

Лабораторная работа №2

Информационная безопасность

Якушевич А.Ю.

1 октября 2022

Российский университет дружбы народов, Москва, Россия

Информация

- Якушевич Артём Юрьевич
- студент кафедры прикладной информатики и теории вероятностей
- Российский университет дружбы народов
- 1132223466@rudn.ru



Вводная часть

- Широкое распространение шифрования данных
- Важность знания основ шифрования для работы в сфере информационных технологий

- Шифры перестановки
 - Столбцовая перестановка
 - Таблица Виженера

- Ознакомиться с шифрами перестановки
- Обучиться их программной реализации

- Курс “Математические основы информационной безопасности” в ТУИС
- Язык программирования python

Выполнение лабораторной работы

Столбцовая перестановка

```

// description
//
// @param {number} n - number of elements in the array
// @return {number} - the sum of the squares of the elements of the array
//
// Example:
// Input: 1
// Output: 1
// Input: 2
// Output: 5
// Input: 3
// Output: 14
// Input: 4
// Output: 30
// Input: 5
// Output: 55
// Input: 6
// Output: 91
// Input: 7
// Output: 140
// Input: 8
// Output: 204
// Input: 9
// Output: 285
// Input: 10
// Output: 385
// Input: 11
// Output: 506
// Input: 12
// Output: 648
// Input: 13
// Output: 811
// Input: 14
// Output: 995
// Input: 15
// Output: 1200
// Input: 16
// Output: 1427
// Input: 17
// Output: 1676
// Input: 18
// Output: 1947
// Input: 19
// Output: 2240
// Input: 20
// Output: 2555
// Input: 21
// Output: 2892
// Input: 22
// Output: 3251
// Input: 23
// Output: 3632
// Input: 24
// Output: 4035
// Input: 25
// Output: 4460
// Input: 26
// Output: 4907
// Input: 27
// Output: 5376
// Input: 28
// Output: 5867
// Input: 29
// Output: 6380
// Input: 30
// Output: 6915
// Input: 31
// Output: 7472
// Input: 32
// Output: 8051
// Input: 33
// Output: 8652
// Input: 34
// Output: 9275
// Input: 35
// Output: 9920
// Input: 36
// Output: 10587
// Input: 37
// Output: 11276
// Input: 38
// Output: 11987
// Input: 39
// Output: 12720
// Input: 40
// Output: 13475
// Input: 41
// Output: 14252
// Input: 42
// Output: 15051
// Input: 43
// Output: 15872
// Input: 44
// Output: 16715
// Input: 45
// Output: 17580
// Input: 46
// Output: 18467
// Input: 47
// Output: 19376
// Input: 48
// Output: 20307
// Input: 49
// Output: 21260
// Input: 50
// Output: 22235
// Input: 51
// Output: 23232
// Input: 52
// Output: 24251
// Input: 53
// Output: 25292
// Input: 54
// Output: 26355
// Input: 55
// Output: 27440
// Input: 56
// Output: 28547
// Input: 57
// Output: 29676
// Input: 58
// Output: 30827
// Input: 59
// Output: 31999
// Input: 60
// Output: 33193
// Input: 61
// Output: 34409
// Input: 62
// Output: 35646
// Input: 63
// Output: 36905
// Input: 64
// Output: 38186
// Input: 65
// Output: 39489
// Input: 66
// Output: 40814
// Input: 67
// Output: 42161
// Input: 68
// Output: 43530
// Input: 69
// Output: 44921
// Input: 70
// Output: 46334
// Input: 71
// Output: 47769
// Input: 72
// Output: 49226
// Input: 73
// Output: 50705
// Input: 74
// Output: 52206
// Input: 75
// Output: 53729
// Input: 76
// Output: 55274
// Input: 77
// Output: 56841
// Input: 78
// Output: 58430
// Input: 79
// Output: 60041
// Input: 80
// Output: 61674
// Input: 81
// Output: 63329
// Input: 82
// Output: 65006
// Input: 83
// Output: 66705
// Input: 84
// Output: 68426
// Input: 85
// Output: 70169
// Input: 86
// Output: 71934
// Input: 87
// Output: 73721
// Input: 88
// Output: 75530
// Input: 89
// Output: 77361
// Input: 90
// Output: 79214
// Input: 91
// Output: 81089
// Input: 92
// Output: 82986
// Input: 93
// Output: 84905
// Input: 94
// Output: 86846
// Input: 95
// Output: 88809
// Input: 96
// Output: 90794
// Input: 97
// Output: 92801
// Input: 98
// Output: 94830
// Input: 99
// Output: 96881
// Input: 100
// Output: 98954
// Input: 101
// Output: 101049
// Input: 102
// Output: 103166
// Input: 103
// Output: 105295
// Input: 104
// Output: 107436
// Input: 105
// Output: 109589
// Input: 106
// Output: 111764
// Input: 107
// Output: 113961
// Input: 108
// Output: 116180
// Input: 109
// Output: 118421
// Input: 110
// Output: 120684
// Input: 111
// Output: 122969
// Input: 112
// Output: 125276
// Input: 113
// Output: 127605
// Input: 114
// Output: 129956
// Input: 115
// Output: 132329
// Input: 116
// Output: 134724
// Input: 117
// Output: 137141
// Input: 118
// Output: 139580
// Input: 119
// Output: 142041
// Input: 120
// Output: 144524
// Input: 121
// Output: 147029
// Input: 122
// Output: 149556
// Input: 123
// Output: 152105
// Input: 124
// Output: 154676
// Input: 125
// Output: 157269
// Input: 126
// Output: 159884
// Input: 127
// Output: 162521
// Input: 128
// Output: 165180
// Input: 129
// Output: 167861
// Input: 130
// Output: 170564
// Input: 131
// Output: 173289
// Input: 132
// Output: 176036
// Input: 133
// Output: 178805
// Input: 134
// Output: 181596
// Input: 135
// Output: 184409
// Input: 136
// Output: 187244
// Input: 137
// Output: 190101
// Input: 138
// Output: 192980
// Input: 139
// Output: 195881
// Input: 140
// Output: 198804
// Input: 141
// Output: 201749
// Input: 142
// Output: 204716
// Input: 143
// Output: 207705
// Input: 144
// Output: 210716
// Input: 145
// Output: 213749
// Input: 146
// Output: 216804
// Input: 147
// Output: 219881
// Input: 148
// Output: 222980
// Input: 149
// Output: 226101
// Input: 150
// Output: 229244
// Input: 151
// Output: 232409
// Input: 152
// Output: 235596
// Input: 153
// Output: 238805
// Input: 154
// Output: 242036
// Input: 155
// Output: 245289
// Input: 156
// Output: 248564
// Input: 157
// Output: 251861
// Input: 158
// Output: 255180
// Input: 159
// Output: 258521
// Input: 160
// Output: 261884
// Input: 161
// Output: 265269
// Input: 162
// Output: 268676
// Input: 163
// Output: 272105
// Input: 164
// Output: 275556
// Input: 165
// Output: 279029
// Input: 166
// Output: 282524
// Input: 167
// Output: 286041
// Input: 168
// Output: 289580
// Input: 169
// Output: 293141
// Input: 170
// Output: 296724
// Input: 171
// Output: 300329
// Input: 172
// Output: 303956
// Input: 173
// Output: 307605
// Input: 174
// Output: 311276
// Input: 175
// Output: 314969
// Input: 176
// Output: 318684
// Input: 177
// Output: 322421
// Input: 178
// Output: 326180
// Input: 179
// Output: 329961
// Input: 180
// Output: 333764
// Input: 181
// Output: 337589
// Input: 182
// Output: 341436
// Input: 183
// Output: 345305
// Input: 184
// Output: 349196
// Input: 185
// Output: 353109
// Input: 186
// Output: 357044
// Input: 187
// Output: 361001
// Input: 188
// Output: 364980
// Input: 189
// Output: 368981
// Input: 190
// Output: 373004
// Input: 191
// Output: 377049
// Input: 192
// Output: 381116
// Input: 193
// Output: 385205
// Input: 194
// Output: 389316
// Input: 195
// Output: 393449
// Input: 196
// Output: 397604
// Input: 197
// Output: 401781
// Input: 198
// Output: 405980
// Input: 199
// Output: 410201
// Input: 200
// Output: 414444
// Input: 201
// Output: 418709
// Input: 202
// Output: 422996
// Input: 203
// Output: 427305
// Input: 204
// Output: 431636
// Input: 205
// Output: 435989
// Input: 206
// Output: 440364
// Input: 207
// Output: 444761
// Input: 208
// Output: 449180
// Input: 209
// Output: 453621
// Input: 210
// Output: 458084
// Input: 211
// Output: 462569
// Input: 212
// Output: 467076
// Input: 213
// Output: 471605
// Input: 214
// Output: 476156
// Input: 215
// Output: 480729
// Input: 216
// Output: 485324
// Input: 217
// Output: 489941
// Input: 218
// Output: 494580
// Input: 219
// Output: 499241
// Input: 220
// Output: 503924
// Input: 221
// Output: 508629
// Input: 222
// Output: 513356
// Input: 223
// Output: 518105
// Input: 224
// Output: 522876
// Input: 225
// Output: 527669
// Input: 226
// Output: 532484
// Input: 227
// Output: 537321

```

[illegible][illegible]

Таблица Виженера

Таблица Виженера

```
In [7]: # получение (формат) Введи ключ до тех пор, пока не получишь  
# столько же, сколько у сообщения
```

```
def genkey(msg, key):  
    # key.reverse(' ', '')  
    # msg.reverse(' ', '')  
    key = list(key)  
    if len(msg) > len(key):  
        return(key)  
    else:  
        for i in range(len(msg) -  
                        len(key)):  
            key.append(key[i % len(key)])  
    return('' . join(key))
```

```
In [8]: # шифрование  
def vig(msg, key):  
    cipher_text = []  
    # шифруем сообщение, потому что мы не знаем длину  
    # msg.reverse(' ', '')  
    for i in range(len(msg)):  
        x = (ord(msg[i]) + ord(key[i])) % 26  
        x = ord('A' +  
                cipher_text.append(chr(x))  
    return('' . join(cipher_text))
```

```
In [9]: def cipherText(string, key):  
    cipher_text = []  
    for i in range(len(string)):  
        x = (ord(string[i]) +  
              ord(key[i])) % 26  
        x = ord('A' +  
                cipher_text.append(chr(x))  
    return('' . join(cipher_text))
```

```
In [10]: # расшифровка  
def unvig(cipher_text, key):  
    orig_text = []  
    #msg.reverse(' ', '')  
    for i in range(len(cipher_text)):  
        x = (ord(cipher_text[i]) - ord(key[i]) + 26) % 26  
        x = ord('A' +  
                orig_text.append(chr(x))  
    return('' . join(orig_text))
```

```
while True:  
    msg = input(bold + "what message do you want to encrypt?\n" + end + "Note that only english characters and space are allowed")  
    if (False in [x in al for x in msg]):  
        continue  
    else:  
        msg = msg.upper()  
        break
```

```
while True:  
    key = input(bold + "\nEnter the key\n" + end + "Note that only english characters are allowed:\n")  
    if (False in [x in al for x in msg]):  
        continue  
    else:  
        key = key.upper()  
        break
```

```
keyg = genkey(msg, key)  
print("\nour encrypted message is: " + bold + ul + vig(msg, keyg))
```

```
what message do you want to encrypt?  
Note that only english characters and space are allowed:  
nomet  
what message do you want to encrypt?  
Note that only english characters and space are allowed:  
hello
```

```
Enter the key  
Note that only english characters are allowed:  
ura
```

```
Your encrypted message is: WVLF
```

```
while True:  
    msg = input("what message do you want to decrypt? ")  
    if (False in [x in al for x in msg]):  
        continue  
    else:  
        msg = msg.upper()  
        break
```

```
while True:  
    key = input("\nEnter the key: ")  
    if (False in [x in al for x in msg]):  
        continue  
    else:  
        key = key.upper()  
        break
```

```
keyg = genkey(msg, key)  
print("\nour decrypted message is: " + bold + ul + unvig(msg, keyg))
```

```
what message do you want to decrypt? WVLF
```

```
Enter the key: ura
```

```
Your decrypted message is: HELLO
```

Шифрование

```

while True:
    msg = input(bold + "what message do you want to encrypt?\n" + end + "Note that only russian and english characters and spa
    if (false in [x in a for x in msg]):
        continue
    else:
        break

while True:
    key = input(bold + "Enter the key\n" + end + "Note that repeated characters are prohibited:\n")
    if len(set(key)) != len(key):
        continue
    else:
        break

print("\nour encrypted message is: " + bold + ui + encryptMessage(msg, key))

```

what message do you want to encrypt?
Note that only russian and english characters and space are allowed:
inform

Enter the key
Note that repeated characters are prohibited:
odod

Enter the key
Note that repeated characters are prohibited:
fudfg

Enter the key
Note that repeated characters are prohibited:
yandex

Your encrypted message is: **naoarafoaain**

Таблица Виженера

```

In [7]: # проверка (убираем?) буквы ключа до тех пор, пока не совпадет
def getkey(msg, key):
    # столько же, сколько у сообщения
    key.replace(' ', '')
    msg.replace(' ', '')
    key = list(key)
    if len(msg) == len(key):
        return(key)
    else:
        for i in range(len(msg) - len(key)):
            key.append(key[i % len(key)])
    return("".join(key))

```

```

In [8]: # шифрование
def vig(msg, key):
    cipher_text = []
    # убираем пробелы, потому что мы их не выводим
    msg.replace(' ', '')
    key.replace(' ', '')
    for i in range(len(msg)):
        x = (ord(msg[i]) + ord(key[i]) % 26)
        x += ord('A')
        cipher_text.append(chr(x))
    return("".join(cipher_text))

```

```

In [9]: def cipherText(string, key):
    cipher_text = []
    for i in range(len(string)):
        x = (ord(string[i]) + ord(key[i]) % 26)
        x += ord('A')
        cipher_text.append(chr(x))
    return("".join(cipher_text))

```

```

In [10]: # расшифровка
def unvig(cipher_text, key):
    orig_text = []
    key.replace(' ', '')
    for i in range(len(cipher_text)):
        x = (ord(cipher_text[i]) - ord(key[i]) + 26) % 26
        x += ord('A')
        orig_text.append(chr(x))
    return("".join(orig_text))

```

Результаты

- Ознакомился с шифрами простой замены

- Ознакомился с шифрами простой замены
- Программно их реализовал