KATTA SONLARNI FAKTORIZATSIYA QILISH MUAMMOSINI SAMARALI HAL QILISHDA SHOR ALGORITIMDA FOYDALANISHNING BOSHQA ALGORITMLARDA SAMARADORLIGI

[1]**Toirov Sh.A.**
[2]**Saidakhmedov E.I.**
techmespeaker@gmail.com
[1]"Toshkent menejment va iqtisodiyot instetuti" muxandis va axborot texnologiyalari kafedrasining dotsent
[2]"Denov tadbirkorlik va pedogogika instituti" axborot texnologiyalari kafedrasi o'qituvchisi
https://doi.org/10.5281/zenodo.14870688

Kirish qismi.

Faktorizatsiya — bu biror butun sonni uning *Tub bo'luvchilariga* ajratish jarayonidir. Tub bo'luvchilar — bu faqat 1 va o'ziga bo'linadigan sonlardir. Boshqacha qilib aytganda, faktorizatsiya biror sonni faqat *tub sonlar* ko'paytmasi sifatida ifodalashni anglatadi.

Misol uchun, agar siz 12 sonini faktorizatsiya qilmoqchi bo'lsangiz, uni asosiy bo'luvchilarga ajratishingiz kerak:

$$12 = 2 \times 2 \times 3$$

Bu yerda 2 va 3 — asosiy sonlardir.

Shunday qilib, faktorizatsiya orqali 12 ni 2 va 3 kabi tub sonlarning ko'paytmasi sifatida ifodalanadi.

Faktorizatsiya jarayonida ma 'lum bir sonni topishga harakat qilamiz:

1. Sonni 2 ga bo'linadi.
2. Agar bo'linmasa, keyingi kichik asosiy son — 3, 5, 7, 11 va boshqalar bilan bo'linadi.
3. Bu jarayonni har safar, bo'lish mumkin bo'lgan tub sonni tanlab, davom ettiriladi.

Asosiy qism: Faktorizatsiya matematika va kriptografiyada muhim rol o'ynaydi, chunki bu jarayon sonlarning tuzilishini o'rganish va xavfsiz ma 'lumot uzatish tizimlarini yaratishda asosiy rol o'ynaydi.

Matematikada va faktorizatsiyaning axamiyat shundaki sonlarni tahlil qilish va ularning xususiyatlarini o'rganishga, biror sonni asosiy bo'luvchilarga ajratish orqali, uning matematik strukturasini tushunishga va algebraik masalalarni yechishda polinomlarni faktorizatsiya qilish orqali algebraik ifodalarni soddalashtirish mumkin.

Kriptografiya turli algoritmlardan foydalanib ma 'lumotlarni xavfsiz tarzda uzatish va saqlash mumkin va bu jarayondan faktorizatsiyadan foydalanadi.

RSA algoritmi (Rivest–Shamir–Adleman) — bu asosan katta sonlarning faktorizatsiyasiga asoslangan shifrlash algoritmi. RSA algoritmining xavfsizligi katta sonlarni asosiy bo'luvchilarga ajratishning qiyinligiga tayanadi.

RSA tizimi ikkita katta asosiy sonni tanlab, ularning ko'paytmasi sifatida jamoatcha kalitni hosil qiladi. Agar siz bu kalitni faktorizatsiya qilsangiz, shifrlangan ma 'lumotni ochish mumkin bo'ladi. Shuning uchun, faktorizatsiya qilish juda qiyin bo'lgani sababli, RSA tizimi xavfsiz bo'lib, ma 'lumotlarni himoya qiladi.

Bu algoritmi **katta sonlarni faktorizatsiya qiyinligiga** asoslangan va u quydagicha amalga oshiriladi.

1. **Kalit juftligini yaratish:** RSA algoritmida ikkita kalit mavjud:
2. **Jamoat kaliti (public key)**: Bu kalit ma 'lumotlarni shifrlashda ishlatiladi va keng tarqatiladi.
3. **Maxfiy kalit (private key)**: Bu kalit shifrlangan ma 'lumotlarni deshifrlashda ishlatiladi va faqat kalit egasi tomonidan saqlanadi.

Kalitlarni yaratish uchun quyidagi bosqichlar bajariladi:

Ikkita asosiy sonni tanlash: p va q kabi ikkita tasodifiy asosiy sonni tanlang. Bu sonlar bir-biriga mustaqil va katta bo'lishi kerak.

N ni hisoblash: N ni quyidagicha hisoblang:

$$\boldsymbol{N = p \times q}$$

N bu RSA algoritmining tub soni bo'lib, u **umumiy_kaliti** va **maxfiy_kalit**ni yaratishda ishlatiladi. N soni juda katta bo'lishi kerak, bu uning xavfsizligini ta 'minlaydi.

Euler funktsiyasini hisoblash: Eulerning totient funktsiyasini (phi) quyidagi tarzda hisoblang:

$$\phi(N) = (p-1) \times (q-1)$$

Bu yerda p va q tanlangan tub sonlar.

Umumiy_kaliti uchun e ni tanlash: e soni **1 < e < \phi(N)** oraliqda bo‘lib, e va ϕ(N) ning **eng katta umumiy bo‘luvchisi (GCD)** 1 bo‘lishi kerak. Bu e ni tanlash uchun shart bo‘ladi. e odatda kichik, masalan, 65537 soni tanlanadi, chunki u xavfsiz va tez hisoblanadi.

Maxfiy kalit uchun ddd ni hisoblash: d maxfiy kalitni hosil qilish uchun quyidagi tenglikdan foydalaniladi:

$$d \times e \equiv 1 \ (mod \ \phi(N))$$

Bu yerda d ni **modulyar teskari** (modular inverse) sifatida hisoblash kerak. d maxfiy kalit sifatida ishlatiladi.

Shifrlash jarayoni (Jamoat kaliti bilan): Ma ‘lumotlarni shifrlash uchun **Umumiy kaliti** (public key) ishlatiladi. Agar M — bu shifrlanadigan xabar bo‘lsa, uni quyidagicha shifrlash mumkin:

$$C = M^e \ (mod \ N)$$

Bu yerda:
C — shifrlangan xabar (kryptogramma),
e — jamoat kalitining birinchi qismini tashkil etuvchi son,
N — jamoat kalitining ikkinchi qismini tashkil etuvchi son,
M — asl ma ‘lumot.

Deshifrlash jarayoni (Maxfiy kalit bilan): Shifrlangan xabarni deshifrlash uchun **maxfiy kalit** (private key) ishlatiladi. Deshifrlash jarayoni quyidagicha amalga oshiriladi:

$$M = C^d (mod \ N)$$

Bu yerda:
C — shifrlangan xabar (kryptogramma),
d — maxfiy kalitning birinchi qismini tashkil etuvchi son
N — jamoat kalitining ikkinchi qismini tashkil etuvchi son,
M — asl ma ‘lumot.

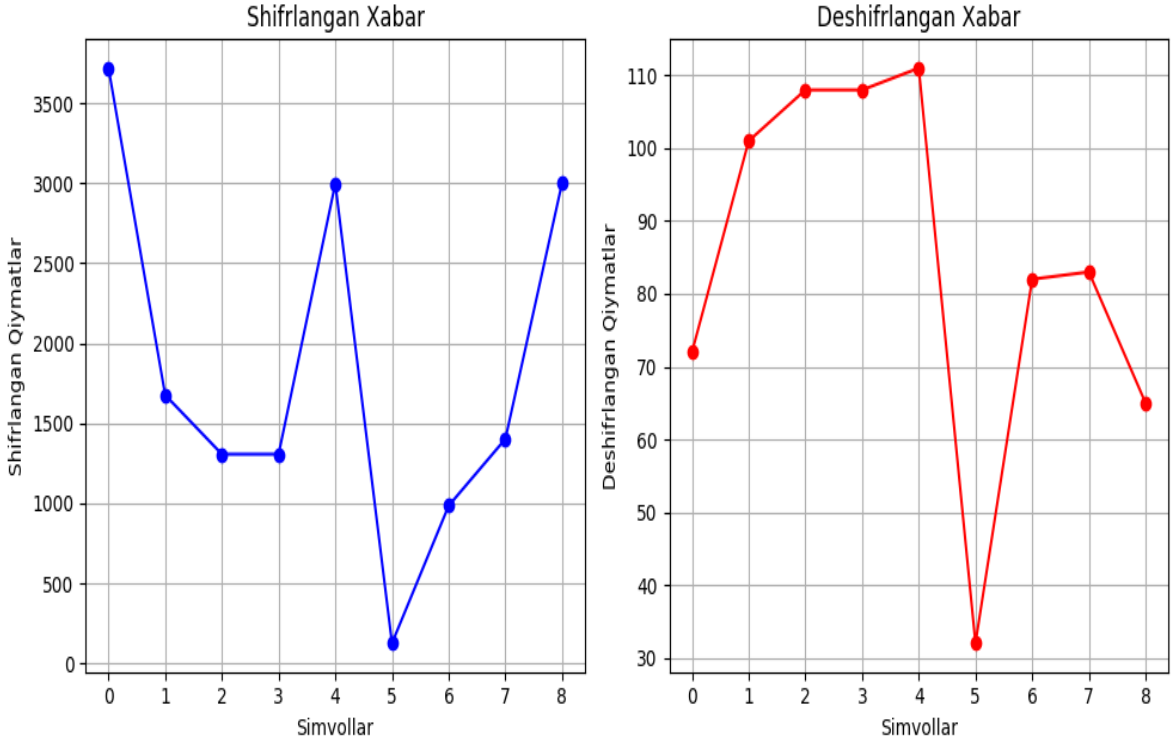
RSA algoritmining python Jupyter notebookda Pythoning PyPlot kutubxonasidan foydalangan holdagi kodi.

```
import random
import matplotlib.pyplot as plt
from sympy import mod_inverse

# 1. Ikkita tasodifiy asosiy sonni tanlash (p va q)
def generate_prime(bits=8):
    while True:
        num = random.getrandbits(bits)  # tasodifiy son yaratish
        if is_prime(num):
            return num

def is_prime(n):
    """Sonning asosiy son ekanligini tekshiradi."""
    if n <= 1:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True

# 2. Kalitlarni yaratish
def generate_rsa_keys(bits=8):
    # Ikkita asosiy sonni tanlash
```

```
    p = generate_prime(bits)
    q = generate_prime(bits)

    # N ni hisoblash
    N = p * q
    phi = (p - 1) * (q - 1)

    # Jamoat kaliti e ni tanlash
    e = 65537  # Bu umumiy tanlangan e qiymati, bu tez ishlaydi
    d = mod_inverse(e, phi)  # d ni hisoblash: d * e ≡ 1 (mod φ(N))

    # Jamoat kaliti (e, N) va maxfiy kalit (d, N)
    public_key = (e, N)
    private_key = (d, N)
return public_key, private_key

# 3. Xabarni shifrlash
def encrypt(public_key, message):
    e, N = public_key
    # Xabarni shifrlash: C = M^e % N
    cipher_text = [pow(ord(char), e, N) for char in message]
return cipher_text

# 4. Shifrlangan xabarni deshifrlash
def decrypt(private_key, cipher_text):
    d, N = private_key
    # Xabarni deshifrlash: M = C^d % N
    plain_text =  ‘ ‘.join([chr(pow(char, d, N)) for char in cipher_text])
return plain_text

# 5. RSA kalitlarini yaratish
public_key, private_key = generate_rsa_keys(bits=8)
print(f"Umumiy kalit: {public_key}")
print(f"Hususiy kalit: {private_key}")

# 6. Xabarni shifrlash va deshifrlash
message = "Hello RSA"
print(f"Shifrlanmagan habar: {message}")

cipher_text = encrypt(public_key, message)
print(f"Shifrlangan matn: {cipher_text}")

decrypted_message = decrypt(private_key, cipher_text)
print(f"Dishifrlangan matn: {decrypted_message}")

# 7. Shifrlangan xabarlarni vizual tarzda ko‘rsatish (matplotlib yordamida)
# Shifrlangan va deshifrlangan xabarlarni chizish
plt.figure(figsize=(10, 5))

# Shifrlangan xabar grafiği
plt.subplot(1, 2, 1)
plt.plot(cipher_text, marker= ‘o‘, color= ‘b ‘)
```

```
plt.title("Shifrlangan Xabar")
plt.xlabel("Simvollar")
plt.ylabel("Shifrlangan Qiymatlar")
plt.grid(True)

# Deshifrlangan xabar grafiği
plt.subplot(1, 2, 2)
plt.plot([ord(char) for char in decrypted_message], marker= ‘o‘, color= ‘r ‘)
plt.title("Deshifrlangan Xabar")
plt.xlabel("Simvollar")
plt.ylabel("Deshifrlangan Qiymatlar")
plt.grid(True)

# Grafiklarni ko‘rsatish
plt.tight_layout()
plt.show()
```

```
Umumiy kalit: (65537, 4033)
Hususiy kalit: (2177, 4033)
Shifrlanmagan habar: Hello RSA
Shifrlangan matn: [3719, 1676, 1307, 1307, 2997, 126, 985, 1402, 3001]
Dishifrlangan matn: Hello RSA
```



1.

Naiv Yondashuv (**Brute Force**)

- ***Prinsipi***: Sonni 2 dan boshlab, o‘sib boruvchi barcha butun sonlar bilan bo‘lib, uning asosiy bo‘luvchilarini topish.
- ***Afzalliklari***: Oson va intuitiv usul.
- ***Kamchiliklari***: Katta sonlar uchun juda sekin. Masalan, 1024bit sonlar uchun faktorizatsiya qilish deyarli imkonsiz bo‘ladi.
- ***Murakkablik***: $O(\sqrt{N})$

Brute Force algoritmi matematik masalalarni hal qilishda "xom kuch" yondashuvidan foydalanadi. Faktorizatsiya uchun bu yondashuv **berilgan sonni barcha mumkin bo‘lgan bo‘luvchilar orqali bo‘lib**,

asosiy omillarni topishga qaratilgan. U juda oddiy, ammo katta sonlar uchun samaradorligi past bo‘lgan algoritmdir.

1. **Boshlanishi**: Faktorizatsiya qilish kerak bo‘lgan sonni, N-ni aniqlang.
2. **Bo‘luvchilarni tekshirish**:

- N-ni ketma-ket sonlar bilan bo‘ling, odatda 2 dan boshlanadi.
- Har safar i-ga bo‘linganda qoldiq nol bo‘lsa (N%i==0), demak, i - bu N-ning omili.

3. **Sonni kichraytirish**:

- Agar bo‘linish muvaffaqiyatli bo‘lsa, N-ni i-ga bo‘ling (N=N//i).
- Agar bo‘linish muvaffaqiyatsiz bo‘lsa, i-ni oshiring (i+=1).

4. **Jarayonni takrorlash**:

- Yuqoridagi jarayon N=1 bo‘lgunga qadar davom etadi. Bu vaqtda barcha omillar topilgan bo‘ladi.

5. **Natija**:

- Topilgan barcha bo‘luvchilar ro‘yxati N-ning asosiy omillari hisoblanadi.

Misol: $N = 120$

Faktorizatsiya jarayoni:

1. $N = 120, i = 2: 120\%2 == 0$. Omil $i = 2, N = 120//2 = 60$.
2. $N = 60, i = 2: 60\%2 == 0$. Omil $i = 2, N = 60//2 = 30$.
3. $N = 30, i = 2: 30\%2 == 0$. Omil $i = 2, N = 30//2 = 15$.
4. $N = 15, i = 3: 15\%3 == 0$. Omil $i = 3, N = 15//3 = 5$.
5. $N = 5, i = 5: 5\%5 == 0$. Omil $i = 5, N = 5//5 = 1$.
6. $N = 1$: Jarayon tugadi.

Natija: $120 = 2 \times 2 \times 2 \times 3 \times 5$.

Algoritmning afzalliklari va kamchiliklari

Afzalliklari:

Oddiylik: Algoritmning tuzilishi va ishlashi tushunish uchun juda sodda.

Haqiqiy natijalar: Har qanday sonni faktorizatsiya qilishi mumkin.

Kamchiliklari:

Sekin ishlaydi: Katta sonlar uchun juda ko‘p vaqt talab qiladi, chunki barcha mumkin bo‘lgan bo‘luvchilar tekshiriladi.

Noefektivlik: N-ning kattaligi oshgani sari, bo‘luvchilarni tekshirish jarayoni juda ko‘p hisoblashlarni talab qiladi.

Murakkablik

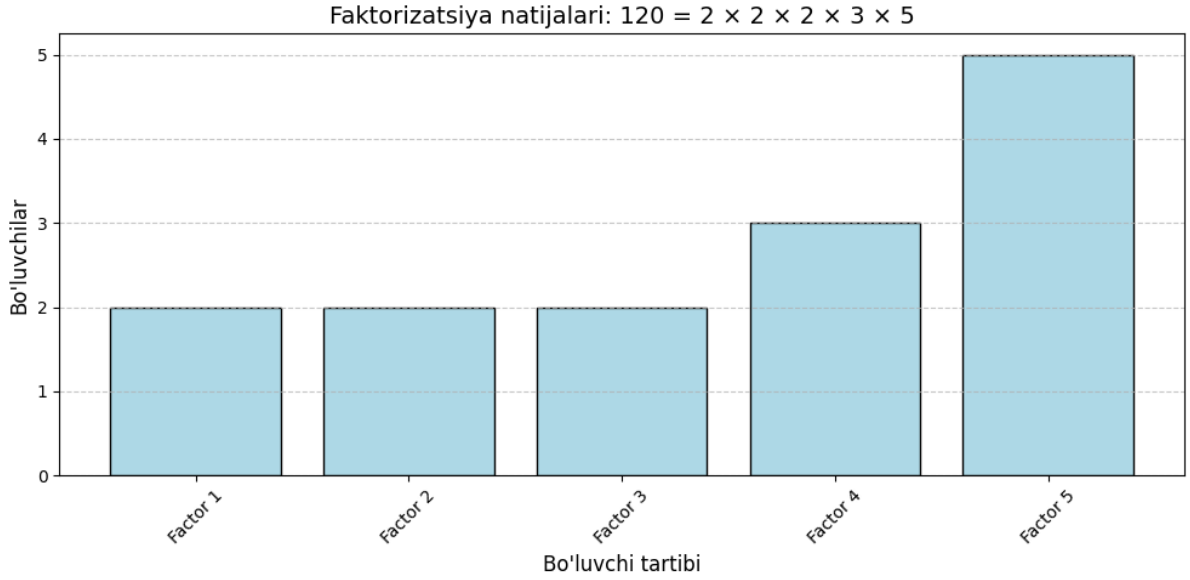
Eng yomon holatda murakkablik: $\boldsymbol{O(\sqrt{N})}$, chunki har qanday sonning eng katta asosiy omili uning kvadrat ildizidan katta bo‘la olmaydi.

O‘rtacha holat murakkabligi: Oddiy sonlar uchun jarayon uzoq davom etadi, chunki ular faqat o‘zlariga va 1 ga bo‘linadi.

```
import torch
# Brute Force algoritmi: PyTorch yordamida asosiy omillarni topish
def brute_force_factorization_torch(n):
    n = torch.tensor(n, dtype=torch.long)
    factors = []
    for i in range(2, n.item() + 1):
        while n % i == 0:
            factors.append(i)
            n //= i
        if n == 1:
            break
    return factors
```

```
# Faktorizatsiya qilinadigan son
number = 120 # Bu yerni o‘zgartirib, boshqa sonni faktorizatsiya qiling
factors = brute_force_factorization_torch(number)

print(f"Number: {number}")
print(f"Factors: {factors}")

# Vizualizatsiya uchun PyTorch tensorlardan foydalanish
import matplotlib.pyplot as plt

# Faktorlarni tensor shaklida yaratamiz
factors_tensor = torch.tensor(factors, dtype=torch.long)

# Grafikni yaratish
plt.figure(figsize=(10, 5))

# Faktorlarni chizish
plt.bar(
    torch.arange(len(factors_tensor)).tolist(),  # Faktorlarning indekslari
    factors_tensor.tolist(),  # Faktorlarning qiymatlari
    color="lightblue",
    edgecolor="black",
)
plt.xticks(range(len(factors)), [f"Factor {i+1}" for i in range(len(factors))], rotation=45)
plt.title(f"Faktorizatsiya natijalari: {number} = { ‘ × ‘.join(map(str, factors))}", fontsize=14)
plt.ylabel("Bo‘luvchilar", fontsize=12)
plt.xlabel("Bo‘luvchi tartibi", fontsize=12)
plt.grid(axis="y", linestyle="--", alpha=0.7)

# Grafikni ko‘rsatish
plt.tight_layout()
plt.show()
```

```
Number: 120
Factors: [2, 2, 2, 3, 5]
```



Trial Division algoritmi

Trial Division algoritmi oddiy va tushunarli faktorizatsiya usulidir. Uning ishlash prinsipi sonni ketma-ket kichik asosiy (prime) sonlarga bo‘lib chiqish orqali faktorizatsiya qilishga asoslangan.Quyida **Trial Division** algoritmini ishlashni tushuntirib beraman:

Algoritmning Asosi

1. Berilgan son N dan kichik bo‘lgan barcha asosiy sonlarni (prime numbers) topish.
2. Ushbu asosiy sonlar yordamida N ni ketma-ket bo‘lish orqali faktorizatsiya qilish.
3. Har bir bo‘luvchi topilganda, N ni topilgan bo‘luvchiga bo‘lib, jarayonni davom ettirish.

Algoritm Qadam-baqadam

1. **Boshlash:** Faktorizatsiya qilish kerak bo‘lgan son N ni kiriting.
2. **Bo‘luvchi tayyorlash:** Bo‘luvchi d-ni 2-dan boshlang.
3. **Bo‘luvni tekshirish:**
   - Agar $N \ mod \ d = 0$ bo‘lsa, d asosiy bo‘luvchi hisoblanadi.
   - N-ni d-ga bo‘lib (ya ‘ni N=N/d), jarayonni davom ettiring.
4. **Bo‘luvchini oshirish:**
   - Agar Nmod d≠0 bo‘lsa, d-ni keyingi sondan davom ettiring.
5. **To‘xtash:** Jarayon $d > \sqrt{N}$ bo‘lganda tugaydi yoki N-ning o‘zi asosiy son bo‘lib qoladi.

Misol: N=120

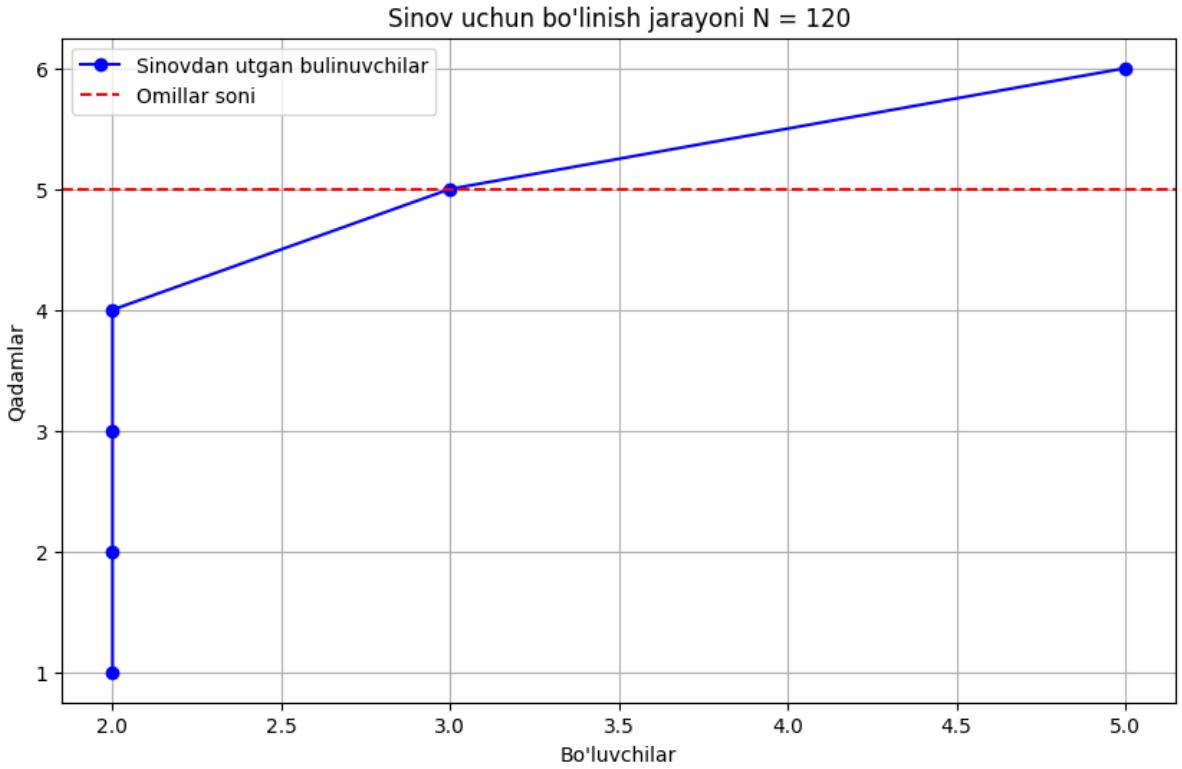
1. **Boshlash:** N=120.d =2
2. **Bo'luvni tekshirish:** 120 mod 2= 0 demak 2 asosiy bo'luvchi.
   - 120/2=60. Qolgan N=60
3. **Jarayonni davom ettirish:**
   - 60mod 2=0, 60/2=30, N=30
   - 30mod 2=0, 30/2=15 , N=15
4. **Bo'luvchini oshirish:** Endi d=3
   - 15mod 3=0 15/3=5, N=5
5. **Yakun:** N=5, bu asosiy son, shuning uchun natija: 2,2,2,3,5

Ushbu misolni pythonning pyplot kutubxonasidan foydalangan holdagi dasturiy kodi.

```
import matplotlib.pyplot as plt

def trial_division(n):
    factors = []  # Asosiy bo'luvchilar ro'yxati
    divisors = [] # Har bir qadamda sinab ko'rilgan bo'luvchilar
    d = 2

    while d * d <= n:
        divisors.append(d)  # Qadamda sinab ko'rilgan bo'luvchini yozib olish
        if n % d == 0:
            factors.append(d)
            n //= d
        else:
            d += 1

    if n > 1:
        factors.append(n)
        divisors.append(n)  # Qolgan asosiy sonni qo'shish

    return factors, divisors

# Sonni faktorizatsiya qilish
N = 120  # Faktorizatsiya qilinadigan son
factors, divisors = trial_division(N)
```

```
# Natijalarni grafikda tasvirlash
plt.figure(figsize=(10, 6))

# Har bir qadam uchun grafikni chizish
plt.plot(divisors, range(1, len(divisors) + 1), 'o-', color='blue', label='Sinovdan utgan bulinuvchilar')
plt.axhline(len(factors), color='red', linestyle='--', label='Omillar soni')

# Grafikni sozlash
plt.title(f"Sinov uchun bo'linish jarayoni N = {N}")
plt.xlabel("Bo'luvchilar")
plt.ylabel("Qadamlar")
plt.legend()
plt.grid(True)

# Natijalarni chiqarish
print(f"Asosiy bo'luvchilar: {factors}")
print(f"Tekshirilgan bo'luvchilar: {divisors}")

# Grafikni ko'rsatish
plt.show()
```

```
Asosiy bo'luvchilar: [2, 2, 2, 3, 5]
Tekshirilgan bo'luvchilar: [2, 2, 2, 2, 3, 5]
```



Fermat's Factorization Method: Algoritm qanday ishlaydi?

Fermatning faktorizatsiya usuli sonni ikki kvadratlar farqi sifatida tasvirlashga asoslangan:

$$N = a^2 - b^2$$

Bu ifoda matematik tarzda qayta yozilishi mumkin:

$$N = (a - b)(a + b)$$

Bu yerda N - faktorizatsiya qilinadigan son, a va b- izlanayotgan butun sonlar. Ushbu usul katta bo'luvchilari bir-biriga yaqin bo'lgan sonlarni faktorizatsiya qilishda samarali ishlaydi.

Algoritmning ishlash qadam-baqadam tavsifi

1. **Boshlash**: Berilgan N son uchun a ni boshlang'ich qiymat sifatida $\lceil\sqrt{N}\rceil$ qilib oling.
   - Bu N-dan katta yoki teng eng kichik butun kvadrat ildizdir.
2. **Kvadrat farqni tekshirish**:
   - Hisoblang: $b^2 = a^2 - N$.
   - Agar b^2 butun sonning kvadrati bo'lsa, faktorizatsiya topilgan.
3. **To'g'rilash**:
   - Agar b^2kvadrat bo'lmasa, a-ni 1 ga oshiring (a=a+1) va qayta hisoblang.
4. **Faktorizatsiyani topish**:
   - Faktorizatsiya topilgach: $p = a - b, q = a + b$ Bu yerda p va q N-ning bo'luvchilari.
5. **To'xtash**: Algoritm b^2 kvadratga aylanguncha davom etadi.

Misol: $N = 5959$

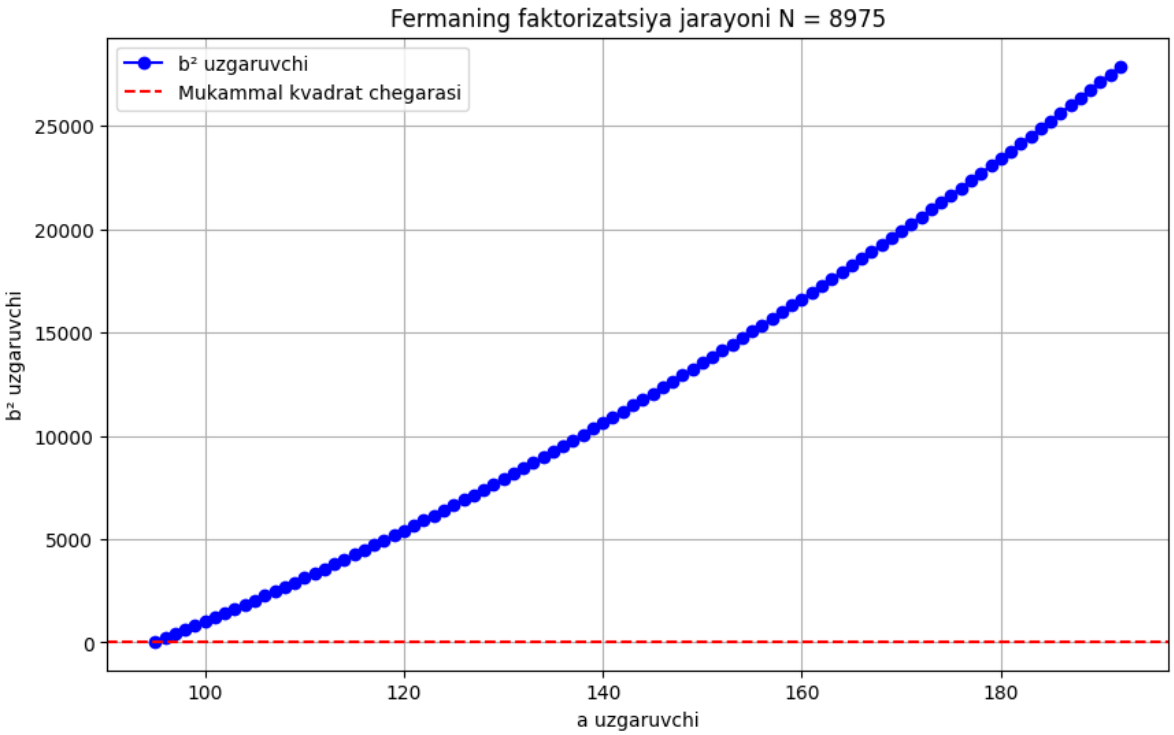
1. **Boshlash**:
   $N = 5959, a = \lceil\sqrt{5959}\rceil = 78.$
2. **Kvadrat farqni hisoblash**:
   $b^2 = 78^2 - 5959 = 6084 - 5959 = 125.$
3. **Kvadratni tekshirish**:
   $b^2 = 125$ kvadrat emas. Shuning uchun a-ni oshiramiz: $a = 79.$
4. **Yangi hisoblash**:
   $b^2 = 79^2 - 5959 = 6241 - 5959 = 282.$
   Yana kvadrat emas. $a = 80$ deb oshiramiz.
5. **Keyingi hisoblash**:
   $b^2 = 80^2 - 5959 = 6400 - 5959 = 441.$
   $b = 21$, chunki $441 = 21^2$.
6. **Faktorizatsiya topish**:
   $p = a - b = 80 - 21 = 59, q = a + b = 80 + 21 = 101.$
7. **Natija**: $5959 = 59 \times 101.$

Pythondagi dasturiy kodi

```
mport matplotlib.pyplot as plt
import math

# Fermat's Factorization Function
def fermat_factorization(N):
    steps = []  # Har bir qadamda 'a' va 'b^2' qiymatlarini saqlash
    a = math.ceil(math.sqrt(N))  # a ni boshlang'ich qiymati
    while True:
        b2 = a**2 - N  # b^2 ni hisoblash
        steps.append((a, b2))  # Qadamni saqlash
        if b2 >= 0 and math.isqrt(b2)**2 == b2:  # b^2 mukammal kvadrat bo'lsa
            b = int(math.sqrt(b2))
            return a - b, a + b, steps  # Bo'luvchilar va qadamlar qaytariladi
        a += 1  # a ni oshirish

# Faktorizatsiya qilinadigan son
```

```
N = 8975

# Algoritmni ishga tushirish
p, q, steps = fermat_factorization(N)

# Qadamlarni grafikda tasvirlash
a_values = [step[0] for step in steps]
b2_values = [step[1] for step in steps]

plt.figure(figsize=(10, 6))
plt.plot(a_values, b2_values, 'o-', color='blue', label='b² uzgaruvchi')
plt.axhline(0, color='red', linestyle='--', label='Mukammal kvadrat chegarasi')

# Grafik sozlamalari
plt.title(f"Fermaning faktorizatsiya jarayoni N = {N}")
plt.xlabel("a uzgaruvchi")
plt.ylabel("b² uzgaruvchi")
plt.legend()
plt.grid(True)

# Natijalarni chiqarish
print(f"Faktorizatsiya natijasi: {N} = {p} * {q}")
print("Qadamlar:")
for i, (a, b2) in enumerate(steps):
    print(f"Step {i + 1}: a = {a}, b² = {b2}")

# Grafikni ko'rsatish
plt.show()
```

```
Faktorizatsiya natijasi: 8975 = 25 * 359
Qadamlar:
Step 1: a = 95, b² = 50
Step 2: a = 96, b² = 241
Step 3: a = 97, b² = 434
Step 4: a = 98, b² = 629
Step 5: a = 99, b² = 826
Step 6: a = 100, b² = 1025
Step 7: a = 101, b² = 1226
Step 8: a = 102, b² = 1429
Step 9: a = 103, b² = 1634
Step 10: a = 104, b² = 1841
Step 11: a = 105, b² = 2050
Step 12: a = 106, b² = 2261
Step 13: a = 107, b² = 2474
Step 14: a = 108, b² = 2689
Step 15: a = 109, b² = 2906
Step 16: a = 110, b² = 3125
Step 17: a = 111, b² = 3346
Step 18: a = 112, b² = 3569
Step 19: a = 113, b² = 3794
Step 20: a = 114, b² = 4021
Step 21: a = 115, b² = 4250
Step 22: a = 116, b² = 4481
Step 23: a = 117, b² = 4714
...
Step 95: a = 189, b² = 26746
Step 96: a = 190, b² = 27125
Step 97: a = 191, b² = 27506
Step 98: a = 192, b² = 27889
```



Pollard's Rho Algorithm**: Algoritm qanday ishlaydi?**

Pollard's Rho Algorithm - bu faktorizatsiya uchun probabilistik algoritm bo'lib, asosan o'rta kattalikdagi sonlarni tezkor faktorizatsiya qilish uchun ishlatiladi. U matematik funksiya yordamida sikl (cycle) topish va GCD (Greatest Common Divisor) orqali bo'luvchi aniqlashga asoslangan.Algoritmning nomi "Rho" (yunoncha ρ) ga o'xshash siklni tasvirlovchi shakl bilan bog'liq.

Ishlash prinsipi

1. **Funksiya tanlash**: Oddiy kvadrat funksiya ishlatiladi, masalan:

$$f(x) = (x^2 + c) mod N$$

- Bu yerda:
- N- faktorizatsiya qilinadigan son,
- c- biror o'zgarmas butun son.
- **Ikkita iteratsion nuqta yaratish**:
- x va y boshlang'ich qiymati sifatida tanlanadi (odatda x=2 va y=2).
- x bir marta iteratsiya qilinadi: $x \leftarrow f(x)$
- y ikki marta iteratsiya qilinadi: $y \leftarrow f(f(y))$
- **GCD hisoblash**: Har bir iteratsiyada:

$$d = GCD(|\ x - y\ |, N)$$

- Agar $d > 1$ bo'lsa, d N-ning bo'luvchisi hisoblanadi.
- **To'xtash**:
- Agar d=N, sikldan chiqib boshqa funksiya yoki boshlang'ich qiymatlarni tanlash kerak.
- Agar d asosiy bo'luvchi bo'lsa, u qaytariladi.

Misol: $N = 8051$

1. **Boshlash:**
   - Funksiya: $f(x) = (x^2 + 1) \mod 8051$,
   - $x = 2, y = 2$.
2. **Iteratsiyalar:**
   - 1-qadam: $x = f(2) = 5, y = f(f(2)) = 26$,

     $$d = \text{GCD}(|5 - 26|, 8051) = \text{GCD}(21, 8051) = 1$$

     Demak, $d = 1$, davom etamiz.
   - 2-qadam: $x = f(5) = 26, y = f(f(26)) = 677$,

     $$d = \text{GCD}(|26 - 677|, 8051) = \text{GCD}(651, 8051) = 97$$
3. **Natija:**
   - $d = 97$, bu N-ning bir bo'luvchisi.
   - $8051 = 97 \times 83$.

```
import matplotlib.pyplot as plt
from math import gcd

# Pollard's Rho Algorithm
def pollards_rho(N, c=1):
    def f(x):
        return (x**2 + c) % N  # Iteratsiya funksiyasi

    x, y, d = 2, 2, 1  # Boshlang'ich qiymatlar
    steps = []  # Qadamlarni saqlash

    while d == 1:
        x = f(x)  # x ni bitta iteratsiya
        y = f(f(y))  # y ni ikkita iteratsiya
        d = gcd(abs(x - y), N)  # GCD hisoblash
        steps.append((x, y, d))  # Har bir qadamni saqlash

    if d == N:
        return None, steps  # Bo'luvchi topilmadi
    return d, steps  # Bo'luvchi va qadamlar

# Faktorizatsiya qilinadigan son
N = 8051

# Algoritmni ishga tushirish
result, steps = pollards_rho(N)

# Natijalarni chiqarish
if result:
    print(f"Faktorizatsiya natijasi: {N} = {result} * {N // result}")
else:
    print("Bo'luvchi topilmadi, boshqa parametrlarni sinab ko'ring.")
```

```
print("Qadamlar:")
for i, (x, y, d) in enumerate(steps):
    print(f"Step {i + 1}: x = {x}, y = {y}, GCD = {d}")

# Grafik vizualizatsiya
x_values = [step[0] for step in steps]
y_values = [step[1] for step in steps]

plt.figure(figsize=(10, 6))
plt.plot(range(1, len(x_values) + 1), x_values, 'o-', label='x values', color='blue')
plt.plot(range(1, len(y_values) + 1), y_values, 'x-', label='y values', color='orange')

# Grafik sozlamalari
plt.title(f"Pollardning Rho jarayoni N = {N}")
plt.xlabel("qadamlar")
plt.ylabel("uzgaruvchilar")
plt.legend()
plt.grid(True)

# Grafikni ko'rsatish
plt.show()
```

```
Faktorizatsiya natijasi: 8051 = 97 * 83
Qadamlar:
Step 1: x = 5, y = 26, GCD = 1
Step 2: x = 26, y = 7474, GCD = 1
Step 3: x = 677, y = 871, GCD = 97
```