The Kings problem

Mark McClure
Department of Mathematics
Universiry of North Carolina at Asheville
Asheville, NC 28804
mcmcclur@unca.edu

Abstract

The m n Kings problem asks how many ways non-attacking may Kings be placed on an m n chess board. Recent work by Neil Calkin and his undergraduate research students has produced an elegant way to approach the problem, which also leads to the enumeration of all possible boards for small m and n. Furthermore, a surprising connection to self-similar fractals arises leading to challenging open questions.

1. The basic problem

The basic problem of the Kings is the following: How many different ways may we place kings on an $m \times n$ chessboard so that no two are attacking one another? This is a natural combinatorial question which has been considered by several authors. It's first appearance in print seems to be in a paper of Wilf [1], who attributes the problem to Donald Knuth. Substantial progress on the asymptotics of the problem have been made by Neil Calkin and his students Shannon Purvis and Keith Schneider. It is this work and, in particular, the occurence of fractals in the solution which sparked the interest of this column.

One dimensional boards

We begin our attack on the problem with the simple m 1 case. We call such a board a column and will construct the more general m n board by stringing n columns together. It is not hard to construct the set C_m of all possible columns of length m using recursion on m. We will represent a column as a list of zeros and ones of length m. Zero represents an empty square and one represents an occupied square. Clearly, there are two 1×1 boards, the empty square and the occupied square.

```
In[1]:= C1 = {{0}, {1}};
```

It's also fairly clear that there are three possible columns of length 2.

```
In[2]:= C2 = {{0, 0}, {1, 0}, {0, 1}};
```

Now the general $m \times 1$ board may be generated in one of two ways. We may place an empty square at the end of an $(m-1) \times 1$ board or an empty square and then a king at the end of a $(m-2) \times 1$ board. This leads to the following recursive defintion.

```
In[3]:= Cm_ := Cm = Join[
            Join[#, {0}] &    Cm-1,
            Join[#, {0, 1}] &    Cm-2];
        MatrixForm    C4
```

Out[4]= $\left\{\begin{pmatrix}0\\0\\0\\0\end{pmatrix}, \begin{pmatrix}1\\0\\0\\0\end{pmatrix}, \begin{pmatrix}0\\1\\0\\0\end{pmatrix}, \begin{pmatrix}0\\0\\1\\0\end{pmatrix}, \begin{pmatrix}1\\0\\1\\0\end{pmatrix}, \begin{pmatrix}0\\0\\0\\1\end{pmatrix}, \begin{pmatrix}1\\0\\0\\1\end{pmatrix}, \begin{pmatrix}0\\1\\0\\1\end{pmatrix}\right\}$

It's easy to see from this recursion that the number of columns of length m is F_{m+2}, the $m+2^{\text{nd}}$ Fibonacci number.

Adjacency matrices

An m n board may be generated by stringing n columns of length m together. However, care must be taken not to introduce an illegal configuration. We will keep track of which columns may be placed next to one another using an adjacency matrix A_m. The rows and columns of A_m will be indexed by C_m. The entry in row i and column j will be 1 if column i may be placed next to column j and 0 otherwise.

For $m = 1$ note that the empty square may be placed next to itself or the occupied square, but the occupied square may only be placed next to the empty square. Thus A_1 is the following matrix.

In[5]:= $\mathbf{A_1} = \begin{pmatrix}1 & 1\\1 & 0\end{pmatrix};$

Similarly, A_2 is given by

In[6]:= $\mathbf{A_2} = \begin{pmatrix}1 & 1 & 1\\1 & 0 & 0\\1 & 0 & 0\end{pmatrix};$

The general matrix A_m may be also be computed by a recursive procedure.

$$A_m = \begin{pmatrix}A_{m-1} & A_{m-2}\\A_{m-2} & 0\end{pmatrix}$$

In this formula, we assume that the 0 is a zero matrix of size F_m and the copies of A_{m-2} are padded with zeros appropriately to make the result a square matrix of size F_{m+2}.

The recurrence may be derived as follows. Recall that there are $F_{m+2} = F_{m+1} + F_m$ possible columns of length m in C_m. Furthermore, the first F_{m+1} elements in C_m are exactly the columns in C_{m-1} with an empty square at the end and these columns of length m have the exact same adjacency conditions as the corresponding in C_{m-1}. This leads to the copy of A_{m-1} in the upper left hand corner of the recurrence. Similarly, the first F_m columns in C_m are exactly the columns in C_{m-2} with two empty squares appended and the last F_m columns in C_m are exactly the columns in C_{m-2} with an empty square and then a king. These columns have the adjacency relationships described by the copy of A_{m-2} in the upper right of the recurrence. The copy of A_{m-2} in the lower left appears by symmetry. Of course, the final F_m columns in C_m all have a king in the last square so no two of these may appear next to one another. This leads to the zero block in the lower right.

The matrix A_m will be quite sparse for large m, thus we will use **`SparseArray`**s to implement the recurrence.

```
In[7]:= F = Fibonacci;
        A_m_ := A_m = Module[
           {rules1, rules2, newRules},
           rules1 = Drop[ArrayRules[A_m-1], -1];
           rules2 = Drop[ArrayRules[A_m-2], -1];
           newRules = Join[rules1,
             rules2  . {x_Integer, y_Integer} → {x + F[m + 1], y},
             rules2  . {x_Integer, y_Integer} → {x, y + F[m + 1]}];
           SparseArray[newRules, {F[m + 2], F[m + 2]}]];
        A_3   MatrixForm
Out[9]//MatrixForm=
```

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Counting the boards

Our objective is to count how many different ways we may place non-attacking kings on an $m \times n$ board. We can generate any such board by placing n columns of length m next to one another. Recall that A_m is an adjacency matrix telling us which columns may be placed next to one another. This adjacency matrix corresponds to a graph and a board with n columns corresponds to a walk through n nodes. The length of this walk is $n-1$, since length of a walk corresponds to the number of edges traversed. It is a standard result in graph theory that we count the total number of such walks by summing the entries in A_m^{n-1}. For example, the following computation shows that there are 21 different 4×2 boards.

```
In[10]:= Plus   A_4.Table[1, {F[6]}]
Out[10]= 21
```

We may square A_m to obtain the number of paths of length 2 or, equivalently, the $m \times 3$ boards. For example, there are 35 different 3×3 boards.

```
In[11]:= Plus   Flatten[A_3.A_3.Table[1, {F[5]}]]
Out[11]= 35
```

There are over a billion ways to place non-attacking kings on an actual 8×8 chessboard.

```
In[12]:= Plus   MatrixPower[A_8, 7].Table[1, {F[10]}]
Out[12]= 1355115601
```

2. Enumerating the boards

Basic enumeration

By replacing standard matrix multiplication with a more general inner product, we can actually enumerate the boards for small m and n. The following matrix describes the paths of length one through the graph represented by A_3. The element in row i and column j is the set of all paths of length 1 from vertex i to vertex j.

```
In[13]:= length1PathsMatrix =
          MapIndexed[{#1 * #2} &, A3, {2}]   . {{0, 0}} → {};
         length1PathsMatrix    MatrixForm
```

```
⎛ {{1, 1}}  {{1, 2}}  {{1, 3}}  {{1, 4}}  {{1, 5}} ⎞
⎜ {{2, 1}}  {}        {}        {{2, 4}}  {}       ⎟
⎜ {{3, 1}}  {}        {}        {}        {}       ⎟
⎜ {{4, 1}}  {{4, 2}}  {}        {}        {}       ⎟
⎝ {{5, 1}}  {}        {}        {}        {}       ⎠
```

Here are the paths of length 2. By examining the element in the first row and column, we see that there are five paths of length two from the first vertex to itself. This is because the first vertex A_3 corresponds to the empty column and may therefore be adjacent to any column.

```
In[15]:= nextSteps = MapIndexed[#1 * #2[[2]] &, A3, {2}];
         takeStep[{}, _] := {};
         takeStep[_, 0] := {};
         takeStep[partialPaths_, nextStep_Integer] :=
          Join[#, {nextStep}] &    partialPaths;
         length2PathsMatrix =
          Inner[takeStep, length1PathsMatrix , nextSteps , Join];
         length2PathsMatrix[[1, 1]]

Out[20]= {{1, 1, 1}, {1, 2, 1}, {1, 3, 1}, {1, 4, 1}, {1, 5, 1}}
```

Note that each of these paths corresponds to a legal 3 3 board. It is very easy to construct the board from the path; we simply extract the columns determined by the path from C_m. Here is the board generated by the path {1, 5, 1} through A_3 represented as a zero-one matrix.

```
In[21]:= Board[m_Integer, path_List] := Transpose[Cm[[path]]];
         Board[3, {1, 5, 1}]    MatrixForm

Out[22]//MatrixForm=
⎛ 0  1  0 ⎞
⎜ 0  0  0 ⎟
⎝ 0  1  0 ⎠
```

In general, we can write a function **`Paths`** which returns a matrix of paths. The element in row i and column j of **`Paths[m,n]`** will be the list of all possible paths of length **`n`** from column i to column j.

```
In[23]:= takeStep[{}, _] := {};
         takeStep[_, 0] := {};
```

```
takeStep[partialPaths_, nextStep_Integer] :=
 Join[#, {nextStep}] &    partialPaths;
Paths[m_Integer, n_Integer] := Module[
  {length1PathsMatrix, nextSteps},
  length1PathsMatrix =
   MapIndexed[{#1 * #2} &, A_m, {2}]   . {{0, 0}} → {};
  nextSteps = MapIndexed[#1 * #2[[2]] &, A_m, {2}];
  Nest[Inner[takeStep, # , nextSteps , Join] &,
   length1PathsMatrix, n - 1]];
```

Now that we can enumerate all possible paths, we can generate the correpsonding boards. Here are all 35 of the 3×3 boards.

```
In[27]:= Boards[m_Integer, n_Integer] := Board[m, #] &
           Partition[Flatten[Paths[m, n - 1]], n];
         MatrixForm    Boards[3, 3]
```

Out[28]=

$$\left\{\begin{pmatrix}0&0&0\\0&0&0\\0&0&0\end{pmatrix}, \begin{pmatrix}0&1&0\\0&0&0\\0&0&0\end{pmatrix}, \begin{pmatrix}0&0&0\\0&1&0\\0&0&0\end{pmatrix}, \begin{pmatrix}0&0&0\\0&0&0\\0&1&0\end{pmatrix}, \begin{pmatrix}0&1&0\\0&0&0\\0&1&0\end{pmatrix},\right.$$
$$\begin{pmatrix}0&0&1\\0&0&0\\0&0&0\end{pmatrix}, \begin{pmatrix}0&0&1\\0&0&0\\0&1&0\end{pmatrix}, \begin{pmatrix}0&0&0\\0&0&1\\0&0&0\end{pmatrix}, \begin{pmatrix}0&0&0\\0&0&0\\0&0&1\end{pmatrix}, \begin{pmatrix}0&1&0\\0&0&0\\0&0&1\end{pmatrix},$$
$$\begin{pmatrix}0&0&1\\0&0&0\\0&0&1\end{pmatrix}, \begin{pmatrix}1&0&0\\0&0&0\\0&0&0\end{pmatrix}, \begin{pmatrix}1&0&0\\0&0&0\\0&1&0\end{pmatrix}, \begin{pmatrix}1&0&1\\0&0&0\\0&0&0\end{pmatrix}, \begin{pmatrix}1&0&1\\0&0&0\\0&1&0\end{pmatrix},$$
$$\begin{pmatrix}1&0&0\\0&0&1\\0&0&0\end{pmatrix}, \begin{pmatrix}1&0&0\\0&0&0\\0&0&1\end{pmatrix}, \begin{pmatrix}1&0&1\\0&0&0\\0&0&1\end{pmatrix}, \begin{pmatrix}0&0&0\\1&0&0\\0&0&0\end{pmatrix}, \begin{pmatrix}0&0&1\\1&0&0\\0&0&0\end{pmatrix},$$
$$\begin{pmatrix}0&0&0\\1&0&1\\0&0&0\end{pmatrix}, \begin{pmatrix}0&0&0\\1&0&0\\0&0&1\end{pmatrix}, \begin{pmatrix}0&0&1\\1&0&0\\0&0&1\end{pmatrix}, \begin{pmatrix}0&0&0\\0&0&0\\1&0&0\end{pmatrix}, \begin{pmatrix}0&1&0\\0&0&0\\1&0&0\end{pmatrix},$$
$$\begin{pmatrix}0&0&1\\0&0&0\\1&0&0\end{pmatrix}, \begin{pmatrix}0&0&0\\0&0&1\\1&0&0\end{pmatrix}, \begin{pmatrix}0&0&0\\0&0&0\\1&0&1\end{pmatrix}, \begin{pmatrix}0&1&0\\0&0&0\\1&0&1\end{pmatrix}, \begin{pmatrix}0&0&1\\0&0&0\\1&0&1\end{pmatrix},$$
$$\left.\begin{pmatrix}1&0&0\\0&0&0\\1&0&0\end{pmatrix}, \begin{pmatrix}1&0&1\\0&0&0\\1&0&0\end{pmatrix}, \begin{pmatrix}1&0&0\\0&0&1\\1&0&0\end{pmatrix}, \begin{pmatrix}1&0&0\\0&0&0\\1&0&1\end{pmatrix}, \begin{pmatrix}1&0&1\\0&0&0\\1&0&1\end{pmatrix}\right\}$$

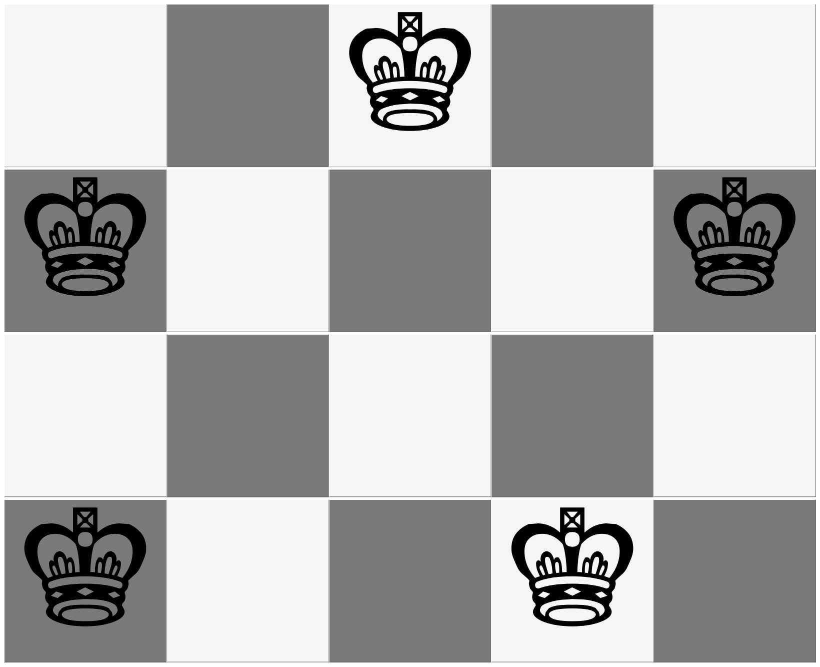
Making the boards pretty

As this is a graphics column, we would like our boards to look as nice as possible. One approach is to install a special font which renders certain characters as chess pieces. A large number of possible fonts are available at http://www.enpassant.dk/chess/fonteng.htm. The code below assumes that the Chess Leipzig font is installed on the system, altough the images are not so bad without it. The board may then be set up using a **`GridBox`**.

```
In[29]:= PrettySquare[val : 0 | 1, pos_] := ButtonBox[
            StyleForm[If[val   0, " ", "k"], FontSize → 55,
             FontColor → Black,
             FontFamily → "Chess Leipzig"],
            ButtonMinHeight → 3.5,
            Background → If[EvenQ[Plus   pos], GrayLevel[0.95],
              GrayLevel[0.4]]];
         PrettyBoard[m_Integer, path_List] := GridBox[
            MapIndexed[PrettySquare, Board[m, path], {2}],
            RowSpacings → 0, ColumnSpacings → 0,
            RowsEqual → True, ColumnsEqual → True];
```

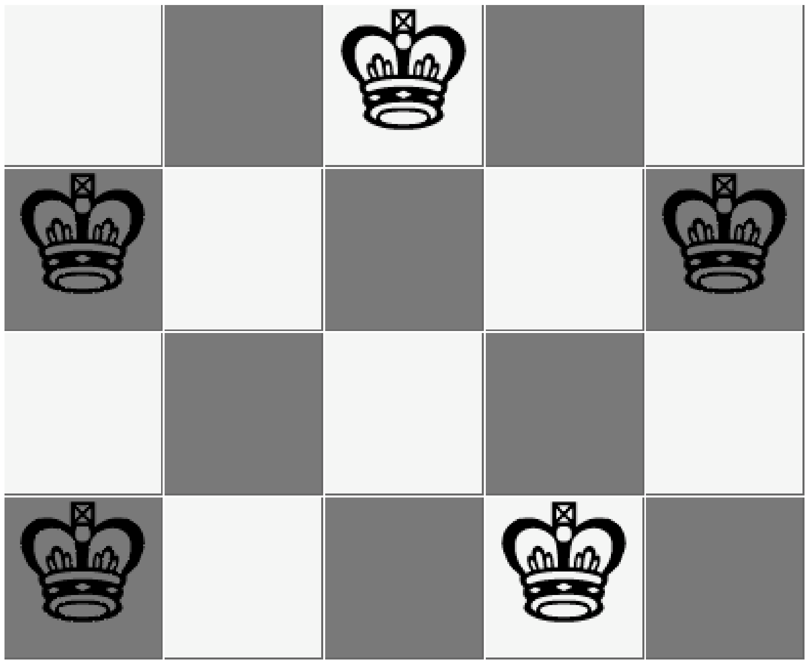
Here is a 4   5 board, for example.

```
In[32]:= a4x5Board = PrettyBoard[4, {8, 1, 2, 6, 3}]    DisplayForm
```



Without the special fonts installed, the above command should yield a chessboard with each king rendered as a "k". With the fonts, the kings should look quite nice. A gif image of the result is shown below.

```
In[47]:= Show[Import["a4x5Board.gif"]];
```



A list of all possible m n boards may be generated using the **PrettyBoards** command defined below.

```
In[48]:= PrettyBoards[m_Integer, n_Integer] :=
           DisplayForm[PrettyBoard[m, #]] &
            Partition[Flatten[Paths[m, n - 1]], n];
```

A 1 GHz iMac will typeset all 6427 5   5 boards in about 10 seconds.

3. Self-similar matrices

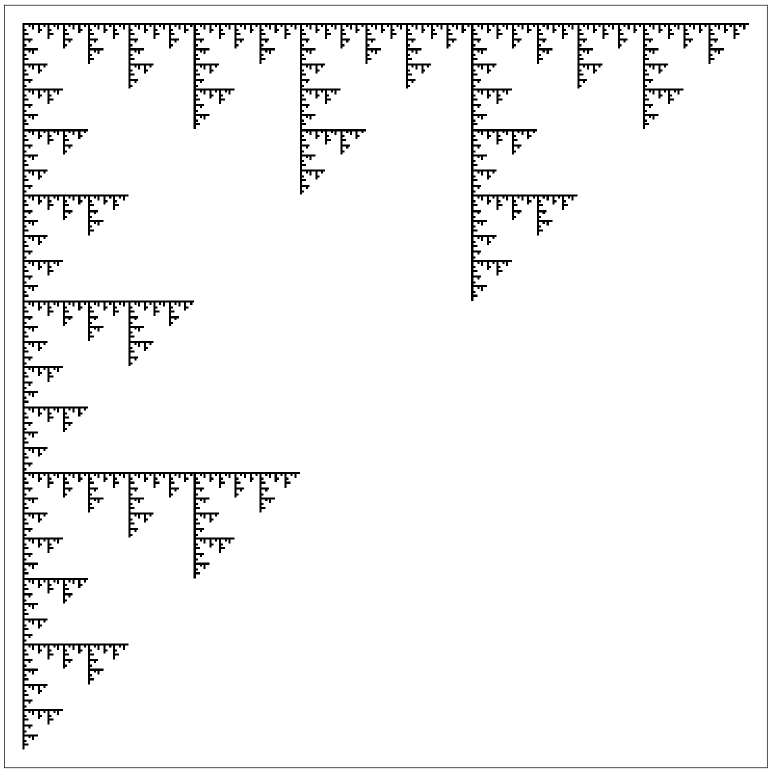
It turns out that the number of possible configurations on an m n chessboard grows asymptotically as λ_m^n, where λ_m is the larges eigenvalue of A_m. Furthermore, λ_m grows exponentially: $\lambda_m \approx \eta^m$, where $\eta \approx 1.34$. These results were discovered experimentally and then proved. To do this, it is necessary to experiment with A_m for somewhat larger values of m. The need for this type of computation is the main reason for the implementation using sparse matrices. For example, the following computation provides some justification that $\lambda_m \approx \eta^m$.

```
In[64]:= Eigenvalues[N[A16], 1]  Eigenvalues[N[A15], 1]

Out[64]= {1.34263}
```

Such experimentation leads to some intriguing connections with fractal geometry. Here is a simple array plot of A_{12}, for example.

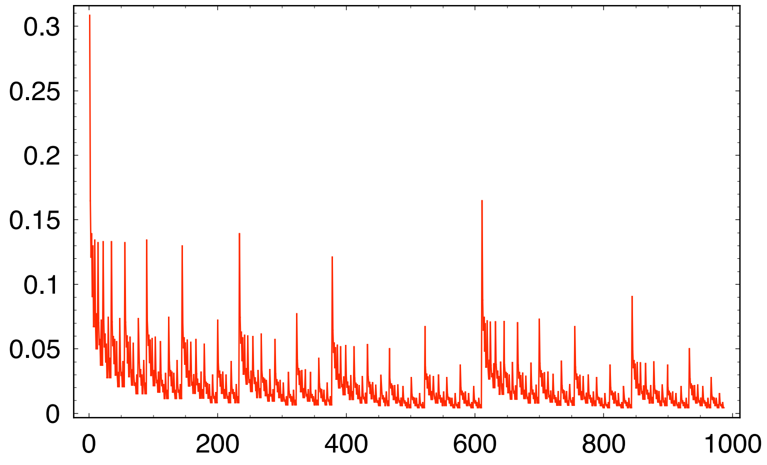
```
In[63]:= ArrayPlot[A12];
```



It's quite easy to prove that there is a natural limiting object of the sequence of matrices. This limit is a self-similar set consisting of three pieces - one scaled by the factor 1 ϕ and two scaled by the factor 1 ϕ^2, where ϕ is the golden ratio.

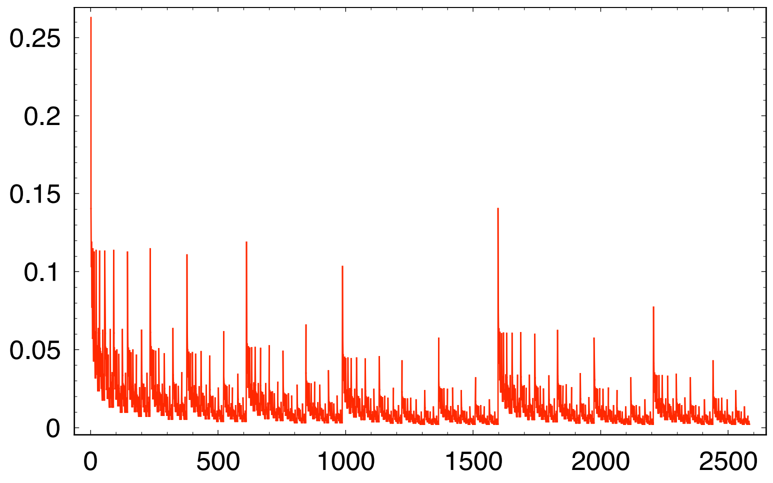
Also of tremendous interest is the apparent limit of the dominant eigenvectors. Here is a **ListPlot** of the dominant eigenvector of A_{14}.

```
In[69]:= ListPlot[-Eigenvectors[N[A14], 1][[1]],
           PlotJoined → True, PlotRange → All];
```



Here is a similar picture for A_{14}.

```
In[70]:= ListPlot[-Eigenvectors[N[A16], 1][[1]],
           PlotJoined → True, PlotRange → All];
```



It certainly appears that there is a similarity between these pictures. However, the definite existence of a limit has not been proved.

References

[1] This is a reference cell.