Finding Small Sizes of Modulo Difference Covers

Jacob Charboneau, and Dr. Ankur Gupta

Butler University

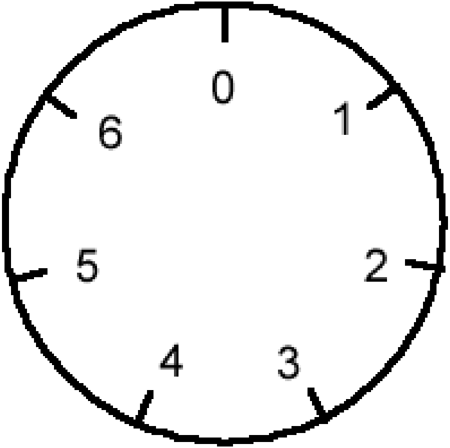
April 29, 2020

Introduction

Let $P = \{0, 1, ..., p-1\}$

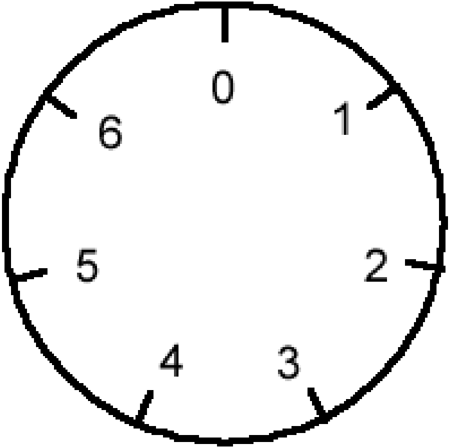
Let $S \subseteq P$ be a **modulo difference cover** of P when $\forall n \in P, \exists s_i, s_j \in S$ such that $s_i - s_j = n \bmod p$.

Example: p=7



Let $P = \{0, 1, 2, 3, 4, 5, 6\}$

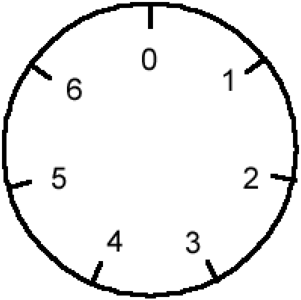
Example: p=7



Let $P = \{0, 1, 2, 3, 4, 5, 6\}$

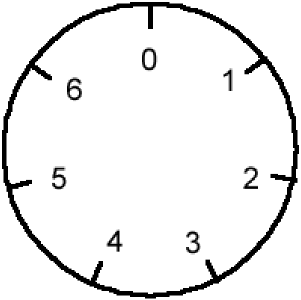
We want to determine whether $S = \{0, 1, 3\}$ is a modulo difference cover of P.

Example: p=7



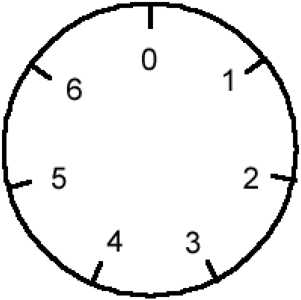
S = {0, 1, 3}
0 mod 7 = 0

Example: p=7



S = {0, 1, 3}
0 mod 7 = 0
(1-0) mod 7 = 1
(0-1) mod 7 = 6

Example: p=7



S = {0, 1, 3}
0 mod 7 = 0
(1-0) mod 7 = 1
(0-1) mod 7 = 6
(3-1) mod 7 = 2
(3-0) mod 7 = 3
(0-3) mod 7 = 4
(1-3) mod 7 = 5

Greedy Algorithm

```
int [] Greedy(int p)
    P = {0, 1, ..., p-1}
    Pcov = {1, 1, 0, 0, ..., 0, 1}
    S = {0, 1}
    x = 2; best = 0; bestdiff = 0; diff=0;
    while S is not a Cover of P do {
        while x<p do {
            if x ∉ S
                for i from 0 to |S| do {
                    if Pcov[x-S[i] mod p]==0
                        diff=diff+1;
                    if Pcov[S[i]-x mod p]==0
                        diff=diff+1;
                if diff>bestdiff
                    bestdiff=diff; best=x; diff=0;}
            x=x+1;}
        S = S+{best};Update(Pcov, S);
        x=2; best=0; bestdiff = 0;}
    return S;
```

Greedy Algorithm Data

P size	√P	Greedy Algorithm	2√P
10000	100	161	200
20000	141	240	283
30000	173	300	346
40000	200	352	400
50000	224	397	447
60000	245	441	490
70000	265	483	529
80000	283	517	566
90000	300	553	600
100000	316	586	632

This algorithm has a runtime of $O(p^2)$.

Chinese Remainder Theorem

Theorem (Chinese Remainder Theorem)

Let $n_1, \ldots, n_i$ be coprime integers greater than 1.
Then there exists uniquely one integer $x \in \mathbb{Z}_N$ such that $x = a_1$ mod $n_1, x = a_2$ mod $n_2, \ldots, x = a_i$ mod n_i, where $N = \prod_1^i n_k$.

How is it used?

Theorem

if $p = q * r$ where q, r are coprime, then $S = S_q \times S_r$ is a cover of $P = \mathbb{Z}_p$ and $S_q \in \mathbb{Z}_q, S_r \in \mathbb{Z}_r$ are covers for their parent sets.

How is it used?

Proof.

Let us assume that S does not cover $\mathbb{Z}_p$. So there is some element $z \in \mathbb{Z}_p$ that is not covered by S.
This means there exists some (x, y) such that $x \in \mathbb{Z}_q, y \in \mathbb{Z}_r$ that is congruent to $z \in \mathbb{Z}_p$ under the Chinese Remainder Theorem.
But because S_q covers $\mathbb{Z}_q$ and S_r covers $\mathbb{Z}_r$, $\exists q_i, q_j, r_k, r_l$ such that $(x, y) = (q_i - q_j, r_k - r_l) = (q_i, r_k) - (q_j, r_l)$, where $(q_i, r_k), (q_j, r_l) \in S$.
Thus $(x, y) \in S$ and S is a cover of $\mathbb{Z}_p$. $\square$

Chinese Remainder Algorithm

In the following pseudocode, `getCover(int n)` retrieves the recorded cover for $|P| = n$ from a list, and `findBest(int [] factors)` returns two coprime integers m, n where $p = mn$ and m and n yield the smallest possible $|S_m||S_n|$ where S_m, S_n are retrieved from a list.
$T(n) \in O(1)$ for getCover(int n)
$T(n) \in O(2^k)$ for findBest(int [] factors), where k is the number of primes that p factors into.

Chinese Remainder Algorithm

```
int [] CRAlg(int p)
  int k = 0;
  factors[] = sieve(p);
  Best[2] = findBest(factors);
  S1 = getCover(Best[0]);S2 = getCover(Best[1]);
  S = new int[Best[0]*Best[1]];
  for(i=0; i<|S1|; i++)
     for(j=0; j<|S2|; j++)
        S[k]=CRT(S1[i], S2[j], Best[0], Best[1]);k++;
  return S;
```

$T(p) \in O(p^{3/2})$

Chinese Remainder Algorithm

```
int CRT(int x, int y, int m, int n)
  while(x!=y)
     while(x<m*n)
        x=x+m;
        if(x==y)
           break;
     if(x!=y)
        x=x%m; y=y+n;
  return x;
```

$T(p) \in O(p)$

Greedy Algorithm Data

P size	CRA Cover Size	Greedy Cover Size
6	4	3
21	6	6
35	9	8
50	12	9
74	14	12
75	12	11
92	18	12
100	18	13

THANK YOU FOR LISTENING