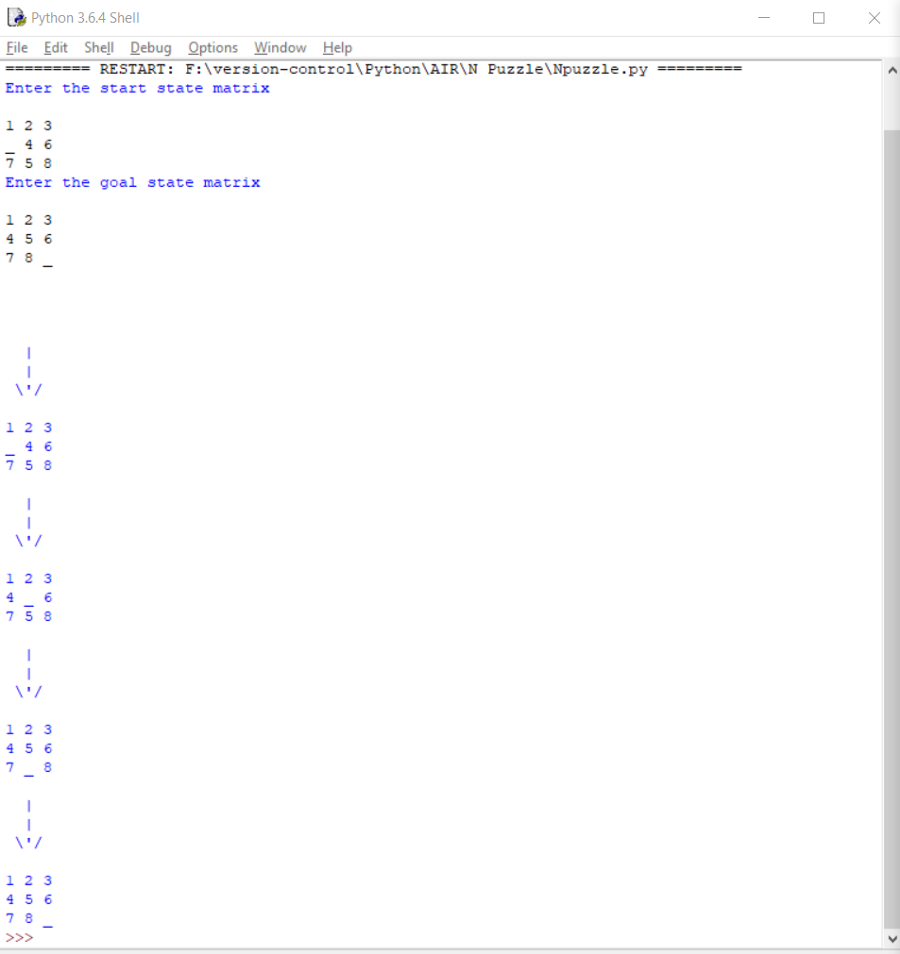
for i in cur.generate\_child():i.fval = self.f(i,goal)self.open.append(i)

self.closed.append(cur)delself.open[0]

self.open.sort(key = lambda x:x.fval,reverse=False)puz=Puzzle(3)

puz.process()

**Output-**



**1.7 Aim- Write a PROLOG program to Program to categorize animal characteristics.**

predicates

small(symbol)

large(symbol)

color(symbol,symbol)

clauses

small(rat).

small(cat).

large(lion).

color(dog,black).

color(rabbit,white).

color(X,dark):- color(X,black);

color(X,brown).

**Output:-**

Graphical user interface, application

Description automatically generated

**1.8 Aim- Write a PROLOG program that answers questions about family members and relationships including predicates and rules which define sister, brother, father, mother, grandchild, grandfather and uncle. The program should be able to answer queries such as the following:**

**father(x, Amit)**

**grandson(x, y)**

**uncle (sumit, puneet)**

**mother (anita, x)**

female(pam).

female(liz).

female(pat).

female(ann).

male(jim).

male(bob).

male(tom).

male(peter).

parent(pam,bob).

parent(tom,bob).

parent(tom,liz).

parent(bob,ann).

parent(bob,pat).

parent(pat,jim).

parent(bob,peter).

parent(peter,jim).

mother(X,Y):- parent(X,Y),female(X).

father(X,Y):- parent(X,Y),male(X).

haschild(X):- parent(X,\_).

sister(X,Y):- parent(Z,X),parent(Z,Y),female(X),X\==Y.

brother(X,Y):-parent(Z,X),parent(Z,Y),male(X),X\==Y.

**Output-**

| ?- [family].

compiling D:/TP Prolog/Sample\_Codes/family.pl for byte code...

D:/TP Prolog/Sample\_Codes/family.pl compiled, 23 lines read - 3088 bytes written, 9 ms

yes

| ?- parent(X,jim).

X = pat ? ;

X = peter

yes

| ?-

mother(X,Y).

X = pam

Y = bob ? ;

X = pat

Y = jim ? ;

no

| ?- haschild(X).

X = pam ? ;

X = tom ? ;

X = tom ? ;

X = bob ? ;

X = bob ? ;

X = pat ? ;

X = bob ? ;

X = peter

yes

| ?- sister(X,Y).

X = liz

Y = bob ? ;

X = ann

Y = pat ? ;

X = ann

Y = peter ? ;

X = pat

Y = ann ? ;

X = pat

Y = peter ? ;

(16 ms) no

| ?-