Sundaramovo síto

A srovnání se sítem Eratosthenovým

Autor: **Marco Souza de Joode**, G. Nad Štolou

1. května 2021

Eratosthenovo síto

- Předpokládáme, že všechna čísla jsou prvočísla
- Postupně odstraňujeme všechny celočíselné násobky
- Případně pouze celočíslené násobky čísel, které nebyly odstraněny.

Myšlenka za Sundaramovým sítem

Všechna přirozená čísla jsou buď sudá nebo lichá. Všechna prvočísla, až na číslo 2, jsou lichá. **Sundaramovo síto** selektivně vybírá složená lichá čísla a odstraňuje je.

Součiny čísel podle parity

Označíme-li si sudá čísla jako S a lichá čísla jako L, pak platí

$$
\begin{aligned}
S \cdot S &= S \\
S \cdot L &= S \\
L \cdot S &= S \\
L \cdot L &= L
\end{aligned}
$$

protože

$$
\begin{aligned}
2n \cdot 2m &= 4nm = 2 \cdot 2nm \\
2n \cdot (2m+1) &= 4nm + 2n = 2 \cdot (2nm + n) \\
(2n+1) \cdot (2m+1) &= 4nm + 2n + 2m + 1 = 2 \cdot (n + m + 2nm) + 1
\end{aligned}
$$

Složená lichá čísla

Všechna lichá čísla jsou buď prvočísla, nebo čísla složená. Každé liché složené číslo lze zapsat jako

$$\begin{aligned}&(2i+1)(2j+1)\\&=4ij+2i+2j+1\\&=2\underbrace{(i+j+2ij)}_{U}+1\end{aligned}$$

Základní princip Sundaramova síta

- Mějme $m = \lfloor N/2 \rfloor$
- Pro všechna $i \in \{1, 2, ..., m\}$ a pro všechna $j \in \{1, 2, ..., m\}$ nalezněme množinu čísel L, které **nelze** zapsat jako $U = i + j + 2ij$, když $U < m$.
- Všechna lichá prvočísla lze zapsat jako $2U + 1$ pro každé U v množině L.

Optimalizace Sundaramova síta

- Mějme $m = \lfloor N/2 \rfloor$
- Pro všechna $i \in \{1, 2, ..., m\}$, aby $U = i + j + 2ij < m$, musí platit omezení na j:

$$\begin{aligned} i + j + 2ij &< m \\ j(2i+1) &< m - i \\ j &< \frac{m-i}{2i+1} \end{aligned}$$

  a zároveň nemusíme prověřovat $j \leq i$. Pak $j \in \{i, i+1, ..., \frac{m-i}{2i+1}\}$
- Všechna lichá prvočísla lze zapsat jako $2U + 1$ pro každé U v množině L.

Eratosthenovo síto

```python
def eratosthenes(n):
    multiples = []
    primes = []
    for i in range(2, n+1):
        if i not in multiples:
            primes.append(i)
            for j in range(i*i, n+1, i):
                multiples.append(j)
    return np.array(primes)
```

Budované pomocí dvou seznamů

- Seznamu násobků
- Seznamu složených čísel

Naivní Eratosthenovo síto pomocí Booleovských masek

```python
def eratosthenes_boolean(n):
    L = np.ones((n,), dtype="bool")
    for i in range(2,n):
        k = 2
        while k*i < n:
            L[k*i] = False
            k += 1
    L[0], L[1] = False, False
    return np.array([i for i, l in enumerate(L) if l])

```

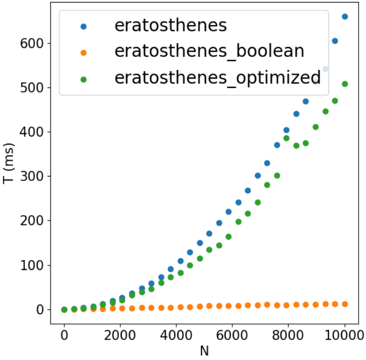
Používá pouze jedno `numpy` pole zaplněné booleovskými hodnotami `True`, poté odstraňuje (naivně) všechny násobky všech čísel.

Optimalizované Eratosthenovo síto

```python
def eratosthenes_optimized(n):
    primes = [i for i in range(2,n+1)]
    for i in primes:
        k = 2
        while k*i <= n:
            try:
                primes.remove(k*i)
            except ValueError:
                pass
            k+= 1
    return primes
```

Na začátku mají všechna čísla status prvočísla. Následně jsou odstraňovány všechny násobky zbývajících čísel s tímto statusem.

Vzájemné srovnání



Vyhrává síto využívající booleovské masky.

Časová složitost Eratosthenova síta

Něco jako nejlepší nebo nejhorší případ zde nehraje roli - vždy procházíme tentýž seznam čísel.Za předpokladu, že odstranění složeného čísla provedeme v čase $O(1)$, tak odstranění všech násobků vyžaduje minimálně

$$O(\frac{N}{2} + \frac{N}{3} + \frac{N}{5} + \frac{N}{7} + ... + \frac{N}{p}) \tag{1}$$

kde p je nejvyšší prvočíslo takové, že $p < N$.

Časová složitost Eratosthenova síta

Euler (1737) dokazuje, že

$$\sum_{\text{p je prv.}<N} \frac{1}{p} \geq \log\log(N+1) - \log\frac{\pi^2}{6} \tag{2}$$

Což pro velké hodnoty N

$$\log\log(N+1) - \log\frac{\pi^2}{6} \approx \log\log(N)\,, \tag{3}$$

a tak

$$O(\frac{N}{2} + \frac{N}{3} + \frac{N}{5} + \frac{N}{7} + \ldots + \frac{N}{p}) \approx O(N\log\log N)\,. \tag{4}$$

Naivní Sundaramovo síto

```python
def sundaram_naive_boolean(n):
    m = n//2
    L = [True] * n

    for i in range(1, m):
        L[2*i] = False
        for j in range(1, m):
            U = i + j + 2*i*j
            if U < m:
                L[2*U+1] = False

    L[0], L[1] = False, False
    L[2] = True
    return np.array([i for i, l in enumerate(L) if l])
```

Pomocí seznamu Booleovských hodnot. Odstraňuje čísla na indexech $U = i + j + 2ij$ menší než $m = \lfloor N/2 \rfloor$.

Časová složitost naivního Sundaramova síta

Prochází všechny body v kartézském součinu

$$i \in \{2, ..., \lfloor N/2 \rfloor\} \times j \in \{2, ..., \lfloor N/2 \rfloor\}$$

a operuje tedy minimálně s časem

$$O(\lfloor N/2 \rfloor^2) \approx O(\frac{1}{4}N^2) \approx O(N^2) \tag{5}$$

což oproti nejlepšímu Eratosthenovu sítu představuje výrazné zhoršení.

Naivní Sundaramovo síto, implementace pomocí množin

```
def sundaram_naive_set(n):
    m = n // 2
    L = {l for l in range(1,m)}
    for i in range(1,m):
        for j in range(i,m):
            U = i + j + 2*i*j
            if U < m:
                L.discard(U)
    primes = deque([2*l+1 for l in L])
    primes.extendleft([2])
    return np.array(primes)
```

Povšimneme si, že není třeba procházet celý prostor

$$i \in \{2, \dots, \lfloor N/2 \rfloor\} \times j \in \{2, \dots, \lfloor N/2 \rfloor\}$$

ale stačí pouze

$$i \in \{2, \dots, \lfloor N/2 \rfloor\} \times j \in \{i, i+1, \dots, \lfloor N/2 \rfloor\}$$

Optimalizace Sundaramova síta

```python
def sundaram_optimized_set(n):
    m = n // 2
    L = {l for l in range(1,m)}
    for i in range(1,m):
        limit = (m-i)//(2*i+1)
        for j in range(i, limit+1):
            U = i + j + 2*i*j
            L.discard(U)
    primes = deque([2*l+1 for l in L])
    primes.extendleft([2])
    return np.array(primes)
```

Optimalizace Sundaramova síta

Protože

$$\begin{aligned} i + j + 2ij &< m \\ j(2i+1) &< m - i \\ j &< \frac{m-i}{2i+1} \end{aligned}$$

a zároveň nemusíme prověřovat $j \leq i$. Pak $j \in \{i, i+1, ..., \frac{m-i}{2i+1}\}$

Optimalizace Sundaramova síta pomocí Booleovské masky

```
def sundaram_optimized_boolean(n):
    m = n//2
    L = [True] * n

    for i in range(1, m):
        L[2*i] = False
        limit = (m-i)//(2*i+1)
        for j in range(i, limit+1):
            U = i + j + 2*i*j
            if U < m:
                L[2*U+1] = False

    L[0], L[1], L[2] = False, False, True
    return np.array([i for i, l in enumerate(L) if l])
```

Idealizovaná časová složitost

Časová složitost je závislá na počtu kroků, který si můžeme spočíst uměle:

```
steps = []
M = np.linspace(0,10**5, 25, dtype="int64")

for m in M:
    s = 0
    for i in range(1,m):
        limit = (m-i)//(2*i+1) + 1
        for j in range(i, limit):
            s += 1
    steps.append(s)
```

Idealizovaná časová složitost

Bylo dokázáno (Sundaram, 1934), že časová složitost Sundaramova síta v této podobě se chová jako

$$O(\frac{m}{5}\log m) = O(\frac{\lfloor N/2 \rfloor}{5}\log\lfloor N/2 \rfloor) < N\log N \qquad (6)$$