PRAKTIKUM KRIPTOGRAFI

Tugas 3

Disusun Oleh:

Prames Ray Lapian *140810210059*

UNIVERSITAS PADJADJARAN

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

PROGRAM STUDI TEKNIK INFORMATIKA

JATINANGOR

2023

SOAL

Tugas

1. Kumpulkan Exercise (dalam format pdf) tadi di Classroom
2. Buatlah program untuk enkripsi, dekripsi, dan mencari kunci Hill Cipher (bahasa pemrograman bebas)
3. Push program tersebut ke repository **NPM-Kripto23** dan sertakan juga screenshot di dalamnya.
4. Jelaskan program yang sudah dibuat di dalam 1 file pdf lalu kumpulkan di classroom

Jawaban

Source Code:

```
#include <iostream>
using namespace std;

int key[3][3];

int mod26(int x)
{
  return x >= 0 ? (x%26) : 26-(abs(x)%26);
}

// Find Determinant Function
int findDet(int m[3][3], int n)
{
  int det;
  if(n == 2)
  {
    det = m[0][0] * m[1][1] - m[0][1]*m[1][0];
  }
  else det = 0;

  return mod26(det);
}

// Find Inverse Matrix Function
int findDetInv(int R, int D = 26)
{
  int i = 0;
  int p[100] = {0,1};
  int q[100] = {0};

  while(R!=0)
  {
    q[i] = D/R;
    int oldD = D;
    D = R;
    R = oldD%R;
```

```
    if(i>1)
    {
        p[i] = mod26(p[i-2] - p[i-1]*q[i-2]);
    }

    i++;
  }

  if (i == 1)
  {
    return 1;
  }
  else
  {
    return p[i] = mod26(p[i-2] - p[i-1]*q[i-2]);
  }
}

// Find GCD Function
int gcd(int m, int n)
{
  if (n > m)
  {
    swap(m,n);
  }

  do
  {
    int temp = m % n;
    m = n;
    n = temp;
  } while (n != 0);

  return m;
}

// Multiplying Matrix Function
void multM(int a[1000][3], int a_rows, int a_cols,  int b[1000][3], int
b_rows, int b_cols, int res[1000][3])
{
```

```
  for(int i=0; i < a_rows; i++)
  {
    for(int j=0; j < b_cols; j++)
    {
      for(int k=0; k < b_rows; k++)
      {
        res[i][j] += a[i][k]*b[k][j];
      }
      res[i][j] = mod26(res[i][j]);
    }
  }
}

// Find the Key Function
void findKey()
{
  string plainteks, cipherteks;
  int key[2][2], det, detInv, adj[2][2], plainteksInv[2][2],
plainMatrix[2][2], cipMatrix[2][2], counter;
  int p, c;
  int transpose[2][2];

  cout << "Input Plaintext : ";
  cin.ignore();
  getline(cin, plainteks);

  // Plaintext assigned to Plainmatrix
  counter = 0;
  for(int i = 0; i < 2; i++)
  {
    for(int j = 0; j < 2; j++)
    {
      p = toupper(plainteks[counter]) - 65;
      plainMatrix[i][j] = p;
      counter++;
    }
  }

  cout << "Input Ciphertext : ";
  getline(cin, cipherteks);
```

```
  // Ciphertext assigned to Ciphermatrix
  counter = 0;
  for(int i = 0; i < 2; i++)
  {
    for(int j = 0; j < 2; j++)
    {
      c = toupper(cipherteks[counter]) - 65;
      cipMatrix[i][j] = c;
      counter++;
    }
  }

  // Determinant
  det = (plainMatrix[0][0] * plainMatrix[1][1]) - (plainMatrix[0][1] *
plainMatrix[1][0]);
  if(gcd(det, 26) == 1)
  {
    detInv = findDetInv(det, 26);

    // Calculate Adjoint
    adj[0][0] = plainMatrix[1][1];
    adj[0][1] = (-1) * plainMatrix[0][1];
    adj[1][0] = (-1) * plainMatrix[1][0];
    adj[1][1] = plainMatrix[0][0];

    for(int i = 0; i < 2; i++)
    {
      for(int j = 0; j < 2; j++)
      {
        plainteksInv[i][j] = detInv * adj[i][j];
        if (plainteksInv[i][j] < 0)
        {
          plainteksInv[i][j] = 26 - (abs(plainteksInv[i][j])%26);
        }
        else
        {
          plainteksInv[i][j] = plainteksInv[i][j];
          plainteksInv[i][j] = plainteksInv[i][j] % 26;
        }
```

```
      }
    }

    // Find the Key
    for(int i = 0; i < 2; i++)
    {
      for(int j = 0; j < 2; j++)
      {
        key [i][j] = 0;
        for(int k = 0; k < 2; k++)
        {
          key [i][j] += (plainteksInv[i][k] * cipMatrix[k][j]);
        }
        key [i][j] %= 26;
      }
    }


    for (int i = 0; i < 2; i++)
    {
      for (int j = 0; j < 2; j++)
      {
        transpose[j][i] = key[i][j];
      }
    }

    for(int i = 0; i < 2; i++)
    {
      for (int j = 0; j < 2; j++)
      {
        cout << (transpose[i][j]) << "\t";
      }
      cout << endl;
    }
  }
  else
  {
    cout << "Determinant not relative " << endl;
    cout << "Key not found" << endl << endl;
  }
```

```
}

// Find Invers Function
void findInv(int m[3][3], int n, int m_Inv[3][3] )
{
  int adj[3][3] = {0};

  int det = findDet(m, n);
  int detInv = findDetInv(det);

  if(n == 2)
  {
    adj[0][0] = m[1][1];
    adj[1][1] = m[0][0];
    adj[0][1] = -m[0][1];
    adj[1][0] = -m[1][0];
  }

  for(int i = 0; i < n; i++)
  {
    for(int j = 0; j < n; j++)
    {
      m_Inv[i][j] = mod26(adj[i][j] * detInv);
    }
  }
}

// Encryption Function
string encrypt(string pTeks, int n)
{
  // C = Cipher, P = Plain
  int P[1000][3] = {0};
  int C[1000][3] = {0};
  int pTeksIter = 0 ;

  while(pTeks.length()%n != 0)
  {
    pTeks += "x";
  }
```

```
  int row = (pTeks.length())/n;

  for(int i = 0; i < row; i++)
  {
    for(int j = 0; j < n; j++)
    {
      P[i][j] = pTeks[pTeksIter++] - 'a';
    }
  }

  multM(P, row, n, key, n, n, C);

  string cTeks = "";
  for(int i = 0; i < row; i++)
  {
    for(int j = 0; j < n; j++)
    {
      cTeks += (C[i][j] + 'a');
    }
  }

  return cTeks;
}

// Decryption Function
string decrypt(string cTeks, int n)
{
  int P[1000][3] = {0};
  int C[1000][3] = {0};
  int cTeksIter = 0;

  int row = cTeks.length()/n;

  for(int i = 0; i < row; i++)
  {
    for(int j = 0; j < n; j++)
    {
      C[i][j] = cTeks[cTeksIter++] - 'a';
    }
  }
```

```
  int k_Inv[3][3] = {0};
  findInv(key, n, k_Inv);
  multM(C, row, n, k_Inv, n, n, P);

  string pTeks = "";
  for(int i = 0; i < row; i++)
  {
    for(int j = 0; j < n; j++)
    {
      pTeks += (P[i][j] + 'a');
    }
  }

  return pTeks;
}

int main(void)
{
  bool stay = true;
  string pTeks, cTeks;
  int n;
  int menu;

  while(stay)
  {
    cout << "\n\n[ PROGRAM HILL CIPHER ] : " << endl;
    cout << "1. Enkripsi" << endl;
    cout << "2. Dekripsi" << endl;
    cout << "3. Cari Kunci" << endl;
    cout << "4. Keluar" << endl;
    cout << "> ";
      cin >> menu;

    switch(menu)
    {
      case 1:
        cout << "Teks : ";
        cin >> pTeks;
```

```
        cout << "Input Ordo Matrix Persegi K : ";
        cin >> n;

        for(int i = 0; i < n; i++)
        {
          for(int j = 0; j < n; j++)
          {
            cout << "Input Matrix K (" << i+1 << "," << j+1 << ") : ";
            cin >> key[i][j];
          }
        }

        cout << "\nPlain Teks : " << pTeks << endl;
        cout << "Cipher Teks : " << encrypt(pTeks, n) << endl;
        break;

      case 2:
        cout << "Teks : ";
        cin >> cTeks;

        cout << "Input Ordo Matrix Persegi K : ";
        cin >> n;

        for(int i = 0; i < n; i++)
        {
          for(int j=0; j<n; j++)
          {
            cout << "Input Matrix K (" << i+1 << "," << j+1 << ") : ";
            cin >> key[i][j];
          }
        }

        cout << "\nCipher Teks  : " << cTeks << endl;
        cout << "Plain Teks : " << decrypt(cTeks, n) << endl;
        break;

      case 3:
        cout << endl;
        findKey();
        break;
```

```
      case 4:
        return(0);

      default:
        cout << "\nInvalid" << endl;
        break;
    }
  }
}
```

Screenshot Program:

```
[ PROGRAM HILL CIPHER ] :
1. Enkripsi
2. Dekripsi
3. Cari Kunci
4. Keluar
> 1
Teks : PRAMES
Input Ordo Matrix Persegi K : 2
Input Matrix K (1,1) : 1
Input Matrix K (1,2) : 2
Input Matrix K (2,1) : 3
Input Matrix K (2,2) : 4

Plain Teks : PRAMES
Cipher Teks : qkmmis
```