sciendo FCT Fundação para a Ciência e a Tecnologia CIUHCT Ludus

Two eggs any style
generalizing egg-drop experiments

Harold R. Parks[1] *and Dean C. Wills*[2]

[1] *Department of Mathematics, Oregon State University, Corvallis, Oregon 97331 USA,* hal.parks@oregonstate.edu
[2] *AppDynamics,, San Francisco, California 94107 USA,* dean@lifetime.oregonstate.edu

Abstract

The egg-drop experiment introduced by Konhauser, Velleman, and Wagon, later generalized by Boardman, is further generalized to two additional types. The three separate types of egg-drop experiment under consideration are examined in the context of binary decision trees. It is shown that all three types of egg-drop experiment are binary decision problems that can be solved efficiently using a non-redundant algorithm—a class of algorithms introduced here. The preceding theoretical results are applied to the three types of egg-drop experiment to compute, for each, the maximum height of a building that can be dealt with using a given number of egg-droppings.

1 Introduction

We generalize the famous egg-drop experiment introduced in [KVW96].

> "Suppose we wish to know which windows in a 36-story building are safe to drop eggs from, and which will cause the eggs to break on landing. ... Suppose two eggs are available. What is the least number of egg-droppings that is guaranteed to work in all cases?"

This experiment was used as an exposition of dynamic programming techniques in [Sni03], examined as a form of weighted binary search in [Wil09], and assigned as a exercise in [Ski09]; it has also enjoyed wide circulation on the internet as a possible interview question for programmers as either the egg or light bulb dropping problem [AIP]. In the typical internet version of the problem, the building has been increased in size to be 100-stories tall.

DOI 10.2478/rmm-2025-0001

Since real eggs seldom survive even being dropped a few feet, some clarification is called for: We agree that the two eggs are identically strong, each can be reused unless it breaks, an egg that breaks from one height will break from any greater height, and an egg that survives from one height will survive from any lower height.

Some years after the problem appeared in [KVW96], Michael E. Boardman [Boa04] came up with his own solution and, at the urging of a colleague, went on to use his method to solve the problem when the egg-drop experiment starts with k-eggs. So indeed we will be further generalizing beyond Boardman's generalization of the original egg-drop experiment.

Boardman's approach revolves around carefully planning ahead to consider what will need to be done when an egg breaks. We might say this is the tactical approach. We want to be more strategic and optimize over the full range of possible outcomes.

Our approach to solving the problem is based on thinking about a binary decision tree that encapsulates a strategy. If the egg breaks, then the tree branches to the left. If the egg survives the fall, then the tree branches to the right. The next node visited either determines where an egg should be dropped, or is a terminal node that reveals the strength of the egg.

Using the binary tree as our guide we are able to handle some interesting variations on the egg-drop experiment with relative ease. The two variations that lead to the nicest results are the following:

- **Replacement Eggs.** The supply of eggs is restored to the original number k whenever the egg that is dropped does not break.
- **Bonus Eggs.** A new egg is received whenever the egg that is dropped does not break.

We will refer to Boardman's k-egg version of the original problem as Standard Eggs:

- **Standard Eggs.** No new eggs will be forthcoming; you break it, you lose it.

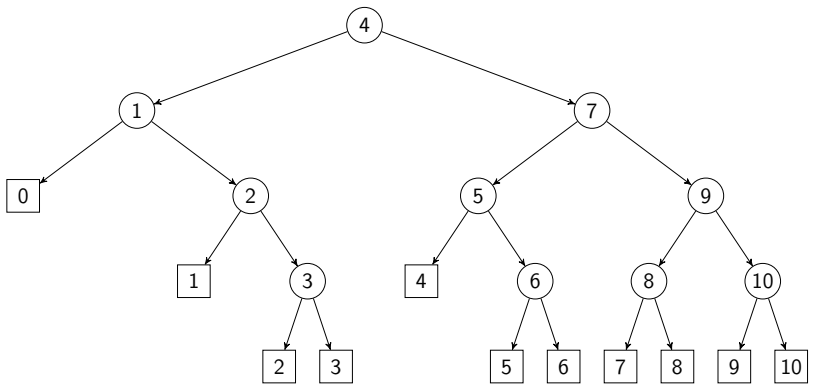

Figure 1: Starting with Two Eggs. No Replacement Eggs. No Bonus Eggs.

The tree in Figure 1 is a binary decision tree solution for two Standard Eggs with a 10-story building. Since 4 is the number of the root node, the first egg should be dropped from the 4th floor window. If the egg breaks, we follow the left branch to the node numbered 1 and that tells us to drop the next egg from the 1st floor window. On the other hand, if the original egg does not break, we follow the right branch from the root to the node numbered 7 and that tells us to drop the next egg from the 7th floor window. To summarize: A node with a number in a circle indicates the experiment to perform, i.e., the floor from which to drop the egg; a node with a number in a square (always a leaf) gives the solution, i.e., the strength of the eggs.

We can see from the tree in Figure 1 that four egg drops will suffice for a 10-story building. We can also see that three egg drops can only be enough for a 7-story building, because if we remove the nodes at depth 4, then there only 7 leaves left to represent the strength of the eggs.

In the next section, we define the notions of a normal binary decision problem (Definition 2.1) and a non-redundant algorithm for solving such problems (Definition 2.4). We then show in Theorem 2.7 that any non-redundant algorithm for solving a normal problem can be represented by a full binary search tree. Of course, the point of that work is that it applies to the three egg-drop problems described above (and certainly to many others not yet devised). In the third section, we use the tree representation of algorithms to show, for all three variations, the maximum height of a building that can be dealt with using a given number of egg-droppings. See Figure 2. With that information in hand, one can answer the fundamental question of egg-drop experiments: "What is the least number of egg-droppings that is guaranteed to work in all cases?"

2 Theory

In this section, we consider binary decision problems on integers, i.e., problems where we have the ability to make a binary decision at each stage of execution of an algorithm based on an integral input. We call these decisions **experiments**. Of course, we will later apply this theory to egg-drop problems. For such problems, it is appropriate to add the following qualifications which simplify and focus the discussion.

Definition 2.1. *We will call a binary decision problem* ***normal*** *if it meets these criteria:*

(1) *(**complete**) The solution set is the integers between* 0 *and some finite* N *inclusive. Every solution is possible and constitutes an instance of the problem.*

(2) *(**partition**) If an experiment is performed at an integer* x*, success indicates the solution is within the set* $[x, N]$*, and failure indicates that the solution is within the set* $[0, x-1]$*.*

Definition 2.2. *We say that* x^* *is the solution of a normal binary decision problem if both of the following hold:*

(1) $x^* = 0$*, or experiment* x^* *is a success,*

Two Eggs									
Drops	1	2	3	4	5	6	7	8	*OEIS*
Standard	1	3	6	10	15	21	28	36	A000217
Replacement	1	3	6	11	19	32	53	87	
Bonus	1	3	6	12	22	42	77	147	

Three Eggs									
Drops	1	2	3	4	5	6	7	8	*OEIS*
Standard	1	3	7	14	25	41	63	92	A004006
Replacement	1	3	7	14	27	51	95	176	
Bonus	1	3	7	14	28	53	103	194	

Four Eggs									
Drops	1	2	3	4	5	6	7	8	*OEIS*
Standard	1	3	7	15	30	56	98	162	A055795
Replacement	1	3	7	15	30	59	115	223	
Bonus	1	3	7	15	30	60	116	228	

Figure 2: Maximum height of a building that can be dealt with for small values.

and

(2) $x^* = N$*, or experiment* $x^* + 1$ *is a failure.*

Definition 2.3. *We say that an algorithm* ***solves a problem*** *if the algorithm is finite and arrives at a solution for every instance of the problem.*

Definition 2.4. *We will call a binary decision algorithm* ***non-redundant*** *if no experiment is performed that is guaranteed to either succeed or fail.*

Deterministically choosing experiments given the results of previous experiments allows us to form a binary decision tree[1] over all possible experiments. In this way, we are performing a search of an ordered table of experiments from 1 to N, trying to determine where the solution is in 0 to N.

We can use the notation $\textcircled{x}$ to indicate that an experiment is to be performed at x, and the notation $\boxed{x}$ to indicate that the conclusion of our sequence of experiments is the solution x. We can define an order on

$$\left\{ \textcircled{x} : 1 \leqslant x \leqslant N \right\} \cup \left\{ \boxed{x} : 0 \leqslant x \leqslant N \right\}$$

[1]We follow the convention that the smallest binary tree consists of a single root node.

that conforms to the constraints of Definition 2.2 first by the value of the number then with $\textcircled{x} < \boxed{x}$.

We then find ourselves searching for the solution in an ordered table:

$$\boxed{0}\,,\,\textcircled{1}\,,\,\boxed{1}\,,\,\textcircled{2},\,\boxed{2}\,,\,\cdots\,,\,\boxed{N-1}\,,\,\textcircled{N},\,\boxed{N}$$

This was studied by Knuth in §6.2.1 of [Knu98], where he stated

> "... *any* algorithm for searching an ordered table of length N by means of comparisons can be represented as an N-node binary tree in which the nodes are labeled with the numbers 1 to N (unless the algorithm makes redundant comparisons)."

Knuth goes on to present (what we might call) hybrid trees, with internal nodes representing comparisons in circles, and external nodes representing conclusions in boxes, as we have defined above. Knuth then asserts,

> "Conversely, any binary tree corresponds to a valid method for searching an ordered table; we simply label the nodes
>
> $$\boxed{0}\;\textcircled{1}\;\boxed{1}\;\textcircled{2}\;\boxed{2}\;\cdots\;\boxed{N-1}\;\textcircled{N}\;\boxed{N}$$
>
> in symmetric order, from left to right."[2]

A summary of what Knuth is telling us to do is the following: Create an arbitrary binary tree with N nodes, label the nodes inorder with circled numbers, then add leaf nodes until each of the original nodes has two children, finally number these leaf nodes inorder with boxed numbers. The result will be a full binary tree that is labeled as described above by Knuth.

There is still a bit more work to do to prove that following the sequence of experiments dictated by the labels on the inner nodes does in fact lead to the leaf labeled with the solution.

Lemma 2.5. *The inorder successor of an inner node in a full binary tree is the first element of the inorder traversal of the right child subtree. Likewise, the predecessor of an inner node in a full binary tree is the last element of the inorder traversal of the left child subtree.*

Proof. Since the tree is full, an inner node has both left and right child sub-trees, which are non-empty by definition. The result then follows by the definition of inorder. □

Corollary 2.6. *The inorder successor of a leaf node in a full binary tree is either nil or an ancestor, whose left child subtree contains the leaf. Likewise, the inorder predecessor of a leaf node in a full binary tree is either nil or an ancestor, whose right child subtree contains the leaf.*

[2]Symmetric order is also called inorder.

Proof. Follows from Lemma 2.5. □

Theorem 2.7. *Any non-redundant algorithm that solves a normal problem over N experiments can be represented by a full binary search tree as above.*

Proof. Utilization of the binary decision tree will always terminate at a leaf, labeled as a $\boxed{x}$ for some x. It remains to be shown that that x satisfies the conditions of Definition 2.2.

If $x > 0$, then it has a inorder predecessor of $\textcircled{x}$, which by Corollary 2.6 is an ancestor whose right child subtree contains $\boxed{x}$. This means that in execution of the binary decision tree, an experiment was performed at x and succeeded.

If $x < N$, then it has a inorder successor of $\textcircled{x+1}$, which by Corollary 2.6 is an ancestor whose left child subtree contains $\boxed{x}$. This means that in execution of the binary decision tree, an experiment was performed at $x + 1$ and failed. □

2.1 Normality and Non-Redundancy

We would like to apply the theory developed above to egg-drop problems. Clearly, we have the next result.

Proposition 2.8. *All three egg-drop problem variations, Replacement, Bonus, and Standard, are normal. In addition, any variation that loses a constant number of eggs on failure and gains a constant number of eggs on success is also normal.*

If we want to minimize the number of consecutive experiments required, then it *seems* evident that we should restrict our attention to non-redundant algorithms. Proving that that restriction can be made requires carefully analyzing an arbitrary solution algorithm. For that analysis, we will assume the algorithm is represented by a tree, with the experiments to be preformed in (metaphorical) circles and the solutions in (metaphorical) squares. Failure of an experiment branches to the left in the tree and success branches to the right. The tree is to represent what the algorithm actually does, so it is required that for every node there is at least one solution that would cause the algorithm to reach that node: Any inaccessible nodes and their descendants are to be removed.

Definition 2.9. *Consider a normal problem and a binary tree representing a solution algorithm for the problem. For every inner node in the tree we associate with it the closed interval $[y, z]$ where y and z are the smallest and largest labels of leaves that are accessible descendants of the node. We will call that interval the* ***solution range*** *of the node.*

Observe the following facts:

(1) If the node is the root node, then the solution range is necessarily $[0, N]$, where N is the largest experiment.

(2) If the solution range of a node is $[y, z]$, then y is either 0 or the largest number that is known to succeed based on the outcomes of previous experiments.

(3) If the solution range of a node is $[y, z]$, then z is either N or one less than the smallest number known to fail based on the outcomes of previous experiments.

Proposition 2.10. *Any algorithm for solving any of the egg-drop problem variations can be modified so as to be non-redundant and to never require more experiments than the original algorithm.*

Proof. We will show that any algorithm that requires an experiment for which the outcome is guaranteed can be modified so that that experiment is either omitted altogether or is replaced by another experiment that does not have a guaranteed outcome. In any case, no more experiments will be required than were required by the original algorithm. Further modifications can be made until all experiments with guaranteed outcomes are eliminated. This process is explained in detail below:

In the tree representation, an experiment for which the outcome is guaranteed will correspond to an inner node with only one child—recall inaccessible nodes have all been removed. We consider such an inner node of least depth, say d. After all nodes at depth d have been processed, we progress to nodes of greater depth. When modifications are made to the algorithm, new inaccessible nodes might be created, so when dealing with a particular node, removing inaccessible descendants is the first thing to do.

For a node with a guaranteed outcome, let x be the floor from which the egg is to be dropped and let $[y, z]$ be the solution range associated with that node. Since the situations are different, we will need to consider separately the case when the guaranteed outcome is failure and when the guaranteed outcome is success.

If $y = z$, then the solution is already known to be y. Since y is known to be the solution, the node we are considering should instead be a leaf with y inside the square to indicate that y is the solution. The entire left or right child subtree below the node should be eliminated. The node is now a leaf, and the height of the tree has not been increased.

Assume now that $y < z$, so the solution s could be any integer in $[y, z]$, and the algorithm cannot yet terminate.

Guaranteed failure. Since dropping on x will give no new information and will lose an egg, this experiment could simply be omitted. Instead, let us note that for failure to be guaranteed, it must hold that $x > z$. By dropping the egg on z instead of x, we might have a success and that would tells us that the solution is z. If the egg breaks, then we are no worse off than before. We have gained the information that the egg breaks on z, but we can ignore the additional information for now and follow the original algorithm. For the tree, the label of the node should be changed to z, a right-child leaf labeled z should be added, and the left-child subtree is unchanged. The node now has two children, and the height of the tree has not been increased.

Guaranteed success. Because success is guaranteed, it must hold that $x \leqslant y$. The drop on x is guaranteed to result in the egg surviving, so it may be beneficial in that replacement or bonus eggs will be acquired. But since no new information is gained, at least one additional experiment beyond the drop on x will be required.

Consider what happens if the drop is made from $y+1$ instead of from x. If the egg breaks, then the solution is known to be y. No additional experiments are required and the modified algorithm can terminate. If the egg survives, then the benefit of acquiring replacement or bonus eggs is achieved as would have happened with a drop on x. We have gained the additional information that the egg survives when dropped on $y+1$, but we may ignore that information for now and simply follow the original algorithm. For the tree, the label of the node should be changed to $y+1$, a left-child leaf labeled y should be added, and the right-child subtree is unchanged. The node now has two children, and the height of the tree has not been increased.

The information that the egg broke when dropped on z or that the egg survived when dropped on $y+1$ was temporarily ignored when we chose to leave the subtree unchanged. Nonetheless the information will affect the solution range of descendant nodes. We might well find newly inaccessible nodes and new instances of guaranteed outcome experiments in the subtree. Such issues occur at depth greater than d and are to be dealt with after all depth d nodes have been processed. When all nodes at depth d have been processed, they are all either leaves or inner nodes with two children. Then when all nodes in the tree have been processed the tree will be a full binary tree with height no greater than the original tree. □

This process of modifying the tree described in the preceding proof is also codified in the following algorithm.

Algorithm 2.11. *Non-Redundancy. We assume branch left on failure, $N > 0$ and $N+1$ integral solutions in $[0, N]$. For each node, let $\mathcal{N}$ be its solution range $[y, z]$ and label x. For nodes other than the root, let $\mathcal{P}$ be the parent of $\mathcal{N}$, so that* ***reference***$(\mathcal{P}) = \mathcal{N}$. *Note that* ***reference*** *resolves to either* ***left*** *or* ***right****. Let $\mathcal{L}(x)$ be the leaf containing x.*

1: ***for*** *each Node $\mathcal{N}$, Breadth first,* ***do***
2: *Unlink all Nodes from $\mathcal{N}$, and its children, that are impossible to visit.*
3: ***if*** *$\mathcal{N}$ is not a leaf* ***then***
4: ***if*** *$y = z$* ***then***
5: *unlink $\mathcal{L}(z)$ and set* ***reference***$(\mathcal{P}) = \mathcal{L}(z)$.
6: ***else if*** *$x > z$* ***then***
7: *set* ***label***$(\mathcal{N}) = z$, *unlink $\mathcal{L}(z)$, set* ***right***$(\mathcal{N}) = \mathcal{L}(z)$.
8: ***else if*** *$x \leqslant y$* ***then***
9: *set* ***label***$(\mathcal{N}) = y + 1$, *unlink $\mathcal{L}(y)$, set* ***left***$(\mathcal{N}) = \mathcal{L}(y)$.
10: ***end if***
11: ***end if***
12: ***end for***

Proof. Line 2 implements the removal of inaccessible descendants. The condition in line 4 is true when the node should be a leaf, and line 5 converts the node to an appropriately labeled leaf. The condition in line 6 is true when the outcome is guaranteed to be failure, and line 7 changes the experiment to z and adds the new leaf that reports the solution is z if the drop on z is a success. The condition in line 8 is true when the outcome

guaranteed to be success, and line 9 changes the experiment to $y+1$ and adds the new leaf that reports the solution is y if the drop on $y+1$ is a failure.

When all nodes are processed, and they will be, then each will either be converted to a leaf or will have two children. □

3 Egg Drop Numbers

Finding solutions for the egg-drop problems with the minimum number of drops turns out to be straightforward. Proposition 2.10 and Theorem 2.7 tells us that the algorithm we seek can be represented by a full binary search tree. Keeping track of eggs in hand, we recursively add nodes to the tree breadth first until there are sufficient inner nodes, representing floors, while not violating the constraints of the problem. For instance, for Standard Eggs starting with k eggs, this means no path from the root can branch left more than k times. This process is carried out in Figure 3 starting with 2 eggs with the number of eggs remaining shown at each node. We then determine the maximum number of inner nodes for depth d, call it $H_{\mathcal{P},k}(d)$, where the subscript $\mathcal{P}$ indicates the problem we are trying to solve, and the subscript k indicates starting with k eggs; we use $\mathcal{S}$ for Standard eggs. Then $H_{\mathcal{S},k}(d)$ is the highest floor that can be distinguished starting with k eggs; Boardman calls these the "egg-drop numbers" in [Boa04]. Counting the inner nodes in Figure 3, we see that $H_{\mathcal{S},2}(4) = 10$. If it is impossible for eggs to be exhausted after d drops, then the maximum is attained and $H_{*,*}(d) = 2^{d-1}$.[3]

We note that the type of problem constrains the possible topology of the tree, i.e., since one can not drop an egg once one has exhausted the supply of eggs, nodes with no eggs are necessarily leaves.

(1) The left child always has one fewer egg than the parent.

(2) The right child has:

- **(a)** Standard Eggs: the same number of eggs as the parent, as in Figure 3;
- **(b)** Replacement Eggs: the same number of eggs as the root node, as in Figure 4;
- **(c)** Bonus Eggs: one more egg than the parent, as in Figure 5.

Applying the numbering scheme $\{0, 1, 1, \cdots x, x \cdots 10, 10\}$ to Figure 3 with an inorder traversal and using circles for inner nodes, squares for leaves, gives us the annotated binary decision tree Figure 1. We also note that by counting the inner nodes in Figures 4 and 5, we learn that $H_{\mathcal{R},2}(4) = 11$ and $H_{\mathcal{B},1}(4) = 7$, the subscripts $\mathcal{R}$ and $\mathcal{B}$ indicating Replacement Eggs and Bonus Eggs, respectively.

To handle a taller building, we will need to grow a bigger tree. Specifically, to determine how many egg drops are required for a building with N-stories, we must construct a tree that has enough depth that the number of inner nodes is N and the number of leaves is $N+1$.

[3] In terms of egg-drop problems, this fact is equivalent to the unsurprising statement that if you have d eggs available, then you can determine their strength for a building up to $2^d - 1$ stories tall.

This procedure will quickly outgrow the use of paper and pencil, and will require a computer. Trees are a standard data structure and inorder transversal is a standard algorithm. We might say the problem is solved. But as mathematicians we would like to do more than simply turn the problem over to a computer. In particular, we would like to answer the original question of how many egg-drop experiments will be required or to at least estimate that number.

3.1 Standard Eggs.

Although complex and clever derivations have been made in the literature for this result, we present a combinatorial argument, which is both concise and elucidating.

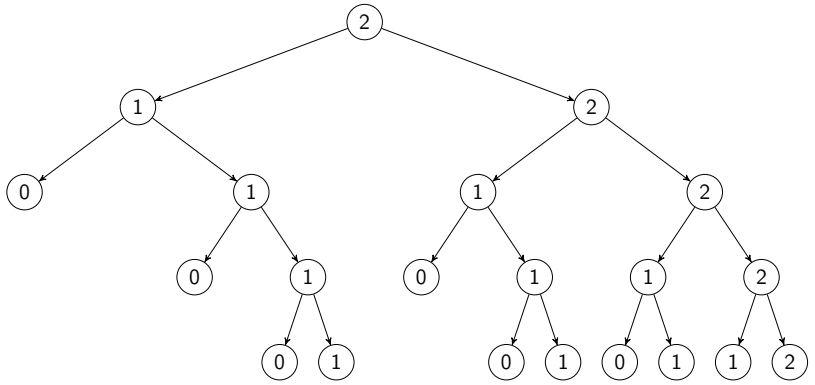

Figure 3: Counting the number of eggs in a Standard Egg tree with 2 initial eggs.

Theorem 3.1. *With Standard Eggs, the height of the tallest building for which the strength of the eggs can be determined starting with k eggs and using no more than d egg drops[4] is*

$$H_{\mathcal{S},\,k}(d) = \sum_{i=1}^{k} \binom{d}{i}$$

Proof. The number of leaves in the tree representing the algorithm is identical to the number of paths from the root to the leaves with i breaks, i running from 0 up to k. Each such path uniquely corresponds to a binary word with a 0 representing going left and a 1 representing going right. If the height of the tree is d, we would like to consider binary words of length d. But if there are k zeros in the word, that means all the eggs were broken, the path ends, and so does the word. Nonetheless, if we put additional 1's after the last of the k zeros, we can bring the length of the word up to d while maintain the one-to-one correspondence between binary words and paths in the tree.

[4]This result is consistent with the egg drop number $\left\langle {d \atop k} \right\rangle$ defined by Boardman, and a combinatorial argument based on words appears in the section "Direct counting approach" of [Boa04].

For a particular i, the number of binary words of length d that contain exactly i 0's is $\binom{d}{i}$. Adding this over all possible i from 0 to k, we get

$$\sum_{i=0}^{k} \binom{d}{i}$$

The number of inner nodes is 1 fewer than the number of leaves, so

$$H_{\mathcal{S},\,k}(d) = \sum_{i=1}^{k} \binom{d}{i}$$

□

3.2 Replacement Eggs.

To determine the egg-drop numbers for Replacement Eggs, we need to make a small digression into the topic of k-bonacci numbers.

Recall that the k-bonacci numbers $F_\ell^{(k)}$, $(k \geqslant 1)$ are a generalization of the Fibonacci numbers defined by the initial values

$$F_\ell^{(k)} = \begin{cases} 0, & 0 \leqslant \ell < k-1 \\ 1, & \ell = k-1 \end{cases} \tag{1}$$

and the recursion

$$F_\ell^{(k)} = \sum_{i=1}^{k} F_{\ell-i}^{(k)}, \quad \text{for} \quad k \leqslant \ell \tag{2}$$

Following the idea used in the proof of Theorem 3.1, we see that we should examine the number of binary words of a given length n that **do not** contain k consecutive 0's.

Lemma 3.2. *There are $F_{n+k}^{(k)}$ binary words of length $n \geqslant 0$ lacking k consecutive 0's.*

Proof. Let n be the length of a binary word. For $n < k$, all words qualify, and recall that $F_{n+k}^{(k)} = 2^k$ for $0 \leqslant n < k$. For $n \geqslant k$, every qualifying word must have a trailing 1 followed by $i < k$ zeros. This word can be formed by a qualifying word of length $n-i-1$ followed by the 1 and i zeros. The total number of ways to do this is

$$\sum_{i=0}^{k-1} F_{n-i-1+k}^{(k)} = \sum_{i=1}^{k} F_{n+k-i}^{(k)} = F_{n+k}^{(k)}$$

□

Remark 3.3. *Lemma 3.2 is also implied by the definition of $p_n(k)$, and equations* (1), (10), *and* (12) *in [Sha73], using the version of the k-bonacci numbers (there n-bonacci) shifted so that $F_{n,0} = 1$ and $F_{n,-r} = 0$. This was built upon and clarified by Lemma 2.2 in [PM82], as $A_n^{(k)} = f_{n+1}^{(k)}$, this time using $f^{(k)}$ for the k-bonacci numbers with the same shift, which translates to the result with a small amount of algebraic manipulation.*

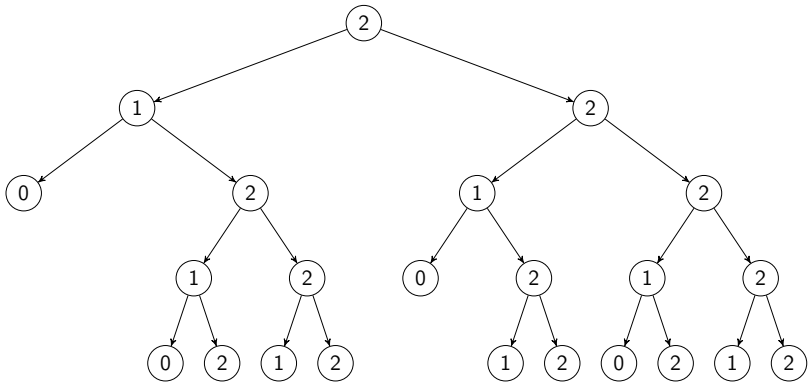

Figure 4: Replacement Eggs.

Theorem 3.4. *With Replacement Eggs, the height of the tallest building for which the strength of the eggs can be determined starting with k eggs and using no more than d egg drops is*

$$H_{\mathcal{R},\,k}(d) = -1 + \sum_{i=0}^{\lfloor d/(k+1)\rfloor} (-1)^i \binom{d-ik}{i} 2^{d-i(k+1)}$$

Proof. For replacement eggs, we can only exhaust our eggs if we get k consecutive breaks; any other sequence will lead to eggs being replenished. We can look at every path from the root as a binary word, using 0 for a break and 1 for a success.

We will count the number of binary words corresponding to a tree of height d. For any of our words that are shorter than d bits, we would like to add 0's to bring them up to length d. Such words shorter than d must end in k consecutive 0's, so if they are distinct before adding 0's, they remain distinct after 0's are added.

One word consists only of 0's. Every other word will contain at least one 1. Since there is a 1 in the word, there is a last 1 in the word and that 1, if followed by anything, is followed by 0's

To count the number of words that contain a 1, remove the last 1 in the word and remove all the 0's that follow that last 1. What remains is a binary word lacking k consecutive 0's having length somewhere between 0 and $d-1$. Thus, the number of leaves in the tree, $L(d)$, is 1 for the single path without any 1's plus the sum over i going from 0 to $d-1$ of the number of binary words lacking k consecutive 0's having length i. By Lemma 3.2, the number of leaves is

$$L(d) = 1 + \sum_{i=0}^{d-1} F_{i+k}^{(k)} = 1 + \sum_{i=k}^{d+k-1} F_i^{(k)} = \sum_{i=0}^{d+k-1} F_i^{(k)}$$

where we have used the k-bonacci initial conditions (1).

In fact, the partial sums of the sequence k-bonacci numbers can be expressed in terms

of a sum of products of binomial coefficients and powers of 2 as follows:

$$\sum_{i=0}^{d+k-1} F_i^{(k)} = \sum_{i=0}^{\lfloor d/(k+1)\rfloor} (-1)^i \binom{d-ik}{i} 2^{d-i(k+1)} \tag{3}$$

Equation 3 is not well-known. While it follows easily from work of Otto Dunkel published in 1925,[5] it was recently rediscovered; a combinatorial proof is given in [PW23].

The height of the tallest building, $H_{\mathcal{R},k}(d)$, is 1 fewer than the number of leaves, completing the proof. □

3.3 Bonus Eggs.

For Standard Eggs or Replacement Eggs, starting with only 1 egg is not interesting, but for Bonus Eggs starting with 1 egg provides some useful information. The tree in Figure 5 shows the number of eggs remaining as we progress through the first four egg drops. If you extend the tree in Figure 5 down to depth 9, you will see that the number of leaves with 0 eggs that occur at depth 1, 3, 5, 7, and 9, respectively, is 1, 1, 2, 5, and 14. The first 5 Catalan numbers are 1, 1, 2, 5, and 14. The Catalan numbers appear because the nth Catalan number is the cardinality of the set of sequences of n $+1$'s and n -1's with non-negative partial sums. To run out of eggs at exactly depth $2n-1$ one must have broken n eggs—those are the n -1's—and have received $n-1$ bonus eggs—those are $n-1$ of the $+1$'s, the nth $+1$ is the starting egg. D. F. Bailey in [Bai96] generalized the Catalan number construction by considering sequences of n $+1$'s and m -1's with non-negative partial sums. We further generalize Bailey's work[6] and apply the results in the present setting of Bonus Eggs.

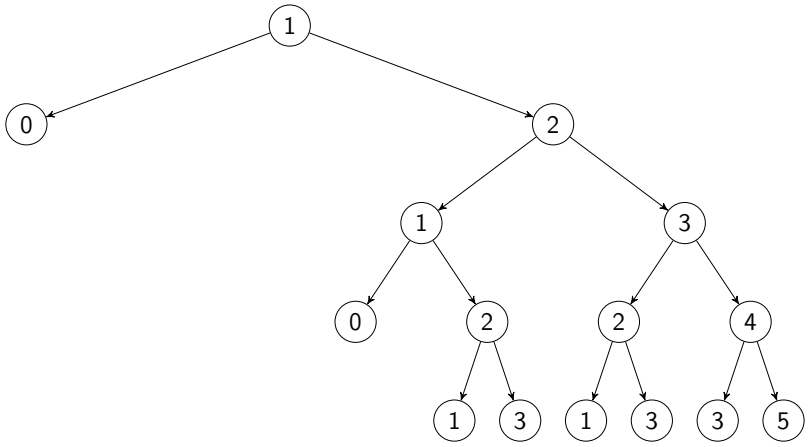

Figure 5: Bonus Eggs.

[5]To obtain (3) from Dunkel's work, compare the equation for $P_2(n)$ in section 6 of [Dun25] to the equation for $P_2(n)$ in section 10 of the same paper.

[6]Relative to Bailey's paper, we will reverse the roles of m and n.

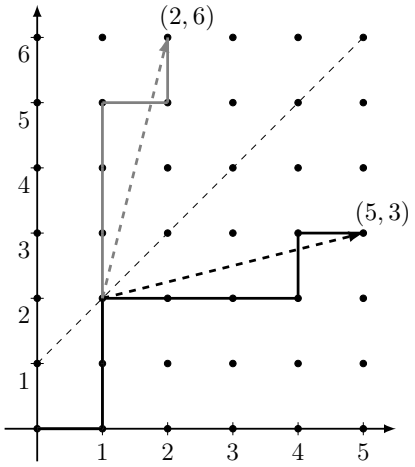

Figure 6: Reflection of the solid path over $y = x + 1$.

Definition 3.5. *Let k, m and n be non-negative integers with $n \leqslant m + k$. Let $G_k(m, n)$ denote the number of sequences $a_1, a_2, \ldots, a_{m+n}$ of m $+1$'s and n -1's for which every partial sum is greater than $-k$, that is,*

$$\text{if } \ 1 \leqslant i \leqslant m + n, \ \text{ then } \ a_1 + a_2 + \cdots + a_i > -k \tag{4}$$

Note that because the hypothesis of (4) is false when $m = n = 0$, the statement itself is satisfied by the empty sequence. Thus $G_k(0, 0) = 1$ holds for all non-negative k.

Theorem 3.6. *For k, m and n non-negative integers with $n \leqslant m + k$, it holds that*

$$G_k(m,n) = \begin{cases} \binom{m+n}{n} & \text{, if } n < k \\ \binom{m+n}{n} - \binom{m+n}{n-k} & \text{, otherwise} \end{cases} \tag{5}$$

Remark 3.7. *Notice that the theorem tells us that $G_1(m, m)$ equals the mth Catalan number, so it should not be surprising that our proof of the theorem is similar to a classic construction used in studying Catalan numbers.*

Proof. We will translate each partial sum of plus and minus 1's into a path in the cartesian plane that begins at $(0, 0)$ and, in some order, takes m steps to the right and n steps up ending at (m, n). We want to count the number of such paths that do not touch the line $y = x + k$.

Assuming $n < k$.
The total number of paths that take m steps to the right and n steps up is $\binom{n+m}{n}$. If $n < k$, then no such path can reach the line $y = x + k$.

Assuming $k \leqslant n$.

As part of the definition we have assumed that $n \leqslant m + k$. If $n = m + k$, then any path that take m steps to the right and n steps up ends on the line $y = x + k$. Thus $G_k(m, m + k) = 0$ and that agrees with the right-hand side of (5). So from now on we assume that $n < m + k$.

The total number of paths that take m steps to the right and n steps up is $\binom{n+m}{n}$. Some of those paths may be "bad" paths that touch the line $y = x + k$. So we need to count the bad paths, and to do that we will show that the bad paths from $(0, 0)$ to (m, n) can be put into one-to-one correspondence with the paths from $(0, 0)$ to $(n - k, m + k)$.

Any bad path has a first point $(x, x + k)$ where it contacts the line $y = x + k$. The vector from $(x, x + k)$ to (m, n) is $(m - x, n - x - k)$. If in that vector we swap steps to the right for steps up and swap steps up for steps to the right, we obtain the vector $(n-x-k, m-x)$. Proceeding from $(x, x+k)$ via the new vector takes us to $(n-k, m+k)$ (see Figure 6), a point above the line $y = x + k$ because $n < m + k$. On this new path from $(0, 0)$ to $(n - k, m + k)$, the point $(x, x + k)$ is the first point where the path hits the line.

Every path of $n-k$ steps to the right and $m+k$ steps up will cross the line $y = x+k$ somewhere, so there will be a first point of contact, say $(x, x + k)$. The vector from $(x, x+k)$ to $(n-k, m+k)$ is $(n-k-x, m-x)$ Again swapping right steps and steps up, we obtain the vector $(m - x, n - k - x)$. Proceeding from $(x, x + k)$ via the new vector takes us to (m, n). On this new path from $(0, 0)$ to (m, n), the point $(x, x+k)$ is the first point where the path hits the line.

From the above constructions, we see that the number of bad paths $(0, 0)$ to (m, n) equals the number of paths with $n - k$ steps to the right and $m + k$ steps up, that is, $\binom{m+n}{n-k}$. It follows that

$$G_k(m, n) = \binom{m + n}{n} - \binom{m + n}{n - k}$$

□

Theorem 3.8. *With Bonus Eggs starting with k eggs, the tree of depth $d < k$ has 2^d leaves and $2^d - 1$ inner nodes. For the tree of depth $k \leqslant d$, set*

$$\alpha = \left\lfloor \frac{d - k}{2} \right\rfloor \quad \text{and} \quad \beta = \left\lfloor \frac{d - k + 1}{2} \right\rfloor$$

then the tree has

$$Z_k(d) = \sum_{m=0}^{\alpha} \frac{k}{2m + k} \binom{2m + k}{m}$$

leaves with no remaining eggs and

$$M_k(d) = \sum_{n=\beta}^{\beta+k-1} \binom{d}{n}$$

leaves that still have eggs.

The height of the tallest building for which the strength of the eggs can be determined starting with k eggs and using no more than d egg drops is $2^d - 1$ when $d < k$ and when $k \leqslant d$ is

$$H_{\mathcal{B},\,k}(d) = \sum_{m=1}^{\alpha} \frac{k}{2m+k} \binom{2m+k}{m} + \sum_{n=\beta}^{\beta+k-1} \binom{d}{n}$$

Proof. The result is clear for $d < k$, so we will assume $k \leqslant d$.

Paths in the tree that end with no eggs left. The first depth at which all k eggs can be broken is clearly k. Also notice that the number of eggs remaining unbroken at depth i always has the same parity as $i + k$. So we need to consider paths that end with no eggs left at depth $k + 2m$, where m ranges from 0 to $\lfloor (d-k)/2 \rfloor$.

Each failure results in -1 egg and each success results in $+1$ egg. The supply of eggs is exhausted when there have been k more failures than successes. Suppose the supply of eggs was exhausted exactly on trial $k + 2m$, where $k + 2m \leqslant d$. Immediately before trial $k + 2m$ we know there must have been exactly one egg left. Say on those previous $k + 2m - 1$ trials there had been a successes and b failures. To have one egg left, we must have $k + a - b = 1$. To account for $k + 2m - 1$ trials, we must have $a + b = k + 2m - 1$. Solving those equations for a and b, we find that $a = m$ and $b = m + k - 1$.

Because we ran out of eggs on trial $k + 2m$ and not earlier, the a successes and b failures can be thought of as a sequence of a $+1$'s and b -1's such that the partial sums are all greater than $-k$. The number of such sequences is $G_k(a, b) = G_k(m, m + k - 1)$.

Thus the number of paths of depth not exceeding d that end with no eggs remaining equals

$$\sum_{m=0}^{\alpha} G_k(m, m + k - 1)$$

By (5) we see that for $1 \leqslant m$

$$\begin{aligned} G_k(m, m+k-1) &= \binom{2m+k-1}{m+k-1} - \binom{2m+k-1}{m-1} \\ &= \frac{(2m+k-1)!}{m!\,(m+k-1)!} - \frac{(2m+k-1)!}{(m-1)!\,(m+k)!} \\ &= \frac{(2m+k-1)!}{m!\,(m+k)!} \Big[(m+k) - m\Big] \\ &= \frac{k}{2m+k} \binom{2m+k}{m} \end{aligned}$$

Also note that the equation $G_k(m, m+k-1) = \frac{k}{2m+k} \binom{2m+k}{m}$ remains valid when $m = 0$.

Paths in the tree that reach depth d with eggs still remaining. For this part of the proof, let n be the number of failures that occur in the d experiments. The number of successes is $d - n$, and because there are still eggs remaining unbroken at depth d, we

have $1 \leqslant k + (d-n) - n$. Thus the values of n range from 0 up to $\lfloor (d+k-1)/2 \rfloor$. Let n^* denote this last value.

A path in the tree that reaches depth d with eggs remaining corresponds to sequence of $d-n$ $+1$'s and n -1's such that the partial sums are all greater than $-k$. The number of such sequences is $G_k(d-n, n)$.

We see that the number of nodes at depth d where there are still remaining eggs is

$$\begin{aligned} M_k(d) &= \sum_{n=0}^{n^*} G_k(d-n,n) = \sum_{n=0}^{k-1} \binom{d}{n} + \sum_{n=k}^{n^*} \left[\binom{d}{n} - \binom{d}{n-k} \right] \\ &= \sum_{n=0}^{n^*} \binom{d}{n} - \sum_{n=k}^{n^*} \binom{d}{n-k} = \sum_{n=0}^{n^*} \binom{d}{n} - \sum_{n=0}^{n^*-k} \binom{d}{n} \\ &= \sum_{n=n^*-k+1}^{n^*} \binom{d}{n} \end{aligned}$$

It is then immediate that $\lfloor (d+k-1)/2 \rfloor - k + 1 = \lfloor (d-k+1)/2 \rfloor = \beta$ □

References

[AIP] Gabriel Alves, James Innes, and Geoff Pilling. Egg dropping. *Brilliant Math & Science Wiki*. URL: `https://brilliant.org/wiki/egg-dropping`.

[Bai96] D. F. Bailey. Counting arrangements of 1's and -1's. *Mathematics Magazine*, 69(2):128–131, 1996. `arXiv:https://doi.org/10.1080/0025570X.1996.11996408`, `doi:10.1080/0025570X.1996.11996408`.

[Boa04] Michael Boardman. The egg-drop numbers. *Mathematics Magazine*, 77(5):368–372, December 2004.

[Dun25] Otto Dunkel. Solutions of a probability difference equation. *The American Mathematical Monthly*, 32(7):354–370, 1925.

[Knu98] Donald E. Knuth. *The Art of Computer Programming: Volume 3: Sorting and Searching*. Pearson Education, 1998. URL: `https://books.google.com/books?id=cYULBAAAQBAJ`.

[KVW96] J.D.E. Konhauser, D. Velleman, and S. Wagon. *Which Way Did the Bicycle Go?: And Other Intriguing Mathematical Mysteries*. Dolciani Mathematical Expositions. Mathematical Association of America, 1996.

[PM82] A. N. Philippou and A. A. Muwafi. Waiting for the K th consecutive success and the Fibonacci sequence of order K. *Fibonacci Quarterly*, 20(1):28–32, February 1982. URL: `http://www.fq.math.ca/Scanned/20-1/philippou.pdf`.

[PW23] Harold R. Parks and Dean C. Wills. Sums of k-bonacci numbers. *The Fibonacci Quarterly*, 61(2):129–134, 2023.

[Sha73] Harold D. Shane. A Fibonacci probability function. *Fibonacci Quarterly*, 11(5):517–522, December 1973. URL: `http://www.fq.math.ca/Scanned/11-5/shane.pdf`.

[Ski09] S.S. Skiena. *The Algorithm Design Manual*. Springer London, 2009. URL: `https://books.google.com/books?id=7XUSn0IKQEgC`.

[Sni03] Moshe Sniedovich. OR/MS Games: 4. the joy of egg-dropping in Braunschweig and Hong Kong. *INFORMS Transactions on Education*, 4(1):48–64, 2003.

[Wil09] Dean Connable Wills. *Connections between Combinatorics of Permutations and Algorithms and Geometry*. PhD thesis, Oregon State University, Corvallis, OR, USA, 2009. AAI3376756.