# MTH 461 H1

## Exercise 1.

▶ see problem

1. AB

For matrix multiplication AB, we have

$$[AB]_{ik} = \sum_{j \in J} a_{ij} b_{jk}$$

yielding

| $AB$ | $0$ | $1$ | $2$ |
|---|---|---|---|
| $0$ | $1$ | $0$ | $0$ |
| $1$ | $1$ | $2$ | $2$ |

For parts 2-4 treat each tuple like a matrix

2. Au

$$Au = \begin{matrix} 0 & 1 \\ 1 & 2 \end{matrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

3. vB

$$vB = \begin{matrix} 0 & 1 & 2 \\ 1 & 1 & 2 \end{matrix} = \begin{pmatrix} 1 \\ 1 \\ 2 \end{pmatrix}$$

4. vu

$$vu = uv = 1 * 1 + 1 * 0 + 0 * 1 + 1 * 0 = 1$$

## Exercise 2.

▶ see problem

We will solve this exhaustively by shifting the letters backwards 1 to 25 places

In [ ]:
```
#encoded = 'AB XYZ'
encoded = 'QY QBOOX QY GRSDO'
```

```
for k in range(-25, 0):

    decoded = str()
    for symbol in encoded:
        if symbol == ' ':
            decoded += symbol

        elif ord(symbol)+k < ord('A'):
            decoded += chr(ord(symbol)+k+26)

        else:
            decoded += chr(ord(symbol)+k)

    print(-k, decoded)
```

```
25 RZ RCPPY RZ HSTEP
24 SA SDQQZ SA ITUFQ
23 TB TERRA TB JUVGR
22 UC UFSSB UC KVWHS
21 VD VGTTC VD LWXIT
20 WE WHUUD WE MXYJU
19 XF XIVVE XF NYZKV
18 YG YJWWF YG OZALW
17 ZH ZKXXG ZH PABMX
16 AI ALYYH AI QBCNY
15 BJ BMZZI BJ RCDOZ
14 CK CNAAJ CK SDEPA
13 DL DOBBK DL TEFQB
12 EM EPCCL EM UFGRC
11 FN FQDDM FN VGHSD
10 GO GREEN GO WHITE
9 HP HSFFO HP XIJUF
8 IQ ITGGP IQ YJKVG
7 JR JUHHQ JR ZKLWH
6 KS KVIIR KS ALMXI
5 LT LWJJS LT BMNYJ
4 MU MXKKT MU CNOZK
3 NV NYLLU NV DOPAL
2 OW OZMMV OW EPQBM
1 PX PANNW PX FQRCN
```

Since the only sensible message is "GO GREEN GO WHITE," the k-value is 10. We can verify by encoding this text with k=10 and see if it yields the encoded message

In [ ]:
```
encoded = 'GO GREEN GO WHITE'
k = 10

decoded = str()
for symbol in encoded:
    if symbol == ' ':
        decoded += symbol
    elif ord(symbol)+k > ord('Z'):
        decoded += chr(ord(symbol)+k-26)
    else:
        decoded += chr(ord(symbol)+k)
```

```
print(k, decoded)
```

```
10 QY QBOOX QY GRSDO
```

# Exercise 3.

▶ see problem

The coding rule used converts a word from the alphabet $\mathbb{A}$ to base 10 according to their usual order in the alphabet, then represents those numbers as binary. We can run this process in reverse to return the original letters

In [ ]:
```
encoded = [0b00111, 0b01111, 0b01111, 0b00100, 0b11011, 0b01100, 0b10101, 0b00011,
print(encoded)
decoded = str()
for symbol in encoded:
    decoded += chr(symbol + ord('A')-1)
print(decoded)
```

```
[7, 15, 15, 4, 27, 12, 21, 3, 11]
GOOD[LUCK
```

The fifth symbol, 11011, is the binary representation of 27, which represents the 27th symbol of the alphabet $\mathbb{A}$, the space. As such, the actual decoded result is "good luck"

# Exercise 4.

▶ see problem

I overheard this one walking out of class on wednesday. Consider an alphabet $\mathbb{X}$ with only a single symbol $x$. Then the irregular code $c : \mathbb{X} \to T^*$ such that

$$c(x) = \emptyset$$

Thus every codeword has length 0. Then the extension is not one-to-one

$$c^*(xxx\ldots) = \emptyset = c(x)$$

and the code is not uniquely decodeable.

In other circumstances this code is UD. Since each codeword has some length k, a message of length n will be encoded into a codeword of length n*k, which can be separated into n blocks of length k and decoded if one knows the injective code that was used.

# Exercise 5.

▶ see problem

$c$ is uniquely decodable.

This is because while $c(1) = 0$ is a prefix of $c(0) = \underline{0}1$, each $1$ reveals that the previous $0$ isn't the result of encoding a $1$, essentially allowing the code

$$0 \mapsto 01, \quad 1 \mapsto 0$$

to become

$$0 \mapsto 1, \quad 1 \mapsto 0$$

which is trivially UD.

The algorithm below uses this to uniquely decode any codeword

Consider any codeword $c^*(s_1 s_2 \dots s_n) = c(s_1)c(s_2)\dots c(s_n) = cw$. To decode the concatenation, create a message $|s'| = |cw|, s' = 111\dots$, and for each $\{1 \leq i \leq n | cw_i = 1\}$ replace $s'_i$ with $0$. Then remove all $s'_{i-1}$ (without updating index locations). Then $s' = s_1 s_2 \dots s_n$

# Exercise 6.

▶ see problem

6. No. One cannot discriminate $c(H)$ from $c(S)c(E)$ since

$$c(H) = \cdot\cdot\cdot\cdot$$

and

$$c(S) + c(E) = \cdot\cdot\cdot + \cdot = \cdot\cdot\cdot\cdot = c(H)$$