Tema 3. Criptografie

1. Demonstrați că dacă $n = \prod_{i=1}^{k} p_i^{\alpha_i}$ și $a^{p_i} \equiv a \pmod{p_i}$, $\forall p_i$, atunci $a^n \equiv a \pmod{n}$

2. Folosind exercițiul anterior, arătați că numerele 1729, 10585 și 75361 sunt numere Carmichael.

3. Arătați că dacă $2^n - 1$ este prim, atunci n este prim

4. Demonstrați legea reciprocității pătratice.

5. Implementați o funcție care să calculeze simbolul lui Jacobi

6. Implementați algoritmul Solovay-Strassen

7. Implementați algoritmul AKS

8. Descrieți simbolul lui Kronecker

9. Algoritmul cu primalitate

15. Folosiți algoritmul Miller-Rabin pentru a verifica dacă numărul 93281 este prim sau compus.

5.
```
#include <iostream>
#include <stdexcept>   -> using namespace std;

int jacobiSymbol (int a, int n) {
    if (n <= 0 || n % 2 == 0) {
        throw invalid_argument ("n trebuie să fie impar pozitiv");
    }
    a = a % n;
    int result = 1;
    while (a != 0) {
        while (a % 2 == 0) {
            a /= 2;
            int r = n % 8;
            if (r == 3 || r == 5) {
                result = -result;
            }
        }
        swap (a, n);
        if (a % 4 == 3 && n % 4 == 3) {
            result = -result;
        }
```

```
        a = a % n;
    }
    if (n == 1) {
        return result;
    } else {
        return 0;
    }
}
int main() {
    int a = 5; int n = 11;
    try {
        int jacobi = jacobiSymbol(a, n);
        cout << "Simbolul Jacobi (" << a << "/" << n << ") = " << jacobi << endl;
    } catch (const invalid_argument & e) {
        cout << e.what() << endl;
    } return 0;
}
```

6)

```
#include <iostream>
#include <cstdlib>
#include <cmath>
using namespace std;
// functia care calculeaza simbolul Jacobi
int jacobiSymbol(int a, int n) {
    if (n <= 0 || n % 2 == 0)
        return 0;
    int result = 1;
    a %= n;
    while (a != 0) {
        while (a % 2 == 0) {
            a /= 2;
            int r = n % 8;
            if (r == 3 || r == 5)
                result = -result;
        }
        swap(a, n);
        if (a % 4 == 3 && n % 4 == 3)
            result = -result;
        a %= n;
    }
```

```
    if (n == 1)
        return result;
    return 0;
}
// functia care verifica daca un numar este prim folosind algoritmul Solovay-Strassen
bool SolovayStrassen (int n, int k) {
    if (n < 2) return false;
    if (n != 2 && n % 2 == 0) return false;
    for (int i = 0; i < k; i++) {
        int a = rand() % (n-1) + 1; // alege un numar aleator intre 1 si n-1
        int x = jacobiSymbol (a, n);
        int y = (int) pow (a, (n-1)/2) % n;
        if (x == 0 || y != x) return false;
    } return true;
}
int main () {
    int n;
    cout << "Introduceti numarul pentru a verifica daca este prim: ";
    cin >> n;
    int k;
    cout << " Introduceti numarul de iteratii (k): "; cin >> k;
    if (solovayStrassen (n, k))
        cout << n << " este probabil prim. \n";
    else
        cout << n << " este compus. \n";
    return 0;
}
```

7)
```
#include <iostream>
#include <vector>
#include <cmath>
using namespace std;
// funcția pt. a calcula combinările C(n,k) % p
long long binomialCoeff(int n, int k, int p) {
    vector<vector<long long>> C(n+1, vector<long long>(k+1, 0));
    for(int i=0; i<=n; i++) {
        for(int j=0; j<=min(i,k); j++) {
            if (j==0 || j==i)
                C[i][j]=1;
            else
                C[i][j] = (C[i-1][j-1] + C[i-1][j]) % p;
        }
    } return C[n][k];
}
// funcția pt. a verifica dacă n este puterea unei baze prime
bool isPowerOfPrime(int n) {
    // verificăm dacă n este puterea unui număr prim
    if (n<=1) return false;
    // determinăm factorii primi ai lui n;
    for(int i=2; i<=sqrt(n); i++){
        if(n%i==0) {
            // verificăm dacă i este prim
            bool isPrime = true;
            for(int j=2; j<=sqrt(i); j++) {
                if(i%j==0){
                    isPrime = false;
                    break;
                }
            }
            // dacă i este prim și n/i este o putere a lui i,
returnăm true
            if(isPrime && pow(i, floor(log(n)/log(i)))==n)
                return true;
        }
    } return false;
}
```

```
// funcția pt. a verifica dacă n este prim
bool isPrime(int n){
    // verificăm dacă n este mai mic sau egal cu 1
    if(n<=1) return false;
    // verificăm dacă n are vreun divizor în intervalul [2, sqrt(n)]
    for(int i=2; i<=sqrt(n); i++){
        if(n%i==0) return false;
    } return true;
}
// funcția pt. a implementa alg. AKS
bool isPrime_AKS(int n){
    // 1. verificăm dacă n este puterea unui număr prim
    if(isPowerOfPrime(n)) return false;
    // 2. găsim cel mai mic r care satisface ord(r,n) > log2(n)^2.
    int r=2;
    while(r<=log2(n)*log2(n)){
        // verificăm dacă r și n sunt coprime
        if(gcd(r,n)!=1) return false;
        // 3. verificăm dacă (x-r)^n este echivalent cu x^n-
r (mod x^r-1, n)
        for(int a=1; a<=sqrt(phi(r))*log2(n); a++){
        // (x-r)^n
            long long par1 = binomialCoeff(n,a,n);
            long long par2 = binomialCoeff(n,a,r);
            if(par1%n!=par2%n) return false;
        } r++;
    }
    // 4. dacă n ≤ r, returnăm true
    if(n<=r) return true;
    return false;
}
int main(){
    int n; cout<<"Introduceți numărul n pt. a verifica dacă
este prim: "; cin>>n;
```

```
    if(isPrime_AKS(n))
        cout<< n << "este prim" << endl;
    else
        cout<< n << "nu este prim" << endl;
    return 0;
}
```

g)
```
#include <iostream>
#include <cmath>
using namespace std;
// funcția pt. calculul puterii în mod eficient
long long int power (long long int x, unsigned long long int y,
long long int p) {
        long long int res = 1; // inițializare rezultat
        x = x % p; // modificăm x dacă este mai mare sau egal cu p
        while (y > 0) {
            // dacă y este impar, înmulțim rezultatul cu x
            if (y & 1) res = (res * x) % p;
            // y trebuie să fie acum par
            y = y >> 1; // y = y/2
            x = (x * x) % p;
        } return res;
}
// funcția pt. testul Miller Rabin
bool millerRabinTest (long long int d, long long int n) {
        // alegem un număr aleatoriu între [2, n-2]
        long long int a = 2 + rand() % (n-4);
        // calculăm a^d % n
        long long int x = power (a, d, n);
        if (x == 1 || x == n-1) return true;
        // continuăm să calculăm a^(2^r * d) % n pt. a verifica dacă este
compus
        while (d != n-1) {
            x = (x * x) % n;
            d *= 2;
            if (x == 1) return false;
            if (x == n-1) return true;
        }
```

```
    // numar compus
    return false;
}
// functia principala pt. testul primalitatii folosind Miller Rabin
bool isPrime (long long int n, int k){
        // cazuri de baza
        if (n <= 1 || n == 4) return false;
        if (n <= 3) return true;
        // gasim d ai. n-1 = d*2^r pt. un anumit d si un r max
        long long int d = n-1;
        while (d%2 == 0)
            d /= 2;
        // repetam testul de k ori pt. a creste precizia
        for (int i = 0; i<k; i++)
            if (!millerRabinTest(d, n)) return false;
        return true;
}
int main(){
    long long int numar = 73281;
    int k=5; // nr. de ~~itte~~ iteratii
    if (isPrime (numar, k)){
        cout << "numar << " prim";
    else
        cout << numar << " compus";
    return 0;
}
```