In [15]:
```
import numpy as np
import inspect

def f_0(a, b): #s = 10
    no_args = 2
    if(a==0 and b==0):
        return format(1, '#0'+str(no_args+2)+'b')
    if(a==0 and b==1):
        return format(3, '#0'+str(no_args+2)+'b')
    if(a==1 and b==0):
        return format(1, '#0'+str(no_args+2)+'b')
    if(a==1 and b==1):
        return format(3, '#0'+str(no_args+2)+'b')

def f_1(a, b): #s = 11
    no_args = 2
    if(a==0 and b==0):
        return format(2, '#0'+str(no_args+2)+'b')
    if(a==0 and b==1):
        return format(3, '#0'+str(no_args+2)+'b')
    if(a==1 and b==0):
        return format(3, '#0'+str(no_args+2)+'b')
    if(a==1 and b==1):
        return format(2, '#0'+str(no_args+2)+'b')

def f_2(a, b, c): #s = 110
    no_args = 3
    if(a==0 and b==0 and c==0):
        return format(5, '#0'+str(no_args+2)+'b')
    if(a==0 and b==0 and c==1):
        return format(2, '#0'+str(no_args+2)+'b')
    if(a==0 and b==1 and c==0):
        return format(0, '#0'+str(no_args+2)+'b')
    if(a==0 and b==1 and c==1):
        return format(6, '#0'+str(no_args+2)+'b')
    if(a==1 and b==0 and c==0):
        return format(0, '#0'+str(no_args+2)+'b')
    if(a==1 and b==0 and c==1):
        return format(6, '#0'+str(no_args+2)+'b')
    if(a==1 and b==1 and c==0):
        return format(5, '#0'+str(no_args+2)+'b')
    if(a==1 and b==1 and c==1):
        return format(2, '#0'+str(no_args+2)+'b')


def f_3(a, b, c): #s = 010
    no_args = 3
    if(a==0 and b==0 and c==0):
        return format(5, '#0'+str(no_args+2)+'b')
    if(a==0 and b==0 and c==1):
        return format(2, '#0'+str(no_args+2)+'b')
    if(a==0 and b==1 and c==0):
        return format(5, '#0'+str(no_args+2)+'b')
    if(a==0 and b==1 and c==1):
        return format(2, '#0'+str(no_args+2)+'b')
    if(a==1 and b==0 and c==0):
```

```
        return format(0, '#0'+str(no_args+2)+'b')
    if(a==1 and b==0 and c==1):
        return format(6, '#0'+str(no_args+2)+'b')
    if(a==1 and b==1 and c==0):
        return format(0, '#0'+str(no_args+2)+'b')
    if(a==1 and b==1 and c==1):
        return format(6, '#0'+str(no_args+2)+'b')

def f_4(a, b, c): #s = 111
    no_args = 3
    if(a==0 and b==0 and c==0):
        return format(5, '#0'+str(no_args+2)+'b')
    if(a==0 and b==0 and c==1):
        return format(2, '#0'+str(no_args+2)+'b')
    if(a==0 and b==1 and c==0):
        return format(6, '#0'+str(no_args+2)+'b')
    if(a==0 and b==1 and c==1):
        return format(0, '#0'+str(no_args+2)+'b')
    if(a==1 and b==0 and c==0):
        return format(0, '#0'+str(no_args+2)+'b')
    if(a==1 and b==0 and c==1):
        return format(6, '#0'+str(no_args+2)+'b')
    if(a==1 and b==1 and c==0):
        return format(2, '#0'+str(no_args+2)+'b')
    if(a==1 and b==1 and c==1):
        return format(5, '#0'+str(no_args+2)+'b')

def f_5(a, b, c, d): #s = 1010
    no_args = 4
    if(a==0 and b==0 and c==0 and d==0):
        return format(1, '#0'+str(no_args+2)+'b')
    if(a==0 and b==0 and c==0 and d==1):
        return format(2, '#0'+str(no_args+2)+'b')

    if(a==0 and b==0 and c==1 and d==0):
        return format(3, '#0'+str(no_args+2)+'b')
    if(a==0 and b==0 and c==1 and d==1):
        return format(4, '#0'+str(no_args+2)+'b')

    if(a==0 and b==1 and c==0 and d==0):
        return format(5, '#0'+str(no_args+2)+'b')
    if(a==0 and b==1 and c==0 and d==1):
        return format(6, '#0'+str(no_args+2)+'b')

    if(a==0 and b==1 and c==1 and d==0):
        return format(7, '#0'+str(no_args+2)+'b')
    if(a==0 and b==1 and c==1 and d==1):
        return format(8, '#0'+str(no_args+2)+'b')

    if(a==1 and b==0 and c==0 and d==0):
        return format(3, '#0'+str(no_args+2)+'b')
    if(a==1 and b==0 and c==0 and d==1):
        return format(4, '#0'+str(no_args+2)+'b')

    if(a==1 and b==0 and c==1 and d==0):
        return format(1, '#0'+str(no_args+2)+'b')
    if(a==1 and b==0 and c==1 and d==1):
```

```
        return format(2, '#0'+str(no_args+2)+'b')

    if(a==1 and b==1 and c==0 and d==0):
        return format(7, '#0'+str(no_args+2)+'b')
    if(a==1 and b==1 and c==0 and d==1):
        return format(8, '#0'+str(no_args+2)+'b')

    if(a==1 and b==1 and c==1 and d==0):
        return format(5, '#0'+str(no_args+2)+'b')
    if(a==1 and b==1 and c==1 and d==1):
        return format(6, '#0'+str(no_args+2)+'b')
```

In [16]:
```
def getDecimalNo(arr_1, arr_2):
    integer = 0
    n = 2*len(arr_1)
    final_arr = np.concatenate((arr_1,arr_2))
    for i in range(0,n):
        integer = integer + (2**i)*final_arr[n-1-i]
    return integer
```

In [17]:
```
def create_Uf(f, n):
    binary1 = format(1, '#0'+str(n+2)+'b') #results in a binary string r
epresenting 1 with adequate number of 0s given by arity of f
    binary2 = format(0, '#0'+str(n+2)+'b') #results in a binary string r
epresenting 0 with adequate number of 0s given by arity of f
    result = {}
    query_list = [int(d) for d in str(binary2)[2:]] #bitstring to send f
unction
    for i in range(pow(2,n)):
        result[binary2] = f(*query_list)
        integer_sum = int(binary1, 2) + int(binary2, 2)
        binary2 = format(integer_sum, '#0'+str(n+2)+'b') #updating the s
econd binary string by adding 1 to get the next input sequence to send f
        query_list = [int(d) for d in str(binary2)[2:]]
    #with the above code we obtained the results of f for all 2**N bitst
rings, now we proceed to get Uf for 2**2N bitstrings

    U_f = np.zeros((2**(2*n), 2**(2*n)))
    binary1 = format(1, '#0'+str(n+2)+'b') #results in a binary string r
epresenting 1 with adequate number of 0s given by arity of f
    binary2 = format(0, '#0'+str(n+2)+'b') #results in a binary string r
epresenting 0 with adequate number of 0s given by arity of f
    for i in range(pow(2,n)):
        b = format(0, '#0'+str(n+2)+'b')
        bin_num1_arr = np.array([int(d) for d in str(result[binary2])[2
:]]) #f(x)

        for i in range(pow(2,n)):
            bin_num2_arr = np.array([int(d) for d in str(b)[2:]]) #b
            f_x_b = np.add(bin_num1_arr,bin_num2_arr) %2

            column = getDecimalNo(np.array([int(d) for d in str(binary2)
[2:]]), f_x_b)
            row = getDecimalNo(np.array([int(d) for d in str(binary2)[2
:]]), np.array([int(d) for d in str(b)[2:]]))

            U_f[row][column] = 1

            integer_sum = int(binary1, 2) + int(b, 2)
            b = format(integer_sum, '#0'+str(n+2)+'b')
        integer_sum = int(binary1, 2) + int(binary2, 2)
        binary2 = format(integer_sum, '#0'+str(n+2)+'b')
    return U_f
```

In [18]:
```
from pyquil import Program
from pyquil.gates import *
from pyquil import get_qc
from pyquil.quil import DefGate
from pyquil.quilatom import unpack_qubit

def run_program(f, n):
    p = Program()

    U_f = create_Uf(f, n)
    N = n*2

    qc = get_qc("{}q-qvm".format(N)) #create qc with the number of qubit
s required
    qc.compiler.client.timeout = 1500 # number of seconds

    #adding Hadamard gates to all main qubits
    for i in range(0,int(N/2)):
        p += H(i)

    #create the UF gate
    uf_gate_definition = DefGate("UF_GATE", U_f)
    qubits = [unpack_qubit(i) for i in range(0,N)]

    #adding Uf gate to the program
    p+=Program(uf_gate_definition,Gate(name="UF_GATE", params=[],qubits=
qubits))

    #adding the remaining Hadamards to the main qubits
    for i in range(0,int(N/2)):
        p += H(i)

    try:
        results = qc.run_and_measure(p, trials=1)
    except TimeoutError:
        print("Could not obtain results for N: {}".format(N/2))
        return[]

    return results
```

In [19]:
```
run_program(f_1,2)
```

Out[19]:
```
{0: array([1]), 1: array([1]), 2: array([1]), 3: array([0])}
```

In [20]:
```
import time

#had difficulty coding this section on solving the linear equations,
#took help from a friend who did the course the previous quarter to unde
rstand the functions and implement them
#also referenced book on solving linear equations with mod 2 using gauss
ian elimination and back substitution

def isAllZeros(list_):
    for i in list_:
        if i!=0:
            return False
    return True

def addRow(vectors, z, MSBDict):
    while isAllZeros(z) is False:
        msbOne = z.index(1)
        if msbOne in MSBDict.keys():
            otherRow = MSBDict[msbOne]
            z = list(np.add(np.array(vectors[otherRow]),np.array(z))%2)
 #update z (adding those two rows with same msb to get reduced row echol
on form)
        else:
            break

    if(isAllZeros(z)):
        return vectors, MSBDict
    else:
        msbOne = z.index(1)
        keysList = list(MSBDict.keys())
        row = 0
        for row in range(len(keysList)):
            if keysList[row] > msbOne:
                break
            row+=1
        vectors.insert(row,z)
        MSBDict[z.index(1)]=row

        return vectors, MSBDict

def addLastVector(vectors):
    lastRow = [0 for i in range(len(vectors[0]))]
    lastOne = 0
    for i in range(len(vectors)):
        if vectors[i][i]==1:
            lastOne = i+1
        else:
            break
    lastRow[lastOne]=1 #insert 1 at appropriate msb position
    vectors.insert(lastOne, lastRow) #insert the lastRow in the proper i
ndex corresponding to lastOne+1
    return lastRow, vectors

def getS(f):
    MSBDict = {}
    vectors = []
```

```
    n = len(inspect.signature(f).parameters)

    start = time.time()

    countOfCallsToGetS = 0
    while len(vectors) != n-1 :
        countOfCallsToGetS+=1
        results_dict = run_program(f, n)
        results = []
        for i in range(n): #interested in the first half qubit results
 (measuring the top half bits)
            results.append(results_dict[i][0]) #limiting to only one tri
al
        print(results)
        vectors, MSBDict = addRow(vectors, results, MSBDict)

    end = time.time()
    total_time = end - start

    newRow, vectors = addLastVector(vectors)
    arr_1 = np.array(vectors)
    arr_2 = np.zeros(n)
    arr_2[newRow.index(1)] = 1
    y = np.linalg.solve(arr_1,arr_2)
    y = [i%2 for i in y]
    s = ''
    for i in range(len(y)):
        s+=str(int(y[i]))
    return s, countOfCallsToGetS, total_time
```

In [22]:
```
print(getS(f_0))
```

```
[0, 1]
('10', 1, 2.6914660930633545)
```

In [23]:
```
print(getS(f_2))
```

```
[0, 0, 0]
[0, 0, 1]
[1, 1, 1]
('110', 3, 293.8817310333252)
```

In [7]:
```
print(getS(f_0))
print(getS(f_1))
print(getS(f_2))
print(getS(f_3))
print(getS(f_4))
```

```
[0, 0]
[0, 0]
[0, 0]
[0, 1]
('10', 4, 11.894580841064453)
[0, 0]
[0, 0]
[0, 0]
[0, 0]
[1, 1]
('11', 5, 16.190147161483765)
[0, 0, 1]
[1, 1, 1]
('110', 2, 212.2325758934021)
[1, 0, 1]
[0, 0, 1]
('010', 2, 178.0379877090454)
[1, 0, 1]
[0, 0, 0]
[0, 1, 1]
('111', 3, 297.64015316963196)
```

In [8]:
```
print(getS(f_0))
print(getS(f_1))
print(getS(f_2))
print(getS(f_3))
print(getS(f_4))
```

```
[0, 0]
[0, 0]
[0, 1]
('10', 3, 8.987928867340088)
[0, 0]
[0, 0]
[1, 1]
('11', 3, 9.844814777374268)
[1, 1, 1]
[1, 1, 0]
('110', 2, 231.71567225456238)
[0, 0, 1]
[1, 0, 0]
('010', 2, 185.76187372207642)
[1, 0, 1]
[0, 0, 0]
[1, 1, 0]
('111', 3, 308.3330101966858)
```

In [9]:
```
print(getS(f_0))
print(getS(f_1))
print(getS(f_2))
print(getS(f_3))
print(getS(f_4))
```

```
[0, 0]
[0, 1]
('10', 2, 5.985502004623413)
[1, 1]
('11', 1, 3.429965019226074)
[0, 0, 1]
[1, 1, 0]
('110', 2, 229.4894552230835)
[1, 0, 1]
[0, 0, 0]
[1, 0, 0]
('010', 3, 268.4213659763336)
[0, 1, 1]
[1, 1, 0]
('111', 2, 198.01419973373413)
```

In [12]:
```
print(getS(f_5))
```

```
[1, 0, 1, 0]
```

```
---------------------------------------------------------------------
----
RPCError                                  Traceback (most recent call l
ast)
<ipython-input-12-8dfbc98f4843> in <module>
----> 1 print(getS(f_5))

<ipython-input-11-3ffab91f2ffb> in getS(f)
     50     while len(vectors) != n-1 :
     51         countOfCallsToGetS+=1
---> 52         results_dict = run_program(f, n)
     53         results = []
     54         for i in range(n): #interested in the first half qubit
 results (measuring the top half bits)

<ipython-input-9-afa15e471e74> in run_program(f, n)
     30
     31     try:
---> 32         results = qc.run_and_measure(p, trials=1)
     33     except TimeoutError:
     34         print("Could not obtain results for N: {}".format(N/2))

/anaconda3/lib/python3.7/site-packages/pyquil/api/_error_reporting.py i
n wrapper(*args, **kwargs)
    249             global_error_context.log[key] = pre_entry
    250
--> 251         val = func(*args, **kwargs)
    252
    253         # poke the return value of that call in

/anaconda3/lib/python3.7/site-packages/pyquil/api/_quantum_computer.py
 in run_and_measure(self, program, trials)
    418             program.inst(MEASURE(q, ro[i]))
    419         program.wrap_in_numshots_loop(trials)
--> 420         executable = self.compile(program)
    421         bitstring_array = self.run(executable=executable)
    422         bitstring_dict = {}

/anaconda3/lib/python3.7/site-packages/pyquil/api/_error_reporting.py i
n wrapper(*args, **kwargs)
    249             global_error_context.log[key] = pre_entry
    250
--> 251         val = func(*args, **kwargs)
    252
    253         # poke the return value of that call in

/anaconda3/lib/python3.7/site-packages/pyquil/api/_quantum_computer.py
 in compile(self, program, to_native_gates, optimize, protoquil_positio
nal, protoquil)
    479
    480         if quilc:
--> 481             nq_program = self.compiler.quil_to_native_quil(prog
ram, protoquil=protoquil)
    482         else:
    483             nq_program = program

/anaconda3/lib/python3.7/site-packages/pyquil/api/_error_reporting.py i
```

```
n wrapper(*args, **kwargs)
    249             global_error_context.log[key] = pre_entry
    250
--> 251         val = func(*args, **kwargs)
    252
    253         # poke the return value of that call in

/anaconda3/lib/python3.7/site-packages/pyquil/api/_compiler.py in quil_
to_native_quil(self, program, protoquil)
    412         self.connect()
    413         request = NativeQuilRequest(quil=program.out(), target_
device=self.target_device)
--> 414         response = self.client.call("quil_to_native_quil", requ
est, protoquil=protoquil).asdict()
    415         nq_program = parse_program(response["quil"])
    416         nq_program.native_quil_metadata = response["metadata"]

/anaconda3/lib/python3.7/site-packages/rpcq/_client.py in call(self, me
thod_name, rpc_timeout, *args, **kwargs)
    203
    204         if isinstance(reply, RPCError):
--> 205             raise utils.RPCError(reply.error)
    206         else:
    207             return reply.result

RPCError: Unhandled error in host program:
Heap exhausted (no more space for allocation).
2981888 bytes available, 8388624 requested.

PROCEED WITH CAUTION.
```

In [ ]: