- `mat2bits(M)`, which is the inverse of `bits2mat`

You can use `str2bits` to transform a string, say `"Rosebud"`, into a list of bits, and use `bits2mat` to transform the list of bits to a $2 \times n$ matrix.

For each column of this matrix, you can use the procedure `choose_secret_vector(s,t)` of Task 7.7.1 to obtain a corresponding secret vector $\boldsymbol{u}$, constructing a matrix U whose columns are the secret vectors.

To compute the shares of the TAs, multiply the matrix

$$\begin{bmatrix} \boldsymbol{a}_0 \\ \hline \boldsymbol{b}_0 \\ \hline \boldsymbol{a}_1 \\ \hline \boldsymbol{b}_1 \\ \hline \boldsymbol{a}_2 \\ \hline \boldsymbol{b}_2 \\ \hline \boldsymbol{a}_3 \\ \hline \boldsymbol{b}_3 \\ \hline \boldsymbol{a}_4 \\ \hline \boldsymbol{b}_4 \end{bmatrix}$$

times U. The second and third rows of the product form the share for TA 1, and so on.

7.8 *Lab: Factoring integers*

7.8.1 *First attempt to use square roots*

In one step towards a modern factorization algorithm, suppose you could find integers a and b such that

$$a^2 - b^2 = N$$

for then

$$(a-b)(a+b) = N$$

so $a-b$ and $a+b$ are divisors of N. We hope that they happen to be nontrivial divisors (ie. that $a-b$ is neither 1 nor N).

Task 7.8.1: To find integers a and b such that $a^2 - b^2 = N$, write a procedure `root_method(N)` to implement the following algorithm:

- Initialize integer a to be an integer greater than $\sqrt{N}$
- Check if $\sqrt{a^2 - N}$ is an integer.
- If so, let $b = \sqrt{a^2 - N}$. Success! Return $a - b$.
- If not, repeat with the next greater value of a.

The module `factoring_support` provides a procedure `intsqrt(x)` with the following spec:

- *input:* an integer x
- *output:* an integer y such that $y * y$ is close to x and, if x happens to be a perfect square, $y * y$ is exactly x.

You should use `intsqrt(x)` in your implementation of the above algorithm. Try it out with 55, 77, 146771, and 118. Hint: the procedure might find just a trivial divisor or it might run forever.

7.8.2 *Euclid's algorithm for greatest common divisor*

In order to do better, we turn for help to a lovely algorithm that dates back some 2300 years: Euclid's algorithm for greatest common divisor. Here is code for it:

```
def gcd(x,y): return x if y == 0 else gcd(y, x % y)
```

Task 7.8.2: Enter the code for gcd or import it from the module `factoring_support` that we provide. Try it out. Specifically, use Python's pseudo-random-number generator (use the procedure `randint(a,b)` in the module `random`) or use pseudo-random whacking at your keyboard to generate some very big integers r, s, t. Then set $a = r * s$ and $b = s * t$, and find the greatest common divisor d of a and b. Verify that d has the following properties:

- a is divisible by d (verify by checking that $a\%d$ equals zero)
- b is divisible by d, and
- $d \geq s$

7.8.3 *Using square roots revisited*

It's too hard to find integers a and b such that $a^2 - b^2$ equals N. We will lower our standards a bit, and seek integers a and b such that $a^2 - b^2$ is divisible by N. Suppose we find such integers. Then there is another integer k such that

$$a^2 - b^2 = kN$$

That means

$$(a-b)(a+b) = kN$$

Every prime in the bag of primes whose product is kN

- belongs either to the the bag of primes whose product is k or the bag of primes whose product is N, and
- belongs either to the the bag of primes whose product is $a-b$ or the bag of primes whose product is $a+b$.

Suppose N is the product of two primes, p and q. If we are even a little lucky, one of these primes will belong to the bag for $a-b$ and the other will belong to the bag for $a+b$. If this happens, the greatest common divisor of $a-b$ with N will be nontrivial! And, thanks to Euclid's algorithm, we can actually compute it.

Task 7.8.3: Let $N = 367160330145890434494322103$, let $a = 67469780066325164$, and let $b = 9429601150488992$, and verify that $a*a-b*b$ is divisible by N. That means that the greatest common divisor of $a-b$ and N has a chance of being a nontrivial divisor of N. Test this using the `gcd` procedure, and report the nontrivial divisor you found.

But how can we find such a pair of integers? Instead of hoping to get lucky, we'll take matters into our own hands. We'll try to create a and b. This method starts by creating a set `primeset` consisting of the first thousand or so primes. We say an integer x factors over *primeset* if you can multiply together some of the primes in S (possibly using a prime more than once) to form x.

For example:

- 75 factors over $\{2,3,5,7\}$ because $75 = 3 \cdot 5 \cdot 5$.
- 30 factors over $\{2,3,5,7\}$ because $30 = 2 \cdot 3 \cdot 5$.
- 1176 factors over $\{2,3,5,7\}$ because $1176 = 2 \cdot 2 \cdot 2 \cdot 3 \cdot 7 \cdot 7$.

We can represent a factorization of an integer over a set of primes by a list of pairs (prime, exponent). For example:

- We can represent the factorization of 75 over $\{2,3,5,7\}$ by the list of pairs $[(3,1),(5,2)]$, indicating that 75 is obtained by multiplying a single 3 and two 5's.
- We can represent the factorization of 30 by the list $[(2,1),(3,1),(5,1)]$, indicating that 30 is obtained by multiplying 2, 3, and 5.

- We can represent the factorization of 1176 by the list $[(2, 3), (3, 1), (7, 2)]$, indicating that 1176 is obtained by multiplying together three 2's, one 3 and two 7's.

The first number in each pair is a prime in the set *primeset* and the second number is its exponent:

$$\begin{aligned} 75 &= 3^1 5^2 \\ 30 &= 2^1 3^1 5^1 \\ 1176 &= 2^3 3^1 7^2 \end{aligned}$$

The module `factoring_support` defines a procedure `dumb_factor(x, primeset)` with the following spec:

- *input:* an integer x and a set *primeset* of primes
- *output:* if there are primes $p_1, \ldots, \boldsymbol{p}_s$ in *primeset* and positive integers $e_1, e_2, \ldots, \boldsymbol{e}_s$ (the exponents) such that $x = p_1^{e_1} p_2^{e_2} \cdots p_s^{e_s}$ then the procedure returns the list $[(p_1, e_1), (p_2, e_2), .$ of pairs (prime, exponent). If not, the procedure returns the empty list.

Here are some examples:

```
>>> dumb_factor(75, {2,3,5,7})
[(3, 1), (5, 2)]
>>> dumb_factor(30, {2,3,5,7})
[(2, 1), (3, 1), (5, 1)]
>>> dumb_factor(1176, {2,3,5,7})
[(2, 3), (3, 1), (7, 2)]
>>> dumb_factor(2*17, {2,3,5,7})
[]
>>> dumb_factor(2*3*5*19, {2,3,5,7})
[]
```

Task 7.8.4: Define *primeset*$=\{2, 3, 5, 7, 11, 13\}$. Try out `dumb_factor(x, primeset)` on integers $x = 12, x = 154, x = 2 * 3 * 3 * 3 * 11 * 11 * 13, x = 2 * 17, x = 2 * 3 * 5 * 7 * 19$. Report the results.

Task 7.8.5: From the `GF2` module, import the value `one`. Write a procedure `int2GF2(i)` that, given an integer i, returns `one` if i is odd and 0 if i is even.

```
>>> int2GF2(3)
one
>>> int2GF2(4)
0
```

The module `factoring_support` defines a procedure `primes(P)` that returns a set consisting of the prime numbers less than P.

Task 7.8.6: From the module `vec`, import `Vec`. Write a procedure `make_Vec(primeset, factors)` with the following spec:

- *input:* a set of primes *primeset* and a list *factors*$=[(p_1, a_1), (p_2, a_2), \ldots, (p_s, a_s)]$ such as produced by `dumb_factor`, where every p_i belongs to *primeset*
- *output:* a *primeset*-vector $\boldsymbol{v}$ over $GF(2)$ with domain *primeset* such that $v[p_i] =$ `int2GF2`(a_i) for $i = 1, \ldots, s$

For example,

```
>>> make_Vec({2,3,5,7,11}, [(3,1)])
Vec({3, 2, 11, 5, 7},{3: one})
>>> make_Vec({2,3,5,7,11}, [(2,17), (3, 0), (5,1), (11,3)])
Vec({3, 2, 11, 5, 7},{11: one, 2: one, 3: 0, 5: one})
```

Now comes the interesting part.

Task 7.8.7: Suppose you want to factor the integer $N = 2419$ (easy but big enough to demonstrate the idea).

Write a procedure `find_candidates(N, primeset)` that, given an integer N to factor and a set $primeset$ of primes, finds `len(primeset)+1` integers a for which $a \cdot a - N$ can be factored completely over `primeset` The procedure returns two lists:

- the list `roots` consisting of $a_0, a_1, a_2, \ldots$ such that $a_i \cdot a_i - N$ can be factored completely over `primeset`, and
- the list `rowlist` such that element i is the `primeset`-vector over $GF(2)$ corresponding to a_i (that is, the vector produced by `make_vec`).

The algorithm should initialize

```
roots = []
rowlist = []
```

and then iterate for $x =$ `intsqrt(N)+2`, `intsqrt(N)+3`, $\ldots$, and for each value of x,

- if $x \cdot x - N$ can be factored completely over *primeset*,
  - append x to `roots`,
  - append to `rowlist` the vector corresponding to the factors of $x \cdot x - N$

continuing until at least *len(primeset)*$+1$ roots and vectors have been accumulated.

Try out your procedure on $N = 2419$ by calling `find_candidates(N, primes(32))`.

Here's a summary of the result of this computation:

x	x^2-N	factored	result of `dumb_factor`	vector.`f`
51	182	$2 \cdot 7 \cdot 13$	$[(2,1),(7,1),(13,1)]$	$\{\texttt{2 : one, 13 : one, 7 : one}\}$
52	285	$3 \cdot 5 \cdot 19$	$[(3,1),(5,1),(19,1)]$	$\{\texttt{19 : one, 3 : one, 5 : one}\}$
53	390	$2 \cdot 3 \cdot 5 \cdot 13$	$[(2,1),(3,1),(5,1),(13,1)]$	$\{\texttt{2 : one, 3 : one, 5 : one, 13 : one}\}$
58	945	$3^3 \cdot 5 \cdot 7$	$[(3,3),(5,1),(7,1)]$	$\{\texttt{3 : one, 5 : one, 7 : one}\}$
61	1302	$2 \cdot 3 \cdot 7 \cdot 31$	$[(2,1),(3,1),(7,1),(31,1)]$	$\{\texttt{31 : one, 2 : one, 3 : one, 7 : one}\}$
62	1425	$3 \cdot 5^2 \cdot 19$	$[(3,1),(5,2),(19,1)]$	$\{\texttt{19 : one, 3 : one, 5 : 0}\}$
63	1550	$2 \cdot 5^2 \cdot 31$	$[(2,1),(5,2),(31,1)]$	$\{\texttt{2 : one, 5 : 0, 31 : one}\}$
67	2070	$2 \cdot 3^2 \cdot 5 \cdot 23$	$[[(2,1),(3,2),(5,1),(23,1)]$	$\{\texttt{2 : one, 3 : 0, 5 : one, 23 : one}\}$
68	2205	$3^2 \cdot 5 \cdot 7^2$	$[(3,2),(5,1),(7,2)]$	$\{\texttt{3 : 0, 5 : one, 7 : 0}\}$
71	2622	$2 \cdot 3 \cdot 19 \cdot 23$	$[(2,1),(3,1),(19,1),(23,1)]$	$\{\texttt{19 : one, 2 : one, 3 : one, 23 : one}\}$
77	3510	$2 \cdot 3^3 \cdot 5 \cdot 13$	$[(2,1),(3,3),(5,1),(13,1)]$	$\{\texttt{2 : one, 3 : one, 5 : one, 13 : one}\}$
79	3822	$2 \cdot 3 \cdot 7^2 \cdot 13$	$[(2,1),(3,1),(7,2),(13,1)]$	$\{\texttt{2 : one, 3 : one, 13 : one, 7 : 0}\}$

Thus, after the loop completes, the value of `roots` should be the list

$$[51, 52, 53, 58, 61, 62, 63, 67, 68, 71, 77, 79]$$

and the value of `rowlist` should be the list

```
[Vec({2,3,5, ..., 31},{2:  one, 13:  one, 7:  one}),
                          ⋮
Vec({2,3,5, ...  , 31},{2:  one, 3:  one, 5:  one, 13:  one}),
 Vec({2,3,5, ..., 31}, {2:  one, 3:  one, 13:  one, 7:  0})]
```

Now we use the results to find a nontrivial divisor of N.

Examine the table rows corresponding to 53 and 77. The factorization of $53 * 53 - N$ is $2 \cdot 3 \cdot 5 \cdot 13$. The factorization of $77 * 77 - N$ is $2 \cdot 3^3 \cdot 5 \cdot 13$. Therefore the factorization of the product $(53 * 53 - N)(77 * 77 - N)$ is

$$(2 \cdot 3 \cdot 5 \cdot 13)(2 \cdot 3^3 \cdot 5 \cdot 13) = 2^2 \cdot 3^4 \cdot 5^2 \cdot 13^2$$

Since the exponents are all even, the product is a perfect square: it is the square of

$$2 \cdot 3^2 \cdot 5 \cdot 13$$

Thus we have derived

$$\begin{aligned} (53^2 - N)(77^2 - N) &= (2 \cdot 3^2 \cdot 5 \cdot 13)^2 \\ 53^2 \cdot 77^2 - kN &= (2 \cdot 3^2 \cdot 5 \cdot 13)^2 \\ (53 \cdot 77)^2 - kN &= (2 \cdot 3^2 \cdot 5 \cdot 13)^2 \end{aligned}$$

Task 7.8.8: To try to find a factor, let $a = 53 \cdot 77$ and let $b = 2 \cdot 3^2 \cdot 5 \cdot 13$, and compute $\text{gcd}(a - b, N)$. Did you find a proper divisor of N?

Similarly, examine the table rows corresponding to 52, 67, and 71. The factorizations of $x * x - N$ for these values of x are

$$\begin{array}{c} 3 \cdot 5 \cdot 19 \\ 2 \cdot 3^2 \cdot 5 \cdot 23 \\ 2 \cdot 3 \cdot 19 \cdot 23 \end{array}$$

Therefore the factorization of the product $(52 * 52 - N)(67 * 67 - N)(71 * 71 - N)$ is

$$(3 \cdot 5 \cdot 19)(2 \cdot 3^2 \cdot 5 \cdot 23)(2 \cdot 3 \cdot 19 \cdot 23) = 2^2 \cdot 3^4 \cdot 5^2 \cdot 19^2 \cdot 23^2$$

which is again a perfect square; it is the square of

$$2 \cdot 3^2 \cdot 5 \cdot 19 \cdot 23$$

Task 7.8.9: To again try to find a factor of N (just for practice), let $a = 52 \cdot 67 \cdot 71$ and let $b = 2 \cdot 3^2 \cdot 5 \cdot 19 \cdot 23$, and compute $\text{gcd}(a - b, N)$. Did you find a proper divisor of N?

How did I notice that the rows corresponding to 52, 67, and 71 combine to provide a perfect square? That's where the linear algebra comes in. The sum of the vectors in these rows is the zero vector. Let A be the matrix consisting of these rows. Finding a nonempty set of rows of A whose $GF(2)$ sum is the zero vector is equivalent, by the linear-combinations definition of vector-matrix multiplication, to finding a nonzero vector $\boldsymbol{v}$ such that $\boldsymbol{v} * A$ is the zero vector. That is, $\boldsymbol{v}$ is a nonzero vector in the null space of A^T.

How do I know such a vector exists? Each vector in `rowlist` is a `primeset`-vector and so lies in a K-dimensional space where $K = \textit{len(primelist)}$. Therefore the rank of these vectors is at most K. But `rowlist` consists of at least $K+1$ vectors. Therefore the rows are linearly dependent.

How do I find such a vector? When I use Gaussian elimination to transform the matrix into echelon form, the last row is guaranteed to be zero.

More specifically, I find a matrix M representing a transformation that reduced the vectors in `rowlist` to echelon form. The last row of M, multiplied by the original matrix represented by `rowlist`, yields the last row of the matrix in echelon form, which is a zero vector.

To compute M, you can use the procedure `transformation_rows(rowlist_input)` defined in the module `echelon` we provide. Given a matrix A (represented by as a list `rowlist_input` of rows), this procedure returns a matrix M (also represented as a list of rows) such that MA is in echelon form.

Since the last row of MA must be a zero vector, by the vector-matrix definition of matrix-vector multiplication, the last row of M times A is the zero vector. By the linear-combinations definition of vector-matrix multiplication, the zero vector is a linear combination of the rows of A where the coefficients are given by the entries of the last row of M. The last row of M is

```
Vec({0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11},{0: 0, 1: one, 2: one, 4: 0,
     5: one, 11: one})
```

Note that entries 1, 2, 5, and 11 are nonzero, which tells us that the sum of the corresponding rows of `rowlist` is the zero vector. That tells us that these rows correspond to the factorizations of numbers whose product is a perfect square. The numbers are: 285, 390, 1425, and 3822. Their product is 605361802500, which is indeed a perfect square: it is the square of 778050. We therefore set $b = 778050$. We set a to be the product of the corresponding values of x (52, 53, 62, and 79), which is 139498888. The greatest common divisor of $a - b$ and N is, uh, 1. Oops, we were unlucky–it didn't work.

Was all that work for nothing? It turns out we were not so unlucky. The rank of the matrix A could have been *len(rowlist)* but turned out to be somewhat less. Consequently, the second-to-last row of MA is also a zero vector. The second-to-last row of M is

```
Vec({0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11},{0: 0, 1: 0, 10: one, 2: one})
```

Note that entries 10 and 2 are nonzero, which tells us that combining row 2 of `rowlist` (the row corrresponding to 53) with row 10 of `rowlist` (the row corresponding to 77) will result in a perfect square.

Task 7.8.10: Define a procedure `find_a_and_b(v, roots, N)` that, given a vector $\boldsymbol{v}$ (one of the rows of M), the list `roots`, and the integer N to factor, computes a pair (a, b) of integers such that $a^2 - b^2$ is a multiple of N.

Your procedure should work as follows:

- Let `alist` be the list of elements of `roots` corresponding to nonzero entries of the vector $\boldsymbol{v}$. (Use a comprehension.)
- Let a be the product of these. (Use the procedure `prod(alist)` defined in the module `factoring_support`.)
- Let c be the product of $\{x \cdot x - N : x \in \texttt{alist}\}$.
- Let `b` be `intsqrt(c)`.
- Verify using an assertion that `b*b == c`
- Return the pair (a, b).

Try out your procedure with $\boldsymbol{v}$ being the last row of M. See if $a-b$ and N have a nontrivial common divisor. If it doesn't work, try it with $\boldsymbol{v}$ being the second-to-last row of M, etc.

Finally, you will try the above strategy on larger integers.

Task 7.8.11: Let $N = 2461799993978700679$, and try to factor N

- Let *primelist* be the set of primes up to 10000.
- Use `find_candidates(N, primelist)` to compute the lists `roots` and `rowlist`.
- Use `echelon.transformation_rows(rowlist)` to get a matrix M.
- Let $\boldsymbol{v}$ be the last row of M, and find a and b using `find_a_and_b(v, roots, N)`.
- See if $a - b$ has a nontrivial common divisor with N. If not, repeat with $\boldsymbol{v}$ being the second-to-last row of M or the third-to-last row....

Give a nontrivial divisor of N.

Task 7.8.12: Let $N = 20672783502493917028427$, and try to factor N. This time, since N is a lot bigger, finding $K + 1$ rows will take a lot longer, perhaps six to ten minutes depending on your computer. Finding M could take a few minutes.

Task 7.8.13: Here is a way to speed up finding M: The procedure `echelon.transformation_rows` takes an optional second argument, a list of column-labels. The list instructs the procedure in which order to handle column-labels. The procedure works much faster if the list consists of the primes of `primelist` in descending order:

```
>>> M_rows = echelon.transformation_rows(rowlist,
                                          sorted(primelist, reverse=True))
```

Why should the order make a difference? Why does this order work well? *Hint:* a large prime is less likely than a small prime to belong to the factorization of an integer.

7.9 Review questions

- What is echelon form?
- What can we learn about the rank of a matrix in echelon form?
- How can a matrix be converted into echelon form by multiplication by an invertible matrix?
- How can Gaussian elimination be used to find a basis for the null space of a matrix?
- How can Gaussian elimination be used to solve a matrix-vector equation when the matrix is invertible?