Kaushik Tota

Avirath Sundaresan

CS 121 Final Project

Designing a Pokemon hack-checking algorithm with imperfect information

Introduction

The function `is_pkmn_hacked()` aims to determine whether a Pokemon's stat values are "legitimate" - that is, whether the stat values inputted by a user for a particular Pokemon are possible per the in-game mechanics. A Pokemon possesses six stat value: HP, Attack, Special Attack, Defense, Special Defense, and Speed. A Pokemon also possesses a level (ranging from 1 to 100), which causes its stat values to increase across all stat categories as the Pokemon levels up. Finally, a Pokemon possesses a nature, an attributes of an individual Pokemon which simultaneously apply a "buffing" (1.1) multiplier to one stat and a "debuffing" (0.9) multiplier to a second stat. These are the "expressed" attributes of a Pokemon; that is, these attributes of a Pokemon are directly visible to a user. However, there are two additional factors which *internally* impact the observed stat value of a Pokemon: EVs and IVs. EVs, or Effort Values, are a pool of "additional" stat points which can be distributed across a Pokemon's stat categories. A Pokemon can have a maximum of 508 EVs which can be distributed amongst all six stats. A Pokemon can only invest a maximum of 252 EVs in a given stat. Meanwhile, IVs, short for Individual Values, are a value which can range from 0 to 31 for each stat which are randomly assigned to a Pokemon when caught and *cannot* be selectively determined for a particular stat (unlike EVs, which can be distributed intentionally by a user through specific training).

When a user seeks to store a Pokemon in our implementation of the Pokemon Storage System, they input the species name of the Pokemon, the stat values of their Pokemon, their Pokemon's level, and their Pokemon's nature. This is analogous to the information involved in the process of storing a Pokemon in the main-series Pokemon games. Note that EVs and IVs are *not* inputted by a user - this also reflects the in-game process of storing Pokemon in a box (as these values are not directly presented to a user, and a user does not directly interface with them as numerical values they can view). Below is an example of a Pokemon as viewed from the Pokemon Storage System in the main-series Pokemon video games:

Here, we observe that the Pokemon's level, stat values, and nature are all visible; however, its EVs and IVs are *not* visible. In actuality, the stat values displayed on this screen are *derived* values based on the Pokemon's EVs, IVs, nature, and a set of special values called "base stats." Base stats constitute the "archetypal" stat

distribution for a Pokemon (or the "potential" for the stats of a particular Pokemon species). For example, below are the base stats of Alakazam (the Pokemon shown above) from which the stats of the specific Level 40 Alakazam shown above are derived:

Stat		Range	
		At Lv. 50	At Lv. 100
HP:	55	115 - 162	220 - 314
Attack:	50	49 - 112	94 - 218
Defense:	45	45 - 106	85 - 207
Sp. Atk:	135	126 - 205	247 - 405
Sp. Def:	95	90 - 161	175 - 317
Speed:	120	112 - 189	220 - 372
Total:	500	Other Pokémon with this total	

- Minimum stats are calculated with 0 EVs, IVs of 0, and (if applicable) a hindering nature.
- Maximum stats are calculated with 252 EVs, IVs of 31, and (if applicable) a helpful nature.

Here, we observe that, while not a perfect match, the observed stats of the specific Alakazam generally follow the base stat distribution. The "discrepancies" between the base stats and the true stat values of the Alakazam are governed by two equations which actually compute a Pokemon's stats from its base stats, EVs, IVs, and nature:

$$S_{HP} = \lfloor \frac{(2 \cdot B + I + \lfloor \frac{E}{4} \rfloor) \cdot L}{100} \rfloor + L + 10$$

$$S_O = \lfloor (\lfloor \frac{(2 \cdot B + I + \lfloor \frac{E}{4} \rfloor) \cdot L}{100} \rfloor + 5) \cdot N \rfloor$$

where S is the observed stat value, B is the base stat value, I is the IV value (0-31), E is the EV value (0-252), L is the level (1-100), and N is the multiplier conferred on that stat by the nature (0.9, 1, or 1.1). Note that nature cannot affect a Pokemon's HP stat, meaning that there are two different stat-calculating equations: one for HP and one for every other non-HP stat.

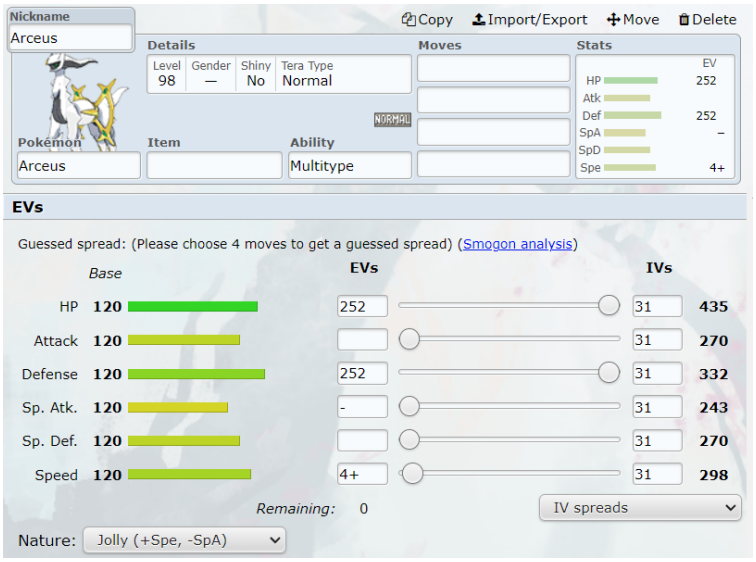
We can thus use these formulae to determine whether an inputted Pokemon's stat distribution is feasible, which in turn determines whether the Pokemon is "hacked" or not. A hacked Pokemon will contain at least one stat which is impossible for that Pokemon to possess (constrained by its level, nature, IVs, EVs, and base stats).

Motivation

Since a user in the main-series Pokemon games does not truly have to interact with EVs and IVs, our Pokemon Storage System does not store EVs or IVs as numerical values. This is a scenario with *imperfect* information, in that we lack the values of all the variables which we must use in order to compute the Pokemon's stat values. Thus, we are left in a position where we must *infer* a Pokemon's EVs and IVs from the stat values inputted by the user. More specifically, since EVs are constrained by a total value (508 distributed across all 6 stats), we particularly care about inferring the distribution of EVs across an inputted Pokemon's stats.

Building intuition and constraining the problem

To constrain our problem space slightly and to clarify why the EVs are the most relevant value for us to infer in the process of detecting whether a Pokemon is hacked, let us consider an example. Below is a Level 98 Arceus a user would like to store in our Pokemon Storage System which we will use as an example for the remainder of this writeup:



The base stats of Arceus are 120 across all stats (shown on the LHS of the bottom panel). The EV allocations are shown in the EV column, the IVs are shown in the IV column, and the resulting stat values are shown on the RHS of the bottom panel. Finally, the nature (Jolly) and its effects (1.1 multiplier on speed, 0.9 multiplier on special attack) are shown in the bottom-left corner.

Consider the HP stat. Using the formula for computing the HP stat value and the values in the example above, we have:

$$S_{HP} = \lfloor \frac{(2 \cdot 120 + 31 + \lfloor \frac{252}{4} \rfloor) \cdot 98}{100} \rfloor + 98 + 10 = 435$$

which is precisely the value shown on the RHS for the HP stat. Note that since we have allocated 31 IVs and 252 EVs to the HP stat, this is actually the *maximum possible value* that this particular Arceus can have as its HP stat. Now, suppose we *hacked* this Arceus to include an extra 4 HP EVs (beyond the maximum allowed amount). Now, our HP stat would be:

$$S_{HP} = \lfloor \frac{(2 \cdot 120 + 31 + \lfloor \frac{256}{4} \rfloor) \cdot 98}{100} \rfloor + 98 + 10 = 436$$

Since this value is larger than the maximum possible value of 435, we can conclude that this Arceus is hacked. However, suppose we reduced the Arceus's HP IV value to 30, and kept the EV value at 256. Now, the computed HP stat would be:

$$S_{HP} = \lfloor \frac{(2 \cdot 120 + 30 + \lfloor \frac{256}{4} \rfloor) \cdot 98}{100} \rfloor + 98 + 10 = 435$$

Now, the Arceus would have an HP stat of 435, which is the allowed maximum, despite having an illegally-high EV allocation. **This is okay.** We simply care about whether the *observed* stat values are feasible for a Pokemon. Even if the hidden values (IVs and EVs) generating an observed stat value are illegal, if the generated stat value is feasible, the "benefit" of hacking those stats is effectively negated (as it is possible to have a non-hacked Pokemon with exactly those stats). Thus, we constrain our problem space by not necessarily worrying about IV and EV values themselves being "legitimate" - instead, we simply care whether the observed stat values of a Pokemon could be generated by some legitimate IV and EV distribution.

This also implicitly demonstrates the importance of identifying EV allocations - we can always freely adjust the IV value of a particular stat (as long as its value lies between 0 and 31); however, our EVs are constrained by a total maximum of 508 allocated EVs.

Designing the algorithm: an initial condition

A simple initial condition to consider when determining whether a Pokemon is hacked is whether the stat even lies between the minimum and maximum possible value that the Pokemon could possess for that stat. The minimum possible value of a stat is given by the scenario where a Pokemon has 0 IVs *and* 0 EVs invested in that stat. Meanwhile, the maximum possible value of a stat is given by the scenario where a Pokemon has 31 IVs *and* 252 EVs invested in that stat. Since we track the nature of every Pokemon, we do not need to worry about the nature "multiplier" (that is, we do not need to consider a multiplier of 1.1 or 0.9 in the maximum or minimum stat, since we will know for every Pokemon what the nature multiplier is for that particular stat). Then, we simply compare the inputted stat values against the minimum and maximum stat values implied by the aforementioned parameters. If the stat is smaller than the minimum or larger than the maximum, we terminate early and return that the Pokemon is hacked. If all stat values are checked and we pass through the entire function successfully, that must mean that all stats are within the minimum-maximum range, and we can return that the Pokemon is not hacked.

(pseudocode on next page)

Algorithm 1: A first pass at the hack-checking algorithm. Base stat variables are prepended with a "b" and stat multipliers conferred by the Pokemon's nature are prepended with an "m." Returns **true** if the Pokemon is hacked and **false** if the Pokemon is not hacked.

function *compute_hp*:
  B, L, E, I
return $\lfloor \frac{(2 \cdot B + I + \lfloor \frac{E}{4} \rfloor) \cdot L}{100} \rfloor + L + 10$
function *compute_other_stat*:
  B, L, E, I, N
return $\lfloor (\lfloor \frac{(2 \cdot B + I + \lfloor \frac{E}{4} \rfloor) \cdot L}{100} \rfloor + 5) \cdot N \rfloor$
function *is_pkmn_hacked*:
  hp, atk, spa, def, spd, spe, lvl, bhp, $batk$, $bspa$, $bdef$, $bspd$, $bspe$, $matk$, $mspa$, $mdef$, $mspd$, $mspe$
if $hp < compute\_hp(bhp, lvl, 0, 0)$ **or** $hp > compute\_hp(bhp, lvl, 252, 31)$ **then**
  return true
if $atk < compute\_other\_stat(batk, lvl, 0, 0, matk)$ **or** $atk > compute\_other\_stat(batk, lvl, 252, 31, matk)$ **then**
  return true
if $spa < compute\_other\_stat(bspa, lvl, 0, 0, mspa)$ **or** $spa > compute\_other\_stat(bspa, lvl, 252, 31, mspa)$ **then**
  return true
if $def < compute\_other\_stat(bdef, lvl, 0, 0, mdef)$ **or** $def > compute\_other\_stat(bdef, lvl, 252, 31, mdef)$ **then**
  return true
if $spd < compute\_other\_stat(bspd, lvl, 0, 0, mspd)$ **or** $spd > compute\_other\_stat(bspd, lvl, 252, 31, mspd)$ **then**
  return true
if $spe < compute\_other\_stat(bspe, lvl, 0, 0, mspe)$ **or** $spe > compute\_other\_stat(bspe, lvl, 252, 31, mspe)$ **then**
  return true
return false;

This seems like a good start; however, it contains a critical flaw (which will be discussed in the subsequent section).

Designing the algorithm: going one step further

While the algorithm described in the previous part seems good at first glance, it ignores the fact that there can only be a maximum of 508 EVs allocated across all the stats of the Pokemon. For example, consider a Level 98 Arceus with 31 IVs in every stat and 252 EVs allocated to every stat. In this scenario, the computed stat value for each of Arceus's stats would be the "maximum" bound in the conditions outlined in the above psuedocode. Then, the stat would "pass" the check and move on to the next stat. All stats would pass in this manner, and the function would finally return **false**. However, $252 \cdot 6 > 508$, meaning that this Pokemon is certainly hacked.

Thus, we must determine a way to "track" the total number of EVs which must have been allocated to a Pokemon. More specifically, we must infer the *lowest-possible* number of EVs which *must* have been allocated to a stat in order to achieve the observed stat value. Then, if the sum of the minimized EV allocations is greater than 508, we should return **true**, since this indicates that the Pokemon *must* have allocated greater than 508 EVs to achieve its stat values. This requires us to "reverse-engineer" an EV value from the observed

stat value and the other parameters in the stat-calculating formulae.

We begin by isolating the EV variable in the stat-calculating formulae (we have to drop the floor operators to do this, since it is not bijective and thus does not have an inverse operator):

$$E = (\frac{4 \cdot (100 \cdot S_{HP} - 100 \cdot L - 1000)}{L}) - 8 \cdot B - 4 \cdot I$$

$$E = (\frac{(400 \cdot S_O - 2000 \cdot N)}{L \cdot N}) - 8 \cdot B - 4 \cdot I$$

Let's try this formula out with the HP stat of our example Arceus:

$$E = (\frac{4 \cdot (100 \cdot 435 - 100 \cdot 98 - 1000)}{98}) - 8 \cdot 120 - 4 \cdot 31 = 250.69$$

We know that 252 EVs have actually been allocated to the HP stat of this Arceus. Thus, our "reverse-engineered" value is pretty close, but we must now make the *inferential* step to get to the correct EV value.

First, we should recall that we simply seek to compute the *lowest-possible* number of EVs that must be allocated to a stat in order to generate an observed stat value (i.e. giving the "benefit of doubt" to the user by maximizing the available remaining EVs after allocating a minimized amount to some stat). Thus, we can "fix" the number of EVs at 31. To see why, note that if the observed value of the stat is *less* than the predicted stat value with 31 IVs and 0 EVs (i.e. with no EV investment into that stat), we don't need to worry about computing a "minimum" required number of EVs, since we can achieve that stat value by simply reducing the IV value. In other words, when the the observed value of the stat is below the predicted stat value with max IVs and no EVs, the minimum required number of EVs to generate that observed stat value is *zero*. Thus, if the observed stat value is *larger* than the predicted stat value with 31 EVs and 0 EVs, it means that *some* EV investment into that stat must have occurred. By finding the number of EVs necessary to reach the observed stat value starting at this max-IV, min-EV stat value, we can determine the minimum number of EVs which were necessarily allocated to a stat.

Next, we must consider the way the floor operators affect the behavior of the stat-calculating formulae. There are two important floor operators: the floor operator around the larger fractional expression, and the floor operator around just the $\frac{E}{4}$ expression. The floor operator around the larger fractional expression in either formula simply ensures that the result is an integer, since all stat values are integers. However, $\lfloor \frac{E}{4} \rfloor$ has the interesting effect of only making each increase of 4 EVs relevant to a Pokemon's stat value. Consider our example Arceus. Suppose we allocated only 251 EVs to the HP stat. The computed HP stat would then be:

$$S_{HP} = \lfloor \frac{(2 \cdot 120 + 31 + \lfloor \frac{251}{4} \rfloor) \cdot 98}{100} \rfloor + 98 + 10 = 434$$

We observe that the HP stat has now dropped to 434. Suppose we remove 3 more EVs, reducing the total allocated HP EVs to 248:

$$S_{HP} = \lfloor \frac{(2 \cdot 120 + 31 + \lfloor \frac{248}{4} \rfloor) \cdot 98}{100} \rfloor + 98 + 10 = 434$$

Notice that the computed stat value is *still* 434, despite subtracting three EVs. Now, if we remove yet another EV, we observe that the computed HP stat once again drops:

$$S_{HP} = \lfloor \frac{(2 \cdot 120 + 31 + \lfloor \frac{247}{4} \rfloor) \cdot 98}{100} \rfloor + 98 + 10 = 433$$

Thus, only *multiples of 4* are relevant to determining how many EVs must have been allocated to generate a certain observed stat value. In fact, if the number of EVs returned by our "reverse-engineered" formulae is not a multiple of 4, we *must* round to the nearest multiple of 4 which is larger than the computed EV value, since it must be the case that *at least* those many EVs had to have been allocated in order to generate the observed stat value. The fact that we do not land on exactly that multiple of 4 is simply an artifact of removing the floor operators. To empirically illustrate this concept, consider our "reverse-engineered" value of 250.69 we found previously for Arceus's HP stat. Let's plug this EV value back into the stat-calculating formula to compute the HP stat:

$$S_{HP} = \lfloor \frac{(2 \cdot 120 + 31 + \lfloor \frac{250.69}{4} \rfloor) \cdot 98}{100} \rfloor + 98 + 10 = 434$$

Oops! The computed value is off-by-one relative to the actual stat value of 435. However, rounding the EV value up to the nearest multiple of 4 (i.e. 250.69 becomes 252) resolves this issue:

$$S_{HP} = \lfloor \frac{(2 \cdot 120 + 31 + \lfloor \frac{252}{4} \rfloor) \cdot 98}{100} \rfloor + 98 + 10 = 435$$

This gives us a new set of formulae for computing the EV allocations for a particular stat:

For HP:

$$E_{raw} = \lceil (\frac{4 \cdot (100 \cdot S_{HP} - 100 \cdot L - 1000)}{L}) \rceil - 8 \cdot B - 4 \cdot I$$

$$E_i = \begin{cases} E_{raw} + (4 - (E_{raw} \mod 4)) & E_{raw} \mod 4 \neq 0 \\ E_{raw} & E_{raw} \mod 4 = 0 \end{cases}$$

For other stats:

$$E_{raw} = \lceil (\frac{(400 \cdot S_O - 2000 \cdot N)}{L \cdot N}) \rceil - 8 \cdot B - 4 \cdot I$$

$$E_i = \begin{cases} E_{raw} + (4 - (E_{raw} \mod 4)) & E_{raw} \mod 4 \neq 0 \\ E_{raw} & E_{raw} \mod 4 = 0 \end{cases}$$

Note that we use the ceiling function to round our fractional subexpression up, since *any* value that is even slightly larger than a multiple of 4 *must* round up to the nearest multiple of 4 (since not rounding or rounding down would lead to an off-by-one error, as illustrated above). Then, we find the "distance" from the nearest larger multiple of 4 through the subexpression containing the modulus operator, then add that "distance" to get to that multiple of 4. The exception is when the result E_{raw} is already a multiple of 4 - in that case, there is no need to adjust the value.

Using this logic, we can infer the smallest number of EVs which had to have been allocated to a specific stat for all stats. Then, if the sum of these EV allocations exceeds 508, we assert that the Pokemon is hacked. If the sum is less than or equal to 508, the Pokemon's stat distribution could legitimately be generated by a pair of valid IV and EV distributions (provided no stat value exceeds the minimum of maximum possible value for that stat - same check as in the previous part).

(pseudocode on next page)

Algorithm 2: The revised hack-checking algorithm. Base stat variables are prepended with a "b" and stat multipliers conferred by the Pokemon's nature are prepended with an "m." Returns **true** if the Pokemon is hacked and **false** if the Pokemon is not hacked.

function *compute_hp*:
  B, L, E, I
return $\lfloor \frac{(2 \cdot B + I + \lfloor \frac{E}{4} \rfloor) \cdot L}{100} \rfloor + L + 10$
function *compute_other_stat*:
  B, L, E, I, N
return $\lfloor (\lfloor \frac{(2 \cdot B + I + \lfloor \frac{E}{4} \rfloor) \cdot L}{100} \rfloor + 5) \cdot N \rfloor$
function *infer_hp_evs*:
  B, L, I
$E_{raw} = \lceil (\frac{4 \cdot (100 \cdot S_{HP} - 100 \cdot L - 1000)}{L}) \rceil - 8 \cdot B - 4 \cdot I$
if $E_{raw} \mod 4 = 0$ **then**
  return E_{raw}
else
  return $E_{raw} + (4 - (E_{raw} \mod 4))$
function *infer_other_evs*:
  B, L, I, N
$E_{raw} = \lceil (\frac{(400 \cdot S_O - 2000 \cdot N)}{L \cdot N}) \rceil - 8 \cdot B - 4 \cdot I$
if $E_{raw} \mod 4 = 0$ **then**
  return E_{raw}
else
  return $E_{raw} + (4 - (E_{raw} \mod 4))$
function *is_pkmn_hacked*:
  hp, atk, spa, def, spd, spe, lvl, bhp, $batk$, $bspa$, $bdef$, $bspd$, $bspe$, $matk$, $mspa$, $mdef$, $mspd$, $mspe$
$T = 508$
if $hp \geq compute\_hp(bhp, lvl, 0, 0)$ **and** $hp \leq compute\_hp(bhp, lvl, 252, 31)$ **then**
  if $hp - compute\_hp(bhp, lvl, 0, 31) > 0$ **then**
    $T = T - infer\_hp\_evs(bhp, lvl, 31)$
else
  return true
if $atk \geq compute\_other\_stat(batk, lvl, 0, 0, matk)$ **and**
 $atk \leq compute\_other\_stat(batk, lvl, 252, 31, matk)$ **then**
  if $atk - compute\_other\_stat(batk, lvl, 0, 31) > 0$ **then**
    $T = T - infer\_other\_evs(batk, lvl, 31, matk)$
else
  return true
do the same for spa, def, spd, and spe...
if $T < 0$ **then**
  return true
else
  return false

The final algorithm (written in SQL)

```
CREATE FUNCTION is_pkmn_hacked (pid INT, hp INT, atk INT, spa INT, def INT,
                          spd INT, spe INT, lvl INT, evs DECIMAL(10, 6))
                          RETURNS TINYINT DETERMINISTIC
BEGIN

DECLARE base_h INT;
DECLARE base_atk INT;
DECLARE base_spa INT;
DECLARE base_def INT;
DECLARE base_spd INT;
DECLARE base_spe INT;
DECLARE atk_mult DECIMAL(2, 1);
DECLARE spa_mult DECIMAL(2, 1);
DECLARE def_mult DECIMAL(2, 1);
DECLARE spd_mult DECIMAL(2, 1);
DECLARE spe_mult DECIMAL(2, 1);

SELECT base_hp, base_attack, base_special_attack, base_defense,
base_special_defense, base_speed INTO base_h, base_atk, base_spa, base_def,
base_spd, base_spe FROM pokedex
WHERE pkmn_name = (SELECT pkmn_name FROM has_species WHERE pkmn_id = pid);

SELECT attack_mult, special_attack_mult, defense_mult, special_defense_mult,
speed_mult INTO atk_mult, spa_mult, def_mult, spd_mult, spe_mult FROM nature
WHERE nature_name = (SELECT nature_name FROM has_nature WHERE pkmn_id = pid);

IF hp BETWEEN FLOOR(2 * base_h * lvl / 100) + lvl + 10
AND FLOOR((2 * base_h + 94) * lvl / 100) + lvl + 10
    THEN IF hp - (FLOOR((2 * base_h + 31) * lvl / 100) + lvl + 10) > 0
        THEN SET evs = evs -
        (CEILING(4 * (100 * hp - 100 * lvl - 1000) / lvl) - 8 * base_h - 124);
        IF MOD((CEILING(4 * (100 * hp - 100 * lvl - 1000) / lvl)
        - 8 * base_h - 124), 4) <> 0
            THEN SET evs = evs -
            (4 - MOD((CEILING(4 * (100 * hp - 100 * lvl - 1000) / lvl)
            - 8 * base_h - 124), 4));
        END IF;
    END IF;
ELSE
    RETURN 1;
END IF;

IF atk BETWEEN FLOOR((FLOOR(2 * base_atk * lvl / 100) + 5) * atk_mult)
AND FLOOR((FLOOR((2 * base_atk + 94) * lvl / 100) + 5) * atk_mult)
    THEN
    IF atk - FLOOR((FLOOR((2 * base_atk + 31) * lvl / 100) + 5) * atk_mult) > 0
        THEN SET evs = evs -
        (CEILING((400 * atk - 2000 * atk_mult) / (lvl * atk_mult))
        - 8 * base_atk - 124);
        IF MOD((CEILING((400 * atk - 2000 * atk_mult) / (lvl * atk_mult))
        - 8 * base_atk - 124), 4) <> 0
            THEN SET evs = evs -
            (4 - MOD((CEILING((400 * atk - 2000 * atk_mult) / (lvl * atk_mult))
            - 8 * base_atk - 124), 4));
```

```
        END IF;
    END IF;
ELSE
    RETURN 1;
END IF;

IF spa BETWEEN FLOOR((FLOOR(2 * base_spa * lvl / 100) + 5) * spa_mult)
AND FLOOR((FLOOR((2 * base_spa + 94) * lvl / 100) + 5) * spa_mult)
    THEN
    IF spa - FLOOR((FLOOR((2 * base_spa + 31) * lvl / 100) + 5) * spa_mult) > 0
        THEN SET evs = evs -
        (CEILING((400 * spa - 2000 * spa_mult) / (lvl * spa_mult))
        - 8 * base_spa - 124);
        IF MOD((CEILING((400 * spa - 2000 * spa_mult) / (lvl * spa_mult))
        - 8 * base_spa - 124), 4) <> 0
            THEN SET evs = evs -
            (4 - MOD((CEILING((400 * spa - 2000 * spa_mult) / (lvl * spa_mult))
            - 8 * base_spa - 124), 4));
        END IF;
    END IF;
ELSE
    RETURN 1;
END IF;

IF def BETWEEN FLOOR((FLOOR(2 * base_def * lvl / 100) + 5) * def_mult)
AND FLOOR((FLOOR((2 * base_def + 94) * lvl / 100) + 5) * def_mult)
    THEN
    IF def - FLOOR((FLOOR((2 * base_def + 31) * lvl / 100) + 5) * def_mult) > 0
        THEN SET evs = evs -
        (CEILING((400 * def - 2000 * def_mult) / (lvl * def_mult))
        - 8 * base_def - 124);
        IF MOD((CEILING((400 * def - 2000 * def_mult) / (lvl * def_mult))
        - 8 * base_def - 124), 4) <> 0
            THEN SET evs = evs -
            (4 - MOD((CEILING((400 * def - 2000 * def_mult) / (lvl * def_mult))
            - 8 * base_def - 124), 4));
        END IF;
    END IF;
ELSE
    RETURN 1;
END IF;

IF spd BETWEEN FLOOR((FLOOR(2 * base_spd * lvl / 100) + 5) * spd_mult)
AND FLOOR((FLOOR((2 * base_spd + 94) * lvl / 100) + 5) * spd_mult)
    THEN
    IF spd - FLOOR((FLOOR((2 * base_spd + 31) * lvl / 100) + 5) * spd_mult) > 0
        THEN SET evs = evs -
        (CEILING((400 * spd - 2000 * spd_mult) / (lvl * spd_mult))
        - 8 * base_spd - 124);
        IF MOD((CEILING((400 * spd - 2000 * spd_mult) / (lvl * spd_mult))
        - 8 * base_spd - 124), 4) <> 0
            THEN SET evs = evs -
            (4 - MOD((CEILING((400 * spd - 2000 * spd_mult) / (lvl * spd_mult))
            - 8 * base_spd - 124), 4));
        END IF;
```

```
    END IF;
ELSE
    RETURN 1;
END IF;

IF spe BETWEEN FLOOR((FLOOR(2 * base_spe * lvl / 100) + 5) * spe_mult)
AND FLOOR((FLOOR((2 * base_spe + 94) * lvl / 100) + 5) * spe_mult)
    THEN
    IF spe - FLOOR((FLOOR((2 * base_spe + 31) * lvl / 100) + 5) * spe_mult) > 0
        THEN SET evs = evs -
        (CEILING((400 * spe - 2000 * spe_mult) / (lvl * spe_mult))
        - 8 * base_spe - 124);
        IF MOD((CEILING((400 * spe - 2000 * spe_mult) / (lvl * spe_mult))
        - 8 * base_spe - 124), 4) <> 0
            THEN SET evs = evs -
            (4 - MOD((CEILING((400 * spe - 2000 * spe_mult) / (lvl * spe_mult))
            - 8 * base_spe - 124), 4));
        END IF;
    END IF;
ELSE
    RETURN 1;
END IF;

IF evs < 0
    THEN RETURN 1;
ELSE
    RETURN 0;
END IF;

END !
```

Testing the algorithm

Let's test this algorithm out in our Pokemon Storage System! Suppose we insert an Arceus with all of the stats shown in the example above:

```
Enter an option: a
Whose boxes would you like to add to?
  (s) - your own boxes
  (u) - a different user's boxes
Enter an option: s
Which box would you like to add a Pokemon to?
Box number (1-16): 1
Enter the species of the Pokemon you'd like to add.
Pokemon species name: arceus
What is this Pokemon's nickname? (30 character max, will default to Pokemon species name if left blank)
Pokemon's nickname: god
Enter the Pokemon's HP stat: 435
Enter the Pokemon's Attack stat: 270
Enter the Pokemon's Special Attack stat: 243
Enter the Pokemon's Defense stat: 332
Enter the Pokemon's Special Defense stat: 270
Enter the Pokemon's Speed stat: 298
Enter the Pokemon's level: 98
Enter the Pokemon's nature: jolly
Pokemon successfully added!
```

Now, let's view the box we inserted this Arceus into:

```
What box number would you like to view?
Box number (1-16): 1
Box 1
+-----------+---------------+--------+--------------+-----+--------+---------------+---------+-----------------+-------+-----+-----------+
| pkmn_name | pokedex_number | pkmn_id | pkmn_nickname |  hp | attack | special_attack | defense | special_defense | speed | lvl | is_hacked |
+-----------+---------------+--------+--------------+-----+--------+---------------+---------+-----------------+-------+-----+-----------+
|    onix   |       95      |    3   |     Rocky    | 100 |   77   |       59      |   178   |        53       |   88  |  45 |     0     |
|   golem   |       76      |    4   |    Baller    | 239 |  240   |      104      |   236   |       132       |  101  |  80 |     0     |
|   arceus  |      493      |    6   |     god      | 435 |  270   |      243      |   332   |       270       |  298  |  98 |     0     |
+-----------+---------------+--------+--------------+-----+--------+---------------+---------+-----------------+-------+-----+-----------+
```

Nice! The Arceus shows up as not hacked. Now, what if we try adding an extra stat point to the Attack stat? This would cause the total allocation of EVs to exceed 508, meaning the Arceus should now show up as hacked. Let's try inserting this Pokemon:

```
Enter an option: a
Whose boxes would you like to add to?
  (s) - your own boxes
  (u) - a different user's boxes
Enter an option: s
Which box would you like to add a Pokemon to?
Box number (1-16): 1
Enter the species of the Pokemon you'd like to add.
Pokemon species name: arceus
What is this Pokemon's nickname? (30 character max, will default to Pokemon species name if left blank)
Pokemon's nickname: god-hacked
Enter the Pokemon's HP stat: 435
Enter the Pokemon's Attack stat: 271
Enter the Pokemon's Special Attack stat: 243
Enter the Pokemon's Defense stat: 332
Enter the Pokemon's Special Defense stat: 270
Enter the Pokemon's Speed stat: 298
Enter the Pokemon's level: 98
Enter the Pokemon's nature: jolly
Pokemon successfully added!
```

Now, let's view the box we inserted this second Arceus into:

```
What box number would you like to view?
Box number (1-16): 1
Box 1
+-----------+---------------+--------+--------------+-----+--------+---------------+---------+-----------------+-------+-----+-----------+
| pkmn_name | pokedex_number | pkmn_id | pkmn_nickname |  hp | attack | special_attack | defense | special_defense | speed | lvl | is_hacked |
+-----------+---------------+--------+--------------+-----+--------+---------------+---------+-----------------+-------+-----+-----------+
|    onix   |       95      |    3   |     Rocky    | 100 |   77   |       59      |   178   |        53       |   88  |  45 |     0     |
|   golem   |       76      |    4   |    Baller    | 239 |  240   |      104      |   236   |       132       |  101  |  80 |     0     |
|   arceus  |      493      |    6   |     god      | 435 |  270   |      243      |   332   |       270       |  298  |  98 |     0     |
|   arceus  |      493      |    7   | god-hacked   | 435 |  271   |      243      |   332   |       270       |  298  |  98 |     1     |
+-----------+---------------+--------+--------------+-----+--------+---------------+---------+-----------------+-------+-----+-----------+
```

Our algorithm detected that it's hacked!