```
 1 /*
 2 Keenan Brab
 3 5862990
 4 Assignment 4
 5
 6 For this program use ass4in for Euler circuit case
 7 Use test for Largest Euler Subset case
 8
 9  */
10
11 import java.util.LinkedList;
12 import java.util.Stack;
13 import java.io.*;
14 import java.util.* ;
15
16 public class Graphy {
17
18     private int V;
19     private boolean[][] adjacencyMatrix;
20
21     Queue<Integer> q = new LinkedList<>();
22     Stack<Integer> stack = new Stack<>();
23
24     private static String FILENAME = "D:\\Assignment4\\src
   \\assn4in.txt";
25     public boolean [] visited;
26     public int data = 0;
27
28     Graphy(int V,boolean[][]matrix) {
29         this.V = V;
30         this.adjacencyMatrix = new boolean[V+1][V+1];
31
32         for (int i = 1; i <= V; i++) {
33             for (int j = 1; j <= V; j++) {
34                 if (matrix[i][j] == true) {
35                     addEdge(i, j);
36                 }
37
38             }
39
40         }
41
42
43
44     }
```

```
45
46
47
48      private void addEdge(Integer u, Integer v) {
49          adjacencyMatrix[u][v] = true;
50          adjacencyMatrix[v][u] = true;
51
52
53
54      }
55      public void removeEdge (int u, int v)
56      {
57          adjacencyMatrix[u][v]  = false;
58          adjacencyMatrix[v][u] = false;
59      }
60
61
62
63      public static void main(String[] args) throws
    FileNotFoundException {
64          int p = 0;
65          FileReader file = new FileReader(FILENAME);
66          try {
67              Scanner input = new Scanner(file);
68              p = input.nextInt();
69              input.close();
70          } catch (Exception e) {
71              e.printStackTrace();
72          }
73
74
75          //P is number of vertices
76
77          boolean[][] matrix = new boolean[p+1][p+1];
78
79
80          int k = 0;
81          FileReader full = new FileReader(FILENAME);
82          try {
83              Scanner input = new Scanner(full);
84              p = input.nextInt();
85
86      //place adjmatrix from text file to array
87              for (int i = 1; i <= p; i++) {
88                  for (int j = 1; j <= p; j++) {
```

```
 89                      k = input.nextInt();
 90
 91                      if (k == 1) {
 92                          matrix[i][j] = true;
 93                      } else if (k == 0) {
 94                          matrix[i][j] = false;
 95                      } else {
 96
 97                      }
 98                  }
 99
100              }
101              input.close();
102          } catch (Exception e) {
103              e.printStackTrace();
104          }
105
106
107          boolean check;
108
109      //create graph
110          Graphy g1 = new Graphy(p,matrix);
111          check = rowCheck(matrix,p);
112          int subset;
113          if(check==true){
114              //build circuit if one exists
115              g1.findCircuit(1);
116              g1.printstack();
117          }else{
118              //find subset if exists at certain
    cardinality
119           subset = g1.largestEulerSubset(matrix,p);
120              System.out.println("Largest Euler subset: "+
    g1.data);
121          }
122
123      }
124          //Sheridans algorithm for finding largest subset
125      private int largestEulerSubset(boolean[][] matrix,int
     p) {
126          boolean success = false;
127          success = rowCheck(matrix,p);
128
129          if(success == true)data = p;
130          if(success == true)return p;
```

```
131
132         for(int i = 1; i <=V; i++){
133
134
135             for(int j = 1;j<=V;j++){
136
137             matrix[j][i] = false;
138             matrix[i][j] = false;
139         }
140             largestEulerSubset(matrix,p-1);
141
142         }
143         return 1;
144     }
145     //print stack result from circuit finder
146     private void printstack() {
147         int startnode = (V-V)+1;
148         stack.add(startnode);
149         int size = stack.size();
150         int circuit [] = new int[size];
151         int count = size-1;
152         char[] comp = {' ','A','B','C','D','E','F'};
153         while(!stack.isEmpty()){
154             int element = stack.pop();
155             circuit[count] = element;
156             count--;
157         }
158         System.out.print("Euler Circuit: ");
159         for(int i = 0; i < size; i++){
160             int temp = circuit[i];
161             System.out.print(comp[temp]);
162         }
163
164
165
166
167
168     }
169     //recursive  algorithm finding path
170     private  void findCircuit(int vertex) {
171
172         for(int dest = 1; dest<=V;dest++){
173             //next edge finds the next available edge to
    traverse too
174             if(!(adjacencyMatrix[vertex][dest] == false)
```

```
174 && NextEdge(vertex, dest)){
175                 q.add(vertex);
176                 stack.push(vertex);
177                 //removes edges onces traversed
178                 removeEdge(vertex, dest);
179                 //recurse
180                 findCircuit(dest);
181 
182             }
183         }
184 
185 
186 
187     }
188         //next available edge to traverse too
189     private boolean NextEdge(int s, int dest) {
190         int count = 0;
191         for (int vertex = 1; vertex <= V; vertex++)
192         {
193             if (!(adjacencyMatrix[s][vertex] == false))
194             {
195                 count++;
196             }
197         }
198 
199         if (count == 1 )
200         {
201             return true;
202         }
203 
204         int visited[] = new int[V+1];
205         int count1 = DFSC(s, visited);
206 
207         removeEdge(s, dest);
208         for (int vertex = 1; vertex <= V; vertex++)
209         {
210             visited[vertex] = 0;
211         }
212 
213         int count2 = DFSC(s, visited);
214         addEdge(s, dest);
215 
216         return (count1 > count2 ) ? false : true;
217 
218 
```

```
219     }
220 //depth first recursive algorothm
221     private int DFSC(int index, int[] visited) {
222         visited[index] = 1;
223         int count = 1;
224         int destination = 1;
225 
226         while (destination <= V)
227         {   //recurses to unvisited vertices
228             if(!(adjacencyMatrix[index][destination] ==
    false) && visited[destination] == 0)
229             {
230                 count += DFSC(destination, visited);
231             }
232             destination++;
233         }
234         return count;
235 
236     }
237     //check rows to ensure they are even
238 
239     private static boolean rowCheck(boolean[][] matrix,
    int p) {
240         int count = 0;
241         boolean testDegree = true;
242         for (int i = 1; i <= p; i++) {
243             for (int j = 1; j <= p; j++) {
244                 if(matrix[i][j]==true){
245                     count++;
246                 }
247             }
248             if(count % 2 == 0){
249                 testDegree = true;
250                 count = 0;
251             }else{
252                 testDegree = false;
253                 return testDegree;
254             }
255 
256 
257         }
258 
259         return testDegree;
260 
261     }
```

```
262 }
```