2019-11-26 For today

Necessary stuff from Crypto.mw

```
> with(StringTools) :
> StringToList:=proc(str::string, {base::posint:=1},
{keepbad::truefalse:=false})
    local numlist;
    global Alphabet;
    numlist:=map( s->SearchText(s,Alphabet)-1, Explode(str));
    if (not(keepbad)) then
      numlist:=remove(x->x<0,numlist);
      if (nops(numlist)<length(str)) then
        WARNING("%1 characters have been ignored because they
don't occur in the Alphabet", length(str)-nops(numlist));
        fi;
     fi;
    if (base>1) then
      numlist:=convert(numlist, ':-base', length(Alphabet),
base);
     fi;
    return(numlist);
  end:
> ListToString:=proc(nums::list(nonnegint), {base::posint:=1})
    local numlist;
    global Alphabet;
    if (base>1) then
      numlist:=convert(nums, ':-base', base, length(Alphabet));
     else
      numlist:=nums;
     fi;
    return(Implode(map(k->Alphabet[k+1],numlist)) );
  end:



>
```

```
> Alphabet := "ABCDEFGHIJKLMNOPQRSTUVWXYZ"; length(Alphabet);
                    Alphabet := "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
                                        26                                  (1)
```

A brief aside about using larger bases. I won't put this in here, since it was in a previous class.

```
> StringToList("THISISLONG")
                      [19, 7, 8, 18, 8, 18, 11, 14, 13, 6]                  (2)
```

```
> StringToList("THISISLONG", base=26^4)
                            [321977, 253976, 169]                           (3)
```

We will stir things up using vectors/matrices.
The notes are using an old interface for linear algebra, which is deprecated by MapleSoft. While they still work (so far), we will discuss implementation using the newer interface, which is both more flexible and more efficient.

> $with(LinearAlgebra):$

a Vector in maple is like a list, but has more structure. We can enter data for Vectors in several ways. Here are a few:
A column vector:

> $V := \langle 1, 2, 3\rangle$

$$V := \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \tag{4}$$

or a row vector (note that the | is used to separate the columns of a Vector (or Matrix):

> $W := \langle 4|5|6\rangle;$

$$W := \begin{bmatrix} 4 & 5 & 6 \end{bmatrix} \tag{5}$$

Or build it from a list:

> $V1 := Vector([1, 2, 3]);$
$W1 := Vector[row]([4, 5, 6, 7]);$

$$V1 := \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$
$$W1 := \begin{bmatrix} 4 & 5 & 6 & 7 \end{bmatrix} \tag{6}$$

Similarly, we can enter a Matrix in a couple of ways:

> $M := Matrix([[1, 2], [3, 4]])$

$$M := \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \tag{7}$$

> $N := \langle\langle 1, 2\rangle\,|\langle 3, 4\rangle\rangle$

$$N := \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} \tag{8}$$

> $N1 := Transpose(M)$

$$N1 := \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} \tag{9}$$

Goal is to use matrix to encrypt, that is V -> A*V mod L

Is it true that A*B mod L = (A mod L)*(B mod L)   (almost.... as long as we reduce mod L at the end)

> $A := \langle\langle 27, 2\rangle|\langle 3, 4\rangle\rangle;$
$B := \langle\langle 5, 32\rangle|\ \langle 7, 8\rangle\rangle;$

$$A := \begin{bmatrix} 27 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B := \begin{bmatrix} 5 & 7 \\ 32 & 8 \end{bmatrix} \tag{10}$$

Matrix multiplication is indicated by a . between the items, enter as A.B

> $A \cdot B$

$$\begin{bmatrix} 231 & 213 \\ 138 & 46 \end{bmatrix} \quad (11)$$

> $(A \textbf{ mod } 26) \cdot (B \textbf{ mod } 26)$

$$\begin{bmatrix} 23 & 31 \\ 34 & 46 \end{bmatrix} \quad (12)$$

> $\% \textbf{ mod } 26$

$$\begin{bmatrix} 23 & 5 \\ 8 & 20 \end{bmatrix} \quad (13)$$

> $A \cdot B \textbf{ mod } 26$

$$\begin{bmatrix} 23 & 5 \\ 8 & 20 \end{bmatrix} \quad (14)$$

One can prove (sorry, I won't, but it is illustrated by the previous example) that equivalence classes mod L are preserved by matrix/vector arithmetic

Next question: when are matrices invertible mod L?

Recall that a Matrix is invertible (ie, non-singular) if the determinant is not zero.   The same is true in modular matrix arithmetic.

> $Binv := MatrixInverse(B)$

$$Binv := \begin{bmatrix} -\frac{1}{23} & \frac{7}{184} \\ \frac{4}{23} & -\frac{5}{184} \end{bmatrix} \quad (15)$$

> $Binv \textbf{ mod } 26$

Error, (in Matrix) the modular inverse does not exist

> $Determinant(Binv)$

$$-\frac{1}{184} \quad (16)$$

> $-\frac{1}{184} \textbf{ mod } 26$

Error, the modular inverse does not exist

> $Determinant(B)$

$$-184 \quad (17)$$

> $gcd(\%, 26)$

$$2 \quad (18)$$

Again, remember that -1/184 mod 26 means "the solution to the equation $-184 \cdot x = 1 \textbf{ mod } 26$ ". Since -184 is even and our base (26) is even, we can't multiply any number times an even number and get an odd number (1) as the solution. So -184 has no multiplicative inverse mod 26.

A matrix mod L is invertible <=> Determinant(A) mod L is coprime to L   (ie, Determinant has an inverse mod L)

Time is running short.
Let's write a StringToVect procedure.

```
> msg := "HAPPYTHANKSGIVING";
  length(message);
                    msg := "HAPPYTHANKSGIVING"
                               7                                    (19)
```

```
> numlist := StringToList(msg);
   numlist := [7, 0, 15, 15, 24, 19, 7, 0, 13, 10, 18, 6, 8, 21, 8, 13, 6]   (20)
```

A problem: If we are to have 2-vectors or 3-vectors (or any vectors that aren't a divisor of 7), we're going to run into a problem.
We will need to pad out the message with something so the last vector gets filled. The class likes using a "Q" as padding.

For now, let's let the size of the vectors be 2, but keep in mind it can be anything.

Here's a by-hand version:

```
> Vsize := 2 :
   pmsg := msg : # let's let pmsg be our padded message.
  while (length(pmsg) mod Vsize ≠ 0) do
    pmsg := cat(pmsg, "Q");
   od:
   pmsg;

                    "HAPPYTHANKSGIVINGQ"                            (21)
```

```
> Vsize := 4; pmsg := msg :
  while (length(pmsg) mod 4 ≠ 0) do
    pmsg := cat(pmsg, "Q");
   od:
   pmsg;
                          Vsize := 4
                   "HAPPYTHANKSGIVINGQQQ"                           (22)
```

Now here is a general procedure to pad out our message:

```
> PadString:=proc(msg::string, factor::posint,
  {padchar::character:="Q"})
    local pmsg;
    pmsg:=msg;
    # as long as the length of the string is not divisible by
  factor
    # stick another padding character on the end.
    while ((length(pmsg) mod factor) <> 0) do
      pmsg:=cat(pmsg,padchar)
     od:
    return(pmsg);
  end:
```

Let's confirm it works:

```
> PadString(msg,2)
                    "HAPPYTHANKSGIVINGQ"                            (23)
```

```
> PadString(msg,5,padchar="x");  #pad with an "x" instead of "Q".
                    "HAPPYTHANKSGIVINGxxx"                                  (24)
> PadString(msg,17);  # this shouldn't pad, since the message
  length is divisible by 17
                    "HAPPYTHANKSGIVING"                                     (25)
```

At this point, I ran out of time in the class. But since I am cleaning this up after the fact, let me finish the job.

We need to write a StringToVects procedure, the analogue of StringToList, as well as its inverse. An obvious way to do this is to
1) pad out the message to be a multiple of the size of our desired Vector length
2) convert the message to a list of numbers using StringToList
3) grab the elements N at a time and put them into Vectors,
4) return a list of the Vectors.

But rather than do it that way, I'm going to do step 3 & 4 in a non-obvious way, sticking them into a Matrix and return the columns of the resulting Matrix.
Here's an example (not in a procedure) of how that works.

First, the Matrix and Vector types are special versions of the Array type, and a collection of things for messing with them is called ArrayTools.
I only need one of those (the Reshape command [we could also use Alias]), so I'm not going to load the whole package.

Before writing the general procedure, let's play around to get a feel for what we're doing. I want 9 2-vectors from my "HAPPYTHANKSGIVING" message (adding a Q as padding).

> $pmsg := PadString(msg, 2)$

$$pmsg := \text{"HAPPYTHANKSGIVINGQ"} \qquad (26)$$

> $numlist := StringToList(pmsg)$

$$numlist := [7, 0, 15, 15, 24, 19, 7, 0, 13, 10, 18, 6, 8, 21, 8, 13, 6, 16] \qquad (27)$$

> $M1 := Matrix(numlist)$ # this is a 1x18 matrix

$$M1 := \underset{1 \times 18 \text{ Matrix}}{\begin{bmatrix} 7 & 0 & 15 & 15 & 24 & 19 & 7 & 0 & 13 & 10 & \dots \end{bmatrix}} \qquad (28)$$

> $M := ArrayTools{:-}Reshape(M1, 2, 9)$

$$M := \begin{bmatrix} 7 & 15 & 24 & 7 & 13 & 18 & 8 & 8 & 6 \\ 0 & 15 & 19 & 0 & 10 & 6 & 21 & 13 & 16 \end{bmatrix} \qquad (29)$$

We can reference individual columns of M by specifying the first index (the row number) as a range, and the second as our desired column. For example, to get the 5th column, we can do the following:

> $M[1..2, 5]$

$$\begin{bmatrix} 13 \\ 10 \end{bmatrix} \qquad (30)$$

We could either choose to leave our Vectors as the columns of a Matrix, or split them into a list of Vectors. I think it is less confusing to do the latter, so we can do something like

> $\left[seq\left(M[1..2, i], i = 1 .. \frac{length(pmsg)}{2} \right) \right]$

$$\left[\begin{bmatrix}7\\0\end{bmatrix}, \begin{bmatrix}15\\15\end{bmatrix}, \begin{bmatrix}24\\19\end{bmatrix}, \begin{bmatrix}7\\0\end{bmatrix}, \begin{bmatrix}13\\10\end{bmatrix}, \begin{bmatrix}18\\6\end{bmatrix}, \begin{bmatrix}8\\21\end{bmatrix}, \begin{bmatrix}8\\13\end{bmatrix}, \begin{bmatrix}6\\16\end{bmatrix}\right] \quad (31)$$

OK, now let's put that all together, with some generality.
Since I'm building the padded-out string inside another command and I need to know how long it is, I'll just grab the dimension of the
1 by n intermediate matrix instead. The Dimensions command tells us how big a Matrix is.

```
> StringToVects:=proc(msg::string,vsize::posint,
  {padchar::character:="Q"})
    local  M,i;
    M:= Matrix( StringToList( PadString(msg,vsize,':-padchar'=
  padchar)));
    M:= ArrayTools:-Reshape(M,vsize,Dimensions(M)[2]/vsize);
    # now we have a matrix that has a bunch of columns of vsize-
  vectors.
    # Convert to a list of vectors instead.
    return( [seq(M[1..vsize,i],i=1..Dimensions(M)[2])]);
  end:
```

That should work; let's try it out.

```
> TwoVects:=StringToVects(msg,2)
```

$$TwoVects := \left[\begin{bmatrix}7\\0\end{bmatrix}, \begin{bmatrix}15\\15\end{bmatrix}, \begin{bmatrix}24\\19\end{bmatrix}, \begin{bmatrix}7\\0\end{bmatrix}, \begin{bmatrix}13\\10\end{bmatrix}, \begin{bmatrix}18\\6\end{bmatrix}, \begin{bmatrix}8\\21\end{bmatrix}, \begin{bmatrix}8\\13\end{bmatrix}, \begin{bmatrix}6\\16\end{bmatrix}\right] \quad (32)$$

```
> FiveVects:=StringToVects(msg,5,padchar="X")
```

$$FiveVects := \left[\begin{bmatrix}7\\0\\15\\15\\24\end{bmatrix}, \begin{bmatrix}19\\7\\0\\13\\10\end{bmatrix}, \begin{bmatrix}18\\6\\8\\21\\8\end{bmatrix}, \begin{bmatrix}13\\6\\23\\23\\23\end{bmatrix}\right] \quad (33)$$

To undo this, we collapse the vectors into a list of numbers (by first turning them back into a Matrix, then converting to a list), then convert using ListToString. We can use convert to flatten the Matrix, using nested=true. There are other ways to do this.
We could also drop the trailing padding characters, but let's not.

```
> VectsToString:=proc(Vlist::list(Vector))
    return(ListToString( convert(Matrix(Vlist),list,nested=false)))
  ;
  end:
> VectsToString(TwoVects);
                         "HAPPYTHANKSGIVINGQ"                      (34)
> VectsToString(FiveVects);
                        "HAPPYTHANKSGIVINGXXX"                     (35)
```

OK, now that the houskeeping is done (that was the hard part), let's write a version of Affine that will encrypt/decrypt given a Matrix and a Vector.
We need to check that the given Matrix is invertible, though.

The inverse of Y=A.V+B is just W.(Y-B) = W.Y -W.B, where W is the inverse of A.

```
> AffineMat:=proc(msg::string, A::Matrix, B1::Vector:=0,
  {decrypt::truefalse:=false})
    global Alphabet;
    local modbase, size, vects, Ainv, B;
    modbase:=length(Alphabet);
    size:=Dimensions(A)[1];

    if (B1=0) then # if B is not specified, make it the zero vector
       B:=Vector(size,fill=0);
    else
       B:=B1;
     fi;
    if (gcd(Determinant(A), modbase)<>1) then
       error(sprintf("encrypting matrix is not invertible mod %d;
  cannot decrypt", modbase));
     fi;
    if (decrypt) then
      Ainv:=MatrixInverse(A) mod modbase;
      return(AffineMat(msg, Ainv, -Ainv.B mod modbase));
     fi;

    vects:=StringToVects(msg,size);
    # now do the encryption; for each vector v, compute A.v+B mod
  modbase
    vects:=map(v->A.v+B mod modbase, vects);
    return(VectsToString(vects));
  end:
```

This should work. Let's give it a try or two.

Note that the matrix <<0,1>|<1,0>> should exchange adjacent characters. Let's confirm.

```
> AffineMat("TWOBYTWO",<<0,1>|<1,0>>);
  AffineMat(%,<<0,1>|<1,0>>,decrypt)
```

"WTBOTYOW"

"TWOBYTWO" (36)

Let's encrypt with some random biggish key (a random 7x7 matrix with a random 7 vector shift). (Note that since the matrix is random, it might not be invertible when you do this).

```
> RandA:=RandomMatrix(7) mod 26: RandB:=RandomVector(7) mod 26:
  RandA, RandB;
```

$$\left[\begin{array}{ccccccc} 14 & 2 & 3 & 3 & 11 & 1 & 7 \\ 19 & 10 & 9 & 0 & 6 & 10 & 21 \\ 17 & 10 & 24 & 8 & 19 & 5 & 17 \\ 8 & 25 & 23 & 0 & 4 & 2 & 19 \\ 24 & 6 & 13 & 0 & 21 & 15 & 17 \\ 25 & 10 & 3 & 11 & 12 & 1 & 3 \\ 4 & 8 & 0 & 5 & 7 & 9 & 24 \end{array}\right], \left[\begin{array}{c} 22 \\ 12 \\ 21 \\ 20 \\ 2 \\ 14 \\ 20 \end{array}\right] \qquad (37)$$

```
> crypted:=AffineMat(msg,RandA,RandB);
  AffineMat(crypted,RandA,RandB,decrypt);
```

$crypted$:= "WHQMZZGPJGAHVAKETLCOS"

(38)

```
"HAPPYTHANKSGIVINGQQQQ"                                        (38)
```

Observe how the matrix encryption really stirs up the message. Let's encrypt a message that is just 14 Xs; since our key is 7x7, we will see the first half the same as the second, but that's about it.

```
> rept:=AffineMat("XXXXXXXXXXXXXX",RandA)
                    rept := "HJMRYNLHJMRYNL"                   (39)
> AffineMat("FAIL",<<1,2>|<3,4>>)
Error, (in AffineMat) encrypting matrix is not invertible mod 
26; cannot decrypt
```

There are lots of improvements we can make, such as using a larger base, getting rid of padding when representing as characters, etc. But that's enough for now.
I will add this routine to Crypto.mw, in case you want to use it.