```
#!/usr/bin/env python3
"""Goldbach par Dancing Links, version minimale.

Matrice : lignes = impairs s dans [3, n/2] ; colonnes = premiers p <= sqrt(n).
La ligne s porte un 1 en colonne p ssi p ne divise ni s (sauf s = p), ni n - s.
Couverture exacte = ensemble de lignes couvrant chaque colonne une fois.
Comme la colonne 2 est couverte par toute ligne, les couvertures exactes
sont des singletons {s} : exactement les décomposants (s, n-s premiers).
Usage : python goldbach_dlx.py 98            (tableaux + décomposants)
        python goldbach_dlx.py 98 --test 300 (vérifie tous les n pairs jusqu'à
300)
"""
import sys
from math import isqrt

class N:  # nœud ; les colonnes sont aussi des nœuds
    def __init__(s, c=None, r=None):
        s.L = s.R = s.U = s.D = s
        s.C, s.row, s.size = c, r, 0

def build(cols, rows):
    root = N()
    head = {}
    for c in cols:
        h = N(); h.C = h
        h.L, h.R = root.L, root
        root.L.R = h; root.L = h
        head[c] = h
    for r, cs in rows.items():
        first = None
        for c in cs:
            h = head[c]
            x = N(h, r)
            x.U, x.D = h.U, h
            h.U.D = x; h.U = x; h.size += 1
            if first is None:
                first = x
            else:
                x.L, x.R = first.L, first
                first.L.R = x; first.L = x
    return root

def cover(c):
    c.R.L, c.L.R = c.L, c.R
    i = c.D
    while i is not c:
        j = i.R
        while j is not i:
            j.D.U, j.U.D = j.U, j.D
            j.C.size -= 1
            j = j.R
        i = i.D

def uncover(c):
    i = c.U
    while i is not c:
        j = i.L
        while j is not i:
            j.C.size += 1
            j.D.U, j.U.D = j, j
            j = j.L
        i = i.U
    c.R.L, c.L.R = c, c  # le relien de la colonne : indispensable
```

```
def search(root, sol):
    if root.R is root:
        yield list(sol)
        return
    c = min(iter_cols(root), key=lambda h: h.size)
    cover(c)
    r = c.D
    while r is not c:
        sol.append(r.row)
        j = r.R
        while j is not r:
            cover(j.C); j = j.R
        yield from search(root, sol)
        j = r.L
        while j is not r:
            uncover(j.C); j = j.L
        sol.pop()
        r = r.D
    uncover(c)

def iter_cols(root):
    c = root.R
    while c is not root:
        yield c
        c = c.R

def primes(m):
    return [p for p in range(2, m + 1) if all(p % q for q in range(2, isqrt(p) +
1))]

def decomposants(n):
    ps = primes(isqrt(n))
    rows = {}
    for s in range(3, n // 2 + 1, 2):
        cs = [p for p in ps if (s % p or s == p) and (n - s) % p]
        if cs:
            rows[s] = cs
    return sorted(sol[0] for sol in search(build(ps, rows), []))

def naif(n):
    pr = lambda k: k > 1 and all(k % q for q in range(2, isqrt(k) + 1))
    return [s for s in range(3, n // 2 + 1) if pr(s) and pr(n - s)]

def tableaux(n):
    """Affiche les tableaux à booléens (1 = cellule présente) puis le résultat
de la danse."""
    ps = primes(isqrt(n))
    dec = set(decomposants(n))
    # 1. matrice de couverture réellement donnée à la danse (colonnes =
premiers)
    print(f"Matrice de couverture, n = {n} (1 = p ne divise ni s ni n-s) ; * =
ligne pleine")
    print("   s | " + " ".join(f"{p:>2}" for p in ps))
    for s in range(3, n // 2 + 1, 2):
        cells = [int((s % p or s == p) and (n - s) % p != 0) for p in ps]
        if any(cells):
            star = "*" if all(cells) else " "
            print(f"{s:>4}{star}| " + " ".join(f"{c if c else ' ':>2}" for c in
cells))
    # 2. tableau unique de la note : colonnes r(p), résidus permis, module 2 mis
de côté
    ps3 = [p for p in ps if p > 2]
    cols = [(p, r) for p in ps3 for r in range(1, p) if r != n % p]
    print(f"\nTableau unique de la note : 1 en colonne r(p) si s = r (mod p), r
```

```
permis")
    print("   s | " + " ".join(f"{r}({p})" for p, r in cols))
    for s in range(3, n // 2 + 1, 2):
        line = [int(s % p == r) for p, r in cols]
        star = "*" if s in dec else " "
        print(f"{s:>4}{star}| " + " ".join(f"{'1' if c else '':>{len(f'{r}
({p})')}}" for c, (p, r) in zip(line, cols)))
    print(f"\nCouvertures exactes trouvées par la danse : {sorted(dec)}")

if __name__ == "__main__":
    args = sys.argv[1:]
    n = int(args[0]) if args else 98
    tableaux(n)
    print(n, decomposants(n), "(naïf :", naif(n), ")")
    if "--test" in args:
        m = int(args[args.index("--test") + 1])
        for k in range(6, m + 1, 2):
            assert decomposants(k) == naif(k), k
            if k % 100 == 0:
                print("  ok jusqu'à", k, flush=True)
        print(f"OK : DLX = test naïf pour tout n pair de 6 à {m}")
```