INSTITUTO POLITÉCNICO NACIONAL ESCUELA SUPERIOR DE CÓMPUTO

Análisis de Algoritmos

Práctica 1

Algoritmos Básicos

Edgar Adrián Nava Romo

Maestra: Sandra Díaz Santiago
Grupo: 3CM3

1. Implementa un algoritmo para factorizar un número entero n. Prueba tu programa con enteros de 5, 10, 15, 20, 25 y 30 dígitos.

Algoritmo:

1. Elegir un entero aleatorio a entre 1 y n-3.
2. Elegir un entero aleatorio s entre 0 y n-1.
3. Inicializar U=V=s;
4. Calcular U=F(U), V=F(F(V)));
5. Si MCD(U-V, n) = 1, volver al punto 4.
6. Si MCD(U-V, n) = n, volver al punto 1.
7. MCD(U-V, n) es un factor de n.

Veamos un ejemplo para n=534, empezando por ejemplo con x=2.

$F(2) = 2^2 \bmod 534 = 4$
$F(4) = 4^2 \bmod 534 = 16$
$F(16) = 16^2 \bmod 534 = 256$
$F(256) = 256^2 \bmod 534 = 388$
$F(388) = 290^2 \bmod 534 = 490$
$F(490) = 120^2 \bmod 534 = 334$
$F(334) = 188^2 \bmod 534 = 484$
$F(484) = 137^2 \bmod 534 = 364$
$F(364) = 35^2 \bmod 534 = 64$
$F(64) = 16^2 \bmod 534 = 358$
$F(358) = 290^2 \bmod 534 = 4$

Aquí se puede observar como se cierra un ciclo entero, es decir, $290^2 = 2^2 \bmod 534$, este algoritmo ayuda a encontrar las llamadas “colisiones”, las cuales en realidad son factores de n.

```c
/*                                  Ejercicio 2
 *                  Factorizes N using Pollard's Rho Algorithm
 *
 * We get as parameters:
 * N: Number given by user to factorize
 * factors[]: Created array to store all the factors of N
 * Steps:
 * We search in the table of primes in "Constants.h" all the n to save time
 *
 * Complexity: its expected running time is proportional to the square root of the
 * size of the smallest prime factor of the composite number being factorized.
 * Due to the use of probabilistic numbers in this algorithm, the complexity in Pollard's Roh
 * algorithm is O(n^1/4 polylog(n)) with a probability of 0.5.
 */

void factorize(mpz_t N, mpz_t factors[]) {
  mpz_t x;
  int num_factors = 0;
  num_factors = find_in_table(N, factors);
  int result;
  mpz_init(x);
  do {
      /** mpz_probab_prime_p performs some trial divisions, a Baillie-PSW probable prime test,then reps-24
      *  Miller-Rabin probabilistic primality tests. A higher reps value will reduce the chances of a non-prime being identified as
      *  “probably prime”. A composite number will be identified as a prime with an asymptotic probability of less than 4^(-reps).
      *   Reasonable values of reps are between 15 and 50. **/
    if(mpz_probab_prime_p(N, 50)) {

      gmp_printf(" %Zd ", N);
      mpz_set(x,N);
      mpz_set_ui(N, 1);
      break;
    }
    result = pollards(N, factors, num_factors);
    if (!result) {
      break;
    }
    num_factors++;
  } while (mpz_cmp_si(N, 1) != 0);
  // If N != 1 (it's not fully factorized)
  if (mpz_cmp_si(N, 1)) {
    printf("Aborted, factorization not possible \n");
  }
  else {
    print_factors(factors, num_factors, x);
  }
  printf("\n");
}
```

Pollard's Rho Function

```c
int pollards(mpz_t N, mpz_t factors[], int num_factors) {

  // Initialize random number container
  mpz_t xi_last;
  mpz_init(xi_last);

  // Initialize and get random first factor
  gmp_randstate_t rand_state;
  gmp_randinit_default(rand_state);
  gmp_randseed_ui(rand_state, time(NULL));

  // Set random number
  mpz_urandomm(xi_last, rand_state, N);
  //Get last random factor
  mpz_t x2i_last;
  mpz_init(x2i_last);
  nextprime(x2i_last, xi_last, N);

  mpz_t xi, x2i, diff;
  mpz_init(xi); mpz_init(x2i); mpz_init(diff);

  mpz_t d;
  mpz_init(d);

  mpz_t count;
  mpz_init_set_ui(count, 0);
  mpz_t limit;
  mpz_init(limit);
  if (mpz_sizeinbase(N,2) > 40) {
    mpz_set_ui(limit, 100000);
  } else if (mpz_sizeinbase(N,2) > 20) {  //Size in bits
    mpz_set_ui(limit, 90000);
  } else {
    mpz_set_ui(limit, 20000);
  }

  while(mpz_cmp(count, limit) < 0) {
    nextprime(xi, xi_last, N);

    nextprime(x2i, x2i_last, N);
    nextprime(x2i, x2i, N);

    mpz_sub(diff, x2i, xi);
    mpz_gcd(d, diff, N);

    if(mpz_cmp_si(d, 1) > 0) {
      mpz_set(factors[num_factors],d);
      mpz_fdiv_q(N, N, d);
      return 1;
    }
    mpz_set(xi_last, xi);
    mpz_set(x2i_last, x2i);
    mpz_add_ui(count,count,1);
  }
  // Clear variables
  mpz_clear(xi_last); mpz_clear(x2i_last); mpz_clear(xi); mpz_clear(x2i); mpz_clear(diff);

  return 0;
}
```

2. Implementa una función para encontrar el máximo común divisor de dos enteros, encontrando la factorización de cada entero. Prueba tu función con enteros de 5, 10, 15, 20, 25 y 30 dígitos.

```
* factoresFirstN -> number of factors that are in factors[]
* factoresSecondN -> number of factors that are in factors2[]
* factors[] -> array of factors in number "a"
* factors2[] -> array of factors in number "b"
* This function first detects wich number has more factors, and store it in fac1 or fac2 depending the case
* then, we took the largest array and compare it with the other, finding the common numbers in both arrays
* At last, all the common numbers are multiplied and stored in "add".
* Complexity: The complexity of the alforithm to compare the two arrays is of O(n+m) at the worst case
***/
void GCD_Factors(int factoresFirstN, int factoresSecondN, mpz_t factors[], mpz_t factors2[]){
  int i = 0, j = 0, savePosition = 0, fact1, fact2;
  mpz_t add, fac1[200], fac2[200];
  mpz_init(add);
  mpz_set_ui(add,1);
  printf("\n\nCommon Factors = ");
  for (size_t k = 0; k < 200; k++) {
    mpz_init(fac1[k]);
    mpz_init(fac2[k]);
  }
  if (mpz_cmp(factors2[100], factors[100]) == 0) {
    gmp_printf(" * %Zd", factors[100]);
    mpz_mul(add,add,factors[100]);
  }
  if (factoresFirstN >= factoresSecondN) {
    fact1 = factoresFirstN;
    fact2 = factoresSecondN;
    for (size_t k = 0; k < 100; k++) {
      mpz_set(fac1[k], factors[k]);
      mpz_set(fac2[k], factors2[k]);
    }

  } else{
    fact2 = factoresFirstN;
    fact1 = factoresSecondN;
    for (size_t k = 0; k < 100; k++) {
      mpz_set(fac1[k], factors2[k]);
      mpz_set(fac2[k], factors[k]);
    }
  }
  while(fact1 > i && fact2 > j){
    if (mpz_cmp(fac1[i], fac2[j]) < 0) {
      i++;
    }else if(mpz_cmp(fac2[j], fac1[i])){
      j++;
    } else {
      gmp_printf(" * %Zd", fac1[i]);
      mpz_mul(add,add,fac1[i]);
      i++;
      j++;
    }
  }
  if(mpz_get_ui(add) == 1)
  printf(" is the only Factor in common\n");
  else
  gmp_printf("\nGCD = %Zd\n", add);
```

```
}
```

```c
/**                      nextprime function
* here, we're gonna get the next prime according to the formula (x^2+1) mod N,
* this will find the prime number next to N and N = nextprime to get factors,
* There are 2 ways to solve it, the (x^2 +1) mod N will repeat somewhere or
**/
void nextprime(mpz_t next, mpz_t prev, mpz_t N) {
  mpz_pow_ui(next, prev, 2); // X^2
  mpz_add_ui(next, next, 1); // X^2 + 1
  mpz_mod(next, next, N);    // (X^2 + 1) mod N
}
```

3. Implementa una función para encontrar el máximo común divisor de dos enteros, utilizando la regla de Euclides. Prueba tu función con enteros de 5, 10, 15, 20, 25 y 30 dígitos.

```c
/***                                 Ejercicio 4
*            Implement a function with Euclidean Factorization Algorithm
* We get as parameters:
* numero1 -> first number given by user
* numero2 -> second number given by user
* Description:
***/
void euclidean(mpz_t numero1, mpz_t numero2){
  mpz_t a, b, r;
  mpz_init(a); mpz_init(b); mpz_init(r);

  mpz_set_ui(r,1);
  if (mpz_cmp(numero1, numero2) > 0) {
    mpz_set(a,numero1);
    mpz_set(b,numero2);
  } else{
    mpz_set(b,numero1);
    mpz_set(a,numero2);
  }
  while (mpz_cmp_ui (b, 0)!= 0){
    mpz_sub(r, a, b); // r = a - b

    if (mpz_cmp(r, b) >= 0){ mpz_set (a,r); }
    else { mpz_set (a,b); mpz_set (b,r); }

  }
  gmp_printf(": %Zd\n" , a);
  mpz_clear(a); mpz_clear(b); mpz_clear(r);
}
```

Capturas de Pantalla en Funcionamiento

Con 5 Dígitos:

```
Insert a = 12346
Insert b = 23456

GCD with Euclides Algorithm: 2

Factorization:

a = * 2 * 6173
b = * 2 * 2 * 2 * 2 * 2 * 733

Common Factors =  * 2 * 1
GCD = 2

took 0.015763 seconds to execute
```

Con 10 Dígitos:

```
Insert a = 1234567894
Insert b = 1112345678

GCD with Euclides Algorithm: 2

Factorization:

a = * 2 * 7 * 47 * 479 * 3917
b =  5197877 * 2 * 107

Common Factors =  * 2 * 1
GCD = 2

took 0.010179 seconds to execute
```

Con 15 Dígitos:

```
Insert a = 234567564738294
Insert b = 987654321876542

GCD with Euclides Algorithm: 2

Factorization:

a =  144260494919 * 2 * 3 * 271
b =  493827160938271 * 2

Common Factors =  * 2
GCD = 2

took 0.001313 seconds to execute
```

Con 20 Dígitos:

```
Insert a = 98765432109876543212
Insert b = 99999999998765432104

GCD with Euclides Algorithm: 4

Factorization:

a =  107822524137419807 * 2 * 2 * 229
b =  876548519369 * 2 * 2 * 2 * 7 * 11 * 43 * 59 * 73

Common Factors =  * 2 * 2
GCD = 4

took 0.036298 seconds to execute
```

Con 25 Dígitos:

```
Insert a = 87345909857463728194
Insert b = 98765748392019283746

GCD with Euclides Algorithm: 2

Factorization:

a =  43672954928731864097 * 2
b =  609665113530983233 * 2 * 3 * 3 * 3 * 3

Common Factors =  * 2
GCD = 2

took 0.001861 seconds to execute
```

Con 30 Dígitos:

```
Insert a = 283746575849302928374657483928
Insert b = 987657483928374657483928172384

GCD with Euclides Algorithm: 8

Factorization:

a =  2223514295773 * 2 * 2 * 2 * 241 * 9358169 * 7072823
b =  10388521162154731755763297 * 2 * 2 * 2 * 2 * 2 * 2971

Common Factors =  * 2 * 2 * 2
GCD = 8

took 0.003933 seconds to execute
```