Genetic Algorithm made intuitive with natural selection and python

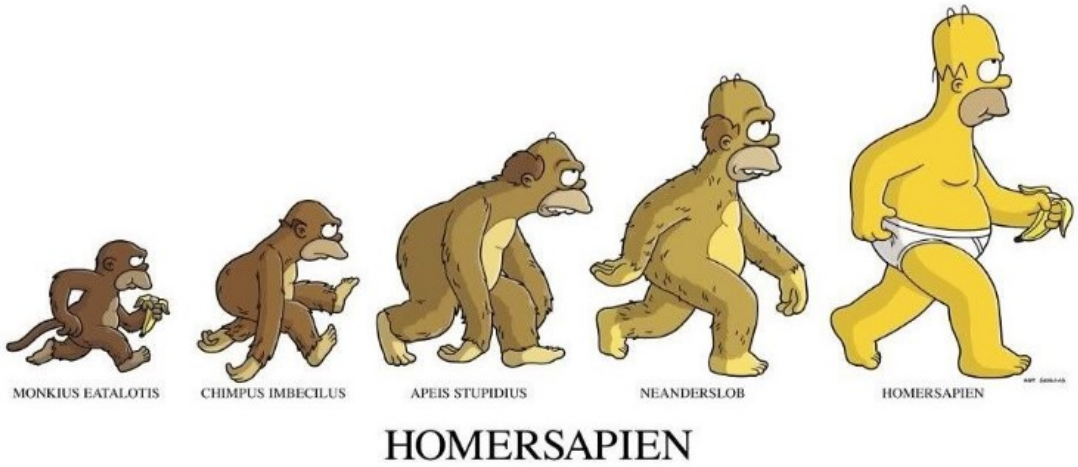


Image source — https://www.pinterest.com/pin/295126581801065984/?lp=true

What to expect

My goal is to help you gain an **intuitive** understanding of the genetic algorithm in the *context of evolution*. We will take a look at birds flying in a **V** shape, then use our understanding of evolution to write a code for finding someone's password.

Quick Introduction

Genetic algorithm is from the family of **Reinforcement learning** which is a subset of AI.

Genetic algorithm is the ***hero***, who made the journey from a silly monkey on the tree to clever monkey which is capable of understanding evolution.

[Disclaimer]

[I'm not at all a biologist neither a Machine Learning engineer yet, so please check the Learn more section for more experienced people than me. I'm gonna extremely simplify and sometimes maybe lie a bit, but that's only for making it stick in your brain. 😉]

Birds example 🐦

So how do we and all other leaving creatures evolved so much? There is no special function for making neither nice unicorns nor hippos. So let us break down stage by stage how it was done.

I won't explain the science behind the reason why this structure is so gorgeous read it here [1] [2].

1. Starting random

Let's imagine that there were lots of birds which had to fly to far and warm places every winter for **surviving**.

Those birds didn't have anything special, their genes(something that would define their behaviour) were pretty much **random**.

The first winter came and some birds were not able to make it to warm places, because they were let's say very aggressive and got kicked out by other birds. Some of the birds that succeeded journey were kind and collaborative. (*just an example characteristics*)

Okay now make sure you understood everything above and let those birds have some sexy times.

2. Examining their children

Well so now what about the next generation.

Aggressive birds died so they hadn't had any babies. But ones that where collaborating had. So those new birds will have again pretty much random genes but this time their parents were collaborating, so most likely children will collaborate as well. Also, they're parents where not aggressive => children won't have a tendency to be aggressive, hopefully.

Some of them won't get "being collaborative gene", some will be aggressive, anyways, let's not focus on them.

Damn! Winter again

This time even a colder one. Those poor birds are once more flying to warmer places, but this time collaborating isn't enough, so let's hope some of them won't just fly straight but will form some kind of shapes and develop strategies with their fellow travellers.

Sadly aggressive birds, birds that flew backwards(don't know why), those who were not enduring cold passed away. We're interested in the strongest ones, those who made the journey. They still have some "bad genes" but the bottom line is that they are flying in a better way than their ancestors.

3. Repeating all that stuff

Those survivors passed their genes to children, children got also some random genes, and when tough times come again those who were lucky enough to receive a good set of genes will have some even better children.

Repeat this for 1000s of times and you hopefully get most aerodynamically perfect birds.

Nature works roughly this way, if you want to survive than be the strongest one. We all have to take a page from nature's book.

Some "human" examples.

If you get to be alone for a really long time you might start feeling miserable, maybe staying alone is the best choice for you, but it sure wasn't for your old-old ancestors that just weren't able to fight mammoths by their own. Also, your ancestors weren't loners, because they ended up finding a partner and having babies.

Love dogs? Me too. That's because humans "created" them. We domesticated those wolfs which were friendly and devoted. Being friendly to people become key for surviving and those who weren't flexible to change didn't get selected.

Important points

1. We need lots of birds(if we have few they might die before figuring anything out)
2. Birds should have some goal or objective of survival(in our case being able to fly far)
3. Bad birds must get punished(e.g. aggressive birds died)
4. Good birds must get rewarded(staying alive and reproducing with their strong partners)
5. Birds must have a way to pass their genes to further generations.

Talk is cheap. Show me the code[4]

Here is the task. We want to get some secret password. We know that passport can contain only lowercase and uppercase English letters, digits and let's say a space as well.

We can try to guess it as long as we want and, instead of getting answer that the password is wrong we will also get how many characters matched in our guess and the correct password.

For every character, there are 26 + 26 + 10 + 1 = 63 possibilities. Let's say the password is 32 letters long. That means that there are 63^{32} possibilities which is approximately $64^{32} = 2^{256}$

That's an enormously enormous number, if you can crack so secure password, you might as well be able to destroy whole internet security[3]

So if we try every possible guess, it won't work, our sun would have already exploded by the time we finished checking a tiny tiny fraction of the guesses.

Nature doesn't try every possible combination of genes, it's a lot smarter, so let's play some nature.

Let's treat those password guesses like they are birds.

So what we need to start our evolution process.

1. We need to have a population of birds(guesses)
2. We need to understand which guesses are good(like birds that were able to fly to warm places, were the good ones)
3. We need to make new generations from those good guesses.
4. We need to repeat the process.

Disclaimer

[Code is going to be very slow and not optimal, my goal is to make everything easy to understand and I'm compromising it with compilation speed, and more lines writtern]

Choo-Choo-Choo, let's start

First of all importing some libraries and declaring globals

```
import random
import string
import numpy as np

PASSWORD = "Unicorns"
POPULATION_SIZE = 1_000
```

1.So how do we create(generate) some population?

We can get our possible characters of the password from python String module, and use the random module to select some random chars.

Our function has to know how long guess to generate. Let's do it with one function if it's not clear for you, consider improving your python skills or maybe dropping me an email.

```
def generate_initial_population(password_length, population_size):
    """
    Function takes as input length of the guess to generate, and
quantity
```

```
    of them. Returns a list of randomly generated guesses
    """
    population = []
    letters = string.ascii_letters + string.digits + " "

    for i in range(population_size):
        guess = ""
        for i in range(password_length):
            guess += random.choice(letters)

        population.append(guess)

    return population


print (generate_initial_population(5,3))
# outputs something like ['4a5jO', ' VQDc', 'x4bSr']
```

2. Evaluating guesses

So now when we have our population, how do we find out which of the guesses are the best birds. The simplest way to do is just count how many characters match in the password and individual guess

```
def match_score(password, guess):
     """
     Functons takes two strings, and counts how many characters matched.
     Retruns an integer
     """
     score = 0
     for i in range(len(target)):
         if target[i] == word[i]:
             score += 1
     return score

 # some examples
 print (match_score("1dsaSf", "e34aSy")) # ->  prints  2 ('a' and "S" matched)
 print (match_score("Elephant", "dlepghfd")) # -> prints 3 ("l","e", "p" matched)

```

Okay, good enough. We have a function to generate a population, and a function to evaluate how good individual “population member “ is. Next step is to evaluate our full population.

```
def score_for_population(password, population):
     """
     Functions uses match_score() and iterates through whole population
```

```
    returns list of ntegers representng score of every populaton member
    """
    scores = []
    for i in population:
        scores.append(match_score(password, i))
    return scores

# example
password = "Elephant"
population = ["dlepghfd", "12345678", "asdehant"]

print (score_for_population(password, population))
"prints [3, 0, 4]"
```

3. Reproducing new guesses

Now when we know how good every guess is, let's pick 2 best ones.(***we pick 2 for simplicity, like picking mother and father, experimenting with this number can be good for improving the model***)

We have 2 lists, one with all the guesses and one with their corresponding scores. So score for the N th member of population is stored in N th place of scores list.

So let's just pick the indexes of the highest values of the scores list, and extract those population members stored in those positions.

```
def choose_parents(population, scores):
    """
    Function takes list of population members and list of their corresponding
    scores, and returns 2 members with the highest scores
    """
    father_index = np.array(scores).argmax()
    father = population[father_index]

    # we are removing the biggest elements from list,
    # than doing the same thing above to find second biggest.
    population.remove(father)
    scores.remove(scores[father_index])

    mother_index = np.array(scores).argmax()
    mother = population[mother_index]

    return [father, mother]

# example
population = ["dlepghfd", "12345678", "asdehant"]
scores = [3, 0, 4]

print(choose_parents(population, scores))
# print ['dlepghfd', 'asdehant']
```

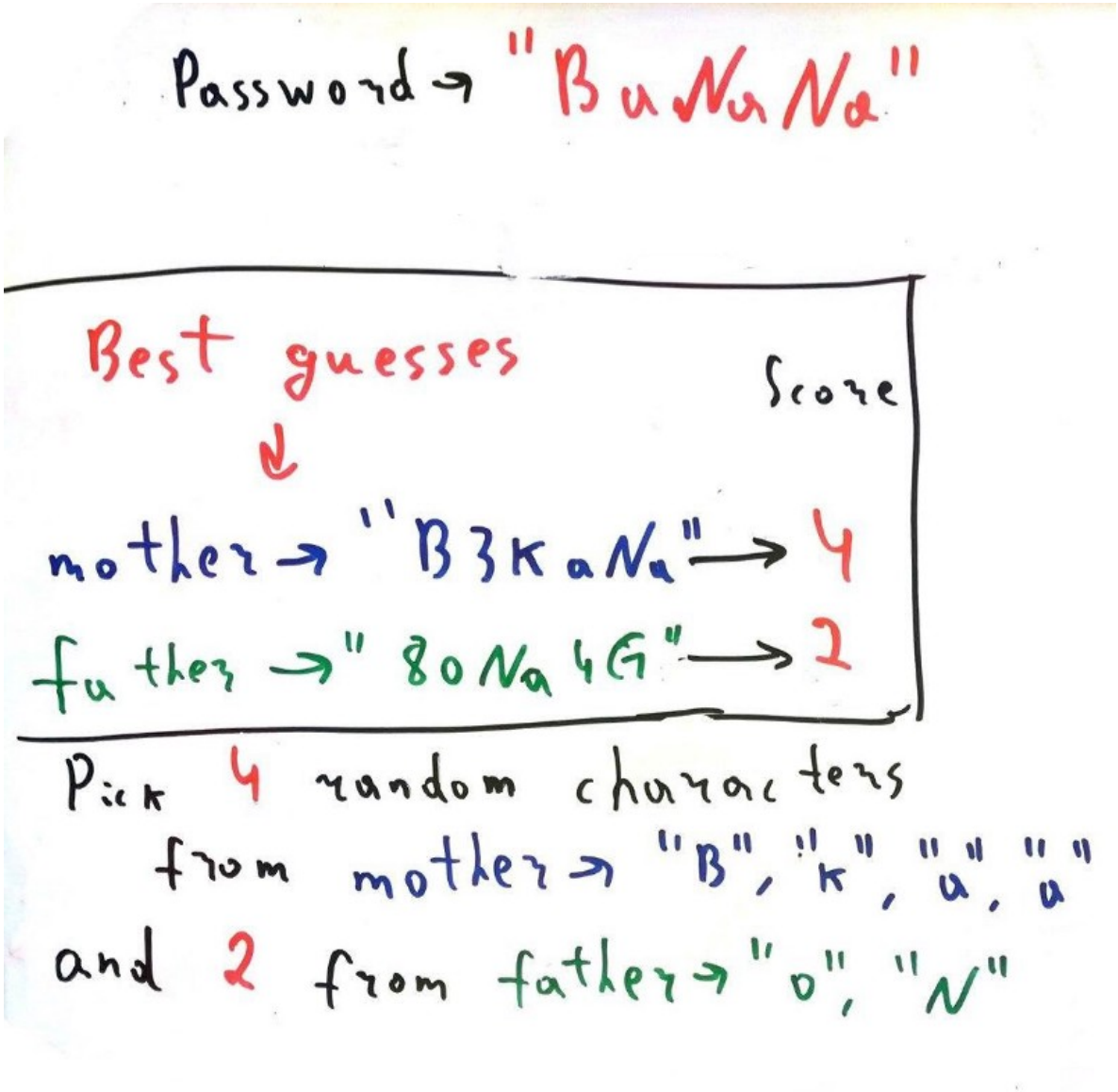
Summing up with birds analogy.

We have the first population of birds, we evaluated how strong each bird is, and picked the best 2 birds. Now those best birds should have some fun passing genes to the next population, so how we do it.

This part is great for experimenting. This time my method isn't the simplest so you may need a bit more time to understand this part, but it worth it.

How do we create a whole population out of 2 best guesses? Parents are the strongest birds from the previous generation, so we should mix their genes in order to improve the next population.

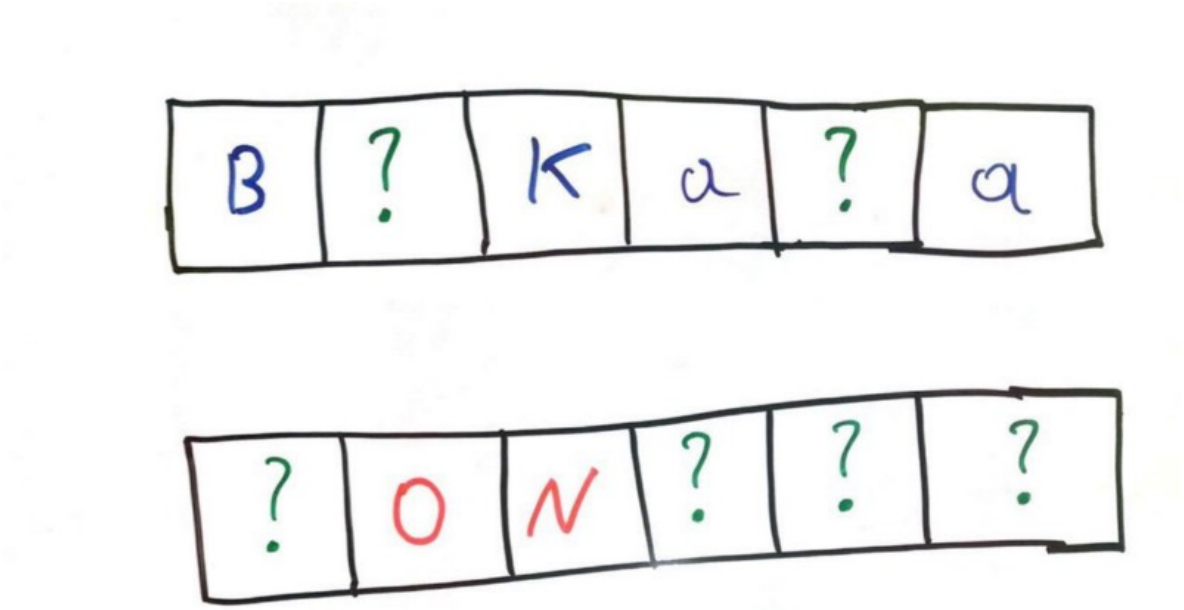
[**NOTE:**] For simplicity, we won't mix fathers and mothers genes simultaneously, which would have been a good thing to do. Instead, we will create half of the population from mothers genes, and a half from fathers(again, very arguable, you can try other approaches.).



explanation of what is going on in image below

For each parent we are gonna take some of their genes and fill the remaining letters with some random character. We could have picked equal quantity of genes from each parent, but as you can see in the example, mother's score is 4, and father's 2, that means that mother has 2 times more "correct genes" so it's wiser to pick more genes from her, instead of treating both guesses the same way.

Eventually, we decided to pick as many characters as big the score is. So as you see we are picking 4 random characters from mother and 2 from the father(*randomly picking is just my approach, you can pick let's say only first one and last one, only first half, and so on, there are lots of options*).



Our problem came to fill this **?** signs, with random characters. Lets code that stuff.

```
def generate_new_member(parent, parent_score):
    """
    Takes as input string which we will, pick letters from(parent),
    and quantity of letters to pick(parent_score)
    Returns a newly generated string"""

    new_guess = ""

    letters = string.ascii_letters + string.digits + " "
    random_indexes = random.sample(range(len(parent)),
parent_score)

    for i in range(len(parent)):
        if i in random_indexes:
            new_guess += parent[i]
        else:
            new_guess += random.choice(letters)

    return new_guess

# example
parent = "aaaaa"
```

```
parent_score = 3

print(generate_new_member("aaaaa", 3))
# prints something like "aGaai", which means 0,2,3 indexes were picked
# and the rest was filled with random characters
```

Now all we have to do is call this function on the half of the population with mother as an argument and for half as a father. We can split the population, just by looping through the first half, then second, but I'm gonna split by index. One group those with even index, one group with odd index. That's not important.

```
def generate_new_population(father, mother):
    """
    Takes as input parents and reproduces new population from them, using
    generate_new_member() function with alternatively father and mother.
    Returns list with new population members
    """
    population = []
    father_score = match_score(PASSWORD, father)
    mother_score = match_score(PASSWORD, mother)

    for i in range(POPULATION_SIZE):
        if i % 2 == 0:
            new_member = generate_new_member(father, father_score)
        else:
            new_member = generate_new_member(mother, mother_score)
        population.append(new_member)
    return population
```

I'm just referring to global variables here. If you want to use this snippet don't forget about adding population_size and target as parameters.

All the wheels are ready, you can use those snippets to build your better algorithm

Here is **full code**, see the output below.

```
# by Hayk Tarkhanyan, finished 11.02.2020

import random
import string
import numpy as np

PASSWORD = "Unicorn"
POPULATION_SIZE = 1000


def generate_initial_population(password_length, population_size):
    """
```

```
    Function takes as input length of the guess to generate, and quantity
    of them. Returns a list of randomly generated guesses
    """
    population = []
    letters = string.ascii_letters + string.digits + " "

    for i in range(population_size):
        guess = ""
        for i in range(password_length):
            guess += random.choice(letters)

        population.append(guess)

    return population


def match_score(password, guess):  # +
    """
    Function takes two strings, and counts how many characters matched.
    Returns an integer
    """
    score = 0
    for i in range(len(password)):
        if password[i] == guess[i]:
            score += 1
    return score


def score_for_population(password, population):  # +
    """
    Functions uses match_score() and iterates through whole population
    returns list of integers representing score of every population member
    """
    scores = []
    for i in population:
        scores.append(match_score(password, i))
    return scores


def choose_parents(population, scores):
    """
    Function takes list of population members and list of their corresponding
    scores, and returns 2 members with the highest scores
    """
    father_index = np.array(scores).argmax()
    father = population[father_index]

    # we are removing the biggest elements from list,
    # than doing the same thing above to find second biggest.
    population.remove(father)
```

```
    scores.remove(scores[father_index])

    mother_index = np.array(scores).argmax()
    mother = population[mother_index]

    return [father, mother]


def generate_new_member(parent, parent_score):
    """
    Takes as input string which we will, pick letters
from(parent),
    and quantity of letters to pick(parent_score)
    Returns a newly generated string"""

    new_guess = ""

    letters = string.ascii_letters + string.digits + " "
    random_indexes = random.sample(range(len(parent)),
parent_score)

    for i in range(len(parent)):
        if i in random_indexes:
            new_guess += parent[i]
        else:
            new_guess += random.choice(letters)

    return new_guess


def generate_new_population(father, mother):
    """
    Takes as input parents and reproduces new population from
them, using
    generate_new_member() function with alternatively father and
mother.
    Returns list with new population members
    """
    population = []
    father_score = match_score(PASSWORD, father)
    mother_score = match_score(PASSWORD, mother)

    for i in range(POPULATION_SIZE):
        if i % 2 == 0:
            new_member = generate_new_member(father, father_score)
        else:
            new_member = generate_new_member(mother, mother_score)
        population.append(new_member)
    return population


def main():
    """
    better way of doing would be using a class.
    """
```

```
    generation_number = 0

    initial_population = generate_initial_population(
        len(PASSWORD), POPULATION_SIZE)
    scores = score_for_population(PASSWORD, initial_population)

    father, mother = choose_parents(initial_population, scores)

    for i in range(10_000):
        generation_number += 1
        new_population = generate_new_population(father, mother)
        new_scores = score_for_population(PASSWORD,
new_population)

        father, mother = choose_parents(new_population,
new_scores)
        father_score = match_score(father, PASSWORD)
        mother_score = match_score(mother, PASSWORD)

        if max(father_score, mother_score) == len(PASSWORD):
            print(generation_number, father, mother)
            print("TOOK {} generations to
finish".format(generation_number))
            break

        print(generation_number, father, mother, father_score)

    return generation_number


if __name__ == "__main__":
    main()
```

I print generation number, best 2 guesses, and score of the best guess.

1. hVcNpn ndpVorB Score: 2
2. WdtcorD nBcIyn Score: 3
3. lntcorA UHgcora Score: 4
4. UGicorW Uuvcorn Score: 5
5. UWicorn Unhcorn Score: 6
6. Unicorn

Took 6 generations to finish

Cool! there were 4 902 227 890 625(6^{57}) possibilities and we cracked that password just by looking on 6000(6 generations * 1000 population members).

We can feel proud of ourselves, we all deserved some cookies. 🍪🍪🍪🍪🍪

What to do next

1. Experiment with the population size.

2. Try selecting more than 2 parents from each generation(or maybe less, it's up to you)
3. Make the code better, maybe use NumPy library
4. Most important — Change the mechanism of passing the genes
5. Maybe add mutation — sometimes change some of the genes randomly
6. Change the way of choosing parents(instead of always choosing the highest scores, choose chem most of the time, like if scores are 2, 10 make choosing member with score 10, 5 times more likely than with 2.)
7. Give your imagination freedom, there is so much to improve and improvise here.

Outro

Thank you! This is my first try of explaining something so please share your feedback, it's important for me. Tell me what I did wrong, what I should have simplified more, what I shouldn't. If you have any questions I'd love to try to help. And of course, check the reference and more to learn sections.

Good luck on your journey.

May the force be with you

Contact info

email — hayk_tarkhanyan@outlook.com

LinkedIn — https://www.linkedin.com/in/hayktarkhanyan/

Learn more.

Evolution related

- How evolution works — https://www.youtube.com/watch?v=hOfRN0KihOU
- Simulating natural selection — https://www.youtube.com/watch?v=0ZGbIKd0XrM
- Examples of human evolution — https://www.businessinsider.com/recent-human-evolution-traits-2016-8#5-missing-wisdom-teeth-5
- Genetic engineering and much more — https://www.youtube.com/watch?v=n___42UNIhvU&t=8s

Programming related

- Playlist on Genetic Algorithm with p5.js — https://www.youtube.com/playlist?list=PLRqwX-V7Uu6bJM3VgzjNV5YxVxUwzALHV
- Intro to Genetic Algorithm — https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3
- After Genetic algorithm, you can move on to Neuroevolution(NEAT) -https://www.youtube.com/playlist?list=PLRqwX-V7Uu6Yd3975YwxrR0x40XGJ__KGO
- Flappy bird Neuroevolution with python — https://www.youtube.com/watch?v=MMxFDaIOHsE

References

1. Why birds fly in a V formation, video explanation — https://www.youtube.com/watch?v=EcX56-E842Q

2. Why birds fly in a V formation article — https://www.nationalgeographic.com/science/phenomena/2014/01/15/birds-that-fly-in-a-v-formation-use-an-amazing-trick/
3. How big is 2^{256} — https://www.youtube.com/watch?v=S9JGmA5_unY
4. Quote by Linus Torvalds — https://en.wikipedia.org/wiki/Linus_Torvalds