

Contents

1	Instructions	1
2	Overview	1
2.1	Greedy Algorithms	1
2.2	Hill-Climbing Algorithms	3
2.3	Mono-Alphabetic Substitution Ciphers & Hill-Climbing Analysis	3
3	Program Specifications	5

1 Instructions

When you are finished submit all your work through the MyClasses page for this class. Create a directory called Project2, put the program in this directory, compress the directory into a single zip file, and then submit this zip file to the Project #2 assignment. Also remember to create make files for each program and make sure the programs compile and run on the lab Linux system.

Projects are to be done strictly on your own and as with all assignments the sharing of files and code is strictly prohibited and constitutes an act of Academic Misconduct. Furthermore the use of any electronic medium, such as code repositories, forums, blogs, message boards, email, etc. is strictly prohibited and constitutes an act of Academic Misconduct.

You may use the course textbooks, class notes, and materials I have provided on the MyClasses page.

2 Overview

2.1 Greedy Algorithms

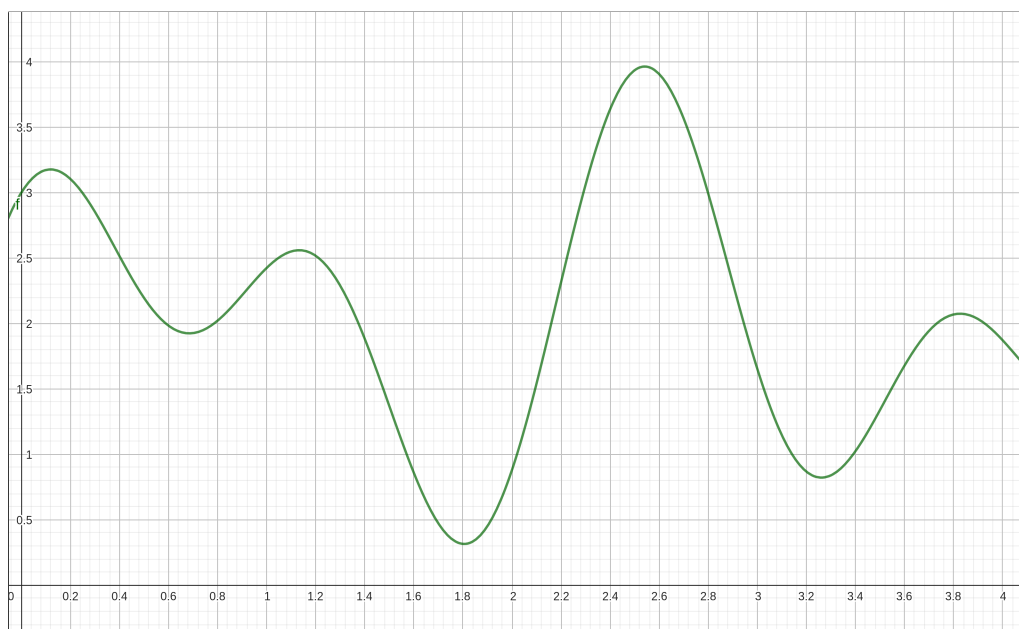
This exercise it to explore, at least a little, on what a greedy algorithm is, and to use a particular type of greedy algorithm to solve a classical cryptographic cipher, the substitution cipher. We will see other examples of this approach when investigating graph algorithms. From the algorithms text book we can define a greedy algorithm as,

Algorithms for optimization problems typically go through a sequence of steps, with a set of choices at each step. For many optimization problems, using dynamic programming to determine the best choices is overkill; simpler, more ef-

efficient algorithms will do. A **greedy algorithm** always makes the choice that looks best at the moment. That is, it makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution.

As an example, let's say we wanted to find the absolute maximum of $f(x) = \sin(3x) + \cos(5x) + 2$ on the interval $[0, 4]$. Let's also say that we forgot our derivative tests or that finding the critical numbers is too hard, who needs derivatives anyway. Here is how we might approach this problem using a greedy algorithm.

Our algorithm will be to select an x value and then look at the y values of the graph that are in the interval $[x - 0.1, x + 0.1]$, that is, within 0.1 of the x we chose. We will take the x value of the point that has the largest y value in that small interval. This is what is meant by “a locally optimal choice”. We then repeat the algorithm with that new x value, and so on until the x values settle down (or get close to) one particular number. As you can see from the graph of the function below, our absolute maximum is around $x = 2.55$.



Say we choose our initial guess at $x = 2.4$. So we look in the interval from 2.3 to 2.5 and it appears that the biggest y value is at 2.5. Now we repeat the process with $x = 2.5$ and we look at the interval from 2.4 to 2.6. It appears that the maximum y is at around $x = 2.55$, so that is our next guess. Now we look at the interval from 2.45 to 2.65. It looks like the maximum y is still at $x = 2.55$, so the x values have settled down to one number $x = 2.55$.

One thing about greedy algorithms is that although they iteratively zero in on an optimal solution, it may not be the globally optimal solution. Let's say that we did the same algorithm but started at $x = 1$. This would lead to a final x value of about $x = 1.1$, a local maximum but not a global one. And if we started at $x = 0.7$ we might end up at $x = 1.1$ or possibly $x = 0.1$. This is what the textbook meant by “...it makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution.” In all, we need to pick our starting point well. Even then we could end up with a non-global solution.

2.2 Hill-Climbing Algorithms

A special case of a greedy algorithm that can be applied to cryptography is a Hill-climbing algorithm. It works very much like the hill we climbed in the algorithm above to find the maximum (top of the hill).

‘Hill-climbing’ algorithm helps to find the correct key. It is an iterative algorithm that starts with an arbitrary solution to a problem, then attempts to find a better solution by making an incremental change to the solution. If the change produces a better solution, another incremental change is made to the new solution, and so on until no further improvements can be found.¹

Hill-climbing has many applications in many different areas of mathematics and computer science. It is one of many techniques used in deep learning and AI to make the machine make better (optimal) decisions. In this exercise we will be using this method to break a mono-alphabetic substitution cipher.

Hill-climbing does have its weaknesses.

- It does not work well on ciphers with less than 100 characters in length, or so. This is because the statistics of short messages can deviate significantly from the long-term statistics of English, which is what our n-gram data is based on.
- This algorithm fails when the true plaintext does not have statistics similar to English. For example, from Simon Singh’s “The Code Book”: “From Zanzibar to Zambia to Zaire, ozone zones make zebras run zany zigzags”.
- You could get stuck in a local maxima before reaching a desired global maximum. The use of the frequency analysis key as a starting point is done to try to avert this from happening.
- Many references to hill-climbing will tell you to start with a randomly selected initial key. This has a higher probability of getting stuck in a local maximum and hence is why we start with the frequency analysis key.

2.3 Mono-Alphabetic Substitution Ciphers & Hill-Climbing Analysis

At the end of this document there is a portion of a chapter of a set of notes I created a few years back on classical cryptography, specifically on the mono-alphabetic substitution cipher and the hill-climbing method for breaking the cipher. There is also a bit of history on the subject if you are interested. It is a fairly long read, and admittedly a bit boring in parts. It will help you understand how the cipher is encrypted and decrypted, how to use frequency

¹Hill-climbing cipher by Daleel Hagy, King’s College London, <https://www.researchgate.net/publication/340633483>

analysis to break one of these by hand, and the algorithm of the hill-climbing method is discussed.

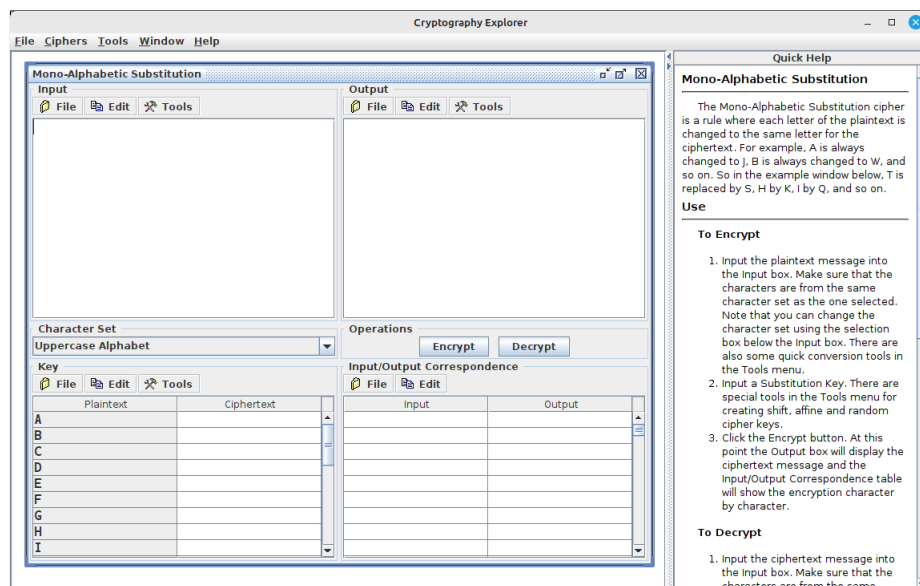
The notes describe several special cases of the mono-alphabetic substitution cipher (shift, affine, ...) but we will be attacking the most general form, the “random substitution” key. From the reading you know that there are a total of $26! = 403291461126605635584000000$ possible keys to this type of cipher which is far too many for even a modern (super) computer to check, so a brute force approach (trying all possible keys) is out of the question. In the reading there is a 4 step process for the hill-climbing analysis that you will be following. Make sure that you understand all facets of this algorithm. In particular, one character frequency analysis and the English letter relative frequencies, the fitness measure and its calculation, the transpositions done for a single pass, the pass number stopping condition.

In the handout, the fitness measure is calculated using tri-grams (three letter sequences), you will also be using quad-grams and quint-grams (4 and 5 letter sequences respectively). These will be given as files along with the exercise and handout.

There is a program that goes along with this set of notes, Cryptography Explorer, which you can get from my web site at,

<https://faculty.salisbury.edu/~despickler/distros/CryptographyExplorer.html>

There are releases for Windows, Mac, and Linux. If the Windows version gives you any trouble the jar file from the Mac and Linux download should also run on Windows machines. This package has a bunch of tools I use in teaching but there is a tool for creating mono-alphabetic substitution ciphertexts that will help you in your testing of your program. In the main menu select Ciphers > Mono-Alphabetic Substitution. The Mono-Alphabetic Substitution window will open in the program desktop and the quick help screen on the right will explain its use.



3 Program Specifications

You will be creating a program that will take an input file of English letter patterns along with a frequency count, and a file containing the ciphertext of an encrypted message. The program will apply the hill-climb algorithm to the ciphertext and output the final best key along with the decrypted message using that best key. The following are some specifics about the program run and structure.

1. From the reading you know that this method requires an enormous number of searches, of strings, on the same data set. Hence you need a fast searching routine. The reading discusses using the binary search for this but we will be using a map structure that is based on the Red-Black tree, which you created in the last homework assignment. Since the Red-Black tree is a balanced binary search tree the data should be searched fairly quickly. Also each data item will consist of two values, a key that is a string and a value that is a numeric frequency. This makes it perfect for the map structure. The n-gram data files you will be loading in are provided but you are to write the program so that the user can input any file of the correct format and use it for hill-climbing. The beginning of the data files look like the following, respectively.

AAA 31081	AAAA 6705	AAAAA 2679
AAB 34294	AAAB 1038	AAAAB 155
AAC 68958	AAAC 1592	AAAAC 240
AAD 53262	AAAD 692	AAAAD 114
AAE 3877	AAAE 176	AAAAE 22
AAF 42562	AAAF 1025	AAAAF 274

The first entry on any line is the trigram, quadgram, or quintgram. The second entry is an integer which is the frequency of that letter sequence in the data set of literature that was used in the calculation. The way these numbers were calculated was that a large number of English literature works were taken (over 4 billion characters in all), the punctuation and spaces were removed and all characters were converted to uppercase. Hence this was a long sequence of English words put together so that the ends of words were next to the beginnings of the next word. The string was then used to get counts of trigrams. For example, the text “the current report table” would be converted to “THECURRENTREPORTTABLE” and then the following trigrams would be counted THE, HEC, ECU, CUR, URR, . . . , ABL, BLE. The same is true for the quadgram and quintgram data files.

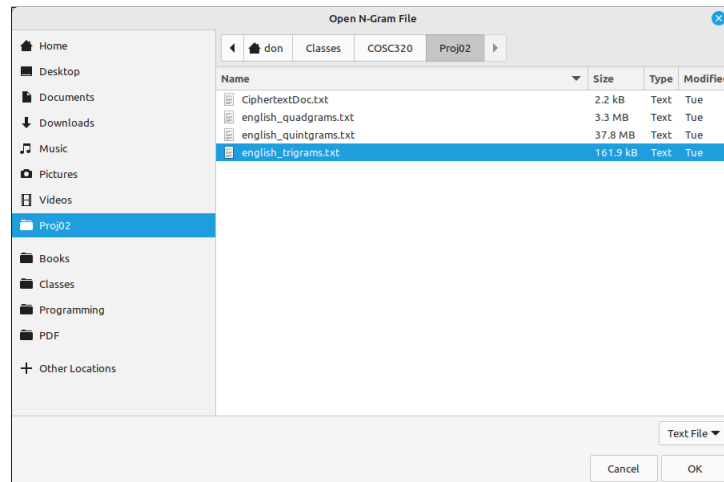
2. The ciphertext file will be a file containing the ciphertext of a single passage that has been encrypted with a mono-alphabetic substitution using a particular random key. You may assume that the text is all uppercase, no punctuation, spaces, whitespace, or numbers. Personally, I did these conversions when the file was read in just to be safe, but you do not need to do that. For example, the test file included with the project is the following. Note that there are really no line breaks in this file.

HFWRHVLSBLURPQSHRKBRHFXSQNRQSIENXSQCQUHRPQNF SBWFUD
 XLVWRILURPQSHRKBEZZQSRUMQYIQWWQUKFZZESDFVWQQDLIFKR
 EURULUDQSMSFDLQKQWRVQSFWSKHHIRQUIQHVLHRUQHHULSHRU
 MCQFWKCHIRQUIQHHEIRFWJESOFUDQDLIFKREUFUDFXXWRQDNFH
 KQSHFUDDEIKESFWXSEMSFNHELSCRMQCHKXLSXEHQRHKEQNXEJQ
 SELSHKLDQUKHJRKCKCQOUEJWQDMQHORWWHFUDIESQPFWLQHKCF
 KIEUKSRVLKQKEFIKRPQIRKRAQUHCRXMFUZZLWQNXWEBNQUKFUD
 WRZQWEUMWQFSURUMRUFQNEISFKRIHEIRQKBFUDRUKQSDQXQUD
 QUKJESWDHFWRHVLSBLURPQSHRKBIWKRPFKQHFUDHLHKFRUHFH
 LXQSRESWQFSURUMIENNLURKBJCQSQHKLQDKHJFILWKBFUDHKF
 ZZFSQPRQJQDFHWQFSUQSHKQFICQSHHICEWFSHFUDZFIRWRKFKE
 SHFUDJCQSQFIENNRKNQUKKEQYIQWWQUIQFUDXQUUQHHEKFVSE
 FDFSSFBZRDQFHFDXQSHXQIKRPQHFSQIQUKSFWKEFWWFHXQIK
 HEZLURPQSHRKBRWZQELSWQFSURUMIENNLURKBRHHKLDQUKIQUK
 QSQDHKLDQUKHWQFSUZSENXSEZQHHREUFWQDLIFKESHUHNFWWI
 WFHHSEENHQKKRUMHZFILWKBFUDXSEZQHHREUFWHKFZZHQSPQFH
 FIFDQNRIFDPRHESHFUDPRSKLFWWBQPSBHKLDQUKCFHFUEXXES
 KLURKBKELUDQSKFOQSQHQSICESQYXQSRQUKRFFWQFSURUMJRK
 CFZFILWKBNUKESKCSSELMCELSXSRPFQWBQUDEJQDHICEEWHFU
 DCEUESHIEWWQMQUFUDKCQIEWWQMQEZCQFWKCFUDCLNFUHQSPRIQ
 HJQZEHKQSFUQUPRSEUNQKJCSQRUDRPRDLFWHXSQXFSQZESIF
 SQQSFUDWRZQRUIWLDKUMKQSRSEIRFWXCBHRIFWEIILXFKREUF
 WQNEKREUFWFUDRUKQWWQIKLFWJQWWVQURUMKQQLURPQSHRKBSQI
 SLRKHQYIQXKREUFWFUDDRPQSHQZFILWKBHKFZZFUDLUDQSMSFD
 LFKQFUDMSFDLQKQHKLDQUKHZSENFISEHHNFSBWFUDFUDKCQLUR
 KQDHKFKQHFUDZSENFSELUDKCQJESWDHLXXESKRUMFWWNQNVQSH
 EZKCQLURPQSHRKBIENNLURKBFHKBQBJESOKEMQKQSKKEFICRQP
 QRUHKKRLKREUFWMEFWHFUDPRHREUVQWRQPRUMKCFKWQFSURUMF
 UDHQSPRIQFSQPRKFWIENXEUQUKHEZIRPRIWRZQHFWRVLSBLUR
 PQSHRKBFIKRPQWBIEUKSRVLKQHKKEKCQWEIFWQFHKQSUHCESQIE
 NNLURKBFUDKCQDLIFKREUFWQIEUENRIILWKLSEFWFUDHEIRFWU
 QQDHEZELSHKFKQFUDUFKREUPFWLQHKCQIESQPFWLQHEZHFWRVH
 LBLURPQSHRKBFSSQYIQWWQUIQHKLDQUKIQUKQSQDUQHHWQFSU
 RUMIENNLURKBIRPRIQUMFMQNUKFUDDRPQSHRKBFUDRUIWLHRE
 UJQVQWRQPQKCQHQPFWLQHNHVKVQWRPQDFUDQYXQSRQUIQDFHRU
 KQMSFWKEQPQSBDFBIFNXLHWRZQHEKCFKHKLQDKHNFQKQKQIEU
 UQIKREUVQKJQQUJCFKKCBWQFSUFUDCEJKCBWRPQKQMEFWHF
 UDEVTQIKRPQHEZELSHKSFQMRIFIFDQNRIZFIRWRKRQHFUDQUS
 EWWNQUKXWFUHFHJQWWFHLSZRHFIFWIENNRKNQUKHSQZWQIKELS
 ZLUDFNQKFWPFWLQHRUFDDRKREUKEKCQHQSRIRXFWPFWLQHK
 CQLURPQSHRKBNVSIQHKCSELMCRKHHCFSQDMEPQSUFUIQVEDR
 QHKCQWEUMCEUESQDKSFDRKREUEZCEUQHKBFUDNLKLFWSQMFSDK
 CFKRHFUDHCELWDVQFDQZRURUMICFSFIKQSRHKRIEZCRMCSQDL
 IFKREU

We will see later that this is the Salisbury University mission and values statement from the website that was converted to uppercase, removed non-alphabetic characters and then was encrypted with a random key mono-alphabetic substitution cipher.

- The file inputs will be done through the open dialog box from the tinyfiledialogs library that you used in the last project.

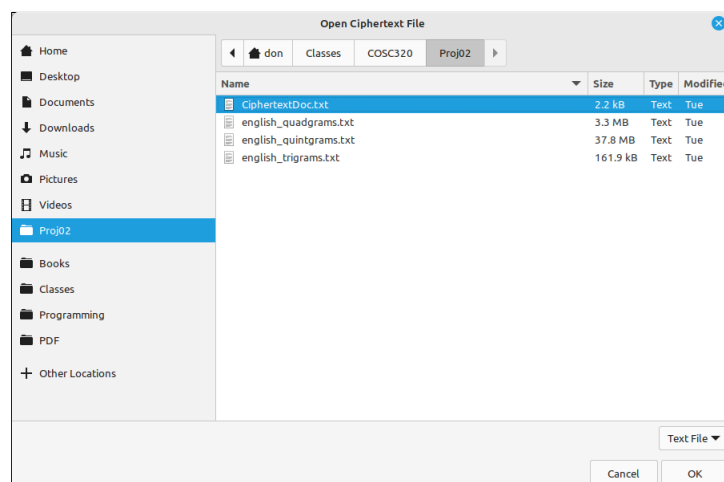
When the program is run the first thing to happen is an open dialog will appear asking for the N-Gram file.



Once that is selected and the user selects OK the console will report which file was opened and when the file read is complete there will be a message like “N-Gram file is loaded.” If the user selects Cancel the console should display something like “N-Gram file is needed to proceed, exiting.” and then exit the program. During the load of the file you will also be populating your map with the n-gram frequency pairs that you are reading from the file. So for the quintgrams this may take a few seconds.

You may assume that the n-gram file has the same structure as the ones given to you for this exercise but you must write the program so that the user can use their own n-gram file. Hence these three filenames are not to be hard-coded into the program.

Once the data is loaded from the file and put into the map, another open file dialog will appear asking for the ciphertext file.



Once that is selected and the user selects OK the console will report which file was opened and when the file read is complete there will be a message like “Ciphertext file is loaded.” If the user selects Cancel the console should display something like “Ciphertext file is needed to proceed, exiting.” and then exit the program.

The program will then start the hill-climbing algorithm, it will print to the console the initial guess of the key doing direct frequency analysis, then it will report the final key after the hill-climbing is finished and the number of iterations (passes) that needed to be done. As was in the reading this is usually fairly quick, not many iterations needed. At the end of the output will be the original ciphertext decrypted with the final key. The result should be readable English text, although without spaces and punctuation. An example of the program output for trigrams and the text ciphertext is below.

```
Opening /home/don/Classes/COSC320/Proj02/english_trigrams.txt
N-Gram file is loaded.
Opening /home/don/Classes/COSC320/Proj02/CiphertextDoc.txt
Ciphertext file is loaded.
```

```
Key from single character frequency analysis:
ABCDEFGHIJKLMNPOQRSTUVWXYZ
QGUDHTZNLVOCYMKWEISXABRPJF
```

```
Key after 4 iterations of the hill climb algorithm:
ABCDEFGHIJKLMNPOQRSTUVWXYZ
FVIDQZMCRTOWNUEXGSHKLPJYBA
```

```
Decryption using the final key:
SALISBURYUNIVERSITYISAPREMIERCOMPREHENSIVEMARYLANDPUBLICUNIV
ERSITYOFFERINGEXCELLENTAFFORDABLEEDUCATIONINUNDERGRADUATELIB
ERALARTSSCIENCESBUSINESSNURSINGHEALTHSCIENCESSOCIALWORKANDED
UCATIONANDAPPLIEDMASTERSANDDOCTORALPROGRAMSOURHIGHESTPURPOSE
ISTOEMPOWEROURSTUDENTSWITHTHEKNOWLEDGESKILLSANDCOREVALUESTHA
TCONTRIBUTETOACTIVECITIZENSHIPGAINFULEMPLOYMENTANDLIFELONGLE
ARNINGINADEMOCRATICSOCIETYANDINTERDEPENDENTWORLDSSALISBURYUNI
VERSITYCULTIVATESANDSUSTAINSASUPERIORLEARNINGCOMMUNITYWHERE
STUDENTS FACULTY AND STAFF ARE VIEWED AS LEARNERSTEACHERSSCHOLARS
AND FACILITATORS AND WHERE A COMMITMENT TO EXCELLENCE AND OPENNES
S TO A BROAD ARRAY OF IDEAS AND PERSPECTIVES ARE CENTRAL TO ALL AS
PECTS OF UNIVERSITY LIFE OUR LEARNING COMMUNITY IS STUDENT CENTERED
STUDENTS LEARN FROM PROFESSIONAL EDUCATORS IN SMALL CLASSROOM SET
TINGS FACULTY AND PROFESSIONAL STAFF SERVE AS ACADEMIC ADVISORS
AND VIRTUALLY EVERY STUDENT HAS AN OPPORTUNITY TO UNDERTAKE RESE
ARCH OR EXPERIENTIAL LEARNING WITH A FACULTY MENTOR THROUGHOUT
THE PRIVATELY ENDOWED SCHOOLS AND HONOR SCOLLEGE AND THE COLLEGE
OF HEALTH AND HUMAN SERVICES WE FOSTER AN ENVIRONMENT WHERE IND
IVIDUALS PREPARE FOR CAREER AND LIFE INCLUDING THEIR SOCIAL PHY
SICAL OCCUPATIONAL EMOTIONAL AND INTELLECTUAL WELL BEING THE UNI
VERSITY RECRUITSEXCEPTIONAL AND DIVERSE FACULTY STAFF AND UNDERGR
ADUATE AND GRADUATE STUDENTS FROM ACROSS MARYLAND AND THE UNITED
STATES A
```


NDFROMAROUNDTHEWORLDSSUPPORTINGALLMEMBERSOFTHEUNIVERSITYCOMMUNITYASTHEYWORKTOGETHERTOACHIEVEINSTITUTIONALGOALSANDVISIONBELIEVINGTHATLEARNINGANDSERVICEAREVITALCOMPONENTSOFCIVICLIFESALISBURYUNIVERSITYACTIVELYCONTRIBUTESTOTHELOCALEASTERNSHORECOMMUNITYANDTHEEDUCATIONALECONOMICCULTURALANDSOCIALNEEDSOFOURSTATEANDNATIONVALUES.THECOREVALUESOFSALISBURYUNIVERSITYAREEXCELLENCESTUDENTCENTEREDNESSLEARNINGCOMMUNITYCIVICENGAGEMENTANDDIVERSITYANDINCLUSIONWEBELIEVETHESEVALUESMUSTBELIVEDANDEXPERIENCEDASINTEGRALTOEVERYDAYCAMPUSLIFESO THATSTUDENTSMAKETHECONNECTIONBETWEENWHATTHEYLEARNANDHOWTHEY LIVETHEGOALSANDOBJECTIVESOFOURSTRATEGICACADEMICFACILITIESANDENROLLMENTPLANSASWELLASOURFISCALCOMMITMENTSREFLECTOURFUNDAMENTALVALUESINADDITIONTOTHESEPRINCIPALVALUES.THEUNIVERSITYEMBRACESTHROUGHITSSHAREDGOVERNANCEBODIESTHELONGHONOREDTRADITIONOFHONESTYANDMUTUALREGARDTHATISANDSHOULDBEADefiningCHARACTERISTICOFHIGHEREDUCATION

If you would like to save the plaintext to a file you are welcome to do so but it is not required. If you do, have it done through the save dialog box and not the command line. Also, these are all text files, the n-grams, the ciphertext, and the plaintext, so you can change the filter to display “Text Files” and use the *.txt extension.

4. You will probably find that there are several facilities that you would like the map class to have, feel free to add them to your map class. With that said, additions to the map class should be functions that are of general use for all maps of any two data types. Functionality that is specific to this application should be in the main and not the map. For example, I found it useful to have a traversal function (in order, although that did not matter) that took as a parameter a function that could update the key and value of the node. I then wrote a function in the main to do the specific alteration I wanted and simply applied the function to the map. Also, things like a decryption function and a fitness measure calculation function were all done in the main.
5. In the reading on the hill-climb I mentioned that when calculating the fitness measure there may be n-grams that are not in the list of n-grams, for example QQQQ. So when searching for n-grams you may encounter a failed search, in all likelihood you will. I suggested using a frequency for these that is around 1/100. I have used values between 1/10 and 1/100 with equal success, so I would keep this value around that range. You do not want to use 0, that is, skipping the ones that are not found. The reason for this is that since you are taking the log of the relative frequencies, the values you are adding to the fitness measure for each n-gram are relatively small negative numbers (e.g. -15.632435) so a 0 addition would actually be increasing the fitness and hence plaintexts with a lot of failed searches (i.e ones that are far from English) will have a larger fitness and the bad key is more likely to be chosen. If on the other hand we add on a large negative number, this will decrease the fitness, which is what we want, but may decrease it too far. In this case a single failed search may make it less likely for the key to be used than one that produces gibberish but has no failed searches. So we want failed searches to decrease the fitness but not overwhelmingly. For moderately sized

ciphertexts a frequency between $1/10$ and $1/100$ has been shown to work in empirical tests.

6. Also from the reading you know that the number of passes should be relatively small and that you stop when the fitness measure no longer increases. Since this is not a set number of iterations you do run the risk of an infinite loop of passes. I would put an upper bound of passes on the program at say 100. For most cases this is more than sufficient. From above you can see that the program needed only 4 passes to process a ciphertext with about 2000 characters.