CS 1511 Homework 25

Mathew Varughese, Justin Kramer, Zach Smith

Friday, Apr 12

51 a. u1 = 0 u2 = 1 u3 = 1

51 b.

```
SOLUTION = "011"

def inner_product(a, b):
        a = "{:05b}".format(int(a, 2))
        b = "{:05b}".format(int(b, 2))
        sum = 0
        for a_i, b_i in zip(a, b):
                sum += (int(a_i) * int(b_i)) % 2
        return sum % 2

def outer_product(a, b):
        assert len(a) == len(b)
        product = ""
        for a_i in a:
                for b_i in b:
                        product += str((int(a_i) * int(b_i)) % 2)
        return product

def walsh_hadamard(x):
        total = len(x)
        binarys = []
        for i in range(2**total):
                binarys.append("{:05b}".format(i))
        encoded = "".join([str(inner_product(x, b)) for b in binarys])
        return encoded


wh_u = walsh_hadamard(SOLUTION)
uxu = outer_product(SOLUTION, SOLUTION)
wh_uxu = walsh_hadamard(uxu)
```

```
print(wh_u, end='')
print(wh_uxu)
```

51 c.

01100110011001101001100110011001011001101100001111
00001100111100001111001111000000001111111100000000
11110000111111110000000011111111000011111111000000
00000000001111111111111111000000000000000011111111
00000000111111111111111100000000000000011111111111
11111100000000111111111111111100000000000000000000
00000000000011111111111111111111111111111111000000
00000000000000000000000000111111111111111100000000
00000000111111111111111111111111111111110000000000
00000000000000000000001111111111111111111111111111
11110000000000000000

51 d.

u1u2 + u2u2 + u3u3 would have 1s in each combo of these. To explain further, the 9 bit long string 000 000 000 contains digits, and each digit corresponds to a combination of the u's. It is really

$$(u_1u_1)(u_1u_2)(u_1u_3)(u_2u_1)(u_2u_2)(u_2u_3)(u_3u_1)(u_3u_2)(u_3u_3)$$

So,

010010001

is the binary string where each corresponding u term in the equation specified in the problem has a 1. This number in base 10 is 145. 145 + the first 8 bits used to store the Walsh Hadamard encoding is 153. So look in this spot.

52. NP = L-PCP(log n)

NP = {L: there is a logspace machine M s.t x $\in$ L iff $\exists$ y : M accepts (x,y) }.

L-PCP(log n) = {L : there is a logspace machine M s.t x $\in$ L iff $\forall$ y : M accepts (x,y) with probability 1 and x $\notin$ L iff $\forall$ y : M rejects (x,y) with probability $\geq 1/2$}

We need to show two things

NP $\subseteq$ L-PCP(log n)

L $\in$ NP

$\exists$ M that decides L

This is simple, have the log space verifier tape of the NP machine M become the random bits that the L-PCP(log n) uses.

This will accept and reject with probability 1, which falls under the L-PCP(log n) conditions.
L $\in$ L-PCP(log n)

L-PCP(log n) $\subseteq$ NP
L $\in$ L-PCP(log n)
$\exists$ M that decides L
Run the machine and build a set R that is the random bits used when the machine accepts for a logirithmic sized R. Then use this set R to build the NP machine with R as the verifier tape.
L $\in$ NP