**Tema 1 ASC - v10**

https://github.com/JustBeYou/ASC/blob/master/tema1/doc.md

**Descriere algoritm**

1. Am construit ciurul lui Eratosthenes pentru a calcula eficient numerele prime mai mici sau egale cu p. O(p \* log log p)

2. Am parcurs ciurul pana la p si am stocat numerele prime intr-un vector. O(p)

3. Generatorul g pe care il cautam satisface urmatoarele doua proprietati:

\* se afla printre numerele prime mai mici decat p

\* g la puterea p - 1 este congruent cu 1

\* in sirul ridicarii sale la putere (g ^ 2, g ^ 3, ..., g ^ (p - 2)) niciunul din rezultate nu este congruent cu 1

4. Iterez prin toate numerele prime mai mici decat pe si verific conditiile anterioare. O((p - 2) \* (p / ln(p)))

5. Dupa determinarea generatorlui, parcurg sirul citit si calculez g ^ (caracter - 'A') modulo p. O(exponent maxim \* lungime sir)

6. Pentru verificare, efectuez si decriptarea sirului abia criptat. Operatia inversa este logaritmul log\_g(caracter) modulo p. O(p)

**Observatii**

1. In descrierea algoritmului am considerat datele de intrare corecte, verificarile apar totusi in program

2. Am stabilit alfabetul ca fiind format din literele mari de la A la Z

**Exemple**

p = 7

g = 3

mesaj = ACAD

criptat = BCBG

decriptat = ACAD

p = 8

p nu este prim

p = 0

p nu este prim

p = 1

p nu este prim

p = 2

p nu poate sa fie 2

p = 100

p trebuie sa fie mai mic decat 26

p = -2

p trebuie sa fie pozitiv

p = 13

g = 2

mesaj = ABCDEF

criptat = BCEIDG

decriptat = ABCDEF

**Cod sursa (MIPS)**

# conventie apelare proceduri: O32

# primele 4 argumente in $a0..4, restul pe stiva

# functia apelata trebuie sa salveze registrii $s0..7 daca ii foloseste

# valorile se returneaza prin $v0..1

PRINT\_INT\_SYSCALL = 1

PRINT\_STR\_SYSCALL = 4

READ\_INT\_SYSCALL = 5

READ\_STR\_SYSCALL = 8

EXIT\_SYSCALL = 10

# dimensiuni hardcodate ale array-urilor

MAX\_P = 256

MAX\_S = 1000

.data # toate sunt intializate cu 0

sieve: .space MAX\_P # ciur, 0 -> prim, 1 -> neprim

primes: .space MAX\_P \* 4 # lista numere prime

primesCnt: .word 0 # contor numere prime

input: .space MAX\_S # sir citit de la utilizator

newline: .asciiz "\n"

space: .asciiz " "

notPrimeMessage: .asciiz "p nu este prim\n"

foundGMessage: .asciiz "g = "

readPMessage: .asciiz "p = "

readInputMessage: .asciiz "mesaj = "

encryptedMessage: .asciiz "criptat = "

decryptedMessage: .asciiz "decriptat = "

negativePMessage: .asciiz "p trebuie sa fie pozitiv"

pTooBigMessage: .asciiz "p trebuie sa fie mai mic decat 26"

pIsTwoMessage: .asciiz "p nu poate sa fie 2"

.text

main:

# p

li $v0, PRINT\_STR\_SYSCALL

la $a0, readPMessage

syscall

# $s0 -> p

li $v0, READ\_INT\_SYSCALL

syscall

move $s0, $v0

# daca p < 0 atunci stop

blt $s0, 0, p\_negative

# daca p >= 26 atunci stop

bge $s0, 26, p\_too\_big

# daca p == 2 atnci stop

beq $s0, 2, p\_is\_two

# Precomputare ciur in O(p \* log log p)

move $a0, $s0

jal compute\_sieve

# Precomputare numere prime in O(p)

move $a0, $s0

jal compute\_primes

# $s1 -> sieve

la $s1, sieve

add $s1, $s1, $s0

lb $s1, ($s1)

# daca sieve[p] == prime, sari la p\_good

beq $s1, 0, p\_good

li $v0, PRINT\_STR\_SYSCALL

la $a0, notPrimeMessage

syscall

j exit

p\_good:

# Cautam generatorul grupului (Zp\*, \*)

# Numarul de numere prime pana la p este ~ p / ln(p)

# Complexitate cautare O(p / ln(p) \* (p - 2)) = O(p ^ 2 / ln(p))

la $a0, foundGMessage

li $v0, PRINT\_STR\_SYSCALL

syscall

move $a0, $s0

jal find\_generator

# $s1 -> g

move $s1, $v0

move $a0, $v0

li $v0, PRINT\_INT\_SYSCALL

syscall

la $a0, newline

li $v0, PRINT\_STR\_SYSCALL

syscall

# citim mesajul pentru criptare

li $v0, PRINT\_STR\_SYSCALL

la $a0, readInputMessage

syscall

la $a0, input

li $v0, READ\_STR\_SYSCALL

syscall

# afisam mesajul criptat

la $a0, encryptedMessage

li $v0, PRINT\_STR\_SYSCALL

syscall

la $a0, input

la $a1, input

move $a2, $s0

move $a3, $s1

jal encrypt

la $a0, input

li $v0, PRINT\_STR\_SYSCALL

syscall

la $a0, newline

li $v0, PRINT\_STR\_SYSCALL

syscall

# afisam mesajul decryptat

la $a0, decryptedMessage

li $v0, PRINT\_STR\_SYSCALL

syscall

la $a0, input

la $a1, input

move $a2, $s0

move $a3, $s1

jal decrypt

la $a0, input

li $v0, PRINT\_STR\_SYSCALL

syscall

la $a0, newline

li $v0, PRINT\_STR\_SYSCALL

syscall

exit:

li $v0, EXIT\_SYSCALL

syscall

p\_negative:

la $a0, negativePMessage

li $v0, PRINT\_STR\_SYSCALL

syscall

la $a0, newline

li $v0, PRINT\_STR\_SYSCALL

syscall

j exit

p\_too\_big:

la $a0, pTooBigMessage

li $v0, PRINT\_STR\_SYSCALL

syscall

la $a0, newline

li $v0, PRINT\_STR\_SYSCALL

syscall

j exit

p\_is\_two:

la $a0, pIsTwoMessage

li $v0, PRINT\_STR\_SYSCALL

syscall

la $a0, newline

li $v0, PRINT\_STR\_SYSCALL

syscall

j exit

compute\_sieve:

# $a0 -> p

# $t0 -> adresa sieve

la $t0, sieve

# $t1 -> folosit temporar pentru stocat valoarea 1 (neprim)

li $t1, 1

sb $t1, 0($t0)

sb $t1, 1($t0)

# $t1 -> i = 2 .. p

li $t1, 2

loop\_1:

bgt $t1, $a0, end\_1

# $t2 -> temporar pentru sieve[i]

add $t0, $t0, $t1

lb $t2, ($t0)

sub $t0, $t0, $t1

# daca sieve[i] == prim, mergi in loop\_2

beq $t2, 1, end\_2

# $t2 -> j = i \* 2 .. p, din i in i

move $t2, $t1

add $t2, $t2, $t1

loop\_2:

bgt $t2, $a0, end\_2

# $t3 -> temporar pentru valoarea 1 (neprim)

li $t3, 1

# sieve[j] = neprim

add $t0, $t0, $t2

sb $t3, ($t0)

sub $t0, $t0, $t2

add $t2, $t2, $t1

j loop\_2

end\_2:

addi $t1, $t1, 1

j loop\_1

end\_1:

jr $ra

compute\_primes:

# $a0 -> p

# $t0 -> sieve

# $t1 -> i = 2 .. p

# $t2 -> primes

li $t1, 2

loop\_3:

bgt $t1, $a0, end\_3

# $t0 -> adresa sieve

la $t0, sieve

add $t0, $t0, $t1

lb $t0, ($t0)

#and $t0, $t0, 1

# data sieve[i] == neprim, omite

beq $t0, 1, skip\_prime

# altfel adauga in lista

# $t0 -> primesCnt

lw $t0, primesCnt

li $t2, 4

mult $t0, $t2

mflo $t0

# $t2 -> adresa primes

la $t2, primes

add $t2, $t2, $t0

# primes[primesCnt++] = i

sw $t1, ($t2)

lw $t0, primesCnt

addi $t0, $t0, 1

# $t2 -> adresa primesCnt

la $t2, primesCnt

sw $t0, ($t2)

skip\_prime:

add $t1, $t1, 1

j loop\_3

end\_3:

jr $ra

find\_generator:

# $a0 -> p

# $t0 -> i = 0..primesCnt - 2

# $t1 -> j = 0..p-4

# $v0 -> generatorul g al lui Zp

# g este un numar prim mai mic decat p sau 1 in cazul in care p = 2

beq $a0, 2, return\_g\_1

li $t0, 0

loop\_4:

# $t2 -> primesCnt - 2

lw $t2, primesCnt

sub $t2, $t2, 2

bgt $t0, $t2, end\_4

# $t2 -> primes[i] -> g

li $t2, 4

mult $t0, $t2

mflo $t2

la $t3, primes

add $t2, $t2, $t3

lw $t2, ($t2)

# $t3 -> rezultat ridicare la putere

move $t3, $t2

# calculam g ^ 2, g ^ 3, .., g ^ (p - 2) mod p

# daca niciunul din rezultate nu este congruent cu 1

# atunci ordinul lui g este p - 1, deci g este generator

li $t1, 0

loop\_5:

# $t4 -> p - 4

sub $t4, $a0, 4

bgt $t1, $t4, end\_5

# result = result \* result % p

mult $t2, $t3

mflo $t3

rem $t3, $t3, $a0

bne $t3, 1, skip\_g\_failure

li $t2, -1

j end\_5

skip\_g\_failure:

addi $t1, $t1, 1

j loop\_5

end\_5:

beq $t2, -1, g\_not\_found

move $v0, $t2

j end\_4

g\_not\_found:

addi $t0, $t0, 1

j loop\_4

end\_4:

jr $ra

return\_g\_1:

li $v0, 1

jr $ra

encrypt:

subu $sp, $sp, 16

sw $ra, 0($sp)

sw $s0, 4($sp)

sw $s1, 8($sp)

sw $s2, 12($sp)

# $a0 -> input[]

# $a1 -> output[]

# $a2 -> p

# $a3 -> g

# $s0 -> i = 0 .. pana cand input[i] == 0 sau 10

li $s0, 0

loop\_6:

# $t0 -> input[i]

add $t0, $a0, $s0

lb $t0, ($t0)

beq $t0, 0, end\_6

beq $t0, 10, end\_6

subu $t0, $t0, 65 # input[i] - 'A'

# salvam argumentele

move $s1, $a0

move $s2, $a1

# pow\_mod(g, char, p)

move $a0, $a3

move $a1, $t0

# $a2 este deja p

jal pow\_mod

move $t0, $v0

# restauram argumentele

move $a0, $s1

move $a1, $s2

addi $t0, $t0, 65

# $t1 -> output[i] = char

add $t1, $a1, $s0

sb $t0, ($t1)

addi $s0, $s0, 1

j loop\_6

end\_6:

# $t0 -> output[len(input)] = 0

add $t0, $a1, $s0

sb $zero, ($t0)

lw $ra, 0($sp)

lw $s0, 4($sp)

lw $s1, 8($sp)

lw $s2, 12($sp)

addiu $sp, $sp, 8

jr $ra

pow\_mod:

# $a0 -> baza

# $a1 -> exponent

# $a2 -> modulo

beq $a1, 0, return\_pow\_1

beq $a1, 1, return\_pow\_base

subu $a1, $a1, 2

# $v0 -> rezultat

rem $v0, $a0, $a2

# $t0 -> i = 0 .. exponent - 2

li $t0, 0

loop\_7:

bgt $t0, $a1, end\_7

mult $v0, $a0

mflo $v0

rem $v0, $v0, $a2

addi $t0, $t0, 1

j loop\_7

end\_7:

jr $ra

return\_pow\_1:

li $v0, 1

jr $ra

return\_pow\_base:

rem $v0, $a0, $a2

jr $ra

decrypt:

subu $sp, $sp, 16

sw $ra, 0($sp)

sw $s0, 4($sp)

sw $s1, 8($sp)

sw $s2, 12($sp)

# $a0 -> input[]

# $a1 -> output[]

# $a2 -> p

# $a3 -> g

# $s0 -> i = 0 .. pana cand input[i] == 0 sau 10

li $s0, 0

loop\_8:

# $t0 -> input[i]

add $t0, $a0, $s0

lb $t0, ($t0)

beq $t0, 0, end\_8

beq $t0, 10, end\_8

subu $t0, $t0, 65 # input[i] - 'A'

# salvam argumentele

move $s1, $a0

move $s2, $a1

# log\_mod(g, char, p)

move $a0, $a3

move $a1, $t0

# $a2 este deja p

jal log\_mod

move $t0, $v0

# restauram argumentele

move $a0, $s1

move $a1, $s2

addi $t0, $t0, 65

# $t1 -> output[i] = char

add $t1, $a1, $s0

sb $t0, ($t1)

addi $s0, $s0, 1

j loop\_8

end\_8:

# $t0 -> output[len(input)] = 0

add $t0, $a1, $s0

sb $zero, ($t0)

lw $ra, 0($sp)

lw $s0, 4($sp)

lw $s1, 8($sp)

lw $s2, 12($sp)

addiu $sp, $sp, 8

jr $ra

log\_mod:

# log\_b(a) modulo n

# $a0 -> baza

# $a1 -> argument

# $a2 -> modulo

# $t0 -> rezultat

li $t0, 1

# $t1 -> modulo - 2

subu $t1, $a2, 2

# $v0 -> i = 0 .. modulo - 2

li $v0, 0

loop\_9:

bgt $v0, $t1, end\_9

beq $t0, $a1, end\_9

mult $t0, $a0

mflo $t0

rem $t0, $t0, $a2

addi $v0, $v0, 1

j loop\_9

end\_9:

jr $ra