31. (expires 4/17) When we implemented RSA in class, we represented our encrypted messages as a list of large numbers, rather than converting them to printable text. Sometimes we want a text representation. One way to do this to use a base-64 representation, where the message $m$ is converted to a base 64 number. This base 64 number is commonly represented with the upper-case characters A–Z representing digits 0 through 25, lower-case a–z representing digits 26 through 51, the characters 0–9 representing 52 through 61, and + and / representing 62 and 63, respectively.

If the message is longer than 64 characters, the encoded line is broken there (i.e., a newline is inserted). In some implementations, padding characters (usually =) are also added to ensure that the encoded text is of a length divisible by 4 (if the input is base 256 ASCII, this means three input characters correspond to four encoded characters). There are several variations of the base-64 encoding in common use.

Write a generalized implementation of this conversion process. Specifically, assume there is a global called AlphabetOut which contains the allowed characters in the encoding, ordered appropriately. Your procedure should take as input two arguments: a list of numbers in base $n$, and the base $n$. Your procedure should return a string representing the message in base $b$, where $b$ is the length of AlphabetOut. Also write another procedure which undoes this conversion. Don’t worry about inserting padding characters.

As an example, the following list of numbers represents some text converted from ASCII (base 256) to base 10$^4$ (that is, without encryption):

```
6669013858395040150291124122141963456189571137, 7908578519506220727272120198112297549231814082,
15624867350454934942834586656586375686490456, 5538768361177927030468952558428915355239550393,
2389695761146543113603100420161706746881540779, 78091587231828327640146863695263953922972490912,
9764606424846784132731915166644883269708742761,
```

When transcribed into the base 64 encoding described above, we get:

```
BBCbv52ZQxatVGih2dsBSauBSYgcWYsFGE5Bi2hJHLgYWyYBSY3FWeu4iLKoQS
0BSazBSYgAXZyl2bkBybnByp2XasBydhJnLgIVZiVGbogcwF2Y1NHapB3sAyc
0JXarlmbnBiZy9WbgEGIo1GZkVmbgogYhNXZsACahZXZgc3buBCdoVWayBiZpJ3c
0BldpNG3vJXegoqYNFWauNHdgQHalBSZ2lGbgcUYsF2Y012YgUUbwlmc15C
```

By the way, this text is from the opening to a well-known movie.

32. (expires 4/17) The procedures StringToKgraph and KgraphToString as defined in Crypto.mw have the following defect. Let $\alpha$ represent the first character of the Alphabet. Then any occurrences of $\alpha$ that appear at the end of a string are lost when converting to $k$-graphs and back.

For this problem, you should think of a way to fix this issue. You need to both implement and explain your solution.

As a concrete example, suppose we use the 10-letter alphabet *123456789. Then the command StringToKgraph("*12***98*456****",3) gives the result [210, 0, 89, 654], which KgraphToString converts back to *12**98*456.

There are a number of ways to solve this issue. Think of one, implement it, and explain why your way works, including some examples.

---

2Using Maple’s convention of least-significant digit first, so the decimal number 123 is [59,1] (or 7B) in base 64.