The problem: We want to decipher a coded message.

"ZTRZY EKXHZR EKX RB NZ BY TBR RB NZ REKR IW REZ FAZWRIBT MEZREZY RIW TBNHZY IT REZ XITV RB WACCZY REZ WHITPW KTV KYYBMW BC BARYKPZBAW CBYRATZ BY RB RKJZ KYXW KPKITWR K WZK BC RYBANHZW KTV NO BSSBWITP ZTV REZX"

The original message was coded using a substitution cipher. Each letter in the alphabet was substituted by another letter.

We want to decipher this message.

We talked about how to evaluate a potential cipher mapping.

Letters:	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Mapping:	H	T	N	R	Y	X	B	O	L	S	Z	C	D	V	E	F	K	W	P	G	U	M	I	Q	J	A

If we assume this is the correct mapping, then the original text, must have been:

"KBDKE OQFAKD OQF DG CK GE BGD DG CK DOQD WR DOK PZKRDWGB VOKDOKE DWR BGCAKE WB D OK FWBN DG RZLLKE DOK RAWBSR QBN QEEGVR GL GZDEQSKGZR LGEDZBK GE DG DQYK QEFR QSQ WBRD Q RKQ GL DEGZCAKR QBN CH GJJGRWBS KBN DOKF"

We evaluate the probability that this was the original text. Using Bayes' rule, we stated that the probability that this is the correct mapping (given the text) is proportional to the likelihood of the text message (given the mapping).

Pr(cipher key mapping | text) is proportional to Pr(text | cipher key mapping)

We evaluate the likelihood of the text as a combination of two likelihoods:

- The likelihood of the two-letter combinations found in the 'original text'
    - How likely is the letter combo "KB" and "BD" and "DK" and "KE", etc.?
    - We used the English translation of War and Peace to estimate the probability of these two-letter combinations.
    - We return the log-likelihood which will be the sum of the log-probabilities of each two-letter combo.
- The likelihood of any real words found in the 'original text'
    - We use a lexical database, which was compiled from Wiktionary and some other unknown sources
    - "ge" appears in the lexical database with a low probability.
    - Hardly any of the other 'words' in the 'original text' appear in the lexical database. We give the non-words a very low probability.
    - We return the log-likelihood which will be the sum of the log-probabilities of each word.

Pr(text | cipher key mapping) = log-likelihood of two-letter combos + scale * log-likelihood of words

The scale is a factor to balance whether we favor common letter combinations more or less than real words. The scale factor for the words will be small because the log-likelihood of the words is very negative (because there are on the order of 80,000 words) while the log-likelihood of even the very uncommon letter combos is not too small considering there are only a total of $27^2 = 729$ possible combos.

There are 26! (approx. 4.033×10^{26}) possible cipher key mappings.

It would be very inefficient to simply randomly try a mapping and then evaluate its likelihood.

The probability if "hitting" a good mapping by random chance is very low. I've produced three random mappings to see if any of them are good.

Coded text: "ZTRZY EKXHZR EKX RB NZ BY TBR RB NZ REKR IW REZ FAZWRIBT MEZREZY RIW TBNHZY IT REZ XITV RB WACCZY REZ WHITPW KTV KYYBMW BC BARYKPZBAW CBYRATZ BY RB RKJZ KYXW KPKITWR K WZK BC RYBANHZW KTV NO BSSBWITP ZTV REZX"

Letters:	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Mapping:	H	T	N	R	Y	X	B	O	L	S	Z	C	D	V	E	F	K	W	P	G	U	M	I	Q	J	A

If this was the mapping, then the original text was: "KBDKE OQFAKD OQF DG CK GE BGD DG CK DOQD WR DOK PZKRDWGB VO KDOKE DWR BGCAKE WB DOK FWBN DG RZLLKE DOK RAWBSR QBN QEEGVR GL GZDEQSKGZR LGEDZBK GE DG DQYK QEFR QSQWBRD Q RKQ GL DEGZCAKR QBN CH GJJGRWBS KBN DOKF"

Letters:	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Mapping:	U	X	J	D	E	W	Z	K	G	P	T	M	N	R	F	I	C	B	S	A	H	Y	V	O	Q	L

If this was the mapping, then the original text was: "GKNGV EHBUGN EHB NR MG RV KRN NR MG NEHN PF NEG OTGFNPRK LEGNEGV NPF KRMUGV PK NEG BPKW NR FTQQGV NEG FUPKJF HKW HVVRLF RQ RTNVHJGRTF QRVNTKG RV NR NHCG HVBF HJHPKFN H FGH RQ NVRTMUGF HKW MX RSSRFPKJ GKW NEGB"

Letters:	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Mapping:	X	U	L	J	Q	T	P	W	S	A	Z	B	Y	F	I	R	K	O	V	H	G	N	E	D	C	M

If this was the mapping, then the original text was: "KFPKM WQATKP WQA PL VK LM FLP PL VK PWQP OH PWK NJKHPOLF ZW KPWKM POH FLVTKM OF PWK AOFS PL HJYYKM PWK HTOFGH QFS QMMLZH LY LJPMQGKLJH YLMPJFK LM PL PQDK QMAH QGQOFHP Q HKQ LY PMLJVTKH QFS VR LIILHOFG KFS PWKA"

They are all junk! How are we to find a good mapping?

Use the Metropolis Algorithm!

Instead of trying a completely new random mapping, we take our current mapping and swap two letters and see if the resulting mapping is any better.

Coded text:

"ZTRZY EKXHZR EKX RB NZ BY TBR RB NZ REKR IW REZ FAZWRIBT MEZREZY RIW TBNHZY IT REZ XITV RB WACCZY REZ WHITPW KTV KYYBMW BC BARYKPZBAW CBYRATZ BY RB RKJZ KYXW KPKITWR K WZK BC RYBANHZW KTV NO BSSBWITP ZTV REZX"

Here is our current mapping:

Letters:	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Mapping:	H	T	N	R	Y	X	**B**	O	L	S	Z	C	D	V	**E**	F	K	W	P	G	U	M	I	Q	J	A

If this was the mapping, then the original text was:
"KBDKE OQFAKD OQF DG CK GE BGD DG CK DOQD WR DOK PZKRDWGB VOKDOKE DWR BGCAKE WB DOK FWBN DG RZ LLKE DOK RAWBSR QBN QEEGVR GL GZDEQSKGZR LGEDZBK GE DG DQYK QEFR QSQWBRD Q RKQ GL DEGZCAKR QBN CH GJJGRWBS KBN DOKF"
The log-prob of this “original” text using our current mapping is: -1807.769

I propose a new mapping by randomly selecting two letters (the B and the E) and swapping them.
Proposed mapping

Letters:	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Mapping:	H	T	N	R	Y	X	**E**	O	L	S	Z	C	D	V	**B**	F	K	W	P	G	U	M	I	Q	J	A

If the proposed mapping was the correct mapping, then the original text was:
"KBDKE GQFAKD GQF DO CK OE BOD DO CK DGQD WR DGK PZKRDWOB VGKDGKE DWR BOCAKE WB DGK FWBN DO RZ LLKE DGK RAWBSR QBN QEEOVR OL OZDEQSKOZR LOEDZBK OE DO DQYK QEFR QSQWBRD Q RKQ OL DEOZCAKR QBN CH OJJORWBS KBN DGKF"
The log-prob of this “original” text using our proposed mapping is: -1788.834

The proposed mapping produces an “original” text that has higher probability than the current mapping. That is Pr(proposed)/Pr(current) > 1. According to the Metropolis algorithm, we accept the proposed mapping.

We can also accept proposed mappings that are not as good. We accept these proposed mappings with Pr(proposed)/Pr(current)

Because we are dealing with very small probabilities, we can run into machine-epsilon problems. To avoid this, we will operate using log probabilities.

U < Pr(proposed)/Pr(current)

Is equivalent to using

Log(U) < Log(Pr(proposed)) – Log(Pr(current))

The Metropolis Algorithm is great because we build off previous states. If we just did true random sampling, we have a 1 in 26! chance of ever hitting the correct mapping by random chance. Instead, we take our current mapping and propose swapping two letters. If the letter swap results in a better map, the Metropolis Algorithm will accept it. If the letter swap results in a lower probability, the Metropolis Algorithm might accept it. The Metropolis Algorithm naturally gravitates towards areas of higher probability even if you give it a poor starting location. (You saw in the homework the Metropolis algorithm still work even when you started the Markov chain at 100 for a normal distribution with mean 15.)

Technically, our deciphering is more of an optimization problem. We want to find the letter mapping that maximizes the probability of the "original" text. Meanwhile the Metropolis algorithm is generally used to produce random samples from a distribution. The Metropolis algorithm still works here because it's effectively searching the state space of 26! Possible states for states (i.e. mappings) with high probabilities. While not a strict optimization algorithm, it has the effect of finding regions of high probability.

When using the algorithm, sometimes the chain can get stuck in a "local max" area where the current mapping is still terrible but all proposed letter swaps result in even worse letter mappings. If this happens, you might need to restart the chain with a different random seed.