The Mathematica® Journal

Buckets and Jumping Pets

Jaime Rangel-Mondragón

In this article we examine three well-known problems from the recreational mathematics literature dealing with goal-oriented strategies set on a discrete state space. We use the backtrack paradigm to implement an exhaustive search and deduce some theoretical results, which in some cases allows us to provide an explicit description of their solutions.

The Problem of the Buckets

> Jack and Jill went up the hill,
> To fetch a pail of water;
> Jack fell down and broke his crown,
> and Jill came tumbling after.
>
> Nursery Rhyme
> J. W. Elliot, 1765

Consider the following question. Given two unmarked buckets of capacities 3 and 5 liters, how can we obtain exactly 1 liter of water from an inexhaustible well? The answer is to fill the 3-liter bucket first and empty it into the other bucket. Then, refill the 3-liter bucket from the well and use it to completely fill the other one, leaving the 1 liter desired. This problem is one of a famous category of "decanting" problems that have always appealed to a wide audience, including recreational mathematicians and computer scientists. Similar problems have found valuable applications in the analysis and teaching of algorithmic techniques, typifying many of the difficulties involved in the design of goal-oriented strategies. In the problem of the buckets, it is easy to show that if we have buckets of capacities a and b liters, and we start with both empty, the problem is solvable if and only if a and b are coprime. If this is the case, we have several ways to proceed. Each of them can be described by a sequence of ordered pairs accounting for the states through which we pass at each step towards our goal. Consider for instance, one of the shortest sequences that solves the problem for capacities 2 and 5: $\{\{0,0\},\{0,5\},\{2,3\},\{0,3\},\{2,1\}\}$.

In solving this problem we notice that any proper move we can make at each step is one of three types:

- Fill an empty bucket from the well. We do not fill a partially filled bucket because that wastes the moves that made it partially full.
- Empty a full bucket into the well.

- Pour one bucket into another, in such a way that we completely empty the first or fill the second.

The general ideas in solving these types of problems are as follows. There is a starting state and one or more final states that we aim to reach. If we are able to do so, we then want to have a description of the actual paths taken from the start to each one of the final states; moreover, we want those paths of minimal length. In order to direct a current partial path towards the goal, we follow guidelines that help to avoid entering blind alleys or performing wasteful moves. For instance, we should avoid reaching a previously visited state. In general, an explicit list describing all the traversed states to the goal would suffice, upgrading an upper bound on the length of future solution paths to alleviate the phenomenon of combinatorial explosion that this approach entails. A "Markovian" point of view is often adopted; the choice of the next state only depending on the current one, and not on the history of all or any previous ones. We can thus encapsulate the process of a single change of state by using the following list of productions. Note the conditionals appearing in the last two productions; without them we might, in certain cases, stay in the same current state.

```
pour := {{0, j_} → {a, j}, {i_, 0} → {i, b}, {a, j_} → {0, j},
   {i_, b} → {i, 0},
   {i_, j_ /; j ≠ b} → If[b < i + j, {i + j - b, b}, {0, i + j}],
   {i_ /; i ≠ a, j_} → If[a < i + j, {a, i + j - a}, {i + j, 0}]}
```

We use the function `ReplaceList` to gather all possible next states.

```
nextStates[state : {_Integer, _Integer}] :=
 Union[ReplaceList[state, pour]]
```

For example, if we have managed to get to state {2, 3} in our first example, the states we can then access are computed as:

```
{a, b} = {2, 5};
nextStates[{2, 3}]
```

```
{{0, 3}, {0, 5}}
```

That is, we could empty the first bucket by either pouring its content into the well or into the other bucket. Note that this list of candidates is returned in lexicographic order.

From the computational point of view, a good approach to solving this kind of problem is to use the backtrack technique. Backtrack is a basic method in the computer scientist repertoire, often used in problems in which it is required to find all solutions satisfying a given property. The general framework is that of a selection of all finite paths traversing a (possibly infinite) discrete state space. By its exhaustive nature, backtrack is often modified in order to overcome large time constraints that are typically exponential in nature (not so its space restrictions, which are always relatively affordable) [1]. However, its popularity arises mainly due to its straightforward quality and simplicity in solving an extensive class of computational problems. Often the main reason for choosing backtrack

is simply the fact that, for many important problems, there is no other feasible approach discovered so far. It is our aim in this article to illustrate an interesting instance of such problems and, in the process, to show how the general backtrack method can be modified in an ad-hoc way, leading to a significant improvement in execution time. This tailoring can arise from some theoretical insight or from a study of the computational burden imposed by its particular use. In general, the backtrack method is sufficiently flexible to give the programmer considerable scope in exercising skills for solving important combinatorial problems in an effective way.

Although there have been efforts in providing general backtrack routines—notably those included in Skiena's excellent books [2, 3], the value of backtrack lies not in its generality per se but in its capacity of adaptation. Because of drawbacks in the languages implementing it, its role has been largely overlooked. In fact, there was a time when the recursive style of programming was discouraged on the grounds that it was "unnatural." The truth was that what was unnatural were the features of the primitive programming languages available.

In its more general setup, the backtrack mechanism grows a currently acceptable partial result *r* by choosing from a set of candidates *S*, gathered by a function such as the function `nextStates`. Each element of *S* is successively appended to *r* and the process is recursively iterated with each of these new partial results. This growing either leads to the finding of a solution or to a stage in which *S* is void, in which case we backtrack to consider another candidate instead of the last one appended. By repeating this process we arrive at fully spanning the whole state space in a goal-oriented and efficient way; however, as we mentioned before, on occasion this is not enough to counterbalance a potential combinatorial explosion. The following pseudocode implements this idea.

```
backtrack[r]
    If r is a solution Print[r] and stop
    Compute S
    Map[backtrack[Append[r,#]]&, S]
```

So our query for a solution would be `backtrack[initial state]`. We can easily modify this scheme to search for all possible solutions, thus attempting to find the "best one" in an exhaustive manner. By its very nature, backtrack allows us to prune the branches of the tree spanned by this traversal in a systematic way, thus cutting unfeasible branches as soon as possible, which, in a full searching of the global state space results is vital.

The following function `buckets[a, b, d]` computes the solution list corresponding to the problem of obtaining 1 liter with buckets of capacities *a* and *b* liters, limiting the exhaustive search to a depth *d*. When $\gcd(a, b) \neq 1$, it returns the empty list.

```
buckets[a_, b_, d_] :=
 Module[{back, pour, nextStates, lim = d, ans = {}, i, j},

  back[x_] := 1 /; Length[x] > lim;
  back[x : {___, {___, 1, ___}}] :=
   (AppendTo[ans, x]; lim = Min[lim, Length[x]]);
  back[feasible : {{_Integer, _Integer} ..}] :=
   Module[{candidates},
    candidates = Select[nextStates[Last[feasible]],
      Not[MemberQ[feasible, #]] &];
    Map[back[Join[feasible, {#}]] &, candidates]];

  pour := {{0, j_} → {a, j}, {i_, 0} → {i, b}, {a, j_} → {0, j},
    {i_, b} → {i, 0},
    {i_, j_ /; j ≠ b} → If[b < i + j, {i + j - b, b}, {0, i + j}],
    {i_ /; i ≠ a, j_} → If[a < i + j, {a, i + j - a}, {i + j, 0}]};

  nextStates[state : {_Integer, _Integer}] :=
   Union[ReplaceList[state, pour]];

  back[{{0, 0}}];
  If[ans == {}, {}, Last[ans]]]
```

We verify our original example.

```
buckets[2, 5, 5]
```

```
{{0, 0}, {0, 5}, {2, 3}, {0, 3}, {2, 1}}
```

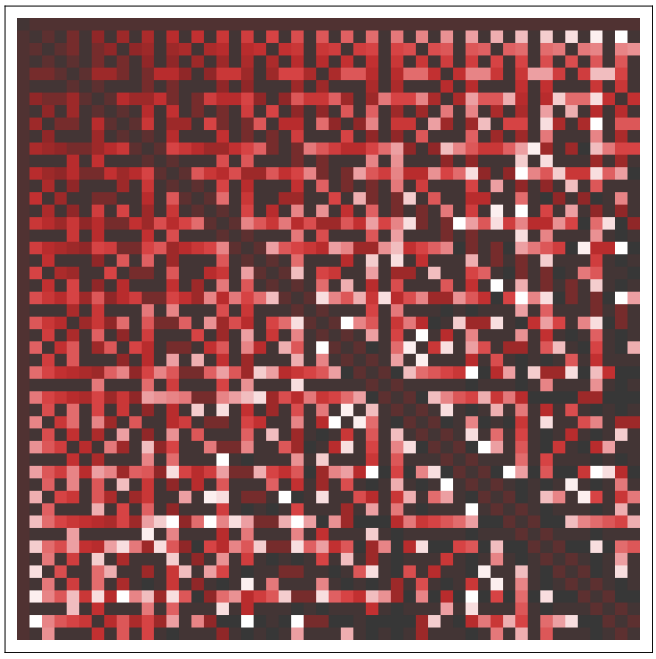
The following is a table of the length of the solutions corresponding to all possible combinations of the values a and b of at most size 21. Note that we still do not have a bound on the number of steps taken, so we use $d = 50$ for all cases; as the matrix B is symmetric with zeroes on the main diagonal, the computing time can be shortened.

```
B = Table[0, {21}, {21}];
Do[
  If[GCD[a, b] == 1,
   B⟦a,b⟧ = B⟦b,a⟧ = Length[buckets[a, b, 50]] - 1], {a, 21},
  {b, a + 1, 21}];
MatrixForm[B]
```

0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	2	0	4	0	6	0	8	0	10	0	12	0	14	0	16	0	18	0	20
1	2	0	2	4	0	4	6	0	6	8	0	8	10	0	10	12	0	12	14	0
1	0	2	0	2	0	4	0	4	0	6	0	6	0	8	0	8	0	10	0	10
1	4	4	2	0	2	8	8	4	0	4	12	12	6	0	6	16	16	8	0	8
1	0	0	0	2	0	2	0	0	0	4	0	4	0	0	0	6	0	6	0	0
1	6	4	4	8	2	0	2	12	8	8	14	4	0	4	18	12	12	20	6	0
1	0	6	0	8	0	2	0	2	0	12	0	14	0	4	0	4	0	18	0	20
1	8	0	4	4	0	12	2	0	2	16	0	8	8	0	20	4	0	4	24	0
1	0	6	0	0	0	8	0	2	0	2	0	12	0	0	0	14	0	4	0	4
1	10	8	6	4	4	8	12	16	2	0	2	20	16	12	8	8	14	20	26	4
1	0	0	0	12	0	14	0	0	0	2	0	2	0	0	0	22	0	24	0	0
1	12	8	6	12	4	4	14	8	12	20	2	0	2	24	16	12	22	8	8	24
1	0	10	0	6	0	0	0	8	0	16	0	2	0	2	0	20	0	12	0	0
1	14	0	8	0	0	4	4	0	0	12	0	24	2	0	2	28	0	16	0	0
1	0	10	0	6	0	18	0	20	0	8	0	16	0	2	0	2	0	20	0	12
1	16	12	8	16	6	12	4	4	14	8	22	12	20	28	2	0	2	32	24	16
1	0	0	0	16	0	12	0	0	0	14	0	22	0	0	0	2	0	2	0	0
1	18	12	10	8	6	20	18	4	4	20	24	8	12	16	20	32	2	0	2	36
1	0	14	0	0	0	6	0	24	0	26	0	8	0	0	0	24	0	2	0	2
1	20	0	10	8	0	0	20	0	4	4	0	24	0	0	12	16	0	36	2	0

Enlarging the possible size of the buckets and assigning a cherry tone to the length of the solutions leads to the following picture.

```
n = 50; B = Table[0, {n}, {n}];
Do[
 If[GCD[a, b] == 1, B[[a,b]] = B[[b,a]] = Length[buckets[a, b, n]] - 1],
 {a, n - 1}, {b, a + 1, n}];
ArrayPlot[B, ColorFunction → "CherryTones"]
```



Note that the values on the first row and columns account for the solutions $\{\{0, 0\}, \{0, 1\}\}$ and $\{\{0, 0\}, \{1, 0\}\}$. The bands of 2s above and below the main diagonal establish the solutions $\{\{0, 0\}, \{0, b\}, \{a, 1\}\}$ and $\{\{0, 0\}, \{a, 0\}, \{1, b\}\}$.

Perhaps the reader will find it interesting to prove that, if $n = \frac{b+1}{a}$ is an integer and $a < b$, then $B_{[\![a,b]\!]} = n$. What strategy does this result imply? What happens in the case when $\frac{b-1}{a}$ is an integer?

Note that the longest entry in the previous matrix corresponds to the case $a = 19$, $b = 21$ (or vice versa) of length 36.

```
buckets[19, 21, 50]
```

```
{{0, 0}, {19, 0}, {0, 19}, {19, 19}, {17, 21}, {17, 0},
 {0, 17}, {19, 17}, {15, 21}, {15, 0}, {0, 15}, {19, 15},
 {13, 21}, {13, 0}, {0, 13}, {19, 13}, {11, 21}, {11, 0},
 {0, 11}, {19, 11}, {9, 21}, {9, 0}, {0, 9}, {19, 9},
 {7, 21}, {7, 0}, {0, 7}, {19, 7}, {5, 21}, {5, 0}, {0, 5},
 {19, 5}, {3, 21}, {3, 0}, {0, 3}, {19, 3}, {1, 21}}
```

Inspection of this list suggests an easy procedure to solve the case where a is odd and $b = a + 2$ (they will always be coprime because a linear combination of them, namely $a((b - 1) / 2) - b((a - 1) / 2)$, equals 1). The length in this case will be equal to $2 (a - 1)$.

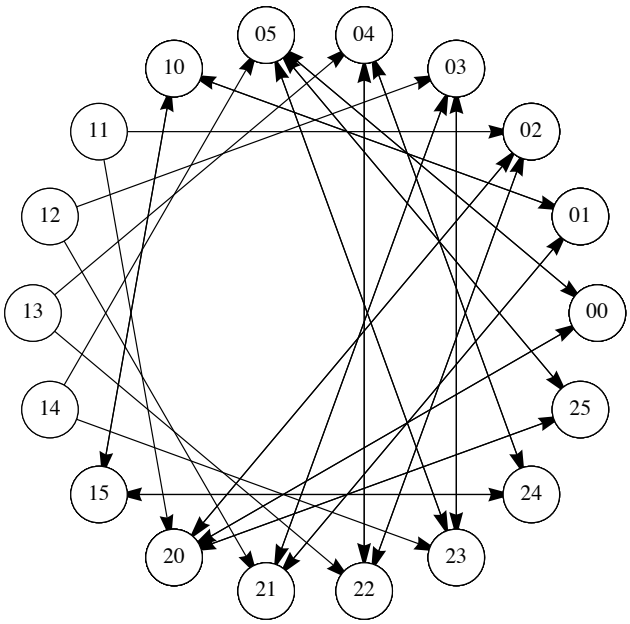
The next function helps to visualize the entire state space involved in solving the problem of the buckets. We associate a labeled disk with each state $\{a , b\}$ and place them at the vertices of a regular polygon. We draw an arrow from state i to state j if state j can be reached from state i. The following function `sG` computes the resulting graphic object.

```
sG[a_, b_] :=
 Module[{ind, pour, nextStates, n = (a + 1) (b + 1), t, c,
   i, j, k, s},

  ind[k_] := With[{{i = ⌊(k - 1)/(b + 1)⌋}, {i, k - (b + 1) i - 1}}];

  pour := {{0, j_} → {a, j}, {i_, 0} → {i, b}, {a, j_} → {0, j},
    {i_, b} → {i, 0},
    {i_, j_ /; j ≠ b} → If[b < i + j, {i + j - b, b}, {0, i + j}],
    {i_ /; i ≠ a, j_} → If[a < i + j, {a, i + j - a}, {i + j, 0}]};

  nextStates[state_] := Union[ReplaceList[state, pour]];

  t = 2. π/n;
  c = Table[{Cos[k t], Sin[k t]}, {k, 0, n - 1}];
```

```
m = Table[0, {n}, {n}];
Do[
 s = Map[((b + 1) First[#] + Last[#] + 1) &,
   nextStates[ind[k]]];
 Map[(m[[k, #]] = 1) &, s], {k, n}];

Table[
 If[m[[i,j]] == 1, {Arrow[{c[[i]], 0.96 c[[j]] + 0.04 c[[i]]}],
   {White, Disk[c[[i]], 0.1], Disk[c[[j]], 0.1], Black,
    Circle[c[[i]], 0.1], Circle[c[[j]], 0.1]},
   {Text[ind[i] /. {x_, y_} :> ToString[x] <> ToString[y],
     c[[i]]],
    Text[ind[j] /. {x_, y_} :> ToString[x] <> ToString[y],
     c[[j]]]}}, {}], {i, n}, {j, n}]]
```

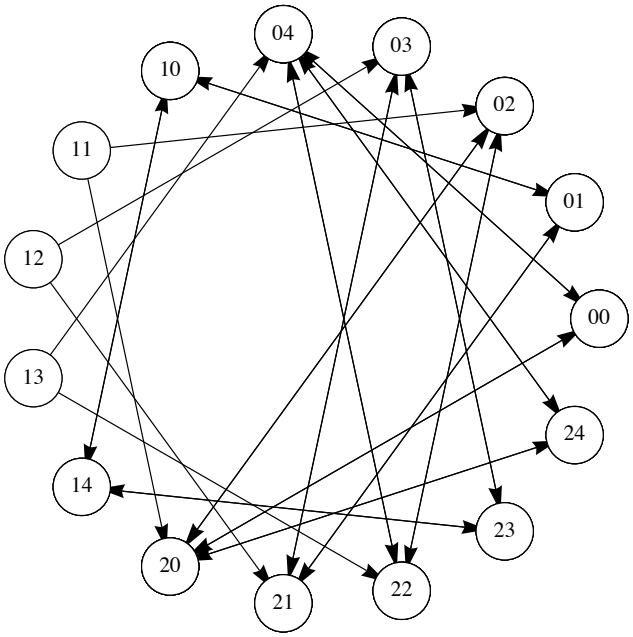
Here is the graph for buckets of capacities 2 and 5. Can the reader follow the path describing our previous solution? Note also that some states, like 11, are inaccessible.

```
{a, b} = {2, 5};
Graphics[sG[a, b]]
```



In the case $a = 2$, $b = 4$, the following graph shows that the goal states are inaccessible from 00. In fact this graph is the superposition of two graphs formed by those states reachable and unreachable from 00.

```
{a, b} = {2, 4};
Graphics[sG[a, b]]
```



□ A Bouquet of Buckets

The following code generalizes our problem to that of obtaining 1 liter from handling a number of buckets of given capacities.

```
buckets[b : {_Integer ..}, depth_] :=
 Module[{back, pour, getCandidates, ans = {}, lim = depth},

  back[x_] := 1 /; Length[x] > lim;
  back[x : {___, {___, 1, ___}}] :=
   (AppendTo[ans, x]; lim = Min[lim, Length[x]]);
  back[feasible : {{_Integer ..} ..}] := Module[{candidates},
    candidates = Select[getCandidates[Last[feasible]],
      Not[MemberQ[feasible, #]] &];
    Map[back[Join[feasible, {#}]] &, candidates]];

  pour := {{x___, {_, bi_}, y___} → {x, {0, bi}, y},
    {x___, {_, bi_}, y___} → {x, {bi, bi}, y},
    {x___, {i_, bi_}, y___, {j_, bj_}, z___} →
```

```
    If[bj < i + j, {x, {i + j - bj, bi}, y, {bj, bj}, z},
     {x, {0, bi}, y, {i + j, bj}, z}],
   {x___, {i_, bi_}, y___, {j_, bj_}, z___} →
    If[bi < i + j, {x, {bi, bi}, y, {i + j - bi, bj}, z},
     {x, {i + j, bi}, y, {0, bj}, z}]};

getCandidates[state : {_Integer ..}] :=
 Union[(First /@ #) & /@
   ReplaceList[Thread[List[state, b]], pour]];

back[{Table[0, {Length[b]}]}];
If[ans == {}, {}, Last[ans]]]
```

For instance, with three buckets of capacities 3, 11, and 20, the steps necessary to obtain 1 liter are as follows.

```
buckets[{3, 11, 20}, 15]
```

```
{{0, 0, 0}, {3, 0, 0}, {3, 0, 20},
 {3, 11, 9}, {3, 0, 9}, {3, 9, 0}, {1, 11, 0}}
```

Some problems are more time-consuming than others.

```
Timing[buckets[{3, 3, 11, 20}, 7]]
```

```
{24.944, {{0, 0, 0, 0}, {3, 0, 0, 0}, {3, 0, 0, 20},
  {3, 0, 11, 9}, {3, 0, 0, 9}, {3, 0, 9, 0}, {1, 0, 11, 0}}}
```

The capacity of the extra buckets can substantially affect this timing.

```
Timing[buckets[{3, 11, 11, 20}, 7]]
```

```
{3.77556, {{0, 0, 0, 0}, {3, 0, 0, 0}, {3, 0, 0, 20},
  {3, 11, 0, 9}, {3, 11, 9, 0}, {1, 11, 11, 0}}}
```

Having an extra bucket might not be useful in shortening the solution.

```
buckets[{2, 7}, 8]
buckets[{2, 7, 9}, 8]
```

```
{{0, 0}, {0, 7}, {2, 5}, {0, 5}, {2, 3}, {0, 3}, {2, 1}}
```

```
{{0, 0, 0}, {0, 7, 0}, {2, 5, 0},
 {0, 5, 2}, {2, 3, 2}, {0, 3, 4}, {2, 1, 4}}
```

But sometimes it does.

```
buckets[{2, 4, 7}, 7]
```

```
{{0, 0, 0}, {0, 0, 7}, {2, 0, 5}, {2, 4, 1}}
```

A better choice would be the following.

```
buckets[{2, 3, 7}, 10]
```

```
{{0, 0, 0}, {0, 3, 0}, {2, 1, 0}}
```

So, given a set of buckets, what would be the best choice if we are allowed to add one new bucket to the set? Our aim would be to obtain the shortest solution possible. Note that there is a maximum capacity for this new bucket beyond which we cannot shorten the solution any further. For example, if we start with buckets of capacities 2 and 7, adding a bucket of capacity 3 provides the shortest solution (the other two solutions are buckets of capacities 6 or 8); choosing a capacity greater than 17 is equivalent to using only the two original buckets. Our previous longest case $a = 19$, $b = 21$ would greatly benefit from the addition of an extra bucket of capacity 18. In general, looking at the previous matrix B we conclude that the addition of a bucket of capacity $a - 2$ substantially speeds up the process that starts with two buckets of capacities a, b $(a \leq b)$.

There are many fascinating variants of the problem of the buckets and we recommend references [4, 5]. They address problems like the following: we are given a filled vessel of capacity 12 and two empty vessels of capacities 9 and 5. How can we divide the liquid into two equal portions? By traversing paths in a network of equilateral triangles, a solution (or its absence) is found for the case of three jugs.

The Problem of the Toads and the Frogs

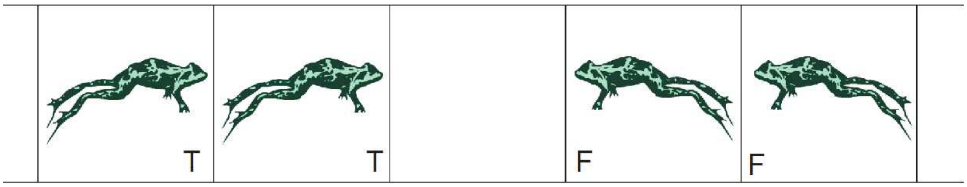
> The clever men at Oxford
> Know all that is to be knowed.
> But none of them knows half as much
> As intelligent Mr. Toad!
>
> *The Wind in the Willows*
> Kenneth Grahame, 1908

The Problem of the Buckets introduced in the previous section deals with traversing a state space and solving the problem of finding the shortest route connecting two particular nodes in this space. The problem here considers the same traversal, but with a different goal, providing an interesting new background from which to deduce properties that this time allows us to explicitly describe an optimal strategy.

Consider the Problem of the Toads and the Frogs: n toads and m frogs are placed in a linear grid of $n + m + 1$ squares. The toads are arranged in the first n squares and the frogs in

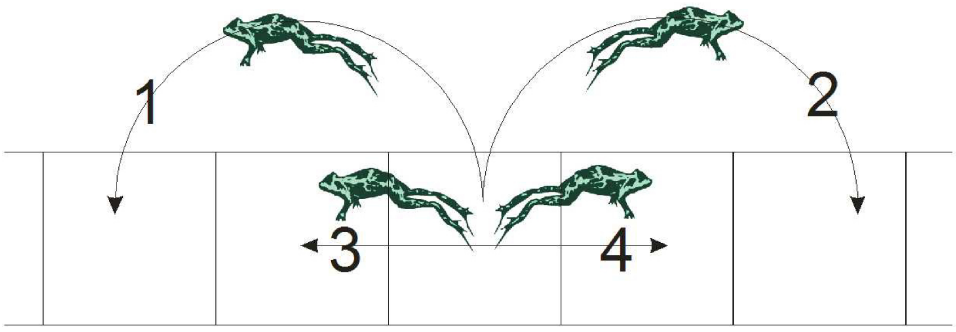
the last m squares, each pet facing each other with a single vacancy in between as depicted in Figure 1. From this initial configuration our problem is to transfer all toads to where the frogs are and vice versa by means of two kinds of movements performed sequentially by a single pet at a time; moreover, we want the fastest way of achieving this. Any pet can only move to the empty space if it happens to be next to it; if this is not the case, it can jump in the direction it is facing over an adjacent pet regardless of its type, provided that the empty space is at the square it lands on. At all times the process is performed within the limits of the squares. The movements resemble those in the game of checkers, but without any piece removal. Denoting a toad by T, a frog by F, and the empty space by an underscore _, we can describe a solution to the problem in Figure 1 by the following sequence of states, where an arrow indicates a transition to the next state:

TT_FF → T_TFF → TFT_F → TFTF_ → TF_FT →_FTFT → F_TFT → FFT_T → FF_TT.



▲ **Figure 1.** The Problem of the Toads and the Frogs for the case $n = m = 2$.

In order to describe a solution, we can regard each stage as a movement of the vacant square; this allows us to unambiguously recover the positions of all the pets at each stage and focus on the essential characteristics of the problem. Let us denote jumps to the left and right and moves to the left and right performed by the empty square by the digits 1, 2, 3, and 4, respectively (see Figure 2). Then the solution previously given for the case $n = m = 2$ can be described more concisely by the word 32411423, or equivalently as $3241^2 423$, where we use powers to denote repetitions of the same sequence. We will also denote this process as $\text{TT\_FF} \xrightarrow{3241^2 423} \text{FF\_TT}$.



▲ **Figure 2.** The movements of the toads and frogs can be interpreted as movements of the empty cell according to this code.

Let us make the following observations.

- If `w` is a solution, then `w /. {1 → 2, 2 → 1, 3 → 4, 4 → 3}` is also a solution.

- In any minimum-length solution, the symbols 1 and 2 and the symbols 3 and 4 cannot appear together. This condition is, however, not sufficient. The solution 1423142323132 for the case $n = m = 2$ complies with it even though it is not of minimal length.
- Despite appearances, $14 \neq 3$, that is, performing movement 1 and then movement 4 is not equivalent to performing movement 3. Similarly, $23 \neq 4$. If we denote by ε the empty word (equivalent to "do nothing"), we have $332 \neq \varepsilon$, $12 = \varepsilon$, $34 = \varepsilon$, $323 = 414$, and $(332)^2 = \varepsilon$.
- Any solution can be made to contain only the digits 2 and 3 or only the digits 1 and 4. This comes from the identities
  $1 = 33\,233, \quad 2 = 44\,144, \quad 3 = 14\,414, \quad 4 = 23\,323,$
  but their presence helps in the shortening of solutions.

We consider now the following productions corresponding to each of the movements of the empty space.

```
moves := {{x___, a_, b_, "_", y___} → {{x, "_", b, a, y}, 1},
    {x___, "_", a_, b_, y___} → {{x, b, a, "_", y}, 2},
    {x___, a_, "_", y___} → {{x, "_", a, y}, 3},
    {x___, "_", a_, y___} → {{x, a, "_", y}, 4}};
```

Function `toadsAndFrogs` takes a string describing a starting state and attempts to find a solution to the problem of swapping the positions of the frogs and toads. As in the Problem of the Buckets, it makes use of backtrack to traverse the state space up to a given depth.

```
a =.; b =.;
toadsQ[{"T" ..}] := True
toadsQ[_] := False
frogsQ[{"F" ..}] := True
frogsQ[_] := False

toadsAndFrogs[TF_String, depth_] :=
 Module[{back, moves, getCandidates, ans = {}, lim = depth},

  back[x_] := 1 /; Length[x] > lim;
  back[x : {___, {{F___, "_", T___}, _}}] :=
   (AppendTo[ans, x]; lim = Min[lim, Length[x]]) /;
    frogsQ[{F}] && toadsQ[{T}];
  back[feasible : {{{_String ..}, _Integer} ..}] :=
   Module[{candidates},
    candidates = Select[getCandidates[Last[feasible]],
      Not[MemberQ[First /@ feasible, First[#]]] &];
    Map[back[Join[feasible, {#}]] &, candidates]];

  moves :=
   {{x___, a_, b_, "_", y___} → {{x, "_", b, a, y}, 1},
    {x___, "_", a_, b_, y___} → {{x, b, a, "_", y}, 2},
    {x___, a_, "_", y___} → {{x, "_", a, y}, 3},
    {x___, "_", a_, y___} → {{x, a, "_", y}, 4}};
```

```
getCandidates[state : {{_String ..}, _Integer}] :=
 ReplaceList[First[state], moves];

back[{{Characters[TF], 0}}];
{StringJoin[First[#]], Last[#]} & /@ Last[ans]]
```

For instance, for the problem depicted in Figure 1 we obtain the actual sequence with the coding of the movements of the empty space involved as follows.

```
toadsAndFrogs["TT_FF", 20]
```

```
{{TT_FF, 0}, {TTF_F, 4}, {T_FTF, 1}, {_TFTF, 3}, {FT_TF, 2},
 {FTFT_, 2}, {FTF_T, 3}, {F_FTT, 1}, {FF_TT, 4}}
```

We can compute a table comprising all solutions for each one of the members of the state space.

```
space = StringJoin /@ Permutations[Characters["TT_FF"]];
Partition[
  Map[{#, u = toadsAndFrogs[#, 10]; FromDigits[Last /@ u]} & ,
   space], 6] // MatrixForm
```

TT_FF 41 322 314	TTF_F 1 322 314	TTFF_ 31 322 314	T_TFF 2 411 423	T_FTF 322 314	T_FFT 41 423
TFT_F 411 423	TFTF_ 11 423	TF_TF 4 411 423	TF_FT 1423	TFFT_ 331 423	TFF_T 31 423
_TTFF 42 411 423	_TFTF 22 314	_TFFT 2414	_FTTF 442 314	_FTFT 423	_FFTT 2
FTT_F 41 414	FTTF_ 1414	FT_TF 2314	FT_FT 414	FTFT_ 314	FTF_T 14
F_TTF 42 314	F_TFT 23	F_FTT 4	FFTT_ 1	FFT_T 3	FF_TT 0

□ Amphibia in Theory

Unlike the space of buckets, the Problem of the Toads and Frogs has an extensive state space including $\frac{(n+m+1)!}{n!\,m!}$ states, in which any state can be reached from any other one. Specifically, let $\mathcal{TF}_{nm}$ be the set of all possible configurations of n toads and m frogs. Then the following property applies.

Theorem 1. *For all $A, B \in \mathcal{TF}_{nm}$, there exists a word w such that $A \overset{w}{\to} B$.*

Proof. Because of the nature of the transitions, the proof relies in being able to find a word h such that $\_T^n F \xrightarrow{h} \_FT^n$. Such a transformation is given by:

$$h = \begin{cases} 2^{(n+1)/2}\, 3\, (14)^{n-1}\, 3 & n \text{ odd} \\ 2^{n/2}\, 4\, (14)^n\, 3 & n \text{ even} \end{cases} . \square$$

(We use the symbol □ to indicate end of proof.)

Note that this proof does not provide a way to obtain the shortest sequence we seek.

We need to apply words describing moves to configurations of toads and frogs. These words are juxtapositions of powers of symbols taken from the set {1, 2, 3, 4}, the powers being used as a shorthand to indicate repetitions of their respective base. Words will be described by a list of pairs similarly as those expressing the canonical factorization of a positive integer number. Because these words grow quite rapidly, we will work with them factorized, their type being `{{_String .., _Integer} ..}`. For example, if $n = 11$, the transformation mentioned in the previous proof will be structured as `h = {{"2", 6}, {"3", 1}, {"14", 10}, {"3", 1}}`. States will have the type `{_String ..}`, where each of their members will be one of "T", "F", or "_" and only one occurrence of the underscore is allowed and required. Their action by a word will be computed by the function `action`. In order to display patterns and words more concisely, we use predicates `showPattern` and `showWord` (for an alternative way regarding the design of the latter, see [6]).

```
action[pattern_, word_] := Module[{w, step},

  step[{x___, a_, b_, "_", y___}, "1"] := {x, "_", b, a, y};
  step[{x___, "_", a_, b_, y___}, "2"] := {x, b, a, "_", y};
  step[{x___, a_, "_", y___}, "3"] := {x, "_", a, y};
  step[{x___, "_", a_, y___}, "4"] := {x, a, "_", y};

  w = Flatten[Map[Table[Characters[First[#]], {Last[#]}] &,
      word]];
  Fold[step, Characters[pattern], w]]

showPattern[p_] := StringJoin[p]

showWord[w_] :=
 DisplayForm[
  RowBox[
   Map[
    SuperscriptBox[First[#], If[Last[#] == 1, "", Last[#]]] &,
    w]]]
```

To verify our previous assertion we proceed as:

```
action["_TTTTTTTTTTTF",
  {{"2", 6}, {"3", 1}, {"14", 10}, {"3", 1}}] //
 showPattern

_FTTTTTTTTTTT
```

Definition 1. *Let w_n be the following sequence*

$$w_0 = \varepsilon$$
$$w_n = \begin{cases} w_{n-1}\, 41^n & n \text{ even} \\ w_{n-1}\, 32^n & n \text{ odd} \end{cases}$$

the family w_n is readily coded as:

```
w0 := {{"", 1}}
wn_?EvenQ := Join[wn-1, {{"4", 1}, {"1", n}}]
wn_ := Join[wn-1, {{"3", 1}, {"2", n}}]
```

The first six members of this sequence are:

```
Table[showWord[wn], {n, 6}]
```

$$\{\ 3\ 2\ ,\ \ 3\ 2\ 4\ 1^2,\ \ 3\ 2\ 4\ 1^2\ 3\ 2^3,\ \ 3\ 2\ 4\ 1^2\ 3\ 2^3\ 4\ 1^4,$$
$$3\ 2\ 4\ 1^2\ 3\ 2^3\ 4\ 1^4\ 3\ 2^5,\ \ 3\ 2\ 4\ 1^2\ 3\ 2^3\ 4\ 1^4\ 3\ 2^5\ 4\ 1^6\}$$

The word w_n is used on our problem as follows.

Theorem 2. $T^n\_F^n \xrightarrow{w_n} \begin{cases} \_(FT)^n & n \text{ even} \\ (FT)^n\_ & n \text{ odd} \end{cases}$.

Proof. (induction on n) For $n = 1$, we have $T^1\_F^1 \xrightarrow{w_1=32} FT\_$.

Let us assume the result is true for the index n. If n is even, $T^{n+1}\_F^{n+1} = TT^n\_F^n\, F \xrightarrow{w_{n+1}=w_n\, 32^{n+1}} X$, where X is such that $T\_(FT)^n\, F \xrightarrow{32^{n+1}} X$, that is, $X = (FT)^{n+1}\_$.

The case n odd is handled similarly. □

For example, the action of sequence w_5 on the case $n = m = 5$ is:

```
showPattern[action["TTTTT_FFFFF", w5]]

FTFTFTFTFT_
```

Denoting the reverse of w by w^R, we have the following result.

Theorem 3. $\left.\begin{matrix}(TF)^n\_ & n\ even\\ \_\,(TF)^n & n\ odd\end{matrix}\right\} \xrightarrow{w_n^R} F^n\,\_T^n.$

Proof. According to the definition:

$$w_0^R = \varepsilon$$

$$w_n^R = \begin{cases} 2^n\, 3\, w_n^R & n \text{ even} \\ 1^n\, 4\, w_n^R & n \text{ odd} \end{cases}.$$

(induction on n) For $n = 1$, we have $\_\,(\mathrm{TF})^1 = \_\mathrm{TF} \xrightarrow{w_1^R=23} \mathrm{F}\_\mathrm{T}$.

Assuming the result is true for index n, we have two cases.

n even:

$$(\mathrm{TF})^{n+1}\,\_ \xrightarrow{1^{n+1}} \_\,(\mathrm{FT})^{n+1} \xrightarrow{4} \mathrm{F}\_\,(\mathrm{TF})^n\,\mathrm{T} \xrightarrow{w_n^R} \mathrm{FF}^n\,\_\mathrm{T}^n\,\mathrm{T}$$

n odd:

$$\_\,(\mathrm{TF})^{n+1} \xrightarrow{2^{n+1}} (\mathrm{FT})^{n+1}\,\_ \xrightarrow{3} \mathrm{F}(\mathrm{TF})^n\,\_\mathrm{T} \xrightarrow{w_n^R} \mathrm{FF}^n\,\_\mathrm{T}^n\,\mathrm{T}.\ \square$$

This result is used to recover from the effect of w_n. For example:

```
showPattern[action["TFTFTFTF_", Reverse[w4]]]
```

```
FFFF_TTTT
```

Unifying Theorems 2 and 3, we can explicitly solve our problem for the case of an equal number of pets.

Theorem 4. $T^n\,\_F^n \xrightarrow{w_n\, S\, w_{n-1}^R} F^n\,\_T^n$ *where* $S = \begin{cases} 4 & n\ even \\ 3 & n\ odd \end{cases}$.

Proof. As

$$\_\,(\mathrm{FT})^n \xrightarrow{4} \mathrm{F}\_\,(\mathrm{TF})^{n-1}\,\mathrm{T}$$

$$(\mathrm{FT})^n\,\_ \xrightarrow{3} \mathrm{F}\,(\mathrm{TF})^{n-1}\,\_\mathrm{T}$$

it suffices to show that

$$\begin{matrix}(\mathrm{TF})^{n-1} & \xrightarrow{w_{n-1}^R} \mathrm{F}^{n-1} & \text{if } n \text{ is even} \\ (\mathrm{TF})^{n-1}\,\_ & \xrightarrow{w_{n-1}^R} \mathrm{F}^{n-1}\,\_\mathrm{T}^{n-1} & \text{if } n \text{ is odd}\end{matrix}.\ \square$$

Testing this result for the case $n = m = 8$, we obtain:

```
showPattern[action["TTTTTTTT_FFFFFFFF",
  Join[w8, {{"4", 1}}, Reverse[w7]]]]

FFFFFFFF_TTTTTTTT
```

The case where we have an unbalanced number of pets is more involved. First, we deduce the following result from Theorem 4.

Corollary 1. *Let $k = Min[n, m]$. Then*

$$\mathrm{T}^n\_\mathrm{F}^m \xrightarrow{w_k} \begin{cases} n \le m \begin{cases} \_\,(\mathrm{FT})^n\,\mathrm{F}^{m-n} & n \text{ even} \\ (\mathrm{FT})^n\,\_\mathrm{F}^{m-n} & n \text{ odd} \end{cases} \\ m < n \begin{cases} \mathrm{T}^{n-m}\,\_\,(\mathrm{FT})^m & m \text{ even} \\ \mathrm{T}^{n-m}\,(\mathrm{FT})^m\,\_ & m \text{ odd} \end{cases} \end{cases}$$

Definition 2. *Let $x_{n,m}$ be the word defined as:*

$$x_{n,0} = \varepsilon$$

$$x_{n,m} = \begin{cases} 42^n\, x_{n,m-1} & m \text{ even} \\ 41^n\, x_{n,m-1} & m \text{ odd} \end{cases}.$$

Let $\overline{x}_{n,m}$ stand for the word $x_{n,m}$ acted upon by the transposition

```
barX = {"1" → "2", "2" → "1"};
```

Similarly, let $\overline{w}_n$ stand for the previously introduced word w_n after applying the transposition

```
barW = {"1" → "2", "2" → "1", "3" → "4", "4" → "3"};
```

The family $x_{n,m}$ can be readily coded as:

```
xn_,0 := {{"", 1}}
xn_,m_?EvenQ := Join[{{"4", 1}, {"2", n}}, xn,m-1]
xn_,m_ := Join[{{"4", 1}, {"1", n}}, xn,m-1]
```

For example, the first six members of $x_{n,3}$ are:

```
Table[showWord[xn,3], {n, 6}]
```

$$\{4\ 1\ 4\ 2\ 4\ 1\ ,\ 4\ 1^2\ 4\ 2^2\ 4\ 1^2\ ,\ 4\ 1^3\ 4\ 2^3\ 4\ 1^3\ ,$$
$$4\ 1^4\ 4\ 2^4\ 4\ 1^4\ ,\ 4\ 1^5\ 4\ 2^5\ 4\ 1^5\ ,\ 4\ 1^6\ 4\ 2^6\ 4\ 1^6\ \}$$

and the first six members of the family $\overline{w}_n$ are:

```
Table[showWord[w_n /. barW], {n, 6}]
```

```
{  4 1 ,  4 1 3 2^2,  4 1 3 2^2 4 1^3,  4 1 3 2^2 4 1^3 3 2^4,
   4 1 3 2^2 4 1^3 3 2^4 4 1^5,  4 1 3 2^2 4 1^3 3 2^4 4 1^5 3 2^6}
```

We state Theorems 5 through 7 without their proofs, which are somewhat laborious. They progressively allow us to obtain an explicit word solving our general problem.

Theorem 5. $\left.\begin{matrix} \_\,(TF)^n & n\ even \\ (TF)^n\,\_ & n\ odd \end{matrix}\right\} \xrightarrow{\overline{w}_n^R} F^n\,\_\,T^n.$

For example, for the case $n = 5$ we have:

```
showPattern[action["TFTFTFTFTF_", Reverse[w_5] /. barW]]
```

```
FFFFF_TTTTT
```

Theorem 6. *Let $0 < n \le m$. Then*

$$\mathrm{T}^n\,\_\,\mathrm{F}^m \xrightarrow{w_n\, x_{n,m-n}} \mathrm{F}^{m-n}\,\_\,(\mathrm{FT})^n \quad m \text{ even}$$

$$\mathrm{T}^n\,\_\,\mathrm{F}^m \xrightarrow{w_n\, \overline{x}_{n,m-n}} \mathrm{F}^{m-n}\,(\mathrm{FT})^n\,\_ \quad m \text{ odd}.$$

This case involves the action of families w and x together. Consider, for example, the following two cases.

```
showPattern[action["TTT_FFFFFFFF", Join[w_3, x_3,5]]]
```

```
FFFFF_FTFTFT
```

```
showPattern[action["TT_FFFFFFF", Join[w_2, x_2,5 /. barX] ]]
```

```
FFFFFFTFT_
```

Now we only need to consider word A as given in the following theorem.

Theorem 7. *Let $0 < n \le m$. Then $T^n\,\_\,F^m \xrightarrow{w_n\, A} F^m\,\_\,T^n$, where*

$$A = \begin{cases} x_{n,m-n}\, 4\, w_{n-1}^R & n \text{ even},\ m \text{ even} \\ \overline{x}_{n,m-n}\, 3\, \overline{w}_{n-1}^R & n \text{ even},\ m \text{ odd} \\ x_{n,m-n}\, 4\, \overline{w}_{n-1}^R & n \text{ odd},\ m \text{ even} \\ \overline{x}_{n,m-n}\, 3\, w_{n-1}^R & n \text{ odd},\ m \text{ odd} \end{cases}.$$

Examples covering each one of these four cases follow.

```
showPattern[action["TTTT_FFFFFF",
  Join[w4, x4,2, {{"4", 1}}, Reverse[w3]]]]
```

```
FFFFFF_TTTT
```

```
showPattern[action["TT_FFFFFFFFF",
  Join[w2, x2,7 /. barX, {{"3", 1}}, Reverse[w1] /. barW]]]
```

```
FFFFFFFFF_TT
```

```
showPattern[action["TTT_FFFF",
  Join[w3, x3,1, {{"4", 1}}, Reverse[w2] /. barW]]]
```

```
FFFF_TTT
```

```
showPattern[action["TTTTT_FFFFFFFFFFF",
  Join[w5, x5,6 /. barX, {{"3", 1}}, Reverse[w4]]]]
```

```
FFFFFFFFFFF_TTTTT
```

We finally arrive at our main result.

Corollary 2. *We can solve the n-Toads and m-Frogs problem using* $\frac{n(n+m)}{2}$ *jumps and* $n+m$ *moves.*

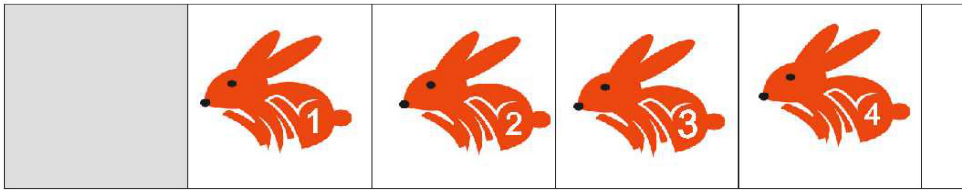
Our results provide the word 3241423 to solve T_FFF and the word 32411423 to solve TT_FF, which we previously attacked exhaustively with the help of backtrack.

Using a different approach to reach a different aim, Berlekamp et al. [7] apply the theory of surreal numbers to analyze the Problem of Toads and Frogs.

The Problem of the Rabbits

> Oh dear! Oh dear!
> I shall be too late!
>
> The white rabbit in *Alice in Wonderland*
> Lewis Carroll, 1865

Consider the following interesting variation of the Problem of the Toads and Frogs in the previous section. There are n rabbits confined in a grid of size $n+1$. All rabbits are labeled sequentially and are initially arranged as shown in Figure 3, leaving only a left-most empty space labeled 0. They can jump and move as in the Problem of the Toads and Frogs. The Problem of the Rabbits consists of orchestrating the series of state changes needed to take the rabbits from positions $01\,234\ldots n$ to positions $0\,n(n-1)\ldots 21$ in the shortest number of moves.



▲ **Figure 3.** The problem of the rabbits for the case $n = 4$. Although they are displayed here looking to the left, they are free to turn and jump about within the confines of their world.

Any state is reachable from any other and the search space grows rapidly with n, which renders a trial-and-error approach impractical. The following function `rabbits` adapts our backtrack paradigm to this problem. Given the number n of rabbits and the depth of search, it exhaustively looks for a solution. The coding that we follow is the same as the one adopted to describe the empty space in the previous section as depicted in Figure 2.

```
rabbits[n_Integer, depth_] :=
 Module[{goal, back, move, getCandidates, ans = {},
   lim = depth},

  back[x_] := 1 /; Length[x] > lim;
  back[x : {___, {g_, _}}] :=
   (AppendTo[ans, x]; lim = Min[lim, Length[x]]) /;
    g == goal;
  back[feasible : {{{_String ..}, _Integer} ..}] :=
   Module[{candidates},
    candidates = Select[getCandidates[Last[feasible]],
      Not[MemberQ[First /@ feasible, First[#]]] &];
    Map[back[Join[feasible, {#}]] &, candidates]];

  move := {{x___, a_, b_, "0", y___} → {{x, "0", b, a, y}, 1},
    {x___, "0", a_, b_, y___} → {{x, b, a, "0", y}, 2},
    {x___, a_, "0", y___} → {{x, "0", a, y}, 3},
    {x___, "0", a_, y___} → {{x, a, "0", y}, 4}};

  getCandidates[state : {{_String ..}, _Integer}] :=
   ReplaceList[First[state], move];

  goal = Join[{"0"}, ToString /@ Range[n, 1, -1]];
  back[{{ToString /@ Range[0, n], 0}}];
  If[ans == {}, {}, Last[ans]]]
```

For example, if there are two rabbits we obtain:

```
u = rabbits[2, 10];
Map[{StringJoin[First[#]], Last[#]} &, u]
```

```
{{012, 0}, {102, 4}, {120, 4}, {021, 1}}
```

With three rabbits, the solution sequence grows as:

```
u = rabbits[3, 10];
Map[{StringJoin[First[#]], Last[#]} &, u]

{{0123, 0}, {2103, 2}, {2130, 4},
 {2031, 1}, {2301, 4}, {0321, 1}}
```

Compiling a table of solutions for the cases of up to five rabbits, we obtain:

```
Table[u = rabbits[n, 18]; {n, FromDigits[Last /@ u]},
  {n, 2, 5}] // MatrixForm
```

$$\begin{pmatrix} 2 & 441 \\ 3 & 24\,141 \\ 4 & 4\,241\,142\,411 \\ 5 & 2\,441\,322\,311\,422\,311 \end{pmatrix}$$

To get some insight into the problem, consider the following results.

Definition 3. *Let X_n be the word defined as: $X_1 = 4$, $X_2 = 2$ and*

$$X_n = \begin{cases} 2^{n/2}\,3\,(14)^{n-2}\,2^{(n/2)-1}\,4\,(14)^{n-2}\,X_{n-2} & n \text{ even} \\ 2^{(n-1)/2}\,4\,(14)^{n-1}\,2^{(n-3)/2}\,4\,(14)^{n-3}\,X_{n-2} & n \text{ odd} \end{cases}.$$

Theorem 8. $0123\ldots n \xrightarrow{X_n\,3^n} 0\,n\,(n-1)\,(n-2)\ldots 21.$

Proof. For n even:

$$\begin{aligned} &012\ldots n \xrightarrow{2^{n/2}} 214\,365\ldots n\,(n-1)\,0 \\ &\xrightarrow{3} 214\,365\ldots n\,0\,(n-1) \\ &\xrightarrow{(14)^{n-2}} n\,0\,214\,365\ldots (n-2)\,(n-3)\,(n-1) \\ &\xrightarrow{2^{(n/2)-1}} n\,123\ldots (n-2)\,0\,(n-1) \\ &\xrightarrow{4} n\,123\ldots (n-2)\,(n-1)\,0 \\ &\xrightarrow{(14)^{n-2}} n\,(n-1)\,0123\ldots (n-2) \end{aligned}$$

the case n odd is handled similarly. □

Although useful, this result gives a large upper bound for the solution. For instance, instead of the 21 movements required to get from 0123456 to 0654321, we need 43 by using Theorem 8. Just to appreciate the subtleties involved, let us consider the steps used in applying one of the shortest solutions 222311322231132223113:

$$\begin{aligned} &0\,123\,456 \xrightarrow{222} 2\,143\,650 \xrightarrow{311} 2\,041\,635 \xrightarrow{322} 4\,261\,035 \xrightarrow{231} \\ &\quad 4\,260\,513 \xrightarrow{132} 6\,402\,513 \xrightarrow{223} 6\,452\,301 \xrightarrow{113} 0\,654\,321. \end{aligned}$$

If instead of describing the movement of the 0 marker, we explicitly mention the label of the rabbit that has to move/jump, we have the following result.

Theorem 9. *Let e be the increasing sequence of even labels* 0246 … *and o the decreasing sequence of odd labels* … 531 *of the* `n` *rabbits involved in the rabbits problem.*

Then the following holds:

$$0123\ldots n \xrightarrow{(e\,o)^{n+1}} 0123\ldots n$$

$$0123\ldots n \xrightarrow{(e\,o)^{n/2}\,e} 0\,n\,(n-1)\ldots 321 \qquad n \text{ even}$$

$$0123\ldots n \xrightarrow{(e\,o)^{(n+1)/2}\,e} n(n-1)\ldots 210 \qquad n \text{ odd.}$$

This provides sequence $(246\,531)^3\,246$ as a 21-step solution for the case $n = 6$. In general, it provides a sequence significantly smaller than the one given by Theorem 8.

Once the training of our rabbits is over, they can be put in a circular board connecting the last square to the first one to study their reactions to a more difficult setup. This new arrangement is reflected in the following list of transformations that gives rise to the following function `Crabbits` (circular rabbits).

```
moves := {{"0", x___, a_, b_} → {{"0", b, a, x}, 1},
    {"0", a_, b_, x___} → {{"0", x, b, a}, 2},
    {"0", x___, a_} → {{"0", a, x}, 3},
    {"0", a_, x___} → {{"0", x, a}, 4}};


Crabbits[n_Integer, depth_] :=
 Module[{goal, back, moves, getCandidates, ans = {},
   lim = depth},

  back[x_] := 1 /; Length[x] > lim;
  back[x : {___, {g_, _}}] :=
   (AppendTo[ans, x]; lim = Min[lim, Length[x]]) /;
    g == goal;
  back[feasible : {{{_String ..}, _Integer} ..}] :=
   Module[{candidates},
    candidates = Select[getCandidates[Last[feasible]],
      Not[MemberQ[First /@ feasible, First[#]]] &];
    Map[back[Join[feasible, {#}]] &, candidates]];

  moves := {{"0", x___, a_, b_} → {{"0", b, a, x}, 1},
    {"0", a_, b_, x___} → {{"0", x, b, a}, 2},
    {"0", x___, a_} → {{"0", a, x}, 3},
    {"0", a_, x___} → {{"0", x, a}, 4}};

  getCandidates[state : {{_String ..}, _Integer}] :=
   ReplaceList[First[state], moves];

  goal = Join[{"0"}, ToString /@ Range[n, 1, -1]];
```

```
    back[{{ToString /@ Range[0, n], 0}}];
    If[ans == {}, {}, Last[ans]]]
```

The results for up to six rabbits are:

```
Table[u = Crabbits[n, 15];
  {n, FromDigits[Last /@ u]}, {n, 2, 6}] // MatrixForm
```

$$\begin{pmatrix} 2 & 4 \\ 3 & 2 \\ 4 & 4422 \\ 5 & 422\,232 \\ 6 & 44\,422\,322\,322 \end{pmatrix}$$

Let us note that the symbol "0" in the previous transformation `move` is not really necessary. In looking at the resulting 0-less transformations for the first time, who would realize that these strange transformations correspond to movements performed by rabbits in a circular one-dimensional board!

Acknowledgments

This work was completed while the author was a visiting scholar at Wolfram Research, Inc., whose assistance and enthusiastic support is gratefully acknowledged. I would also like to thank the University of Queretaro in Mexico for their continued support.

References

[1] D. E. Knuth, "Estimating the Efficiency of Backtrack Programs" in *Selected Papers on the Analysis of Algorithms*, Stanford, CA: CSLI Publications, 2000, pp. 55–75.

[2] S. S. Skiena, *The Algorithm Design Manual*, New York: Springer-Verlag, 1997.

[3] S. S. Skiena, *Implementing Discrete Mathematics: Combinatorics and Graph Theory in Mathematica*, Reading, MA: Addison-Wesley, 1990.

[4] H. S. M. Coxeter and S. L. Greitzer, *Geometry Revisited*, Washington, DC: The Mathematical Association of America, 1967.

[5] T. H. O'Beirne, "Jug and Bottle Department," in *Puzzles and Paradoxes*, New York: Oxford University Press, 1965 pp. 49–75.

[6] P. Abbott, ed., "Tricks of the Trade," *The Mathematica Journal*, **6**(4), Fall 1996, pp. 17–26.

[7] E. R. Berlekamp, J. H. Conway, and R. Guy, *Winning Ways: For Your Mathematical Plays*, Vol. 2, London: Academic Press, 1982.

J. Rangel-Mondragón, "Buckets and Jumping Pets," *The Mathematica Journal*, 2011.
dx.doi.org/10.3888/tmj.11.1-3.

About the Author

Jaime Rangel-Mondragón received M.Sc. and Ph.D. degrees in applied mathematics and computation from the University College of North Wales in Bangor, UK. He has been a visiting scholar at Wolfram Research, Inc. and held research positions in the Faculty of Informatics at UCNW, the College of Mexico, the Center of Research and Advanced Studies, the Monterrey Institute of Technology, the Queretaro Institute of Technology, and the University of Queretaro in Mexico, where he is presently a member of the Faculty of Informatics. A prolific contributor to *MathSource*, he currently writes the column *Geometric Themes* for the *Mathematica in Education and Research* journal. His research interests include recreational mathematics, combinatorics, the theory of computing, computational geometry, and functional languages.

Jaime Rangel-Mondragón
Universidad Autónoma de Querétaro
Mexico
jrangelmondragon@gmail.com