EE 599 Homework 3

Neema Badihian

In [1]:
```
import numpy as np
import itertools
```

In [2]:
```
mat = [[1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1],
    [1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1],
        [1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1],
        [1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0],
        [0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0],
        [0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0],
        [0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1],
        [1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0],
        [0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1],
        [1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0]]
```

In [3]:
```
def kones(n, k):
    result = []
    for bits in itertools.combinations(range(n), k):
        s = [0] * n
        for bit in bits:
            s[bit] = 1
        result.append(s)
    return result
```

In [4]:
```
def reduce_mat(mat, rows):
    for i in range(len(mat)):
        combos = kones(rows, i)
        for combo in combos:
            stab_prod = []
            prod_count = 0
            for j in range(len(combo)):
                if prod_count == 0 and combo[j] == 1:
                    stab_prod = mat[j]
                    prod_count += 1
                elif combo[j] == 1:
                    new_prod = (np.array(stab_prod) + np.array(mat[j])) % 2
                    stab_prod = list(new_prod)
                    prod_count += 1
            if prod_count > 1 and stab_prod in mat:
                print(f"combo = {combo}")
                print(f"Stabilizer Product = {stab_prod}")
                print(f"Stabilizer Index in Matrix = {mat.index(stab_prod)}")
                print()
                return mat.index(stab_prod)

    return -1
```

In [5]:
```
mat_indices = []
found_reduction = True
rows = 10
while found_reduction == True:
    mat_index = reduce_mat(mat, rows)
    if mat_index > -1:
        mat_indices.append(mat_index)
        mat.pop(mat_index)
        rows -= 1
    else:
        found_reduction = False
```

```
combo = [1, 1, 1, 0, 1, 1, 0, 0, 1, 0]
Stabilizer Product = [1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0]
Stabilizer Index in Matrix = 9
```

In [6]:
```
print(f"New Matrix =")
for row in mat:
    print(row)
print()
```

```
New Matrix =
[1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1]
[1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1]
[1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1]
[1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0]
[0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0]
[0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0]
[0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1]
[1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
[0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1]
```

k = 21 - (9*2) = 3

In [7]:
```
def does_commute(op1, op2):
    op_sum = list(np.array(op1) + np.array(op2))
    count = op_sum.count(2)
    return count % 2 == 0
```

In [8]:
```
def find_logical_x(zlogicals, xlogicals, error):
    commutes = True
    for zlog in zlogicals:
        if does_commute(error, zlog) == False:
            commutes = False
            break
    if commutes and error not in xlogicals:
        return error
    return None
```

In [9]:

```
def find_logical_z(xlogical, zlogicals, error):
    if does_commute(error, xlogical) == False and error not in zlogicals:
        return error
    return None
```

In [10]:

```
def find_commute_with_stab(stabilizers, errors):
    new_errors = []
    for error in errors:
        commutes = True
        for stab in stabilizers:
            if does_commute(error, stab) == False:
                commutes = False
                break
        if commutes:
            new_errors.append(error)
    return new_errors
```

In [11]:

```
xlogicals = []
xlogical_groups = []
zlogicals = []
zlogical_groups = []
x_errors = kones(21, 5)
x_errors2 = kones(21, 6)
x_errors = x_errors + x_errors2
z_errors = kones(21, 5)
z_errors2 = kones(21, 6)
z_errors = z_errors + z_errors2

x_errors_stabilizers = find_commute_with_stab(mat, x_errors)
z_errors_stabilizers = find_commute_with_stab(mat, z_errors)
x_errors = x_errors_stabilizers
z_errors = z_errors_stabilizers
```

In [12]:

```
for i in range(len(x_errors)):
    if x_errors[i].count(1) > 5:
        break
    xlogical1 = find_logical_x(zlogicals, xlogicals, x_errors[i])

    if xlogical1 != None:
        all_x = xlogicals + [xlogical1]
        print(len(all_x))
        for j in range(len(z_errors)):
            zlogical1 = find_logical_z(xlogical1, zlogicals, z_errors[j])

            if zlogical1 != None:
                all_z = zlogicals + [zlogical1]
                for k in range(i, len(x_errors)):
                    xlogical2 = find_logical_x(all_z, all_x, x_errors[k])

                    if xlogical2 != None:
                        all_x = xlogicals + [xlogical1] + [xlogical2]
                        for l in range(j, len(z_errors)):
                            zlogical2 = find_logical_z(xlogical2, all_z, z_errors[l]]

                            if zlogical2 != None:
                                all_z = zlogicals + [zlogical1] + [zlogical2]
                                for q in range(k, len(x_errors)):
                                    xlogical3 = find_logical_x(all_z, all_x, x_errors

                                    if xlogical3 != None:
                                        all_x = xlogicals + [xlogical1] + [xlogical2]
                                        for r in range(l, len(z_errors)):
                                            zlogical3 = find_logical_z(xlogical3, all
                                            if zlogical3 != None:
                                                zlogicals = all_z + [zlogical3]
                                                xlogicals = all_x
                                                xlogical_groups.append([xlogical1, xl
                                                zlogical_groups.append([zlogical1, zl
```

```
1
```

In [13]:

```
print(f"Number of logical operator sets = {len(xlogical_groups)}")
for i in range(5):
    print(f"XL1 = {xlogical_groups[i][0]}, weight = {xlogical_groups[i][0].count(1)}"
    print(f"ZL1 = {zlogical_groups[i][0]}, weight = {zlogical_groups[i][0].count(1)}"
    print(f"XL2 = {xlogical_groups[i][1]}, weight = {xlogical_groups[i][1].count(1)}"
    print(f"ZL2 = {zlogical_groups[i][1]}, weight = {zlogical_groups[i][1].count(1)}"
    print(f"XL3 = {xlogical_groups[i][2]}, weight = {xlogical_groups[i][2].count(1)}"
    print(f"ZL3 = {zlogical_groups[i][2]}, weight = {zlogical_groups[i][2].count(1)}"
    print()
```

```
Number of logical operator sets = 6216
XL1 = [1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0], weight = 5
ZL1 = [1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0], weight = 5
XL2 = [1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0], weight = 6
ZL2 = [1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1], weight = 6
XL3 = [1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1], weight = 6
ZL3 = [1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0], weight = 6
```

```
XL1 = [1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0], weight = 5
ZL1 = [1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0], weight = 5
XL2 = [1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0], weight = 6
ZL2 = [1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1], weight = 6
XL3 = [1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1], weight = 6
ZL3 = [1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1], weight = 6

XL1 = [1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0], weight = 5
ZL1 = [1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0], weight = 5
XL2 = [1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0], weight = 6
ZL2 = [1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1], weight = 6
XL3 = [1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1], weight = 6
ZL3 = [1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0], weight = 6

XL1 = [1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0], weight = 5
ZL1 = [1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0], weight = 5
XL2 = [1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0], weight = 6
ZL2 = [1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1], weight = 6
XL3 = [1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1], weight = 6
ZL3 = [1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0], weight = 6

XL1 = [1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0], weight = 5
ZL1 = [1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0], weight = 5
XL2 = [1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0], weight = 6
ZL2 = [1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1], weight = 6
XL3 = [1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1], weight = 6
ZL3 = [1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0], weight = 6
```

Distance = 5

[[n, k, d]] = [[21, 3, 5]]

In [14]:

```
x_expression = [0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1]
z_expression = [0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0]
z_exp_syndrome = [1 if does_commute(x_expression, stab) else 0 for stab in mat]
x_exp_syndrome = [1 if does_commute(z_expression, stab) else 0 for stab in mat]

results = []
solution_found = False

for weight in range(2,21):
    errors = kones(21, weight)
    for xerror in errors:
        z_err_syndrome = [1 if does_commute(xerror, stab) else 0 for stab in mat]
        if z_exp_syndrome == z_err_syndrome:
            for zerror in errors:
                x_err_syndrome = [1 if does_commute(zerror, stab) else 0 for stab in
                if x_exp_syndrome == x_err_syndrome:
                    x_log_operator = list((np.array(xerror) + np.array(x_expression)
                    z_log_operator = list((np.array(zerror) + np.array(z_expression)
                    results.append((xerror.count(1), xerror, zerror.count(1), zerror
                    solution_found = True
    if solution_found:
        break
```

In [15]:

```
for result in results:
    print(f"X Error = {result[1]}, Weight = {result[0]}")
    print(f"Z Error = {result[3]}, Weight = {result[2]}")
    print(f"X Operator = {result[5]}, Weight = {result[4]}")
    print(f"Z Operator = {result[7]}, Weight = {result[6]}")
    print()
```

```
X Error = [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0], Weight = 2
Z Error = [1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], Weight = 2
X Operator = [0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1], Weight
= 9
Z Operator = [1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0], Weight
= 9
```

In [ ]: