A Mathematica Program for Classifying Geodesics With k Self-Intersections on the Once-Punctured Torus

Susan Garner Garille
Lafayette College
gs60@lafibm.lafayette.edu

August 9, 1995

Abstract

This program develops a list of possible geodesics on the once-punctured torus with k self-intersections. It generates a list of words, each having $k+1$ initial elements, as all combinations of six simple loops. It then continually eliminates impossible combinations and reduces words.

Keywords: geodesics, intersections, punctured torus, *Mathematica*

1 Introduction

At the end of June, I was presented with the possibility of working on the problem of classifying the geodesics that self-intersect non trivially three times on the once-punctured torus. Including myself, there were four people working on this project. Using results obtained by previous investigators of the subject, we each worked diligently on developing a method for determining the answer to the question. Once some possible methods were established, the work that then needed to be done was tedious and time consuming. I considered it the kind of work where it was very easy to make an error. As I wrote down strings of a's, b's, a^{-1}'s, and b^{-1}'s I couldn't help but continually think that

there must be a way to program a computer to do all of this. Although developing the names of the geodesics on your own can be a rewarding research experience, the results are what people want. The results are what people will look at so that new information can be developed. Needless to say, I decided that it could be done especially with the new theorem we developed in the first weeks looking at the three intersector case.

This report describes an algorithm that generates a list of words that describe geodesics on the once-punctured torus with k self-intersections. The algorithm uses a lemma and a theorem from (Dziadosz, et al.) and [Butler, et al.), respectively. The main idea behind the algorithm is to develop a "big" list of all possible words then eliminating impossible combinations and reducing and eliminating duplicate words. I wrote the program in Mathematica and from here on (except for the theorem statements) I will refer to a^{-1} as A and b^{-1} as B, since this is the notation I used in the program. The Appendix contains a copy of the program for the two-intersector case which takes less than two minutes to run.

2 Background Information

The following results are necessary for understanding and verifying that my method for developing the initial list of all possible "words" is allowed. A word is a sequence of a's, b's, A's, and B's that describes a composition of curves.

Lemma 1 (Dziadosz, Insel, and Wiles) *Up to free homotopy, any loop on T with k transverse self-intersection points can be formed as the composition of $k+1$ simple loops, which intersect at one point [called the "basepoint"].*

Theorem 1 (Butler, Carton, and Kraft) *Given a generating pair a, b for $\Pi_1(T)$, any other generator representable by a simple loop is equivalent to one of the following six, up to orientation and reflection:*

(i) a

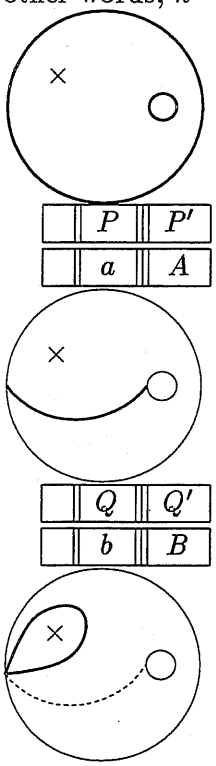
(ii) b

(iii) bab^{-1}

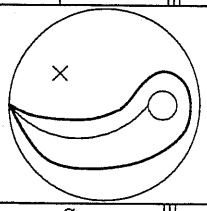
(iv) aba^{-1}

(v) ab

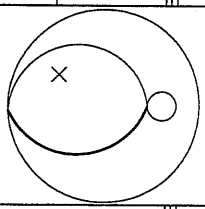
With this information, I know that a geodesic with k self-intersections will be a composition of $k+1$ of the loops below (represented by bold lines). Each of these loops can be described in terms of a and b according to its orientation (clockwise (CW), counterclockwise (CCW), left to right ($l \rightarrow r$), or right to left ($r \rightarrow l$)) and the orientation of a (CW or CCW) and b ($l \rightarrow r$) or ($r \rightarrow l$). Below each picture are all possible namings of each curve. The letter at the top of each box denotes the name of the set of all elements in that box. Each non-reduced word has four elements, each element being the descriptor of one of the simple loops. In the program, n is the number of elements in the word. For example, $n = 3$ for the two intersector cases and $n = 4$ for the three intersector cases. In other words, $n = k + 1$.



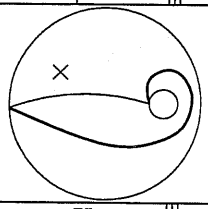
	R		R'	
	a		A	
b	CW	CCW	CW	CCW
$l \to r$	$abAB$	$baBA$	$AbaB$	$bABa$
$r \to l$	$aBAb$	$BabA$	$ABab$	$BAba$



	S		S'	
	a		A	
b	CW	CCW	CW	CCW
$l \to r$	baB	bAB	bAB	baB
$r \to l$	Bab	BAb	BAb	Bab



	T		T'	
	a		A	
b	$l \to r$	$r \to l$	$l \to r$	$r \to l$
$l \to r$	Aba	ABa	abA	aBA
$r \to l$	ABa	Aba	aBA	abA



	U		U'	
	a		A	
b	$l \to r$	$r \to l$	$l \to r$	$r \to l$
$l \to r$	Ab	aB	ab	AB
$r \to l$	AB	ab	aB	Ab

When these loops are combined, some rules apply. One rule is that the "default" descriptors are a and b. For example, if a composition of curves does not contain loop a or loop b, then the other curves should be described by a and b, not A and B.

3 Developing the "Big" List

The first step of the program is to generate a list of all possible combinations of the six simple loops for words of n elements. This "big" list is made up of three smaller lists. The first list is that of all curves that do not have any generators (i.e. only have loops around the puncture). Since these types of compositions do not contain loops a, b, A or B, then the only possible elements in the words are the names of the loops that are formed using a and b as descriptors: $abAB$, $baBA$. *Mathematica* has a command, **Outer[List,** $list_1$, $list_2$, **...**], that gives you all possible combinations of elements. I used this to develop this list and the other two that make up the "big" list.

The second two lists are those words that contain at least one generator. Without loss of generality, assume that one of the generators is A or a. Since cyclic permutations of a word are equivalent, the generator (A or a) can always be the first element (denoted **w1** or **v1** in the program) of these words. A only needs to be considered if the picture of the composition of the curves is asymmetric. This only occurs when R or R' are elements of the word. So, generate a list where the first element (**v1**) is A and the rest of the elements (**v2**,**v3** ... ,**vn**) are anything from P', Q, R', S', T', or U'. From this list, eliminate all words that do not contain anything from R' (since those are symmetric and we can re-orient the pictures and force the first generator to be a).

Next, generate a list of words where the first element (**w1**) is a and the rest of the elements (**w2**, **w3**, ... **wn**) are anything from P, Q, R, S, T, or U.

Once these three lists are generated and joined, the entire list (denoted **List1** in the program) of possible words for curves with k self-intersections has been formed. It is a list of ALL possible words for geodesics with k self-intersections.

4 Eliminating Impossible Combinations

Some combinations of elements are not possible and need to be eliminated from the big list. The next steps of the program find and delete these cases from **List1**. The *Mathematica* commands used to find and eliminate these words were **Select**[*list*, *criteria*], which finds all elements in *list* that fit the indicated *criteria*], and **Complement**[*eall*, *e*], which eliminates elements in *e* from *eall*.

Words that do not have anything from Q cannot have anything from T or U. These words can be eliminated from the **List1** to form **List2**.

If a word has b as an element, then is cannot also have $aBAb$, $BabA$, Bab, or BAb as elements since these loops are described using B as a generator. b takes priority over B as a generator to describe loops. Eliminating these words from **List2** forms **List3**.

If word has B as an element, but not b, then the loops on this word are described by B. So, these words cannot have $abAB$, $baBA$, baB, bAB, Ab, or aB as elements since these are formed using b, not B as a generator. **List4** is formed by deleting these cases from **List3**.

If a word has neither B or b, then since b is the default descriptor, these words cannot have, as elements, $aBAb$, $BabA$, Bab, BAb, AB, or ab. Removing these from **List4** yields **List5**.

Words cannot have $abAB$, $baBA$, baB, Ab, or aB as an element and also have $BabA$, $BabA$, Bab, BAb, AB, or ab as an element since the elements in the first group are described using b as a generator and the second group of elements are all described using B as a generator. **List6** is formed by eliminating these cases from **List5**.

5 Comparing Words

The original list of words, **List1** is drastically reduced now that all of the impossible combinations have been eliminated. The next step is to compare cyclic permutations of each word with other words in **List6** and eliminate duplicates. To do this, I defined a **Module** called **checklist**. **checklist** used **RotateLeft**[*list*], which cycles the elements in *list* one position to the left. It also used a while loop that used the commands **!MemberQ**[*list*, *form*], which checks if *form*] is not a member of *list*. After performing **checklist** on **List6** for the

length of **List6**, **List9** is formed.

6 Simple Reductions

At this point, the n elements of each word are joined, using **Map[StringJoin** [*list*]], so that they form one string. This new list is called **List10**. The strings in this list can be reduced and made shorter. Anytime a is followed by A or vice versa, or b is followed by B or vice versa, they cancel and the string is shortened by two characters. These simplifications can be performed repeatedly to a word until no further reduction can be done. To do this, I defined another **Module** called **reducing**. **reducing** uses do loops (each of which runs 4n times since that is the maximum possible number of reductions per word) and if statements to apply **StringPosition**["*string*", "*substring*"], which returns the position of "*substring*" in "*string*", and **StringDrop**["*string*", m, n], which deletes positions m through n in **"string"**, to each of the elements in **List10**.

7 Rotate and Compare

List10 has been reduced to **List11**. After the last step, some words have been reduced to the identity while others have remained the same. Many words have been made shorter and may be equivalent to other words in **List11**. Once again comparisons must be made between the words. Each word is cyclically permuted and compared to the other words in the list and is discarded if it is a duplicate. This is done using the same procedure described in Section 5. **List13** is the new list created after performing the eliminations.

8 Dividing the List

Before moving on to the more complicated reductions, a division of the most recently made list would be helpful. The list is divided into sublists; each sublist contains words of equal length. Since we are interested in curves which intersect at least twice, words of length 1 and 2 can be eliminated since these are either a generator or a square.

A **Module** that uses a simple if statement is defined to generate each sublist.

9 Substitutions

On each of the remaining sublists, substitutions and comparisons must be performed to again eliminate duplicates. The following substitutions, using **StringReplace**["*string*", "*substitution*$_1$", "*substitution*$_2$", ...], can be made for a, b, A, and B.

Substitutions							
a	A	A	a	b	B	b	B
b	B	b	B	a	a	A	A
A	a	a	A	B	b	B	b
B	b	B	b	A	A	a	a

For example $aaBab = bbAba = BBaBA$, etc.

These substitutions need to be made on each word of each sublist and compared to the other words in the sublist so that duplicate words can be eliminated.

10 More Reductions

It is possible to use more complicated substitutions to determine more duplications in **List27**. If I had more time this summer, I would have continued the process of reducing the list. The algorithm for the next procedure should be simple since it incorporates procedures already executed. It should be possible that the final list for the two-intersector case, after all reductions, contain only 15 words: 8 two-intersector words, 4 one-intersector words, and 3 simple loops.

11 The Final Step

Once the program has been run, the user is left with a list of words – not all of which describe curves with k self-intersections. The names of all of the curves with k self intersections are in this list, but extraneous curves are included. These other curves have greater than or less than k self-intersections. In order to eliminate these unwanted words, a *Mathematica* program used during the Summer of 1994 by the REU

students must be implemented. The program allows the user to input a word and see the described curve represented on the upper half plane. A quick analysis of these pictures allows the user to determine how many intersections a given curve has.

12 Conclusion

My goal was to write a program that would make it easier for future interested parties to classify the geodesics that self-intersect k times on the once punctured torus. Hopefully this program will be implemented or refined or used as a "check" to those researchers who like to stick with the pencil and paper method for determining these names.

13 References

Mande Butler, Jeanne Carton, and Emil Kraft, *Geodesics With Three Intersections on the Punctured Torus*, NSF REU Program at Oregon State University (Summer 1995).

Susan Dziadosz, Thomas Insel, and Peter Wiles, *Geodesics With Two Self-Intersections on the Punctured Torus*, NSF REU Program at Oregon State University (Summer 1994).

John W. Gray, *Mastering Mathematica: Programming Methods and Applications*, AP Professional, New York, 1994.

Stephen Wolfram, *Mathematica: A System for Doing Mathematics by Computer*, Second Edition, Addison-Wesley Publishing Company, Inc., New York, 1991.

```
n=3;
w1 = {"a"};
w2 = {"a","A",
      "b","B",
      "abAB","aBAb","baBA","BabA",
      "baB","bAB","Bab","BAb",
      "Aba","ABa",
      "Ab","AB","aB","ab"};
w3 = {"a","A",
      "b","B",
      "abAB","aBAb","baBA","BabA",
      "baB","bAB","Bab","BAb",
      "Aba","ABa",
      "Ab","AB","aB","ab"};
v1 = {"A"};
v2 = {"A",
       "b","B",
       "AbaB","ABab","bABa","BAba",
       "bAB","BAb","baB","Bab",
       "abA","aBA",
       "ab","aB","AB","Ab"};
v3 = {"A",
       "b","B",
       "AbaB","ABab","bABa","BAba",
       "bAB","BAb","baB","Bab",
       "abA","aBA",
       "ab","aB","AB","Ab"};
u1 = {"abAB","baBA"};
u2 = {"abAB","baBA"};
u3 = {"abAB","baBA"};

aGenerators = Outer[List, w1,w2,w3];
AGenerators = Outer[List, v1,v2,v3];
NoGenerators = Outer[List, u1,u2,u3];
ASymmetries = Select[AGenerators, (#[[n-1]] == "AbaB"
                                || #[[n-1]] == "ABab"
                                || #[[n-1]] == "bABa"
                                || #[[n-1]] == "BAba"
                                || #[[n]] == "AbaB"
                                || #[[n]] == "ABab"
                                || #[[n]] == "bABa"
                                || #[[n]] == "BAba" &)];
Symmetries = Complement[AGenerators, ASymmetries];
BigList = Join[aGenerators, Symmetries, NoGenerators];
List1 = Flatten[BigList,2];
```



```
Print["Length List1 = ", Length[List1]];

WordsWithTU = Select[List1, (#[[n-1]] == "Aba"
                          || #[[n-1]] == "ABa"
                          || #[[n-1]] == "Ab"
                          || #[[n-1]] == "AB"
                          || #[[n-1]] == "aB"
                          || #[[n-1]] == "ab"
                          || #[[n]] == "Aba"
                          || #[[n]] == "ABa"
                          || #[[n]] == "Ab"
                          || #[[n]] == "AB"
                          || #[[n]] == "aB"
                          || #[[n]] == "ab" &)];
WordsWithTUQ = Select[WordsWithTU, (#[[n-1]] == "b"
                                 || #[[n-1]] == "B"
                                 || #[[n]] == "b"
                                 || #[[n]] == "B" &)];
WordsWithTUnoQ = Complement[WordsWithTU,WordsWithTUQ];
List2 = Complement[List1,WordsWithTUnoQ];


Print["Length List2 = ", Length[List2]];
WordsWithb = Select[List2, (#[[n-1]] == "b"
                         || #[[n]] == "b" &)];
NonWordsWithb = Select[WordsWithb, (#[[n-1]] == "aBAb"
                                 || #[[n-1]] == "BabA"
                                 || #[[n-1]] == "Bab"
                                 || #[[n-1]] == "BAb"
                                 || #[[n-1]] == "AB"
                                 || #[[n-1]] == "ab"
                                 || #[[n]] == "aBAb"
                                 || #[[n]] == "BabA"
                                 || #[[n]] == "Bab"
                                 || #[[n]] == "BAb"
                                 || #[[n]] == "AB"
                                 || #[[n]] == "ab" &)];
List3 = Complement[List2, NonWordsWithb];

Print["Length List3 = ", Length[List3]];

WordsWithB = Select[List3, (#[[n-1]] == "B"
                         || #[[n]] == "B" &)];
WordsWithBb = Select[WordsWithB, (#[[n-1]] == "b"
                               || #[[n]] == "b" &)];
```



```
WordsWithOnlyB = Complement[WordsWithB, WordsWithBb];

NonWordsWithB = Select[WordsWithOnlyB, (#[[n-1]] == "abAB"
                                     || #[[n-1]] == "baBA"
                                     || #[[n-1]] == "baB"
                                     || #[[n-1]] == "bAB"
                                     || #[[n-1]] == "Ab"
                                     || #[[n-1]] == "aB"
                                     || #[[n]] == "abAB"
                                     || #[[n]] == "baBA"
                                     || #[[n]] == "baB"
                                     || #[[n]] == "bAB"
                                     || #[[n]] == "Ab"
                                     || #[[n]] == "aB" &)];
List4 = Complement[List3, NonWordsWithB];

Print["Length List4 = ", Length[List4]];

WordsWithoutBandb = Complement[List4,WordsWithBb];
NonWordsWithoutBandb = Select[WordsWithoutBandb,
                        (#[[n-2]] == "aBAb"
                      || #[[n-2]] == "BabA"
                      || #[[n-2]] == "Bab"
                      || #[[n-2]] == "BAb"
                      || #[[n-2]] == "AB"
                      || #[[n-2]] == "ab"
                      || #[[n-1]] == "aBAb"
                      || #[[n-1]] == "BabA"
                      || #[[n-1]] == "BAb"
                      || #[[n-1]] == "AB"
                      || #[[n-1]] == "ab"
                      || #[[n]] == "aBAb"
                      || #[[n]] == "BabA"
                      || #[[n]] == "Bab"
                      || #[[n]] == "AB"
                      || #[[n]] == "ab" &)];
List5 = Complement[List4, NonWordsWithoutBandb];

Print["Length List5 = ", Length[List5]];

WordsWithRSUb = Select[List5, (#[[n-2]] == "abAB"
                            || #[[n-2]] == "baBA"
                            || #[[n-2]] == "baB"
                            || #[[n-2]] == "bAB"
                            || #[[n-2]] == "Ab"
```



```
                            || #[[n-2]] == "aB"

                            || #[[n-1]] == "abAB"
                            || #[[n-1]] == "baBA"
                            || #[[n-1]] == "baB"
                            || #[[n-1]] == "bAB"
                            || #[[n-1]] == "Ab"
                            || #[[n-1]] == "aB"
                            || #[[n]] == "abAB"
                            || #[[n]] == "baBA"
                            || #[[n]] == "baB"
                            || #[[n]] == "bAB"
                            || #[[n]] == "Ab"
                            || #[[n]] == "aB" &)];
NonWordsWithRSUb = Select[WordsWithRSUb, (#[[n-2]] == "aBAb"
                                       || #[[n-2]] == "BabA"
                                       || #[[n-2]] == "Bab"
                                       || #[[n-2]] == "BAb"
                                       || #[[n-2]] == "AB"
                                       || #[[n-2]] == "ab"
                                       || #[[n-1]] == "aBAb"
                                       || #[[n-1]] == "BabA"
                                       || #[[n-1]] == "Bab"
                                       || #[[n-1]] == "BAb"
                                       || #[[n-1]] == "AB"
                                       || #[[n-1]] == "ab"
                                       || #[[n]] == "aBAb"
                                       || #[[n]] == "BabA"
                                       || #[[n]] == "Bab"
                                       || #[[n]] == "BAb"
                                       || #[[n]] == "AB"
                                       || #[[n]] == "ab" &)];
List6 = Complement[List5,NonWordsWithRSUb];

Print["Length List6 = ", Length[List6]];

List9 = {List6[[1]]};

checklist[m_] :=
   Module[{Word1 = List6[[m]]},
      j = 1;
      X = RotateLeft[Word1];
      While[!MemberQ[List9,X] && j < n,
         X = RotateLeft[X]; j = j + 1];
      If[X == List6[[m]], AppendTo[List9,X]];
```

```
Do[checklist[m], {m,2,Length[List6]}];

Print["Length List9 = ", Length[List9]];

List10 = Map[StringJoin,List9];

List11 = List[ ];

reducing[i_] :=
    Module[{Word1=List10[[i]]},
      Do[PosaA = StringPosition[Word1, "aA"];
         If[PosaA == List[ ],
            Word1,
            Word1 = StringDrop[Word1,PosaA[[1]]]];
         PosbB = StringPosition[Word1, "bB"];
         If[PosbB == List[ ],
            Word1,
            Word1 = StringDrop[Word1, PosbB[[1]]]];
         PosAa = StringPosition[Word1, "Aa"];
         If[PosAa == List[ ],
            Word1,
            Word1 = StringDrop[Word1, PosAa[[1]]]];
         PosBb = StringPosition[Word1,"Bb"];
         If[PosBb == List[ ],
            Word1,
            Word1 = StringDrop[Word1,PosBb[[1]]]
            ,{j,6}];
       AppendTo[List11,Word1]];

For[i=1, i<=Length[List10], i++, reducing[i]];

Print["Length List11 = ", Length[Union[List11]]];

For[m=1, m<13, m++,
    List12 = Union[List11];
    Print[Length[List12]];
    List13 = Map[Characters,List12];
    List14 = Map[RotateLeft,List13];
    List15 = Map[StringJoin,List14];
    List10 = List15;
    List11 = List[ ];
    For[i=1, i<=Length[List12], i++, reducing[i]]];

List12 = Map[Characters,List11];
```



```
List13 = {List12[[2]]};

rotatencompare[k_] :=
   Module[{Word1 = List12[[k]]},
       j = 1;
       Y = RotateLeft[Word1];
       While[!MemberQ[List13,Y] && j < Length[Word1],
          Y = RotateLeft[Y]; j = j + 1];
       If[Y == List12[[k]], AppendTo[List13,Y]]];

For[k=3, k<=Length[List12], k++, rotatencompare[k]];

Print["Length List13 = ", Length[List13]];

List14 = Map[StringJoin,List13];

Print["Length List14 = ", Length[List14]];

Singles = List[ ];

FindSingles[i_] :=
   Module[{Word = List14[[i]]},
      If[StringLength[Word] == 1,
         AppendTo[Singles, Word]]];

For[i=1, i<= Length[List14], i++, FindSingles[i]];

List15 = Complement[List14,Singles];

Doubles = List[ ];

FindDoubles[i_] :=
   Module[{Word = List15[[i]]},
      If[StringLength[Word] == 2,
         AppendTo[Doubles, Word]]];

For[i=1, i<= Length[List15], i++, FindDoubles[i]];

List16 = Complement[List15, Doubles];

Print["List16 = ", Length[List16]];

Triples = List[ ];

FindTriples[i_] :=
   Module[{Word = List16[[i]]},
```



```
      If[StringLength[Word] == 3,
         AppendTo[Triples, Word]]];

For[i=1, i<= Length[List16], i++, FindTriples[i]];

List17 = Complement[List16, Triples];

trisubstitutions[i_] :=
   Module[{Sub = Triples[[i]]},
      ListX = List[ ];
      Sub1a = StringReplace[Sub,   {"a" -> "c",
                                    "b" -> "d",
                                    "A" -> "a",
                                    "B" -> "b"}];
      Sub1  = StringReplace[Sub1a, {"c" -> "A",
                                    "d" -> "B"}];
      Sub2a = StringReplace[Sub,   {"a" -> "c",
                                    "A" -> "a"}];
      Sub2  = StringReplace[Sub2a, "c" -> "A"];
      Sub3a = StringReplace[Sub,   {"b" -> "d",
                                    "B" -> "b"}];
      Sub3  = StringReplace[Sub3a, "d" -> "B"];
      Sub4a = StringReplace[Sub,   {"a" -> "c",
                                    "b" -> "a",
                                    "A" -> "C",
                                    "B" -> "A"}];
      Sub4  = StringReplace[Sub4a, {"c" -> "b",
                                    "C" -> "B"}];
      Sub5a = StringReplace[Sub,   {"a" -> "c",
                                    "b" -> "d",
                                    "A" -> "C",
                                    "B" -> "D"}];
      Sub5  = StringReplace[Sub5a, {"c" -> "B",
                                    "d" -> "a",
                                    "C" -> "b",
                                    "D" -> "A"}];
      Sub6a = StringReplace[Sub,   {"a" -> "c",
                                    "b" -> "d",
                                    "A" -> "C",
                                    "B" -> "D"}];
      Sub6  = StringReplace[Sub6a, {"c" -> "b",
                                    "d" -> "A",
                                    "C" -> "B",
                                    "D" -> "a"}];
      Sub7a = StringReplace[Sub,   {"a" -> "c",
                                    "b" -> "d",
```



```
                                    "A" -> "b",
                                    "B" -> "a"}];
      Sub7  = StringReplace[Sub7a, {"c" -> "B",
                                    "d" -> "A"}];
      ChSub1 = Characters[Sub1];
      ChSub2 = Characters[Sub2];
      ChSub3 = Characters[Sub3];
      ChSub4 = Characters[Sub4];
      ChSub5 = Characters[Sub5];
      ChSub6 = Characters[Sub6];
      ChSub7 = Characters[Sub7];

      For[k=1, k<=Length[ChSub1], k++,
          Z = RotateLeft[ChSub1,k];
          AppendTo[ListX, Z]];

      For[k=1, k<=Length[ChSub2], k++,
          Z = RotateLeft[ChSub2,k];
          AppendTo[ListX, Z]];

      For[k=1, k<=Length[ChSub3], k++,
          Z = RotateLeft[ChSub3,k];
          AppendTo[ListX, Z]];

      For[k=1, k<=Length[ChSub4], k++,
          Z = RotateLeft[ChSub4,k];
          AppendTo[ListX, Z]];

      For[k=1, k<=Length[ChSub5], k++,
          Z = RotateLeft[ChSub5,k];
          AppendTo[ListX, Z]];

      For[k=1, k<=Length[ChSub6], k++,
          Z = RotateLeft[ChSub6,k];
          AppendTo[ListX, Z]];

      For[k=1, k<=Length[ChSub7], k++,
          Z = RotateLeft[ChSub7,k];
          AppendTo[ListX, Z]];

      StringListX = Map[StringJoin,ListX];

      Discards = List[ ];

      For[j=i+1, j<=Length[Triples], j++,
```

```
            If[MemberQ[StringListX, Triples[[j]]],
               AppendTo[Discards, Triples[[j]]]];

         Triples = Complement[Triples,Discards]];

   For[i=1, i<=Length[Triples], i++, trisubstitutions[i]];

   Quadruples = List[ ];

   FindQuadruples[i_] :=
      Module[{Word = List17[[i]]},
         If[StringLength[Word] == 4,
            AppendTo[Quadruples, Word]]];

   For[i=1, i<= Length[List17], i++, FindQuadruples[i]];

   List18 = Complement[List17, Quadruples];

   quadsubstitutions[i_] :=
      Module[{Sub = Quadruples[[i]]},
         ListX = List[ ];
         Sub1a = StringReplace[Sub,   {"a" -> "c",
                                       "b" -> "d",
                                       "A" -> "a",
                                       "B" -> "b"}];
         Sub1  = StringReplace[Sub1a, {"c" -> "A",
                                       "d" -> "B"}];
         Sub2a = StringReplace[Sub,   {"a" -> "c",
                                       "A" -> "a"}];
         Sub2  = StringReplace[Sub2a,  "c" -> "A"];
         Sub3a = StringReplace[Sub,   {"b" -> "d",
                                       "B" -> "b"}];
         Sub3  = StringReplace[Sub3a,  "d" -> "B"];
         Sub4a = StringReplace[Sub,   {"a" -> "c",
                                       "b" -> "a",
                                       "A" -> "C",
                                       "B" -> "A"}];
         Sub4  = StringReplace[Sub4a, {"c" -> "b",
                                       "C" -> "B"}];
         Sub5a = StringReplace[Sub,   {"a" -> "c",
                                       "b" -> "d",
                                       "A" -> "C",
                                       "B" -> "D"}];
         Sub5  = StringReplace[Sub5a, {"c" -> "B",
                                       "d" -> "a",
                                       "C" -> "b",
```



```
                                       "D" -> "A"}];
         Sub6a = StringReplace[Sub,   {"a" -> "c",
                                       "b" -> "d",
                                       "A" -> "C",
                                       "B" -> "D"}];
         Sub6  = StringReplace[Sub6a, {"c" -> "b",
                                       "d" -> "A",
                                       "C" -> "B",
                                       "D" -> "a"}];
         Sub7a = StringReplace[Sub,   {"a" -> "c",
                                       "b" -> "d",
                                       "A" -> "b",
                                       "B" -> "a"}];
         Sub7  = StringReplace[Sub7a, {"c" -> "B",
                                       "d" -> "A"}];
         ChSub1 = Characters[Sub1];
         ChSub2 = Characters[Sub2];
         ChSub3 = Characters[Sub3];
         ChSub4 = Characters[Sub4];
         ChSub5 = Characters[Sub5];
         ChSub6 = Characters[Sub6];
         ChSub7 = Characters[Sub7];

         For[k=1, k<=Length[ChSub1], k++,
             Z = RotateLeft[ChSub1,k];
             AppendTo[ListX, Z]];

         For[k=1, k<=Length[ChSub2], k++,
             Z = RotateLeft[ChSub2,k];
             AppendTo[ListX, Z]];

         For[k=1, k<=Length[ChSub3], k++,
             Z = RotateLeft[ChSub3,k];
             AppendTo[ListX, Z]];

         For[k=1, k<=Length[ChSub4], k++,
             Z = RotateLeft[ChSub4,k];
             AppendTo[ListX, Z]];

         For[k=1, k<=Length[ChSub5], k++,
             Z = RotateLeft[ChSub5,k];
             AppendTo[ListX, Z]];

         For[k=1, k<=Length[ChSub6], k++,
             Z = RotateLeft[ChSub6,k];
             AppendTo[ListX, Z]];
```



```
         For[k=1, k<=Length[ChSub7], k++,
             Z = RotateLeft[ChSub7,k];
             AppendTo[ListX, Z]];

         StringListX = Map[StringJoin,ListX];

         Discards = List[ ];

         For[j=i+1, j<=Length[Quadruples], j++,

             If[MemberQ[StringListX, Quadruples[[j]]],
                AppendTo[Discards, Quadruples[[j]]]];

         Quadruples = Complement[Quadruples,Discards]];

   For[i=1, i<=Length[Quadruples], i++, quadsubstitutions[i]];

   Quintuples = List[ ]

   FindQuintuples[i_] :=
      Module[{Word = List18[[i]]},
         If[StringLength[Word] == 5,
            AppendTo[Quintuples, Word]]];

   For[i=1, i<= Length[List18], i++, FindQuintuples[i]];

   List19 = Complement[List18, Quintuples];

   quintsubstitutions[i_] :=
      Module[{Sub = Quintuples[[i]]},
         ListX = List[ ];
         Sub1a = StringReplace[Sub,   {"a" -> "c",
                                       "b" -> "d",
                                       "A" -> "a",
                                       "B" -> "b"}];
         Sub1  = StringReplace[Sub1a, {"c" -> "A",
                                       "d" -> "B"}];
         Sub2a = StringReplace[Sub,   {"a" -> "c",
                                       "A" -> "a"}];
         Sub2  = StringReplace[Sub2a,  "c" -> "A"];
         Sub3a = StringReplace[Sub,   {"b" -> "d",
                                       "B" -> "b"}];
         Sub3  = StringReplace[Sub3a,  "d" -> "B"];
         Sub4a = StringReplace[Sub,   {"a" -> "c",
                                       "b" -> "a",
```



```
                                       "A" -> "C",
                                       "B" -> "A"}];
         Sub4  = StringReplace[Sub4a, {"c" -> "b",
                                       "C" -> "B"}];
         Sub5a = StringReplace[Sub,   {"a" -> "c",
                                       "b" -> "d",
                                       "A" -> "C",
                                       "B" -> "D"}];
         Sub5  = StringReplace[Sub5a, {"c" -> "B",
                                       "d" -> "a",
                                       "C" -> "b",
                                       "D" -> "A"}];
         Sub6a = StringReplace[Sub,   {"a" -> "c",
                                       "b" -> "d",
                                       "A" -> "C",
                                       "B" -> "D"}];
         Sub6  = StringReplace[Sub6a, {"c" -> "b",
                                       "d" -> "A",
                                       "C" -> "B",
                                       "D" -> "a"}];
         Sub7a = StringReplace[Sub,   {"a" -> "c",
                                       "b" -> "d",
                                       "A" -> "b",
                                       "B" -> "a"}];
         Sub7  = StringReplace[Sub7a, {"c" -> "B",
                                       "d" -> "A"}];
         ChSub1 = Characters[Sub1];
         ChSub2 = Characters[Sub2];
         ChSub3 = Characters[Sub3];
         ChSub4 = Characters[Sub4];
         ChSub5 = Characters[Sub5];
         ChSub6 = Characters[Sub6];
         ChSub7 = Characters[Sub7];

         For[k=1, k<=Length[ChSub1], k++,
             Z = RotateLeft[ChSub1,k];
             AppendTo[ListX, Z]];

         For[k=1, k<=Length[ChSub2], k++,
             Z = RotateLeft[ChSub2,k];
             AppendTo[ListX, Z]];

         For[k=1, k<=Length[ChSub3], k++,
             Z = RotateLeft[ChSub3,k];
             AppendTo[ListX, Z]];
```

```
        For[k=1, k<=Length[ChSub4], k++,
            Z = RotateLeft[ChSub4,k];
            AppendTo[ListX, Z]];

        For[k=1, k<=Length[ChSub5], k++,
            Z = RotateLeft[ChSub5,k];
            AppendTo[ListX, Z]];

        For[k=1, k<=Length[ChSub6], k++,
            Z = RotateLeft[ChSub6,k];
            AppendTo[ListX, Z]];

        For[k=1, k<=Length[ChSub7], k++,
            Z = RotateLeft[ChSub7,k];
            AppendTo[ListX, Z]];

        StringListX = Map[StringJoin,ListX];

        Discards = List[ ];

        For[j=i+1, j<=Length[Quintuples], j++,

            If[MemberQ[StringListX, Quintuples[[j]]],
               AppendTo[Discards, Quintuples[[j]]]];

        Quintuples = Complement[Quintuples,Discards]];

    For[i=1, i<=Length[Quadruples], i++, quadsubstitutions[i]];

    Sextuples = List[ ];

    FindSextuples[i_] :=
       Module[{Word = List19[[i]]},
          If[StringLength[Word] == 6,
             AppendTo[Sextuples, Word]]];

    For[i=1, i<= Length[List19], i++, FindSextuples[i]];

    List20 = Complement[List19, Sextuples];

    sextsubstitutions[i_] :=
       Module[{Sub = Sextuples[[i]]},
          ListX = List[ ];
          Sub1a = StringReplace[Sub,   {"a" -> "c",
                                        "b" -> "d",
                                        "A" -> "a",
```



```
                                        "B" -> "b"}];
          Sub1  = StringReplace[Sub1a, {"c" -> "A",
                                        "d" -> "B"}];
          Sub2a = StringReplace[Sub,   {"a" -> "c",
                                        "A" -> "a"}];
          Sub2  = StringReplace[Sub2a, "c" -> "A"];
          Sub3a = StringReplace[Sub,   {"b" -> "d",
                                        "B" -> "b"}];
          Sub3  = StringReplace[Sub3a, "d" -> "B"];
          Sub4a = StringReplace[Sub,   {"a" -> "c",
                                        "b" -> "a",
                                        "A" -> "C",
                                        "B" -> "A"}];
          Sub4  = StringReplace[Sub4a, {"c" -> "b",
                                        "C" -> "B"}];
          Sub5a = StringReplace[Sub,   {"a" -> "c",
                                        "b" -> "d",
                                        "A" -> "C",
                                        "B" -> "D"}];
          Sub5  = StringReplace[Sub5a, {"c" -> "B",
                                        "d" -> "a",
                                        "C" -> "b",
                                        "D" -> "A"}];
          Sub6a = StringReplace[Sub,   {"a" -> "c",
                                        "b" -> "d",
                                        "A" -> "C",
                                        "B" -> "D"}];
          Sub6  = StringReplace[Sub6a, {"c" -> "b",
                                        "d" -> "A",
                                        "C" -> "B",
                                        "D" -> "a"}];
          Sub7a = StringReplace[Sub,   {"a" -> "c",
                                        "b" -> "d",
                                        "A" -> "b",
                                        "B" -> "a"}];
          Sub7  = StringReplace[Sub7a, {"c" -> "B",
                                        "d" -> "A"}];
          ChSub1 = Characters[Sub1];
          ChSub2 = Characters[Sub2];
          ChSub3 = Characters[Sub3];
          ChSub4 = Characters[Sub4];
          ChSub5 = Characters[Sub5];
          ChSub6 = Characters[Sub6];
          ChSub7 = Characters[Sub7];

          For[k=1, k<=Length[ChSub1], k++,
```



```
              Z = RotateLeft[ChSub1,k];
              AppendTo[ListX, Z]];

          For[k=1, k<=Length[ChSub2], k++,
              Z = RotateLeft[ChSub2,k];
              AppendTo[ListX, Z]];

          For[k=1, k<=Length[ChSub3], k++,
              Z = RotateLeft[ChSub3,k];
              AppendTo[ListX, Z]];

          For[k=1, k<=Length[ChSub4], k++,
              Z = RotateLeft[ChSub4,k];
              AppendTo[ListX, Z]];

          For[k=1, k<=Length[ChSub5], k++,
              Z = RotateLeft[ChSub5,k];
              AppendTo[ListX, Z]];

          For[k=1, k<=Length[ChSub6], k++,
              Z = RotateLeft[ChSub6,k];
              AppendTo[ListX, Z]];

          For[k=1, k<=Length[ChSub7], k++,
              Z = RotateLeft[ChSub7,k];
              AppendTo[ListX, Z]];

          StringListX = Map[StringJoin,ListX];

          Discards = List[ ];

          For[j=i+1, j<=Length[Sextuples], j++,

              If[MemberQ[StringListX, Sextuples[[j]]],
                 AppendTo[Discards, Sextuples[[j]]]];

          Sextuples = Complement[Sextuples,Discards]];

    For[i=1, i<=Length[Sextuples], i++, sextsubstitutions[i]];

    Septuples = List[ ];

    FindSeptuples[i_] :=
       Module[{Word = List20[[i]]},
          If[StringLength[Word] == 7,
             AppendTo[Septuples, Word]]];
```



```
    For[i=1, i<= Length[List20], i++, FindSeptuples[i]];

    List21 = Complement[List20, Septuples];

    septsubstitutions[i_] :=
       Module[{Sub = Septuples[[i]]},
          ListX = List[ ];
          Sub1a = StringReplace[Sub,   {"a" -> "c",
                                        "b" -> "d",
                                        "A" -> "a",
                                        "B" -> "b"}];
          Sub1  = StringReplace[Sub1a, {"c" -> "A",
                                        "d" -> "B"}];
          Sub2a = StringReplace[Sub,   {"a" -> "c",
                                        "A" -> "a"}];
          Sub2  = StringReplace[Sub2a, "c" -> "A"];
          Sub3a = StringReplace[Sub,   {"b" -> "d",
                                        "B" -> "b"}];
          Sub3  = StringReplace[Sub3a, "d" -> "B"];
          Sub4a = StringReplace[Sub,   {"a" -> "c",
                                        "b" -> "a",
                                        "A" -> "C",
                                        "B" -> "A"}];
          Sub4  = StringReplace[Sub4a, {"c" -> "b",
                                        "C" -> "B"}];
          Sub5a = StringReplace[Sub,   {"a" -> "c",
                                        "b" -> "d",
                                        "A" -> "C",
                                        "B" -> "D"}];
          Sub5  = StringReplace[Sub5a, {"c" -> "B",
                                        "d" -> "a",
                                        "C" -> "b",
                                        "D" -> "A"}];
          Sub6a = StringReplace[Sub,   {"a" -> "c",
                                        "b" -> "d",
                                        "A" -> "C",
                                        "B" -> "D"}];
          Sub6  = StringReplace[Sub6a, {"c" -> "b",
                                        "d" -> "A",
                                        "C" -> "B",
                                        "D" -> "a"}];
          Sub7a = StringReplace[Sub,   {"a" -> "c",
                                        "b" -> "d",
                                        "A" -> "b",
                                        "B" -> "a"}];
```

```
Sub7  = StringReplace[Sub7a, {"c" -> "B",
                              "d" -> "A"}];
ChSub1 = Characters[Sub1];
ChSub2 = Characters[Sub2];
ChSub3 = Characters[Sub3];
ChSub4 = Characters[Sub4];
ChSub5 = Characters[Sub5];
ChSub6 = Characters[Sub6];
ChSub7 = Characters[Sub7];

For[k=1, k<=Length[ChSub1], k++,
    Z = RotateLeft[ChSub1,k];
    AppendTo[ListX, Z]];

For[k=1, k<=Length[ChSub2], k++,
    Z = RotateLeft[ChSub2,k];
    AppendTo[ListX, Z]];

For[k=1, k<=Length[ChSub3], k++,
    Z = RotateLeft[ChSub3,k];
    AppendTo[ListX, Z]];

For[k=1, k<=Length[ChSub4], k++,
    Z = RotateLeft[ChSub4,k];
    AppendTo[ListX, Z]];

For[k=1, k<=Length[ChSub5], k++,
    Z = RotateLeft[ChSub5,k];
    AppendTo[ListX, Z]];

For[k=1, k<=Length[ChSub6], k++,
    Z = RotateLeft[ChSub6,k];
    AppendTo[ListX, Z]];

For[k=1, k<=Length[ChSub7], k++,
    Z = RotateLeft[ChSub7,k];
    AppendTo[ListX, Z]];

StringListX = Map[StringJoin,ListX];

Discards = List[ ];

For[j=i+1, j<=Length[Septuples], j++,

    If[MemberQ[StringListX, Septuples[[j]]],
       AppendTo[Discards, Septuples[[j]]]];
```



```
Septuples = Complement[Septuples,Discards]];


For[i=1, i<=Length[Septuples], i++, septsubstitutions[i]];

Octets = List[ ];

FindOctets[i_] :=
   Module[{Word = List21[[i]]},
      If[StringLength[Word] == 8,
         AppendTo[Octets, Word]]];

For[i=1, i<= Length[List21], i++, FindOctets[i]];

List22 = Complement[List21, Octets];

octsubstitutions[i_] :=
   Module[{Sub = Octets[[i]]},
      ListX = List[ ];
      Sub1a = StringReplace[Sub,   {"a" -> "c",
                                    "b" -> "d",
                                    "A" -> "a",
                                    "B" -> "b"}];
      Sub1  = StringReplace[Sub1a, {"c" -> "A",
                                    "d" -> "B"}];
      Sub2a = StringReplace[Sub,   {"a" -> "c",
                                    "A" -> "a"}];
      Sub2  = StringReplace[Sub2a,  "c" -> "A"];
      Sub3a = StringReplace[Sub,   {"b" -> "d",
                                    "B" -> "b"}];
      Sub3  = StringReplace[Sub3a,  "d" -> "B"];
      Sub4a = StringReplace[Sub,   {"a" -> "c",
                                    "b" -> "a",
                                    "A" -> "C",
                                    "B" -> "A"}];
      Sub4  = StringReplace[Sub4a, {"c" -> "b",
                                    "C" -> "B"}];
      Sub5a = StringReplace[Sub,   {"a" -> "c",
                                    "b" -> "d",
                                    "A" -> "C",
                                    "B" -> "D"}];
      Sub5  = StringReplace[Sub5a, {"c" -> "B",
                                    "d" -> "a",
                                    "C" -> "b",
                                    "D" -> "A"}];
```



```
Sub6a = StringReplace[Sub,   {"a" -> "c",
                              "b" -> "d",
                              "A" -> "C",
                              "B" -> "D"}];
Sub6  = StringReplace[Sub6a, {"c" -> "b",
                              "d" -> "A",
                              "C" -> "B",
                              "D" -> "a"}];
Sub7a = StringReplace[Sub,   {"a" -> "c",
                              "b" -> "d",
                              "A" -> "b",
                              "B" -> "a"}];
Sub7  = StringReplace[Sub7a, {"c" -> "B",
                              "d" -> "A"}];
ChSub1 = Characters[Sub1];
ChSub2 = Characters[Sub2];
ChSub3 = Characters[Sub3];
ChSub4 = Characters[Sub4];
ChSub5 = Characters[Sub5];
ChSub6 = Characters[Sub6];
ChSub7 = Characters[Sub7];

For[k=1, k<=Length[ChSub1], k++,
    Z = RotateLeft[ChSub1,k];
    AppendTo[ListX, Z]];

For[k=1, k<=Length[ChSub2], k++,
    Z = RotateLeft[ChSub2,k];
    AppendTo[ListX, Z]];

For[k=1, k<=Length[ChSub3], k++,
    Z = RotateLeft[ChSub3,k];
    AppendTo[ListX, Z]];

For[k=1, k<=Length[ChSub4], k++,
    Z = RotateLeft[ChSub4,k];
    AppendTo[ListX, Z]];

For[k=1, k<=Length[ChSub5], k++,
    Z = RotateLeft[ChSub5,k];
    AppendTo[ListX, Z]];

For[k=1, k<=Length[ChSub6], k++,
    Z = RotateLeft[ChSub6,k];
    AppendTo[ListX, Z]];
```



```
For[k=1, k<=Length[ChSub7], k++,
    Z = RotateLeft[ChSub7,k];
    AppendTo[ListX, Z]];

StringListX = Map[StringJoin,ListX];

Discards = List[ ];

For[j=i+1, j<=Length[Octets], j++,

    If[MemberQ[StringListX, Octets[[j]]],
       AppendTo[Discards, Octets[[j]]]];

Octets = Complement[Octets,Discards]];

For[i=1, i<=Length[Octets], i++, octsubstitutions[i]];

Nonets = List[ ];

FindNonets[i_] :=
   Module[{Word = List22[[i]]},
      If[StringLength[Word] == 9,
         AppendTo[Nonets, Word]]];

For[i=1, i<= Length[List22], i++, FindNonets[i]];

List23 = Complement[List22, Nonets];

nonsubstitutions[i_] :=
   Module[{Sub = Nonets[[i]]},
      ListX = List[ ];
      Sub1a = StringReplace[Sub,   {"a" -> "c",
                                    "b" -> "d",
                                    "A" -> "a",
                                    "B" -> "b"}];
      Sub1  = StringReplace[Sub1a, {"c" -> "A",
                                    "d" -> "B"}];
      Sub2a = StringReplace[Sub,   {"a" -> "c",
                                    "A" -> "a"}];
      Sub2  = StringReplace[Sub2a,  "c" -> "A"];
      Sub3a = StringReplace[Sub,   {"b" -> "d",
                                    "B" -> "b"}];
      Sub3  = StringReplace[Sub3a,  "d" -> "B"];
      Sub4a = StringReplace[Sub,   {"a" -> "c",
                                    "b" -> "a",
                                    "A" -> "C",
```

```
                                    "B" -> "A"}];
    Sub4  = StringReplace[Sub4a, {"c" -> "b",
                                  "C" -> "B"}];
    Sub5a = StringReplace[Sub,   {"a" -> "c",
                                  "b" -> "d",
                                  "A" -> "C",
                                  "B" -> "D"}];
    Sub5  = StringReplace[Sub5a, {"c" -> "B",
                                  "d" -> "a",
                                  "C" -> "b",
                                  "D" -> "A"}];
    Sub6a = StringReplace[Sub,   {"a" -> "c",
                                  "b" -> "d",
                                  "A" -> "C",
                                  "B" -> "D"}];
    Sub6  = StringReplace[Sub6a, {"c" -> "b",
                                  "d" -> "A",
                                  "C" -> "B",
                                  "D" -> "a"}];
    Sub7a = StringReplace[Sub,   {"a" -> "c",
                                  "b" -> "d",
                                  "A" -> "b",
                                  "B" -> "a"}];
    Sub7  = StringReplace[Sub7a, {"c" -> "B",
                                  "d" -> "A"}];
    ChSub1 = Characters[Sub1];
    ChSub2 = Characters[Sub2];
    ChSub3 = Characters[Sub3];
    ChSub4 = Characters[Sub4];
    ChSub5 = Characters[Sub5];
    ChSub6 = Characters[Sub6];
    ChSub7 = Characters[Sub7];

    For[k=1, k<=Length[ChSub1], k++,
        Z = RotateLeft[ChSub1,k];
        AppendTo[ListX, Z]];

    For[k=1, k<=Length[ChSub2], k++,
        Z = RotateLeft[ChSub2,k];
        AppendTo[ListX, Z]];

    For[k=1, k<=Length[ChSub3], k++,
        Z = RotateLeft[ChSub3,k];
        AppendTo[ListX, Z]];

    For[k=1, k<=Length[ChSub4], k++,
```



```
        Z = RotateLeft[ChSub4,k];
        AppendTo[ListX, Z]];

    For[k=1, k<=Length[ChSub5], k++,
        Z = RotateLeft[ChSub5,k];
        AppendTo[ListX, Z]];

    For[k=1, k<=Length[ChSub6], k++,
        Z = RotateLeft[ChSub6,k];
        AppendTo[ListX, Z]];

    For[k=1, k<=Length[ChSub7], k++,
        Z = RotateLeft[ChSub7,k];
        AppendTo[ListX, Z]];

    StringListX = Map[StringJoin,ListX];

    Discards = List[ ];

    For[j=i+1, j<=Length[Nonets], j++,

        If[MemberQ[StringListX, Nonets[[j]]],
          AppendTo[Discards, Nonets[[j]]]];

    Nonets = Complement[Nonets,Discards]];

For[i=1, i<=Length[Nonets], i++, nonsubstitutions[i]];

Tens = List[ ];

FindTens[i_] :=
  Module[{Word = List23[[i]]},
    If[StringLength[Word] == 10,
      AppendTo[Tens, Word]]];

For[i=1, i<= Length[List23], i++, FindTens[i]];

List24 = Complement[List23, Tens];

Elevens = List[ ];

FindElevens[i_] :=
  Module[{Word = List24[[i]]},
    If[StringLength[Word] == 11,
      AppendTo[Elevens, Word]]];
```



```
For[i=1, i<= Length[List24], i++, FindElevens[i]];

List25 = Complement[List24, Elevens];

Twelves = List[ ];

FindTwelves[i_] :=
  Module[{Word = List25[[i]]},
    If[StringLength[Word] == 12,
      AppendTo[Twelves, Word]]];

For[i=1, i<= Length[List25], i++, FindTwelves[i]];

List26 = Complement[List25, Twelves];

twelvesubstitutions[i_] :=
  Module[{Sub = Twelves[[i]]},
    ListX = List[ ];
    Sub1a = StringReplace[Sub,   {"a" -> "c",
                                  "b" -> "d",
                                  "A" -> "a",
                                  "B" -> "b"}];
    Sub1  = StringReplace[Sub1a, {"c" -> "A",
                                  "d" -> "B"}];
    Sub2a = StringReplace[Sub,   {"a" -> "c",
                                  "A" -> "a"}];
    Sub2  = StringReplace[Sub2a, "c" -> "A"];
    Sub3a = StringReplace[Sub,   {"b" -> "d",
                                  "B" -> "b"}];
    Sub3  = StringReplace[Sub3a, "d" -> "B"];
    Sub4a = StringReplace[Sub,   {"a" -> "c",
                                  "b" -> "a",
                                  "A" -> "C",
                                  "B" -> "A"}];
    Sub4  = StringReplace[Sub4a, {"c" -> "b",
                                  "C" -> "B"}];
    Sub5a = StringReplace[Sub,   {"a" -> "c",
                                  "b" -> "d",
                                  "A" -> "C",
                                  "B" -> "D"}];
    Sub5  = StringReplace[Sub5a, {"c" -> "B",
                                  "d" -> "a",
                                  "C" -> "b",
                                  "D" -> "A"}];
    Sub6a = StringReplace[Sub,   {"a" -> "c",
                                  "b" -> "d",
```



```
                                  "A" -> "C",
                                  "B" -> "D"}];
    Sub6  = StringReplace[Sub6a, {"c" -> "b",
                                  "d" -> "A",
                                  "C" -> "B",
                                  "D" -> "a"}];
    Sub7a = StringReplace[Sub,   {"a" -> "c",
                                  "b" -> "d",
                                  "A" -> "b",
                                  "B" -> "a"}];
    Sub7  = StringReplace[Sub7a, {"c" -> "B",
                                  "d" -> "A"}];
    ChSub1 = Characters[Sub1];
    ChSub2 = Characters[Sub2];
    ChSub3 = Characters[Sub3];
    ChSub4 = Characters[Sub4];
    ChSub5 = Characters[Sub5];
    ChSub6 = Characters[Sub6];
    ChSub7 = Characters[Sub7];

    For[k=1, k<=Length[ChSub1], k++,
        Z = RotateLeft[ChSub1,k];
        AppendTo[ListX, Z]];

    For[k=1, k<=Length[ChSub2], k++,
        Z = RotateLeft[ChSub2,k];
        AppendTo[ListX, Z]];

    For[k=1, k<=Length[ChSub3], k++,
        Z = RotateLeft[ChSub3,k];
        AppendTo[ListX, Z]];

    For[k=1, k<=Length[ChSub4], k++,
        Z = RotateLeft[ChSub4,k];
        AppendTo[ListX, Z]];

    For[k=1, k<=Length[ChSub5], k++,
        Z = RotateLeft[ChSub5,k];
        AppendTo[ListX, Z]];

    For[k=1, k<=Length[ChSub6], k++,
        Z = RotateLeft[ChSub6,k];
        AppendTo[ListX, Z]];

    For[k=1, k<=Length[ChSub7], k++,
        Z = RotateLeft[ChSub7,k];
```

```
            AppendTo[ListX, Z]];

        StringListX = Map[StringJoin,ListX];

        Discards = List[ ];

        For[j=i+1, j<=Length[Twelves], j++,

            If[MemberQ[StringListX, Twelves[[j]]],
               AppendTo[Discards, Twelves[[j]]]];

        Twelves = Complement[Twelves,Discards]];

For[i=1, i<=Length[Twelves], i++, twelvesubstitutions[i]];

List27 = Join[Triples,
              Quadruples,
              Quintuples,
              Sextuples,
              Septuples,
              Octets,
              Nonets,
              Tens,
              Elevens,
              Twelves];

Print["List27 = ", List27];

Print["Length List27 = ", Length[List27]];
```



```
Length List1 = 621
Length List2 = 361
Length List3 = 341
Length List4 = 321
Length List5 = 233
Length List6 = 221
Length List9 = 200
Length List11 = 165
165
154
145
145
145
145
145
145
145
145
145
145
145
Length List13 = 105
Length List14 = 105
List16 = 95
List27 = {aaa, aba, AAbb, abAb, AbAb, baBA, aaBab, abAAb,

   aBAAB, Abaab, ABaaB, aBAba, AbaBA, ABabA, baBaa, bABaa,

   bABAA, BAbAA, baBab, baBAb, bABab, baBBA, bABBA, BAbbA,

   aaabAB, aabAAB, AAbaBB, ababAB, bABAba, bbABBA, abAbABB,

   bAAbaBa, bAAbABa, BAAbABa, bABABab, aabAAbAB, aabAABAb,

   abaBAbAB, abaBABAb, abABabaB, abABABAb, AbaBAbaB,

   ABabAbAB, babAAABa, baBAAABa, bbaBAABa, bbABAABa,

   abAbaBAAB, abABabAAB, abABabABabAB}
Length List27 = 50
```