Notebook

October 21, 2024

1 Problem 11: «A simple hash function»

1.1 Problem

- Carol invented a keyed hash algorithm. The key $k = (k_1, k_2, ..., k_6)$ for this hash function is a binary vector of length 6, the input of hash function p is a sequence of digits. It should be divided into blocks of length 6. If the length of the sequence is not a multiple of 6 then it can be completed with 1, 2, 3, and so on up to the necessary length. It should be divided into blocks of length 6. Let say the input is $p = (p_1, p_2, ..., p_6)$, the hash value of each block is calculated by this formula:

$$H(k, p) = \sum_{i=1}^{6} (-1)^{k_i} * p_i$$

  Say $n_1, n_2, ..., n_k$ is the result of such calculations of each block, then the final hash value is calculated as

$$H = \sum_{i=1}^{k} (-1)^{i} * n_i$$

- Our goal is to propose a simplest algorithm how to get a collision of the first order for any known input sequence P if the key K is unknown. By the way, find the shortest collision for the sequence from the example, $P = 134875512293$.

1.2 Solution

- We can form the hash function into a formula like this:

$$H(k, p) = \sum_{i=1}^{6} (-1)^{k_i} * \left(\sum_{j=0}^{k} (-1^j) * p_{i+6j} \right)$$

  Because the key K is unknown, we have to find an algorithm that find collision for all case of K, which means all sums of $(-1^j) * p_{i+6j}$ of 2 message have to be respectively equal.
- In my algorithm, I firstly find the minimum numbers of blocks in the collision. I do it like this:
  - Calculate the range of sum $(-1^j) * p_{i+6j}$ in each case of number of blocks.
  - Calculate all sums $(-1^j) * p_{i+6j}$ of input sequence, say n_i
  - Find the biggest number of blocks such that its ranges can fit all calculated sum of input sequence. That is the minimum of number of blocks in collision.

[1]:
```
def find_bound(num_blocks):
    '''
```

```
    find the range of sum (-1^j)*p_{i+6j} in each case of num_blocks
    '''
    min = 0
    max = 0
    for i in range(num_blocks):
        if i % 2:
            min -= 9
        else:
            max += 9
    return min, max

def find_min_num_block(x_i: list[int]):
    '''
    Find the minimum of number of blocks in collision
    '''
    num = 1
    while not all(x in range(*find_bound(num)) for x in x_i):
        num += 1
    return num
```

- After having the number of blocks, suppose it is k, we handle each sum $(-1^j)*p_{i+6j}$ seperately. Now we have to solve a sub-problem like this: > Find a sequence of number p_i, with i from 1 to k such that $\sum_{i=1}^{k}(-1)*p_i = x$ with known x and the number $\overline{p_1p_2...p_k}$ is minimum.
- To solve that, I simply bruteforce the value $\overline{p_1p_2...p_k}$ from 0 to the maximum to find the first number satisfying the condition. We can have a Python script to simulate this process

[3]:
```
def calculate(n : list[str]) -> int:
    '''
    Calculate the sum (-1^i)*n_i
    '''
    return sum((-1)**i * int(x) for i, x in enumerate(n))

def brute(x : int, num_blocks : int) -> list[str]:
    candidate = 0
    if num_blocks == 1:
        if 0 <= x <= 9:
            return [str(x)]
        else:
            return None
    while True:
        candidate_str = str(candidate).rjust(num_blocks - 1, '0')
        known = calculate(candidate_str)
        if num_blocks % 2:
            unknown = x - known
        else:
            unknown = known - x
        if 0 <= unknown <= 9:
```

```
            return list(candidate_str + str(unknown))
            break
        else:
            candidate += 1
        if candidate > 10**(num_blocks-1):
            return None
```

- Now, we have to consider to the case of padding. We can take advantage of padding to reduce the length of collision to find the shortest one. So, we can consider each case of padding in turn from 12345, 1234, 123, 12, 1 and no padding. Once we found a collision in a such case of padding, we can stop and return the value.

[4]:
```
def find_collision(original_data : str):
    # padding the original data
    i = 1
    while len(original_data) % 6:
        original_data += str(i)
        i += 1
    blocks = [original_data[i:i+6] for i in range(0, len(original_data), 6)]
    x_i = [calculate([block[i] for block in blocks]) for i in range(6)] # sums␣
 ↪of (-1)^i * p_{i+6j}
    num_blocks = find_min_num_block(x_i) # find the minimum number of blocks in␣
 ↪collision
    print(x_i)
    if num_blocks == 1:
        return "".join(str(x) for x in x_i)
    # consider all padding cases
    for num_pad in [5,4,3,2,1,0]:
        coll = []
        last_block_padding = (6 - num_pad) * [None] + list(range(1,num_pad + 1))
        for i,x in enumerate(x_i):
            if last_block_padding[i]:
                if num_blocks % 2:
                    x -= last_block_padding[i]
                else:
                    x += last_block_padding[i]
                result = brute(x, num_blocks - 1)
                if result == None:
                    break
                else:
                    result += [str(last_block_padding[i])]
            else:
                result = brute(x, num_blocks)
            coll.append(result)
        if len(coll) < 6:
            continue
        ans = ""
```

```
        for i in range(num_blocks):
            ans += "".join(c[i] for c in coll)
        return ans[:-num_pad]
```

[5]:
```
print(find_collision("134875512293"))
```

```
[-4, 2, 2, 6, -2, 2]
0349274
```

So, the shortest collision for the sequence $P = 134875512293$ is 0349274. The algorithm I just present above is also a simplest algorithm how to get a collision of the first order for any known input sequence P if the key K is unknown. QED