Kodas reikalingomis dalimis:

Patikrinimas ar skaičius yra pirminis:

```
def isPrime(num):
    if num > 1:
        if num == 2:
            return True
        if num % 2 == 0:
            return False
        i = 3
        while i*i <= num:
            if num % i == 0:
                return False
            i += 2
        return True
    else:
        return False
```

Patikrinu dvejetą atskirai ar jis yra pirminis, tada tikrinu ar num dalinasi su visais nelyginiais skaičiais nuo 3 iki šaknies iš tikrinamo skaičiaus. Tai greičiausias metodas kokį radau nerealizuojant jūsų skaidrėse pateiktų algoritmų.

8 bitų p ir q generavimas:

```
def generate8BitPQ(e):
    if not isPrime(e):
        print("The given exponent is not a prime number")
        return
    p = 0
    q = 0
    check = 0
    while getGCD(e, p*q - p - q + 1)[0] != 1 or p*q < 0b1111111111111111:
        p = random.randrange(128, 256, 2)
        q = random.randrange(128, 256, 2)
        p = 227
        p = 239

        if p == q:
            p = 0
            q = 0

        check += 1
        if check == 1000:
            print ("Something is wrong, the program took too long")
            return
```

Generuojami p ir q taip, kad duota exponentė būtų tinkama ir kad modulis būtų 16 bitų skaičius (32767 < p*q < 65536). Nežinau kodėl, bet ne visi taip sugeneruoti skaičiai taip veikė, bet gavau porą veikiančių su kuriais atlinksiu šifravimą.

Teksto šifravimas:

```
def RSA_Encrypt(text, e, p = None, q = None, blockLength = None):
    if p == None and q == None:
        print("p and q not provided, they will be generated randomly")
        p, q = generate8BitPQ(e)
    elif p == None:
        print("Only one argument of p and q is provided, it will not be used, both p and q will be generated randomly")
        p, q = generate8BitPQ(e)

    n = p*q
    fi = n - p - q + 1

    d = calcReverseMod(e, fi)
    cipher = ""
    encodingLength = 3
    if blockLength == None:
        blockLength = len(str(n)) - 1


    print ("public key = ({}, {})".format(n, e))
    print ("private key = ({}, {})".format(n, d))

    asciiBlocks = []
    block = ""
    for char in text:
        tempOrd = str(ord(char))
        while len(tempOrd) < encodingLength:
            tempOrd = "0" + tempOrd
        asciiBlocks.append(tempOrd)

    asciiText = ''.join(asciiBlocks)[::-1]

    print("")
    print("Text:", text)
    print("The text in ASCII:", ''.join(asciiBlocks))

    blocks = textwrap.wrap(asciiText, blockLength)
    for i in range(len(blocks)):
        blocks[i] = blocks[i][::-1]
    blocks = blocks[::-1]
```

```
originalBlockLength = blockLength
blockLength = len(str(n))
for block in blocks:
    cipherBlock = str(getQuickRemainder(int(block), e, n))
    while len(cipherBlock) < blockLength:
        cipherBlock = "0" + cipherBlock

    cipher += cipherBlock

print ("Cipher:", cipher)
print ("Decyphered text:", RSA_Decrypt(cipher, n, d, blockLength, originalBlockLength))
print("")
print("-----------------------------------------------")
print("")

return cipher
```

Pagal jūsų patarimą šifruoju taip, kad gauti užšifruoti blokai bus didesni nei blokai su kuriais pradėjome jei bloko ilgis yra mažesnis jei modulio skaitmenų skaičius (kadangi 16 bitų modulis yra penkių skaitmėnų skaičius kai šifruojami blokai kurie yra mažesni pvz 4 skaitmenų ilgio, galimai gautas rezultatas bus 5 skaitmėnų), tokiais atvėjais dešifruojant yra gaunami blokai, kurie prasideda nuliais, tad prieš konveruojant iš ASCII atgal į tekstą reikia panaikinti tiek nulių iš blokų pradžios, kad blokai taptu pradinių blokų ilgio (pvz. užšifravus 8610 bloką, dešifruojant gaunamas 08610 blokas).

Dešifravimas:

```
def RSA_Decrypt(cipher, n, d, blockLength, originalBlockLength = None):
    blocks = textwrap.wrap(cipher, blockLength)

    if originalBlockLength == None:
        originalBlockLength = blockLength

    asciiText = ""
    for block in blocks:
        m = str(getQuickRemainder(int(block), d, n))
        while len(str(m)) < originalBlockLength:
            m = "0" + m
        asciiText += m

    print("Decyphered text in ASCII:", asciiText)

    text = ""
    asciiBlocks = textwrap.wrap(asciiText[::-1], 3)
    asciiBlocks = asciiBlocks[::-1]
    isFrontZeroes = True
    for block in asciiBlocks:
        block = block[::-1]
        while len(block) < 3:
            block = "0" + block
        if block != "000":
            isFrontZeroes = False
        if not isFrontZeroes:
            text = text + chr(int(block))

    return text
```

Atliekamas dešifravimas ar atliekami papildomi veiksmai aprašyti prie šifravimo.

Kur reikalinga naudojami greito kėlimo laipsnio ir atvikštinio elemento paieškos algoritmai suprogramuoti praeituose darbuose.

Išvestis:

```
RSA_Encrypt("Vilnius", 59, 131, 137, 4)
RSA_Encrypt("Marius Bieliauskas", 59, 131, 137, 4)

RSA_Encrypt("Vilnius", 59, 239, 227, 4)
RSA_Encrypt("Marius Bieliauskas", 59, 239, 227, 4)

RSA_Encrypt("Vilnius", 7, 11, 13, 3)
RSA_Encrypt("Daugai", 39827, 18409199, 18409201, 15)
RSA_Encrypt("Vilnius", 939391, 993319, 999331, 12)

RSA_Encrypt("Marius Bieliauskas", 59, 757, 1321, 6)
RSA_Encrypt("Vilnius", 59, 2, 49999, 5)
RSA_Encrypt("Marius Bieliauskas", 59, 2, 49999, 5)
```

Užšifruoju savo vardą ir žodį „Vilnius“ su skirtingomis p ir q reikšmėmis ir blokų ilgiais. Pirmi 4 variantai yra gauti naudojant 8 bitų p ir q su blokų ilgiu 4 ir pagal eksponentę 59 kuri gauta pagal užduoties salygą (p_{n+15}). Sekantys 3 yra skaidrėsė rodyti pavyzdžiai ir pagal duotą egzamino pavyzdinį klausimą parašytas miesto „Daugai“ pavadinimo šifravimas. Likę 3 yra pavyzdžai su didžiausiais šifravimui tinkamais moduliais kai modulis yra 6 ir 5 skaitmenų ilgio, kad parodyti, kad kai modulio ilgis ir bloko ilgis yra vienodi ir atitinka salygą m < n algoritmas veikia.

Rezultatai:

Pirmi 4:

```
public key = (17947, 59)
private key = (17947, 899)

Text: Vilnius
The text in ASCII: 086105108110105117115
Cipher: 000000699917553133201538106334
Decyphered text in ASCII: 000086105108110105117115
Decyphered text: Vilnius

-----------------------------------------------

public key = (17947, 59)
private key = (17947, 899)

Text: Marius Bieliauskas
The text in ASCII: 07709711410511711503206610510110810509711711510709711 5
Cipher: 120641351514900153810633412375024031557910253062851300712003040280 6334
Decyphered text in ASCII: 000770971141051171150320661051011081050971171151070971 15
Decyphered text: Marius Bieliauskas

-----------------------------------------------

public key = (54253, 59)
private key = (54253, 2735)

Text: Vilnius
The text in ASCII: 086105108110105117115
Cipher: 000003529624202230541165504257
Decyphered text in ASCII: 000086105108110105117115
Decyphered text: Vilnius

-----------------------------------------------

public key = (54253, 59)
private key = (54253, 2735)

Text: Marius Bieliauskas
The text in ASCII: 07709711410511711503206610510110810509711711510709711 5
Cipher: 435872032116357116550425746552488694649237247483443991604202019470 4257
Decyphered text in ASCII: 000770971141051171150320661051011081050971171151070971 15
Decyphered text: Marius Bieliauskas
```

3 pavyzdiniai:

```
public key = (143, 7)
private key = (143, 103)

Text: Vilnius
The text in ASCII: 086105108110105117115
Cipher: 070118004033118039080
Decyphered text in ASCII: 086105108110105117115
Decyphered text: Vilnius

-----------------------------------------------

public key = (338898644639999, 39827)
private key = (338898644639999, 18456601811963)

Text: Daugai
The text in ASCII: 068097117103097105
Cipher: 134110512159248177845645444842
Decyphered text in ASCII: 000000000000068097117103097105
Decyphered text: Daugai

-----------------------------------------------

public key = (992654469589, 939391)
private key = (992654469589, 344351421211)

Text: Vilnius
The text in ASCII: 086105108110105117115
Cipher: 529211959060867523619443
Decyphered text in ASCII: 000086105108110105117115
Decyphered text: Vilnius
```

3 mano pasirinkti:

```
public key = (999997, 59)
private key = (999997, 169139)

Text: Marius Bieliauskas
The text in ASCII: 077097114105117115032066105101108105097117115107097115
Cipher: 848087213752984095498730147409800604666702457673488992
Decyphered text in ASCII: 0770971141051171150320661051011081050971171151070971l5
Decyphered text: Marius Bieliauskas

-----------------------------------------------

public key = (99998, 59)
private key = (99998, 27965)

Text: Vilnius
The text in ASCII: 086105108110105117115
Cipher: 000000548537773705457921 7
Decyphered text in ASCII: 000086105108110105117115
Decyphered text: Vilnius

-----------------------------------------------

public key = (99998, 59)
private key = (99998, 27965)

Text: Marius Bieliauskas
The text in ASCII: 077097114105117115032066105101108105097117115107097115
Cipher: 233906510648235660508358295808005836633188391000107366 3
Decyphered text in ASCII: 0077097114105117115032066105101108105097117115107097115
Decyphered text: Marius Bieliauskas
```