Elective 1 & 2 Lab

(Automata & Artificial Intelligence)

COE-408

By - Divjot Singh
262/CO/12
COE-1

Index

#	Experiment	Signature
1	Water Jug Problem (8, 5, 3 Liter Jugs)	
2	Elevator (3 Story)	
3	Highway Traffic Light Controller	
4	One Item Vending Machine	
5	Push Down Automata for given language	
6	Palindrome Detector Turing Machine	
7	Unary Multiplier Turing Machine	
8	Travelling Salesman	
9	Chess Problem - 8 Queen Puzzle	
10	Tic Tac Toe	

Experiment 1: Water Jug Problem

Analysis:

This famous problem has 3 jugs of 3, 5 and 8 liter jugs, and the goal is to have 4 liters of water in 5-liter and 8-liter jug, from the initial state where only 8 liter jug is completely filled while others are empty. The conditions here are that we can't measure water, and can only transfer from one jug to another.

Algorithm:

The algorithm is fairly simply here, we'll use generate and test paradigm to solve it. We will have a graph of states, each of which denote the status of all 3 jugs.

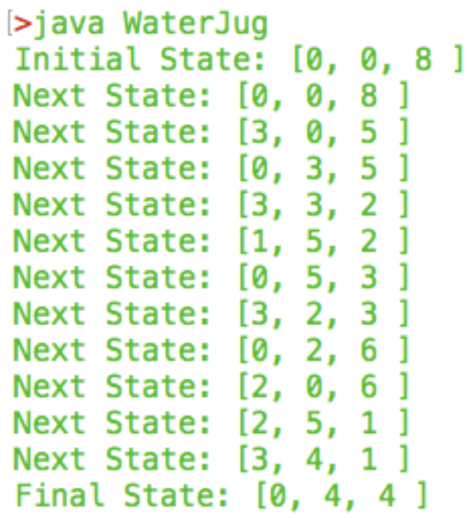
Generate: We will generate new states from initial state by trying all combinations of pouring water from one jug to another. Then we will test these states, and recursively call generate on them. We also need to ensure that same state isn't computed twice, so we will use a matrix *status* which will help in this bookkeeping.

Test: This is the easiest part for the algorithm, we just check whether the current state is equal to final state. If it is, then we stop our algorithm and print the path, else we continue testing for other generated states.

Code:

```
import java.util.*;

public class WaterJug {
  static State initialState = new State(0, 0, 8);
  static State finalState = new State(0, 4, 4);
  static LinkedList<State> solution = new LinkedList<State>();
  static int status[][] = new int[4][6];
  static final int SOLUTION = 1;
  static final int UNCHECKED = 0;
  static final int CHECKED = -1;

  public static void main (String args[]) {
    System.out.println("Initial State: " + initialState.toString());
    if(solveFrom(initialState)) {
      for(State s: solution) {
        System.out.println(
          (s.equalsTo(finalState) ? "Final State: " : "Next State: ") + s.toString());
      }
    }
  }
  public static boolean solveFrom (State s) {
    boolean foundSolution = false;

    if(status[s._3][s._5] == UNCHECKED) {
      if(status[s._3][s._5] == SOLUTION || s.equalsTo(finalState)) {
        foundSolution = true;
      } else {
        status[s._3][s._5] = CHECKED;

        if(s._3 > 0) {
          foundSolution = solveFrom (new State(s._3 - Math.min(s._3, 5 - s._5), s._5 +
Math.min(s._3, 5 - s._5)))
            || solveFrom (new State(s._3 - Math.min(s._3, 8 - s._8), s._5));
        }

        if (!foundSolution && s._5 > 0) {
          foundSolution = solveFrom (new State(s._3 + Math.min(s._5, 3 - s._3), s._5 -
Math.min(s._5, 3 - s._3)))
            || solveFrom (new State(s._3, s._5 - Math.min(s._5, 8 - s._8)));
        }


        if (!foundSolution && s._8 > 0) {
          foundSolution = solveFrom (new State(s._3 + Math.min(s._8, 3 - s._3), s._5))
            || solveFrom (new State(s._3, s._5 + Math.min(s._8, 5 - s._5)));
        }
      }

      if(foundSolution) {
        solution.addFirst(s);
        status[s._3][s._5] = SOLUTION;
      }
    }
    return (status[s._3][s._5] == SOLUTION) || foundSolution;
  }
}
class State {
  int _3, _5, _8;
  State(int v3, int v5) {
    this(v3, v5, 8 - v3 - v5);
  }
  State(int v3, int v5, int v8) {
    _3 = v3;
    _5 = v5;
    _8 = v8;
  }
  boolean equalsTo(State s) {
    return s._3 == _3 && s._5 == _5 && s._8 == _8;
  }
  public String toString() {
    return "[" + _3 + ", " + _5 + ", " + _8 + " ]";
  }
}
```

Output:



```
>java WaterJug
 Initial State: [0, 0, 8 ]
Next State: [0, 0, 8 ]
Next State: [3, 0, 5 ]
Next State: [0, 3, 5 ]
Next State: [3, 3, 2 ]
Next State: [1, 5, 2 ]
Next State: [0, 5, 3 ]
Next State: [3, 2, 3 ]
Next State: [0, 2, 6 ]
Next State: [2, 0, 6 ]
Next State: [2, 5, 1 ]
Next State: [3, 4, 1 ]
 Final State: [0, 4, 4 ]
```

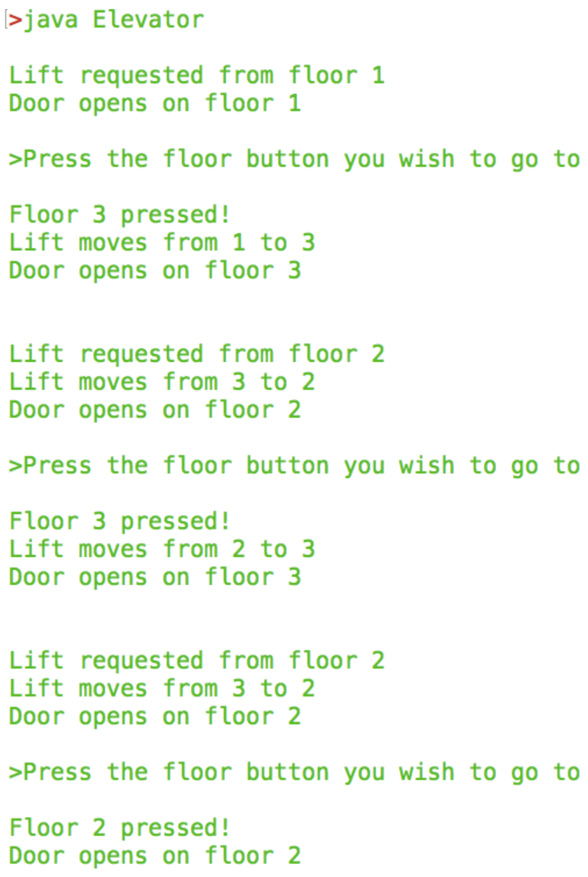
Experiment 2: Elevator (3 Story)

Code:

```
import java.util.*;
import java.lang.*;

public class Elevator {

  static int currentFloor = 1;

  public static void main (String args[]) {
    requestFromFloor(1);
    requestToFloor(3);
    requestFromFloor(2);
    requestToFloor(3);
    requestFromFloor(2);
    requestToFloor(2);
  }

 private static void gotoFloor(int floor) {
    if(floor != currentFloor) {
      System.out.println("Lift moves from " + currentFloor + " to " + floor);
      sleep(2000 * Math.abs(currentFloor - floor));
      currentFloor = floor;
    }
    System.out.println("Door opens on floor " + currentFloor);
    System.out.println();
  }

  public static void requestToFloor(int floor) {
    System.out.println("Floor " + floor + " pressed!");
    gotoFloor(floor);
  }

  public static void requestFromFloor(int floor) {
    System.out.println();
    System.out.println("Lift requested from floor " + floor);
    gotoFloor(floor);
    System.out.println(">Press the floor button you wish to go to");
    System.out.println();
  }
```

```java
public static boolean isValidFloor(int floor) {
    return floor == 1 || floor == 2 || floor == 3;
  }

  public static void sleep(int value) {
    try { Thread.sleep(value); } catch (InterruptedException e) {
      System.out.println(e.getMessage());
    }
  }
}
```

Output:



```
>java Elevator

Lift requested from floor 1
Door opens on floor 1

>Press the floor button you wish to go to

Floor 3 pressed!
Lift moves from 1 to 3
Door opens on floor 3


Lift requested from floor 2
Lift moves from 3 to 2
Door opens on floor 2

>Press the floor button you wish to go to

Floor 3 pressed!
Lift moves from 2 to 3
Door opens on floor 3


Lift requested from floor 2
Lift moves from 3 to 2
Door opens on floor 2

>Press the floor button you wish to go to

Floor 2 pressed!
Door opens on floor 2
```

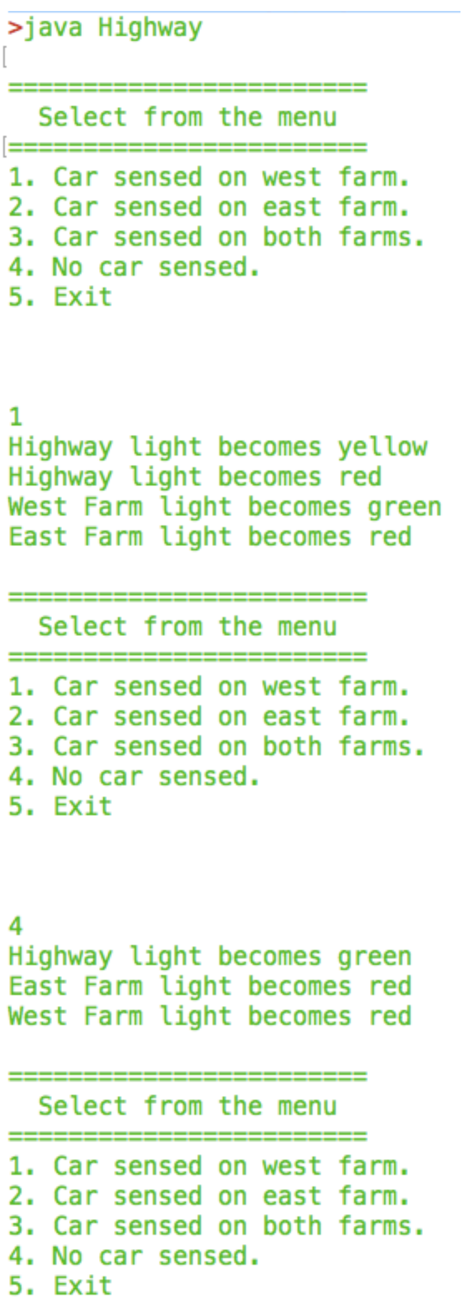
Experiment 3: Highway Traffic Light Controller

Code:

```
import java.util.*;

public class Highway {
  public static void main (String args[]) {
    Scanner sc = new Scanner(System.in);
    int choice;

    TrafficLight highway = new TrafficLight(TrafficLightStatus.green);
    TrafficLight westFarm = new TrafficLight(TrafficLightStatus.red);
    TrafficLight eastFarm = new TrafficLight(TrafficLightStatus.red);

    do {
      displayMenu();
      switch (choice = sc.nextInt()) {
        case 1:
          if(highway.currentStatus != TrafficLightStatus.red) {
            System.out.println("Highway light becomes yellow");
            highway.set(TrafficLightStatus.yellow);
            highway.set(TrafficLightStatus.red, 5);
          }
          System.out.println("Highway light becomes red");
          System.out.println("West Farm light becomes green");
          System.out.println("East Farm light becomes red");
          westFarm.set(TrafficLightStatus.green);
          eastFarm.set(TrafficLightStatus.red);
          break;
        case 2:
          if(highway.currentStatus != TrafficLightStatus.red) {
            System.out.println("Highway light becomes yellow");
            highway.set(TrafficLightStatus.yellow);
            highway.set(TrafficLightStatus.red, 5);
          }
          System.out.println("Highway light becomes red");
          System.out.println("West Farm light becomes red");
          System.out.println("East Farm light becomes green");
          westFarm.set(TrafficLightStatus.red);
          eastFarm.set(TrafficLightStatus.green);
          break;
        case 3:
          if(highway.currentStatus != TrafficLightStatus.red) {
            System.out.println("Highway light becomes yellow");
            highway.set(TrafficLightStatus.yellow);
            highway.set(TrafficLightStatus.red, 5);
          }
          westFarm.set(TrafficLightStatus.red);
          System.out.println("Highway light becomes red");
          System.out.println("West Farm light becomes red");
          System.out.println("East Farm light becomes green");
          eastFarm.set(TrafficLightStatus.green);
          westFarm.set(TrafficLightStatus.green, 20);
          System.out.println("West Farm light becomes green");
          System.out.println("East Farm light becomes red");
          eastFarm.set(TrafficLightStatus.red);
          break;
        case 4:
          if(highway.currentStatus != TrafficLightStatus.green) {
            System.out.println("Highway light becomes green");
            System.out.println("East Farm light becomes red");
            System.out.println("West Farm light becomes red");
          }
          break;
        case 5: break;
        default: System.out.println("Invalid option. Pick from `1`, `2`, `3`, `4` or `5`");
      }
    } while (choice != 5);
  }
  static void displayMenu () {
    System.out.println("\n========================");
    System.out.println("  Select from the menu  ");
    System.out.println("========================");
    System.out.println("1. Car sensed on west farm.");
    System.out.println("2. Car sensed on east farm.");
    System.out.println("3. Car sensed on both farms.");
    System.out.println("4. No car sensed.");
    System.out.println("5. Exit");
    System.out.println("\n\n");
  }
}
```

```
class TrafficLight {
  TrafficLightStatus currentStatus;
  TrafficLight(TrafficLightStatus s) { currentStatus = s; }
  public void set (TrafficLightStatus newStatus) { currentStatus = newStatus; }
  public void set (TrafficLightStatus newStatus, int afterTheseManySeconds) {
    try {
      Thread.sleep(1000 * afterTheseManySeconds);
      set(newStatus);
    } catch (InterruptedException e) {
      System.out.println (e.getMessage());
    }
  }
}
enum TrafficLightStatus { red, yellow, green }
```

Output:



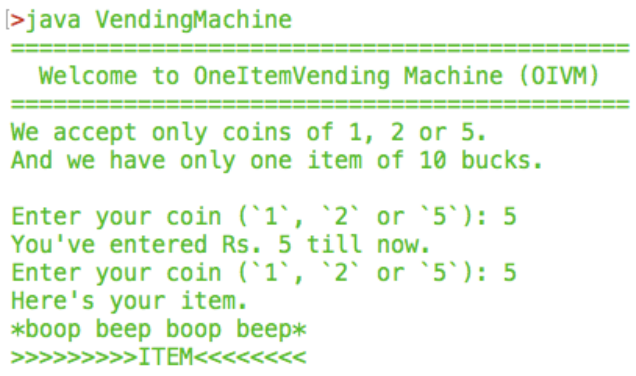
```
>java Highway

========================
  Select from the menu
========================
1. Car sensed on west farm.
2. Car sensed on east farm.
3. Car sensed on both farms.
4. No car sensed.
5. Exit



1
Highway light becomes yellow
Highway light becomes red
West Farm light becomes green
East Farm light becomes red

========================
  Select from the menu
========================
1. Car sensed on west farm.
2. Car sensed on east farm.
3. Car sensed on both farms.
4. No car sensed.
5. Exit



4
Highway light becomes green
East Farm light becomes red
West Farm light becomes red

========================
  Select from the menu
========================
1. Car sensed on west farm.
2. Car sensed on east farm.
3. Car sensed on both farms.
4. No car sensed.
5. Exit
```

Experiment 4: One Item Vending Machine

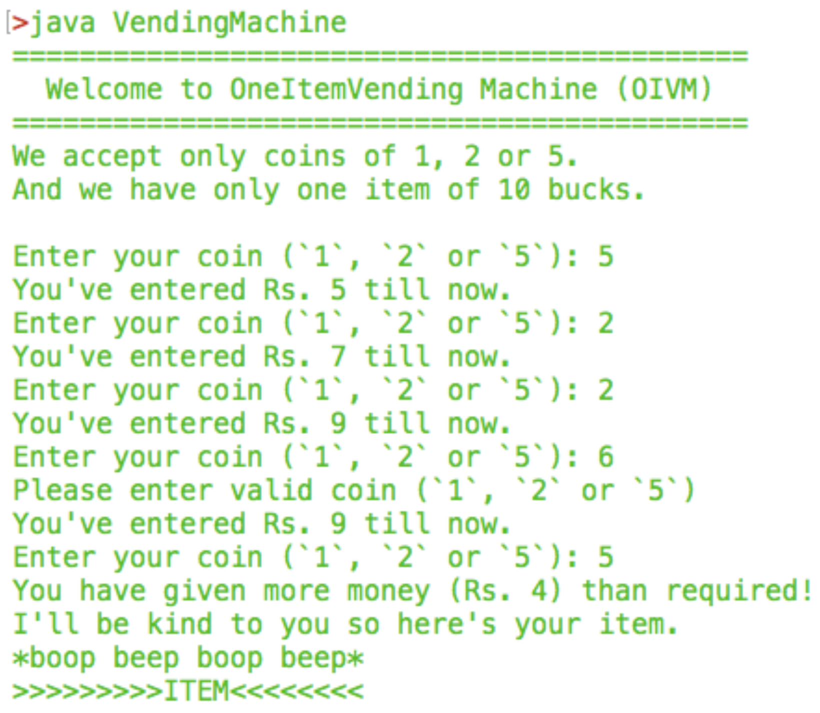
Code:

```java
import java.util.*;

public class VendingMachine {
  public static void main (String args[]) {
    Scanner sc = new Scanner(System.in);
    int currentState = 0;
    int input;

    System.out.println(
"================================================"
);
    System.out.println("  Welcome to OneItemVending Machine (OIVM)  ");
    System.out.println("
");
    System.out.println("We accept only coins of 1, 2 or 5.");
    System.out.println("And we have only one item of 10 bucks.\n");

    do {
      if(currentState != 0) {
        System.out.println("You've entered Rs. " + currentState + " till now.");
      }

      System.out.print("Enter your coin (`1`, `2` or `5`): ");

      if(isValidInput(input = sc.nextInt())) {
        currentState += input;
      } else {
        System.out.println("Please enter valid coin (`1`, `2` or `5`)");
      }

    } while(currentState < 10);

    if(currentState == 10) { System.out.println("Here's your item."); } else {
      System.out.println("You have given more money (Rs. " +
          (currentState - 10) + ") than required!");
      System.out.println("I'll be kind to you so here's your item.");
    }
    System.out.println("*boop beep boop beep*");
    System.out.println(">>>>>>>>>ITEM<<<<<<<<");
  }
  public static boolean isValidInput (int i) { return i == 1 || i == 2 || i == 5; }
}
```

Output:



```
[>java VendingMachine
============================================
  Welcome to OneItemVending Machine (OIVM)
============================================
We accept only coins of 1, 2 or 5.
And we have only one item of 10 bucks.

Enter your coin (`1`, `2` or `5`): 5
You've entered Rs. 5 till now.
Enter your coin (`1`, `2` or `5`): 5
Here's your item.
*boop beep boop beep*
>>>>>>>>>ITEM<<<<<<<<
```

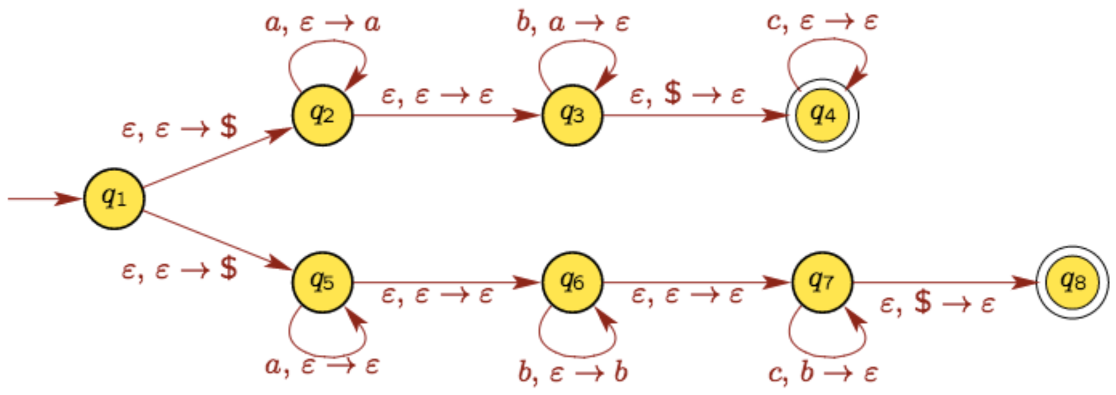


```
[>java VendingMachine
============================================
  Welcome to OneItemVending Machine (OIVM)
============================================
We accept only coins of 1, 2 or 5.
And we have only one item of 10 bucks.

Enter your coin (`1`, `2` or `5`): 5
You've entered Rs. 5 till now.
Enter your coin (`1`, `2` or `5`): 2
You've entered Rs. 7 till now.
Enter your coin (`1`, `2` or `5`): 2
You've entered Rs. 9 till now.
Enter your coin (`1`, `2` or `5`): 6
Please enter valid coin (`1`, `2` or `5`)
You've entered Rs. 9 till now.
Enter your coin (`1`, `2` or `5`): 5
You have given more money (Rs. 4) than required!
I'll be kind to you so here's your item.
*boop beep boop beep*
>>>>>>>>>ITEM<<<<<<<<
```

Experiment 5: Push Down Automata for given language

Q: PDA for L = ($a^i b^j c^k$), i = j or j = k

We will use non-deterministic PDA to solve this question, The PDA will have two non-deterministic branches:



We can write the transition as follows:

Branch 1 (for i = j)	Branch 2 (for j = k)
(q1, e, $) |- (q2, $)	(q1, e, $) |- (q5, $)
(q2, a, $) \|- (q2, a$)	(q5, a, $) |- (q5, $)
(q2, a, a) |- (q2, aa)	(q5, b, $) \|- (q6, b$)
(q2, b, a) |- (q3, ^)	(q6, b, b) |- (q6, bb)
(q3, b, a) |- (q3, ^)	(q6, c, b) |- (q7, ^)
(q3, c, $) |- (q4, $)	(q7, c, b) |- (q7, ^)
(q4, c, $) |- (q4, $)	(q7, e, $) |- (q8, $)
(q4, e, $) |- (q4, $)	

Experiment 6: Palindrome Detector Turing Machine

Code:

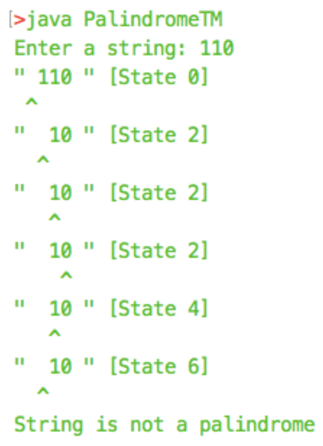
```
import java.util.*;

public class PalindromeTM {
  static Scanner sc = new Scanner (System.in);
  static int currentState = 0;
  static int pointerLocation = 1;
  static StringBuilder input = new StringBuilder(" ");

  public static void main (String args[]) {
    System.out.print ("Enter a string: ");
    input.append(sc.next() + " ");

    System.out.println ( "String is" + (isPalindrome() ? " " : " not ") + "a
palindrome");
  }

  static boolean isPalindrome () {
    boolean found = false;
    boolean halt = false;

    System.out.println("\"" + input.toString() + "\" [State " + currentState + "]");
    System.out.println(spaces(pointerLocation) + "^");

    switch (currentState) {

      case 0: // Start State
        switch(input.charAt(pointerLocation)) {
          case '0':
            currentState = 1;
            input.setCharAt(pointerLocation, ' ');
            pointerLocation++; break;
          case '1':
            currentState = 2;
            input.setCharAt(pointerLocation, ' ');
            pointerLocation++; break;
          case ' ':
            currentState = 5;
            input.setCharAt(pointerLocation, ' ');
            pointerLocation--; break;
        }
        break;
```

```
case 1: // Seen a 0, move to right end of input
 switch(input.charAt(pointerLocation)) {
  case '0':
   currentState = 1;
   input.setCharAt(pointerLocation, '0');
   pointerLocation++; break;
  case '1':
   currentState = 1;
   input.setCharAt(pointerLocation, '1');
   pointerLocation++; break;
  case ' ':
   currentState = 3;
   input.setCharAt(pointerLocation, ' ');
   pointerLocation--; break;
 }
 break;

case 2: // Seen a 1, move to right end of input
 switch(input.charAt(pointerLocation)) {
  case '0':
   currentState = 2;
   input.setCharAt(pointerLocation, '0');
   pointerLocation++; break;
  case '1':
   currentState = 2;
   input.setCharAt(pointerLocation, '1');
   pointerLocation++; break;
  case ' ':
   currentState = 4;
   input.setCharAt(pointerLocation, ' ');
   pointerLocation--; break;
 }
 break;
```

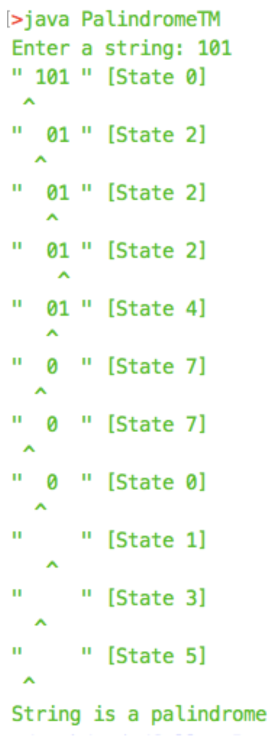
```
case 3: // Test right end of input for 0
    switch(input.charAt(pointerLocation)) {
     case '0':
       currentState = 7;
       input.setCharAt(pointerLocation, ' ');
       pointerLocation--; break;
     case '1':
       currentState = 6;
       input.setCharAt(pointerLocation, '1');
       pointerLocation--; break;
     case ' ':
       currentState = 5;
       input.setCharAt(pointerLocation, ' ');
       pointerLocation--; break;
    }
    break;

   case 4: // Test right end of input for 1
    switch(input.charAt(pointerLocation)) {
     case '0':
       currentState = 6;
       input.setCharAt(pointerLocation, '0');
       pointerLocation--; break;
     case '1':
       currentState = 7;
       input.setCharAt(pointerLocation, ' ');
       pointerLocation--; break;
     case ' ':
       currentState = 5;
       input.setCharAt(pointerLocation, ' ');
       pointerLocation--; break;
    }
    break;

       // Found a palindrome
   case 5: halt = true; found = true; break;

       // Did not find a palindrome
   case 6: halt = true; found = false; break;
```

```
      case 7: // Matched a character, move to left end of input
       switch(input.charAt(pointerLocation)) {
        case '0':
          currentState = 7;
          input.setCharAt(pointerLocation, '0');
          pointerLocation--; break;
        case '1':
          currentState = 7;
          input.setCharAt(pointerLocation, '1');
          pointerLocation--; break;
         case ' ':
          currentState = 0;
          input.setCharAt(pointerLocation, ' ');
          pointerLocation++; break;
       }
      break;
    }
    return halt ? found : isPalindrome();
  }

  // Helper function to print pointer head properly
  static String spaces (int n) {
    StringBuilder s = new StringBuilder("");
    while (n-- > 0) s.append(" ");
    return s.toString();
  }
}
```

Output:



```
>java PalindromeTM
Enter a string: 110
" 110 " [State 0]
 ^
"  10 " [State 2]
  ^
"  10 " [State 2]
   ^
"  10 " [State 2]
    ^
"  10 " [State 4]
   ^
"  10 " [State 6]
  ^
String is not a palindrome
```



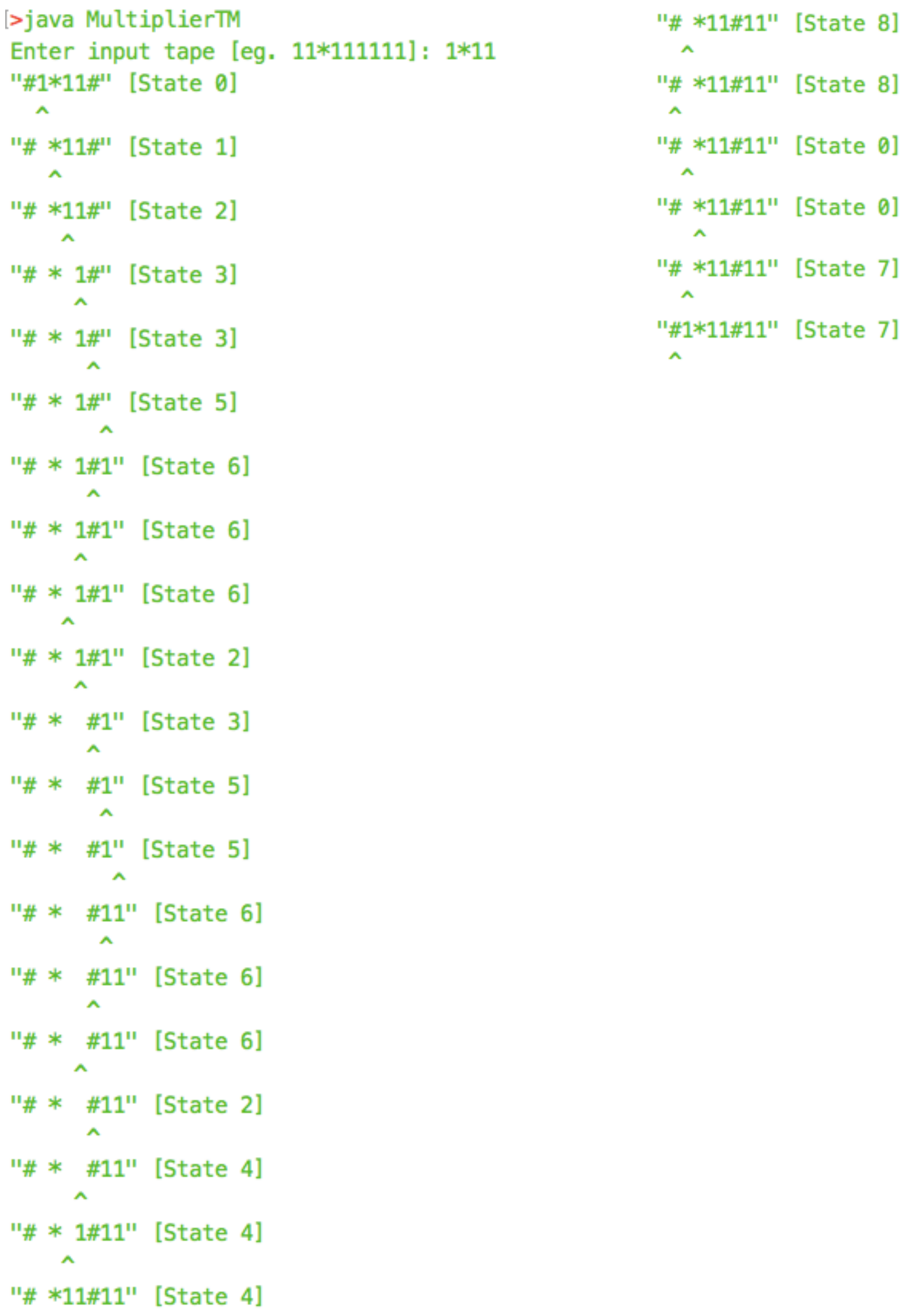
```
>java PalindromeTM
Enter a string: 101
" 101 " [State 0]
 ^
"  01 " [State 2]
  ^
"  01 " [State 2]
   ^
"  01 " [State 2]
    ^
"  01 " [State 4]
   ^
"  0  " [State 7]
  ^
"  0  " [State 7]
 ^
"  0  " [State 0]
  ^
"     " [State 1]
   ^
"     " [State 3]
  ^
"     " [State 5]
 ^
String is a palindrome
```

Experiment 7: Unary Multiplier Turing Machine

Code:

```
import java.util.*;

public class MultiplierTM {
  static Scanner sc = new Scanner (System.in);
  static int currentState = 0;
  static int pointerLocation = 1;
  static int inputLength = 1;
  static StringBuilder input = new StringBuilder("#");

  public static void main (String args[]) {
    System.out.print ("Enter input tape [eg. 11*111111]: ");
    input.append(sc.next() + "#");
    inputLength = input.length();
    input.append(spaces(1000)); // assuming max product would be 1000

    unaryMultiply();
  }

  static void unaryMultiply () {
    boolean halt = false;

    System.out.println("\"" + input.substring(0, inputLength).toString() + "\" [State
" + currentState + "]");
    System.out.println(spaces(pointerLocation + 1) + "^");

    switch (currentState) {

      case 0: // Start State, look for multiplicand
        switch(input.charAt(pointerLocation)) {
          case '1': currentState = 1; input.setCharAt(pointerLocation, ' ');
pointerLocation++; break;
          case ' ': currentState = 0; input.setCharAt(pointerLocation, ' ');
pointerLocation++; break;
          case '*': currentState = 7; input.setCharAt(pointerLocation, '*');
pointerLocation--; break;
        }
        break;
      case 1: // Multiplicand found, looking for *
        switch(input.charAt(pointerLocation)) {
          case '1': currentState = 1; input.setCharAt(pointerLocation, '1');
pointerLocation++; break;
          case '*': currentState = 2; input.setCharAt(pointerLocation, '*');
pointerLocation++; break;
        }
        break;

      case 2: // Seen a *, looking for multiplier
        switch(input.charAt(pointerLocation)) {
          case '1': currentState = 3; input.setCharAt(pointerLocation, ' ');
pointerLocation++; break;
          case ' ': currentState = 2; input.setCharAt(pointerLocation, ' ');
pointerLocation++; break;
          case '#': currentState = 4; input.setCharAt(pointerLocation, '#');
pointerLocation--; break;
        }
        break;

      case 3: // Found a multiplier, looking for right most #, end of input
        switch(input.charAt(pointerLocation)) {
          case '#': currentState = 5; input.setCharAt(pointerLocation, '#');
pointerLocation++; break;
          case '1': currentState = 3; input.setCharAt(pointerLocation, '1');
pointerLocation++; break;
        }
        break;

      case 4: // End of multiplier, restore multiplier
        switch(input.charAt(pointerLocation)) {
          case ' ': currentState = 4; input.setCharAt(pointerLocation, '1');
pointerLocation--; break;
          case '1': currentState = 4; input.setCharAt(pointerLocation, '1');
pointerLocation--; break;
          case '*': currentState = 8; input.setCharAt(pointerLocation, '*');
pointerLocation--; break;
        }
        break;
      case 8: // Multiplier restored, go back left most #, start of multiplier
       switch(input.charAt(pointerLocation)) {
        case '#': currentState = 0; input.setCharAt(pointerLocation, '#');
pointerLocation++; break;
        case ' ': currentState = 8; input.setCharAt(pointerLocation, ' ');
pointerLocation--; break;
        case '1': currentState = 8; input.setCharAt(pointerLocation, '1');
pointerLocation--; break;
       }
       break;

      case 5: // Found right most #, printing input towards the end
       switch(input.charAt(pointerLocation)) {
        case ' ': currentState = 6; input.setCharAt(pointerLocation, '1');
pointerLocation--; inputLength++; break;
        case '1': currentState = 5; input.setCharAt(pointerLocation, '1');
pointerLocation++; break;
        case '#': currentState = 5; input.setCharAt(pointerLocation, '#');
pointerLocation++; break;
       }
       break;

      case 6: // Printed partial product, looking for first blank at left
       switch(input.charAt(pointerLocation)) {
        case ' ': currentState = 2; input.setCharAt(pointerLocation, ' ');
pointerLocation++; break;
        case '1': currentState = 6; input.setCharAt(pointerLocation, '1');
pointerLocation--; break;
        case '#': currentState = 6; input.setCharAt(pointerLocation, '#');
pointerLocation--; break;
       }
       break;

      case 7: // End of multiplicand, restore it and halt
       switch(input.charAt(pointerLocation)) {
        case ' ': currentState = 7; input.setCharAt(pointerLocation, '1');
pointerLocation--; break;
        case '#': halt = true; break;
       }
       break;
     }

     if (!halt) {
      unaryMultiply();
     }
    }
    // Helper function to print pointer head properly
    static String spaces (int n) {
      StringBuilder s = new StringBuilder("");
      while (n-- > 0) s.append(" ");
      return s.toString();
    }
}
```

Output:



```
>java MultiplierTM
Enter input tape [eg. 11*111111]: 1*11
"#1*11#" [State 0]
  ^

"# *11#" [State 1]
   ^

"# *11#" [State 2]
    ^

"# * 1#" [State 3]
     ^

"# * 1#" [State 3]
      ^

"# * 1#" [State 5]
       ^

"# * 1#1" [State 6]
      ^

"# * 1#1" [State 6]
     ^

"# * 1#1" [State 6]
    ^

"# * 1#1" [State 2]
     ^

"# *  #1" [State 3]
      ^

"# *  #1" [State 5]
       ^

"# *  #1" [State 5]
        ^

"# *  #11" [State 6]
       ^

"# *  #11" [State 6]
      ^

"# *  #11" [State 6]
     ^

"# *  #11" [State 2]
      ^

"# *  #11" [State 4]
     ^

"# * 1#11" [State 4]
    ^

"# *11#11" [State 4]
```

```
"# *11#11" [State 8]
  ^

"# *11#11" [State 8]
 ^

"# *11#11" [State 0]
  ^

"# *11#11" [State 0]
   ^

"# *11#11" [State 7]
  ^

"#1*11#11" [State 7]
 ^
```

Experiment 8: Travelling Salesman

Analysis:

The problem is that a salesman has to visit a set of cities that are strongly interconnected to each other by not travelling one city more than once (except for the starting city) and minimize the total travelling cost.

Algorithm:

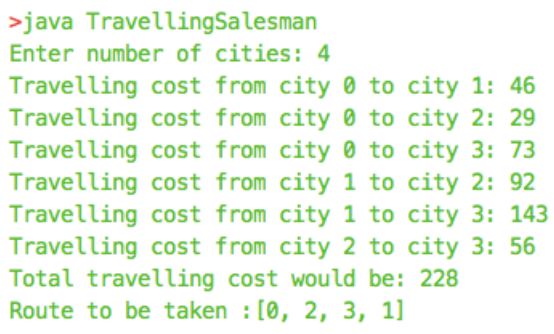
We will use a greedy heuristic based traversal search to solve the problem. It isn't the best solution but will be optimal one in practical time complexities. We augment the Depth First Search algorithm to find the nearest neighboring city, and traverse for it. While traversing we update the *visited* array, *totalCost* which has the total travelling cost and a linked list *route* which keeps track of the path taken by the salesman. The data about the costs of travelling from one city to another is saved in a matrix *costOf*. Once all nodes are visited, or we can't find any new city which is closest to the current one, we stop our algorithm.

Code:

```
import java.util.*;

public class TravellingSalesman {
  static Scanner sc = new Scanner(System.in);
  static boolean visited[];
  static int costOf[][];
  static long totalCost;
  static LinkedList <Integer> route = new LinkedList <Integer> ();

  public static void main (String args[]) {
    int nCities;

    System.out.print("Enter number of cities: ");
    nCities = sc.nextInt();

    visited = new boolean [nCities];
    costOf  = new int     [nCities][nCities];
```

```
   // Getting undirected graph
   for (int cityA = 0; cityA < nCities; cityA++) { // For each city c1...cN
     for (int cityB = cityA + 1; cityB < nCities; cityB++) { // For each city cj..cN
where j = i + 1
       System.out.print("Travelling cost from city " + cityA + " to city " + cityB + ":
");
       costOf[cityA][cityB] = sc.nextInt();
       costOf[cityB][cityA] = costOf[cityA][cityB]; // Undirected graph
     }
   }

   DFS (0); // Greedy Solution based on heuristc

   System.out.println ("Total travelling cost would be: " + totalCost);
   System.out.println ("Route to be taken :" + route.toString());
 }
 static void DFS (int cityA) {
   int closestCity = cityA;
   int costToClosestCity = Integer.MAX_VALUE;

   route.addLast (cityA);
   visited[cityA] = true;

   for (int cityB = 0; cityB < visited.length; cityB++) {
     if(!visited[cityB] && costOf[cityA][cityB] < costToClosestCity) {
       closestCity = cityB;
       costToClosestCity = costOf[cityA][cityB];
     }
   }

   if (closestCity != cityA) {
     totalCost += costToClosestCity;
     DFS (closestCity);
   }
 }
}
```

Output:



```
>java TravellingSalesman
Enter number of cities: 4
Travelling cost from city 0 to city 1: 46
Travelling cost from city 0 to city 2: 29
Travelling cost from city 0 to city 3: 73
Travelling cost from city 1 to city 2: 92
Travelling cost from city 1 to city 3: 143
Travelling cost from city 2 to city 3: 56
Total travelling cost would be: 228
Route to be taken :[0, 2, 3, 1]
```

Experiment 9: Chess Problem - 8 Queen Puzzle

Analysis:

In this problem we will have to find a configuration of chess board such that it contains 8 queens in it which don't cut each other.

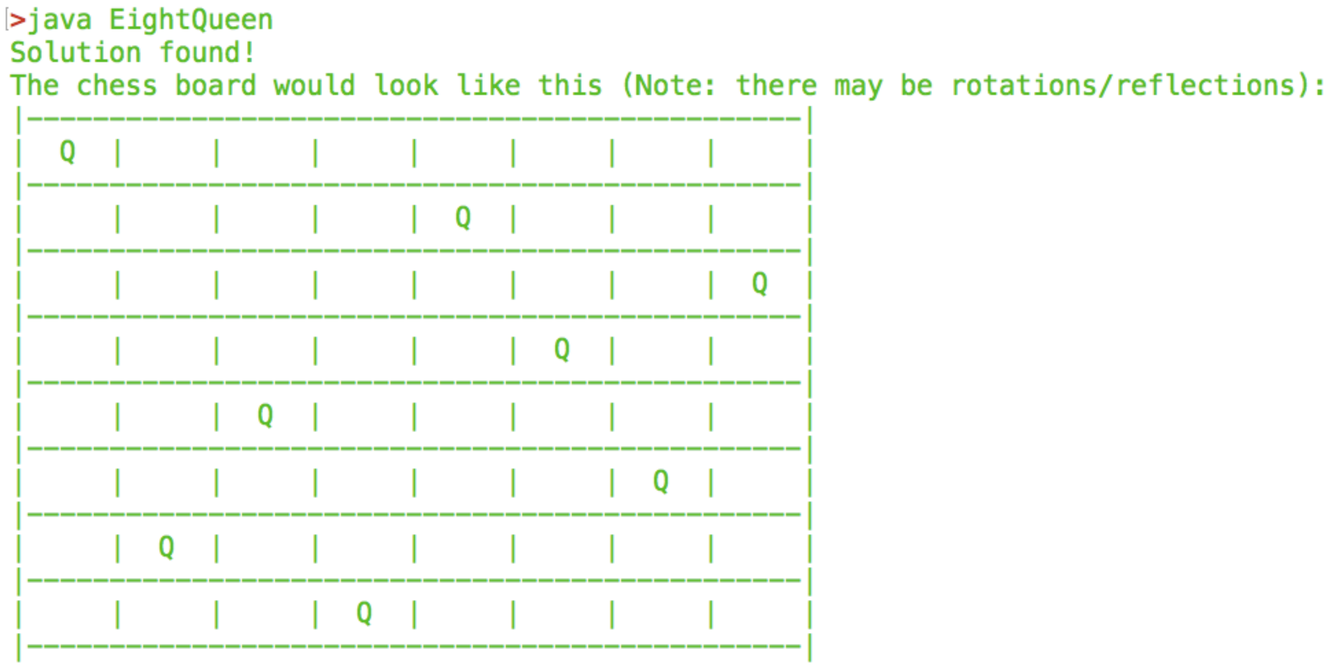
Algorithm:

We make use of backtracking to solve this problem. We generate new possible queen positions, and if it is valid or *anythingCuts* on the board, we undo the configuration, and try another new one. Once the eight queen is placed, we end our algorithm.

Code:

```
import java.util.*;

public class EightQueen {
  static boolean chessBoard[][];

  public static void main (String args[]) {
    chessBoard = new boolean[8][8];

    boolean solution = solveFor(chessBoard, 0);

    System.out.println("Solution" + (solution ? "" : " not") + " found!");

    if(solution) { printArray(chessBoard); }
  }

  public static boolean solveFor(boolean board[][], int row) {
    boolean solution = row >= board.length;

    if(isStrictlyBetween(row, -1, board.length)) {
      for(int col = 0; !solution && col < board[0].length; col++) {
        if(!board[row][col] && !anythingCuts(board, row, col)) {
          board[row][col] = true;
          board[row][col] = (solution = solveFor(board, row + 1));
        }
      }
    }
    chessBoard = board;
    return solution;
  }
  private static boolean anythingCuts(boolean board[][], int x, int y) {
    boolean doesCut = false;
    for(int i = 0; !doesCut && i < board.length; i++) // column & row
      doesCut = ((board[i][y] && i != x) || (board[x][i] && i != y));

    for(int i = 1; !doesCut && i < board[0].length; i++) // diagonal
      doesCut = ((isStrictlyBetween(x - i, -1, 8) && isStrictlyBetween(y - i, -1, 8) &&
board[x - i][y - i]) ||
        (isStrictlyBetween(x + i, -1, 8) && isStrictlyBetween(y + i, -1, 8) && board[x
+ i][y + i]) ||
        (isStrictlyBetween(x + i, -1, 8) && isStrictlyBetween(y - i, -1, 8) && board[x +
i][y - i]) ||
        (isStrictlyBetween(x - i, -1, 8) && isStrictlyBetween(y + i, -1, 8) && board[x -
i][y + i]));

    return doesCut;
  }

  private static boolean isStrictlyBetween(int x, int from, int to) {
    return x > from && x < to;
  }

  static void printArray(boolean m[][]) {
    System.out.println("The chess board would look like this (Note: there may be
rotations/reflections): ");
    StringBuilder sb = null;
    for(boolean n[] : m) {
      sb = new StringBuilder("|");
      for(boolean o: n) sb.append((o ? "  Q  " : "     ")  + "|");
      printLine(sb);
      System.out.println(sb.toString());
    }
    printLine(sb);
  }
  private static void printLine(StringBuilder sb) {
    if(sb != null)
      for(int i = 0; i < sb.length() - 2; i++)
        System.out.print((i == 0 ? "|" : "") + "-" + (i == sb.length() - 3 ? "|\n" : ""));
  }
}
```

Output:



```
>java EightQueen
Solution found!
The chess board would look like this (Note: there may be rotations/reflections):
|-----------------------------------------------|
|  Q  |     |     |     |     |     |     |     |
|-----------------------------------------------|
|     |     |     |     |  Q  |     |     |     |
|-----------------------------------------------|
|     |     |     |     |     |     |     |  Q  |
|-----------------------------------------------|
|     |     |     |     |     |  Q  |     |     |
|-----------------------------------------------|
|     |     |  Q  |     |     |     |     |     |
|-----------------------------------------------|
|     |     |     |     |     |     |  Q  |     |
|-----------------------------------------------|
|     |  Q  |     |     |     |     |     |     |
|-----------------------------------------------|
|     |     |     |  Q  |     |     |     |     |
|-----------------------------------------------|
```