**Practical No 2**

|  |
| --- |
| **Aim:** Implement 3 missionaries and 3 cannibals problem depicting appropriate graph. Use A\* algorithm. |

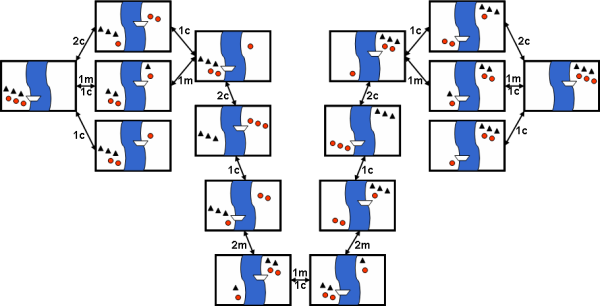
**Software Requirement:-**

**Theory: -** The Missionaries and Cannibals problem is a classic AI puzzle that can be defined as follows:

On one bank of a river are three missionaries and three cannibals. There is one boat available that can hold up to two people and those they would like to use to cross the river. If the cannibals ever outnumber the missionaries on either of the river’s banks, the missionaries will get-eaten. How can the boat be used to safely carry all the missionaries and cannibals across the river? The initial state is shown to the right here, where black triangles represent missionaries and red circles represent cannibals.



This problem can be solved by searching for a solution, which is a sequence of actions that leads from the initial state to the goal state. The goal state is effectively a mirror image of the initial state. The complete search space is shown in Graph below Arrows in graph represent state transitions and are labelled with actions, e.g. 2c represents the action of two cannibals crossing the river. The initial state is shown again on the left, whereas the goal state is all the way to the right.



**Example:-**

**Steps Moves Initial State  *Final state***

0 Initial setup: MMMCCC B -

1 Two cannibals cross over: MMMC B CC

2 One comes back: MMMCC B C

3 Two cannibals go over again: MMM B CCC

4 One comes back: MMMC B CC

5 Two missionaries cross: MC B MMCC

6 A missionary & cannibal return: MMCC B MC

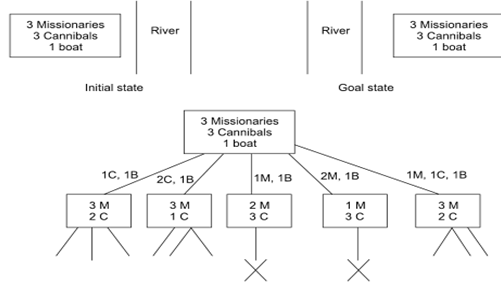
7 Two missionaries cross again: CC B MMMC

8 A cannibal returns: CCC B MMM

9 Two cannibals cross: C B MMMCC

10 One returns: CC B MMMC

11 And brings over the third: - B MMMCCC



**Algorithm**:-

1. Place the starting node s on the queue. If the queue is empty, return failure and stop.
2. If the first element on the queue is a goal node g, return success and stop. Otherwise,
3. Remove and expand the first element from the queue and place all the children at the
4. End of the queue in any order.
5. Return to step 2

**Program Code**:-

/\*AIM: Implement 3 missionaries and 3 cannibals problem depicting appropriate graph.

\* Use A\* algorithm.

\*/

import java.util.Vector;

import java.util.LinkedList;

public class MandC

{

static State initial\_state=new State(3,3,'L',0,0);

public static void main(String[] args)

{

Node node=solve(initial\_state);

System.out.println();

if(node==null)

System.out.println("No solution exists.");

else

{

System.out.println("The solution is:\n");

node.printBackTrace();

}

System.out.println();

}

static class State

{

int ml,cl;

int mr,cr;

char boat;

public State(int ml,int cl,char boat,int mr,int cr)

{

this.ml=ml;

this.cl=cl;

this.boat=boat;

this.mr=mr;

this.cr=cr;

}

public boolean goal\_test()

{

return ml==0 && cl==0;

}

public String toString()

{

return "("+ml+""+cl+""+boat+""+mr+""+cr+")";

}

public boolean equals(Object obj)

{

if(!(obj instanceof State))

return false;

State s=(State)obj;

return(s.ml==ml && s.cl==cl && s.boat==boat && s.cr==cr && s.mr==mr);

}

public Vector successor\_function()

{

Vector v=new Vector();

if(boat=='L')

{

testAndAdd(v,new StateActionPair(new State(ml-2,cl,'R',mr+2,cr),new Action("Two missioneries cross left to right.")));

testAndAdd(v,new StateActionPair(new State(ml,cl-2,'R',mr,cr+2),new Action("Two cannibals cross left to right.")));

testAndAdd(v,new StateActionPair(new State(ml-1,cl-1,'R',mr+1,cr+1),new Action("One missionary and one cannibal cross left to right.")));

testAndAdd(v,new StateActionPair(new State(ml-1,cl,'R',mr+1,cr),new Action("One missionary cross left to right.")));

testAndAdd(v,new StateActionPair(new State(ml,cl-1,'R',mr,cr+1),new Action("One cannibal cross left to right.")));

}

else

{

testAndAdd(v,new StateActionPair(new State(ml+2,cl,'L',mr-2,cr),new Action("Two missionaries cross right to left.")));

testAndAdd(v,new StateActionPair(new State(ml,cl+2,'L',mr,cr-2),new Action("Two cannibals cross right to left.")));

testAndAdd(v,new StateActionPair(new State(ml+1,cl+1,'L',mr-1,cr-1),new Action("One missionary and one cannibal cross right to left.")));

testAndAdd(v,new StateActionPair(new State(ml+1,cl,'L',mr-1,cr),new Action("One Missionary crosses right to left.")));

testAndAdd(v,new StateActionPair(new State(ml,cl+1,'L',mr,cr-1),new Action("One cannibal crosses right to left.")));

}

return v;

}

private void testAndAdd(Vector v,StateActionPair pair)

{

State state=pair.state;

if(state.ml>=0 && state.mr>=0 && state.cl>=0 && state.cr>=0 && (state.ml==0 || state.ml>=state.cl) && (state.mr==0 || state.mr>=state.cr))

v.addElement(pair);

}

}//end class State

static class Action

{

String text;

public Action(String text)

{

this.text=text;

}

public String toString()

{

return text;

}

public double cost()

{

return 1;

}

}

static class StateActionPair

{

public State state;

public Action action;

public StateActionPair(State state,Action action)

{

this.state=state;

this.action=action;

}

}

static class Node

{

public State state;

public Node parent\_node;

public Action action;

public double path\_cost;

public int depth;

public Node(State state)

{

this.state=state;

parent\_node=null;

action=new Action("Initial state");

path\_cost=0;

depth=0;

}

public Node(State state,Node parent,Action action)

{

this.state=state;

this.parent\_node=parent;

this.action=action;

this.path\_cost=action.cost()+parent.path\_cost;

this.depth=1+parent.depth;

}

public void printBackTrace()

{

if(parent\_node !=null)

parent\_node.printBackTrace();

System.out.println(" "+depth+", "+action+" ----->"+state);

}

}

public static Node solve(State initial\_state)

{

LinkedList fringe=new LinkedList();

Vector visited=new Vector();

fringe.add(new Node(initial\_state));

while(true)

{

if(fringe.isEmpty())

return null;

Node node=(Node)fringe.removeFirst();

Vector successors=node.state.successor\_function();

for(int i=0;i<successors.size();i++)

{

StateActionPair successor=(StateActionPair)successors.elementAt(i);

if(! containState(visited,successor.state))

{

Node newNode=new Node(successor.state,node,successor.action);

if(successor.state.goal\_test())

return newNode;

fringe.add(newNode);

visited.add(successor.state);

}

}

}

}

public static boolean containState(Vector visitedStates,State state)

{

for(int i=0;i<visitedStates.size();i++)

{

if(visitedStates.elementAt(i).equals(state))

return true;

}

return false;

}

}

/\*OUTPUT:

\*

The solution is:

0, Initial state ----->(33L00)

1, Two cannibals cross left to right. ----->(31R02)

2, One cannibal crosses right to left. ----->(32L01)

3, Two cannibals cross left to right. ----->(30R03)

4, One cannibal crosses right to left. ----->(31L02)

5, Two missioneries cross left to right. ----->(11R22)

6, One missionary and one cannibal cross right to left. ----->(22L11)

7, Two missioneries cross left to right. ----->(02R31)

8, One cannibal crosses right to left. ----->(03L30)

9, Two cannibals cross left to right. ----->(01R32)

10, One Missionary crosses right to left. ----->(11L22)

11, One missionary and one cannibal cross left to right. ----->(00R33)

**Conclusion**: - Thus we have Implement 3 missionaries and 3 cannibals problem Using A\* algorithm.