Public Key Cryptography - Bonus Assignment

Continued Fractions Method (Integer Factorization)

Student: Cristian Mititean
Assigned number: N = 7871

Goal

Factor the integer N using the continued-fraction expansion of sqrt(N) and its convergents.

How the continued-fraction terms a_i are generated

We write sqrt(N) as a continued fraction sqrt(N) = [a0; a1, a2, ...]. The coefficients a_i are produced using the standard recurrence with (m_k, d_k, a_k):

```
a0 = floor(sqrt(N))
m0 = 0,  d0 = 1
m_{k+1} = d_k * a_k - m_k
d_{k+1} = (N - m_{k+1}^2) / d_k
a_{k+1} = floor((a0 + m_{k+1}) / d_{k+1})
```

For N = 7871, a0 = floor(sqrt(7871)) = 88. Below are the first two iterations (to show concretely how a1 and a2 are obtained):

```
k = 0:
  m1 = d0*a0 - m0 = 1*88 - 0 = 88
  d1 = (N - m1^2)/d0 = (7871 - 88^2)/1 = 127
  a1 = floor((a0 + m1)/d1) = floor(176/127) = 1

k = 1:
  m2 = d1*a1 - m1 = 127*1 - 88 = 39
  d2 = (N - m2^2)/d1 = (7871 - 39^2)/127 = 50
  a2 = floor((a0 + m2)/d2) = floor(127/50) = 2
```

This matches the start of the sequence a0=88, a1=1, a2=2 used to compute the convergents.

Convergents p_i / q_i

Let p_i/q_i be the i-th convergent corresponding to the continued-fraction terms. We compute them using the standard recurrence:

```
p_{-2} = 0,  p_{-1} = 1
q_{-2} = 1,  q_{-1} = 0
p_i = a_i * p_{i-1} + p_{i-2}
q_i = a_i * q_{i-1} + q_{i-2}
```

Because p_i/q_i is a very accurate approximation of sqrt(N), the integers p_i^2 and N*q_i^2 are close to each other. We define:

```
r_i = p_i^2 - N * q_i^2
```

When r_i is small, we get a useful relation between squares. In particular, if r_i = 1 then:

```
p_i^2 - N*q_i^2 = 1  =>  p_i^2 ≡ 1 (mod N)
```

which is a congruence of squares modulo N that can be used to recover non-trivial factors.

Computed table of convergents

i	a_i	p_i	q_i	r_i = p_i^2 - N*q_i^2
0	88	88	1	-127
1	1	89	1	50
2	2	266	3	-83
3	1	355	4	89
4	1	621	7	-38
5	4	2839	32	17
6	10	29011	327	-38
7	4	118883	1340	89
8	1	147894	1667	-83
9	1	266777	3007	50
10	2	681448	7681	-127
11	**1**	**948225**	**10688**	**1**

Extracting the factors

From the table, the first index where r_i = 1 is i = 11. Therefore:

```
p_11^2 - N*q_11^2 = 1
=> p_11^2 ≡ 1 (mod N)
```

Rewrite the congruence as p_11^2 - 1 ≡ 0 (mod N), and factor the left-hand side:

```
p_11^2 - 1 = (p_11 - 1)(p_11 + 1)
=> (p_11 - 1)(p_11 + 1) ≡ 0 (mod N)
```

So N divides the product (p_11 - 1)(p_11 + 1). For a composite N, typically one factor shares a non-trivial gcd with N. We compute gcd(p_11 - 1, N) and gcd(p_11 + 1, N). To keep the arithmetic small, reduce p_11 modulo N first.

```
p_11 = 948225
x = p_11 mod N = 948225 mod 7871 = 3705
x^2 mod N = 1
```

Since $x \equiv p_{11} \pmod N$, $\gcd(p_{11} \pm 1, N) = \gcd(x \pm 1, N)$. Then:

```
gcd(x - 1, N) = gcd(3704, 7871) = 463
gcd(x + 1, N) = gcd(3706, 7871) = 17
```

Both values are strictly between 1 and N, so they are non-trivial factors.

Result

$$7871 = 17 * 463$$

Therefore the factorization of N is $N = 17 \cdot 463$.

Appendix: Python implementation

Full script used to generate the continued-fraction terms, convergents table, and the gcd step:

```
import math
from dataclasses import dataclass


@dataclass
class Row:
    i: int
    a: int
    p: int
    q: int
    r: int
    is_square: bool


def is_square(x: int) -> bool:
    if x < 0:
        x = -x
    s = math.isqrt(x)
    return s * s == x


def cf_sqrt_terms(N: int, max_iters: int):
    a0 = math.isqrt(N)
    m, d, a = 0, 1, a0
    for _ in range(max_iters):
        yield a
        m = d * a - m
        d = (N - m * m) // d
        if d == 0:
            return
        a = (a0 + m) // d


def convergents_and_r(N: int, max_iters: int):
    p_m2, p_m1 = 0, 1
    q_m2, q_m1 = 1, 0

    for i, a in enumerate(cf_sqrt_terms(N, max_iters)):
        p = a * p_m1 + p_m2
        q = a * q_m1 + q_m2
        r = p * p - N * q * q

        yield Row(i=i, a=a, p=p, q=q, r=r, is_square=is_square(r))

        p_m2, p_m1 = p_m1, p
        q_m2, q_m1 = q_m1, q


def factor_by_pell_step(N: int, max_iters: int = 5000, print_first_rows: int = 12):
    print(f"N = {N}\n")
    print(f"{'i':>3} {'a_i':>4} {'p_i':>12} {'q_i':>12} {'r_i=p^2-Nq^2':>16}")
    print("-" * 55)

    found = None
    for row in convergents_and_r(N, max_iters):
        if row.i < print_first_rows:
            print(f"{row.i:3d} {row.a:4d} {row.p:12d} {row.q:12d} {row.r:16d}")

        if row.r == 1:
            x = row.p % N
            g1 = math.gcd(x - 1, N)
            g2 = math.gcd(x + 1, N)
```

```python
            if 1 < g1 < N or 1 < g2 < N:
                found = (row, x, g1, g2)
                break

    if not found:
        print("\nNo factor found within max_iters. Increase max_iters.")
        return None, None

    row, x, g1, g2 = found
    print("\n--- Found step ---")
    print(f"Index i = {row.i}")
    print(f"r_i = p_i^2 - N*q_i^2 = {row.r}  (so p_i^2 ≡ 1 (mod N))")
    print(f"x = p_i mod N = {x}")
    print(f"Check: x^2 mod N = {(x*x) % N}")
    print(f"gcd(x-1, N) = gcd({x-1}, {N}) = {g1}")
    print(f"gcd(x+1, N) = gcd({x+1}, {N}) = {g2}")

    factors = []
    for g in (g1, g2):
        if 1 < g < N:
            factors.append(g)

    if not factors:
        return None, None

    f = factors[0]
    return f, N // f


if __name__ == "__main__":
    N = 7871
    f1, f2 = factor_by_pell_step(N, max_iters=10000, print_first_rows=12)

    print("\nResult:")
    if f1 is not None:
        print(f"{N} = {f1} * {f2}")
    else:
        print("Failed.")
```