信息安全数学基础 第九次作业

BY 18340087 李晨曦

(1)

$$\begin{aligned}\sigma_1\sigma_2 &= \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 6 & 4 & 5 & 3 & 1 & 2 \end{pmatrix} \\ &= (2,4,3,5,1,6)\end{aligned}$$

$$\begin{aligned}\sigma_2\sigma_1 &= \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 3 & 4 & 2 & 6 & 1 & 5 \end{pmatrix} \\ &= (1,3,2,4,6,5)\end{aligned}$$

$$\begin{aligned}\sigma_1^{-1} &= \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 6 & 1 & 2 & 3 & 4 & 5 \end{pmatrix} \\ &= (1,6,5,4,3,2)\end{aligned}$$

(2)

由于3是质数， 所以S_4的所有三阶子群的元素都是某个形式为(a_0,a_1,a_2)的S_4中元素作为生成元的循环子群。

证明. 记一个S_4的三阶子群为G. 任取一个$a\in G$.

a可以被表示为不交的$(a_1,\ldots a_{n_1})(a_{n_0+1},\ldots a_{n_1})\ldots(a_{n_{k-1}}\ldots a_{n_k})$， 而以$a$作为生成元的循环子群$H$的阶数$m$满足

$$m=[n_1,n_2\ldots n_k]$$

由Lagrange定理， 我们知道若设包含这个循环子群的对称群G的阶数为一个质数p， 那么有：

$$[n_1,n_2\ldots n_k]|p$$

所以$k=1$显然成立， 且$n_1=p$或$n_1=1$. 这里显然有$n_1=p$. 即$|H|=|G|$， 又因为$H\subset G$， 所以$H=G$.

也就是说， G是a作为生成元的循环子群， 且G中只可能含有(a_0,a_1,a_2)形式的cycle， 否则将会与$[n_1,n_2\ldots n_k]|p$的条件抵触。所以，G一定是某个形式为(a_0,a_1,a_2)的S_4中元素作为生成元的循环子群。 □

由此， 我们可以首先得到S_4中所有与上述形式共轭的元素：

$$\begin{gathered}(4,2,3)\\(3,2,4)\\(3,1,2)\\(4,1,2)\\(4,1,3)\\(2,1,3)\\(3,1,4)\\(2,1,4)\end{gathered}$$

然后求其所有元素作为生成元的循环子群，可以得到四个阶为3的子群：

$$\{e,(4,2,3),(3,2,4)\}$$
$$\{e,(3,1,2),(2,1,3)\}$$
$$\{e,(4,1,2),(2,1,4)\}$$
$$\{e,(4,1,3),(3,1,4)\}$$

(3)

对于p是合数的情况，上面的讨论仍然有一些可以使用：

- $[n_1,n_2...n_k]|p$仍然成立，本题中$p=4$，可以得到任何满足(a_0,a_1,a_2)形式的cycle都不是S_4的四阶子群的元素
- 形式为(a_0,a_1)的元素的生成子群只有这个元素和e，两个不交的形式为(a_0,a_1)的元素生成的新元素的生成子群也只有这个新元素和e，所以这三个元素和e构成一个四阶子群
- 上条中所说的『新元素』的生成子群也只有这个元素和e，两个这样的元素的乘积是一个与之共轭的元素，且它的生成子群也只有这个元素和e，与另外两个元素的乘积在这三个元素中封闭（因为$a\,ab=(aa)b=eb=b$），这样的三个元素和e构成一个四阶子群。
- 一个形式为(a_0,a_1,a_2,a_3)的元素的生成子群构成一个四阶子群。

这三种四阶子群枚举如下：

$$\{e,(1,2),(3,4),(1,2)(3,4)\}$$
$$\{e,(1,3),(2,4),(1,3)(2,4)\}$$
$$\{e,(1,4),(2,3),(1,4)(2,3)\}$$
$$\{e,(1,4)(2,3),(1,2)(3,4),(1,3)(2,4)\}$$
$$\{e,(1,2,3,4),(1,3)(2,4),(1,4,3,2)\}$$
$$\{e,(1,3,2,4),(1,2)(3,4),(1,4,2,3)\}$$
$$\{e,(1,2,4,3),(1,4)(2,3),(1,3,4,2)\}$$

(5)

一个有限循环群G，$|G|=m$，我们可以知道，G与$\mathbb{Z}/m\mathbb{Z}$同构. 实际上，现在只需要证明F_p^*是一个有限循环群即可。

由于p是奇素数，存在原根r，使得

$$H=\{r^t,1\leqslant t\leqslant p-1\}$$

有$|H|=p-1$. 而$|G|=p-1,H\subset G$，所以$H=G$.

这也就是说，G是一个有限循环群，所以G与$\mathbb{Z}/(p-1)\mathbb{Z}$同构

(7)

下面用$[n]$表示代表元为n的、满足$\forall n_1,n_2\in[n]\rightarrow n_1+p^2\mathbb{Z}=n_2+p^2\mathbb{Z}$的等价类。

由上面的定义可以知道，这里的等价类实际上就是模p^2的剩余类。

对任意的$[n_1],[n_2],[n_3]$， 有：

$$\begin{aligned}[n_1]\times([n_2]\times[n_3]) &= [n_1]\times[n_2\times n_3]\\ &= [n_1\times n_2\times n_3]\\ &= ([n_1]\times[n_2])\times[n_3]\end{aligned}$$

所以有交换律。

$$[n_1]=[n_1]\times[1]$$

所以有单位元$[1]$。

由题意， 所有的元素可逆， 故有逆元。

而如果$[n_1]$有逆元，$[n_2]$有逆元， 我们可以知道：

$$\begin{aligned}\mathrm{GCD}(n_1,p^2) &= 1\\ \mathrm{GCD}(n_2,p^2) &= 1\end{aligned}$$

那么， 显然有

$$\mathrm{GCD}(n_1n_2,p^2)=1$$

故$[n_1n_2]$也有逆元，$[n_1n_2]\in G$成立

而

$$[n_1]\times[n_2]=[n_1n_2]$$

所以有封闭性。结合以上分析可知这是一个群。

这个群G的阶数即$\{x|1\leqslant x\leqslant p^2-1,\mathrm{GCD}(x,p^2)=1\}$这个集合的元素个数。从1到$p^2-1$一共有$p^2-1$个元素， 其中整除$p^2$的因数$p$的有$p-1$个， 所以群$G$的阶数为$p^2-p$.

由原根的存在性知， 存在r， 使得$\mathrm{Ord}(r)=\varphi(p^2)=p^2-p$. 这样一来就证明了群$G$是循环群。

(9)

这个问题可以用(2)中介绍的方法解决。但是由于规模比较大， 我们写程序解决它：

```racket
#lang racket

(define (permutations s)
  (cond [(empty? s) empty]
        [(empty? (rest s)) (list s)]
        [else
         (let splice [(l '()) (m (first s)) (r (rest s))]
           (append
            (map (lambda (x) (cons m x))
                 (permutations (append l r)))
            (if (empty? r)
                empty
                (splice (cons m l) (car r) (cdr r)))))]))

(define (vector-permutations vec)
  (map (λ (x) (list->vector x))
       (permutations (vector->list vec))))
```

```
(struct circle (list order)
  #:inspector #f)

(define (simply->normal cir)
  (let* ([vec (make-vector (circle-order cir) -1)]
         [atom->normal
          (λ (x)
            (let loop ([i 0])
              (if (= i (length x))
                  #t
                  (begin
                    (vector-set! vec (list-ref x i)
                                 (list-ref x (modulo (+ i 1) (length x))))
                    (loop (+ i 1))))))])
    (for-each (λ (x) (atom->normal x)) (circle-list cir))
    (let loop ([i 0])
      (if (< i (vector-length vec))
          (begin
            (if (= (vector-ref vec i) -1)
                (vector-set! vec i i)
                #t)
            (loop (+ i 1)))
          #t))
    (circle vec (circle-order cir))))

(define (normal->simply cir)
  (define mark (make-vector (+ (circle-order cir) 1) #f))
  (define (set-mark! x) (vector-set! mark x #t))
  (define (get-mark x) (vector-ref mark x))
  (define vec (circle-list cir))
  (define (normal->atom vec i first)
    (if (get-mark i)
        '()
        (let ([value (vector-ref vec i)])
          (set-mark! i)
          (if (= first i)
              (list value)
              (cons value
                    (normal->atom vec value first))))))
  (define (search i)
    (if (< i (vector-length vec))
        (cons (normal->atom vec (vector-ref vec i) i)
              (search (+ i 1)))
        '()))
  (circle
   (reverse
    (foldl (λ (x res) (if (or (null? x) (= (length x) 1))
                           res
                           (cons x res))) '() (search 0)))
   (circle-order cir)))

(define (vector-merge! x y)
  (let loop ([i 0])
    (if (< i (vector-length x))
        (begin
```

```
          (vector-set! x i (vector-ref y (vector-ref x i)))
          (loop (+ i 1)))
        #t)))

(define (circle-mult cir-y cir-x)
  (let ([normal-x (simply->normal cir-x)]
        [normal-y (simply->normal cir-y)])
    (vector-merge! (circle-list normal-x)
                   (circle-list normal-y))
    (normal->simply normal-x)))

(define (circle^n cir n)
  (if (= n 1)
      cir
      (circle-mult cir (circle^n cir (- n 1)))))

(define (circle-e n) (circle '(()) n))

(define (vector-equal? x y)
  (let loop ([i 0])
    (if (> (vector-length x) i)
        (if (= (vector-ref x i)
               (vector-ref y i))
            (loop (+ i 1))
            #f)
        #t)))

(define (circle-equal? cir-x cir-y)
  (let ([normal-x (simply->normal cir-x)]
        [normal-y (simply->normal cir-y)])
    (vector-equal? (circle-list normal-x)
                   (circle-list normal-y))))

(define (all-Sym_n n)
  (define all-vec (vector-permutations (build-vector n (λ (x) x))))
  (map (λ (x) (normal->simply (circle x n))) all-vec))

(define (all-gen-subgroup cir)
  (define (iter n)
    (let ([value (circle^n cir n)])
      (if (circle-equal? value (circle-e (circle-order cir)))
          (list value)
          (cons value (iter (+ n 1))))))
  (iter 1))

(define (circle-n? cir n)
  (if (null? (circle-list cir))
      #f
      (if (= (length (car (circle-list cir))) n)
          #t
          #f)))

(define (equal?-l l value)
  (if (equal? (member value l circle-equal?) #f)
      #f
      #t))
```

```scheme
(define (all-prime-subgroup n prime)
  (define all-group (all-Sym_n n))
  (define (iter groups)
    (if (null? groups)
        '()
        (if (circle-n? (car groups) prime)
            (let ([subgroup (all-gen-subgroup (car groups))])
              (cons subgroup
                    (iter (filter (λ (x) (not (equal?-l subgroup x)))
                                  (cdr groups)))))
            (iter (cdr groups)))))
  (iter all-group))

(all-prime-subgroup 6 5)
```

得到结果：（如何阅读此结果已在注释中）

```scheme
(list ;所有的子群
 (list ;一个子群
  (circle '((3 4 5 1 2)) 6)  ;(4 5 6 2 3)
  (circle '((5 2 4 1 3)) 6)  ;(6 3 5 2 4)
  (circle '((2 5 3 1 4)) 6)  ;(3 6 4 2 5)
  (circle '((4 3 2 1 5)) 6)  ;(5 4 3 2 6)
  (circle '() 6))            ; e
 (list
  (circle '((3 5 4 1 2)) 6)
  (circle '((4 2 5 1 3)) 6)
  (circle '((2 4 3 1 5)) 6)
  (circle '((5 3 2 1 4)) 6)
  (circle '() 6))
 (list
  (circle '((4 5 3 1 2)) 6)
  (circle '((3 2 5 1 4)) 6)
  (circle '((2 3 4 1 5)) 6)
  (circle '((5 4 2 1 3)) 6)
  (circle '() 6))
 (list
  (circle '((4 3 5 1 2)) 6)
  (circle '((5 2 3 1 4)) 6)
  (circle '((2 5 4 1 3)) 6)
  (circle '((3 4 2 1 5)) 6)
  (circle '() 6))
 (list
  (circle '((5 3 4 1 2)) 6)
  (circle '((4 2 3 1 5)) 6)
  (circle '((2 4 5 1 3)) 6)
  (circle '((3 5 2 1 4)) 6)
  (circle '() 6))
 (list
  (circle '((5 4 3 1 2)) 6)
  (circle '((3 2 4 1 5)) 6)
  (circle '((2 3 5 1 4)) 6)
  (circle '((4 5 2 1 3)) 6)
  (circle '() 6))
 (list
```

```
 (circle '((2 3 4 0 1)) 6)
 (circle '((4 1 3 0 2)) 6)
 (circle '((1 4 2 0 3)) 6)
 (circle '((3 2 1 0 4)) 6)
 (circle '() 6))
(list
 (circle '((2 3 5 0 1)) 6)
 (circle '((5 1 3 0 2)) 6)
 (circle '((1 5 2 0 3)) 6)
 (circle '((3 2 1 0 5)) 6)
 (circle '() 6))
(list
 (circle '((2 4 5 0 1)) 6)
 (circle '((5 1 4 0 2)) 6)
 (circle '((1 5 2 0 4)) 6)
 (circle '((4 2 1 0 5)) 6)
 (circle '() 6))
(list
 (circle '((2 4 3 0 1)) 6)
 (circle '((3 1 4 0 2)) 6)
 (circle '((1 3 2 0 4)) 6)
 (circle '((4 2 1 0 3)) 6)
 (circle '() 6))
(list
 (circle '((2 5 4 0 1)) 6)
 (circle '((4 1 5 0 2)) 6)
 (circle '((1 4 2 0 5)) 6)
 (circle '((5 2 1 0 4)) 6)
 (circle '() 6))
(list
 (circle '((2 5 3 0 1)) 6)
 (circle '((3 1 5 0 2)) 6)
 (circle '((1 3 2 0 5)) 6)
 (circle '((5 2 1 0 3)) 6)
 (circle '() 6))
(list
 (circle '((3 4 5 0 1)) 6)
 (circle '((5 1 4 0 3)) 6)
 (circle '((1 5 3 0 4)) 6)
 (circle '((4 3 1 0 5)) 6)
 (circle '() 6))
(list
 (circle '((3 5 4 0 1)) 6)
 (circle '((4 1 5 0 3)) 6)
 (circle '((1 4 3 0 5)) 6)
 (circle '((5 3 1 0 4)) 6)
 (circle '() 6))
(list
 (circle '((3 4 2 0 1)) 6)
 (circle '((2 1 4 0 3)) 6)
 (circle '((1 2 3 0 4)) 6)
 (circle '((4 3 1 0 2)) 6)
 (circle '() 6))
(list
 (circle '((3 5 2 0 1)) 6)
 (circle '((2 1 5 0 3)) 6)
```

```
 (circle '((1 2 3 0 5)) 6)
 (circle '((5 3 1 0 2)) 6)
 (circle '() 6))
(list
 (circle '((3 2 4 0 1)) 6)
 (circle '((4 1 2 0 3)) 6)
 (circle '((1 4 3 0 2)) 6)
 (circle '((2 3 1 0 4)) 6)
 (circle '() 6))
(list
 (circle '((3 2 5 0 1)) 6)
 (circle '((5 1 2 0 3)) 6)
 (circle '((1 5 3 0 2)) 6)
 (circle '((2 3 1 0 5)) 6)
 (circle '() 6))
(list
 (circle '((4 2 3 0 1)) 6)
 (circle '((3 1 2 0 4)) 6)
 (circle '((1 3 4 0 2)) 6)
 (circle '((2 4 1 0 3)) 6)
 (circle '() 6))
(list
 (circle '((4 5 3 0 1)) 6)
 (circle '((3 1 5 0 4)) 6)
 (circle '((1 3 4 0 5)) 6)
 (circle '((5 4 1 0 3)) 6)
 (circle '() 6))
(list
 (circle '((4 3 5 0 1)) 6)
 (circle '((5 1 3 0 4)) 6)
 (circle '((1 5 4 0 3)) 6)
 (circle '((3 4 1 0 5)) 6)
 (circle '() 6))
(list
 (circle '((4 3 2 0 1)) 6)
 (circle '((2 1 3 0 4)) 6)
 (circle '((1 2 4 0 3)) 6)
 (circle '((3 4 1 0 2)) 6)
 (circle '() 6))
(list
 (circle '((4 5 2 0 1)) 6)
 (circle '((2 1 5 0 4)) 6)
 (circle '((1 2 4 0 5)) 6)
 (circle '((5 4 1 0 2)) 6)
 (circle '() 6))
(list
 (circle '((4 2 5 0 1)) 6)
 (circle '((5 1 2 0 4)) 6)
 (circle '((1 5 4 0 2)) 6)
 (circle '((2 4 1 0 5)) 6)
 (circle '() 6))
(list
 (circle '((5 2 4 0 1)) 6)
 (circle '((4 1 2 0 5)) 6)
 (circle '((1 4 5 0 2)) 6)
 (circle '((2 5 1 0 4)) 6)
```

```
 (circle '() 6))
(list
 (circle '((5 2 3 0 1)) 6)
 (circle '((3 1 2 0 5)) 6)
 (circle '((1 3 5 0 2)) 6)
 (circle '((2 5 1 0 3)) 6)
 (circle '() 6))
(list
 (circle '((5 3 4 0 1)) 6)
 (circle '((4 1 3 0 5)) 6)
 (circle '((1 4 5 0 3)) 6)
 (circle '((3 5 1 0 4)) 6)
 (circle '() 6))
(list
 (circle '((5 4 3 0 1)) 6)
 (circle '((3 1 4 0 5)) 6)
 (circle '((1 3 5 0 4)) 6)
 (circle '((4 5 1 0 3)) 6)
 (circle '() 6))
(list
 (circle '((5 3 2 0 1)) 6)
 (circle '((2 1 3 0 5)) 6)
 (circle '((1 2 5 0 3)) 6)
 (circle '((3 5 1 0 2)) 6)
 (circle '() 6))
(list
 (circle '((5 4 2 0 1)) 6)
 (circle '((2 1 4 0 5)) 6)
 (circle '((1 2 5 0 4)) 6)
 (circle '((4 5 1 0 2)) 6)
 (circle '() 6))
(list
 (circle '((3 4 5 0 2)) 6)
 (circle '((5 2 4 0 3)) 6)
 (circle '((2 5 3 0 4)) 6)
 (circle '((4 3 2 0 5)) 6)
 (circle '() 6))
(list
 (circle '((3 5 4 0 2)) 6)
 (circle '((4 2 5 0 3)) 6)
 (circle '((2 4 3 0 5)) 6)
 (circle '((5 3 2 0 4)) 6)
 (circle '() 6))
(list
 (circle '((4 5 3 0 2)) 6)
 (circle '((3 2 5 0 4)) 6)
 (circle '((2 3 4 0 5)) 6)
 (circle '((5 4 2 0 3)) 6)
 (circle '() 6))
(list
 (circle '((4 3 5 0 2)) 6)
 (circle '((5 2 3 0 4)) 6)
 (circle '((2 5 4 0 3)) 6)
 (circle '((3 4 2 0 5)) 6)
 (circle '() 6))
(list
```

```
 (circle '((5 3 4 0 2)) 6)
 (circle '((4 2 3 0 5)) 6)
 (circle '((2 4 5 0 3)) 6)
 (circle '((3 5 2 0 4)) 6)
 (circle '() 6))
(list
 (circle '((5 4 3 0 2)) 6)
 (circle '((3 2 4 0 5)) 6)
 (circle '((2 3 5 0 4)) 6)
 (circle '((4 5 2 0 3)) 6)
 (circle '() 6)))
```