Genetic Algorithms and Substitution Ciphers

Eckel, TJHSST AI2, Spring 2020

# Background and Explanation

Given a strategy, it’s often possible to improve the strategy by making random changes, testing the result of those changes, and keeping only the better outcomes. This is called **hill climbing**. This is relatively simple to code, but doesn’t work in every situation – it can often get caught in a local maximum, a strategy that is not ideal but from which any single change is not an improvement on its own.

**Genetic algorithms** are a powerful tool for improving on the idea of hill climbing to optimize a solution in situations where all of the following are true:

* A strategy can be precisely quantified by a specific set of variables given certain numeric values.
* The outcome of the strategy can also be precisely quantified.
* There are interactions between the variables that make simple hill climbing inefficient or unlikely to succeed.

The components of a genetic algorithm are as follows:

1. A **population** of strategies. Instead of improving one strategy, genetic algorithms create a large set of strategies and recombine them in each generation.
2. A **fitness function** that evaluates the success of any particular strategy.
3. A **selection method** that makes better strategies (ie, strategies that return more desirable values from the fitness function) more likely to breed. To increase genetic diversity, it shouldn’t be true that the very best strategies are the *only* ones that breed, but better strategies should be more *likely* to.
4. A **breeding process** that selects characteristics from two different strategies to form a new child strategy.
5. A small chance of a **mutation**, an unpredictable change in one of the strategy’s variables, as each new child is created.

This together has the effect of simulating something much like natural selection (survival of the fittest) in nature, allowing a diversity of strategies that helps avoid getting stuck in a local maximum while still generally improving. Genetic algorithms are remarkably effective at some problems that seem untenable otherwise!

We will implement a genetic algorithm to decode substitution ciphers.

# Substitution Ciphers

A simple substitution cipher is a cryptographic cipher that replaces each letter systematically with another. For instance, a substitution cipher could be defined by the following origin and cipher alphabets:

ABCDEFGHIJKLMNOPQRSTUVWXYZ  
XRPHIWGSONFQDZEYVJKMATUCLB

To encode a message, simply replace each letter with the corresponding letter in the cipher alphabet. Using this particular cipher, “HELLO” would be encrypted as “SIQQE”.

There are 26! = 403,291,461,126,605,635,584,000,000 possible substitution ciphers. Given an encoded block of text, a brute force solution is obviously not feasible!

Traditionally, a substitution cipher is cracked using frequency analysis (for instance, the most common letter is likely to be “E”) and then judgment as you, the human decoder, look for fragments that look like words and try possibilities until it works. In my cryptography class, most students, working in pairs, could do 5 messages in a class period. That’s clearly too much work; let’s use a genetic algorithm instead. Why figure something out the long way when a computer can do it for you in seconds?

# Genetic Algorithm Application and Parameters

There are a lot of decisions to make when implementing a genetic algorithm. As this is our first exercise, I’ll give a set of decisions that definitely works along with a list of parameters you can vary to produce better results.

**If you would like to try a different implementation, feel free! I can guarantee that my implementation below will work, but there are certainly other valid solutions. If you can find something better I’d love to hear about it!**

In any case, here’s one implementation that works. Let’s go step by step.

## 1 – Population

This is straightforward – just **generate random permutations of the alphabet**. We will need to choose our population size.

## 2 – Fitness Function

Define an n-gram to be a consecutive sequence of n letters inside a word. So, for example, the word “example” has four different 4-grams: “exam”, “xamp”, “ampl”, and “mple”. The word “example” has five different 3-grams: “exa”, “xam”, “amp”, “mpl”, and “ple”. Etc, etc.

We can choose a value for n and then take a string of text that we have “decoded” using a member of our population of ciphers and find all the n-grams in it. We can then score our “decoded” text by finding the frequency of each of those n-grams in English and adding them together. The closer we get to English, the higher the number should be.

On Blackboard, you’ll find a data file that is the result of looping over an enormous corpus of English and counting the occurrence of each n-gram. Use that as the source of your frequency numbers.

It does turn out that there’s a problem with this approach, which is that some n-grams in English occur *way, way* more than others, and so a wrong answer that happens to create one or two common n-grams by mistake can overwhelm a correct answer with many more n-grams that are less frequent. This problem can be solved by summing the log of each frequency number instead of the frequency numbers themselves, as this shrinks the gap between most frequent and less frequent n-gram scores. Of course, in addition to choosing the value for n, we will have to pick the base we want to use for the log. My best results usually came from log base 2 and 4-grams, though when a ciphertext contains a lot of short words, sometimes 3-grams work better. Feel free to experiment with that yourself.

To summarize: **our fitness function will be the sum of the logs, base 2, of the frequencies of each 4-gram that occurs in the text “decoded” by the cipher.**

## 3 – Selection Method

We want better strategies to be selected more often, but we also want a healthy element of randomness. It’s obviously a bad idea to just pick the best two strategies to breed all the children themselves; our population will quickly converge and get stuck at what is most likely a local maximum but not an ideal solution.

So, for each child that needs to be generated as we create the next generation, **we will use a “tournament” to choose each child’s two parents**. We will decide on a tournament size and then choose twice that many distinct strategies from our population, splitting them into two tournaments. Within each tournament, we will score each strategy. Then, the winning strategy will be selected with a certain probability. If that fails, the next highest strategy will be selected with that same probability. If that fails, the next, and so on.

We will need to choose tournament size and the probability by which the winner will be selected.

In addition to the tournaments, we may wish to keep some number of the best strategies precisely identical from one generation to the next so that the most successful strategies have more chances to breed. As such, we will also need to choose how many **exact clones are kept from one generation to the next**. To be clear: clones are NOT randomly selected; they are the best strategies from one generation, with no random variation.

## 4 – Breeding Process

Once we select the two parents to breed, we need to generate a child. This process should *also* have a healthy element of randomness – if the same two parents are selected again, they should still generate a *different* child. We do, however, want to ensure that the same letter doesn’t appear in our decoding alphabet twice, so we can’t just take some of the letters from one parent and some from the other.

So, what we will do is **choose a certain number of locations and copy the letter in each of those locations to the child** in the same location as the parent. Then, we will loop over the *other* parent’s alphabet, in order, and each time we encounter a letter that isn’t already in the child, we will place that letter in the next available open space.

We will need to choose the number of crossover locations.

## 5 – Mutation

After each child is generated, there should be a certain probability of a random mutation. In our case, a random mutation would simply be **two randomly chosen letters swapping locations**.

Of course, we will need to choose the probability of mutation.

**NOTE:** After generating a child, **make sure it is unique in the new population** before adding it. Don’t duplicate children!

## Final List of Parameters

That leaves us with the following list of parameters that we must set. **Make these GLOBAL CONSTANTS that you set at the top of your code so that they can be modified easily. Do NOT hardcode any of these values inside any method.**

POPULATION\_SIZE – the number of strategies in each generation  
CLONES – the number of precisely cloned strategies retained from each generation to the next  
TOURNAMENT\_SIZE – how many strategies selected for each tournament  
TOURNAMENT\_WIN – the probability with which the best strategy in a tournament is selected  
CROSSOVER\_POINTS – how many exact letters from parent 1 are copied to the child in the same locations  
MUTATION\_RATE – the chance that a child experiences a mutation after being generated

## My Values

Through a fair amount of entirely unscientific experimentation, these are the values that seemed to produce the best results for me, though I haven’t rigorously tested and am sure this is not ideal. Almost everything I’ve found online suggests a much lower mutation rate, for instance; .8 is incredibly high. Most often, I see numbers like .05 or .1. But it seemed to help my code… In any case, feel free to modify and see what happens! (It’s particularly fun to watch what happens when you have zero clones.)

POPULATION\_SIZE = 500  
CLONES = 1  
TOURNAMENT\_SIZE = 20  
TOURNAMENT\_WIN = .75  
CROSSOVER\_POINTS = 5  
MUTATION\_RATE = .8

## Helpful Python Methods

You will need to import math and find the log function there. Also the following methods from random are essential:

if random.random() < prob: This will trigger a random event with probability prob.

things = random.sample(iter\_obj, n) This will randomly select n **DISTINCT** items from any iterable object.

It might also be a good idea to re-familiarize yourself with how to use lambda functions as keys in a sort call.

# Required Task

Remember what you learned from Crosswords – start small, have a plan, and test your code piece by piece!

First, make methods that will encode and decode a certain piece of text given a certain cipher alphabet. **Any non-alphabetic characters should be left unchanged; only replace letters.** How you deal with upper and lower cases is up to you; my code just upper-cases everything, which is a fine solution. **TEST THIS** before you move on – encode a message and then decode it and make sure it comes out correctly. (Also, if you’re looking for a totally unnecessary bonus challenge, try getting each of these methods in one line!)

Second, code the fitness function, selection process, and breeding function **separately** and test them all **individually**. In particular, ensure that an actual English string is scored more highly than an encoded string and make sure that all children contain each letter precisely once.

Third, code the full genetic algorithm. Remember not to allow duplicate children in each new generation!

Fourth, **decode 10 of the messages** on the following pages. Put all of your decoded messages into a document.

Fifth, submit your **document and code** to the given link, meeting the specification below.

# Specification

Submit your **document and code** to <https://tinyurl.com/S20EckelGenSub>.

This assignment is **complete** if:

* The “First Name” field on the Dropbox submission form contains your **class period**, not your name.
* The “Last Name” field on the Dropbox submission form contains your **last name then a comma then your first name** (like, for example, “Eckel, Malcolm”).
* You submit a document containing 10 decoded messages.
* Your code accepts **a single command line argument** – a long string of text. (I’ll use quotes so that spaces are part of the input.)
* Your code runs a **500 generation or less** genetic process, printing the best result **after each generation**. I do not expect this to work every time, but it should have **a high chance of succeeding**. The highest difficulty will be “Medium”; if it took you like 30 tries to decode each Medium message, you need to tweak your parameters!

For **resubmission**:

* Complete the specification correctly.

# Specification for Outstanding Work

The basic idea: find a way to make your decodes work 100% of the time. Be creative. Otherwise, same as above.

Submit your **document and code** to <https://tinyurl.com/S20EckelGenSubOW>.

This assignment is **complete** if:

* You meet all the above requirements, **and:**
* In your document, you have decoded **all 12 of the Easy, Medium, and Hard messages**.
* Your code **solves the given input correctly every single time in 500 generations or less**. (I will run your code on 10 different messages; all must be correct. The highest difficulty will be “Medium”.)

For **resubmission**:

* Complete the specification correctly.

# Decoding problems

In the document you submit, you must include any 10 of these decoded correctly. They’re listed in what seems to me, based on the way my code has performed, to be an approximate order of difficulty, but who knows really.

## Approximately Easy:

1. PF HACYHTTRQ VF N PBYYRPGVBA BS SERR YRNEAVAT NPGVIVGVRF GUNG GRNPU PBZCHGRE FPVRAPR GUEBHTU RATNTVAT TNZRF NAQ CHMMYRF GUNG HFR PNEQF, FGEVAT, PENLBAF NAQ YBGF BS EHAAVAT NEBHAQ. JR BEVTVANYYL QRIRYBCRQ GUVF FB GUNG LBHAT FGHQRAGF PBHYQ QVIR URNQ-SVEFG VAGB PBZCHGRE FPVRAPR, RKCREVRAPVAT GUR XVAQF BS DHRFGVBAF NAQ PUNYYRATRF GUNG PBZCHGRE FPVRAGVFGF RKCREVRAPR, OHG JVGUBHG UNIVAT GB YRNEA CEBTENZZVAT SVEFG. GUR PBYYRPGVBA JNF BEVTVANYYL VAGRAQRQ NF N ERFBHEPR SBE BHGERNPU NAQ RKGRAFVBA, OHG JVGU GUR NQBCGVBA BS PBZCHGVAT NAQ PBZCHGNGVBANY GUVAXVAT VAGB ZNAL PYNFFEBBZF NEBHAQ GUR JBEYQ, VG VF ABJ JVQRYL HFRQ SBE GRNPUVAT. GUR ZNGREVNY UNF ORRA HFRQ VA ZNAL PBAGRKGF BHGFVQR GUR PYNFFEBBZ NF JRYY, VAPYHQVAT FPVRAPR FUBJF, GNYXF SBE FRAVBE PVGVMRAF, NAQ FCRPVNY RIRAGF. GUNAXF GB TRAREBHF FCBAFBEFUVCF JR UNIR ORRA NOYR GB PERNGR NFFBPVNGRQ ERFBHEPRF FHPU NF GUR IVQRBF, JUVPU NER VAGRAQRQ GB URYC GRNPUREF FRR UBJ GUR NPGVIVGVRF JBEX (CYRNFR QBA'G FUBJ GURZ GB LBHE PYNFFRF – YRG GURZ RKCREVRAPR GUR NPGVIVGVRF GURZFRYIRF!). NYY BS GUR NPGVIVGVRF GUNG JR CEBIVQR NER BCRA FBHEPR – GURL NER ERYRNFRQ HAQRE N PERNGVIR PBZZBAF NGGEVOHGVBA-FUNERNYVXR YVPRAPR, FB LBH PNA PBCL, FUNER NAQ ZBQVSL GUR ZNGREVNY. SBE NA RKCYNANGVBA BA GUR PBAARPGVBAF ORGJRRA PF HACYHTTRQ NAQ PBZCHGNGVBANY GUVAXVAT FXVYYF, FRR BHE PBZCHGNGVBANY GUVAXVAT NAQ PF HACYHTTRQ CNTR. GB IVRJ GUR GRNZ BS PBAGEVOHGBEF JUB JBEX BA GUVF CEBWRPG, FRR BHE CRBCYR CNTR. SBE QRGNVYF BA UBJ GB PBAGNPG HF, FRR BHE PBAGNPG HF CNTR. SBE ZBER VASBEZNGVBA NOBHG GUR CEVAPVCYRF ORUVAQ PF HACYHTTRQ, FRR BHE CEVAPVCYRF CNTR.
2. LTQCXT LRJJ HJRDECD, EZT CDJP SXTFRYTDE EC ZNKT LTTD RASTNHZTY VNF NDYXTV WCZDFCD. ZT VNF NHUBREETY LP N FRDGJT KCET VZTD N LXNKT FTDNECX QXCA ONDFNF XTQBFTY EC PRTJY QXCA SXTFFBXT EC HCDKRHE EZT SXTFRYTDE. ZNY WCZDFCD LTTD HCDKRHETY, EZT FSTNOTX CQ EZT ZCBFT VCBJY ZNKT LTHCAT SXTFRYTDE FRDHT WCZDFCD ZNY DC KRHTSXTFRYTDE. RDHXTYRLJP, RE VNF EZRF FNAT FSTNOTX VZC JTY EZT RASTNHZATDE RD EZT ZCBFT CQ XTSXTFTDENERKTF. EZBF, ZNY EZT FTDNET HCDKRHETY EZT SXTFRYTDE, EZRF VCBJY ZNKT NACBDETY EC N SCJRERHNJ HCBS.
3. ZRTGO Y JPEYPGZA, RP'J IKPGO HIJJRMWG PI RSHEITG PUG JPEYPGZA MA SYDROZ EYOBIS XUYOZGJ, PGJPROZ PUG EGJLWP IK PUIJG XUYOZGJ, YOB DGGHROZ IOWA PUG MGPPGE ILPXISGJ. PURJ RJ XYWWGB URWW XWRSMROZ. PURJ RJ EGWYPRTGWA JRSHWG PI XIBG, MLP BIGJO'P CIED RO GTGEA JRPLYPRIO - RP XYO IKPGO ZGP XYLZUP RO Y WIXYW SYFRSLS, Y JPEYPGZA PUYP RJ OIP RBGYW MLP KEIS CURXU YOA JROZWG XUYOZG RJ OIP YO RSHEITGSGOP IO RPJ ICO. ZGOGPRX YWZIERPUSJ YEG Y HICGEKLW PIIW KIE RSHEITROZ IO PUG RBGY IK URWW XWRSMROZ PI IHPRSRVG Y JIWLPRIO RO JRPLYPRIOJ CUGEG YWW IK PUG KIWWICROZ YEG PELG: Y JPEYPGZA XYO MG HEGXRJGWA QLYOPRKRGB MA Y JHGXRKRX JGP IK TYERYMWGJ ZRTGO XGEPYRO OLSGERX TYWLGJ. PUG ILPXISG IK PUG JPEYPGZA XYO YWJI MG HEGXRJGWA QLYOPRKRGB. PUGEG YEG ROPGEYXPRIOJ MGPCGGO PUG TYERYMWGJ PUYP SYDG JRSHWG URWW XWRSMROZ ROGKKRXRGOP IE LOWRDGWA PI JLXXGGB.
4. CWQ KHTTQKC TFAZJAB HS FGG HS CWQ ECFT YFTE PHRJQE TQGQFEQM EH SFT JE CWQ QPXJTQ ECTJZQE VFKZ, F AQY WHXQ, CWQ GFEC OQMJ, TQCLTA HS CWQ OQMJ, THBLQ HAQ, EHGH, TQRQABQ HS CWQ EJCW, CWQ SHTKQ FYFZQAE, TJEQ HS CWQ EZNYFGZQT, CWQ XWFACHP PQAFKQ, FCCFKZ HS CWQ KGHAQE. CWQ KHTTQKC TFAZJAB HS CWQ CWTQQ JAMJFAF OHAQE PHRJQE JE CWQ GFEC KTLEFMQ, TFJMQTE HS CWQ GHEC FTZ, CQPXGQ HS MHHP. CWQTQ JE AH SHLTCW JAMJFAF OHAQE PHRJQ, FAM FANHAQ YWH CQGGE NHL HCWQTYJEQ JE F GJFT. OLEC CQGG CWQP CH CLTA FTHLAM FAM YFGZ FYFN VQSHTQ CWQN KFA VGQEE NHL YJCW FAN HCWQT JAKHTTQKC HXJAJHAE. FANYFN, EH EFNQCW PN STJQAM VJGG, YWH WFXXQAQM CH VQ HAGJAQ YWJGQ J YFE PFZJAB CWJE FEEJBAPQAC, YWQA J FEZQM WJP 'YWFC YHLGM VQ F BHHM EQKTQC PQEEFBQ SHT PN ECLMQACE CH MQKHMQ?' XGQFEQ CFZQ LX FAN KHPXGFJACE YJCW WJP.

## Approximately Medium:

1. XMTP CGPQR BWEKNJB GQ OTGRB EL BEQX BWEKNJB, G RFGLI. GR GQ BEQX ABSETQB RFGQ QBLRBLSB TQBQ EJJ RBL KMQR SMKKML VMPYQ GL BLDJGQF: 'G FEUB RM AB E DMMY QRTYBLR GL RFER SJEQQ GL RFB PMMK MC RFER RBESFBP.' (If, after decoding this problem, you’re confused as to why it’s in Medium difficulty – I originally made this for my Cryptography class, where this was an easy challenge for the *traditional* method of decoding, but is very short and so would appear to be a bit harder for these programs to get right.)
2. XTV B CHDQCL BHF GCVIVDGDHWPN ABVF ZABPPLHWL, ZTHGDFLV MBJDHW B PTHW BHF XCPPN VLBFBYPL GLVDLG TX UTVFG HLRLV CGDHW B GDHWPL LEBMIPL TX TCV ULPP-PTRLF LHWPDGA WPNIA UADZA TZZCVG GLZTHF IPBZL DH TRLVBPP XVLQCLHZN. DX D BM WLHCDHL, D UDPP GBN MBHN, MBHN GLZTHFG ABRL IBGGLF UADPL D ABRL YLLH ALVL ITHFLVDHW MBJDHW GCZA B UTVJ. FDGZTRLVDHW NTC ZVBZJLF MN YVBDHZADPF, ALVL, DH B GMBPPLV HCMYLV TX GLZTHFG UTCPF WDRL ML HT GCVIVDGL.
3. NU XTZEIMYTNEVZ INUHU YM, ZML SPYVI NXILNFFZ XNFF IVPU N API VNTD. NU PI ILTWU MLI, P XNW YM N FMWY JNZ JPIVMLI LUPWY NWZ MC IVNI YFZEV IVNI ITNDPIPMWNFFZ CMFFMJU 'D' NI NFF. PUW'I IVNI ULTETPUPWY? P CMLWD IVPU ULTETPUPWY, NWZJNZ! NW NLIVMT JVM NFUM CMLWD IVPU ULTETPUPWY, FMWY NYM, NXILNFFZ FMUI SNWZ SMWIVU JTPIPWY N AMMH - N CLFF CPXIPMWNF UIMTZ - JPIVMLI IVNI YFZEV NI NFF. NSNRPWY, TPYVI?
4. RHNJJCBXVCXJYQJNEJNDYDCELTHNBFTVTHNJJREFCLBEECANOTREFDNEBXTHJTNXTXECPCBAPZNSSPXTNYTXFVZCNXTSXRKRJTGTYECJRKTRDFSNHTRANGRDTNKNFEFZTTECQSNSTXCDVZRHZFEXNRGZEJRDTFEXRNDGJTFFUBNXTFSTDENGCDFZTINGCDFNDYCEZTXQRGBXTFRDFETNYCQXTANRDRDGQRITYRDEZTRXSJNHTFACKTQXTTJPNLCBECDCXRDEZTFBXQNHTLBEVREZCBEEZTSCVTXCQXRFRDGNLCKTCXFRDORDGLTJCVREKTXPABHZJROTFZNYCVFCDJPZNXYVREZJBARDCBFTYGTFNDYPCBVRJJEZTDZNKTNSXTEEPHCXXTHEDCERCDCQAPHCBDEXPNDYHCBDEXPATDNJNFNQTVPTNXFNGCRFZCBJYZNKTFNRYAPBDRKTXFTLBEDCVAPARDYZNFLTTDCSTDTYECZRGZTXKRTVFCQEZRDGF

## Approximately Hard:

A note with these; if you find that the decoded message is correct except for one or two obvious letter swaps that occur rarely (like “KUST” instead of “JUST” but everything else is fine, for instance) it’s ok to make edits by hand after the decode is finished.

1. W CTZV VYQXDVD MCWJ IVJJTHV, TYD VYQXDVD WM BVAA, FXK WM QXYMTWYJ MCV JVQKVM XF MCV PYWZVKJV! YX KVTAAS, WM DXVJ! SXP DXY'M NVAWVZV IV? BCS BXPAD SXP YXM NVAWVZV MCTM MCWJ RVKFVQMAS QKXIPAVYM JVQKVM MVGM QXYMTWYJ MCV NV TAA, VYD TAA, HKTYDVJM JVQKVM XF TAA MCV QXJIXJ? YXB W FVVA DWJKVJRVQMVD! CTZV SXP DWJQXZVKVD SXPK XBY NVMMVK PAMWITMV MKPMC XF VZVKSMCWYH? W DWDY'M MCWYL JX. JX BCS TKV SXP HVMMWYH TAA PRRWMS TM IV? CXYVJMAS. YX XYV CTJ TYS ITYYVKJ MCVJV DTSJ. ...BCTM'J MCTM? SXP BTYM IV MX MVAA SXP MCV JVQKVM? YXM TFMVK MCWJ LWYD XF DWJKVJRVQM! HXXDYVJJ HKTQWXPJ IV. NTQL BCVY W BTJ T SXPMC W BTJ YXM JX QTAAXPJ. BCVY JXIVXYV BVAA KVJRVQMVD TYD WIRXKMTYM MXAD IV MCTM MCVS CTD JXIVMCWYH BXKMC MVAAWYH IV, W OPJM AWJMVYVD! W DWDY'M DXPNM MCVI! JX KPDV, CXYVJMAS. OPJM PYTQQVRMTNAV.
2. CIJUQCIZQFIZALSBPUFCRLIPBFIEIHYLXQIHYLOLILEIBFQXBZJSXDJXXDLSOLILELIPLNXASUBICJAXDJUMSBFYLISCFNDXDLSOLILXDLAJZXELBEALSBFQLKELNXXBRLHUYBAYLQHUJUSXDHUWZXIJUWLBICSZXLIHBFZRLNJFZLXDLSVFZXQHQUXDBAQOHXDZFNDUBUZLUZL
3. ZFNNANWJWYBZLKEHBZTNSKDDGJWYLWSBFNSSJWYFNKBGLKOCNKSJEBDWZFNGKLJKJNQFJPFJBXHBZTNRDKNZFNPDEJWYDRPDEGCNZNWJYFZZFLZTCNBBNBZFNNLKZFSLKONWBLCCKJANKBPHGBZFNGNLOBLWSRDCSBZFNRJWLCBFDKNJWLWSWDTDSUWDTDSUOWDQBQFLZBYDJWYZDFLGGNWZDLWUTDSUTNBJSNBZFNRDKCDKWKLYBDRYKDQJWYDCSJZFJWODRSNLWEDKJLKZUJNANWZFJWODRDCSSNLWEDKJLKZUZFNRLZFNKQNWNANKRDHWSJZFJWODRSNLWEDKJLKZU
4. FBYSNRBIYVNIJRJZSRSRJZNQCQNIJXCGTNEBSJNYKCUXCGTNONNIRNBUMZSIVSIJZNYBUAXCGURENBJRCBASIVJZUCGVZJZNKFCCUBIYOGUSNYSIXCGUOCINRJZNUNRBIBMZNJZBJXCGMBIJSVICUNJBASIVXCGUOUNBJZRJNBFSIVXCGUQSIYBIYBFFJZBJEBRUNBFSRFNKJONZSIYYCIJKSVZJSJSJRMCQSIVKCUXCGUGIISIVBJXCGSJRCIFXJZSRQCQNIJYCIJMBUNEZBJMCQNRBKJNUXCGUKNTNUYUNBQMBIJXCGRNNSJVNJJSIVMFCRNUPGRJRGUUNIYNUMBGRNXCGKNNFJZNKNNFSIVJBASIVCTNUSJRKSUNSJRKUNNYCQSJRKFCCYSIVCLNISJRJZNLUNBMZNUSIJZNLGFLSJBIYXCGUOFSIYYNTCJSCIJZNUNRRCQNJZSIVOUNBASIVBJJZNOUSMACKNTNUXEBFFSJRZCFYSIVBFFJZBJXCGAICERCJNFFQNYCXCGEBIIBVCEZNUNSJRMCTNUNYSIBFFJZNMCFCUNYFSVZJREZNUNJZNUGIBEBXRBUNUGIISIVJZNISVZJSQLCRRSOFNMCQNRJUGNSJRJBASIVCTNUXCGCZJZSRSRJZNVUNBJNRJRZCEENFSVZJSJGLENECIJMCQNYCEIBIYJZNRGIMBIJRJCLGRICEEBJMZSIVSJMCQNJUGNSJRJBASIVCTNUXCGCZJZSRSRJZNVUNBJNRJRZCE

## Just for fun, problems that my code actually can’t solve correctly.

Good luck?

1. KDBULISCWSCTLISJLXTNJULIUJBTALISNYULDEIDJCKTCGLISKLAUKLIMBFUGTCGLIMCGUJUGLIJUTLUCSCWLIULIJUUGLIDMWILKDEBTLLIUQLIULIMWTALIDMWILIUTLJSNTAAXSLQTKDCAXLIULISJLUUCLIDMKTCGLISKLAUKTCGLIDJCKLIJDMWILIUMCGUJCUTLIDEISKLISWILITLLIULISJLXXUTJDAGLIMWLIDMWILDELITLBDJCSCW
2. ANUYJKHNFL JLNBL, NBENJK YNK KHNKIONS: 'JNYNHYJ - SNJKONS, INHKONS, JNHDSNJ ONBRNJ!'
3. MBLBJJBV, HUBBSKDBM SBIMBJK BO BUS JBIBIB KIDBBUK BO LBJBIIB, BXOBJS, IBUBLHB KBV CBCIBJA-IBUBJ LBJLBA HUBKKBL CBSK.
4. STHTGHTCTITGHTUSKSTHHTOBTGUKTETIWSTBITSQTWKTESTKWSSTYTGUTGETHHTHHTCTETETWHSTESTUSBTUSKHTGSTHWBTGTWTWTGUTGHTIMOTYTGTHITUTKHTGHTIMTBBTWTKTFTHJFTUTIYTRTCTWGTW