Problem:

Suppose we have three different colors of knights: black, white and green. Maximize the number of knights that we can put on a m*n chess board, such that no knights of different colors could attack each other, and there are equal numbers of all three colors.

Create the LP:

Define k = 1 means a blue knight, k = 2 means a black knight, k = 3 means a red knight.

Let x_{ijk} (i, j, k $\in$ Z) be the binary variable such that x_{ijk} = 0 means there won't be a knight on (i, j) with color k and x_{ijk} = 1 means there will be a knight on (i, j) with color k.

Since there are equal numbers of all 3 colors of knights, the maximum number of knights on the chess board equals to three times the number of knights with the same color on the chess board, so our object function could be $3*\sum_{j=1}^{n}\sum_{i=1}^{m} x_{ij1}$.

There are three constraints in this LP problem (x_{ijk} is always a binary variable):

(a) For any square that has a knight, the square at the right side and can attack it must be either empty or a knight of the same color. Assume there is a knight with color k at (i, j) and (l, m) is one of the squares at the right side and can attack (i, j). The following constraint should be added: $x_{ijk} + x_{lmk'} + x_{lmk''} \leq 1$where $k \neq k'$, $k \neq k''$, and $k' \neq k''$.

(b) There should be at most one knight at a square with a particular color, so the following constraint should be added: $\sum_{k=1}^{3} x_{ijk} \leq 1$ for all i, j $\in$ Z such that i $\leq$ m and j $\leq$ n.

(c) There are equal numbers of all 3 colors of knights, so the following constraint should be added: $\sum_{j=1}^{n}\sum_{i=1}^{m} x_{ij1} = \sum_{j=1}^{n}\sum_{i=1}^{m} x_{ij2} = \sum_{j=1}^{n}\sum_{i=1}^{m} x_{ij3}$.

Code to generate the lpsolve input file (JAVA, use 5*5 chess board as example in this case):

```
1 public class math381hw2 {
2   // This gives the number of rows for a chess board.
3   public static int m = 5;
4   // This gives the number of columns for a chess board.
5   public static int n = 5;
6   public static void main (String[] args) {
7     // Print out the object function.
8     System.out.print("max: ");
9     for (int i = 1; i <= m; i++) {
10      for (int j = 1; j <= n; j++) {
11        if (i == m && j == n) {
12          System.out.println("3x_" + i + "_" + j + "_" + 1 + ";");
13        } else {
14          System.out.print("3x_" + i + "_" + j + "_" + 1 + "+");
15        }
16      }
17    }
18    // Print the first constraint: for any square that has a knight, the square that is at the right
19    // side of it and can attack it must be either empty or have a knight with the same color.
20    for (int i = 1; i <= m; i++) {
21      for (int j = 1; j <= n; j++) {
22        if (i - 2 >= 1 && j + 1 <= n) {
23          System.out.println("x_" + i + "_" + j + "_1+x_" + (i - 2) + "_" + (j + 1) + "_2<=1;");
24          System.out.println("x_" + i + "_" + j + "_1+x_" + (i - 2) + "_" + (j + 1) + "_3<=1;");
25          System.out.println("x_" + i + "_" + j + "_2+x_" + (i - 2) + "_" + (j + 1) + "_1<=1;");
26          System.out.println("x_" + i + "_" + j + "_2+x_" + (i - 2) + "_" + (j + 1) + "_3<=1;");
27          System.out.println("x_" + i + "_" + j + "_3+x_" + (i - 2) + "_" + (j + 1) + "_1<=1;");
28          System.out.println("x_" + i + "_" + j + "_3+x_" + (i - 2) + "_" + (j + 1) + "_2<=1;");
29        }
30        if (i - 1 >= 1 && j + 2 <= n) {
31          System.out.println("x_" + i + "_" + j + "_1+x_" + (i - 1) + "_" + (j + 2) + "_2<=1;");
32          System.out.println("x_" + i + "_" + j + "_1+x_" + (i - 1) + "_" + (j + 2) + "_3<=1;");
33          System.out.println("x_" + i + "_" + j + "_2+x_" + (i - 1) + "_" + (j + 2) + "_1<=1;");
34          System.out.println("x_" + i + "_" + j + "_2+x_" + (i - 1) + "_" + (j + 2) + "_3<=1;");
35          System.out.println("x_" + i + "_" + j + "_3+x_" + (i - 1) + "_" + (j + 2) + "_1<=1;");
36          System.out.println("x_" + i + "_" + j + "_3+x_" + (i - 1) + "_" + (j + 2) + "_2<=1;");
37        }
38        if (i + 1 <= m && j + 2 <= n) {
39          System.out.println("x_" + i + "_" + j + "_1+x_" + (i + 1) + "_" + (j + 2) + "_2<=1;");
40          System.out.println("x_" + i + "_" + j + "_1+x_" + (i + 1) + "_" + (j + 2) + "_3<=1;");
41          System.out.println("x_" + i + "_" + j + "_2+x_" + (i + 1) + "_" + (j + 2) + "_1<=1;");
42          System.out.println("x_" + i + "_" + j + "_2+x_" + (i + 1) + "_" + (j + 2) + "_3<=1;");
43          System.out.println("x_" + i + "_" + j + "_3+x_" + (i + 1) + "_" + (j + 2) + "_1<=1;");
44          System.out.println("x_" + i + "_" + j + "_3+x_" + (i + 1) + "_" + (j + 2) + "_2<=1;");
45        }
46        if (i + 2 <= m && j + 1 <= n) {
47          System.out.println("x_" + i + "_" + j + "_1+x_" + (i + 2) + "_" + (j + 1) + "_2<=1;");
48          System.out.println("x_" + i + "_" + j + "_1+x_" + (i + 2) + "_" + (j + 1) + "_3<=1;");
49          System.out.println("x_" + i + "_" + j + "_2+x_" + (i + 2) + "_" + (j + 1) + "_1<=1;");
50          System.out.println("x_" + i + "_" + j + "_2+x_" + (i + 2) + "_" + (j + 1) + "_3<=1;");
51          System.out.println("x_" + i + "_" + j + "_3+x_" + (i + 2) + "_" + (j + 1) + "_1<=1;");
```

```
52            System.out.println("x_" + i + "_" + j + "_3+x_" + (i + 2) + "_" + (j + 1) + "_2<=1;");
53          }
54        }
55      }
56      // Print the second constraint: there should be at most one knight at a square with a particular color.
57      for (int i = 1; i <= m; i++) {
58        for (int j = 1; j <= n; j++) {
59          System.out.println("x_" + i + "_" + j + "_1+x_" + i + "_" + j + "_2+x_" + i + "_" + j + "_3<=1;");
60        }
61      }
62      // Print the third constraint: there are equal numbers of all 3 colors of knights.
63      for (int i = 1; i <= m; i++) {
64        for (int j = 1; j <= n; j++) {
65          if (i == m && j == n) {
66            System.out.print("x_" + i + "_" + j + "_1=");
67          } else {
68            System.out.print("x_" + i + "_" + j + "_1+");
69          }
70        }
71      }
72      for (int i = 1; i <= m; i++) {
73        for (int j = 1; j <= n; j++) {
74          if (i == m && j == n) {
75            System.out.println("x_" + i + "_" + j + "_2;");
76          } else {
77            System.out.print("x_" + i + "_" + j + "_2+");
78          }
79        }
80      }
81      for (int i = 1; i <= m; i++) {
82        for (int j = 1; j <= n; j++) {
83          if (i == m && j == n) {
84            System.out.print("x_" + i + "_" + j + "_2=");
85          } else {
86            System.out.print("x_" + i + "_" + j + "_2+");
87          }
88        }
89      }
90      for (int i = 1; i <= m; i++) {
91        for (int j = 1; j <= n; j++) {
92          if (i == m && j == n) {
93            System.out.println("x_" + i + "_" + j + "_3;");
94          } else {
95            System.out.print("x_" + i + "_" + j + "_3+");
96          }
97        }
98      }
99      // Print the condition that every variable is binary.
100     System.out.print("bin ");
101     for (int i = 1; i <= m; i++) {
102       for (int j = 1; j <= n; j++) {
103         for (int k = 1; k <= 3; k++) {
104           if (i == m && j == n && k == 3) {
105             System.out.println("x_" + i + "_" + j + "_" + k + ";");
```

```
106             } else {
107               System.out.print("x_" + i + "_" + j + "_" + k + ",");
108             }
109           }
110         }
111       }
112   }
113 }
```

*LP input file to solve the LP problem for a 5*5 chess board:*

max:
3x_1_1_1+3x_1_2_1+3x_1_3_1+3x_1_4_1+3x_1_5_1+3x_2_1_1+3x_2_2_1+3x_2_3_1+3x_2_4_1+3x_2_5_1+3x_3_1_1+3x_3_2_1+3x_3_3_1+3x_3_4_1+3x_3_5_1+3x_4_1_1+3x_4_2_1+3x_4_3_1+3x_4_4_1+3x_4_5_1+3x_5_1_1+3x_5_2_1+3x_5_3_1+3x_5_4_1+3x_5_5_1;
(288 lines of the following type: ensure that for any square that has a knight, the square that is at the right side of it and can attack it must be either empty or have a knight with the same color)
x_1_1_1+x_2_3_2<=1;
.
.
.
x_5_4_3+x_3_5_2<=1;
(25 lines of the following type: ensure that there should be at most one knight at a square with a particular color)
x_1_1_1+x_1_1_2+x_1_1_3<=1;
.
.
.
x_5_5_1+x_5_5_2+x_5_5_3<=1;
(ensure that number of knights on the chess board with the same color is equal)
x_1_1_1+x_1_2_1+...x_5_4_1+x_5_5_1=x_1_1_2+x_1_2_2+...+x_5_4_2+x_5_5_2;
x_1_1_2+x_1_2_2+...+x_5_4_2+x_5_5_2=x_1_1_3+x_1_2_3+...+x_5_4_3+x_5_5_3;
(ensure all variables are binary)
bin x_1_1_1, x_1_1_2, ..., x_5_5_2, x_5_5_3;

Output file by using lp_solve to generate the LP input file:

Value of objective function: 15.00000000

Actual values of the variables:

x_1_1_1	1
x_1_3_1	1
x_1_5_1	1
x_2_1_1	1
x_2_3_1	1
x_2_4_2	1
x_1_2_2	1
x_1_4_3	1
x_2_2_3	1
x_4_5_2	1
x_5_2_3	1

x_5_3_2 1
x_5_4_3 1
x_5_5_2 1
x_5_1_3 1
All other variables are zero.
./lp_solve -ia hw2_55.txt 1.46s user 0.01s system 99% cpu 1.478 total

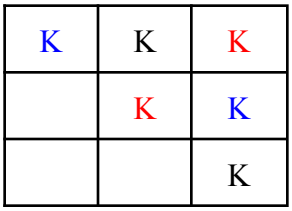
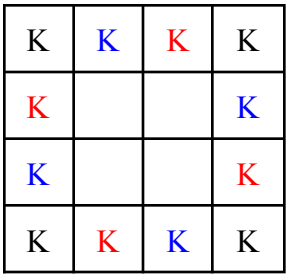
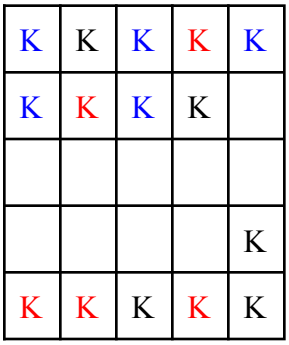
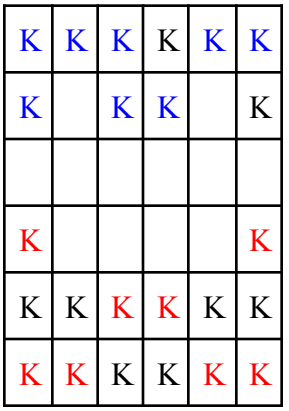
Since 1.46s < 10 minutes, I continue to use lp_solve to solve the LP problem for a 6*6 chess board, and I get the following output file after generating the new LP input file:

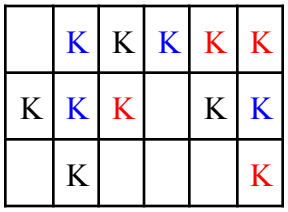
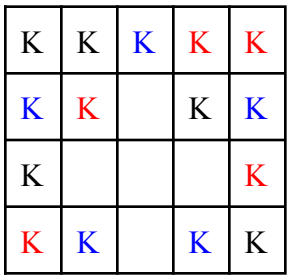
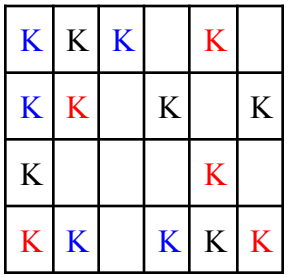
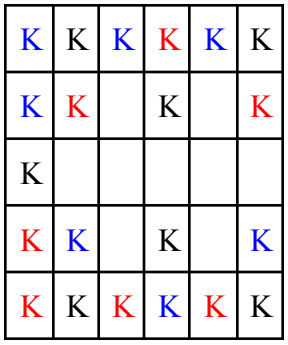
Value of objective function: 24.00000000

Actual values of the variables:
x_1_1_1 1
x_1_2_1 1
x_1_3_1 1
x_1_5_1 1
x_1_6_1 1
x_2_1_1 1
x_2_3_1 1
x_2_4_1 1
x_2_6_2 1
x_1_4_2 1
x_4_6_3 1
x_5_2_2 1
x_5_3_3 1
x_5_4_3 1
x_5_5_2 1
x_5_6_2 1
x_4_1_3 1
x_6_2_3 1
x_6_3_2 1
x_6_4_2 1
x_6_5_3 1
x_6_6_3 1
x_5_1_2 1
x_6_1_3 1
All other variables are zero.
./lp_solve -ia hw2_66.txt 27.20s user 0.02s system 99% cpu 27.246 total

Since 27.2s < 10 minutes, I continue to use lp_solve to solve the Lp problem for a 7*7 chess board. Yet, it couldn't give me the answer even in 30 minutes, so I terminated it.

Below is the information table for 8 individual chess boards. 4 of them are square chess boards, and the other 4 chess boards are not:

Size of Chess board	Max num of knights	Running Time	Solution Board
3*3	6	0.01s	
4*4	12	0.02s	
5*5	15	1.46s	
6*6	24	27.20s	

3*6	12	0.06s	
4*5	15	0.07s	
4*6	15	0.80s	
5*6	21	2.00s	

Personal Conclusion and some interesting facts:

By checking the running time for square chess boards, we could see that the larger the board is, the longer time is required for the lp_solve to run to solve for the solution. However, the decisive factor for the running time of lp_solve could be either the number of knights on the board or the number of squares the board has. By comparing the information for the 4*5 and 4*6 chess board, I figure out that the maximum number of knights on chess board 4*5 in same as the maximum number of knights on chess board 4*6 when there is a significant difference between the running time for chess board 4*5 and 4*6, so ***the decisive factor for the running time of lp_solve should be the number of squares on the chess board***. I also find three interesting facts: there are at least two knights that are placed at the corner on either chess board, the relationship between the change of running time for lp_solve to solve the chess board and the size of the chess board is not linear, and the redundant inequalities in the input file won't affect running time too much. For the first fact, I have a hypothesis: ***placing knights at corners as much as possible might help to get a feasible solution for the LP problem***, since there are only 2 attacking spaces for knights at the corner. For the second fact, the average time to solve a square on 4*4, 5*5, 6*6 chess boards, by calculation, is about 0.0016s, 0.097s, and 1.13s separately. So I have a hypothesis for such a chess board problem: ***the running time to solve the related LP problem would be much longer by only increasing a few spaces on the chess board as the chess board becomes larger and larger***. For the third fact, I make the first constraint for the Lp problem for 5*5 chess board to be redundant and run it (I copy and paste the 288 lines to make the first constraint be 576 lines) and find out that the running time for the redundant one is similar to the running time for the simplified one. Thus, I make the following hypothesis: ***lp_solve could automatically omit repeated inequalities in a very short time (maybe in milliseconds)***.