Project Part-II

CSM-322: Information and Coding Theory

October 31, 2022

Gandharv Jain
20124018

Question 1, 2, 3

```python
import numpy as np
q = 2

def modInv(a):
	for x in range(1, q):
		if (a * x) % q == 1: return x

def swapCol(A, i, j):
	A[:, i], A[:, j] = A[:, j], A[:, i].copy()

def rref(G):
	m, n = G.shape
	i, j = 0, 0

	rank = 0
	while i < m and j < n:
		t = np.argmax(G[i:m, j]) + i
		p = G[t, j]
		if p == 0:
			j += 1
			continue

		rank += 1
		if i != t:
			G[[i, t]] = G[[t, i]]

		G[i] *= modInv(G[i, j])
		G[i] %= q

		for l in range(m):
			if l != i:
				G[l] -= G[l, j] * G[i]
				G[l] %= q

		i += 1
		j += 1
	return G[:rank]

def generatorToParityCheck(G):
	swaps_seq = []
	k, n = G.shape
	j = 0
	for i in range(k):
		if G[i, i] != 1:
			while j < n:
				j += 1
				if G[i, j] == 1:
					break
			swapCol(G, i, j)
			swaps_seq.append((i, j))

	X = G[:,k:n]
	H = np.concatenate((-1*X.T % q, np.identity(n - k, int)), axis=1)
	for s in reversed(swaps_seq):
		swapCol(H, s[0], s[1])
	return H
```

```python
def main():
        S = input("Enter codewords (space-seperated): ").strip().split()
        G = np.array([list(c) for c in S], int)
        G = rref(G)
        print(f'G = \n{G}\n')
        H = generatorToParityCheck(G)
        print(f'H = \n{H}\n')
        return G, H

if __name__ == '__main__':
        main()
```

```
Input: 11101 10110 01011 11010
Output:
    G =
    [[1 0 0 0 1]
     [0 1 0 1 1]
     [0 0 1 1 1]]

    H =
    [[0 1 1 1 0]
     [1 1 1 0 1]]
Input: 1101 0111 1111 1000
Output:
    G =
    [[1 0 0 0]
     [0 1 0 1]
     [0 0 1 0]]

    H =
    [[0 1 0 1]]
```

Question 4

```python
import numpy as np
q = 2
G = np.array([
        [1, 0, 0, 0, 1],
        [0, 1, 0, 0, 1],
        [0, 0, 1, 1, 1]
        ])

print("All the codewords of C are:")
for a_0 in range(q):
        for a_1 in range(q):
                for a_2 in range(q):
                        print((G[0] * a_0 + G[1] * a_1 + G[2] * a_2) % q)
```

```
Output:
    [0 0 0 0 0]
    [0 0 1 1 1]
    [0 1 0 0 1]
    [0 1 1 1 0]
    [1 0 0 0 1]
    [1 0 1 1 0]
    [1 1 0 0 0]
    [1 1 1 1 1]
```

Question 5

```
import numpy as np
q = 2
H = np.array([
        [1, 0],
        [1, 1],
        [0, 1],
        [1, 1]
        ])
C = np.array([
        [0, 0, 0, 0],
        [1, 1, 1, 0],
        [1, 0, 1, 1],
        [0, 1, 0, 1],
        ])

print("The given H is not the parity check matrix as n - k cannot be greater than n but")
print("for transpose of H:")
for c in C:
        p = (c @ H) % q
        print(f'{str(c)}*H = {p}')

print("Hence, transpose of H is a parity check matrix for the given code")
```

```
Output:
    The given H is not the parity check matrix as n - k cannot be greater than n but
    for transpose of H:
    [0 0 0 0]*H = [0 0]
    [1 1 1 0]*H = [0 0]
    [1 0 1 1]*H = [0 0]
    [0 1 0 1]*H = [0 0]
    Hence, transpose of H is a parity check matrix for the given code
```

Question 6

```
import numpy as np
import A1to3_Linear as lin
q = 2

def genCodewords(codeword, indx, bits_left):
        C = []
        if bits_left > 0:
                for i in range(indx, len(codeword)):
                        c = codeword.copy()
                        c[i] = 1
                        C += genCodewords(c, i + 1, bits_left - 1)
        else:
                C.append(codeword)
        return C
```

```
def syndromeTable(H):
        S = {}
        t, n = H.shape
        k = n - t
        total_syndromes = q ** (n - k)
        weight = 0

        while len(S) < total_syndromes:
                for u in genCodewords(np.zeros(n, int), 0, weight):
                        S_u = u @ H.T
                        if tuple(S_u) not in S:
                                S[tuple(S_u)] = tuple(u)
                weight += 1
        return S

def decode(w, H_t, S_H):
        S_w = (w @ H_t) % q
        e = np.array([S_H.get(tuple(S_w))])
        u = (w - e) % q
        return u

def main():
        G, H = lin.main()
        w = np.array(list(input("Enter received word: ")), int)

        S_H = syndromeTable(H)
        sent_word = decode(w, H.T, S_H)

        print(f'Syndrome Table: {S_H}')
        print(f'Sent word: {sent_word}')

if __name__ == '__main__':
        main()
```

```
Input:
    0000 1011 0101 1110
    1101
Output:
    G =
    [[1 0 1 1]
     [0 1 0 1]]

    H =
    [[1 0 1 0]
     [1 1 0 1]]

    Syndrome Table: {(0, 0): (0, 0, 0, 0), (1, 1): (1, 0, 0, 0),
                     (0, 1): (0, 1, 0, 0), (1, 0): (0, 0, 1, 0)}
    Sent word: [[0 1 0 1]]
```

Question 7

```
import numpy as np
q = 2
```

```python
def generateHadamard(n):
        H = [None] * (n + 1)
        H[0] = np.array([1], int)
        H[1] = np.array([[1, 1], [1, -1]], int)
        hadamard(H, n)
        return H

def hadamard(H, n):
        t = n - 1
        l = 2 ** t
        if H[n] is not None:
                return
        if H[t] is None:
                hadamard(H, t)

        H[n] = np.zeros((2**n, 2**n), int)
        H[n][:l, :l] = H[t]
        H[n][:l, l:] = H[t]
        H[n][l:, :l] = H[t]
        H[n][l:, l:] = -1 * H[t]

def printCodewords(H_n):
        C = []
        H_n[H_n == -1] = 0
        for c in H_n:
                C.append(np.array2string(c))

        H_n[H_n == 0] = -1
        H_n[H_n == 1] = 0
        H_n[H_n == -1] = 1
        for c in H_n:
                C.append(np.array2string(c))
        print(C)

def main():
        n = int(input("Enter value of n in 2^n: "))
        H = generateHadamard(n)
        print(f'Hadamard matrix of order 2^n: \n{H[n]}')
        printCodewords(H[n])

if __name__ == '__main__':
        main()
```

```
Input: 3
Output:
    Hadamard matrix of order 2^n:
    [[ 1  1  1  1  1  1  1  1]
     [ 1 -1  1 -1  1 -1  1 -1]
     [ 1  1 -1 -1  1  1 -1 -1]
     [ 1 -1 -1  1  1 -1 -1  1]
     [ 1  1  1  1 -1 -1 -1 -1]
     [ 1 -1  1 -1 -1  1 -1  1]
     [ 1  1 -1 -1 -1 -1  1  1]
     [ 1 -1 -1  1 -1  1  1 -1]]
    ['[1 1 1 1 1 1 1 1]', '[1 0 1 0 1 0 1 0]', '[1 1 0 0 1 1 0 0]', '[1 0 0 1 1 0 0 1]',
     '[1 1 1 1 0 0 0 0]', '[1 0 1 0 0 1 0 1]', '[1 1 0 0 0 0 1 1]', '[1 0 0 1 0 1 1 0]',
     '[0 0 0 0 0 0 0 0]', '[0 1 0 1 0 1 0 1]', '[0 0 1 1 0 0 1 1]', '[0 1 1 0 0 1 1 0]',
     '[0 0 0 0 1 1 1 1]', '[0 1 0 1 1 0 1 0]', '[0 0 1 1 1 1 0 0]', '[0 1 1 0 1 0 0 1]']
```